

CICS Transaction Server for z/OS  
Version 6

*Security for CICS*



**Note**

Before using this information and the product it supports, read the information in [Product Legal Notices](#).

This edition applies to the IBM® CICS® Transaction Server for z/OS®, Version 6 (product number 5655-YA1) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1974, 2024.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>About this PDF.....</b>	<b>xi</b>
<b>Chapter 1. What does security mean for CICS?.....</b>	<b>1</b>
<b>Chapter 2. CICS security is a team sport.....</b>	<b>5</b>
<b>Chapter 3. How it works: Identification in CICS.....</b>	<b>7</b>
Identity propagation.....	14
<b>Chapter 4. How it works: Authentication in CICS.....</b>	<b>17</b>
Which authentication method can I use with CICS access methods?.....	19
Passwords and passphrases.....	20
PassTickets.....	21
Multi-Factor Authentication (MFA).....	22
ICRX (Extended Identity Context Reference).....	24
X.509 certificates.....	24
JSON Web Token (JWT) .....	26
Configuring RACF for JWT.....	30
OAuth 2.0.....	30
OpenID Connect.....	32
Kerberos.....	33
Configuring RACF for Kerberos.....	35
Lightweight Third-Party Authentication (LTPA).....	38
SAML.....	39
Configuring CICS for SAML.....	40
User registries.....	47
<b>Chapter 5. How it works: Authorization in CICS.....</b>	<b>49</b>
Transaction security.....	49
Resource security.....	50
Use of the WARNING option.....	52
Command security.....	52
Security for CICS shutdown.....	54
Intercommunication security.....	55
Application-specific security (QUERY SECURITY).....	58
Post-initialization processing.....	60
Surrogate security.....	60
Role authorization.....	62
<b>Chapter 6. How it works: Integrity and confidentiality in CICS.....</b>	<b>63</b>
Data in motion.....	63
Transport Layer Security (TLS) for IP connections.....	63
TLS cipher suites.....	65
Implementation options for TLS.....	66
Performance considerations for TLS connections.....	69
SOAP messages.....	71
3270 sessions.....	71
Data in memory.....	72
Data at rest.....	73

<b>Chapter 7. How it works: Auditing in CICS .....</b>	<b>75</b>
<b>Chapter 8. How it works: Zero trust in CICS.....</b>	<b>79</b>
How it works: CICS security discovery.....	82
How it works: Capturing and validating security definitions during the development process.....	83
<b>Chapter 9. How it works: Securing CICS with RACF .....</b>	<b>89</b>
RACF administration.....	92
RACF users and roles.....	92
The CICS segment in user profiles.....	93
The LANGUAGE segment in user profiles.....	94
Obtaining CICS-related data for a user.....	95
Changing the RACF profile of a user.....	95
RACF classes and profiles for resources.....	96
Caching of RACF classes and their profiles.....	98
Defining generic profiles for resources.....	98
Defining your own resource classes.....	99
Setting up installation-defined classes in RACF.....	100
Resource classes for DB2ENTRY resources.....	100
Prefixing RACF profiles with SECPRFX.....	101
RACF data set profiles.....	101
Controlling access to fields in RACF profiles.....	101
Security administration tasks and the RACF commands to run them.....	102
Security classification of data and users.....	103
Invoking an external security manager.....	104
<b>Chapter 10. Implementing a zero trust strategy.....</b>	<b>105</b>
What is security metadata?.....	106
Migrating to zero trust with CICS security discovery.....	113
Creating input security metadata for security discovery analysis.....	115
Capturing security discovery data (SDD).....	116
Activating CICS security discovery during CICS startup.....	119
Estimating performance cost of additional resource security using security discovery statistics..	120
Analyzing security definitions using security discovery in CICS Explorer.....	122
Creating RACF commands from output security metadata of security discovery analysis.....	150
CICS security discovery considerations.....	152
Capturing security definitions during development using security definition capture.....	154
<b>Chapter 11. Security through the network layers.....</b>	<b>157</b>
Transport Layer Security (TLS).....	157
Application Transparent Transport Layer Security (AT-TLS).....	158
Changing TLS protocol level or ciphers safely.....	158
Enabling TLS 1.3 in CICS.....	162
Security for TCP/IP clients.....	164
Configuring CICS to use SSL.....	165
Setting up profiles in RACF.....	166
Requesting a certificate from a certificate authority.....	167
Building a key ring manually.....	167
Building a key ring with certificates using DFH\$RING.....	168
Creating new RACF certificates.....	170
Associating a RACF user ID with a certificate.....	171
Using an existing certificate that is not owned by the CICS region user ID.....	171
Configuring a RACF site certificate for use with CICS TS.....	172
Making a certificate untrusted.....	172
Setting the System SSL environment for CICS.....	173
System initialization parameters for SSL.....	176

TCIPSERVICE attributes for SSL.....	177
Creating a TLS cipher suite specification file.....	178
Customizing encryption negotiations.....	179
Making your CICS TS system conformant to NIST SP800-131A.....	180
Configuring LDAP for CICS use.....	181
Using certificate revocation lists (CRLs).....	182
Intrusion Detection Services.....	185
<b>Chapter 12. Security for MRO.....</b>	<b>187</b>
How it works: CICS MRO security.....	187
MRO connection (bind-time) security.....	187
MRO link security.....	190
MRO user security.....	192
Designing security for MRO.....	193
Design example: MRO connection – using only the task user ID.....	195
Design example: MRO connection – using only the link user ID.....	197
Design example: MRO connection – using both the task ID and the link user ID.....	200
Configuring security for MRO.....	203
<b>Chapter 13. Security for IPIC (IP interconnectivity).....</b>	<b>205</b>
How it works: CICS IPIC security.....	205
IPIC connection security .....	205
IPIC transport security .....	208
IPIC link security .....	209
IPIC user security.....	209
Designing security for IPIC.....	211
Design example: Securing CICS-to-CICS with an IPIC connection within a sysplex.....	213
Design example: Securing CICS-to-CICS with an IPIC connection that uses TLS.....	217
Design example: Securing client to CICS with a trusted IPIC connection.....	222
Configuring security for IPIC.....	226
Installing a resource with a user ID attribute.....	226
<b>Chapter 14. Security for LU6.1 and LU6.2 connections.....</b>	<b>229</b>
<b>Chapter 15. Security for Java applications.....</b>	<b>231</b>
How it works: CICS Liberty security.....	231
How it works: Liberty angel process.....	235
How it works: Securing Liberty web applications.....	237
How it works: Securing Link to Liberty applications.....	246
How it works: Securing JCICSX applications.....	247
How it Works: OSGi JVM server security.....	249
How it works: Restricting access to Java APIs.....	250
How it Works: Security for new Java threads and tasks.....	250
Designing security for CICS Liberty applications.....	252
Designing application security for Liberty with the angel process.....	252
Designing security for Liberty web applications.....	253
Designing security for Link to Liberty applications.....	274
Designing security for JCICSX applications.....	278
Configuring security for CICS Liberty.....	290
Configuring the angel process.....	290
Configuring authentication in CICS Liberty.....	297
Configuring authorization in CICS Liberty.....	312
Configuring confidentiality and integrity in CICS Liberty.....	327
<b>Chapter 16. Security for SOAP web services.....</b>	<b>341</b>
How it works: SOAP message security .....	341
How it works: Authentication for CICS with SOAP message security.....	344

How it works: Signing SOAP messages .....	346
How it works: SOAP message encryption .....	348
Designing security for CICS web service providers.....	350
Design example: Securing a direct request with TLS client authentication.....	354
Design example: Asserting an identity to the CICS web service provider.....	358
Designing security for CICS web service requesters.....	362
Design example: Securing a web service request with TLS client authentication.....	364
Design example: Securing a web service request with identity assertion.....	367
Configuring SOAP message security for CICS web services.....	371
Installing the prerequisites for WS-Security support.....	372
Configuring the pipeline for WS-Security.....	373
Configuring provider mode web services for identity propagation.....	376
Configuring RACF for WS-Security.....	378
Invoking the Trust client from a message handler.....	380
Writing a custom security handler.....	380
<b>Chapter 17. Security for 3270.....</b>	<b>383</b>
How it works: 3270 security.....	383
3270 connections to z/OS.....	383
CICS connection to VTAM to enable 3270 terminals.....	384
3270 devices that connect to CICS.....	385
3270 terminal authentication.....	390
3270 terminal sign-off.....	391
Designing 3270 terminal security.....	392
Design example: Dynamically defined 3270 terminal connects and authenticates to a CICS region.....	393
Design example: 3270 terminal with a preset user ID connects.....	396
Configuring 3270 terminal security.....	399
<b>Chapter 18. Security for CICSplex SM.....</b>	<b>401</b>
Security planning for CICSplex SM .....	401
Implementing CICSplex SM security.....	402
Determining who requires access to CICSplex SM resources.....	402
General requirements for CICSplex SM security.....	406
Creating profiles for the CICSplex SM data sets.....	406
Determining the agent user ID for CICSplex SM components.....	408
Defining the CICSplex SM started tasks.....	410
Defining the CICSplex SM transactions in a CMAS.....	410
Defining the transactions in a managed CICS region.....	411
Setting up CICSplex SM Web User Interface security.....	411
Specifying CICSplex SM resource names in profiles.....	416
Security for platforms and applications.....	436
Activating simulated CICS security.....	439
BAS security considerations.....	440
Considerations for CICS surrogate security checks.....	441
Activating security for CICSplex SM.....	441
Refreshing RACF profiles for CICSplex SM.....	442
CICSplex SM security checking sequence.....	443
Invoking a user-supplied external security manager.....	447
Invoking an external security manager.....	447
CICSplex SM security control points.....	448
Example RACF commands for securing CICSplex SM.....	448
<b>Chapter 19. Security for Node.js applications.....</b>	<b>451</b>
<b>Chapter 20. Security for CICS web support.....</b>	<b>453</b>
User IDs for access to document templates and z/OS UNIX files used by CICS Web support.....	453

CICS as an HTTP server: authentication and identification.....	454
CICS as an HTTP client: authentication and identification.....	455
Identifying HTTP users.....	456
Authenticating HTTP users.....	458
Password expiry management for HTTP basic authentication.....	459
Support for HTTP strict transport security (HSTS).....	460
CICS system and resource security for CICS web support.....	462
Security for inbound ports.....	462
Security for CICS system components.....	463
Resource and transaction security for application-generated responses.....	463
Resource-level security for static responses using document templates.....	466
SSL with CICS web support.....	467
Introduction to Application Transparent Transport Layer Security (AT-TLS).....	467
Security for Atom feeds.....	477
<b>Chapter 21. Security for platforms and applications.....</b>	<b>479</b>
<b>Chapter 22. Security in BTS.....</b>	<b>483</b>
Process and activity user IDs.....	483
<b>Chapter 23. Security for the CICS-MQ adapter.....</b>	<b>485</b>
Implementing security for CICS-MQ adapter transactions.....	485
CICS-MQ adapter user IDs.....	486
Command security for MQCONN and MQMONITOR resources.....	486
Surrogate user security for MQMONITOR resources.....	487
IBM MQ connection security for the CICS-MQ adapter.....	487
<b>Chapter 24. Security for the CICS-MQ bridge.....</b>	<b>489</b>
<b>Chapter 25. Security for data sources.....</b>	<b>493</b>
Security for Db2.....	493
Controlling access to Db2-related resources in CICS.....	494
Providing authorization IDs to Db2 for the CICS region and for CICS transactions.....	501
Authorizing users to access resources in Db2.....	508
Db2 multilevel security and row-level security.....	511
Security for DBCTL.....	511
Security for data sets: encryption.....	512
<b>Chapter 26. Security for EXCI.....</b>	<b>515</b>
Using MRO logon and bind-time security.....	515
Link security for EXCI.....	516
User security for EXCI.....	516
Surrogate user checking for EXCI.....	516
<b>Chapter 27. Security for ONC RPC.....</b>	<b>519</b>
Security in CICS and its effect on CICS ONC RPC operations.....	519
RACF Secured Sign-on for ONC RPC clients.....	520
Writing the resource checker.....	521
Reference information for the resource checker.....	521
<b>Chapter 28. Security for data tables.....</b>	<b>525</b>
Security for CICS shared data tables.....	525
SDT server authorization security check.....	525
CONNECT security checks for AORs.....	526
Security for coupling facility data tables.....	528

<b>Chapter 29. Security for CICS resources.....</b>	<b>531</b>
CICS system resource security.....	531
CICS installation requirements for RACF.....	531
Specifying the CICS region user ID.....	532
Authorizing CICS to start up under RACF.....	533
Defining user profiles for CICS region user IDs.....	534
Defining the default CICS user ID to RACF.....	536
Authorizing access to z/OS log streams.....	538
Authorizing access to CICS data sets.....	538
Authorizing access to the temporary storage pools.....	542
Authorizing access to temporary storage servers.....	542
Authorizing access to named counter pools and servers.....	543
Authorizing access to SMSVSAM servers.....	545
Authorizing access to the CICS region.....	545
Controlling the opening of a CICS region's z/OS Communications Server ACB.....	546
Controlling userid propagation.....	547
Surrogate job submission in a CICS environment.....	547
JES spool protection in a CICS environment.....	548
Authorizing use of <b>HPO</b> in <b>PARM</b> parameter on an EXEC PGM=DFHSIP statement or in SYSIN.....	548
Setting up PassTickets.....	548
Supporting parallel PassTicket requests for the same user ID.....	549
Using PassTickets in FEPI applications.....	550
The sign-on process.....	550
The sign-off process.....	552
Controlling access to CICS from specific ports of entry.....	553
Defining port of entry profiles.....	553
Terminal profiles.....	553
Example terminal profiles.....	554
Universal access authority for undefined terminals.....	555
Conditional access processing.....	555
Console profiles.....	556
Preset terminal security.....	556
Controlling the use of preset security.....	557
Using a z/OS system console as a CICS terminal.....	558
Security for transient data.....	560
Security for journals and log streams.....	561
Security for started transactions.....	562
Transactions started at terminals.....	563
Transactions started by EXEC CICS START.....	563
Security for transactions started with EXEC CICS RUN TRANSID.....	565
Security for XPCT-checked transactions.....	566
Security for CICS programs.....	566
Security for temporary storage.....	568
Security for z/OS UNIX files.....	569
Implementing security for z/OS UNIX files.....	570
Security for program specification blocks.....	572
Security checking of transactions running under CEDF, CEDG, CEDX, or CEDY.....	573
Security for submitting a JCL job to the internal reader.....	573
Configuring RACF for identity propagation.....	576
<b>Chapter 30. Auditing CICS.....</b>	<b>579</b>
Compliance data collection.....	579
Retrieving SMF 1154 subtype 80 records from CICS.....	579
Example of formatted SMF 1154 subtype 80 records.....	579
Auditing CICS configuration with IBM Health Checker for z/OS.....	580
Identifying changes to resources (resource signatures).....	582



Resource signature field values.....	582
Classifying CICS regions with region tagging.....	587
Auditing installed resource.....	588
Auditing SPI commands.....	589
Auditing sign-on and sign-off .....	592
Auditing bind time security (LU6.2).....	593
Auditing authorization.....	594
Auditing RACF.....	595
<b>Chapter 31. Developing SAML applications.....</b>	<b>597</b>
Developing a SAML-aware initial program.....	597
Developing a program that uses a validated SAML token in an outbound request.....	598
Developing a program that creates or augments SAML tokens.....	598
<b>Chapter 32. Troubleshooting security.....</b>	<b>601</b>
Problem determination in a CICS-RACF security environment.....	601
Access is denied incorrectly.....	601
Access is allowed incorrectly.....	606
Diagnosing access issues with security request recording (SRR).....	607
CICS initialization failures related to security.....	615
Password expiry management problems.....	620
Troubleshooting Kerberos.....	620
Troubleshooting Passtickets.....	622
<b>Chapter 33. Security reference.....</b>	<b>623</b>
Available CICS health checks supported by IBM Health Checker for z/OS.....	623
CICS_CAT3_CONFIGURATION.....	623
CICS_CEDA_ACCESS.....	625
CICS_JOBSUB_SPOOL.....	626
CICS_JOBSUB_TDQINTRDR.....	627
CICS_REGION_CONFIGURATION.....	628
CICS_RESOURCE_CONFIGURATION.....	630
CICS_RESOURCE_SECURITY.....	632
CICS_STABILIZED_FUNCTIONS.....	634
CICS_USS_CONFIGURATION.....	635
Summary of RACF classes for CICS resources.....	636
CICS transactions subject to security checking.....	641
Commands subject to command security.....	644
Resources subject to command security checking.....	645
Resource and command check cross-reference.....	655
Responses to SDC HTTP calls.....	673
Data captured by CICS security discovery.....	675
Security discovery statistics.....	676
Security-related system initialization parameters.....	678
Security standards.....	680
SMF 1154 subtype 80 record.....	681
SRR reason codes.....	682
The SAML linkable interface, DFHSAML.....	683
<b>Notices.....</b>	<b>691</b>
<b>Index.....</b>	<b>697</b>



## About this PDF

---

This PDF describes how to plan and implement security across your CICS systems. It is intended for security administrators responsible for controlling access to resources used by CICS. These resources are used by CICS terminals, users, or transactions in CICS regions, and by CICS application programs running in those regions. This PDF will also be of interest for CICS system programmers who need to communicate their requirements to the security administrator for their installation.

This PDF replaces *RACF Security Guide*, SC34-7423-00.

For details of the terms and notation used, see [Conventions and terminology used in the CICS documentation](#) in IBM Documentation.

### **Date of this PDF**

This PDF was created on 2024-06-11 (Year-Month-Date).



---

# Chapter 1. What does security mean for CICS?

CICS Transaction Server itself has security facilities and it also benefits from security capabilities that are integrated into the z/OS operating system and IBM Z hardware. You choose how to apply these facilities, depending on the security objectives for your organization and on the types of CICS environments and applications that you run.

## Who do you trust?

Security is based on trust. CICS Transaction Server supports different levels of security, depending on your connectivity and the trust associated with it. The lower your trust, the higher the level of security that you need.

If CICS is directly connected to systems outside your enterprise, you need a higher level of security. If CICS connects only to trusted systems inside your enterprise and the users of the applications are trusted (for example, they are all employees of your enterprise), a medium level of security can be used. If CICS is isolated and can be reached only by trusted systems and trusted users, a lower level of security can be used.

If you are operating a Zero Trust strategy, then your aim is a principle of never trust, always verify. In this case access is minimized to only what users' need for their role, and access checks made on all resources that a user accesses. This checking is independent of whether a request to CICS originates inside or outside of the enterprise.

When you consider the security design for your CICS application, you need to consider the following questions:

- How can you ensure that the users of the application are properly authenticated?
- What authorization mechanisms are used to protect access to the CICS system and access to resources such as transactions, files, and databases?
- How can you protect the confidentiality of data that is transported between the different tiers of the physical configuration?
- How can you meet audit and compliance regulations?

## Security principles

Software security follows some fundamental guiding principles. Here's how CICS supports those principles:

### Identification

Identification is the ability to assign an identity to the entity that is attempting to access the system. Typically, the identity is used to control access to resources and to audit requests. Depending on the authentication model, the identity might come from the authentication credentials or it might be asserted from another server.

CICS is designed to work with user IDs that are stored in the z/OS user registry. It can also work with non-z/OS user IDs, such as an X.500 ID, and these non-z/OS user IDs are known as *distributed IDs*. For authorization purposes, distributed IDs can be mapped to a z/OS user ID on entry to CICS. The distributed ID might also flow alongside the z/OS user ID so that it is available to the application and can be used for auditing.

*User registries* store user account information so that it can be accessed during authentication and authorization. RACF is the primary z/OS user registry, which is accessed through the SAF interface. Some other types of user registry can also be used with CICS, for example, if you use CICS Liberty, LDAP.

See [How it works: Identification in CICS](#) and [User registries](#) for more information.

## Authentication

Authentication is the process of validating the identity that is claimed by the accessing entity: is this entity who it claims to be? Authentication is done by verifying authentication information that is provided with the claimed identity. The authentication information is generally referred to as the accessor's *credentials*. Authentication is usually the first step in security processing. After it is authenticated, an identity can be *asserted* to the downstream security processing, meaning that these steps trust that the identity was successfully authenticated by the upstream steps.

In CICS, authentication can be omitted if the asserted identity comes from a trusted system that already authenticated the user.

CICS Transaction Server supports the following types of authentication:

- Basic authentication, which uses credentials in the form of a password or passphrase. The passwords or passphrase are verified by the user registry. A special version of a basic authentication token is a *PassTicket*, which is a time-limited, one-time-use password for a specific system.
- Multi-factor authentication, which requires the user to provide multiple pieces of information: something you know, something you have, and something you are. This information is provided in the form of a *Multi Factor Authentication (MFA)* token. They can generally be used in the same interfaces in CICS as passwords and passphrases.
- Client certificate authentication, which requires the user to provide a digital certificate that is then mapped to a user ID. The certificate must be issued by a trusted third party or certificate authority.
- Third-party authentication, where the client authenticates with a third-party authentication server. This server generates an authentication token such as a JSON Web Token (JWT), Security Assertion Markup Language (SAML) token, or a Kerberos token. The authentication token is then used to authenticate with CICS.

See [How it works: Authentication in CICS](#) for more information.

## Authorization

Authorization is the process of checking whether an identity that is already authenticated should be given access to the resource that it is requesting. A typical implementation of authorization is to pass a security context that contains the identity that was authenticated, to the access control mechanism.

CICS Transaction Server supports different types of authorization to:

- Run CICS transactions (*transaction security*)
- Access resources (*resource security*)
- Use of CICS system programming commands (*command security*)
- Communicate between CICS systems (*intercommunication security*)
- Control access to a resource in an application served by Liberty (*role authorization*).

See [How it works: Authorization in CICS](#) for more information.

## Integrity

Integrity ensures that information was not altered in an unintended way during transmission or storage. When data is transmitted over a network, integrity checks that the received data is the same as the data that was sent.

CICS Transaction Server uses security protocols, such as SSL or, more recently, TLS, AT-TLS, or JSSE, and message-based security to provide data integrity for the transmission of data. See [How it works: Confidentiality and integrity in CICS](#) for more information.

## Confidentiality

Confidentiality, or *privacy* ensures that an unauthorized party cannot obtain access to data. Typically, confidentiality is achieved by encrypting the data, for example, by using TLS or WS-Security.

See [How it works: Confidentiality and integrity in CICS](#) for more information.

## Auditing

Auditing is a key part of a security infrastructure. It involves the validation of security-related events, such as the sign-on of a user, to ensure that security standards were correctly applied to the processing of requests.

See [How it works: Auditing in CICS](#) for more information.

## Nonrepudiation

Nonrepudiation means that a data sender and a data receiver are able to provide proof to a third party that the sender sent the information, and the receiver received the identical information. Neither side can then deny that it happened.

## Security facilities available in CICS

Figure 1 on page 3 summarizes the options that are available in CICS Transaction Server. If Liberty is installed with CICS to run as an application server in a JVM server, more facilities become relevant and these facilities are shown in purple in the diagram.

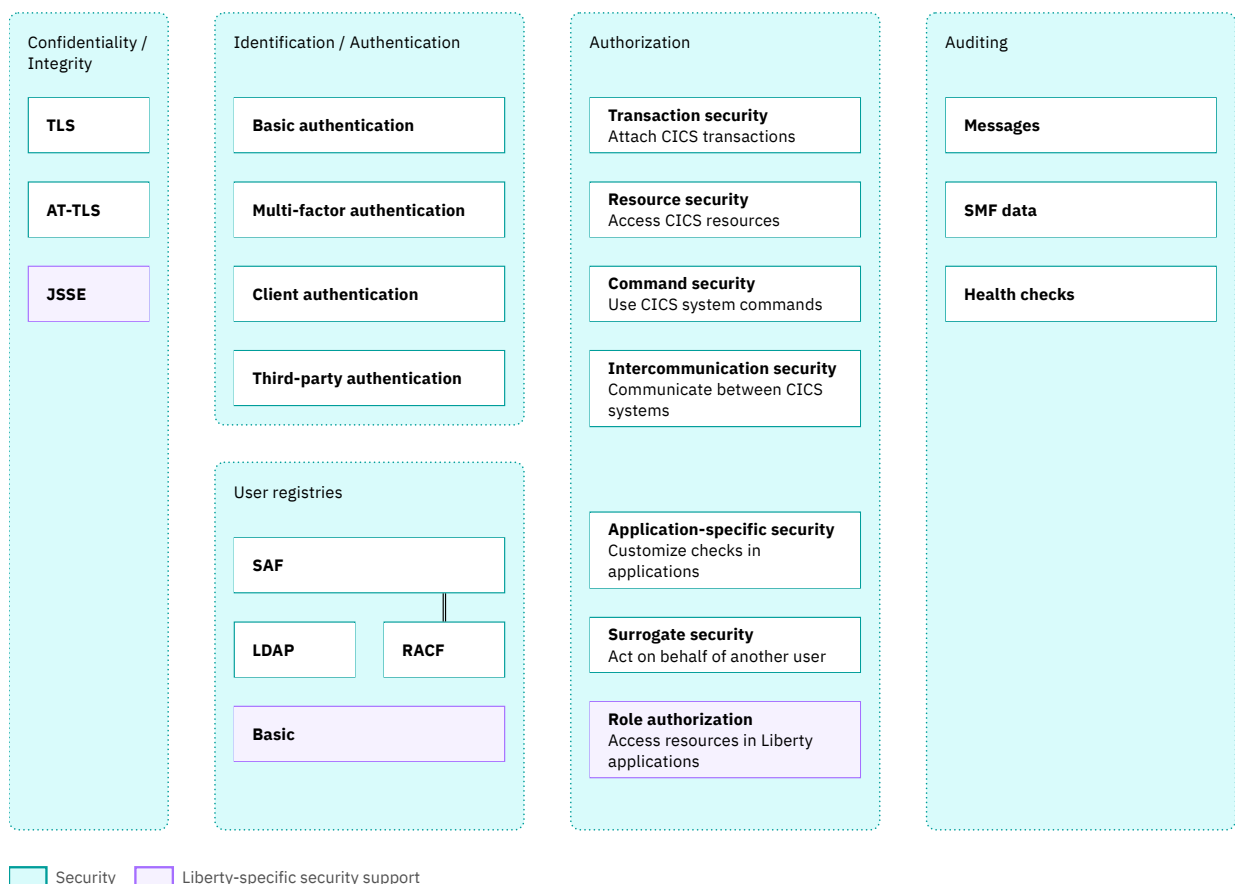


Figure 1. Security options that are available in CICS

## What CICS security doesn't cover

CICS provides various security and control mechanisms to limit the activities of CICS users to only the CICS functions that each user is authorized to use. However, CICS Transaction Server does not directly provide facilities to secure the following:

### External access

Although z/OS provides good security controls to prevent access to data sets and files, CICS itself does not provide facilities to protect its own assets from external access. You must restrict

access to the program libraries, to the CICS regions, and to those users who are responsible for incorporating approved application and system changes. Similarly, the data sets, databases, and zFS-based configuration files that are used by CICS and by CICS applications must be accessible only by approved batch processing and operations procedures.

#### **Applications that use undocumented or unsupported interfaces**

CICS does not protect your system from application programs that use undocumented or unsupported interfaces to bypass CICS security. You are responsible for ensuring that such programs are not installed on your system.

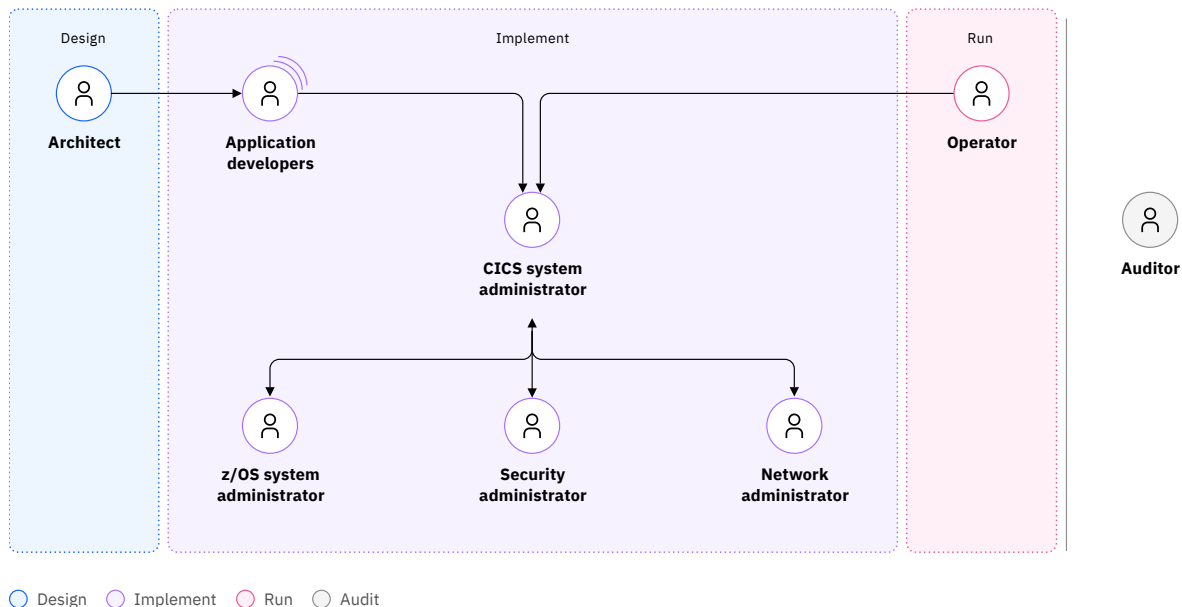
#### **Application source libraries**

Although CICS provides facilities to prevent you running unauthorized code (for example, by restricting access to authorized services and by using the identity of the caller only for calls through the EXEC CICS API so that calls to other z/OS APIs can be protected with the CICS region ID), CICS does not directly protect your application source libraries. Make sure that you have procedures to prevent the introduction of unauthorized or untested application programs into your production application base. You must also protect the integrity of your system by exercising control over libraries that are installed in the system and changes to those libraries.



## Chapter 2. CICS security is a team sport

Whose job is security in CICS? Decisions about security in CICS and its implementation are made by a team of people who fulfill different roles. It's important to understand the different roles and to identify the people who fulfill them in your organization.



### Architect

The architect designs a solution to meet business needs. The solution balances many different requirements, for example: the business outcome, technologies or products to be used, compliance with industry or local standards, and so on.

### Application developers

Application developers implement solutions based on the guidelines provided by the architect. They need to be aware of the security requirements of the application and any application programming interfaces that have security implications. They connect with the system administrators either directly or through their DevOps processes.

### CICS system administrator

The CICS system administrator is responsible for the CICS definitions and configuration that is associated with applications. In many cases, regulations that require separation of duties mean that they are not usually directly responsible for the security definitions. Instead, they either receive requirements from the application team and work with the other administrators to implement those requirements, or they set up their delivery pipeline to do so.

### z/OS system administrator

Much of the infrastructure that supports CICS security comes from the z/OS operating system and the IBM Z hardware on which it runs. The z/OS system administrator is responsible for securely configuring the z/OS system for CICS and other subsystems.

### Security administrator

The security administrator plans and implements definitions, for example, in RACF, to secure applications and the data that they use.

**Network administrator**

The network administrator is responsible for securely configuring the network access to CICS and other IBM Z subsystems. Some security mechanisms used by CICS, for example, AT-TLS, are applied as part of the network architecture.

**Operator**

The operator is responsible for operating and monitoring the system. Problems are then reported back to the administrators.

**Auditor**

The auditor tests the adequacy and effectiveness of the security controls that are implemented in the organization.

## Chapter 3. How it works: Identification in CICS

All CICS tasks are associated with one or more user IDs. The primary user ID is known as the *task user ID*. The task user ID is derived from whoever or whatever started the task and that can be one of many different things. To differentiate the user ID that comes from each of these starting points, CICS uses different names for them. These names are explained in this document.

CICS calls RACF to determine whether the user ID has the authority to complete the request. For example, when a task issues a request to access a resource or to attach a CICS transaction to run another task,

Individual users make requests of CICS and supply identification to do so. The following users (in alphabetical order) are identified with RACF user IDs that reflect how they accessed CICS:

Table 1. Simple RACF user IDs	
Definition	Brief description
<a href="#">Asserted user ID</a>	User ID flowed from a trusted remote system.
<a href="#">Authenticated user ID</a>	User ID shared from a remote system that includes credentials that CICS uses to authenticate the user.
<a href="#">“Client user ID” on page 11</a>	An identity that comes from outside of CICS with some credentials to be authenticated.
<a href="#">EDF user ID</a>	The user ID under which the Execution Diagnostic Facility is run to test a transaction.
<a href="#">Functional user ID</a>	A functional user ID is a user ID used by all users of a function instead of identifying individual client user IDs.
<a href="#">Link user ID</a>	A user ID passed as part of the bind process or defined in a resource definition that is associated with the connection.
<a href="#">Resource user ID</a>	A user ID defined in a resource definition and used to run a task or used to authorize a request.
<a href="#">Session user ID</a>	A user ID that can be associated with a session, such as a 3270 terminal session.
<a href="#">Task user ID</a>	The task user ID is obtained from whoever or whatever started the task.
<a href="#">Technical user ID</a>	Alternative name for <a href="#">functional user ID</a> .

CICS supports surrogate security, where one user can act on behalf of another. The following users (in alphabetical order) are identified with RACF user IDs when CICS is using surrogate security:

Table 2. RACF IDs that use surrogate security	
Definition	Brief description
<a href="#">Batch region user ID</a>	A batch region user ID, is the user ID under which a request will run. It is typically a <a href="#">functional user ID</a> .
<a href="#">“Execution user ID” on page 12</a>	An execution user ID, typically a <a href="#">functional user ID</a> , is paired with a <a href="#">surrogate user ID</a> .
<a href="#">“Surrogate user ID” on page 12</a>	A surrogate user has the authority to start work on behalf of another user (the <a href="#">execution user</a> ).

Not all user IDs originate as RACF user IDs. Remote systems that have different user identities that come from external security realms can connect to CICS. Similarly, the Liberty JVM servers that are used by

CICS use different types of user identity. These different user identities can be mapped to RACF user IDs to enable them to access CICS resources.

Table 3. Remote user IDs that are mapped to RACF	
Definition	Brief description
<a href="#">Distributed identity</a>	A distributed identity represents user identity information from a different security realm that is flowed into CICS.
<a href="#">Java™ security subject</a>	A Java security subject is used by Liberty to represent a user.

“How users are identified in CICS” on page 8 explains how these user IDs are used.

A number of users are associated with the CICS region itself. These users are identified with RACF user IDs that reflect the configuration of CICS:

Table 4. User IDs associated with the CICS region and its configuration	
Definition	Brief description
<a href="#">CICS default user ID</a>	One of the user IDs that is associated with the CICS region to protect resources in the absence of other more specific user identification.
<a href="#">CICS region user ID</a>	One of the user IDs that is associated with the CICS region that is associated with the tasks started when CICS starts.
<a href="#">Kerberos service principal user ID</a>	One of the user IDs that is associated with the CICS region that is related to Kerberos use.
<a href="#">PLT user ID</a>	One of the user IDs that is associated with the CICS region when requests are made by second phase PLT programs.

Information about users is stored in a user registry. See [User registries](#) for the options that CICS supports.

## How users are identified in CICS

[Figure 2 on page 9](#) shows different ways in which user IDs can come into CICS and how they map to the task user ID of the task that processes their requests. When requests come into CICS, various system tasks do processing before the user task is started; these system tasks are not shown in the diagram.

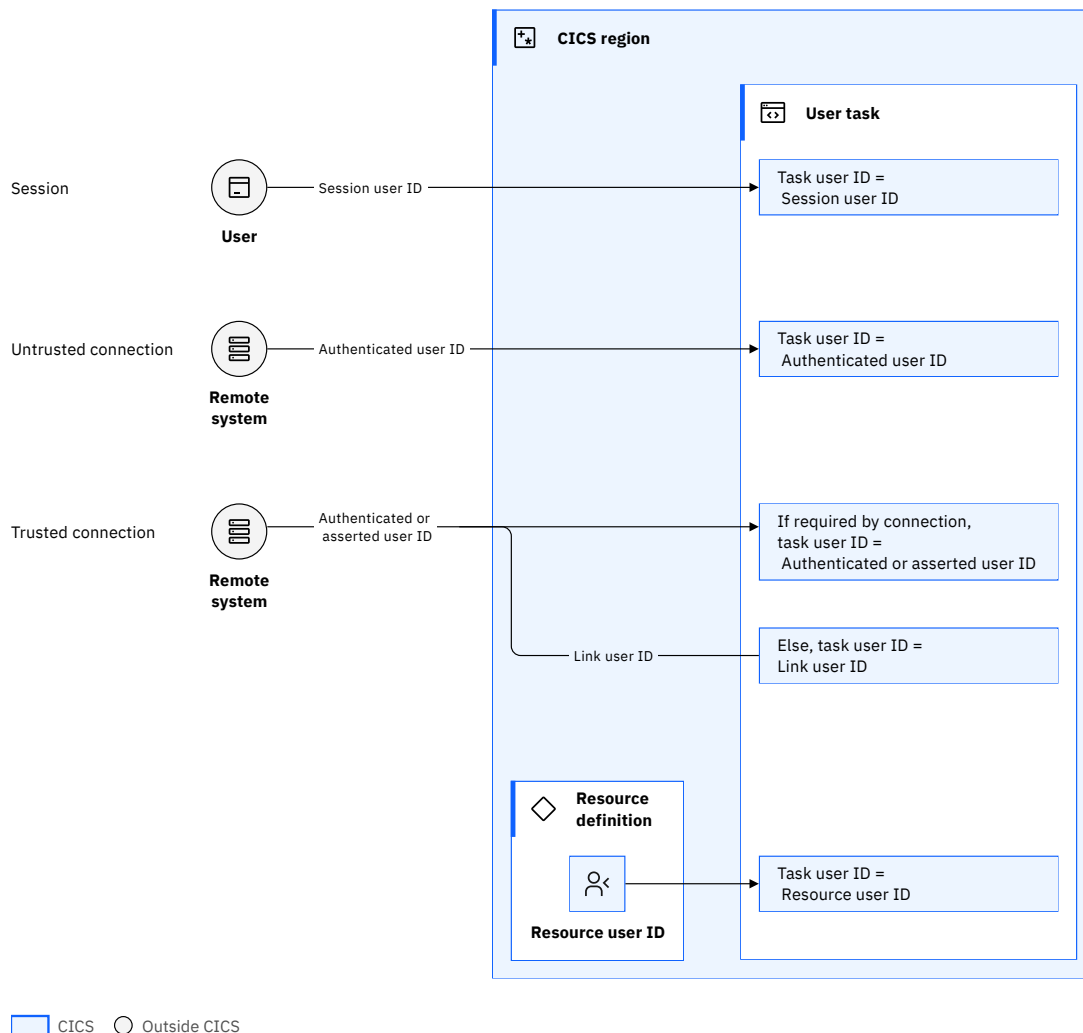


Figure 2. How users are identified in CICS

### From a session

Each user supplies credentials that contain a user ID and a secret, such as a password, when they start the session. This user ID is known as the *session user ID*. Examples of sessions include interaction with CICS through a 3270 terminal, from CICS Explorer (by using CMCI JVM server) or the CICSplex SM WUI. Every request that the user makes during the session goes back to the same connection and runs under the same user ID. Any task that is initiated through the session runs under the session user ID that the user supplied at the start of the session.

### Over an untrusted connection

An untrusted connection is one in which trust is not established for the connection itself. Requests are stateless, so each request must establish trust by supplying credentials that can be authenticated in CICS before the request is processed. The credentials contain an *authenticated user ID*. The *authenticated user ID* is used as the task user ID.

### Over a trusted connection

Two user IDs are associated with requests over a connection:

- A link user ID is associated with the connection itself.
- A user ID that is associated with the request.

The remote system and the CICS system must establish trust when the connection is bound, which can be done in one of two ways:

- The remote system supplies credentials that are authenticated when the connection is established. These credentials can be a certificate, which is associated with the link user ID.
- The session can be securely defined in z/OS, for example, in VTAM definitions. Depending on the type of connection, the link user ID can flow into CICS or be defined in CICS.

The CICS connection definition can require that, in addition to a trust relationship of the connection, the user ID that is associated with the request must also be authenticated. The request must then supply credentials that can be authenticated, resulting in an *authenticated user ID*. Alternatively, the CICS connection definition can specify that it trusts the remote system. In this case, the remote system needs only to supply an *asserted user ID* as part of the request.

The CICS connection definition specifies whether the user ID that is associated with the request (either the *authenticated user ID* or the *asserted user ID*) is required. If so, it is used as the task user ID. If it is not required, the link user ID is used as the task user ID.

### **Inside CICS (during CICS processing)**

As part of CICS processing, CICS uses information in some resource definitions to start a task with a specific user ID or to use a specific user ID for authorization for a request. That task is associated with the user ID that is defined on an attribute of the resource. This user ID is known as a *resource user ID*.

A similar concept is the *PLT user ID* that is used to start a task to run PLTs.

## **Types of user ID in CICS**

The user IDs are listed alphabetically.

### **Asserted user ID**

A message or a request that is received over a trusted connection or component can contain a user ID that has been previously authenticated by another component. The user ID does not need to be authenticated in CICS because the other component is trusted. The user ID is therefore *asserted*. Compare this to an *authenticated user ID* where the user ID that is associated with the request must also be authenticated.

### **Authenticated user ID**

A CICS connection definition can require that, in addition to a trust relationship of the connection, the user ID that is associated with the request must also be authenticated. The request must then supply credentials that can be authenticated, resulting in an *authenticated user ID*. Compare this to an *asserted user ID* where the message or request that is received over a trusted connection can contain a user ID that is previously authenticated by the remote system.

### **CICS default user ID**

One of the user IDs that is associated with the CICS region.

The *CICS default user ID* is used to protect resources in the absence of other more specific user identification. For example, when a 3270 terminal session is started, the session is associated with the *CICS default user ID* until the user signs on.

You must define the *CICS default user ID* with no authority to use secured resources. It is recommended that the default user ID is a member of a RACF group that contains only CICS default user IDs. It must not be a member of any other group.

Because the *CICS default user ID* is key to the running of CICS, it must be protected. It is recommended that this user ID is defined to RACF with the PROTECTED attribute. *Protected user IDs* cannot be used to log on to the system, and are protected from being revoked through incorrect system access attempts. This setting prevents failed password attempts that cause a denial of service attack.

### **Batch region user ID**

A user ID that is used when running batch processes.

This user ID is usually configured as a functional user ID.

## CICS region user ID

One of the user IDs that is associated with the CICS region.

When you start a CICS region in a z/OS environment, either by submitting JCL or by using a started task, the job or task is associated with a user ID. This user ID is known as the *region user ID*. When the started job is CICS, it is known as the *CICS region user ID*. The CICS region user ID accesses the resources that are listed on behalf of the task user. Therefore, the CICS region user ID must have authorization to the following resources:

- z/OS log streams
- Data sets: CICS system data sets and CICS user data sets, JES spool data sets
- Temporary storage data sharing servers
- SMSVSAM server
- Named counter servers
- z/OS Communications Server ACB
- Coupling facility data tables
- RACF key rings
- JVM server and Liberty configuration files and logging directories in zFS

The CICS *region user ID* is used for authorization in the following cases:

- Submitting jobs to the JES internal reader
- Authorizing a surrogate for other user IDs used during CICS execution
- As a prefix for resource names that are passed to RACF
- Running CICS system transactions (also known as Category 1 transactions)
- In CICS intersystem communication
- In Liberty JVM server, to control registration with the angel process

**Recommendation:** Because the CICS *region user ID* is a powerful user ID, it must be protected. This user ID must be defined to RACF with the PROTECTED attribute. *Protected user IDs* cannot be used to log on to the system, and are protected from being revoked through incorrect system access attempts. This setting prevents failed password attempts that cause a denial of service attack.

## Client user ID

An identity that comes from outside of CICS with some credentials to be authenticated. This ID is a RACF-based identity; otherwise, it would be a Distributed identity.

## Distributed identity

A *distributed identity* represents user identity information from a different security realm that is flowed into CICS. This user identity information contains a user ID from a different user registry than the RACF registry used by CICS and might be in a different form, such as an X.509 distinguished ID. The distributed identity is created and authenticated in one system and is passed to one or more other systems over a network. A distributed identity only ever originates outside CICS. CICS is never the source of a distributed identity but can propagate the distributed identity onwards.

A distributed identity is often used with *identity propagation*. For more information, see [Identity propagation](#).

Except for CICS Liberty, a distributed identity is not associated with a password so it must be passed to CICS over a trusted connection or passed in a signed third-party authentication token. Distributed identities can be flowed into CICS by using a third-party authentication token, such as a [How it works: JSON Web Token \(JWT\)](#) or in [Extended Identity Context Reference \(ICRX\)](#) over trusted IPIC connections.

A distributed identity can also be mapped to a RACF user ID outside CICS, for example by IBM DataPower. In this case, only the RACF user ID is passed to CICS as an *asserted user ID*.

### EDF user ID

Execution diagnostic facility (EDF) can be used to test a transaction. When a transaction is under test in dual-screen mode, the user ID under which EDF runs (the *EDF user ID*) becomes significant. When authorization checks are made in the transaction under test, the access authority of the EDF user ID is checked in addition to the user IDs that are normally associated with the transaction (when it is not under test).

### Execution user ID

In surrogate security, an execution user ID is paired with a *surrogate user ID*. The execution user ID is typically a *functional user ID*. A surrogate user is authorized to act for the execution user without authenticating as that user. See [Surrogate security](#).

### Functional user ID

A *functional user ID* is a user ID used by all users of a function instead of identifying individual client user IDs. Typically you can achieve this configuration by using the *link user ID* as the *task user ID*.

### Java security subject

A *Java security subject* is used by Liberty to represent a user. The CICS Liberty security feature can be used to synchronize the Java security subject and the CICS *task user ID*.

### Kerberos service principal user ID

One of the user IDs that is associated with the CICS region. If you use Kerberos, you must define a Kerberos service principal to register the CICS region as a service to Kerberos. This user ID must have a password that is associated with it. As a consequence, it cannot be defined to RACF as a protected user ID and it must not be the same as the CICS *region user ID*.

### Link user ID

The *link user ID* is either passed from the remote system as part of the secure establishment (the bind) of the connection, or it is defined in the resource definition that is associated with the connection. See [“Resources that have link user IDs” on page 13](#) for a list of the resources for which this situation applies.

### PLT user ID

One of the user IDs that is associated with the CICS region.

*PLT* programs run during CICS initialization. Requests that are made by second phase *PLT* programs are associated with the *PLT user ID*. The user ID is defined by the **PLTPIUSR** system initialization parameter.

### Resource user ID

A *resource user ID* is a user ID that is defined on a resource definition. Its purpose is either to run a task or to be used in the authorization of a request. See [“Resources that have user IDs” on page 13](#) for a list of the resources that are associated with resource user IDs.

### Session user ID

A user ID can be associated with a session, such as a 3270 terminal session. This user ID is normally associated with the session when the user signs on to the terminal.

### Surrogate user ID

A surrogate user has the authority to start work on behalf of another user (the *execution user*). The execution user is typically a functional user ID. A surrogate user is authorized to act for the execution user without authenticating as that user. See [Surrogate security](#).

### Task user ID

All CICS tasks are associated with a user ID. This user ID is obtained from whoever or whatever started the task.

### Technical user ID

Alternative name for *functional user ID*. The documentation refers to *Functional user ID* throughout.

## User IDs associated with a CICS task

A CICS task can be associated with up to three user IDs:



- Task user ID
- Link user ID
- EDF user ID

The task user ID is the primary user ID. Only the *task user ID* is displayed in CICS commands that show the user ID that is associated with a task. However, all three user IDs are used in authorization.

When a task is initiated over a connection, it can have an *asserted user ID* and a *link user ID*. The connection can be defined so that an *asserted user ID* is not needed. In this case, the task user ID is the link user ID. This type of *link user ID* is often known as a *functional user ID*.

When a task is being debugged by EDF, the user ID that controls the EDF session is known as the *EDF user ID*.

When a task is started in a CICS Liberty server or links to a program running in a CICS Liberty server, an assertion can be made between the Java security subject and the CICS task user ID. For more information, see [CICS Liberty security feature](#).

## Resources that have user IDs

Table 5 on page 13 lists the resources that have user IDs (excluding link user IDs) associated, their attributes, and the purpose of these user IDs. For more information, see [Resource definition attributes](#).

Table 5. Resources that have user IDs		
Resource	User ID attributes	Purpose of user ID
BUNDLE with transaction start EP adapter	USERID in the properties of the <a href="#">transaction start EP adapter</a> .	User ID under which the transaction is started
<a href="#">DB2CONN</a>	AUTHID (used in API requests) AUTHTYPE	For authorization of pool threads
	COMAUTHID (used in API requests), COMAUTHTYPE	For authorization of command threads
	SIGNID	To sign on to IBM Db2 for authorization of pool and DB2ENTRY threads
<a href="#">DB2ENTRY</a>	AUTHID (used in API requests), AUTHTYPE	For DB2ENTRY authorization (not RACF)
<a href="#">MQMONITOR</a>	MONUSERID	Task user ID of the MQ monitor
	USERID	Task user ID associated with messages
<a href="#">TDQUEUE</a>	JOBUSERID	Job user ID for JCL that is submitted by using transient data queues
	USERID	Task user ID of triggered task
<a href="#">URIMAP</a>	USERID	Task user ID for the request

## Resources that have link user IDs

Table 6 on page 14 lists the resources that use associated link user IDs, their attributes, and the purpose of these user IDs. For more information, see [Resource definition attributes](#).

Table 6. Resources that have link user IDs		
Resource	User ID attributes	Purpose of user ID
<u>CONNECTION (APPC)</u>	SECURITYNAME	Link user ID
<u>IPCONN</u>	SECURITYNAME	Link user ID
<u>SESSIONS</u>	USERID	Link user ID
<u>TERMINAL</u>	SECURITYNAME	Link user ID
	USERID	Preset user ID

## Identity propagation

z/OS identity propagation allows a user identity from an external security realm (a *distributed identity*) to be mapped to, and asserted as, a specific identity on the target security realm, without using any other means of authentication.

Figure 3 on page 14 gives a high-level view of how identity propagation works with CICS.

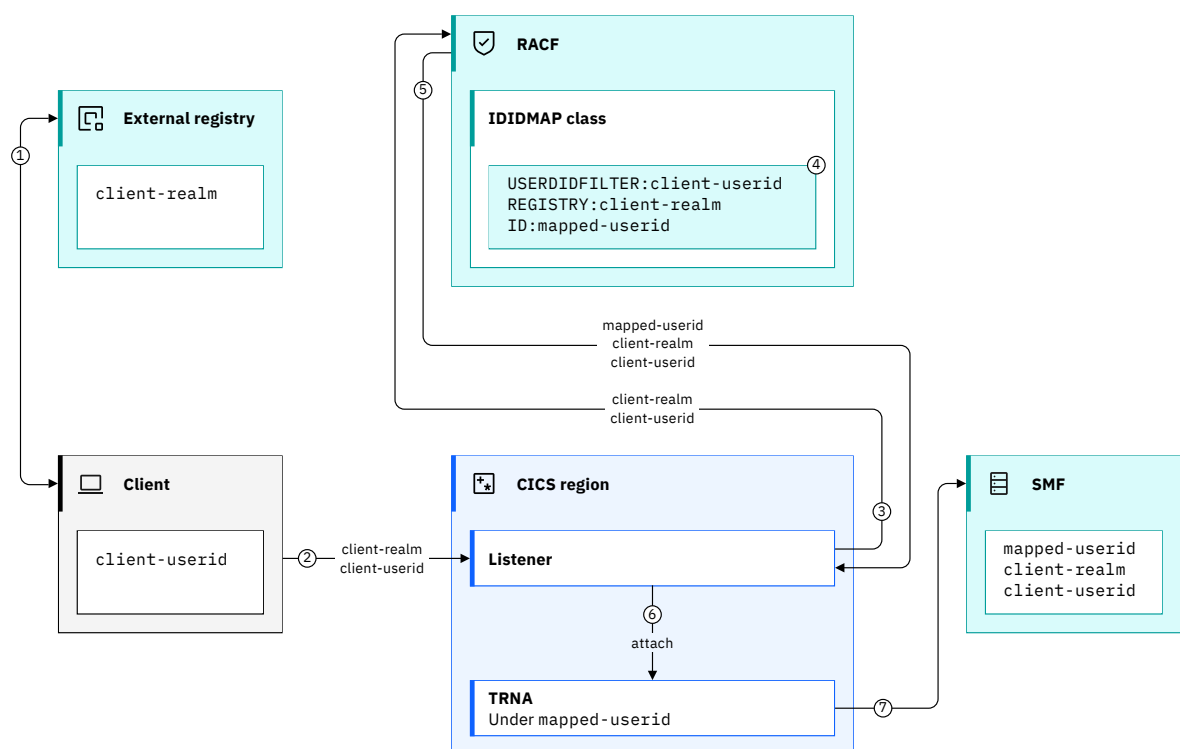


Figure 3. How identity propagation works with CICS

1. The client *client-userid* authenticates with an external registry with the name *client-realm*.
2. A request is sent to CICS containing the distributed identity (*client-userid* and *client-realm*)
3. Depending on how the request is received in CICS (for example, a CICS IPIC request) a CICS listener system task calls RACF to verify that the distributed identity definition exists .
4. RACF maps the distributed identity to a *mapped-userid* (a RACF user ID) by using a distributed identity filter. The mapping is based on the distributed user's name (*client-userid*) and the name of the registry where the user ID is defined (*client-realm*).
5. RACF returns the mapped-userid and the distributed identity to CICS.

6. CICS attaches the transaction to run under the *mapped-userid*.
7. Both the *mapped-userid* and the distributed identity can be audited (for example in CICS SMF 110 subtype 1 or RACF 80 records), strengthening accountability across the different environments.

The rules for mapping a distributed identity to a RACF user ID are created by the RACF system programmer by using the RACMAP command. The relationship between distributed identities and RACF user IDs can be one to one, or many to one. For more information about using the RACMAP command, see [Configuring RACF for identity propagation](#).

Identity propagation can be used with CICS in different scenarios:

- When a CICS IPIC request is received in CICS, the distributed identity can be passed in an ICRX (Extended Identity Context Reference) token. The CICS transaction runs with the mapped user ID but CICS audits both the mapped user ID and the distributed identity.

If the CICS transaction links to other programs, or starts other transactions, in remote CICS regions within the sysplex, the distributed identity is automatically propagated across these CICS regions. The automatic propagation does not happen for STARTs with TERMID or USERID.

CICS security handles the distributed identity as additional information that relates to the user ID in the task [association data](#).

For an example scenario that shows how identity propagation is used with CICS IPIC, see [Design example: Securing client to CICS with a trusted IPIC connection](#).

- When a SOAP web service request is received in CICS, the distributed identity can also be passed in an ICRX token, in a SOAP header. The ICRX is created by an IBM DataPower Gateway.

If the CICS transaction links to other programs, or starts other transactions, in remote CICS regions within the sysplex, the distributed identity is automatically propagated across these CICS regions. The automatic propagation does not happen for STARTs with TERMID or USERID.

CICS security handles the distributed identity as additional information relating to the user ID in the task [association data](#).

- When identity propagation is used with CICS Liberty, the distributed identity can be flowed in a third-party authentication token, such as a JSON Web Token (JWT). The distributed identity can then be mapped to a RACF user ID by the Liberty JVM server.

If the Enterprise Java application uses JCICS to link to a COBOL program, both the program and the Enterprise Java application run with the mapped user ID.

**Note:** The distributed identity information is not available in the CICS SMF 110 subtype 1 record. If the CICS transaction links to, or starts programs in remote CICS regions, the distributed identity is not propagated across these CICS regions.

For an example scenario that shows how identity propagation is used with CICS Liberty, see [Design example: Securing a web application with a JWT and CICS transaction security](#).

- When a Liberty JVM server is configured to use an LDAP (Lightweight Directory Access Protocol) registry, a user can authenticate directly with the Liberty JVM server by using an LDAP user ID. The LDAP user ID can then be mapped to a RACF user ID.

If the Enterprise Java application uses JCICS to link to a COBOL program, both the program and the Enterprise Java application run with the mapped user ID.

**Note:** The distributed identity information is not available in the CICS SMF 110 subtype 1 record. If the CICS transaction links to, or starts programs in remote CICS regions, the distributed identity is not propagated across these CICS regions.

For more information, see [Configuring Liberty to use an LDAP user registry](#).



## Chapter 4. How it works: Authentication in CICS

CICS handles the authentication process. It requests credentials from a user, decodes the authentication information if necessary, calls RACF® or a third-party authentication server to authenticate the supplied credentials, and rejects the request if the authentication fails. It supports different forms of authentication. Your options for authentication depend on the way that you access CICS; see [Which authentication method can I use with which access method?](#) for details.

CICS authenticates users in the following ways:

- “Basic authentication” on [page 17](#)
- “Multi-factor authentication (MFA)” on [page 17](#)
- “Client authentication” on [page 17](#)
- “Third-party authentication” on [page 18](#)

### Basic authentication

This form of authentication uses credentials in the form of a user ID and password, a passphrase, or a PassTicket.

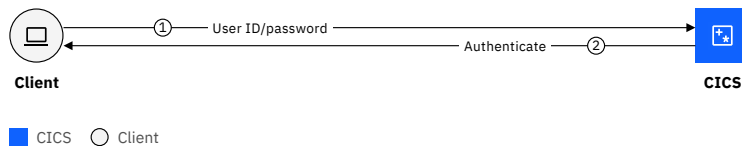


Figure 4. Basic authentication

For more information, see [How it works: Passwords and passphrases](#).

### Multi-factor authentication (MFA)

This form of authentication uses credentials in the form of a user ID and an MFA token that is generated by an external device.

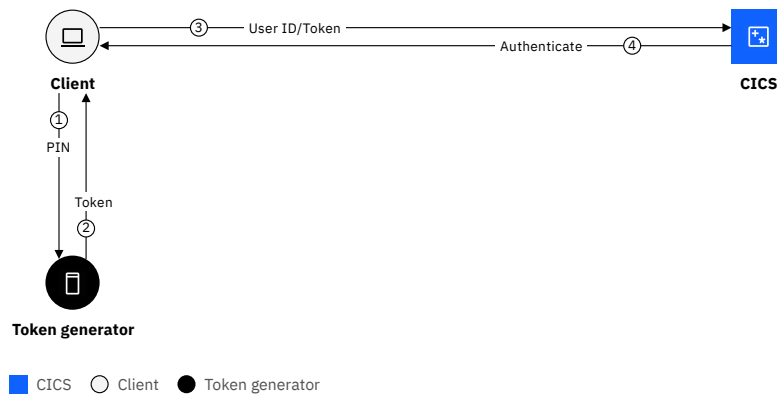


Figure 5. Multi-factor authentication

For more information, see [How it works: Multi-factor authentication \(MFA\)](#).

### Client authentication

This form of authentication uses a TLS certificate to identify the client. Either CICS Liberty, CICS TLS support or Application Transparent Transport Layer Security (AT-TLS) can be used.

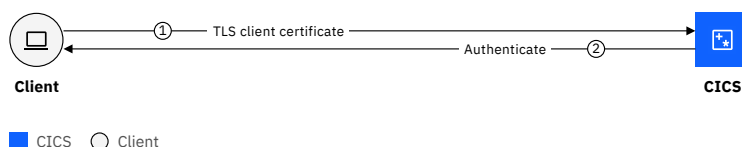


Figure 6. Client authentication

For more information, see [How it works: X.509 certificates](#).

### Third-party authentication

This form of authentication is an architecture that enables a user to authenticate with an authentication server to obtain a token. The authentication token is sent to CICS and CICS validates the token. The identity in the token can also be mapped to a RACF user ID. In some cases, this form can be used for Single Sign-on (SSO) solutions, which allow the client to have access to several servers. CICS supports the following third-party tokens and architectures:

- [How it works: ICRX \(Extended Identity Context Reference\)](#)
- [How it works: JSON Web Token \(JWT\)](#)
- [How it works: OAuth 2.0](#)
- [How it works: OpenID Connect](#)
- [How it works: Kerberos](#)
- [How it works: LTPA \(Lightweight Third Party Authentication\)](#)
- [How it works: SAML \(Security Assertion Markup Language\)](#)

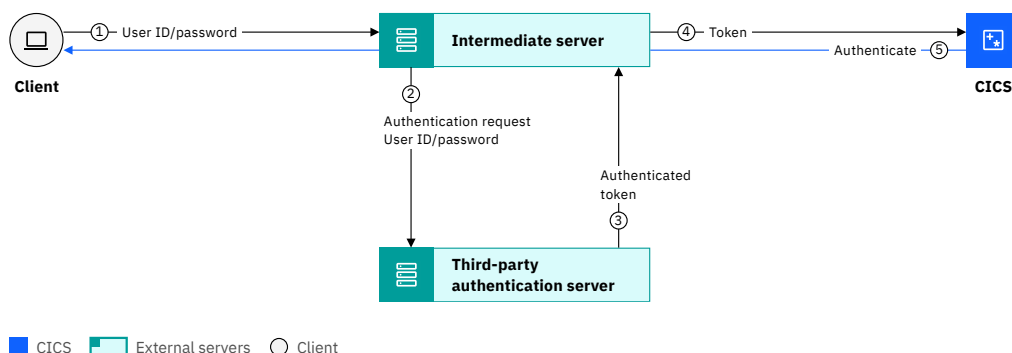


Figure 7. Third-party authentication

The information about users that CICS needs for authentication is stored in a user registry. See [User registries](#) for the options that CICS supports.

## Which authentication method can I use with CICS access methods?

Your options for authentication depend on how you access CICS. This topic describes the security tokens that can be configured in CICS for different access methods. Many of the access methods provide exit points where you can write your own code to provide additional support for security tokens.

*Table 7. Authentication options for different ways into CICS.* The authentication options are shown across the top. The types of access method are shown down the side.

	<b><u>Passw</u> <u>rds and</u> <u>passph</u> <u>rases</u></b>	<b><u>PassTic</u> <u>kets</u> <u>(“1” on</u> <u>page</u> <u>20)</u></b>	<b><u>MFA</u></b>	<b><u>X.509</u> <u>certific</u> <u>ates</u></b>	<b><u>JWT</u></b>	<b><u>OAuth</u> <u>2.0 or</u> <u>OpenI</u> <u>D</u> <u>Connec</u> <u>t</u></b>	<b><u>Kerber</u> <u>os</u></b>	<b><u>SAML</u></b>	<b><u>LTPA</u></b>	<b><u>ICRX</u></b>
Web service provider	✓	✓		✓			✓	✓		✓
CICS Liberty web application	✓	✓ (“2” on page 20)	✓ (“2” on page 20)	✓	✓	✓			✓	
IPIC (non-CICS client - “4” on page 20)	✓	✓		✓						✓
CICS web support	✓	✓		✓						
CICS-MQ Bridge	✓ (“3” on page 20)	✓								
3270	✓	✓	✓				✓			
APPC	✓	✓								

Table 7. Authentication options for different ways into CICS. The authentication options are shown across the top. The types of access method are shown down the side. (continued)

	<u>Passw rds and passph rases</u>	<u>PassTic kets ("1" on page 20)</u>	<u>MFA</u>	<u>X.509 certific ates</u>	<u>JWT</u>	<u>OAuth 2.0 or OpenID Connect</u>	<u>Kerber os</u>	<u>SAML</u>	<u>LTPA</u>	<u>ICRX</u>
CICS MQ adapter	No security tokens are used. Trusted connections are used to pass user IDs.									
Db2										
EXCI										
IMS										
IPIC (CICS to CICS)										
MRO										
Node.js										

1. PassTickets can be used if there is a low transaction rate (<1 per second). Multi-use PassTickets can also be used.
2. PassTickets and MFA tokens can be used to authenticate the first time, if used with LTPA tokens.
3. Passphrases are not supported.
4. An example of a non-CICS IPIC client is CICS Transaction Gateway.

## Passwords and passphrases

Passwords are the most basic form of authentication. Passwords and passphrases are supported by most of the capabilities in CICS.

CICS supports the following types of password in its logon and authentication interfaces:

- 1-8 character uppercase passwords (not recommended)
- 1-8 character mixed case passwords
- 9-100 character mixed case password phrases (*passphrases*)

If the security manager supports mixed-case passwords, CICS passes the password to the security manager unchanged. If not, CICS converts the password to uppercase before passing it to the security manager.

To see which CICS access methods support passwords and passphrases, see [Which authentication method can I use with which access method?](#)

### Why use passwords and passphrases?

Although passwords and passphrases are widely supported in CICS, passwords are not as secure as other authentication methods.

**Recommended:** Avoid 1-8 character uppercase passwords because they are subject to dictionary attacks if users have poor password hygiene.



## PassTickets

A PassTicket is a one-time only password that is generated for a specific RACF user ID to sign on to a specific application, by using a shared secret key. A specific PassTicket can be used for authentication only once and it must be used within 10 minutes of being generated. A PassTicket can be used anywhere in z/OS where a password can be used.

To see which CICS access methods support PassTickets, see [Which authentication method can I use with which access method?](#)

### Why use PassTickets?

Using a PassTicket in place of a password means that applications do not have to store passwords (or ask users to reenter them) in order to authenticate to the destination system. Because a PassTicket is for one-time-use only, it is safer than a password because it cannot be captured and replayed.

You can configure CICS to allow an application to generate a PassTicket for the user that issues the request. This capability is useful when issuing outbound requests to run on z/OS that require basic authentication.

For more information about why you should consider PassTickets, see [The RACF PassTicket](#).

### How PassTickets work?

The *originating system* is the system where a PassTicket is generated. The *destination system* is the system that the signed-on user ID attempts to access with the PassTicket and where the PassTicket is authenticated.

The client on the originating system generates a PassTicket for the destination system by using the RACF PassTicket generator algorithm. For more information, see [Incorporating the PassTicket generator algorithm into your program in z/OS Security Server RACF Macros and Interfaces](#).

If the originating system is CICS, to create a PassTicket for the signed-on user, your application issues the **EXEC CICS REQUEST PASSTICKET**, **REQUEST ENCRYPTPTKT**, or **FEPI REQUEST PASSTICKET** command to request RACF to generate a PassTicket. The regions that can generate a PassTicket are specified with the system initialization parameter **XPTKT=YES**.

The signed-on user ID attempts to access the destination system. That system authenticates the user ID and PassTicket with RACF. Existing commands and procedures can be used for authentication.

Figure 8 on page 21 shows how one CICS region creates a PassTicket for a signed-on user to access another system.

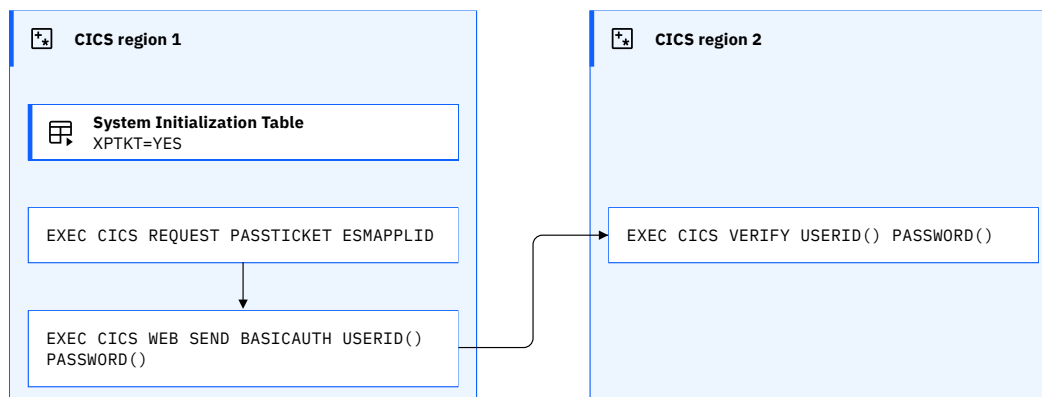


Figure 8. PassTickets

When CICS authenticates the user ID and PassTicket, it calls RACF to check whether the PassTicket supplied is for the specified user ID for that region. A check might verify that the specified user ID is connected to the specified group ID.

If the PassTicket times out because, for example, of a session failure, your application must generate another before it attempts to sign on again. Repeated failed sign-on attempts with PassTickets can result in the user ID being revoked. If a user ID is revoked, a request for a PassTicket succeeds but an attempt to sign on with that user ID and PassTicket fails.

PassTickets are not displayed when the CICS execution diagnostic facility (EDF) is used.

For information about configuring PassTickets, see [Implementing PassTickets for secure sign-on](#).

## Multi-Factor Authentication (MFA)

---

Multi-factor authentication (MFA) requires that two or more authentication factors are presented during logon in order to verify a user's identity. Each authentication factor must be from a separate category of credential types: something that you know, something that you have, and something that you are. IBM Multi-Factor Authentication for z/OS (IBM MFA) enables CICS to authenticate with multiple authentication factors.

The following types of credential are used in Multi-Factor Authentication:

**Something that you know**

For example, a password or a security question.

**Something that you have**

For example, a physical access control such as an ID badge, or a cryptographic token device.

**Something that you are**

For example, a fingerprint or other biometric data.

To see which CICS access methods support MFA, see [Which authentication method can I use with which access method?](#).

### Why use MFA?

MFA is more secure than single factor authentication because it is not compromised if one of the factors is discovered. You should use MFA whenever possible, particularly to protect access to high value data or sensitive data. MFA should also be used whenever non-repudiation is required. MFA is required for compliance with many security standards, such as PCI DSS (Payment Card Industry Data Security Standard).

### IBM Multi-Factor Authentication for z/OS (IBM MFA)

IBM Multi-Factor Authentication for z/OS (IBM MFA) provides support for authenticating with multiple authentication factors. You can configure profiles for RACF users to require authentication through IBM MFA. RACF calls IBM MFA to help make the authentication decision during logon processing. See [Multi-Factor Authentication for z/OS in z/OS Security Server RACF Security Administrator's Guide](#) for an overview of MFA.

MFA tokens are supported on CICS session-based logon interfaces: CESN, CESL, CICSplex SM WUI, and CICS Explorer.

There are two ways of using MFA:

- [“In-band authentication” on page 22](#)
- [“Out-of-band authentication ” on page 23](#)

**In-band authentication**

You generate a token by using one of the IBM MFA options and use that token directly to log on. CICS supports in-band MFA tokens if they can be entered as a single character string.

[Figure 9 on page 23](#) illustrates in-band authentication with an MFA solution, such as RSA SecurID:

1. The user logs on with a user ID and an RSA SecurID token and PIN.
2. CICS calls RACF to authenticate this information.
3. When RACF determines that the user is an MFA user, RACF calls the RACF MFA server.
4. The RACF MFA server calls RACF to retrieve the user's MFA factor details.
5. The RACF MFA server validates the user's authentication factors by calling the RSA server.
6. The MFA server passes back a return code to RACF.
7. RACF passes back the return code to CICS.
8. The user is either authenticated or rejected.

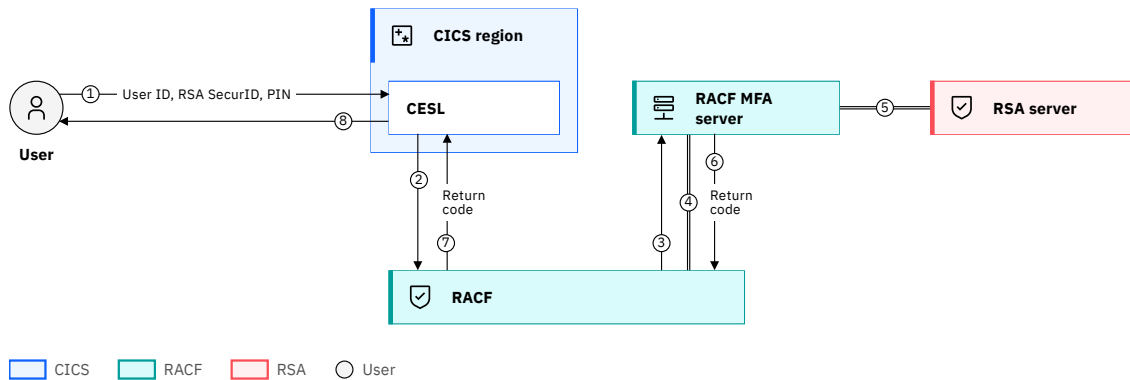


Figure 9. MFA in-band authentication

### Out-of-band authentication

Allows you to authenticate on a user-specific web page with one or more factors, possibly in a sequence to obtain a one-time-use token that you use to log in. CICS supports out-of-band MFA tokens.

Figure 10 on page 24 illustrates out-of-band authentication:

1. The user pre-authenticates through the RACF MFA web server.
2. The RACF MFA web server calls the RACF MFA server, which stores the pre-authentication record in a session cache.
3. The RACF MFA web server returns a One Time Passcode (OTP).
4. The users logs on with a user ID and the OTP
5. CICS calls RACF to authenticate this information.
6. When RACF determines that the user is an MFA user, RACF calls the RACF MFA server.
7. The RACF MFA server calls RACF to retrieve the user's MFA factor details.
8. The RACF MFA server validates the user's authentication factors by checking the session cache.
9. The MFA server passes back a return code to RACF.
10. RACF passes back the return code to CICS.
11. The user is either authenticated or rejected.

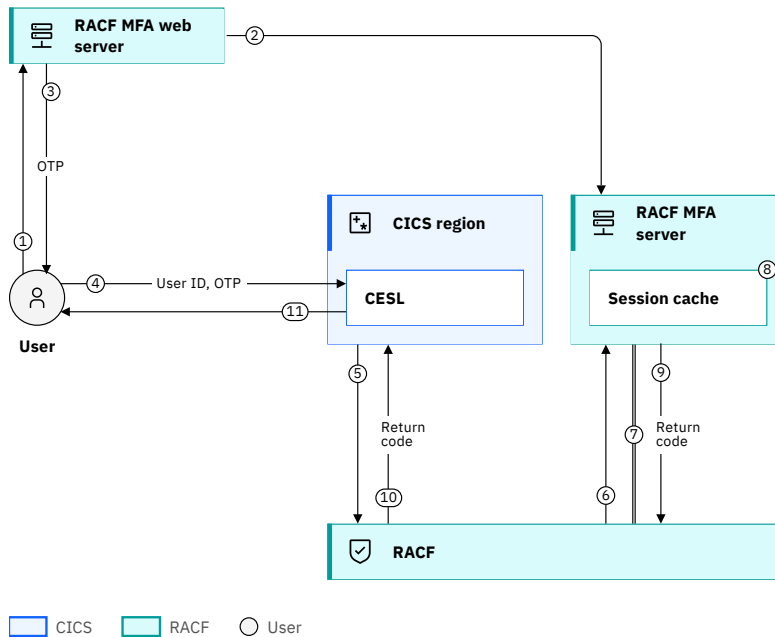


Figure 10. MFA out-of-band authentication

## ICRX (Extended Identity Context Reference)

A distributed identity can be mapped to, and associated with, a RACF user ID through Identity propagation. In CICS, the distributed identity is passed in a token that is known as an *ICRX*, term that is taken from *Extended Identity Context Reference*.

The ICRX contains the user identity information, for example, an X.500 distinguished name and associated LDAP realm name (for example, the name of an LDAP registry) that identifies the user externally. The distributed identity, a combination of the user identity and the realm, is passed with the request from the distributed application to CICS.

To see which CICS access methods support ICRX, see [Which authentication method can I use with which access method?](#)

### Related reference

[ICRX: Extended identity context reference in z/OS documentation](#)

## X.509 certificates

TLS uses *certificates* for authentication. Authentication happens at connection time and is independent of the application or the application protocol.

To see which CICS access methods support certificates, see [Which authentication method can I use with which access method?](#)

### Why use certificates?

TLS is a widely adopted security protocol that is used to secure communications over the internet. The primary use case of TLS is encrypting the communication between servers. It can also be used to establish trust between servers when enabling client authentication (sometimes referred to as *mutual authentication*).

TLS client authentication might not be practical when many clients need to connect to CICS because the creation and administration of certificates can be a significant effort, for example, when certificates expire.

For more information about TLS, see [TLS](#).

## How certificates work

TLS client authentication involves making sure that sites with which you communicate are who they claim to be. With TLS, authentication is done by an exchange of certificates, which are blocks of data in a format that is described in ITU-T standard X.509. The X.509 certificates are issued and digitally signed by an external authority, which is known as a *certificate authority*.

A certificate contains:

- Two *distinguished names*, which uniquely identify the *issuer* (the certificate authority that issued the certificate) and the *subject* (the individual or organization to whom the certificate was issued). The distinguished names contain several optional components:
  - Common name
  - Organizational unit
  - Organization
  - Locality
  - State or Province
  - Country
- A digital signature. The signature is created by the certificate authority by using the public-key cryptography technique:
  1. A secure hashing algorithm is used to create a digest of the certificate's contents.
  2. The digest is encrypted with the certificate authority's private key.

The digital signature assures the receiver that no changes are made to the certificate since it was issued:

1. The signature is decrypted with the certificate authority's public key.
  2. A new digest of the certificate's contents is made, and compared with the decrypted signature. Any discrepancy indicates that the certificate might have been altered.
- The subject's domain name. The receiver compares this name with the actual sender of the certificate.
  - The subject's public key.

Certificates are used to authenticate clients to servers, and servers to clients; the mechanism that is used is essentially the same in both cases. However, the server certificate is mandatory - that is, the server must send its certificate to the client - but the client certificate is optional: some clients might not support client certificates; others might not have installed certificates. Servers can decide whether to require client authentication for a connection.

With client certificate authentication, the client is prompted by the server to supply a certificate that the server then validates and associates with a user ID in the user registry. The identity in the certificate must be mapped to a user ID in the user registry. [Figure 11 on page 26](#) illustrates the exchange.

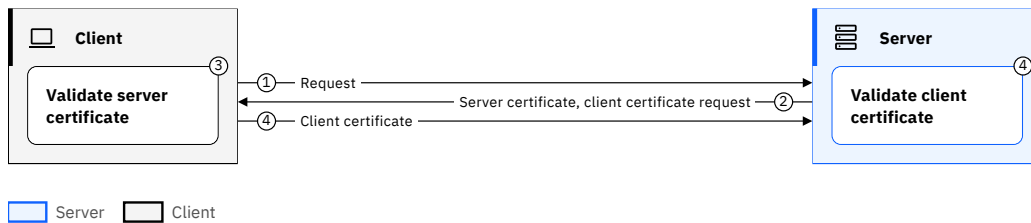


Figure 11. TLS client authentication flow

1. The client makes a request.
2. During the handshake, the server sends its certificate and other information to the client.
3. The client validates the server client.
4. If client (mutual) authentication is required, the client sends its certificate and other information to the server. And the server validates the client certificate.

A client certificate is normally signed by the private key of a certificate authority (CA). To validate the certificate, the server needs to use the public key of the certificate authority. It is also possible that the CA certificate is itself signed by another CA. Such a sequence is called a *certification path*, and the server must have access to the public keys of all the CA certificates in the path.

## Support in CICS for TLS client authentication

CICS can play the role of a server or a client in the client authentication flow. When CICS is acting as a server, the client is prompted by CICS to supply a certificate that CICS validates and associates with a user ID in RACF. When CICS is acting as a client, the server prompts CICS to supply a certificate that is validated by the server.

You can implement TLS client authentication with CICS in the following ways:

- CICS Liberty

CICS Liberty uses Java Secure Sockets Extension (JSSE) as the TLS implementation. JSSE provides a framework and Java implementation that handles the handshake negotiation, including the exchange of certificates. Certificates are stored in a keystore or RACF key ring. For more information on using TLS with CICS Liberty, see [Configuring confidentiality and integrity](#).

- CICS

CICS provides TLS support for IPIC requests, web requests, and CICS web services. Certificates are stored in a RACF key ring. For more information about using TLS with different TCP/IP clients, see [Data in motion](#).

- Application Transparent Transport Layer Security (AT-TLS)

Application Transparent Transport Layer Security (AT-TLS) is a capability of z/OS Communications Server. Certificates are stored in a RACF key ring. For more information, see [AT-TLS](#).

## JSON Web Token (JWT)

JSON Web Tokens (JWT) are an open standard, which is defined in [JSON Web Token \(JWT\) Specification RFC 7519](#). They securely represent *claims* between two parties. Claims can be related to any business process but are typically used to represent an identity and its associations. For example, a user whose identity the JWT represents belongs to an administrator role or group.

For an introduction to JWT, see [JWT \(RFC 7519\)](#).

The claims in a JWT are encoded as a JSON object and are normally digitally signed with a Message Authentication Code (MAC). The most common scenario for using a JWT is authentication. When the user

is logged in, each subsequent request includes the JWT, which allows the user to access services that are permitted by that token.

To see which CICS access methods support JWT, see [Which authentication method can I use with which access method?](#).

## Why use JWT?

Advantages of using JWTs include the following:

- They are lightweight and easy to use by client applications: for example, mobile applications.
- They are self-contained, which means that the Liberty JVM server can consume the token directly and use a claim from the token as the identity for running the request.
- They can be symmetrically signed by a shared secret by using the HMAC algorithm, or asymmetrically by using a private key.
- They have a built-in expiry mechanism.
- They can be extended to contain custom claims.
- They are widely adopted by different Single Sign-On solutions and well-known standards, such as [How it works: OpenID Connect](#).

## Anatomy of a JWT

Figure 12 on page 28 shows that a JWT consists of three parts: a header, payload, and signature.

### Header

The header typically consists of two parts: the type of the token, which is JWT, and the algorithm that is used, such as HMAC SHA256 or RSA SHA256. It is Base64Url encoded to form the first part of the JWT.

### Payload

The payload contains the claims. There is a set of registered claims, for example, `iss` (issuer), `exp` (expiration time), `sub` (subject), and `aud` (audience). These claims are not mandatory but recommended to provide a set of useful, interoperable claims. The payload can also include extra attributes that define custom claims, such as employee role. Typically, the subject claim is used to create the OpenID Connect user subject. However, the Liberty JVM server can be configured to use an alternative claim. The payload is Base64Url encoded to form the second part of the JWT.

### Signature

To create the signature part, the encoded header and encoded payload are signed by using the signature algorithm from the header. The signature is used to verify that the issuer of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

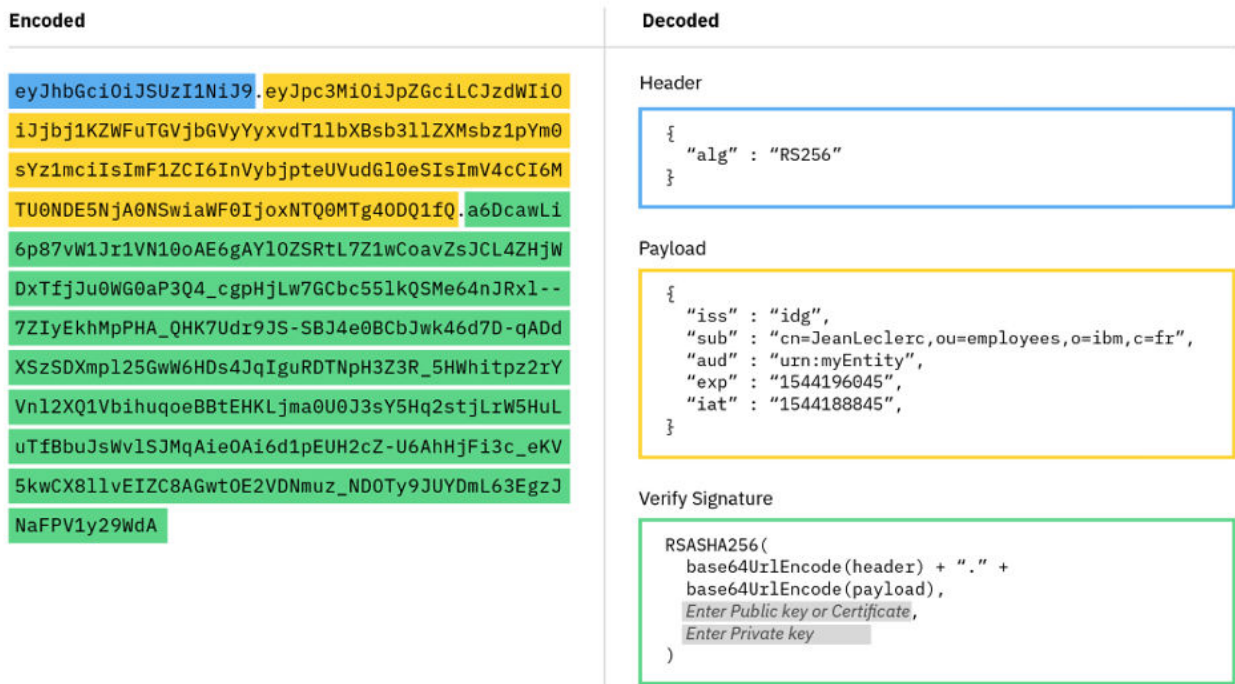


Figure 12. Anatomy of a JWT

Figure 13 on page 28 shows a typical JWT authentication flow.

1. The user logs in using credentials.
2. When the user is authenticated, a JWT is created and returned to the user.
3. When the user wants to access a protected resource, the client application sends the JWT, typically in the HTTP Authorization header.
4. The JWT is then used by the application server, such as CICS, to identify the user and allow access to the resource.

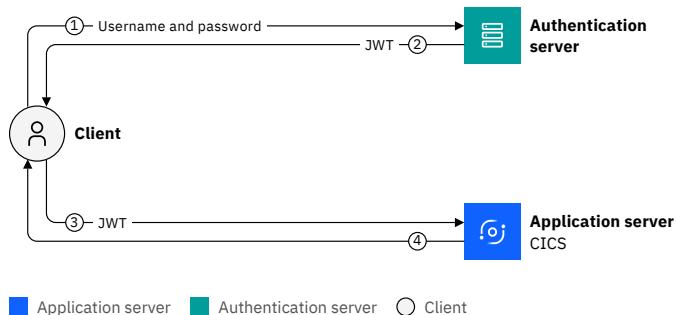


Figure 13. JWT authentication flow

## Support for JWT in CICS Liberty

A JWT can be used for identity propagation with CICS Liberty. If the subject claim contains a distributed identity, you can configure CICS Liberty to map the distributed identity to a RACF user ID. You can configure JWT authentication with a Liberty JVM server in a number of different ways, the most common of which include:

- OpenID Connect Client (OIDC) feature



CICS Liberty supports OpenID Connect 1.0. You can use the `openidConnectClient-1.0` feature to configure a Liberty JVM server to accept a JWT as an authentication token. For more information, see [How it works: OpenID Connect](#).

- JWT feature

The Liberty `jwt-1.0` feature provides a set of APIs that you can use to work with JWTs. You can use the feature to build or to consume a JWT.

- Enterprise Java Security API

The Enterprise Java Security API is an Enterprise Java standard service provider API that enables the implementation of authentication mechanisms and identity stores into Enterprise Java web applications. You can develop a custom HTTP authentication mechanism that authenticates a request that uses a JWT.

- Java Authentication Service Provider Interface for Containers (JASPIC)

JASPIC is an Enterprise Java standard service provider API that enables the implementation of authentication mechanisms into Enterprise Java web applications. You can develop a custom JASPIC authentication provider that authenticates a request that uses a JWT.

- Trust Association Interceptor (TAI)

A Trust Association Interceptor (TAI) is a WebSphere® proprietary service provider API that enables the integration of third-party security service. You can implement a TAI that inspects HTTP requests for a specific security token, for example, a JWT, and validates it. The CICS Liberty server needs to be configured to use the implemented TAI.

**Recommended:** When you use nonencrypted JWTs, it is highly recommended that you use HTTPS to transport requests that contain a JWT.

## Support for JWT in CICS

CICS supports only JWTs created by RACF. With this capability, you can convert basic authentication credentials of a user to a JWT that is then used as a time-limited secure session token. You can validate this secure token on subsequent requests by using the CICS API command, **VERIFY TOKEN**. This support can also be used to convert an MFA token to a JWT to support the use of MFA tokens on stateless requests that cache credentials. Passwords, passphrases, MFA tokens, and PassTickets are all supported credentials.

If you require HTTPS or SOAP requests to accept a JWT created by RACF, you need to write custom headers.

The following diagram shows a scenario where basic authentication credentials are converted to a JWT for subsequent requests, where `user` is the user ID and `pw` is the password.

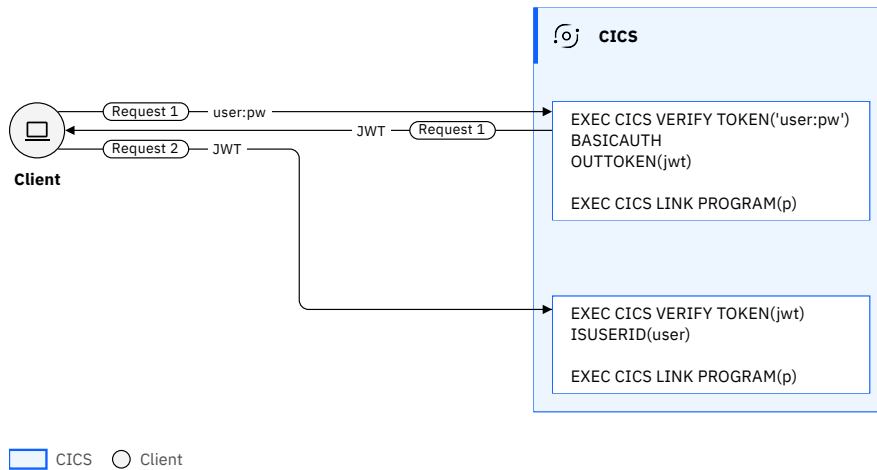


Figure 14. Converting basic authentication credentials to a JWT

## Configuring RACF for JWT

CICS Transaction Server for z/OS provides support for JSON Web Tokens (JWTs) using RACF. For information, see [How it works: JSON Web Token \(JWT\)](#).

This capability requires RACF APAR OA55926 and SAF APAR OA55927.

For a CICS region to support JWTs, you must create profiles in the IDTDATA class. Ensure to specify the IDTPARMS SIGTOKEN option because CICS supports only signed JWTs. The IDTDATA class must be active and RACLISTed.

Figure 15 on page 30 shows example RDEFINE statements to create such profiles. For details of the commands, see [Security Server RACF Command Language Reference](#).

```

SETROPTS CLASSACT(IDTDATA)
RDEFINE IDTDATA JWT.applid.userid.SAF IDTPARMS(SIGTOKEN(icsftoken))
  
```

### **applid**

Specify the APPLID of the CICS region. If all CICS regions are supported, specify an asterisk \*.

### **userid**

Specify the CICS task user ID that is allowed to process JWTs. If all user IDs are supported, specify an asterisk \*.

Figure 15. Example RDEFINE statements to create profiles for JWT support

## OAuth 2.0

OAuth 2.0 is an open standard for delegated authorization. The OAuth 2.0 authorization framework enables a user, the resource owner, to grant a third-party client application access to information that is stored on a HTTP resource server without sharing the resource owner's credentials.

The following roles are defined in the OAuth 2.0 specification.

### **Client**

The application that requests access to a protected resource on behalf of the resource owner.

### **Resource owner**

The entity that grants access to the protected resource. It's normally an end user.

### **Authorization server**

The server that authenticates the resource owner and issues the access tokens.

## Resource server

The server that hosts the protected resources

OAuth 2.0 uses access tokens to provide authorization. An access token represents the authorization to access a resource on behalf of the resource owner. OAuth 2.0 defines four different flows that can be used to provide an access token. These flows are called *grant types*. Each grant type is designed for a specific use case.

- Authorization code flow
- Client credentials flow
- Resource owner password flow
- Implicit flow

The following diagram shows a typical OAuth 2.0 flow for a grant type of *authorization code*. For further details on using the authorization code flow in CICS Liberty, see [Configuring OAuth 2.0 for authorization](#).

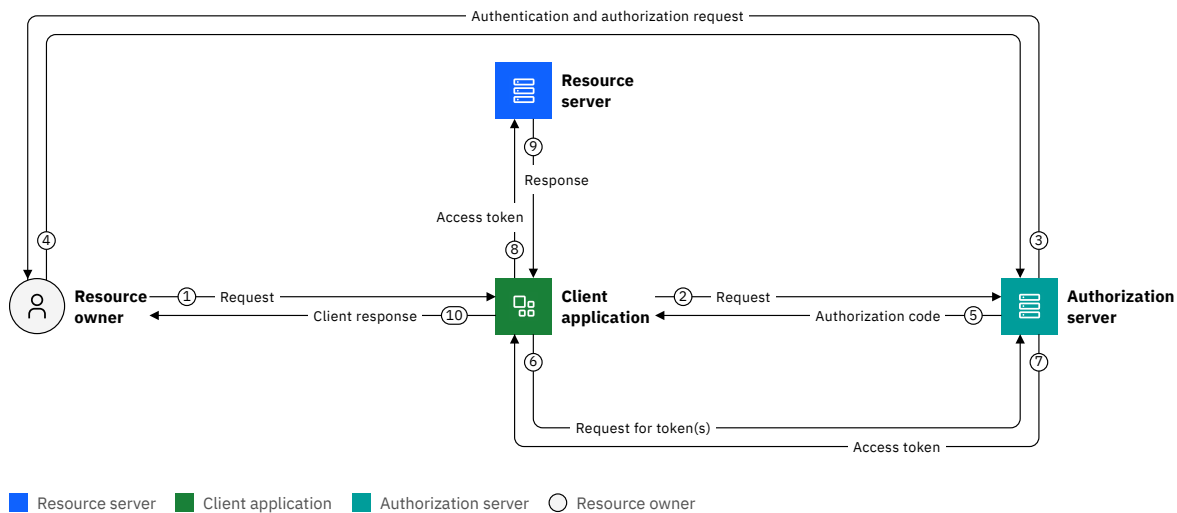


Figure 16. OAuth 2.0 flow

1. The resource owner sends a request to the client application.
2. The client application redirects the request to the authorization server for authorization.
3. The authorization server sends an authentication and authorization request to the resource owner.
4. The resource owner authenticates and authorizes the client application to access the resource.
5. The authorization server sends an authorization code to the client application.
6. The client application sends a request to the authorization server to exchange the authorization code for an access token.
7. The authorization server sends the access token to the client application.
8. The client application makes the request to the resource server with the access token, which is used to verify the authorization granted to the client application to access the resource.
9. The response is sent from the resource server to the client application.
10. The response is sent from the client application to the resource owner.

For details on the standards, see [RFC 6749](#) and [RFC 6750](#).

## Related information

[Which authentication method can I use with which access method?](#)

## OpenID Connect

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol in which the identity of the user is also transmitted to client applications. OpenID Connect allows applications to verify the identity of the user based on the authentication that is performed by an OpenID Connect Provider and to obtain basic profile information about the user in an interoperable and REST-like way. The client application retrieves an ID token (in the form of a [JWT](#)) from the OpenID Connect Provider that is then used to access a resource on behalf of the user.

For more information about OpenID Connect, see its [specification](#).

To see which CICS access methods support OpenID Connect, see [Which authentication method can I use with which access method?](#)

### Why use OpenID Connect?

OpenID Connect 1.0 is widely used as an open identity protocol. Using OpenID Connect with CICS allows you to use the same authentication and identification mechanisms that you use with other platforms and application servers. Configuring the `openidConnectClient-1.0` feature with CICS Liberty allows you to authenticate requests by using a [JWT](#) without needing to write custom authentication code.

### How OpenID Connect works

OAuth 2.0, which underpins OpenID Connect, allows a user to securely share access to information with a third party. It simplifies the user experience by avoiding the need to manage many different credentials for multiple websites. OAuth 2.0 aims to enable one website to request user information from another site, while allowing the user to control access to the information.

OAuth 2.0 provides different types of *grant*, or method, for a client application to acquire an access token. The grants cover different use cases; see the [OAuth 2.0 specification](#) for details. [Figure 17 on page 32](#) show the steps in a typical OpenID Connect flow for a grant type of *authorization code*.

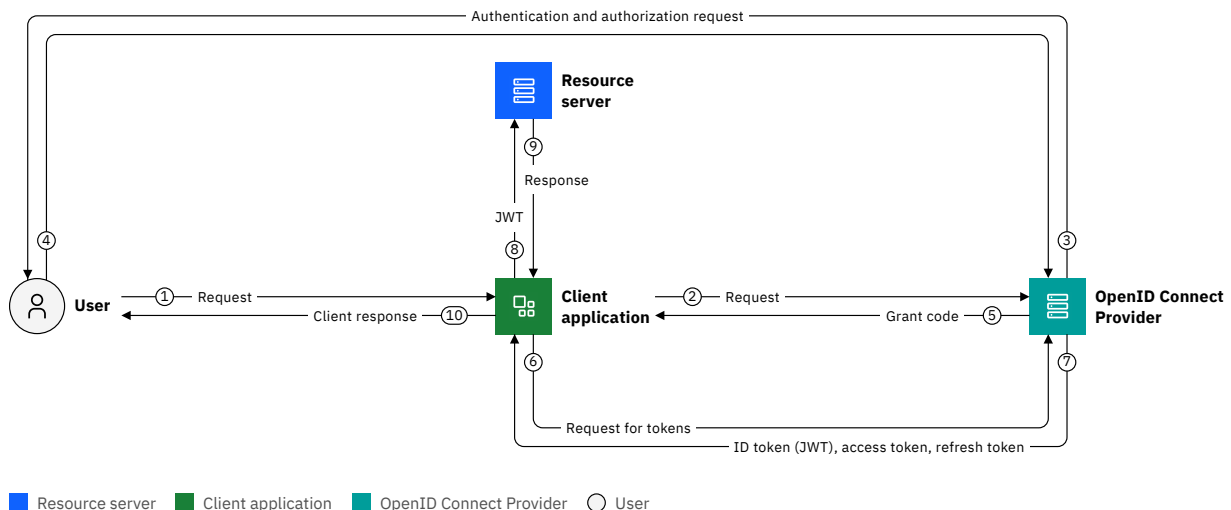


Figure 17. OpenID Connect flow

1. The user makes a request to the client application.
2. The client application redirects the request to the OpenID Connect Provider (OP) for authorization.
3. The OP sends an authentication and authorization request to the user.
4. The user authenticates and authorizes the client application to access the resource.
5. The OP sends a grant code to the client application.

6. The client application sends a request to the OP to exchange the grant code for an ID token (in the form of a JWT), an access token, and a refresh token.
7. The OP sends the ID token (JWT), access token, and refresh token to the client application.
8. The client application makes the request to the resource server with the JWT, which is used to authenticate the user and to authorize access to the resource.
9. The response is sent from the resource server to the client application.
10. The response is sent from the client application to the user.

## Support in CICS Liberty for OpenID Connect

CICS Liberty supports OpenID Connect 1.0 and can play a role as a client, OpenID Connect Provider, or resource server. For example, you can use the `openidConnectClient-1.0` feature to configure a Liberty JVM server to accept a JWT as an authentication token. For instructions, see [Configuring JWT authentication](#).

## Kerberos

---

Kerberos is a network authentication protocol that is designed to provide strong authentication for client or server applications by using secret-key cryptography. The Kerberos network authentication protocol assumes that services and workstations communicate over an insecure network. A user authenticates to an authentication server. The user can then request service tickets, which can be used by that user on a specific application on a server.

To see which CICS access methods support Kerberos, see [Which authentication method can I use with which access method?](#)

### Why use Kerberos?

Kerberos avoids the need for passwords to be flowed to authenticate to multiple servers over an insecure network. This feature can help reduce the need to manage multiple passwords. It allows clients and servers to do either one-way authentication (the server authenticates the client), or two-way (mutual) authentication, where the client also authenticates the server.

### How Kerberos works

The Kerberos system consists of three components: a client, a server, and a trusted third party, which is also known as a Key Distribution Center (KDC). The KDC interacts with both a client and server to accept the client's request, authenticate its identity, and issue tickets to the service. The domain that is served by a single KDC is referred to as a realm. A principal is used to identify each client and server in a realm. The principal name is uniquely assigned for all clients and servers by the Kerberos administrator. All principals must be known to the KDC.

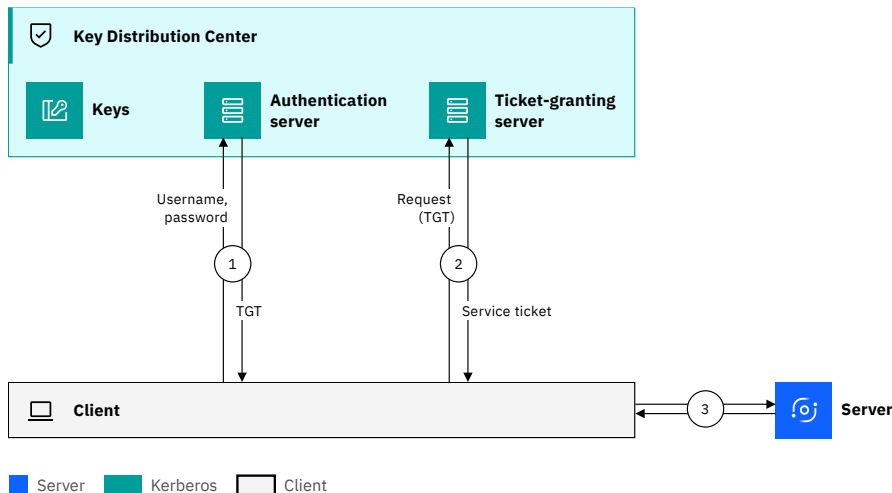


Figure 18. How Kerberos works

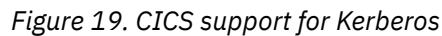
1. The first-phase exchange takes place between a client and the authentication server. In this phase, the authentication server authenticates the user (for example, by validating the user ID and password). After a successful login, the authentication server obtains the user's secret keys and returns a ticket-granting ticket (TGT) to the client.
2. On receiving the TGT, the client sends a request (containing the TGT) for a service ticket to the ticket-granting server (TGS). The TGS authenticates the TGT and then returns a service ticket to the client.
3. The service ticket allows the client to communicate with the server that is providing a service that the client wants to use. The server can verify the client without contacting the KDC by using the service ticket. An extension to the protocol is *mutual authentication*. When this is configured, additional information flows back to the client in this step to authenticate the server.

## Support in CICS for Kerberos

CICS supports Kerberos through RACF. Support is based on Kerberos Version 5 as defined in [RFC 4120 – The Kerberos Network Authentication Service \(V5\)](#) and Generic Security Services (GSS). To use Kerberos with RACF, use the Network Authentication Service component of the z/OS Integrated Security Services base element. For more information, see [z/OS Integrated Security Services Network Authentication Service Administration](#) in the z/OS product documentation.

CICS can verify a Kerberos token by configuring a web service provider pipeline or by using the CICS API command, [VERIFY TOKEN](#).

**Stabilized feature:** CICS web services support for pipeline configuration for WS-Security infrastructure is stabilized. See also [Stabilization notices](#).



You must configure an external security manager, such as RACF, to enable support for Kerberos.

## Before you begin

- You must set up a Kerberos environment for z/OS Security Server RACF by using the z/OS Integrated Security Services Network Authentication Service. For information about RACF, Kerberos and z/OS Integrated Security Services Network Authentication Service, see [RACF and z/OS Integrated Security Services Network Authentication Service](#).
- The configuration `/etc/skrb/krb5.conf` defines the local realm for the LPAR or sysplex.
- The SKRBKDC started task must be running. For more information, see [Setting up a Kerberos Key Distribution Center in z/OS Network File System Guide and Reference](#).

## Procedure

1. Enable RACF protection for Kerberos. Activate RACF protection for the KERBLINK class by using the RACF **SETROPTS** command:

For more information about the **SETROPTS** command, see [z/OS Security Server RACF Command Language Reference](#).

2. To set up a CICS region to use Kerberos, define a service principal name and associate it with a user ID, as follows:
  - a) Specify the **KERBEROSUSER** system initialization parameter.

This parameter enables support for the Kerberos service in the region and specifies a user ID to be associated with the Kerberos service principal. You should use a non-protected user ID.

- b) Use the **ALTUSER** command to associate the service principal name with the user ID you specify in **KERBEROSUSER**.

```
ALTUSER user_id KERB(KERBNAME(service_principal))
```

3. Associate a RACF user ID with the client principal by using the **ALTUSER** command:

```
ALTUSER userid PASSWORD(password) NOEXPIRED KERB(KERBNAME(client_principal))
```

Alternatively, you can associate a default user ID with all unassociated principals by using a command as follows:

```
RDEFINE KERBLINK /.../realm APPLDATA('userid')
```

where *userid* is the local user ID to be associated with all unmapped principals for the realm *realm*.

4. In order for this to be activated, a Kerberos key must be created. This is done automatically when the user next changes their password.

## Configuring CICS web services for Kerberos

To configure a provider pipeline to implement Kerberos authentication, you must add a security handler to your pipeline configuration file.

### Before you begin

You must configure an external security manager, such as RACF, to enable support for Kerberos. For more information, see [Configuring RACF for Kerberos](#).

You must identify or create the pipeline configuration file to add the configuration information for Kerberos.

### Procedure

1. Add a `<wsse_handler>` element to your pipeline. The handler must be included in the `<service_handler_list>` element. Code the following elements:

```
<wsse_handler>
  <dfhwsse_configuration version="1">
  </dfhwsse_configuration>
</wsse_handler>
```

The `<dfhwsse_configuration>` element is a container for the other elements in the configuration.

2. Code an `<authentication>` element.
  - a) Code the **trust** attribute to specify `trust="basic"`.
  - b) Code the **mode** attribute to specify `mode="basic-kerberos"`.
  - c) Optional: Code an empty `<suppress/>` element.

If you do not specify this element, the work runs under the user ID associated with the token.

### Results

The CICS provider pipeline is configured for Kerberos authentication. Inbound web service requests received by the pipeline must include a valid Kerberos token. If they do not, the request is rejected with the appropriate SOAP fault. Depending on pipeline configuration options, the target application runs under the user ID associated with the token.



## Example

The following example shows how you might configure the provider pipeline:

```
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="basic" mode="basic-kerberos"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
</apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

## Verifying a Kerberos token

You can write a custom security handler to verify a Kerberos token, or you can write your own application to verify the token.

### Before you begin

In the following steps, you can extract the RACF user ID of the Kerberos principal, by using the **ISUSERID** option of the **EXEC CICS VERIFY TOKEN** command. To do this, you must first define an association between the user ID and the principal. Use the RACF **RACLINK** command to set up the association. For more information, see [Defining User ID Associations in z/OS Security Server RACF Security Administrator's Guide](#).

### Procedure

Use one of the following techniques, according to your requirements:

- Write a security handler that uses the **VERIFY TOKEN** command, as described in [Writing a custom security handler](#). If you want to run under the user ID of the Kerberos principal that is associated with the token, use the **ISUSERID** option of the **VERIFY TOKEN** command.
- Write your own front-end security program. Such a program might extract the Kerberos token from an HTTP header or an IBM MQ message and then issue the **VERIFY TOKEN** command. If you want to run under the user ID of the Kerberos principal that is associated with the token, use the **ISUSERID** option of the **VERIFY TOKEN** command to obtain the user ID. The new request can then be started with that user ID.

## Using a Kerberos security token in a 3270 emulator sign-on

The use of Kerberos provides stronger security because passwords are not required to flow over the network.

The process is described as follows:

1. The Client terminal emulator applies to a Kerberos authentication server to obtain a Kerberos token.
2. The Kerberos token is returned to the Client terminal emulator, and the content is encoded in Base64 format.
3. The token is then forwarded in a message to the CICS server, where a sign-on transaction receives the Base64 encoded Kerberos token and issues the **SIGNON TOKEN** command.
4. The RACF Kerberos registry validates the Kerberos token and returns the associated RACF USERID to CICS. This USERID is associated with the terminal session for subsequent tasks.

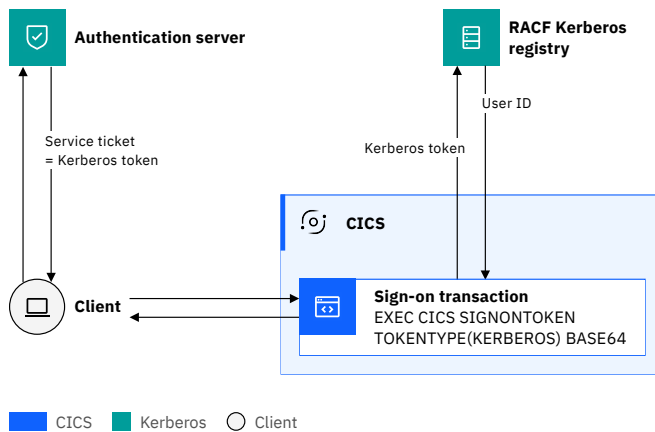


Figure 20. Flow of requests between the Client terminal emulator, the authentication server, and CICS TS

**Note:** Logon data cannot be used to send the Kerberos token since it is limited to 255 characters.

## Lightweight Third-Party Authentication (LTPA)

The Lightweight Third Party Authentication (LTPA) is an IBM single-sign on technology that reduces the number of times a user's credentials are checked against a user registry. When web users access application servers that use LTPA, they can reuse their logged-in credentials across different applications or servers. When a new authentication request occurs, the user's credentials, such as user ID and password, are authenticated as normal, but in response the server returns a signed authentication token to the requester.

To see which CICS access methods support LTPA, see [Which authentication method can I use with which access method?](#).

### Why use LTPA?

LTPA enables web users to reuse their logged-in credentials across different application or servers. LTPA can be used to improve the performance of security authentication and can simplify the authentication process in systems that use components that are spread across multiple servers. It can also be used in the deployment of Multi-Factor Authentication (MFA) solutions, by avoiding the need to request security factors for every request.

### How LTPA works

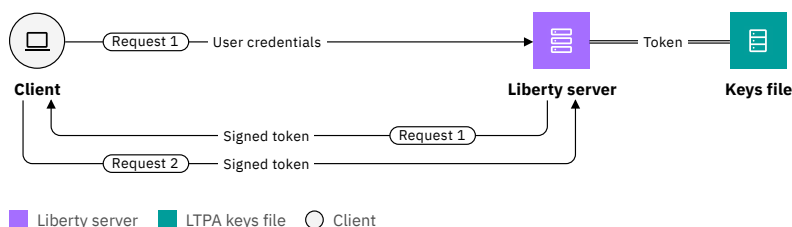


Figure 21. Creating and validating an LTPA token

When a user passes authentication on a Liberty server, a signed token is generated by using the LTPA protocol and transported to the web browser in a cookie called `1tpaToken2`. The token contains user

information and an expiration time, and is signed by keys that are owned by the server. An LTPA token is a binary security token. Liberty supports the LTPA Version 2.

After receiving an LTPA token, if the user then accesses an application either on the same server or another server that is a member of the same authentication realm, the user is automatically authenticated by using the LTPA token that is flowed by the browser in a cookie. This scenario is called a *Single-Sign-On (SSO)* environment.

## Support in CICS Liberty for LTPA

LTPA tokens are supported for use with all Liberty authentication mechanisms, including HTTP basic authentication, form login, and TLS client certificates. They are enabled by default and the Liberty server manages the creation and expiration of the certificates in an underlying key file. The default LTPA key file is located in `${server.output.dir}/resources/security/ltpa.keys` and can be modified by using the `<webAppSecurity>` element in `server.xml`.

LTPA is required by the Liberty CMCI server. For information, see [How the CMCI JVM server authenticates clients](#).

You can set up Liberty to allow the sharing of LTPA tokens among multiple servers. HTTP client users can authenticate once and have access to other applications on Liberty servers that share the same LTPA keys. If HTTP endpoints are shared across cloned CICS regions that use a TCP/IP load balancer, then the Liberty servers must share a common LTPA keys file.

An LTPA token has a fixed lifetime. It cannot be extended or renewed, even if a user is active in a session. Upon timeout, the user is logged out and must provide login credentials again to get a new token. The expiration time of the LTPA token is configurable. For instructions, see [Configuring LTPA in Liberty](#).

For more information, see [Customizing SSO configuration using LTPA cookies in Liberty](#).

## SAML

---

Security Assertion Markup Language (SAML) is an XML-based framework for describing and exchanging security information between multiple service partners. This security information is expressed in the form of SAML *assertions* that can be trusted by applications that work across security domain boundaries. The OASIS SAML standard defines the SAML syntax and the rules for defining and by using SAML assertions.

**Stabilized feature:** Support for SAML using the CICS Security Token Service is stabilized. See also [Stabilization notices](#).

To see which CICS access methods support SAML, see [Which authentication method can I use with which access method?](#).

### Why use SAML?

SAML is a commonly used single sign-on (SSO) standard. The SAML framework is used to provide a common source of user role or authority-based security information that can be securely communicated between Service Providers. This concept is also known as Federated Identity. SAML uses Public Key Infrastructure cryptography to protect these asserted identities. SAML is a framework that combines both distributed authentication and authorization.

### How SAML works

An assertion is a collection of one or more statements about a principal that are made by a SAML authority. These assertion statements can be of three types:

#### Authentication

A statement that a specified subject is authenticated, with details of the means of authentication and the time it took place.

#### Attribute

A statement that a specified subject has the specified attributes.

## Authorization decision

A statement that a request to allow the specified subject to access a specified resource is granted or denied.

A user or principal authenticates with an Identity Provider or Security Token Service, which provides a signed SAML token that contains assertions to define the principal's role and authority to a Service Provider.

## Support in CICS for SAML

CICS supports the SAML Core1.1 and SAML Core2.0 standards. It does not support the protocols that are described in those standards. CICS supports SAML in a number of ways:

- As a web service provider, by configuring the WS-Security element of the pipeline
- Through an API, which uses a channel-based program, provided for user-written solutions.

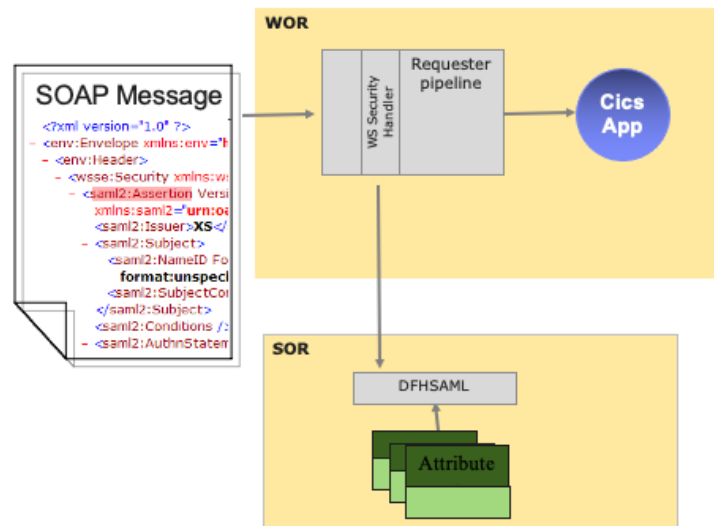


Figure 22. CICS support for SAML

The diagram shows the flow of a SAML assertion in a CICS environment that is configured to perform single sign-on (SSO) through web services. The assertion is augmented at some point in the flow through the addition of new attributes. The provider and requester pipelines are configured to support SAML and to use the transaction channel.

For more information about setting up SAML, see [Configuring CICS for SAML](#).

## Configuring CICS for SAML

CICS supports the use of Security Assertion Markup Language (SAML) for describing and exchanging security information.

### Before you begin

CICS supports the SAML Core1.1 and SAML Core2.0 standards. It does not support the protocols that are described in those standards.

You can configure provider and requester pipelines to use SAML tokens, but you must first deploy the CICS Security Token Service (STS). You must identify the regions where you want to deploy the

CICS Security Token Service (STS). Install the STS in regions without any application code. If you have application code in the region where you will be validating your SAML token, define the STS remotely. You might also choose to define the region remotely if you prefer to separate regions that run Java code from other regions. Another reason for having a separate region for the STS is that you could define that region with its own keyring, which contains only those certificates that are required for signature validation and signing SAML tokens.

## About this task

CICS provides a linkable interface called DFHSAML. The interface allows CICS web services pipelines and applications to validate and extract information from SAML assertions. CICS support for SAML requires a JVM server that is installed and configured on your system.

Java 11Java 17 Running a SAML JVM server with Java 11 or Java 17 is not supported.

## Procedure

1. Create a JVM server profile for the JVM server.

You can copy the appropriate supplied profile, DFHJVMST, from the installation directory to the directory that is specified by the **JVMPROFILEDIR** system initialization parameter.

2. Install CSD group DFHSAML in the chosen configuration:

- a) Install DFHSAML in the region that is chosen to run the STS.
- b) If you want to use SAML remotely, define a remote program definition for DFHSAML pointing to the region that runs the STS.

**Note:** If you are using your own JVM server definition, copy DFHSAML, customize this group, and install the customized group instead of the DFHSAML group. The new group must point to your own JVM server definition. All programs that call the security token extensions support must create DFHSAML JVMSERVER containers with the name of their JVM server.

## Results

CICS is configured for SAML.

## What to do next

You can validate your configuration, as described in [“Validating your configuration of CICS for SAML” on page 41](#).

## Validating your configuration of CICS for SAML

A sample is provided, which you can use to verify that CICS is configured correctly for SAML. Two programs are provided, which can be compiled and then invoked through a transaction.

## Before you begin

You must configure your JVM server, as described in [“Configuring CICS for SAML” on page 40](#).

## About this task

A sample is provided in CSD group DFH\$SAML, which contains a program definition for sample programs, a transaction, and a template. You can use this sample to validate your configuration. When you compile and deploy the sample application, it provides an example SAML token assertion to be processed by the CICS security token extensions. The application is started by a CICS transaction.

## Procedure

1. Optional: If you customized and installed a JVM server with a name other than DFHXSTS, update program DFH0XST2 to reflect the new server name.

2. Compile the programs DFH0XST1 and DFH0XST2, which are in the samples library, SDFHSAMP. For information about compiling COBOL programs, see [Batch compilation for COBOL programs](#).
3. Install the group DFH\$SAML in a region that calls the DFHSAML program.
4. Run transaction XST1.

## Results

If the sample transaction XST1 runs successfully, SAML support is configured correctly.

The sample outputs the parsed containers into TSQ DFH0XSTO.

To look at these containers use **CEBR DFH0XSTO**.

If the installation validation is not successful, the DFHSAML-RESPONSE container contains a return code that indicates the reason. For more information about container response codes, see [SAML support containers](#).

If an abend code is returned read the sample for further information.

## What to do next

- You can replace the sample SAML token with your own. Create and install a DOCTEMPLATE resource definition that names the file that contains your SAML token. Specify this DOCTEMPLATE 48-byte TEMPLATENAME after the transaction identifier when you run the sample:

```
XST1 templatename
```

If no *templatename* is specified, the default TEMPLATENAME of DFH0XSTI is used.

- If you want to use signature validation, update program DFH0XST2. For more information, see the comments within that program.

## Configuring a provider pipeline to use SAML tokens

Configure a provider pipeline that enables the validation and extraction of SAML token assertions and make them available to the CICS application.

### Before you begin

If you are adding SAML support to an existing pipeline, identify the pipeline configuration file. If you are creating a new pipeline, create a new pipeline configuration file. CICS provides two sample configuration files, `samlprovider.xml` and `propagatesamlprovider.xml`. The latter contains the **tran\_channel** attribute.

### Procedure

1. Code an `<sts_authentication>` element in your pipeline configuration file.

This element specifies that a Security Token Service (STS) is used for authentication and determines the type of request that is sent. Do not code an `<authentication>` element.

  - a) Code the **action="validate"** attribute to specify that the STS verifies a security token.

If you do not specify this attribute, CICS assumes that the action is to request an identity token.
  - b) Optional: Code the **token\_signature="ignored"** attribute to specify that CICS ignores the signature.

If you do not specify this attribute, the default value is `required`, which means that a valid signature must be supplied.
  - c) Optional: Code the **extract="no"** attribute to specify that the elements of the SAML token are not put into containers.

If you do not specify this attribute, the default value is `yes`, which means that CICS creates containers with the main elements of the SAML token. For full details of these containers, see [SAML support containers](#).

- d) Optional: Code the **tran\_channel="yes"** attribute to specify that the SAML assertions are copied into containers in the DFHTRANSACTION channel to allow propagation of SAML information through a CICS application.

If you do not specify this attribute, the default value is no, which means that the assertions are made available in containers in the channel that is passed from the pipeline.

- e) Within the <sts\_authentication> element, code an <auth\_token\_type> element.

Within the <auth\_token\_type> element, code the following elements:

**<namespace>**

Set the content of this element to either urn:oasis:names:tc:SAML:1.0:assertion or urn:oasis:names:tc:SAML:2.0:assertion, depending on the version of SAML you are using.

**<element>**

Set the content of this element to Assertion.

2. Code an <sts\_endpoint> element.

- a) In the <sts\_endpoint> element, code an <endpoint> element.

This element contains a URI that points to the location of the Security Token Service (STS) on the network.

To use the CICS Security Token Service to process SAML tokens, set the endpoint to cics://PROGRAM/DFHSAML.

The CICS Security Token Service is called by the security handler. If **extract="yes"** is configured in the <sts\_authentication> element, containers prefixed with "DFHSAML" are copied to the pipeline channel and are therefore available to later pipeline handlers and the target application.

- b) Optional: In the <sts\_endpoint> element, code a <jvmserver> element.

This element identifies the server on which to run. If this element is not included, the default sample resource JVM server DFHXSTS is assumed.

## Example

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <sts_authentication action="validate" token_signature="required"
            extract="yes" tran_channel="yes">
            <auth_token_type>
              <namespace>urn:oasis:names:tc:SAML:2.0:assertion</namespace>
              <element>Assertion</element>
            </auth_token_type>
          </sts_authentication>
          <sts_endpoint>
            <endpoint>cics://PROGRAM/DFHSAML</endpoint>
            <jvmserver>DFHXSTS</jvmserver>
          </sts_endpoint>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

## Configuring a requester pipeline to use SAML tokens

Configure a requester pipeline to insert SAML tokens in the SOAP header of outbound messages.

### Before you begin

If you are adding SAML support to an existing pipeline, identify the pipeline configuration file. If you are creating a new pipeline, create a new pipeline configuration file. CICS provides two sample configuration files, `samlrequester.xml` and `propagatesamlrequester.xml`. The latter contains the **tran\_channel** attribute.

### About this task

In this task, you configure a requester pipeline to automatically attach SAML tokens to the outbound SOAP messages. CICS allows only previously validated tokens to be sent out in a SOAP message. The validated token is stored in the DFHSAML-OUTTOKEN container and the token inserted in the SOAP header is taken from this container.

### Procedure

1. Code an `<sts_authentication>` element in your pipeline configuration file.

This element specifies that a Security Token Service (STS) is used for authentication and determines the type of request that is sent. Do not code an `<authentication>` element.

- a) Optional: Code the **tran\_channel="yes"** attribute to specify that the contents of the DFHTRANSACTION channel's DFHSAML-OUTTOKEN container are used as the SAML token for the request.

If you do not specify this attribute, the default value is no, which means that the SAML token is taken from the DFHSAML-OUTTOKEN container in the channel that is passed to the SOAP pipeline.

- b) Within the `<sts_authentication>` element, code an `<auth_token_type>` element.

Within the `<auth_token_type>` element, code the following elements:

#### **<namespace>**

Set the content of this element to either `urn:oasis:names:tc:SAML:1.0:assertion` or `urn:oasis:names:tc:SAML:2.0:assertion`, depending on the version of SAML you are using.

#### **<element>**

Set the content of this element to Assertion.

2. Code an `<sts_endpoint>` element.

- a) In the `<sts_endpoint>` element, code an `<endpoint>` element.

Set the value of the endpoint to `cics://PROGRAM/DFHSAML`.

- b) Optional: In the `<sts_endpoint>` element, code a `<jvmserver>` element.

This element identifies the server on which to run. If this element is not included, the default sample resource JVM server DFHXSTS is assumed.

### Example

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <sts_authentication tran_channel="yes">
            <auth_token_type>
              <namespace>urn:oasis:names:tc:SAML:2.0:assertion</namespace>
              <element>Assertion</element>
            </auth_token_type>
          </sts_authentication>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
  </service>
  <sts_endpoint>
```



```

        <endpoint>cics://PROGRAM/DFHSAML</endpoint>
        <jvmserver>DFHXSTS</jvmserver>
      </sts_endpoint>
    </dfhwsse_configuration>
  </wsse_handler>
</service_handler_list>
</service>
<service_parameter_list/>
</requester_pipeline>

```

## Configuring the CICS Security Token Service

To control the behavior of the CICS STS, update your STS configuration file and perhaps your `java.policy` file and system initialization parameters.

### About this task

CICS uses a file referred to as the *STS configuration file* to control the behavior of the CICS STS. By default, CICS uses a file called `sts.xml` in the directory that is identified by the **JVMPROFILEDIR** SIT parameter. If that file does not exist, CICS uses the file specified by the JVM property **com.ibm.cics.sts.config** in the JVM server profile. For example: `-Dcom.ibm.cics.sts.config=/var/security/sts/sts-config.xml`. A sample file `sts-config.xml` is provided.

### Procedure

1. Update the STS configuration file according to your requirements.
  - a) If you want to update the issuer of the SAML token when you add attributes to the token, specify an *issuer*.
    - i) Code an `<issuer>` element in your STS configuration file.
    - ii) Within the `<issuer>` element, code a `<format>` element.
    - iii) Within the `<issuer>` element, code a `<uri>` element.
  - b) To allow for any time difference in system clocks on different computers, you can specify a tolerance value or *clock skew*.
 

If you find that SAML tokens are regularly being rejected as expired, or as not yet valid, it might be caused by a discrepancy in clock time between the issuing and receiving systems. Setting a skew time can prevent SAML tokens from appearing to arrive before they are issued, or arriving already expired when they have, in reality, recently been issued. However, if you set too large a value, you might accept tokens that have genuinely expired.

    - i) Code a `<clock_skew>` element in your STS configuration file. The value is set in milliseconds, so to set a time of 90 seconds, code `<clock_skew>90000</clock_skew>`.
  - c) If you intend to re-sign a SAML token, specify the certificate label.
    - i) Code a `<signature>` element in your STS configuration file. Set the `hash_algorithm` attribute to `sha-1` or `sha-2` according to the hash algorithm you want to use.
 

**Note:** The hash algorithm `sha-3` is not supported.
    - ii) Optional: Within the `<signature>` element, code a `<certificate>` element.
    - iii) Optional: Within the `<certificate>` element, code a `<label>` element. Set it to the value of the RACF certificate label.
  - d) If you intend to verify signatures or to re-sign SAML tokens, you can specify the keystore type.
    - i) Code a `<keystore>` element in the STS configuration file.
    - ii) Within the `<keystore>` element, code a `<type>` element. If you want to use cryptographic hardware to validate signatures, give it the value `JCECCARACFKS`. Otherwise, use the value `JCERACFKS`, which is the default value.

2. Optional: If you intend to verify signatures or to re-sign SAML tokens, update the `java.policy` file and set the **KEYRING** SIT parameter.

a) Update the `java.policy` file to give the CICS user permission to alter the list of Java security providers.

A sample file, `sts-java.policy`, is provided.

i) To use the sample file, edit it to change `/&USSHOME/` to the appropriate value for your installation, and set the JVM property **java.security.policy** in the JVM server profile to specify its path and file name.

ii) If you use an existing `java.policy` file, update it to include the following rule:

```
// All permissions granted to CICS codesource protection domain
grant codeBase "file:///&USSHOME/---" {
  permission java.security.AllPermission;
};
```

where `USSHOME` is the name and path of the root directory for CICS Transaction Server files on z/OS UNIX.

b) Set the SIT parameter **KEYRING** to specify the RACF keyring that contains the set of certificates you want to use.

### ***The STS configuration file***

The STS configuration file specifies various aspects of the CICS Security Token Service (STS).

CICS uses a file called `sts.xml` in the directory that is identified by the **JVMPROFILEDIR** SIT parameter. If that file does not exist, CICS uses the file that is specified by the JVM property **com.ibm.cics.sts.config** in the JVM server profile. For example: `-Dcom.ibm.cics.sts.config=/var/security/sts/sts-config.xml`. A sample STS configuration file, `sts-config.xml`, is supplied for reference.

The XML schema file, `sts.xsd`, is supplied in the `/usr/lpp/cicsts/cicsts52/schemas/sts` directory.

The STS configuration file contains the following elements:

#### **<keystore>**

Defines the RACF keystore type. The possible values are `JCERACFKS` and `JCECCARACFKS`. The default value is `JCERACFKS`.

#### **<issuer>**

Defines the STS as an asserting party. This element contains the following elements:

##### **<format>**

Contains any string. There is no default value.

##### **<uri>**

Contains any string. The default value is `http://cics`.

#### **<signature>**

Specifies the hash algorithm and the certificate label. This element has the following attribute:

##### **hash\_algorithm**

Possible values are `sha-1` and `sha-2`. The default value is `sha-2`.

This element contains the following element:

##### **<certificate>**

This element contains the following element:

##### **<label>**

The value of the RACF certificate label. The default value is `CICSCERT`.

#### **<clock\_skew>**

The clock skew time, in milliseconds. The default value is `180000 ms` (3 minutes).

A clock skew allows for any time difference in system clocks on different computers. It is applied to all timing conditions in a SAML token.

The following figure shows an example of an STS configuration file with all elements set:

```
<sts_configuration xmlns="http://www.ibm.com/xmlns/prod/cics/JVMSEVER/stsconfig">
  <keystore>
    <type>JCECCARACFKS</type>
  </keystore>

  <issuer>
    <format>urn:oasis:names:tc:SAML:2.0:nameid-format:entity</format>
    <uri>http://cics</uri>
  </issuer>

  <signature hash_algorithm="sha-1">
    <certificate>
      <label>CICSCERT</label>
    </certificate>
  </signature>

  <clock_skew>90000</clock_skew>
</sts_configuration>
```

## User registries

User registries, also known as security registries, store user account information, such as user ID and password, that can be accessed during authentication and authorization. CICS Transaction Server for z/OS supports different user registries.

### SAF registry

RACF is a SAF registry and the primary z/OS user registry. RACF is an *external security manager (ESM)* and it provides more functions than user registry. For more information, see [How it works: Securing CICS with RACF](#). An ESM is accessed through the SAF interface. In documentation about CICS Liberty support, you might see the term *SAF registry*. It means RACF or an alternative ESM.

On z/OS, access to the SAF registry is considered an *authorized service*. To access such authorized services, the caller needs to use one of the following methods:

1. An SVC routine to call an authorized service.
2. A program call (PC) instruction to another address space, which is itself authorized.

When the CICS security domain makes calls to SAF, it can use its SVC routine, DFHCSVC, which is loaded from the authorized LPA to make authorized calls to SAF (option 1). However, this option is not available to Liberty JVM servers as the CICS SVC is a private interface. Instead, option 2 is the model that is used by Liberty on z/OS. The *angel process* is a long-running started task that is required for the Liberty JVM server in CICS TS to use SAF-authorized services. The angel process is configured to run authorized and Liberty servers can connect to it to call authorized services. The access to its services is controlled by the SAFCREDS resource profiles. When run in a CICS region, use of profiles is the only way for Liberty servers to authenticate users with the SAF registry.

Liberty also offers the ability to fail over to unauthorized UNIX System Services to authenticate requests when the angel process is unavailable. However, this option is not supported when you run a Liberty JVM server in CICS.

### Lightweight Directory Access Protocol (LDAP) registry

LDAP is an open industry standard application protocol for accessing distributed directory information services. It is widely used in enterprises to authenticate users and retrieve user groups. CICS TS can use LDAP to retrieve a Certificate Revocation List (CRL) or to create basic authentication credentials for web requests through the LDAP XPI functions and the XWBAUTH global user exit. Liberty JVM servers that run in CICS TS can also connect to an LDAP registry to perform authentication and authorization.

## Basic registry

If you use CICS TS with Liberty, an extra option is available with the basic registry. The basic registry provides a simple, text-based registry in the `server.xml` file. Access to `server.xml` is not controlled by RACF.

**Recommendation:** It is not advisable to use this registry type for any purpose other than testing because this registry is not integrated with CICS Liberty security or synchronized with RACF.

---

## Chapter 5. How it works: Authorization in CICS

CICS Transaction Server for z/OS uses RACF to authorize a user identifier (user ID) to a specified resource. Authorization is based on an identifier that is either trusted or has been previously authenticated.

CICS checks authorization to:

- Run CICS transactions (*transaction security*). For more information, see [Transaction security](#).
- Access resources (*resource security*). For more information, see [Resource security](#).
- Use CICS system programming commands (*command security*). For more information, see [Command security](#).
- Communicate between CICS systems (*intercommunication security*) if you are using MRO, IPIC, or ISC connections between multiple regions. For connected systems, the same principles of protecting CICS resources, commands, and transactions apply, but you also have to consider the resource definitions for the connections. For more information, see [Intercommunication security](#).
- Control access to a resource in an application served by Liberty (*role authorization*). For more information, see [Role authorization](#).

You can further manipulate these capabilities to:

- Customize the CICS authorization checks in your application. The application can use the CICS API to determine a user's level of access to a resource (*application-specific authorization*). For more information, see [Application-specific security \(QUERY SECURITY\)](#).
- Issue requests on behalf of other users (*surrogate authorization*). For more information, see [Surrogate security](#).

Usually the same RACF definitions apply to all CICS regions. Each region would normally use the same security configuration definitions. If you have a specific requirement for some regions to have different definitions, then this can be achieved using the [SECPRFX SIT option](#).

---

### Transaction security

Transaction security ensures that users that attempt to run a transaction are entitled to do so. You might come across the alternative terms of *attach-time security* or *transaction-attach security* to describe transaction security. Transaction security is the most fundamental form of security checking that is required to secure a CICS region and its application; you should **always** enable transaction security. Without transaction security, any user who has access to CICS can run any transaction without even needing to sign on.

The security section of the documentation assumes that transaction security is enabled.

Transaction security checking applies to all user transactions and CICS transactions in Category 2. (Category 1 and Category 3 transactions are not checked.) You cannot turn transaction security on or off for an individual transaction.

See [Transactions in CICS](#) for an introduction to CICS transactions and their categorization.

The **XTRAN** system initialization parameter controls transaction security. CICS issues an authorization request for every transaction, regardless of how it was started. The user ID that is used for authorization is the user ID that is associated with the request. For details of user IDs, see [How it works: Identification in CICS](#).

[Figure 23 on page 50](#) shows how transaction security works.

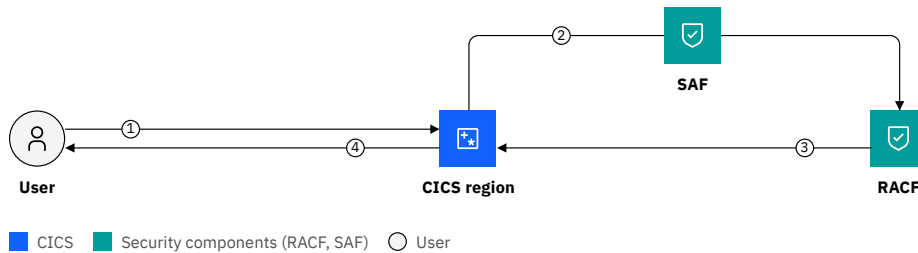


Figure 23. Transaction security

In this process:

1. The user initiates a transaction.
2. CICS checks that the transaction is active (XTRAN=ON) and calls RACF to determine if the user has enough access rights for the transaction.
3. RACF checks the currently active transaction resource profile and returns a yes or no decision to CICS. RACF also logs the activity.
4. CICS then allows or denies the user access.

CICS and RACF process the authorization request using profiles for each transaction in RACF resource classes. You define these profiles either in the default RACF resource classes for CICS (TCICSTRN or GCICSTRN), or in your own classes. The CICS transactions, except sample transactions in Category 2, are generated in the designated groups when you initialize the CICS system definition data set (CSD) or during installation. You identify your transactions to RACF using the transaction names that you have assigned to them. See [RACF classes and profiles for resources](#) for more information about the RACF resource classes.

The *currently active* transaction profile is used for authorization checking. To understand this, see [Refreshing profiles for SETROPTS RACLIST processing](#).

To set up transaction security:

- Set the **XTRAN** system initialization parameter. **Always have XTRAN on.**
- Set up RACF profiles to specify which user is authorized to run a transaction. For an example, see [How it works: Securing CICS with RACF](#).

## Resource security

Resource security provides an additional layer of security on top of transaction security. It controls access to the resources, such as files, queues, or programs, that are used by the transaction.

Unlike transaction security that cannot be turned off for individual transactions, resource security can be applied at the individual transaction level. You can control access to the resources that are associated with a particular transaction. A user who is authorized to invoke a particular transaction might not be authorized to access some resources used by the transaction, or might have limited access: for example, authority to read but not to update.

System initialization parameters (**RESSEC** and the set of parameters that start with X, known as the **Xnnn** parameters) and attributes on a TRANSACTION resource definition control when CICS issues an authorization request for access to a resource. The **Xnnn** parameters tell CICS that you want resource security checking on a class of resource: for example, the **XFCT** system initialization parameter specifies resource security checking for files. For a list of the **Xnnn** parameters, see [Security-related SIT parameters](#).

CICS checks the authority to access the resource every time that a transaction tries to access it. RESSEC gives you additional flexibility to control resource security at the transaction level. The RESSEC attribute on a TRANSACTION resource specifies that you want resource security to apply to that

transaction. The **RESSEC** system initialization parameter determines whether CICS honors the setting on the TRANSACTION resource definition or overrides it and always performs resource security checking.

**Important:** Because resource security is enabled at the transaction level, it applies to every program resource that the transaction might use. Care should be taken when you determine which programs are to use resource security.

The user ID that is used for authorization is the user ID that is associated with the request. For details of user IDs, see [How it works: Identification in CICS](#).

Figure 24 on page 51 shows how resource security works, where the system administrator has defined the transaction AP01 to require resource security checking.

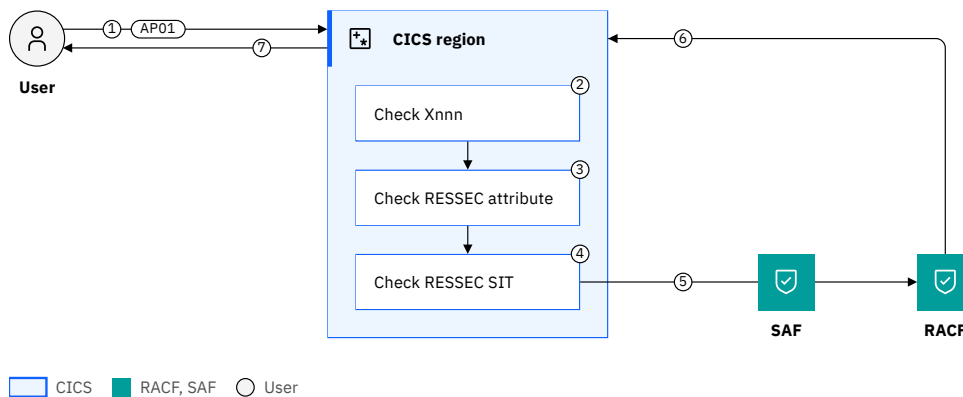


Figure 24. Resource security

In this scenario:

1. The user initiates Transaction AP01. Because AP01 is defined to run a program APADD, APADD then issues a request to access a CICS resource.
2. CICS checks whether resource security is active for this class of resource (**Xnnn** parameters).
3. CICS checks whether resource security in the transaction resource definition is active, for example:

```
DFHCSD
TRANSACTION (AP01)
RESSEC (YES)
```

4. CICS checks the SIT parameter **RESSEC**. If it is set to ALWAYS, the resource security check is always performed regardless of the RESSEC value in the transaction resource definition.
5. CICS calls RACF to determine whether the user has enough access rights for the resource.
6. RACF checks the currently active resource profile and returns a yes or no decision to CICS. RACF also logs the activity.
7. CICS then allows or denies the user access.

CICS and RACF process the authorization request using profiles for each resource in RACF resource classes. You define these profiles either in the default RACF resource classes for CICS or in your own classes. See [RACF classes and profiles for resources](#) for more information about the RACF resource classes.

The *currently active* resource profile is used for authorization checking. To understand this, see [Refreshing profiles for SETROPTS RACLIST processing](#).

To set up resource security:

- Set the **Xnnn** system initialization parameter for each class of resource that you want CICS to check.
- Control resource security for individual transactions by setting the RESSEC attribute on the TRANSACTION resource definition.

- Set the **RESSEC** system initialization parameter to tell CICS whether to honor or override the RESSEC attribute on a TRANSACTION resource definition.
- Set up RACF profiles to specify if a user is authorized to access a resource.

## Use of the WARNING option

The RACF WARNING option, if used on RACF profiles, is honored by CICS. The WARNING option allows users access to resources that otherwise would be denied. RACF logs to SMF those accesses that would have failed had WARNING not been in effect.

The selective use of WARNING may be useful when initially implementing resource security for an application.

When WARNING results in an SMF type 80 record being recorded, you should verify whether the user should be added to the access list for the resource, and modify the RACF profiles accordingly.

### Recommendation:

- In general, use WARNING in test environments.  
The only situation where you might need to use WARNING in a production environment is when you make security changes to production and want to ensure those changes do not break the existing valid behavior. In this case, make sure you use it only for a short period and address any warnings promptly.
- If you want to be notified at once when access is denied to a user, specify the NOTIFY option. If you do not have a suitable test environment, specify the NOTIFY option so that you can take action immediately.
- You must strictly limit the time during which resources are accessed with the WARNING option in force, and keep logging to a minimum during the warning period.

## Command security

---

Command security applies to CICS system programming commands. Security checking is performed for these commands, when they are issued from a CICS application, and for the equivalent commands that you can issue with the CEMT main terminal transaction or CECI. Command security operates in addition to any transaction security or resource security that you define for a transaction. You should **always** enable command security. Without command security, either applications or any user who has access to CEMT or CICS can issue any SPI command.

For a list of the system programming commands, see [System commands](#).

Like resource security, you can control command security checking at the individual transaction level. You can control access to the SPI commands that are associated with a particular transaction. A user who is authorized to invoke a particular transaction might be allowed only to inquire on the status of a resource but not to change it. A primary use of this support is to restrict what users can do with powerful transactions such as CEMT, which you can use to inquire on resources, to change them, and to shut down CICS.

System initialization parameters (**XCMD** and **CMDSEC**) and attributes on a TRANSACTION resource definition control when CICS issues an authorization request for access to a SPI command. The **XCMD** parameters tell CICS that you want command security checking on CICS system commands. CICS checks the authority to run the command every time that a transaction tries to access it. **CMDSEC** gives you additional flexibility to control command security at the transaction level. The **CMDSEC** attribute on a TRANSACTION resource specifies that you want command security to apply to that transaction. The **CMDSEC** system initialization parameter determines whether CICS honors the setting on the TRANSACTION resource definition or overrides it and always performs command security checking.

The user ID that is used for authorization is the user ID that is associated with the request. For details of user IDs, see [How it works: Identification in CICS](#).

[Figure 25 on page 53](#) shows how command security works, where the system administrator has defined the transaction CEMT to require command security checking.



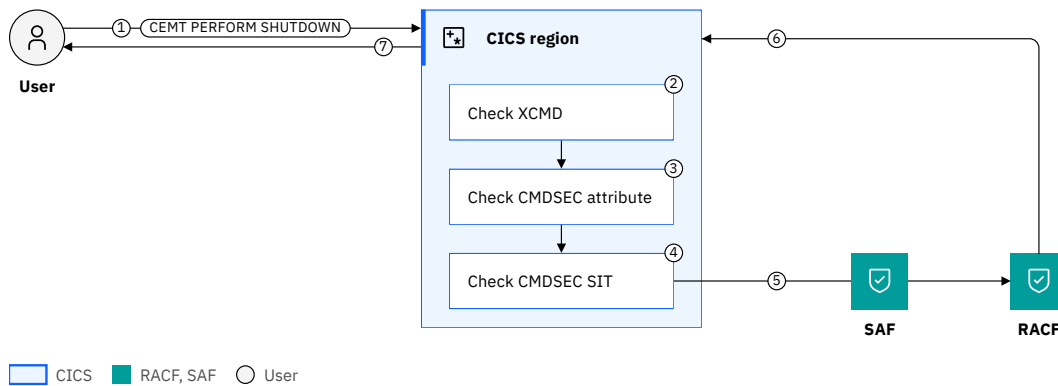


Figure 25. Command security

In this scenario:

1. The user initiates a CEMT request, for example, CEMT PERFORM SHUTDOWN.
2. CICS checks whether command security is active for CICS system commands (**XCMD** SIT parameter).
3. CICS checks whether command security in the transaction resource definition is active (CMDSEC), for example:

```

DFHCSD
TRANSACTION (CEMT)
CMDSEC (YES)

```

4. CICS checks the SIT parameter **CMDSEC**. If it is set to ALWAYS, the command security check is always performed regardless of the CMDSEC value in the transaction resource definition.
5. CICS calls RACF to determine whether the user has enough access rights to run the command by checking the resource identifier for the command. Resource identifiers for each command are described in [CICS resources subject to command security checking](#).
6. RACF checks the currently active resource profile and returns a yes or no decision to CICS. RACF also logs the activity.
7. CICS then allows or denies the user access to run the command.

CICS and RACF process the authorization request using profiles for each transaction in RACF resource classes. You define these profiles either in the default RACF resource classes for CICS (CCICSCMD or VCICSCMD) or in your own classes. See [RACF classes and profiles for resources](#) for more information about the RACF resource classes.

The *currently active* profile is used for authorization checking. To understand this, see [Refreshing profiles for SETROPTS RACLIST processing](#).

To set up command security:

- Set the **XCMD** system initialization parameter. **Always have XCMD on.**
- Control command security for individual transactions by setting the CMDSEC attribute on the TRANSACTION resource definition.
- Set the **CMDSEC** system initialization parameter to tell CICS whether to honor or override the CMDSEC attribute on a TRANSACTION resource definition.
- Set up RACF profiles to specify if a user is authorized to access a resource.

To help you decide which commands to protect with command security, see [CICS commands subject to command security checking](#).

## Security for CICS shutdown

---

When planning security for CICS shutdown, you need to consider the security required by the shutdown assist program if any is in use, and the security for the shutdown commands themselves.

Security for CICS shutdown consists of two aspects:

- Security for the shutdown assist program if you used any.
- Security for the shutdown operation itself by using the [CEMT PERFORM SHUTDOWN](#) transaction or the [EXEC CICS PERFORM SHUTDOWN](#) command (referred to as **PERFORM SHUTDOWN** in the following text).

### Security for the shutdown assist program

You can specify a CICS shutdown assist transaction to run at the beginning of a normal or an immediate shutdown, to shut down CICS in a controlled manner.

CICS supplies a default shutdown assist transaction, CESD, which invokes the CICS-supplied shutdown assist program, DFHCESD. You can also specify a transaction of your own. This is controlled by the [SDTRAN system initialization parameter](#) or the SDTRAN (or NOSDTRAN) option on **PERFORM SHUTDOWN** commands. For more information about how to specify the shutdown assist transaction, see [Shutdown assist program \(DFHCESD\)](#).

The transaction runs under the user ID authority of the issuer of the shutdown command. Security checking is conducted as follows, depending on the shutdown assist transaction specified:

#### The default transaction, CESD, which invokes the default program DFHCESD

- Transaction security for CESD is always in force
- If DFHCESD is loaded either from a library in the DFHRPL concatenation or from the MVS™ link pack area, command security and resource security are exempted. In this case, external security profiles such as those used with RACF are not required for access to SPI commands in DFHCESD or resources it accesses.
- If DFHCESD is not loaded from either of the places above, command security and resource security are conducted for SPI commands in DFHCESD or resources it accesses. Therefore, external security profiles such as those used with RACF are required. See [Command security](#) and [Resource security](#) for more information.

**Note:** By using the default shutdown assist transaction and program, you can minimize what is required to set up the necessary security.

#### A custom shutdown assist transaction

- Transaction security for the specified transaction is in force.
- Command security and resource security are also in force.

#### No shutdown assist transaction

In this case, no shutdown assist program is run. Therefore, no security checking is conducted for the assist program. However, security checking is still conducted for the shutdown operation itself. See [“Security for the shutdown operation \(PERFORM SHUTDOWN\)”](#) on page 54.

### Security for the shutdown operation (PERFORM SHUTDOWN)

Regardless whether you use a shutdown assist transaction, or which shutdown transaction you use, the following security checking is conducted depending on how you shut down CICS.

If you use **CEMT PERFORM SHUTDOWN** or the CICS Explorer® equivalent in the **Regions** view, transaction security for CEMT and command security for the **PERFORM** command are in force.

If you use **EXEC CICS PERFORM SHUTDOWN**, command security for **PERFORM** is in force.

## Intercommunication security

---

Intercommunication security ensures that users of one CICS system are entitled to run transactions and access resources in another CICS system. Intercommunication security in CICS is concerned with *incoming requests* for access to CICS resources instead of requests that are sent out to other systems. Security issues with incoming requests arise when a given user at a given *remote system* tries to access resources of your *local system*. You need to determine if this access is authorized or whether the access should be rejected.

For connected systems, the same principles of protecting CICS resources, commands, and transactions apply, but you also have to consider the resource definitions for the connections. You need to allow for the fact that users of one CICS system can initiate transactions and access resources in another CICS system. You might have to define a RACF user profile or group profile more than once: you might have to define these profiles in each CICS system that is using a separate RACF database, and in which a user is likely to want to attach a transaction or access a resource.

When planning the RACF profiles, you must consider all cases in which a user could initiate function shipping, transaction routing, asynchronous processing, distributed program link, distributed transaction processing, or external call interface (EXCI). For descriptions of these methods of intercommunication, see [Distributed transaction processing overview](#).

There are three types of link that you can use between CICS systems:

### **MRO (Multiregion operation)**

MRO is a CICS-to-CICS facility that does not require a network connection such as TCP/IP or SNA.

MRO can be used between CICS regions that are in the same z/OS LPAR or in the same z/OS sysplex and joined by the z/OS coupling facility.

### **IPIC**

For communication between CICS and non-CICS systems, or between CICS systems that are not in the same LPAR or z/OS sysplex, a network access method provides the necessary communication protocols. When this protocol is TCP/IP, communication is known as IP interconnectivity or IPIC.

### **ISC**

For communication between CICS and non-CICS systems, or between CICS systems that are not in the same LPAR or z/OS sysplex, a network access method provides the necessary communication protocols. When this protocol is SNA (either LU6.1 or LU6.2), communication is known as intersystem interconnectivity or ISC.

You are recommended to connect systems through IPIC instead of ISC. ISC is listed here only for legacy purposes and no further documentation about it is provided in this section.

There are three elements to intercommunication security:

### **Bind time security**

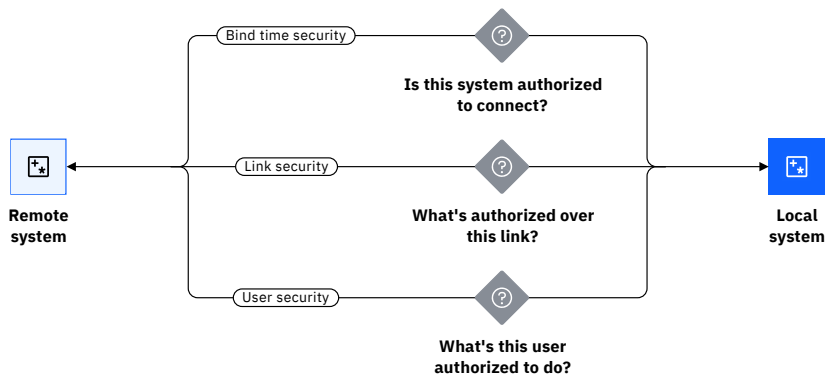
Bind time security prevents an unauthorized system from connecting to CICS.

### **Link security**

Link security controls what access is authorized over the link between the two systems.

### **User security**

User security controls what the requester is authorized to do.



Requests that run in the local system are security-checked by both the *asserted user ID* and the link user ID. Both user IDs are optional, depending on how you have configured CICS. If both are available, then the task user ID is derived from the asserted user ID. For information about the different user IDs used in CICS security processing, see [How it works: Identification in CICS](#).

## Bind time security

For any communication between systems, the first requirement is a session to be established between the two systems. A session, once established, is usually long-lived. The connection request that establishes the session can, depending on the circumstances, be issued either by the remote system or by your local CICS system.

Your security concern at bind time is to prevent unauthorized remote systems from being connected to your CICS system; that is, to ensure that the remote system is really the system that it claims to be. This level of security is called bind time security. It prevents an unauthorized remote system from connecting to CICS and verifies that the remote system is really the system that it claims to be. Bind time security can be applied when a request to establish a session is received from, or sent to, a remote system.

Table 8. How bind time security is enforced	
Link type	Security check
MRO	DFHAPPL profile in the RACF FACILITY class
IPIC	Exchange of TLS client certificate with partner system

You are recommended to connect systems through IPIC instead of ISC.

## Link security

Each link between systems is given an authority that is defined by a *link user ID*. Link security defines the transactions and resources that the remote system is allowed to access across the connection. For information about how the link user ID is set for IPIC, see [How it works: IPIC link security](#).

Users cannot access any transactions or resources over a link that is itself unauthorized to access them. This means that each user's authorization is a subset of the authority of the link as a whole.

Link security works with the other forms of authorization, such as Transaction security or Resource security to govern what the link user ID can access or do in CICS. For example, resource security controls the authority of the link user ID to access resources.

Link authority is established just after the connection (IPIC) or session (MRO) is bound, by sign on of a user ID that is determined by the attributes below.

<i>Table 9. How link security is enforced</i>		
Link type	Resource definition	Attributes
MRO	SESSION	USERID
IPIC	IPCONN	LINKAUTH and SECURITYNAME

## Remote user security

In addition to the security profile that you set up for the link user ID, you can further restrict each remote user's access to the transactions, commands, and resources in your system. For information for IPIC, see [How it works: IPIC user security](#).

User security works with the other forms of authorization, such as [Transaction security](#) or [Resource security](#) to govern what the remote user can access or do in CICS. For example, resource security controls the authority of the remote user to access resources in the local system.

<i>Table 10. How remote user security is enforced</i>		
Link type	Resource	Attributes
MRO	CONNECTION	ATTACHSEC
IPIC	IPCONN	USERAUTH

## When do I use link security and user security?

User security causes CICS® to make a second check against a user, in addition to the link security check. There are use cases for different combinations of link security and user security.

<i>Table 11.</i>		
	Link user ID	No link user ID
Authenticated or asserted user ID	<p>If users can enter the system from multiple locations, you can use the link user ID to give different access to different users, depending on their entry point.</p> <p>For example, a user can access a resource if they issue the request at work but not if they make the request from home.</p>	<p>Use this where none of the security checks need to take account of where a request came from.</p> <p>Security is based only on the authenticated or asserted user ID.</p>

Table 11. (continued)

	Link user ID	No link user ID
No authenticated or asserted user ID	<p>Use this where all user requests are made under a single link user ID, often known as a functional ID. All security checking must be performed by the remote region, or earlier. Although this simplifies the security setup, requests cannot be secured or audited by CICS for different users. Auditing requires tying up information from the local region with information in the remote region and earlier.</p> <p>Security is based only on the link user ID.</p>	<p>This is generally not recommended. It is useful only if no security is required to access resources on the local system. All requests run under the CICS default userid.</p>

## Application-specific security (QUERY SECURITY)

In general, it is not advisable for application code to do all its own security checking; this should be the responsibility of CICS. But in some cases you might want to include your own security checking in an application program, perhaps to enhance the user experience or to simplify the application.

The **QUERY SECURITY** command determines the level of access that a user has to a particular resource. Then the application program uses the values returned by the command to determine what action to take. The **QUERY SECURITY** command does not itself grant or deny access to a resource.

Here are some examples of how you might use **QUERY SECURITY**:

- Produce a menu application of only those options that a user is allowed to use. This can simplify the application design because all users have the same initial transaction but the navigation through the application depends on their individual authority. [Figure 26 on page 59](#) shows the difference that this makes to the flow of the application.
- Securing non-CICS resources. For example, you might have all customer records in one file but only allow users to access records in their region. You can define your own RACF class, with a series of profiles for each region.
- Check if a user has access to all resources required before executing a series of API requests. This can simplify error recovery and improve the user experience. For example, an application requires access to three files. The user updates the first two files but finds that they don't have access to the last file. Correcting this situation requires a syncpoint rollback. It could be easier to return a message before the API requests start.



Figure 26. Application flows without and with QUERY SECURITY

You can query whether a user has READ, UPDATE, CONTROL, or ALTER authority on a resource.

There are two forms of the **QUERY SECURITY** command:

### QUERY SECURITY RESTYPE

QUERY SECURITY RESTYPE enables an application program to request from RACF the level of access a terminal user has to a specified resource for the environment in which the transaction is running. You can query access levels to CICS resources (including Db2® resource definitions) that are contained in the RACF classes activated at initialization by RACLIST. RESTYPE is a resource type that corresponds to one of the set of system initialization parameters that start with X, known as the **Xnnn** parameters. These **Xnnn** parameters activate resource security checking with the exception of z/OS UNIX files. There is an additional resource type, SPCOMMAND, that corresponds to the **XCMD** system initialization parameter used to activate command security checking.

Before calling RACF, for all resources except PSBs, CICS checks that the resource is installed. If the resource does not exist, CICS does not call RACF and returns the NOTFND condition.

### QUERY SECURITY RESCLASS

Queries access levels for non-CICS resources that are contained in RACF general resource classes, such as `TERMINAL`, `FACILITY`, or a similar resource class that you have defined.

For information about using `QUERY SECURITY`, see [QUERY SECURITY](#).

## Post-initialization processing

---

If you specify a program list table on the `PLTPI` system initialization parameter, CICS checks that the region user ID is authorized as a surrogate user of the user ID specified in the `PLTPIUSR` system initialization parameter.

The **`PLTPIUSR`** system initialization parameter specifies the user ID that CICS is to use for PLT programs that run during CICS initialization. All PLT programs run under the authority of the specified user ID, including the `CPLT` transaction, which must be authorized to all the resources referenced by the programs.

The scope of PLT security checking is defined by the `PLTPISEC` system initialization parameter. This parameter specifies whether command security checks and resource security checks are to apply to `PLTPI` programs.

If you do not specify the **`PLTPIUSR`** parameter, CICS runs `PLTPI` programs under the authority of the CICS region user ID, in which case CICS does not perform a surrogate user check. However, the CICS region user ID must be authorized to all the resources referenced by the PLT programs. Furthermore, the CICS region user ID is associated with any transactions started by PLT programs, and therefore must be authorized to run such transactions.

## Surrogate security

---

A *surrogate user* has the authority to start work on behalf of another user (the *execution user*). The execution user is typically a functional user ID. A surrogate user is authorized to act for the execution user without authenticating as that user.

**Recommendation: *Always*** enable surrogate security. Without surrogate security, any user can run requests on behalf of any other. Transaction security can be used in conjunction with trusted applications to ensure that this is only used for authorized cases.

The **`XUSER`** system initialization parameter and the RACF `SURROGAT` class control surrogate security. There are a number of types of surrogate authority, depending on what the surrogate user attempts to do on behalf of the execution user. CICS issues an authorization request for every eligible situation (see “Types of surrogate authority” on page 61).

[Figure 27 on page 61](#) shows how surrogate security works for starting transactions.



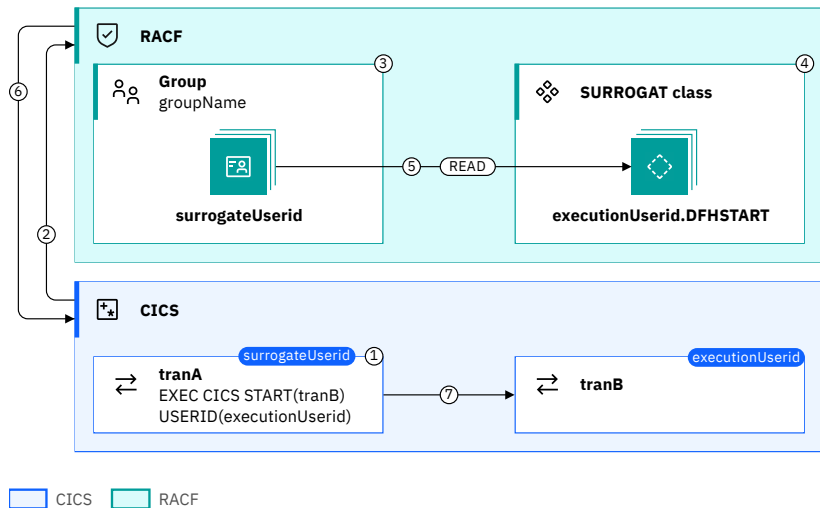


Figure 27. Surrogate security

1. A task that runs transaction *tranA* under the user ID of *surrogateUserId* issues an **EXEC CICS START** command to run transaction *tranB* under the user ID of *executionUserId*.
2. CICS calls RACF to check whether *surrogateUserId* has surrogate authority to start a transaction that runs under user ID *executionUserId*.
3. RACF finds a resource profile in the RACF SURROGAT class that matches *executionUserId.DFHSTART*. See [RACF classes and profiles for resources](#) for more information about the RACF resource classes.
4. RACF finds a group or groups permitted to use that profile and checks whether the *surrogateUserId* is in that group.
5. RACF checks that the group has READ access to the *executionUserId.DFHSTART* profile, which gives it the required surrogate security access.
6. RACF returns an OK response to CICS.
7. CICS starts the transaction *tranB* under user ID *executionUserId*.

To query a user's surrogate authority, you can use the QUERY SECURITY command with the RESCLASS('SURROGAT') option. For information, see [Application-specific security \(QUERY SECURITY\)](#).

The *currently active* transaction profile is used for authorization checking. To understand this, see [Refreshing profiles for SETROPTS RACLIST processing](#).

To set up surrogate security:

- Set the **XUSER** system initialization parameter. **Always have XUSER on.**
- Set up RACF profiles to specify which user is authorized to run as a surrogate.

## Types of surrogate authority

A surrogate user ID is granted access to perform a specific type of request for an execution user ID by granting access to a profile in the RACF SURROGAT class. [Table 12 on page 62](#) gives the list of the profiles that apply to CICS and the actions that the surrogate user ID can do with the authority of the execution user ID, if given access to that profile. For information about the different user IDs in CICS, including resources that have user IDs, see [How it works: Identification in CICS](#).

Table 12. Which profile controls what a surrogate user ID can do with authority of an execution user ID

Profile in RACF SURROGAT	The surrogate user	What the surrogate user can do
<i>execution-userID.DFHSTART</i>	Task user ID	Start a transaction running under the <i>execution-userID</i> . This applies to the API commands <a href="#">START</a> , <a href="#">DEFINE PROCESS</a> , and <a href="#">DEFINE ACTIVITY</a> .
<i>execution-userID.DFHINSTL</i>	At startup, the CICS region user ID, or the task user ID that issues a <a href="#">DEFINE</a> , <a href="#">ALTER</a> , or <a href="#">INSTALL</a> command, or issue a <a href="#">SET</a> command that changes the specific user ID.	Install or change a resource that has the ability to perform function under the <i>execution-userID</i> . See <a href="#">How it works: Identification in CICS</a> .
<i>default-userID.DFHINSTL</i>	CICS region user ID	Use the CICS default user ID. See <a href="#">Defining the default CICS user ID to RACF</a> .
<i>plt-userID.DFHINSTL</i>	CICS region user ID	Allows the PLTPUIUSR user ID to run PLT processing. See <a href="#">Surrogate security</a> .
<i>execution-userID.DFHEXCI</i>	The batch job's user ID	Run an EXCI call in CICS by using the <i>execution-userID</i> .
<i>execution-userID.DFHQUERY</i>	Task user ID	Issue the <a href="#">QUERY SECURITY</a> command on behalf of the <i>execution-userID</i> .
<i>execution-userID.SUBMIT</i>	Task user ID	Submit JCL to run a job under the <i>execution-userID</i> . See <a href="#">Security for submitting a JCL job to the internal reader</a> .

## Role authorization

The Enterprise Java model used in Liberty leverages subjects, resources, and roles to determine what can and cannot be allowed.

When a user is authenticated, the source of a request is represented as a Subject object within the Liberty server. The roles are specified within the Java application, either in the code or within its deployment descriptor.

Role authorization then uses user-to-role mapping to find out whether a user or group of users belongs to a certain role, and therefore what part of the application they can access.

Role authorization mappings are defined in one of three places:

1. Within each application's <application-bnd> element
2. Using the Liberty SAF role mapper and SAF registry profiles in the RACF EJBROLE class
3. With the role-to-user registry group mapping

For more information about implementing role security in CICS Liberty, see [Authorizing users to run CICS Liberty web applications](#) and [How it works: Securing Link to Liberty applications](#).

---

## Chapter 6. How it works: Integrity and confidentiality in CICS

*Integrity* ensures that transmitted or stored information is not altered in an unauthorized or accidental manner. *Confidentiality* ensures that an unauthorized party cannot obtain the information in the transferred or stored data, even if they are able to access this data. Typically, confidentiality is achieved by encrypting the data. The integrity and confidentiality of data is assured in different ways, depending on whether that data is *in motion* (flowing between systems), *in memory* (in the CICS address space memory), or *at rest* (in a storage device).

---

### Data in motion

In a security context, *data in motion* refers to data that flows between systems: for example, IP data that flows between a client and a server. Other forms of data in motion include SOAP messages and 3270 terminal sessions. You must consider how to protect the integrity and confidentiality of this data.

For information about other situations in which data must be protected, see [How it works: Confidentiality and integrity in CICS](#).

Integrity and confidentiality for TCP/IP connections is provided by the Transport Layer Security (TLS) protocol. In addition, individual SOAP messages can be signed or encrypted by using XML signature and XML encryption standards. Intrusion Detection Services protect 3270 data streams.

### Transport Layer Security (TLS) for IP connections

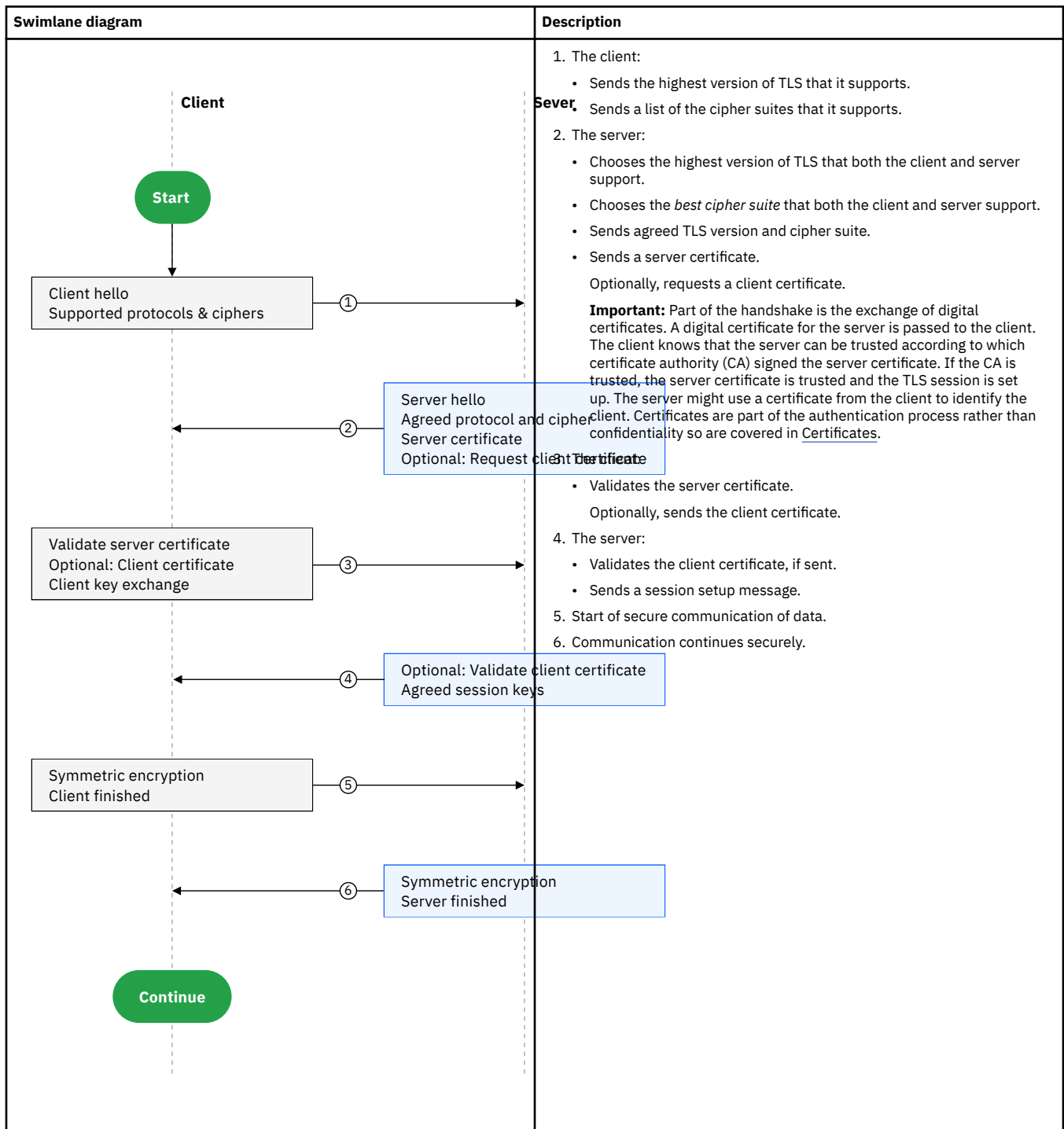
TLS is a cryptographic protocol that provides security for connections over the Internet Protocol network. It is used for sessions over the web and for IPIC sessions. In both cases, the concept exists of a client that initiates a request and a server that provides the response. CICS can operate as a client or a server. A TLS session has two protocols: a handshake protocol to establish a secret key and a record protocol to encrypt data that flows in the session.

TLS is the modern version of a protocol originally called Secure Sockets Layer (SSL). SSL protocols are no longer supported, but the term is sometimes used interchangeably with TLS, especially in API options and resource definitions.

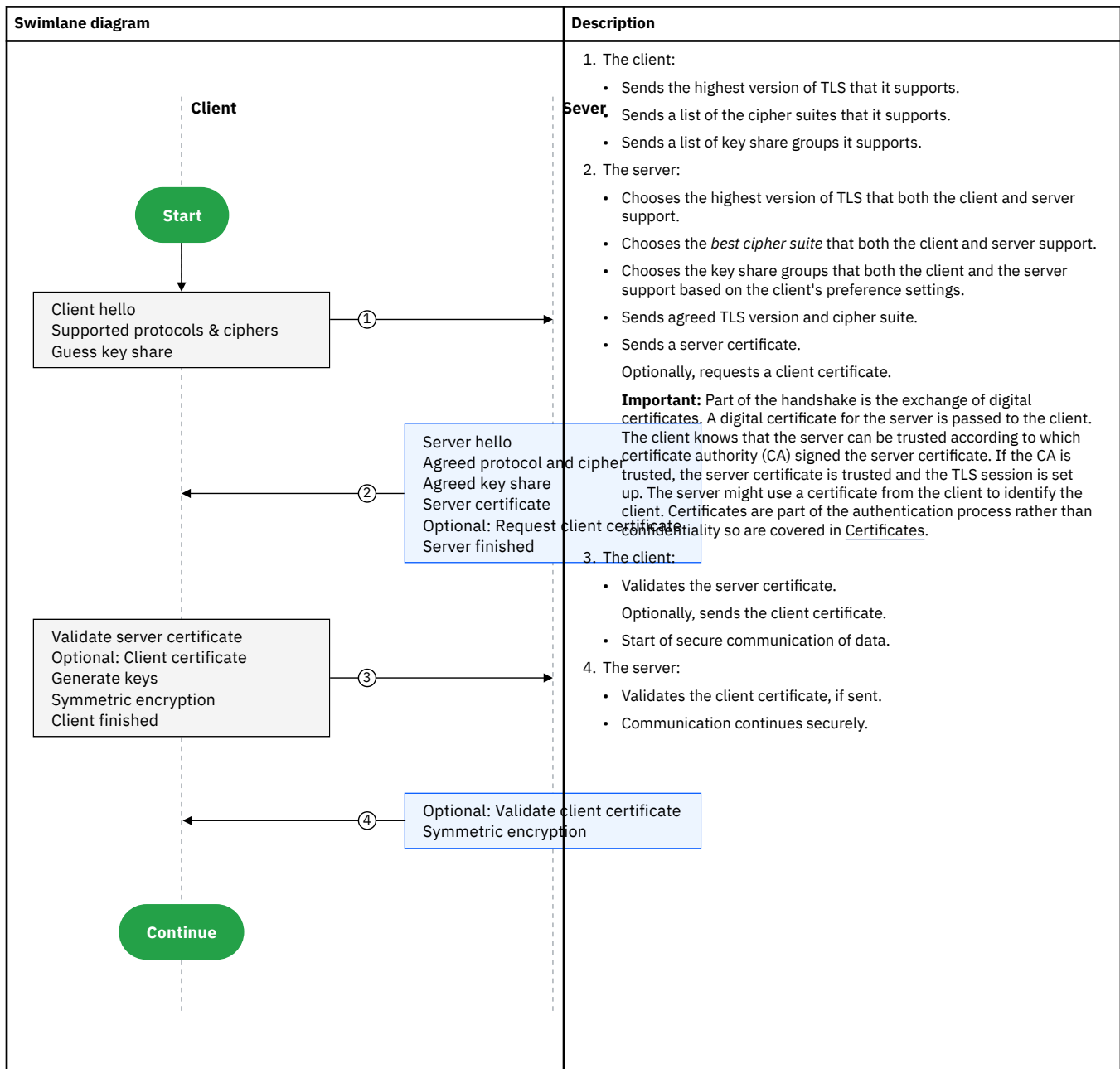
The TLS handshake is for the client and server to agree characteristics about the session: the TLS protocol and a cipher. The protocol that is used is the highest level that is supported by both sides. The cipher that is selected is a cipher that is common to both sides. The cipher that is selected is the first one in the list that is supplied by the server. If no common protocol or cipher is identified, a connection cannot be made. The exact mechanism of the handshake depends on the protocol level. [“The TLS 1.2 handshake” on page 64](#) and [“The TLS 1.3 handshake” on page 65](#) show how the configurable parameters flow between the client and the server.

For a more in-depth view of the TLS protocol, including keys and message authenticate codes (MACs), see [TLS 1.2 Protocol](#) and [TLS 1.3 Protocol](#) in the IBM SDK Java Technology Edition documentation.

## The TLS 1.2 handshake



## The TLS 1.3 handshake



## TLS cipher suites

A cipher suite is a set of cryptographic algorithms that are used to create keys and encrypt information.

Cipher suites define the following algorithms:

- The key exchange algorithm used to exchange a symmetric key.
- The data exchange algorithm that uses the symmetric key to encrypt the data.
- The message authentication code (MAC) algorithm used to ensure message integrity.

In addition, the cipher suites include the lengths of the various keys used.

Some cipher suites are more secure than others. Which cipher suites are used might also depend on the compliance regulations to which a company needs to adhere. The choice of cipher suite has an impact on performance.

For information about how the cipher suites are specified, see [Implementation options for TLS](#).

## Related concepts

[Transport Layer Security \(TLS\) for IP connections](#)

## Implementation options for TLS

Learn the three complementary options to implement TLS in CICS.

- [“CICS TLS support by using System SSL” on page 66](#)
- [“CICS Liberty TLS support by using JSSE” on page 67](#)
- [“Application Transparent TLS by using AT-TLS” on page 68](#)

The method that is used depends on your requirements. You can mix these methods on different connections:

- Generally, AT-TLS is recommended because AT-TLS rules and policies enable a network administrator to handle all the TLS definitions in a single place.
- If you use client certificates for IPIC connections, you must use CICS TLS support by using System SSL.
- If you use certificates for authentication in CICS Liberty, then CICS Liberty TLS support is required.

## CICS TLS support by using System SSL

TLS can be used for securing TCP/IP connections to and from CICS that involve URIs and ports that are defined in the CICS resource definitions TCPIPService, URIMAP, and IPCONN. [Figure 28 on page 66](#) shows an example for an inbound connection that uses a TCPIPService definition. [Figure 29 on page 67](#) shows an example for an outbound connection that uses a URIMAP definition.

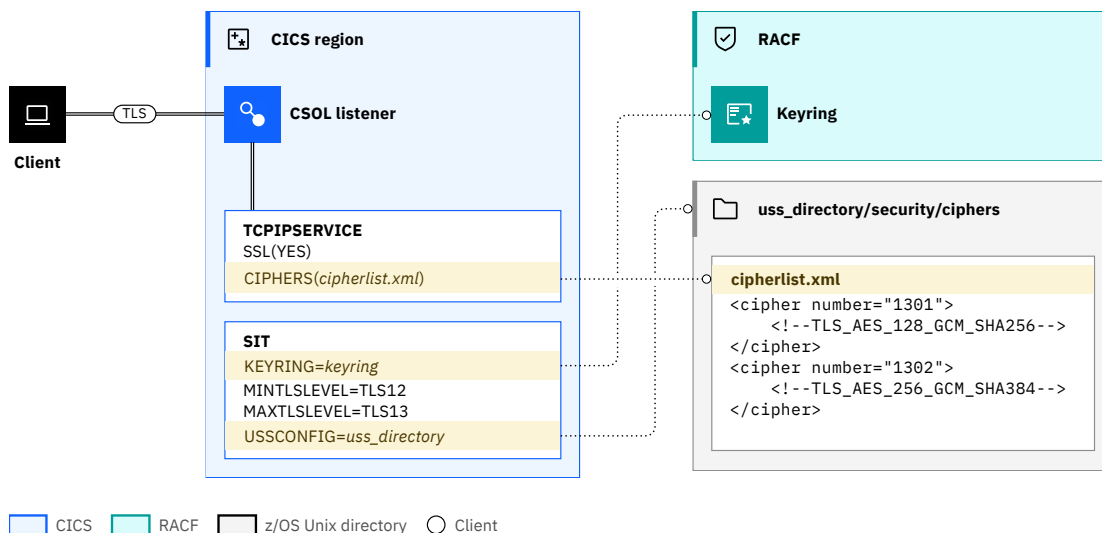


Figure 28. Inbound TLS support in CICS

The range of TLS protocols that can be used for securing the connection is obtained from the SIT. The SIT parameters [MINTLSLEVEL](#) and [MAXTLSLEVEL](#) define the minimum and maximum level of TLS that is supported by the CICS region.

The resource definition TCPIPService has SSL(YES) and the CIPHERS options. The CIPHERS option on the resource definition references a z/OS UNIX file that contains ciphers. Each connection can specify its own list of ciphers to allow configuration between CICS and clients and other servers that might have different capabilities.

**Important:** Older releases of CICS allowed the specification of lists of numbers to represent the ciphers instead of a z/OS UNIX file. This method is deprecated.

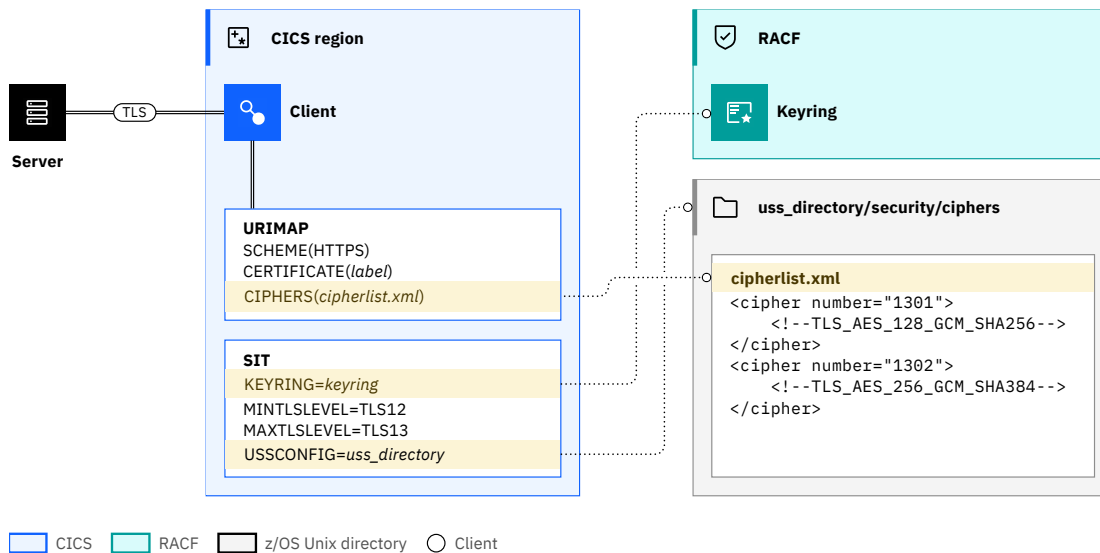


Figure 29. Outbound TLS support in CICS

A CICS transaction that is acting as a client uses a URIMAP to define the connection as SCHEME(HTTPS) to use TLS. It provides the CIPHERS and optionally the label of a certificate in the region's key ring.

The function that provides TLS support is [System SSL](#).

## CICS Liberty TLS support by using JSSE

TLS can be configured in Liberty, as shown in [Figure 30 on page 67](#).

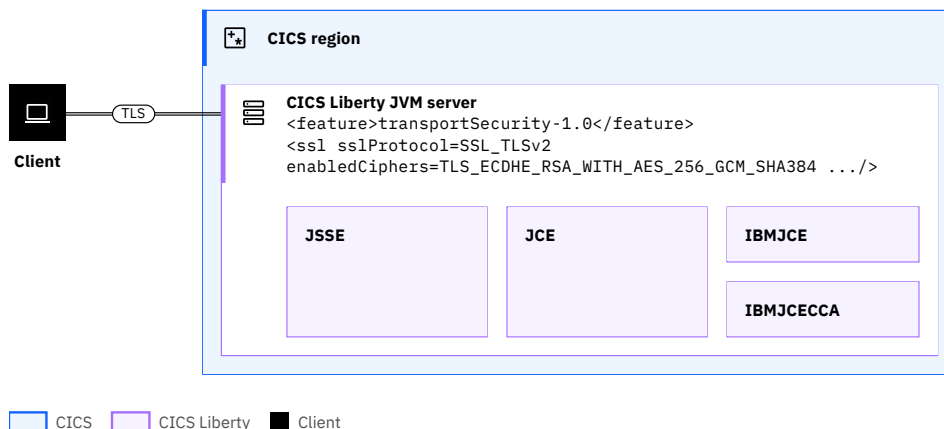


Figure 30. TLS in Liberty

You must enable the Liberty transportSecurity feature. The SSL element can be configured to specify the protocol and ciphers. For more information about configuration, see [Enabling SSL communication in Liberty](#).

JSSE provides a framework and Java implementation of the SSL and TLS protocols.

Java Cryptography Extension (JCE) is a standard extension to the Java Platform that provides the underlying implementation for cryptographic services, including encryption, key generation, and Message Authentication Codes (MAC).

The IBM Java SDK for z/OS provides two main JCE providers: IBMJCE, and IBMJCECCA. The default provider is IBMJCE, which can use the CPACF IBM Z cryptographic hardware. The IBM JCECCA provider is an optional provider that can also use ICSF to drive the IBM Crypto Express cards. For more information, see [Performance considerations for hardware cryptography](#).

## Application Transparent TLS by using AT-TLS

Application Transparent Transport Layer Security (AT-TLS) can be used to create TLS sessions on behalf of CICS.

Figure 31 on page 68 shows an example of this configuration inbound with CICS.

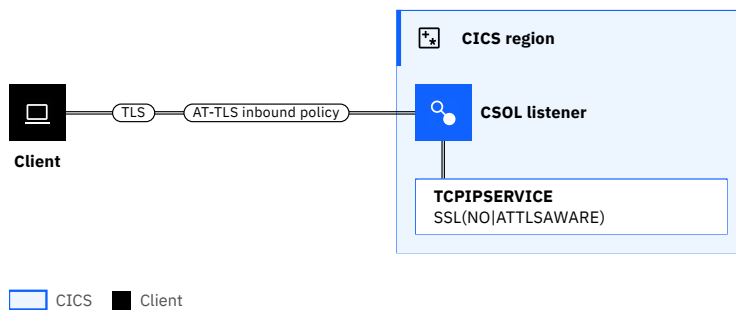


Figure 31. Inbound TLS support in CICS by using AT-TLS

Instead of implementing TLS in CICS, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the TCP/IP Policy Agent. When AT-TLS is active for a CICS socket connection, CICS sends and receives unencrypted data, while AT-TLS encrypts and decrypts data at the TCP transport layer.

As a result, CICS SIT parameters and resource configuration options that are related to TLS are not relevant because the implementation of TLS is provided by the AT-TLS policy statements. All encryption and decryption is done outside of the CICS address space.

For more information about AT-TLS and AT-TLS policy setup, see [AT-TLS policy configuration](#) and [Policy Agent and policy applications in z/OS Communications Server: IP Configuration Reference](#).

You can configure AT-TLS in CICS in two ways:

### AT-TLS basic

The port is unaware that AT-TLS is encrypting or decrypting data. This configuration is implemented in CICS by using the SSL(NO) option on the [TCPIP SERVICE](#) resource definition.

### AT-TLS aware

The port is aware of AT-TLS and can query information such as AT-TLS status and access items such as the client certificate and the certificate user ID. This configuration is implemented in CICS by using the SSL(ATTLASWARE) option on the TCPIP SERVICE definition. AT-TLS aware mode can be used only for CICS as an HTTP server. This option is not available on URIMAP definitions for outbound sessions or IPCONN definitions for IPIC sessions.

Figure 32 on page 69 shows an example outbound from CICS.



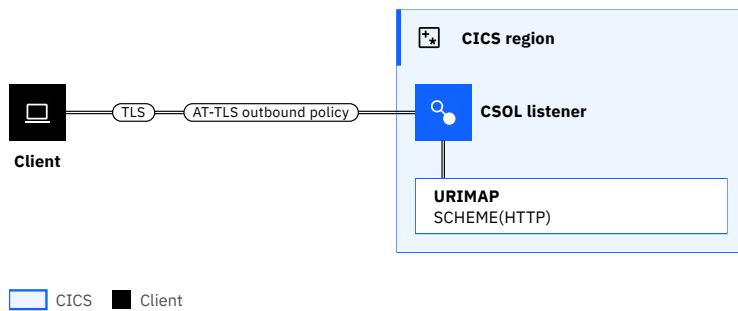


Figure 32. Outbound support for TLS in CICS by using AT-TLS

Figure 33 on page 69 shows an example in CICS Liberty.

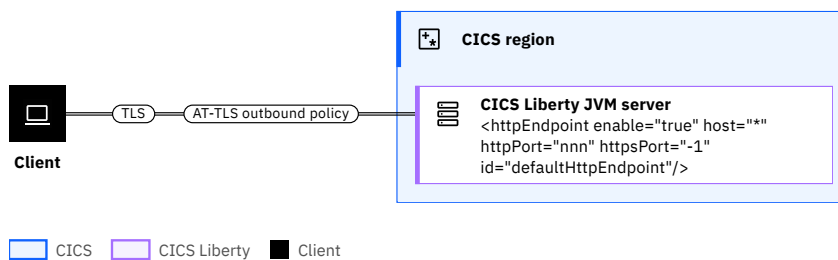


Figure 33. TLS in CICS Liberty by using AT-TLS

Implementing TLS in CICS Liberty by using AT-TLS is similar to CICS. The only difference is that you are unable to define ports as being AT-TLS aware. Therefore, it is not possible to identify the client certificate within a CICS Liberty application.

## Performance considerations for TLS connections

The TLS handshake can be CPU-expensive and can have a significant impact on throughput. When you expect a high number of messages, consider the following hardware and configuration options to improve performance.

- “Hardware cryptography” on page 69
- “Persistent connections” on page 70
- “Session caching” on page 70

### Hardware cryptography

The performance of TLS is considerably improved by using cryptographic hardware rather than using software encryption.

Two cryptographic hardware devices are available on IBM Z®:

- CP Assist for Cryptographic Function (CPACF)
- IBM Crypto Express® cards

CPACF is a set of cryptographic instructions that is available on all CPs, including zIIPs, IFLs, and General Purpose CPUs. Various symmetric algorithms are supported by the CPACF, including DES, 3DES, and AES-CBC, and SHA-based digest algorithms. CPACF provides the potential for significantly improved performance for these operations.

The IBM Crypto Express cards are optional I/O attached cards that implement extra cryptographic functions. On an IBM z14®, this feature is available as a Crypto Express 6S (CEX6S) adapter, or Crypto Express 5S (CEX5S).

By default, the Crypto Express card is a coprocessor (CEX6C) and can support a wider range of callable services. These services include secure key and clear key support for PKA decrypt, digital signature verify, digital signature generate, and include RSA and ECC variants. Alternatively, the card can be configured as an accelerator (CEX6A). In this mode, the card supports only three clear key cryptographic APIs, associated with RSA public key encryption, decryption, and verification. When the cryptographic coprocessor is configured as an accelerator, it provides better throughput at the expense of supporting fewer services.

TLS in CICS and AT-TLS uses System SSL to implement hardware cryptographic features. For more information about support for cryptographic hardware with System SSL, see [Guidelines for using hardware cryptographic features](#).

TLS in Liberty uses the JCE components for implementing hardware cryptography. The implementation in CICS is the same as z/OS Connect. For more information, see [Hardware cryptography](#).

## Persistent connections

Establishing TCP/IP connections between a client and server is expensive. If you encounter repeated traffic with the same server, persistent connections can reduce the need for regular TLS handshakes.

However, persistent connections can have downsides because they exist only between a client and a single server. You must limit workload balancing when you use cloned systems in a sysplex that uses shared ports. Therefore, CICS limits the number of times a persistent session can be reused. This function is enabled by default. For more information, see [SOTUNING](#).

[Web services performance](#) shows the outcome of performance tests on CICS TS 5.4.

## Session caching

The TLS handshake is a relatively expensive process. Therefore, information about TLS sessions is cached so that the handshake can be optimized, which can have a significant impact on the performance.

The mechanism for caching depends on how TLS sessions are made to CICS:

- [“Sysplex caching when you use TLS in CICS” on page 70](#)
- [“Local session caching when you use TLS in CICS” on page 71](#)
- [Session caching when using AT-TLS](#)
- Session caching is local to a JVM when you use TLS in CICS Liberty.

See this article about session caching that uses TLS 1.2 in

CICS: <https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/ian-mitchell1/2020/09/02/tls-12-session-id-caching-for-cics-in-a-sysplex>.

## Sysplex caching when you use TLS in CICS

Use sysplex caching if you have multiple CICS web-owning regions that accept TLS connections at the same IP address. Sharing TLS session IDs across different CICS regions on a sysplex is important when HTTP requests are being routed across a set of CICS regions. For example, by using TCP/IP connection workload balancing techniques, such as TCP/IP port sharing or Sysplex Distributor.

To enable sysplex caching, activate the z/OS System SSL started task GSKSRVR and specify the CICS system initialization parameter SSLCACHE=SYSPLEX for the CICS regions.

For more information about the SSL started task GSKSRVR and its configuration, see [The SSL started task GSKSRVR](#). For information about SSLCACHE, see [SSLCACHE system initialization parameter](#).

## Local session caching when you use TLS in CICS

Local caching in a single CICS region can be used when sysplex caching is not available. It is managed by z/OS® System SSL as part of the SSL environment, which exists within this enclave.

Local caching is implemented by the CICS SIT parameter SSLCACHE=CICS. See [SSLCACHE system initialization parameter](#).

## SOAP messages

In addition to TLS encryption, to secure the transport of SOAP messages, individual SOAP messages can be signed or encrypted by using the XML signature and XML encryption standards.

*Signing* a SOAP messages ensures the integrity of a message, preventing a man-in-the-middle attack from changing the messages. For more information, see [How it works: Signing SOAP messages](#).

*Encrypting* a SOAP messages ensures the confidentiality of a message between the client and CICS, irrespective of the number of intermediate systems that the message might flow through. For more information, see [How it works: SOAP message encryption](#).

If you need to ensure that the person knows the message came from you, sign the message. If your intent is to ensure no one sees the message content, encrypt it.

### Related concepts

[Web services overview](#)

### Related information

[Security for SOAP web services](#)

## 3270 sessions

Most 3270 data flows now come from terminal emulators. These emulators are connected to IBM z/OS Communications Server by using a TCP/IP interface. To protect confidentiality, these emulators must be connected by using TLS. In addition, *Intrusion Detection Services* (IDS) protect 3270 data streams and CICS BMS screens.

When most 3270 applications were written, the networking infrastructure used proprietary hardware components. Using these components avoided the need for any integrity or confidentiality considerations because no opportunity existed to intercept or to corrupt 3270 data flows.

Integrity was supported by 3270 hardware's conformance to the 3270 data stream protocol. This conformance includes preventing protected fields from being overwritten. Compliant 3270 emulators maintain these controls, but a rogue emulator, which is connected to the network, might bypass this protection.

To protect 3270 data streams, IBM provides two complementary solutions:

- IBM z/OS Communications Server Intrusion Detection Service to protect 3270 data streams.
- CICS BMS 3270 Intrusion Detection Services to protect BMS screens.

CICS BMS 3270 Intrusion Detection works together with the 3270 Intrusion Detection Service that is provided by IBM z/OS Communications Server. If configured, IBM z/OS Communications Server handles protection of all 3270 applications. When both services are enabled, BMS-generated 3270 data is handled by CICS, and non-BMS 3270 data is handled by IBM z/OS Communications Server. Enabling both gives you full coverage of all 3270 applications, but uses BMS to maximize performance and to enhance the information that is returned about any intrusion.

[Figure 34 on page 72](#) illustrates the flow of data in a 3270 session and the capabilities for integrity and confidentiality at each stage.

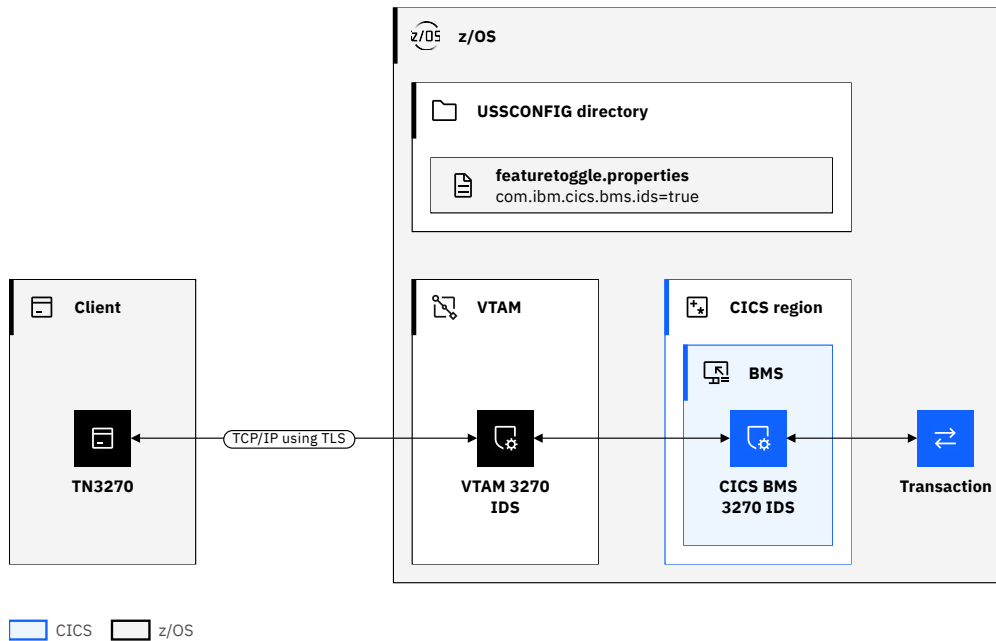


Figure 34. Integrity and confidentiality for 3270 sessions

## IBM z/OS Communications Server Intrusion Detection Service (IDS)

IDS offers support for scan and attack detection, reporting and traffic regulation for TCP connections, and UDP receive queues. You can use IDS policies to specify event conditions and the actions to take for particular events.

For more information, see [3270 Intrusion Detection Service](#).

## CICS BMS 3270 Intrusion Detection Services (IDS)

CICS provides 3270 IDS detection and protection for any applications that use CICS basic mapping support (BMS) interfaces to create and parse their 3270 screens.

The CICS BMS 3270 Intrusion Detection Service allows CICS to detect whether a 3270 emulator invalidly modifies a protected field that is generated by a BMS map. You can opt into this capability by configuring the `com.ibm.cics.bms.ids` feature toggle as described in [Specifying feature toggles](#). When this feature is activated, CICS monitors the 3270 data streams to detect any attempted modifications to protected fields on the screen. CICS can then provide warning messages or prevent the application from processing the data by abending the transaction.

By introducing a user-replaceable module `DFHBMX` into the BMS 3270 IDS configuration, you can get more granular and target specific applications or maps. But this configuration is generally only necessary if an application made unusual use of the 3270 data stream and reported false hits.

## Data in memory

In a security context, *data in memory* refers to data in the CICS address space memory, such as CICS control blocks and application storage. You must consider how to protect the integrity and confidentiality of this data.

For information about other situations in which data must be protected, see [How it works: Confidentiality and integrity in CICS](#).

Integrity of data in memory is implemented by ensuring that unauthorized users or programs cannot accidentally or maliciously change data. z/OS contains many features to protect programs; see [z/OS and system integrity](#) for an overview of these functions.

CICS provides *storage protection* and *transaction isolation* to prevent accidental or malicious overwrites of data between programs. For information, see [Storage protection](#) and [Transaction isolation](#).

Assembly language programs can use non-authorized instructions to switch between CICS-key and user-key storage. To maintain integrity, review all assembly language programs before you install them in a CICS region.

## Data at rest

---

In a security context, *data at rest* refers to data in storage devices, such as data in VSAM or Db2®. You must consider how to protect the integrity and confidentiality of this data.

For information about other situations in which data must be protected, see [How it works: Confidentiality and integrity in CICS](#).

Although data sets that CICS uses must be protected from unauthorized access, this protection extends beyond just the CICS region user IDs that handle the data. For example, a system programmer might need access to the data sets for backup and other functions.

To protect this data from unauthorized viewing, data sets can be encrypted. Keys can be defined, and the CICS region user IDs given access to these keys so that the data can be accessed and updated by CICS, but not by anyone else. This control exists even if they somehow managed to obtain a copy of the data.

CICS supports this capability for all VSAM and VSAM RLS data sets, and also the QSAM data sets used as TD queues, and BSAM data sets used for sequential devices. BDAM datasets are not supported.

Only data sets that contain sensitive information need to be encrypted. [Planning for data set encryption](#) provided information about the recommended CICS data sets to encrypt.

For more information about data set encryption, see [Data Set Encryption in z/OS DFSMS Using Data Sets](#).

The keys for data set encryption are stored securely in ICSF. The keys are referenced by a key label. This is in itself not secure. It is made secure by only allowing authorized users, such as the region user ID, to use the key label to decrypt the data. For more information, see [Managing Cryptographic Keys Using the Key Generator Utility Program in z/OS Cryptographic Services ICSF Administrator's Guide](#).



# Chapter 7. How it works: Auditing in CICS

Auditing is a key part of a security infrastructure. Its purpose is to validate that a company’s information assets are protected. Auditing CICS is an important part of this process.

At a high-level, auditors typically ask:

- Are procedures and practices consistent with documented policies?
- Are procedures and practices consistent with the requirements of regulation or legislation?
- Are procedures and practices generally consistent with a policy of “least privilege”: that is, users have access to the resources that are required to do their jobs, but no more.
- Are duties sufficiently controlled and separated?

There are many sources of information that need to be considered in auditing CICS. See the [z/OS Security Server RACF Auditor's Guide](#) for details. This section focuses on information that is specific to CICS and of interest to auditors.

If you use Liberty, there is auditing that is specific to Liberty. See [Auditing Liberty events](#) in the WebSphere Liberty documentation.

The following tables in [“Audit information for configuration”](#) on page 75 to [“Audit information for authorization”](#) on page 76 outline the information that you might want to audit for CICS configuration, operation, authentication, and authorization and where to find that information. The repositories and tools that are referenced in the table are defined in [“Sources of information”](#) on page 76. For more information about using the audit information, see [Auditing CICS](#).

## Audit information for configuration

Table 13. Finding audit information for CICS configuration		
What you want to audit	Where to find audit information	How to use the audit information
CICS configuration	<ul style="list-style-type: none"><li>• z/OS compliance data collection</li><li>• IBM Health Checker for z/OS</li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Compliance data collection</a></li><li>• <a href="#">Auditing CICS configuration with IBM Health Checker for z/OS</a></li></ul>
Who last changed a CICS resource	<ul style="list-style-type: none"><li>• CICS Explorer, WUI, or CEDA displays</li><li>• DFHCSDUP output</li></ul>	<a href="#">Identifying changes to resources (resource signatures)</a>
Changes that are made to CICS resources	<ul style="list-style-type: none"><li>• Transient Data Queue CSDL</li><li>• EYULOG of the CICSplex SM maintenance point CMAS</li></ul>	Change (before or after)
Resource installation	<ul style="list-style-type: none"><li>• Transient Data Queue CSDL or resource-specific TDQs,</li><li>• CICSplex SM MAS MSGUSR</li></ul>	<a href="#">Auditing installed resources</a>

## Audit information for operation

Table 14. Finding audit information for operation		
What you want to audit	Where to find audit information	How to use the audit information
Changes that are made to CICS by operator commands	Transient Data Queue CADS	<a href="#">Auditing SPI commands</a>

## Audit information for authentication

Table 15. Finding audit information for authentication		
What you want to audit	Where to find audit information	How to use the audit information
User sign-on and sign-off	<ul style="list-style-type: none"><li>• z/OS security console</li><li>• SMF type 80 records</li><li>• Transient Data Queue CSCS</li><li>• EYULOG for the CICSplex SM WUI</li><li>• Liberty messages log</li></ul>	<a href="#">Auditing sign-on and sign-off</a>
Failed authentication attempts	<ul style="list-style-type: none"><li>• z/OS security console</li><li>• SMF type 80 records</li><li>• CICS statistics</li><li>• Liberty messages log</li></ul>	<a href="#">Auditing sign-on and sign-off</a>
Connection attempts	<ul style="list-style-type: none"><li>• z/OS security console</li><li>• SMF type 80 records</li></ul>	

## Audit information for authorization

Table 16. Finding audit information for authorization		
What you want to audit	Where to find audit information	How to use the audit information
Failed access attempts	<ul style="list-style-type: none"><li>• z/OS security console</li><li>• SMF type 80 records</li><li>• Transient Data Queue CSCS</li><li>• CICSplex SM WUI EYULOG</li></ul>	<a href="#">Auditing authorization</a>
User changes to CICS resources	Log streams	
SAF requests	SMF type 80 records	
WARNING option used	RACF profiles	<a href="#">Listing the profiles in a class</a>

## Sources of information

### DFHCSDUP

The CSD utility program, DFHCSDUP, provides offline services for you to list and modify the resource definitions in the CICS system definition (CSD) file. See [Defining resources with DFHCSDUP](#).



## EYULOG

EYULOG is the output log that is associated with a component of CICSplex SM.

## IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that helps simplify and automate the identification of potential configuration problems before they impact availability or cause outages. See [IBM Health Checker for z/OS User's Guide](#) for details.

## Liberty messages log

Each CICS Liberty server writes messages in a zFS file, usually named messages.log. This log file can contain information about the server initialization, the status of application deployments and the errors that occurred. The level of information provided depends on the logging configuration. See [Liberty - Logging and Trace](#)

## MSGUSR

Is the log for messages, defined by the transient data destination, CSSL and used by a number of CICS services. For more information about how MSGUSR is used, see [Auditing installed resources](#).

## SMF type 80 records

Provide audit information for RACF.

z/OS collects data for each task when certain events occur in the life of the task. The System Management Facility (SMF) formats the information that it gathers into system-related (or job-related) records. Record type 80 is produced during RACF processing and Public Key Infrastructure (PKI) Services processing. For more information, see [Record type 80: RACF processing record](#).

## SMF 1154 subtype 80 records

Provide audit information for CICS.

The type 1154 records provide compliance evidence data for IBM Z. A different subtype is assigned to each participating z/OS component or product. CICS regions collect data and write an SMF 1154 subtype 80 with the information to SMF. For a list of records, see [SMF 1154 subtype 80 record](#). For instructions on how to retrieve the records, see [Compliance data collection](#).

## Transient data queues (TDQ)

Data can be stored in transient data queues for subsequent internal or external processing (see [Transient data control](#)). CICS services use certain TDQs, as shown in [Required TDQUEUE definitions](#). Audit messages are sent to the following TDQs:

- CADL for a log of each z/OS Communications Server resource definition installed in the active CICS system.
- CADS for messages that are produced when auditing the system programming interface commands.
- CRDI for a log of installed resource definitions.
- CSCS for messages that give details of each sign-on and sign-off. It also receives a message about each rejected attempt at sign on and each resource authorization failure.
- CSDL for a log of all commands that result in changes to the CICS system definition (CSD) file or active CICS system. Resource-specific installation messages are also sent to the TDQ that is associated with messages for that resource. See [Required TDQUEUE definitions](#) for the resource TDQs.

TDQs for messages are defined in group DFHDCTG. Output written to the CICS® transient data queues appears in JCL DDNAMEs that are allocated to a CICS region. The TDQs default to being redirected to TDQ CSSL, which is the MSGUSR DD in the job output.

## z/OS security console

z/OS Security Server RACF routes system operator messages to a system console or a *security console*. The security console is any MVS console with a routing code of 9.



---

## Chapter 8. How it works: Zero trust in CICS

Zero trust is a framework that assumes a complex network's security is always at risk to external and internal threats. It helps organize and strategize a thorough approach to counter those threats. You need to review several important aspects, including identity, data, devices, analytics, and network, before implementing a zero trust strategy in CICS.

### What is zero trust?

Traditionally, the IT industry has relied on perimeter security strategies to protect its most valuable resources such as user data and intellectual property. These security strategies involve using firewalls and other network-based tools to inspect and validate users going into and out of the network. However, changes to working patterns have made this model less reliable. For example, the increased use of a global workforce connecting from home means that relying on a network perimeter is no longer sufficient.

A zero trust strategy is built around the assumption that the threat can also exist inside the organization. This might be from an insider or an external hacker who has obtained the identity of an insider. To identify who is attacking, all access needs to be verified. It also assumes that attacks will occur. Therefore, constant monitoring is required to detect such an attack when it occurs and act immediately.

A simple statement of zero trust is "never trust and always verify". The aim is to enforce security policies consistently and detect and respond to threats faster and precisely.

Zero trust is an industry wide initiative. For more information about the benefits of implementing a zero trust strategy, see the [Why use a zero trust model](#) section in [What is zero trust?](#)

The standard zero trust strategy covers the following portfolio of security capabilities:

- Identity
- Data
- Devices and workloads
- Analytics and visibility
- Automation and orchestration
- Network and endpoint

Implementation of a zero trust strategy therefore needs an enterprise wide approach with solutions to potential problems in each of these areas. For more information, see the [Zero trust minimum requirements](#) section in [What is zero trust?](#) Accordingly, you need to analyze each of the components in the enterprise before implementing a zero trust strategy.

To move to a zero trust strategy for CICS, you need to consider the following questions:

- [“What data needs protection?” on page 79](#)
- [“Who is accessing the data?” on page 80](#)
- [“What controls are available to protect the data?” on page 80](#)
- [“What user roles need to access the data?” on page 81](#)
- [“How to monitor who is accessing the data?” on page 81](#)
- [“What are the performance considerations for enabling resource security?” on page 81](#)
- [“How to audit to make sure best practices are used?” on page 81](#)

### What data needs protection?

CICS uses data from many sources. This includes external resource types (for example, Db2, VSAM, IMS DB, and MQ) and internal resource types (for example, temporary storage queues and transient data queues). Some resources, especially the internal resource types, are transient. Transient resources are less of a threat than the external resources, but they might still require protection from rogue

applications. Data held in internal storage such as containers is better from a security point of view, especially if transaction isolation is used.

You need to identify a list of resources that contain high-value data, such as payment card industry (PCI) data or commercially sensitive data. This list will most likely contain one or more of the external resource types, but might also include internal resource types.

CICS requires definitions for each of these resources. Resource definitions can be specified in the CICS system definition data set (CSD) or CICSplex SM data repository, or be defined dynamically. It is also useful to know what definitions are installed in each CICS region. You can use CICS Explorer to obtain such information.

When the list is complete, application architects can use the list to identify the content of every data source. As a result, the application architects can identify what user roles need which level of access (usually read or update) to each of the resources.

CICS provides security for resources but it doesn't provide security at any lower level, such as specified records or fields in a resource. If that is required, you are recommended to use Db2.

It is also useful to identify what resources are used in each region. Resources must only be installed, and accessible in the regions where they are used.

## Who is accessing the data?

Conceptually it is not a user that needs to access data, it is a user who has a specific role that needs access to this data in order to do their job. The zero trust statement on data access is "Enable the right user, to have the right access, to the right data, for the **right reasons**". The reason in this statement is the user's role. A principle of zero trust is that users have access to the data only if they have a need for their role. By default, users do not have access to data. They must only be granted access if their role requires it.

The initial stage of this process is obtaining the identity of the user. A user's request can enter CICS in several ways. You can define the entry points, such as URIMAPs, to verify the user's credentials or trust a user ID that has been verified by a trusted system connecting to CICS.

The need to identify the specific user that is accessing the data means that the use of functional user IDs does not conform to a zero trust strategy. A user ID must be flowed end-to-end in an application, wherever the application accesses data. This user ID can either be a RACF user ID, a distributed identity, or a Java security subject.

It is recommended that user IDs that access high-value data use "Multi-Factor Authentication (MFA)" on [page 22](#) as an additional level of protection. Define specific user groups to contain only user IDs that are defined with MFA.

## What controls are available to protect the data?

### Transaction security

Always configure CICS regions using transaction security. However, transaction security is in effect a form of boundary checking. Therefore, it's not sufficient for a zero trust strategy on its own. Unless additional controls are applied, a transaction can access any resource in the CICS region.

### Resource security

Transaction security is not sufficient for a zero trust strategy on its own. It is recommended that you use resource security to protect data that is accessed by an application for an additional layer of security. Enable resource security for all classes of resources that you need to protect in a region. For instructions about enabling resource security, see [Resource security](#).

**Note:** Although you can disable resource security by specifying the RESSEC(NO) attribute in transaction definitions, this does not conform to a zero trust strategy. You can specify the SIT parameter RESSEC=ALWAYS to override any transaction definitions with RESSEC(NO) attributes.

## Command security

System programming interface (SPI) commands can be used to discover installed resources or change their state. This can be exploited by a rogue user if they had access to these commands. It is recommended that you use command security to control who is allowed to issue the commands. Most applications do not need to issue SPI commands. For instructions about enabling command security, see [Command security](#).

**Note:** Although you can disable command security by specifying the CMDSEC(NO) attribute in transaction definitions, this does not conform to a zero trust strategy. You can specify the SIT parameter `CMDSEC=ALWAYS` to override any transaction definitions with CMDSEC(NO) attributes.

## What user roles need to access the data?

To ensure correct access for different user roles, you need to:

- Define the RACF classes that CICS uses.
- Create groups representing all the user roles and add the user IDs with those roles to the groups.
- Create RACF profile definitions for member lists containing groups of all the transactions, resources and commands that you need to protect.
- Grant appropriate access to the user groups representing the roles that need access to the member lists.

**Recommended:** [Generic profiles](#) are not recommended as part of a zero trust strategy because they require other controls to ensure that definitions are added during development.

Use universal access (UACC) only for resources that everyone is allowed to access, for example, a customer written sign-on transaction.

A special set of roles are system programmers and operators. These roles need access to create, install, monitor, and change resources, but should not have access to the data. CICSplex SM is commonly used by these roles. CICSplex SM uses a combination of resource object security and *simulated security* to enhance the security required by these roles. For more information, see [Activating simulated CICS security](#).

## How to monitor who is accessing the data?

With resource security and command security enabled as described in [“What controls are available to protect the data?” on page 80](#), you are in a position to monitor data access using [RACF SMF 80 records](#). These records include the resource, access level, user ID, and distributed identity (if used) along with many other fields to identify the user and their access patterns. This information can be used by products such as IBM Z Operational Log and Data Analytics to constantly monitor access patterns. For more information, see [SMF 80 data in IBM Z Operational Log and Data Analytics documentation](#).

System commands that change the state of installed resources are also audited with a message DFHAP1900.

## What are the performance considerations for enabling resource security?

For customers that currently only use transaction security, there will be a cost to implementing resource security and command security because of the increased number of RACF calls to authorize access.

You can estimate the performance cost of activating additional resource security by using [security discovery statistics](#).

## How to audit to make sure best practices are used?

If a region has been configured for the resource security and command security aspects of zero trust, you can refer to the best practices described in [compliance data collection to audit](#). You might need to follow detailed advice in [Center for Internet Security benchmark for CICS TS 6.1](#).

Check MRO and IPIC links to ensure that they are propagating the user ID on the flow. Do not use ATTACHSEC (LOCAL) for MRO, USERAUTH (LOCAL) for IPIC, or unauthenticated users in Liberty.

If you use user groups that contain only MFA users, check that all the user IDs in that group have MFA defined as their only access mechanism. You can use the **RACF LISTUSER** command to check that.

### Related information

[Implementing a zero trust strategy](#)

## How it works: CICS security discovery

CICS security discovery helps you identify security definitions required for resource security. It does this by analyzing your existing external security manager (ESM) definitions and the usage pattern of resources in your production regions. You can then manage and refine your security definitions based on its recommendations. RACF is used as an example ESM in CICS documentation.

### Why use CICS security discovery?

To conform with a zero trust strategy as well as various compliance regulations such as PCI-DSS, you need to protect your sensitive data. In CICS this means switching on resource security and command security in regions where sensitive data exists. For more information about zero trust, see [How it works: Zero trust in CICS](#).

**About the term:** To avoid repeating the phrase [resource security](#) and [command security](#), and because commands are just a type of resource, the following text refers to both by using the term *resource security*.

For regions that don't currently have resource security, methods used prior to the security discovery feature of migrating to resource security in production can be difficult. Any errors will result in applications being unavailable for some or all users, or even worse allowing users access to data that they shouldn't have access to. Other approaches such as using SMF records or RACF warning mode for all definitions can generate a large amount of data that is difficult to process. Therefore, a process is required that allows system programmers, security administrators, and application owners to safely review definitions based on the usage pattern and produce definitions that are safe to implement in a production region.

### An overview of the CICS security discovery process

The following diagram shows the process of using CICS security discovery to implement resource security.

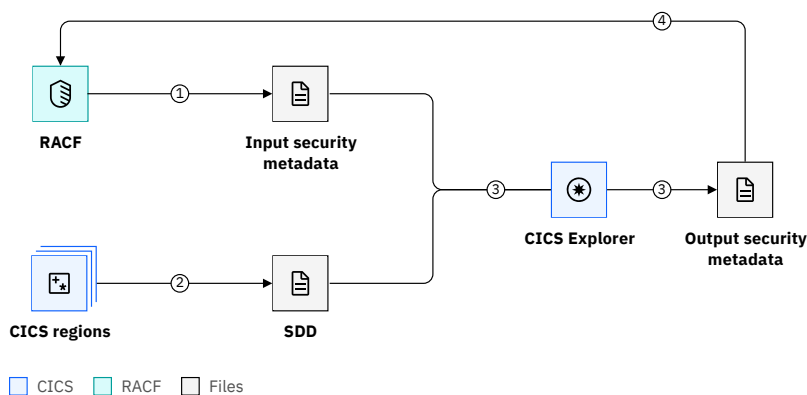


Figure 35. Overview of the CICS security discovery process

The steps involved are as follows:

### 1. Extract RACF definitions

A security administrator runs a JCL job to extract the group and the transaction class definitions from RACF and converts them into input security metadata in a readable, ESM agnostic format on zFS.

### 2. **6.2** Capture security discovery data (SDD)

A system programmer switches security discovery on in production regions that are to be converted to resource security to capture usage data in those regions. Security discovery needs to be left on for an extended period of several weeks to capture most user usage. This should include periods such as the end-of-quarter and end-of-year processing.

At the end of the capture period, the system programmer runs another JCL job to generate the usage data into SDD on zFS.

### 3. Analyze security definitions in CICS Explorer and export them for review

The system programmer transfers the files through FTP from zFS and imports them into the CICS Explorer. The **Security discovery** perspective in CICS Explorer can be used to identify and refine groups of users (roles), transactions, and other resources, one resource type at a time.

To simplify the task, the system programmer can use the transaction data in the security metadata and SDD to identify applications. They can then work on the data one application at a time.

After updating the security definitions in CICS Explorer, the system programmer exports them into output security metadata.

Multiple roles including the security administrator and application owner then need to review the output security metadata to ensure the security definitions are correct.

### 4. Create RACF commands from reviewed security definitions

The security administrator transfers the security metadata through FTP to zFS and runs a JCL job to convert it to RACF commands. These commands are used to create new classes. CICS regions can then have a staged migration with the new classes. RACF warning mode is recommended for the migration period so that any errors can be corrected without disrupting production usage.

The security discovery process can be integrated into the zero trust migration journey. For more information about each step, see [Migrating to zero trust with CICS security discovery](#).

## How it works: Capturing and validating security definitions during the development process

---

### **6.2** Applies to 6.2.

To conform to a zero trust strategy, it is vital that from a security perspective you must not only understand the model of security in your enterprise, but also be able to maintain it. CICS security definition capture (SDC) can capture the security definitions that would have been required if resource security and command security were turned on during the development process. It can run in a development region with minimal CICS security.

The required security definitions can then be reviewed and flowed through the development process to be implemented in production on a region with full CICS security. You can further incorporate the whole process into a CI/CD pipeline as part of your enterprise's DevSecOps strategy.

**About the term:** To avoid repeating the phrase resource security and command security, and because commands are just a type of resource, the following text refers to both by using the term *resource security*.

## What is DevSecOps, and how is SDC related?

DevSecOps stands for development, security, and operations. It's a methodology that extends DevOps. DevSecOps automates the integration of security at every phase of the software development lifecycle, from initial design through integration, testing, and deployment, to software delivery.

CICS security definition capture enables you to identify required security definitions and implement them earlier within the development process. If your enterprise is adopting a DevOps (or DevSecOps) approach, you can incorporate SDC into your CI/CD (continuous integration/continuous delivery) pipeline to align with the DevSecOps strategy.

## Why capture security definitions using SDC?

To conform with a zero trust strategy as well as various compliance regulations such as PCI-DSS, you need to protect your sensitive data. In CICS this means switching on resource security and command security in regions where sensitive data exists. For more information about zero trust, see [How it works: Zero trust in CICS](#).

When writing new applications or changing applications, it is necessary to identify the security definitions required for promoting the application into a production environment.

One way of doing this is to identify all of the definitions before starting development, implement them on a test system, and then develop and test the application. Testing might identify changes to the security definitions, which will then need to be implemented before further work can be done. This can result in a lengthened software development lifecycle and hence inefficient product delivery.

With CICS security definition capture, development can take place on an isolated test system with minimal security. When the developer changes the application or a tester runs the automation tests, CICS identifies the required security definitions. This can then be implemented on a preproduction test system and the application further tested with all security correctly configured first time. This process minimizes the chances of definitions being incorrect or even worse, over permissive.

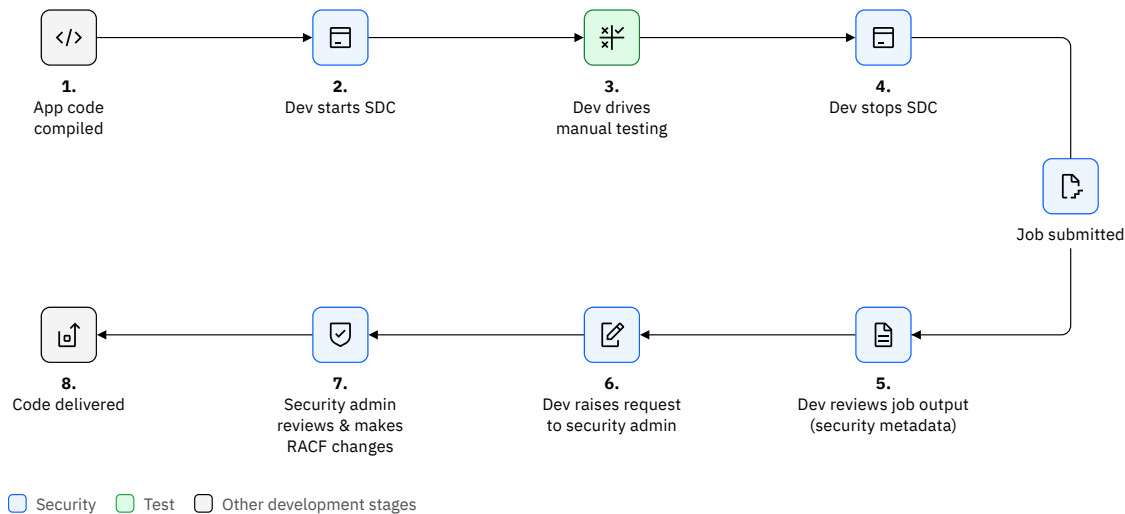
If security definition capture is incorporated into a CI/CD pipeline as part of the DevSecOps approach, the whole process can be automated so that the required security definitions can be captured, reviewed, and implemented on a continuous basis.

## Manually capturing security definitions using SDC

Although automating the process of capturing and validating security definitions renders added value, you can still simplify your zero trust maintenance by manually capturing required security definitions using SDC before reviewing and promoting them.

The following diagram shows the process of generating RACF definitions to implement resource security based on the output from CICS security definition capture, where RACF can be replaced by any other supported ESM.

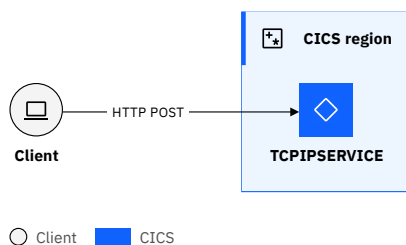




*Figure 36. Flow of capturing security definitions using SDC*

The flow is as follows:

1. The application is written or changed, compiled, and then passes basic unit testing.
2. Prior to testing, the developer starts SDC for the developers' test entry point. SDC provides two interfaces: the CXSD transaction and the HTTP REST interface. This is done on a region with SEC=YES, but all other security definitions are optional.
  - For 3270 applications, the developer signs on and invokes CXSD ON. This captures security definitions required by the 3270 terminal based on its usage.
  - For the other applications, the developer sends an HTTP POST request to the TCPIP SERVICE installed in the CICS region, as shown in the following diagram. The user ID used for requests to the TCPIP SERVICE must be the same as that used for testing.



*Figure 37. Starting SDC through the HTTP interface*

3. The developer runs through the test paths. The test environment might be a single region or a collection of regions (TOR, AORs, FORs, and so on). These tests exclude specific tests that are to test security failures.
4. The developer stops SDC:
  - For 3270 applications, the developer invokes CXSD OFF.
  - For the other applications, the developer sends an HTTP DELETE request to the TCPIP SERVICE installed in the CICS region.
5. When SDC stops, CICS automatically submits a job to produce the security metadata that contains required security definitions. The output is sent to either the developer or a security administrator depending on the configuration. The remaining steps assume the output goes to the developer.
6. The developer reviews this security metadata to verify whether it meets the expectation.

7. The developer then submits this file, or a copy of the information, to the security administrator.  
Developers must include the roles who need to access the application. Such roles need to replace the user ID(s) that are used in the testing.
8. The security administrator reviews the information and creates the RACF commands to update the database.
9. The application code can then be delivered.

After this process, it is expected that there will be preproduction testing with security on. This will test out the new security definitions including any error paths that will result in security failures.

The definitions and the application can then be moved to production.

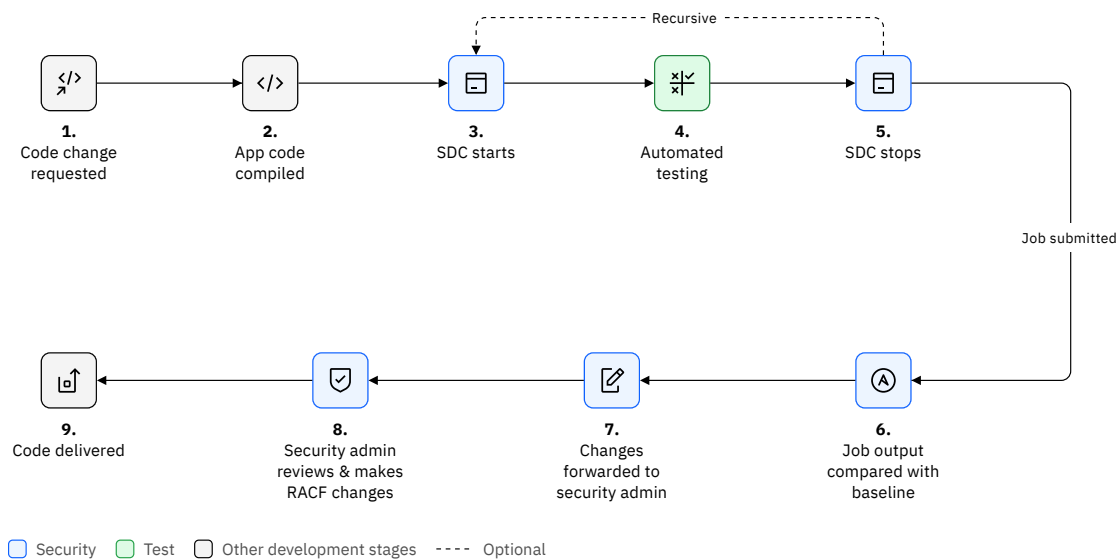
For more information about running SDC, see [Capturing security definitions during development using security definition capture](#).

## Moving towards a DevSecOps process

Further value can be obtained by automating some of this process.

For example, you can use Galasa SDV Manager to automate the testing, starting and stopping of SDC, and the generation of the security metadata as part of your CI/CD pipeline. Other DevSecOps tools like Ansible® and GitHub can be used to automate the comparison and validation of security definitions.

The following diagram shows one example:



*Figure 38. Automating the starting and stopping of SDC to identify security definitions in a DevSecOps process*

As a prerequisite to this example, a full automated regression test suite run is performed with SDC on, submitting a job for each test class. This creates security metadata for each test class, acting as the security baseline to detect differences from.

Each time a developer makes a request to change the application or test code, the pipeline compiles the code and then runs automated regression tests against the code changes, generating updated security metadata files. These files can be compared against the baseline files to identify differences. The security administrator can then review these differences and make the security changes

CICS provides an example implementation of a full CI pipeline called security definition validation (SDV). It utilizes SDC within its automated testing stage and initiates an approval process. When changes to security are identified, the approval process must complete before code is delivered. For more

information about the example, see [Introducing Security Definition Validation to your CICS TS Application CI pipeline](#).

**Related tasks**

[Capturing security definitions during development using security definition capture](#)

[Configuring security definition capture \(SDC\)](#)



---

## Chapter 9. How it works: Securing CICS with RACF

Resource Access Control Facility (RACF), part of z/OS' security server, is an IBM provided security manager. Other external security managers (ESMs) are available and provide similar functions. The CICS TS documentation refers only to RACF unless it is to explain the interfaces to ESMs.

For more information about the terminology used by other ESM tools, see the following:

- [IBM RACF to ACF2 Translation](#)
- [IBM RACF-to-Top Secret Translation](#)

RACF provides the following facilities:

- The necessary functions to record information that identifies individual *users* who access system resources, and information that identifies the resources that require protection.
- The facilities to define which users, or groups of users, are either permitted a level of *access*, or excluded from access, to the resources for which profiles are defined.
- A method to process requests to *authenticate* the identity of users who are defined to RACF.
- A method to process requests to check a user's access *authorization* to resources.
- The facilities for logging security-related events, such as authentication and authorization to the MVS System Management Facility (SMF).

The basic concepts and terms of RACF security, as they apply to CICS, are as follows:

### Resource profile

Sometimes simply referred to as a *profile*. It defines resources to be protected.

A resource profile can protect a discrete or generic resource (for example, a single transaction CEMT or all resources starting with AB) or a resource group (for example, a named member list containing multiple resources).

### Class

Also called a *class profile*, contains a set of resource profiles for resources of the same type. For example, transactions.

### User

Also called a *user profile*, defines characteristics about a user.

### Group

Also called a *group profile*, contains a set of users with the same role, for example, bank tellers. A user can be associated with one or more roles.

A group can be given a level of access to a resource profile. For example, bank tellers can run the banking transaction.

When CICS tries to perform an operation that is subject to a security check, it calls RACF. RACF checks whether the user (in the group) has at least the level of access that is required to perform that operation on the profile that represents that resource. Checks can be audited in SMF records, and failures can result in ICH408I security failure messages.

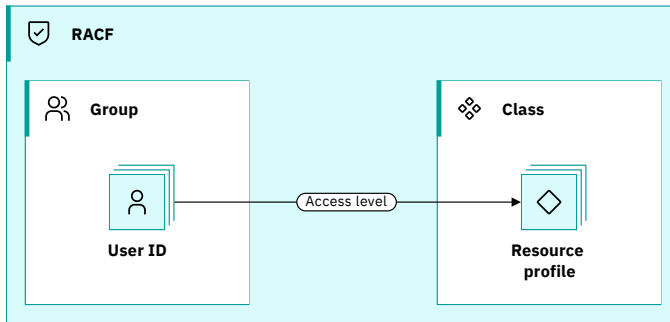


Figure 39. RACF security elements

To learn more about RACF basics, watch the 9-minute video: [IBM Security: Resource Access Control Facility \(RACF\) Introduction](#).

## An example of basic concepts in RACF

The following example explains RACF concepts of user IDs, groups, and the access they have to resource profiles in classes and how these relate to CICS resources such as transactions.

The user OCTAVIA is a system administrator who should have access to run the CEMT transaction. In CICS terms, CEMT is a named resource, in this case a transaction.

In RACF, access is controlled by resource profiles in a class. A resource profile is a named resource. In this example, the resource profile is the transaction name CEMT, which can be specifically defined in the general resource class TCICSTRN.

The following RACF definitions specifies OCTAVIA has access to run the CEMT transaction:

```
RDEFINE TCICSTRN 2 CEMT 3 UACC(NONE) 4
PERMIT CEMT CLASS(TCICSTRN) ID(OCTAVIA) ACCESS(READ) 6
```

### Understand the snippet:

1. The order is important. First you define a resource profile in a class using the **RDEFINE** command. Then you permit users to access it using one or more **PERMIT** commands.
2. TCICSTRN is a class that contains a set of resource profiles representing resources.
3. The **RDEFINE** command explicitly uses the transaction name as the resource profile. CEMT is the resource profile that protects the transaction CEMT.
4. UACC(NONE) means that no one has access unless they are explicitly permitted to have access. UACC other than none means that everyone needs some access for their job, such as a transaction that displays help information.
5. The format of the **RDEFINE** and **PERMIT** commands are different, but the TCICSTRN class and the CEMT resource profile are the same in both.
6. The user ID OCTAVIA is given READ access to the CEMT profile in the TCICSTRN class. To CICS, READ means OCTAVIA can run the CEMT transaction.

Therefore, with that definition alone, only OCTAVIA has access to run the CEMT transaction.

In most cases, OCTAVIA is one of several system administrators. Instead of issuing **PERMIT** commands for each of the system administrators, it is recommended to define OCTAVIA in a group of system administrators, and issue **PERMIT** for the whole group:

```
ADDGROUP ADMIN 2
CONNECT OCTAVIA GROUP(ADMIN) 3
CONNECT OLIVER GROUP(ADMIN) 3
```

### Understand the snippet:

1. The order is important. First you define a group using the **ADDGROUP** command. Then add users to that group using one or more **CONNECT** commands.
2. ADMIN is the name of the group. It can be used in subsequent RACF commands in the same way that a user ID is used.
3. The **CONNECT** commands add OCTAVIA and OLIVER to the ADMIN group.

After you have defined the ADMIN group, you can define their permission to run CEMT as follows:

```
RDEFINE TCICSTRN CEMT UACC(NONE)
PERMIT CEMT CLASS(TCICSTRN) ID(ADMIN) ACCESS(READ) 1
```

### Understand the snippet:

1. The **PERMIT** command gives READ access to the CEMT profile for all users in the ADMIN group.
2. There is no difference in the format of the **PERMIT** command to define access for users or groups.

By granting the ADMIN group access to multiple resources, you can allow all system administrators access to those resources. If OCTAVIA changes jobs and stops being a system administrator, it's easy to remove her from access to all the system administrator resources by removing her from the group using the following command:

```
REMOVE OCTAVIA GROUP(ADMIN)
```

The transaction CEMT is just one of several transactions that a system administrator needs to use as part of their job. Rather than adding definitions for each of these transactions, it is simpler to define the set of transactions in a group of resources known as a *member list*. This is shown in the following command:

```
RDEFINE GCICSTRN 1 SYSADM 2 ADDMEM(CEMT,CESD,CETR) 3 UACC(NONE)
PERMIT SYSADM CLASS(GCICSTRN) ID(ADMIN) ACCESS(READ) 4
```

### Understand the snippet:

1. The class name GCICSTRN has a prefix of G. This class is related to the TCICSTRN class mentioned before. Whereas TCICSTRN contained members, GCICSTRN contains member lists (resource groups). For more information about different types of resource class, see [“RACF classes and profiles for resources”](#) on page 96.
2. SYSADM is the name of the member list containing the group of transactions.
3. The ADDMEM option lists the transactions.
4. The **PERMIT** command allows all the users in the ADMIN group access to run all the transaction in the SYSADM member list.

Using groups of users and groups of transactions makes subsequent administration a lot easier. For example, to add transaction CEDA to the list of transactions in the SYSADM member list, you issue the following command:

```
RDEFINE GCICSTRN SYSADM ADDMEM(CEDA)
```

This gives all the users in the ADMIN group access to run CEDA.

Each CICS region or set of CICS regions is configured to use these classes by using the **Xnnn SIT** parameters. For example, the transaction classes that a CICS region uses is defined by the [XTRAN SIT parameter](#). Classes come in pairs, a *general resource class* that contains members and a *resource group class* that contains member lists. In this example, these are TCICSTRN and GCICSTRN. The SIT parameter that configures CICS to use these transaction classes is XTRAN=CICSTRN. RACF knows that it needs to add the prefixes T and G to the name specified in the SIT parameter to find the name of the general resource class and the resource group class.

In practice, after you have updated all the profiles in a class, you need to refresh the RACF cache for the changes to take effect. For instructions, see [Caching of RACF classes and their profiles](#).

## RACF administration

---

The RACF security administrator is responsible for ensuring that an installation's data is properly protected. This is achieved by protecting all system resources and specifically CICS resources.

A key feature of RACF is its hierarchical management structure. This allows delegation of tasks from a RACF security administrator to other users who have administrator authority over a subset of the resources.

The roles are:

### **RACF security administrator**

Users with this role have the authority to control security for the whole system. They have the group-SPECIAL attribute that permits them to issue any command, except for some audit-related commands.

### **RACF group administrator**

This role can add or remove users from a group. They have a group-SPECIAL attribute or are the owner of a group. This configuration limits their commands to the groups for which they are responsible.

### **RACF class administrator**

Users with this role are able to define resource profile in a class, and give users or groups access to the profiles. They have the CLAUTH (class authority) attribute or are the owner of a class. This configuration limits their commands to the classes for which they are responsible.

### **RACF auditor**

Users with this role can produce reports of security-relevant activity based on auditing records that are generated by RACF.

For more information about the authorities that are required to issue RACF commands, and for information on delegating authority and on the scope of a RACF group, see the [z/OS Security Server RACF Auditor's Guide](#).

For more information on the requirements for issuing RACF commands, see the descriptions of the commands in the [z/OS Security Server RACF Command Language Reference](#).

## **Separation of duties**

A key security principle is the separation of duties between different users so that no one person has sufficient access privilege to perpetrate damaging fraud. This configuration is required by various audit regulations such as the United States Federal Law known as the [Sarbanes-Oxley Act of 2002](#).

An example of this separation of duties, is that someone with the role of CICS system programmer must not also have the role of RACF security administrator.

## RACF users and roles

---

A user ID is an identifier associated with an end user, or with functions provided by the CICS region. User IDs have profiles that describe the users. A group profile describes a set of users with the same role. A user ID can be associated with one or more roles.

For a basic example of defining the security elements of RACF (users, groups, resource profiles, and classes) and their interrelationships, see [How it works: Securing CICS with RACF](#).

### **User profiles**

A user profile is a description of a user in the RACF database. The information in the profile includes the user ID, the username, the user's credentials or authentication mechanisms, the profile owner, user attributes, and other data. The user profile also contains user-related information for subsystems, including CICS.



The user profile consists of a base segment and optionally, a number of extra segments. The segments relevant to CICS are as follows:

**The base segment in user profiles**

Contains basic information that is needed to define a user to RACF, such as the user ID, the user name, the user's authority, the user's credentials, and so on.

**“The CICS segment in user profiles” on page 93**

Contains data for CICS users.

**“The LANGUAGE segment in user profiles” on page 94**

Holds information about the national language in which the user receives CICS-issued messages.

To read these profiles, it is necessary for the CICS region user ID to have READ access to a profile that defines the relevant field in the segment. For more information, see [“Defining authorities required by security administrators” on page 102](#).

## **Roles (group profiles)**

A group profile, also simply called a group, defines a group of users who have the same role. Users in a group have access to all the resources to which the group has access. Users can belong to one or more groups. A group can also contain other groups.

A group profile can contain information about the group, for example, who owns it, which subgroups it has, and a list of connected users. For more information about how to define and use group profiles, see the [z/OS Security Server RACF Security Administrator's Guide](#).

Users who are members of the same group can share common access authorities to protected resources. For example, you might want to set up groups as follows:

- Users who have the same role, such as bank tellers.
- Users who work in the same department with the same role.

In a CICS environment, group profiles offer a number of advantages:

- Easier control of access to resources because you define only the group or groups that have access to the resource.
- Easier management of role-based security because membership of a group decides a user's access.
- Fewer refreshes to in-storage profiles.

If you connect or remove a user from a group that is already in the access list, that user acquires or loses the authority of the group. This situation is achieved without needing to refresh the profile. By default, CICS is automatically notified if a user ID is added or removed from a group. This notification occurs so that its security tokens are refreshed to use the user ID's current level of access. However, for users who signed onto 3270 terminals, it is necessary to sign off and sign back on again for this condition to take effect.

**Recommendation:** Define access lists in resource profiles for groups rather than individual user IDs. This way, when someone leaves a department or stops performing a role, you can remove the user ID from the group to revoke all privileges owned by the user. No administration of profiles is required. Thus, you keep RACF administration to a minimum.

For more information about other benefits obtained from creating groups, see the [z/OS Security Server RACF Security Administrator's Guide](#).

## **The CICS segment in user profiles**

The CICS segment of the RACF user profile contains data for CICS users. The information is only applicable to signed on 3270 terminals.

The fields can be set for an individual user, default values can also be set and some information can be supplied at signon. For details about the order in which this information is used see [Obtaining CICS-related data for a user](#).

The fields that you can specify in the CICS segment are as follows:

**OPCLASS({1|number})**

CICS uses the operator classes when routing basic mapping support (BMS) messages initiated within a CICS transaction. The operator classes are numeric values in the range 1–24.

Specify operator classes for users who use CICS transactions that issue EXEC CICS ROUTE commands with the (optional) OPCLASS parameter. For automatic routing to occur, you specify the corresponding value as an operator class in the CICS segment of the user profile.

**OPIDENT({blank|name})**

The 1- to 3-character operator identification code that you assign to each operator.

This operator ID is displayed in certain CICS messages and can also be used in the EXEC CICS ROUTE command for routing BMS messages. It is also used when using the CEDA LOCK command.

**OPPRTY({0|number})**

The operator priority value is a decimal number that you want CICS to use when determining the task priority for CICS transactions that the operator invokes at a CICS terminal. The priority value can be in the range 0 through 255, where 255 is the highest priority.

CICS uses the sum of operator priority, terminal priority, and transaction priority to determine the dispatching priority of a transaction.

**TIMEOUT({0000|hhmm})**

The time that must elapse since the user last used the terminal before CICS "times-out" the terminal.

The time must be a decimal integer in the range 0 through 9959 (the last two digits represent a number of minutes, and must be 00 through 59. Any digits to the left of these represent hours).

**XRFSOFF({NOFORCE|FORCE})**

XRF is stabilised (see [Stabilization notices and discontinued functions](#)).

**Related information**

[Adding a CICS User with the default CICS options](#)

[Creating or updating segment data for a CICS user](#)

[Message routing](#)

## The LANGUAGE segment in user profiles

The language segment holds information about the national language in which the user receives CICS-issued messages. When a user specifies a national language during sign-on, the sign-on option overrides the language specified in the user's RACF CICS segment.

You can specify two languages, but CICS assigns each user only one language. CICS assigns the primary language if the language is specified and corresponds to the single-character code specified on the **NATLANG** system initialization parameter. Otherwise, CICS assigns the secondary language by the same standard. If both the primary and the secondary language do not correspond to a **NATLANG** value, the language must be provided from elsewhere. See [“National language and non-terminal transactions” on page 95](#).

The information that you can specify is as follows:

**LANGUAGE**

Specifies primary and secondary languages for CICS users. CICS accepts and uses the languages that you define in the segment. If these values are not specified, the system default applies.

**PRIMARY(language\_code)**

Specifies the user's primary language.

**SECONDARY(language\_code)**

Specifies the user's secondary language.

**Notes:**

1. CICS messages are supported only in US English, Simplified Chinese, and Japanese. If any other language other is specified, English is used by default.
2. CICS ignores the RACF default national language that is defined by the SETROPTS command

## National language and non-terminal transactions

When a user specifies a national language during sign-on, the sign-on option overrides the language specified in the user's RACF CICS segment.

The language thus specified is set for the time that the user is signed on at the terminal. Any transaction invoked by the signed-on user runs with the national language specified on the sign-on.

However, if a transaction uses the [START](#) command to start another transaction, the national language attribute for the started transaction is derived as follows:

1. If the USERID parameter is specified on the START command, the national language is taken from the RACF CICS segment of the specified userid.
2. If the user is signed on at a terminal with a preset national language specified on the terminal definition, this preset national language is assigned to the started transaction.
3. If there is no userid on the START command, and no preset national language on the terminal, the started transaction inherits the national language specified in the RACF CICS segment of the signed-on user (not the national language used in the sign-on).

If the national language of the original terminal is required, the terminal's national language can be inquired about before the START command is issued. The information can then be passed as data in the START command for the use of the transaction that has been started.

### LANGUAGE segment in z/OS Security Server RACF Security Administrator's Guide

[The LANGUAGE segment in user profiles](#)

## Obtaining CICS-related data for a user

CICS obtains CICS-related data from the RACF profile by calling RACF when a 3270 user signs on to CICS. CICS obtains the same information for the CICS default user ID at CICS initialization.

CICS obtains the user information in the following order:

1. If the CICS segment or the LANGUAGE segment data is present for the signed on user, RACF returns this data to CICS.
2. If none of the segments are available, CICS uses the information of the default user, which is defined during system initialization.
3. If none of the previous information is available, the following default values are used:
  - National language from the [NATLANG](#) system initialization parameter
  - OPCLASS=1
  - OPIDENT= ' '
  - OPPRTY=0
  - TIMEOUT=0

## Changing the RACF profile of a user

CICS can be notified when certain changes are made to a user ID. These changes include when a user ID is added or removed from a group, and also when a user ID is revoked.

The mechanism to notify interested parties that a user ID is affected is a [type 71 ENF](#). RACF sends type 71 ENFs under the following circumstances:

- A user ID is revoked that uses the RACF ALTUSER command with the REVOKE option.
- The user ID is revoked when an incorrect password is tried too many times.

- A user ID is deleted by the RACF DELUSER command.
- A user ID is added or removed from a group that uses the RACF CONNECT or REMOVE command.
- A group that includes the user ID is deleted by using the RACF DELGROUP command.

The effect of the notification is to discard any cached copy of a security token for the user ID. It does not affect inflight tasks or signed on users. Notification of a change to the user ID overrides any setting that is specified in the USRDELAY system initialization parameter.

When a RACF profile change occurs and CICS receives a new request to run a transaction that uses this user ID, CICS performs an implicit sign-on for the user ID and the new RACF profile information is used.

## RACF classes and profiles for resources

---

In RACF, a resource profile defines resources to be protected. And a class, or a class profile, contains a set of resource profiles for resources of the same type, such as transactions or files. RACF classes and resource profiles are the means of defining resources that need to be protected. Users or groups of users can then be assigned the access that they need to those resources.

For a basic example of defining the security elements of RACF (users, groups, resource profiles, and classes) and their interrelationships, see [How it works: Securing CICS with RACF](#).

Two types of class exist: *general resource classes* and *resource group classes*. The former contains discrete profiles or generic profiles protecting members. The latter contains named profiles protecting groups of resources (member lists).

### General resource classes and members

*General resource classes*, also known as *member classes*, can contain either discrete profiles or generic profiles.

#### Discrete profile

Protects a single resource, for example, a transaction name CEMT.

#### Generic profile

Protects several resources with similar names. Wildcard characters \* and % are used to define these profiles. For example, C\* represents all resources that start with letter C. See [Other rules for generic profile names in z/OS Security Server RACF Security Administrator's Guide](#) for details.

To use generic profiles, it is necessary to be rigorous in maintaining naming conventions. Naming conventions can help minimize the risk of problems when you combine systems, such as during a company merger.

Resources protected by such profiles, either discrete or generic, are also referred to as *members*.

General resource classes are sometimes easier to manage, but require complete adherence to a strict naming convention. This configuration can lead to complexity when company mergers require the integration of two different naming conventions.

### Resource group classes and member lists

*Resource group classes*, also known as *member list classes* or *grouping classes*, include named profiles protecting sets of resources. These sets of resources are also referred to as *member lists*.

Member lists are often associated with a user role to manage access based on roles. For example, the SYSADM member list in the CICS supplied DFH\$CAT2 CLIST contains a set of associated transactions (CEDA, CEMT ...) that can be used by users with a system administrator role.

Resource group classes are generally safer to manage since it is possible in theory to identify all profiles for each application. With that knowledge, you can limit user access to only the resources of the specific applications that they are allowed to use.

If duplicate resources exist in the resource groups, these are merged according to the algorithm defined in [Resolving conflicts among grouping profiles](#).

The following diagram shows some example profiles and classes in RACF, where TCICSTRN and GCICSTRN are both classes for transactions.

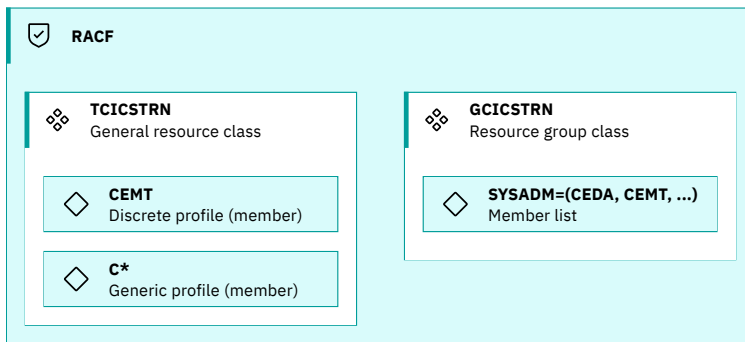


Figure 40. RACF classes and profiles for resources

## Class pair

Most CICS resources are protected by a pair of associated classes: a general resource class and an associated resource group class. They have the same class name apart from the initial character (prefix) that is used to identify the type of class. For example, TCICSTRN is the general resource class for transactions while GCICSTRN is the resource group class for transactions.

You configure a CICS region or set of CICS regions to use RACF classes and optionally the class names with the **Xnnn** SIT parameters. RACF knows that it needs to add different prefixes to the name specified in the SIT parameter to find the name of the corresponding general resource class and the resource group class.

For more information, see [Summary of RACF classes for CICS resources](#).

## Breakdown of a resource profile

A resource profile includes the following information:

### Profile name

A description of the resources protected. The profile name depends on the type of class that it relates to.

If it's a generic profile or discrete profile in the general resource class, the profile name is derived from the protected resource or resources, for example, CEMT or C\* in the previous example.

If it's a resource group profile (member list) in the resource group class, a name can be specified, for example, SYSADM in the previous example. Usually, a profile name consists of eight uppercase alphanumeric characters. However, when defining classes in RACF, you can define alternative naming conventions for profiles. See [ICHERCDE macro](#).

### Profile owner

The security administrator who is responsible for maintaining the profile.

### Universal access authority (UACC)

The default access for users not in an access list. This access is usually defined as NONE.

### Access list

- The users or groups that are permitted to access the resource.
- The level of access that each user or group is allowed.

**Recommendation:** Define access lists in resource profiles for groups rather than individual user IDs. This way, when someone leaves a department or stops performing a role, you can remove the user

ID from the group to revoke all privileges owned by the user. No administration of profiles is required. Thus, you keep RACF administration to a minimum.

### Reference

[z/OS Security Server RACF Security Administrator's Guide](#)

[Defining profiles for general resources](#)

[Creating resource group profiles](#)

## Caching of RACF classes and their profiles

Refresh the RACF cache when the definition of a profile is changed.

CICS's RACF classes are RACLISTed, which means that their profiles are cached for performance. If the profiles in a class are changed, it is necessary to refresh the cache.

Taking the transaction class as an example, this change is done by issuing the following command:

```
SETROPTS RACLIST(TCICSTRN) REFRESH
```

The general resource class name (TCICSTRN) must be specified regardless whether the profile that's changed is in the general resource group (TCICSTRN) or the associated resource group class (GCICSTRN). The resource group class is not specifically RACLISTed.

### Related concepts

[How it works: Securing CICS with RACF](#)

### Related tasks

[Refreshing RACF profiles for CICSplex SM](#)

### Related reference

[Summary of RACF classes for CICS resources](#)

[Security administration tasks and the RACF commands to run them](#)

## Defining generic profiles for resources

A zero trust strategy means that resources should be protected by only granting groups (roles) access to member list profiles. The member lists then contain the sets of resources that users in the group have a specific need to access. Whilst migrating to this strategy, there can be regions where the security are based on a naming convention for the resources rather than listing all of the resources.

For example, there might be just a few programs in the CICS application program resource class that are particularly sensitive, and a much larger number that constitute no significant risk. In this case, you could protect the few by defining specific RACF profiles for only those programs that are sensitive. You ensure that everyone can access the remaining, nonsensitive, programs by defining a completely generic resource profile, as follows:

```
RDEFINE MCICSPPT * UACC(READ) ...
```

This profile applies to any authorization request for programs not covered by one of the specific profiles. RACF processing logic is such that the most specific profile for any given resource name is always used.

For an example of defining other types of profile, see [How it works: Securing CICS with RACF](#).

Note that to determine whether a profile is generic, you need only check if 'G' appears after the name of the profile when it is listed with RLIST or SEARCH. For example:

```
SEARCH CLASS(TCICSTRN)
```

might give the following output:

```
C*  
CED% (G)  
** (G)
```

This output shows that both CED% and \*\* are generic profiles. The C\* profile is not generic because it is not followed by (G). This could have occurred if the C\* profile was created before generic profiles had been enabled with a SETROPTS command. The C\* profile can be deleted and redefined as a proper generic profile as follows:

```
SETROPTS NOGENERIC(TCICSTRN)
SETROPTS NOGENCMD(TCICSTRN)
RDEL TCICSTRN C*
SETROPTS GENERIC(TCICSTRN)
RDEFINE TCICSTRN C* UACC(NONE)
```

## Access to all or access to none?

If RACF can find neither a specific nor generic profile, it returns a `no profile found` condition.

CICS treats this return code exactly the same as the `user not authorized` return code, and returns the `NOTAUTH` condition to the CICS application program. If RACF cannot find the APPL class, it returns a `READ access intent` condition.

You can either use the completely generic profile to permit access to any resources not otherwise covered by more specific profiles, or, to prevent any access, use the `UACC(READ|UPDATE)` or `UACC(NONE)` options. For example,

```
RDEFINE DCICSDCT * UACC(NONE)
```

prevents access to any transient data queue not covered by any of the other profiles defined to RACF, and results in RACF writing an SMF record.

On the other hand, you can define files as "public" by the following command:

```
RDEFINE FCICSFCT * UACC(READ)
```

If you are using generic profiles, ensure that generic profile checking has been activated for the CICS RACF resource classes (both the IBM-supplied classes and any installation-defined classes added to the RACF class descriptor table) by issuing a `SETROPTS GENERIC(classname)` command for any one of the CICS classes having the same POSIT value. This ensures generic checking for all other CICS classes with the same POSIT value. If you change a generic profile, you must issue a `SETROPTS GENERIC(classname) REFRESH` command. For more information about POSIT values and defining generic classes, see the [z/OS Security Server RACF System Programmer's Guide](#).

## Defining your own resource classes

You can also define your own resource classes for user-defined security checks using the `QUERY SECURITY` command.

If you don't have a suitable class defined in RACF, see [Setting up installation-defined classes in RACF](#).

Defining your own resource class names can have the following benefits:

### Controlling access from other regions

You can prevent users who work in one CICS region from accessing the resources of other CICS regions that have different class names specified. (You can also achieve this outcome by using prefixing; see [SECPRFX system initialization parameter](#).)

### Group administrator for each region

For each CICS region with installation-defined classes, you can authorize a different group administrator to create profiles to be used by that region.

To get this benefit, define the installation-defined classes with a POSIT number other than 5 (the POSIT number of the IBM-supplied CICS classes). Then give the group administrator the `CLAUTH` (class authority) for at least one of those classes.

Use the `SETROPTS GENERIC` command before defining generic profiles, as described in [Security administration tasks and the RACF commands to run them](#).



With prefixing active, you can also assign different administrators without fear of conflict. To achieve this outcome, create a generic profile in each class, by using the prefix as a high-level qualifier. For example:

```
RDEFINE TCICSTRN cics_region_id.** UACC(NONE)
OWNER(cics_region_administrator_userid)
```

The administrator specified as the OWNER of each such profile can create and maintain more specific profiles. The other administrators cannot do so.

## Setting up installation-defined classes in RACF

To set up installation-defined classes, work with your RACF system programmer to add new class descriptors to the installation-defined part (module ICHRRCDE) of the RACF class descriptor table (CDT).

For details about adding installation-defined classes to the CDT, see [Adding installation-defined classes to the static class descriptor table](#).

All installation-defined classes that are defined in the CDT must also be defined in the MVS router table. This requirement is because the MVS router checks any class that is used in a router request to determine whether it exists. If it does not, no request is sent to RACF. To define classes to the MVS router, add them to ICHRR01, the user-modifiable portion of the MVS router table, as described in the [z/OS Security Server RACROUTE Macro Reference](#).

When you set up installation defined classes, it is recommended that you copy the IBM-supplied defaults from the CDT, an example of which is in [z/OS Security Server RACF Macros and Interfaces](#). You then need to change the name, group or member name, POSIT number, and ID. See the description of the ICHERCDE macro in [ICHERCDE macro](#) for details of valid values for these operands. For more information about creating installation-defined resource classes, see the same manual. For an example of how to add resource classes, see the sample, DFH\$RACF in the CICSTSnn.CICS.SDFHSAMP library, where CICSTSnn is your version of CICS. As an example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is CICSTS62.CICS.SDFHSAMP.

For CICS resources, the first character of the resource class name is predefined by CICS, consistent with the default resource class name. You can define the second through eighth characters of the resource class name. You must specify the same characters for both the member class (general resource class) and the grouping class (resource group class). The seven characters that are specified for the member class are the part of the resource class name you define to CICS in the various Xnnn parameters, except for the following instances:

- XDB2, which has no CICS-defined prefix letter, so any defined class name of 1- to 8-characters can be specified. It is recommended that you use a specific class or classes that are dedicated to these resources.
- XAPPC and XUSER, which have no name option, and are either YES or NO to control whether security is active or not.

You must avoid using the letters "CICS" in the second through fifth characters in any class name you define. RACF requires that at least one of the characters in the class name must be a national or numeric character.

## Resource classes for DB2ENTRY resources

CICS supports resource security checking for CICS-defined DB2ENTRY resources, for which there are no IBM-supplied RACF resource classes.

For DB2ENTRY resources, you define security profiles in user-defined class names, and use the **XDB2** system initialization parameter to specify the class name to CICS.

As an example of how to define Db2 resource class names for CICS, use the DFH\$RACF sample job in the CICSTSnn.CICS.SDFHSAMP library, where CICSTSnn is your version of CICS. As an example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is CICSTS62.CICS.SDFHSAMP.



## Prefixing RACF profiles with SECPRFX

Use SECPRFX to increase the granularity of security rules across CICS regions.

By default, all CICS regions that use the same classes also use the same profiles. This configuration means that the same security rules are applied to all regions and although it is the simplest way to configure regions, it might not fit your requirements. Examples can include:

- Regions or sets of regions might have resources that have the same names as other regions.
- A single RACF database is used for different types of region usage, such as production and QA.

The SECPRFX SIT parameter is used to define a prefix that is inserted at the beginning of the name of the resource that is being checked. The prefix can be defined explicitly or set to match the CICS region user ID.

For example, if SECPRFX is set to PROD and a check is made on transaction ABCD, then CICS checks that the user has the appropriate authority on the profile PROD.ABCD in the transaction class.

Prefixes apply to all CICS classes, both general resource classes and resource group classes.

### Related reference

[SECPRFX system initialization parameter](#)

## RACF data set profiles

Using RACF facilities, you can protect data sets by defining profiles for the data sets you want to protect. Data set profiles apply only to the CICS region user ID, not to terminal users.

The rules for defining data set profiles to RACF are described in the [z/OS Security Server RACF Security Administrator's Guide](#), and the [z/OS Security Server RACF Command Language Reference](#). For examples, see the [z/OS Security Server RACF General User's Guide](#).

You define profiles to protect two RACF categories of data sets:

1. Profiles for **user data sets**, where the high-level qualifier is a RACF userid. All RACF-defined users can protect their own data sets.
2. Profiles for **group data sets**, where the high-level qualifier is a RACF group name (see [“RACF users and roles”](#) on page 92 for information about RACF groups). A RACF-defined user can RACF-protect group data sets provided the user has the necessary authority or attributes. (See the [z/OS Security Server RACF Security Administrator's Guide](#) for details.)

## Controlling access to fields in RACF profiles

Use the FIELD resource class to define profiles that control access to fields in the RACF database.

By creating profiles in the RACF FIELD class, in the following form, you can permit listing or updating of the CICS or LANGUAGE segments in the user profiles, and of appropriate fields in partner-LU profiles.

```
USER.CICS.OPIDENT
USER.CICS.OPCLASSN
USER.CICS.OPPRTY
USER.CICS.TIMEOUT
USER.CICS.XRFSOFF
USER.LANGUAGE.USERNL1
USER.LANGUAGE.USERNL2
APPCLU.SESSION.SESSKEY
APPCLU.SESSION.KEYINTVL
APPCLU.SESSION.SLSFLAGS
```

Alternatively, you can set up a generic profile USER.CICS.\*\*, to control access to all fields in the CICS segment. Before defining generic profiles use the **SETROPTS GENERIC** command, as described in [“Security administration tasks and the RACF commands to run them”](#) on page 102.

You need READ access to list these profiles, and UPDATE access to change them. For further guidance, see the section on field level access checking in the [z/OS Security Server RACF Security Administrator's Guide](#).

## Security administration tasks and the RACF commands to run them

Reference the list of common security tasks that are carried out by a security administrator and the RACF commands that are required to implement them.

The RACF commands can be issued on TSO or by using this JCL:

```
//jobname JOB
//RACF EXEC PGM=IKJEFT01,REGION=6M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACF commands
/*
//
```

These tables are lists of common tasks that a security administrator would need.

Table 17. Types of RACF administration	
Table	Example tasks
<a href="#">Defining authorities required by security administrators</a>	Delegation of RACF administrative responsibility, controlling access to fields in RACF profiles
<a href="#">Managing users</a>	Specifying default values in the CICS segment, creating or updating segment data for a CICS user, defining terminal users and user groups to RACF
<a href="#">Managing groups</a>	Creating groups, creating subgroups, adding a user to a group, removing a user from a group
<a href="#">Managing profiles and classes</a>	Listing profiles in a class, activating the CICS classes

### Defining authorities required by security administrators

Task	RACF command	Reference
What level of authority do you have?	LISTUSER	<a href="#">LISTUSER</a>
Delegate authority to administer a class.	ALTUSER userid CLAUTH(class)	<a href="#">ALTUSER</a>
Delegate authority to administer users.	ALTUSER userid CLAUTH(USER)	<a href="#">ALTUSER</a>
Delegate authority to administer a group.	CONNECT userid GROUP(group) SPECIAL	<a href="#">CONNECT</a>
Giving access to control CICS fields in the user profile.	RDEFINE FIELD USER.CICS.** UACC(NONE)  PERMIT USER.CICS.** CLASS(FIELD)GROUP(group) ACCESS(UPDATE)	<a href="#">RDEFINE</a> <a href="#">PERMIT</a> <a href="#">Field-level access checking</a>
Giving access to control LANGUAGE fields in the user profile.	RDEFINE FIELD USER.CICS.** UACC(NONE)  PERMIT USER.LANGUAGE.** CLASS(FIELD)GROUP(group) ACCESS(UPDATE)	<a href="#">RDEFINE</a> <a href="#">PERMIT</a> <a href="#">Field-level access checking</a>
Installing a resource with user ID attributes.	DEFINE SURROGAT *.DFHINSTAL UACC(NONE)  PERMIT *.DFHINSTAL CLASS(SURROGAT) GROUP(group) ACCESS(UPDATE)	<a href="#">RDEFINE</a> <a href="#">PERMIT</a> <a href="#">Surrogate Security</a>

### Managing users

Task	RACF command	Reference
Adding a CICS User with the default CICS options.	ADDUSER userid DFLTGRP(group) CICS	<a href="#">ADDUSER</a> <a href="#">The CICS segment in user profiles</a>

Task	RACF command	Reference
Changing CICS options for a user.	ALTUSER userid CICS(field1 field2 .. fieldN)	<a href="#">ALTUSER</a> The CICS segment in user profiles
Removing CICS options for a user.	ALTUSER userid NOCICS	<a href="#">ALTUSER</a>
Listing the CICS options for a user.	LISTUSER userid CICS	<a href="#">LISTUSER</a>
Specifying the language in the LANGUAGE segment.	ALTUSER userid LANGUAGE (PRIMARY(language_code) SECONDARY(language_code))	<a href="#">ALTUSER</a> The LANGUAGE segment in user profiles
Listing the LANGUAGE segment for a user.	LISTUSER userid LANGUAGE	<a href="#">LISTUSER</a>

## Managing groups

Task	RACF command	Reference
Creating groups .	ADDGROUP group OWNER(owner)	<a href="#">ADDGROUP</a>
Creating subgroups.	ADDGROUP subgroup OWNER(subowner) SUPGROUP(group)	<a href="#">ADDGROUP</a>
Adding a user to a group.	CONNECT userid GROUP(group)	<a href="#">CONNECT</a>
Removing a user from a group.	REMOVE userid GROUP(group)	<a href="#">REMOVE</a>

## Managing profiles and classes

Task	RACF command	Reference
Listing the profiles in a class.	RLIST class * ALL	<a href="#">RLIST</a>
Defining a resource by using a generic profile.	REDEFINE class profile UACC(NONE)	<a href="#">RDEFINE</a> <a href="#">RACF profiles for CICS classes</a>
Defining a resource by using a member profile.	REDEFINE class profile ADDMEM(memberList) UACC(NONE)	<a href="#">RDEFINE</a> <a href="#">RACF profiles for CICS classes</a>
Giving access to a group of users to a profile.	PERMIT profile CLASS(class) ID(group) ACCESS(access)	<a href="#">PERMIT</a>
Defining resource definitions when you use SECPRFX.	REDEFINE class secprfx.profile UACC(NONE)	<a href="#">RDEFINE</a> <a href="#">Prefixing RACF Profiles with SECPRFX</a>
Refreshing resource profiles in main storage.	SETROPTS RACLIST(class) REFRESH	<a href="#">SETROPTS</a> <a href="#">Caching of RACF classes and their profiles</a>
Activating the CICS classes.	SETROPTS CLASSACT(class)	<a href="#">SETROPTS</a> <a href="#">Caching of RACF classes and their profiles</a>
Making an installation defined class support generic resources.	SETROPTS GENERIC(class)	<a href="#">SETROPTS</a>

When you have a pair of classes, class is the general resource class, for example, TCICSTRN.

## Security classification of data and users

RACF gives you the means to classify some or all of the resources on your system. You can use security levels, security categories, or both, to protect any CICS-related resource.

Consider classifying resources if you want to control access to them without having to specify access lists in each resource profile. If you classify a resource, only users whose user profiles are appropriately classified will be able to access that resource. For information on using security levels and security categories, see the [z/OS Security Server RACF Security Administrator's Guide](#). Because CICS uses the RACROUTE REQUEST=FASTAUTH function, some services such as security labels and global access checking are not available under CICS. See the [z/OS Security Server RACF Security Administrator's Guide](#) for information on what is available with FASTAUTH.

You can also put users with the same access or logging requirements into groups. A user can belong to one or more groups, one of which is their default. The sign-on process allows the user to override the default RACF user group name. If “list of groups checking” is inactive, signing on with different group names might give a user different authorities. For more information, see [Activating list-of-groups checking \(GRPLIST option\)](#).

## Invoking an external security manager

---

CICS provides an interface to an external security manager (ESM), which can be RACF, a vendor product, or user-written. CICS security uses a base component of z/OS, the System Authorization Facility (SAF) to route authorization requests to the external security manager (ESM).

For information about RACF, see [How it works: Securing CICS with RACF](#). The information in this section is intended for users of other ESMs.

SAF builds an interface between CICS and the external security manager by using a z/OS system service called the *MVS router*. The System Authorization Facility and the SAF router are present on all z/OS systems, even if no ESM is installed. Resource managers, such as CICS, call the MVS router as part of certain decision-making functions in their processing, for example access control checking and authorization-related checking. These functions are called *control points*. Using the MVS router, SAF routes the questions to the ESM and routes the answer back to the resource manager. The final decision on whether access is granted, is made by the resource manager, not by SAF nor by the ESM.

For more information about the MVS router, see [ICHRTX00 - MVS Router Exit in z/OS MVS Installation Exits](#) and [z/OS Security Server RACF Messages and Codes](#).

Typically, if RACF is present, the MVS router passes control to the RACF router, which in turn invokes the appropriate RACF function. However, you can modify the action of the MVS router by invoking the router exit. The router exit can be used, for example, to pass control to a user-written or vendor-supplied ESM. If you want to use your own security manager, you must supply an MVS router exit routine.

When CICS invokes the ESM, it passes information about the current CICS environment, for use by an ESM exit program, in an *installation data parameter list*. How your exit programs access the installation data parameter list depends on whether or not your ESM is RACF. If you use an ESM exit program that issues a **RACROUTE REQUEST=AUTH** and the exit is running because of a security request from CICS, you must ensure that any CLASS which is referenced is listed in the RACLIST.

For more information, see [Customizing security processing](#).

---

## Chapter 10. Implementing a zero trust strategy

Zero trust is a strategy rather than something that you implement by a specific configuration option. It is essentially an increasing series of best practices that you need to review, implement, and maintain as part of your IT strategy.

There are a number of best practices that are described in [How it works: Zero trust in CICS](#). Implementing zero trust might seem a daunting task. It is not something that can be implemented quickly or all at once. Therefore, it is essential to develop a plan to prioritize this work. This will probably be done in conjunction with your CISO looking at zero trust across the enterprise.

You're encouraged to adopt a zero trust strategy regardless of your current CICS release. However, some functions such as security discovery and security definition capture (SDC) that are provided in more recent releases of CICS can ease this process.

Follow steps in this roadmap to start the journey of implementing a zero trust strategy in CICS.

- [“Identifying the data that needs protection” on page 105](#)
- [“Identifying who needs to access the data” on page 105](#)
- [“Configuring controls to protect the data” on page 105](#)
- [“Monitoring who is accessing the data” on page 106](#)
- [“Auditing CICS to make sure best practices are used” on page 106](#)
- [“Maintaining security when developing new applications or changing applications” on page 106](#)

### Identifying the data that needs protection

Some data is more sensitive or valuable to the company than other data. You might already have reviewed your data for GDPR or similar regulation, or for PCI-DSS compliance by following best practices in [How it works: Zero trust in CICS](#).

If you have separate sets of regions that access this data, it is recommended to concentrate on these regions first. If there are particular resource types that store this data, such as files, concentrate on these resources first.

### Identifying who needs to access the data

As described in [How it works: Zero trust in CICS](#), you need to ensure access to applications and their data is associated with roles.

You might have already implemented transaction security for users who need access to an application that then needs access to run certain transactions. If this is done on an individual user ID basis, you need to migrate your security definitions to be role based.

To do that, you can use [CICS security discovery](#) to identify and refine the groups of user IDs (roles) associated with groups of transactions (applications).

6.2

After identifying the roles and applications, you can further refine roles with usage data. The CICS security discovery data (SDD) helps you identify the resource groups that are accessed by each role.

### Configuring controls to protect the data

In order to implement resource security, you need to define a RACF class for the resource with profiles that define which roles have access to which groups of data. You then need to specify the RACF class on the corresponding CICS SIT parameter (**Xnnn** parameter) for the regions that use this data.

**Recommended:** To conform with a zero trust strategy, it's recommended that you define resource group classes (member lists) in RACF. Generic profiles are not recommended as part of a zero trust strategy because they require other controls to ensure that definitions are added during development.

CICS security discovery allows you to convert required resource security definitions back into RACF commands.

After you have applied required security definitions, you need to make sure command security (CMDSEC) and resource security (RESSEC) are enabled in such regions:

- **6.2** As of CICS TS 6.2, all CICS transactions, excluding CJXA and CICSplex® SM transactions (CO\*\*), are defined with RESSEC (YES) and CMDSEC (YES) by default, which requires extra security configuration for some CICS transactions. See [CICS transactions subject to security checking](#).
- Your transactions that are running in these CICS regions also need to be defined with RESSEC (YES) and CMDSEC (YES).
- When all of your transactions definitions implement RESSEC (YES) and CMDSEC (YES), you can set the SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS** to override transaction settings to prevent future transaction definitions from breaking your zero trust policy.

## Monitoring who is accessing the data

By implementing the previously mentioned command and resource security, you will be able to monitor key resources using the RACF SMF 80 records or by using [zSecure Access Monitor](#).

Zero trust is about monitoring the end user's access. Therefore, the user ID that is used for the authentication request must either be the end user's user ID, or have the [distributed identity](#) associated with it if it is a functional user ID.

## Auditing CICS to make sure best practices are used

[SMF 1154-80 records](#) includes auditing of the best practices' compliance. See [Compliance data collection](#).

## Maintaining security when developing new applications or changing applications

After your production CICS regions are migrated to use resource and command security, you need to ensure that the security definitions of new or changed applications are identified and added whenever a new set of code is delivered to production.

- **6.2** [CICS security definition capture \(SDC\)](#) provides a manual way of identifying required security definitions for changed applications during development, before promoting the changes to production. You can also integrate it into a CI/CD pipeline by leveraging other automation solutions.

### Related tasks

[Migrating to zero trust with CICS security discovery](#)

### Related information

[How it works: Zero trust in CICS](#)

## What is security metadata?

Security metadata represents the security definitions defined in an external security manager (ESM), in this case RACF, in a more readable format.

Security metadata can be used for analysis of security definitions by the [security discovery](#) function in CICS Explorer. You generate security metadata from your RACF definitions, analyze and update it in CICS Explorer, and then convert the output security metadata back to RACF commands to update your security definitions.

It is also the output from [CICS security definition capture \(SDC\)](#). In that case, it contains the required security definitions captured by SDC.

**Note:** Diagrams about the look of security metadata in CICS Explorer in this topic are for illustrative purposes only, where colors and annotations are added for explanation. For an actual example and screen captures of managing security metadata in CICS Explorer, see [“Analyzing security definitions using security discovery in CICS Explorer”](#) on page 122.

## The file format

Security metadata provides an easy to read version of the essential elements of the security definitions. CICS Explorer requires the security metadata file to have a file extension of `esm`.

The file is in ASCII and can be read by any text editor. However, because it is in YAML format, you can also read it in a YAML editor for improved syntax highlighting. You might need to associate the `.esm` files with the YAML language in the editor settings for correct syntax highlighting.

In a YAML file, a key-value pair is identified by a colon (:), in the format of `key: value`. A hyphen (-) indicates a list item and indentation (spaces) indicates nesting or hierarchy. Comment lines start with a hash sign (#).

## Syntax of security metadata

The following code snippet shows a basic example of security metadata:

```
--- # Security Metadata---
version: 2
file_created:
  - date: "17 Mar 2023"
  - time: "15:27:26"
  - user: SUE
  - esm: "RACF"
application:
  - name: "Current Account"
  - owner: "Claude Gordon"
  - description: "Front office account application"
    origin_transactions:
      - MENU
group_list:
  - name: MANAGER
    users:
      - MAINWRN
user_list:
  - user: MAINWRN
    username: "George Mainwaring"
secpfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: MENU
        access_lists:
          - access: READ
            groups:
              - MANAGER
```

*Figure 41. A basic example of security metadata*

As shown in [Figure 41 on page 107](#), a security metadata file contains several main sections:

### version

Is the version number of the security metadata syntax.

### file\_created

Contains information about when the file was created by which user ID, and for which ESM.

### application

Applicable only if an application filter has been defined in CICS Explorer for the security metadata.

This part contains basic information about the application, for example, the name, owner, description, and origin transactions of the application.

**Note:** If a security metadata file contains an application section, that means this metadata contains only security definitions that are associated with the defined application.

For more information about the application filter, see [“Defining applications to segment work”](#) on page 136.

### group\_list

Includes a list of RACF user groups (roles), which can nest user profiles (in the format of user ID). In this case, MANAGER is the RACF group, which contains the user ID MAINWRN.

A group can be empty; that is, it contains only the name element.

### user\_list

Optional. Maps the user ID to the username defined in RACF.

In this case, the user ID MAINWRN is mapped to the username George Mainwaring.

### secprfx

Optional. Is the value of the SECPRFX that is used to extract the RACF data. This value corresponds to the **SECPRFX** SIT parameter for the region or regions using this data.

### classes

The list of RACF classes. Each class has a class type that matches the [Xnnn SIT parameter](#) of the resource type (such as XTRAN for transactions), and the name of the class that the security metadata was extracted from.

Each class has a series of profiles. The profile can be defined for either a single resource or sets of resources in member lists. Each profile then has an access list section that defines who has access at each level.

In the following snippet, a transaction class has a profile for the MENU transaction, to which the MANAGER group has READ access. The name of the transaction classes is derived from CICSTRN (see [XTRAN system initialization parameter](#)). In this case, it's TCICSTRN because it's a general resource class.

```
classes:
- class: XTRAN                # The Xnnn SIT parameter for the type of resource
  name: CICSTRN               # Optional. The class name as defined on the SIT parameter
  profiles:                   # Contains either discrete profiles or member lists
    - name: MENU              # Name of the resource or member list
      access_lists:           # Contains access lists for the named profile
        - access: READ        # RACF access level
          groups:              # Optional. Contains groups with the defined access level to
the named profile
            - MANAGER
```

If the profile is a member list instead of a resource, it looks something like this:

```
profiles:
- name: BANKING
  members:                    # Optional. If the profile is a member list, it contains
resources in the member list
  - BNK1
  - BNK2
```

where the BANKING member list contains resources (in this case, transactions) BNK1 and BNK2.

In addition to groups, individual users can also appear in the access list. However, you're recommended to assign every user a role when analyzing the security definitions in CICS Explorer.

```
access_lists:
- access: READ
  users:                      # Optional. Contains users with the defined access level to
the named profile
  - JONES                     # User ID
```



## Example security metadata: Transactions only

In the following code snippet:

- Two groups are defined: group MANAGER that contains the user MAINWRN, and group TELLER that contains PIKE and WILSON. An ungrouped user JONES is also defined.
- RACF classes are defined for transactions, whose names are derived from CICSTRN. The classes contain profiles for member lists BANKING and NOSEC, and a profile for transaction BNK3. BANKING further contains transactions BNK1 and BNK2. NOSEC further contains transactions CHLP, CMAC, CRTX, and CSGM.
- Groups MANAGER and TELLER are defined with READ access to the BANKING member list. The MANAGER group also has READ access to the BNK3 transaction. The NOSEC member list has the universal access (UACC) set to READ.

```
--- # Security Metadata ---
version: 2
file_created:
  - date: "17 Mar 2023"
  - time: "16:27:26"
  - user: SUE
  - esm: "RACF"
group_list:
  - name: MANAGER
    users:
      - MAINWRN
  - name: TELLER
    users:
      - WILSON
      - PIKE
user_list:
  - user: JONES
    username: "Jack Jones"
  - user: MAINWRN
    username: "George Mainwaring"
  - user: PIKE
    username: "Frank Pike"
  - user: WILSON
    username: "Arthur Wilson"
secprfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: BANKING
        members:
          - BNK1
          - BNK2
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
            users:
              - JONES
      - name: BNK3
        access_lists:
          - access: READ
            groups:
              - MANAGER
```

Figure 42. Example security metadata with only transaction definitions

The security definitions extracted from RACF can include a `universal_access` tag if the UACC value isn't NONE. A zero trust strategy recommends that access is only granted for roles that need to access a resource. This means that normally in your profile definitions universal access (UACC) should be set to NONE. The **Security Discovery** editor in CICS Explorer ignores the `universal_access` tag. Therefore, any definitions requiring a UACC value other than NONE need to be handled outside of the tool.

When opened in the **Security Discovery** editor in CICS Explorer, it looks something like this:

Member list with included transactions

		BANKING		Ungrouped
		BNK1	BNK2	BNK3
Group	MANAGER	R	R	
Users within a given group, with mapped usernames	MAINWRN	George Mainwaring	R	R
	TELLER		R	
	PIKE	Frank Pike	R	
	WILSON	Arthur Wilson	R	
Ungrouped users	Unresolved			
	JONES	Jack Jones	R	

Ungrouped transactions

Figure 43. Breakdown of example security metadata with transaction definitions in CICS Explorer

## Example security metadata: Transactions and resources

Resources of other types can be included in the security metadata too. For example, the following code snippet defines two profiles. One belongs to the GCICSTRN class and is defined for the two transactions BNK1 and BNK2 that are in the BANKING member list. The other belongs to the HCICSFCT class and is defined for the two files CURRENT and SAVING that are also in the BANKING member list.

```

--- # Security Metadata ---
version: 2
file_created:
  - date: "17 Mar 2023"
  - time: "17:27:26"
  - user: SUE
  - esm: "RACF"
group_list:
  - name: MANAGER
    users:
      - MAINWRN
  - name: TELLER
    users:
      - WILSON
      - PIKE
user_list:
  - user: JONES
    username: "Jack Jones"
  - user: MAINWRN
    username: "George Mainwaring"
  - user: PIKE
    username: "Frank Pike"
  - user: WILSON
    username: "Arthur Wilson"
secprfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: BANKING
        members:
          - BNK1
          - BNK2
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
            users:
              - JONES
  - class: XFCT
    name: CICSFCT
    profiles:
      - name: BANKING
        members:
          - CURRENT
          - SAVING
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
            users:
              - JONES
          - access: UPDATE
            groups:
              - MANAGER

```

*Figure 44. Example security metadata with transaction and file definitions*

When opened in the **Security Discovery** editor in CICS Explorer, it looks something like this:

Security definitions for FILES

		BANKING			
		CURRENT	SAVING	BNK1	BNK2
MANAGER		U	U	R	R
MAINWRN	George Mainwaring	U	U	R	R
TELLER		R	R	R	R
PIKE	Frank Pike	R	R	R	R
WILSON	Arthur Wilson	R	R	R	R
Unresolved					
JONES	Jack Jones	R	R	R	R

Figure 45. Breakdown of example security metadata with transaction and file definitions in CICS Explorer

### Example security metadata: Application filter

When the security metadata is too large, for example, it contains security definitions for multiple applications across several CICS regions, you might want to segment the work. You can define an application filter for your security definitions in CICS Explorer so that the resulting security metadata contains only the security definitions that are associated with the defined application filter.

The following example shows example security definitions with an application filter. The application filter includes the following definitions:

- Origin transactions that the application uses: BNK1
- Other transactions invoked by the origins on the execution path, either directly or indirectly: BNK2
- Resources used by those transactions: CURRENT, SAVING
- Users authorized to use the application, with different levels of access: MAINWRN, JONES, PIKE, WILSON

The users are grouped into roles (MANAGER and TELLER), and resources and transactions into member lists (BANKING).

```

--- # Security Metadata ---
version: 2
file_created:
  - date: "20 Mar 2023"
  - time: "18:24:26"
  - user: SUE
  - esm: "RACF"
application:
  - name: "Current Account"
  - owner: "Claude Gordon"
  - description: "Front office account application"
  origin_transactions:
    - BNK1
group_list:
  - name: MANAGER
    users:
      - MAINWRN
  - name: TELLER
    users:
      - WILSON
      - PIKE
      - JONES
user_list:
  - user: JONES
    username: "Jack Jones"
  - user: MAINWRN
    username: "George Mainwaring"
  - user: PIKE
    username: "Frank Pike"
  - user: WILSON
    username: "Arthur Wilson"
secprfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: BANKING
        members:
          - BNK1
          - BNK2
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
  - class: XFCT
    name: CICSFCT
    profiles:
      - name: BANKING
        members:
          - CURRENT
          - SAVING
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
          - access: UPDATE
            groups:
              - MANAGER

```

Figure 46. Example security metadata with an application filter

## Related concepts

[How it works: Securing CICS with RACF](#)

## Related tasks

[Creating input security metadata for security discovery analysis](#)

# Migrating to zero trust with CICS security discovery

This section contains a series of tasks and an example that demonstrate each stage in the migration to zero trust, by using CICS security discovery.

The overall migration process, the sequence of tasks within, roles responsible for them, and where security discovery is involved are shown in the following diagram:



The security discovery process highlighted by the light blue box is explained in detail in [How it works: CICS security discovery](#).

## Identify a subset of production regions for migration

Unless your production CICS environment contains only a few CICS regions, it's recommended that you divide the work into a set of manageable chunks that can be addressed separately. Sets of regions can be identified based on their similar security definitions and application usage. These regions would have the following characteristics:

- The same RACF database.
- The same transaction class (the XTRAN parameter in the SIT) and other CICS classes if used.
- The same SECPRFX value in the SIT.
- A complete or representative set of regions to cover one or more applications. This might include one or more of:
  - a TOR or WOR
  - an AOR
  - an FOR

You might want to select a small set of regions for a proof of concept, and then repeat the procedure for the other selected sets based on risk.

For detailed instructions about other main tasks in the migration journey, see the following topics:

## Creating input security metadata for security discovery analysis

A security administrator runs a JCL job to extract the RACF definitions and convert them into input security metadata, which represents the security definitions for roles and transactions in an ESM agnostic format.

For an overview of the whole security discovery process, see [How it works: CICS security discovery](#). This task is Step 1 in that flow.

### Before you begin

To create the security metadata for each set of CICS regions that share security definitions, make sure the regions have the same RACF class(es) and **SECPRFX** SIT parameter.

### Procedure

1. The security administrator needs to customize the DFH\$R2SM JCL, which is a sample in hlq.SDFHSAMP.

Customize the following parameters as needed:

#### **SECMETA EXEC DFHXSMET,SAMPLIB=hlq.SDFHSAMP**

The JCL uses REXX programs in the CICS sample library. On this line, change *hlq* to the installed library's high-level qualifier.

#### **DIR**

Specifies the directory to one on zFS into which the output file will be created. This directory must already exist.

#### **FN**

Specifies the file name of the output file. The file will be created in ASCII with a file extension of .esm.

#### **CYL**

Specifies the number of cylinders that allows for the size of the RACF data to be extracted. The default allows for two million lines of output, which is sufficient for most installations.

#### **Xnnn**

The name of the CICS classes as defined on the SIT security class parameters (**Xnnn**), for example, XTRAN=CICSTRN, XFCT=CICSFCT.

The transaction classes are always required. If XTRAN=NO is specified in the JCL, an error is returned:

ERROR: XTRAN is not supported and terminates with Return code 12

For other classes, Xnnn=NO is assumed unless the option is otherwise specified.

#### **SECPRFX**

The security prefix as defined on the **SECPRFX** SIT parameter.

#### **HIDE**

Obfuscates the output if you need to send it to IBM for diagnostic purposes. Set it to NO for customer use.

#### **DEBUG**

Use it for diagnostic purposes if you need IBM support. Set it to NO for customer use.

2. The job needs to be submitted by the security administrator who has at least ROAUDIT authority so that the RACF commands extract all of the transaction and group data.

The security metadata looks like the following example:

```

--- # Security Metadata---
version: 2
file_created:
  - date: "17 Mar 2023"
  - time: "15:27:26"
  - user: SUE
  - esm: "RACF"
group_list:
  - name: MANAGER
    users:
      - MAINWRN
user_list:
  - user: MAINWRN
    username: "George Mainwaring"
secprfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: MENU
        access_lists:
          - access: READ
            groups:
              - MANAGER

```

Figure 47. An example of generated security metadata

For more information about security metadata, see [What is security metadata?](#)

3. Job steps return a completion code:

- 0 - successful completion
- 4 - warning messages issued (the job will continue)
- 8, 12 - error messages issued (the job will terminate)

4. Each job step might result in messages being issued to SYSTSPRT. Warning messages are issued for unsupported definitions that are found in the RACF data.

## Result

The security metadata is generated in UTF8 on zFS.

## What to do next

The security administrator passes the security metadata to the system programmer for analysis in CICS Explorer. For instructions, see [“Analyzing security definitions using security discovery in CICS Explorer” on page 122](#).

- 6.2 Optionally, the system programmer can [capture security discovery data \(SDD\)](#) as additional input data for analysis in CICS Explorer.

## Capturing security discovery data (SDD)

- 6.2 Applies to 6.2.

Security discovery records security access requests within CICS, contained in what's called the security discovery data (SDD). Records in the SDD can be access requests that were performed as a result of the existing security settings. They can also be access requests that would have been performed if the security settings were changed to allow it. With SDD, you can identify what security definitions would be required if the current CICS resource security were extended to include another resource classes.

For a list of data that is captured by security discovery, see [Data captured by CICS security discovery](#).

For an overview of the whole security discovery process, see [How it works: CICS security discovery](#). This task is Step 2 in that flow.



## Before you begin

You must have configured your CICS regions for security discovery. For instructions, see [Setting up CICS security discovery](#).

### Important:

The security discovery state persists over a warm or an emergency restart. On a cold or an initial start of CICS, security discovery is inactive.

In case of a cold or an initial start, you need to manually reissue the **SET SECDISCOVERY** commands or add them to a program that runs as part of PLT to turn it on. See [“Activating CICS security discovery during CICS startup”](#) on page 119.

## About this task

CICS always records transaction attach requests when security discovery is active. Such requests are protected by the **XTRAN** system initialization parameter. You need to decide what additional access requests are recorded based on business requirements to protect the associated resources.

Security discovery data (SDD) is never reset while CICS is running. When the data is written to the DFHSECD log stream, it is the complete set of data since security discovery was first activated in the region.

**Recommendation:** You are advised to use a single shared log stream for regions with identical security settings. The chance of contention is low because the data is usually only written to the log stream at end of day. A randomization process means that multiple regions do not write their data at the same time.

## Procedure

1. Decide what security requests you want to discover in addition to transaction attach requests.
2. Decide the set of regions where security discovery needs to be activated.

A good starting point is to activate security discovery across a set of regions that run a similar workload, such as an application, and have identical CICS security settings.

3. Activate security discovery in either of the following ways:
  - In CICS Explorer, navigate to the **Security Discovery Records** view. Right-click the row of the region in which you want to activate security discovery and select **Set Discovery Status > Switch On Security Discovery**. For more options in CICS Explorer, see [Security Discovery Records view in CICS Explorer documentation](#).
  - Use a PLT program to activate security discovery during CICS startup. For instructions, see [Activating CICS security discovery during CICS startup](#).
  - Write your own transaction by using the **SET SECDISCOVERY** SPI command to switch recording on. For more information about options on the command, see [SET SECDISCOVERY](#).
4. CICS captures the SDD in memory and writes the data out to the DFHSECD log stream at end of day or when the region shuts down.
  - If needed, you can cause CICS to write the SDD to the log stream immediately with either of the following methods. Security discovery statistics are written to the DFHSECD log stream at the same time and the values are reset.

### CICS Explorer

In the **Security Discovery Records** view, right-click the row of the region against which you want to perform the action. Then click **Write Security Discovery Records**. For more information, see [Security Discovery Records view in CICS Explorer documentation](#).

### SPI commands

Issue the [PERFORM SECDISCOVERY WRITE](#) command from your transaction.

- To check whether any new data has been recorded since the last time CICS wrote data to the log stream, use the DFHXS1602 message and either of the following methods to gain status

information about security discovery. You can know when most or all of the SDD in the region is captured and the volume of the data that is to be processed by CICS Explorer.

### CICS Explorer

The **Security Discovery Records** view shows the security discovery information per CICS region.

### SPI commands

Issue the `INQUIRE SECDISCOVERY` command from your transaction.

5. To process the log stream, copy the DFH\$SDDP JCL from the SDFHSAMP library and customizing the following variables:

#### HLQ

Is the high-level qualifier of your CICS libraries.

#### SDD

Is the file name to which the SDD file is written. The directory containing this file must already exist. The file extension must be `.sdd`.

#### LOGSTRM

Is the name of the log stream that contains the SDD.

To affect how the log stream data is processed, specify parameters for the DFHSDD program in the SYSIN data set. If none of the parameters are specified, every record in the log stream will be processed. Lines beginning with an asterisk are ignored.

#### APPLID=*applid*

Processes records with a matching APPLID. A trailing wildcard (\*) can be specified.

#### STARTTIME=YYYYMMDDHHMMSS

Specifies a local date and time. All records before this time are ignored.

#### ENDTIME=YYYYMMDDHHMMSS

Specifies a local date and time. All records after this time are ignored.

#### TRACE

If you encounter problems, specify the TRACE option to capture a trace to the DFHTRACE destination.

This trace data can be used by IBM support to diagnose the problem.

If you need to process data from multiple log streams, update the DFHSECD DDNAME within the SDD step to add a DD statement for each additional log stream that is to be processed:

```
//DFHSECD DD DSNAME=&LOGSTRM.,DCB=BLKSIZE=32760,  
// SUBSYS=(LOGR,DFHLGCV)  
// DD DSNAME=HLQ.SECOND.LOGSTRM,DCB=BLKSIZE=32760,  
// SUBSYS=(LOGR,DFHLGCV)
```

6. Look at the SUMMARY output from the job. The output shows how many records were processed and how many were ignored as a result of the selection parameters or failing to match the security settings of the first record that was processed.

For an example of the SUMMARY output, see [Example](#).

## Results

You have successfully captured the SDD. You can then use the `.sdd` file as input for the security discovery analysis in CICS Explorer.

CICS security discovery statistics were also written to DFHSECD. The statistics were formatted by DFH\$SDDP and the output was written to the STATS destination. The statistics can be used to estimate the cost of activating additional resource security within CICS.

### Example

The following snippet shows an example of SUMMARY output:

```

Writing trace output to DFHTRACE.
Selecting APPLID=CICSAPP*
Selecting STARTTIME=20230310112500
Selecting ENDTIME=20230320115500

*****
* Record counts *
*****

SDD records read.....: 12
SDD records selected.....: 10
SDD records rejected.....: 0
Userids processed.....: 5
Transactions processed.....: 2

*****
* Security Settings *
*****

SECPRFX = PROD
XCMD    = CICSAMD
XDB2    = CICSDB2
XDCT    = CICSDCCT
XFCT    = CICSFCT
XHFS    = YES
XJCT    = CICSJCT
XPCT    = CICSPCCT
XPPT    = CICSPPCT
XPSB    = CICSPSB
XRES    = CICSRES
XTRAN   = CICSSTRN
XTST    = CICSSTST
XUSER   = SURROGAT

```

The first section confirms the parameters that are in effect.

Within the Record counts section:

#### **SDD records read**

Shows the total number of records read from the DFHSECD log stream.

#### **SDD records selected**

Shows the number of records that met the selection criteria based on the input parameters.

#### **SDD records rejected**

Shows the number of records rejected due to having different security settings.

#### **Userids processed**

Shows the number of unique user IDs found within the processed records.

#### **Transactions processed**

Shows the number of unique transaction IDs found within the processed records.

The Security Settings section lists the SECPRFX and the RACF class names for each system initialization parameter. These values come from the first record processed. All subsequent records must match these settings.

## **Activating CICS security discovery during CICS startup**

**6.2** Applies to 6.2.

When a CICS region starts up with a cold or an initial start, CICS security discovery is in an inactive state. Instead of manually activating it every time, you can use a PLT program to activate it automatically during CICS startup. CICS provides sample program DFH\$SECD, which activates security discovery and sets all resource access requests to be discovered by default.

If you want to activate CICS security discovery by manually issuing the **SET SECDISCOVERY** SPI command, see [SET SECDISCOVERY](#).

**Note:** The CICS security discovery state, whether it is active or not, and the set of resource access requests being discovered, are restored on a warm or an emergency start. You need to set the startup PLT program to control the security discovery behavior only on a cold or an initial start.

## Procedure

1. Review the sample program DFH\$SECD to check whether it meets your requirements for security discovery. DFH\$SECD is located in the hlq.SDFHSAMP data set.

You can use the sample program unchanged, modify the sample program to alter the [SET SECDISCOVERY](#) command being issued, or use the sample as the base for your own program.

2. Add the name of your PLT program to your PLTPI table.

The program needs to run in the third phase of initialization so must appear after the DFHDELIM entry.

```
DFHPLT TYPE=INITIAL,SUFFIX=XS
DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
DFHPLT TYPE=ENTRY,PROGRAM=DFH$SECD Activate security discovery
DFHPLT TYPE=FINAL
```

3. Authorize the PLTPIUSR user ID, if specified, or the region user ID with UPDATE access to the **SET SECDISCOVERY** SPI command. The resource type is SECURITY.

4. Add a program definition for your PLT program. A sample definition for DFH\$SECD is provided in the DFHSECD group.

If you are using the sample program, copy the sample definition to your own group for customization. The definition needs to be in a group that is installed as part of the GRPLIST processing.

5. Set the [PLTPI system initialization parameter](#) to the suffix of the PLT table that contains your new PLT program.

## Results

You have successfully configured CICS to activate CICS security discovery during startup.

## Estimating performance cost of additional resource security using security discovery statistics

6.2

Applies to 6.2.

You can use CICS security discovery statistics to estimate the performance cost of activating additional resource security checks within CICS.

### What are security discovery statistics

CICS security discovery statistics show the number of security calls that have, or would have, been made while resource security is active. This allows you to work out an approximate cost of activating additional resource security for the workload that was running.

Security discovery statistics are written out to the DFHSECD log stream whenever the [security discovery data \(SDD\)](#) is written out. The statistics fields are also reset at that time. The statistics are formatted using the DFH\$SDDP job. The values in the statistics output are the accumulated total of all the DFHSECD statistics records that are selected and processed by DFH\$SDDP.

The statistics are captured for **all** resource types as long as security discovery is active, regardless of the resource type that is being recorded. Therefore, before recording access requests for other resources, you can initially run security discovery with only **XTRAN** access requests recorded, which is the default. The statistics then show you how many resource security checks you are dealing with before you start discovering and analyzing other resource access requests.

**Note:** Security discovery statistics are captured only while security discovery is active.

### Sample security discovery statistics

The following snippet shows an example of security discovery statistics.

```

*****
* Statistics *
*****

Security Discovery records      :          20
Transactions executed          :          24  1
Successful resource checks     :         179  2
  Checks that bypassed DFHXSRC :           7  3
  Checks that called DFHXSRC   :         172
  Exempt resources             :          32
  Resource class not active     :           2  4
  FASTAUTH calls               :         138
  AUTH calls                   :           0
Resource class counts  5
  XCMD checks                  :          24
  XDB2 checks (class not active) :           0
  XDCT checks                  :           6
  XFCT checks                  :           0
  XHFS checks                  :           0
  XJCT checks                  :           0
  XPCT checks                  :           0
  XPPT checks                  :         120
  XPSB checks                  :           0
  XRES checks (class not active) :           2
  XTRAN checks                 :          21
  XTST checks (class not active) :           5
  XUSER checks                 :           1

```

For a complete list of the statistics fields and their description, see [Security discovery statistics](#).

## How to use security discovery statistics

As shown in “[Sample security discovery statistics](#)” on page 120, you can use the statistics to make the following calculations:

### Average number of resource security checks per transaction

Equals the number of successful resource security checks that were made (2) divided by the number of transactions that were executed (1).

### Increase in resource security checks if CMDSEC(YES) and RESSEC(YES) were set or TS queues defined using TSMODELS were secured

Equals the number of resource security checks that bypassed DFHXSRC because CMDSEC(NO) or RESSEC(NO) was specified, or because TS queues that use a model or TST entry didn't have security active (3).

### Average increase in resource security checks per transaction as a result of setting CMDSEC(YES) and RESSEC(YES), and securing existing TS queues that have TSMODELS

Equals the number of resource security checks that bypassed DFHXSRC (3) divided by the number of transactions that were executed (1).

### Increase in ESM FASTAUTH calls if all resource classes were activated

Equals the number of resource security checks that were successful because the associated security class was not active (4), for example, an attempt to access a file when XFCT=NO.

### Average increase in ESM FASTAUTH calls per transaction as a result of activating all resource classes

Equals the number of resource security checks that were successful because the associated security class was not active (4) divided by the number of transactions that were executed (1).

### Cost per transaction

(the number of Xnnn checks / the number of XTRAN checks) \* 2 micro seconds \* *CPUTypeScaling*

where *CPUTypeScaling* is the ratio of the CPU that your enterprise uses to the CPU benchmark reported in 6.2: Cost of resource security. By using the data obtained from the CPU MF and the Large Systems Performance Reference (LSPR), you can predict how this might perform in your environment. For more information, see [Relating LSPR values to a CICS workload](#).

The individual resource class counts (5) allow you to work out which classes to secure first. With this information, you can identify those that will have the biggest impact and perhaps those that are hardly used or even not used at all.

## Related information

[Capturing security discovery data \(SDD\)](#)

[Security discovery statistics](#)

## Analyzing security definitions using security discovery in CICS Explorer

The **CICS Security Discovery** perspective in CICS Explorer allows you to view security definitions for CICS, improve grouping of users, transactions, and other resources, and potentially identify anomalies.

It is assumed that you already have some transactions security definitions set up, which allow users to access the transaction they need for their job. There is no assumption about whether your users are in groups, or your transactions are in member lists. The process enables you to create or improve your user groups (roles) and member lists, and by using the security discovery data (SDD), to identify anomalies, including unused permissions.

This process is likely to take several days, and can be done in a number of stages. You can save your work at any stage. Also, it is possible to export your data for review and implementation in stages.

Choose a route depending on your scenario:

- You have a single application in a set of CICS regions. See [“Analyzing security definitions for a single application in multiple regions”](#) on page 122.
- You have multiple applications in a set of CICS regions. See [“Analyzing security definitions for several applications in multiple regions”](#) on page 123.

### Analyzing security definitions for a single application in multiple regions

Table 18. Workflow of analyzing security definitions for a single application				
Stage	Input	Definitions analyzed	Result	Roles responsible
<a href="#">“Analyzing security definitions based on similarity”</a> on page 124	Security metadata containing RACF definitions (.esm file)	<ul style="list-style-type: none"><li>• Roles</li><li>• Transactions (XTRAN)</li></ul>	<ul style="list-style-type: none"><li>• Users grouped into roles based on the similarity pattern</li><li>• Transactions grouped into member lists</li><li>• All security definitions are for a single application</li></ul>	System programmer
<a href="#">“Refining security definitions for transactions with SDD”</a> on page 130	<ul style="list-style-type: none"><li>• Security definitions from the previous stage</li><li>• SDD</li></ul>	<ul style="list-style-type: none"><li>• Roles</li><li>• Transactions (XTRAN)</li></ul>	<ul style="list-style-type: none"><li>• Users removed from or added to roles based on usage</li><li>• Transactions in member lists</li><li>• All security definitions are for a single application</li></ul>	System programmer

Table 18. Workflow of analyzing security definitions for a single application (continued)

Stage	Input	Definitions analyzed	Result	Roles responsible
<a href="#">“Identifying or improving security definitions for resources with SDD” on page 132</a>	<ul style="list-style-type: none"> <li>Security definitions from the previous stage</li> <li>SDD</li> </ul>	<ul style="list-style-type: none"> <li>Roles</li> <li>Other resources, such as files (XFCT)</li> </ul>	<ul style="list-style-type: none"> <li>Roles (refined)</li> <li>Resources in member lists</li> <li>All security definitions are for a single application</li> </ul>	<ul style="list-style-type: none"> <li>System programmer</li> <li>Application owner</li> </ul>

At each stage, you can export the security definitions from CICS Explorer as security metadata (.esm) and convert it back into RACF definitions to leverage whatever analysis has been achieved. See [“Exporting security metadata for review” on page 148](#).

## Analyzing security definitions for several applications in multiple regions

In this case, you need to identify your transactions and resources associated with each application to work on a subset of data at a time. The **Security Discovery** editor in CICS Explorer provides an application filter function for you to define an application as well as its associated transactions and resources, making it easier to manage security definitions per application. This allows you to enable additional command and resource security (CMDSEC, RESSEC) for one application at a time.

The workflow is as follows:

Table 19. Workflow of analyzing security definitions for multiple applications

Stage	Input	Definitions analyzed	Result	Roles responsible
<a href="#">“Analyzing security definitions based on similarity” on page 124</a>	Security metadata containing RACF definitions (.esm file)	<ul style="list-style-type: none"> <li>Roles</li> <li>Transactions (XTRAN)</li> </ul>	<ul style="list-style-type: none"> <li>Users grouped into roles based on the similarity pattern</li> <li>Transactions grouped into member lists</li> <li>No application segmentation</li> </ul>	System programmer
<a href="#">“Refining security definitions for transactions with SDD” on page 130</a>	<ul style="list-style-type: none"> <li>Security definitions from the previous stage</li> <li>SDD</li> </ul>	<ul style="list-style-type: none"> <li>Roles</li> <li>Transactions (XTRAN)</li> </ul>	<ul style="list-style-type: none"> <li>Users removed from or added to roles based on usage</li> <li>Transactions in member lists</li> <li>No application segmentation</li> </ul>	System programmer

Table 19. Workflow of analyzing security definitions for multiple applications (continued)

Stage	Input	Definitions analyzed	Result	Roles responsible
<a href="#">“Defining applications to segment work” on page 136</a>	<ul style="list-style-type: none"> <li>Security definitions from the previous stage</li> <li>SDD</li> </ul>	<ul style="list-style-type: none"> <li>Roles</li> <li>Transactions (XTRAN)</li> </ul>	<ul style="list-style-type: none"> <li>Applications defined by: <ul style="list-style-type: none"> <li>A set of origin transactions</li> <li>A set of associated transaction member lists</li> </ul> </li> <li>Transactions in member lists</li> <li>Roles (refined)</li> </ul>	<ul style="list-style-type: none"> <li>System programmer</li> <li>Application owner</li> </ul>
<a href="#">“Defining security for resources associated with each application” on page 144</a>	<ul style="list-style-type: none"> <li>Security definitions from the previous stage</li> <li>SDD</li> </ul>	<ul style="list-style-type: none"> <li>Roles</li> <li>Other resources, such as files (XFCT)</li> </ul>	<ul style="list-style-type: none"> <li>Applications defined by associated transactions and resources</li> <li>Resources in member lists</li> <li>Transactions in member lists</li> <li>Roles (refined)</li> </ul>	<ul style="list-style-type: none"> <li>System programmer</li> <li>Application owner</li> </ul>

At each stage, you can export the security definitions from CICS Explorer as security metadata (.esm) and convert it back into RACF definitions to leverage whatever analysis has been achieved. See [“Exporting security metadata for review” on page 148](#).

## Analyzing security definitions based on similarity

In CICS Explorer, you can organize security definitions imported from security metadata by using the **CICS Security Discovery** perspective. It groups ungrouped users that have similar access levels to transactions into roles, and ungrouped transactions into member lists if they are accessed by the same roles.

For an overview of the workflow in CICS Explorer, see [“Analyzing security definitions using security discovery in CICS Explorer” on page 122](#).

### Before you begin

- Ensure that you have installed CICS Explorer Fix Pack 5.5.36. For this task, however, it is not necessary to configure CICS Explorer to connect to CICS regions.
- Make sure that you have exported definitions from RACF as input security metadata and stored the security metadata as a .esm file on your workstation ready to be loaded into CICS Explorer.

### About this task

You work with security definitions mainly in the security definitions table in the **Security Discovery** editor.

In the table, uppercase letters in the cells indicate access, for example, R indicates READ access defined in the security definitions. A plus (+) suffix indicates access that is proposed by the analysis as a result of the user being moved to a role.



For more information about the **CICS Security Discovery** perspective as well as the **Security Discovery** editor and the **Security Discovery Details** view within, see [CICS Security Discovery perspective in CICS Explorer documentation](#).

## Procedure

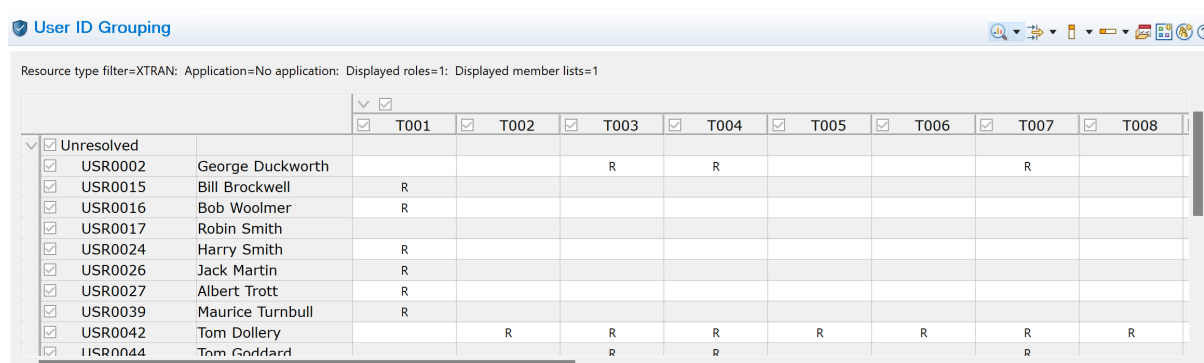
1. Open the **CICS Security Discovery** perspective by clicking **Window > Perspective > Open perspective > Other**. Then select **CICS Security Discovery** in the dialog that opens.

**Recommendation:** The **CICS Security Discovery** perspective comes with a view and an editor that are related with security discovery. For the best user experience, it's recommended that you switch to this perspective before working with the **Security Discovery** editors or views.

2. To import the RACF definitions as input security metadata into CICS Explorer, click **File > Import ESM file**, and then select the .esm file.

This opens the **Security Discovery** editor in CICS Explorer.

The editor shows the security definitions in a table:



User ID Grouping

Resource type filter=XTRAN: Application=No application: Displayed roles=1: Displayed member lists=1

		<input checked="" type="checkbox"/> T001	<input checked="" type="checkbox"/> T002	<input checked="" type="checkbox"/> T003	<input checked="" type="checkbox"/> T004	<input checked="" type="checkbox"/> T005	<input checked="" type="checkbox"/> T006	<input checked="" type="checkbox"/> T007	<input checked="" type="checkbox"/> T008
Unresolved									
<input checked="" type="checkbox"/> USR0002	George Duckworth			R	R			R	
<input checked="" type="checkbox"/> USR0015	Bill Brockwell	R							
<input checked="" type="checkbox"/> USR0016	Bob Woolmer	R							
<input checked="" type="checkbox"/> USR0017	Robin Smith								
<input checked="" type="checkbox"/> USR0024	Harry Smith	R							
<input checked="" type="checkbox"/> USR0026	Jack Martin	R							
<input checked="" type="checkbox"/> USR0027	Albert Trott	R							
<input checked="" type="checkbox"/> USR0039	Maurice Turnbull	R							
<input checked="" type="checkbox"/> USR0042	Tom Dollery		R	R	R	R	R	R	R
<input checked="" type="checkbox"/> USR0044	Tom Goddard			R	R			R	

Figure 48. RACF definitions imported into the **Security Discovery** editor

The table shows the users (rows) and the resources (columns), in this case transactions, they have access to. The users' access is shown by the letter R (for READ) in the corresponding cells.

If the transactions and users are already grouped, the grouping is shown in the table through nesting. In this example, the users are not yet grouped into roles, and instead access to resources is assigned individually to users. Although it is a valid approach, it can make management of authorities time consuming and error- prone. Similarly, it is better to group transactions into member lists that represent the transactions accessible to one or more roles in an application.


3. By default, only transactions are shown. If not, click the **Filter resources** icon (  ) on the editor tool bar and select **XTRAN (transactions)**.



Figure 49. Filtering security definitions to show transactions

4. If **all** your users and transactions are already grouped into appropriate roles and member lists, skip to Step “7” on page 127.

5. On the editor toolbar, click the **Analyze** icon (🔍) and select **Run similarity analysis**.

This creates roles and member lists as required, moving ungrouped users with similar access into roles without removing their existing permissions. It does not move users from existing roles or transactions from existing member lists.

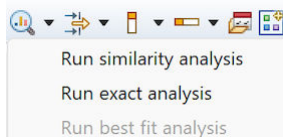


Figure 50. Analyze menu

Alternatively, you can use the other analysis option **Run exact analysis**, which groups users and resources with exact matching access into proposed roles and member lists.

**Note:** The best fit analysis is applicable only to resources other than transactions.

6. Example output of the similarity analysis is as follows:

User ID Grouping

</

Figure 51. Transaction security definitions grouped by similarity

As shown in Figure 51 on page 126, additional permissions that are proposed are marked by a plus (+) suffix in the cell, for example R+. This is a result of moving a user to a role with additional access.

An unchecked or semi-checked checkbox also appears in the row header (user or role) or the column header (resource or member list) to indicate unresolved proposed changes for the corresponding user, role, resource, or member list.

All proposed roles and member lists are indicated by lowercase names. In this example:

- User USR0016 has at least one unresolved proposed change to the permissions.
  - User USR0015 has all proposed changes to the permissions, if any, resolved.
  - For the g005 role, proposed changes to some (but not all) of its users' permissions are not resolved.
  - g004 has all proposed changes, if any, resolved.
  - Proposed changes to the permissions for resource T001 are unresolved.
  - Proposed changes, if any, to the permissions for resource T009 are all resolved.
7. You can use the **Security Discovery Details** view that's included in the **Security Discovery** perspective to understand details about a role, member list, or anything by clicking it.

The view automatically shows the details of the selected item, and its relationships with other definitions. For more information, see [Security Discovery Details view in CICS Explorer documentation](#).

For example:

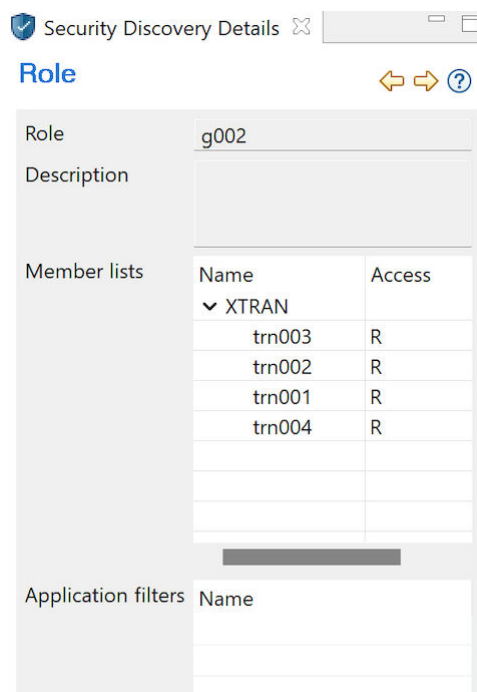


Figure 52. **Security Discovery Details** view for a role

8. If you want to adjust the grouping, right-click the users, roles, resources, or member lists to perform more actions. Only actions applicable to the selected item are displayed.

For example, you might want to use the **Select roles for user** option when your similarity analysis has identified a correct set of roles but certain users need to be in multiple roles. It adds a user to one or more roles without changing the permissions of either the source or the target roles. That is, the user inherits the permissions of the target roles.

As shown in [Figure 53 on page 128](#), roles g001 and g005 both have access to member list trn000, but role g001 also has access to trn005. You can collapse individual roles or members by using the twistie next to the role or member list, to focus on those of interest. In addition, you can collapse or expand all roles ( ) or member lists ( ).



Resource type filter=XTRAN: Application=No application: Displayed roles=6: Displayed member lists=6

		trn000						trn001	trn002	trn003	trn004	trn005			
		T010	T011	T013	T014	T016	T003	<...>	<...>	<...>		T012	T015	T017	
g000															
USR0068	Arnie Sidebottom														
g001		R+	R	R	R	R						R+	R+	R+	
USR0017	Robin Smith	R+	R	R	R	R						R+	R+	R+	
USR0039	Maurice Turnbull	R	R	R	R	R						R	R	R	
USR0065	Peter Lever	R	R	R	R	R						R	R	R	
USR0077	Sam Staples	R	R	R	R	R						R	R	R	
g002							R	R	R	R					
g003							R	R	R	R					
g004							R	R	R	R					
g005		R+	R+	R+	R	R+									
USR0015	Bill Brockwell	R	R	R	R	R									
USR0016	Bob Woolmer	R	R+	R	R	R									
USR0017	Robin Smith	R+	R	R	R	R									
USR0024	Harry Smith	R	R	R	R	R									
USR0026	Jack Martin	R	R	R+	R	R									

Figure 56. User USR0017 copied to g001

As a result, additional READ access (R+) to member list trn005 is proposed for user USR0017 (Robin Smith) in g001. The user now has access to member list trn000 by belonging to both role g001 and role g005.

- Continue adjusting the grouping by using all the applicable options, until you are happy with the grouping of users and resources.

For example, the **Move user with permissions** option moves a user to another role and updates the source and target role's permissions according to the permissions of the users within these roles. This option is useful when a proposed role contains users who actually need different access.

The **Move intersecting permissions from role** option moves a subset of the permissions in the selected role into a target role. Any users in the current role that are not already in the target role are added to it. The set of permissions to be removed must already be present in the target role and the target role must be a subset of the current role. You might want to use this option when you have a number of roles having overlapping permissions. Instead of defining each role with all the permissions that it requires, you can have a base role that contains all the users and each additional role with only access to additional resources required by that role.

- When the users and resources are in appropriate groups and member lists with proposed access, approve proposed changes that you want.
  - To approve a single proposed change (R+), right-click the cell and select **Approve proposed change**. The + suffix is removed.
  - To quickly approve all proposed changes to a user, role, resource, or member list, check the checkbox for the corresponding row (for a user or role) or column (for a resource or member list).
- You can rename the generated names of roles and member lists to meaningful names anytime. When you're happy with the adjustments, ensure that all your roles and member lists have appropriate names. They must be renamed for the security metadata to be imported back into RACF.

The following example shows the updated security definitions with all proposed changes resolved:



- To import the SDD into the security definitions table, click the **Load SDD File** icon (📎) on the editor toolbar.

The .sdd file must be from a CICS region running on the same z/OS system from which the security metadata (the .esm file) was exported so that the user IDs, resource names, classes, and SECPRFX values match.

- The editor then shows the grouped RACF definitions that are overlaid with SDD:

		T001	T009	T010	T011	T013	T014	T016	T003	T005	T006	T012	T015
HR	USR0068 Arnie Sidebottom	Rr	Rr	Rr	Rr	Rr	Rr	Rr				Rr	Rr
MANAGERS	USR0017 Robin Smith	R	Rr	R	Rr	Rr	Rr	R				R	R
	USR0039 Maurice Turnbull	Rr	Rr	Rr	Rr	Rr	Rr	Rr				Rr	Rr
	USR0065 Peter Lever	Rr	R	Rr	Rr	Rr	Rr	Rr				R	R
	USR0077 Sam Staples	Rr	Rr	Rr	Rr	Rr	Rr	Rr				Rr	Rr
SUPER	USR0042 Tom Dollyery								Rr	Rr	Rr		
	USR0090 Charles Leslie								Rr	Rr	Rr		
TELLERS	USR0015 Bill Brockwell	Rr	R	Rr	Rr	Rr	Rr	Rr					
	USR0016 Bob Woolmer	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0017 Robin Smith	R	Rr	R	Rr	Rr	Rr	R					
	USR0024 Harry Smith	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0026 Jack Martin	R	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0027 Albert Trott	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0052 Ray Robinson	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0057 Ian Davis	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0062 Percy Hornbrook	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0084 Don Blackie	R	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0087 H. D. G. Leveson G...	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
Unresolved	USR0002 George Duckworth												
	USR0015 Bill Brockwell												
	ICSDN16 Bob Woolmer												

Figure 58. RACF definitions overlaid with SDD

where the R symbol indicates that a user or group has READ access to a transaction. The r symbol indicates that a READ access request was performed at least once for that user or group while security discovery data was being captured in the running CICS regions.

Figure 58 on page 131 also shows multiple users in the Unresolved section. These rows show no access on this screen because although these users' transaction access is in appropriate roles, the SDD indicates they accessed other resource types. You will manage such resource access at a later stage.

- Optional: Loading the SDD file also enables the application filter section of the editor. This is not needed in this task, but you'll use it at a later stage to define application segmentation if you have multiple applications across your regions.

Figure 59. Application filter section

- Review and update access based on usage shown by the SDD.

For example, you might consider the following actions:

- Identify and review users who are in roles that grant them access beyond what they need in practice. These users would have R's to indicate access, but no r to indicate usage.
- Identify and review any user who used a transaction that they appear not to have access to. These users would have r's to indicate usage, but no R to indicate access. This can occur because the user was granted access after the .esm file was generated, or you removed the user's access in the [previous step](#).
- Ensure that roles do not grant users authority that they do not need.
- Ensure that member lists represent suitable grouping of transactions, such as those used by a particular role who needs to run an application.

As in the previous stage, you can create or delete roles and move users between roles to ensure that the users have only the access they require. For details, see Steps “7” on page 127 to “12” on page 129 in “Analyzing security definitions based on similarity”.

- When you're happy with the adjustments, ensure that all your roles and member lists have appropriate names.

The following example shows the updated security definitions with all proposed changes resolved:

		T001	T009	T010	T011	T013	T014	T016	T003	T005	T006	T012	T01
CLERK	USR0083 Peter Such								R	Rr	Rr		
CLERK2	USR0002 George Duckworth								R				
	USR0044 Tom Goddard								Rr				
	USR0061 Ian Chappell								Rr				
	USR0069 Des Hoare								Rr				
HR	USR0068 Annie Sidebottom												
MANAGERS	USR0017 Robin Smith	Rr	Rr	Rr	Rr	Rr	Rr	Rr				Rr	Rr
	USR0039 Maurice Turnbull	R	Rr	R	R	Rr	Rr	R				R	R
	USR0065 Peter Lever	Rr	R	Rr	Rr	Rr	Rr	Rr				Rr	Rr
	USR0077 Sam Staples	Rr	Rr	Rr	Rr	Rr	Rr	Rr				Rr	Rr
SUPER	USR0042 Tom Dollery								Rr	Rr	Rr		
	USR0090 Charles Leslie								Rr	Rr	Rr		
TELLERS	USR0015 Bill Brockwell	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0016 Bob Woolmer	Rr	Rr	Rr	Rr	Rr	Rr	Rr					
	USR0017 Robin Smith	R	Rr	R	Rr	Rr	Rr	Rr					
	USR0024 Harry Smith	Rr	Rr	R	Rr	Rr	Rr	Rr					
	USR0026 Jack Martin	R	Rr	Rr	Rr	R	R	Rr					

Figure 60. Transaction security definitions with all proposed changes resolved

- You can save changes to the security definitions anytime to resume analysis at a later point. Click **File > Save**.

This saves the current state of security definitions into a .sdm file, which you can open later in CICS Explorer.

## Results

The transaction security definitions are refined into roles and member lists based on actual usage.

## What to do next

You can continue analyzing the security definitions with security discovery data (SDD).

- If you have a single application in the regions being analyzed, see “Identifying or improving security definitions for resources with SDD” on page 132.
- If you have multiple applications in the regions being analyzed, see “Defining applications to segment work” on page 136.

If you want to use the security definitions you've got at this point in RACF, see “Exporting security metadata for review” on page 148.

## Identifying or improving security definitions for resources with SDD

6.2 Applies to 6.2.

It is assumed that you have RACF security definitions for transaction security. Unlike transactions, you might not have any RACF definitions for some or all of the other resource types. You can use the security discovery data (SDD) to identify these definitions. Therefore, after you finish adjusting the transaction definitions with SDD in CICS Explorer, you can go through the other resource types one at a time to identify the member lists and possibly additional roles.

For an overview of the workflow in CICS Explorer, see “Analyzing security definitions using security discovery in CICS Explorer” on page 122.



## Before you begin

- You have completed “Refining security definitions for transactions with SDD” on page 130 and stored the resulting .sdm file on your workstation for further analysis in CICS Explorer.


## About this task

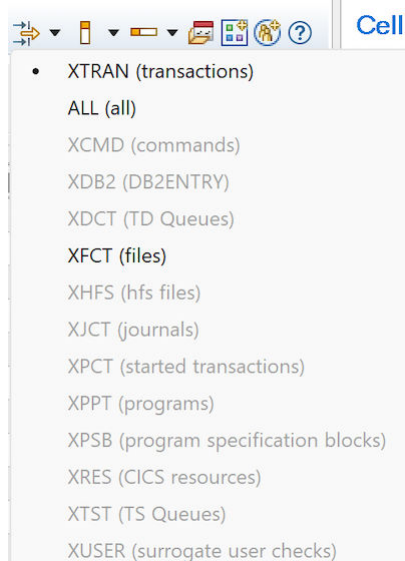
This example analyzes access to FILE resources. In this case, there are no existing RACF security definitions for files. The analysis is based on usage information from the SDD.

Although files are used as an example resource type, you need to repeat the procedure for all your resource types, one type at a time, by selecting a different resource in Step “2” on page 133 each time.

## Procedure

1. Ensure the security definitions are ready in the **Security Discovery** editor for further analysis.  
In CICS Explorer, click **File > Open file** to open the .sdm file. The **Security Discovery** editor opens, showing the security definitions saved last time in a table.
2. Ensure that the table shows the resource type you want to manage.

In this example, to manage FILE resources, click the **Filter resources** icon (  ) on the editor tool bar and select **XFCT (files)**.



*Figure 61. Filtering security definitions to show files*

Some resource types are disabled in the list because the resource was neither in the .esm file or the .sdd file.

3. In this case, because no RACF definitions exist for files, the security definitions table only shows file access data from the SDD. Prior to analysis, the user access appears in the Unresolved section and files in the Ungrouped section.

User ID Grouping

Figure 62. Security definitions overlaid with file access data from SDD

Lowercase r's and u's indicate READ and UPDATE requests that were detected in the running CICS region.

- To group resource definitions, click the **Analyze** icon (🔍) and select **Run best fit analysis**.

CICS Explorer then associates the resource access with the roles found in previous transaction analysis.

- The best fit analysis creates new roles for the file access based on the existing roles for the transaction access. The transaction roles are not changed. If a transaction-access-based role contains subsets of users that need different levels of access to resources, new roles are derived from the transaction-access-based roles, with a suffix in their names indicating the access level.

User ID Grouping

Figure 63. Role groups with proposed access to FILE resources

As shown in [Figure 63 on page 134](#), a role named MANAGERS was created during [previous analysis of transactions](#).

In this step, the best fit analysis proposes two more roles based on access to files: MANAGERS\_XFCT+r and MANAGERS\_XFCT+u. One role (MANAGERS\_XFCT+r) contains the users in MANAGERS who have READ access to certain files, while the other (MANAGERS\_XFCT+u) contains users in MANAGERS with UPDATE access to certain files. Because the analysis suggests that users and roles be given access, R+ and U+ are added to indicate different proposed access. Accompanying r's and u's show that the access requests were found in the SDD.

- You can then adjust the proposed grouping of file access as you did with transactions. For example, you might want to merge some of the roles that are generated as a result of file access into the transaction-access-based roles if the users need the same access.

You can also delete a role or approve its proposed changes as before. You might need to consult an application owner to understand who needs what access to the application.

For example, you might want to merge role MANAGERS\_XFCT+r into role MANAGERS because all the users who use the transactions also need to read the files at least. Right-click the role and select **Merge role MANAGERS\_XFCT+r into**.

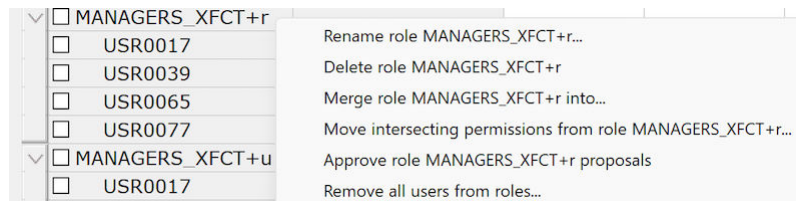


Figure 64. Right-click options for a role

- The **"Merge role"** dialog shows a list of roles that the selected role can be merged into. Select MANAGERS and click **OK**.

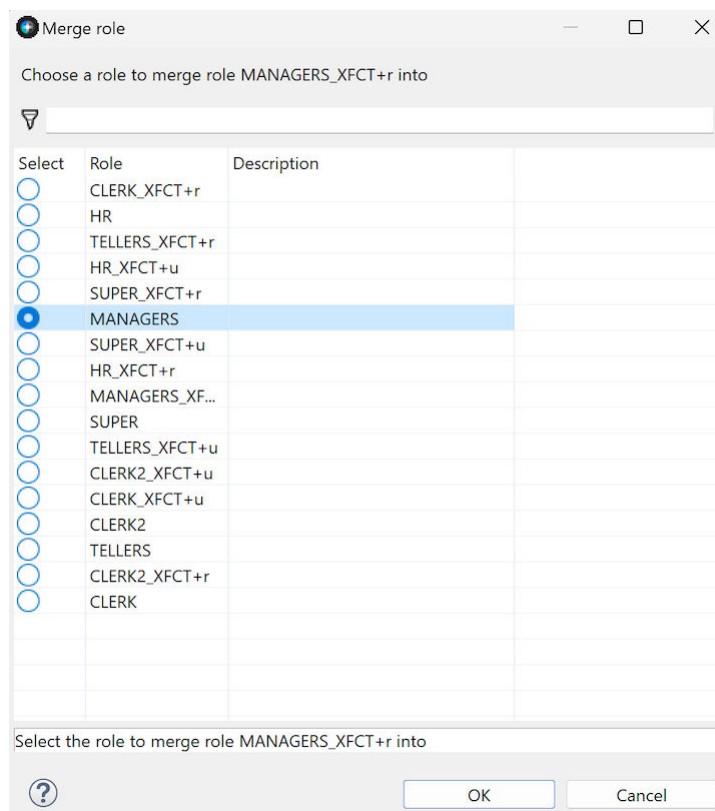


Figure 65. Merge role dialog

- The security definitions table shows that MANAGERS\_XFCT+r is merged into MANAGERS:

MANAGERS				R+r	R+r		R+r	R+r	R+r	R+r
USR0017	Robin Smith			R+r	R+r		R+r	R+r	R+	R+r
USR0039	Maurice Turnbull			R+r	R+r		R+r	R+r	R+r	R+r
USR0065	Peter Lever			R+	R+		R+r	R+	R+r	R+r
USR0077	Sam Staples			R+r	R+r		R+	R+r	R+r	R+r
MANAGERS_XFCT+u				U+u	U+u	U+u			U+u	U+u
USR0017	Robin Smith			U+u	U+u	U+u			U+	U+u
USR0039	Maurice Turnbull			U+u	U+u	U+u			U+u	U+u
USR0065	Peter Lever			U+	U+	U+u			U+u	U+u
USR0077	Sam Staples			U+u	U+u	U+u			U+u	U+u

Figure 66. Role MANAGERS+r merged into MANAGERS

9. For the proposed MANAGERS\_XFCT+u role, rename it to MNGRSUPD because it is a separate subset of the MANAGERS role that needs UPDATE access to certain files.
10. You might need to make changes to the member lists, either merging or splitting them, to ensure that users only have access to the resources that they need for their role.
11. When you're happy with the adjustments, ensure that all your roles and member lists have appropriate names.

The following example shows the updated security definitions with all proposed changes resolved:

		fct006	fct007	fct008	fct009	fct010	fct011	fct012	fct013
USR0069	Des Hoare	Uu							
HR		Uu	Rr						
USR0068	Arnie Sidebottom	Uu	Rr						
MANAGERS				Rr	Rr	Rr	Rr	Rr	Rr
USR0017	Robin Smith			Rr	Rr				
USR0039	Maurice Turnbull			Rr	Rr				
USR0065	Peter Lever			R	R				
USR0077	Sam Staples			Rr	Rr				
MNGRSUPD				Uu	Uu	Uu		Uu	Uu
USR0017	Robin Smith			Uu	Uu	Uu		U	Uu
USR0039	Maurice Turnbull			Uu	Uu	Uu		Uu	Uu
USR0065	Peter Lever			U	U	Uu		Uu	Uu
USR0077	Sam Staples			Uu	Uu	Uu		Uu	Uu
SUPER			Rr						
USR0042	Tom Dollery		Rr						
USR0090	Charles Leslie		Rr						
SUPERUPD			Uu						Uu
USR0042	Tom Dollery		Uu						Uu
TELLERS				Rr	Rr		Rr	Rr	Rr
USR0015	Bill Brockwell			Rr	Rr		Rr	Rr	Rr
USR0016	Bob Woolmer			R	R		R	R	Rr
USR0017	Robin Smith			Rr	Rr		Rr	R	Rr
USR0024	Harry Smith			Rr	Rr		Rr	Rr	Rr

Figure 67. Security definitions with all proposed changes resolved

12. You can save changes to the security definitions anytime to resume analysis at a later point. Click **File** > **Save**.

This saves the current state of security definitions into a .sdm file, which you can open later in CICS Explorer.

## Results

The security definitions are refined for the FILE resources based on actual usage. You can then refine security definitions for other resources, one type at a time. To refine other resource types, select a different resource type from Step “2” on page 133 and repeat the process.

## What to do next

“Exporting security metadata for review” on page 148

## Defining applications to segment work

6.2 Applies to 6.2.

If you have multiple applications in a set of CICS regions, it might be easier to analyze your data one application at a time. You can define applications based on their associated transactions. This allows you to analyze and manage security definitions application by application, making the process manageable.

For an overview of the workflow in CICS Explorer, see “Analyzing security definitions using security discovery in CICS Explorer” on page 122.

## Before you begin

You have completed “Refining security definitions for transactions with SDD” on page 130 and stored the resulting .sdm file on your workstation for further analysis in CICS Explorer.

## About this task

A CICS application can often be viewed as a set of entry points accessed by users in one or more roles. It can also be viewed as the transactions and resources that are used on its execution path.

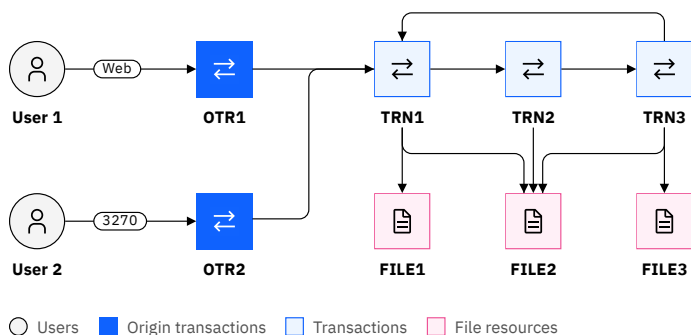


Figure 68. Application with associated transactions and resources

Figure 68 on page 137 shows an application that can be accessed through both a web interface and the 3270 terminal. As the execution path shows, the following users, transactions, and resources are associated with the application:

- Origin transactions: OTR1 and OTR2
- Other associated transactions: TRN1, TRN2, TRN3
- Used resources (in this case, all are files): FILE1, FILE2, FILE3
- Users accessing the application: User 1, User 2

It might be the case that some resources are shared between multiple applications. However, each application has a defined set of resources. Similarly, some roles might be shared by multiple applications.

In most cases, an entry point is associated with an origin transaction, usually to assist with accounting or workload balancing. Therefore, you can define an *application filter* to limit the *application transactions* to origin transactions and other transactions that were initiated (directly or indirectly) by the origins, and to limit the resources to those used by any of those transactions. These transactions and resources need to be in member lists. The application filter provided in CICS Explorer's **Security Discovery** editor helps you identify such member lists. For example, the transactions involved in Figure 68 on page 137 might be defined as follows in an application filter, where all the associated application transactions are found to be included in a member list APP1, which you should have created in the previous step:

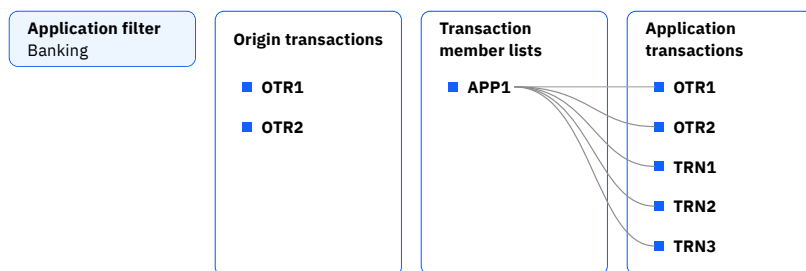


Figure 69. An example application filter


Another important aspect is the users who are to use the application. These users need to be associated with roles who are allowed to use the member lists that contain the application transactions. This can be done in the security definitions table as before. Example security definitions for the use case in Figure 68 on page 137 can be as follows:

		APP1				
		OTR1	OTR2	TRN1	TRN2	TRN3
ROLE		<b>Rr</b>	<b>Rr</b>	<b>Rr</b>	<b>Rr</b>	<b>Rr</b>
USER1	Username1	Rr	Rr	Rr	Rr	Rr
USER2	Username2	Rr	Rr	Rr	Rr	Rr

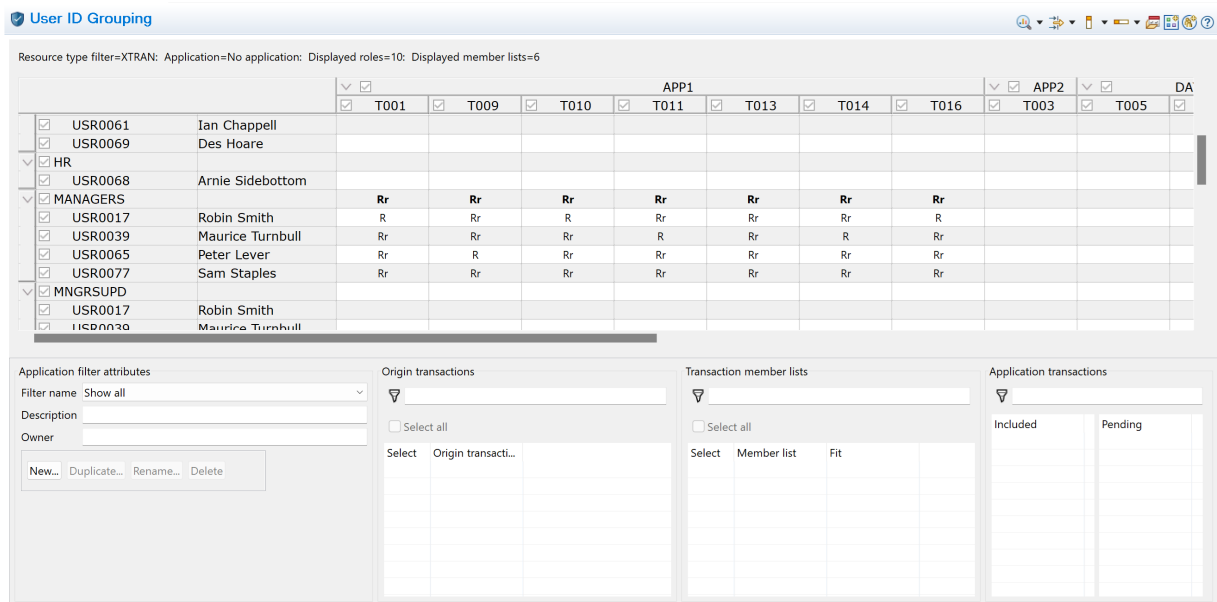
Figure 70. Example security definitions for the application filter

This task creates an application filter that identifies the origin transactions and other associated transactions for an application. It then refines the transaction member lists so that they contain only transactions for this application. You might need to work with an application owner to understand the origin transactions and other associated transactions of the application.

## Procedure

1. Ensure the security definitions are ready in the **Security Discovery** editor for further analysis.  
In CICS Explorer, click **File > Open file** to open the .sdm file. The **Security Discovery** editor opens, showing the security definitions saved last time in a table.
2. Ensure that the table shows transactions for analysis. If not, click the **Filter resources** icon (  ) on the editor tool bar and select **XTRAN (transactions)**.
3. To define an application, click **New** in the "**Application filter attributes**" pane.

**Note:** The application filter section is available only when transactions (XTRAN) are filtered to show and SDD is loaded.



The screenshot shows the 'User ID Grouping' window in CICS Explorer. The main table displays security definitions for APP1, with columns for roles (Rr) and transactions (T001, T009, T010, T011, T013, T014, T016, T003, T005). Below the table, the 'Application filter attributes' pane is visible, showing fields for Filter name, Description, and Owner. The 'Origin transactions' pane shows a list of transactions with columns for Select, Origin transaction, and Fit. The 'Transaction member lists' pane shows a list of member lists with columns for Select, Member list, and Fit. The 'Application transactions' pane shows a list of transactions with columns for Included and Pending.

Figure 71. Application filter section at the bottom

4. In the dialog that opens, enter the name of the application and click **OK**.

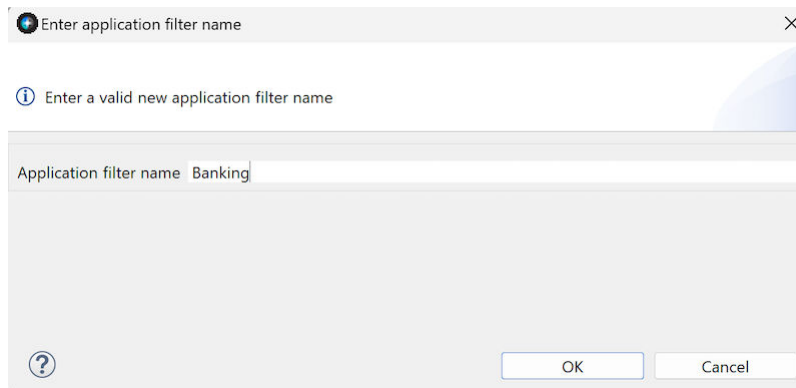


Figure 72. Entering an application filter name

5. Once a new application filter is created, it becomes selected in the **Filter name** list and the security definitions table shows member lists that are included in the application filter automatically.

In this case, no member lists have been defined to the application filter yet. Therefore, no member lists are shown in the table.

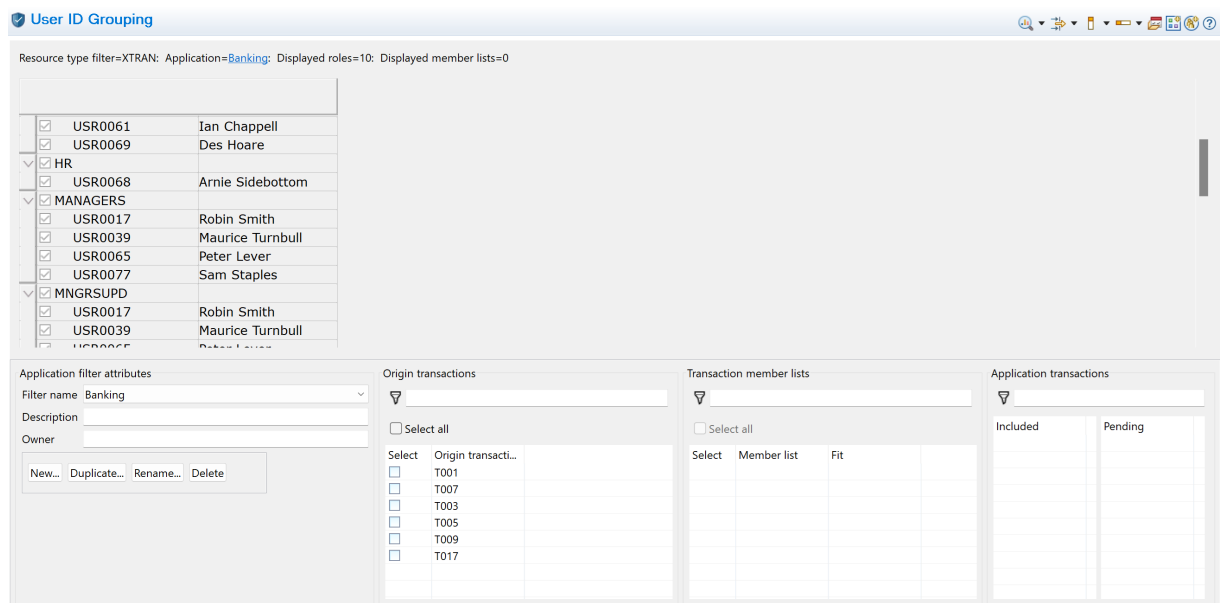


Figure 73. Security definitions table for the new application filter

6. Optional: Fill in the description and the owner of the application in the "**Application filter attributes**" pane. These fields can be a useful reminder of who needs to review the application security definitions.
7. The "**Origin transactions**" pane lists all the origin transactions that were captured from the running CICS regions when security discovery was active.

Select the origin transactions that are relevant to this application. If you do not know your origin transactions, you can select all origin transactions in the list and work from the list to deselect those that don't apply.

Alternatively, if origin transactions are not relevant, you can select all origin transactions and select the transaction member lists that define the application.

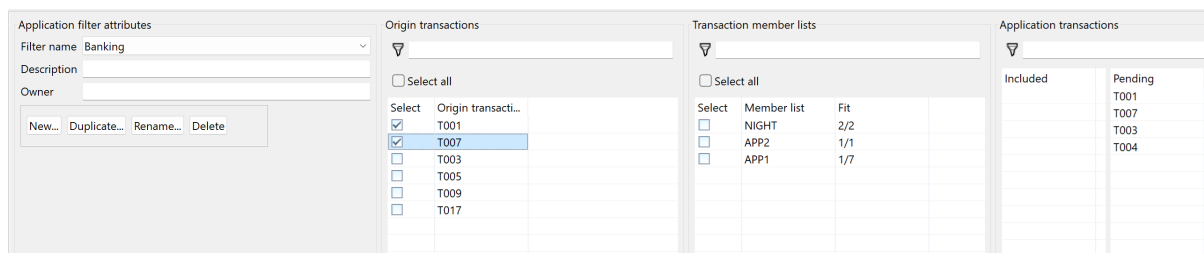


Figure 74. Selecting origin transactions

As origin transactions are selected, the "**Application transactions**" pane is populated with all the transactions that are associated with the selected origin transactions based on the SDD. These are initially added to the **Pending** list, to be processed later.

As new transactions are added to the **Pending** list, the **Member list** in the "**Transaction member lists**" pane is populated with member lists that contain one or more of the identified application transactions.

8. In the "**Transaction member lists**" pane, for a given member list, the **Fit** score is the number of its transactions that are associated with the selected origin transactions based on the SDD (that is, those listed in the "**Application transactions**" pane) divided by the total number of its members.

For example, in Figure 75 on page 140, member list NIGHT contains five transactions, all of which are associated with the selected origin transactions. Therefore, it has a fit score of 2/2. The higher the fit score, the higher the chance that the member list is relevant to the application.

As member lists are selected in the application filter, transactions that are in that member lists are moved to the **Included** list. The member lists and roles associated with those transactions will appear in the security definitions table.

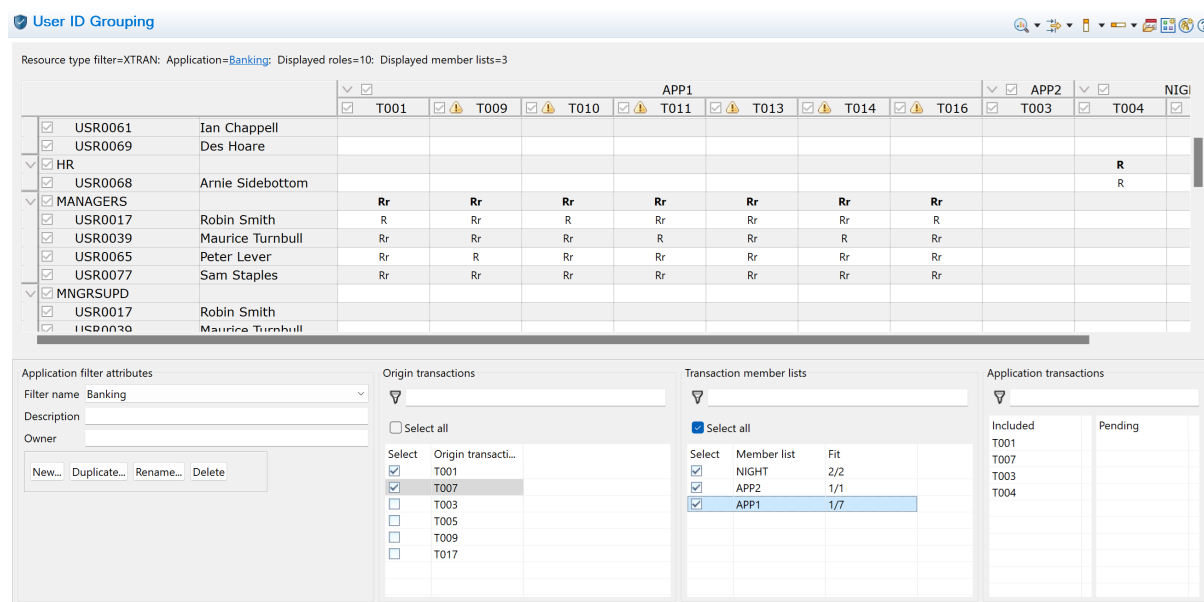


Figure 75. Selecting member lists

**Note:** When defining application filters, ensure that all the transactions that are part of the application end up in the **Included** list. Transactions that are not part of the application must not appear in the **Included** list. If an application shares one or more origin transactions with another application, the **Pending** list might contain transactions that are used by the other application. You might need to refine the transaction member lists to either create a common set of transaction, or move the common transactions into multiple applications member lists.

9. The member lists shown in the security definition table might include transactions that are not part of that application. These are flagged with a warning icon.



For example, as shown in Figure 75 on page 140, member list APP1 has a fit score of 1/7. That means out of its seven transactions, only one is associated with the selected origin transactions based on the captured SDD. Therefore, the six "irrelevant" transactions are flagged with a warning icon in the security definitions table.

**Notes:**

- There is a chance that these warned about transactions are not part of the application and should not be included in the current member list. However, it's also possible there wasn't enough SDD getting captured to prove the association. You might need to take your own use case into consideration.
  - In general, applications should have their own member lists containing only the transactions that are used by one or more roles who use the application.
10. You can use the **Security Discovery Details** view to better understand the relationships between the transactions and member lists.

The screenshot shows the 'Security Discovery Details' window. The 'Member list' section is expanded, showing details for 'APP1'. The 'Resource type' is 'XTRAN'. The 'Description' field is empty. The 'Resources' section is expanded, showing a list of transaction IDs: T001, T009, T010, T011, T013, and T014. The 'Roles' section is also expanded, showing a list of roles: TELLERS and MANAGERS. The 'Application filters' section is expanded, showing a list of filters: Banking.

*Figure 76. Details of a member list*

11. Repeat steps “7” on page 139 to “10” on page 141 until the application filter contains all the transactions that are used by the application.
12. When you're happy with the adjustments, ensure that all your roles and member lists have appropriate names.

User ID Grouping

Resource type filter=XTRAN: Application=Banking: Displayed roles=10: Displayed member lists=4

			APP1							APP2	
			T001	T009	T010	T011	T013	T014	T016	T003	T005
<input checked="" type="checkbox"/>	USR0061	Ian Chappell									
<input checked="" type="checkbox"/>	USR0069	Des Hoare									
<input checked="" type="checkbox"/>	HR										
<input checked="" type="checkbox"/>	USR0068	Arnie Sidebottom									
<input checked="" type="checkbox"/>	MANAGERS		Rr	Rr	Rr	Rr	Rr	Rr	Rr		
<input checked="" type="checkbox"/>	USR0017	Robin Smith	R	Rr	R	Rr	Rr	Rr	R		
<input checked="" type="checkbox"/>	USR0039	Maurice Turnbull	Rr	Rr	Rr	R	Rr	R	Rr		
<input checked="" type="checkbox"/>	USR0065	Peter Lever	Rr	R	Rr	Rr	Rr	Rr	Rr		
<input checked="" type="checkbox"/>	USR0077	Sam Staples	Rr	Rr	Rr	Rr	Rr	Rr	Rr		
<input checked="" type="checkbox"/>	MNGRSUPD										
<input checked="" type="checkbox"/>	USR0017	Robin Smith									
<input checked="" type="checkbox"/>	USR0039	Maurice Turnbull									

Application filter attributes

Filter name Banking

Description

Owner

New...

Duplicate...

Rename...

Delete

Origin transactions

Select all

Select	Origin transacti...
<input checked="" type="checkbox"/>	T001
<input checked="" type="checkbox"/>	T007
<input checked="" type="checkbox"/>	T003
<input checked="" type="checkbox"/>	T005
<input checked="" type="checkbox"/>	T009
<input type="checkbox"/>	T017

Transaction member lists

Select all

Select	Member list	Fit
<input checked="" type="checkbox"/>	APP1	7/7
<input checked="" type="checkbox"/>	NIGHT	2/2
<input checked="" type="checkbox"/>	DAY	2/2
<input checked="" type="checkbox"/>	APP2	1/1

Application transactions

Included

Pending

T001

T007

T011

T016

T003

T009

T004

T014

T010

Figure 77. Transaction security definitions defined by the Banking application filter

- You can also check the details of your final application filter in the **Security Discovery Details** view by selecting the application filter:

142 CICS TS for z/OS: Security for CICS

**Security Discovery Details**

**Application**

Name: Banking

Owner:

Description:

Origin transactions

Name
T001
T007
T003
T005
T009

Roles

Name
CLERK2
TELLERS
HR
MANAGERS
SUPER
C I FRK

Member lists

Name	Access
▼ XTRAN	
APP1	Rr
NIGHT	Rr
APP2	Rr
APP2	Rr

Figure 78. Details of the application filter

14. Define other applications as needed.

15. You can save changes to the security definitions anytime to resume analysis at a later point. Click **File** > **Save**.

This saves the current state of security definitions into a .sdm file, which you can open later in CICS Explorer.

## Results

You have defined applications by using the application filter to segment work. You have also adjusted the security definitions for transactions associated with each application.

## What to do next

After you have identified transactions that are associated with the application, you need to group resources that are used by these transactions into member lists that are accessed by roles who can run the application. For instructions, see [“Defining security for resources associated with each application” on page 144](#).

## Defining security for resources associated with each application

6.2

Applies to 6.2.

After you have defined an application filter with associated transaction member lists, you need to identify resources associated with each application. Then you can identify the security required for those resources using the security discovery data (SDD).

It is assumed that you have RACF security definitions for transaction security. Unlike transactions, you might not have any RACF definitions for some or all of the other resource types. You can use the security discovery data (SDD) to identify these definitions.

For an overview of the workflow in CICS Explorer, see [“Analyzing security definitions using security discovery in CICS Explorer”](#) on page 122.

### Before you begin

You have completed [“Defining applications to segment work”](#) on page 136 and stored the resulting .sdm file on your workstation for further analysis in CICS Explorer. This means you have identified all the origin transactions and other associated transactions as part of member lists, and have identified all the roles associated with those transactions for your application.

### About this task

Based on the roles identified for the application transactions, this task further identifies the member lists containing the resources that are used by the application and the roles associated with those resources. The procedure uses the FILE resources as an example. You need to repeat the procedure for all your resource types, one type at a time, by selecting a different resource in Step [“3”](#) on page 145 each time.

Based upon the simple application in [“Defining applications to segment work”](#) on page 136:

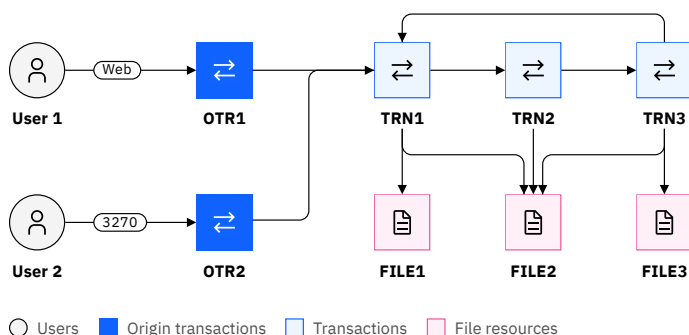


Figure 79. Application with associated transactions and resources

You will gain security definitions like this for its application filter, where definitions in yellow are achieved through this task.

		FI001			APP1				
		FILE1	FILE2	FILE3	OTR1	OTR2	TRN1	TRN2	TRN3
ROLE		Rr	Rr	Rr	Rr	Rr	Rr	Rr	Rr
USER1	Username1	Rr	Rr	Rr	Rr	Rr	Rr	Rr	Rr
USER2	Username2	Rr	Rr	Rr	Rr	Rr	Rr	Rr	Rr


Figure 80. Example security definitions for the application filter

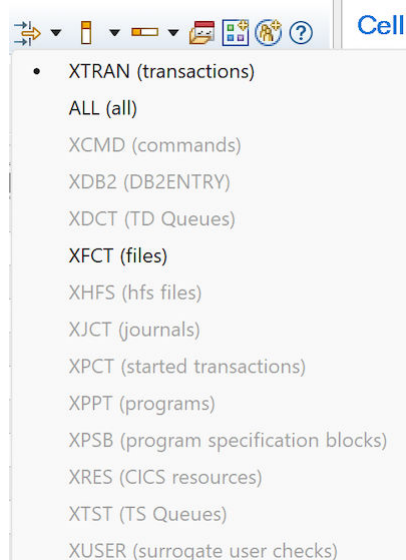
The roles identified in this task will probably be based on the previously identified transaction-access-based roles in the task “Defining applications to segment work” on page 136. This is because to access a resource, users need to use an applications. There might also be new roles that are subsets of the previously identified roles.

All of the resources used by the application, perhaps except the obscure error paths, will have been captured in the SDD.

## Procedure

1. Ensure the security definitions are ready in the **Security Discovery** editor for further analysis.  
In CICS Explorer, click **File > Open file** to open the .sdm file. The **Security Discovery** editor opens, showing the security definitions saved last time in a table.
2. Ensure that the application filter you want to work with is selected in the "**Application filter attributes**" pane.
3. Ensure that the table shows the resource type you want to manage.

In this example, to manage FILE resources, click the **Filter resources** icon (  ) on the editor tool bar and select **XFCT (files)**.



*Figure 81. Filtering security definitions to show files*

Some resource types are disabled in the list because the resource was neither in the .esm file or the .sdd file.

4. The security definitions table then shows the FILE resources that were accessed by the transactions associated with the application filter.  
In this case, because no RACF definitions exist for files, the security definitions table only shows file access data from the SDD. Prior to analysis, the user access appears in the Unresolved section and files in the Ungrouped section.

User ID Grouping

Figure 82. Security definitions of application Banking overlaid with file access data from SDD

Lowercase r's and u's indicate READ and UPDATE requests that were detected in the running CICS region.

- To group resource definitions for the selected application, click the **Analyze** icon (🔍) and select **Run best fit analysis**.

CICS Explorer then associates the resource access with the roles found in previous transaction analysis.

- The best fit analysis creates new roles for the file access based on the existing roles for the transaction access. The transaction roles are not changed. If a transaction-access-based role contains subsets of users that need different levels of access to resources, new roles are derived from the transaction-access-based roles, with a suffix in their names indicating the access level.

User ID Grouping

Figure 83. Role groups with proposed access to FILE resources for application Banking

As shown in Figure 83 on page 146, a role named MANAGERS was created during previous analysis of transactions for the application.

In this step, the best fit analysis proposes two more roles based on access to files: MANAGERS\_XFCT+r and MANAGERS\_XFCT+u. One role (MANAGERS\_XFCT+r) contains the users in MANAGERS who have READ access to certain files, while the other (MANAGERS\_XFCT+u) contains users in MANAGERS with UPDATE access to certain files. Because the analysis suggests that users

and roles be given access, R+ and U+ are added to indicate different proposed access. Accompanying r's and u's show that the access requests were found in the SDD.

7. You can then adjust the proposed grouping of file access as you did with transactions. For example, you might want to merge some of the roles that are generated as a result of file access into the transaction-access-based roles if the users need the same access.

You can also delete a role or approve its proposed changes as before. You might need to consult an application owner to understand who needs what access to the application.

For example, you might want to merge role MANAGERS\_XFCT+r into role MANAGERS because all the users who use the transactions also need to read the files at least. Right-click the role and select **Merge role MANAGERS\_XFCT+r into MANAGERS**. This option is offered because MANAGERS\_XFCT+r and MANAGERS\_XFCT+u were created as child roles of MANAGER.

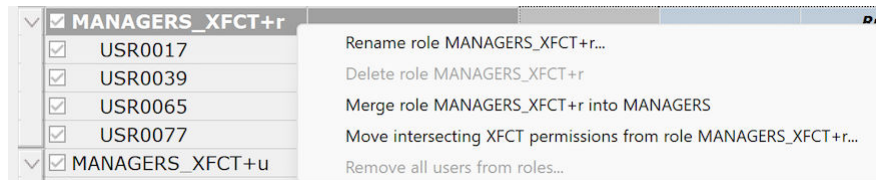


Figure 84. Right-click options for a role in application Banking

8. The security definitions table shows that MANAGERS\_XFCT+r is merged into MANAGERS:

<input checked="" type="checkbox"/>	MANAGERS				R+r	R+r		R+r	R+r	R+r	R+r
<input type="checkbox"/>	USR0017	Robin Smith			R+r	R+r		R+r	R+r	R+	R+r
<input type="checkbox"/>	USR0039	Maurice Turnbull			R+r	R+r		R+r	R+r	R+r	R+r
<input type="checkbox"/>	USR0065	Peter Lever			R+	R+		R+r	R+	R+r	R+r
<input type="checkbox"/>	USR0077	Sam Staples			R+r	R+r		R+	R+r	R+r	R+r
<input checked="" type="checkbox"/>	MANAGERS_XFCT+u				U+u	U+u	U+u			U+u	U+u
<input type="checkbox"/>	USR0017	Robin Smith			U+u	U+u	U+u			U+	U+u
<input type="checkbox"/>	USR0039	Maurice Turnbull			U+u	U+u	U+u			U+u	U+u
<input type="checkbox"/>	USR0065	Peter Lever			U+	U+	U+u			U+u	U+u
<input type="checkbox"/>	USR0077	Sam Staples			U+u	U+u	U+u			U+u	U+u

Figure 85. Role MANAGERS+r merged into MANAGERS

9. For the proposed MANAGERS\_XFCT+u role, rename it to MNGRSUPD because it is a separate subset of the MANAGERS role that needs UPDATE access to certain files.
10. You might need to make changes to the member lists, either merging or splitting them, to ensure that users only have access to the resources that they need for their role.
11. When you're happy with the adjustments, ensure that all your roles and member lists have appropriate names.
12. You can also check the details of your final application filter in the **Security Discovery Details** view by selecting the application filter:

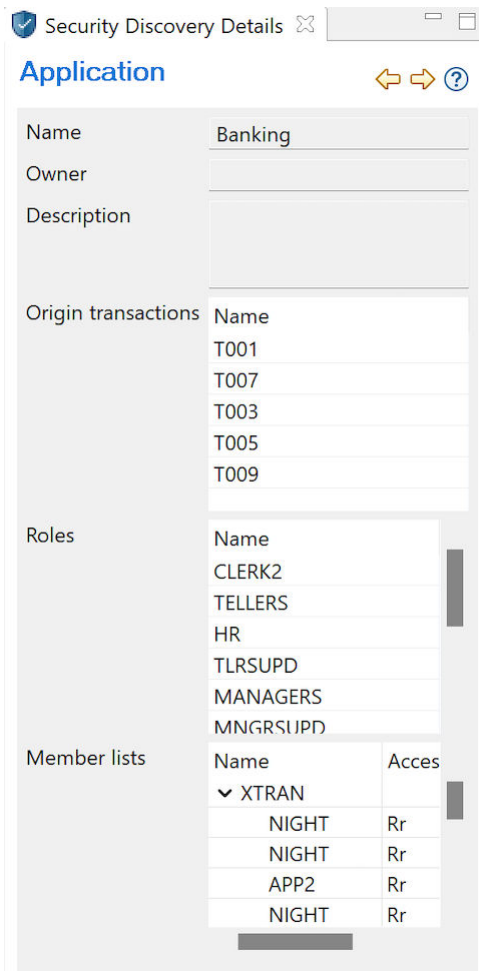


Figure 86. Details of the application filter

13. You can save changes to the security definitions anytime to resume analysis at a later point. Click **File > Save**.

This saves the current state of security definitions into a .sdm file, which you can open later in CICS Explorer.

## Results

You have identified FILE resources that are associated with each application and defined security for those resources.

You can then repeat that process for other resources, one type at a time. To manage other resource types, select a different resource type from Step “3” on [page 145](#) and repeat the process.

## What to do next

[“Exporting security metadata for review” on page 148](#)

## Exporting security metadata for review

In CICS Explorer, you can export the analyzed and updated security definitions into output security metadata for the application owner to review. It can then be converted back to RACF commands that a security administrator can use to update RACF definitions.

## Procedure

1. When the security definitions are ready for review, click **File > Export ESM file**.



**Note:** <sup>6.2</sup> If you have applied an application filter, the output security metadata (the .esm file) shows security definitions that are only applicable to the selected application.

2. The application owner reviews the .esm file to determine whether the updated security definitions are in line with their understanding of the application.

An example of security metadata that defines an application filter is as follows. For more information about the structure of security metadata, see [What is security metadata?](#).

```
--- # Security Metadata ---
version: 2
file_created:
  - date: "20 Mar 2023"
  - time: "18:24:26"
  - user: SUE
  - esm: "RACF"
application:
  - name: "Current Account"
  - owner: "Claude Gordon"
  - description: "Front office account application"
  origin_transactions:
    - BNK1
group_list:
  - name: MANAGER
    users:
      - MAINWRN
  - name: TELLER
    users:
      - WILSON
      - PIKE
      - JONES
user_list:
  - user: JONES
    username: "Jack Jones"
  - user: MAINWRN
    username: "George Mainwaring"
  - user: PIKE
    username: "Frank Pike"
  - user: WILSON
    username: "Arthur Wilson"
secprfx: NO
classes:
  - class: XTRAN
    name: CICSTRN
    profiles:
      - name: BANKING
        members:
          - BNK1
          - BNK2
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
  - class: XFCT
    name: CICSFCT
    profiles:
      - name: BANKING
        members:
          - CURRENT
          - SAVING
        access_lists:
          - access: READ
            groups:
              - MANAGER
              - TELLER
          - access: UPDATE
            groups:
              - MANAGER
```

When reviewing the security metadata, the application owner is advised to conform with the zero trust principle, to ensure that users are associated with roles that are granted access only to the member lists as their roles require.

To be more specific, they might ask these questions:

## 6.2 Is the list of origin transactions correct?

- If origin transactions are missing, it is possible that parts of the application are also missing. Therefore, the definitions are incomplete.
- If extraneous origin transactions are included, the definitions can include transactions or resources that are not part of the application and users can be granted more authority than needed.

### Are the proposed roles correct?

- Are the proposed roles appropriate to the application? Are roles missing? Should any of the roles be merged, or split?
- Does each role have correct users?

### Are the proposed member lists correct?

- Are the resources in each member list related?
  - Do the roles have correct access to each of the member lists?
  - Are there other member lists that need to be defined?
  - Should any of the member lists be merged or split?
  - Are there any resources missing?
3. After the review is complete, the application owner passes the reviewed security metadata back to the system programmer. The system programmer needs to manually make changes to his .sdm file in CICS Explorer accordingly.

## What to do next

The system programmer can export the definitions into a .esm file. The security administrator then converts the .esm file to RACF commands. For instructions, see [“Creating RACF commands from output security metadata of security discovery analysis”](#) on page 150.

## Creating RACF commands from output security metadata of security discovery analysis

The security administrator runs a JCL job to convert the output security metadata from security discovery analysis to RACF commands. These commands are used to create new classes.

For an overview of the whole security discovery process, see [How it works: CICS security discovery](#). This task is Step 4 in that flow.

## Before you begin

You must have the output security metadata produced from CICS Explorer reviewed and transferred back to zFS. See [Analyzing security definitions using security discovery in CICS Explorer](#).

## Procedure

1. The security administrator reviews the sample conversion REXX program DFH\$XSR and makes changes to the RDEFINE or PERMIT section of the code to add any customer specific information such as OWNER.
2. The security administrator needs to customize the DFH\$SM2R JCL, which is a sample in hlq.SDFHSAMP. What's included in the sample JCL is explained in [Figure 87 on page 151](#). Always go to hlq.SDFHSAMP for the latest sample.

```

//DFHSM2R JOB REGION=0M
//RACFCMD EXEC PGM=IRXJCL,PARM='DFH$XSR'
//SYSEXEC DD DISP=SHR,DSN=h1q.SDFHSAMP 1
//YAML DD PATH='/u/userid/cics/regionsA.ESM', File on zFS 2
// PATHOPTS=ORDONLY,
// FILEDATA=BINARY Data is in ascii
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//XCMD DD SYSOUT=*
//XDB2 DD SYSOUT=*
//XDCT DD SYSOUT=*
//XFCT DD SYSOUT=* 3
//XJCT DD SYSOUT=*
//XPCT DD SYSOUT=*
//XPPT DD SYSOUT=*
//XPSB DD SYSOUT=*
//XRES DD SYSOUT=*
//XTRAN DD SYSOUT=* 3
//XTST DD SYSOUT=*
//GROUPS DD SYSOUT=* 4
//SECMETA DD SYSOUT=* Sec metadata in ebcdic
//SYSIN DD *
SECPRFX=NO 5
XTRAN=TESTTRN 6
XFCT=TESTFCT 6
//

```

Figure 87. Example sample JCL DFH\$SM2R

The parameters are as follows:

- 1 The JCL uses REXX programs in the CICS sample library. Change h1q to the installed library's high-level qualifier.
  - 2 Specifies the name of the output security metadata file in the USSCONFIG directory.
  - 3 Output DD statements used for each of the new class definitions that are generated.
  - 4 An output DD statement used for the group definitions.
  - 5 The SECPRFX to be used to generate the profiles. This would be as specified on the SIT parameter.
  - 6 The CICS classes to be generated as they would be specified on the SIT parameters. It is recommended that you use different class names from those that you currently use. This allows you to migrate to the new RACF by changing the SIT value and restarting the CICS regions. Likewise, if a problem occurs, it's easy to back out the changes.
3. The security administrator submits the job. There are output DD statements for each class and the groups.

The command output looks like the following examples:

Figure 88. Example command output

Groups:

```

ADDGROUP MANAGER
CONNECT MAINWRN GROUP(MANAGER)
ADDGROUP TELLER
CONNECT PIKE GROUP(TELLER)
CONNECT WILSON GROUP(TELLER)
...

```

XTRAN:

```

RDEFINE GPRODTRN APP1 ADDMEM(MENU,ABCD) UACC(NONE)
PERMIT APP1 CLASS(GPRODTRN) ACC(READ) ID(MANAGER)
PERMIT APP1 CLASS(GPRODTRN) ACC(READ) ID(TELLER)
...

```

4. The security administrator reviews the definitions to check whether they impact existing definitions. When happy with these definitions, the security administrator can run the commands.

**Recommended:**

- a. Security definitions are created only for resources that are found in the ESM and SDD files. If there are resources that are not found in those files, this results in a NOTAUTH failure with associated CICS messages DFHXS1111 and DFHXS1117. It does not result in an ICH408I. It is therefore recommended that you add a generic definition to report on these failures.

```
RDEFINE TPRODTRN * UACC(NONE)
```

- b. For migration purposes, you might want to add WARNING to the generic definition to allow the application to work unchanged, but report missing security definitions. You only need to do this for resource types that you didn't previously protect. Never apply it to the transaction class.

```
RDEFINE MPRODPT * UACC(NONE) WARNING
```

## Results

The relevant CICS regions are updated with the new classes and the regions are restarted.

## What to do next

Implement required security definitions in the regions by setting the Xnnn SIT parameters to the new classes defined in the previous steps.

Then change the transaction definitions to RESSEC (YES) and CMDSEC (YES) to enable resource and command security checking. This can be implemented in stages by changing the transactions for one application at a time, which helps minimize the risk. If there are transactions that are shared between applications, you might want to handle these as a separate stage.

When all transaction definitions are changed to RESSEC (YES) and CMDSEC (YES) in a region, set the SIT parameters **RESSEC** and **CMDSEC** to ALWAYS. This ensures that future transactions are always security checked.

## CICS security discovery considerations

Support for CICS security discovery has some restrictions, which you need to consider when using security discovery.

Affected areas and recommendations are listed as follows.

### Generic profiles

Generic profiles are not recommended as part of a zero trust strategy because they require other controls to ensure that definitions are added during development. Generic profiles can be extracted into security metadata, but cannot be analyzed in the **Security Discovery** perspective provided by CICS Explorer, or used to create RACF definitions from the security metadata.

The tool that extracts RACF definitions into security metadata identifies any generic profiles in the data with the following message:

```
WARNING: Profile C* in class CICSTRN is generic
```

The **Security Discovery** perspective in CICS Explorer displays generic profiles and the RACF access that is granted to them but cannot associate them with SDD for analysis.

6.2

Although the security discovery data (SDD) identifies the transactions and used resources to which access is granted by a generic profile, it does not associate them with the generic profile.

### Generic SECPRFX values

Only discrete SECPRFX values are supported. Generic SECPRFX values in resource profiles are not supported. Generic SECPRFX values might occur when you have different SECPRFX values in CICS regions

that are processed by a single RACF profile. For example, PROD\*.TRAN is the transaction TRAN for SECPRFX values PROD1 and PROD2.

The tool that extracts RACF definitions into security metadata identifies all the SECPRFX values in the classes that it scans with the following message:

```
Additional SECPRFX value AC* found in class CICSTRN
```

It identifies generic SECPRFX values with the following message:

```
WARNING: Generic SECPRFX value AC* found in class CICSTRN
```

If you are extracting security metadata for a SECPRFX that doesn't overlap this generic value, the warning can be ignored.

## Universal access authority (UACC)

A zero trust strategy recommends that access is only granted for roles that need to access a resource. This means that normally in your profile definitions universal access (UACC) should be set to NONE.

The tool that extracts RACF definitions into security metadata identifies UACC values other than NONE in the data with the following message:

```
WARNING: Profile ALLUSER in class CICSTRN has UACC(READ)
```

In addition, the **Security Discovery** perspective in CICS Explorer ignores the UACC values in the security metadata and assumes UACC (NONE).

The only case where the security metadata CICS Explorer outputs contains a `universal_access: NONE` tag is when a profile is defined with UACC (NONE), but no groups (roles) or users are granted access. In this case, the following message is issued:

```
WARNING: Profile NOUSER in class CICSTRN cannot be accessed by any users
```

## Supported classes

Only CICS classes are supported. That is, resource classes that are controlled by the **Xnnn** SIT parameters. For a list of CICS classes, see "RACF-provided resource classes for CICS resources" in [Summary of RACF classes for CICS resources](#).

Information about [surrogate security](#) (controlled by the **XUSER** SIT parameter) cannot be extracted from RACF definitions into security metadata, or analyzed by the **Security Discovery** perspective in CICS Explorer. When you submit a job to extract RACF definitions to security metadata with the **XUSER** parameter specified in the JCL, CICS ignores requests to extract surrogate information and outputs this message:

```
Invalid line ignored: XUSER=YES
```

**6.2** Information about CICS surrogate security is collected in the SDD.

## Special user IDs

[Special user IDs](#), such as link user IDs, default user IDs, and region user IDs are not identified by the **Security Discovery** perspective in CICS Explorer. You might want to include them in separate groups from user groups. These special user IDs can be identified in the [Security Discovery Records](#) view in CICS Explorer, or in the CSD definitions and SIT parameters.

## Performance Considerations

Analysis of roles and resource groups is a complex process. For this reason, it's recommended that you run the **Security Discovery** perspective in CICS Explorer on a machine with at least 32 G of physical memory, 6 cores, and a clock speed of at least 2.5 GHz.

The processing time increases with the size of the RACF member lists and user groups. In particular, member lists with over 100 resources will be computationally expensive.

For CPU intensive operations, a process bar appears in CICS Explorer. Long operations can be canceled to restore the system back to the previous state.

## Support for other ESMs

The following capabilities of security discovery are supported regardless of the external security manager (ESM) in use:

- **6.2** SDD. However, some differences can exist:
  - The SDD file might contain additional resource accesses if a non-RACF ESM is used.
  - Some statistics are not recorded for a non-RACF ESM. The affected fields are **Exempt resources**, **Resource class not active**, **FATAUTH calls**, and **AUTH calls**.
- The **Security Discovery** perspective in CICS Explorer that allows you to analyze security definitions

Only RACF is supported in terms of extracting RACF definitions into security metadata and converting the metadata back to RACF commands. If you are interested in support for other ESMs, please contact your vendor.

### Related information

[How it works: Zero trust in CICS](#)

## Capturing security definitions during development using security definition capture

---

**6.2** Applies to 6.2.

Security definition capture (SDC) enables an application developer to capture the security definitions required by an application during their development process. You start SDC before testing an application to record security definitions that are required by the application if security were on. You can invoke SDC through either the CXSD transaction or an HTTP REST interface, depending on the applications to be tested.

These instructions describe how to start and stop SDC manually in a manual testing scenario. Alternatively, you can automate the control of SDC and application testing, for example, by using the [Galasa SDV Manager](#). See [Introducing Security Definition Validation to your CICS TS Application CI pipeline](#) for an example.

For more information about how SDC or SDV works, see [How it works: Capturing and validating security definitions during the development process](#).

### Before you begin

- You must have already [configured SDC in the CICS region](#) where the application is to be invoked.
- Developers have installed the CICS applications to be tested in the development region as usual.
- The region where SDC runs is defined with SEC=YES. However, other security SIT parameters are optional. SDC captures the required security definitions irrespective of those SIT parameters.

### About this task

You can start or stop SDC in either of the following ways:

- For 3270 applications, by running the [CXSD](#) transaction
- For the other applications, by calling the HTTP REST interface

**Recommendation:** It's recommended that whoever invokes SDC use their user ID that is dedicated to testing or development and do not perform other tasks in the system during the testing. This way, SDC records only the security definitions that are relevant to the application. This means:

- If you're using the CXSD transaction, you need to sign on to CICS using a dedicated user ID before running CXSD.
- If you're calling the HTTP REST interface, you need to supply a dedicated user ID when prompted for a user ID.

You can follow the procedure to collect a baseline of the application's required security definitions, or repeat it to continuously compare required changes to the security definitions for maintenance.

## Capturing security definitions for 3270 applications

To test 3270 applications, the developer runs the CXSD transaction to start and stop SDC.

1. When the application is ready for testing, invoke CXSD ON to start SDC of the terminal.
2. While SDC is on, perform manual testing as usual, ensuring the range of paths are tested.
3. Invoke CXSD OFF to stop SDC, which results in a message that indicates the SDC job has been submitted.

## Capturing security definitions for the other applications

To test applications of the other types, the developer starts and stops SDC through the HTTP REST interface. The endpoint takes the form of `http://endpoint:port/dfhsdc`, depending on the configuration described in [Configuring security definition capture \(SDC\)](#).

1. To start SDC, send a POST request without any input parameters to the endpoint.

If SDC is started successfully, you will see this response.

```
HTTP/1.1 201 Created
{
  "srr_id": "_1053208",
  "message": "SDC HAS BEEN SWITCHED ON"
}
```

For a complete list of all responses of the SDC calls, see [Responses to SDC HTTP calls](#).

2. While SDC is on, perform manual testing as usual, ensuring the range of paths are tested.
3. When testing is complete, send a DELETE request to stop SDC and submit a job automatically. The request body must be in the JSON format, for example:

```
{
  "submit": true
}
```

### submit

Controls whether a job is submitted when SDC stops.

It is a boolean value: true or false. If no parameter is specified, true is the default.

When set to true, a job is submitted automatically when SDC stops.

If you want to submit the JCL yourself, you can set it to false to stop SDC without submitting a job.

## Result

You have captured required security definitions for the tested applications. The required security definitions are output into [security metadata](#).

You can check the job is submitted and completes successfully based on configuration described in [Configuring security definition capture \(SDC\)](#).

## What's next

1. Review the results of the batch job.

If you have configured SPOOL to submit the job that generates the security metadata, the job is submitted under your user ID. You need to pass the results to the reviewer.

If you have configured the CXST TDQUEUE to submit the job, the job is submitted under the user ID specified on the JOBUSERID attribute of the TDQUEUE. This user is often the reviewer.

The reviewer must be in a role that has authority to view sensitive security definitions, such as a security administrator. The reviewer then ensures that the required security definitions as indicated in the security metadata are expected.

2. When the security metadata is verified, the security administrator updates security definitions in the ESM. The user ID in the security metadata is replaced by a group name representing the role of the user who uses the applications.

### Related concepts

[How it works: Capturing and validating security definitions during the development process](#)



---

# Chapter 11. Security through the network layers

Security mechanisms used by CICS apply at different layers of a networking architecture such as TCP/IP.

## **Application layer**

Security measures at this layer are application-specific.

## **API layer**

Security protocols such as TLS operate above the TCP transport layer. These protocols use sockets to interface with the transport layer.

## **Transport layer**

Security at the transport layer protects communication between two hosts, commonly web (HTTP) traffic.

## **Network layer**

Security at this layer can provide protection transparently to applications.

---

## Transport Layer Security (TLS)

CICS TS supports the Transport Layer Security (TLS) protocol. The TLS protocol supersedes the Secure Sockets Layer (SSL) protocol and the terms are often used interchangeably. This documentation uses the term TLS to refer to either TLS or SSL, unless it is explicitly referencing a version of the specification or the name of a `server.xml` configuration file element. Specifically, CICS supports TLS 1.1, 1.2 and 1.3. TLS 1.1 is stabilized. For more information, see [Stabilization notices and discontinued functions](#).

The TLS protocol consists of two layers, the TLS Record Protocol and the TLS Handshake Protocol.

- The TLS Record Protocol provides connection security and has the following properties:
  - The connection is private. Secret key cryptography is used for data encryption. The keys for this secret key encryption are generated uniquely for each connection and are based on a secret that is negotiated by a handshake.
  - The connection is reliable. Message transport includes a message integrity check by using a keyed-hash MAC (HMAC). Secure hash functions such as SHA-1 or MD5, are used for MAC computations.
- The TLS Handshake Protocol operates with the TLS Record Protocol to allow the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before any data is transmitted. The TLS Handshake Protocol provides connection security that has the following properties:
  - The peer's identity can be authenticated by using public key cryptography.
  - The negotiation of a shared secret is secure and the negotiated secret is not available to eavesdroppers. For any authenticated connection, the secret cannot be obtained, even by an attacker who can intercept the connection.
  - The negotiation is reliable. No attacker can modify the negotiation communication without being detected by the parties in communication.

When CICS acts as an HTTP client and attempts to communicate over TLS with a peer that acts as a virtual host, the hostname is passed in the Server Name Indication (SNI) extension during the handshake. The passing of the hostname is in conformance with the TLS handshake protocol that is specified in Internet Engineering Task Force RFC 6066.

An encrypted TLS connection is established after a handshake takes place between the client and the server. A digital certificate for the server is passed to the client. The client knows that the server can be trusted by which certificate authority (CA) signed the server certificate. If the CA is trusted, then the server certificate is trusted, and the TLS session is set up. For more information about the TLS 1.2 handshake, see [TLS 1.2 Protocol](#). For more information about the TLS 1.3 handshake, see [TLS 1.3](#).

For TLS requests sent to a server, the z/OS Connect server acts as the "server" and the "client" is either the REST client when z/OS Connect acts as an API provider, or CICS, IMS or a z/OS application when z/OS Connect acts as an API requester.

For TLS requests sent from a z/OS Connect server, the z/OS Connect server acts as the "client" and the "server" is either the System of Record when z/OS Connect acts as an API provider, or a RESTful API endpoint when z/OS Connect acts as an API requester.

## Application Transparent Transport Layer Security (AT-TLS)

---

Application Transparent Transport Layer Security (AT-TLS) can be used to create secure socket sessions on behalf of CICS. Instead of implementing Transport Layer Security (TLS) in CICS, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the Policy Agent. When AT-TLS is active for a CICS socket connection, CICS sends and receives cleartext (unencrypted data), while AT-TLS encrypts and decrypts data at the TCP transport layer. As a result, CICS system initialization parameters that are related to SSL/TLS, such as KEYRING and MINTLSLEVEL/ENCRYPTION are not required because the implementation of TLS is provided by the AT-TLS policy statements and all encryption and decryption is done outside of the CICS address space.

For more information about AT-TLS and AT-TLS policy setup, see [AT-TLS policy configuration in z/OS Communications Server: IP Configuration Guide and Policy Agent](#) and [policy applications in z/OS Communications Server: IP Configuration Reference](#).

Address spaces such as CICS that are taking advantage of AT-TLS can be separated into three different types, based on whether awareness of the service is needed and, if so, the amount of control that the address space is given over the security functions. These types are:

### AT-TLS basic

The address space is unaware that AT-TLS is encrypting or decrypting data. This is the only option available for releases of CICS TS earlier than V5.3.

CICS cannot access client certificates as it is unaware of a certificate that is associated with the socket. Also, when CICS uses HTTP redirection with this mode of operation, there are failures because CICS assumes that the client connection is HTTP rather than HTTPS. A problem that might arise (for example) is when you are using **AUTHENTICATE(BASIC)** on an HTTP TCPIP SERVICE, and CICS discovers that the user's password expired. A dialog is triggered with the user to request a new password. At the end of this dialog, there is an error because the resubmission of the original HTTP request specifies HTTP as the scheme instead of HTTPS.

### AT-TLS aware

The CICS address space is aware of AT-TLS and can query information such as AT-TLS status and access items such as the client certificate and the certificate user ID. This is supported by CICS TS from Version 5.3. When a TCPIP SERVICE is defined as AT-TLS aware, CICS issues an AT-TLS query to obtain information such as AT-TLS security status, negotiated CIPHER suite, partner certificate, and derived RACF user ID.

All new client connections for the AT-TLS aware TCPIP SERVICE are queried to extract their AT-TLS attributes. When a client establishes a new connection to a CICS TCPIP SERVICE defined with **SSL(ATTLSAWARE)**, it triggers the query and CICS accepts the new client connection.

### AT-TLS controlling

The address space is aware of AT-TLS and can control the secure session on a socket. This is not supported by CICS TS.

All new client connections for the **SSL(ATTLSAWARE)** TCPIP SERVICE are queried to extract their AT-TLS attributes. When a client establishes a new connection to a CICS TCPIP SERVICE defined with **SSL(ATTLSAWARE)**, it triggers the query and CICS accepts the new client connection.

## Changing TLS protocol level or ciphers safely

---

If a TLS protocol is compromised, you need to switch to a higher minimum TLS level. If a cipher is compromised, you need to remove the cipher from all TLS connections. To safely make a change, you

need to identify if the compromised protocol or ciphers is in use. When you know it is in use, you need to identify who is using it so that the client or server that CICS is communicating with can also be upgraded. CICS statistics and monitoring can be used to help you evaluate the TLS protocol levels and ciphers in use and identify the impact of a change to you and your users.

## Why evaluate TLS protocol levels or ciphers in use?

Using older or compromised TLS protocol levels or ciphers is a risk. Using higher TLS protocol levels or newer ciphers can mitigate such risks. You might also be asked to switch to a newer TLS protocol level or cipher as a result of a security alert or audit requirement. To achieve this switchover safely, it is important for you to know which protocols or ciphers are in use and by who. With this knowledge, you can plan and move to more secure options with minimal risk.

Several steps are involved in making such a move:

1. Identify what TLS protocols or ciphers are in active use by gathering relevant statistics data.
2. Evaluate whether the older protocols or ciphers are still in use.
3. Identify applications that need to upgrade from the older TLS protocol level or ciphers that are no longer to be used by reviewing monitoring data.
4. Address the applications and ensure they are updated to use the new higher-level TLS protocols.
5. When all previous steps are complete, update your configuration to use a higher MINTLSLEVEL or remove a cipher.

For more information about TLS protocol information within TCP/IP statistics, see [TCP/IP global and TCP/IP Service statistics](#).

For more information about cipher statistics, see [Cipher statistics](#).

For more information about the data fields SOTLSLVL and SOCIPHER, see [Performance data in group DFH SOCK](#).

Consider these two scenarios, one for [removing a lower TLS level](#) and the other for [removing a compromised cipher](#).

## Example scenario for removing a previous TLS level

You're advised by a security alert or an auditor that TLS 1.1 is no longer sufficient for your business and you must remove this protocol level. To make this change, all connections between CICS regions and clients or servers must be running at TLS 1.2 or higher.

Connections use the highest level that is supported by both sides. Therefore, if a connection is at TLS 1.1, one side needs to be upgraded or configured to support TLS 1.2.

You can use the available CICS statistics and monitoring to help you with this exercise.

1. The CICS region must have its MAXTLSLEVEL set to *TLS12* or higher.
2. All connections definitions (TCPIP SERVICE, IPCONN, URIMAP) must include ciphers that are supported by TLS 1.2 or higher.

**Recommended:** It is recommended that you use the cipher files rather than numeric ciphers.

3. Gather statistics for the TLS protocols. Set up gathering of statistics to identify the TLS protocols in use by using CEMT PERFORM STATISTICS RECORD. This data gathering needs to be done on all regions with inbound or outbound connections. Only data from new handshakes is collected during the defined gathering period. You must ensure that the gathering period is sufficient to capture the data you require.
4. Check whether the gathered statistics identified any handshakes for the TLS 1.1 level protocol. See [Figure 89 on page 161](#) for an example of how the statistics output shows which TLS protocols were involved in handshakes for the selected time period. If you find no cases, you can proceed to step 8.
5. For more detailed analysis, use the monitoring information to determine which TLS protocol levels were used during the selected monitoring period. The performance data field SOTLSLVL shows which

applications used which TLS protocols. You can also use other data in the monitoring results to identify IP addresses and port numbers to assist you with identifying the client or server that is associated with the connection.

6. Update the associated client or server that is reported as using TLS 1.1. See [Figure 90 on page 161](#) to see the SOTLSLVL information. Other TLS information such as the cipher used, TCP/IP address, and port number is also available.
7. Run the statistics and monitoring steps again to confirm that no remaining applications use the TLS protocol you want to remove.
8. Update MINTLSLEVEL to the new minimum protocol level.

## Example scenario for removing a compromised cipher

You're advised by a security alert or an auditor that a specific cipher is no longer sufficient for your business and you must remove this cipher.

The cipher depends on the configuration and capabilities of both partners in a connection. The client and the server each provides a list of one or more ciphers that are used as part of the TLS handshake. The cipher that is used is the first one in the server's list, which is also in the client's list. Therefore, it is important that before you remove a cipher, other ciphers are available, and used.

You can use the available CICS statistics and monitoring to help you with this exercise.

1. Move the cipher that you want to remove to the end of either:
  - The list in cipher files.
  - The numeric ciphers specified in the CIPHERS options of connection definitions (TCPIP SERVE, IPCONN, and URIMAP).
- Recommended:** It is recommended that you use the cipher files rather than numeric ciphers.
2. Gather statistics for the cipher. Set up gathering of statistics to identify new handshakes that use ciphers by using CEMT PERFORM STATISTICS.RECORD. This data gathering needs to be done on all regions with inbound or outbound connections. Only data from new handshakes is collected during the defined gathering period. You must ensure that the gathering period is sufficient to capture the data you require.
3. Check whether the gathered statistics identified any relevant data for the selected cipher. See [Figure 89 on page 161](#) for an example of how the statistics output shows which ciphers were used for the selected time period. If you find no cases, then you can proceed to step 7.
4. For more detailed analysis, use the monitoring information to determine which ciphers were used during the selected monitoring period. See [Figure 90 on page 161](#) to see the SOCIPHER information. You can also use other data in the monitoring results to identify IP addresses and port numbers to assist you with identifying the client or server that is associated with the connection.
5. Either update the associated clients or servers to add a cipher is acceptable to the CICS region. Or update the CICS cipher files to add a cipher that is acceptable to the clients or servers. This cipher must be in front of the cipher that you want to remove.
6. Run the statistics and monitoring steps again to confirm that no remaining applications use the ciphers you want to remove.
7. Remove the cipher definition.

## Example screens

-----			
TLS LEVEL			
-----			
CICS Configured TLS Level and Handshake Type		Inbound	Outbound
Full		1	0
Abbreviated		0	0
TLS 1.1		1	0
TLS 1.2		0	0
TLS 1.3		0	0
TOTAL		1	0
AT-TLS SSL 3		0	0
AT-TLS 1.0		0	0
AT-TLS 1.1		0	0
AT-TLS 1.2		1	1
AT-TLS 1.3		0	0
TOTAL		1	1
CICS 7.4.0 Statistics Utility Program		Report Date 02/23/2022	
Report Time 16:11:39 Page 106			
Requested Statistics Report		Collection Date-Time 02/23/2022-16:10:49	
Applid IYK4ZON1 Jobname D1		Last Reset 16:10:34	
-----			
TLS CIPHER STATISTICS			
-----			
AT-TLS		TLS	TLS
Number	Cipher Name	Inbound	Outbound
Outbound			Inbound
-----			
009D	TLS_RSA_WITH_AES_256_GCM_SHA384	0	0
1	1		
0035	TLS_RSA_WITH_AES_256_CBC_SHA	1	0
0	0		

Figure 89. Statistics data (formatted output)

-----									
SDSF OUTPUT DISPLAY MONDSERV JOB68349 DSID 111 LINE CHARS									
COMMAND INPUT ==>									
DFHCICS	P362	OTRANNUM	0000045C						45
DFHCICS	C363	OTRAN	C3E6E7D5						CWXN
DFHCICS	C364	OUSERID	C7C2F1F2	F2F04040					GB1220
...									
DFHCICS	C366	OTCPSVCE	E6C5C2E3	D3E24040					WEBTLS
DFHCICS	A367	OPORTNUM	0000271F						10015
DFHCICS	C372	OCLIPADR	F94BF2F0	40404040	40404040	40404040	40404040	40404040	9.20.5.0
		+X0014	40404040	40404040	40404040	40404040	40404040	40404040	
DFHCICS	A369	OCLIPORT	0000FA66						641020
DFHCICS	A370	OTRNFLAG	80008040	09800000					
DFHCICS	C371	OFCTYNME	5C5CE2E3	C55C5C40					**STE**
DFHWEBB	C380	WBURIMNM	00000039	F14BF240					DFH\$URI4
...									
DFHWEBB	C385	WBPROGNM	00000039	F14BF240					DFH\$WBHC
...									
DFHCICS	P376	PHTRANNO	0000000C						0
...									
DFH\$OCK	A458	SOFLAG	40000000						
DFH\$OCK	A457	SOTLSLVL	E3D3E2E5	F14BF240					TLSV1.2
DFH\$OCK	A320	SOCIPHER	00000039	F14BF240					57
DFHTASK	C430	CECMCHTP	F3F9F0F6						3906

Figure 90. Monitoring data (formatted output)

## Enabling TLS 1.3 in CICS

---

TLS 1.3 differs from earlier versions of the protocol. To migrate to TLS 1.3 from an earlier version, you need to update your resource definitions to use the cipher suite specification file before enabling TLS 1.3.

### Before you begin

To enable TLS 1.3, you must ensure:

- You are at z/OS 2.4 or later.
- ICSF must be active.
- If you use RSA certificates, the RSA keys held in the PKDS must be secured using the ECC master key.
- You are at CICS TS 6.1 or later.

**Important:** If you're at a CICS release earlier than 6.1, it is important to avoid enabling TLS 1.3 until you complete your upgrade to CICS TS 6.1. You can then focus on enabling TLS 1.3 in isolation from any other work.

**Recommended:** The increased security provided by TLS 1.3 can have a higher CPU cost than earlier versions of TLS. To avoid performance problems, it's recommended that you fulfill the following requirements before enabling TLS 1.3 in production:

- **6.2** Enable sysplex caching for TLS 1.3. See [SSLCACHE system initialization parameter](#).
- Use RSA certificates.

### About this task

TLS 1.3 differs from earlier versions of the protocol as follows. For more information, see [RFC 8446 The Transport Layer Security \(TLS\) Protocol Version 1.3](#).

- No ciphers exist in common between TLS 1.3 and earlier versions of the TLS protocol.
- The ciphers for TLS 1.3 are 4-digit ciphers that can be defined in CICS only by using the TLS cipher suite specification files, which are XML files.
- Enabling TLS 1.3 might affect performance.
- There is an increase in the number of CWXN transactions that execute when you use TLS 1.3 compared to when you use TLS 1.2.

### Procedure

#### Prepare the TLS cipher suite specification file

1. Copy the sample default cipher suite specification file (`defaultciphers.xml`) from `usshome/security/ciphers` to `ussconfig/security/ciphers`.

where

**usshome**

Is the directory specified on the **USSHOME** SIT parameter.

**ussconfig**

Is the directory specified on the **USSCONFIG** SIT parameter. It must be different from the value on **USSHOME**.

You need to create the folders in `ussconfig` if there are none.

This `defaultciphers.xml` file is used as the default cipher suite specification file for new resource definitions that require the **CIPHERS** options. In CICS TS 5.6 and earlier, it defaulted to two-digit ciphers.

2. In `ussconfig/security/ciphers`, customize the `defaultciphers.xml` file to ensure that the ciphers in use conform to your compliance rules.

**Note:** The `defaultciphers.xml` file is used in this example to show a simple route. You can define multiple cipher suite specification files for different connections.

### Prepare resource definitions for TLS 1.3

3. Specify at least one of the following new TLS 1.3 ciphers in your cipher suite specification file, in this case, `ussconfig/security/ciphers/defaultciphers.xml`.

```
<cipher number="1301">
  <!-- TLS_AES_128_GCM_SHA256 -->
</cipher>
<cipher number="1302">
  <!-- TLS_AES_256_GCM_SHA384 -->
</cipher>
<cipher number="1303">
  <!-- TLS_CHACHA20_POLY1305_SHA256 -->
</cipher>
```

If you have multiple cipher suite specification files, ensure that they all have at least one of the TLS 1.3 ciphers, in addition to your existing TLS 1.2 ciphers.

4. Ensure the CIPHERS option in all IPCONN, TCIPSERVICE, and URIMAP resource definitions references your cipher suite specification file, in this case `defaultciphers.xml`, regardless of whether TLS 1.3 is used for the connection.

Because the TLS 1.3 ciphers can be defined only in the cipher suite specification file, you must update any definitions that use ciphers to use an cipher suite specification file on the CIPHERS option.

If you're using CEDA to alter definitions, you can blank out the ciphers in a resource definition. CEDA replaces the existing ciphers with `defaultciphers.xml`.

If you use applications with the **WEB OPEN** commands, you might still have some ciphers defined and see a message DFHWB0767 that indicates the CIPHERS and NUMCIPHERS options are deprecated on the command. The CIPHERS or NUMCIPHERS option on **WEB OPEN** is ignored for TLS 1.3. The **WEB OPEN** connection obtains the cipher either from the URIMAP defined on the URIMAP option of the **WEB OPEN** command, or if that option is not defined, from `defaultciphers.xml`. Ensure that the cipher suite specification cipher file in use contains ciphers appropriate for this connection.

Any existing URIMAP resources defined with `CIPHERS()` will use ciphers from the `defaultciphers.xml` file as soon as TLS 1.3 is enabled.

If you need specific ciphers for a specific connection, ensure you have the corresponding cipher suite specification file in the ciphers subdirectory and it is referenced correctly in the CIPHERS option on the matching IPCONN, TCIPSERVICE, and URIMAP resource definition.

### Update your certificates

5. TLS 1.3 requires certificates to have a minimum key size of 2048.

Check that any certificates in your key rings are at least `SIZE(2048)`. If they are smaller, they need to be reissued.

You can use the `RACDCERT` command to list key rings and certificates.

### Enable TLS 1.3

6. Set the SIT parameter `MAXTLSLEVEL=TLS13`.

The SIT parameter `MAXTLSLEVEL` defaults to `TLS12`. This value enables an upgrade to CICS TS 6.1 or later to take place without affecting any existing TLS connections or programs that open TLS connections.

7. Check that all of your connections are established. The connection uses the highest level of TLS supported by both sides. If any connection does not use TLS 1.3, you need to reconfigure or upgrade that client or server.

You can see which clients or servers are not using TLS 1.3 by using diagnostics. See [Changing TLS protocol level or ciphers safely](#).

### Disable earlier versions of TLS

8. When you are sure that all connections are using TLS 1.3, disable earlier versions of TLS by setting **MINTLSLEVEL=TLS13**.
9. After you complete setting **MINTLSLEVEL=TLS13** in all of your regions, remove non-TLS 1.3 ciphers from all the cipher suite specification files.

## Security for TCP/IP clients

---

This page discusses how you can secure your applications when CICS participates in a client-server configuration, using TCP/IP communication protocols.

### Authenticating ECI users

For the ECI protocol you can use basic authentication to authenticate the user.

Specify **ATTACHSEC (VERIFY)** in the **TCPIPSERVICE** definition for the ECI client. Specify **ATTACHSEC (LOCAL)** if you do not want to authenticate the user.

### SSL encryption

The SSL protocol operates between the application layer and the TCP/IP layer. This allows it to encrypt the data stream itself, which can then be transmitted securely, using any of the application layer protocols.

Many different algorithms can be used for encrypting data, and for computing the message authentication code. Some algorithms provide high levels of security but require a large amount of computation for encryption and decryption. Other algorithms are less secure but provide rapid encryption and decryption. The length of the key that is used for encryption affects the level of security; the longer the key, the more secure the data. SSL defines cipher suites to specify cryptographic algorithms that are used during an SSL connection.

SSL uses two encryption techniques:

- Public key cryptography standard (PKCS), which encrypts and decrypts certificates during the SSL handshake. Encryption keys are created in pairs, a public key and its associated private key. Data encrypted with a given public key can be decrypted only with the associated private key; this means that data is readable by only the intended recipient. Data encrypted with a given private key can be decrypted only with the associated public key; this means that authentication data is assured to originate from the owner of the private key.
- A mutually agreed symmetric encryption technique, such as DES (data encryption standard), or triple DES, is used in the data transfer following the handshake.

PKCS, as used by SSL, works briefly as follows:

1. When a certificate is created, an algorithm based on two random numbers is used to create a private key and public key for the certificate owner. The private and public keys which result are related to each other such that:

- **It is not feasible to deduce the value of the private key from the public key, nor the public key from the private key**

The private key is stored securely, and is not made known to anyone but its owner. The public key can be made freely available to any user, with no risk of compromising the security of the private key.

- **Information encrypted using the public key can be decrypted only with the private key**

Information can be encrypted by any user, and sent securely to the holder of the private key. A third party cannot use the public key to read the information.

- **Information encrypted using the private key can be decrypted only with the public key**

Only the holder of the private key can encrypt information that can be decrypted with the public key. A third party cannot pose as the sender of the information.



## Configuring CICS to use SSL

---

CICS can use the Secure Sockets Layer (SSL) or the Transport Layer Security (TLS) security protocols to support secure TCP/IP connections. To authenticate servers to clients, create certificates and key rings in RACF and ensure that the CICS region and resources are correctly configured to support security.

### Before you begin

Before you begin to configure CICS, decide which type of certificates to use in SSL handshakes.

### About this task

You can use RACF to create certificates, but you must configure your clients to ensure that they can recognize the RACF server certificate. If you cannot configure your clients in this way, for example when clients are external to your organization, use a certificate signed by an external certificate authority.

### Procedure

1. Set the correct authorizations in RACF to create a key ring, create a signing certificate (certificate authority certificate), and to add certificates to the key ring.
2. Optional: If you decide to use a certificate from a certificate authority, create a certificate request using RACF and send it to the certificate authority.

You might have to wait a number of days to receive a signing certificate from the certificate authority. If your chosen certificate authority does not have its certificate built in to RACF, you might have to import it.

3. Create a key ring.

You must create a key ring in the RACF database. The key ring contains:

- Your public and private keys
- Your server certificates
- Signing certificates for the server certificates
- If the client certificate is not associated with a valid RACF userid, the signing certificates for any client certificates owned by clients with which you expect CICS to communicate using client authentication should be added to the keyring.

4. Create the certificates and add them to the key ring.

5. Ensure that the CICS region has access to the z/OS system SSL library SIEALNKE.

You can use STEPLIB or JOBLIB statements, or use the system link library.

6. Define the CICS system initialization parameters that are related to security.

In particular, specify the name of the key ring that you created in the **KEYRING** system initialization parameter.

7. Define TCPIP SERVICE resources.

You can also specify the level of security for an SSL connection by using the **CIPHERS** attribute. You can specify this attribute either with a string that is interpreted as a list of cipher suite codes or with the name of a cipher suite specification file.

### Example

CICS supplies a sample REXX program, DFH\$RING, that contains all of the RACF commands to create a key ring, create a signing certificate, create additional certificates, and add them to the key ring. DFH\$RING contains sample values which are suitable for building a test key ring only. You must edit all the values if you want to create a key ring that is suitable for a production environment.

## Setting up profiles in RACF

To build a RACF key ring in the RACF database that is suitable for use in a CICS region, you must grant access to the appropriate profiles in the FACILITY class.

### About this task

You must grant this access only to users who administer CICS systems and not to general CICS users. The following profiles are available:

#### CONTROL

- IRR.DIGTCERT.GENCERT (to allow certificates to be signed by a CERTAUTH certificate)
- IRR.DIGTCERT.ADD on first execution (to allow a CERTAUTH certificate to be generated)
- IRR.DIGTCERT.CONNECT to connect CERTAUTH certificates for other users

#### UPDATE

- IRR.DIGTCERT.CONNECT (to connect CERTAUTH certificates to your keyring).
- IRR.DIGTCERT.\* (to manage certificates for other users).

#### READ

IRR.DIGTCERT.\* (to manage certificates for your own user ID).

IRR.DIGTCERT.\* contains the wildcard asterisk, and is intended as a generic profile. To allow generic profiles to be created in the FACILITY class:

### Procedure

1. Issue the command **SETROPTS GENERIC(FACILITY)**.
2. Issue the following command:

```
RDEFINE FACILITY(IRR.DIGTCERT.*)
RDEFINE FACILITY(IRR.DIGTCERT.ADD)
RDEFINE FACILITY(IRR.DIGTCERT.CONNECT)
RDEFINE FACILITY(IRR.DIGTCERT.GENCERT)
```

3. Depending upon whether the FACILITY class is RACLISTed or not, issue one of the following commands:

```
SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS GENERIC(FACILITY) REFRESH
```

4. To permit a user ID or group *ringuser* to use the commands contained in DFH\$RING issue the following commands:

```
PERMIT IRR.DIGTCERT.*      CLASS(FACILITY) ID(ringuser) ACCESS(READ)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(ringuser) ACCESS(UPDATE) (for self)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(ringuser) ACCESS(CONTROL) (for another user)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(ringuser) ACCESS(CONTROL)
```

DFH\$RING is the sample that is provided with CICS to help you set up a suitable key ring.

5. You must give the first user of DFH\$RING, *certauser*, authority to create a certificate authority certificate:

```
PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(certauser) ACCESS(CONTROL)
```

This certificate is then used to sign all the other certificates created by DFH\$RING.

### Results

If you have READ access to the IRR.DIGTCERT.ADD profile in the FACILITY class, you can add certificate information for your own user ID. If you have UPDATE access to the IRR.DIGTCERT.ADD profile in the FACILITY class, you can add certificate information for other user IDs. If you have RACF SPECIAL authority, you can execute RACDCERT ADD for any user ID.

## Requesting a certificate from a certificate authority

You can use RACF to request a signing certificate (certificate authority certificate) from a certificate authority such as Verisign. Use an external certificate to authenticate your server to clients that cannot recognize RACF certificates.

### Before you begin

You must have authorization to use the **RACDCERT** command. This command installs and maintains digital certificates, key rings, and digital certificate mappings in RACF.

### About this task

RACF supplies certificates for various certificate authorities, so you do not have to define them yourself. These certificates are listed in [Supplied digital certificates in z/OS Security Server RACF Security Administrator's Guide](#).

### Procedure

1. Create a self-signed certificate in RACF as a placeholder:

```
RACDCERT ID(foruser) GENCERT,  
  SUBJECTSDN(CN('username')  
    T ('username's certificate')  
    OU('department')  
    O ('organization')  
    L ('city')  
    SP('state')  
    C ('country'))  
  NOTBEFORE(DATE(start) TIME(00:00:00))  
  NOTAFTER (DATE(finish) TIME(23:59:59))  
  WITHLABEL(self-signed-certlabel)  
  SIZE (1024)
```

2. Generate a certificate request, based on the placeholder certificate, to send to your external certificate authority. Use the **RACDCERT GENREQ** command:

```
RACDCERT ID(cics-region-userid) GENREQ(LABEL('label'))  
  DSN('request.dataset')
```

where *label* is the placeholder self-signed certificate.

RACF saves the certificate request in the data set specified in the **DSN** parameter.

3. Send the certificate request to the certificate authority, using a method that the certificate authority accepts.
4. When you receive the certificate, save it in a new data set.
5. Optional: If you are using a certificate authority that is not one of the default certificate authorities, for which certificates are already stored in the key database, you must import the certificate authority's certificate into your RACF database.
6. Replace the self-signed certificate with your new CA-signed certificate:

```
RACDCERT ID(cics-region-userid) ADD('response.dataset') TRUST
```

### What to do next

Create the key ring in the RACF database and add your CA-signed certificate.

## Building a key ring manually

In CICS, the required server certificate and related information about certificate authorities are held in a *key ring* in the RACF database. The key ring contains your system's private and public key pair, together with your server certificate and the certificates for all the certificate authorities that might have signed the certificates you receive from your clients.

## Before you begin

Before you can use SSL with CICS, you must create a key ring that contains a private and public key pair and a server certificate.

To create a key ring you must have UPDATE authority to the IRR.DIGTCERT.ADDRING resource in the FACILITY class.

If you want to share certificates in a key ring between CICS regions, make sure either of the following conditions is met:

- The CICS regions share the same user ID that owns the key ring.
- If the region user ID does not own the key ring, grant that region user ID authority to access the key ring.

## About this task

The **RACDCERT** command installs and maintains public key infrastructure (PKI) private keys and certificates in RACF. You can either manually issue the **RACDCERT** command to create a new key ring or you can use the DFH\$RING sample program, see [Building a key ring with certificates using DFH\\$RING](#).

To create a key ring manually, follow these steps:

## Procedure

Issue the following **RACDCERT** command:

```
RACDCERT ID(cics-region-userid) ADDRING(ringname)
```

The key ring must be associated with the CICS region user ID.

## Results

RACF creates the key ring in the RACF database. If there is a key ring of the same name already in the RACF database, it is replaced with the new key ring.

## What to do next

Create a signing certificate (certificate authority certificate) and add it to the key ring.

## Building a key ring with certificates using DFH\$RING

DFH\$RING is a sample REXX program that builds a key ring, creates a signing certificate (certificate authority certificate), creates additional certificates, and adds the certificates to the key ring.

## Before you begin

You must have the required authorization to run the RACF commands:

- CONTROL authority to IRR.DIGTCERT.GENCERT
- UPDATE authority to IRR.DIGTCERT.CONNECT
- READ authority to IRR.DIGTCERT.\* in the FACILITY class
- On first execution, CONTROL authority to IRR.DIGTCERT.ADD
- If the FORUSER parameter is used to specify a different user ID from the issuer, UPDATE authority to IRR.DIGTCERT.\* is required

If you uncomment the sample EXPORT commands from the EXEC, then you also need the following authorities to IRR.DIGTCERT.EXPORT:

- CONTROL to export a CA certificate
- UPDATE to export a certificate owned by a different user

- READ to export your own certificate

## About this task

DFH\$RING is in the CICSTSnn.CICS.SDFHSAMP library, where CICSTSnn is your version of CICS. As an example, for CICS Transaction Server for z/OS, Version 6 Release 2, this library is CICSTS62.CICS.SDFHSAMP.

## Procedure

Edit the values in DFH\$RING to create a suitable key ring and certificates:

1. Update the variables *organization*, *department*, *city*, *state*, and *country*. These variables are used in SUBJECTDN for the certificate authority and user certificates that are generated.  
The values for *firstname*, *lastname*, and *hostname* are set from user input when you run DFH\$RING. This means that they do not need to be updated per the previously listed variables. The *firstname* and *lastname* values are concatenated together to form the name of the key ring. Enter the hostname of your web server for the *hostname* variable.
2. If you have a signing certificate (certificate authority certificate), enter the label in the *certifier* variable.
3. If you do not have a signing certificate, replace the variables for the **RACDCERT CERTAUTH GENCERT** command with suitable values and RACF can create it for you:

```
"RACDCERT CERTAUTH GENCERT",
" SUBJECTSDN(CN('CICS Sample Certification Authority' ) ",
" OU('department' ) ",
" O ('organization' ) ",
" L ('city' ) ",
" SP('state' ) ",
" C ('country' ) ",
" NOTBEFORE (DATE("start") TIME(00:00:00) ) ",
" NOTAFTER (DATE("finish") TIME(23:59:59) ) ",
" WITHLABEL ("certifier" ) ",
" SIZE (2048 ) "
```

These values define appropriate fields in the distinguished names of the generated certificates. The country code for the *country* variable must be an ISO 3166-1 code. For a list of valid codes, see [International Organization for Standardization Country Codes - ISO 3166](#). *start* and *finish* determine the validity of the certificate. *certifier* is the label of the self-signed certificate authority certificate that is used to sign the other certificates.

The **SIZE** parameter specifies the size, in bits, of the private key that is associated with the certificate. The larger the size, the more secure the key. A minimum of 2048 is required for when TLS 1.3 is used, otherwise 2048 is a recommended value.

DFH\$RING creates the signing certificate only if it does not already exist.

4. Edit the variables for the **RACDCERT GENCERT RACF** command to create the certificate to add to your key ring.
5. Edit the labels for the **RACDCERT CONNECT RACF** commands to match your certificates. Ensure that the signing certificate is also added to the key ring, because it signs all the other certificates.
6. Uncomment the **RACDCERT EXPORT RACF** commands if you need to export the certificate authority or user certificates. You might want to edit the DSN parameter on the commands to control which dataset is used for the export. You need to export the certificate authority certificate and make it available to any clients that need to connect securely to your CICS region.
7. Run DFH\$RING to create the key ring and certificates as follows:

```
EXEC 'CICSTSnn.CICS.SDFHSAMP(DFH$RING)' 'firstname lastname hostname [ FORUSER(userid) ] '
```

where CICSTSnn is the CICS release (for example, CICSTS62) and *userid* is the CICS region user ID.

## Results

The DFH\$RING program creates a key ring with name *Firstname.Lastname*, which is owned by the *userid* user ID. Any existing key ring with that name is replaced. If you omit the **FORUSER** parameter, the key ring is owned by the user ID that you used to run the program. DFH\$RING creates a signing certificate if required and adds it to the key ring, followed by the other certificates.

## Example

If you run DFH\$RING with the default values, DFH\$RING creates a certificate with the following label:

### Lastname-Default-Certificate

This certificate is marked as the default certificate for the key ring and is the one that is used for all TCPIP SERVICE, IPCONN, and URIMAP resources that do not specify a CERTIFICATE attribute. This certificate also contains a common name of *hostname*.

## What to do next

You can create and add further certificates to the key ring.

## Creating new RACF certificates

Use the **RACDCERT** command to create and add new certificates to a key ring.

### About this task

The certificates in the key ring must be associated with the CICS region user ID. The region user ID that will use the key ring must either own the key ring or have the authority to use the key ring if it is owned by a different region user ID.

**Note:** Multiple certificates with the same Distinguished Name on the same **KEYRING** are not supported.

### Procedure

1. Create a certificate, specifying the CICS region user ID. Enter the **RACDCERT GENCERT** command as follows:  
Provide values for the variables. The country code for the *country* variable must be an ISO 3166-1 code. For a list of valid codes, see [International Organization for Standardization Country Codes - ISO 3166](#). The value of *certifier* is the label of the signing certificate in the key ring.

```
RACDCERT ID(foruser) GENCERT
  SUBJECTSDN(CN('username')
    T ('username's certificate')
    OU('department')
    O ('organization')
    L ('city')
    SP('state')
    C ('country'))
  NOTBEFORE (DATE(start) TIME(00:00:00))
  NOTAFTER  (DATE(finish) TIME(23:59:59))
  SIGNWITH (CERTAUTH LABEL('certifier'))
  WITHLABEL ('certlabel')
  SIZE      (1024)
```

2. Add the certificate to the key ring using the RACDCERT CONNECT command.
  - a) If you want to share the certificate across multiple CICS regions, add it to the key ring specified in the **KEYRING** system initialization parameter for that CICS region and specify USAGE(PERSONAL). Any CICS region that has the same region user ID and is using the same key ring can access the certificate.

```
RACDCERT ID(foruser) CONNECT( RING(ringname) LABEL('label') USAGE('PERSONAL'))
```

- b) If you want to add a certificate to the key ring as the default certificate, add it to the key ring specified in the **KEYRING** system initialization parameter for that CICS region and specify DEFAULT.

```
RACDCERT ID(foruser) CONNECT( RING(ringname) LABEL('label') DEFAULT)
```

When a client or server requests a certificate from CICS, the default certificate is used unless you have specified otherwise. For inbound HTTP requests, specify the certificate in the TCIPSERVICE resource.

3. After running any of the RACDCERT commands that update certificates or key rings, if the DIGTCERT and DIGTRING classes are RACLISTed, you must issue the following command:

```
SETROPTS RACLIST(DIGTCERT DIGTRING) REFRESH
```

4. After you make any updates or additions to the certificates in the key ring, issue the **PERFORM SSL REBUILD** command for the CICS region.

The command rebuilds the SSL environment for the CICS region and refreshes the cache of certificates with the new information from the key ring.

## Associating a RACF user ID with a certificate

The client certificate can only be used to determine the user ID for the CICS transaction if the certificate is associated with a RACF user ID.

You can associate a certificate with a RACF user ID in two ways:

- Users can register their certificates online through their web browser program. You enable clients to register their certificates themselves by specifying AUTHENTICATE(AUTOREGISTER) on the TCIPSERVICE definition. Users connecting to CICS through such a TCIPSERVICE must have a client certificate. If that certificate is already registered to a user ID, then that user ID is used; if not, the client is prompted for a user ID and password with HTTP basic authentication. If the client then enters a valid user ID and password, that user ID is registered to the certificate, and the client will not be prompted for a password again. The rules are summarized in [How it works: Identification in CICS](#).

Once a certificate has been registered in this way, it can be used for all inbound TCP/IP connections.

- You can use the RACDCERT command. If you do not want to allow your clients to register their own certificates, you must register them with the RACDCERT command. Before executing RACDCERT, you must download the certificate that you want to process into an MVS sequential file with RECFM=VB that is accessible from TSO. The syntax of RACDCERT is:

```
RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
```

where *datasetname* is the name of the data set containing the client certificate, and *userid* is the user ID that is to be associated with the certificate. If the optional ID(userid) parameter is omitted, the certificate is associated with the user issuing the RACDCERT command.

You can add certificate information for your own user ID if you have READ access to the IRR.DIGTCERT.ADD profile in the FACILITY class. You can add certificate information for other user IDs if you have UPDATE access to the IRR.DIGTCERT.ADD profile in the FACILITY class or if you have RACF SPECIAL authority.

For further information on the RACDCERT command, including the format of data allowed in the downloaded certificate data set, see [z/OS Security Server RACF Command Language Reference](#).

## Using an existing certificate that is not owned by the CICS region user ID

You can share a single certificate between CICS systems by using the appropriate RACF facilities.

### About this task

For any CICS resource that has the CERTIFICATE attribute and for Web Services Security, by default the certificate that is used must be owned by the CICS region user ID. If CICS needs to use a certificate that it does not own, for example a single certificate that is shared by multiple CICS systems where each system has a different region user ID, you can use the RACF Facility Class RDATA LIB to allow multiple CICS systems to share a single certificate.

## Procedure

1. Connect the certificate to its key ring with the PERSONAL usage option.
2. If the certificate is a USER certificate, grant to the CICS region user ID that you want to use the certificate UPDATE authority for the *ring\_owner.ring\_name.LST* resource in the RDATA LIB class.
3. Activate the RDATA LIB class by using the **RACLIST** command.

## Results

CICS can use the certificate that is owned by the other user ID. For more information, see [z/OS Security Server RACF Callable Services](#).

## Configuring a RACF site certificate for use with CICS TS

If you want to enable SSL in your CICS Transaction Server for z/OS regions, but you do not want to define separate SSL Certificates for each region, you can use a *site certificate*.

## Procedure

Build a keyring by following the instructions in “Building a key ring manually” on page 167 or “Building a key ring with certificates using DFH\$RING” on page 168.

The key ring must be completely configured, that is, it must contain not only the certificate pointed to by your TCPIP SERVICE RDO definition, but also every certificate that was used to sign that certificate. These signing certificates must be in the KEYRING with USAGE=CERTAUTH.

The following figure shows an example of a completely configured key ring as listed by the RACF command: `RACDCERT ID(ring_owner) LISTRING(ring_name)`

Ring: *ring\_name*

Certificate Label Name	Cert Owner	USAGE	DEFAULT
Verisign Class 1 Primary CA	CERTAUTH	CERTAUTH	NO
IBM World Registry CA	CERTAUTH	CERTAUTH	NO
CICS-Sample-Certification	CERTAUTH	CERTAUTH	NO
Verisign Class 2 Primary CA	CERTAUTH	CERTAUTH	NO
SITECERT	SITE	PERSONAL	YES

The certificate that you want to use as a site certificate (SITECERT) must be owned by SITE and have a usage of PERSONAL. This certificate is the one used by any TCPIP SERVICE definition that needs SSL encryption.

The key ring must be owned by the CICS region user ID. If multiple CICS regions use the same region user ID, they can share the same key ring. If they run under different region user IDs, then you must build separate key rings. However, you can use the same site certificate in each ring.

The site certificate must have a private key or the TCPIP SERVICE will either fail to install or will fail when an attempt is made to use it.

The CICS region user ID must have CONTROL access or greater to the profile IRR.DIGTCERT.GENCERT in the FACILITY class.

For more information, see the sections about RACF Callable Services Authorization and RACF Callable Services Usage Notes in [z/OS Security Server RACF Callable Services](#).

## Making a certificate untrusted

If a certificate has been registered in the RACF database, but you do not want it to be used by clients, you can mark it as UNTRUSTED using the RACDCERT command.

## Procedure

1. Enter the command `RACDCERT ID(userid) LIST` to find the label associated with the certificate.



2. Enter the command `RACDCERT ID (userid) ALTER(LABEL(label)) NOTRUST` to mark the certificate as untrusted.
  3. If you amended the certificate while a running CICS region was using a key ring containing the certificate, issue the `PERFORM SSL REBUILD` command for the CICS region.  
The command rebuilds the SSL environment for the CICS region and refreshes the cache of certificates with the new information from the key ring.
- Note:** The **PERFORM SSL REBUILD** command does not apply to SSL/TLS environments where CICS is using a TCPIP SERVICE that is defined with `SSL(ATTLSAWARE)`, mandating AT-TLS secured client connections. If you want to refresh such SSL environments and cache, follow the instructions in [Implementation options for TLS](#).

## Results

Clients are prevented from establishing CLIENTAUTH connections with this certificate.

## Setting the System SSL environment for CICS

CICS managed TLS connections are processed using the z/OS System SSL services. The System SSL environment is controlled by using either environment variables or the GSK API commands. CICS uses the API commands directly, which means that values set by CICS cannot be overridden by externally set environment variables.

You update System SSL environment variables in different ways, depending on whether they are set by CICS or not.

For a complete list of environment variables used by System SSL, see [Environment variables in z/OS documentation](#).

## Updating System SSL variables that are set by CICS

The following table lists the System SSL variables that are set by CICS and the value they are set to. Some values are fixed and some can be modified by changing the value specified on the associated CICS parameter.

Table 20. System SSL variables set by CICS	
Variable name	Value
GSK_KEYRING_FILE	The <b>KEYRING</b> SIT parameter.
GSK_KEYRING_LABEL	The CERTIFICATE label specified on the associated command or resource for a particular request.
GSK_V3_CIPHER_SPECS	The list of 2-digit ciphers if a cipher file is not used.
GSK_V3_CIPHER_SPECS_EXPANDED	The list of 4-digit ciphers from the associated cipher file.
GSK_V2_SESSION_TIMEOUT	60
GSK_V3_SESSION_TIMEOUT	The <b>SSLDELAY</b> SIT parameter.
GSK_PROTOCOL_SSLV2	OFF
GSK_PROTOCOL_SSLV3	OFF
GSK_PROTOCOL_TLSV1	OFF
GSK_PROTOCOL_TLSV1_1	ON or OFF based on the <b>MINTLSLEVEL</b> and <b>MAXTLSLEVEL</b> SIT parameters.

Table 20. System SSL variables set by CICS (continued)

Variable name	Value
GSK_PROTOCOL_TLSV1_2	ON or OFF based on the <a href="#">MINTLSLEVEL</a> and <a href="#">MAXTLSLEVEL</a> SIT parameters.
GSK_PROTOCOL_TLSV1_3	ON or OFF based on the <a href="#">MINTLSLEVEL</a> and <a href="#">MAXTLSLEVEL</a> SIT parameters.
GSK_CLIENT_EPHEMERAL_DH_GROUP_SIZE	<ul style="list-style-type: none"> <li>6.2 2048 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <a href="#">system default value</a> is used instead.</li> <li>6.1 2048 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <a href="#">system default value</a> is used.</li> </ul>
GSK_SERVER_EPHEMERAL_DH_GROUP_SIZE	<ul style="list-style-type: none"> <li>6.2 2048 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <a href="#">system default value</a> is used instead.</li> <li>6.1 2048 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <a href="#">system default value</a> is used.</li> </ul>
GSK_PEER_DH_MIN_KEY_SIZE	<ul style="list-style-type: none"> <li>6.2 2048 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <a href="#">system default value</a> is used instead.</li> <li>6.1 2048 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <a href="#">system default value</a> is used.</li> </ul>
GSK_PEER_DSA_MIN_KEY_SIZE	<ul style="list-style-type: none"> <li>6.2 2048 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <a href="#">system default value</a> is used instead.</li> <li>6.1 2048 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <a href="#">system default value</a> is used.</li> </ul>
GSK_PEER_RSA_MIN_KEY_SIZE	<ul style="list-style-type: none"> <li>6.2 2048 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <a href="#">system default value</a> is used instead.</li> <li>6.1 2048 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <a href="#">system default value</a> is used.</li> </ul>

Table 20. System SSL variables set by CICS (continued)

Variable name	Value
GSK_PEER_ECC_MIN_KEY_SIZE	<ul style="list-style-type: none"> <li>6.2 256 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <u>system default value</u> is used instead.</li> <li>6.1 256 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <u>system default value</u> is used.</li> </ul>
GSK_CLIENT_ECURVE_LIST	<ul style="list-style-type: none"> <li>6.2 00250024002300300029 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024 and TLS 1.3 is not being used, the <u>system default value</u> is used instead.</li> <li>6.1 00250024002300300029 either when feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024) or when TLS 1.3 is used. Otherwise, the <u>system default value</u> is used.</li> </ul>
GSK_SERVER_ALLOWED_KEX_CURVES	<ul style="list-style-type: none"> <li>6.2 00250024002300300029 by default. If feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 1024, the <u>system default value</u> is used instead.</li> <li>6.1 00250024002300300029 only if feature toggle <code>com.ibm.cics.tls.minimumkeystrength</code> is set to 2048 (the default is 1024). Otherwise, the <u>system default value</u> is used.</li> </ul>
GSK_CLIENT_TLS_KEY_SHARES	00250024002300300029 if TLS 1.3 is used.
GSK_SERVER_TLS_KEY_SHARES	00250024002300300029 if TLS 1.3 is used.
6.2 GSK_CERT_DIAG_INFO	GSK_CERT_DIAG_INFO_SUCCESS_OR_FAILURE
GSK_LDAP_SERVER	The LDAP server URL that is obtained from the profile specified on the <b>CRLPROFILE</b> SIT parameter. It is set only if CRL processing is required and <b>CRLPROFILE</b> specifies a valid profile in the LDAPBIND resource class.
GSK_LDAP_USER	The LDAP user distinguished name that is obtained from the profile specified on the <b>CRLPROFILE</b> SIT parameter. It is set only if CRL processing is required and <b>CRLPROFILE</b> specifies a valid profile in the LDAPBIND resource class.
GSK_LDAP_USER_PW	The LDAP user password that is obtained from the profile specified on the <b>CRLPROFILE</b> SIT parameter. It is set only if CRL processing is required and <b>CRLPROFILE</b> specifies a valid profile in the LDAPBIND resource class.
6.2 GSK_SESSION_TICKET_CLIENT_MAXCACHED	The <b>MAXSSLTCBS</b> SIT parameter, with a maximum value at 128.

Table 20. System SSL variables set by CICS (continued)

Variable name	Value
6.2 GSK_SYSPLEX_SESSION_TICKET_CACHE	ON if the <b>MAXTLSLEVEL</b> SIT parameter is set to TLS13.
GSK_SYSPLEX_SIDCACHE	ON if the <b>SSLCACHE</b> SIT parameter is set to SYSPLEX.

## Updating System SSL variables that are not set by CICS

If a System SSL variable is not set by CICS, CICS uses the System SSL default value for that variable. You can change the default values that are used by CICS in either of the following ways:

1. Update CELQDOPT in the CEEPRMxx parmlib member to add an ENVAR statement that sets the required variable. This affects all 64-bit applications that use System SSL services, not just CICS applications. For detailed instructions, see [Creating system-level runtime options and keyword defaults with CEEPRMxx in z/OS documentation](#).
2. Add a CEEOPTS DD statement to the CICS JCL. CEEOPTS needs to reference a data set or member that contains an ENVAR statement. This affects System SSL and JVM servers within CICS.

```
ENVAR("GSK_PEER_CERT_MIN_VERSION=3")
```

## System initialization parameters for SSL

Descriptions of system initialization parameters that relate to SSL.

The following system initialization parameters relate to SSL:

### **CRLPROFILE** system initialization parameter

Specifies the name of the profile that authorizes CICS to access certificate revocation lists that are stored in an LDAP server. For more information about certificate revocation lists and setting up this profile, see [“Configuring an LDAP server for CRLs” on page 183](#).

### **KEYRING** system initialization parameter

Specifies the name of a key ring in the RACF database that contains keys and certificates used by CICS. The region user ID that will use the key ring must either own the key ring or have the authority to use the key ring if it is owned by a different region user ID. You can create an initial key ring with the DFH\$RING exec that is supplied in the CICSTSnn.CICS.SDFHSAMP library, where CICSTSnn is your version of CICS. For example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is CICSTS62.CICS.SDFHSAMP

### **MAXSSLTCBS** system initialization parameter

Specifies the maximum number of S8 TCBs that are available to CICS to process secure sockets layer connections and requests to LDAP using the DFHDDAPX XPI interface. This value is a number in the range 0 through 999, and has a default value of 8. The S8 TCBs are created and managed in the SSL pool. An S8 TCB is used by a task only for the duration of the SSL or LDAP processing.

### **MINTLSLEVEL** system initialization parameter

Specifies the minimum TLS protocol that CICS uses for secure TCP/IP connections.

### **SSLCACHE** system initialization parameter

Specifies whether CICS should use a local cache of SSL sessions for the CICS region, or share the cache across multiple CICS regions by using the coupling facility. Caching across a sysplex can only take place when the regions accept SSL connections at the same IP address. The cache contains session IDs that enable CICS to perform abbreviated handshakes with clients that it has previously authenticated. A local cache is replaced when you issue the PERFORM SSL REBUILD command for the CICS region, but a sysplex cache is unaffected.

### **SSLDELAY system initialization parameter**

Specifies the length of time in seconds for which CICS retains session IDs for secure socket connections in a local CICS region. Session IDs are tokens that represent a secure connection between a client and an SSL server. While the session ID is retained by CICS within the SSLDELAY period, CICS can continue to communicate with the client without the significant overhead of an SSL handshake. The value is a number of seconds in the range 0 through 86400. The default value is 600.

## **TCPIPSERVICE attributes for SSL**

Descriptions of the attributes of the TCPIPSERVICE resource that relate to SSL.

### **About this task**

The following attributes of the [TCPIPSERVICE](#) resource relate to SSL:

#### **AUTHENTICATE**

Specifies the authentication and identification scheme to be used for inbound TCP/IP connections for the HTTP protocol. The HTTP protocol supports the following authentication scheme:

##### **NO**

The client is not required to send authentication or identification information.

##### **BASIC**

HTTP basic authentication is used to obtain a user ID and password from the client.

##### **CERTIFICATE**

TLS client certificate authentication is used to authenticate and identify the client.

##### **AUTOREGISTER**

TLS client certificate authentication is used to authenticate the client. If the client sends a valid certificate that is not registered to the security manager, then CICS will register the certificate.

##### **AUTOMATIC**

If the client sends a certificate, TLS client certificate authentication is used to authenticate the client. If the client sends a valid certificate that is not registered to the security manager, then CICS will register the certificate. If the client does not send a certificate, then HTTP Basic authentication is used to obtain a user ID and password from the client.

#### **CERTIFICATE**

Specifies the label of the server certificate used during the TLS handshake. If this attribute is omitted, the default certificate defined in the key ring for the CICS region user ID is used.

#### **CIPHERS**

The CIPHERS attribute is specified by referencing the name of the SSL cipher suite specification file, which is a z/OS UNIX file in the `security/ciphers` subdirectory of the directory that is specified by the **USSCONFIG** system initialization parameter. The default value is `defaultciphers.xml`. For example, if **USSCONFIG** is set to `/var/cicsts/dfhconfig` and **CIPHERS** is set to `defaultciphers.xml`, the fully qualified file name is `/var/cicsts/dfhconfig/security/ciphers/defaultciphers.xml`. For more information, see [Creating a TLS cipher suite specification file](#).

It is recommended that you copy the sample default ciphers file from `ussHOME/security/ciphers/defaultciphers.xml` to `ussconfig/security/ciphers/defaultciphers.xml` and customize to ensure that the ciphers used conform to your compliance rules.

If TLS is enabled, the CIPHERS attribute is based on the defined cipher suites in `ussconfig/security/ciphers/`.

Any unsupported ciphers are removed at run time. A list of the removed ciphers is reported in messages DFHSO0145 and DFHSO0146.

#### **PORTNUMBER**

Specifies the number of the port on which CICS is to listen for incoming client requests. The well known port for SSL services supported by CICS is 443, for HTTP with TLS.

## SSL

Specifies whether the TCP/IP service is to use SSL (TLS) for encryption and authentication:

### NO

SSL is not to be used.

### YES

An SSL session is to be used; CICS will send a server certificate to the client.

### CLIENTAUTH

An SSL session is to be used; CICS will send a server certificate to the client, and the client must send a client certificate to CICS.

### ATTLSAWARE

CICS queries the client connection to determine whether AT-TLS is active. CICS retrieves a client certificate from TCP/IP if one was provided by the partner.

**Note:** If you specify SSL(ATTLSAWARE), you must also specify PROTOCL(HTTP).

## Creating a TLS cipher suite specification file

You can create a TLS cipher suite specification file to specify a list of cipher suites to be used by TLS. If TLS is used for TCP/IP connections, you can specify the name of cipher suite specification file in the **CIPHERS** attribute for resources that define TCP/IP connections.

### Procedure

1. Create a TLS cipher suite specification file either by editing a sample specification file or by creating your own one:
  - To modify a sample TLS cipher suite specification file, copy one of the sample files located in the *usshome/security/ciphers* directory to the *ussconfig/security/ciphers* directory, where
    - usshome**  
is the value of the SIT parameter **USSHOME**.
    - ussconfig**  
is the value of the SIT parameter **USSCONFIG**.
  - Note:** The TLS cipher suite specification file must be in the *ussconfig/security/ciphers* directory.
  - To create your own TLS cipher suite specification file, create an XML file in the *ussconfig/security/ciphers* directory and ensure the file follows these rules:
    - The file name is up to 28 characters in length, including the .xml extension.
    - The file name is case-sensitive. It must be a valid name for a UNIX file and contain only the following characters: A-Z a-z 0-9 # - . @ \_
    - The file must use the EBCDIC 037 encoding.
2. Specify your list of cipher suites in the specification file as indicated in [Customizing encryption negotiations](#).

If you edit the sample file, you can remove unwanted cipher suites that do not meet your security requirements, or that are not supported by your hardware. You can also add cipher suites, but only those cipher suites that are supported by z/OS.
3. For the file to be effective in a TLS connection, ensure that the CICS region has permission to access z/OS UNIX, and that the region has read and execute access to the directory that contains the specification file, and read access to the file itself.

### Results

You have created a cipher suite specification file. A TLS cipher suite file can be used by multiple resources. The first time when a resource that uses a specification file is installed, the file is read from

zFS and parsed. Any errors are flagged during this parse. If the file is valid, the resource is installed and the cipher information is stored in a new control block that is associated with the file. When subsequent resources that use the same cipher file are installed, cached information in the control block is used.

## What to do next

If you want to update the list of cipher suites in a cipher suite specification file, you can edit the file directly, but you must restart CICS for the updated list to take effect. The file is reread for any type of start, whether the **START** system initialization parameter is set to INITIAL, COLD, or AUTO.

To update the list of cipher suites for a resource without restarting CICS, you must use a new specification file:

1. Create a new cipher suite specification file. Ensure that the file name has not been loaded by this CICS system.
2. Update the existing resource definition to refer to the new file. For example, issue a **CREATE TCPIP SERVICE** command with CIPHERS(newciphers.xml) specified.
3. Reinstall the resource definition.

## Customizing encryption negotiations

You can select the cipher suites that are used in the encryption negotiation process for TLS connections to set a minimum level and a maximum level of encryption.

### About this task

The CIPHERS attribute in the resource definitions TCPIP SERVICE, IPCONN, and URIMAP specifies the cipher suites that can be used for each encryption level. The default is to use the cipher suite specification file of defaultciphers.xml for the cipher suites that are used in encryption negotiations. You have the option of customizing the cipher suites to meet your compliance requirements.

**Note:** You must copy allvalidciphers.xml and defaultciphers.xml from USSHOME to USSCONFIG and customize them to your company requirements. This customization includes balancing security and performance. Some cipher algorithms require an IBM Crypto Express card so might cause increased CPU if they are used on servers without this capability.

If you have different security strengths for different connections, create separate cipher files for these connections.

The TLS cipher suite specification (cipher) file is a z/OS UNIX file in the security/ciphers subdirectory of the directory that is specified by the **USSCONFIG** system initialization parameter. For more information, see [Creating a TLS cipher suite specification file](#).

If you need to check which cipher suites are in use, for example, to remove an outdated or unused suite, see [Changing TLS protocol level or ciphers safely](#).

For a description of the cipher file structure and where it is used by CICS and z/OS, see [Implementation options for TLS](#).

### Procedure

1. Create or use an existing cipher file in the /security/ciphers directory of **USSCONFIG**.  
The CIPHERS attribute displays the default value. For CICS to display the default value, the **KEYRING** system initialization parameter must be specified in the CICS region where you are working with the resource definition.
2. Edit the attribute value to specify the name of the cipher file.  
For example, you might specify myciphers.xml if that was the name of the file that is created in step 1.
3. Save the resource definition.

## Related tasks

[“Enabling TLS 1.3 in CICS” on page 162](#)

TLS 1.3 differs from earlier versions of the protocol. To migrate to TLS 1.3 from an earlier version, you need to update your resource definitions to use the cipher suite specification file before enabling TLS 1.3.

## Making your CICS TS system conformant to NIST SP800-131A

To make your system SP800-131A conformant, update various SIT parameters and resource attributes to use suitable cipher suites and certificates.

### About this task

Conformance to the National Institute of Standards and Technology (NIST) SP800-131A security standard strengthens security by requiring the use of stronger cryptographic keys and more robust algorithms.

For more information about the NIST standards, see the [NIST Computer Security Resource Center \(nist.gov\)](http://nist.gov).

**Note:** This configuration is ignored if **MAXTLSLEVEL** is set to TLS13.

### Procedure

To make your system conformant to NIST SP800-131A, complete the following steps:

1. Set the **NISTSP800131A** system initialization parameter to NISTSP800131A=CHECK.
2. Set the **MINTLSLEVEL** and **MAXLSLEVEL** system initialization parameters to TLS12.
3. Set the **KEYRING** system initialization parameter to the name of a key ring that is populated with NIST SP800-131A conformant certificates.
4. Set the **USSCONFIG** system initialization parameter to the name and path of the root directory for CICS Transaction Server configuration files on z/OS UNIX.  
This directory must have a `/security/ciphers/` subdirectory that contains at least one SSL cipher suite specification file. For more information, see [Customizing encryption negotiations](#).
5. Create `defaultciphers.xml` and ensure it contains SP800-131A conformant cipher suites. The ciphers in `fipsciphers.xml` is suitable.
6. Update any TCIPSERVICE, IPCONN, or URIMAP definitions, setting the **CIPHERS** attribute to the name of an SSL cipher suite specification file that contains SP800-131A conformant cipher suites. The sample file `fipsciphers.xml` is suitable.
7. Update any TCIPSERVICE, IPCONN, or URIMAP definitions, setting the **CERTIFICATE** attribute to the name of an SP800-131A conformant certificate label.  
Also, any outbound HTTP application that uses SSL must also use an SP800-131A conformant certificate on any **EXEC CICS WEB OPEN** command. If you use the key ring default certificate with any of these resources definitions or the **WEB OPEN** command, ensure that the key ring default certificate is SP800-131A conformant.
8. If you use the CICSplex SM Web User Interface (WUI) to connect to CICS, set the **TCIPSSLCIPHERS** WUI server initialization parameter to the name of an SSL cipher suite specification file that contains SP800-131A conformant cipher suites. The sample file `fipsciphers.xml` is suitable.
9. If you use the CICSplex SM WUI to connect to CICS, set the **TCIPSSLCERT** WUI server initialization parameter to the name of an SP800-131A conformant certificate label.  
If you use the key ring default certificate, ensure that it is SP800-131A conformant.



## What to do next

Any clients that connect to CICS must be SP800-131A-conformant and must support TLS 1.2. To be conformant, they must be capable of using SP800-131A conformant cipher suites and, if they use certificates, SP800-131A conformant certificates.

Any partner CICS system must use MINTLSLEVEL=TLS12 to talk to a MINTLSLEVEL=TLS12 system and it must be configured to use cipher suites and certificates that are SP800-131A conformant.

**Note:** If you set NISTSP800131A=CHECK, CICS takes the following actions:

- When a JVM server is started, CICS sets the Java properties to make Java NIST SP800-131A conformant.
- If you use SAML and sign outbound messages, CICS issues message DFHXS1300 to warn you to check that the certificates used are conformant.
- If you use WS-Security, CICS issues message DFHXS1301 because CICS support of WS-Security is not conformant with NIST SP800-131A.
- When a TCPIP SERVICE defined with SSL(ATTLSAWARE) is opened, CICS issues message DFHSO0148 to warn you that the AT-TLS policy used to secure this TCPIP SERVICE must be conformant with **NIST SP800-131A**. See [AT-TLS policy configuration in z/OS Communications Server: IP Configuration Guide](#).

## Configuring LDAP for CICS use

You can use LDAP for storing CRLs (certificate revocation lists) or Basic Authentication credentials. When certificate revocation lists or credentials are stored in the LDAP server, you must authorize CICS to access them.

### About this task

Certificate revocation lists and passwords are stored in the LDAP server with an access class of *critical* and can only be accessed by a user who has provided authentication credentials at LDAP bind time. These credentials are a user's distinguished name and an associated password. You can save these details in a specialized profile in the LDAPBIND RACF class. To set up the profile, follow these steps:

### Procedure

1. The password that is used in the profile must be encrypted before it is stored in the RACF database. To encrypt the password, you must store a password encryption key in the KEYSMSTR RACF class by issuing one of the following RACF commands:

- ```
RDEFINE KEYSMSTR LDAP.BINDPW.KEY OWNER(userid)
  SSIGNON(KEYENCRYPTED(keyvalue))
```

Use this command when the password encryption key is stored by the integrated cryptographic service facility (ICSF).

- ```
RDEFINE KEYSMSTR LDAP.BINDPW.KEY OWNER(userid)
  SSIGNON(KEYMASKED(keymask))
```

Use this command when ICSF is not active.

2. Create the profile using the following RACF command:

```
RDEFINE LDAPBIND profile-name
  PROXY(LDAPHOST(ldap-url)
    BINDDN('ldap-distinguished-name')
    BINDPW(password))
  UACC(NONE)
```

where:

#### profile-name

is the name of the RACF profile whose PROXY segment contains the following LDAP bind parameters.

**ldap-url**

is a fully qualified URL of the LDAP server to be accessed; for example, LDAP://EXAMPLE.COM:3389.

**ldap-distinguished-name**

is the distinguished name of an LDAP user authorized to inquire on certificate revocation list attributes from the server; for example, CN=LDAPADMIN.

**password**

is the password that authenticates the LDAP user. The password is case-sensitive.

3. Authorize each CICS region user ID to access appropriate bind credentials in the LDAPBIND class by issuing one or more commands of the following form:

```
PERMIT profile-name CLASS(LDAPBIND)
                        ACCESS(READ)
                        ID(region-userid)
```

4. Specify the profile name in the system initialization parameter **CRLPROFILE** for each applicable CICS region.

**Results**

When you start a CICS region with the profile name specified in the **CRLPROFILE** system initialization parameter, the bind information for the LDAP server is cached in the SSL environment for the CICS region, which is managed by z/OS System SSL. When you issue the PERFORM SSL REBUILD command for the CICS region, the bind information for the LDAP server is refreshed from the external security manager.

**What to do next**

If the **CRLPROFILE** parameter is specified for a CICS region but is invalid, or if the specified profile contains invalid data, or if the LDAP server identified by the profile is unavailable when the CICS region starts, the CICS region disables its own access to the LDAP server. Messages DFHSO0128 and DFHSO0129 report this problem.

To restore access, you must fix the error and restart the CICS region. The PERFORM SSL REBUILD command cannot restore access to the LDAP server if the CICS region has disabled it. The refresh only takes place for an LDAP server that was available to the CICS region at the time when the command was issued.

**Using certificate revocation lists (CRLs)**

You can configure CICS to use certificate revocation lists (CRLs) to check the validity of client certificates being used in SSL negotiations.

**Before you begin**

To use certificate revocation lists, you must install and configure an LDAP server. See [z/OS Cryptographic Services PKI Services Guide and Reference](#). You also need to authorize CICS to access the LDAP server, as described in [Configuring LDAP for CICS use](#).

**About this task**

A certificate revocation list details the revoked certificates from a certificate authority. Certificate authorities keep these lists in CRL repositories that are available on the World Wide Web and can be downloaded and stored in an LDAP server. To populate the LDAP server and update certificate revocation lists, use the CICS-supplied transaction CCRL.

## Configuring an LDAP server for CRLs

To use certificate revocation lists (CRLs), you must have an LDAP server running. You will also need to perform some configuration steps before you download the CRLs.

### Before you begin

If you need to install and configure an LDAP server, see [z/OS Cryptographic Services PKI Services Guide and Reference](#).

### About this task

#### Procedure

1. Ensure that the LDAP server is running. The default started task name is LDAPSRV.
2. In the file system in `etc/ldap`, edit the configuration file `slapd.conf` as follows:
  - a) Create an administrator distinguished name and password, by providing values for `adminDN` and `adminPW`.

The CICS-supplied CCRL transaction requires this information to update the LDAP server with the certificate revocation lists.
  - b) Create a suffix entry for every certificate authority that you want to download CRLs from using CCRL. For each suffix, use the syntax `"O=certificate authority"`.

The suffix is comprised of the Certificate Authority's distinguished name that contains the organization or "O=" keyword, together with any other keywords to the right of this. If the suffix contains any of the special characters `<, +; >\` you must escape them by using **two** backslash characters. If you are using the z/OS LDAP server and the suffix contains any characters that are not in the required 1047 code page, the characters should be escaped by encoding them as the 3-digit octal number of their Unicode representation, preceded by an ampersand.

#### Example

For example you could specify the following suffixes in the file `slapd.conf`:

```
suffix "O=CompanyName"
suffix "O=CompanyName plc"
suffix "O=CompanyName,L=CompanyLocation,ST=CompanyArea,C=CompanyCountry"
suffix "O=CompanyName\\, Inc."
suffix "O=CompanyName\\, Inc.,C=CompanyCountry"
```

### What to do next

When you have configured the LDAP server to include all of your certificate authorities, run the CCRL transaction. For details, see [“Running the CCRL transaction” on page 183](#).

## Running the CCRL transaction

The CICS-supplied transaction CCRL allows you to download and store certificate revocation lists (CRLs) that can be used in the SSL handshake to determine if client certificates are valid.

### Before you begin

You need to configure an LDAP server to specify which certificate authorities you want to use and to create an administrator id and password. See [“Configuring an LDAP server for CRLs” on page 183](#) for detailed instructions.

### About this task

Certificate revocation lists are available from certificate authorities such as Verisign. They are kept in CRL repositories that are available on the World Wide Web and can be downloaded and stored in an LDAP server. To populate the LDAP server and update certificate revocation lists, use the CICS-supplied

transaction CCRL. You can run the CCRL transaction from a terminal or using a **START** command. Use the **START** command to schedule regular updates.

## Procedure

1. Specify the name of the LDAP server in the system initialization parameter **CRLPROFILE**.
2. Run the CCRL transaction
  - from a terminal. See [“Running CCRL from a terminal” on page 184](#).
  - using a command. See [“Running CCRL from a START command” on page 184](#).

### ***Running CCRL from a terminal***

You can run the CICS-supplied transaction CCRL using a terminal to download certificate revocation lists (CRLs).

## Before you begin

Read [“Running the CCRL transaction” on page 183](#) to find out about the prerequisites before running this transaction from a terminal.

## Procedure

1. From a terminal, enter the command `CEOT TRANIDONLY` so that you can enter the list of URLs in mixed case.
2. Enter `CCRL url-list`, where *url-list* is the URL that specifies the location of the certificate revocation list file that you want to download. You can specify more than one URL by leaving a space between each URL in the list.  
For example, you could specify: `CCRL http://crl.verisign.com/ATTCClass1Individual.crl http://crl.verisign.com/ATTCClass2Individual.crl`.
3. You are prompted to enter the administrator distinguished name and password for the LDAP server. This allows CICS to update the LDAP server with the CRLs that it downloads.  
The administrator name and password are specified in the file `slapd.conf`. For more information about configuring this file, see [“Configuring an LDAP server for CRLs” on page 183](#).

## Results

CICS downloads the CRLs from the URLs that you have specified and store them in the LDAP server. You will receive confirmation that all of the lists were downloaded. If CICS experiences a problem, for example the URL is not valid, you will receive an error message.

## What to do next

To set up regular updates, you can use a START command. See [“Running CCRL from a START command” on page 184](#).

### ***Running CCRL from a START command***

You can schedule the CCRL transaction to run at regular intervals using a START command.

## Before you begin

Read [“Running the CCRL transaction” on page 183](#) to find out about the prerequisites before running this transaction from a terminal.

## Procedure

- To use a START command, enter `EXEC CICS START TRANSID(CCRL) FROM (admin://adminDN:adminPW url-list) LENGTH (url-list-length) [INTERVAL(hhmmss)|TIME(hhmmss)]` where *url-list* is a space-delimited list of URLs from where certificate revocation

lists can be downloaded, *url-list-length* is the length of the URL list (including `admin: / /`), and *hhmmss* is the interval or expiration time at which the CCRL transaction is to be scheduled. For example, you could specify:

```
EXEC CICS START TRANSID(CCRL)
FROM('admin://cn=ldapadmin:cics31ldap
http://crl.verisign.com/ATTCClass1Individual.crl
http://crl.verisign.com/ATTCClass2Individual.crl')
LENGTH(124) INTERVAL(960000)
```

This example schedules the CCRL transaction to run in 96 hours.

## Intrusion Detection Services

---

IBM z/OS Communications Server provides Intrusion Detection Services (IDS) provides support for scan and attack detection and reporting and traffic regulation for TCP connections and UDP receive queues. You can use IDS policies to specify event conditions and the actions to take for particular events. CICS provides 3270 IDS detection and protection for any applications that exploit CICS basic mapping support (BMS) interfaces to create and parse their 3270 screens.

The BMS 3270 Intrusion Detection feature allows CICS to detect if a 3270 emulator has invalidly modified a protected field generated by a BMS map. You can opt into this capability with a feature toggle, as described in [Specifying feature toggles](#). When this feature is activated, CICS monitors the 3270 data streams to detect any attempted modifications to protected fields on the screen. CICS can then provide warning messages or prevent the application from processing the data by abending the transaction.

CICS BMS 3270 Intrusion Detection works together with the 3270 Intrusion Detection Service provided by IBM z/OS Communications Server. If configured, IBM z/OS Communications Server handles protection of all 3270 applications. When both services are enabled, BMS-generated 3270 data is handled by CICS, and non-BMS 3270 data is handled by IBM z/OS Communications Server. Enabling both gives you full coverage of all 3270 applications, but makes use of BMS, to maximize performance and to enhance the information returned about any intrusion.

By introducing a user-replaceable module DFHBMSX into the BMS 3270 IDS configuration, you can get more granular and target specific applications or maps. But this is generally only necessary if an application made unusual use of the 3270 data stream and reported false hits.

See [3270 Intrusion Detection Service in z/OS Communications Server: SNA Network Implementation Guide](#) for the configuration and usage of 3270 IDS.



---

## Chapter 12. Security for MRO

CICS multiregion operation (MRO) enables CICS regions that are running in the same z/OS image, or in the same z/OS sysplex, to communicate with each other. MRO security determines which CICS regions can use MRO, which CICS regions can connect to other regions, and what security is applied to transactions when the regions communicate with each other.

For more information about the architecture of MRO connections, see [Multiregion operation](#).

For information about securing IPIC connections and some additional MRO information, see [Security for IPIC \(IP interconnectivity\)](#).

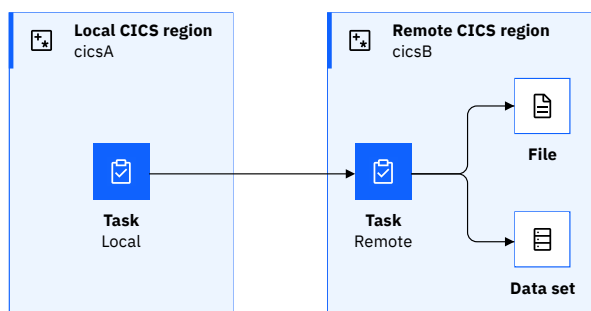
When you decide how to secure MRO connections, you need to consider the standard security principles: authentication, identification, authorization, integrity, confidentiality, and audit. For a description of these terms, see [What does security mean in CICS?](#).

---

### How it works: CICS MRO security

CICS MRO security is based on a combination of security controls. These controls determine what CICS regions can be connected, and also what security context is used.

CICS multiregion operation (MRO) connections pass requests between a *local* CICS system and a *remote* CICS system. [Figure 91 on page 187](#) defines the terms local CICS region and remote CICS region.



*Figure 91. Definition of local and remote CICS regions*

### The components of MRO security

#### MRO connection (bind-time) security

Connection or bind-time security is the method by which you control whether a CICS region is authorized to use inter-region communication (IRC) and whether a CICS region can connect to another CICS region using MRO.

You see two phases in establishing the connection security for an MRO connection:

- [Logon security](#)
- [Connect security](#)

#### Logon security

Logon security is the process CICS uses to confirm that a CICS region is authorized to use IRC, thus enabling it to use MRO. This check is made by validating that the CICS region user ID has UPDATE access

to the RACF DFHAPPL . *applid* profile in the FACILITY class. The *applid* value of the profile is defined by the APPLID of the CICS region.

**Recommendation:** It's recommended that you define DFHAPPL.applid profiles. If you don't, the default is to allow all regions to logon.

To prevent unauthorized CICS region connections, this check is made regardless of the setting on the **SEC SIT** parameter.

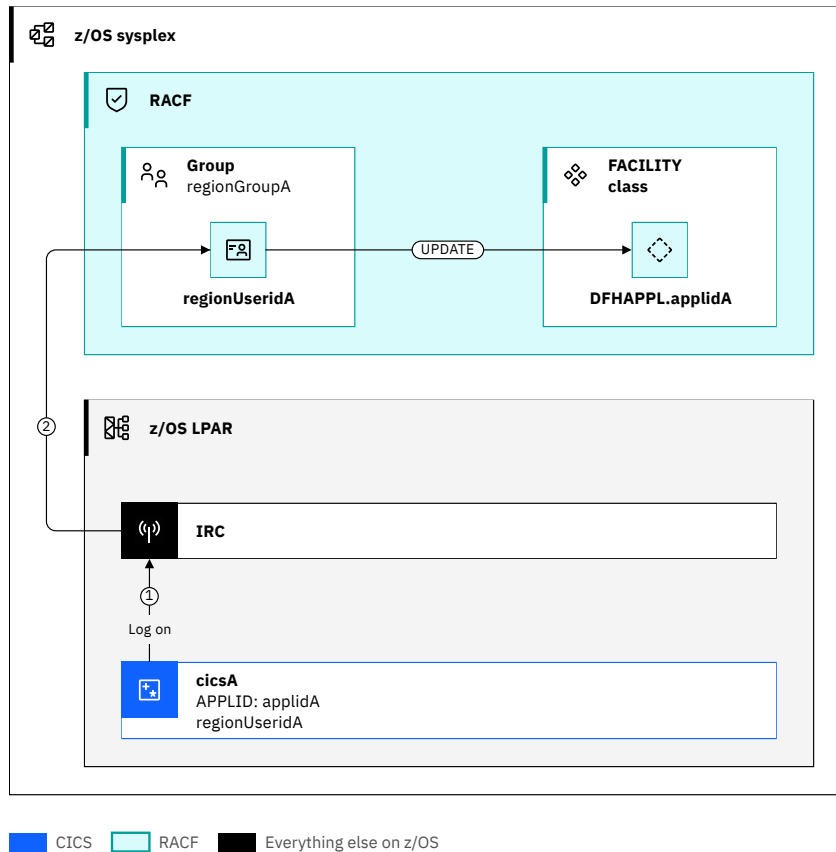


Figure 92. CICS region *cicsA* connects to IRC

1. CICS attempts to log on to IRC when IRC is opened, typically at initialization of the CICS region.
2. RACF verifies that the region user ID for the CICS region (*regionUseridA*) has UPDATE authority to the DFHAPPL . *applidA* profile in the FACILITY class.

## Changing a CICS region's authority to connect to IRC

If the authority for a *region user ID* to connect to IRC is changed, it is necessary to recycle the state of the connection to IRC. This recycling is needed because the authorization check is only made at the initial logon of the region to IRC.

The logon procedure can be recycled by a SET IRC CLOSED system programming command followed by a **SET IRC OPEN** command. The **SET IRC OPEN** command triggers the logon process and if successful open MRO connections.



## Connect security

Connect security occurs when one CICS region attempts to establish a connection to another CICS region by using MRO. Each CICS region must be authorized to connect to the partner region because connections are bidirectional.

The following diagrams show region *cicsA* makes the connection to *cicsB*, and region *cicsB* makes the reciprocal connection to *cicsA* respectively.

Figure 93 on page 189 shows the flow of a connect request from the local region *cicsA* to the remote CICS region *cicsB*.

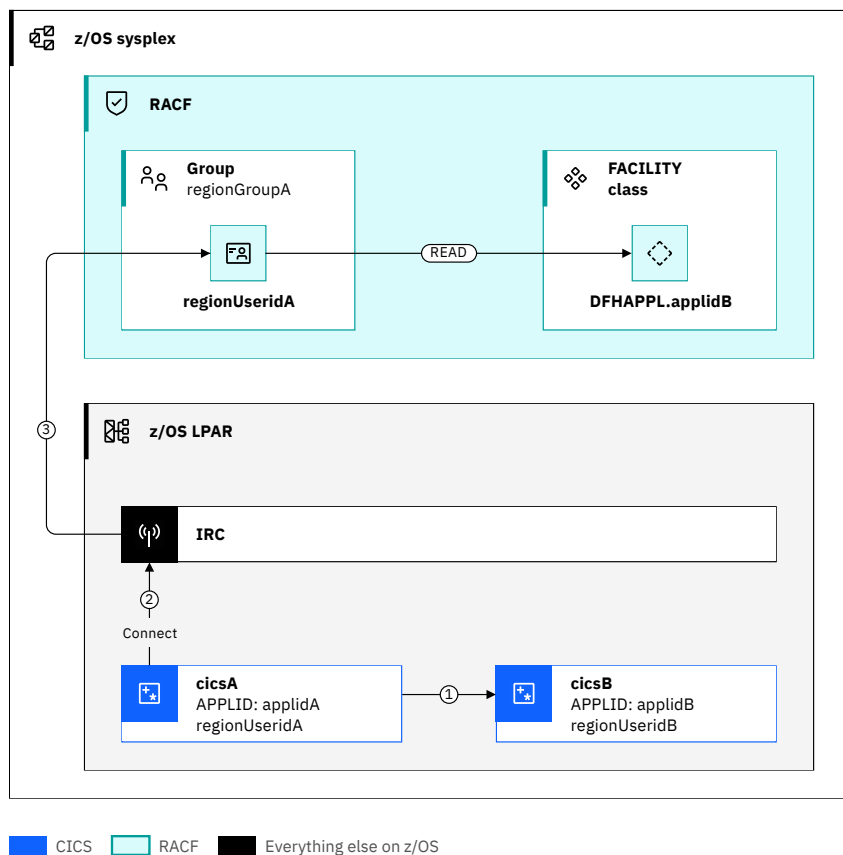


Figure 93. CICS region *cicsA* connects to CICS region *cicsB*

1. CICS region *cicsA* attempts to establish a connection to CICS region *cicsB*.
2. A request is made to IRC to verify that the region user ID (*regionUseridA*) for the CICS region that initiates the connection (*cicsA*) is authorized to connect to the target CICS region (*cicsB*).
3. IRC makes an authorization check to verify that *regionUseridA* has READ access to profile *DFHAPPL.applidB* in the FACILITY class.

Figure 94 on page 190 shows the flow of a connect request from the remote CICS region *cicsB* to the local CICS region *cicsA*.

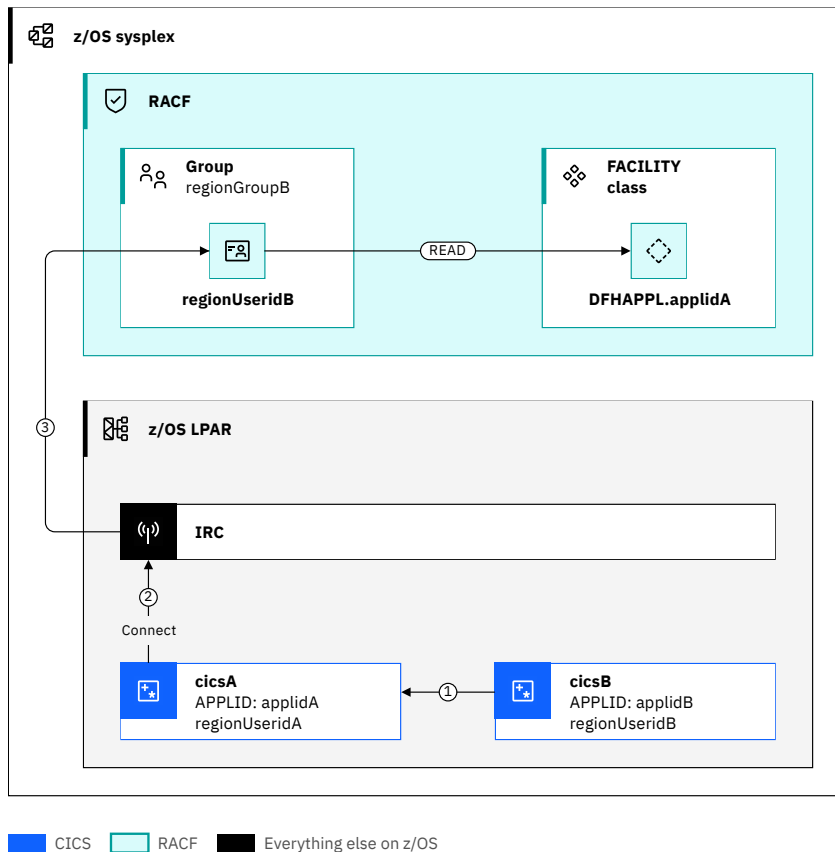


Figure 94. CICS region *cicsB* connects to CICS region *cicsA*

1. A reciprocal check is made by *cicsB*, which attempts to establish a connection to CICS region *cicsA*.
2. A request is made to connect to IRC to verify that the region user ID (*regionUseridB*) for the CICS region that initiates the reciprocal connection (*cicsB*) is authorized to connect to the target CICS region (*cicsA*).
3. IRC makes an authorization check to verify that *regionUseridB* has READ access to profile `DFHAPPL.applidA` in the FACILITY class.

Only when both connections are authorized will the connection between the CICS regions be established.

## MRO link security

*Link security* establishes the user ID, if any, that is associated with requests that are passed across the connection from the local CICS region to the remote CICS region. This user ID is known as the *link user ID*. The remote CICS region verifies that the *link user ID* has the authority to initiate requests in the remote CICS region and that it has access to the resources used by the initiated tasks.

Link security is independent of [user security](#).

### Purpose of link security

Link security has two purposes. First, it identifies where a request came from, which can be useful both in problem diagnosis and in audit. Second, it allows the remote CICS region to authorize requests based on whether the local CICS region is trusted.

## Determining link security

The link user ID is determined by a combination of attributes:

- The *region user ID* of the local and remote CICS regions
- The ATTACHSEC value in the CONNECTION definition in the remote CICS region
- The USERID value in the SESSION definition in the remote CICS region

The following diagram shows the resource definition attributes in use for link security.

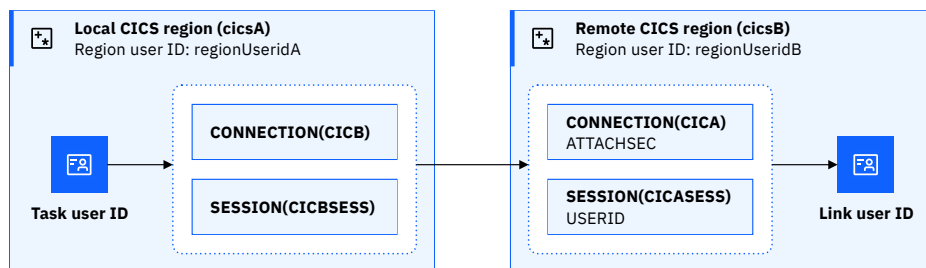


Figure 95. Resource definition attributes used for link security

How these values can be combined is shown in Table 21 on page 191.

Where a user ID is specified by the USERID value in the SESSION definition in the remote CICS region and it is different from the region user ID of the local CICS region. The link user ID is often known by the term *functional user ID* in the absence of a task ID.

Table 21. How the link user ID is determined					
Local CICS region cicsA	Remote CICS region cicsB Default User: <i>defaultUseridB</i>			Notes or examples	
Region user ID	Region user ID	Resource definition		Link user ID	
		ATTACHSEC attribute on CONNECTION	USERID attribute on SESSION		
<i>regionUseridA</i>	Same as <i>regionUseridA</i>	LOCAL		No link user ID	<b>1</b> Not recommended.
<i>regionUseridA</i>	<i>regionUseridB</i>	LOCAL		<i>regionUseridA</i>	<b>2</b> Not recommended.
<i>regionUseridA</i>	<i>regionUseridB</i>	LOCAL	<i>functionalUseridB</i>	<i>functionalUseridB</i>	Design example <a href="#">2</a>
<i>regionUseridA</i>	<i>regionUseridB</i>	LOCAL	Same as <i>regionUseridA</i>	<i>regionUseridA</i>	<b>1</b> Not recommended.
<i>regionUseridA</i>	Same as <i>regionUseridA</i>	IDENTIFY		No link user ID	Design example <a href="#">1</a>
<i>regionUseridA</i>	<i>regionUseridB</i>	IDENTIFY		<i>regionUseridA</i>	<b>2</b> Not recommended.
<i>regionUseridA</i>	<i>regionUseridB</i>	IDENTIFY	<i>functionalUseridB</i>	<i>functionalUseridB</i>	Design example <a href="#">3</a>

Table 21. How the link user ID is determined (continued)					
Local CICS region cicsA	Remote CICS region cicsB Default User: <i>defaultUseridB</i>				Notes or examples
<i>regionUseridA</i>	<i>regionUseridB</i>	IDENTIFY	Same as <i>regionUseridA</i>	<i>regionUseridA</i>	<b>2</b> Not recommended.

#### Recommended:

**1** ATTACHSEC(LOCAL) results in no propagation of the task user ID from the local region. In addition, because there is no link user ID, the task runs on the remote region with the default user ID. It is not recommended because the default user ID must have no authority.

**2** A region user ID must not be used as a link user ID. Instead, a functional user ID must be used to maintain a separation between the authorization of system tasks (category 1 transactions) and user tasks.

## MRO user security

User security is about using an authenticated or asserted user ID that represents the user. This user ID is known as the *task user ID*. The CICS region verifies that the task user ID has the authority to initiate requests and that it has access to the resources used by the initiated tasks.

User security is independent of [link security](#).

### Purpose of user security

User security has multiple purposes:

- It allows the propagation of the task user ID security context from the local CICS region to the remote CICS region, even when new work is started. Therefore, it allows tasks in the remote CICS region to continue to run under the initial task user ID.
- It allows the remote CICS region to authorize the requests that are received from the local CICS region based on the task user ID.
- It allows auditing at the task user ID level.

### Determining user security

User security is only used if the remote CICS region's CONNECTION definition specifies ATTACHSEC (IDENTIFY).

The following diagram shows how the task user ID is determined based on the resource definitions over an MRO connection:

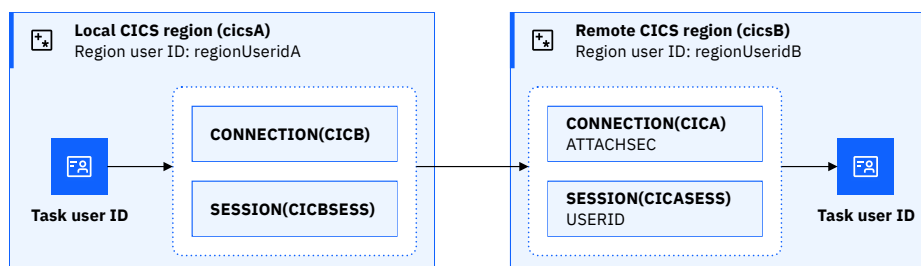


Figure 96. Resource definition attributes used for determining the task user ID

User security authorization checks are made in addition to any link user ID checks.

When ATTACHSEC (IDENTIFY) is specified in the remote region, the user ID is flowed as the task user ID across the connection. When ATTACHSEC (LOCAL) is specified, the task user ID is not flowed and the link user ID is used. This information is summarized in the following table.

Table 22. How the task user ID is determined

Local CICS region cicsA		Remote CICS region cicsB			Notes or examples
Region user ID	Task user ID	Region user ID	Resource definition		Task user ID
			ATTACHSEC attribute on CONNECTION	USERID attribute on SESSION <sup>a</sup>	
regionUseridA	TaskUseridA	Same as regionUseridA	LOCAL		defaultUseridB
regionUseridA	TaskUseridA	regionUseridB	LOCAL		regionUseridA <sup>b</sup>
regionUseridA	TaskUseridA	regionUseridB	LOCAL	functionalUseridB	functionalUseridB <sup>b</sup>
regionUseridA	TaskUseridA	regionUseridB	LOCAL	regionUseridA	regionUseridA <sup>b</sup>
regionUseridA	TaskUseridA	Same as regionUseridA	IDENTIFY		TaskUseridA
regionUseridA	TaskUseridA	regionUseridB	IDENTIFY		TaskUseridA
regionUseridA	TaskUseridA	regionUseridB	IDENTIFY	functionalUseridB	TaskUseridA
regionUseridA	TaskUseridA	regionUseridB	IDENTIFY	regionUseridA	TaskUseridA

<sup>a</sup> The USERID value specified in the SESSION definition is used for [MRO link security](#).

<sup>b</sup> When ATTACHSEC(LOCAL) is specified, the link user ID specified on the USERID option is also the task user ID.

#### Recommended:

<sup>1</sup> ATTACHSEC(LOCAL) results in no propagation of the task user ID from the local region. In addition, because there is no link user ID, the task runs on the remote region with the default user ID. It is not recommended because the default user ID must have no authority.

<sup>2</sup> A region user ID must not be used as a link user ID. Instead, a functional user ID must be used to maintain a separation between the authorization of system tasks (category 1 transactions) and user tasks.

### Distributed identity

If the task user ID in the local CICS region has an associated *distributed identity*, this information is also automatically flowed in requests to the remote CICS region. For more information, see [Identity propagation](#).

## Designing security for MRO

MRO connections can be used to connect CICS to CICS. The connections can be trusted or untrusted. To secure each of these scenarios, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

For more information about configuring security for MRO, see [Configuring security for MRO](#).

### Security design implications for MRO

When you design security for CICS across an MRO connection, consider the implications for:

- [“Authentication and identification” on page 194](#)
- [“Authorization” on page 194](#)
- [“Confidentiality and integrity” on page 194](#)
- [“Trust” on page 194](#)
- [“Audit” on page 194](#)

These considerations are explored as follows:

## Authentication and identification

You have a trust relationship between the local and remote CICS regions. Therefore, the remote CICS region is not involved in authenticating the user. The user ID is asserted by the local CICS region and trusted by the remote CICS region.

## Authorization

### Will you configure the CICS task in the remote CICS region to run with the RACF user ID of the user?

Configuring the CONNECTION with ATTACHSEC(IDENTIFY) enables fine-grained authorization and auditing of each request, based on the supplied security context.

### Do you need to limit the authority of requests from specific connections?

You might need to restrict what a user can do based on where the work came from.

You can use the link user ID to identify which connection was used, and use RACF profile definitions to limit access on the less trusted connection.

For example, work from one connection might come from mobile devices outside of the enterprise. Whereas work from another connection might come from secure devices inside the enterprise.

You can restrict the external mobile devices to be able to only READ data, whereas the internal secure devices can UPDATE the data.

### Do you want to limit access to CICS resources to a specific, functional user ID?

Configure the CONNECTION in the remote CICS region with ATTACHSEC(LOCAL) and configure its associated SESSION with the USERID set to the *functional user ID*.

## Confidentiality and integrity

Network traffic that is sent across MRO connections is not encrypted because the traffic cannot be sent outside the sysplex environment.

## Trust

Trust is provided by authorizing systems to access IRC, and authorizing regions to connect to each other. See [MRO connection \(bind-time\) security](#).

## Audit

By auditing MRO security, you can be sure that the configuration specified is correctly implemented.

- If a functional user ID is used, it is reported in the MSGUSR log. Look for the DFHZC5966 message for the connection to the local CICS region. If you use a functional user ID, the DFHZC5966 message is immediately followed with a DFHSN1400 message that contains the functional user ID.
- For other configurations, you need to look at the **CONNECTION** and **SESSION** definitions in the remote CICS regions and refer to [table 1 in MRO link security](#).

Review these design examples that offer different configurations.

## Design example: MRO connection – using only the task user ID

In this scenario, the requirement is that authorization in the remote CICS region is only based on the local *task user ID*.

For more information about configuring this scenario, see the configuration task [Configuration example: MRO connections - using only the task user ID](#).

The scenario is used where one or more of the following conditions apply:

- The task user ID that initiated the request in the local CICS region needs to be identifiable as the user ID under which the request is authorized in the remote CICS region.
- The auditing of individual user access to resources in the remote CICS region is required.
- The identity of the original user needs to be flowed through multiple tasks or outbound calls.

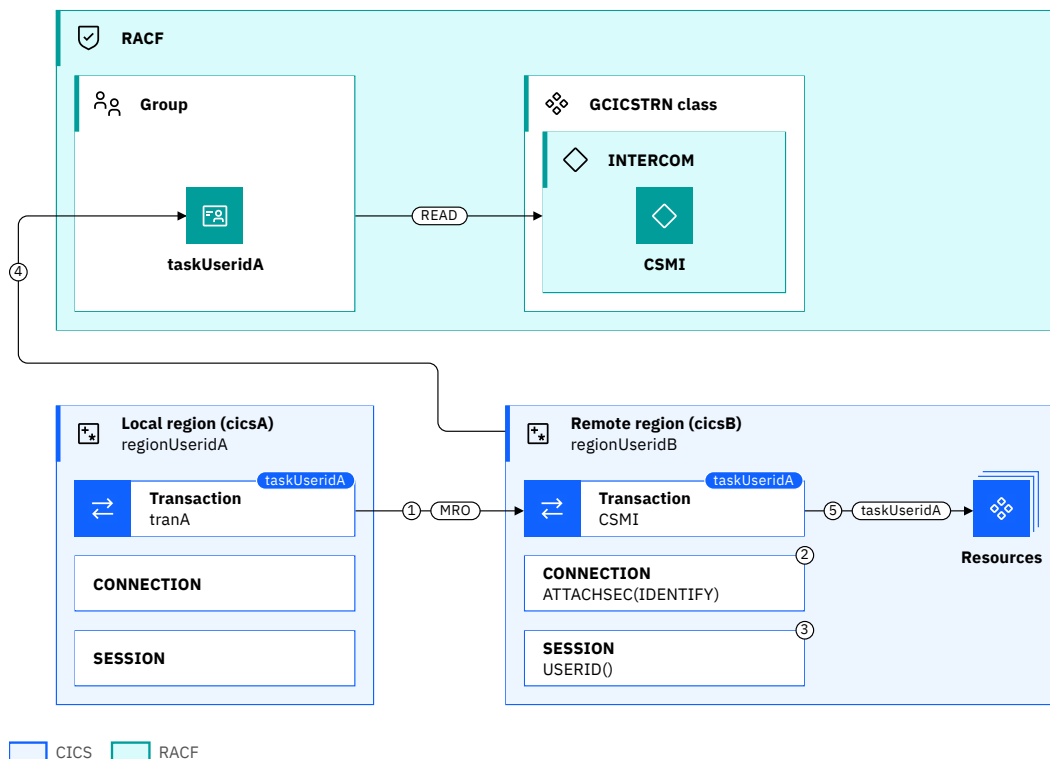


Figure 97. MRO connections - using only the task user ID

1. A DPL request is made by a task in *cicsA* that runs under *taskUseridA*.
2. *regionUseridA* is flowed from *cicsA* because the **CONNECTION** definition in the remote CICS region *cicsB* is defined with **ATTACHSEC( IDENTIFY)**.
3. The **SESSION** definition in *cicsB* does not specify a **USERID** value. Therefore, no *link user ID* is used.
4. *cicsB* checks that the *taskUseridA* is authorized to attach the transaction by calling RACF. CICS does this check by calling RACF to verify whether the *taskUseridA* belongs to a group that has **READ** access to a member list that contains the **CSMI** transaction in the **GCICSTRN** class. In this case, the **INTERCOM** member list in the CICS-supplied **DFH\$CAT2** sample is used. **INTERCOM** contains a set of intercommunication related transactions.
5. *taskUseridA* is associated with any additional request that is initiated by the task in *cicsB*, for example, by an **EXEC CICS START** command or an outbound request (if the other region requires a task user ID to flow).

## Related concepts

[MRO link security](#)

## Related information

[MRO user security](#)

## Configuration example: MRO Connections - using only the task user ID

Configure an MRO connection so work that is routed to the remote region runs under the task user ID of the local region.

### Before you begin

This configuration task is based on the example security scenario [Design example: MRO connection – using only the task user ID](#).

You must complete the following tasks:

- Understand the [Design example: MRO connection – using only the task user ID](#) and terminology used.
- Allow both regions to use MRO and allow them to connect to each other via MRO. For more information, see [Configuring security for MRO](#).

You need to know:

- How to define CICS resource definitions (examples show DFHCSDUP definitions). For more information, see [How you can define CICS resources](#).
- How to install definitions. For more information, see [Resource definition installation](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you learn how to configure the CICS resource definitions and the RACF security definitions so that a task on local region *cicsA* can DPL to a remote region *cicsB*. The work runs under the same task user ID as the local region.

This task assumes the following definitions:

- *connA*, *connB* are the names of CONNECTION definitions.
- *sessA*, *sessB* are the names of SESSION definitions.
- *groupA*, *groupB* are RDO group names.
- *applidA*, *applidB* are the APPLIDs of the CICS regions.
- *taskUserGroup* is the RACF group that contains the user IDs, such as *taskUseridA*, that are allowed to run work on *cicsB*.

### Procedure

1. Define in *cicsA* the MRO definitions to connect *cicsA* to *cicsB*.

```
DEFINE CONNECTION(connB) GROUP(groupA) ACCESSMETHOD(IRC) NETNAME(applidB) AUTOCONNECT(YES)
DEFINE SESSION(sessB) GROUP(groupA) CONNECTION(connB) PROTOCOL(LU61) AUTOCONNECT(YES)
```

2. Define in *cicsB* the MRO definitions to connect *cicsB* to *cicsA* specifying ATTACHSEC(IDENTIFY) on the connection definitions.



```
DEFINE CONNECTION(connA) GROUP(groupB) ACCESSMETHOD(IRC) NETNAME(applidA) AUTOCONNECT(YES)
ATTACHSEC(IDENTIFY)
DEFINE SESSION(sessA) GROUP(groupB) CONNECTION(connA) PROTOCOL(LU61) AUTOCONNECT(YES)
```

3. The task user IDs in CICS region A need to be able to run the CSMI transaction in *cicsB*. The following RACF definitions are based on the DFH\$CAT2 sample.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

4. If you use resource or command security, *taskUserGroup* also need to be given access to any resources or commands that are used by the DPLed programs.
5. Install *groupA* in *cicsA* and *groupB* in *cicsB*.

## Results

If you inquire on the connection by using the CICS Explorer **ISC/MRO Connections** view or CEMT INQUIRE CONNECTION, you see that they have a connection status of acquired.

To validate the security environment is functioning correctly, you need a transaction that a signed-on user on *cicsA* can run. This transaction needs the ability to issue a DPL request to a program on *cicsB*.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **3270** tab and set the **User ID** field to the user ID of the signed-on user. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: MRO connection – using only the link user ID

In this scenario, the requirement is that authorization in the remote CICS region is solely based on the user ID associated with the connection from the local CICS region. All requests from the local CICS region run under the same user ID, often known as a *functional user ID*.

For more information about configuring this scenario, see the configuration task [Configuration example: MRO connections - using only the link user ID](#).

The scenario is a configuration where no end-to-end auditing of resources is required and where authorization checks are managed in the local CICS region.

The scenario is used where one or more of the following conditions apply:

- You need no differentiation of the user that is required for the transaction attach, command, and resource authorization checks in the remote CICS region.
- No auditing of user access to resources in the remote CICS region is necessary.

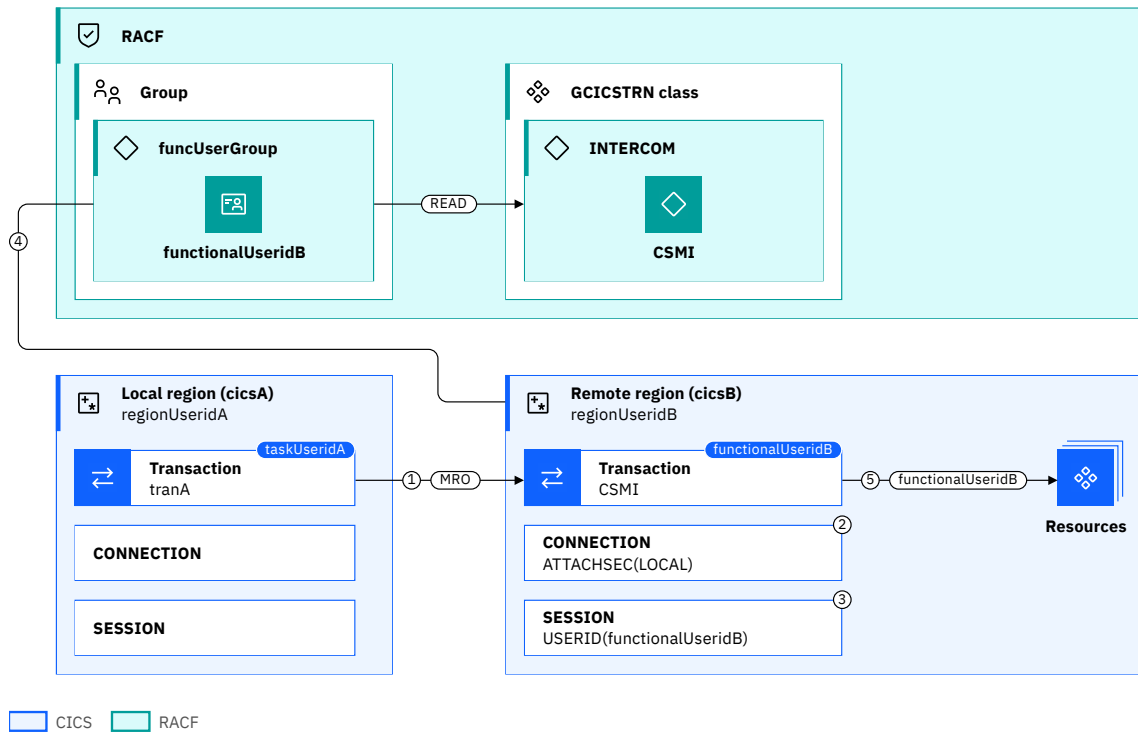


Figure 98. MRO connection – using only the link user ID

1. A DPL request is made by a task in *cicsA* that runs under *taskUseridA*.
2. *taskUseridA* does not flow from the local CICS region because the **CONNECTION** definition in the remote CICS region is defined with **ATTACHSEC(LOCAL)**.
3. *functionalUseridB* is used as the *functional user ID* because the **SESSION** definition in *cicsB* is defined with a **USERID** value of *functionalUseridB*.
4. *cicsB* checks that the functional user ID is authorized to attach the transaction by calling RACF. RACF verifies that *functionalUseridB* belongs to a group (*funcUserGroup*) that has **READ** access to a member list that contains the CSMI transaction in the GCICSTRN class. In this case, the INTERCOM member list in the CICS-supplied DFH\$CAT2 sample is used. INTERCOM contains a set of intercommunication related transactions.
5. *functionalUseridB* is associated with any additional request that is initiated by the task in the remote CICS region, for example by an **EXEC CICS START** command or an outbound request.

### Related concepts

[MRO link security](#)

## Configuration example: MRO Connections - using only the link user ID

Configure an MRO connection so that all work that is routed to the remote region runs under the same user ID. This user ID is often known as a functional user ID.

### Before you begin

This configuration task is based on the example security scenario [Design example: MRO connection – using only the link user ID](#).

You must complete the following tasks:

- Understand the [Design example: MRO connection – using only the link user ID](#) and terminology used.

- Allow both regions to use MRO and allow them to connect to each other via MRO. For more information, see [Configuring security for MRO](#).

You need to know how to define CICS resource definitions (examples show DFHCSDUP definitions)

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

## About this task

In this example, you learn how to configure the CICS resource definitions and the RACF security definitions so that a task on local region *cicsA* can DPL to a remote region *cicsB*. The work runs under a functional user ID.

This task assumes the following definitions:

- *connA*, *connB* are the names of CONNECTION definitions.
- *sessA*, *sessB* are the names of SESSION definitions.
- *groupA*, *groupB* are RDO group names.
- *applidA*, *applidB* are the APPLIDs of the CICS regions.
- *funcUserGroup* is the RACF group that contains the functional user IDs, such as *functionalUseridB*, that are allowed to run work on *cicsB*.
- *installGroup* is the RACF group of user IDs that can install RDO groups after CICS initialization.

## Procedure

1. Define in *cicsA* the MRO definitions to connect *cicsA* to *cicsB*.

```
DEFINE CONNECTION(connB) GROUP(groupA) ACCESSMETHOD(IRC) NETNAME(applidB) AUTOCONNECT(YES)
DEFINE SESSION(sessB) GROUP(groupA) CONNECTION(connB) PROTOCOL(LU61) AUTOCONNECT(YES)
```

2. Define in *cicsB* the MRO definitions to connect *cicsB* to *cicsA* specifying ATTACHSEC(LOCAL) on the connection definitions.

```
DEFINE CONNECTION(connA) GROUP(groupB) ACCESSMETHOD(IRC) NETNAME(applidA) AUTOCONNECT(YES)
ATTACHSEC(LOCAL)
DEFINE SESSION(sessA) GROUP(groupB) CONNECTION(connA) PROTOCOL(LU61) AUTOCONNECT(YES)
USERID(functionalUseridB)
```

3. The *functionalUseridB* needs to be able to run the CSMI transaction in *cicsB*. The following definitions are based on the DFH\$CAT2 sample.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)
PERMIT INTERCOM CLASS(GCICSTRN) ID(funcUserGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

4. If you use resource or command security, *functionalUseridB* also needs to be given access to any resources or commands that are used by the DPLed programs.
5. If the installation of the group is done after initialization, it is necessary for the installer to have surrogate access.

```
RDEFINE SURROGAT functionalUseridB.DFHINSTL UACC(NONE)
PERMIT functionalUseridB.DFHINSTL CLASS(SURROGAT) ID(installGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

6. Install *groupA* in *cicsA* and *groupB* in *cicsB*.

## Results

If you inquire on the connections, on you see that they have a connect status of acquired.

To validate the security environment is functioning correctly, you need a transaction that a signed-on user on *cicsA* can run. This transaction needs the ability to issue a DPL request to a program on *cicsB*.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **3270** tab and set the **User ID** field to the user ID of the signed on user. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: MRO connection – using both the task ID and the link user ID

In this scenario, the requirement is that authorization in the remote CICS region is based on both the local task user ID and the user ID associated with the connection from the local CICS region.

For more information about configuring this scenario, see the configuration task [Configuration example: MRO connections - using both the task user ID and the link user ID](#).

The scenario is used where one or more of the following conditions apply:

- The *task user ID* that initiated the request in the local CICS region needs to be identifiable as the user ID under which the request is authorized in the remote CICS region.
- The auditing of individual user access to resources in the remote CICS region is necessary.
- The identity of the original user needs to be flowed through multiple tasks or outbound calls.
- You need to restrict access for requests that come from some regions.
- You need to audit where requests originated.

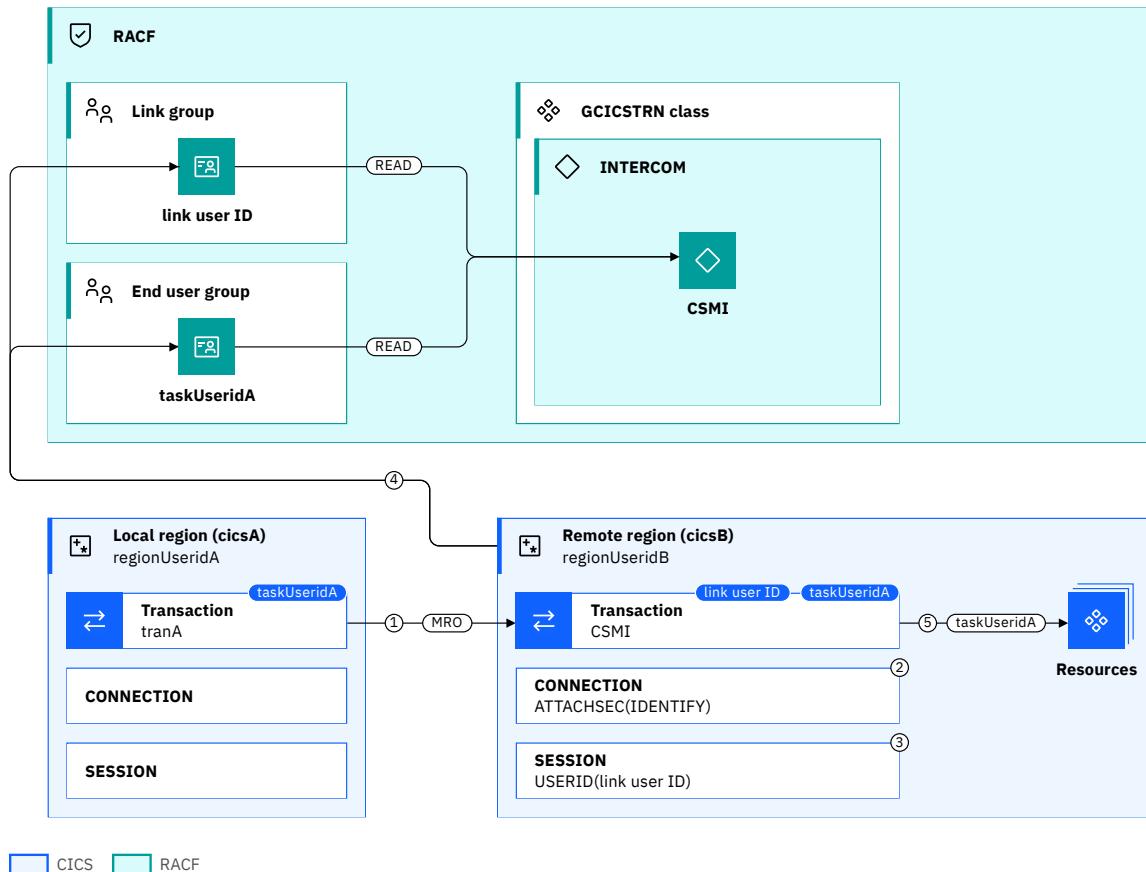


Figure 99. MRO connection – using both the task ID and the link user ID

1. A DPL request is made by a task in *cicsA* that runs under *taskUseridA*.
2. *taskUseridA* flows because the CONNECTION definition in *cicsB* is defined with ATTACHSEC(IDENTIFY).
3. A link user ID is also used because the SESSION definition in *cicsB* is defined with a USERID value of the *link user ID*.
4. *cicsB* checks that both *taskUseridA* and the *link user ID* are authorized to attach the transaction by calling RACF. RACF verifies that both *taskUseridA* and the *link user ID* belong to groups that have READ access to a member list that contains the CSMI transaction in the GCICSTRN class. In this case, the INTERCOM member list in the CICS-supplied DFH\$CAT2 sample is used. INTERCOM contains a set of intercommunication related transactions.
5. *taskUseridA* is associated with any additional request that is initiated by the task in *cicsB*, for example by an **EXEC CICS START** command or an outbound request.

## Related information

### [Security for MRO](#)

## Configuration example: MRO Connections - using both the task user ID and the link user ID

Configure an MRO connection so that all work is routed to the remote region runs under the task user ID of the local region. In addition, all security is carried out on the link user ID.

### Before you begin

This configuration task is based on the example security scenario [Design example: MRO connection – using both the task ID and the link user ID](#).

You must complete the following tasks:

- Understand the [Design example: MRO connection – using both the task ID and the link user ID](#) and terminology used.
- Allow both regions to use MRO and allow them to connect to each other via MRO. For more information, see [Configuring security for MRO](#).

You need to know how to define CICS resource definitions (examples show DFHCSDUP definitions)

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you learn how to configure the CICS resource definitions and the RACF security definitions so that a task on local region *cicsA* can DPL to a remote region *cicsB*. The work runs under the same task user ID as the local region. In addition, all security checks are done for the link user ID as well.

This task assumes the following definitions:

- *connA*, *connB* are the names of CONNECTION definitions.
- *sessA*, *sessB* are the names of SESSION definitions.
- *groupA*, *groupB* are RDO group names.
- *applidA*, *applidB* are the APPLIDs of the CICS regions.
- *taskUserGroup* is the RACF group that contains the user IDs, such as *taskUseridB*, that are allowed to run work on *cicsB*.
- *linkUserGroup* is the RACF group that contains the link user IDs, such as *linkUseridB*, that define which links are allowed to run work on *cicsB*.
- *installUserid* as the user ID of the installer of the groups.

### Procedure

1. Define in *cicsA* the MRO definitions to connect *cicsA* to *cicsB*.

```
DEFINE CONNECTION(connB) GROUP(groupA) ACCESSMETHOD(IRC) NETNAME(applidB) AUTOCONNECT(YES)
DEFINE SESSION(sessB) GROUP(groupA) CONNECTION(connB) PROTOCOL(LU61) AUTOCONNECT(YES)
```

2. Define in *cicsB* the MRO definitions to connect *cicsB* to *cicsA* specifying ATTACHSEC(IDENTIFY) on the connection definitions.

```
DEFINE CONNECTION(connA) GROUP(groupB) ACCESSMETHOD(IRC) NETNAME(applidA) AUTOCONNECT(YES)
ATTACHSEC(IDENTIFY)
DEFINE SESSION(sessA) GROUP(groupB) CONNECTION(connA) PROTOCOL(LU61) AUTOCONNECT(YES)
USERID(linkUseridA)
```

3. The task user ID and the link user ID need to be able to run the CSMI transaction in *cicsB*. The following definitions are based on the DFH\$CAT2 sample.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)
PERMIT INTERCOM CLASS(GCICSTRN) ID(linkUserGroup) ACCESS(READ)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

4. If you use resource or command security *linkUserGroup* and *taskUserGroup* need to be given access to any resources or commands that are used by the DPLed programs.

- 5.

```
RDEFINE SURROGAT linkUserid.DFHINSTL UACC(NONE)
PERMIT linkUserid.DFHINSTL CLASS(SURROGAT) ID(installUserid) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

6. Install *groupA* in *cicsA* and *groupB* in *cicsB*.

## Results

If you inquire on the connections, you see that they have a connection status of acquired.

To validate the security environment is functioning correctly, you need a transaction that a signed-on user on *cicsA* can run. This transaction needs the ability to issue a DPL request to a program on *cicsB*.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **3270** tab and set the **User ID** field to the user ID of the signed on user. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Configuring security for MRO

The following design examples describe sample MRO security topologies to demonstrate recommended configurations. Each design example has a dedicated configuration example that highlights the key configuration steps that are needed for implementation.

- [Design example: MRO connection – using only the task user ID](#)
- [Design example: MRO connection – using only the link user ID](#)
- [Design example: MRO connection – using both the task ID and the link user ID](#)

The following sections describe common configuration tasks and additional information.

### Allowing the regions to use MRO

For each CICS region with APPLID *applid* and region user ID *region user ID*, you can allow it to use MRO using the following RACF commands:

```
RDEFINE FACILITY DFHAPPL.applid UACC(NONE)
PERMIT DFHAPPL.applid CLASS(FACILITY) ID(region user ID) ACCESS(UPDATE)
```

### Allowing regions to connect to each other using MRO

For two CICS regions with APPLIDs *applidA* and *applidB*, and region user IDs *regionUseridA* and *regionUseridB*, allow them to connect to each other via MRO by using the following RACF commands:

```
RDEFINE FACILITY DFHAPPL.applidA UACC(NONE)
PERMIT DFHAPPL.applidA CLASS(FACILITY) ID(regionUseridB) ACCESS(READ)
RDEFINE FACILITY DFHAPPL.applidB UACC(NONE)
PERMIT DFHAPPL.applidB CLASS(FACILITY) ID(regionUseridA) ACCESS(READ)
```

The RACF definitions can be simplified. This configuration is possible if you have a naming conventions for a set of APPLIDs, and want to allow any of these regions to be able to connect to each other,

For example, you have a set of production regions whose APPLIDs start with the characters PROD. You also have a RACF *productionRegionGroup* with the same set of production region IDs. The RACF definitions for this configuration will be:

```
RDEFINE FACILITY DFHAPPL.PROD* UACC(NONE)
PERMIT DFHAPPL.PROD* CLASS(FACILITY) ID(productionRegionGroup) ACCESS(READ)
```

Which regions are to connect is still subject to connection definitions.

**Recommendation:**

- Do not mix definitions for production and test or development regions.
- Where possible, development regions must use a separate RACF database. This configuration avoids any possibility of development regions that connect to production regions.



---

## Chapter 13. Security for IPIC (IP interconnectivity)

IPIC is the way that CICS Transaction Server for z/OS provides intersystem communication over TCP/IP. IPIC connections can be used in many scenarios: for example, to connect CICS to CICS, CICS to IMS, CICS to CICS TX on Cloud or TXSeries®, and clients to CICS across TCP/IP networks. For more information about the architecture of IPIC connections, see [Intercommunication using IP interconnectivity](#). IPIC connections provide a comprehensive set of security controls for both *transport security* and CICS user authentication.

For information about securing MRO connections and some additional IPIC information, see [Security for MRO](#).

When you decide how to secure IPIC connections, you need to consider the standard security principles: authentication, identification, authorization, integrity, confidentiality, and audit. For a description of these terms, see [What does security mean in CICS?](#)

---

### How it works: CICS IPIC security

CICS IPIC security is based on a combination of IP networking security controls and CICS security controls. TCP/IP network security controls operate at the connection level, whereas CICS security controls operate at the CICS resource or transaction level.

#### Components of IPIC security

#### IPIC connection security

*IPIC connection security* provides the ability to control which remote systems can connect to a CICS region, based on the supplied connection parameters. This process can be controlled either by using the installation process of the IPCONN resource or by using TLS client and server authentication.

When an IPIC connection is acquired between two CICS regions, three sets of network flows occur for each socket that is established.

##### TCP/IP 3-way handshake

Used by the TCP/IP stack to open the socket connection between the local CICS region and the remote CICS region. “[When the socket is established \(TCP/IP 3-way handshake\)](#)” on page 206 describes how connection security is implemented at this stage.

##### TLS handshake

Negotiates the cipher suite, validates the client and server certificates, then creates an encrypted session for the exchange of further requests. “[When the TLS session is established \(TLS handshake\)](#)” on page 207 describes how connection security is implemented at this stage.

The following diagram shows a TLS 1.2 handshake. For comparison between a TLS 1.2 and a TLS 1.3 handshake, see [Transport Layer Security \(TLS\) for IP connections](#).

##### CICS IPIC capability exchange (CAPEX)

The first request to flow between the local and remote CICS regions. “[At the IPIC capability exchange \(CAPEX\) request](#)” on page 207 describes how connection security is implemented at this stage.

These flows are shown in [Figure 100 on page 206](#). Connection security can be implemented at each of these flows.

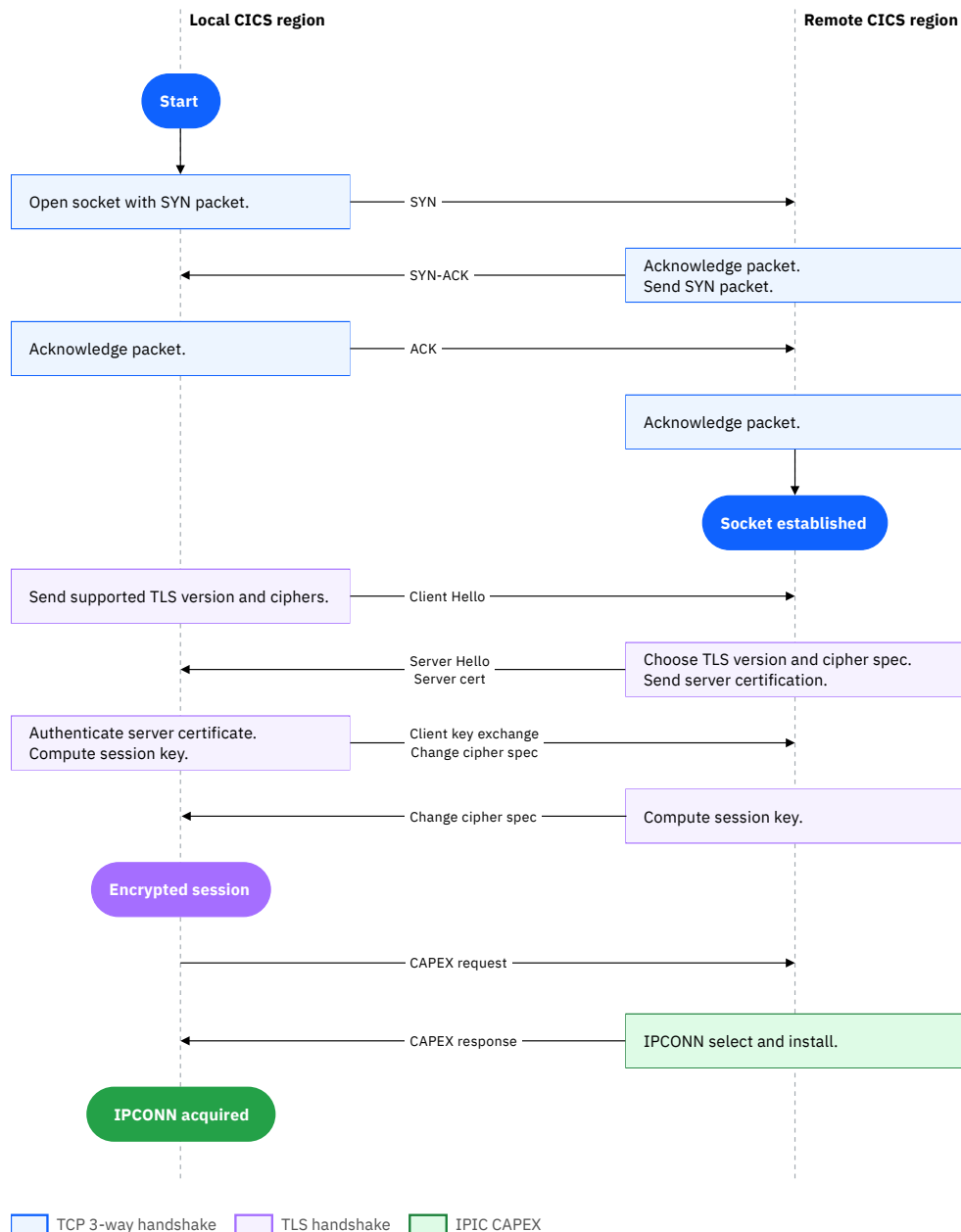


Figure 100. Network flows to establish an IPIC socket

Connections between CICS systems usually have two inbound sockets in each direction, making a total of four sockets per IPIC connection: two inbound and two outbound in each direction.

Connections between CICS and a client, such as IBM z/OS Connect, use one-way IPIC connections. These one-way connections have only inbound sockets. Either one or two of these sockets are used depending on how many were initiated by the client.

For each socket, connection security can be implemented at any of the following stages:

#### When the socket is established (TCP/IP 3-way handshake)

A 3-way handshake is used by the TCP/IP stack to open the socket connection between the local CICS region and the remote CICS region. This handshake uses the remote host and port information from the local IPCONN resource definition. If the local system is an IPIC client, such as z/OS Connect or CICS Transaction Gateway, then the equivalent values are used from the client configuration file. It is

a three-step process that requires both the client and server to exchange synchronization (SYN) and acknowledgment (ACK) packets before the real data communication process starts.

You can control the ability to create a TCP/IP socket connection between two IP addresses in a z/OS sysplex by using SAF SERVAUTH class. This class prevents unauthorized user access to TCP/IP resources, including TCP/IP stacks, ports, and networks. Additionally, external network security devices, such as firewalls, can be used to allow or block incoming and outgoing network traffic.

### **When the TLS session is established (TLS handshake)**

After the TCP/IP socket is established, a TLS handshake is performed if TLS support is specified in the CICS TCPIP SERVICE definition. The TLS handshake negotiates the cipher suite, validates the client and server certificates, and then creates an encrypted session for the exchange of further requests. [Figure 100 on page 206](#) shows a TLS handshake without client authentication at TLS 1.2 or earlier.

TLS can be used to control if a client and server system can connect to each other, and provides similar function to that provided by CICS MRO and APPC *bind time security* (see [Intercommunication security](#)). This control is based on *public key cryptography*. Authentication is performed by an exchange of X.509 certificates. The X.509 certificates are issued and digitally signed by an external authority, which is known as a *certificate authority* (CA). Certificates are used to authenticate clients to servers and servers to clients. The mechanism that is used is essentially the same in both cases. However, the server certificate is mandatory: that is, the server must send its certificate to the client, but the client certificate is optional, and servers decide whether to require client authentication for a connection.

Certificates are stored in a RACF key ring and the CICS region user ID must be authorized to access a specific key ring that contains all the certificates that it needs to establish TLS connections with its partners. The personal certificates in the key ring are used to identify client and server. The signer certificates are used by the partner system to authenticate these certificates by using the public key from the CA.

For more information, see [Transport Layer Security \(TLS\) for IP connections](#).

### **At the IPIC capability exchange (CAPEX) request**

After the socket and its TLS session are established, the CICS IPIC capability exchange (CAPEX) request is the first request to flow between the local and remote CICS regions. The remote CICS region validates the CAPEX and uses the supplied information either to locate a matching IPCONN resource definition, or, if the IPIC autoinstall program is active, to install a new IPCONN resource definition.

The CAPEX response, including any negotiated IPIC session values, is returned from the remote CICS region to the local CICS region and, if required, a further socket is created by the local CICS region. If the connection is a 2-way IPCONN for CICS-to-CICS connectivity, the same process is repeated from the remote CICS region to the local CICS region to establish the inbound sockets and install the IPCONN resource definition in the local CICS region.

By default, TCPIP SERVICE resource definitions that have a PROTOCOL of IPIC specify the use of the default autoinstall program DFHISAIP. This program allows IPIC CAPEX requests from any system to connect dynamically if no matching IPCONN resource definition can be located.

**Security best practice (validated by IBM Health Checker for z/OS):** It is recommended that you do not use the default autoinstall program DFHISAIP in production. This can be done by setting the user-replaceable module (URM) attribute in the relevant TCPIP SERVICE resource definition to NO (meaning no URM is used) or to a user written URM. This is to prevent connections from any system.

A custom autoinstall URM can be written to validate incoming CAPEX requests. A sample program, DFH\$ISAI, is supplied by CICS to validate connections based on a pre-defined network name, then uses this to locate an IPCONN resource definition template. For more information about the IPIC autoinstall user replaceable modules (URM), see [Sample autoinstall user program to support predefined connection templates](#).

## IPIC transport security

Transport Level Security (TLS) support for the TCP/IP sockets that are used by IPIC is provided by System SSL. TLS can be used to provide authentication of client and server systems and confidentiality of transmitted data. TLS client and server authentication perform a similar role to SNA session security and MRO bind-time security (see [Intercommunication security](#)).

A CICS-to-CICS IPIC connection consists of two socket connections in each direction. One set of sockets is used for inbound requests and the other for outbound requests. TLS support must be configured for both the outbound and inbound sets of sockets. The settings, such as cipher suites and client authentication support, can be different for each set if required. The TLS protocol level is determined by the highest level that is supported by both CICS regions, as determined by their [MAXTLSLEVEL](#) SIT values.

Transport security is controlled by the SSL, CERTIFICATE, and CIPHERS attributes in the region's TCPIP SERVICE resource definition, along with equivalent settings in the client. For more information, see [TCPIP SERVICE resource](#).

Figure 101 on page 208 summarizes the resources that are used to define TLS connectivity between two CICS regions.

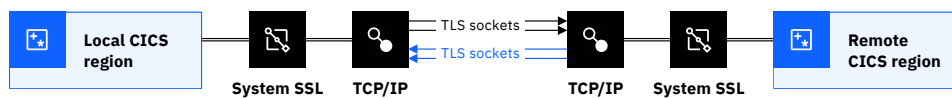


Figure 101. TLS support for inbound and outbound IPCONN resource definitions

The TCPIP SERVICE resource definition defines the inbound side of the connection. These settings tell CICS which server certificate to send to the client and if it must receive a valid TLS client certificate before it allows the client to acquire the IPCONN.

The IPCONN resource definition defines the outbound side of the connection. These settings tell CICS to initiate a TLS handshake. During the TLS handshake, CICS asks the partner region for the server certificate that is specified in the TCPIP SERVICE resource definition. If SSL (CLIENTAUTH) is specified in the TCPIP SERVICE resource definition in the remote system, then the remote system also requests the certificate of the originating system as part of the handshake. The IPCONN definition does not enable unless a valid client certificate is provided.

Figure 102 on page 208 summarizes the resources that are used to define TLS with a one-way IPCONN between an IPIC client and a CICS region.

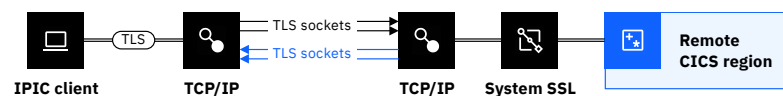


Figure 102. TLS support for IPIC client connections

A one-way IPCONN has only inbound sockets and either one or two sockets are used, depending on how many were initiated by the client. Transport security is controlled by the SSL, CERTIFICATE, and CIPHERS attributes in the region's TCPIP SERVICE resource definition, along with equivalent settings in the client's configuration.

## IPIC security and AT-TLS

AT-TLS (Application Transparent Transport Layer Security) is an alternative implementation of TLS on z/OS, provided by IBM z/OS Communications Server. However, AT-TLS is supported only in basic mode for IPIC connections and means that the CICS region cannot query requests so it is unaware that the TLS

protocol is in use. As a result, support is not available for user authentication by using client certificates. See [Implementation options for TLS](#).

## IPIC link security

*Link security* establishes the user ID, if any, that is associated with requests that are passed across the connection from the local CICS region to the remote CICS region. This user ID is known as the *link user ID*. The remote CICS region verifies that the *link user ID* has the authority to initiate requests in the remote CICS region and that it has access to the resources used by the initiated tasks.

### Purpose of link security

Link security has two purposes. First, it identifies where a request came from, which can be useful both in problem diagnosis and in audit. Second, it allows the remote CICS region to authorize requests based on whether the local CICS region is trusted.

### Determining link security

Link security is configured with the LINKAUTH and SECURITYNAME attributes on the IPCONN resource definition of the receiving system. It acts as an extra security check for all intercommunication requests from the connected system. The practical effect of link security is to prevent remote users from attaching a transaction or accessing a resource for which the link user ID has no authority, regardless of the authority of the user ID that is flowed.

Additionally, if you use TLS client authentication, link security can set the task user ID based on the user ID that is associated with the client's TLS certificate. This is configured by using the attributes USERAUTH(LOCAL) and LINKAUTH(CERTUSER).

For more information about the types of user ID used in CICS, see [How it works: Identification in CICS](#).

The link user ID is determined according to the following table.

Local CICS region: <i>cicsA</i>	Remote CICS Region: <i>cicsB</i> Default User: <i>DefaultUseridB</i>			
Region user ID	Region user ID	IPCONN resource definition		Link user ID
		LINKAUTH attribute	SECURITYNAME attribute	
<i>regionUseridA</i>	<i>regionUseridB</i>	CERTUSER		<i>certUseridA</i>
<i>regionUseridA</i>	<i>regionUseridB</i>	SECUSER	<i>regionUseridB</i>	No link user ID
<i>regionUseridA</i>	<i>regionUseridB</i>	SECUSER	<i>functionalUseridB</i>	<i>functionalUseridB</i>
<i>regionUseridA</i>	<i>regionUseridB</i>	SECUSER		<i>defaultUseridB</i>

**Recommended:** A *default user ID* must not be used as a *link user ID*. Instead, a *functional user ID* must be used. The default user ID must not have authority to run transactions or access resources.

## IPIC user security

User security is about using an authenticated or asserted user ID that represents the user. This user ID is known as the *task user ID*. The CICS region verifies that the task user ID has the authority to initiate requests and that it has access to the resources used by the initiated tasks.

### Purpose of user security

User security has multiple purposes:

- It allows the propagation of the task user ID security context from the local CICS region to the remote CICS region, even when new work is started. Therefore, it allows tasks in the remote CICS region to continue to run under the initial task user ID.
- It allows the remote CICS region to authorize the requests that are received from the local CICS region based on the task user ID.
- It allows auditing at the task user ID level.

## Determining the user security

User security causes a check of the security context that flows with each transaction request from the sending system to the target CICS region. It is controlled by the USERAUTH attribute in the [IPCONN](#) resource definition in the remote CICS region.

Figure 103 on page 210 shows the security attributes defined in the resource definitions for user security.

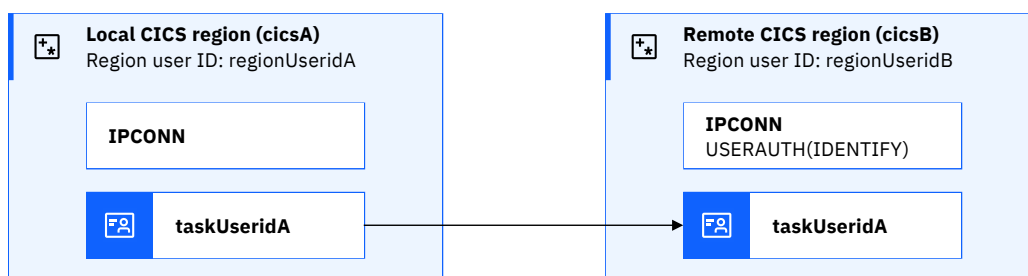


Figure 103. Resource definition attributes used for user security

The security context can identify one of the following types of user:

- The user ID of the CICS transaction that issues the remote request.
- The user ID that is flowed in a client request from a connected client system, such as CICS Transaction Gateway or z/OS Connect.
- The distributed identity that is flowed in a request from a connected CICS region or client system. For more information, see [Identity propagation](#).

The user security check validates the flowed user security context either by verifying a flowed password or by using a TLS client certificate. Alternatively, in specific circumstances (see [1](#)), the server can trust that the link from the sending system has already verified the identity of the request. In this case, no authentication check occurs and the server validates that only the user ID that is flowed exists in the local user registry. This condition is known as a *trusted connection*.

## Identification and authentication

User security validates the user ID that is flowed and its password based on the USERAUTH attributes on the [IPCONN](#) resource definition. These checks are summarized in [Table 23 on page 210](#).

Table 23. Identification and authentication	
USERAUTH attribute	Description
IDENTIFY (see <a href="#">1</a> )	The <i>asserted user ID</i> is trusted by the remote region, but it is validated to check that RACF knows about this user ID.
VERIFY	The client user ID and password are verified by RACF in the remote region.
LOCAL	No user ID is flowed, so no authentication is required.

Table 23. Identification and authentication (continued)	
USERAUTH attribute	Description
DEFAULTUSER	No user ID is flowed, so no authentication is required.

1 USERAUTH(IDENTIFY) is only valid on a trusted link. Therefore, it is restricted to the following conditions:

- IPIC connections that use TLS client authentication: that is, the TCPIPService resource definition specifies SSL (CLIENTAUTH).
- The remote socket is established by a trusted route within the sysplex, as determined by TCP/IP and is reported in CICS by the SO\_CLUSTERCONNTYPE options that are returned by z/OS Communications Server. You can inquire on this value with the CICS command **INQUIRE IPCONN() CLIENTLOC()**.

## How is the task user ID determined?

Depending on the setting of the link user ID and the USERAUTH option on the IPCONN resource definition, each user request can run under the task user ID that is shown in Table 1. However, in cases where a link user ID is also provided, it is associated with the task as a secondary user ID. This secondary user ID is used in addition to the task user ID for all checks for transaction security and resource security.

Table 24. How to identify the secondary ID when a link user ID is provided					
Local CICS region: <i>cicsA</i>		Remote CICS region: <i>cicsB</i> Default User ID: <i>defaultUseridB</i>			
Region user ID	Task user ID	Region user ID	USERAUTH on IPCONN	Link user ID *	Task user ID
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	LOCAL	<i>linkUseridB</i>	<i>linkUseridB</i>
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	LOCAL	<i>regionUseridA</i>	<i>defaultUseridB</i> **
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	IDENTIFY or VERIFY	<i>linkUseridB</i>	<i>taskUseridA</i>
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	IDENTIFY or VERIFY	<i>regionUseridA</i>	<i>taskUseridA</i>
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	DEFAULTUSER	<i>linkUseridB</i>	<i>defaultUseridB</i> **
<i>regionUseridA</i>	<i>taskUseridA</i>	<i>regionUseridB</i>	DEFAULTUSER	<i>regionUseridA</i>	<i>defaultUseridB</i> **

\* The link user ID can be either pre-defined in the IPCONN resource definition or set dynamically based on the TLS client certificate that is used for the inbound IPCONN resource definition. If the link user ID is not set in these ways, it defaults to the region user ID. See [How it works: IPIC link security](#).

**Recommended:** \*\* Avoid use of the default user ID as the task user ID. The default user ID must not have authority to access resources or run transactions other than CAT 3 transactions.

## Designing security for IPIC

IPIC connections can be used in many scenarios, for example, to connect CICS to CICS and to connect clients to CICS across Internet Protocol networks. The connections can be trusted or untrusted. To secure each of these scenarios, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

For more information on configuring security for IPIC, see [Configuring security for IPIC](#).

## Security design considerations for IPIC

When you design security for CICS web service provider scenarios, consider the implications for:

- [Authentication and identification](#)
- [Authorization](#)
- [“Confidentiality and integrity” on page 213](#)
- [Trust](#)
- [Audit](#)

These considerations are explored as follows.

### Authentication and identification

#### When to use links with password authentication?

When you use client-to-CICS IPIC connections, you can choose to authenticate with either password or passphrases by setting the `USERAUTH(VERIFY)` attribute on the `IPCONN` resource definition. This method is best used with transport security to provide confidentiality for the nonencrypted password..

#### When to use trusted connections with `USERAUTH(IDENTIFY)`?

`USERAUTH(IDENTIFY)` on an **IPCONN** resource definition requires incoming attach requests to specify a user ID. Both CICS-to-CICS IPIC connections and client-to-CICS IPIC connections can use a [trusted connection with `USERAUTH\(IDENTIFY\)`](#). When you use trusted `IPCONN` connections, the task user security context is established without any password or certificate authentication mechanism. This method is designed for scenarios where the local system can authenticate the caller by using its own user registry and is restricted to trusted connections as documented in Table 1 in [How it works: IPIC user security](#). Examples of this kind of scenario include a CICS web owning region, or a WebSphere Application Server with a local user registry.

**Recommendation:** When you use trusted `IPCONN` connections, it is recommended that you apply further security to the connection by using either of the following techniques:

- Transport security with client authentication so that the server can confirm the authenticity of the client.
- Link security to restrict the resources that any authenticated user can access. Link security can also be integrated with TLS client authentication because the link user ID can be established by using the identity of the distinguished name in the client certificate.

Each client can have a dedicated signer certificate. You can permit access to a partner CICS region by adding the public key from each client's signer certificate to the key ring of the partner CICS region.

#### When to use link security?

Link security provides an extra way of authorizing access to transactions and other resources in the receiving system. It is especially useful if you need to restrict authorization based on the connection into the CICS region. For instance, from a specific trusted web owning region or from a trusted IPIC client connection.

### Authorization

#### When to configure the CICS task in the remote CICS region to run with the RACF user ID of the user?

Configuring the `IPCONN` in the remote CICS region with `USERAUTH(IDENTIFY)` enables fine-grained authorization and auditing of each request, based on the supplied security context.

#### Do you need to limit the authority of requests from specific connections?

Using link security and IPIC user security allows you to ensure that no request that is made through a specific `IPCONN` can have more authority than that is granted to the link user ID, whatever authority the flowed user ID has.



### **Do you want to limit access to CICS resources to a specific, functional user ID?**

Instead of configuring the IPCONN to run requests under the user's identity, requests can be allowed to run under the link user ID. This configuration can either be set by the SECURITYNAME attribute on IPCONN or mapped from the TLS client certificate that is associated with the IPCONN definition.

## **Confidentiality and integrity**

### **What TLS implementation to use?**

You can implement TLS by using System SSL or AT-TLS. However, CICS IPIC is an AT-TLS unaware application, which means that a client cannot use a certificate to authenticate with CICS using IPIC.

### **What TLS version and cipher suite to use?**

You are recommended to use the most recent version of TLS that is also supported by the partner system. The most recent version of the TLS specification includes fixes to previous versions and brings enhancements like support for stronger cipher suites. To control the TLS version that is supported by CICS, you can set **MINTLSLEVEL** and **MAXTLSLEVEL** SIT parameters. To set the list of ciphers that are supported, you can modify the contents of the ciphers XML files that are used in the IPCONN or TCPIP SERVICE resource.

## **Trust considerations for systems that connect through IPIC**

TLS client authentication can be used so that the partner system provides a client certificate that is validated by CICS. Successful client authentication requires that the certificate authority (CA) that signed the client certificate is considered trusted by CICS. To be considered trusted, the certificate of the CA must be in the CICS key ring.

## **Audit considerations for IPIC security**

By auditing IPIC security, you can be sure that the configuration specified is correctly implemented.

- To verify that the inbound IPIC sockets are encrypted with TLS, check that the SSLTYPE attribute on the IPCONN resource definition is set to SSL. You can check with the SPI command **INQUIRE IPCONN** after the connection is acquired.
- To verify that TLS client authentication is in use, check that the SSL attribute on the TCPIP SERVICE resource definition is set to CLIENTAUTH. The TCPIP SERVICE resource definition that is used is specified in the inbound IPCONN resource definition.
- To verify the TLS cipher suite that is selected, see the SOCIPHER field in the performance class monitoring data for any request that is attached through the connection.
- To verify the APPLID or IP address of an IPIC partner system, check the [DFHIS2000](#) message that is issued to the MSGUSR log after the connection is enabled.
- To verify whether an IPIC autoinstall user replaceable module is in use, check the value of the URM attribute on each TCPIP SERVICE resource definition that has the attribute of PROTOCL (IPIC). A value other than NONE indicates the name of the URM that is in use.
- To verify the remote APPLID and IP address of any autoinstalled IPCONN resource definitions, view the information that is supplied in message [DFHIS3000](#) in the local CICS region.

Review these design examples that offer different configurations.

## **Design example: Securing CICS-to-CICS with an IPIC connection within a sysplex**

In this scenario, the two CICS regions in the same sysplex are connected by using an IPIC connection, and no network traffic flows outside the sysplex. This configuration enables the remote CICS region to trust the authentication performed in the local CICS region, without requiring an encrypted connection.

For more information about configuring for this scenario, see the configuration task [Configuration example: Securing CICS-to-CICS with an IPIC connection within a sysplex](#).

The IPCONN resource in the remote CICS region specifies `USERAUTH(IDENTIFY)`, `LINKAUTH(SECUSER)`, with the `SECURITYNAME` attribute set to the local CICS region user ID (*regionUseridA*).

Figure 104 on page 214 shows an overview of the scenario.

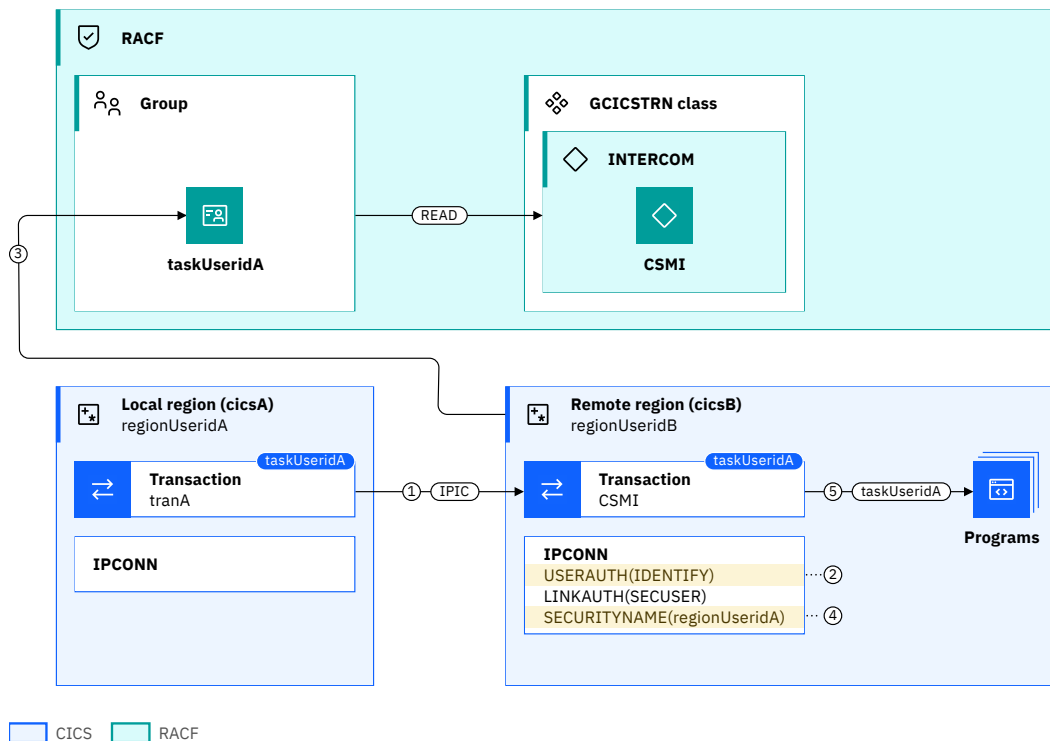


Figure 104. IPIC trusted connection between systems in the same sysplex

The following security flows occur at the points shown:

1. User *taskUseridA* runs transaction *tranA* in the local CICS region. The transaction links to a program in the remote CICS region over the IPCONN that connects the two regions.
2. *taskUseridA* is flowed from the local CICS region because the IPCONN definition in the remote CICS region is defined with `USERAUTH(IDENTIFY)`.
3. Authorization of the transaction in the remote CICS region is performed by checking whether *taskUseridA* has authority to run the specified mirror transaction CSMI. The remote CICS region does this check by calling RACF to check whether *taskUseridA* belongs to a group that has READ authority to a member list that includes CSMI in the GCICSTRN class. In this case, the INTERCOM member list in the CICS-supplied DFH\$CAT2 sample is used. INTERCOM contains a set of intercommunication related transactions.
4. No link user ID is in use because the `SECURITYNAME` attribute in the remote CICS region is set to the region user ID of the local CICS region, *regionUseridA*.
5. *taskUseridA* is associated with any additional request that is initiated by the transaction in the remote CICS region, for example, by an **EXEC CICS START** command or an outbound request (if the other region requires a task user ID to flow).

## Configuration example: Securing CICS-to-CICS with an IPIC connection within a sysplex

Configure an IPIC connection between two CICS regions in the same sysplex. Work routed to the remote region runs under the task user ID of the transaction in the local region.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing CICS-to-CICS with an IPIC connection within a sysplex](#).

You need to know how to define CICS resource definitions (examples show DFHCSDUP definitions)

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you learn how to configure the CICS resource definitions and the RACF security definitions so that a task on local region *cicsA* can DPL (distributed program link) to a remote region *cicsB*. The work runs under the same task user ID as the local region. In addition, all security checks are done for the link user ID as well.

This task assumes the following definitions:

- *ipconnA*, *ipconnB* are the names of IPCONN definitions.
- *tcpipA*, *tcpipB* are the names of TCPIP SERVICE definitions.
- *groupA*, *groupB* are RDO group names.
- *applidA*, *applidB* are the APPLIDs of the CICS regions.
- *hostA*, *hostB* are the hostnames of the LPARs on which CICS runs.
- *portA*, *portB* are the port numbers of the CICS regions.
- *taskUserGroup* is the RACF group that contains the user IDs, such as *taskUseridA*, that are allowed to run work on *cicsB*.
- *regionUseridA* is the user IDs of *cicsA*.
- *installUserid* is the user ID of the installer of the groups.
- *nSess* is the number of sessions.

### Procedure

1. Define in *cicsA* the IPIC definitions to connect *cicsA* to *cicsB*.

```
DEFINE IPCONN(ipconnB) GROUP(groupA)
      APPLID(applidB) HOST(hostB) PORT(portB)
      TCPIP SERVICE(tcpipB)
      SENDCOUNT(nSess) RECEIVECOUNT(nSess)
DEFINE TCPIP SERVICE(tcpipB) GROUP(groupA)
      PORT(portA) PROTOCOL(IPIC) URM(NO)
```

2. Define in *cicsB* the IPIC definitions to connect *cicsB* to *cicsA*.

```
DEFINE IPCONN(ipconnA) GROUP(groupB)
      APPLID(applidA) HOST(hostA) PORT(portA)
      TCPIP SERVICE(tcpipA)
      LINKAUTH(SECUSER) SECURITYNAME(regionUseridA)
      USERAUTH(IDENTIFY)
      SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)
DEFINE TCPIP SERVICE(tcpipA) GROUP(groupB)
      PORT(portB) PROTOCOL(IPIC) URM(NO)
```

3. The task user ID and the link user ID both need authority to run the CSMI transaction in *cicsB*. The following definitions are based on the DFH\$CAT2 sample where INTERCOM is the name of the set of transactions that are associated with intercommunication.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
PERMIT INTERCOM CLASS(GCICSTRN) ID(regionUseridA) ACCESS(READ)
SETOPTS RACLIST(TCICSTRN) REFRESH
```

4. If you use resource or command security, *taskUserGroup* needs to be given access to any resources or commands that are used by the DPLed programs. For more information, see [Resource security](#).
5. Install *groupA* in *cicsA* and *groupB* in *cicsB*.
- If you are making this change manually, *installUserid* needs authority to install a resource with the user ID *regionUseridA*. For more information, see [Installing a resource with user ID attributes](#).

## Results

You see the following messages on MSGUSR on *cicsA*.

```
DFHIS2012 22/02/2022 13:28:01 applidA The connection status in IPCONN ipconnB from applid applidB
is changed from RELEASED to OBTAINING due to a SPI command to ACQUIRE the IPCONN issued on the
remote system.
DFHIS2000 22/02/2022 13:28:01 applidA Server web session 1 with applid applidB on host hostB,
port portB acquired for IPCONN
ipconnB.
DFHIS2000 22/02/2022 13:28:01 applidA Server web session 2 with applid applidB on host hostB,
port portB acquired for IPCONN
ipconnB.
DFHIS2001 22/02/2022 13:28:01 applidA Client web session 1 from applid applidB accepted for
IPCONN ipconnB.
DFHIS2012 22/02/2022 13:28:01 applidA The connection status in IPCONN ipconnB from applid applidB
is changed from OBTAINING to
ACQUIRED.
```

```
DFHIS2001 22/02/2022 13:28:01 applidA Client web session 2 from applid applidB accepted for
IPCONN ipconnB.
```

You see the following messages on MSGUSR on *cicsB*.

```
DFHIS2012 22/02/2022 13:28:01 applidB The connection status in IPCONN ipconnA from applid applidA
is changed from RELEASED to OBTAINING due to a SPI command to ACQUIRE the IPCONN issued on the
local system.
DFHIS2001 22/02/2022 13:28:01 applidB Client web session 1 from applid applidA accepted for
IPCONN ipconnA.
DFHIS2001 22/02/2022 13:28:01 applidB Client web session 2 from applid applidA accepted for
IPCONN ipconnA.
DFHIS2000 22/02/2022 13:28:01 applidB Server web session 1 with
applid applidA on host hostA, port portA acquired for IPCONN
ipconnA.
DFHIS2000 22/02/2022 13:28:01 applidB Server web session 2 with
applid applidA on host hostA, port portA acquired for IPCONN
ipconnA.
DFHIS2012 22/02/2022 13:28:01 applidB The connection status in IPCONN ipconnA from applid applidA
is changed from OBTAINING to ACQUIRED.
```

If you query the IPCONN in the explorer or CEMT, you see that the connections are *inservice* and *acquired*.

To validate the security environment is functioning correctly, you need a transaction that a signed-on user on *cicsA* can run. This transaction needs the ability to issue a DPL request to a program on *cicsB*.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **3270** tab and set the **User ID** field to the user ID of the signed on user. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: Securing CICS-to-CICS with an IPIC connection that uses TLS

In this scenario, the two CICS regions are connected by an encrypted TLS connection and network traffic flows across the network outside the sysplex. A TLS encrypted network connection between the two systems is required. The regions must also use client authentication to identify each other.

For more information about configuring for this scenario, see the configuration task [Configuration example: Securing CICS-to-CICS with an IPIC connection that uses TLS](#).

This scenario describes a client or server example where a program in CICS region *cicsA* can issue a dynamic program link (DPL) to region *cicsB* and get a response. It is not possible to issue a DPL from *cicsB* to *cicsA*.

It is assumed that both regions use a common RACF database. This RACF database can be shared or cloned. The common RACF database is required so that the same user IDs are defined in both regions with the same roles.

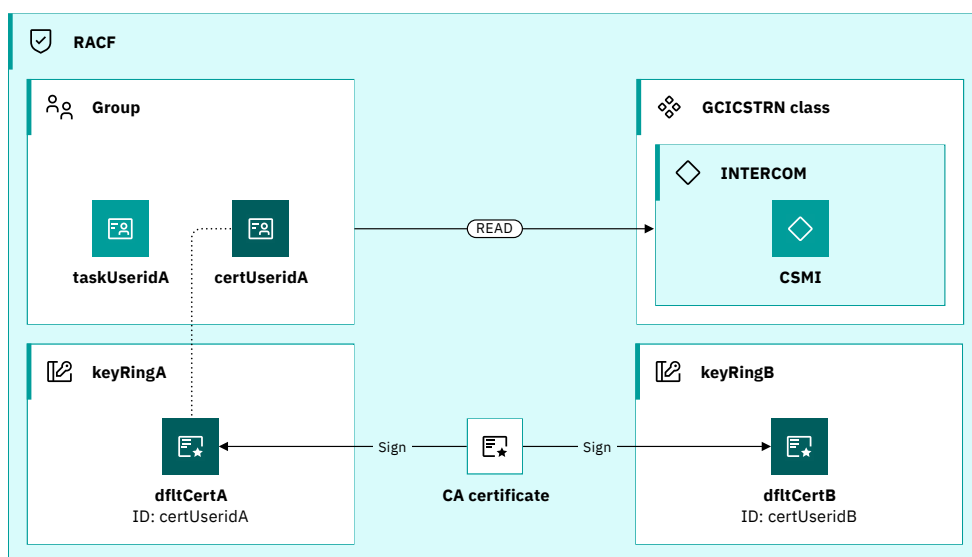



Figure 105. RACF configuration to connect two CICS regions that use TLS

In [Figure 105 on page 217](#), both regions have key rings with a default certificate that is signed by a certificate authority (CA) known to both regions. The default certificate  for *cicsA* (*dfltCertA*) is defined to have a certificate user ID *certUseridA* in *cicsB*, and vice versa.

*certUseridA* is given READ access to CSMI because it belongs to a group that has READ access to a member list that contains CSMI. In this case, the INTERCOM member list is supplied in the DFH\$CAT2 sample and contains a set of transactions for intercommunication. Granting *certUseridA* READ access to CSMI allows *cicsA* to DPL to *cicsB*. *certUseridB*'s access to CSMI is NONE, so *cicsB* is not allowed to DPL to *cicsA*.

The following diagram shows an overview of the CICS to CICS connection that uses IPIC.

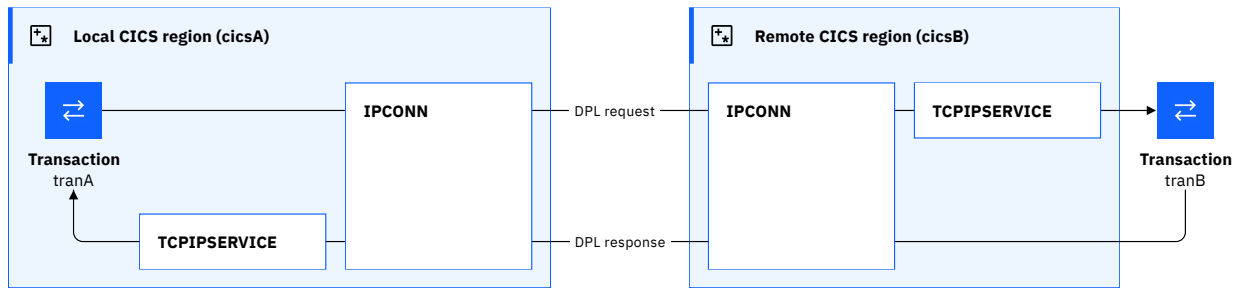


Figure 106. Overview of IPCONN and TCPIP setup to connect two CICS regions that use TLS

Shown in Figure 106 on page 218 are a pair of definitions with IPCONN resources in both regions, where an IPCONN resource in one region connects to a TCPIPService resource in the other region. The DPL request uses one set of definitions, and the response uses the other. The configuration of these two regions is symmetric. The only definition that defines *cicsA* as the client and *cicsB* as the server is the RACF definition that is shown in Figure 105 on page 217.

The flow of a DPL request from *cicsA* to *cicsB*, and the definitions that are required to achieve this configuration are shown in Figure 107 on page 218. The DPL response from *cicsB* to *cicsA* and the definitions required to achieve this flow are shown in Figure 108 on page 219.

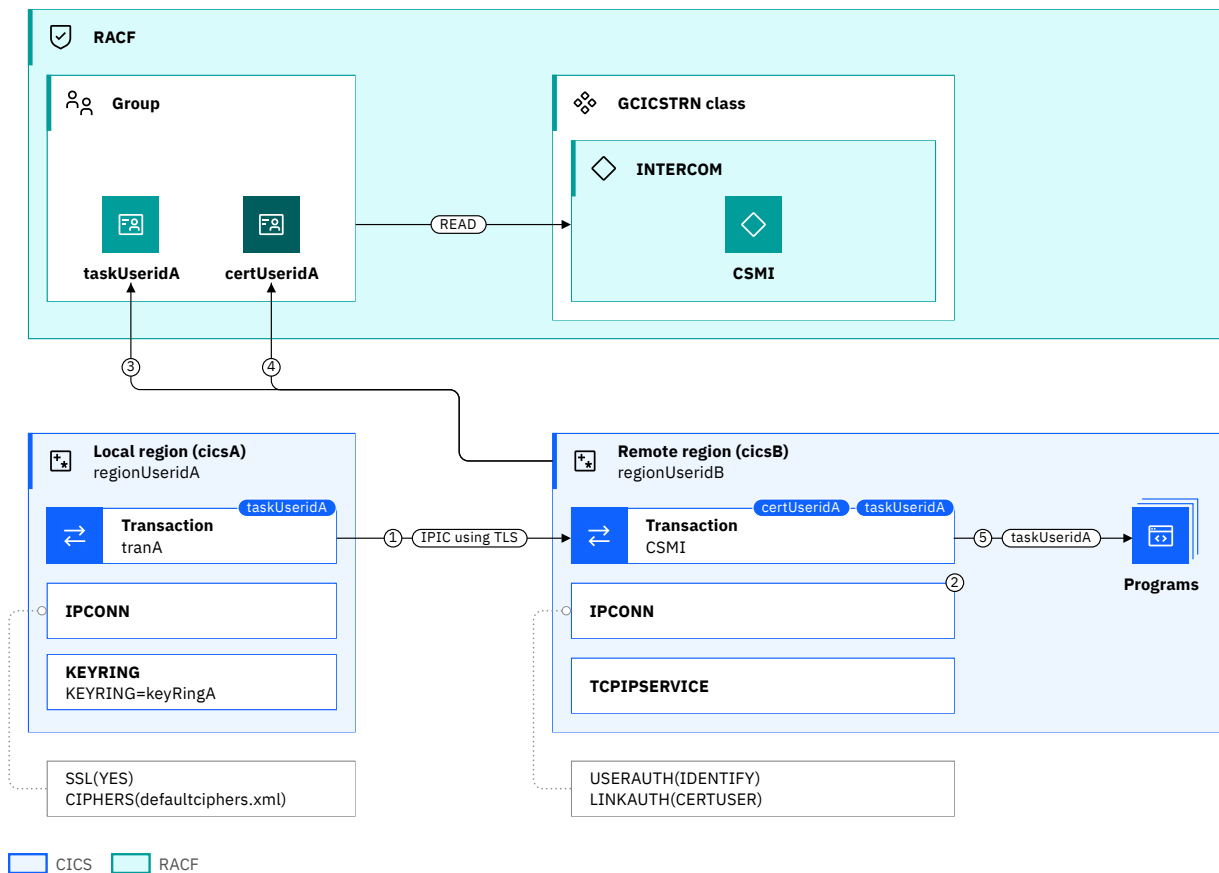


Figure 107. IPCONN and TCPIP setup for a DPL request

Figure 107 on page 218 shows the security flows as follows:

1. User *taskUseridA* runs transaction *tranA* in the local CICS region *cicsA*. The transaction links to a program in the remote CICS region *cicsB* over the IPCONN that connects the two regions.
2. *taskUseridA* is flowed from the local CICS region because the IPCONN definition in the remote CICS region is defined with `USERAUTH(IDENTIFY)`.
3. Authorization of the transaction in the remote CICS region is performed by checking whether *taskUseridA* has authority to run the specified mirror transaction CSMI. The remote CICS region does this check by calling RACF to verify whether *taskUseridA* belongs to a group that has READ authority to a member list that includes CSMI in the GCICSTRN class. In this case, the INTERCOM member list in the CICS-supplied DFH\$CAT2 sample is used. INTERCOM contains a set of intercommunication related transactions.
4. Link security is also in force because the LINKAUTH attribute in the remote CICS region is set to CERTUSER. The *certUseridA* is mapped from the certificate that is passed from *cicsA* \*. *cicsB* does this check by calling RACF to verify whether *certUseridA* belongs to a group that has READ authority to the member list (INTERCOM) that includes CSMI, in the GCICSTRN class.
5. *taskUseridA* is associated with any additional request that is initiated by the transaction in the remote CICS region. For example, by an **EXEC CICS START** command or an outbound request (if the other region requires a task user ID to flow).

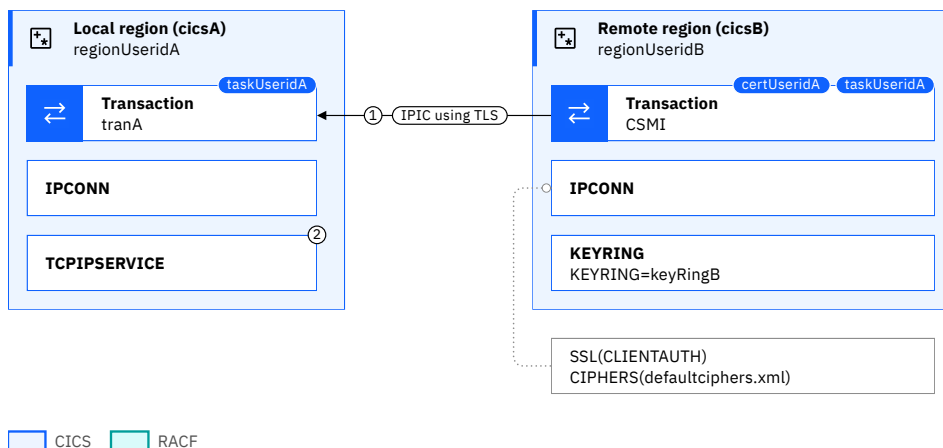


Figure 108. IPCONN and TCPIP setup for the DPL response

Figure 108 on page 219 shows the security flows as follows:

1. The program returns its response to the originating transaction *tranA* in the local CICS region *cicsA* over the IPCONN that connects the two regions.
2. The response is received by the TCPIP SERVICE in *cicsA*.

\* In this example the certificates that are used are the CICS region's default certificates. As an alternative you can specify a specific certificate for a connection. The label of the certificate needs to be added to the CERTIFICATE option of the TCPIP SERVICE definition. For more information, see [Authentication and identification](#).

## Configuration example: Securing CICS-to-CICS with an IPIC connection that uses TLS

Configure an IPIC connection between two CICS regions that are connected by TLS. The local region connects to the remote region by using TLS. Work routed to the remote region runs under the task user ID of the local region.

### Before you begin

This configuration task is based on the [Design example: Securing CICS-to-CICS with an IPIC connection that uses TLS](#).

You must complete these tasks:

- Understand the [Design example: Securing CICS-to-CICS with an IPIC connection that uses TLS](#) and the terminology that is used in that design example.
- Configured the CICS regions to be able to use TLS see [Configuring CICS to use SSL](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you learn how to configure the CICS resource definitions and the RACF security definitions so that a task on local region *cicsA* can DPL to a remote region *cicsB*. The work runs under the same task user ID as the local region. In addition, all security checks are done for the link user ID as well.

This task assumes the following definitions:

- *dfltCertA*, *dfltCertB* are the default certificate labels.
- *certUseridA*, *certUseridB* are the user ID of the default certificates.
- *keyRingA* *keyRingB* are the names of the keyrings
- *ipconnA*, *ipconnB* are the names of IPCONN definitions.
- *tcpipA*, *tcpipB* are the names of TCPIP SERVICE definitions.
- *groupA*, *groupB* are RDO group names.
- *applidA*, *applidB* are the APPLIDs of the CICS regions.
- *hostA*, *hostB* are the hostnames of the LPARs on which CICS runs.
- *lparA*, *lparB* are the LPARs on which CICS runs.
- *portA*, *portB* are the port numbers of the CICS regions.
- *taskUserGroup* is the RACF group that contains the user IDs, such as *taskUseridA*, that are allowed to run work on *cicsB*.

This task assumes that the user ID *taskUseridA* is defined in the RACF database on both *cicsA* and *cicsB*, and has the same role on both systems.

### Procedure

1. Export certificate *dfltCertA* from *lparA* to *dataset* and transfer to *lparB*.

```
RACDCERT EXPORT (LABEL('dfltCertA')) DSN('datasetName') -  
FORMAT(CERTDER)
```

2. Import *dfltCertA* in *datasetName* to *certUseridA* in *lparB*.

```
RACDCERT ADD('datasetName') TRUST ID(certUseridA)
```



3. Export certificate *dfltCertB* from *lparB* to *dataset* and transfer to *lparA*.

```
RACDCERT EXPORT (LABEL('dfltCertB')) DSN('datasetName') -  
FORMAT(CERTDER)
```

4. Import *dfltCertB* in *datasetName* to *certUseridB* in *lparA*.

```
RACDCERT ADD('datasetName') TRUST ID(certUseridB)
```

5. Define in *cicsA* the IPIC definitions to connect *cicsA* to *cicsB*.

```
DEFINE IPCONN(ipconnB) GROUP(groupA)  
  APPLID(applidB) HOST(hostB) PORT(portB)  
  TCPIP SERVICE(tcpipB) SSL(YES)  
  LINKAUTH(CERTUSER) USERAUTH(IDENTIFY)  
  SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)  
DEFINE TCPIP SERVICE(tcpipB) GROUP(groupA)  
  PORT(portA) PROTOCOL(IPIC) URM(NO)  
  SSL(CLIENTAUTH)
```

6. Define in *cicsB* the IPIC definitions to connect *cicsB* to *cicsA*.

```
DEFINE IPCONN(ipconnA) GROUP(groupB)  
  APPLID(applidA) HOST(hostA) PORT(portA)  
  TCPIP SERVICE(tcpipA) SSL(YES)  
  LINKAUTH(CERTUSER) USERAUTH(IDENTIFY)  
  SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)  
DEFINE TCPIP SERVICE(tcpipA) GROUP(groupB)  
  PORT(portB) PROTOCOL(IPIC) URM(NO)  
  SSL(CLIENTAUTH)
```

7. The task user ID and the certificate user ID need to be able to run the CSMI transaction in *cicsB*. The following definitions are based on the DFH\$CAT2 sample where INTERCOM is the name of the set of transactions that are associated with intercommunication.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)  
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)  
PERMIT INTERCOM CLASS(GCICSTRN) ID(certUseridA) ACCESS(READ)  
SETOPTS RACLIST(TCICSTRN) REFRESH
```

8. If you use resource or command security, *taskUserGroup* needs to be given access to any resources or commands that are used by the DPLed programs.
9. Install *groupA* in *cicsA* and *groupB* in *cicsB*.

## Results

You see the following messages on MSGUSR on *cicsA*.

```
DFHIS2012 11/03/2022 13:54:58 applidA The connection status in IPCONN ipconnB from applid applidB  
is changed from RELEASED to OBTAINING due to a SPI command to ACQUIRE the IPCONN issued on the  
remote system.  
DFHIS2000 11/03/2022 13:54:58 applidA Server web session 1 with applid applidB on host hostB,  
port portB acquired for IPCONN  
ipconnB.  
DFHIS2000 11/03/2022 13:54:58 applidA Server web session 2 with applid applidB on host hostB,  
port portB acquired for IPCONN  
ipconnB.  
DFHIS2001 11/03/2022 13:54:58 applidA Client web session 1 from applid applidB accepted for  
IPCONN ipconnB.  
DFHIS2012 11/03/2022 13:54:58 applidA The connection status in IPCONN ipconnB from applid applidB  
is changed from OBTAINING to  
ACQUIRED .
```

```
DFHIS2001 11/03/2022 13:54:58 applidA Client web session 2 from applid applidB accepted for  
IPCONN ipconnB.
```

You see the following messages on MSGUSR on *cicsB*.

```
DFHIS2012 11/03/2022 13:54:58 applidB The connection status in IPCONN ipconnA from applid applidA  
is changed from RELEASED to OBTAINING due to a SPI command to ACQUIRE the IPCONN issued on the  
local system.  
DFHIS2001 11/03/2022 13:54:58 applidB Client web session 1 from applid applidA accepted for  
IPCONN ipconnA.  
DFHIS2001 11/03/2022 13:54:58 applidB Client web session 2 from applid applidA accepted for
```

```
IPCONN ipconnA.  
DFHIS2000 11/03/2022 13:54:58 applidB Server web session 1 with  
applid applidA on host hostA, port portA acquired for IPCONN  
ipconnA.  
DFHIS2000 11/03/2022 13:54:58 applidB Server web session 2 with  
applid applidA on host hostA, port portA acquired for IPCONN  
ipconnA.  
DFHIS2012 11/03/2022 13:54:58 applidB The connection status in IPCONN ipconnA from applid applidA  
is changed from OBTAINING to ACQUIRED.
```

If you query the IPCONN in the explorer or CEMT, you see that the connections are *service* and *acquired*.

To validate the security environment is functioning correctly, you need a transaction that a signed-on user on *cicsA* can run. This transaction needs the ability to issue a DPL request to a program on *cicsB*.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **3270** tab and set the **User ID** field to the user ID of the signed on user. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

### Related information

[Associating a RACF user ID with a certificate](#)

## Design example: Securing client to CICS with a trusted IPIC connection

In this scenario, the client system is IBM z/OS Connect that is co-located in the same sysplex as two CICS regions. This example demonstrates how the client system can assert identities in CICS based on authentication that is performed in the z/OS Connect system. This scenario applies to any client system that supports IPIC connections to CICS TS, including both CICS TG and z/OS Connect.

For more information about configuring this scenario, see the configuration task [Configuration example: Securing client-to-CICS with a trusted IPIC connection](#).

The IPCONN resource in the CICS region specifies USERAUTH(IDENTIFY), LINKAUTH(SECUSER), with the SECURITYNAME attribute set to the CICS region user ID.

The following diagram shows a trusted IPIC connection to two CICS regions in the same sysplex.

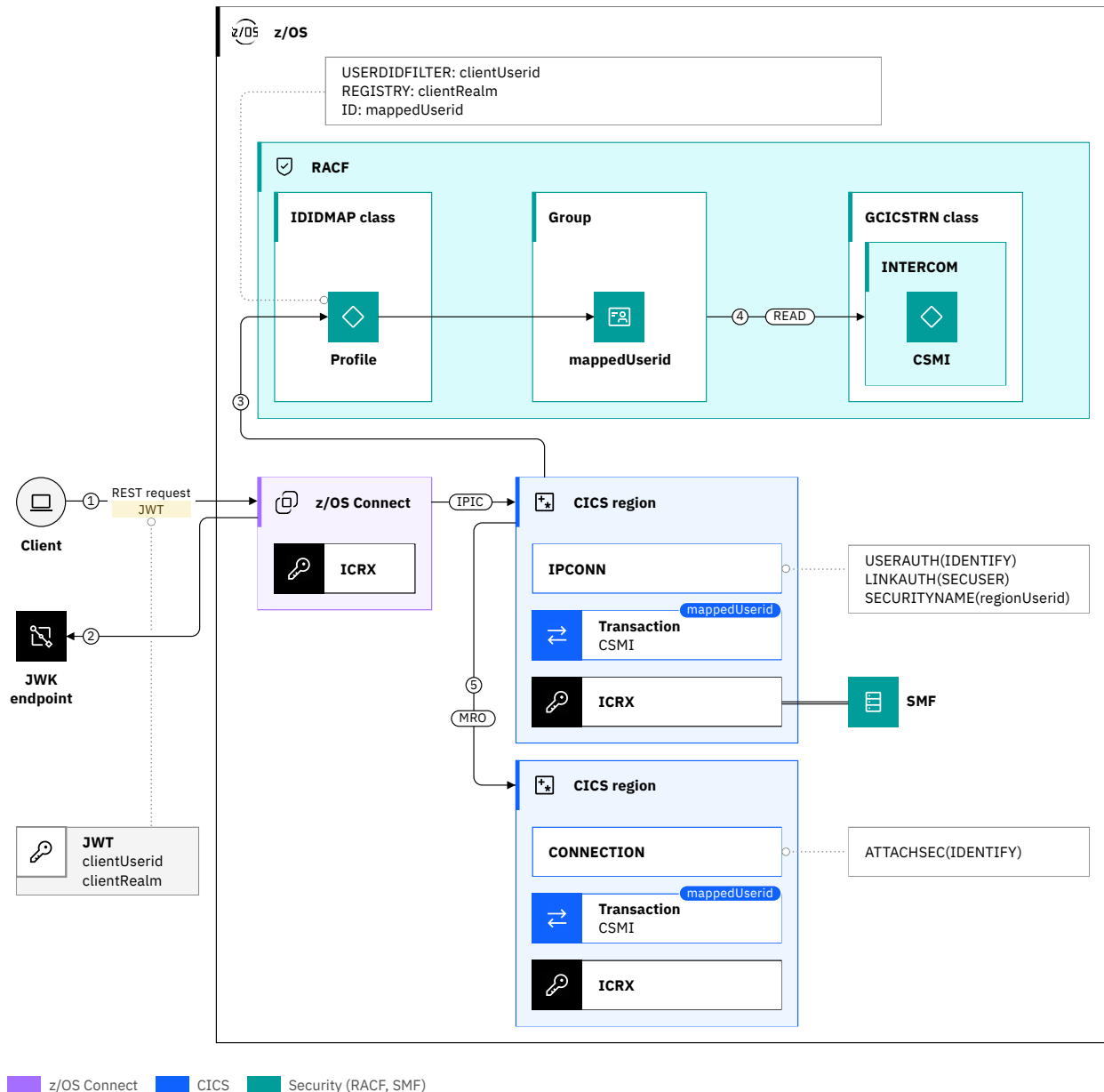


Figure 109. A trusted IPIC client connection to CICS regions

The following security flows occur at the points shown:

1. The client sends a JWT to z/OS Connect. The distributed identity (*clientUserId* and *clientRealm*) is represented by claims in the JWT (normally the subject and issuer claims)
2. z/OS Connect validates the JWT signature by using a JWK (JSON Web Key) endpoint.
3. CICS calls RACF to map the distributed identity to a *mappedUserId* by using an IDIDMAP profile that is created by using the **RACMAP** command. The *mappedUserId* is set as the task user ID.
4. Authorization of the transaction in the CICS regions is performed by checking that the *mappedUserId* has READ authority to the resource profile for the CSMI transaction. The CICS region does this check by calling RACF to validate that the *mappedUserId* belongs to a group that has READ authority to a member list that includes CSMI in the GCICSTRN class. In this case, the INTERCOM member list in the CICS-supplied DFH\$CAT2 sample is used. INTERCOM contains a set of intercommunication related transactions.

5. *mappedUserid* is associated with any additional request that is initiated by the transaction in the CICS regions. For example, by an **EXEC CICS START** command or an outbound request (if the other region requires a task user ID to flow).

Link security is not in force as the link user ID specified in the SECURITYNAME attribute on the IPCONN is set to the CICS region user ID. This configuration means that the link user ID is not used for authorization of CSMI in the CICS regions.

## Configuration example: Securing client-to-CICS with a trusted IPIC connection

Configure a trusted IPIC connection between CICS and z/OS Connect Enterprise Edition supporting distributed identity mapping.

### Before you begin

This configuration task is based on the [Design example: Securing client to CICS with a trusted IPIC connection](#).

You must complete these tasks:

- Configure IBM z/OS Connect to connect to CICS by using a `zosconnect_cicsIpicConnection` element in `server.xml`. For more information, see [Configuring an IPIC connection in z/OS Connect EE](#).
- Configure z/OS Connect to perform JWT authentication and use the identity in the JWT to authorize access to z/OS Connect. For more information, see [How to configure JWT authentication](#).
- Configure z/OS Connect to map the identity in the JWT to a RACF user ID by using a distributed identity filter. For more information, see [Mapping a distributed identity claim to a SAF user ID](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to issue RACF commands.

### About this task

In this example, you learn how to configure CICS to connect to z/OS Connect by using a hardcoded IPCONN definition.

This task assumes the following definitions:

- *ipconnA* is the name of the IPCONN definition.
- *regionUserid* is the region user ID for the CICS region.
- *portA* is the TCP/IP port number that is used to listen for IPIC connections from z/OS Connect.
- *zceeAppl* is the APPLID specified in the z/OS Connect CICS service provider on the `zosConnectApplid` attribute within the `zosconnect_cicsIpicConnection` element.
- *zceeNetid* is the network ID specified in the z/OS Connect `zosConnectNetworkid` attribute within the `zosconnect_cicsIpicConnection` element.
- *nSess* is the number of sessions defined.
- The distributed identity flowed from z/OS Connect is specified in X.509 form as:  
CN=clientUserid,O=IBM,C=US.
- *mappedUserid* is the RACF user ID that is mapped from the distributed identity.
- *taskUserGroup* is the RACF group that contains the user IDs, such as *mappedUserid*, that are allowed to run transactions on *cicsB*.

## Procedure

### 1. Set up IPIC connection with TLS.

- a. In the CICS region, define the TCPIP SERVICE that listens for IPIC connection requests from z/OS Connect. The value URM(NO) is used to disable connection autoinstall.

```
DEFINE TCPIP SERVICE(tcpiPA) GROUP(groupA) URM(NO) PORTNUMBER(portA) PROTOCOL(IPIC)
HOST(ANY)
```

- b. In the CICS region define a one-way IPIC connection definition for z/OS Connect, naming the TCPIP SERVICE defined in Step [“1.a” on page 225](#) and the APPLID and network ID used by the z/OS Connect client.

```
DEFINE IPCONN(ipconnA) GROUP(groupA) APPLID(zceeAppl) NETWORKID(zceeNetid) PORT(NO)
TCPIPS(tcpiPA)
RECEIVECOUNT(nSess) SENDCOUNT(0) LINKAUTH(SECUSER)
SECURITYNAME(regionUserid) USERAUTH(IDENTIFY)
```

- c. Install *groupA* in the CICS region.

### 2. Setup distributed identity mapping in RACF.

- a. Activate the RACF IDIDMAP class to allow distributed identity filters to be created or changed.

Enter the following RACF command:

```
SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
```

- b. Create an IDIDMAP profile using the RACMAP command to map the distributed identity flowed from z/OS Connect to a RACF user ID.

Enter the following RACF command:

```
RACMAP ID(mappedUserid) MAP
USERIDFILTER(NAME('CN=clientUserid,O=IBM,C=US'))
REGISTRY(NAME('*')) WITHLABEL('ZCEE-LABEL')
```

- c. Refresh the IDIDMAP RACF class for the changes to take effect.

```
SETROPTS RACLIST(IDIDMAP) REFRESH
```

3. Add the mirror transaction (CSMI) to the INTERCOM profile group, and grant permission to run the CSMI transaction to users in the *taskUserGroup*, then refresh the RACF class.

```
RDEFINE GCICSTRN INTERCOM ADDMEM(CSMI) UACC(NONE)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

4. Send a request from the z/OS Connect client to CICS to install the IPCONN and link to the target CICS program.

## Results

To check that the IPIC connection is active, issue the command **CEMT INQUIRE IPCONN(*ipconnA*)** and validate that the resource is marked as inservice and acquired.

You should see the following CICS messages in the CICS region's MSGUSR log when the first request from z/OS Connect arrives.

```
DFHIS2012 03/11/2022 16:53:10 IYK2Z32A The connection status in IPCONN ipconnA from applid
zceeAppl is changed from RELEASED to OBTAINING due to a SPI command to ACQUIRE the IPCONN issued
on the remote system.
```

```
DFHIS2001 03/11/2022 16:53:10 IYK2Z32A Client web session 1 from applid zceeAppl accepted for
IPCONN ipconnA.
```

```
DFHIS2012 03/11/2022 16:53:10 IYK2Z32A The connection status in IPCONN zceeAppl from applid
ipconnA is changed from OBTAINING to
ACQUIRED .
```

```
DFHIS2001 03/11/2022 16:53:10 IYK2Z32A Client web session 2 from applid zceeAppl accepted for
IPCONN ipconnA.
```

To validate the security environment is functioning correctly, you need a program that the mapped user ID has authority to run.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions** view in focus, select the **Add Security Request Recording** pop-up menu option. In that window, select the **IPIC** tab and set the **Appl ID** to the *zceeAppl* from the z/OS Connect configuration. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

To check the original distributed identity that mapped to the *mappedUserid*, you can use the Task Association fields for Distinguished Name and Realm that are available in the following locations:

- **Task Association** view in CICS Explorer.
- DNAME and REALM returned by the **INQUIRE ASSOCIATION** command.
- Identity class data in the CICS monitoring data. For more information, see [Identity class data: Listing of data fields](#).

## Configuring security for IPIC

Before you configure security for IPIC, make sure that you are familiar with the information in [Security for IPIC \(IP interconnectivity\)](#) and [Designing security for IPIC](#).

The example configuration tasks listed here are based on their corresponding design examples.

- [Configuration example: Securing CICS-to-CICS with an IPIC connection within a sysplex](#)
- [Configuration example: Securing CICS-to-CICS with an IPIC connection that uses TLS](#)
- [Configuration example: Securing client-to-CICS with a trusted IPIC connection](#)

Other configuration tasks are found here:

## Installing a resource with a user ID attribute

If a resource member contains a user ID, such as the USERID option of the SESSION definition, and the group containing the resource is installed manually, it is necessary to have surrogate authority to permit this.

### Before you begin

You must have:

- Authorization to define RACF commands.
- Authorization to update the SURROGAT class.

### About this task

This task assumes the following definitions:

- *installerUserid* is the user ID of the person who installs the resource
- *installerGroup* is the group of people sharing the same role who are allowed to install the resource
- *resourceUserid* is the user ID on the resource definition

*installerUserid* is member of *installerGroup*

### Procedure

Allow installers in *installerGroup* to be able to install a resource with user ID *resourceUserid*:

```
RDEFINE SURROGAT resourceUserid.DFHINSTL UACC(NONE)
PERMIT resourceUserid.DFHINSTL CLASS(SURROGAT) ID(installerGroup) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

## Results

Installers will now be able to manually install resources with the user ID *resourceUserid*.





---

## Chapter 14. Security for LU6.1 and LU6.2 connections

Information about security for LU6.1 and LU6.2 connections is not included in this version of the documentation.

Information about security for LU6.1 connections is available in previous versions: [Implementing LU6.1 security](#).

Information about security for LU6.2 connections is available in previous versions: [Implementing LU6.2 security](#).



---

## Chapter 15. Security for Java applications

A JVM server can run Java applications that are accessed in a variety of ways, for example a CICS Java program that is linked to from another CICS program, a web application deployed into a Liberty JVM server using HTTP, or a JMS application that uses a message driven bean (MDB).

Java programs in CICS have the same security facilities as other high-level languages and can use CICS transaction and resource security to provide for identification, authorization, and audit. For a description of these terms, see [What does security mean in CICS?](#)

If you run Java applications in a Liberty JVM server, because Liberty has its own security model, you can configure additional security directly in the Liberty server and in the Enterprise Java application deployment descriptor to provide authentication and Java EE based authorization. Different options or combinations of capabilities can be used depending on the types of application to be run in a Liberty JVM server.

---

### How it works: CICS Liberty security

CICS Liberty security is based on the standard Enterprise Java security that is provided by Liberty, which is then integrated with the CICS security infrastructure. You can use all the standard components of CICS security, for example, transaction security and resource security within CICS Liberty. This support gives you a wide variety of options for authentication and authorization. To understand how CICS Liberty security works, it is useful to understand the components involved.

#### Components of Liberty security

The key components of security in a Liberty JVM server are shown in [Figure 110 on page 232](#):

- The user registry, which provides identification and authentication support.
- The angel process, which provides access to authorized System Authorization Facility (SAF) registry services.
- The CICS Liberty security feature, which integrates Liberty and CICS security.
- The Liberty HTTP listener, which provides TLS support using the Java™ Secure Sockets Extension (JSSE) framework. As an alternative, z/OS Communications Server AT-TLS can be used in place of the JSSE TLS support.

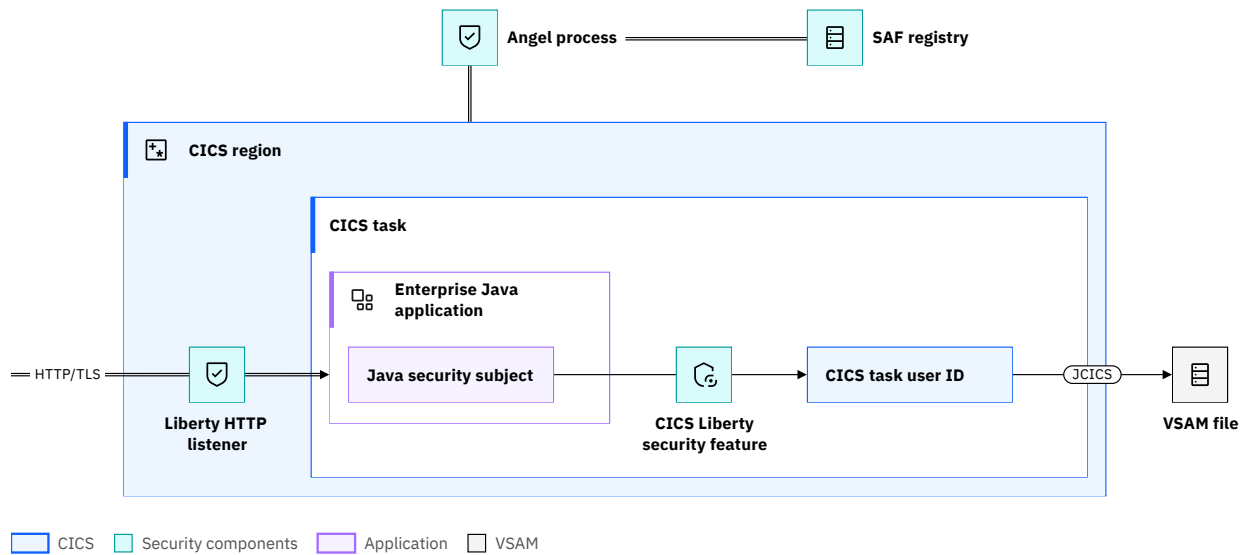


Figure 110. Key components of CICS Liberty security

These components correspond to the security principles described in [What does security mean in CICS?](#):

Security principles	Security components
Identification and authentication	<a href="#">“User registry” on page 232</a> <a href="#">“Angel process” on page 233</a>
Authorization	<a href="#">“CICS Liberty security feature” on page 233</a>
Integrity and confidentiality	<a href="#">“Transport Layer Security (TLS)” on page 233</a> <ul style="list-style-type: none"> <li>• <a href="#">“JSSE” on page 233</a></li> <li>• <a href="#">“Keystores and truststores” on page 235</a></li> <li>• <a href="#">“AT-TLS” on page 235</a></li> </ul>

## User registry

User registries, also known as security registries, store information about users and groups that can be used for authentication and authorization. CICS Liberty supports the following types of user registry.

### SAF registry

[RACF](#) for z/OS is a SAF registry. The angel process is used by Liberty to handle security calls to the SAF registry. For more information about the angel process, see [How it works: Liberty angel process](#).

For more information about configuring a SAF registry, see [Configuring Liberty to use a SAF user registry](#).

### Lightweight Directory Access Protocol (LDAP) registry

LDAP security registries are widely used outside of z/OS to store enterprise security information. It is possible for an LDAP registry to also be used by a Liberty JVM server.

For more information about configuring an LDAP registry, see [Configuring Liberty to use an LDAP user registry](#).

## Basic registry

The basic registry is a simple file-based registry that is provided by the Liberty appSecurity feature. Users, passwords, and groups are defined in the `server.xml` configuration file.

**Note:** Access to the basic registry information stored in `server.xml` is controlled only by zFS permissions. It is not advisable to use this registry type for any purpose other than testing in development environments.

For more information about configuring a basic registry, see [Configuring a basic user registry for Liberty](#).

## Custom user registry

A custom registry can be created by using either the Enterprise Java security API, and a [database identity store](#), a [custom identity store](#), or [using the Liberty custom user registry support](#).

## Angel process

The [angel process](#) is a long-running started task that allows Liberty servers to use z/OS authorized services. It is required if a Liberty JVM server in CICS needs to perform authentication and authorization against the RACF registry.

For recommendations and more information about how the angel process works for a CICS Liberty server, see [How it works: Liberty angel process](#).

For more information about configuring the Liberty angel process, see [Configuring the angel process](#).

To assist with managing permissions and also to authenticate and authorize Java application users, see [RACF profiles used by the Liberty angel process](#).

## CICS Liberty security feature

The CICS Liberty security feature (`cicsts:security-1.0`) enables the integration of Java security in Liberty with CICS security. To use the feature, you must also configure a user registry. When used for web applications this feature can assert the authenticated Java security subject from Liberty as the CICS task user ID, or assign the matched URIMAP user ID as the CICS task user ID. When used for Link to Liberty the CICS task user ID is asserted as the Java security subject when the CICS program calls the Java application.

The CICS Liberty security feature pulls in the required combinations of the `zosSecurity`, `appSecurity`, and `servlet` features to support the Enterprise Java level implied by the current `server.xml` configuration.

For more information about how to configure the CICS security feature, see [Configuring security for CICS Liberty](#).

## Transport Layer Security (TLS)

You can secure communications between an HTTP client or JMS (Java Message Service) client and a Liberty JVM server by using the [TLS](#) protocol. TLS provides transport layer security that includes confidentiality, integrity, and authentication to secure the connection between a client and a Liberty JVM server.

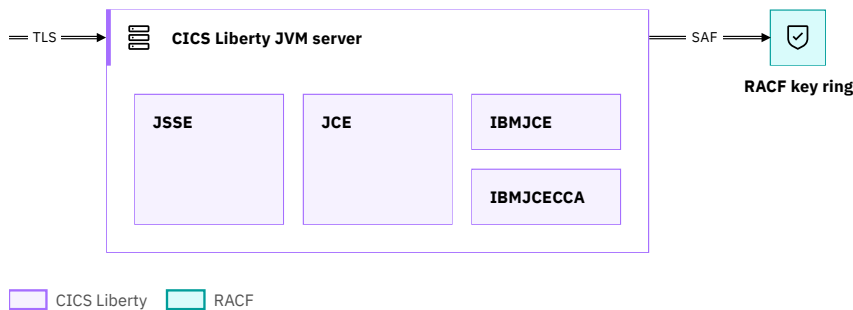
To enforce TLS use for a web application, you can add a security constraint in the application's `web.xml` `<transport-guarantee>CONFIDENTIAL</transport-guarantee>`. This configuration ensures all requests that match the constraint pattern are automatically redirected to the HTTPS endpoint.

## JSSE

CICS Liberty uses Java™ Secure Sockets Extension (JSSE) as the TLS implementation for secure connections. JSSE provides a framework and Java implementation that handles the handshake negotiation and protection capabilities that are provided by TLS.

To configure TLS with CICS Liberty, you must define the Liberty Transport Security 1.0 feature (transportSecurity-1.0) to the `server.xml` and then, use the `<ssl/>` and `<keyStore/>` elements to define the required settings. For more information about configuring JSSE with CICS Liberty, see [Configuring TLS for a Liberty JVM server by using RACF](#).

CICS Liberty TLS support uses the underlying JSSE and Java Cryptography Extension (JCE) frameworks that are provided as part of Java Standard Edition (Java SE), as shown in [Figure 111](#) on page 234. This support can be integrated with the RACF user registry to control access to digital certificates that are stored in RACF.



*Figure 111. TLS support in Java that uses JCE and JSSE*

The JSSE API provides a framework and Java implementation of the TLS protocol that is used when CICS Liberty is configured to use HTTPS. The JSSE API is provided in the `javax.net` and `javax.net.ssl` packages of Java SE.

**Important:** IBMJSSE is IBM's Java implementation of TLS and so does not use the facilities of System SSL as used by CICS web support and CICS web services. Configuration of TLS in CICS Liberty is achieved by setting properties in the [JVM profiles](#) and the Liberty server configuration file (`server.xml`).

Java Cryptography Encryption (JCE) is a standard extension to the Java Platform that provides the underlying implementation for cryptographic services, including encryption, key generation, and Message Authentication Codes (MAC).

The IBM Java SDK for z/OS V8 includes the IBMJCE, IBMJCEPlus, and IBMJCECCA JCE providers. Both IBMJCE (the default) and IBMJCEPlus automatically detect and uses the CP Assist for Cryptographic Function (CPACF), which provides improved performance for symmetric key encryption (for example, AES and 3DES) and hashing algorithms (for example, SHA1 and SHA2). The IBMJCEPlus provider is intended to supersede IBMJCE and supports newer algorithms, some of which are required for TLS 1.3 support. For more information, see [Security guide in IBM SDK Java Technology Edition documentation](#). IBMJCECCA is an alternative provided that is integrated with the z/OS Integrated Cryptographic Service Facility (ICSF) and can use the IBM Z Crypto Express cards to assist with asymmetric key encryption, during TLS handshakes, for example. For instructions on using IBMJCECCA with IBM Java SDK for z/OS V8, see [Enabling hardware cryptography for Liberty for z/OS using Java 8](#).

The IBM Semeru Runtime Certified Edition for z/OS V11 includes the OpenJCEPlus and IBMZSecurity providers by default. The OpenJCEPlus provider replaces the IBMJCEPlus provider and the IBMZSecurity provider is used to provide the JCERACFKS keystore implementation that was previously provided by the IBMJCE provider. For more information, see [Security guide in IBM Semeru Runtime Certified Edition for z/OS documentation](#). For additional instructions on using IBMJCECCA with IBM Semeru Runtime Certified Edition for z/OS V11, see [Enabling hardware cryptography for Liberty for z/OS using Java 11](#).

The use of JCE providers is controlled by editing the list of security providers in the `java.security` file for the JVM.

## Keystores and truststores

Keystores and truststores are repositories that contain certificates and private keys that are used for cryptographic operations, for example, when you use TLS or validating signatures in third-party authentication tokens.

A *keystore* contains personal certificates, plus the corresponding private keys that are used to identify the owner of the certificate. For TLS, a *personal certificate* represents the identity of a TLS endpoint. Both the client (for example, a REST client) and the server (for example, a Liberty JVM server) might have personal certificates to identify themselves.

A *truststore* contains signer certificates (also known as certificate authority certificates) which the endpoint trusts. A *signer certificate* contains a *public key*, which is used to validate personal certificates. By installing the server's signer certificate into the client's truststore, you are allowing the client to trust the server when it establishes a TLS connection. The same principle is true for a server to trust a client when TLS client authentication is enabled.

CICS Liberty supports SAF key rings, Public Key Cryptography Standards #12 (PKCS12), and Java™ keystores (JKS).

For more information about configuring CICS Liberty to use a SAF key ring for TLS, see [Configuring TLS for a Liberty JVM server by using RACF](#). For more information about configuring CICS Liberty to use a JKS for TLS, see [Configuring confidentiality and integrity](#).

## AT-TLS

As an alternative to the CICS Liberty TLS support that uses JSSE, you can use Application Transparent Transport Layer Security (AT-TLS), a capability of z/OS Communications Server, for transport layer security with CICS Liberty. For more information about AT-TLS, see [Implementation options for TLS](#).

**Important:** When you use AT-TLS with CICS Liberty, CICS Liberty acts as a basic application, which means that it is unaware that AT-TLS is performing encryption and decryption of data. You do not configure an HTTPS port, keystore, or truststore for the Liberty JVM server.

## How it works: Liberty angel process

The [angel process](#) is a long-running started task that allows Liberty servers to use z/OS authorized services. It is required if a Liberty JVM server in CICS needs to perform authentication and authorization against the RACF registry.

On z/OS, access to the SAF registry is considered an authorized service. When the CICS security domain makes calls to SAF, it can use its SVC routine, DFHCSVC, which is loaded from the authorized LPA to make authorized calls to SAF. This option is not available to Liberty JVM servers because the CICS SVC is a private interface. Instead, it uses a program call (PC) instruction to another address space, which is itself authorized. The angel process is configured to run authorized, and Liberty servers can connect to it to call authorized services (see [Figure 112 on page 236](#)). When a Liberty server is run in a CICS region, the angel process is the only way for the Liberty server to authenticate users with the SAF registry. The access to its services is controlled by the SAFCREDS resource profiles.

**Important:** Liberty normally offers the ability to fail over to unauthorized z/OS services to authenticate requests when the angel process is disabled. If you use a Liberty server in CICS, this function is not supported because CICS does not support the use of program-controlled libraries.

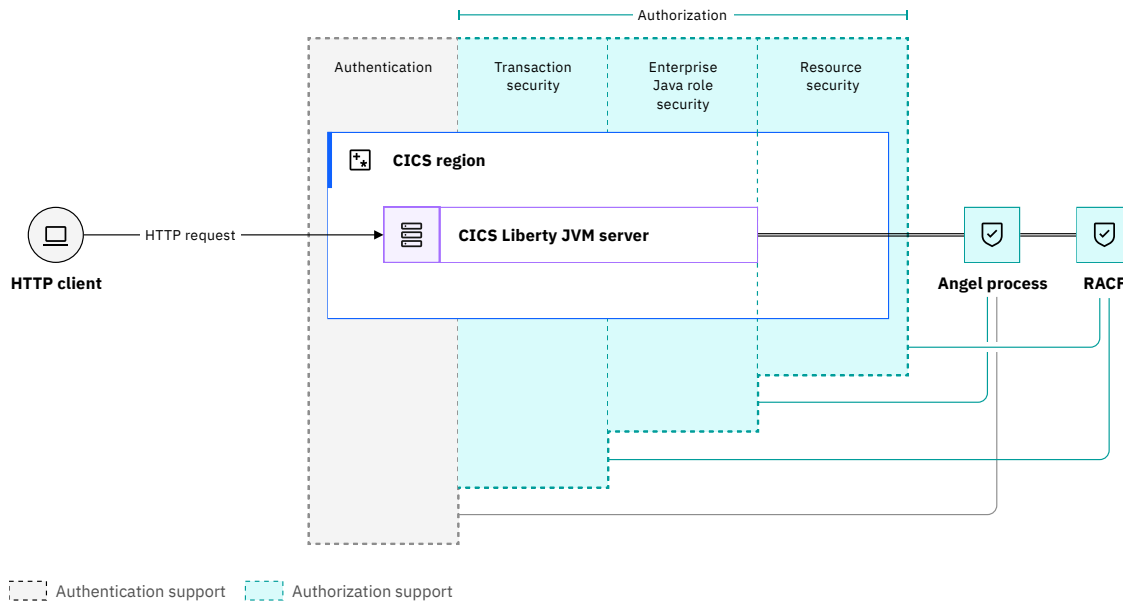


Figure 112. How Liberty servers call the angel process

## Angel process connectivity

A Liberty server can connect to only one angel process at server startup. However, multiple Liberty servers can share the angel process, independent of the level of code that the servers are running. By default, the angel process is unnamed, and all Liberty servers will connect to the default unnamed angel process. However, you can use [named angels](#) to achieve more flexible configuration, including sharing an angel process across a selection of Liberty servers.

An angel process can only service Liberty servers that run on the same LPAR. Each LPAR can have multiple named angel processes but only one default angel process.

## Named angel process

Liberty Fix pack 16.0.0.4 introduced the notion of a *named angel* that allows multiple uniquely named angel processes to run on a single z/OS system, in addition to the default unnamed angel process. A named angel has the same function as the default angel process but it can be used for a selected group of Liberty servers. This isolates different product stacks or Liberty installations from one another so that they can run different service levels or be managed independently.

**Best practice:** It is recommended to use a single named angel process for all the Liberty servers that run inside the same CICS region. This named angel process can also be shared by other Liberty servers within the z/OS LPAR, unless further isolation is required. This condition occurs regardless of the level of code that the Liberty servers are running or whether they are running in a CICS Liberty JVM server.

For more information about named angel processes, see [Named angel](#).

## Angel process version and interoperability

Liberty servers that are running on a z/OS image can share a single angel process, regardless of the level of Liberty code that the servers are using.

**Best practice:** It is best practice to upgrade the level of the angel process before you upgrade the servers that use it. Upgrading in that order ensures continued support for older Liberty servers while continued access is provided to the latest authorized services and functions.



**Important:** You can choose to run a different version of the angel process that is bundled with another product rather than the one bundled with CICS Liberty. However, no matter which product the angel process is bundled with, it must be equal to or later than the version of CICS Liberty. Therefore, if the angel process that is bundled with the other product is earlier than that bundled with CICS Liberty, you must use the angel process bundled with CICS Liberty.

You can identify the Liberty version of the angel process and the Liberty server that's running in CICS as shown in [Managing a Liberty angel process](#).

For more information on the Liberty angel process usage with CICS, see [Using the Liberty angel to access authorized z/OS services](#).

## How it works: Securing Liberty web applications

For web applications accessed by HTTP requests, the web client can be authenticated by using one of a range of authentication options. Access to the web application can then be controlled by using role authorization, and access to CICS resources can be controlled by using CICS transaction and resource security.

The following diagram summaries web authentication and authorization options available with CICS Liberty. You can click each option to learn more.

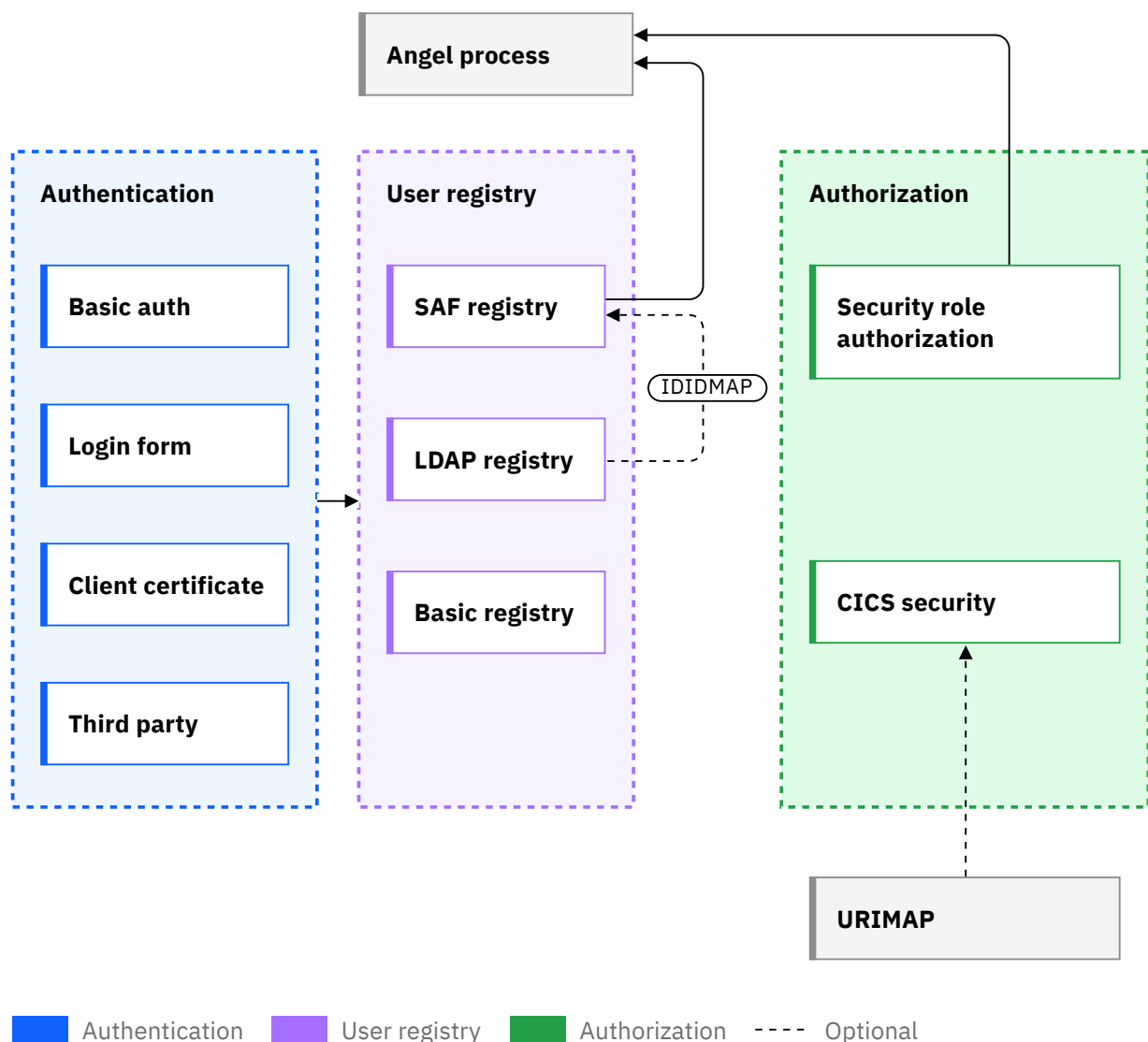


Figure 113. Web authentication and authorization options for CICS Liberty

Options in the diagram are also available with related links in the following table.

Authentication	User registry	Authorization	External components
<ul style="list-style-type: none"> <li>• <a href="#">“Basic authentication” on page 239</a></li> <li>• <a href="#">“Form-based authentication” on page 240</a></li> <li>• <a href="#">“Client-certificate authentication” on page 240</a></li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">Configuring Liberty to use a SAF user registry</a></li> <li>• <a href="#">Configuring Liberty to use an LDAP user registry</a></li> <li>• <a href="#">Configuring basic authentication for Liberty web applications</a></li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">“Security role authorization” on page 244</a></li> <li>• <a href="#">“CICS transaction and resource security” on page 244</a></li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">Angel process</a></li> <li>• <a href="#">URIMAP</a></li> </ul>

Authentication	User registry	Authorization	External components
<ul style="list-style-type: none"> <li>• <a href="#">“Third-party authentication” on page 240</a></li> </ul>			

## Authenticating users to run CICS Liberty web applications

CICS Liberty supports the following authentication mechanisms: basic, form-based, client-certificate, and third-party authentication.

For basic authentication, form-based authentication, and client-certificate authentication, the web application attempts authentication only if a security constraint is specified in its application deployment descriptor (`web.xml`). Third-party authentication is typically configured in the Liberty server configuration file `server.xml`.

**Note:** You can override the authentication method to be used for all applications, or specify which authentication mechanism to use if client authentication fails, with the `webAppSecurity` element in the `server.xml` configuration file of the Liberty JVM server.

For more information about the `webAppSecurity` element, see [Web Container Application Security \(webAppSecurity\) in Liberty](#).

**Note:** If you use `RequestDispatcher.forward()` methods to forward requests from one servlet to another, the security check occurs only on the first servlet that is requested from the client.

- [Basic authentication](#) requires the web client to provide a user ID and password.
- [Form-based authentication](#) requires the application to send an HTML form to the web client, into which the user enters their credentials.
- [Client-certificate authentication](#) requires the web client to provide a certificate that can be validated and associated with a user ID.
- [Third-party authentication](#) requires user to authenticate with a trusted third party that generates an authentication token to be sent to CICS Liberty.
- [“Single Sign-On” on page 241 \(SSO\)](#) can be implemented with each of the above authentication options. It enables users to log in to one server and access applications on other servers without getting prompted to log in again.
- [“Custom authentication options” on page 242](#) enable use of programmatic options for authentication.
- [“Authentication cache” on page 244](#) offers the option to store the subject in an authentication cache to alleviate the performance impact due to security.

## Basic authentication

Basic authentication is a simple authentication scheme that is built into the HTTP protocol. It requires the client to provide a user ID and password when a request is made. The user ID and password are encoded in base64 and sent in the HTTP Authorization header of the request. The Liberty JVM server validates the user ID and password against a configured user registry. The user ID is set as the authenticated user.

The following types of user ID and password are supported for basic authentication with CICS Liberty:

- A user ID and password that is defined in a basic registry.
- An LDAP distinguished name and password, which is defined in an LDAP registry.
- A SAF user ID and password (or passphrase) defined in the SAF registry on the same LPAR as the Liberty server.

Basic authentication encodes but does not encrypt the credentials. You must use HTTPS (TLS) to ensure confidentiality.

Specify the following authentication method in `web.xml` when you want to enable basic authentication for the application:

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

For more information about configuring basic authentication, see [Configuring basic authentication for Liberty web applications](#).

## Form-based authentication

Use form-based authentication when you need to create a customized login form for users to access your application from a browser. You must specify a login page that contains the form to obtain the credentials as well as an error page that displays when login authentication fails.

Specify the following authentication method in `web.xml` when you want to enable form-based authentication for the application:

```
<login-config>
  <auth-method>FORM</auth-method>
</login-config>
```

## Client-certificate authentication

Certificate-based client authentication maps information that is provided in the client's TLS certificate to an associated user ID. It also provides all the normal benefits that are associated with a secure TLS connection.

When client-certificate authentication is configured, for each HTTPS connection, the Liberty JVM server asks the client to provide its certificate. The server validates the chain of trust by checking that the client certificate issuer is in the truststore. This process is standard TLS behavior and if the client certificate is successfully validated, the connection can be established to the Liberty JVM server.

If the client certificate is successfully validated, it is then mapped to a user identity in the user registry.

Specify the following authentication method in `web.xml` when you want to enable client-certificate authentication for the application:

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

For more information about configuring client-certificate authentication with RACF, see [Configuring TLS for a Liberty JVM server by using RACF](#).

## Third-party authentication

### JWT

Third-party authentication can be implemented with CICS Liberty by using a range of different authentication tokens. The most widely used third-party authentication token is the [JSON Web Token \(JWT\)](#).

[Figure 114 on page 241](#) shows a scenario where:

1. A user connects to an intermediate server.
2. The intermediate server authenticates the user with a third-party authentication server.
3. The third-party authentication server creates an authentication token, such as a JWT, which is returned to the intermediate server.
4. The intermediate server then sends this token in the request to CICS Liberty, which validates the token and uses the user identity in the token for authorization. You might also want to map the distributed identity in the token to a user ID in the RACF user registry used by CICS.

5. CICS authenticates the user.

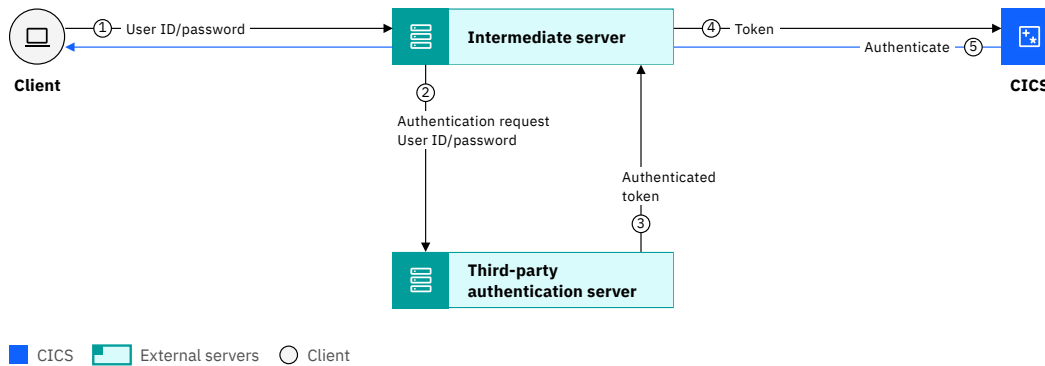


Figure 114. Third-party authentication

The claims in a JWT are encoded as a JSON object and are normally digitally signed with a Message Authentication Code (MAC), which provides two functions:

1. The MAC verifies that the issuer of the JWT (the authentication server) is who it says it is.
2. The MAC ensures that the token remains unchanged between creation on the authentication server and receipt at the Liberty JVM server.

Third-party authentication tokens can also be used as part of a Single Sign-On (SSO) solution (see [Single Sign-on](#)).

**Note:** Client-certificate authentication is often used in this model to establish trust. The client certificate of the intermediate server is used to establish the connection to the Liberty JVM server. The user identity in the token is used as the security principal for the request and for authorization processing.

### OpenID Connect

[OpenID Connect](#) is a simple identity layer built on top of the [OAuth 2.0](#) protocol, which is an open standard for access delegation. Implementing third-party authentication using OpenID Connect allows applications to verify the identity of the user based on the authentication that is implemented by an OpenID Connect Provider. For more information about using OAuth 2.0 with CICS Liberty, see [Configuring OAuth 2.0 for authorization](#).

OpenID Connect avoids the need to write custom authentication code when using a JWT.

CICS Liberty supports OpenID Connect and can play a role as a Relying Party (RP), OpenID Connect Provider (OP), or Resource Server (RS). In most cases, Liberty plays the role of the RS, receiving a JWT in an HTTP Authorization request header field. Users authenticate with an OP and then all back-end services (including CICS Liberty services) are accessed with a trusted JWT.

To configure a Liberty JVM server to accept a JWT as a third-party authentication token, you use the OpenID Connect Client Liberty feature (**openidConnectClient-1.0**). For more information, see [Configuring JWT authentication](#).

If JWT authentication is only required for a subset of requests to a Liberty JVM server, authentication filters can be used to restrict which requests the JWT authentication applies to. Authentication filters specify conditions that are matched against the HTTP request. For more information, see [Authentication Filters](#).

### Single Sign-On

Single Sign-On (SSO) enables users to log in to one server and access applications on other servers without getting prompted to log in again. When a user authenticates on one Liberty server, authentication information that is generated by the server is transported to the web browser in a cookie. The cookie is used to propagate the authentication information to other Liberty servers.

The most commonly used SSO token is the [How it works: LTPA \(Lightweight Third Party Authentication\)](#) token although it is also possible to use a [How it works: JSON Web Token \(JWT\)](#) as an SSO token.

If you use the `cicsts:security-1.0` feature or the `appSecurity-2.0` feature (or a more recent version of this feature) in your Liberty JVM server, Liberty uses LTPA tokens by default in its web authentication processing.

For instructions on how to change the default behavior, see [Configuring LTPA in Liberty](#).

## Custom authentication options

A configuration-only authentication solution is recommended but programmatic options exist that can be used for custom solutions, for example, to support a custom token, user registry, or authentication flow. There are four options to consider:

- [“JWT feature” on page 242](#)
- [“Enterprise Java Security API” on page 242](#)
- [“Trust Association Interceptor \(TAI\)” on page 243](#)
- [“Java Authentication Service Provider Interface for Containers \(JASPIC\)” on page 243](#)

### JWT feature

The Liberty **jwt-1.0** feature provides a set of APIs that you can use to work with JWTs. You can use the feature to build or consume a JWT.

For more information about the JWT feature, see [Building and consuming JSON Web Token \(JWT\) tokens in Liberty](#).

For an example on using the JWT feature with CICS Liberty, see [JWT feature sample with CICS Liberty on GitHub](#).

### Enterprise Java Security API

The JSR 375 specification for Java EE 8 introduced the Java EE Security API 1.0. Jakarta EE 8 offers an identical implementation for Jakarta EE Security API 1.0. Jakarta Security 2.0 in Jakarta EE 9.0 supersedes version 1.0 of the specification for both Enterprise variants. Collectively this function is referred to as the Enterprise Java EE Security API. This API provides portable, plug-in interfaces to enable HTTP authentication in web applications and a standardized API for programmatic security.

In Liberty, `appSecurity-3.0` feature enables Enterprise Java Security API 1.0 and `appSecurity-4.0` feature enables Jakarta Security API 2.0.

You can use the built-in implementations of the plug-in interfaces, or write your own custom implementations.

Using the Enterprise Java EE Security API, you can define all of the security information directly within the application. Declarative security expressed by `web.xml` constraints or annotations, such as `@RolesAllowed`, continue to work as normal.

The three principles of the Enterprise Java EE Security API are as follows:

1. The `HttpAuthenticationMechanism` interface provides an authentication mechanism for the servlet container and offers an alternative to the Java EE function offered by the `<login-config>` element in the web application's deployment descriptor.

There are 3 options that can be used with the `HttpAuthenticationMechanism` interface.

- a. HTTP basic authentication, this is implemented using the `@BasicAuthenticationMechanismDefinition` annotation.
- b. Form-based authentication, this is implemented using the `@FormAuthenticationMechanismDefinition` annotation.
- c. A custom authentication mechanism using a class that implements the `HttpAuthenticationMechanism` interface.

2. An identity store holds usernames, passwords, role membership and other meta-data to allow it to authenticate a caller's credentials. Three identity store types are introduced by the Enterprise Java EE Security API:
  - a. A database identity store, this is implemented using the `@DatabaseIdentityStoreDefinition` annotation. For more information, see [Configuring security by using a database identity store](#).
  - b. An LDAP identity store, this is implemented using the `@LdapIdentityStoreDefinition` annotation. For further details see [Java EE Security API](#).
  - c. A custom identity store using a class that implements the `IdentityStore` interface. For more information, see [Configuring security by using a custom identity store](#).

Liberty also provides the ability to map authenticated identities from an identity store to a SAF user ID using distributed identity mapping. This is configured via the `<safCredentials mapDistributedIdentities="true">` configuration element. If the `cicsts:security-1.0` feature is also installed in a Liberty JVM server then CICS will set the CICS task user ID based on the mapped SAF user ID.

3. The `SecurityContext` class can be used to perform programmatic security checks such as checking access to web resources or checking if the authenticated caller is in a specified role. In this example, the user is forwarded to another page only if they have access to it:

```
@WebServlet("/home")
public class HomeServlet extends HttpServlet
{
    @Inject
    private SecurityContext securityContext;
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        if (securityContext.hasAccessToWebResource("/anotherServlet", "GET"))
        {
            req.getRequestDispatcher("/anotherServlet").forward(req, resp);
        }
        else
        {
            req.getRequestDispatcher("/logout").forward(req, resp);
        }
    }
}
```

For more information about Liberty support for the Enterprise Java security API, see [Java EE Security API](#).

### Trust Association Interceptor (TAI)

A TAI can be used to configure a Liberty JVM server to integrate with a third-party security service. The TAI can inspect the HTTP request to see whether it contains a specific security token, for example, a custom token or a standard token such as a JWT.

For an example on using a TAI with CICS Liberty to validate a JWT, see [Sample Trust Association Interceptor \(TAI\) to validate a JWT with CICS Liberty on GitHub](#).

For more information, see [Configuring TAI for Liberty](#).

### Java Authentication Service Provider Interface for Containers (JASPIC)

JASPIC is an Enterprise Java standard service provider API that enables the implementation of authentication mechanisms into Enterprise Java web applications.

For an example on using a JASPIC implementation with CICS Liberty, see [Sample implementation of JASPIC with CICS Liberty on GitHub](#).

For more information, see [Configuring a Java Authentication SPI for Containers \(JASPIC\) User Feature](#).

## Authentication cache

Because the creation of a Java security subject can affect performance, Liberty provides an authentication cache to store the subject. However, unlike CICS, Liberty does not check for an expired user ID within the cache period. You can configure the cache timeout by using the standard Liberty configuration process. For more information, see [Configuring the authentication cache in Liberty](#).

## Authorizing users to run CICS Liberty web applications

To control users' access to applications that run in a Liberty JVM server, you can leverage one or more authorization methods: CICS transaction and resource security, security role authorization, or operating system thread authorization:

- “[CICS transaction and resource security](#)” on page 244: Use standard CICS transaction and resource security profiles to restrict access to the transaction and its resources that run as part of the web request received by the Liberty JVM server.
- “[Security role authorization](#)” on page 244: In addition to CICS transaction and resource security, you can restrict access to the Java application by using the security roles function provided by Liberty.
- “[Operating system thread authorization](#)” on page 246: You can also control the thread identity that is used for accessing operating system resources by using the Liberty `syncToOSThread` function.

## CICS transaction and resource security

Java applications that run in an integrated-mode Liberty JVM server run as CICS transactions and are therefore subject to CICS transaction and resource security. The `cicsts:security-1.0` feature is used to set the CICS task user ID for each request.

The CICS task user ID that is used to run CICS programs and access CICS resources is determined in the following order:

1. The Java security subject that is set during Liberty application security authentication is used. An example of this scenario is shown in [Design example: Securing a CICS Liberty web application with basic authentication and CICS transaction security](#).
2. If an unauthenticated subject is supplied from Liberty, the user ID that is defined in the matching CICS URIMAP is used. An example of this scenario is shown in [Design example: Securing a web application with a preset URIMAP user ID](#).
3. If no user ID is defined in the URIMAP, the request will run under the CICS default user ID.

By default, HTTP requests that are received by Liberty are run under the CICS Liberty default transaction ID `CJSA`. If you want to use transaction security, define URIMAPs so that requests for different web application URIs run under different transaction IDs. This configuration allows the security profiles that are defined in the SAF resource classes `TCICSTRN` and `GCICSTRN` to be used to authorize access to the web applications.

If transaction and resource security controls are sufficient to control authorization to web applications, you can avoid configuring Enterprise Java authorization. Instead, you grant access to all Java components by using the special subject `ALL_AUTHENTICATED_USERS`. In this case, authorization to the application is controlled entirely by using URIMAPs and CICS transaction security. For more information, see “[Local security roles](#)” on page 245.

## Security role authorization

The authenticated Java security subject can be used by Liberty for authorization of access to Java applications by using security roles. Only members of the defined security role are authorized to access the application or its components. Using role-based security means you can use existing standard Enterprise Java security definitions from another application server.

You have three ways to define these security roles:



1. In the web application's deployment descriptor, specify an `<auth-constraint>` element. This method maps specific URL patterns within the web application to a defined security role.
2. Specify a `@ServletSecurity` annotation with the `rolesAllowed` modifier on either the `@HttpConstraint` or `@HttpMethodConstraint` annotations in a web application. This option is an alternative to Option 1.
3. On an individual Java method, specify the `@RolesAllowed` annotation to define the permissible roles for the specific component. Although this option is primarily for EJB components, if the EJB is also a CDI managed bean and the bean is called from a Servlet or the Link to Liberty (CDI) capability, security role permissions are respected.

#### Notes:

- If no authorization constraint is present in the Java application's deployment descriptor and no security annotations are defined, the Liberty JVM server does not enforce authentication of the Java security subject.
- If both deployment descriptor and security annotations are present for an application, definitions within the deployment descriptor override the annotation settings.

After the roles are defined, user or group membership to a security role is determined by using one of three options:

#### Local security roles

Local security roles are defined in an `<application-bnd>` element. As a result, the mapping is local to the application and takes place outside the SAF registry. You can define the mapping in the `<application>` element of the `server.xml` file. Alternatively, you can configure it in the `ibm-application-bnd.xml` or `ibm-application-bnd.xmi` files if you use an EAR file. Providing you don't configure the `<safAuthorization/>` element in `server.xml`, any RACF EJBROLE profiles and permissions are ignored.

This method is used by default with web applications that are deployed within a CICS bundle. CICS automatically adds the role `cicsAllAuthenticated` and maps it to the special subject `ALL_AUTHENTICATED_USERS`:

```
<application id="com.ibm.cics.server.examples.wlp.tsq.app"
  name="com.ibm.cics.server.examples.wlp.tsq.app" type="eba"
  location="${server.config.dir}/installedApps/
com.ibm.cics.server.examples.wlp.tsq.app.eba">
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</application>
```

By using the special subject `ALL_AUTHENTICATED_USERS` and giving the `cicsAllAuthenticated` role access to all URLs in your web applications deployment descriptor (`web.xml`), you allow any authenticated user ID access to the web application. In this case, authorization to the transaction must be controlled by using CICS transaction security.

If applications are deployed directly into Liberty without a CICS bundle, more complex role mappings can be configured in the `<application-bnd>` element.

If you deploy your application directly to the Liberty dropins directory, by default CICS security is not integrated with Liberty security.

#### SAF role authorization

SAF role authorization is enabled by using the `<safAuthorization/>` element in `server.xml`. When SAF role authorization is active, the `<application-bnd>` no longer acts as the source of user ID to role mapping and if present it will be ignored. Instead, user or group role membership is determined by the Liberty SAF role mapper and corresponding RACF security profiles in the EJBROLE class. You authorize authenticated users to access an application by giving them access to the EJBROLE that is referenced in the `profilePattern` attribute on the `safRoleMapper` element in

server.xml. To access the protected resource, the user or its group must have READ access to the EJBROLE resource profile.

For more information, see [Configuring SAF authorization with an EJBROLE](#).

### Default role to group mapping

If SAF role authorization is not in use and local role mapping information is not provided in an <application-bnd> element, Liberty provides a default role to group mapping that maps the role directly to a group in the user registry. For example, if the role in the <auth-constraint> element that protects the application is defined as MANAGER, any user ID that belongs to the group MANAGER in the local user registry has access to the role and hence to the application.

### Operating system thread authorization

Within a CICS region, the operating system thread identity of any UNIX System Services thread defaults to the CICS region user ID. The CICS region user ID is the identity that is used to authorize access to resources outside of CICS control, such as zFS files or third-party APIs.

When you run a Java application in a Liberty JVM server, you can modify this behavior by using the syncToOSThread function of Liberty. SyncToOSThread enables the Java security subject that is authenticated by Liberty to be synchronized with the operating system thread identity. With syncToOSThread in effect, as long as the SAF registry is in effect, the user's security subject is used to access operating system resources such as zFS files.

For more information, see [Configuring the syncToOSThread function](#).

## How it works: Securing Link to Liberty applications

A CICS program can use the Link to Liberty capability to link to an Enterprise Java, Spring Boot, or CDI application that runs in a Liberty JVM server.

[Figure 115 on page 246](#) shows a view of the security options available with Link to Liberty.

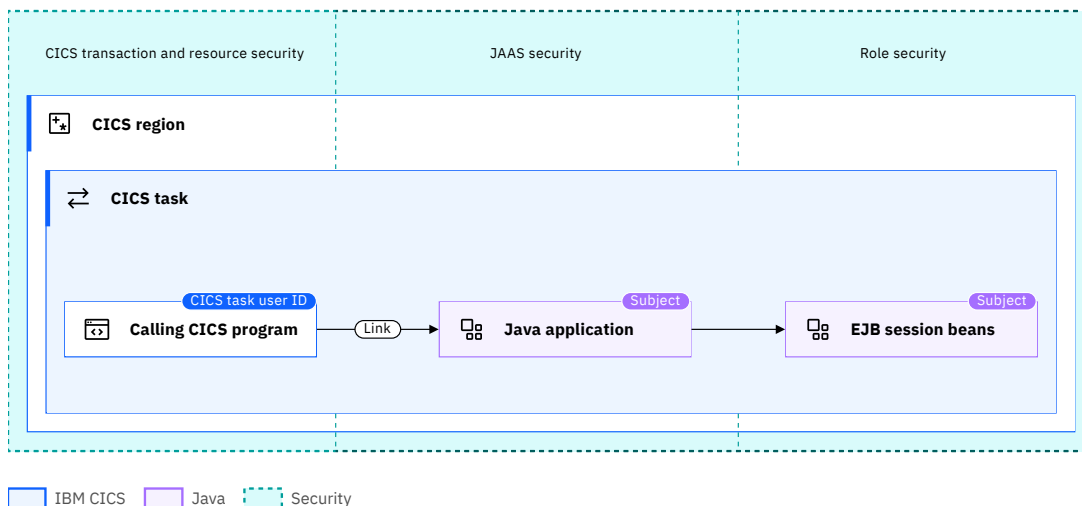


Figure 115. Security options for Link to Liberty

When a CICS program links to a Java application in Liberty, the user ID of the CICS task remains consistent between the calling program and the linked to Java program. This consistency occurs because the linked to program runs within the same CICS task. Access to all CICS resources is authorized by using the CICS task user ID established by the calling program. This configuration means that CICS resource security can be used to control access to the Java application. This access control is achieved by using the CICS PROGRAM resource that is defined or autoinstalled to run the Java application.

If the CICS security feature (`cicsts:security-1.0`) is installed in the Liberty server, the CICS task user ID is asserted as the Java security subject when the calling CICS program links to the Java application. If the CICS task user ID is not present in the Liberty user registry, the Java security subject is set to the Liberty unauthenticated user ID. This condition can occur only if the Liberty server and the CICS region are using different security registries, for instance a RACF registry for CICS and an LDAP registry for Liberty.

## Security role authorization

Because Link to Liberty Java applications do not have a web entry point, they have no means of defining security roles in an `<auth-constraint>` element in the deployment descriptor or using the `@HttpConstraint` annotation as is normally done for Web applications.

Instead the `@RolesAllowed` annotation can be used to authorize access to specific Java methods, based on the authenticated Java security subject. For Link to Liberty entry points the `@RolesAllowed` annotation can be used to authorize access to EJB applications, including CDI beans that are annotated as EJBs. It is not supported for POJO or Spring Boot components.

**Important:** When you link to a Spring Boot application from a CICS program, the CICS task user ID is not passed to Spring security. For more information about integrating Spring security with CICS security, see [Spring Boot Java applications for CICS, Part 2: Security](#).

If security role authorization checks are not required in the Java application, then you can improve performance by bypassing the assertion of the task user ID as the Java security subject. This configuration is enabled by setting the JVM server property `com.ibm.cics.jvmserver.wlp.security.subject.create` to `false`.

## How it works: Securing JCICSX applications

The JCICSX server is a feature of a Liberty JVM server that supports requests to run JCICSX commands from a local development environment. This is useful for remote testing of CICS Java applications. An HTTP connection is established from the JCICSX client in the local development environment to the JCICSX server running in a Liberty JVM server. These connections can be secured by using the standard facilities for authentication, authorization, and confidentiality provided by Liberty.

**When JCICSX feature is not used:** When JCICSX applications are deployed to run within a CICS region without usage of the JCICSX feature, JCICSX adopts an identical security model to that of the corresponding JCICS commands being invoked and the JCICSXUSER role is not used. For more information, see [“How it works: Securing Liberty web applications”](#) on page 237 or [“How it works: Securing Link to Liberty applications”](#) on page 246.

There are two components of JCICSX when deployed in a development environment:

- The `jcicsxClient-1.0` feature that runs in the local development environment and provides the JCICSX client tooling.
- The Liberty `cicsts:jcicsxServer-1.0` feature that runs in a Liberty JVM server and provides the JCICSX server.

[Figure 116 on page 248](#) shows how these components interact.

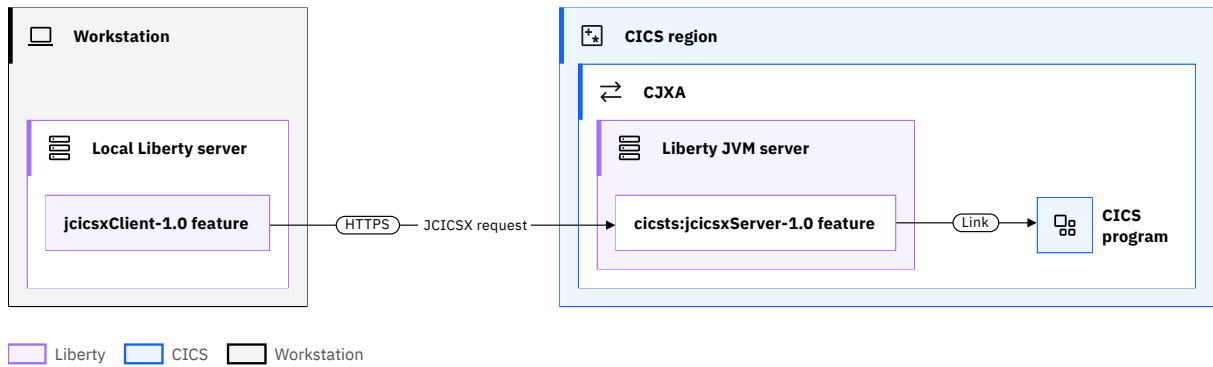


Figure 116. Overview of JCICSX components in a development environment

The JCICSX client tooling uses a set of RESTful API calls to create a long running CICS task by using transaction ID CJXA. All subsequent JCICSX requests from that client will run under this long running CICS task, using the supplied security credentials. The JCICSX server can be configured to use HTTP basic authentication or TLS client certificate authentication. Authorization to run the JCICSX request can use CICS transaction security and security role authorization, and the communication between the client and the remote server can be encrypted by using TLS. [Figure 117 on page 248](#) summarizes the security components that are used by JCICSX.

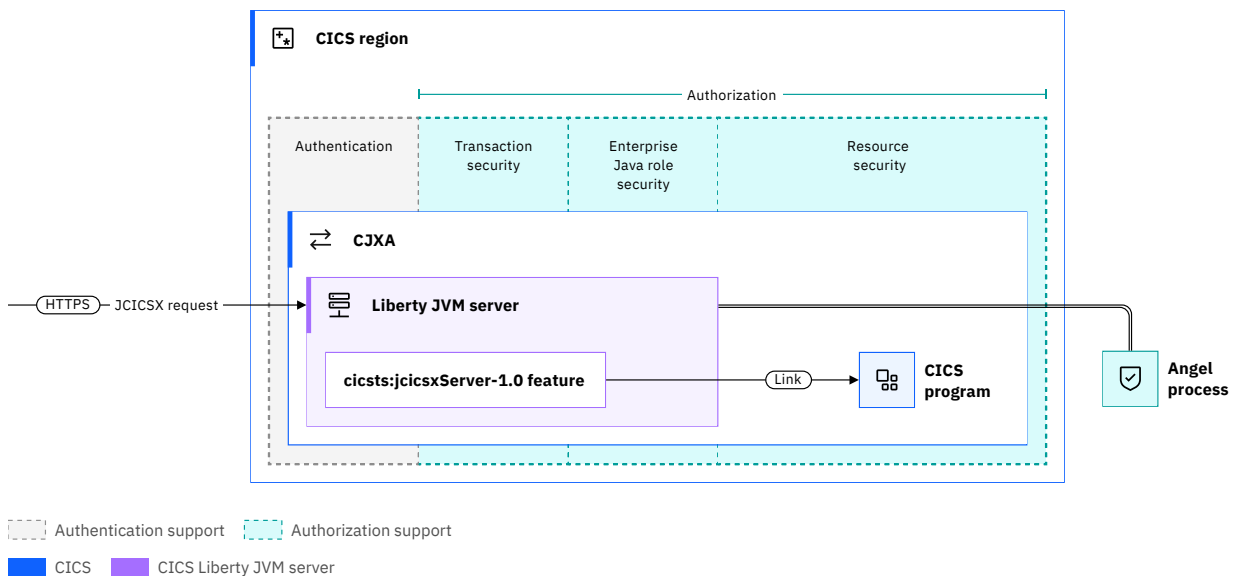


Figure 117. Security components used by JCICSX

## Authentication

When JCICSX requests are sent from the local development environment to the JCICSX server, they are serviced by a dedicated web application provided by the `cicsts:jcicsxServer-1.0` feature. This web application is defined to use the CLIENT-CERT authentication method by default. The Liberty server will attempt to authenticate requests by mapping the client's TLS certificate to a user ID in the configured user registry, such as RACF. However, the JCICSX server can alternatively be configured to fail over to basic authentication. The required credentials must then be defined on the workstation in the local Liberty server configuration file, using the `<usr_jcicsxClient>` element.

## Encryption

The HTTP connection from the local development environment to the JCICSX server can be encrypted by using the TLS support provided by Liberty. This can use either server authentication or mutual authentication in just the same way as other Liberty web applications. For more information, see [“Configuration example: Securing the JCICSX server using TLS client authentication” on page 285.](#)

**Note:** Any TLS support configured for the JCICSX server also applies to any Enterprise Java web applications deployed into the same Liberty server as used by the JCICSX server.

## Authorization

Requests to run JCICSX commands in the Liberty JVM server are run under the CJXA transaction ID. This mapping is defined by the DFHJXSU URIMAP that is dynamically created by the `cicsts:jcicsxServer-1.0` feature. The CJXA transaction can be protected using CICS transaction security in the same manner as any other Liberty web application, as described in [Authorizing users to run CICS Liberty web applications](#). By default, command security and resource security is disabled for CJXA because JCICSX remote development is used for testing. If you want to test security, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. This impacts security settings of other transactions too. For more information, see [Designing security for JCICSX applications](#).

In addition, Enterprise Java role security can be used to restrict access to the JCICSX server's web application. Role security is controlled by using the JCICSXUSER security role defined in the JCICSX web application's `<auth-constraint>` element with access granted to all authenticated users by default.

If SAF role authorization is used by the Liberty server, access can be further restricted by controlling which authenticated users have read access to the EJBROLE profile that is mapped to the JCICSXUSER role and the JCICSX web application ID `com.ibm.cics.wlp.jcicsxserver`. This mapping is controlled by the SAF role mapper as described in [Configuring SAF authorization with an EJBROLE](#). For more information, see [Configuration example: Securing the JCICSX server using basic authentication](#).

**Note:** If SAF role authorization is used by the JCICSX server, it also applies to Enterprise Java web applications that are deployed into the same Liberty server as used by the JCICSX server.

## How it Works: OSGi JVM server security

---

Security for Java applications that are deployed into an OSGi JVM server is based on the standard CICS transaction security and resource security controls.

For more information about CICS transaction security and resource security, see [Transaction security and Resource security](#).

**Note:** Security support for Enterprise Java web applications is not applicable to the OSGi JVM server environment, and only applies to Java applications deployed in a Liberty JVM server. For more information about Liberty JVM server security, see [How it works: CICS Liberty security](#).

Java applications in CICS are accessed by using a CICS program definition. The CICS program definition references the target Java application by using the JVMCLASS attribute. For more information, see [JVMCLASS syntax for Java programs](#).

If the Java application is invoked by using either an **EXEC CICS LINK** command or a 3270 transaction, CICS transaction security provides the authorization of the user ID for all programs that execute within the transaction, including the Java program.

If the Java application is invoked by using an **EXEC CICS START** command, additional surrogate security rules can apply to the starting transaction. For more information, see [Surrogate security](#).

If the transaction that invokes the Java application is started by using an **EXEC CICS RUN TRANSID** command, resource security also applies to the transaction. For more information, see [Security for transactions started with EXEC CICS RUN TRANSID](#).

Further authorization for each Java application can be performed by using CICS resource security on its associated CICS program resource. Checking is performed against the CICS task user ID to ensure it has

authority to run the CICS program and its association Java application. For more information, see [Linking to an OSGi application from a CICS program](#).

## How it works: Restricting access to Java APIs

---

By default, there are no security restrictions on calling Java runtime APIs from Java applications. Sometimes you might want to prevent inadvertent use of certain APIs. For example, the `java.lang.System.exit()` method causes an immediate termination of the JVM and all running applications.

To prevent use of specific APIs within a JVM you can choose from the following two common approaches:

### Code scanning

Code scanning is the practice of analyzing source code to identify issues such as security vulnerabilities and coding errors, and to enforce coding standards. You can use a wide variety of modern tools in combination with a modern development pipeline to restrict the resources and API that a Java application can access.

### Java 2 security

**Deprecation:** Java 2 security has been deprecated by the Java community and will be removed in a future release of Java (see [JEP411](#)). Therefore, it is not recommended for use in the CICS JVM server environment.

Java 2 security provides a policy-based, fine-grain access control mechanism that checks for permissions before allowing access to system resources (such as file I/O or system properties) or specific Java APIs. This function can be enabled at the JVM level by enabling a Java security manager.

## How it Works: Security for new Java threads and tasks

---

Java applications that are running in an OSGi or Liberty JVM server can asynchronously start new CICS-enabled Java threads using a Java executor service. CICS transaction security can be used to authorize access to the new CICS tasks that are using these threads.

For further details on the options for using Java executor services in CICS JVM servers, see [CICS task and thread management](#).

When a new CICS task is started using a suitable `Java ExecutorService`, the task user ID and transaction ID of the calling CICS task is propagated to the new CICS task. However, the transaction ID of the new task can be modified using the `CICSTransactionRunnable` or `CICSTransactionCallable` interface. If the calling thread does not have a CICS task, then the transaction ID and task user ID uses default values.

The following rules summarize this process for threads started in an OSGi JVM server or Liberty JVM server:

1. For threads that are started from a CICS thread, the new CICS task runs under the task user ID from the parent. If the new transaction ID is not specified, it is inherited from the calling CICS task.
2. For threads in an OSGi JVM server that are started from a non-CICS thread, the transaction ID CJSA and default CICS user ID are used for the new thread.
3. For threads in a Liberty server that are started from a non-CICS thread, the default unclassified transaction ID (CJSU) and default CICS user ID are used. These defaults can be modified using the `com.ibm.cics.jvmserver.unclassified.tranid` and `com.ibm.cics.jvmserver.unclassified.userid` JVM server properties. See [JVM system properties](#).

### Security for newly started CICS tasks running on a Java thread

If the new CICS task is started from a CICS-enabled Java thread, CICS transaction security is used to ensure that the calling task's user ID has authorization to attach the new transaction. If late-binding is in effect, this authorization check occurs during the late-binding process.

This is summarized in the following diagram:

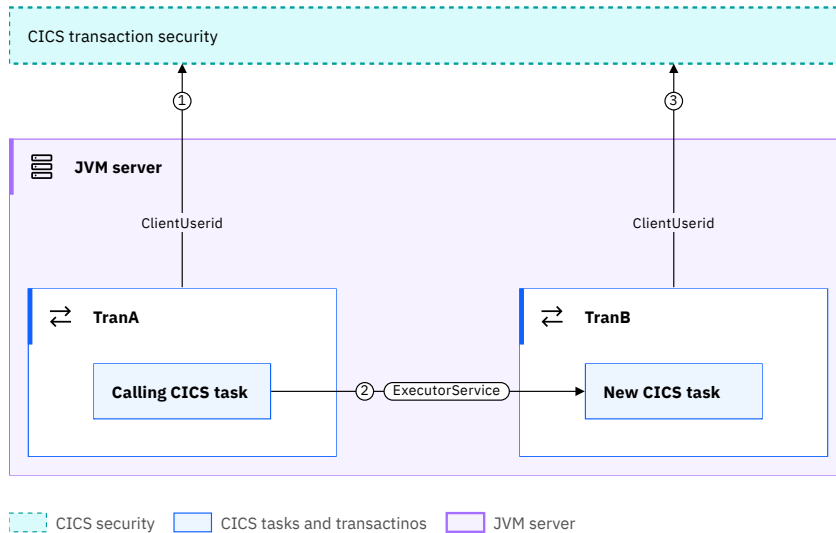


Figure 118. Security for new CICS tasks started from a CICS-enabled Java thread

1. Before starting transaction tranA, CICS performs a transaction security check to verify that clientUserid has authority to attach tranA.
2. The calling CICS task asynchronously starts a new Java thread running on a CICS task by calling the executor service. The transaction ID of the new CICS task is set as tranB by the calling CICS task.
3. Before starting transaction tranB, CICS performs a transaction security check to verify that the clientUserid has authority to attach transaction tranB.

If the new CICS task is started from a non-CICS thread, CICS transaction security is not applicable to the calling thread. The new CICS task will use the CICS default user ID, and the transaction ID will default to CJSU or CJSB depending on what type of JVM server is in use. Transaction security will be used to ensure that the CICS default user ID has authority to attach the specified transaction.

This following diagram shows an example of the non-CICS thread when a Liberty JVM server is in use:

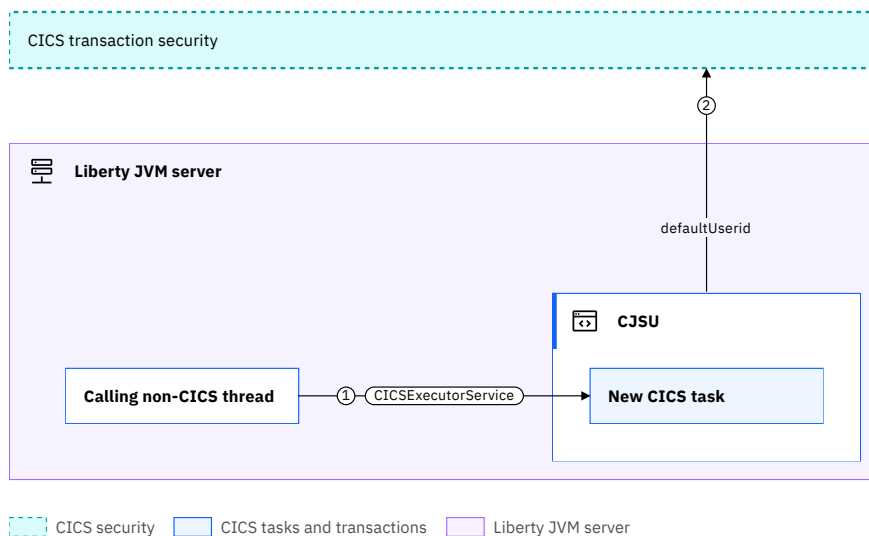


Figure 119. Security for new CICS tasks started from a non-CICS thread

1. The calling Java thread asynchronously starts a new Java thread running on a CICS task by calling the JCICS executor service. The transaction ID of the new CICS task defaults to CJSU as the task is running in a Liberty JVM server.
2. Before starting transaction CJSU, CICS performs a transaction security check to verify that the default CICS user ID has authority to attach transaction CJSU.

## Alternate options

As an alternative to using a Java executor service, the EXEC CICS commands RUN TRANSID and START TRANSID can also be used to asynchronously start new CICS tasks. The new CICS tasks can invoke either Java or non-Java programs. These commands can be called from a Java program using the JCICS `AsyncService` and `StartRequest` classes respectively.

## Related information

[Java threads](#)

[Threads and tasks example](#)

## Designing security for CICS Liberty applications

---

When you are deciding how to secure an application that is running in a Liberty JVM server, you need to consider the following security principals: authentication, identification, authorization, integrity, confidentiality, and audit.

## Designing application security for Liberty with the angel process

Before you use CICS Liberty in production, consideration should be given to security. On the z/OS platform, an angel process provides the integration of Liberty with z/OS authorized services.

The angel process is a started task that allows Liberty servers to use z/OS authorized services. Liberty servers call the angel process when they need to use z/OS authorized services. When run in a CICS region, a Liberty server is able to call the following z/OS authorized services SAFCRE, PRODMGR, ZOSAIO, and for standard-mode with MQ bindings, TXRRS.

**Recommendation:** You are advised to have a Liberty server that is connected to an angel process. The Liberty server can then be authorized to use the selected and relevant z/OS authorized services.

### Which z/OS authorized services should a Liberty server be authorized to use?

The SAFCRE authorized service is required to use the CICS Liberty security feature (*cicsts:security-1.0*). It is recommended to always use the *cicsts:security-1.0* feature in a CICS Liberty server.

This feature can be added automatically to the Liberty server configuration file if the auto-configuration is enabled for the Liberty JVM server and the SEC system initialization parameter is set to YES in the CICS region.

For more information about the *cicsts:security-1.0* feature, see [CICS Liberty security feature](#).

In some cases, the Liberty server can benefit from improved performance. This performance improvement can occur for Java applications that use Link to Liberty when these Java applications do not need to have a Java security subject, for example, for authorization based on security roles. The Liberty server can benefit from improved performance by not creating the Java security subject and by using the Liberty unauthenticated user ID instead. For an example scenario, see [Design example: Securing a Link to Liberty Java application with CICS transaction security](#).

**Recommendation:** When you run applications that use TCP/IP sockets (such as HTTP, JMS or IIOP) use the asynchronous TCP/IP sockets I/O provided by the ZOSAIO authorized service to improve performance.

The PRODMGR authorized service enables the use of the IFAUSAGE macro as an authorized user to register and unregister the product. When a product is registered, its usage is measured, and the measurement data are collected into SMF record types 30 and 89.



The TXRRS authorized service enables the use of Resource Recovery Services (RRS). In a CICS Liberty server, RRS is required when you connect to a local MQ for z/OS queue manager in bindings mode. This type of connection is only supported in CICS standard-mode Liberty server.

### Do you need to run multiple angel processes?

An angel process is a stable process that does not run complex code, so it is not prone to being disrupted. Therefore, a single angel process can be used per software product stack. If multiple Liberty based software products are in use on the same LPAR (such as CICS Liberty and z/OS Connect Enterprise Edition), it is recommended to run a dedicated, named angel process for each software product. Using a dedicated angel process allows more flexibility when performing software maintenance.

For more information about the configuration of an angel process and the connection to it, see the configuration tasks [Configuring the angel process](#).

## Designing security for Liberty web applications

Address the various security considerations when you plan to deploy a Liberty web application.

A web client, such as a browser, REST client, or SOAP client, connects to CICS Liberty over an HTTP (or HTTPS) connection. The web application that is deployed in the Liberty JVM server typically accesses a CICS resource such as a VSAM file or a Db2 connection, or link to a CICS program. [Figure 120 on page 253](#) shows a high-level view of how the different [security principles](#) are supported for web applications.

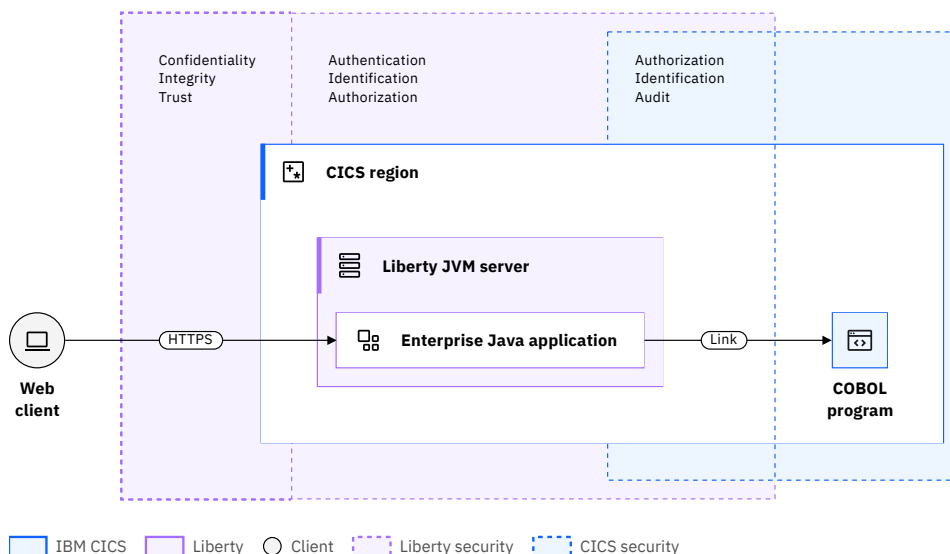


Figure 120.

where:

- HTTPS is used for confidentiality, integrity, and trust.
- Liberty security can be used for authentication, identification, and authorization. You can authenticate users using a variety of authentication mechanisms and authorize access to web applications through Enterprise Java roles.
- CICS security can be used to perform authorization checks and can also create an audit record. You can integrate Liberty security with CICS transaction and resource security, including propagation of the user identity.

When you design a solution for securing a web application that is deployed to a Liberty JVM server, consider the implications for:

- [Authentication and identification](#)
- [Authorization](#)
- [Confidentiality and integrity](#)
- [Trust](#)
- [Audit](#)

## Authentication and identification

### Which user registry to use?

To integrate CICS Liberty security with your existing security settings on z/OS, consider using a SAF user registry such as RACF. Alternatively, an LDAP registry might be more convenient if:

- All users are already present in an enterprise-wide LDAP registry.
- Authorization groups are already defined and centralized in the enterprise-wide LDAP registry.

### Which authentication mechanism to choose?

If you are using form-based authentication or basic authentication, consider using the HTTPS protocol rather than HTTP to ensure credentials remain protected in transport.

TLS client authentication can be used to establish trust between a remote server and a Liberty JVM server. It is most often used when a person (rather than another server or application) is being authenticated. TLS client authentication might not be practical when many clients need to connect to a JVM server. This is because the creation and administration of certificates can be a significant effort, for example, when certificates expire.

When you map a client certificate to a RACF user ID, you can use RACF certificate name filtering or import the client certificate into the RACF registry as a trusted certificate. When many clients use certificates to authenticate, consider using certificate name filtering to avoid storing and administering many client certificates.

Basic authentication, form-based authentication, or client authentication are often suitable when a web client authenticates directly with the Enterprise Java application that is running in a Liberty JVM server. For some scenarios, you might want to use [third-party authentication](#) instead. For example, a web client such as a REST client authenticating with an intermediate server where the intermediate server connects to the Liberty JVM server.

If you don't enable authentication, you can use a single URIMAP to set the RACF user ID for the CICS transaction. In this case, all requests to the Liberty JVM server run with the same CICS transaction and RACF user ID.

### When to use third-party authentication?

Third-party authentication is a common authentication mechanism when the user population is large. Using this architecture with CICS Liberty, the user authenticates with an authentication server to obtain a token. The authentication token is then sent to CICS Liberty, which validates the token and uses the identity in the token for authorization processing. The identity in the token would normally be mapped to a RACF user ID in advance.

Consider selecting third-party authentication when you:

- Want a user identity to be passed to CICS without the user's password.
- Need to flow an identity across different servers that use different registries.
- Want to establish trust by using a signed authentication token.

### Which third-party authentication token to use?

The type of authentication token that you use can be determined by various factors:

- Your enterprise security standards
- The type of information that is transmitted in the token (the *claims*)

- The way that the web client interacts with the Liberty JVM server, for example, through a REST or SOAP request

For simplicity you can configure third-party authentication using a JWT or an LTPA token. For custom tokens, you will need to write more involved authentication code, for example, a Trust Association Interceptor (TAI).

The JWT is the most commonly used third-party authentication token, especially for REST API requests.

#### **Which JWT authentication option to use?**

You have several options for implementing JWT authentication with CICS Liberty. The OpenID Connect Client feature provides a flexible configuration-centric way of implementing JWT authentication. However, if you need programmatic access to the claims in the JWT, consider other JWT authentication options.

#### **What if different authentication mechanisms for different services or different types of client are required?**

Each application that you deploy to a Liberty JVM server can use a different mode of authentication. You can also override authentication mechanisms at the server level.

For basic authentication, form-based authentication, and client authentication, the web application authenticates users only if it includes a security constraint in its application deployment descriptor (`web.xml`).

When you enable JWT authentication by configuring the OpenID Connect Client feature, authentication filters can be used to identify the types of request that are authenticated by JWT.

#### **When to enable Single Sign-On (SSO)?**

Use SSO when you want to optimize the web application authentication process by reducing the number of times a user's credentials need to be checked.

SSO enables users to log in to one Liberty JVM server and then access applications on other Liberty server without being prompted to log in again. LTPA SSO is enabled for a Liberty server by default. However, to use LTPA SSO across multiple Liberty servers, you need to configure each Liberty server to share the LTPA key file. For more information, see [How it works: LTPA \(Lightweight Third Party Authentication\)](#).

A JWT is suitable as an SSO token for REST clients that authenticate with a third-party authentication server. Different Liberty servers can be configured to use the same public key for validating the JWT. For more information, see [How it works: JSON Web Token \(JWT\)](#).

#### **Which RACF user ID is used for running CICS programs and accessing CICS resources?**

The CICS task user ID that is used to run CICS programs and access CICS resources is determined in the following order:

1. The Java security subject that is set during Liberty application security authentication is used. An example of this scenario is shown in [Design example: Securing a CICS Liberty web application with basic authentication and CICS transaction security](#).
2. If an unauthenticated subject is supplied from Liberty, the user ID that is defined in the matching CICS URIMAP is used. An example of this scenario is shown in [Design example: Securing a web application with a preset URIMAP user ID](#).
3. If no user ID is defined in the URIMAP, the request will run under the CICS default user ID.

## **Authorization**

#### **Which authorization option to choose?**

You can use Enterprise Java application security roles to authorize access to web applications. You can also use CICS transaction and resource security.

Using CICS transaction and resource security is convenient if you want to reuse existing CICS authorization procedures. If individual web applications are accessed from different URIMAPs, different transaction IDs can be assigned to each application.

Using [role-based security](#) requires authorization constraints (roles) to be defined in application deployment descriptors or in source code annotations. It is convenient if you want to reuse existing role definitions that are already present in an Enterprise Java application previously running on another platform.

With the `syncToOSThread` function of Liberty in effect, RACF can be used to authorize access to resources that are not controlled by CICS such as zFS files through the operating system thread identity. For more information, see [Configuring the `syncToOSThread` function](#).

### **Which option to use for role-based authorization?**

You can use SAF role authorization, local security role authorization, or role to group mapping.

- SAF role authorization allows security administrators to authorize access to Enterprise Java applications based on RACF profiles.
- Local role authorization can be a useful when you port an application directly from a third-party Enterprise Java application server that already uses this method. However, with local security role authorization, you cannot use CICS bundles to install your applications. This restriction is because a CICS bundle-installed application automatically creates an `<application-bnd>` element and uses the `ALL_AUTHENTICATED_USERS` special-subject, which prevents you from defining the element yourself.
- Role to group mapping can be used when neither SAF role authorization nor local security role authorization is in use. It associates a role to the group of the same name.

## **Confidentiality and Integrity**

### **Which TLS implementation to use?**

You can implement TLS by using JSSE or AT-TLS.

Because JSSE is a Java TLS implementation, the TLS handshake and data encryption processing are zIIP eligible.

AT-TLS processing is not zIIP eligible but offers the benefit that the TLS policy configuration can be centralized for all z/OS subsystems. However, CICS Liberty is an unaware AT-TLS application, which means that use of AT-TLS imposes the following restrictions:

- A client cannot use a certificate to authenticate with a web application when the connection between the client and the Liberty JVM server is secured by using AT-TLS.
- The `transport-guarantee` element in `web.xml` has no effect when you use AT-TLS.

### **Which type of keystore and truststore to use?**

CICS Liberty supports SAF key rings, Public Key Cryptography Standards #12 (PKCS12), and Java KeyStores (JKS). Consider using SAF key rings to benefit from the additional protection of SAF, for example, [RACF](#). If SAF key rings are not an option, then consider using PKCS12 stores as this store type is a standard format and language-neutral.

### **Which cypher suite to use?**

Use the most recent version of TLS that is supported by the partner system. The most recent version of software usually includes fixes to previous versions and enhancements such as support for stronger cipher suites. To control the TLS version that is supported by the JVM server, you can update the `sslProtocol` attribute on the `ssl` element in `server.xml`. To modify the list of ciphers that are supported by the server, you can specify them in the `enabledCiphers` attribute on the `ssl` element.

## **Trust**

### **How to ensure that the remote system that connects to a Liberty JVM server is a trusted server?**

You can use TLS client authentication so that the server provides a client certificate that is validated by the Liberty JVM server. Successful client authentication requires that the certificate authority (CA)

that signed the client certificate is considered trusted by the Liberty JVM server. To be considered trusted, the certificate of the CA must be in the Liberty server's truststore. For more information, see [Configuring TLS for a Liberty JVM server by using RACF](#).

#### **How to ensure that a third-party authentication token is trusted?**

An authentication token, for example a JWT, can be signed by a private key. The signature can be verified by the issuer's public key. This configuration creates a trust relationship that can be used to ensure the integrity of the token.

## **Audit**

#### **How are web requests audited in CICS?**

When a Liberty web request is processed by CICS, the CICS task will initially run under the basic authentication user ID, the URIMAP user ID, or the default user ID. When Liberty establishes the Java security subject for the request, CICS is notified and if the `cicsts:security-1.0` feature is installed, CICS updates the user ID that is being used by the CICS task, ensuring that the correct security context is used by CICS transaction and resource security checks. The task user ID audited in the CICS Monitoring Facility (CMF) records will likewise match the final value of the CICS task user ID, ensuring that the correct security credentials are audited.

Review the design examples that offer different configurations.

### **Design example: Securing a Liberty web application with basic authentication and CICS transaction security**

In this example, the Enterprise Java application is deployed with security constraints in its application deployment descriptor (`web.xml`) that require the web client to provide basic authentication credentials. CICS transaction security is used to authorize access to the Enterprise Java application. The Enterprise Java application is deployed as a CICS bundle so all authenticated users are automatically authorized to run the application in Liberty. A URIMAP that matches the request URI specifies the transaction ID to be used for the request. The transaction is protected such that the authenticated user ID must be authorized to run the transaction.

For more information about configuring this scenario, see the configuration task [Configuration example: Securing a web application with basic authentication and CICS transaction security](#).

[Figure 121 on page 258](#) shows an overview of the scenario.

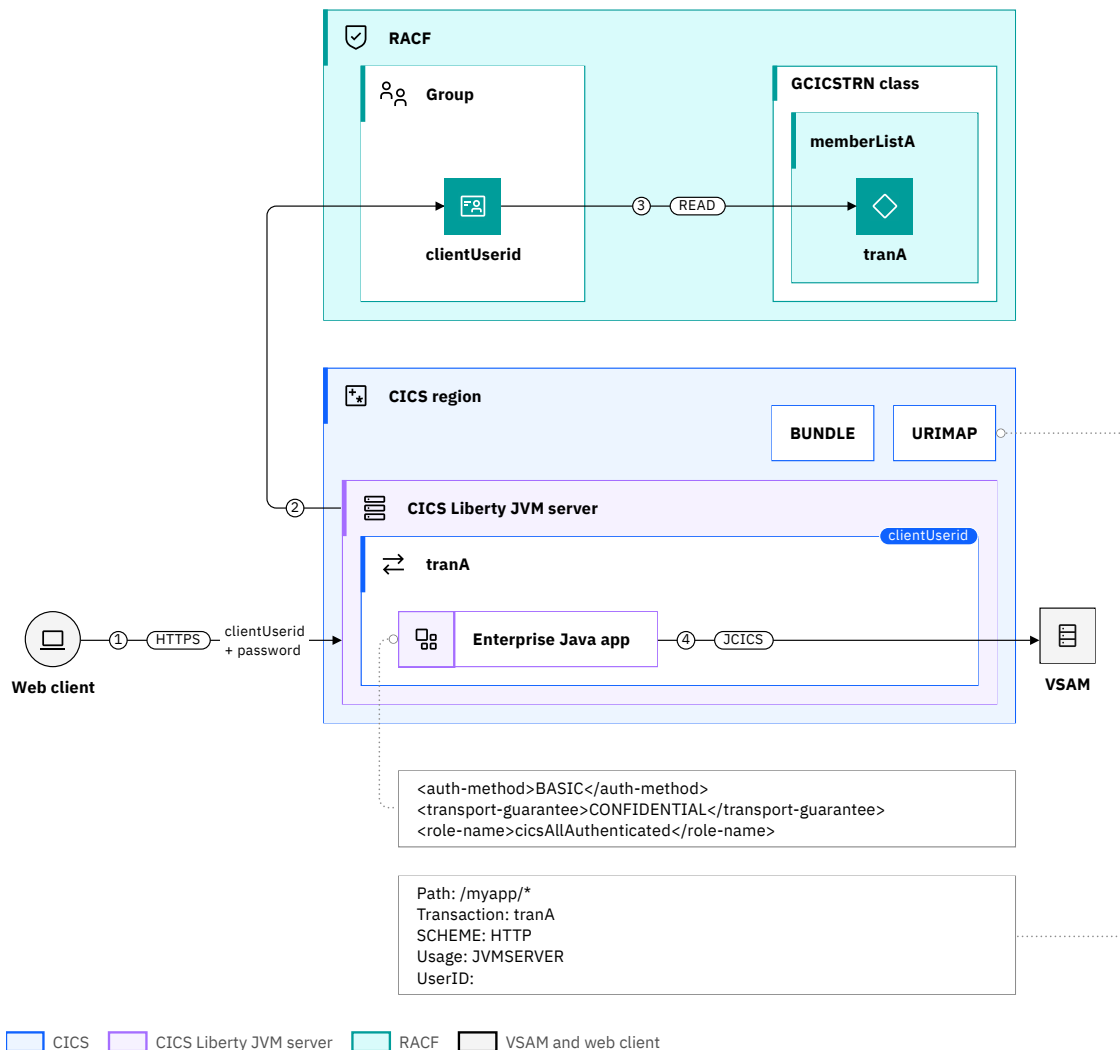


Figure 121. Securing a Liberty web application with basic authentication and CICS transaction security

1. The web client sends an HTTPS request with the *clientUserid* and password in the HTTP Authorization header to the Liberty JVM server.
- Important:** It is recommended to use HTTPS because the basic authentication credentials are not signed or encrypted.
2. To authenticate the request, the Liberty JVM server validates the *clientUserid* and *password* against the configured RACF user registry and sets the Java security subject to the *clientUserid*.
3. CICS transaction security is used to authorize the request. The URIMAP that matches the request determines the transaction *tranA* is to be used. CICS checks that the *clientUserid* is authorized to run the transaction. To do this, CICS calls RACF to verify that the *clientUserid* has READ access to member list *memberListA* that contains *tranA*, in the GCICSTRN class.
4. The CICS transaction (including the Enterprise Java application) runs with transaction *tranA* under the context of the *clientUserid*. Any JCICS calls to CICS resources such as VSAM files use the *clientUserid* security context.

#### Related information

[Security for Java applications](#)

## **Configuration example: Securing a Liberty web application with basic authentication and CICS transaction security**

Configure basic authentication for a web application and then check that the RACF user ID is authorized to run the CICS transaction.

### **Before you begin**

This configuration task is based on the example security scenario [Design example: Securing a CICS Liberty web application with basic authentication and CICS transaction security](#).

Before you begin this task, you must complete these tasks:

- Connect the Liberty server to the angel process, see [Configuring and starting the Liberty angel process](#) [Connecting a Liberty JVM server to the angel process](#).
- Configure the Liberty server to use the RACF registry, see [Configuring Liberty to use a SAF user registry](#).

You need to know the RACF user ID that is used to authenticate. This user ID must exist and have an OMVS segment.

You must have:

- Authorization to define CICS resource definitions.
- Authorization to update the application security constraint in the Enterprise Java deployment descriptor (`web.xml`).
- Write access to the `server.xml` configuration file.

### **About this task**

In this task, you configure an application to use basic authentication with a RACF user registry and CICS transaction security to verify that the authenticated user ID is authorized to run the web application.

This task assumes the following definitions:

- `clientUserId` is the authenticated user ID.

### **Procedure**

1. Add the security controls to the Enterprise Java application's deployment descriptor as follows. The `web.xml` can be found inside the source files for the web application that you are deploying.
  - a. Add a login configuration to the application's `web.xml` file to specify basic authentication as the authentication method.

```
<auth-method>BASIC</auth-method>
```
  - b. Define an authorization constraint in the `web.xml` to restrict access to all URL paths for this application to the special role `cicsAllAuthenticated`:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>myResourceName</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>cicsAllAuthenticated</role-name>
  </auth-constraint>
</security-constraint>
```

For more information, see the configuration task [Configuring basic authentication for Liberty web applications](#).

2. Deploy the Java application as a CICS bundle (see [Deploying a CICS bundle in the CICS Explorer product documentation](#)).
3. Assign a CICS transaction ID to be used for the request.

Define and install a URIMAP of type JVMSERVER for the web application. For example, you can specify a URIMAP to match the generic context root (URI) of the web application to scope the transaction ID to the application. For more information, see the configuration task [Configuring CICS transaction security for a Liberty JVM server](#).

4. Authorize the *clientUserId* to run the CICS transaction defined in step 3.

Authorize all users of the web application to run the transaction that is specified in the URIMAP that uses CICS transaction security. For more information, see the configuration task [Configuring CICS transaction security for a Liberty JVM server](#).

## Results

Requests to this application are authenticated by using a RACF user ID. Authorization is determined by using CICS transaction security profiles. The use of the role `cicsAllAuthenticated` - which maps to the special role `ALL_AUTHENTICATED_USERS` - creates a bypass of Liberty authorization. The authorization verification then relies on CICS transaction security.

To validate the security environment is functioning correctly, you need to send a request to the web application.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction ID defined in step 3 (or CJSA by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: Securing a Liberty web application with basic authentication and SAF role authorization

In this example, the Enterprise Java application is deployed with security constraints in its application deployment descriptor (`web.xml`) that require the web client to provide basic authentication credentials and to use HTTPS. SAF role authorization is used to verify that the *clientUserId* is a member of the authorized EJBROLE profile. An authorization constraint is configured in the `web.xml` so that users must be in the security role *myRole* in order to run the application.

For more information about configuring for this scenario, see [Configuration example: Securing a web application with basic authentication and SAF role authorization](#).

[Figure 122 on page 261](#) shows an overview of the scenario.



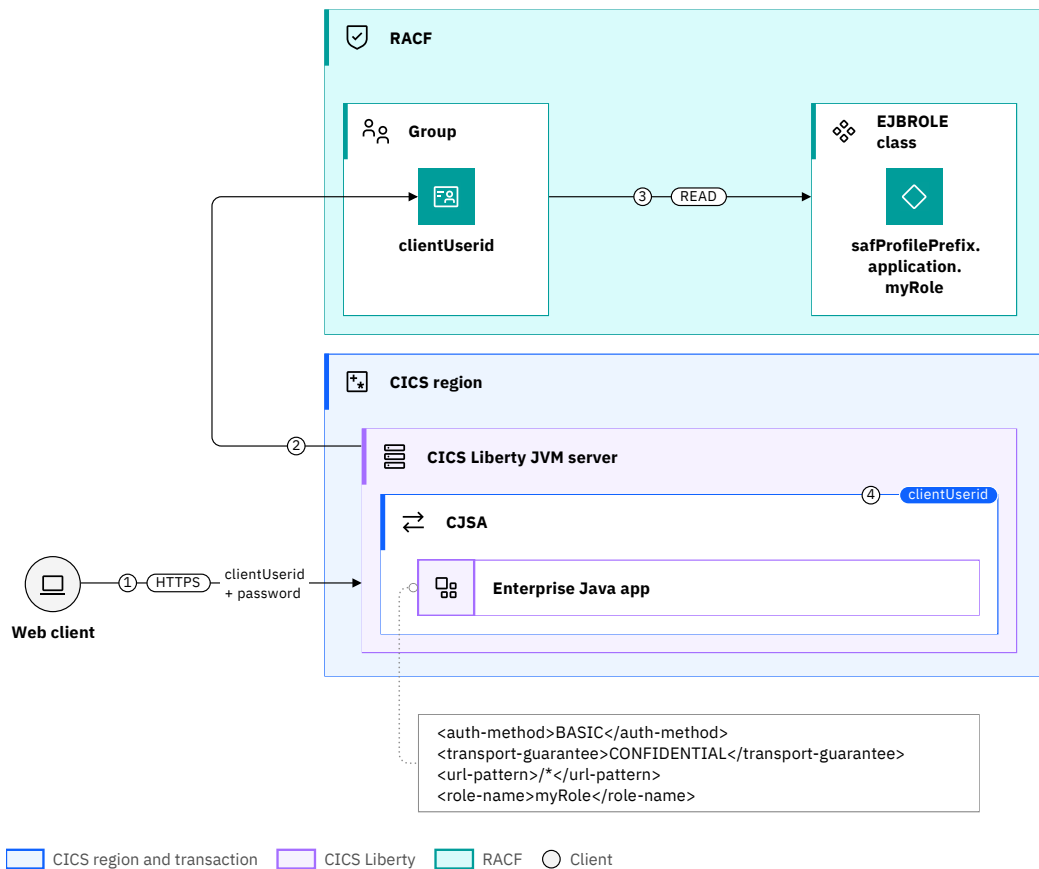


Figure 122. Securing a Liberty web application with basic authentication and SAF role authorization

1. The web client sends an HTTPS request with the *clientUserid* and password in the HTTP Authorization header to the Liberty JVM server.

**Important:** It is recommended to use HTTPS because the basic authentication credentials are not signed or encrypted.

2. To authenticate the request, the Liberty server validates the *clientUserid* and password against the configured RACF user registry and uses this to set the Java security subject.
3. The Liberty server checks that the *clientUserid* is authorized to access the web URL defined in the constraint. The verification is done by calling RACF to verify that the user ID has READ access to the profile *safProfilePrefix.application.myRole* in the class EJBROLE or its corresponding resource group class.
4. The CICS transaction (including the Java application code) runs using the security context of the *clientUserid*. No URIMAP is defined for the request so the transaction runs with the CICS Liberty default transaction ID CJSA.

### Related information

[Security for Java applications](#)

## Configuration example: Securing a Liberty web application with basic authentication and SAF role authorization

Configure basic authentication for a web application and then check that the RACF user ID is a member of an authorized security role.

### Before you begin

This configuration task is based on the example security scenario [Configuration example: Securing a web application with basic authentication and CICS transaction security](#).

Before you begin this task, you must complete these tasks:

- Connect the Liberty server to the angel process, see [Configuring and starting the Liberty angel process](#) [Connecting a Liberty JVM server to the angel process](#).
- Configure the Liberty server to use the RACF registry, see [Configuring Liberty to use a SAF user registry](#).

You need to know the RACF user ID that is used to authenticate. This user ID must exist and have an OMVS segment.

You must have:

- Authorization to create or update the application security constraint in the Enterprise Java deployment descriptor (`web.xml`)
- Write access to the `server.xml` configuration file.
- Authorization to issue the RACF RDEFINE and PERMIT commands.

### About this task

In this task, you configure an application to use basic authentication against the SAF registry, and SAF role authorization to check that the `clientId` is authorized to run the web application.

This task assumes the following definitions:

- `clientId` is the user ID that is used to authenticate.
- `myRole` is the security role that protects the application.
- `safProfilePrefix` is the SAF profile prefix for the Liberty server.
- `myApplication` is the name of the Enterprise Java application.

### Procedure

1. Add the security configuration to the Enterprise Java application's deployment descriptor as follows. The `web.xml` file can be found inside the source files for the web application that you are deploying.
  - a. Add a login configuration to the application's `web.xml` file to specify HTTP basic authentication as the authentication method.

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

- b. Define an authorization constraint in the `web.xml` to restrict access to all URL paths for this application to users in the role `myRole`.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>myResourceName</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>myRole</role-name>
  </auth-constraint>
</security-constraint>
```

2. Deploy the Java application as a CICS bundle (see [Deploying a CICS bundle in the CICS Explorer product documentation](#)).
  3. Update the `Liberty server.xml` configuration file:
    - Add the `<safAuthorization>` element to enable SAF role authorization.
  4. Authorize the `clientUserId` to run the web application:
    - a. Create a RACF EJBROLE resource profile to match the default role mapper pattern for EJBROLE profiles, for instance `safProfilePrefix.myApplication.myRole`.
    - b. Grant the `clientUserId` READ permission to the EJBROLE.
- For detailed instructions on these steps, see the task [Configuring SAF authorization with an EJBROLE](#).
5. Start, or restart the server if it was already running, to pick up the changes that are made to the RACF class profiles.

## Results

Requests to this application are authenticated by using RACF user ID and password. Authorization is determined by using an EJB role RACF security profile.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction ID defined by the URIMAP that matches the request (or CJSA by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: Securing a Liberty web application with a functional user ID from a URIMAP

In this example, a functional user ID is preset as the CICS task user ID using a matching URIMAP. The Enterprise Java application is deployed without a security constraint in its application deployment descriptor (`web.xml`) and with no security annotations. This means no Enterprise Java security mechanisms will be used for authentication or authorization and CICS transaction security is used to authorize the request based solely on the preset functional user ID.

For more information about how to configure for this scenario, see [Configuration example: Securing a web application with a preset URIMAP user ID](#).

[Figure 123 on page 264](#) shows an overview of the scenario.

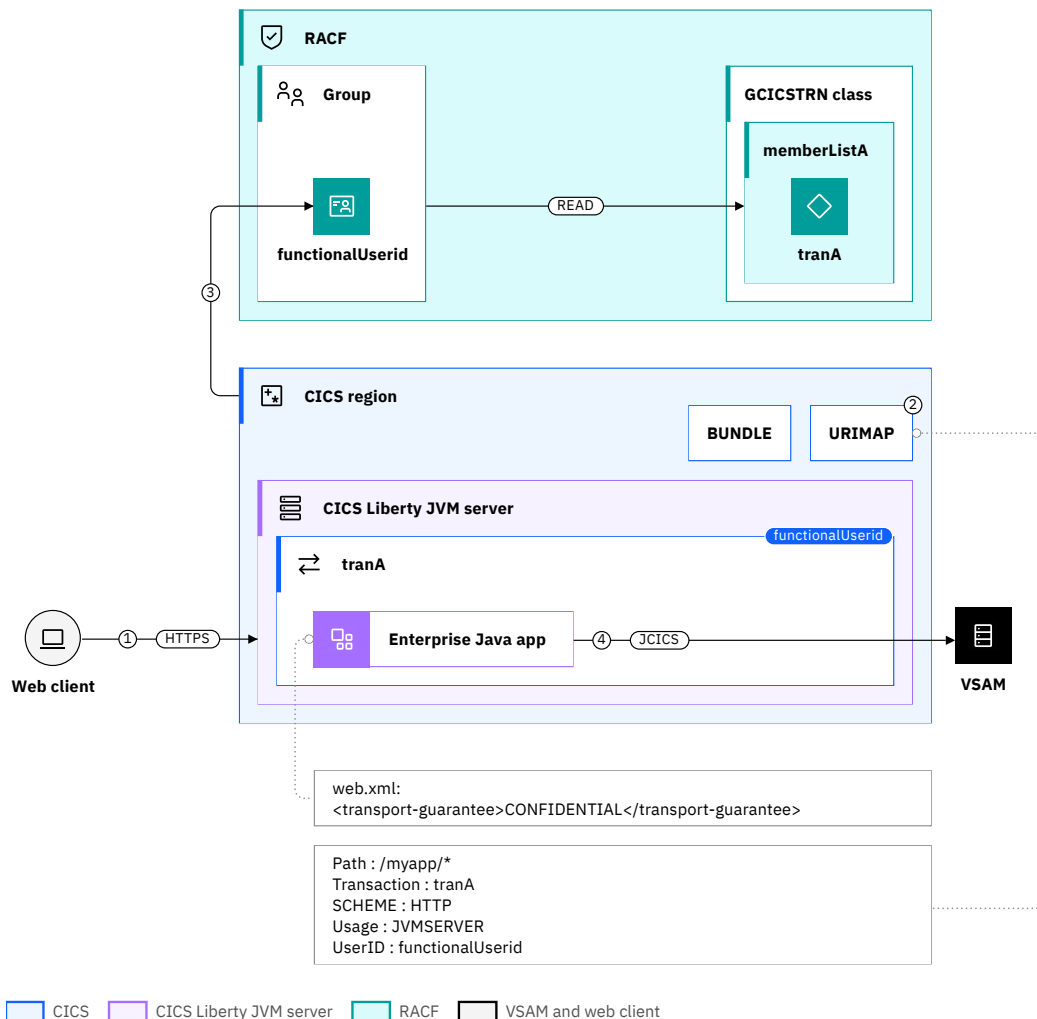


Figure 123. Securing a web application with a preset URIMAP user ID

1. The web client sends an HTTPS request to the Liberty JVM server because `<transport-guarantee>CONFIDENTIAL</transport-guarantee>` is set in the applications deployment descriptor. Liberty does not authenticate the request because no security constraint is defined. Therefore, the Java security subject is set as the Liberty unauthenticated user ID, which is WSGUEST by default.
- Important:** It is recommended that you use HTTPS with client authentication because any requests that match the URIMAP can run under the functional user ID.
2. CICS matches the URI of `/myapp/*` against the web request's URI and sets the *functionalUserid* and *tranA* from the URIMAP as the CICS task user ID and transaction ID respectively.
3. CICS transaction security is used to authorize the request using the *functionalUserid*. CICS calls RACF to verify that the *functionalUserid* belong to a group that has READ authority to the member list (*memberListA*) containing *tranA* in the GCICSTRN class.
4. The CICS transaction, including the Enterprise Java application, runs with transaction ID *tranA* under the context of the *functionalUserid*. Any JCICS calls to CICS resources such as VSAM files use the *functionalUserid* security context.

## ***Configuration example: Securing a Liberty web application with a functional user ID from a URIMAP***

Configure a functional user ID from a URIMAP for a web application and check that the RACF user ID is authorized to run the CICS transaction.

This configuration task is based on the example security scenario: [Design example: Securing a web application with a preset URIMAP user ID](#).

### **Before you begin**

Before you begin this task, you must complete the following tasks:

- Ensure CICS security is activated by setting the SIT parameter SEC=YES.
- You need to know the RACF user ID that is used to authenticate.
- You must have authorization to issue the RACF RDEFINE and PERMIT commands.

### **About this task**

In this task, you configure an application to use a preset user ID and CICS transaction security to verify that the preset user ID is authorized to run the web application.

This task assumes `functionalUserId` is the preset user ID.

### **Procedure**

Add the security controls to the Enterprise Java application's deployment descriptor (`web.xml`) as follows. The `web.xml` file can be found inside the source files for the web application that you are deploying.

1. Add a transport guarantee to the application's `web.xml` file to enforce the use of a TLS connection:

```
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

**Important:** Any requests that match the URIMAP are run under the functional user ID, for added security consider using TLS client-certificate authentication. For more information, see [Configuring TLS client authentication for a Liberty JVM server by using RACF](#).

2. Deploy the Java application in a CICS bundle. For instructions, see [<LINK to be added once question clarified>](#).
3. Assign a CICS transaction ID and user ID to be used for the request.

Define and install a URIMAP of type JVMSERVER and scheme HTTPS for the web application, specifying `functionalUserId` on the USERID attribute and `tranA` on the TRANSACTION attribute. You can specify a URIMAP to match the generic context root (URI) of the web application, for example, `/myapp/*`, by setting the appropriate value on the PATH attribute. For more information, see [Configuring CICS transaction security for a Liberty JVM server](#).

4. Authorize the `functionalUserId` to run transaction `tranA` that is defined in Step “3” on page 265.

Using standard CICS transaction security practices, authorize all users of the web application to run the transaction specified in the URIMAP. For more information, see [Configuring CICS transaction security for a Liberty JVM server](#).

5. Start, or restart the Liberty JVM server if it was already running.

### **Results**

Requests to this application are authenticated using a functional RACF user ID. Authorization is determined by using CICS transaction security profiles.

To validate the security environment is functioning correctly, send a request to the web application.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example:

1. In the **Regions** view, select **Add Security Request Recording** from the pop-up menu.
2. In the dialog, select the **JVM Server** tab and set the **Transaction ID** field to the *tranA* transaction ID that is defined in .

For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Design example: Securing a Liberty web application with TLS client authentication and SAF role authorization

In this example, the Enterprise Java application is deployed with a security constraint in its application deployment descriptor (web.xml). This restraint requires the web client to provide a TLS client certificate. An authorization constraint is configured in the deployment descriptor so that authenticated users must be in role *myRole* to run the application. SAF role authorization is used to verify that the authenticated user ID is a member of the authorized role.

For more information about configuring this scenario, including the definition of role-mapping preferences, see the configuration task [Configuration example: Securing a web application with TLS client authentication and SAF role authorization](#).

Figure 124 on page 266 shows an overview of the scenario.

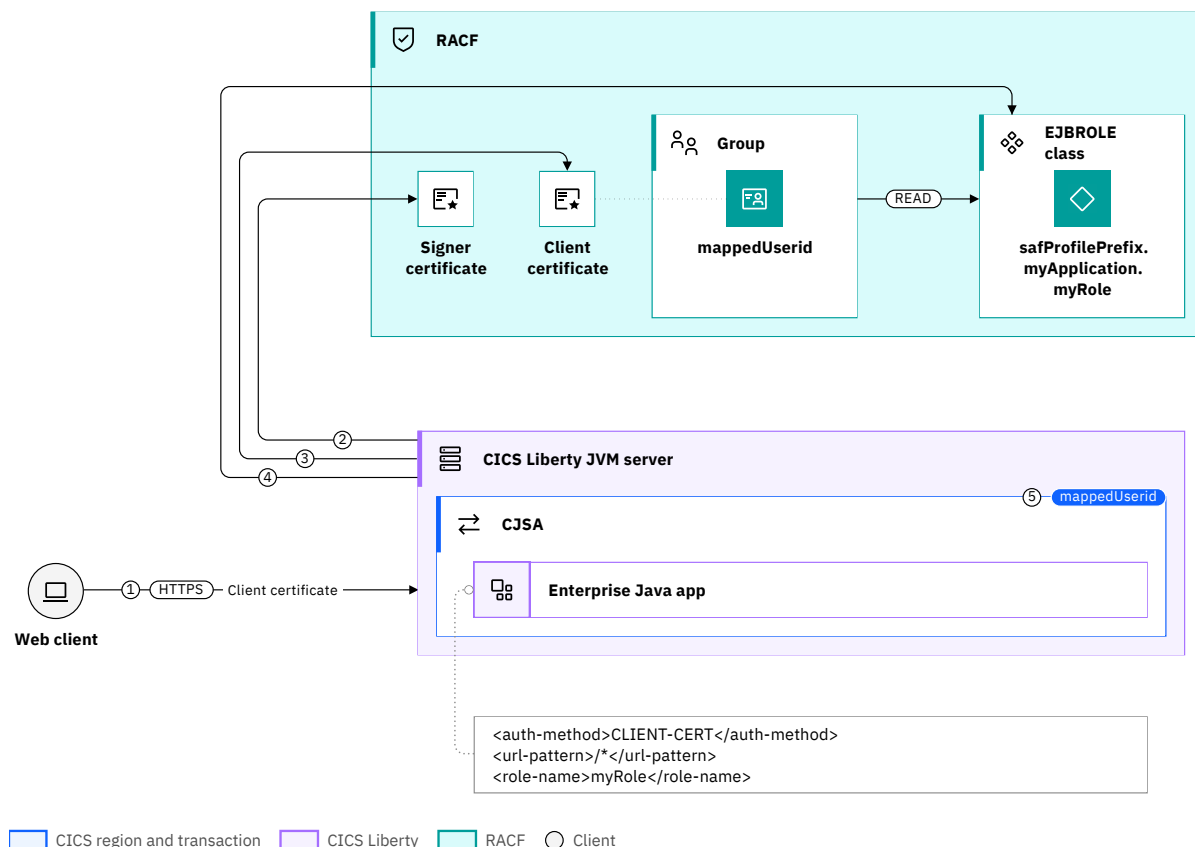


Figure 124. Securing a Liberty web application with TLS client authentication and SAF role authorization

1. The web client sends an HTTPS request to the Liberty JVM server. A TLS handshake is performed. The Liberty JVM server sends its server certificate and asks the client to provide its certificate.

2. The server validates the chain of trust by checking that the client certificate issuer (signer certificate) is in the RACF keyring.
3. To authenticate the request, the Liberty server also maps the web client certificate to a RACF user ID (the *mappedUserid*).
4. The Liberty server checks that the *mappedUserid* is authorized to run the web request. This check is done by calling RACF to verify that the *mappedUserid* has READ access to profile `safProfilePrefix.myApplication.myRole` in the class EJBROLE or its corresponding resource group class.
5. The CICS task (including the Java application code and COBOL program) runs with the *mappedUserid*. No URIMAP is defined for the request so the task runs with the CICS Liberty default transaction ID CJSA.

**Important:** If you don't want to run tasks with the default transaction ID, you can define a single URIMAP for all requests to the Liberty JVM server. This URIMAP specifies another transaction ID (for example, BJSA). The BJSA transaction ID resource definition is created by copying the CJSA resource definition and changing the transaction name.

### Related information

[Security for Java applications](#)

## **Configuration example: Securing a Liberty web application with TLS client authentication and SAF role authorization**

This task covers three areas. Configuring a Liberty JVM server to perform authentication by using a TLS client certificate. Mapping the certificate to a RACF user ID. Checking that the user ID is a member of an authorized role.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web application with a TLS client certificate and SAF role authorization](#).

An [alternative configuration](#) based on importing a client certificate into the RACF registry instead of using certificate name filtering is explained at the end of this document.

Before you begin this task, you must complete these tasks:

- Connect the Liberty server to the angel process, see [Configuring and starting the Liberty angel process](#) [Connecting a Liberty JVM server to the angel process](#).
- Configure a TLS connection between the web client and the Liberty server with TLS client authentication enabled. see [Configuring TLS for a Liberty JVM server by using RACF](#).

You need to know:

- The subject value of the client certificate to be mapped.
- The user ID to which the TLS client certificate is mapped. This user ID must exist and have an OMVS segment.

You must have:

- Authorization to issue the RACDCERT MAP command.
- Write access to the `server.xml` configuration file.
- Ability to update the web application's deployment descriptor `web.xml`.

### About this task

In this task, you configure RACF certificate name filtering to map a TLS client certificate to a RACF user ID. SAF role authorization is then used to check that the authenticated user ID is authorized to run the web application.

This task assumes the following definitions:

- *mappedUserId* is the RACF user ID to which the client certificate is mapped.
- *CN=myClient.host.com, O=IBM, C=US* is the client certificate subject's distinguished name value.
- *clientCertLabel* is the label of the client certificate.

## Procedure

1. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETOPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

2. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(mappedUserId)
SDNFILTER('CN=myClient.host.com.O=IBM.C=US')
WITHLABEL('clientCertLabel')
```

**Important:** In this command the syntax of the SDNFILTER is significant, use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS Security Server RACF Command Language Reference*.

3. Refresh the DIGTNMAP RACF class.

For the changes to take effect, enter the following RACF command:

```
SETOPTS RACLIST(DIGTNMAP) REFRESH
```

4. Deploy the web application with a login-config element for CLIENT-CERT in the deployment descriptor `web.xml`.

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>
```

The `web.xml` file can be found inside the source files for the web application that you are deploying.

5. Authorize the authenticated RACF user ID to run the web application:
  - a. Define an authorization constraint (`<auth_constraint>` element) in the deployment descriptor (`web.xml`).
  - b. Use `<safAuthorization>` in your `server.xml` to allow user or group role membership to be mapped by SAF.
  - c. Create an EJBROLE resource profile for the application that uses the default pattern `safProfilePrefix.<resource>.<role>`, where resource is the application name and role is defined in the application's deployment descriptor.
  - d. Authorize the authenticated RACF user ID to access the EJBROLE. For more information about these steps, see task [Configuring SAF authorization with an EJBROLE](#).
6. Start, or restart the server if it was already running, to pick up the changes that are made to the RACF class profiles.

## Results

The TLS client certificate is mapped to a RACF user ID that is then used for running the CICS task.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction



ID defined by the URIMAP that matches the request (or CJSJA by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Alternative configuration where you import a client certificate into the RACF registry

You have an alternative to using RACF certificate name filtering to map the client certificate to a RACF user ID. You can import the client certificate into the RACF registry as a trusted certificate. The client certificate can be transferred to z/OS as a PKCS #12 certificate package.

This process can be done by using the following command:

```
RACDCERT ADD('dataset') TRUST ID(mappedUserid) WITHLABEL('clientCertLabel') PASSWORD (password)
```

The command uses the following values:

- *dataset* is the name the z/OS sequential file that contains the PKCS #12 certificate package.
- *password* is the password that is associated with the PKCS #12 certificate package.

For the full syntax of the RACDCERT MAP command, see RACDCERT MAP (Create mapping) in the z/OS Security Server RACF Command Language Reference.

## Design example: Securing a Liberty web application with a JWT and CICS transaction security

In this third-party authentication scenario, a web client authenticates with an intermediate server, which then propagates an identity in a JWT to a Liberty JVM server. The identity in the JWT is mapped to a RACF user ID that is used to authorize the user to run the CICS transaction. The Enterprise Java application is deployed as a CICS bundle so all authenticated users are automatically authorized to run the application in Liberty. A URIMAP that matches the request URI specifies the transaction ID to be used for the request.

For more information about configuring this scenario, see [Configuration example: Securing a web application with a JWT and CICS transaction security](#).

[Figure 125 on page 270](#) shows a third-party authentication scenario where a web client authenticates with an intermediate server, which then propagates an identity in a JWT to a Liberty JVM server. The identity in the JWT is mapped to a RACF user ID that is used to authorize the user to run the CICS transaction.

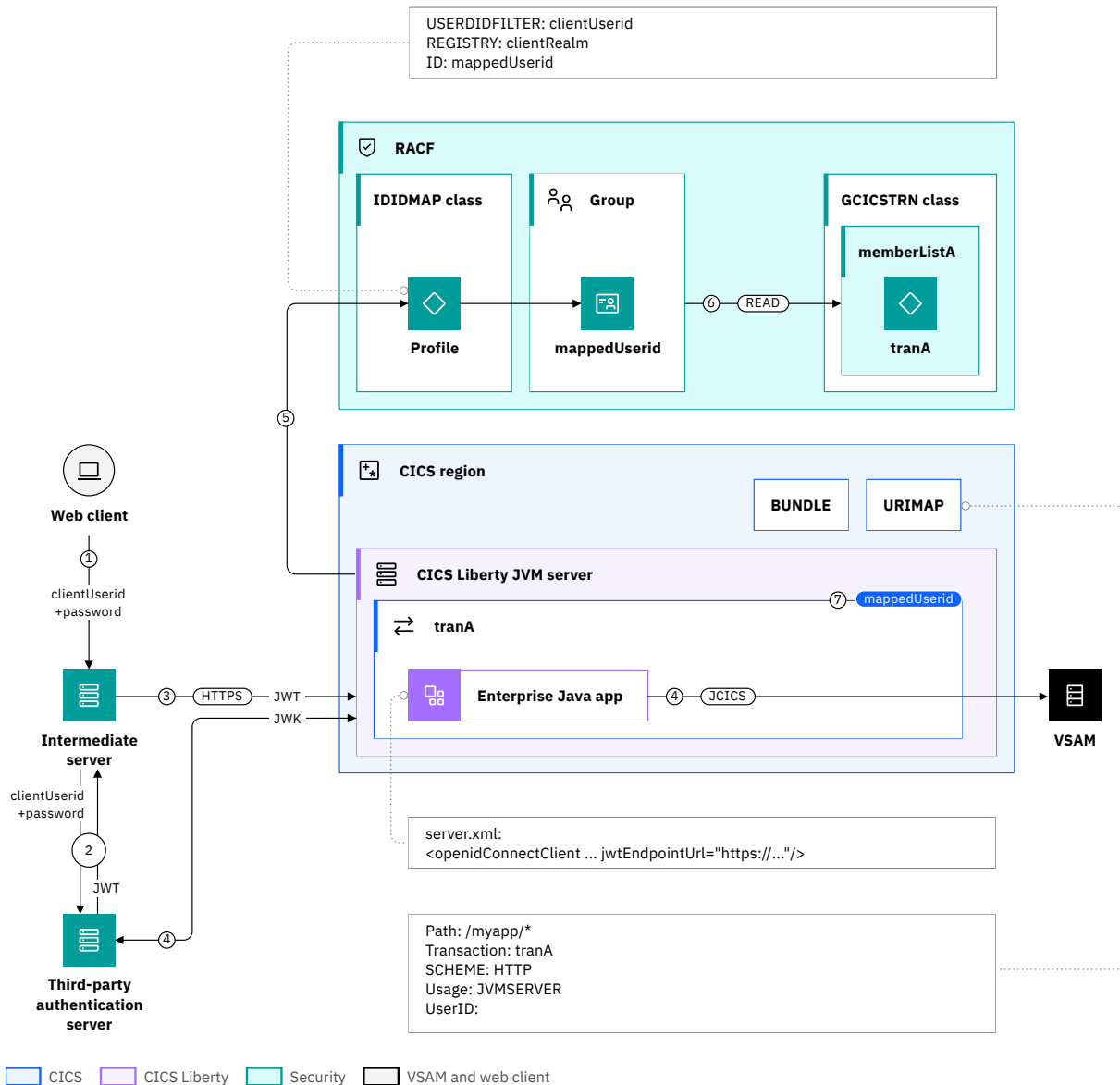


Figure 125. Securing a Liberty web application with a JWT and CICS transaction security

1. The web client sends a request to an intermediate server with its user credentials (`clientUserId` and `password`).
2. The intermediate server authenticates the client with a third-party authentication server, which returns an authentication token in the form of a `JWT`. The `JWT` contains the distributed identity (`clientUserId` and `clientRealm`) of the user. The `JWT` is signed with the private key of the authentication server.
3. The intermediate server sends an `HTTPS` request to the Liberty JVM server with the `JWT` in an HTTP Authorization request header.
4. The Liberty server validates the `JWT` by retrieving the necessary public key from the third-party authentication server. The public key is provided in a JSON Web Key (JWK) format and is publicized by the third-party authentication server on its JWK endpoint. Alternatively the Liberty server can validate the `JWT` by using a locally stored public key, but this option is not illustrated in this scenario.
5. The subject claim in the `JWT` is used to identify the user. The Liberty server also maps the distributed identity to a `mappedUserId` by using an IDIDMAP profile that is created by using the `RACMAP` command.

6. To authorize the request, CICS checks that the *mappedUserid* is authorized to run transaction *tranA*. CICS carries out this check by calling RACF to check that the *mappedUserid* belongs to a group that has READ access to the member list (*memberListA*) containing *tranA* in the GCICSTRN class.
7. The CICS transaction (including the Enterprise Java application) runs with transaction ID *tranA* and the *mappedUserid* as the security context.

### Related information

[Security for Java applications](#)

## **Configuration example: Securing a Liberty web application with a JWT and CICS transaction security**

This task explains how to use the OpenID Connect Client feature to configure a Liberty JVM server to accept a JWT as an authentication token. The task then explains how to run the CICS task with a RACF user ID mapped from a claim in the JWT.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web application with a JWT and CICS transaction security](#). You must be familiar with the information in [Third-party authentication](#).

Before you begin this task, you must complete these tasks:

- Enable the CICS Liberty security feature (`cicsts:security-1.0`).
- Connect the Liberty server to the angel process, see [Configuring and starting the Liberty angel process](#) and [Connecting a Liberty JVM server to the angel process](#).
- Enable RACF to be able to map distributed identities, see [Configuring RACF for identity propagation](#).

You need to know:

- The claims that are present in the JWT.
- The JSON Web Key (JWK) endpoint information to retrieve the JWK used to validate the JWT signature.

You must have write access to the `server.xml` configuration file.

### About this task

In this task, you configure a Liberty JVM server to perform JWT authentication and then use the identity in the JWT for running the CICS task.

This task assumes:

- *regionUserid* is the CICS region user ID.
- *myKeyRing* is the name of the RACF key ring.
- *cn=JeanLeclerc,ou=employees,o=ibm,c=fr* is the distributed user ID from the JWT subject claim.
- The JWT is sent to CICS in an HTTP *Authorization* request header field.
- The RS256 public/private key pair algorithm is used to sign the JWT.
- The identity in the JWT subject claim is a distributed identity.
- The Enterprise Java application is deployed as a CICS bundle so all authenticated users are automatically authorized to run the application in Liberty. For more information, see [Deploying a Enterprise Java application in a CICS bundle to a Liberty JVM server](#).

An example JWT:

```
{
  "alg": "RS256" }.
{
  "iss": "idg",
  "sub": "cn=JeanLeclerc,ou=employees,o=ibm,c=fr",
  "exp": 1496230040,
  "iat": 1496229740 }.
RSASHA256signature
```

In the example:

- The header contains an `alg` (algorithm) claim RS256. RS256 (RSA Signature with SHA-256) is the algorithm that is used to sign the JWT.
- The `iss` (issuer) claim `idg` identifies the principal that issued the JWT.
- The `sub` (subject) claim `cn=JeanLeclerc,ou=employees,o=ibm,c=fr` is a distributed identity. The distributed identity is mapped to a RACF user ID and then mapped RACF user ID is used for running the CICS task.
- The `exp` (expiration time) claim identifies the expiration time on or after which the JWT must not be accepted for processing.
- The `iat` (issued at) claim identifies the time at which the JWT was issued.
- The signature (not shown) is calculated by using the header and the payload. The signature certifies that only the party that holds the private key is the one that created and signed the JWT, and it also verifies that the claims are not altered.

**Important:** If the JWT is issued by a JWT provider that does not support JWK (JSON Web Key) or is signed by using the HMAC-SHA256 algorithm, then some steps in this procedure must be modified. For more information, see [“Alternative configuration when the public key is stored locally or the HS256 algorithm is used” on page 273.](#)

## Procedure

1. Add the `openidConnectClient-1.0` Liberty feature to the `server.xml` file.
2. Configure the `<openidConnectClient>` element in the `server.xml` to use JWT authentication, for example:

```
<openidConnectClient id="RS"
  clientId="RS-JWT-CICS" inboundPropagation="required"
  signatureAlgorithm="RS256" jwkEndpointUrl="https://... "
  userIdentifier="sub" mapIdentityToRegistryUser="false"
  issuerIdentifier="idg"/>
```

In this example:

- `id` and `clientId` are element identifiers.
- `inboundPropagation` is set to `required` to allow Liberty JVM server to use the received JWT as an authentication token.
- `signatureAlgorithm` specifies the algorithm to be used to verify the JWT signature.
- `jwkEndpointUrl` is the URL to the third-party authentication server JWK endpoint.
- `userIdentifier` indicates the claim to use to create the user subject. The default claim to use for the user subject is the `sub` claim.
- `mapIdentityToRegistryUser` indicates whether to map the retrieved identity to the registry user. You set `mapIdentityToRegistryUser` as `false` because the identity in the `sub` claim is not a RACF user ID.
- `issuerIdentifier` defines the expected issuer.

For more information about other `openidConnectClient` attributes, see the WebSphere Application Server for z/OS Liberty IBM Documentation topic [OpenID Connect Client](#).

**Note:** If the JWK endpoint requires the use of HTTPS, the appropriate TLS configurations need to be in place, for example, the TLS truststore might need to be updated.

**Note:** If JWT authentication is required only for a subset of requests to the Liberty JVM server, the `openidConnectClient` element can include an `authFilter` attribute that represents conditions that are matched against the HTTP request.

For more information about authentication filters, see [Authentication Filter \(authFilter\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

3. Set the attribute `mapDistributedIdentities="true"` on the `safCredentials` element in the `server.xml` configuration file.

For example,

```
<safCredentials mapDistributedIdentities="true"/>
```

4. Define a distributed identity filter in RACF to map the distributed user ID from the JWT subject claim to a RACF user ID.

For example, enter this command:

```
RACMAP ID(mappedUserId) MAP USERIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))  
REGISTRY(NAME('*')) WITHLABEL('Mapping Jean Leclerc')
```

In this example, `REGISTRY(NAME(' * '))` matches any registry realm name. Alternatively, to match a specific realm, replace the `*` with the value that is specified on the `realmName` attribute of the `openidConnectClient` element.

This example shows a one-to-one mapping. z/OS Identity Propagation also supports many-to-one mappings. For example, you might use a many-to-one mapping to map all employees to the same RACF user ID.

For more information about the RACMAP command, see [RACMAP \(Create, delete, list, or query a distributed identity filter\)](#) in the *z/OS Security Server RACF Command Language Reference*.

5. Refresh the RACF IDIDMAP class.

For the changes to take effect. Enter the following RACF command:

```
SETOPTS RACLIST(IDIDMAP) REFRESH
```

6. Assign a CICS transaction ID to be used for the request.

Define a URIMAP of type JVMSERVER for the web application. For example, you can specify a URIMAP to match the generic context root (URI) of the web application to scope the transaction ID to the application.

7. Authorize the mapped RACF user ID to run the CICS transaction.

Authorize all users of the web application to use the transaction that is specified in the URIMAP that uses CICS transaction security. For more information, see the configuration task [Configuring CICS transaction security for a Liberty JVM server](#).

For more information about configuring the OpenID Connect Client feature with Liberty z/OS, see [Configuring JSON Web Token authentication for OpenID Connect](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

## Results

A JWT is used for authenticating a request to a Liberty JVM server. The identity in the JWT is used for running the CICS task.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction ID defined by the URIMAP that matches the request (or CJSA by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Alternative configuration when the public key is stored locally or the HS256 algorithm is used

If the JWT is issued by a JWT provider that does not support JWK (JSON Web Key) or is signed by using the HMAC-SHA256 secret key algorithm, you must specify different attributes on the `openidConnectClient` element.

## Alternate process

1. If the public key cannot be retrieved from a JWK endpoint, it can be stored locally and you need to specify the `trustStoreRef` and the `trustAliasName` attributes in the `openidConnectClient` element. These attributes specify the truststore that contains the public certificate, and the label of the certificate. For example,

```
<openidConnectClient id="RS"
  clientId="RS-JWT-CICS" inboundPropagation="required"
  signatureAlgorithm="RS256"
  trustStoreRef="JWTrustStore"
  trustAliasName="JWTTrustCert" userIdentifier="sub"
  mapIdentityToRegistryUser="false" issuerIdentifier="idg"/>
```

For more information about keystores, see [Keystores](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

2. If the JWT is signed by using a shared secret key with the HMAC-SHA256 algorithm, you define the shared secret key on the `sharedKey` or `clientSecret` attributes.

For more information about these alternative `openidConnectClient` attributes, see [OpenID Connect Client](#).

## Designing security for Link to Liberty applications

Address the various security considerations when you plan to deploy a Java EE, Spring Boot or CDI application that uses Link to Liberty.

The Link to Liberty feature enables a CICS program to call into a Java application that runs in a Liberty JVM server by using the EXEC CICS LINK command. The calling program can be written in any CICS supported language and can also run on a remotely connected CICS region. The Java application can be an Enterprise Java application, a Spring Boot application or a CDI application. Figure 126 on page 274 shows a high-level view of how CICS and Liberty security mechanisms work together to address the security principles for this scenario.

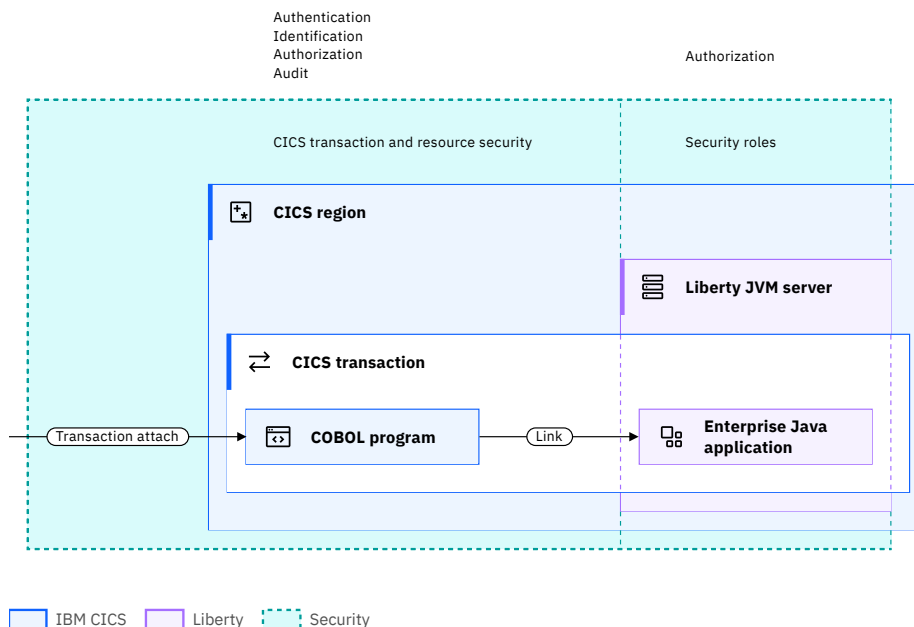


Figure 126. Security for Link to Liberty applications

When you are designing a solution for securing Link to Liberty, consider the implications for:

- [Authentication and identification](#)

- [Authorization](#)

## Authentication and identification

### How does authentication and identification work?

In CICS, both authentication and identification are determined by CICS transaction security. In a Link to Liberty scenario, authentication occurs during the transaction attach phase, which is before the calling task links to the Java program. Then, when the task links to the Java application, it runs under the authenticated CICS task user ID.

The CICS task user ID is used for the lifetime of a CICS task. If the CICS Liberty security feature (`cicsts:security-1.0`) is enabled in the Liberty JVM server, the CICS task user ID is pushed into the Liberty server and asserted as the Java security subject. If the CICS Liberty security feature is not enabled, then the Liberty unauthenticated user ID is used as the Java security subject.

### Which user registry to use?

A RACF user registry in Liberty is recommended because Liberty validates the Java security subject that is passed in from CICS is present in the Liberty user registry. Although other user registries can be used, if the CICS task user ID is not present in the Liberty user registry, then the Java security subject is set to the Liberty unauthenticated user ID.

## Authorization

### How to authorize Link to Liberty requests?

The following authorization checks can be performed:

- Before CICS links to the Java program, CICS transaction security can be used to protect access to the calling transaction. Transaction security grants authority to run the entire transaction. The entire transaction consists of the initial program and any further programs or other CICS resources such as files or databases that the initial program calls.
- When the CICS program links to the Java program, an authorization check can be performed by using CICS resource security. This controls access to the CICS program resource that is autoinstalled based on the name that is specified in the `@CICSProgram` annotation in the Enterprise Java application. Resource security can be useful when a large group of users are authorized to run a transaction and you need to restrict the access to the Java program to selected users. However, resource security needs to be specified on all CICS programs that are used by the transaction as it is enabled at the transaction level.
- A more granular authorization check can be performed by using security roles to ensure that the Java security subject is authorized to access the specified security role. However, security roles only function for EJB components, and do not function when you link to a Java POJO.

To enable security role authorization for Link to Liberty applications, the `cicsts:security-1.0` feature must be installed and the Liberty server must be configured to use SAF role authorization. Then, the EJB application declares within the code which roles have access to specific Java methods by using the `@RolesAllowed` annotation. Access to the Java method is then authorized by Liberty mapping the security role to an EJBROLE profile and then granting the authenticated subject user ID access to the mapped EJBROLE resource profile.

In this example, the calling program links to a Java program by using Link to Liberty. The transaction is protected by using CICS transaction security and the Java security subject is set to the Liberty unauthenticated user ID.

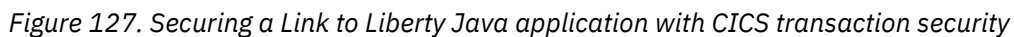


Figure 127 on page 276 shows the following security flow:

1. A request is made to attach a transaction (*tranA*) and CICS transaction security is used to verify that the CICS task user ID is authorized to run the transaction. CICS validates the authorization by calling RACF. RACF verifies that a group to which the task user ID belongs has READ authority to the member list (*memberListA*) containing *tranA* in the GCICSTRN class.
2. The calling program is run as the initial program for the CICS transaction. This program issues a LINK command to the Java program, which is named by using the **@CICSProgram** annotation on the target method of the Enterprise Java application.
3. The CICS task user ID is then used for any further CICS resource security checks to CICS resources accessed by the JCICS API. For example, VSAM files, TS queues or LINKs to other CICS programs.

At step 2, resource security can also be used to control whether the CICS task user ID is authorized to run the initial calling program and the Java program. For more information, see [Resource security](#).



## Related information

[Security for Java applications](#)

### **Configuration example: Securing a Link to Liberty Java application with CICS transaction security**

Configure CICS transaction security authorization for a Link to Liberty application.

## Before you begin

This configuration task is based on the security scenario [Design example: Securing a Link to Liberty Java application with CICS transaction security](#).

Before you begin this task, you need to know:

- The RACF user ID that is used to authenticate. This user ID must exist and have an OMVS segment.
- If your CICS region uses a security prefix.

You must complete the following tasks:

- Define a Java program entry point within the Enterprise Java application by using the **@CICSProgram** annotation.
- Enabled Link to Liberty by adding the **cicsts:link-1.0** feature to the Liberty server.xml.

You must have authorization to issue the RACF RDEFINE and PERMIT commands.

## About this task

In this task, you configure CICS transaction security for an Enterprise Java application that uses Link to Liberty.

This task assumes the following definitions:

- The user that is accessing the application has a RACF user ID *taskUserid*.
- The CICS transaction ID used to run the request is *tranA* and the initial program for this transaction is *progA*.
- The CSD group name that is used for the CICS transaction ID is *groupA*.
- The **cicsts:security-1.0** feature is installed into the Liberty JVM server.

## Procedure

1. Define and install a CSD definition for the CICS transaction ID to run the application.

```
DEFINE TRANSACTION(tranA) GROUP(groupA) PROGRAM(progA)
```

Create a member list (*memberListA*) that contains the *tranA* transaction in RACF and grant the authenticated user ID read access to the member list.

```
RDEFINE GCICSTRN memberListA ADDMEM(tranA) UACC(NONE)
PERMIT memberListA CLASS(GCICSTRN) ID(taskUserid) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Instead of you granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to this group.

If your CICS region uses a security prefix, all GCICSTRN profile names need to be prefixed with this value.

2. Edit the JVM profile for the Liberty JVM server. Add the following property to disable the synchronization of the CICS task user ID with the Java security subject. This configuration gives improved performance for scenarios that do not require authorization by using the Java subject.

```
-Dcom.ibm.cics.jvmserver.wlp.security.subject.create=false
```

3. Restart the JVM server.
4. Validate that Link to Liberty subject creation is disabled by viewing the JVM server log for the following message:  

```
[ECBListener] @Info : static initializer() - Liberty server is configured to bypass Java security Subject creation on Link-to-Liberty operations
```
5. Deploy the Java application by using a CICS bundle resource (see [Deploying a CICS bundle in the CICS Explorer product documentation](#)).
6. Deploy the calling program (*progA*) into the CICS region.

## Results

The user with RACF user ID *taskuserid* is authorized to access the transaction *tranA* including all the CICS programs that it can potentially start.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **All** tab and set the **Transaction ID** field to *tranA* that is defined in step 1 (or the transaction ID that initiates the task). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Designing security for JCICSX applications

Address the various security considerations when you plan to use the JCICSX Liberty feature to run JCICSX commands from a local development environment.

The JCICSX server is a feature of a Liberty JVM server that supports requests to run JCICSX commands from a local development environment. This is useful for remote testing of CICS Java applications. An HTTP connection is established from the JCICSX client in the local development environment to the JCICSX server running in a Liberty JVM server. These connections can be secured by using the standard facilities for authentication, authorization, and confidentiality provided by Liberty.

When you are designing a solution to secure the JCICSX server that is deployed to a Liberty JVM server, consider the implications for:

- [“Authentication and identification” on page 278](#)
- [“Authorization” on page 279](#)
- [“Confidentiality and integrity” on page 279](#)
- [“Trust” on page 280](#)

## Authentication and identification

### Which authentication mechanism to use?

TLS client authentication is the default authentication method for the JCICSX server and can also be used to establish trust between the local development environment and the JCICSX server. When you map a client certificate to a RACF user ID, you can use RACF certificate name filtering, or import the client certificate into the RACF registry as a trusted certificate.

#### Best practices:

When you need to authenticate many clients by certificate, it is recommended that you use certificate name filtering as this avoids having to store and administer too many client certificates.

When you are using basic authentication, it is recommended that you use a HTTPS connection to the Liberty server, otherwise basic authentication credentials are sent over the network in cleartext.

### Which user registry to use?

A SAF registry, for example RACF, is preferred by many CICS customers including those using CICS Liberty. RACF is secure, well understood and offers strong management protocols in an z/OS

environment. However, using an LDAP registry might be more convenient if all users are already present in an enterprise-wide LDAP registry.

### **What RACF user ID is used for running the CICS programs and accessing CICS resources?**

- Provided the `cicsts:security-1.0` feature is installed, and the user is authenticated with either TLS client authentication or basic authentication, then the RACF user ID under which the CICS task runs, will be set based on the Java security subject established by the Liberty authentication process.
- If instead, an unauthenticated subject results from the Liberty authentication process, then the default CICS user ID is used as the CICS task user id. The URIMAP supplied for the JCICSX server does not specify a default user ID.

## **Authorization**

### **Which option to use for authorizing web requests?**

The JCICSX server runs under the transaction CJXA by default. To authorize access to the JCICSX server from a traditional CICS perspective, you can configure CICS transaction security for CJXA. In terms of command security and resource security, CJXA is defined with `CMDSEC(N0)` and `RESSEC(N0)` because JCICSX remote development is used for testing. You might want to test command security and resource security, for example, the access to individual CICS resources such as links to programs or access to VSAM files. In this case, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. Note that this configuration impacts security settings of other transactions too. By using CICS transaction and resource security, you can reuse existing CICS authorization procedures.

Alternatively, or complementary to CICS transaction security, you can control access to the JCICSX server application using Liberty's SAF authorization and Enterprise Java's security role authorization. This approach uses the inbuilt role JCICSXUSER. For the JCICSX server, the security role authorization does not offer any more granular authorization than is available with CICS transaction security. Unless there are other applications in your server that employ SAF role authorization, there is no distinct advantage to using security role authorization in this scenario.

## **Confidentiality and integrity**

### **Which TLS implementation to use?**

You can implement TLS by using JSSE or AT-TLS.

Because JSSE is a Java TLS implementation, the TLS handshake and data encryption processing are zIIP eligible.

AT-TLS processing is not zIIP eligible but offers the benefit that the TLS policy configuration can be centralized for all z/OS subsystems. However, CICS Liberty is an unaware AT-TLS application, which means that use of AT-TLS imposes the following restrictions:

- A client cannot use a certificate to authenticate with a web application when the connection between the client and the Liberty JVM server is secured by using AT-TLS.
- The `transport-guarantee` element in `web.xml` has no effect when you use AT-TLS.
- Although AT-TLS secures the connection with HTTPS, it is still the HTTP port of the Liberty server that is used from the view of Liberty web applications.

### **Which type of keystore and truststore to use?**

CICS Liberty supports SAF key rings, Public Key Cryptography Standards #12 (PKCS12), and Java KeyStores (JKS). SAF key rings are recommended because they benefit from the additional protection of SAF, for example, [RACF](#).

### **Which cypher suite to use?**

It is recommended that you use the most recent version of TLS that is also supported by the partner system. The most recent version of software usually includes fixes to previous versions and brings enhancements such as support for stronger cipher suites. To control the TLS version that is

supported by the Liberty JVM server, you can update the `sslProtocol` attribute on the `ssl` element in `server.xml`. To modify the list of ciphers that are supported by the server, you can specify the list of ciphers on the `enabledCiphers` attribute on the `ssl` element.

## Trust

### How to ensure that the remote system that connects to a Liberty JVM server is a trusted server?

TLS client authentication can be used so that the server provides a client certificate that is validated by the Liberty JVM server. Successful client authentication requires that the certificate authority (CA) that signed the client certificate is considered trusted by the Liberty JVM server. To be considered trusted, the certificate of the CA must be in the Liberty server's truststore. For more information, see [Configuring TLS for a Liberty JVM server by using RACF](#).

### Design example: Securing the JCICSX server using basic authentication

In this example, CICS transaction security is used to authorize access to run JCICSX commands in CICS. The supplied URIMAP DFHJXSU that matches the JCICSX HTTP requests ensures all requests run under the CJXA transaction ID. This transaction is protected such that the authenticated client user ID must be authorized to run the transaction.

For more information about how to configure security for this scenario, see [Configuration example: Securing the JCICSX server using basic authentication](#).

**Note:** This example uses only CICS transaction security because JCICSX remote development is used for testing and that command security and resource security is disabled for CJXA by default. If you want to test security, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. This impacts security settings of other transactions too. For more information, see [Designing security for JCICSX applications](#).

[Figure 128 on page 281](#) shows an overview of the scenario.

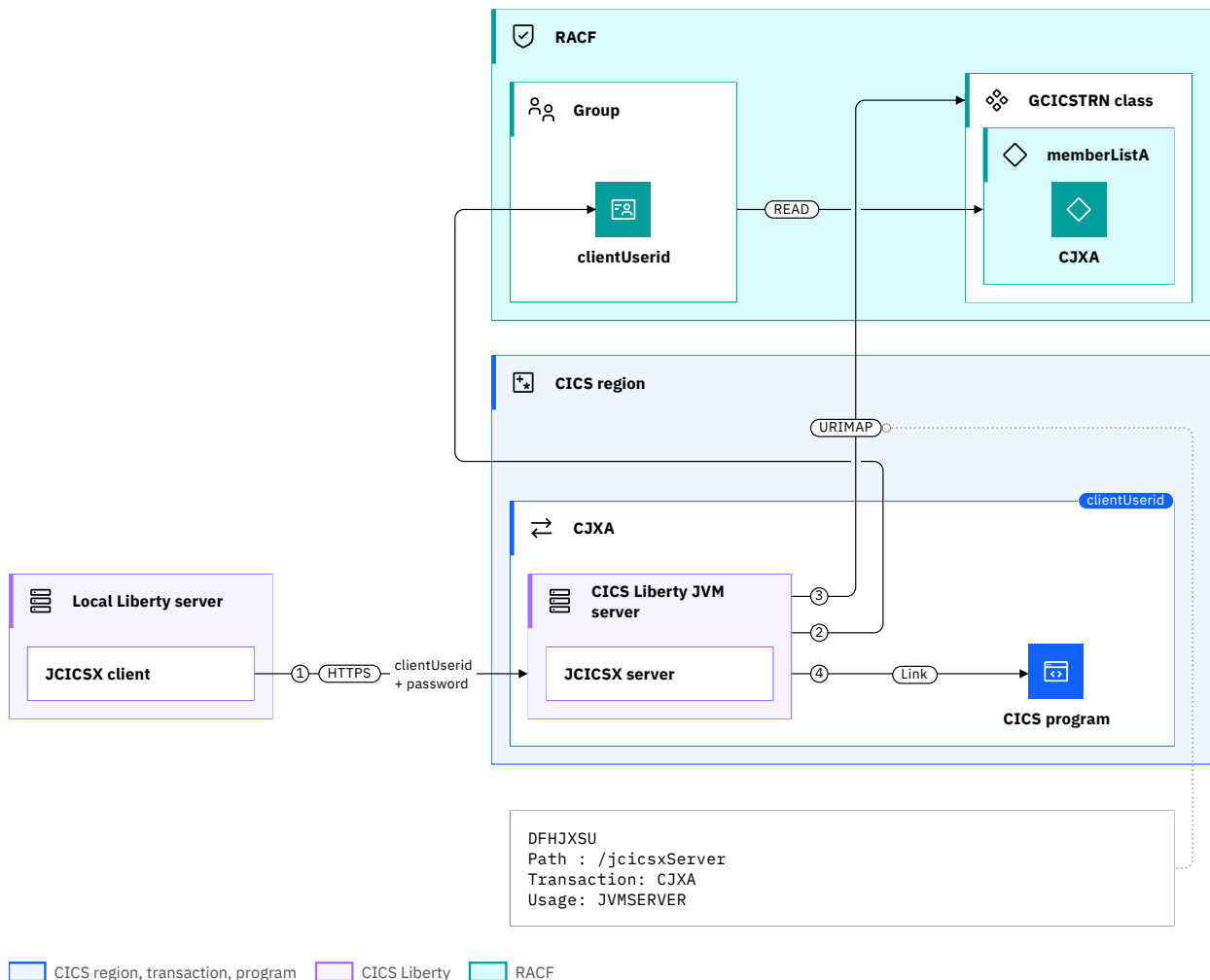


Figure 128. Securing JCICSX with basic authentication

1. The JCICSX client sends an HTTPS request with the *clientUserid* and *password* in the HTTP Authorization header to the Liberty JVM server.
- Important:** It is recommended that you use HTTPS to protect the request. Otherwise the basic authentication credentials will flow across the network in cleartext.
2. To authenticate the request, the Liberty JVM server validates the *clientUserid* and *password* against the configured SAF user registry (RACF) and sets the Java security subject to the *clientUserid*. More specifically, the authentication is delegated to Liberty's authentication process, which in turn uses the Liberty angel process to perform the authorized services. If successful, the Java security subject, and subsequently the CICS task user ID are set to the *clientUserid*.
  3. CICS transaction security is used to authorize the request. The URIMAP DFHJXSU matches the request URL and determines that it runs under the CJXA transaction. CICS checks that the *clientUserid* is authorized to run the transaction by calling RACF to verify that the *clientUserid* has READ authority to the CJXA transaction in RACF. As shown in the diagram, the *clientUserid* is connected to a group that has READ authority to a member list (*memberListA*) containing CJXA in the GCICSTRN class.
  4. The JCICSX HTTP requests run using the CJXA transaction under the security context of the *clientUserid*. Any JCICSX calls to CICS resources such as a link to a CICS program use the *clientUserid* as the CICS task user ID.

## Configuration example: Securing the JCICSX server using basic authentication

Configure basic authentication for JCICSX requests and check the RACF user ID is authorized by using CICS transaction security. Optionally, if there are other web applications that require SAF role authorization, configure the required EJBROLE security profiles.

This configuration task is based on the example security scenario: [Design example: Securing the JCICSX server using basic authentication](#).

**Note:** This example uses only CICS transaction security because JCICSX remote development is used for testing and that command security and resource security is disabled for CJXA by default. If you want to test security, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. This impacts security settings of other transactions too. For more information, see [Designing security for JCICSX applications](#).

## Before you begin

Before you begin this task, you must complete the following tasks in CICS:

- Ensure CICS security is activated by setting the SIT parameter SEC=YES.
- Connect the Liberty server to the angel process. See [Configuring and starting the Liberty angel process](#) and [Connecting a Liberty JVM server to the angel process](#).
- Enable basic authentication for each RACF user ID. See [Configuring Liberty to use a SAF user registry](#).
- Configure a Liberty JVM server with the JCICSX server feature. See [Configuring the environment for JCICSX in Java development using JCICSX](#).

Developers need to configure the local development environment to run Java code locally and to make remote calls for JCICSX. See [Configuring the environment for JCICSX in Java development using JCICSX](#).

You need to know the RACF user ID that is used to authenticate. This user ID must exist and have an OMVS segment.

You must have authorization to issue the RACF **RDEFINE** and **PERMIT** commands.

## About this task

In this task, you configure the JCICSX server application to use a SAF registry, basic authentication, and CICS transaction security to check that the *clientUserid* is authorized to use the JCICSX server. This task assumes that *clientUserid* is the user ID that is used to authenticate.

By default, role-based authorization is bypassed because the JCICSX server's web application is configured to allow access to its role for any authenticated user ID.

## Procedure

1. On the local development workstation, the developer adds the *clientUserid* credentials to the local Liberty server `server.xml`, for example:

```
<usr_jcicsxClient serverUri="https://hostname:port">
  <basicAuthentication user="clientUserid"
    password="{aes}ADwac72WxpSCr2YDUv3hHgjf0a0moXZDj626MmM4DbtT"/>
</usr_jcicsxClient>
```

**Important:** It is recommended that you encrypt the password by using the [securityUtility](#) command provided in the `bin` directory of the local Liberty server. However, encrypting the credentials in this manner does not stop them being reused in other Liberty environments.

2. In the CICS Liberty JVM server:
  - a. Ensure the `cicsts:security-1.0` feature is installed in the Liberty server's feature manager list.  
This ensures that the Liberty authenticated Java security subject is synchronized with the CICS Task user ID.

- b. By default, the JCICSX server only accepts authentication with a valid certificate. To allow users to be authenticated with a username and password, add the following line to the `server.xml` file:

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

If this step is not performed, users will get an HTTP 403 error when they attempt to access the server with a username and password.

**Note:** Be aware that this setting affects all web applications running in this Liberty server.

- c. For CICS transaction security, authorize the *clientUserId*, or a group of which it is a member, to run the CICS transaction CJXA.

For more information, see [Configuring CICS transaction security for a Liberty JVM server](#).

- d. Enable, or disable and re-enable, the Liberty JVM server if it was already running.

## Results

JCICSX requests to CICS are authenticated by Liberty using a RACF user ID and password. Authorization is determined by CICS transaction security.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example:

1. In the **Regions** view, select **Add Security Request Recording** from the pop-up menu.
2. In the dialog, select the **JVM Server** tab and set the **Transaction ID** field to the CJXA transaction ID.

For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Optional configuration for SAF role authorization

This task assumes the following definitions:

- The `<safAuthorization>` element has been added to the Liberty's `server.xml` file.
- *safProfilePrefix* is the SAF profile prefix for the Liberty server.
- `com.ibm.cics.wlp.jcicsxserver` is the name of the JCICSX web application ID.
- JCICSXUSER is the security role that protects the application authorization.

You can configure SAF role authorization as follows:

1. Authorize the *clientUserId* to access the JCICSX server:
  - a. Create a RACF EJBROLE resource profile to match the default role mapper pattern for EJBROLE profiles, for example, *safProfilePrefix*, `com.ibm.cics.wlp.jcicsxserver`, and JCICSXUSER:

```
RDEFINE EJBROLE
  safProfilePrefix.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER
  UACC(NONE)
```

- b. Grant the *clientUserId* READ permission to the EJBROLE:

```
PERMIT safProfilePrefix.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER
  ACCESS(READ)
  CLASS(EJBROLE)
  ID(clientUserId)
```

- c. Refresh the EJBROLE class in RACF:

```
SETROPTS RACLIST(EJBROLE) REFRESH
```

For detailed instructions on these steps, see [Configuring SAF authorization with an EJBROLE](#).

2. Enable, or disable and re-enable, the Liberty JVM server if it was already running, to pick up the changes made to the RACF class profiles.

## What to do next

The application developer can add a JCICSX resource to the local Liberty JVM server to check whether the connection is working. Samples can be found at [JCICSX samples in GitHub](#).

## Design example: Securing the JCICSX server using TLS client authentication

In this example, a mutual TLS connection is established between the local Liberty server and the CICS Liberty server. The authentication is performed with the TLS client certificate. CICS transaction security is used to authorize access to run JCICSX commands in CICS. The supplied URIMAP DFHJXSU that matches the JCICSX HTTP requests ensures all requests run under the CJXA transaction ID. This transaction is protected such that the authenticated client user ID must be authorized to run the transaction.

For more information about how to configure security for this scenario, see [“Configuration example: Securing the JCICSX server using TLS client authentication”](#) on page 285.

**Note:** This example uses only CICS transaction security because JCICSX remote development is used for testing and that command security and resource security is disabled for CJXA by default. If you want to test security, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. This impacts security settings of other transactions too. For more information, see [Designing security for JCICSX applications](#).

The following diagram shows an overview of the scenario.

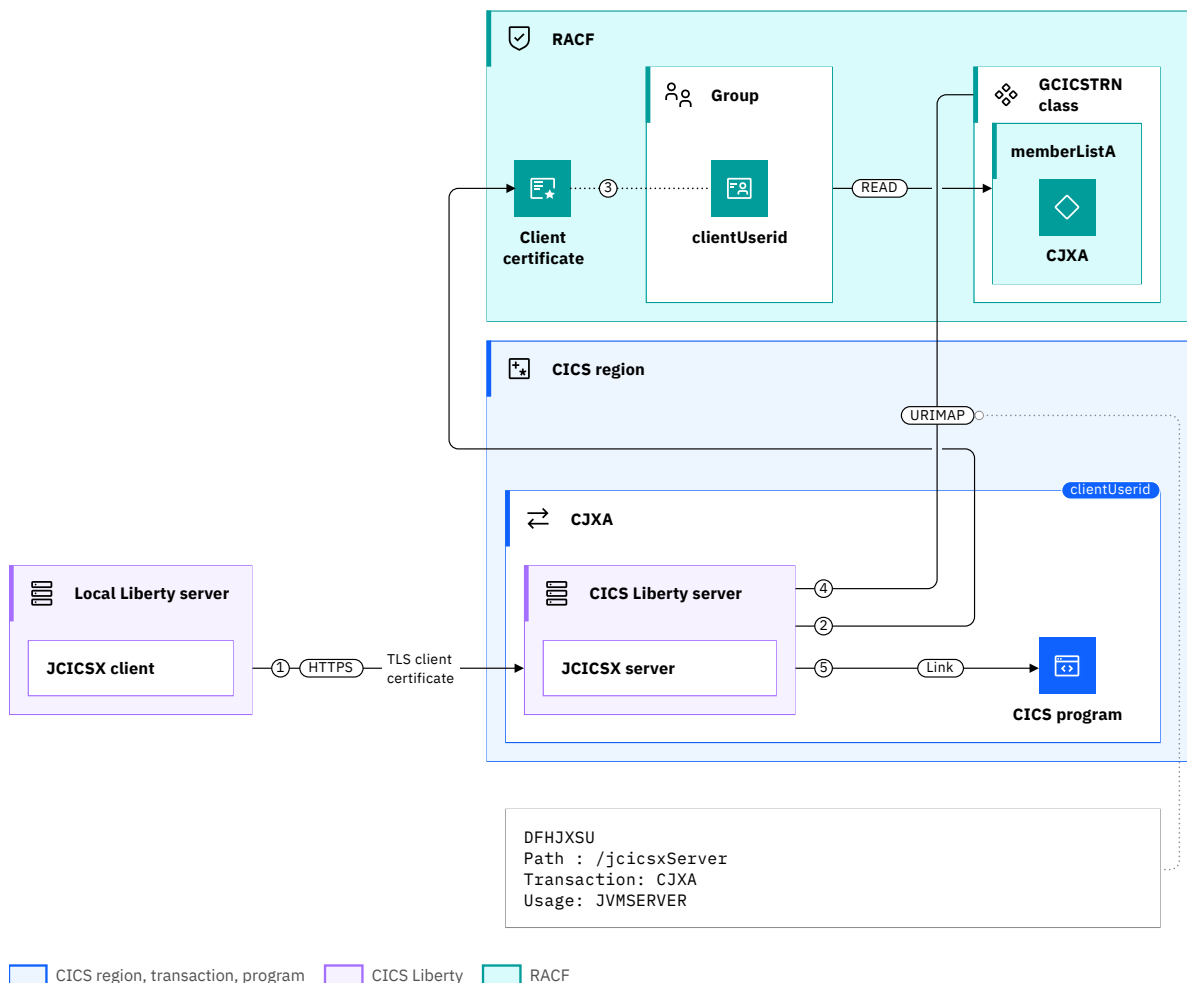


Figure 129. Securing JCICSX with TLS client authentication



1. The JCICSX client establishes an HTTPS connection with the Liberty server that is running within the JVM server in CICS. During the TLS handshake, the server is configured to request the client certificate. Both parties validate they trust each other: the client verifies the server certificate, and the server verifies the client certificate.
2. Once the HTTPS connection is established, the Liberty server uses the client certificate to authenticate the user. The Liberty server sends a request to the SAF registry (in this case, RACF) in order to get the user identity.
3. The SAF registry attempts to map the certificate to a user identity. If the mapping is successful, the *clientUserId* is returned to the Liberty server.  
Upon successful authentication, the Java security subject, and subsequently the CICS task user ID are set to the *clientUserId*.
4. CICS transaction security is used to authorize the request. The URIMAP DFHJXSU matches the request URL and determines that it runs under the CJXA transaction. CICS checks that the *clientUserId* is authorized to run the transaction by calling RACF to verify that the *clientUserId* has READ authority to the CJXA transaction in RACF. As shown in the diagram, the *clientUserId* is connected to a group that has READ authority to the member list (*memberListA*) that contains the CJXA transaction, in the GCICSTRN class.
5. The JCICSX HTTP requests run using the CJXA transaction under the security context of the *clientUserId*. Any JCICSX calls to CICS resources such as a link to a CICS program use the *clientUserId* as the CICS task user ID.

### **Configuration example: Securing the JCICSX server using TLS client authentication**

Configure TLS client authentication for JCICSX requests and check the authenticated RACF user ID is authorized by using CICS transaction security. Optionally, if there are other web applications that require SAF role authorization, configure the required EJBROLE security profiles.

This configuration task is based on the example security scenario: [“Design example: Securing the JCICSX server using TLS client authentication”](#) on page 284.

**Note:** This example uses only CICS transaction security because JCICSX remote development is used for testing and that command security and resource security is disabled for CJXA by default. If you want to test security, set SIT parameters **CMDSEC=ALWAYS** and **RESSEC=ALWAYS**. This impacts security settings of other transactions too. For more information, see [Designing security for JCICSX applications](#).

### **Before you begin**

Before you begin this task, you must complete the following tasks in CICS:

- Ensure CICS security is activated by setting the SIT parameter SEC=YES.
- Connect the Liberty server to the angel process. See [Configuring and starting the Liberty angel process](#) and [Connecting a Liberty JVM server to the angel process](#).
- Enable TLS client authentication in the Liberty JVM server. See [Configuring TLS client authentication for a Liberty JVM server by using RACF](#).
- Configure a Liberty JVM server with the JCICSX server feature. See [Configuring the environment for JCICSX in Java development using JCICSX](#).

Developers need to configure the local development environment to run Java code locally and to make remote calls for JCICSX. See [Configuring the environment for JCICSX in Java development using JCICSX](#).

You need to know:

- The subject value of the client certificate to be mapped.
- The user ID to which the TLS client certificate is mapped. This user ID must exist and have an OMVS segment.

You must have the following authorization:

- Authorization to issue **RACDCERT** commands to manage RACF key rings and certificates.

For more information about the **RACDCERT** commands and the authorization that is required, see [RACDCERT \(Manage RACF digital certificates\)](#).

- Authorization to issue the RACF **RDEFINE** and **PERMIT** commands.

## About this task

In this task, you configure the JCICSX server application to use a SAF registry, TLS client authentication, and CICS transaction security to check that the *mappedUserid* is authorized to use the JCICSX server. The TLS client certificate is mapped to a user ID by using a certificate name filter in this task. Other mapping methods exist and can be used. This task assumes that the client certificate has already been generated, and that *mappedUserid* is the user ID that is used to authenticate.

By default, role-based authorization is bypassed because the JCICSX server's web application is configured to allow access to its role for any authenticated user ID.

## Procedure

During Steps “1” on page 286 to “6” on page 287, the system programmer creates the certificate name filter that allows the client certificate to be mapped to *mappedUserid*, and handles the RACF definitions.

During Steps “7” on page 287 to “10” on page 288, the developer creates the client keystore and configures the local Liberty server to use it.

In Step “11” on page 288, the system programmer configures the Liberty JVM server in CICS.

1. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

2. Map the TLS certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID:

```
RACDCERT MAP ID(mappedUserid)
SDNFILTER('CN=myClient.host.com.O=IBM.C=US')
WITHLABEL('clientMapLabel')
```

### *mappedUserid*

Is the RACF user ID that is associated with the certificate name filter.

### **CN=myClient.host.com.O=IBM.C=US**

Is the certificate subject distinguish name.

### **clientMapLabel**

Is the certificate name filter label.

**Important:** In this command, the syntax of the SDNFILTER line is significant. Use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#).

3. Refresh the DIGTNMAP RACF class for the changes to take effect:

```
SETROPTS RACLIST(DIGTNMAP) REFRESH
```

4. Export the JCICSX server CA certificates so that the client can add them in its keystore.

Use the following command:

```
RACDCERT EXPORT(LABEL('JCICSX-CA'))
CERTAUTH
DSN('hlq.JCICSX.ROOTCA.PEM')
FORMAT(CERTB64)
```

## **JCICSX-CA**

Is the CA certificate label. It needs to match the certificate authority you want to export.

### **hlq.JCICSX.ROOTCA.PEM**

Is the data set where the CA certificate is exported.

Repeat the step for all necessary CA certificates and use a different export data set each time.

5. Transfer the CA certificates to the local workstation. The public certificate exported in CERTB64 requires a transfer in text mode.
6. If the CA certificates that signed the client certificate are not in RACF, import the client CA certificates.

Transfer the CA certificates to z/OS first, usually in text mode.

Then add them in RACF by using the following command:

```
RACDCERT ADD(hlq.JCICSX.CLIENT.CA.PEM)
          CERTAUTH
          TRUST
          WITHLABEL('JCICSX-CLIENT-CA')
```

### **hlq.JCICSX.CLIENT.CA.PEM**

Is the data set that contains the client CA certificate.

## **JCICSX-CLIENT-CA**

Is the client CA certificate label. It needs to match the certificate authority you want to add.

And connect them to the JCICSX server key ring:

```
RACDCERT CONNECT(CERTAUTH
                  LABEL('JCICSX-CLIENT-CA')
                  RING(Keyring.CICS1)
                  USAGE(CERTAUTH))
OWNER(regionUserId)
```

## **JCICSX-CLIENT-CA**

Is the client CA certificate label. It needs to match the certificate authority you want to connect.

### **Keyring.CICS1**

Is the key ring used by the JCICSX server.

### **regionUserId**

Is the CICS region user ID, who owns the key ring.

Refresh the cache for the changes to take effect:

```
SETROPTS RACLIST(DIGTCERT DIGTRING) REFRESH
```

7. On the local development workstation, p12 keystore files can be used to store certificates.

The keystore that contains the TLS client certificate can be used to store the JCICSX server CA certificates. Alternatively a dedicated keystore can be used. For simplicity, this task uses a single keystore to keep the local Liberty server certificate, the JCICSX client certificate and the JCICSX server CA certificates.

Liberty keystore files are usually located in the server resource folder. For example, if the Liberty server is named `defaultServer`, the security resource folder is `defaultServer/resources/security`.

Identify the resource folder used to store p12 files, and use a command line terminal to navigate to it. The following steps assume you are in the correct folder to operate on the keystore. Otherwise use an absolute path in the keytool command to point to the keystore.

8. On the local development workstation, the developer adds the JCICSX server CA certificates to its keystore.

Use the following keytool command to add a certificate to a keystore:

```
keytool -import -alias ROOTCA -file JCICSX.ROOTCA.pem -keystore JCICSX.CLIENT.p12
```

### ROOTCA

Is the alias given to the certificate in the keystore.

### JCICSX.ROOTCA.pem

Is the path to the certificate file to be added.

### JCICSX.CLIENT.p12

Is the path to the p12 file that stores the certificates. By default, this file is stored in the directory you navigated to in Step “7” on page 287.

To perform the keytool operations, the command prompts you for the p12 file password.

When the command asks whether to trust the certificate, reply yes.

Repeat the step for all necessary CA certificates, each time using a different alias.

9. On the local development workstation, verify the content of the p12 file with the following command:

```
keytool -list -v -keystore JCICSX.CLIENT.p12
```

10. On the local development workstation, in the `server.xml` configuration file, the developer configures the keystore and adds the JCICSX client server URI :

```
<usr_jcicsxClient serverUri="https://hostname:port"/>
<keyStore id="defaultKeyStore" location="JCICSX.CLIENT.p12"
  password="{aes}ALcGsFoto940TrpFAp0nuz7WzcpzDyjMDeErXIM6aHo+" type="PKCS12"/>
```

The `serverUri` should be set with the JCICSX server hostname and port, and the `keyStore` location points to the client keystore.

**Important:** It is recommended that you encrypt the password by using the `securityUtility` command provided in the `bin` directory of the local Liberty server. However, encrypting the credentials in this manner does not stop them being reused in other Liberty environments.

11. In the CICS Liberty JVM server:

- a. Ensure the `cicsts:security-1.0` feature is installed in the Liberty server's feature manager list.

This ensures that the Liberty authenticated Java security subject is synchronized with the CICS Task user ID.

- b. By default, the JCICSX server only accepts authentication with a valid certificate, but TLS mutual authentication is not enabled by default.

In the `server.xml` file, ensure that the `ssl` configuration has `clientAuthentication` set to `true`. Otherwise, users get an HTTP 403 error when they attempt to access the server.

The `server.xml` configuration file might look similar to this example:

```
<featureManager>
  <feature>cicsts:core-1.0</feature>
  <feature>cicsts:security-1.0</feature>
  <feature>transportSecurity-1.0</feature>
</featureManager>

<safRegistry enableFailover="false" id="saf"/>

<httpEndpoint host="*" httpPort="9080" httpsPort="9443" id="httpEndpoint1"/>
<sslDefault sslRef="customSSLConfiguration"/>
<ssl id="customSSLConfiguration" keyStoreRef="SAFKeyStore" sslProtocol="TLSv1.2"
  clientAuthentication="true"/>
<keyStore id="SAFKeyStore" fileBased="false"
  location="safkeyring:///Keyring.CICS1"
  password="password" readOnly="true"
  type="JCERACFKS" />
```

Your `server.xml` file might contain more configuration elements.

- c. For CICS transaction security, authorize the *mappedUserid*, or a group of which it is a member, to run the CICS transaction CJXA.

For more information, see [Configuring CICS transaction security for a Liberty JVM server](#).

- d. Enable, or disable and re-enable, the Liberty JVM server if it was already running.

## Results

JCICSX requests to CICS are authenticated by Liberty using a RACF client certificate. Authorization is determined by CICS transaction security.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example:

1. In the **Regions** view, select **Add Security Request Recording** from the pop-up menu.
2. In the dialog, select the **JVM Server** tab and set the **Transaction ID** field to the CJXA transaction ID.

For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Optional configuration for SAF role authorization

This task assumes the following definitions:

- The `<safAuthorization>` element has been added to the Liberty's `server.xml` file.
- *safProfilePrefix* is the SAF profile prefix for the Liberty server.
- `com.ibm.cics.wlp.jcicsxserver` is the name of the JCICSX web application ID.
- JCICSXUSER is the security role that protects the application authorization.

You can configure SAF role authorization as follows:

1. Authorize the *mappedUserid* to access the JCICSX server:
  - a. Create a RACF EJBROLE resource profile to match the default role mapper pattern for EJBROLE profiles, for example, *safProfilePrefix*, `com.ibm.cics.wlp.jcicsxserver`, and JCICSXUSER:

```
RDEFINE EJBROLE
  safProfilePrefix.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER
  UACC(NONE)
```

- b. Grant the *mappedUserid* READ permission to the EJBROLE:

```
PERMIT safProfilePrefix.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER
  ACCESS(READ)
  CLASS(EJBROLE)
  ID(mappedUserid)
```

- c. Refresh the EJBROLE class in RACF:

```
SETROPTS RACLIST(EJBROLE) REFRESH
```

For detailed instructions on these steps, see [Configuring SAF authorization with an EJBROLE](#).

2. Enable, or disable and re-enable, the Liberty JVM server if it was already running, to pick up the changes made to the RACF class profiles.

## What to do next

The application developer can add a JCICSX resource to the local Liberty JVM server to check whether the connection is working. Samples can be found at [JCICSX samples in GitHub](#).

## Configuring security for CICS Liberty

---

This section contains common tasks to configure CICS Liberty security. To understand what scenarios the tasks apply to, you can refer to the design examples. The design examples describe sample CICS Liberty security topologies to demonstrate recommended security configurations. Each design example links to a configuration example that highlights the steps needed to implement security for that example.

Design examples for CICS Liberty are included in each section of [Designing security for CICS Liberty applications](#).

### Configuring the angel process

You can use z/OS authorized services from CICS Liberty JVM servers. These services include the SAF registry and SAF role authorization services, the IFAUSAGE services, and the asynchronous I/O on z/OS services.

For more information about the list of z/OS authorized services, see [Enabling z/OS authorized services on Liberty for z/OS](#). A Liberty angel process is necessary to access these services as it validates whether a Liberty server can access a specific z/OS authorized service. For more information on which services a CICS Liberty needs, see [Designing application security for Liberty with the angel process](#).

For more information about the angel process, see [the angel process](#).

When you start a CICS Liberty server, it tries to connect to an angel process and validates the z/OS authorized services to which it has access. If a connection to an angel process is not possible, the server can be configured to stop itself since the applications might require the use of specific z/OS authorized services to function properly.

The following tasks outline the configuration steps to use the angel process.

### Configuring and starting the Liberty angel process

The Liberty angel process is a started task that allows Liberty servers to use z/OS authorized services. It's long-lived and can be shared among multiple Liberty servers.

#### Before you begin

You must be familiar with:

- The information in [the angel process](#).
- The restrictions that are related to [a named angel](#).

You need to know:

- The angel process version and the Liberty server version, as the angel must be at the same or later level of Liberty than the server.
- The location of the CICS USSHOME directory in zFS.
- Optionally, the name to assign to the angel process.
- The user ID that runs the angel process.

You must have authorization to issue the RACF **RDEFINE** command.

#### About this task

In this task, you configure and start an angel process, optionally providing a name.

This task assumes:

- Your External Security Manager is RACF.
- You have no existing named angel process that can be reused.

**Recommended:** Create a named angel to allow individual angel process to be updated without affecting other Liberty server instances on the LPAR. If you do want to create an unnamed default angel, then omit the NAME configuration.

## Procedure

1. Locate the sample JCL procedure for the started task in the CICS USSHOME directory  
For example: /usr/lpp/cicsts/cicsts61/wlp/templates/zos/procs/bbgzangl.jcl
2. Verify that the level of Liberty you are going to use for the process is at the same or later level of Liberty than the one bundled in CICS TS. This verification is only necessary if you want to use a different level of Liberty for the angel process than the one bundled with CICS TS.

For compatibility reasons, it is important to identify the latest level of Liberty available. This information can be available in the logs of running Liberty servers, or by looking up the README.txt file located in Liberty installation folders.

3. Copy the JCL procedure to a JES procedure library and modify it. To use the angel version provided by CICS TS, set the ROOT variable to the value of USSHOME/wlp.  
For example, ROOT=/usr/lpp/cicsts/cicsts61/wlp

Otherwise, specify the appropriate path to the level of Liberty selected in the previous step.

**Important:** The procedure name can be changed to distinguish the different named angels. You can directly set the name of the angel process in the NAME parameter instead of passing it as argument when you issue the START command.

The angel process name is 1 - 54 characters inclusive, and must use only the following characters: A-Z 0-9 ! # \$ % & ' ( ) \* + , - . / : ; < = ? @ [ \ ] ^ \_ ` { | } ~

4. Define the RACF STARTED profile to associate the angel user ID to the process.  
For example, with the default procedure name BBGZANGL and *angelUserId*:

```
REDEFINE STARTED BBGZANGL.* UACC(NONE) STDATA(USER(angelUserId))  
SETROPTS RACLIST(STARTED) REFRESH
```

5. Define a SERVER profile for the angel (BBG.ANGEL.*angelName* if you are using a named angel process, or BBG.ANGEL for the default angel) in the SERVER class.  
For example, with a named angel:

```
RDEFINE SERVER BBG.ANGEL.angelName UACC(NONE)  
SETROPTS RACLIST(SERVER) REFRESH
```

6. Define the RACF profiles that are needed for accessing z/OS authorized services. These profiles need to be defined only one time. This step can be skipped if the profiles are already defined. For more information about the z/OS authorized services, see [Enabling z/OS authorized services on Liberty for z/OS](#). For more information on which services a CICS Liberty needs, see [Designing application security for Liberty with the angel process](#).

- a) Create a SERVER profile for the authorized module BBGZSAFM:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
```

- b) Create a SERVER profile for the SAF registry and SAF role authorization services (SAFCRED):

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
```

- c) Create a SERVER profile for the IFAUSAGE services (PRODMGR). This profile allows authorized Liberty JVM servers to register and unregister from IFAUSAGE when the CICS JVM server is enabled and disabled:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
```

- d) Create a SERVER profile for the asynchronous I/O on z/OS (ZOSAIO) service. This profile allows authorized Liberty JVM servers to use the asynchronous TCP/IP sockets I/O on z/OS to improve performance:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAI0 UACC(NONE)
```

e) Refresh the SERVER resource:

```
SETROPTS RACLIST(SERVER) REFRESH
```

7. Start the angel process. In the following examples, [.identifier] indicates an optional identifier that can be up to 8 characters long.

a) To start the angel process as a named angel process, you can either code the NAME parameter on the operator START command, or set the NAME parameter in the procedure.  
For example,

```
START BBGZANGL[.identifier],NAME=angelName
```

b) To start the angel process without naming it, don't set the NAME parameter and use this command:

```
START BBGZANGL[.identifier]
```

8. Verify that the angel process started correctly by looking at the JES message log.

For example,

```
CWWKB0079I THE ANGEL BUILD LEVEL IS 2021.6.0.0 20210528-0200 / 2021.6.0.0 20210528-0200  
CWWKB0069I INITIALIZATION IS COMPLETE FOR THE angelName ANGEL PROCESS.
```

9. Optional: Enable printing of RACF audit messages for unauthorized access to SERVER profiles when Liberty servers check their permissions to access z/OS authorized services. This configuration is disabled by default. Specify SAFLOG=Y as a parameter to the START command, or set SAFLOG=Y in the angel procedure.

## Results

A (named) angel process is started on the LPAR and ready to be used by Liberty JVM servers.

## What to do next

You might want to connect a CICS Liberty server to the named angel that you created by following the steps in [Connecting a Liberty JVM server to the angel process](#).

You might also want to know what interactions are available with the angel process or how to verify its version by visiting [Managing a Liberty angel process](#).

## Connecting a Liberty JVM server to the angel process

A CICS Liberty server needs to have the appropriate configurations and authorizations to successfully connect to an angel process.

### Before you begin

You must be familiar with the information in [the angel process](#).

You need to know:

- The name of the angel process to which you are connecting.
- The z/OS authorized services that are required by the Liberty server.

You must have:

- Authorization to issue the RACF **PERMIT** command.
- An angel process that is already configured and started, otherwise follow the steps in [Configuring and starting the Liberty angel process](#).



## About this task

In this task, you configure the connection of a CICS Liberty server to an angel process, and grants authorization to the Liberty server to use z/OS authorized services.

- The name that is used by the angel process is *angelName*.
- The CICS region user ID is *regionUserid*.

## Procedure

1. Specify the named angel process to which the Liberty server connects by setting the `com.ibm.ws.zos.core.angelName` property. If the Liberty server connects to the LPAR default angel process, this configuration is not needed and this step can be skipped.  
For example, if the angel name is *angelName* this value can be set in the JVM profile with:

```
-Dcom.ibm.ws.zos.core.angelName=angelName
```

2. Optionally, specify whether a connection to the angel is needed by setting the `com.ibm.ws.zos.core.angelRequired` property. This option is only recommended for advanced use-cases.  
For example, this property can be set in the JVM profile with:

```
-Dcom.ibm.ws.zos.core.angelRequired=true
```

The default value is `false`. When set to `true`, if the server cannot connect to an angel process during start-up, the server fails to start. This situation can be caused by lack of permissions to access the RACF SERVER class profiles, or if the angel process is not available. The use of this property allows a quicker and cleaner failure.

**Important:** After a number of unsuccessful attempts to connect to the angel process, an MVS system request is issued to prompt whether the Liberty server should continue to retry the connection or should be stopped. Not responding to this prompt can cause unexpected behavior in different situations like when the JVM server status changes or the CICS region stops.

3. Optionally, specify a comma-separated list of z/OS authorized services that are required by the applications, by setting the `com.ibm.ws.zos.core.angelRequiredServices` property. This configuration requires that the property `com.ibm.ws.zos.core.angelRequired` is set to `true`.  
For example, this configuration can be set in the JVM profile with:

```
-Dcom.ibm.ws.zos.core.angelRequired=true  
-Dcom.ibm.ws.zos.core.angelRequiredServices=SAFCRED,PRODMGR,ZOSAIO
```

When not specified, the default behavior is for the server to request access to all z/OS authorized services. If access to any of the services that are requested is not authorized, the server continues to start. Specifying a list of authorized services indicates to the server which services can be used, and only access to these services is validated. If the server doesn't have access to one of the required services, it fails to start. The use of this property avoids printing unnecessary security violation messages.

4. Allow the CICS region user ID (*regionUserid*) to connect to the angel by granting READ access to the angel SERVER profile `BBG.ANGEL.angelName` (or `BBG.ANGEL` for the default angel).  
For example, with a named angel:

```
PERMIT BBG.ANGEL.angelName CLASS(SERVER) ACCESS(READ) ID(regionUserid)  
SETROPTS RACLIST(SERVER) REFRESH
```

5. Allow the CICS region user ID to access z/OS authorized services by granting READ access to the corresponding SERVER profiles. For more information about the list of available z/OS authorized services, see [Enabling z/OS authorized services on Liberty for z/OS](#).
  - a) Allow access to the authorized module `BBGZSAFM`:

```
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(regionUserid)
```

- b) Allow access to the RACF user registry and SAF authorization services (SAFCRED)

```
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(regionUserId)
```

- c) Allow access to the IFAUSAGE services (PRODMGR). This permission allows authorized Liberty JVM servers to register and unregister from IFAUSAGE when the CICS JVM server is enabled and disabled:

```
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(regionUserId)
```

- d) Allow access to the asynchronous I/O on z/OS (ZOSAI0) service. This permission allows authorized Liberty JVM servers to use the asynchronous TCP/IP sockets I/O on z/OS to improve performance:

```
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSAI0 CLASS(SERVER) ACCESS(READ) ID(regionUserId)
```

- e) Refresh and RACLIST the SERVER profiles:

```
SETROPTS RACLIST(SERVER) REFRESH
```

6. Restart the CICS Liberty server to ensure that the changes are active.

7. Verify that the Liberty server is connected to the expected angel and has access to the required services by checking the message .log file.

For example,

```
CWWKB0124I: Angel angelName is required, server was configured to wait up to 0 seconds to
connect to the targeted
CWWKB0122I: This server is connected to the CICSANGL angel process.
CWWKB0103I: Authorized service group KERNEL is available.
CWWKB0103I: Authorized service group PRODMGR is available.
CWWKB0103I: Authorized service group SAFCRED is available.
CWWKB0104I: Authorized service group LOCALCOM is not available.
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group WOLA is not available.
CWWKB0104I: Authorized service group ZOSAI0 is available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
CWWKB0104I: Authorized service group CLIENT.WOLA is not available.
```

Note that the name of the angel process and the availability of the z/OS authorized services might differ based on the configuration you made.

## Results

The CICS Liberty server is now connected to an angel process, and is granted access to use z/OS authorized services.

## What to do next

You might want to configure the CICS Liberty server to authenticate or authorized with the RACF registry by following the steps in [Configuring Liberty to use a SAF user registry](#).

## Managing a Liberty angel process

You might need to identify the version of the Liberty angel process or the Liberty server, view the job names and process identifiers of the angel process, or stop it when you manage the Liberty angel process.

## Interacting with the angel process started task

On the z/OS platform, you can operate the angel process by using MVS operator commands. In the following examples, [.identifier] indicates an optional identifier that can be up to eight characters.

- To display the angel process version, use the following console command:

```
MODIFY BBGZANGL[.identifier],VERSION
```

The angel version is then displayed:

```
15:48:35 STC82204 CWWKB0053I ANGEL VERSION 17
```

- To display the process identifiers (PID) of the Liberty JVM servers that are connected to the angel process:

```
MODIFY BBGZANGL[.identifier],DISPLAY,SERVERS,PID
```

A list of job names and PIDs are displayed:

```
15.48.45 STC82204 CWWKB0067I ANGEL DISPLAY OF ACTIVE SERVERS
15.48.45 STC82204 CWWKB0080I ACTIVE SERVER ASID 5c JOBNAME IYK3ZNA1 PID 83953428
15.48.45 STC82204 CWWKB0080I ACTIVE SERVER ASID 5c JOBNAME IYK3ZNA1 PID 33621002
```

Each Liberty JVM server runs under a unique PID and is returned by the CICS command INQUIRE JVMSERVER. The PID is also printed in the header of the messages .log file.

- To stop the angel process:

```
STOP BBGZANGL[.identifier]
```

**Note:** The Liberty JVM server must be stopped before restarting or applying maintenance to the angel process.

- To enable or disable tracing:

```
MODIFY BBGZANGL[.identifier],TRACE=[Y|N]
```

Select Y to enable tracing, or select N to disable tracing.

## identifying Liberty versions

The angel process has its own version, which is updated at a different pace from the Liberty version. A table mapping the angel version to the Liberty version in which it was first provided is available in [WebSphere Application Server Liberty Process types on z/OS](#).

You can identify the Liberty version of the angel process in different ways as follows:

- **Identifying the angel Liberty version from the started task system log**

If the Liberty angel process is running Liberty 18.0.0.2 or above, the started task system log contains a message that indicates the Liberty version. The following message shows an example of Liberty 22.0.0.3:

```
CWWKB0079I THE ANGEL BUILD LEVEL IS 22.0.0.3 20220302-1433 / 2022.3.0.0 20220302-1433
```

- **Identifying the version of a Liberty JVM server running in CICS from message DFHSJ1405**

The version of a Liberty running in a CICS JVM server is available in the CICS MSGUSR log:

```
DFHSJ1405I 08/30/2022 17:04:39 IYK3ZDRI JVMSERVER EYUCMCIJ is running WebSphere Application Server
Version 22.0.0.3 Liberty - (22.0.0.3-cl220320220302-1100) process ID 67174497.
```

- **Identifying the version of a Liberty server by running a script**

The `wlpenv` script is located in `WORK_DIR/applid/jvmserver`, where `WORK_DIR` is defined in the server JVM profile. Running `wlpenv` allows you to use the scripts and environment variables related to the current version of the Liberty server. The `productInfo` script can display the Liberty version.

To find out the version of Liberty server, run the following script:

```
WORK_DIR/applid/jvmserver/wlpenv productInfo version --verbose
```

The output should be similar to the following example:

```

openliberty.properties:
  com.ibm.websphere.productId=io.openliberty
  com.ibm.websphere.productOwner=Open Liberty Project
  com.ibm.websphere.productVersion=22.0.0.3
  com.ibm.websphere.productName=Open Liberty
  com.ibm.websphere.productInstallType=Archive
  com.ibm.websphere.productEdition=Open
  com.ibm.websphere.productLicenseType=EPL

WebSphereApplicationServer.properties:
  com.ibm.websphere.productId=com.ibm.websphere.appserver
  com.ibm.websphere.productReplaces=io.openliberty
  com.ibm.websphere.productOwner=IBM
  com.ibm.websphere.productVersion=22.0.0.3
  com.ibm.websphere.productName=WebSphere Application Server
  com.ibm.websphere.productInstallType=Archive
  com.ibm.websphere.productEdition=zOS
  com.ibm.websphere.productLicenseType=IPLA

WebSphereApplicationServerZOS.properties:
  com.ibm.websphere.productId=com.ibm.websphere.appserver.zos
  com.ibm.websphere.productOwner=IBM CORP
  com.ibm.websphere.productVersion=22.0.0.3
  com.ibm.websphere.productName=WAS FOR Z/OS
  com.ibm.websphere.productPID=5655-WAS
  com.ibm.websphere.productQualifier=WAS Z/OS
  com.ibm.websphere.productReplaces=com.ibm.websphere.appserver
  com.ibm.websphere.productEdition=
  com.ibm.websphere.gssp=true

```

## RACF profiles used by the Liberty angel process

Multiple RACF profiles are used to permit Liberty servers to use z/OS authorized services, and also to authenticate and authorize Java application users.

The main RACF profiles are listed in [Table 25 on page 296](#).

Table 25. RACF profile table for CICS Liberty security					
Class	Profile	Required for	Access for CICS region user ID <b>1</b>	Access for unauthenticated user ID <b>2</b>	Access for authenticated user ID <b>3</b>
SERVER	BBG.ANGEL	Angel process registration at Liberty server startup	READ		
SERVER	BBG.ANGEL.<namedAngelName>	Angel process registration at Liberty server startup	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM	Angel process registration at Liberty server startup	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM.SAFCRED	Angel process registration at Liberty server startup	READ		
SERVER	BBG.AUTHMOD.BBGZSAFM.PRODMGR	Angel process registration at Liberty server startup	READ		

Table 25. RACF profile table for CICS Liberty security (continued)

Class	Profile	Required for	Access for CICS region user ID <b>1</b>	Access for unauthenticated user ID <b>2</b>	Access for authenticated user ID <b>3</b>
SERVER	BBG.AUTHMOD .BBGZSAFM.ZO SAIO	Angel process registration at Liberty server startup	READ		
SERVER	BBG.SECPFX.B BGZDFLT <b>4</b>	Authentication or authorization	READ		
APPL	BBGZDFLT <b>4</b>	Authentication or authorization		READ	READ
EJBROLE	BBGZDFLT.<resource>.<role> <b>4 5</b>	Authorization			READ

**1** User ID that is associated with the CICS job or started task.

**2** User ID used for unauthenticated requests in Liberty. The value is controlled by using the unauthenticatedUser attribute of the <saFCredentials> element. This value defaults to WSGUEST.

**3** User ID authenticated by the Liberty server.

**4** BBGZDFLT is the default value for the security profile prefix that is set by using the profilePrefix attribute of the <saFCredentials> element, for example: <saFCredentials profilePrefix="BBGZDFLT"/>.

**5** EJBROLE profiles are required if the <saFAuthorization> element is configured. The default pattern for the profile is controlled by the SAF role mapper element, which defaults to <saFRoleMapper profilePattern="%profilePrefix%.%resource%.%role%"/>.

## Configuring authentication in CICS Liberty

You have various options to configure authentication in CICS Liberty. These authentication options include HTTP basic, form-based, client authentication, third-party, and JWT.

Before configuring the authentication method, a user registry needs to be configured. CICS Liberty supports the following user registries:

- RACF: To configure the RACF registry, follow the steps in [Configuring Liberty to use a SAF user registry](#).
- LDAP: To configure the LDAP registry, follow the steps in [Configuring Liberty to use an LDAP user registry](#).
- Basic: To configure a basic registry, see the WebSphere® Application Server for z/OS® Liberty IBM® Documentation topic [Configuring a basic user registry for Liberty](#).

Other types of user registry can be used through the Java security API.

CICS Liberty provides the following common approaches to authentication:

- HTTP basic authentication specifies that the web request can contain a user ID and password.

To configure basic authentication with CICS Liberty, see [Configuring basic authentication for Liberty web applications](#).

- Form-based authentication specifies that an HTML form is sent to the web client by the application, which the user can use to provide credentials.

To configure form-based authentication with CICS Liberty, see the WebSphere Application Server for z/OS Liberty IBM Documentation topic [Configuring a custom form login page](#).

- Programmatic authentication can be enabled programmatically by using the Enterprise Java Security API.
  - To configure basic authentication when you want to use a database identity store see [Using the Java security API with a database identity store](#).
  - To configure basic authentication when you want to use a custom identity store see [Using the Java security API with a custom identity store](#).
- Client authentication specifies that the web client must provide a certificate that is then validated and associated with a user ID.
  - To configure client authentication when you want to use a RACF keyring, see [Configuring TLS client authentication for a Liberty JVM server by using RACF](#).
  - To configure client authentication when you want to use a Java keystore, see [Configuring TLS for a Liberty JVM server by using RACF](#).
- Third-party authentication specifies that the user authenticates with a trusted third party that then sends an authentication token (for example, a JSON Web Token) to CICS Liberty.
 

To configure JWT authentication with CICS Liberty by using the OpenID Connect Client Liberty feature (openidConnectClient-1.0), see [Configuring JWT authentication](#).

For more information about design guidance on which authentication options to use, see [Designing security for CICS Liberty applications](#).

## Configuring Liberty to use a SAF user registry

A CICS Liberty server can be configured to use the SAF registry when it is connected to an angel process and authorized to use the SAFCREDS service.

### Before you begin

You must complete several other tasks:

- Complete the task [Configuring and starting the Liberty angel process](#).
- Complete the task [Connecting a Liberty JVM server to the angel process](#) and grant access to the SAFCREDS service.

You need to know the RACF user ID that is used to authenticate. This user ID must exist and have an OMVS segment.

You must have:

- Authorization to issue RACF RDEFINE and PERMIT commands.
- Write access to the `server.xml` configuration file.

### About this task

In this task, you configure the CICS Liberty server with a SAF registry and define a profile prefix, which is used as the server's APPLID and grants users access to the server.

This task assumes the following definitions:

- The profile prefix that is used by the CICS Liberty server is *safProfilePrefix*.
- The user registry realm is *registryRealm*.
- The user that is accessing the application has a RACF user ID *clientUserid*.
- The CICS region user ID is *regionUserid*.

### Procedure

1. Ensure the **cicsts:security-1.0** feature is added to the `server.xml` file.
2. Add the `<safCredentials>` element to the `server.xml`:

```
<safCredentials profilePrefix="safProfilePrefix"/>
```

This configuration sets the profile prefix to *safProfilePrefix*. You might set the profilePrefix to the APPLID of a CICS region. If you want multiple CICS regions to share identical security configuration, you can set profilePrefix to the same value for those regions. For more information, see [Accessing z/OS security resources using WZSSAD](#).

3. Add the SAF registry element to `server.xml`:

```
<safRegistry id="saf" realm="registryRealm" enableFailover="false"/>
```

This configuration sets the realm name of the registry to *registryRealm* and disables failover from authorized SAF services to unauthorized UNIX System Services. Setting the realm is optional, if this value is not set in the server configuration it is taken from the RACF REALM class.

4. Define the APPL resource and authorize users to the resource. The Liberty server uses an APPL resource profile to authenticate a user to the SAF domain. The user must have READ access to the *safProfilePrefix* resource in the APPL class. In addition, whenever the APPL class is active, the unauthenticated user ID (which is WSGUEST by default) also requires READ access to the *safProfilePrefix* resource in the APPL class. The profilePrefix is used as the Liberty server APPLID.

The *safProfilePrefix* resource name that is used by the server is specified by the profilePrefix attribute in the `<safCredentials>` configuration element. If you do not specify this element, then the default profilePrefix of BBGZDFLT is used.

The following example shows the RACF commands to define the *safProfilePrefix* resource profile and to authorize access to the *clientUserid* and WSGUEST RACF user IDs:

```
SETROPTS CLASSACT(APPL)
PERMIT safProfilePrefix CLASS(APPL) ACCESS(READ) ID(clientUserid)
PERMIT safProfilePrefix CLASS(APPL) ACCESS(READ) ID(WSGUEST)
```

5. Grant permission to the Liberty server to make authentications calls. The server must be granted permission to make authentication calls in the defined APPLID domain. This setting prevents an unauthorized user from using the server's use of authorized SAF services to discover information about which APPLIDs can and cannot be authenticated. To grant the server permission to authenticate in a particular APPLID domain, the CICS region user ID must be granted READ access to the BBG.SECPFX.*safProfilePrefix* profile in the SERVER class:

```
RDEFINE SERVER BBG.SECPFX.safProfilePrefix UACC(NONE)
PERMIT BBG.SECPFX.safProfilePrefix CLASS(SERVER) ACCESS(READ) ID(regionUserid)
```

## Results

Requests for authentication can be done with the SAF registry.

## What to do next

You might want to configure an authentication mechanism for your Java applications by choosing a configuration task in [Configuring authentication in CICS Liberty](#).

## Configuring Liberty to use an LDAP user registry

A CICS Liberty server can be configured to use an LDAP registry in order to authenticate users with their LDAP credentials. Once the user is authenticated, a Java security subject is created based on the LDAP credentials.

## Before you begin

The web application must be configured with an Enterprise Java authentication method in the `<login-config>` element in its `web.xml` file and an authorization constraint to restrict access to URL paths for

this application to specific roles. For more information, see [Configuring basic authentication for Liberty web applications](#).

You need to know the distributed identity (LDAP user ID) that is used to authenticate. This user ID must exist and have an entry in the LDAP registry.

You must have write access to the `server.xml` configuration file.

## About this task

In this task, you configure the CICS Liberty server to use LDAP as the user registry for requests to run web applications.

## Procedure

1. Remove any other user registry configuration from the `server.xml` file if present, such as a `<safRegistry/>` or `<basicRegistry/>` element.
2. Add the `ldapRegistry-3.0` and `appSecurity` features to the feature manager list in the `server.xml` file.

For example:

```
<featureManager>
  <feature>ldapRegistry-3.0</feature>
  <feature>appSecurity-3.0</feature>
</featureManager>
```

**Note:** The `appSecurity` feature version needs to be consistent with the Enterprise Java version used by the Liberty server.

3. Define the LDAP registry element and its associated properties.

For example:

```
<ldapRegistry id="ldap" realm="registryRealm" host="host.domain.com" port="389"
ldapType="IBM Tivoli Directory Server" baseDN="ou=employees,o=ibm,c=fr" ignoreCase="true">
</ldapRegistry>
```

For more details about configuring LDAP user registries with Liberty, see [Configuring LDAP user registries in Liberty](#).

## Results

Requests to the Java application are authenticated by using a distributed identity from the LDAP user registry. The Java subject is created using the LDAP authenticated identity.

The CICS task runs under the default CICS user ID if there is no matching URIMAP with a functional user ID specified.

## What to do next

You might want to:

- Configure an authorization mechanism, such as:
  - [Configuring LDAP using a URIMAP with a functional user ID and transaction security](#)
  - [Configuring LDAP with distributed identity mapping](#)
- Configure a SAF registry in addition to the LDAP registry as federated registries. See [Configuring Liberty to use federated registries](#).



## Configuring Liberty to use federated registries

6.2 Applies to 6.2.

Using the Federated Registry feature in Liberty allows multiple user registries of different types such as System Authorization Facility (SAF), Lightweight Directory Access Protocol (LDAP) and basic registries to be treated as a single registry by Liberty.

### Procedure

1. Add the federatedRegistry-1.0 Liberty feature to the server.xml file.

```
<feature>federatedRegistry-1.0</feature>
```

2. Update the server.xml file to include user registry configurations as needed.

- **LDAP**

Add the ldapRegistry-3.0 Liberty feature to the server.xml file to federate an LDAP registry.

```
<feature>ldapRegistry-3.0</feature>
```

Define the LDAP registry configuration properties in the ldapRegistry element. For example:

```
<ldapRegistry baseDN="dc=your,dc=com"
  bindDN="cn=master,dc=your,dc=com"
  bindPassword="aPassword"
  host="your.host.com"
  id="LdapRegistry"
  ignoreCase="true"
  ldapType="Custom"
  port="nnnnn"
  realm="LDAPREALM"
  recursiveSearch="true">
  <customFilters groupFilter="(&(cn=%v)(objectclass=groupOfNames))"
    groupIdMap="*:cn"
    groupMemberIdMap="ibm-allGroups:member;ibm-
allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember"
    id="customFilters"
    userFilter="(&(cn=%v)(objectclass=person))"
    userIdMap="*:cn"/>
</ldapRegistry>
```

When more than one LDAP registry is configured in the server.xml file, the registries are automatically federated and no additional configuration is needed. Even though multiple LDAP registries are federated by default, the best practice is to add the federatedRegistry-1.0 Liberty feature.

- **Basic registry**

Add the appSecurity-3.0 Liberty feature to the server.xml file to federate a basic registry.

```
<feature>appSecurity-3.0</feature>
```

Define user information in the basicRegistry element. For example:

```
<basicRegistry id="basic" realm="basicRegRealm">
  <user name="basicRegAdminUser" password="pwd1"/>
  <group name="Administrators">
    <member name="basicRegAdminUser"/>
  </group>
  <user name="basicRegUser" password="pwd2"/>
  <group name="Users">
    <member name="basicRegUser"/>
  </group>
</basicRegistry>
```

- **SAF Registry**

Add `safRegistry` and `safCredentials` elements to `server.xml` to federate a SAF registry.

```
<safRegistry id="saf" realm="SAFRealm"/>
<safCredentials mapDistributedIdentities="false"
  profilePrefix="yourprofileprefix"
  suppressAuthFailureMessages="false"/>
```

If the `mapDistributedIdentities` attribute is set to `true`, you need to map each non-SAF user to a SAF user.

Do not use `safAuthorization` when a SAF registry is federated in a CICS Liberty server. If `safAuthorization` is included, CICS Liberty attempts to create an ACEE (accessor environment element) for both non-SAF users and SAF users. Errors occur when a SAF registry is federated with a non-SAF registry unless every non-SAF user is mapped to a SAF user.

3. Update the `server.xml` file to add a `federatedRepository` element to create the Federated Registry.

For example, to federate an LDAP and the SAF registry that is defined in the step [“2” on page 301](#) examples:

```
<federatedRepository>
  <primaryRealm name="FEDREALM">
    <participatingBaseEntry name="dc=your,dc=com"/>
    <participatingBaseEntry name="o=SAFRealm"/>
  </primaryRealm>
</federatedRepository>
```

In this example, the `participatingBaseEntry` name value for an LDAP registry must be set to the baseDN value used in your LDAP connection.

Alternatively, to federate a basic registry and the SAF registry that is defined in the step [“2” on page 301](#) examples:

```
<federatedRepository>
  <primaryRealm name="FEDREALM">
    <participatingBaseEntry name="o=basicRegRealm"/>
    <participatingBaseEntry name="o=SAFRealm"/>
  </primaryRealm>
</federatedRepository>
```

In this example, the name value for the `participatingBaseEntry` is the realm name that is specified on the `basicRegistry` element.

4. Add Users to Roles.

Some applications are created with different roles that allow users access to different parts of the application. For example, the CICS default app has two roles, Administrator and User.

When a SAF registry is federated with an LDAP or Basic registry, SAF users must be included with Basic or LDAP users in an `authorization-roles` element to set the user's role.

The following example shows an `authorization-roles` element with two SAF users that belong to the User role and the Administrator role. The `authorization-roles` element applies to all applications to be run in the Liberty server.

```
<authorization-roles id="com.ibm.cics.wlp.defaultapp">
  <!--Users and groups belonging to the User role-->
  <security-role name="User">
    <group name="Users"/>
    <user name="USR275"/>
  </security-role>

  <!--Users and groups belonging to the Administrator role-->
  <security-role name="Administrator">
    <group name="Administrators"/>
    <user name="USR270"/>
  </security-role>
</authorization-roles>
```

Roles can be defined to be application-specific by defining the user as part of a security-role element within an application-bnd element. The application-bnd element can be found within application, enterpriseApplication, ejbApplication osgiApplication, and webApplication application elements, which are described in the [Liberty Server configuration documentation](#).

The following example shows an enterpriseApplication security role definition for an application that is described in the [IBM Support: Configuring Liberty for z/OS with a Federated Repository using SAF and LDAP document](#):

```
<enterpriseApplication id="LogiEAR" location="${server.config.dir}/LoginEAR.ear"
name="LoginEAR">
  <application-bnd>
    <security-role name="AuthorizedUser">
      <user name="david" access-id="david"></user>
    </security-role>
  </application-bnd>
</enterpriseApplication>
```

## Results

The Liberty JVM server is configured to use federated registries.

### Related information

[Authenticating users to run CICS Liberty web applications](#)

[Federation of user registries for Liberty on z/OS](#)

## Configuring basic authentication for Liberty web applications

Authenticate a user to an application that runs in a Liberty JVM server by using basic authentication.

### Before you begin

You must complete the configuration of a user registry for the Liberty server, see [Configuring authentication in CICS Liberty](#) for a list of related configuration tasks.

**Recommended:** It is recommended to use HTTPS with basic authentication because the credentials are not signed or encrypted. For more information, see the task [Configuring TLS for a Liberty JVM server by using RACF](#).

You must have:

- Authorization to create or update the application security constraint in the deployment descriptor (web.xml).
- Write access to the server.xml configuration file

### About this task

In this task, you configure your Liberty web application to authenticate with basic authentication.

### Procedure

1. Add the **appSecurity-2.0** feature (or a more recent version of this feature) to the server.xml file.
2. Add the security controls to the application's deployment descriptor as follows. The web.xml file can be found inside the source files for the web application that you are deploying.
  - a. Add a login configuration to the application's web.xml file to specify HTTP basic authentication as the authentication method.

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

- b. Define an authorization constraint in the web.xml to restrict access to URL paths for this application to specific roles. For example, the security constraint below restricts the access for any HTTP method to all URL paths to users with the role `cicsAllAuthenticated`.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>myResourceName</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>cicsAllAuthenticated</role-name>
  </auth-constraint>
</security-constraint>
```

The role needs to be defined to the server and be associated to the authorized users. When deploying a Java application in a CICS bundle, the `cicsAllAuthenticated` role is automatically defined and associated with the special subject `ALL_AUTHENTICATED_USERS`.

The resource name `myResourceName` is a logical name that represents the collection of web resources.

### 3. Deploy the Java application.

For example, as a CICS bundle (see [Deploying a CICS bundle in the CICS Explorer product documentation](#)).

If an application already has a security constraint defined but uses a different auth-method, you can override this at the Liberty server scope to use basic authentication by default:

```
<webAppSecurity overrideHttpAuthMethod="BASIC"/>
```

## Results

Requests to the Java application are authenticated by using a user ID and password.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction ID defined by the URIMAP that matches the request (or CJSB by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## What to do next

You might want to authorize the authenticated user to run the web request by following the steps in [Configuring CICS transaction security for a Liberty JVM server](#) or [Configuring SAF authorization with an EJBROLE](#).

## Configuring JWT authentication

Use the OpenID Connect Client feature to configure a Liberty JVM server to accept a JSON Web Tokens (JWT) as an authentication token for a specific web application. You can then run the CICS task with a RACF user ID mapped from a claim in the JWT.

## Before you begin

You must be familiar with the information in [Third-party authentication](#).

You must complete several other tasks:

- Enable the CICS Liberty security feature (**cicsts:security-1.0**).
- Complete the task [Configuring and starting the Liberty angel process](#).
- Complete the task [Connecting a Liberty JVM server to the angel process](#).
- Complete the task [Configuring RACF for identity propagation](#).



- The *iss* (issuer) claim *idg* identifies the principal that issued the JWT.
- The *sub* (subject) claim is the identity **cn=JeanLeclerc,ou=employees,o=ibm,c=fr**.

The *aud* (audience) claim **urn:myEntity** identifies the recipient that the JWT is intended for.

**Important:** The *aud* claim is optional. It can be used to identify a target application, a commercial entity, or any other entity defined by business processes.

- The *exp* (expiration time) claim identifies the expiration time on or after which the JWT must not be accepted for processing.
- The *iat* (issued at) claim identifies the time at which the JWT was issued.

The payload is Base64Url encoded to form the second part of the JWT.

To create the **signature** part the encoded header and payload are signed by using the signature algorithm from the header. The signature is used to verify that the issuer of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

**Important:** If the JWT is issued by a JWT provider that supports JWK (JSON Web Key) or is signed by using the HMAC-SHA256 algorithm, then some steps in this procedure must be modified. For more information, see [“Alternative configuration when you use a JWK or the HS256 algorithm” on page 308](#).

## Procedure

1. Configure a RACF key ring as a truststore that is used to validate the JWT signature. In this example, a private key is used to sign the JWT so the X.509 certificate that contains the public key must be stored in the Liberty JVM server truststore.

Add the X.509 certificate that contains the public key that is needed to validate the JWT signature into a RACF key ring as a trusted certificate. Connect the X.509 certificate with a usage of CERTAUTH.

Enter the following command:

```
RACDCERT ID(CICS1) CONNECT(RING(myKeyRing) LABEL('jwtValidate') CERTAUTH)
```

The command uses the following values:

- **CICS1** is the CICS region user ID.
- **myKeyRing** is the name of the key ring.
- **jwtValidate** is the label or alias of the certificate to be connected to the key ring.
  - Add a `keyStore` element to the `server.xml` configuration file for the truststore.

The `id` attribute value of the `trustStore` element must match the value that is specified on the **trustStoreRef** attribute of the `openidConnectClient` element, for example:

```
<keyStore id="JWTTrustStore" fileBased="false" location="safkeyring://CICS1/myKeyRing"
  password="myPassword" readOnly="true"
  type="JCERACFKS" />
```

If you are running Java 11, the location must be `location="safkeyringjce://CICS1/myKeyring"`. For more information about keystore, see [Keystores](#) in the IBM WebSphere Application Server for z/OS Liberty documentation.

2. Add the **openidConnectClient-1.0** Liberty feature to the `server.xml` file.
3. Configure the **<openidConnectClient>** element in the `server.xml` to use JWT authentication, for example:

```
<openidConnectClient id="RS"
  clientId="RS-JWT-CICS" inboundPropagation="required"
  signatureAlgorithm="RS256" trustStoreRef="JWTTrustStore"
  trustAliasName="JWTTrustCert" userIdentifier="sub"
  mapIdentityToRegistryUser="false" issuerIdentifier="idg"
  audiences="urn:myEntity"/>
```

In this example:

- *id* and *clientId* are element identifiers.
- *inboundPropagation* is set to **required** to allow Liberty JVM server to use the received JWT as an authentication token.
- *signatureAlgorithm* specifies the algorithm to be used to verify the JWT signature.

CICS Liberty supports the RS256 (asymmetric algorithm) and HS256 (symmetric algorithm).

- *trustStoreRef* specifies the name (in the *id* attribute) of the keystore element that defines the location of the validating certificate.
- *trustAliasName* gives the alias or label of the certificate to be used for signature validation.
- *userIdentifier* indicates the claim to use to create the user subject. The default claim to use for the user subject is the **sub** claim.
- *mapIdentityToRegistryUser* indicates whether to map the retrieved identity to the registry user. You set *mapIdentityToRegistryUser* as **false** because the identity in the *sub* claim is not a RACF user ID.
- *issuerIdentifier* defines the expected issuer.
- *audiences* defines a comma-separated list of target audiences (the audience in the JWT must match one of the defined audiences)

For more information about other `openidConnectClient` attributes, see the WebSphere Application Server for z/OS Liberty IBM Documentation topic [OpenID Connect Client \(openidConnectClient\)](#).

**Important:** If JWT authentication is needed only for a subset of requests to the Liberty JVM server, the `openidConnectClient` element can include an `authFilter` attribute that represents conditions that are matched against the HTTP request.

For more information about authentication filters, see [Authentication Filter \(authFilter\)](#).

4. Set the attribute `mapDistributedIdentities="true"` on the `safCredentials` element in the `server.xml` configuration file.

For example,

```
<safCredentials mapDistributedIdentities="true"/>
```

5. Activate the RACF IDIDMAP class. Enter the following RACF command:

```
SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
```

6. Define a distributed identity filter in RACF to map the distributed user ID from the JWT subject claim to a RACF user ID.

For example, enter the following command:

```
RACMAP ID(EMPLOY1) MAP USERIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))  
REGISTRY(NAME('*')) WITHLABEL('Mapping Jean Leclerc')
```

In this example, the following values are used:

- `cn=JeanLeclerc,ou=employees,o=ibm,c=fr` is the distributed user ID from the JWT subject claim.
- `EMPLOY1` is the RACF user ID to which the distributed user ID is to be mapped.
- `REGISTRY(NAME('*'))` matches any registry realm name. Alternatively, to match a specific realm, replace the `*` with the value that is specified on the `realmName` attribute of the `openidConnectClient` element.

**Important:** This example shows a one-to-one mapping. z/OS Identity Propagation also supports many-to-one mappings. For example, you might use a many-to-one mapping to map all employees to the same RACF user ID.



For more information about the RACMAP command, see [RACMAP \(Create, delete, list, or query a distributed identity filter\)](#) in the z/OS Security Server RACF Command Language Reference.

#### 7. Refresh the RACF IDIDMAP class.

For the changes to take effect, enter the following RACF command:

```
SETOPTS RACLIST(IDIDMAP) REFRESH
```

For more information about configuring the OpenID Connect Client feature with Liberty z/OS, see [Configuring JSON Web Token authentication for OpenID Connect](#).

## Results

A JWT is used for authenticating a request to a Liberty JVM server. The distributed identity in the JWT is mapped to a RACF user ID, and the RACF user ID is then used for running the CICS task.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to the transaction ID defined by the URIMAP that matches the request (or CJSA by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Alternative configuration when you use a JWK or the HS256 algorithm

If the JWT is issued by a JWT provider that supports JWK (JSON Web Key) or is signed by using the HMAC-SHA256 secret key algorithm, then a truststore is not necessary. However, you must specify different attributes on the **openidConnectClient** element.

## Alternate process

1. If the public key is retrieved from a JWK endpoint, you specify the JWK endpoint URL on the **jwkEndpointUrl** attribute.
2. If the JWT is signed by using a shared secret key with the HMAC-SHA256 algorithm, you define the shared secret key on the **sharedKey** or **clientSecret** attributes.

For information about these alternative **openidConnectClient** attributes, see the WebSphere Application Server for z/OS Liberty IBM Documentation topic [OpenID Connect Client \(openidConnectClient\)](#).

## Configuring security by using a database identity store

The JSR-375 specification provides the option to use a database identity store. You can use this option to implement your own identity store and control exactly where to obtain user information.

For more information about JSR-375, see [“Enterprise Java Security API” on page 242](#).

## About this task

Follow these steps to authenticate a Liberty web application. This example uses HTTP authentication together with a database identity store in a Db2 database.

## Procedure

1. Add either the appSecurity-3.0 feature or the appSecurity-4.0 to `server.xml` according to whether you are using Java EE 8 or Jakarta EE 9.

**6.2** Add the appSecurity-5.0 feature to `server.xml` if you are using Jakarta EE 10.

2. Ensure that CDI annotation file scanning is enabled. CICS disables it by default in `server.xml`. You can ensure that CDI annotation file scanning is enabled for the web application by:



- Adding the relevant version of the CDI feature to the Liberty feature manager list.
  - Adding a `beans.xml` file to the `WEB-INF` folder of the `.war` file.
3. Create a Java class to implement the database identity store, and deploy this store within your application, as shown in the following example:

```
@DatabaseIdentityStoreDefinition
(
    dataSourceLookup = "jdbc/sec",
    callerQuery = "#{select password from USR where USERNAME = ?'}",
    groupsQuery = "#{select ugroup from USR where USERNAME = ?'}",
    hashAlgorithm = Pbkdf2PasswordHash.class,
    priorityExpression = "#{100}",
    hashAlgorithmParameters =
    {
        "Pbkdf2PasswordHash.Iterations=3072",
        "Pbkdf2PasswordHash.Algorithm=PBKDF2WithHmacSHA512",
        "Pbkdf2PasswordHash.SaltSizeBytes=64"
    }
)
```

4. Create a table in the database. For example, to create a Db2 table by using SQL:

```
CREATE TABLE PXX.USR (
    USERNAME      VARCHAR ( 256 ) NOT NULL,
    PASSWORD      VARCHAR ( 256 ) NOT NULL,
    UGROUP        VARCHAR ( 256 ) NOT NULL
) IN SECU.TSSE;
CREATE UNIQUE INDEX INDXUSRS ON PXX.USR (USERNAME);
```

The password in the database must be encrypted. An example of inserting an encrypted password into a database can be found in [Database Setup](#).

5. Configure the Liberty `server.xml`:

- a. Add the `jdbc-4.2` feature in `server.xml`:

```
<feature>jdbc-4.2</feature>
```

- b. Define a data source in `server.xml` matching the JNDI name referenced on the identity store class. For example:

```
<dataSource id="DefaultDataSource" jndiName="jdbc/sec">
    <jdbcDriver libraryRef=<xxx>/>
    ...
</dataSource>
```

6. Create a servlet that defines access to a role returned by the identity store implementation. For example:

```
@WebServlet("/home")
@ServletSecurity(@HttpConstraint(rolesAllowed = "ROLE1"))
public class Servlet extends HttpServlet {...}
```

Add either the HTTP basic authentication or login-form annotation to the servlet class, for instance to use basic authentication add the following text:

```
@BasicAuthenticationMechanismDefinition(realmName="defaultRealm")
```

7. Determine whether to map the identity store credential to a SAF user ID.

If you want CICS tasks to run under specific SAF users mapped from your database identity store, you need to take the following steps:

- Add the `<safRegistry/>` element to your `server.xml` to enable SAF support.
- Add the `cicsts:security-1.0` feature to your `server.xml` to ensure that the Liberty subject is asserted as the CICS task userid.

- c. Configure the mapping of distributed identities in `server.xml` by setting the following element.

```
<safCredentials mapDistributedIdentities="true" />
```

- d. Create a mapping of the identity from the database store to a SAF user ID. This mapping is achieved by using the RACMAP command.

In the following example *databaseUserId* is the identity from the database identity store and *safUserId* is the RACF user ID that is used to run in Liberty. When you use a database identity store the registry realm name by default is *defaultRealm*.

```
RACMAP ID(safUserId) MAP WITHLABEL('label-name')
USERIDFILTER(NAME('databaseUserId'))
REGISTRY(NAME('defaultRealm'))
```

### Important:

If you deploy the application in a CICS bundle, the security role "cicsAllAuthenticated" is automatically set in the `installedApps.xml` as follows:

```
<application ...>
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</application>
```

This `cicsAllAuthenticated` role takes precedence over the group name that is stored in the database identity store and an HTTP 403 error occurs. You can select from three options to resolve this error:

1. Deploy within a CICS bundle and name the `cicsAllAuthenticated` role on the `@ServletSecurity rolesAllowed` annotation.
2. Deploy within a CICS bundle but use SAF authorization and the SAF role mapper. Your use of these options bypasses the role mappings that are specified in the `<application-bnd>`.
3. Deploy your application directly into Liberty by using an `<application>` element in `server.xml`, thus avoiding use of a CICS bundle.

## Results

You successfully configured the database identity store.

## Configuring security by using a custom identity store

The JSR-375 specification provides the option to use a custom identity store. You can use this option to implement your own identity store and control exactly how to perform user authentication.

For more information about JSR-375, see [“Enterprise Java Security API” on page 242](#).

## About this task

Follow these steps to authenticate a Liberty web application by using a custom identity store, an HTTP authentication mechanism, and mapping the authenticated identity to a SAF user ID to be used for the CICS task.

## Procedure

1. Add either the `appSecurity-3.0` feature or `appSecurity-4.0` to `server.xml` depending on whether you are using Java EE 8, or Jakarta EE 9.

**6.2** Add the `appSecurity-5.0` feature to `server.xml` if you are using Jakarta EE 10.

2. Ensure that CDI annotation file scanning is enabled. CICS® disables it by default in `server.xml`. You can ensure that CDI annotation file scanning is enabled for the web application by:
  - Adding the relevant version of the CDI feature to the Liberty feature manager list.
  - Adding a `beans.xml` file to the WEB-INF folder of the .war file.
3. Create Java™ classes to process the custom identity store logic and build them into a WAR file.
  - a. Create a custom identity store object by creating a class that implements the `IdentityStore` interface, as shown in the following example:

```
@ApplicationScoped
public class MyIdentityStore implements IdentityStore
{
    public CredentialValidationResult validate(UsernamePasswordCredential userCredential)
    {
        if (userCredential.compareTo("USERID", "PASSWORD"))
        {
            return new CredentialValidationResult("USERID", new
HashSet<>(Arrays.asList("ROLE1", "ROLE2")));
        }
        return CredentialValidationResult.INVALID_RESULT;
    }
}
```

- b. Create an HTTP authentication mechanism associated with this identity store, which is used with the identity store class that is created in the previous step:

```
@ApplicationScoped
public class MyAuthMechanism implements HttpAuthenticationMechanism
{
    @Inject
    private IdentityStoreHandler idStoreHandler;

    public AuthenticationStatus validateRequest(HttpServletRequest req,
HttpServletResponse res, HttpContext context)
    {
        CredentialValidationResult result = idStoreHandler.validate(
            new UsernamePasswordCredential(
                req.getParameter("name"),
                req.getParameter("password")));
        if (result.getStatus() == CredentialValidationResult.Status.VALID)
        {
            return context.notifyContainerAboutLogin(result);
        }
        else
        {
            return context.responseUnauthorized();
        }
    }
}
```

4. Create a servlet that defines access to a role returned by the identity store implementation. For example:

```
@WebServlet("/home")
@ServletSecurity(@HttpConstraint(rolesAllowed = "ROLE1"))
public class Servlet extends HttpServlet {...}
```

Optionally, if you are not developing your own HTTP authentication mechanism as in step 3b, then add either the HTTP basic authentication or login-form annotation to the servlet class, for instance:

```
@BasicAuthenticationMechanismDefinition(realName="defaultRealm")
```

5. Determine whether to map the identity store credential to a SAF user ID.

If you want CICS tasks to run under specific SAF users mapped from your custom identity store, you need to take the following steps:

- a. Add the `<safRegistry/>` element to your `server.xml` to enable SAF support.

- b. Add `cicsts:security-1.0` feature to your `server.xml` to ensure that the Liberty subject is asserted as the CICS task user ID.
- c. Configure the mapping of distributed identities in `server.xml` by setting the following element.

```
<safCredentials mapDistributedIdentities="true" />
```

- d. Create a mapping of the identity from the custom registry to a SAF user ID by using the RACMAP command.

In the following example *customUserid* is the identity from the custom registry and *safUserid* is the RACF user ID that is used to run in Liberty. When you use a custom identity store, the registry realm name is *defaultRealm* unless it is overridden.

```
RACMAP ID(safUserid) MAP WITHLABEL('label-name')
USERIDFILTER(NAME('customUserid'))
REGISTRY(NAME('defaultRealm'))
```

### Important:

If you deploy the application in a CICS bundle, the security role "cicsAllAuthenticated" is automatically set in the `installedApps.xml` as follows:

```
<application ...>
  <application-bnd>
    <security-role name="cicsAllAuthenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</application>
```

This `cicsAllAuthenticated` role takes effect and an HTTP 403 error occurs. You can select from three options to resolve this error:

1. Deploy within a CICS bundle and name the `cicsAllAuthenticated` role on the `@ServletSecurity rolesAllowed` annotation.
2. Deploy within a CICS bundle but use SAF authorization and the SAF role mapper. Your use of these options bypasses the role mappings that are specified in the `<application-bnd>`.
3. Deploy your application directly into Liberty by using an `<application>` element in `server.xml`, thus avoiding use of a CICS bundle.

## Results

You successfully configured a custom identity store.

## Configuring authorization in CICS Liberty

You can use Enterprise Java application security roles to authorize access to Enterprise Java applications. Additionally, in a Liberty JVM server you can further restrict access to transactions (run as part of the application) by using CICS transaction and resource security.

When you use role authorization, you can use SAF role authorization with EJB roles or Liberty role authorization with an `<application-bnd>` element that is defined in the `<application>` element of your `server.xml`.

For more information about design guidance on which options to use, see [Designing security for CICS Liberty applications](#).

The following tasks outline the configuration steps for different authorization tasks.

## Configuring CICS transaction security for Liberty web applications

Authorize access to a Java application that you run in a Liberty JVM server by using CICS transaction security.

### Before you begin

Before you begin this task, you must complete these tasks:

- Ensure the CICS Liberty security feature (cicsts:security-1.0) is enabled.
- Complete the task [Configuring and starting the Liberty angel process](#).
- Complete the task [Connecting a Liberty JVM server to the angel process](#).
- Configure authentication for the Java application, see [Configuring authentication in CICS Liberty](#).

If a RACF user ID is not used to authenticate, then you need to complete the task [Configuring RACF for identity propagation](#).

You need to know:

- The application URI that needs to be protected.
- The CICS transaction ID that is defined for running the request.
- The user ID that you want authorized to run the application.

You must have:

- Authorization to issue the RACF RDEFINE and PERMIT commands.
- Authorization to create or update the application security constraint in the deployment descriptor (`web.xml`).

### About this task

In this task, you authorize access to a Liberty application by using CICS transaction security.

Role authorization is bypassed by allowing access to the web application by using any authenticated user ID.

This task assumes:

- The Java web application is deployed with an authorization constraint (by using the `<auth_constraint>` element in `web.xml`) that requires a user to be a member of an authorized role.
- The initial path of the URIs used to access the application is `/mycompany/myapp`.
- The user that accesses the application has a RACF user ID *taskUserid*.
- The CICS transaction ID used to run the request is *tranA*.
- The CICS URIMAP used to map the request is *urimapA*.
- The CEDA group name that is used for the CICS transaction ID is *groupA*.

### Procedure

1. Define a CSD definition for the CICS transaction ID to run the application as follows.

For more information about defining CICS resources, see [How you can define CICS resources](#).

```
DEFINE TRANSACTION(tranA) GROUP(groupA) PROGRAM(DFHSJTHP)
```

You will receive a warning when you define this transaction because the value of the PROGRAM attribute starts with the letters DFH. This warning can be safely ignored.

The program DFHSJTHP acts as a dummy program in this scenario. It does not run any code. Instead, it allows your Java code to run under the specified transaction.

2. Define a URIMAP of type JVMSERVER for the web application:

```
DEFINE URIMAP(urimapA) GROUP(groupA) PATH(/mycompany/myapp/*) SCHEME(HTTP)
USAGE(JVMSEVER) HOST(*) PORT(*) TRANSACTION(tranA)
```

#### ***/mycompany/myapp/\****

Is the path component of the URI to which the URIMAP definition applies.

**Important:** URIMAPs that specify SCHEME (HTTP) apply to HTTP and HTTPS requests. If you specify SCHEME (HTTPS), the URIMAP apply to HTTPS requests only.

3. Install the previously defined *groupA* resources in the CICS region.
4. Create a member list containing the *tranA* transaction in RACF and give user *taskUserid* read access:

```
RDEFINE GCICSTRN memberListA ADDMEM(tranA) UACC(NONE)
PERMIT memberListA CLASS(GCICSTRN) ID(taskUserid) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Instead of you granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to that group.

If your CICS region uses a security prefix, all GCICSTRN profile names need to be prefixed with this value.

5. Optional: If you are using SAF role authorization, and you do not want to define an EJBROLE to authorize access to the application, you can change the role name in the authorization constraint (<auth\_constraint> element in web.xml) to the special role \*\*. Making this change allows all authenticated users to be authorized to run the application. For more information, see [Configuring SAF authorization with an EJBROLE](#).

This allows CICS transaction security to be used exclusively for application authorization. Authorization to the application is controlled entirely by using URIMAPs and CICS transaction security.

6. Optional: If you are using Liberty authorization, you can avoid having to configure security role authorization by using the special subject ALL\_AUTHENTICATED\_USERS.

Using this special subject and giving the `cicsAllAuthenticated` role access to all URLs in your web applications deployment descriptor (web.xml) allows access to the web application by using any authenticated user ID. For more information, see [Authorizing users to run CICS Liberty web applications](#).

This allows CICS transaction security to be used exclusively for application authorization. Authorization to the application is controlled entirely by using URIMAPs and CICS transaction security.

If you deploy a Liberty application in a CICS bundle, CICS automatically configures the `cicsAllAuthenticated` role for you in the `installedApps.xml` file.

7. Deploy the Java application, for example, as a CICS bundle.

## Results

The user with RACF user ID *taskUserid* is authorized to access the application.

This example configuration shows how CICS transaction security can be used to secure access to an entire application. It is also possible to use more fine-grained URIMAP resource definitions that map to different transaction IDs to enable different security levels for different URIs in the same application.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **JVM Server** tab and set the **Transaction ID** field to *tranA* that is defined in step 1 (or CJSB by default). For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

## Configuring SAF role authorization with an EJBROLE

Authorize access to an application that runs in a Liberty JVM server by using a SAF role, in this case, an EJBROLE.

### Before you begin

Before you begin this task, you must complete several other tasks:

- Ensure the CICS Liberty security feature (`cicsts:security-1.0`) is enabled.
- Complete the task [Configuring and starting the Liberty angel process](#).
- Complete the task [Connecting a Liberty JVM server to the angel process](#).
- Configure authentication for the Java application, see [Configuring Authentication](#).

If a RACF user ID is not used to authenticate, then you need to complete the task [Configuring RACF for identity propagation](#).

You need to know the application URI that needs to be protected and the user ID that must be authorized to run the application.

You must have:

- Authorization to issue the RACF RDEFINE and PERMIT commands.
- Write access to the `server.xml` configuration file.
- Authorization to create or update the application security constraint in the deployment descriptor (`web.xml`).

### About this task

In this task, you configure SAF role authorization to check that an authenticated user ID is authorized to run a web application. An [alternative authorisation option](#) is described at the end of the main process.

When you use SAF role authorization, the RACF implementation of SAF uses an EJBROLE profile to hold the mapping of users and groups to roles. A SAF role mapper is used to identify the EJBROLE profile associated with a security role. The server queries RACF to determine whether the user or the user's group has the necessary READ access to the EJBROLE profile.

When you use roles and SAF role authorization, you can continue to use CICS bundles to lifecycle your web applications. The `<application-bnd>` is ignored by Liberty and the SAF role authorization takes precedence.

**Important:** When SAF role authorization is enabled, role mappings are not defined in `server.xml`. Therefore, mapping users to special subjects such as `ALL_AUTHENTICATED_USERS` and `EVERYONE`, cannot be done in `server.xml`. For more information about an alternative approach, see [“Alternative configuration by using special role \\*\\*”](#) on page 317.

This task assumes:

- The URI used to access the application is `/mycompany/myapp`.
- The user who accesses the application has a RACF user ID `taskUserid`.
- The role name used to protect access to the application is `roleA`.
- The default SAF role mapping `%profile_prefix%.%resource%.%role%` is used to determine whether a user is in a role.
- The profile prefix `safProfilePrefix` is used.
- The application name is `MYAPP` and is defined in the server configuration as follows:

```
<application id="MYAPP" location="${server.output.dir}/installedApps/MYAPP.war" name="MYAPP" type="war"/>
```

## Procedure

1. Add an authorization constraint, the `<auth_constraint>` element, to the web applications deployment descriptor (`web.xml`), for example:

```
<security-constraint>
  <display-name>
    myapp.role_restraint
  </display-name>
  <web-resource-collection>
    <web-resource-name>
      myappRestraint
    </web-resource-name>
    <description>
      Protection for urls in myapp
    </description>
    <url-pattern>/mycompany/myapp/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description>Web users only</description>
    <role-name>roleA</role-name>
  </auth-constraint>
</security-constraint>
```

By using the configuration that is shown in the earlier example, the application restricts access to the URL `mycompany/myapp/*` so that only users in the role `roleA` have access.

An alternative to declaring the security constraints in the application deployment descriptor is to use the `ServletSecurity` annotations.

2. Add the `<safAuthorization id="saf"/>` element to your `server.xml`.

If you are using the `cicsts:distributedIdentity-1.0` feature, this addition is made for you.

**Recommended:** You can also add `racRouteLog="ASIS"` to the Liberty `<safAuthorization>` element in the previous step to record the access attempts in the manner that is specified in the RACF EJBROLE profile.

3. Add the `<safCredentials>` element to the `server.xml`:

```
<safCredentials profilePrefix="safProfilePrefix"/>
```

This configuration sets the profile prefix to `safProfilePrefix`.

For example, you might set it to the APPLID of a region. If you want multiple regions to share identical security configuration, you can set `profilePrefix` to the same value for those regions. For more information, see [Accessing z/OS security resources using WZSSAD](#).

If you want authorization failures in the MVS™ job log, you must add `suppressAuthFailureMessages="false"` to the Liberty `<safCredentials>` element.

4. Optionally, configure the SAF role pattern to be used by the Liberty server by using the `<safRoleMapper>` element in the `server.xml`:

```
<safRoleMapper profilePattern="%profilePrefix%.%resource%.%role-mapping%"
  toUpperCase="true"/>
```

### **<safRoleMapper>**

Determines the name of the RACF EJBROLE profiles that are used to validate the user authorization.

### **toUpperCase**

Controls whether the EJBROLE profile name is case-sensitive. `toUpperCase="true"` specifies it to be not case-sensitive.

The default value is `false`.

### **profilePattern**

The default value is `"%profilePrefix%.%resource%.%role%"`. It uses three substitution variables:



- `%profilePrefix%` takes the value of the server `profilePrefix` configured in `<safCredentials>`.
- `%resource%` takes the value of the application name as defined in the `<application>` element, or the name of the archive if no `<application>` element exists.

The application name is also printed by the Liberty server in a CWWKZ0001I message when the application starts.

- `%role%` takes the value of a role that is defined either in the web application deployment descriptor (`web.xml`) or in the `ServletSecurity` annotation.

The `profilePattern` can be customized to allow a less granular authorization. For example, if all the roles are common across the applications and each application role is associated to the same users, you might omit the `%resource%` variable and thus define fewer EJBROLE profiles.

5. Create the EJBROLE profiles in RACF, regarding the prefix scheme described earlier:

```
RDEFINE EJBROLE safProfilePrefix.MYAPP.roleA UACC(NONE)
SETROPTS RACLIST(EJBROLE) REFRESH
```

6. Authorize the user `taskUserId` to call the application:

```
PERMIT safProfilePrefix.MYAPP.roleA CLASS(EJBROLE) ACCESS(READ) ID(taskUserId)
SETROPTS RACLIST(EJBROLE) REFRESH
```

Instead of granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to that group.

7. Deploy the Java application, for example as a CICS bundle.

## Results

The user with RACF user ID `taskUserId` is authorized to access the application.

The example configuration that is outlined in this task shows how an EJBROLE profile can be used to secure access to an entire application. It is also possible to use EJBROLE profiles and groups to set up different security levels for different URIs in the same application.

## Alternative configuration by using special role \*\*

If you don't want to protect access to a specific Java application by using an EJBROLE profile, you can use the special role `**`. This role allows all authenticated users to be authorized to run your application. For example,

```
<auth-constraint>
  <description>Web users only</description>
  <role-name>**</role-name>
</auth-constraint>
```

- The special role name `**` is a shorthand for any authenticated user independent of role.
- The special role name `*` is a shorthand for all role names that are defined in the deployment descriptor. When using Java annotations, the equivalent of the special role `*` is `@PermitAll`.

When the special role name `**` appears in an authorization constraint, it indicates that any authenticated user, independent of role, is authorized to perform the constrained requests. Special roles do not need an extra `<security-role>` declaration in `web.xml`.

**Important:** When SAF role authorization is enabled, an application binding is ignored. For more information, see [Authorizing users to run CICS Liberty web applications](#).

## Configuring LDAP using a URIMAP with a functional user ID and transaction security

A CICS Liberty server can be configured to use an LDAP registry to authenticate users, and a functional user ID to authorize CICS transactions. This provides the ability to authenticate users using LDAP and authorize access to applications using RACF and CICS transaction security based on static mapping defined in a URIMAP resource.

### Before you begin

You must complete the task [Configuring Liberty to use an LDAP user registry](#).

You need to know:

- The distributed identity (LDAP user ID) that is used to authenticate.
- The RACF user ID to be used for authorization.

You must have:

- Authorization to issue RACF **RDEFINE** and **PERMIT** commands.
- Write access to the `server.xml` configuration file.

### About this task

In this task, you setup a URIMAP to define a functional user ID. The functional user ID is used for transaction security for requests to web applications in a Liberty JVM server.

### Procedure

1. Define a CSD definition for the CICS transaction ID to run the application as follows.

For more information about defining CICS resources, see [How you can define CICS resources](#).

```
DEFINE TRANSACTION(tranA) GROUP(groupA) PROGRAM(DFHSJTHP)
```

You will receive a warning when you define this transaction because the value of the PROGRAM attribute starts with the letters DFH. This warning can be safely ignored.

The program DFHSJTHP acts as a dummy program in this scenario. It does not run any code. Instead, it allows your Java code to run under the specified transaction.

2. Define a URIMAP in CICS to map the web application's URI to a functional user ID.

- a) Define a URIMAP of type JVMSERVER for the web application, naming your functional RACF user ID for authorization.

```
DEFINE URIMAP(urimapA) GROUP(groupA) PATH(/mycompany/myapp/*) SCHEME(HTTP)
USAGE(JVMSERVER) HOST(*) PORT(*) USERID(functionalUseridA)
TRANSACTION(tranA)
```

#### ***/mycompany/myapp/\****

Is the path component of the URI to which the URIMAP definition applies.

**Important:** URIMAPs that specify SCHEME(HTTP) apply to HTTP and HTTPS requests. If you specify SCHEME(HTTPS), the URIMAP apply to HTTPS requests only.

- b) Install the previously defined *groupA* resources in the CICS region.
- c) Create a RACF profile for the *tranA* transaction and give user *functionalUseridA* read access:

```
RDEFINE TCICSTRN tranA UACC(NONE)
PERMIT tranA CLASS(TCICSTRN) ID(functionalUseridA) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Instead of granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to that group.

## What to do next

You might want to:

- Configure a SAF registry and federate the LDAP and SAF registries. See [Configuring Liberty to use federated registries](#).
- Configure an alternative authorization method, such as [Configuring LDAP with distributed identity mapping](#).

### Related information

[Design example: Securing a web application with a preset URIMAP user ID](#)

## Configuring LDAP with distributed identity mapping

A CICS Liberty server that uses an LDAP security registry can be further configured to map LDAP credentials to a RACF user ID with by using RACF distributed identity filters. It allows you to authenticate users using LDAP and authorize access using RACF based on a set of mapping rules.

### Before you begin

You must complete the following tasks:

- [Configuring Liberty to use an LDAP user registry](#)
- [Configuring and starting the Liberty angel process](#)
- Setting up a security constraint for the web application. For an example, see [Configuring basic authentication for Liberty web applications](#).

You need to know:

- The distributed identity (LDAP user ID) that is used to authenticate.
- The RACF user ID that the distributed identity will be mapped to. This user ID must exist in the RACF registry and have an OMVS segment.

You must have:

- Authorization to issue RACF **RACMAP**, **RDEFINE**, **RACMAP**, **SETROPTS** and **PERMIT** commands.
- Write access to the `server.xml` configuration file.

### About this task

The authorization to the web application can be controlled by using the CICS transaction security to grant all authenticated users access, or by using SAF role authorization and EJBROLE profiles to ensure role-based access. The SAF role authorization can be used in addition or as an alternative to the CICS transaction security.

Whichever authorization method you decide to use, you follow a common procedure first and then different routes depending on your choice.

### Common procedure

1. Configure the Liberty server to use distributed identity mapping.
  - a. Add the `cicsts:security-1.0` feature to the feature manager list in the `server.xml` file.
  - b. Enable mapping of distributed identities by using the `<safCredentials>` element in `server.xml`.

An example is as follows:

```
<featureManager>
  <feature>ldapRegistry-3.0</feature>
  <feature>cicstsSecurity-1.0</feature>
```

```
</featureManager>
<safCredentials profilePrefix="safProfilePrefix" mapDistributedIdentities="true"/>
```

2. Configure the Liberty server to use SAF services. These services are used to map an identity to a SAF user ID and perform authorization check against SAF profiles.

- a. **Optional** Configure the Liberty server to use the SAFCREDS service by setting the following system property in the JVM profile. By default without the system property, a Liberty server requests access to all the z/OS authorized services. If the server is not authorized to all z/OS authorized services, a security violation message may be issued. Setting the system property restricts the request and usage of z/OS authorized service to the ones listed in the property. For example:

```
-Dcom.ibm.ws.zos.core.angelRequiredServices=PRODMGR,ZOSAI0,SAFCRED
```

- b. Define the APPL resource and authorize users to the resource. The Liberty server uses an APPL resource profile to authenticate a user to the SAF domain. The user must have READ access to the *safProfilePrefix* resource in the APPL class.

In addition, whenever the APPL class is active, the unauthenticated user ID (WSGUEST by default) also requires READ access to the *safProfilePrefix* resource in the APPL class. The *profilePrefix* is used as the Liberty server APPLID.

#### ***safProfilePrefix***

You can specify the *safProfilePrefix* resource name to be used by the server on the *profilePrefix* attribute in the `<safCredentials>` configuration element. If you do not specify this element, the default value BBGZDFLT is used.

The following example shows the RACF commands to define the *safProfilePrefix* resource profile and to authorize access to the *mappedUserid* and WSGUEST RACF user IDs:

```
SETROPTS CLASSACT(APPL)
PERMIT safProfilePrefix CLASS(APPL) ACCESS(READ) ID(mappedUserid)
PERMIT safProfilePrefix CLASS(APPL) ACCESS(READ) ID(WSGUEST)
```

- c. Grant permission to the Liberty server to make authentication calls.

The server must be granted permission to make authentication calls in the defined APPLID domain. This setting prevents an unauthorized user from using the server's authorized SAF services to discover information about which APPLIDs can and cannot be authenticated.

To grant the server permission to authenticate in a particular APPLID domain, the CICS region user ID must be granted READ access to the BBG.SECPFX.safProfilePrefix profile in the SERVER class:

```
RDEFINE SERVER BBG.SECPFX.safProfilePrefix UACC(NONE)
PERMIT BBG.SECPFX.safProfilePrefix CLASS(SERVER) ACCESS(READ)
ID(regionUserid)
```

3. Define a distributed identity filter in RACF.

- a. Activate the RACF IDIDMAP class.

Enter the following RACF command:

```
SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
```

- b. Map the distributed LDAP user ID to a RACF user ID, for example, by entering the following command:

```
RACMAP ID(mappedUserid) MAP
USERIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))
REGISTRY(NAME('registryRealm')) WITHLABEL('Mapping Jean Leclerc')
```

**cn=JeanLeclerc,ou=employees,o=ibm,c=fr**

Is the distinguished name of the distributed identity in LDAP.

***mappedUserid***

Is the RACF user ID to which the distributed identity is to be mapped.

### **registryRealm**

Must match the LDAP registry realm name. Alternatively, to match any realm, specify the value `*`.

- c. Refresh the RACF IDIDMAP class for the changes to take effect.

Enter the following RACF command:

```
SETROPTS RACLIST(IDIDMAP) REFRESH
```

**Note:** This example shows one-to-one mapping. z/OS Identity Propagation also supports many-to-one mapping. For example, you might use many-to-one mapping to map a subset of employees to the same RACF user ID.

4. To configure CICS transaction security to grant all authenticated users access, continue with [“Authorization using CICS transaction security” on page 321.](#)

To configure role-based access in addition or as an alternative to CICS transaction security, continue with [“Authorization using SAF roles” on page 322.](#)

## **Authorization using CICS transaction security**

You can enforce authorization with CICS transaction security. In this situation, the role-based authorization to access the web application can be granted to all authenticated users (the `cicsAllAuthenticated` role). This role-based authorization enforced by Enterprise Java is necessary to insure all requests are run with an authenticated user. Then the CICS region verifies if the authenticated user has the authorization to run the transaction associated with the request.

1. Define a CSD definition for the CICS transaction ID to run the application as follows.

For more information about defining CICS resources, see [How you can define CICS resources.](#)

```
DEFINE TRANSACTION(tranA) GROUP(groupA) PROGRAM(DFHSJTHP)
```

You will receive a warning when you define this transaction because the value of the PROGRAM attribute starts with the letters DFH. This warning can be safely ignored.

The program DFHSJTHP acts as a dummy program in this scenario. It does not run any code. Instead, it allows your Java code to run under the specified transaction.

2. Define a URIMAP of type JVMSERVER for the web application:

```
DEFINE URIMAP(urimapA) GROUP(groupA) PATH(/mycompany/myapp/*) SCHEME(HTTP)
USAGE(JVMSERVER) HOST(*) PORT(*) TRANSACTION(tranA)
```

### **/mycompany/myapp/\***

Is the path component of the URI to which the URIMAP definition applies.

**Important:** URIMAPs that specify SCHEME (HTTP) apply to HTTP and HTTPS requests. If you specify SCHEME (HTTPS), the URIMAP apply to HTTPS requests only.

3. Install the previously defined *groupA* resources in the CICS region.
4. Create a RACF profile for the *tranA* transaction and give user *mappedUserid* READ access:

```
RDEFINE TCICSTRN tranA UACC(NONE)
PERMIT tranA CLASS(TCICSTRN) ID(mappedUserid) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Instead of granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to that group.

5. Configure security role authorization for the web application by using the special subject `ALL_AUTHENTICATED_USERS`.

Using this special subject and giving the `cicsAllAuthenticated` role access to all URLs in your web applications deployment descriptor (`web.xml`) allows access to the web application by using

any authenticated user ID. For more information, see [Authorizing users to run CICS Liberty web applications](#).

This allows CICS transaction security to be used exclusively for application authorization. Authorization to the application is controlled entirely by using URIMAPs and CICS transaction security.

If you deploy a Liberty application in a CICS bundle, CICS automatically configures the `cicsAllAuthenticated` role for you in the `installedApps.xml` file.

## Results

Requests to the Java application are authenticated by using a distributed identity (an LDAP user ID) and then the LDAP user ID is mapped to a RACF user ID, which is used to authorize the transaction by using CICS transaction security.

In the case that a distributed identity filter is not configured, the RACF user ID running the CICS task is the user ID set in the matching URIMAP resource. If no user ID is set in the matching URIMAP, the CICS default user ID is used.

## Authorization using SAF roles

Role-based authorization can be enforced by using Liberty SAF role authorization. This way, you can use the roles defined in the web application to authorize the authenticated users based on the rules defined in the matching EJBROLE profiles.

This can be used as an alternative or in addition to CICS transaction security and authorization based on URI mapping to individual transactions.

1. Enable the `cicsts:distributedIdentity-1.0` feature in the `server.xml`. This feature enables the distributed identity mapping and SAF authorization. If further settings on SAF authorization are necessary the `<safAuthorization id="saf"/>` element can be added in the `server.xml` file with additional attributes.

**Recommended:** You can also add `racRouteLog="ASIS"` to the Liberty `<safAuthorization>` element to record the access attempts in the manner that is specified in the RACF EJBROLE profile.

2. Configure the SAF role pattern to be used by the Liberty server by using the `<safRoleMapper>` element in the `server.xml`:

```
<safRoleMapper profilePattern="%profilePrefix%.%resource%.%role-mapping%"
toUpperCase="true"/>
```

### **<safRoleMapper>**

Determines the name of the RACF EJBROLE profiles that are used to validate the user authorization.

### **toUpperCase**

Controls whether the EJBROLE profile name is case-sensitive. `toUpperCase="true"` specifies it to be not case-sensitive.

The default value is `false`.

### **profilePattern**

The default value is `"%profilePrefix%.%resource%.%role%"`. It uses three substitution variables:

- `%profilePrefix%` takes the value of the server `profilePrefix` configured in `<safCredentials>`.
- `%resource%` takes the value of the application name as defined in the `<application>` element, or the name of the archive if no `<application>` element exists.

The application name is also printed by the Liberty server in a CWWKZ0001I message when the application starts.

- `%role%` takes the value of a role that is defined either in the web application deployment descriptor (`web.xml`) or in the `ServletSecurity` annotation.

3. Create the EJBROLE profiles in RACF, regarding the prefix scheme described earlier:

```
RDEFINE EJBROLE safProfilePrefix.MYAPP.roleA UACC(NONE)
SETROPTS RACLIST(EJBROLE) REFRESH
```

4. Authorize user *mappedUserid* to call the application:

```
PERMIT safProfilePrefix.MYAPP.roleA CLASS(EJBROLE) ACCESS(READ) ID(mappedUserid)
SETROPTS RACLIST(EJBROLE) REFRESH
```

Instead of granting access to a specific RACF user ID, you can also grant access to a group of RACF user IDs and add the user ID to that group.

## Results

Requests to the Java application are authenticated by using a distributed identity (an LDAP user ID) and then the LDAP user ID is mapped to a RACF user ID.

This RACF user ID is used to authorize access to the web application based on the roles defined in the web application and the permissions defined in the EJBROLE profile.

## Related tasks

[Configuring CICS transaction security for a Liberty JVM server](#)

[Configuring SAF authorization with an EJBROLE](#)

## Related reference

[RACMAP \(Create, delete, list, or query a distributed identity filter\)](#)

## Configuring the syncToOSThread function

Use the Liberty syncToOSThread function to authorize access to resources outside of CICS control such as zFS files.

## Before you begin

Before you begin this task, you must complete these tasks:

- Enable the CICS Liberty security feature (`cicsts:security-1.0`).
- Complete the task [Configuring and starting the Liberty angel process](#).
- Complete the task [Connecting a Liberty JVM server to the angel process](#).

You must have:

- Authorization to issue the RACF PERMIT command.
- Write access to the `server.xml` configuration file.
- Authorization to create or update the application security constraint in the deployment descriptor (`web.xml`).
- The SAF registry must be used for authentication, Liberty authorized SAF services must be available and the Liberty server must be connected to the angel process.

## About this task

In this task, you use the syncToOSThread function of Liberty to enable a Java security subject, authenticated by Liberty, to be synchronized with the operating system (OS) thread identity.

Without syncToOSThread, the operating system thread identity defaults to be the CICS region user ID. This identity is used to authorize access to resources outside of CICS control such as zFS files. With syncToOSThread in effect, the user's subject is used to access these operating system resources.

This task assumes the following definitions:

- The CICS region user ID is *regionUserid*.
- The authenticated Java security subject has a RACF user ID of *authenticatedUserid*.

- The profile prefix that is used by the CICS Liberty server is *safProfilePrefix*.

## Procedure

1. Configure the syncToOSThread configuration element in the Liberty `server.xml` and add the required `<env-entry/>` to each web application's deployment descriptor by following steps 1 and 2 in [Enabling syncToOSThread for applications](#).
2. Grant the Liberty server permission to perform syncToOSThread operations by configuring SAF with either of the following profiles:
  - Grant the CICS region user ID CONTROL access to the `BBG.SYNC.safProfilePrefix` profile in the FACILITY class, where *safProfilePrefix* is specified on the `<safCredentials/>` element:

```
PERMIT BBG.SYNC.safProfilePrefix ID(regionUserId) ACCESS(CONTROL) CLASS(FACILITY)
```

This method allows the Liberty server to synchronize any authenticated Java security subject with the OS thread identity, so it can be easier to configure but is a less granular security control.

- Grant the CICS region user ID READ access to the `BBG.SYNC.safProfilePrefix` profile in the FACILITY class. Additionally, grant the CICS region user ID READ access to one or more `BBG.SYNC.authenticatedUserId` profiles in the SURROGAT class:

```
PERMIT BBG.SYNC.safProfilePrefix ID(regionUserId) ACCESS(READ) CLASS(FACILITY)
```

```
PERMIT BBG.SYNC.authenticatedUserId ID(regionUserId) ACCESS(READ) CLASS(SURROGAT)
```

This method requires that permission must be given for each authenticated user ID that needs to be synchronized with the OS identity.

**Restriction:** A servlet that is configured as the welcome page in `web.xml`, does not support the syncToOSThread function.

## Results

The authenticated RACF user ID from the Java security subject is synchronized with the operating system (OS) thread identity and used to authorize access to resources outside of CICS control.

## Configuring OAuth 2.0 for authorization

OAuth 2.0 is an open standard for delegated authorization supported by WebSphere Liberty. In this example a Liberty server is configured to use the OAuth 2.0 authorization code flow to authorize access to a protected web application.

### Before you begin

Ensure that the CICS region is configured to use SAF security and is defined with `SEC=YES`. See [Configuring Liberty to use a SAF user registry](#).

Configure the Liberty angel process to provide authentication and authorization services to the Liberty JVM server. See [Configuring the angel process](#).

For more information about OAuth, see [How it works: OAuth 2.0](#).

### About this task

The following procedure covers how to:

- Define an OAuth service provider.
- Grant an OAuth 2.0 token using the authorization code flow.
- Use this token to access a protected resource in a Liberty web application.



## Procedure

1. Add the `oauth-2.0` and the `cicsts:security-1.0` features to the `featureManager` element in `server.xml`.

```
<featureManager>
  <feature>oauth-2.0</feature>
  <feature>cicsts:security-1.0</feature>
</featureManager>
...
```

OAuth is an HTTP-based protocol and a web application is supplied by Liberty to secure access to the authorization and token endpoints. The web application is built in and is started automatically when you specify the `oauth-2.0` feature.

2. Set up role mapping for the Liberty OAuth web application by using the `oauth-roles` element.

Map the authenticated role to one or more users, groups, or special subjects. Additional roles, `clientManager` and `tokenManager`, are supplied but it is not necessary to map these roles for OAuth authorization to function. The following example shows the special subject of `ALL_AUTHENTICATED_USERS` being used to grant access to the OAuth web application.

```
<oauth-roles>
  <authenticated>
    <special-subject type="ALL_AUTHENTICATED_USERS"/>
  </authenticated>
</oauth-roles>
```

3. Configure an OAuth 2.0 provider and a client using the `oauthProvider` element. The provider must have at least one client defined.

You can define clients in any of the following ways:

- Locally with the `localStore` and `client` elements. This is the approach this example uses.
- In a relational database with the `databaseStore` element.
- In a custom OAuthStore implementation with the `customStore` element.

**Restriction:** Db2 JDBC type 2 connectivity is not supported for persistent OAuth 2.0 services.

In the following example, a client named *ClientApp* is defined locally and the OAuth provider specifies a scope of email and secures access to the URL `oauthapp`. This ensures that only clients with a valid access token can be authorized to access the protected resource.

```
<oauthProvider id="myProvider" filter="request-uri%="oauthapp" oauthOnly="true" >
  <localStore>
    <client name="ClientApp" secret="mySecret" scope="email" redirect="https://
client.example.ibm.com/webapp/redirect" />
  </localStore>
</oauthProvider>
```

The provider must have at least one client defined, with a unique name, a secret password, and the redirect URI of the application.

### Important:

Although it is not shown in this example, it is important to encode passwords and limit access to `server.xml` configuration. Passwords can be encoded by using the Liberty `securityUtility`, found in `USS_HOME/wlp/bin/securityUtility`. For more information, see [securityUtility command](#).

4. To initiate the authorization code flow, the client application accesses the authorization endpoint, passing in the `response_type` of `code` and the `client_id`, `client_secret`, `redirect_uri`, scope configured in Step “3” on page 325 as query parameters on a HTTP GET request.

This flow is normally performed programmatically in the client application. The following query shows an example web request.

```
https://zos.example.ibm.com/oauth2/endpoint/myProvider/authorize?
response_type=code&client_id=ClientApp&client_secret=mySecret&redirect_uri=https://
client.example.ibm.com/webapp/redirect&scope=email
```

5. Upon receiving the request, the Liberty authorization server redirects the resource owner (user) to authenticate using a customizable login form, and then prompts the resource owner to confirm the email authorization scope should be granted.
6. The user is then redirected back to the redirect URI. The Liberty authorization endpoint returns the authorization code as a query parameter of the URI.

The client application must store this authorization code to request an access token.

7. The client application requests an access token by sending the authorization code that was received from the authorization endpoint, along with the `client_id` and the pre-defined secret password in an HTTP POST request to the token endpoint.

This flow is normally performed programmatically in the client application. The following example shows the parameters required.

```
POST https://zos.example.ibm.com/oauth2/endpoint/myProvider/token HTTP/1.1
Content-Type: application/www-form-urlencoded

grant_type=authorization_code&
code=<code>&
client_id=ClientApp&
client_secret=mySecret&
redirect_uri=https://client.example.ibm.com/webapp/redirect
```

8. The authorization server returns a JSON document that contains the access token, its expiry time, and its authorized scope.

For example:

```
{
  "access_token": "AvTQKrk6H6Uqw4E5ffATHRHuq7d7PEmqP5ss32M1",
  "token_type": "Bearer",
  "expires_in": 7199,
  "scope": "email",
  "refresh_token": "8X5cwCDUC1YRw8nbCTEHkciGLozne9woR1tCXNAMQ4aGVSZ1GY"
}
```

9. You can then use this token to access the protected resource on the resource server by passing the request as a bearer token in an HTTP Authorization header.

Liberty then uses the OAuth access token to create a security subject, which can then be used to authorize access to the protected resource.

**Note:** In Enterprise Java applications, roles are used to authorize access to application URIs or Java methods. When using OAuth2.0, you need to define appropriate role security rules to authorize access to the protected resource based on the security subject created from the access token. See [Role authorization](#).

## Results

The OAuth 2.0 access token can be used by the client application to access a protected web application running in the resource server. Requests to the web application are mapped to the Java security subject that was created when the resource owner authenticated with the authorization server. By using the CICS security feature, you ensure that the SAF user ID associated with the security subject is used as the CICS task user ID for any subsequent CICS security authorization checks for the CICS transaction that runs the web request.

## Configuring confidentiality and integrity in CICS Liberty

TLS provides transport layer security that includes confidentiality, integrity, and authentication to secure the connection between a client and a Liberty JVM server.

You configure TLS for a Liberty JVM server by updating the `server.xml` file and by storing certificates in keystores and truststores. The keystore and truststore types that are supported by CICS Liberty are SAF keyrings, Public Key Cryptography Standards #12 (PKCS12), and Java KeyStores (JKS).

For more information about design guidance on which TLS options to use with CICS Liberty, see [Designing security for CICS Liberty applications](#).

The following tasks outline the configuration steps for different TLS configurations.

### Configuring TLS for a Liberty JVM server by using RACF

Configure a TLS connection between a web client and a CICS Liberty JVM server.

#### Before you begin

You must be familiar with:

- The information in [How it works: Integrity and confidentiality in CICS](#).

You must have:

- Authorization to issue **RACDCERT** commands to manage RACF key rings and certificates.

For more information about the **RACDCERT** commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\)](#).

- Access to the `keytool` command for creating Java KeyStores (JKS) and certificates for the client. This command is provided with the IBM SDK for z/OS, Java Technology Edition.

For more information about the `keytool` command, see [Keytool](#) in the *IBM SDK, Java Technology Edition Security Guide*.

- Write access to the `server.xml` configuration file.

#### About this task

This configuration creates the following artifacts. For more information about each asset such as the keystore and truststore, see [How it works: CICS Liberty security](#).

- A CA certificate (`cicsCA`) that is used to sign the server certificate (`cics1Cert`) of the Liberty JVM server.
- The personal RACF certificate (`cics1Cert`) signed by the CA certificate (`cicsCA`), which is used by the Liberty JVM server to identify itself over TLS connections.
- A RACF key ring that acts as the server's keystore to store the server's personal certificates.
- A Java KeyStore (JKS) that acts as the client's truststore so that the client can trust the server by importing the server's CA certificate (`cicsCA`).

It is important to understand that any web request to a Liberty JVM server uses Java Secure Sockets Extension (JSSE) as the TLS implementation, and not the CICS sockets domain.

This task does not include the additional configuration that is required for TLS client authentication. See [Configuring TLS client authentication for a Liberty JVM server by using RACF](#).

The certificates, key rings, and JKS files that are used in this task are described in [Figure 131 on page 328](#).

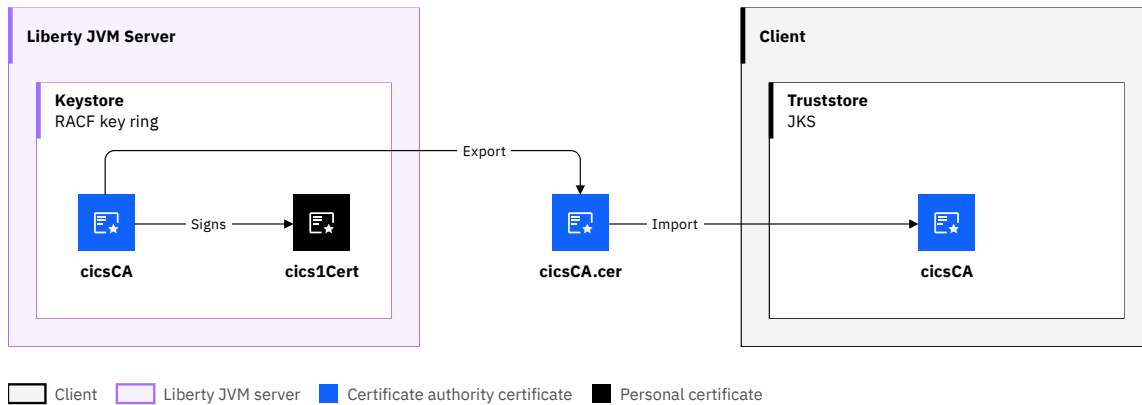


Figure 131. Certificates that are used for TLS connection

In this diagram:

**cicsCA**

Is the label of the CICS CA certificate.

**cics1Cert**

Is the label of the CICS server certificate.

This task makes the following assumptions:

- RACF is the security manager for the Liberty JVM server. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.
- The Liberty JVM server uses RACF to create and store certificates.
- The certificate that is created for the Liberty JVM server is a personal certificate (cics1Cert) that is signed by a certificate authority (CA) certificate (cicsCA).
- The client uses a JKS file to store certificates. Other keystore types or SAF key rings might be used depending on the client and the operating system that it runs on. If the client keystore is a different type (for example, a browser keystore) then for the client TLS configuration you must follow the instructions for the specific keystore type.
- The keytool command is used to create the JKS file for the client.

## Procedure

The following diagram shows the flow of the procedure and where each step is performed. You use RACF to create the keystore and certificates for the server during steps [“1” on page 329](#)–[“5” on page 331](#), and then configure the truststore and certificates for the client during steps [“6” on page 331](#)–[“9” on page 332](#). Finally, you configure the Liberty JVM server to use the RACF key ring during steps [“10” on page 332](#)–[“14” on page 333](#).

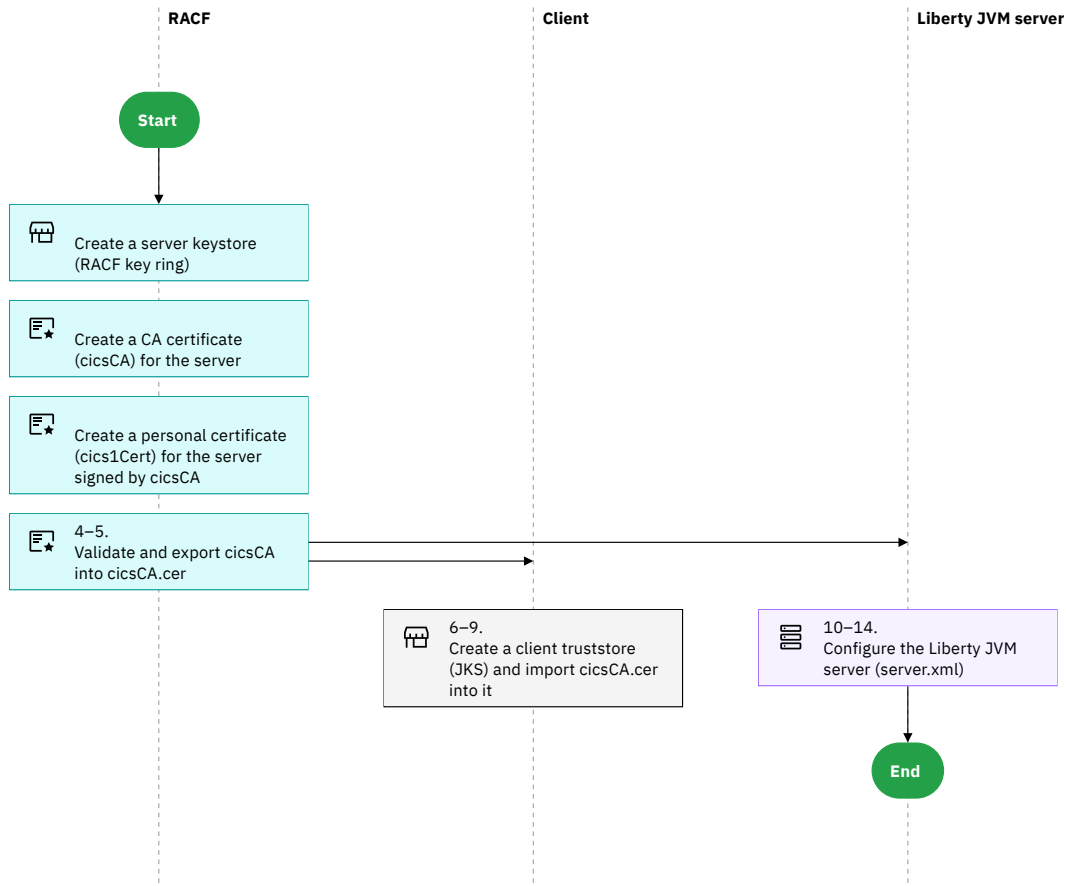


Figure 132. Flow of the task

### Create a keystore and certificates for the Liberty JVM server using RACF

1. Create a RACF key ring for the Liberty JVM server to use as its keystore.

Enter the following command:

```
RACDCERT ID(CICS1) ADDRING(Keyring.CICS1)
```

The command uses the following values:

- CICS1 is the CICS region user ID.
- Keyring.CICS1 is the name of the key ring to be created.

2. Create a CA certificate for the Liberty JVM server.

- a. Create a self-signed RSA key pair to act as a CA certificate. A *key pair* consists of a public and private key.

Enter the following command:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(CN('CA for CICS') OU('CICS') O('IBM') C('US'))
SIZE(2048) WITHLABEL('cicsCA') NOTAFTER(2029-12-31))
```

The command uses the following values:

- CN('CA for CICS') OU('CICS') O('IBM') C('US') is an example distinguished name (DN) for the certificate.
- 2029-12-31 is the expiry date of the certificate.

- `cicsCA` is the *label* or *alias* of the certificate.

The term *label* is used by RACF and *alias* is used by JKS to reference the same artifact, therefore in this documentation, the phrase *label, or alias*, is used for clarity.

- Connect the CA certificate to the key ring.

Enter the following command:

```
RACDCERT ID(CICS1) CONNECT(RING(Keyring.CICS1) LABEL('cicsCA') CERTAUTH)
```

The command uses the following values:

- `CICS1` is the user ID that owns the key ring.
- `Keyring.CICS1` is the name of the key ring.
- `cicsCA` is the label, or alias, of the certificate to be connected to the key ring.

The `SIZE` of the certificate must be a minimum of 2048 bits. For more information, see the [RACF RACDCERT GENCERT \(Generate certificate\) command](#).

- Create a personal certificate, signed by the CA certificate, for the Liberty JVM server.

- Create an RSA key pair for the Liberty JVM server signed by the CA certificate.

Enter the following command:

```
RACDCERT ID(CICS1) GENCERT SUBJECTSDN(CN('myServer.host.com') OU('CICS') O('IBM') C('US')) SIZE(2048) SIGNWITH(CERTAUTH LABEL('cicsCA')) WITHLABEL('cics1Cert') NOTAFTER(DATE(2029-12-31))
```

The command uses the following values:

- `CN('myServer.host.com') OU('CICS') O('IBM') C('US')` is an example distinguished name (DN) for the certificate. The common name (CN) value is typically the hostname of the z/OS LPAR that hosts the Liberty JVM server.
- `cicsCA` is the label, or alias, for the CA certificate that is used to sign the personal certificate.
- `cics1Cert` is the label, or alias, for the personal certificate to be created and signed.

- Connect (add) the personal certificate to the key ring.

Enter the following command:

```
RACDCERT ID(CICS1) CONNECT(RING(Keyring.CICS1) LABEL('cics1Cert'))
```

The command uses the following values:

- `CICS1` is the user ID that owns the key ring.
- `Keyring.CICS1` is the name of the key ring.
- `cics1Cert` is the label or alias of the personal certificate to be connected to the key ring.

- Confirm that the key ring and certificates were created correctly.

- List the certificates in the key ring.

Enter the following command:

```
RACDCERT ID(CICS1) LISTRING(Keyring.CICS1)
```

The following screen capture shows the expected response:

```
Ring:
>Keyring.CICS1<
Certificate Label Name      Cert Owner      USAGE      DEFAULT
-----
cicsCA                     CERTAUTH       CERTAUTH    NO
cics1Cert                  ID(CICS1)      PERSONAL    NO
```

- List the details of the CA certificate.

Enter the command:

```
RACDCERT CERTAUTH LIST(LABEL('cicsCA'))
```

The expected response contains the following information:

```
Issuer's Name:
>CN=CA for CICS.OU=CICS.O=IBM.C=US<
Subject's Name:
>CN=CA for CICS.OU=CICS.O=IBM.C=US<
```

- c. List details of the Liberty JVM server's personal certificate.

Enter the command:

```
RACDCERT ID(CICS1) LIST(LABEL('cics1Cert'))
```

The expected response contains the following information:

```
Issuer's Name:
>CN=CA for CICS.OU=CICS.O=IBM.C=US<
Subject's Name:
>CN=myServer.host.com.OU=CICS.O=IBM.C=US<
```

5. Export the Liberty JVM server's CA certificate, containing the public key, to a z/OS sequential file.

Enter the following command:

```
RACDCERT CERTAUTH EXPORT(LABEL('cicsCA')) DSN('CICS.CERTS.CICSCA') FORMAT(CERTDER)
```

The command uses the following values:

- `cicsCA` is the label or alias for the CA certificate.
- `'CICS.CERTS.CICSCA'` is the name of the target z/OS sequential file.
- `CERTDER` specifies a DER encoded X.509 certificate.

### Configure a truststore and certificates for the client

6. Prepare to use the keytool command.

- a. Create a directory to contain the certificates and JKS files to be created by the keytool command.

In a later step, you copy those files to other locations, so if you run the keytool command on z/OS, make the location a z/OS UNIX System Services (USS) directory. For example, `/u/myuser/cicsJKS`.

- b. Add the `keytool` command to the `PATH` environment variable.

For example,

```
export PATH=$PATH:<javaInstallPath>/bin
```

The value `<javaInstallPath>` is the installation path of the IBM SDK for z/OS that includes the keytool command.

7. Import the CA certificate used to sign the Liberty JVM server's personal certificate into the client's JKS truststore as a trusted certificate.

- a. Transfer in binary format, the z/OS sequential file `'CICS1.CERTS.CICSCA'` containing the exported CA certificate to the client's workstation.

The method that you use depends on the operating system on which the client is running and the file transfer utilities you have access to. You might be able to FTP to the z/OS LPAR and get the z/OS sequential file directly. Alternatively you might first need to copy the z/OS sequential file to a USS file, and then get the UNIX System Services file. For example,

```
cp "'CICS.CERTS.CICSCA'" /u/myuser/cicsJKS/cicsCA.cer
```

- b. Name the CA certificate file that you received onto the client system as `cicsCA.rec`.
- c. Enter the following command from the USS directory that you created in step six of [Configuring TLS for a Liberty JVM server using RACF](#):

```
keytool -importcert
-file cicsCA.cer
-alias cicsCA
-keypass passw0rd
-keystore clientTrust.jks
-storepass passw0rd
-storetype jks
```

The command uses the following values:

- `cicsCA` is the alias to be given to the CA certificate to be imported.
- `cicsCA.cer` is the name of the certificate file that contains the public key of the CA certificate to be imported.
- `clientTrust.jks` is the name of the JKS file into which the certificate is to be imported. This file is used as the client's truststore.

If the JKS file does not exist, the command creates the file and issues the following prompt:

Trust this certificate? [no]:

Enter yes in response.

8. Check that the CA certificate was added to the client's truststore correctly.

List the contents of the client's truststore by issuing the following command from the UNIX System Services directory that you created in step six of [Configuring TLS for a Liberty JVM server using RACF](#):

```
keytool -list -v -keystore clientTrust.jks
```

The expected response contains the following information:

```
Your keystore contains 1 entry
Alias name: cicsca
Entry type: trustedCertEntry
Owner: CN=CA for CICS, OU=CICS, O=IBM, C=US
Issuer: CN=CA for CICS, OU=CICS, O=IBM, C=US
```

9. Make the client's truststore JKS file available to the client.

In this task, the client's truststore is called `clientTrust.jks`. Transfer this file in binary mode to the workstation that hosts the client. The instructions for configuring your client to use these JKS files depend on your client application. For more information, see the documentation for the client application.

### Configure the Liberty JVM server by updating the `server.xml` configuration file

10. Enable the Liberty Transport Security 1.0 feature (`transportSecurity-1.0`):

```
<featureManager>
...
  <feature>transportSecurity-1.0</feature>
</featureManager>
```

11. Configure the HTTPS port for the Liberty JVM server.

Identify an unused TCP/IP port on your z/OS LPAR, which can be used as the HTTPS port for the Liberty JVM server. Specify the port value on the `httpsPort` attribute of the `httpEndpoint` element in the `server.xml` configuration file.

For example, to use 9443 as the default HTTPS port, add the following element to the configuration:

```
<httpEndpoint id="httpEndpoint1" host="*"
  httpPort="9080" httpsPort="9443"/>
```

12. Configure the SSL configuration to be used by the Liberty JVM server.



Create an `ssl` element in the `server.xml` configuration file, with unique values for the `id` and `keyStoreRef` attributes, and name this in an `sslDefault` element. For example,

```
<ssl id="customSSLConfiguration"
    keyStoreRef="SAFKeyStore" />
<sslDefault sslRef="customSSLConfiguration"/>
```

This example uses the `customSSLConfiguration` element `id`. If you use a non-default `ssl` element `id` you should also specify the default `ssl` element using the `sslDefault` element as shown. For more information about the `ssl` element, see [Server configuration](#).

13. Configure the Liberty JVM server to reference the RACF key ring that contains the server's personal certificate and CA certificate.

Create a `keyStore` element in the `server.xml` configuration file using the `SAFKeyStore` element `id`. For example,

```
<keyStore id="SAFKeyStore"
    fileBased="false"
    location="safkeyring://CICS1/Keyring.CICS1"
    password="password"
    readOnly="true"
    type="JCERACFKS" />
```

If you are running Java 11, the location must be `location="safkeyringjce://CICS1/Keyring.CICS1"`.

The element uses the following values:

- `SAFKeyStore` must match the value that is specified on the `keyStoreRef` attribute of the `ssl` element.
- The `location` value must be the RACF key ring that acts as the server's keystore.
- `CICS1` is the user ID that owns the key ring.
- The `password` attribute is mandatory, so a value must be specified. However, the value is not used when `type="JCERACFKS"` because RACF key rings are not secured with passwords.

14. Start, or restart the Liberty JVM server, to pick up the changes.

The `messages.log` file contains the following message:

```
CWWK00219I: TCP Channel httpEndpoint1-ssl has been started and is now
listening for requests on host * (IPv6) port 9443
```

The message uses the following values:

- `httpEndpoint1-ssl` is the `id` attribute value of the `httpEndpoint` element followed by `-ssl`.
- `*` is the value of the `httpEndpoint` element `host` attribute.
- `9443` is the value of the `httpEndpoint` element `httpsPort` attribute.

## Results

When the client connects to the Liberty JVM server by using `https` (`http` over a TLS connection) a TLS handshake occurs. The client knows that the Liberty JVM server can be trusted because the certificate authority (CA) that signed the server's certificate is trusted. A TLS session is set up.

## What to do next

You might want to configure TLS client authentication for this connection by following the steps in [Configuring TLS client authentication for a Liberty JVM server by using RACF](#).

## Configuring TLS client authentication for a Liberty JVM server by using RACF

Configure a TLS client authenticated connection between a client and a CICS Liberty JVM server.

### Before you begin

You must be familiar with the information in [How it works: Integrity and confidentiality in CICS](#).

You must complete the prerequisite task [Configuring TLS for a Liberty JVM server by using RACF](#).

You must have :

- Authorization to issue RACDCERT commands to manage RACF key rings and certificates.  
For more information about the RACDCERT commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\)](#).
- Access to the keytool command for creating a Public Key Cryptography Standards #12 keystore (PKCS#12) and certificates for the client. This command is provided with the IBM SDK for z/OS, Java Technology Edition.  
For more information about the keytool command, see [Keytool](#).
- Write access to the `server.xml` configuration file.

### About this task

This task describes configuration for TLS client authentication, also called mutual TLS authentication, to require the client to provide its personal certificate on the connection. It does not include the steps that enable the Liberty JVM server to use TLS, as described in the prerequisite task [Configuring TLS for a Liberty JVM server by using RACF](#).

This configuration creates these artifacts. For more information about each asset such as the keystore and truststore, see [How it works: CICS Liberty security](#).

- A certificate authority (CA) certificate (clientCA) that is used to sign the client certificate (clientCert).
- The CA signed personal certificate (clientCert) for the client to identify itself on TLS connections.
- A PKCS#12 keystore to act as the client's keystore.

A PKCS#12 keystore file has a `.p12` file name extension, so this type of keystore can be designated as a p12 file.

The following diagram shows certificates that are used in this task. The artifacts that are shown within the client keystore and the Liberty JVM server truststore are the new artifacts created for this task. You create the other artifacts by completing the task [Configuring TLS for a Liberty JVM server by using RACF](#).

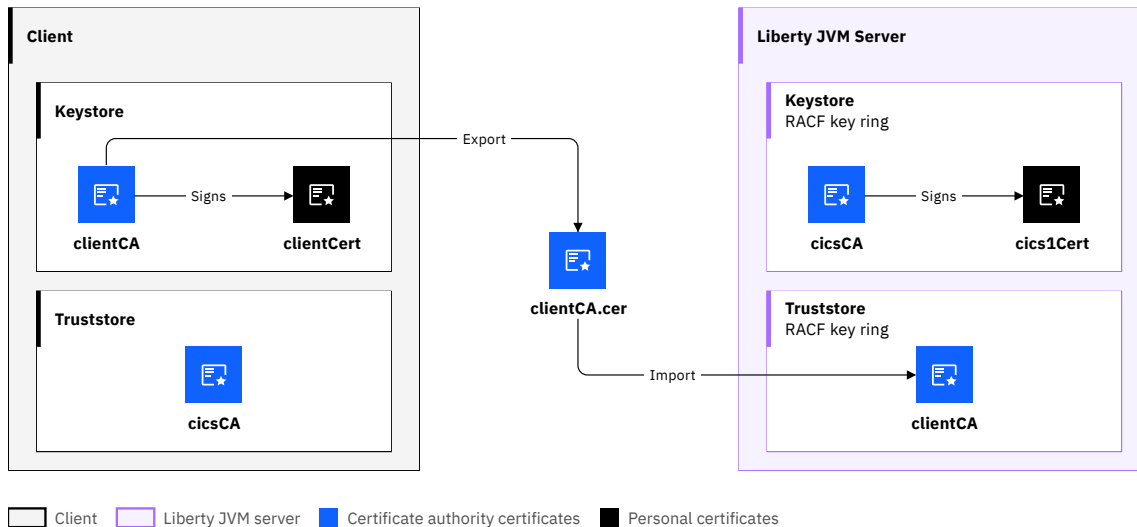


Figure 133. Certificates used for a client authenticated TLS connection

In this diagram:

**cicsCA**

Is the label of the CICS CA certificate.

**cics1Cert**

Is the label of the CICS server certificate.

**clientCA**

Is the label of the client CA certificate.

**clientCert**

Is the label of the client certificate.

This task makes these assumptions:

- RACF is the security manager for the Liberty JVM server. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.
- The Liberty JVM server uses RACF to create and store certificates.
- The client uses a p12 file to store certificates. Other keystore types or SAF key rings might be used depending on the client and the operating system that it runs on. If the client keystore is a different type (for example, a browser keystore) then for the client TLS configuration you must follow the instructions for the specific keystore type.
- The keytool command is used to create the p12 file for the client.

## Procedure

The following diagram shows the flow of the procedure and where each step is performed. You create a keystore and certificates for the client during steps “1” on page 336 to “4” on page 338, and then connect the truststore and certificates in steps “5” on page 338 and “6” on page 339. Finally, you configure the TLS connection by updating the `server.xml` configuration file of the Liberty JVM server during steps “7” on page 339–“9” on page 339.

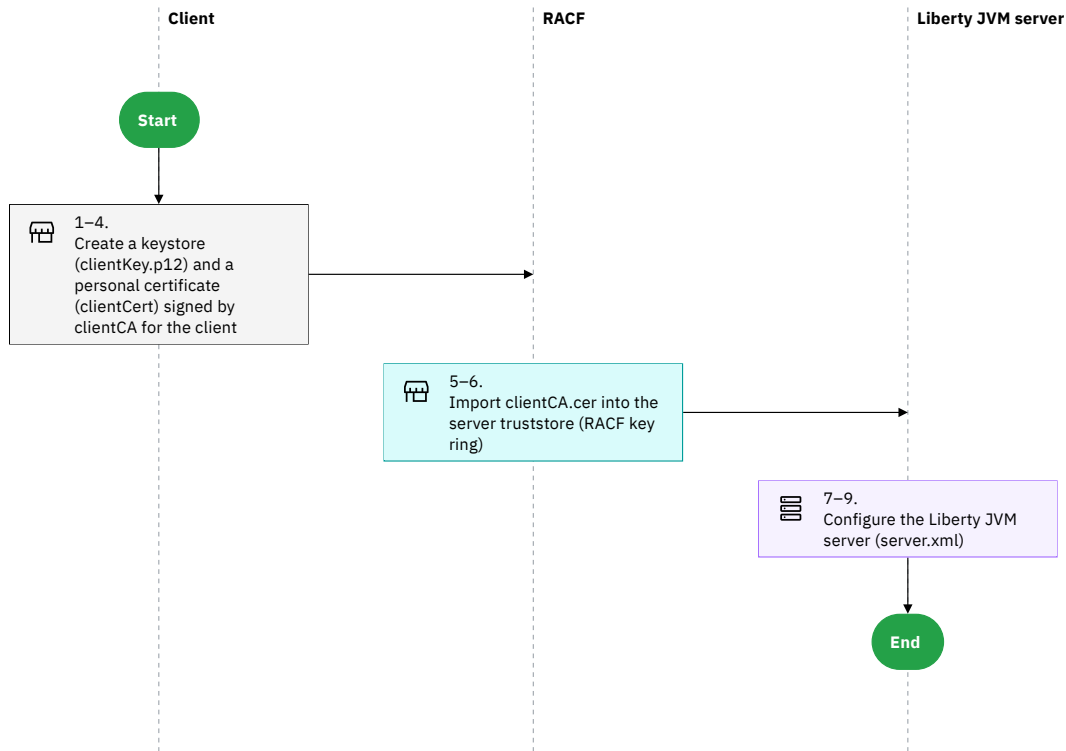


Figure 134. Flow of the task

**Create the TLS client certificate that the client presents to the server.** If the client already has a certificate, these steps can be skipped.

1. Create a self-signed certificate for the client.

Generate a self-signed RSA key pair for the client and add them to a p12 file, which acts as the client's keystore.

Enter this command from USS:

```
keytool -genkeypair
        -alias clientCert
        -dname "CN=myClient.host.com, O=IBM, C=US"
        -keyalg RSA
        -keypass passw0rd
        -keysize 2048
        -keystore clientKey.p12
        -storetype PKCS12
        -storepass passw0rd
        -validity 365
```

The command uses these values:

- `clientCert` is the alias of the personal certificate to be created.
  - `CN=myClient.host.com, O=IBM, C=US` is an example distinguished name (DN) for the certificate. The CN value is typically the hostname of the client that owns the certificate.
  - `clientKey.p12` is the name of the dynamically created p12 file that is used as the client's keystore.
  - `passw0rd` is the example password that is used for the key and the keystore, it is recommended to choose your own password.
2. Sign the client's personal certificate with a CA certificate.

A certificate signing request (CSR) is created for the self-signed personal certificate and sent to a certificate authority (CA). The CA verifies the request and returns a CA signed version of the personal certificate, a root CA certificate and optionally, an intermediate CA certificate.

- Create a certificate signing request (CSR) for the personal certificate.

Enter this command from USS:

```
keytool -certreq
  -alias clientCert
  -keystore clientKey.p12
  -storetype PKCS12
  -file clientCert.csr
  -storepass passw0rd
```

The command uses these values:

- `clientCert` is the alias of the personal certificate to be signed.
  - `clientKey.p12` is the name of the P12 file that contains the personal certificate to be signed.
  - `clientCert.csr` is the name of the CSR file to be created.
- Send the CSR (`clientCert.csr`), to your preferred certificate authority with any additional details they require.

The certificate authority returns a root CA certificate, the signed client's personal certificate, and optionally, an intermediate CA certificate.

- Import the root CA certificate and if present, the intermediate CA certificate, into the client's p12 keystore.

Enter this command from USS:

```
keytool -importcert
  -file clientCA.cer
  -alias clientCA
  -keypass passw0rd
  -keystore clientKey.p12
  -storepass passw0rd
  -storetype PKCS12
```

After you run the command, you see this prompt:

```
Trust this certificate? [no]:
```

Enter yes in response.

The command uses these values:

- `clientCA` is the alias to be given to the root CA certificate to be imported.
- `clientCA.cer` is the name of the certificate file that the certificate authority returns. It contains the root CA certificate to be imported.
- `clientKey.p12` is the name of the p12 file into which the certificate is to be imported. This file is the client's keystore.

Repeat the command for the intermediate CA certificate if necessary.

- Import the signed client's personal certificate.

Enter this command from USS:

```
keytool -importcert
  -file clientCertSigned.cer
  -alias clientCert
  -keypass passw0rd
  -keystore clientTrust.p12
  -storepass passw0rd
  -storetype PKCS12
```

The command uses these values:

- `clientCert` is the alias to be given to the signed personal certificate to be imported. It must match the name of the original self-signed personal certificate so that one is replaced in the keystore.
  - `clientCertSigned.cer` is the name of the certificate file that the certificate authority returns, and contains the signed personal certificate to be imported.
  - `clientKey.p12` is the name of the p12 file into which the certificate is to be imported, which is the client's keystore.
3. Confirm that the client's keystore and personal certificate are created correctly.

Enter this command from USS to list the contents of the client's keystore.

```
keytool -list -v -keystore clientKey.p12 -storetype PKCS12
```

The expected response contains entries for the signed personal certificate with an `Entry` type of `PrivateKeyEntry` and for the root CA certificate, an entry with `Entry` type of `trustedCertEntry`.

4. Make the client's keystore p12 file available to the client.

The client's keystore that is created in this task is called `clientKey.p12`. Transfer this file in binary mode to the workstation that hosts the client. The instructions for configuring your client to use the p12 file depends on your client application. For more information, see the documentation for the client application.

### Connect the client's CA certificate to the server truststore

5. Connect the client's CA certificates to the Liberty JVM server's RACF key ring.

If the client CA certificates are already present in RACF, you can skip steps [“5.a” on page 338](#) and [“5.b” on page 338](#).

- a) Transfer the client's CA certificates to the z/OS LPAR where the Liberty JVM server is hosted, to a z/OS sequential file. Be careful on the transfer mode that you use. Typically a PEM certificate should be transferred in text format, whereas a DER certificate should be transferred in binary format. This example uses the z/OS sequential file `CICS.CERTS.CLIENTCA` to store the client's CA certificate. Use an appropriate sequential file name for your environment.

- b) Import the client's CA certificates into RACF.

Enter this command:

```
RACDCERT ADD('CICS.CERTS.CLIENTCA') CERTAUTH TRUST WITHLABEL('clientCA')
```

The command uses the following values:

- `CICS.CERTS.CLIENTCA` is the dataset that contains the CA certificate.
- `clientCA` is the certificate label to be assigned to the CA certificate.

Since the CA certificates are only used to verify that the client certificate can be trusted, they should be added to `CERTAUTH`.

- c) Enter this command:

```
RACDCERT ID(CICS1) CONNECT(CERTAUTH LABEL('clientCA') RING(Keyring.CICS1) USAGE(CERTAUTH))
```

The command uses the following values:

- `CICS1` is the user ID that owns the key ring.
- `Keyring.CICS1` is the name of the key ring that is used by the Liberty JVM server.
- `clientCA` is the CA certificate label to be connected to the key ring.

Usually a single RACF key ring that is used as keystore and truststore is sufficient for a Liberty JVM server. If you have two different key rings, then connect the client CA certificates in the key ring as truststore.

6. Confirm that the client's CA certificates are connected to the Liberty JVM server's key ring correctly.

Enter this command to list the certificates in the key ring:

```
RACDCERT ID(CICS1) LISTRING(Keyring.CICS1)
```

This screen capture shows the expected response:

```
Ring:
>Keyring.CICS1<
Certificate Label Name      Cert Owner      USAGE      DEFAULT
-----
cicsCA                     CERTAUTH       CERTAUTH    NO
cics1Cert                  ID(CICS1)      PERSONAL    NO
clientCA                   CERTAUTH       CERTAUTH    NO
```

### Configure the Liberty JVM server by updating the `server.xml` configuration file

7. Edit the SSL configuration to be used by the Liberty JVM server.

Edit the existing `ssl` element in the `server.xml` configuration file to set client authentication by using `clientAuthentication="true"`.

```
<ssl id="customSSLConfiguration"
    keyStoreRef="SAFKeyStore"
    clientAuthentication="true" />
<sslDefault sslRef="customSSLConfiguration"/>
```

This example uses the `customSSLConfiguration` element id. If you use a non-default `ssl` element ID, you should also specify the default `ssl` element using the `sslDefault` element as shown.

In cases where some clients have a client certificate, and TLS client authentication is not mandatory for every HTTPS connection, then the `clientAuthenticationSupported` attribute can be set to `true` and `clientAuthentication` doesn't need to be set. If `clientAuthentication` is set to `true`, the Liberty JVM server ignores the setting of `clientAuthenticationSupported`.

8. Optional: If you want to have a unique keyStore entry for both the keystore and the truststore, create a `keyStore` element in the `server.xml` configuration file for the Liberty JVM server's truststore. The key ring that contains the client CA certificates is to be used as truststore.

The `keyStore` element is also used for truststores:

```
<keyStore id="SAFTrustStore"
    fileBased="false"
    location="safkeyring:///Keyring.CICS1"
    password="password"
    readOnly="true"
    type="JCERACFKS" />
```

If you are running Java 11, the location must be `location="safkeyringjce:///Keyring.CICS1"`.

The element uses these values:

- `SAFTrustStore` must match the value that is specified on the `trustStoreRef` attribute of the `ssl` element.
- The `location` value must be the RACF key ring that acts as the server's truststore.
- The `password` attribute is mandatory, so a value must be specified. However, the value is not used when `type="JCERACFKS"` because RACF key rings are not secured with passwords.

Update the `ssl` element to use the newly defined truststore:

```
<ssl id="customSSLConfiguration"
    keyStoreRef="SAFKeyStore"
    trustStoreRef="SAFTrustStore"
    clientAuthentication="true" />
```

9. To pick up the changes, start, or if it was already running, restart the Liberty JVM server.

## Results

When the client connects to the Liberty JVM server by using HTTPS (HTTP over a TLS connection) a TLS handshake occurs. The client knows that the Liberty JVM server can be trusted because the certificate authority (CA) that signed the server's certificate is trusted. Also, the Liberty JVM server knows that the client can be trusted because the CA that signed the client's certificate is trusted. A TLS session is set up.

## What to do next

Once you have configured TLS client authentication, you can use the client certificate for user authentication. For more information about configuring a client certificate to authenticate with a Liberty JVM server, see [Configuration example: Securing a web application with TLS client authentication and SAF role authorization](#).



## Chapter 16. Security for SOAP web services

With the support provided by CICS Transaction Server for z/OS, you can either secure your SOAP messages with *transport security* (used for either SOAP or JSON messages) or *SOAP message security* (for SOAP messages only). For information about the architecture of SOAP web services, see [CICS and SOAP web services](#).

### Transport-based security

Transport-based security relies on technologies that are available as part of the TCP/IP and HTTP protocols. For example, TLS and HTTP basic authentication can be used to secure CICS web services. For information on TCP/IP and HTTP security options, see [Security for TCP/IP clients](#) and [Security for CICS web support](#).

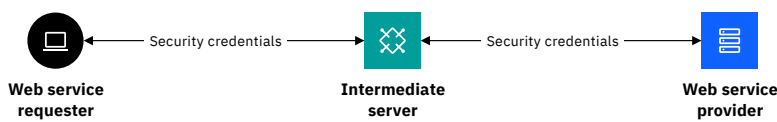


Figure 135. Transport-based security for SOAP messages

### SOAP message security

When you use transport-based security (as illustrated in [Figure 135 on page 341](#)), if the service requester identifies itself to an intermediate gateway, then the intermediate gateway identifies itself to the service provider, the target service normally runs with the identity of the intermediate gateway rather than the service requester. The Web Services Security (WSS): SOAP Message Security 1.0 specification (*WS-Security*) addresses this problem by allowing security credentials to be passed within the SOAP message itself, so that the credentials of the service requester can be passed through an intermediate gateway, and can still be used to identify the requester to the service provider.

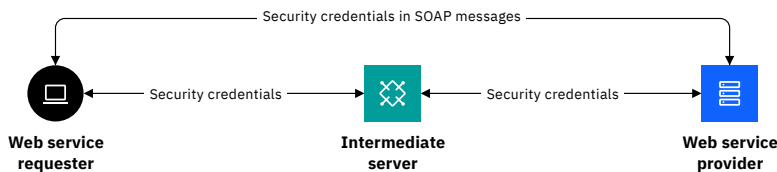


Figure 136. SOAP message security

For more information about SOAP message security, see [How it works: SOAP message security](#).

You can use a combination of transport-based security and SOAP message security to secure CICS web services. To decide which type, or which combination of capabilities, is best for you, see [Designing security for CICS web service providers](#) and [Designing security for CICS web service requesters](#).

## How it works: SOAP message security

CICS security for SOAP messages is based on support for standard web specifications, including the WS-Security and WS-Trust specifications. This support gives you various options for authentication, integrity, and confidentiality of SOAP messages. It is useful to understand the components that provide the support.

- [“Support for WS-Security, WS-Trust, and WS-Policy specifications” on page 342](#)

- [“What SOAP message security offers” on page 342](#)
- [“Components of SOAP message security” on page 343](#)

## Support for WS-Security, WS-Trust, and WS-Policy specifications

The Web Service Security (*WS-Security*) specification provides a foundational set of SOAP message extensions for building secure web services by defining elements to be used in the SOAP header for message-level security. It specifies the use of security tokens, digital signatures, and XML encryption to protect and authenticate SOAP messages. It specifies the use of digital signatures to provide integrity for XML elements in a SOAP message, and it specifies the use of encryption to provide confidentiality for XML elements in a SOAP message. The specification allows you to protect the body of the message or any XML elements within the body or the header. You can give different levels of protection to different elements within the SOAP message. For more information, see the [Web Services Security: SOAP Message Security 1.0 specification](#).

The *Web Services Trust Language (WS-Trust)* specification extends WS-Security by providing a framework for requesting and issuing security tokens, and managing trust relationships between web service requesters and providers. This extension to the authentication of SOAP messages enables web services to validate and exchange security tokens of different types by using a trusted third party. This third party is called a *Security Token Service (STS)*. For more information about WS-Trust, see the [Web Services Trust Language specification](#).

CICS does not support WS-Policy. Security constraints can be specified by using WS-Policy within the WSDL but these specifications are ignored by CICS. CICS uses the pipeline configuration file to specify the security handler. For more information, see [Configuring the pipeline for Web Services Security](#). CICS supports only the WS-Security 1.0 and WS-Trust 1.2 specifications.

## What SOAP message security offers

Figure 137 on page 342 shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the requester and a CICS web service provider. You configure SOAP message security either by using a [CICS security handler](#) or by using the [Trust client interface](#). The CICS security handler and the WS-Sec message header are both using WS-Security.

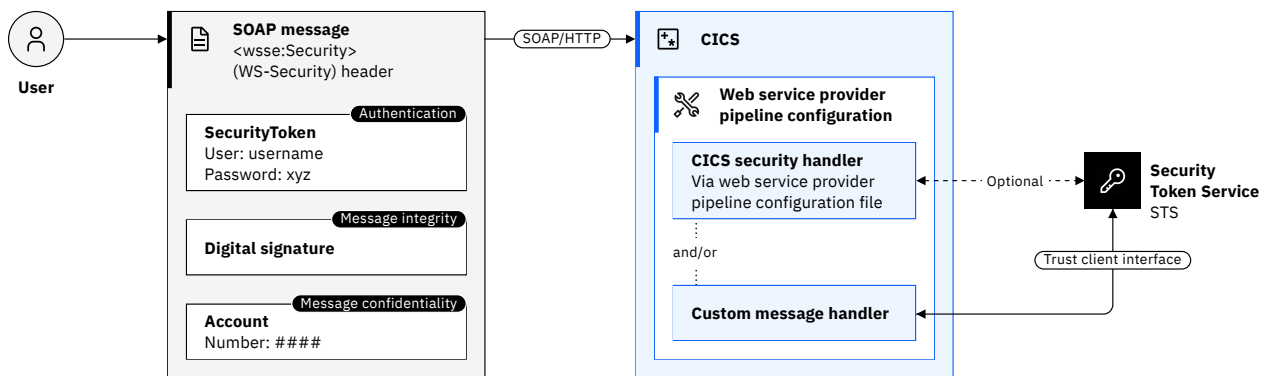


Figure 137. How CICS supports SOAP message security

SOAP message security in CICS offers you the following options for authentication and for message protection:

- Authentication mechanisms for deriving a user ID from an inbound message (CICS web service provider) or attaching a security token to an outbound message (CICS web service requester), including:
  - Basic authentication (for web service provider only)
  - X.509 certificate

- ICRX identity token (web service provider only)
- Identity assertion
- Operation with a trusted third party (Security Token Service)

For more information, see [“How it works: Authentication for CICS with SOAP message security”](#) on page 344.

- For message integrity, signature validation of inbound message signatures and signature generation for the SOAP body on outbound messages. For more information, see [“How it works: Signing SOAP messages”](#) on page 346.
- For message confidentiality, decryption of encrypted data in inbound messages and encryption of the SOAP body content on outbound messages. For more information, see [“How it works: SOAP message encryption”](#) on page 348.
- You can choose to both sign and encrypt a SOAP message to provide both message integrity and confidentiality. CICS always signs the SOAP message body first and then encrypts it.

**Note:** Web Services Security is potentially not conformant with SP800-131A. Web Services Security is configured by adding a security handler into the pipeline and CICS has no control over the processing in a user-written handler. If you use digital signatures, you can specify only the algorithms `dsa-sha1` and `rsa-sha1`. These algorithms are not SP800-131A-conformant. The two-key triple DES encryption algorithm, which can be used to encrypt a SOAP body, is also nonconformant.

## Components of SOAP message security

### CICS security handler

To implement WS-Security in CICS for either a web service provider or a web service requester, you configure a security handler in the pipeline configuration file. CICS provides a security handler to cover the most common scenarios. For more information about using the CICS security handler, see [Configuring the pipeline for Web Services Security](#).

To use your own security procedures and processing, you can write a custom security handler to process secure SOAP messages. For information about writing your own security handler, see [Writing a custom security handler](#).

In addition to the CICS security handler, CICS provides a Trust client interface so that you can interact with a Security Token Service (STS) without using the CICS security handler. For information about using this interface, see [Invoking the Trust client from a message handler](#).

### Security Token Service (STS)

An STS is a web service that acts as a trusted third party to broker trust relationships between a web service requester and a web service provider, allowing web services from different trust domains to communicate successfully. Like a certificate authority in a TLS handshake, the STS guarantees that the service requester and the service provider can trust the credentials that are provided in the message. The trust is represented through the exchange of security tokens. An STS can issue, exchange, and validate these security tokens, and, in doing so, establish trust relationships between different trust domains.

The STS enables CICS to accept and send messages that have security tokens in the message header that are not directly supported by the CICS security handler; for example, LTPA and Kerberos tokens. CICS can be configured as a Security Token Service for SAML assertions. For information about using SAML assertions, see [Configuring CICS for SAML](#).

CICS acts as a Trust client. The CICS security handler uses the information in the pipeline configuration file to send a web service request to the STS.

In a service provider pipeline, depending on how you configure the security handler, the request that is sent to the STS can be used for either of the two things:

- To validate a security token in the WS-Security message header of the inbound message.

- To exchange a security token in the WS-Security message header for a security token that CICS can understand.

In a service requester pipeline, the request can be only one thing: to exchange a security token for a different type of token. The pipeline configuration file defines what type of token the STS issues to the security handler.

The security handler dynamically creates a pipeline to send the web service requests to the STS. This pipeline exists until a response is received from the STS after which it is deleted. If the request is successful, the STS returns an identity token or the status of the validity of the token. The security handler places the RACF ID that is derived from the token in the DFHWS-USERID container. If the STS encounters an error, it returns a SOAP fault to the security handler. The security handler then passes a fault back through the pipeline to the web service requester.

For more information about using an STS, see [Configuring the pipeline for Web Services Security](#).

### **Trust client interface**

The Trust client interface enables you to interact with a Security Token Service (STS) directly, rather than using the CICS security handler. Using an STS offers more advanced processing of tokens than the processing that the CICS security handler offers: for example, you can enable specific processing to handle many tokens in the inbound message headers or to exchange multiple types of token for outbound messages. Using this interface, you can create a custom message handler to send your own web service request to the STS.

You can use the Trust client interface without enabling the CICS security handler in your service provider and service requester pipelines, or you can use the Trust client interface in addition to the CICS security handler.

The Trust client interface is an enhancement to the CICS-supplied program DFHPIRT. This program is usually used to start a pipeline when a web service requester application is not deployed by using the CICS web services assistant. But it can also act as the Trust client interface to an STS.

You can invoke the Trust client interface by linking to DFHPIRT from a message handler or header processing program, passing a channel called DFHWSTC-V1 and a set of security containers, described in [Security containers](#). Using these containers, you request either a validate or issue action from the STS, select which token type to exchange, and pass the appropriate token from the message header. DFHPIRT dynamically creates a pipeline, composes a web service request from the security containers, and sends it to the STS.

DFHPIRT waits for the response from the STS and passes this back in the DFHWS-RESTOKEN container to the message handler. If the STS encounters an error, it returns a SOAP fault. DFHPIRT puts the fault in the DFHWS-STSFault container and returns to the linking program in the pipeline.

For more information about interacting directly with an STS, see [Invoking the Trust client from a message handler](#).

## **How it works: Authentication for CICS with SOAP message security**

WS-Security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. It does not require you to use a specific type of security token. Instead, it is extensible and supports multiple security token formats to accommodate various authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

WS-Security is applied to the message by inserting a SOAP security header. WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element `<wsse:Security>` is the container for security-related information. (*wsse* stands for web services security extension. *wsse* is an arbitrary prefix. Another prefix can be used, if it matches the namespace definition.)

When you use WS-Security for authentication, a security token is embedded in the SOAP header and is propagated from the message sender to the intended message receiver. On the receiving side, it is the responsibility of the server security handler to authenticate the security token and to set up the caller identity for the request.

The simplest form of security token is the username token, which is used to provide a username and (optional) password. A signed security token is one that is digitally signed by a specific authority. For example, an X.509 certificate is a signed security token. Security token usage for WS-Security is defined in separate profiles such as the username token profile and the X.509 token profile.

You can choose from the following authentication options for SOAP message authentication with CICS:

#### **Basic authentication (for web service provider only)**

In service provider mode, CICS can accept a UsernameToken in the SOAP message header for authentication on inbound SOAP messages. The UsernameToken contains a Username element and a Password element. CICS verifies the Username and Password with RACF. If this check is successful, CICS places the Username in container DFHWS-USERID and processes the SOAP message in the pipeline. If CICS cannot verify the UsernameToken, it returns a SOAP fault message to the service requester.

Username tokens that contain passwords are not supported on outbound SOAP messages when CICS is the service requester. A custom security handler can be written to store and send passwords. Any such handler must be responsible for securing the credentials that are used. This pattern is strongly discouraged because it is inherently insecure.

#### **X.509 certificate authentication**

An X.509 certificate that is used for signing SOAP messages can also be used for authentication. In service provider mode, CICS maps the certificate to a RACF user ID and places the user ID in the container DFHWS-USERID. In service requester mode, CICS can send an X.509 certificate in the SOAP message header to the service provider that is then used for authentication purposes. The certificate is identified with an element in the pipeline configuration file.

#### **Trusted authentication (identity assertion)**

In service provider pipelines, CICS can accept a UsernameToken or X.509 certificate in the SOAP message header as trusted. This type of security token typically contains a username and password, but in this case, the password is not required. When an X.509 certificate is used, the certificate is mapped to a RACF user ID. CICS trusts the provided identity and places the user ID in container DFHWS-USERID.

This option is a form of identity assertion in which an already-authenticated user identity is forwarded to CICS, and CICS accepts the identity without having to re-authenticate.

In service requester pipelines, CICS can send a username token without the password in the SOAP message header to the service provider. The user ID placed in the identity token is the contents of the DFHWS-USERID container. CICS can also send an X.509 certificate. The certificate is identified with an element in the pipeline configuration file.

#### **ICRX-based identity propagation (for web service provider only)**

An ICRX identity token is a z/OS identifier that binds a distributed ID to a (potentially shared) SAF user ID. User activities can be audited and tracked by using the identity by which they are known outside of z/OS, but authorized by using their SAF identity. In service provider mode, you can implement identity propagation with IBM DataPower Gateway acting as an intermediate server between web service requester applications and the CICS web service provider. The web service requester applications authenticate with IBM DataPower Gateway. IBM DataPower Gateway then creates a z/OS ICRX (*Extended Identity Context Reference*) identity token and forwards the token to CICS in a WS-Security header over the trusted TLS connection. The ICRX token allows CICS to map the distributed user ID to a RACF user ID. CICS resolves the ICRX identity token to a user ID and places a copy in the DFHWS-ICRX container. CICS also populates the DFHWS-USERID container.

ICRX-based identity propagation allows user access to CICS resources to be audited and tracked through the distributed identity, while using the mapped RACF user ID for resource authorization at the same time.

For more information about using an ICRX identity token, see [Configuring provider mode web services for identity propagation](#).

### Advanced authentication (Security Token Service)

In web service provider and requester pipelines, you can verify or exchange security tokens with a Security Token Service (STS) for authentication purposes. The STS enables CICS to accept and send messages that have security tokens in the message header that are not directly supported by the CICS security handler; for example, LTPA and Kerberos tokens or SAML assertions. For more information, see [Security Token Service](#).

You either configure the CICS security handler to define how CICS interacts with an STS or write your own message handler to use the Trust client interface. Whichever method you select, use TLS to secure the connection between CICS and the STS.

For an inbound message, you can verify or exchange a security token. If the request is to exchange the security token, CICS must receive a username token back from the STS. For an outbound message, you can only exchange a username token for a security token.

## How it works: Signing SOAP messages

Integrity is applied to a SOAP message to ensure that no one illegally modifies the message while it is in transit. Essentially, integrity is provided by generating an XML digital signature on the contents of the SOAP message. If the message data changes illegally, the signature would no longer be valid.

Generating a digital signature involves encrypting a message digest with a private key to create the electronic equivalent of a handwritten signature. You can use a digital signature to verify the identity of the signer and to ensure that nothing altered the SOAP message since it was signed.

For inbound messages, CICS supports digital signatures on elements in the SOAP body and on SOAP header blocks. For outbound messages, CICS signs all elements in the SOAP body.

The [WS-Security specification](#) allows the contents of the <Header> and the <Body> to be signed at the element level. That is, in a message, individual elements can be signed or not, or can be signed with different signatures or by using different algorithms. For example, in a SOAP message that is used in an online purchasing application, it is appropriate to sign elements that confirm receipt of an order because these elements might have legal status. However, to avoid the overhead of signing the entire message, other information might safely remain unsigned.

For inbound messages, the CICS security message handler can verify the digital signature on individual elements in the SOAP <Header> and the <Body>. The CICS security handler verifies the following elements:

- Signed elements that it encounters in the <Header>
- Signed elements in the SOAP <Body>. If the handler is configured to expect a signed body, CICS rejects with a fault any SOAP message in which the body is not signed.

For outbound messages, the security message handler can sign the SOAP <Body> only; it does not sign the <Header>. The security handler's configuration information specifies the algorithm and key that is used to sign the body.

To use CICS XML digital signatures, z/OS Integrated Cryptographic Service Facility (ICSF) must be started and configured with cryptographic devices. For more information, see [z/OS Integrated Cryptographic Service Facility \(ICSF\)](#).

**Note:** The performance overhead of XML signatures is significant.

### Signature algorithms

CICS supports the signature algorithms that are required by the XML Signature specification. Each algorithm is identified by a universal resource identifier (URI).

Algorithm	URI
Digital Signature Algorithm with Secure Hash Algorithm 1 (DSA)	<a href="http://www.w3.org/2000/09/xmldsig#dsa-sha1">http://www.w3.org/2000/09/xmldsig#dsa-sha1</a>

Algorithm	URI
with SHA1) Supported on inbound SOAP messages only.	
Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1)	http://www.w3.org/2000/09/xmldsig#rsa-sha1

## Example of a signed SOAP message

This example shows a SOAP message that is signed by CICS.

```
<?xml version="1.0" encoding="UTF8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      xmlns:xenc="http://www.w3.org/2001/04/xmenc#" SOAP-ENV:mustUnderstand="1">
      <wsse:BinarySecurityToken 1
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
        wsu:Id="x509cert00">MIIChDCC Ae2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHbGJEMMAoGA1UEChMD
SUJNMRRMEQYDVQQDEwpaWxsIF1hdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
MDELMAkGA1UEBhMCRC0xODAKBgNVBAoTA01CTTETMBEGA1UEAxMKV21sbCBZbYXRlczCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYKcGyEArSj/n+3RN75+jaxuOMBWShvZCB0egv8qu2UwLWEeioePsR
6Ku4SuHbBwJtWNr0xATAAS91Ea70yhVdppx0nJB0CiERg7S0HudP7a8JXPfzA+BqV63JqRgJyxN6
msfTAvEMR07LIXmZAt62nwcFrvCKNPFIJ5mkaJ9v1p7jkCAwEAa0BrTCBqJA/BglghkgBhvhC
AQ0EMhMwR2VuZJhdGVkIGJ5IHRoZSB0ZWN1cm10eSB0ZlZJ2ZXIgzM9yIHovT1MgKFJBQ0YpMDGg
ZQVRFU0BVSY5JQk0uQ09ggdJQk0uQ09NhgtXV1cuSUJNLKNPTYcECRR1BjAO
      </wsse:BinarySecurityToken>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          xmlns:xenc="http://www.w3.org/2001/04/xmenc#">
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <c14n:InclusiveNamespaces xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="ds wsu xenc SOAP-ENV "/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#TheBody">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <c14n:InclusiveNamespaces xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="wsu SOAP-ENV "/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/> 2
            <ds:DigestValue>Q0RZEa+gpaf1uShspHxhrrjaFlXE=</ds:DigestValue> 3
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>drDH0XESiyN6YJm27mfK1ZMG4Q4IsZqQ9N9V6kEnw2lk7aM3if77XNFnyKS4deg1bC3ga11kkaFJ 4
          p4jL0mYRqqycDPpQpm+UEu7mzfHRQGe7H0EnFqZpikNqZK5FF6fvY1v2JgTDPwr0SYXmhzwegUDT
          1TVj0vuUgXYrFya03pw=</ds:SignatureValue>
      </ds:Signature>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#x509cert00"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    wsu:Id="TheBody">
    <getVersion xmlns="http://msgsec.wssecvt.ws.ibm.com"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**1** The binary security token contains the base64binary encoding of the X.509 certificate. This encoding includes the public key that the intended recipient of the SOAP message uses to verify the signature.

**2** This is the algorithm that is used during the hashing process to produce the message digest.

**3** This is the value of the message digest.

**4** The digest value is then encrypted with the user's private key and included here as the signature value.



**5** This references the binary security token that contains the public key that is used to verify the signature.

## How it works: SOAP message encryption

XML encryption is applied to a SOAP message to ensure that only the intended recipient of the message can read the message.

The sender uses a secret key (also known as a *symmetric key*) to encrypt the message, and the receiver uses the same key to decrypt the message.

For inbound messages, CICS can decrypt any encrypted elements in the SOAP body, and encrypted SOAP header blocks where the body is also encrypted. For outbound messages, CICS encrypts the entire SOAP body.

The WS-Security specification allows some of the contents of the <Header> and all of the contents of the <Body> to be encrypted at the element level. That is, in a message, individual elements can have different levels of encryption, or can be encrypted by using different algorithms. For example, in a message that is used in an online purchasing application, it is appropriate to encrypt an individual's credit card details to ensure that they remain confidential. However, to avoid the overhead of encrypting the entire message, some information might be safely encrypted by using a less secure (but faster) algorithm and other information might safely remain unencrypted.

CICS supports encryption of the SOAP message body by using a symmetric algorithm such as the Triple DES algorithm or the AES algorithm. A symmetric algorithm is where the same key is used to encrypt and decrypt the data. This key is known as a symmetric key. The symmetric key is included in the message and is itself encrypted using the intended recipient's public key with the asymmetric key encryption algorithm RSA 1.5. This method gives you increased security because the asymmetric algorithm is complex and it is difficult to decrypt the symmetric key. However, you obtain better performance because most of the SOAP message is encrypted with the symmetric algorithm, which is faster to decrypt.

For inbound messages, the CICS security message handler can decrypt individual elements in the SOAP <Body> and can decrypt elements in the SOAP <Header> if the SOAP body is also encrypted. The security message handler always decrypts the following elements:

- Elements that it encounters in the <Header> in the order that the elements are found.
- Elements in the SOAP <Body>.

If the handler is configured to expect an encrypted body (by using the <expect\_encrypted\_body> element), CICS rejects with a fault any SOAP message in which the body is not encrypted.

For outbound messages, the security message handler supports encryption of the contents of the SOAP <Body> only; it does not encrypt any elements in the <Header>. When the security message handler encrypts the <Body>, all elements in the body are encrypted with the same algorithm and use the same key. The algorithm and information about the key are specified in the handler's configuration information.

CICS supports the encryption algorithms that are required by the XML Encryption specification. Each algorithm is identified by a universal resource identifier (URI). For supported algorithms, see [Web Services Security: SOAP Message Security](#).

To use CICS XML encryption, z/OS Integrated Cryptographic Service Facility (ICSF) must be started and configured with cryptographic devices. For more information, see [z/OS Integrated Cryptographic Service Facility \(ICSF\)](#).

**Note:** The performance overhead of XML encryption is significant.

## Encryption algorithms

CICS supports the encryption algorithms that are required by the XML Encryption specification. Each algorithm is identified by a universal resource identifier (URI).





- 2 This is the algorithm that was used to encrypt the symmetric key.
- 3 This references the binary security token that contains the public key that is used to encrypt the symmetric key.
- 4 This is the encrypted symmetric key that was used to encrypt the message.
- 5 This is the encryption algorithm that was used to encrypt the message.
- 6 This is the encrypted message.

## Designing security for CICS web service providers

Two primary scenarios are possible for a CICS web service provider: a direct request to a CICS web service provider or a request through an intermediate server that uses a third-party authentication server. To secure each of these scenarios, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

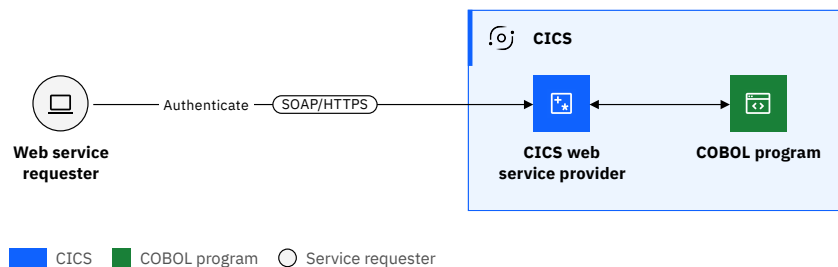
For the equivalent considerations for web service requester applications, see [Designing security for CICS web service requesters](#).

- “[Scenarios for CICS web service providers](#)” on page 350
- “[Security design considerations for CICS web service providers](#)” on page 351

For more information, see [Configuring security for CICS web services](#).

### Scenarios for CICS web service providers

In [Figure 138](#) on page 350, the web service requester sends a request directly to a CICS web service provider application.



*Figure 138. Direct request to CICS web service provider*

In [Figure 139](#) on page 351, the web service requester calls an intermediate server, which authenticates the request with a third-party authentication server, then passes the identity of the client to the CICS web service provider.

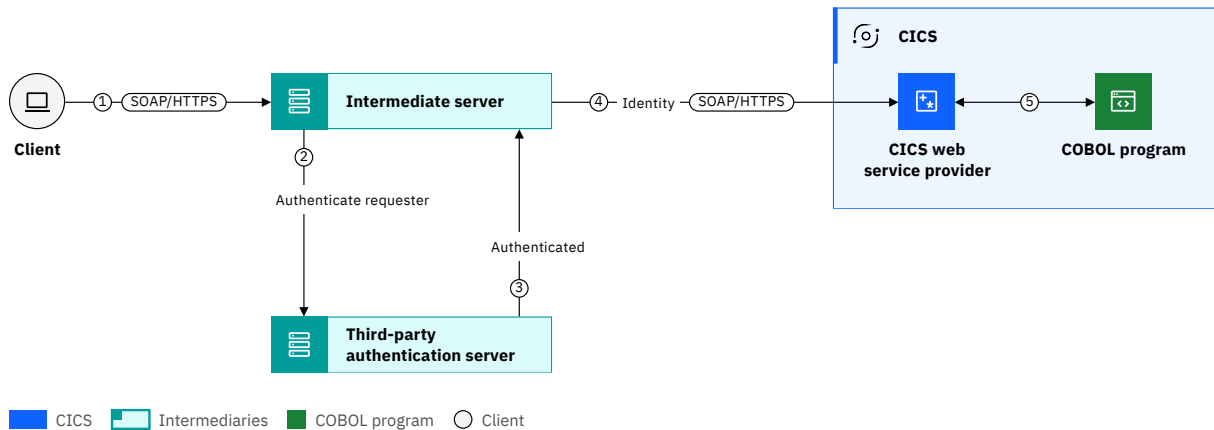


Figure 139. Request from an intermediate server

## Security design considerations for CICS web service providers

When you design security for CICS web service provider scenarios, consider the implications for:

- [Authentication and identification](#)
- [Authorization](#)
- [Confidentiality and integrity](#)
- [Trust](#)
- [Audit](#)

These considerations are explored further in this document.

## Authentication and identification considerations for web service provider security

### Who authenticates the web service requester? Is it CICS or an intermediate server?

CICS can authenticate a web service requester directly, or an intermediary can provide an authentication service to CICS. In this case, the intermediate server authenticates the client and then passes the identity of the client to CICS across a trusted connection. The client's identity can be passed to CICS as an [asserted user ID](#) or a [distributed identity](#).

### Will you use transport-based or SOAP message-based authentication?

Transport security is recommended for point-to-point requests where no intermediate server is involved. If you use transport-based authentication, you can configure HTTP basic authentication or TLS client authentication.

Use basic authentication only with HTTPS because the credentials are sent as cleartext and are vulnerable to packet sniffing. TLS client authentication is more secure than basic authentication because a client certificate is signed.

Use SOAP message-based authentication (WS-Security) when security credentials need to flow in the SOAP message through a number of intermediaries. If you use SOAP message-based security, you can configure the CICS security handler to authenticate with one of the following types of token:

- Username tokens that contain only a username.
- Username tokens that contain a username and password.
- X.509 digital certificate
- ICRX token.

If other token types are used, you can configure the CICS security handler to call a [Security Token Service \(STS\)](#) or you can write a custom security handler to do your own, customized security processing.

**Recommended:** Use TLS to secure the connection to the STS.

For more information about the types of authentication token that can be used with CICS web services, see [How it works: Authentication for CICS with SOAP message security](#).

In third-party authentication scenarios, SOAP-message authentication is often used along with transport-based authentication. The client's identity is flowed in a SOAP security header and the intermediate server is authenticated by using TLS client authentication.

#### **Do you need to implement different authentication mechanisms for different services or different types of client?**

Instead of having a general rule for the application, the type of authentication mechanism can be decided for specific services. It might be appropriate to run read-only services with a generic user ID, where more sensitive services might need the requester to authenticate. In CICS, this split can be made by running secured services on a different pipeline to unsecured services.

### **Authorization considerations for web service provider security**

#### **Will you configure the CICS task in the web service provider to run with the RACF user ID of the end user?**

Configuring the CICS task in this way enables fine-grained authorization and auditing.

- Where no intermediaries are used, CICS authorization processing can be based on the RACF user ID that is used to authenticate, or a RACF user ID that is associated with the security token that is used to authenticate.
- Where intermediaries are used, CICS authorization processing can be based on the asserted RACF user ID or the distributed identity that is passed by the intermediary. CICS can resolve a distributed identity that is contained in an ICRX identity token to a RACF user ID.

Use [Surrogate security](#) to ensure that the intermediary has the correct authority to start work on behalf of the asserted identity.

#### **Do you want to limit access to CICS resources to a specific, technical user ID?**

Instead of configuring the CICS task in the web service provider to run with the RACF user ID of the end users, you can specify a RACF user ID in a URIMAP for the pipeline alias transaction.

If the web service does not need to be secured, requests can be allowed to run under the CICS default user ID.

### **Confidentiality and integrity considerations for web service provider security**

#### **TLS, XML signatures and XML encryption, or non-encrypted?**

TLS can be used for integrity and confidentiality for point-to-point connections by using either the TLS support in CICS or AT-TLS. However, TLS does not ensure end-to-end message integrity or encryption because the message is not protected within intermediate servers, as illustrated in [Figure 139 on page 351](#).

XML signatures and XML encryption can be used to ensure integrity and confidentiality end-to-end, including within intermediate servers. However, this capability is rarely used because the performance overhead of XML signature processing and XML encryption in CICS is significant. As an alternative, these capabilities are often implemented in an intermediate server such as IBM DataPower® Gateway.

**Recommendation:** If you need XML signature processing or XML encryption, consider implementing these capabilities in an intermediate server, such as IBM DataPower Gateway, instead of directly in CICS.

If SOAP messages do not contain critical data, or if the messages are only transmitted within an internal secure network, then it is reasonable to flow requests over non-encrypted HTTP connections.

## **Trust considerations for web service provider security**

### **How do you ensure that the intermediate server is a trusted server?**

TLS client authentication can be used so that the intermediate server provides a client certificate that is validated by CICS. Successful client authentication requires that the certificate authority (CA) that signed the client certificate is considered trusted by CICS. To be considered trusted, the certificate of the CA must be in the CICS key ring.

### **How do you ensure that the authentication token is trusted?**

An authentication token, for example a SAML token, can be signed. The signature is used to verify the issuer of the SAML token and to ensure the integrity of the token.

## **Audit considerations for web service provider security**

### **Does the web service request need to be audited in CICS?**

Typically, you need to capture the RACF user ID that is used for running the CICS task. This information is recorded in the CICS SMF in type 110, subtype 01 records. When identity propagation is used, the distributed identity from the ICRX is also recorded.

### **Will you create an audit trail in CICS or in an intermediate server, or both?**

You might need an audit trail in all the servers that are involved in processing a request. IBM DataPower Gateway is an ideal place to capture audit information because it provides a configurable solution that supports multiple format log records that can be stored on the appliance or transferred off-device.

Review these design examples that offer different configurations.

## Design example: Securing a direct request with TLS client authentication

This scenario shows how TLS client authentication is used to secure a web service request to a CICS web service provider application. In this example, the web service requester application connects directly to CICS.

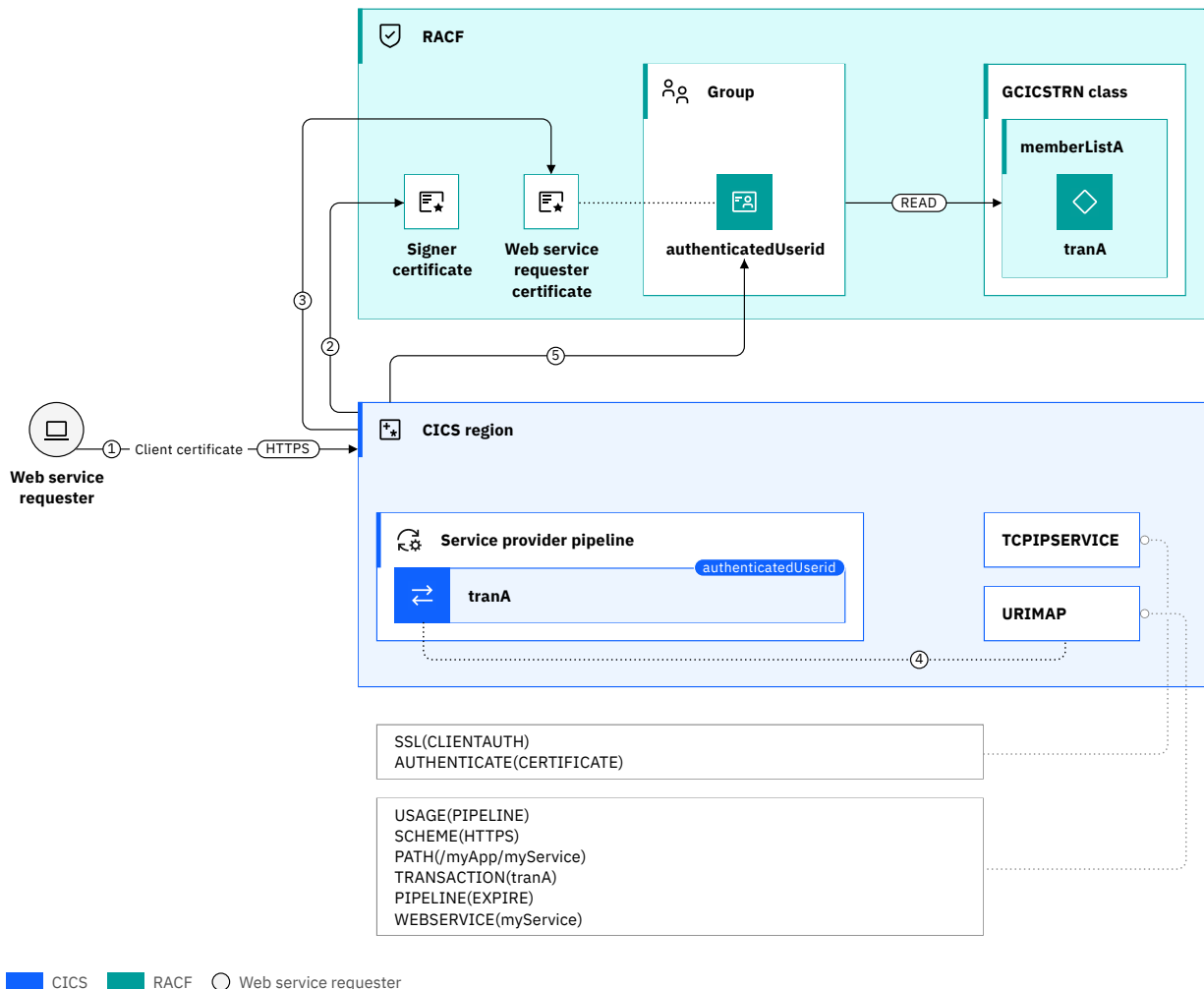


Figure 140. TLS authentication in a web service provider scenario

The TCPIPService resource in CICS specifies SSL(CLIENTAUTH), which requires the service requester to send a client certificate. It also specifies AUTHENTICATE(CERTIFICATE) so that CICS uses the client certificate to determine the authenticated user ID.

The URIMAP resource specifies the web service, pipeline, and transaction ID for the request. It also specifies SCHEME(HTTPS) to restrict access to the web service to HTTPS requests.

1. The web service requester application sends a SOAP/HTTPS request to CICS.
2. A TLS handshake occurs. CICS sends its server certificate and asks the service requester to provide its certificate. CICS validates the chain of trust by checking that the client certificate issuer (signer certificate) is in the RACF keyring.
3. CICS also maps the service requester certificate to a RACF user ID (the *authenticatedUserId*).
4. CICS finds the URIMAP that matches the request and determines the web service to start and the service provider pipeline to process the request. The URIMAP also determines the transaction ID *tranA* to run the request.

5. CICS queries RACF to verify that the *authenticatedUserid* is authorized to run transaction *tranA*. This validation is achieved by calling RACF to check that the *authenticatedUserid* belongs to a group that has READ authority to the member list (*memberListA*) containing *tranA*, in the GCICSTRN class.

When AT-TLS is used, the web attach transaction is not required in most situations. In this case, the pipeline alias transaction runs directly with the authenticated user ID.

## Configuration example: Securing a direct request with TLS client authentication

This task explains how to configure a CICS web service provider that supports authentication by using a TLS client certificate.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a direct request with TLS client authentication](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Implemented a CICS web service provider as described in [Developing SOAP web services](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to accept web service requests over TLS. The CICS region identifies itself to the client by using the region's TLS server certificate. The client identifies itself to CICS by using its TLS client certificate that maps to a RACF user ID. This user ID is used to run the CICS web service attach transaction. The user ID is also used to run the web service when started through the client that sends the web service request.

This task assumes the following definitions:

- *keyRingA* is the name of the RACF key ring that is associated with the CICS region.
- *clientCA* is the RACF key ring label for the client certificate's signing CA.
- *clientCACertificateDataSet* is the data set containing the client certificate's signing CA.
- *clientCertificateDataSet* is the data set containing the client certificate (if a specific client certificate is mapped to a RACF user ID).
- *certificateLabel* is the RACF label for the client certificate in RACF.
- *groupA* is the RDO group name.
- *tcpipService* is the name of a TCPIP SERVICE definition.
- *tcpipPort* is a TCPIP port that is used by CICS to listen for the web service requests.
- *webServicePipe* is the name of a PIPELINE definition.
- *webServiceProvider* is the name of the webServiceProvider program.
- *soapProviderXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.

- *matchID* an identifier for the security request recording (SRR) report.
- *regionUseridA* is the region user ID for the CICS region.
- *clientUserid* is the user ID associated with the client certificate (from the requester) that runs the web attach transaction.
- CN=myClient.host.com, O=IBM, C=US is the client certificate subject's distinguished name value.

## Procedure

1. Import the client's CA certificate into the RACF key ring.

First, export the client's CA certificate (public key) from their keystore and transfer it to a sequential data set on z/OS. Now import the certificate into the CICS region's key ring by using the following RACF commands:

```
RACDCERT ID(regionUseridA) ADD('clientCACertificateDataSet') WITHLABEL('clientCA') TRUST
```

2. Identify the client's CA certificate as a trusted (CERTAUTH) certificate in the RACF keyring:

```
RACDCERT ID(regionUseridA) CONNECT(RING(keyRingA) LABEL('clientCA') USAGE(CERTAUTH))
```

3. The client's certificate must be mapped to a RACF user ID. Client certificates can be associated with a RACF user ID in two ways. Each expected client certificate can be mapped to a RACF user ID individually, or groups of certificates that match a specified pattern can be mapped to a RACF user ID. This latter approach significantly reduces the administration burden of managing digital certificates and is the approach that is described here.

Use the RACDCERT MAP command to define a certificate name filter. This command maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters allowing multiple certificates to be associated with a single RACF user ID.

- a. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETOPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

**Important:** In this command, the syntax of the SDNFILTER is significant. Use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS® Security Server RACF Command Language Reference*.

- b. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(clientUserid) SDNFILTER('CN=myClient.host.com.O=IBM.C=US')  
WITHLABEL('certificateLabel')
```

- c. Refresh the DIGTNMAP RACF class for the changes to take effect.

Enter the following RACF command:

```
SETOPTS RACLIST(DIGTNMAP) REFRESH
```

When this certificate is presented to the CICS region as part of a TLS handshake, the CICS region runs the subsequent transaction under the *clientUserid*, which is associated with the certificate in RACF.

4. Define the TCPIP SERVICE.



A TCIPSERVICE is needed to listen for connections from the client. The connection uses TLS and requires the client to authenticate with a TLS certificate that is mapped to a user ID in RACF.

```
DEFINE TCIPSERVICE(tcpipService) GROUP(groupA)
PORT(tcpipPort)
SSL(CLIENTAUTH) AUTHENTICATE(CERTIFICATE)
URM(NONE) PROTOCOL(HTTP) TRANSACTION(CWXN)
```

#### 5. Define the PIPELINE.

The PIPELINE provides information about the message handler programs that act on a service request and on the response.

```
DEFINE PIPELINE(webServicePipe) GROUP(groupA)
CONFIGFILE(soapProviderXmlFile) SHELF(shelfDir)
WSDIR(wsProviderDir)
```

An example CONFIGFILE can be found in <cics-install-dir>/samples/pipelines/basicsoap11provider.xml.

#### 6. Authorize *clientUserid* to the pipeline alias transaction CPIH and subsequent processing.

The following commands associate the *clientUserid* with the *taskUserGroup* and grant the group READ access to the PIPEUSER member list that contains CPIA, CPIH, CPIL, and CPIQ:

```
RDEFINE GCICSTRN PIPEUSER ADDMEM(CPIA,CPIH,CPIL,CPIQ)
UACC(NONE)
PERMIT GCICSTRN PIPEUSER ID(taskUserGroup) ACCESS(READ)
CONNECT clientUserid GROUP(taskUserGroup)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

If the PIPEUSER member list is already defined and associated with a user group, then you need only to add *clientUserid* to that user group.

#### 7. Install the definitions in group *groupA*. Installing the PIPELINE creates URIMAP and WEBSERVICE resources for the web service.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHS00107 02/07/2022 14:27:00 IYK2ZGV1 TCIPSERVICE tcpipService has been opened on port
tcpipPort at IP address ANY.
DFHPI0701 I 02/07/2022 14:27:00 IYK2ZGV1 CICSUSER PIPELINE webServicePipe has been
created.
DFHPI0204 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe is now ENABLED and
is ready for use.
DFHPI0703 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe is about to scan
the WSDIR directory.
DFHPI0901 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA New WEBSERVICE webServiceProvider is being
created during a scan against PIPELINE webServicePipe.
DFHPI0910 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA WEBSERVICE webServiceProvider within
PIPELINE webServicePipe has been created.
DFHPI0915 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA WEBSERVICE webServiceProvider is now
INSERVICE and is ready for use.
DFHPI0903 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA New URIMAP £055300 is being created during
a scan against PIPELINE webServicePipe for
WEBSERVICE webServiceProvider.
DFHWB1560 02/07/2022 13:47:00 IYK2ZGV1 regionUseridA URIMAP £055300 has been created.
DFHPI0704 I 02/07/2022 14:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The TCIPSERVICE must have an OPENSTATUS of OPEN.
- The PIPELINE must have an ENABLESTATUS of ENABLED.
- The WEBSERVICE must have a STATE of INSERVICE.
- You must have a generated URIMAP with an ENABLESTATUS of ENABLED.

To validate the security environment is functioning correctly when you run this example, you can use the CICS security request recording (SRR) utility. The SRR logs all authorization calls and the matching response codes.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **Web** tab and set the **TCPIP Service** field to *tcpipService*. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

The web service client can now start the CICS web service provider.

## Design example: Asserting an identity to the CICS web service provider

In this scenario, a client authenticates with an intermediate server, which then asserts the client's identity to the CICS web service provider.

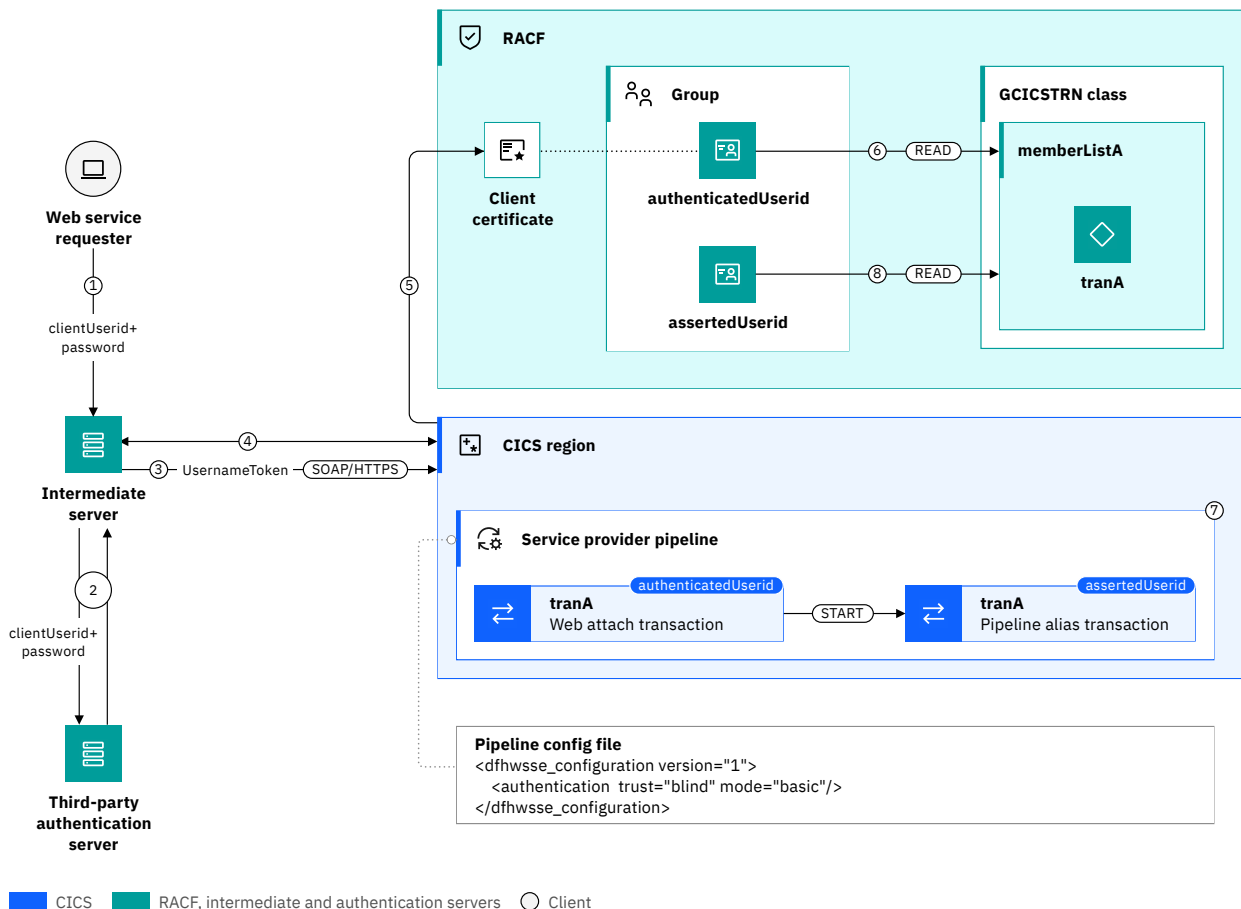


Figure 141. Identity assertion in a web service provider scenario

1. The client sends a request to the web service requester application.
2. The web service requester application authenticates the client with a third-party authentication server.
3. The web service requester application sends a SOAP/HTTPS request to CICS containing a *UsernameToken* in a SOAP header. The *UsernameToken* contains the RACF ID of the client. Note that this scenario requires that the client authenticates with a RACF ID, or the third-party authentication server maps the client's distributed identity to a RACF user ID.

4. CICS responds with its TLS server certificate and asks for the client's certificate. The web service requester application then validates the CICS server certificate and sends its TLS client certificate. The request is sent over HTTPS. Therefore, encryption and integrity are enabled at the transport level.
5. CICS validates the client certificate and maps the client certificate to a RACF user ID (*authenticatedUserid*).
6. The web attach transaction runs with the user ID mapped from TLS client certificate (*authenticatedUserid*).
7. The web service provider pipeline includes the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. CICS puts the asserted user ID in the DFHWS-USERID container.
8. The pipeline alias transaction runs under the asserted user ID (*assertedUserid*) and all further resource security checks for this request (for example, program and database security) are then performed against the asserted user ID.

Use TLS client authentication to establish trust for an identity assertion scenario like the one described here. Extra trust can be configured by using [surrogate security](#), where the user ID mapped from TLS client certificate must have the correct authority to start work on behalf of the asserted identity.

## Configuration example: Asserting an identity to the CICS web service provider

This task explains how to configure a CICS web service provider that supports authentication by using a TLS client certificate, and then identity assertion by using blind trust.

### Before you begin

This configuration task is based on the example security scenario [Configuration example: Asserting an identity to the CICS web service provider](#). You must be familiar with web services and TLS.

Before you begin this task, you must complete these tasks:

- Complete the task [Configuring CICS to use SSL](#).
- Implement a CICS web service provider as described in [Developing SOAP web services](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to accept web service requests over TLS. The CICS region identifies itself to the client by using the region's TLS server certificate. The client identifies itself to CICS by using its TLS client certificate. The certificate maps to a RACF user ID and this user ID is used to run the CICS web service attach transaction. The client then sends the web service request that asserts the user ID that is used to run the requested web service.

This task assumes the following definitions:

- *keyRingA* is the name of the RACF key ring that is associated with the CICS region.
- *clientCA* is the RACF key ring label for the client certificate's signing CA.
- *clientCACertificateDataSet* is the data set containing the client certificate's signing CA.
- *clientCertificateDataSet* is the data set containing the client certificate (if a specific client certificate is mapped to a RACF user ID).
- CN=myClient.host.com, O=IBM, C=US is the client certificate subject's distinguished name value.
- *certificateLabel* is the RACF label for the client certificate in RACF.

- *groupA* is the RDO group name.
- *tcpipService* is the name of a TCPIPSERVICE definition.
- *tcpipPort* is a TCPIP port that is used by CICS to listen for the web service requests.
- *webServicePipe* is the name of a PIPELINE definition.
- *webServiceProvider* is the name of the webServiceProvider program.
- *soapProviderXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *uriMapA* is the generated name of the URIMAP that is created by CICS when the PIPELINE is installed.
- *matchID* an identifier for the security request recording (SRR) report

## Procedure

1. Import the client's CA certificate into the RACF key ring.

First, export the client's CA certificate (public key) from their keystore and transfer it to a sequential data set on z/OS. Now import the certificate into the CICS region's key ring by using the following RACF commands:

```
RACDCERT ID(regionUseridA) ADD('clientCACertificateDataSet') WITHLABEL('clientCA') TRUST
```

2. Identify the client's CA certificate as a trusted (CERTAUTH) certificate in the RACF keyring:

```
RACDCERT ID(regionUseridA) CONNECT(RING(keyRingA) LABEL('clientCA') USAGE(CERTAUTH))
```

3. The client's certificate must be mapped to a RACF user ID. Client certificates can be associated with a RACF user ID in two ways. Each expected client certificate can be mapped to a RACF user ID individually, or groups of certificates that match a specified pattern can be mapped to a RACF user ID. This latter approach significantly reduces the administration burden of managing digital certificates and is the approach that is described here.

Use the RACDCERT MAP command to define a certificate name filter. This process maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters that allow multiple certificates to be associated with a single RACF user ID.

- a. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETOPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

- b. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(clientUserid) SDNFILTER('CN=myClient.host.com.O=IBM.C=US')  
WITHLABEL('certificateLabel')
```

**Important:** In this command, the syntax of the SDNFILTER is significant. Use periods to separate the components of the distinguished name and remove any spaces between DN components.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS® Security Server RACF Command Language Reference*.

- c. Refresh the DIGTNMAP RACF class for the changes to take effect.

Enter the following RACF command:

```
SETOPTS RACLIST(DIGTNMAP) REFRESH
```

When this certificate is presented to the CICS region as part of a TLS handshake, the CICS region runs the subsequent transaction under the *clientUserid*, which is associated with the certificate in RACF.

#### 4. Define the TCPIP SERVICE.

A TCPIP SERVICE is needed to listen for connections from the client. The connection uses TLS and requires the client to authenticate with a TLS certificate that is mapped to a user ID in RACF.

```
DEFINE TCPIP SERVICE(tcpipService) GROUP(groupA)
PORT(tcpipPort)
SSL(CLIENTAUTH) AUTHENTICATE(CERTIFICATE)
URM(NONE) PROTOCOL(HTTP) TRANSACTION(CWXN)
```

#### 5. Define the PIPELINE.

The PIPELINE provides information about the message handler programs that act on a service request and on the response.

```
DEFINE PIPELINE(webServicePipe) GROUP(groupA)
CONFIGFILE(soapProviderXmlFile) SHELF(shelfDir)
WSDIR(wsProviderDir)
```

The CONFIGFILE specifies the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. An example is:

```
<provider_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="blind" mode="basic"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.2_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

This sample must be copied into a zfs named *soapProviderXmlFile*.

#### 6. Authorize *assertedUserid* to the pipeline alias transaction CPIH and subsequent processing:

The following commands associate the *assertedUserid* with the *taskUserGroup* and grant the group READ access to the PIPEUSER member list that contains CPIA, CPIH, CPIL, and CPIQ:

```
RDEFINE GCICSTRN PIPEUSER ADDMEM(CPIA,CPIH,CPIL,CPIQ)
UACC(NONE)
PERMIT GCICSTRN PIPEUSER ID(taskUserGroup) ACCESS(READ)
CONNECT assertedUserid GROUP(taskUserGroup)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

If the PIPEUSER profile is already defined and associated with a user group, then you need only to add *assertedUserid* to that group.

#### 7. Install the definitions in group *groupA*. Installing the PIPELINE creates URIMAP and WEBSERVICE resources for the web service.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and URIMAP resources:

```
DFHS00107 02/07/2022 13:27:00 IYK2ZGV1 TCPIP SERVICE tcpipService has been opened on port
tcpipPort at IP address ANY.
DFHPI0701 I 02/07/2022 13:27:00 IYK2ZGV1 CICSUSER PIPELINE webServicePipe has been
created.
DFHPI0204 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe is now ENABLED and
is ready for use.
DFHPI0703 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe is about to scan
```

the WSDIR directory.  
 DFHPI0901 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA New WEBSERVICE webServiceProvider is being created during a scan against PIPELINE webServicePipe.  
 DFHPI0910 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA WEBSERVICE webServiceProvider within PIPELINE webServicePipe has been created.  
 DFHPI0915 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA WEBSERVICE webServiceProvider is now INSERVICE and is ready for use.  
 DFHPI0903 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA New URIMAP uriMapA is being created during a scan against PIPELINE webServicePipe for WEBSERVICE webServiceProvider.  
 DFHPI1560 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA URIMAP uriMapA has been created.  
 DFHPI0704 I 02/07/2022 13:27:00 IYK2ZGV1 regionUseridA PIPELINE webServicePipe Implicit scan has completed. Number of wsbnd files found in the WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of failed WEBSERVICE creates: 000000.

If you inquire on the created CICS resources:

- The TCPIP SERVICE has an OPENSTATUS of OPEN.
- The PIPELINE has an ENABLESTATUS of ENABLED.
- The WEBSERVICE has a STATE of INSERVICE.
- You have a generated URIMAP with an ENABLESTATUS of ENABLED.

To validate the security environment is functioning correctly when you run this example, you can use the CICS security request recording (SRR) utility. The SRR logs all authorization calls and the matching response codes.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **Web** tab and set the **TCPIP Service** field to *tcpipService*. For more information, see [Checking that a CICS security configuration example is working by using the SRR](#).

The web service client can now start the CICS web service provider.

## Designing security for CICS web service requesters

---

Typically, a CICS web service requester sends a request to the web service provider. To secure this scenario, consider the implications for different aspects of security and decide which options are the best for you. Examples illustrate some recommended options.

For the equivalent considerations for web service requester applications, see [Designing security for CICS web service providers](#).

- [“Scenario for CICS web service requesters” on page 362](#)
- [“Security design considerations for CICS web service requesters” on page 363](#)

For more information about configuring security for a CICS web service requester, see [Configuring security for CICS web services](#).

### Scenario for CICS web service requesters

In [Figure 142 on page 363](#), the CICS web service requester sends a request to a web service provider.

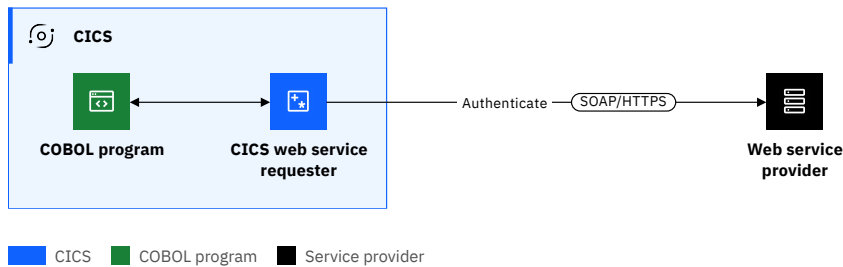


Figure 142. Direct request to the web service provider

## Security design considerations for CICS web service requesters

When you design security for CICS web service requester scenarios, consider the implications for:

- [Authentication and identification](#)
- [Confidentiality and integrity](#)

These considerations are explored below.

## Authentication and identification considerations for web service requester security

### Will you use transport-based or SOAP message-based authentication?

You can use transport-based security or SOAP message-based authentication, depending on how the web service provider application expects to receive authentication credentials.

Transport-based authentication can be implemented by using HTTP basic authentication or TLS client authentication.

The XWBAUTH global user exit is used to specify the credentials to send. See [HTTP client send exit XWBAUTH](#).

Web HTTPS is used to connect to a web service provider. The web service provider can require CICS to provide a client certificate. See [Design example: Securing a web service request with TLS client authentication](#).

Use SOAP message-based authentication (WS-Security) when the web service provider expects to receive an authentication token in a SOAP header.

- If you choose SOAP message-based security, you can configure the CICS security handler to include one of the following authentication tokens in the security header of outbound SOAP message:
  - Username tokens that contain only a username.
  - X.509 digital certificate

If you use other token types, you can configure the CICS security handler to call a Security Token Service (STS). Alternatively you can write your own message handler to use the Trust client interface.

**Recommended:** Use TLS to secure the connection to the STS.

- In some scenarios, SOAP message authentication can be used along with transport-based authentication, for example, the RACF user ID of the CICS task can be flowed in a SOAP security header and the CICS region can be authenticated to the web service provider by using TLS client authentication. See [Design example: Securing a web service request with identity assertion](#).

## Confidentiality and integrity considerations for web service requester security

### TLS, XML signatures and XML encryption, or non-encrypted?

TLS can be used for integrity and confidentiality for point-to-point connections where no intermediate server is involved, but it does not ensure end-to-end message integrity or encryption because the message is not protected within intermediate servers.

XML signatures and XML encryption can be used to ensure integrity and confidentiality end-to-end, including within intermediate servers. However, this capability is rarely used because the performance overhead of XML signature processing and XML encryption in CICS is significant. As an alternative, these capabilities are often implemented in an intermediate server such as IBM DataPower Gateway.

**Recommendation:** If you need XML signature processing or XML encryption, consider implementing these capabilities in an intermediate server, such as IBM DataPower Gateway, instead of directly in CICS.

Shown here are some design examples to demonstrate different configurations.

### Design example: Securing a web service request with TLS client authentication

This scenario shows how TLS client authentication is used to secure a web service request from CICS to a web service provider application.

For instructions about how to configure for this scenario, see [Configuration example: Securing a web service request with TLS client authentication](#).

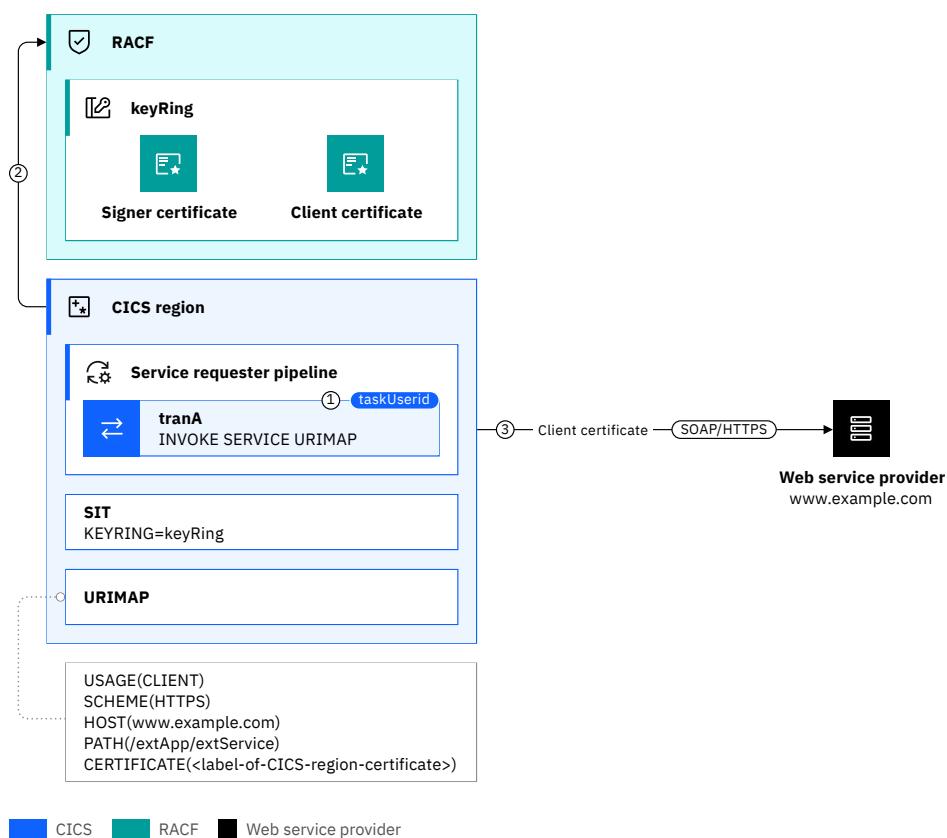


Figure 143. TLS authentication in a web service requester scenario



The URIMAP resource specifies the hostname and path of the external web service. It also specifies SCHEME (HTTPS) so that an HTTPS request is sent, and the label of client certificate that is used in the TLS handshake.

1. The CICS program uses the **EXEC CICS INVOKE SERVICE URIMAP()** command to call the external web service provider.
2. A TLS handshake is performed. The external web service provider sends its server certificate and asks CICS to provide its certificate. CICS validates the chain of trust by checking that the server certificate issuer (signer certificate) is in the RACF key ring (*keyRing*).
3. CICS sends its client certificate that is specified in the URIMAP.

**Recommended:** Use the CIPHERS attribute of the **URIMAP** to define the ciphers that can be used for the SOAP/HTTPS request.

For more information about configuring TLS client authentication for a CICS web service requester, see [CICS as an HTTP client: authentication and identification](#).

## Configuration example: Securing a web service request with TLS client authentication

Learn how to configure a CICS web service requester that authenticates with a web service provider by using a TLS client certificate.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web service request with TLS client authentication](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Set up a CICS web service requester that is implemented as described in [Developing SOAP web services](#).
- Set up a web service provider that is implemented to be started by the CICS web service requester. For example, a CICS web service provider as described in [Design example: Securing a direct request with TLS client authentication](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to issue web service requests over TLS. The CICS region validates the TLS server certificate that is presented by the started server. The web service requester identifies itself to the provider by using a TLS client certificate.

This task assumes the following definitions:

- *clientCertA* is the TLS certificate to be presented to the web service provider to identify the client
- *datasetName* is the data set that *clientCertA* is exported into to allow it to be imported into the web service provider's key store.
- *webserviceA* is the name of the WEBSERVICE resource that is generated from the WSBIND file by CICS when the pipeline is installed.
- *urimapA* is the name of the URIMAP used to specify the URI of the target web service.

- *channelNameA* is the name of the channel that is used to pass the containers that hold the data that is mapped by the application data structure.
- *operationA* is the name of the operation that is to be started as defined in the WSDL file.
- *groupA* is the RDO group name.
- *pipelineA* is the name of a PIPELINE definition.
- *webserviceURIHost* is the hostname for the web service to be requested.
- *webserviceURIPathPort* is the port for the web service to be requested.
- *webserviceURIPath* is the path part of the URI for the web service to be requested.
- *soapRequesterXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *matchID* an identifier for the Security Request Recording (SRR) report
- *txnA* the transaction that starts the web service request.
- *regionUseridA* is the region user ID for the CICS region.
- *taskUserID* is the user that triggers the web service request.

## Procedure

1. Export certificate *clientCertA* to *dataset* and transfer it to the platform on which the web service provider is implemented.

```
RACDCERT EXPORT (LABEL('clientCertA')) DSN('datasetName') -
FORMAT(CERTDER)
```

2. Import the client certificate into the web service provider so it can be used to identify the web service requester. How this import is performed depends on the platform on which the web service provider is implemented.
3. Ensure that the WSBIND file in the *wsProviderDir* directory specifies the correct URI for the web service to be called. As the call is made over a TLS encrypted connection, the URI should start with https (rather than http). This configuration might require the source WSDL file being updated and the tools rerun to generate the updated WSBind file. Options for generating the WSDL file can be found in [Developing SOAP web services](#).
4. Ensure that the code that starts the web service specifies the URIMAP to be used, for example:

```
EXEC CICS INVOKE SERVICE(webserviceA)
      URIMAP(urimapA)
      CHANNEL(channelNameA)
      OPERATION(operationA)
      RESP(resp) RESP2(resp2)
```

5. Update the targeted web service provider to require a TLS connection with the requester that identifies itself with a client certificate.
6. Define the pipeline.

The PIPELINE provides information about the message handler programs that act on an outgoing service request and on the response.

```
DEFINE PIPELINE(pipelineA) GROUP(groupA)
      CONFIGFILE(soapRequesterXmlFile) SHELF(shelfDir)
      WSDIR(wsProviderDir)
```

An example CONFIGFILE can be found in <cics-install-dir>/samples/pipelines/basicsoap11requester.xml.

This configuration is not adding any security to this scenario. The security in this scenario is provided by the encryption of the communications by using TLS and the checking of the TLS certificates at either end of the connection.

#### 7. Define the URIMAP.

The URIMAP matches the URI of the web service request and provides information on how to process the requests. For service requesters, CICS does not create any URIMAP resources automatically when the PIPELINE resource is installed or as a result of a PERFORM PIPELINE SCAN command.

```
DEFINE URIMAP(urimapA) GROUP(groupA)
  USAGE(Client) SCHEME(HTTPS)
  PORT(webbserviceURIPathPort) HOST(webbserviceURIHost)
  PATH(webbserviceURIPath)
  CERTIFICATE(clientCertA)
  AUTHENTICATE(No)
```

#### 8. Install the definitions in group *groupA*.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHWP1560 03/25/2022 21:21:18 IYK2ZGV2 regionUserida URIMAP urimapA has been
created.
DFHPI0701 I 03/25/2022 21:21:18 APPLID regionUserida PIPELINE pipelineA has been
created.
DFHPI0204 I 03/25/2022 21:21:18 APPLID regionUserida PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 03/25/2022 21:21:18 APPLID regionUserida PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 03/25/2022 21:21:18 APPLID regionUserida New WEBSERVICE webbserviceRequesterA is being
created during a scan against
pipelineA.

DFHPI0910 I 03/25/2022 21:21:18 APPLID regionUserida WEBSERVICE webbserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 03/25/2022 21:21:18 APPLID regionUserida WEBSERVICE webbserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 03/25/2022 21:21:18 APPLID regionUserida PIPELINE pipelineA Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The PIPELINE should have an ENABLESTATUS of ENABLED.
- The URIMAP should have an ENABLESTATUS of ENABLED.
- The WEBSERVICE should have a STATE of INSERVICE.

To validate the security environment is functioning correctly, you need a web service requester that is configured to authenticate by using a TLS client certificate. This web service needs to start an appropriate web service provider.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** context menu option. On that dialog, select the **All** tab and set the **Transaction ID** *tranA*. See [Checking that a CICS security configuration example is working by using the SRR](#) for details.

The web service requester can now start the CICS web service provider.

## Design example: Securing a web service request with identity assertion

This scenario shows how you can use identity assertion to secure a web service request from CICS to a web service provider application.

For instructions about how to configure for this scenario, see the [Configuration example: Securing a web service request with identity assertion](#).

Figure 144 on page 368 shows a scenario where CICS asserts the user ID of the running CICS task to the web service provider. TLS client authentication is used in this scenario to establish trust between CICS and the external web service provider.

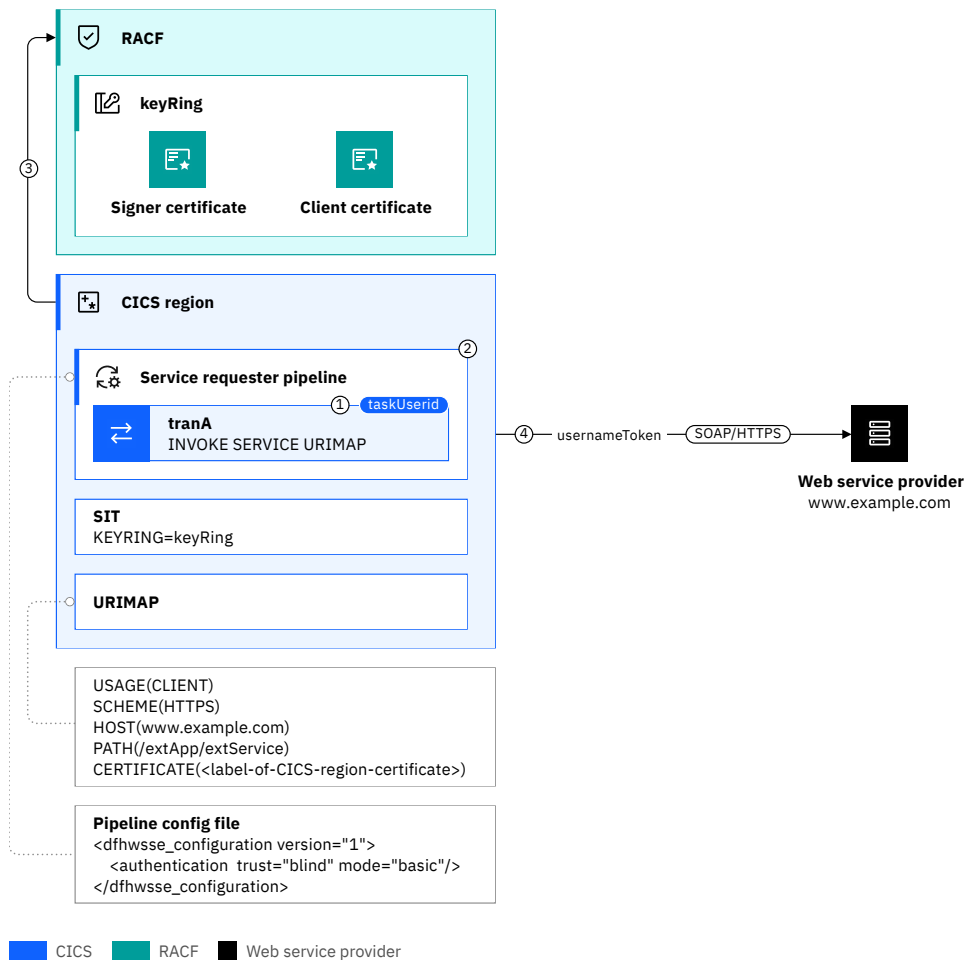


Figure 144. Identity assertion in a web service requester scenario

The web service requester pipeline includes the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic.

The URIMAP resource specifies the hostname and path of the external web service. It also specifies SCHEME (HTTPS) so that an HTTPS request is sent, and the label of client certificate that is used in the TLS handshake.

In Figure 144 on page 368:

1. The CICS program uses the **EXEC CICS INVOKE SERVICE URIMAP()** command to call the external web service provider.
2. The service requester pipeline adds an identity token (*usernameToken*) to the SOAP message. The user ID placed in the identity token is the contents of the DFHWS-USERID container, which by default contains the running task's user ID.
3. A TLS handshake is performed. The web service provider sends its server certificate and asks CICS to provide its client certificate. CICS validates the chain of trust by checking that the server certificate issuer (signer certificate) is in the RACF key ring (*keyRing*).
4. The SOAP request with a *usernameToken* containing the task user ID is sent over HTTPS to the web service provider.

## Configuration example: Securing a web service request with identity assertion

Learn how to configure a CICS web service requester that asserts an identity on the request to a web service provider.

### Before you begin

This configuration task is based on the example security scenario [Design example: Securing a web service request with identity assertion](#).

You must be familiar with web services and TLS.

You must complete these tasks:

- Completed the task [Configuring CICS to use SSL](#).
- Set up a CICS web service requester that is implemented as described in [Developing SOAP web services](#).
- Set up a web service provider that is implemented to be started by the CICS web service requester. For example, a CICS web service provider as described in [Configuration example: Asserting an identity to the CICS web service provider](#).

You must have:

- Authorization to create CICS resource definitions.
- Authorization to install CICS resources.
- Authorization to define RACF commands.

### About this task

In this example, you configure a CICS region to issue web service requests over TLS. The CICS region validates the TLS server certificate that is presented by the started server. The web service requester identifies itself to the provider by using a TLS client certificate.

This task assumes the following definitions:

- *clientCertA* is the TLS certificate to be presented to the web service provider to identify the client.
- *datasetName* is the data set that *clientCertA* is exported into to allow it to be imported into the web service provider's key store.
- *webserviceA* is the name of the WEBSERVICE resource that is generated from the WSBIND file by CICS when the pipeline is installed.
- *urimapA* is the name of the URIMAP used to specify the URI of the target web service.
- *channelNameA* is the name of the channel that is used to pass the containers that hold the data that is mapped by the application data structure.
- *operationA* is the name of the operation that is to be started as defined in the WSDL file.
- *groupA* is the RDO group name.
- *pipelineA* is the name of a PIPELINE definition.
- *webserviceURIHost* is the hostname for the web service to be requested.
- *webserviceURIPort* is the port for the web service to be requested.
- *webserviceURIPath* is the path part of the URI for the web service to be requested.
- *soapRequesterXmlFile* is the zFS file that contains information about the processing nodes that act on a service request, and on the response.
- *shelfDir* is the zFS directory for the web service binding file.
- *wsProviderDir* is the web service binding directory on zFS for this pipeline.
- *matchID* an identifier for the security request recording (SRR) report.

- *regionUseridA* is the user ID that the CICS region is running under.
- *taskUserID* is the user that triggers the web service request.

## Procedure

1. Export certificate *clientCertA* to *dataset* and transfer it to the platform on which the web service provider is implemented.

```
RACDCERT EXPORT (LABEL('clientCertA')) DSN('datasetName') -
FORMAT(CERTDER)
```

2. Import the client certificate into the web service provider so it can be used to identify the web service requester. How this import is performed depends on the platform on which the web service provider is implemented.
3. Ensure that the WSBIND file in the *wsProviderDir* directory specifies the correct URI for the web service to be called. As the call is made over a TLS encrypted connection, the URI should start with https (rather than http). This configuration might require the source WSDL file being updated and the tools rerun to generate the updated WSBind file. Options for generating the WSDL file can be found in [Developing SOAP web services](#).
4. Ensure that the code that starts the web service specifies the URIMAP to be used, for example:

```
EXEC CICS INVOKE SERVICE(webServiceA)
      URIMAP(urimapA)
      CHANNEL(channelNameA)
      OPERATION(operationA)
      RESP(resp) RESP2(resp2)
```

5. Update the targeted web service provider to require a TLS connection with the requester that identifies itself with a client certificate. The targeted web service provider should also be configured for identity assertion with blind trust and an authentication mode of basic.
6. Define the pipeline.

The PIPELINE provides information about the message handler programs that act on an outgoing service request and on the response.

```
DEFINE PIPELINE(pipelineA) GROUP(groupA)
      CONFIGFILE(soapRequesterXmlFile) SHELF(shelfDir)
      WSDIR(wsProviderDir)
```

The CONFIGFILE specifies the CICS-supplied security handler that is configured for identity assertion with blind trust and an authentication mode of basic. An example is:

```
<requester_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="blind" mode="basic"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
  </service>
</requester_pipeline>
```

This sample must be copied into a zfs file named *soapProviderXmlFile*.

This configuration asserts that a user ID is asserted to the web service provider, but no password is passed. The user ID is either explicitly specified in the container DFHWS-USERID, or if that is not specified then it is *taskUserID*.

7. Define the URIMAP.

The URIMAP matches the URI of the web service request and provides information on how to process the requests. For service requesters, CICS does not create any URIMAP resources automatically when the PIPELINE resource is installed or as a result of a PERFORM PIPELINE SCAN command.

```
DEFINE URIMAP(urimapA) GROUP(groupA)
  USAGE(Client) SCHEME(HTTPS)
  PORT(webserviceURIPathPort) HOST(webserviceURIHost)
  PATH(webserviceURIPath)
  CERTIFICATE(clientCertA)
  AUTHENTICATE(No)
```

8. Install the definitions in group *groupA*.

## Results

Installing the resources in group *groupA* causes CICS to issue messages that report the installation of the PIPELINE and creation of the WEBSERVICE and a URIMAP resource:

```
DFHWB1560 04/05/2022 15:26:10 IYK2ZGV2 regionUserida URIMAP urimapA has been
created.
DFHPI0701 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA has been
created.
DFHPI0204 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 04/05/2022 15:26:10 APPLID regionUserida New WEBSERVICE webserviceRequesterA is being
created during a scan against PIPELINE
pipelineA.
```

```
DFHPI0910 I 04/05/2022 15:26:10 APPLID regionUserida WEBSERVICE webserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 04/05/2022 15:26:10 APPLID regionUserida WEBSERVICE webserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 04/05/2022 15:26:10 APPLID regionUserida PIPELINE pipelineA Implicit scan has
completed. Number of wsbind files found in the
WSDIR directory: 000001. Number of successful WEBSERVICE creates: 000001. Number of
failed WEBSERVICE creates: 000000.
```

If you inquire on the created CICS resources:

- The PIPELINE should have an ENABLESTATUS of ENABLED.
- The URIMAP should have an ENABLESTATUS of ENABLED.
- The WEBSERVICE should have a STATE of INSERVICE.

To validate the security environment is functioning correctly, you need a web service requester that is configured to authenticate by using a TLS client certificate. This web service needs to start an appropriate web service provider.

You can use the CICS security request recording (SRR) feature from within CICS Explorer to validate this example. With the **Regions view** in focus, you select the **Add Security Request Recording** context menu option. On that dialog, select the **All** tab and set the **Transaction ID** *txnA*. See [Checking that a CICS security configuration example is working by using the SRR for details](#).

The web service requester can now start the CICS web service provider.

## Configuring SOAP message security for CICS web services

Before you configure SOAP message security for CICS web services, make sure that you are familiar with the information in [How it works: SOAP message security](#), [Designing security for CICS web service providers](#), and [Designing security for CICS web service requesters](#).

These are common configuration examples that are based on the Design examples:

- Web service provider examples.
  - [Configuration example: Securing a direct request with TLS client authentication](#)
  - [Configuration example: Asserting an identity to the CICS web service provider](#)

- Web service requester examples.
  - [Identity assertion](#)
  - [Identity assertion](#)

When you have chosen how to secure web services, follow the instructions for the relevant tasks from [Table 26 on page 372](#).

<i>Table 26. Configuration activities to set up security for CICS web services</i>			
Activity	For web service providers	For web service requesters	Instructions
Review prerequisites for implementing WS-Security	✓	✓	<a href="#">Installing the prerequisites for WS-Security support</a>
Configure a pipeline for SOAP message security	✓	✓	<a href="#">Configuring the pipeline for Web Services Security</a>
Configure ICRX-based identity propagation	✓	n/a	<a href="#">Configuring provider mode web services for identity propagation</a>
Sign or encrypt SOAP messages	✓	✓	<a href="#">Configuring RACF for Web Services Security</a> and the instructions for signing or encrypting SOAP messages in <a href="#">Configuring the pipeline for Web Services Security</a>
Use the Trust client interface to invoke a Security Token Service (STS)	✓	✓	<a href="#">Invoking the Trust client from a message handler</a>
Write a custom security handler to secure SOAP messages	✓	✓	<a href="#">Writing a custom security handler</a>

## Installing the prerequisites for WS-Security support

To implement WS-Security support in CICS, you must install the IBM XML Toolkit for z/OS and add 3 libraries to the DFHRPL concatenation.

### Procedure

1. Install the free IBM XML Toolkit for z/OS.

You can download it from the following site: <https://www.ibm.com/servers/eserver/zseries/software/xml/>.

- **6.2** You must install version 1.11.
- **6.1** You must install version v1.10.

Other versions of the IBM XML Toolkit for z/OS do not work with Web Services Security support in CICS.

2. Add the following libraries to the DFHRPL concatenation:

- *hlq*.SIXMLOD1, where *hlq* is the high-level qualifier of the XML Toolkit.
- *hlq*.SCEERUN, where *hlq* is the high-level qualifier of the Language Environment®.
- *hlq*.SDFHWSLD, where *hlq* is the high-level qualifier of the CICS installation; for example CICSTS62 for CICS TS 6.2.



The first two libraries contain DLLs that are required at run time by the CICS security handler. IXM4C58 is provided by the XML Toolkit and is found in *hlq.SIXMLOD1*. C128N is provided by the Language Environment run time and is found in *hlq.SCEERUN*.

The *hlq.SDFHWSLD* library enables CICS to find the DFHWSSE1 and DFHWSXXX Web Services Security modules.

3. You might need to increase the value of the [EDSALIM system initialization parameter](#).

The three DLLs that are loaded require approximately 15 MB of EDSA storage.

## Results

If you do not have the libraries specified, you see the following message:

CEE3501S The module *module\_name* was not found.

The *module\_name* varies depending on which library is missing.

## Configuring the pipeline for WS-Security

To configure a pipeline to support WS-Security, you must add a security handler to your pipeline configuration files. You can use the security handler supplied with CICS, as described, or create your own. Remember that CICS does not support WS-Policy and ignores any security constraints that you configure in the WSDL document.

For information about the elements of pipeline configuration files, see [Pipeline configuration for WS-Security](#).

## Procedure

1. Add a `<wsse_handler>` element to your pipeline for a security handler.

For information about the structure of this element, see [<wsse\\_handler> pipeline configuration element](#). The security handler must be included in the `<service_handler_list>` element in a service provider or requester pipeline. The order of handler elements in the `<service_handler_list>` element determines the order that each handler is called at run time. In a pipeline that supports WS-Security, encrypted SOAP messages remain encrypted until the `<wsse_handler>` element is called. Therefore, you must specify the `<wsse_handler>` element before any other handler program that processes unencrypted messages.

Code the following elements:

```
<wsse_handler>
  <dfhwsse_configuration version="1">

  </dfhwsse_configuration>
</wsse_handler>
```

The `<dfhwsse_configuration>` element is a container for the other elements in the configuration.

2. Optional: Code an `<authentication>` element.

For information about the structure of this element, see [The <authentication> element](#).

- In a service requester pipeline, the `<authentication>` element specifies the type of authentication that must be used in the security header of outbound SOAP messages.
- In a service provider pipeline, the element specifies whether CICS uses the security tokens in an inbound SOAP message to determine the user ID under which work is processed.
  - a) Code the **trust** attribute to specify whether asserted identity is used and the nature of the trust relationship between service provider and requester.
  - b) Optional: If you specified **trust=none**, code the **mode** attribute to specify how credentials found in the message are processed.
  - c) In the `<authentication>` element, code these elements:
    - i) An optional, empty `<suppress/>` element.

If this element is specified in a service provider pipeline, the handler does not attempt to use any security tokens in the message to determine under which user ID the work runs.

If this element is specified in a service requester pipeline, the handler does not attempt to add to the outbound SOAP message any of the security tokens that are required for authentication.

- ii) In a requester pipeline, an optional `<algorithm>` element that specifies the URI of the algorithm that is used to sign the body of the SOAP message. You must specify this element if the combination of trust and mode attribute values indicate that the messages are signed. You can specify only the RSA with SHA1 algorithm in this element. The URI is `http://www.w3.org/2000/09/xmlsig#rsa-sha1`.
- iii) An optional `<certificate_label>` element that specifies the label that is associated with an X.509 digital certificate installed in RACF. If you specify this element in a service requester pipeline and the `<suppress>` element is not specified, the certificate is added to the security header in the SOAP message. If you do not specify a `<certificate_label>` element, CICS uses the default certificate in the RACF key ring.

This element is ignored in a service provider pipeline.

- 3. Optional: Code an `<sts_authentication>` element as an alternative to the `<authentication>` element.

Do not code both in your pipeline configuration file. For information about the structure of this element, see [The `<sts\_authentication>` element](#). This element specifies that a Security Token Service (STS) is used for authentication and determines the type of request that is sent.

- a) Optional: In service provider mode only, code the **action** attribute to specify whether the STS verifies or exchanges a security token.
- b) Within the `<sts_authentication>` element, code these elements:
  - i) An `<auth_token_type>` element. This element is required when you specify a `<sts_authentication>` element in a service requester pipeline and is optional in a service provider pipeline. For more information, see [The `<auth\_token\_type>` element](#).
    - In a service requester pipeline, the `<auth_token_type>` element indicates the type of token that STS issues when CICS sends it the user ID contained in the DFHWS-USERID container. The token that CICS receives from the STS is placed in the header of the outbound message.
    - In a service provider pipeline, the `<auth_token_type>` element is used to determine the identity token that CICS takes from the message header and sends to the STS to exchange or validate. CICS uses the first identity token of the specified type in the message header. If you do not specify this element, CICS uses the first identity token that it finds in the message header. CICS does not consider the following as identity tokens:
      - `wsu:Timestamp`
      - `xenc:ReferenceList`
      - `xenc:EncryptedKey`
      - `ds:Signature`
  - ii) In a service provider pipeline only, an optional, empty `<suppress/>` element. If this element is specified, the handler does not attempt to use any security tokens in the message to determine the user ID that the work runs under. The `<suppress/>` element includes the identity token that is returned by the STS.

- 4. Optional: Code an `<sts_endpoint>` element.

Use this element only if you have also specified an `<sts_authentication>` element. In the `<sts_endpoint>` element, code the following elements:

- An `<endpoint>` element. This element contains a URI that points to the location of the Security Token Service (STS) on the network. It is recommended that you use TLS to keep the connection to the STS secure, rather than using HTTP.

To use SAML support, set the endpoint to `cics://PROGRAM/DFHSAML`.

You can also specify an IBM MQ endpoint, by using the JMS format of URI.

- An optional `<jvmserver>` element. This element identifies the JVM server that is configured to run the SAML token service. If this element is not included, the default sample resource JVM server DFHXSTS is assumed. This element is valid only if you are using SAML: if you use it in other situations, an error occurs.
5. Optional: If you require inbound SOAP messages to be digitally signed, code an empty `<expect_signed_body/>` element.

The `<expect_signed_body/>` element indicates that the `<body>` of the inbound message must be signed. If the body of an inbound message is not correctly signed, CICS rejects the message with a security fault.

6. Optional: If you want to reject inbound SOAP messages that are digitally signed, code an empty `<reject_signature/>` element.
7. Optional: If you require inbound SOAP messages to be encrypted, code an empty `<expect_encrypted_body/>` element.

The `<expect_encrypted_body/>` element indicates that the `<body>` of the inbound message must be encrypted. If the body of an inbound message is not correctly encrypted, CICS rejects the message with a security fault.

8. If you want to reject inbound SOAP messages that are partially or fully encrypted, code an empty `<reject_encryption/>` element.
9. Optional: If you require outbound SOAP messages to be signed, code a `<sign_body>` element.
- a) In the `<sign_body>` element, code an `<algorithm>` element.
  - b) Following the `<algorithm>` element, code a `<certificate_label>` element.

Here is an example of a completed `<sign_body>` element:

```
<sign_body>
  <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
  <certificate_label>SIGCERT01</certificate_label>
</sign_body>
```

10. Optional: If you require outbound SOAP messages to be encrypted, code an `<encrypt_body>` element.
- a) In the `<encrypt_body>` element, code an `<algorithm>` element.
  - b) Following the `<algorithm>` element, code a `<certificate_label>` element.

Here is an example of a completed `<encrypt_body>` element:

```
<encrypt_body>
  <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
  <certificate_label>ENCCERT02</certificate_label>
</encrypt_body>
```

## Example

The following example shows a completed security handler in which most of the optional elements are present:

```
<wsse_handler>
  <dfhwsse_configuration version="1">
    <authentication trust="signature" mode="basic">
      <suppress/>
      <certificate_label>AUTHCERT03</certificate_label>
    </authentication>
    <expect_signed_body/>
    <expect_encrypted_body/>
    <sign_body>
      <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
      <certificate_label>SIGCERT01</certificate_label>
    </sign_body>
    <encrypt_body>
      <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
      <certificate_label>ENCCERT02</certificate_label>
    </encrypt_body>
  </dfhwsse_configuration>
</wsse_handler>
```

```
</encrypt_body>
</dfhwsse_configuration>
</wsse_handler>
```

## Configuring provider mode web services for identity propagation

Identity propagation with a web service request relies on trust-based configurations; for example, using a client-certified TLS connection from IBM DataPower Gateway. In this task, you configure a PIPELINE resource to expect an ICRX identity token in the WS-Security header, sent from a trusted client.

### Before you begin

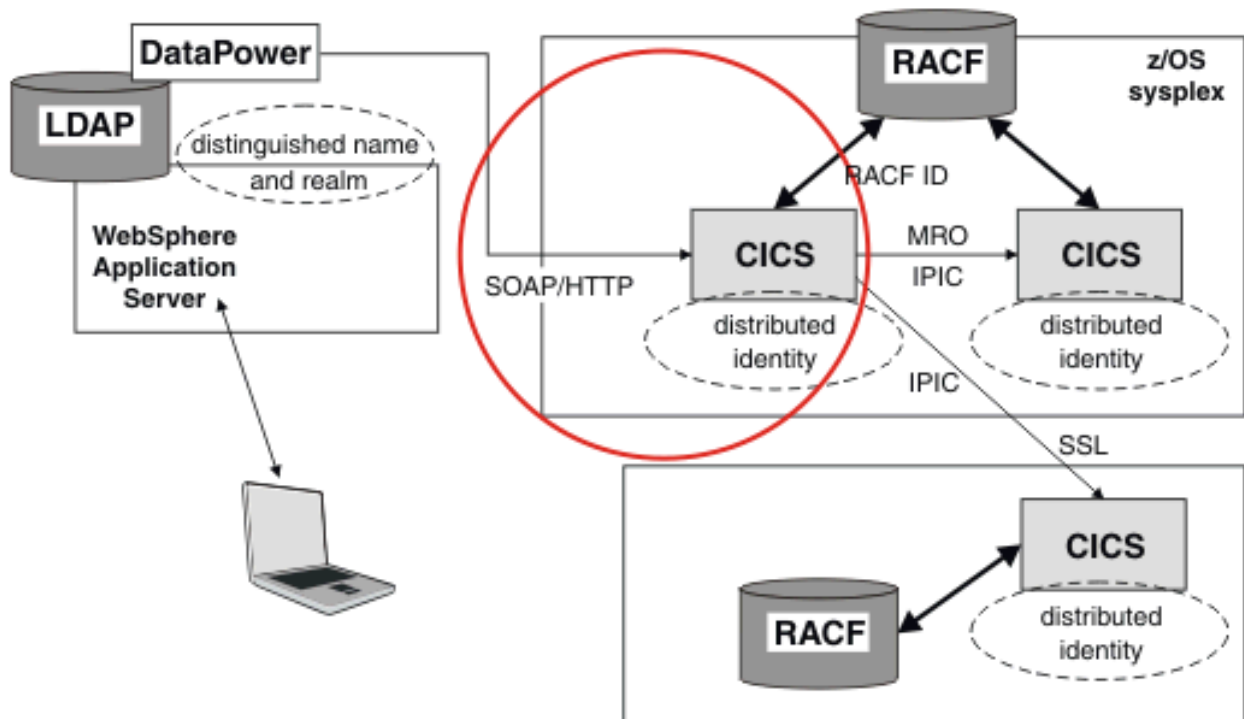
Configure your RACF RACMAP settings before you configure your web service connections. Otherwise, you receive the RACF ICH408I message for every unmapped request that is sent to RACF. For more information about configuring the RACF **RACMAP** command, see [Configuring RACF for identity propagation](#).

You must configure a *trust* relationship between the IBM DataPower Gateway appliance and CICS, for example, using TLS client certification between IBM DataPower Gateway and CICS. The digital certificate that IBM DataPower Gateway uses to identify itself must be associated with a user ID, and that user ID must be granted surrogate authority to assert identities. For more information about surrogate authority, see [Surrogate security](#).

### About this task

This task explains how to use CICS with a IBM DataPower Gateway appliance to provide a web service configuration that can propagate distributed identities in a secure and robust way. The circle in the diagram indicates that this task explains the CICS-specific configuration.

Figure 145. Configuring CICS to expect an ICRX identity token from IBM DataPower Gateway.



IBM DataPower Gateway acts as an intermediary between CICS and other applications. Remote web service requester applications connect to the IBM DataPower Gateway appliance using the SOAP protocol. IBM DataPower Gateway authenticates the credentials supplied by the remote client and mapping the credentials to a z/OS ICRX identity token, which identifies the distributed identity of a user.

The SOAP message is then forwarded to CICS over the trusted TLS connection with an ICRX identity token in a WS-Security header. For more information about ICRX identity tokens, see [z/OS Security Server RACF Data Areas](#).

CICS receives the SOAP message from IBM DataPower Gateway. The PIPELINE configuration file specifies *blind* trust, because the only possible client is the IBM DataPower Gateway appliance, and IBM DataPower Gateway is communicating with CICS over a secure TLS connection. Therefore, you do not need to specify additional authentication in the PIPELINE configuration file. The WS-Security handler program locates the first ICRX found in the WS-Security header and uses the ICRX to identify the user.

## Procedure

1. Create a PIPELINE resource, or edit an existing PIPELINE resource to specify the basic-ICRX mode, which allows the PIPELINE to receive an ICRX.

The most typical combination is the blind trust with the basic-ICRX mode. For more information about the PIPELINE resource element, see [The <authentication> element](#).

Here is an example PIPELINE configuration file, showing blind trust with the basic-ICRX mode:

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="blind" mode="basic-ICRX"/>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.2_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Here is an example SOAP message with an ICRX identity, using blind trust:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
      SOAP-ENV:mustUnderstand="1">

      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
        wsu:Id="ICRX"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd"
        ValueType="http://www.IBM.com/xmlns/prod/zos/saf#ICRXV1">

ICRX IS HERE

      </wsse:BinarySecurityToken>

    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

**APPLICATION SPECIFIC XML IS HERE**

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. Ensure that IBM DataPower Gateway is configured to be able to send ICRX information. See [Identity propagation](#).

## Results

Web service requests from IBM DataPower Gateway with an ICRX identity token in the WS-Security header, connected over a client-certified TLS connection, can now flow.

## Configuring RACF for WS-Security

You must configure an external security manager, such as RACF, to create public-private key pairs and X.509 certificates for signing and encrypting outbound SOAP messages and to authenticate and decrypt signed and encrypted inbound SOAP messages.

### Before you begin

Before you perform this task, you must have RACF set up to work with CICS. Specify the **DFLTUSER**, **KEYRING**, and **SEC=YES** system initialization parameters in the CICS region that contains your web services pipelines.

**Note:** Multiple certificates with the same Distinguished Name on the same **KEYRING** are not supported.

### Procedure

1. To authenticate inbound SOAP messages that are signed:
  - a) Import the X.509 certificate into RACF as an ICSF key.
  - b) Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter, using the **RACDCERT** command:

```
RACDCERT ID(userid1)
CONNECT(ID(userid2) LABEL('label-name') RING(ring-name))
```

where:

- *userid1* is the default user ID of the key ring or has authority to attach certificates to the key ring for other user IDs.
- *userid2* is the user ID that you want to associate with the certificate.
- *label-name* is the name of the certificate.
- *ring-name* is the name of the key ring that is specified in the **KEYRING** system initialization parameter.

- c) Optional: If you want to use asserted identities, ensure that the user ID associated with the certificate has surrogate authority to allow work to run under other user IDs.

Also, make sure that any additional certificates included in the SOAP message header are also imported into RACF.

The SOAP message can contain a binary security token in the header that either includes the certificate or contains a reference to the certificate. This reference can be the KEYNAME (the certificate label in RACF), a combination of the ISSUER and SERIAL number, or the SubjectKeyIdentifier. CICS can recognize the SubjectKeyIdentifier only if it has been specified as an attribute in the definition of the certificate in RACF.

2. To sign outbound SOAP messages:
  - a) Create an X.509 certificate and a public-private key pair using the following **RACDCERT** command:

```
RACDCERT ID(userid2) GENCERT
SUBJECTSDN(CN('common-name')
T('title')
OU('organizational-unit')
O('organization'))
```

```
L('locality')
SP('state-or-province')
C('country'))
WITHLABEL('label-name')
```

where *userid2* is the user ID that you want to associate with the certificate.

When you specify the certificate *label-name* value, do not use the following characters:

```
< > : ! =
```

- b) Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter. Use the **RACDCERT** command.

- c) Export the certificate and publish it to the intended recipient of the SOAP message.

You can edit the pipeline configuration file so that CICS automatically includes the X.509 certificate in the binary security token of the SOAP message header for the intended recipient to validate the signature.

- 3. To decrypt inbound SOAP messages that are encrypted, the SOAP message must include the public key that is part of a key pair, where the private key is defined in CICS.

- a) Generate a public-private key pair and certificate in RACF for encryption.

The key pair and certificate must be generated using ICSF.

- b) Attach the certificate to the key ring specified in the **KEYRING** system initialization parameter. Use the **RACDCERT** command.

- c) Export the certificate and publish it to the generator of the SOAP messages that you want to decrypt.

The generator of the SOAP message can then import the certificate that contains the public key and use it to encrypt the SOAP message. The SOAP message can contain a binary security token in the header that either includes the public key or contains a reference to it. This reference can be the KEYNAME, a combination of the ISSUER and SERIAL number, or the SubjectKeyIdentifier. CICS can recognize the SubjectKeyIdentifier only if it has been specified as an attribute in the definition of the public key in RACF.

- 4. To encrypt outbound SOAP messages:

- a) Import the certificate that contains the public key that you want to use for encryption into RACF as an ICSF key.

The intended recipient must have the private key associated with the public key to decrypt the SOAP message.

- b) Attach the certificate that contains the public key to the key ring specified in the **KEYRING** system initialization parameter. Use the **RACDCERT** command.

CICS uses the public key in the certificate to encrypt the SOAP body and sends the certificate containing the public key as a binary security token in the SOAP message header. The public key is defined in the pipeline configuration file.

## What to do next

This configuration for signing and encrypting outbound messages requires that the certificate used is owned by the CICS region user ID. The certificate must be owned by the CICS region *userid* because RACF allows only the certificate owner to extract the private key, which is used for the signing or encryption process.

If CICS needs to sign or encrypt a message using a certificate that it does not own, you can share a single certificate between CICS systems by following the instructions in [Using an existing certificate that is not owned by the CICS region user ID](#).

## Invoking the Trust client from a message handler

CICS provides an interface so that you can write your own message handler to invoke a Security Token Service (STS). With this interface you can perform more advanced processing than the CICS security handler. You can use the Trust client instead of the CICS security handler or in addition to it.

### Procedure

1. Extract the correct token from the security message header of the inbound or outbound message.
2. Link to program DFHPIRT, passing the channel DFHWSTC-V1 and the following required containers:
  - DFHWS-STSUR, containing the location of the STS on the network.
  - DFHWS-STSACTION, containing the URI of the type of request that the STS must perform. The two supported actions are `issue` and `validate`.
  - DFHWS-IDTOKEN, containing the token that must either be verified or exchanged by the STS.
  - DFHWS-TOKENTYPE, containing the type of token that the STS must send back in the response.
  - DFHWS-SERVICEURI, containing the URI of the web service operation that is being invoked.

You can optionally include the DFHWS-XMLNS container to provide the namespaces of the SOAP message that contains the security token. This container is described in more detail in [The header processing program interface](#).

3. DFHPIRT returns with the response from the STS.

A successful response is stored in the DFHWS-RESTOKEN container.

If the STS encounters a problem with the request, it returns a SOAP fault. DFHPIRT puts the SOAP fault in the DFHWS-STSFault container. If the STS provides a reason for issuing the SOAP fault, the reason is put in the DFHWS-STSREASON container.

If an abend occurs, a DFHERROR container is returned that contains details of the processing error.

Your message handler must handle these responses and perform suitable processing in the event of an error. For example, the message handler might return a suitable SOAP fault to the web service requester.

4. Process the response as appropriate.

In provider mode, your pipeline processing must ensure that a user name that CICS can understand is placed in the DFHWS-USERID container by the time the message reaches the application handler. In requester mode, your message handler must add the correct token to the outbound message security header.

### What to do next

When you have written your message handler, deploy the program in CICS and update the appropriate pipeline configuration files. In service requester pipelines, define your message handler to occur at the end of the pipeline processing but before the CICS security handler. In service provider pipelines, define your message handler at the beginning of the pipeline but after the CICS security handler.

## Writing a custom security handler

To use your own security procedures and processing, write a custom message handler to process secure SOAP messages in the pipeline.

You need to decide the level of security that your security handler must support, and ensure that an appropriate SOAP fault is returned when a message includes security that is not supported. The message handler must also be able to cope with security on inbound and outbound messages.

Here is a likely set of steps that your security handler would implement:

1. Retrieve the DFHREQUEST or DFHRESPONSE container using an **EXEC CICS GET CONTAINER** command.



2. Parse the XML to find the security token that is in the WS-Security message header. The header starts with the <wsse:Security> element. The security token might be a user name and password, a digital certificate, or an encryption key. A message can have many tokens in the security header, so your handler needs to identify the correct one to process.
3. Perform the appropriate processing, depending on the security that is implemented in the message:
  - To perform basic authentication of a Kerberos token, issue an **EXEC CICS VERIFY TOKEN** command. This command checks that the supplied Kerberos token is valid. If the command is successful, update the DFHWS-USERID container with an **EXEC CICS PUT CONTAINER**. Otherwise, issue an **EXEC CICS SOAPFAULT CREATE** command.
  - To perform basic authentication of a password or password phrase, issue an **EXEC CICS VERIFY PHRASE** command. This command checks the user name and password in the security header of the message. If the command is successful, update the DFHWS-USERID container with an **EXEC CICS PUT CONTAINER**. Otherwise, issue an **EXEC CICS SOAPFAULT CREATE** command.
  - You might also want to write an audit record each time a service is requested, for example, you could write a message to a CICS user journal.
  - To perform advanced authentication, either by exchanging or validating a range of tokens with a Security Token Service, use the Trust client interface which enables you to interact with the STS directly. . See [“Invoking the Trust client from a message handler” on page 380](#) for details.
  - Validate the credentials of the digital certificate if the message is signed.
  - If parts of the message are encrypted, decrypt the message using the information in the security header. The [How CICS complies with Web Services Security specifications](#) specification provides information about how to do this
4. Define your security handler program in CICS and update the pipeline configuration file, ensuring that it is correctly placed in the XML. In a service requester pipeline configuration file, the security handler must be configured to run at the end of the pipeline. In a service provider pipeline configuration file, the security handler must be configured to run at the beginning of the pipeline.

For examples of custom message handlers, see [IBM Redbooks: Implementing CICS Web services](#).



---

## Chapter 17. Security for 3270

The 3270 terminal or emulator is used to connect to CICS. You need to configure it for secure operations within CICS.

When you decide how to secure the 3270 terminal or emulator, you need to consider the standard security principles: authentication, identification, authorization, integrity, confidentiality, and audit. For a description of these terms, see [What does security mean in CICS?](#).

This section explains the mechanism and options of 3270 security, as well as how to design and configure 3270 security with examples.

---

### How it works: 3270 security

Understand how 3270 terminals or terminal emulators are configured for secure operation within CICS.

The following terms are specific to 3270 terminals. These terms are described here and referenced in the 3270 security documentation.

The default user ID, *defaultUser*, is used for any terminal that has yet to supply any credentials. This value is specified by SIT parameter **DFLTUSER**.

Sign on transaction, *signOn*, refers to a CICS transaction used by a 3270 terminal user to authenticate themselves to the system. CICS provides transactions CESN and CESL although environments can use a locally written transaction.

Sign off transaction, *signOff*, refers to a CICS transaction used by a 3270 terminal user to remove their credentials from the terminal and can also disconnect the terminal from the CICS system. CICS provides transaction CESF although environments can use a locally written transaction.

The Good Morning Transaction, *gmTran*, is a transaction that is started on a newly connected 3270 terminal. The transaction that is used is set by the SIT parameter **GMTRAN** and defaults to the CICS supplied transaction CSGM.

The CICS region VTAM® Generic application identifier, *grApplid* specified by the SIT parameter **GRNAME**, is used when CICS checks whether a user has permission to access the CICS region. If **GRNAME** is not specified for the CICS region, the region's VTAM application identifier referenced as *applid* that is specified by SIT parameter **APPLID** is used. Whenever *grApplid* is used, unless stated otherwise, it references the value *applid* if *grApplid* is blank.

When a terminal device connects into CICS by using 3270 data protocols, multiple facilities are available:

- Users authenticate by using a *signOn* transaction or for fixed devices, a statically defined user identification can be set for the device.
- Authorization ensures that only permitted users and devices can connect to the CICS region.
- Confidentiality of the data that passes between a device and CICS is protected by encryption while the data passes over the network infrastructure. Physical security of the network hardware also provides further protection.

### 3270 connections to z/OS

3270 devices communicate with CICS by using the Virtual Telecommunications Access Method (VTAM) on the z/OS system that hosts the CICS system.

In some instances, devices are physically wired to a z/OS system and the cabling permits the device to connect to the VTAM infrastructure. This physical cabling provides a level of trust on the identity of the physical device and confidentiality of the data that passes between it and the z/OS system. These devices have a fixed *Logical Unit* (LU) name that is assigned to them in the VTAM definitions, often referred to as the *Network Identifier* or NETID. Typically, such devices are generally not terminals that are used by

individual users but physical devices, which are used by multiple users, such as printers. This device always has the same LU.

Usually, an individual user's device connects over a TCP/IP network by using *Telnet 3270* (TN3270) or Telnet 3270 Enhanced (TN3270E) protocols to a TN3270 server on a z/OS system. Unless explicitly stated, use of TN3270 also applies to TN3270E.

TN3270 server configuration can provide:

- Confidentiality of data that passes between the remote device and the target system by requiring the connection be encrypted.
- Authorization that the client device is permitted to access the system by one or more techniques:
  - Restricting the client IP addresses allowed to connect.
  - Requiring a client certificate.
  - Determining where within the network it is initially routed (for example an application selection menu) and to what it is ultimately permitted to connect.

When a TN3270 device is connected to the VTAM infrastructure, it is allocated a VTAM LU. The TN3270 server configuration controls if the VTAM LU is a specific value or selected from a pool. Depending on the server configuration, a reconnecting client might not have the same LU as previously. A client can request a specific value or pool of values is used for the LU but the server configuration controls if the specification is accepted or rejected. For more information on TN3270 server security, see [TN3270E Telnet server security](#).

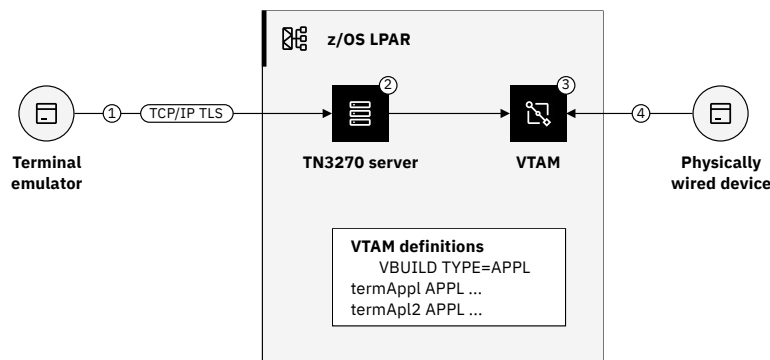


Figure 146. 3270 connection for physical terminals and emulators

Figure 146 on page 384 shows how a terminal emulator or physically wired device connects to z/OS.

1. A 3270 terminal emulator on a user's device connects over an encrypted TLS session to a z/OS LPAR.
2. The TN3270 server on the LPAR accepts the connection and allocates an LU of *termAppl*.
3. Incoming connection is then passed to VTAM for connection to target.
4. A physically wired device connects through cabling to VTAM with fixed LU *termAppl2*.

## CICS connection to VTAM to enable 3270 terminals

3270 terminals or terminal emulators connect to CICS using the default user.

When CICS starts, a check is made that *defaultUser* is authorized to use the CICS region as shown in [Figure 147 on page 385](#). For more information about 3270 terms used in this topic, see [How it works: 3270 security](#).

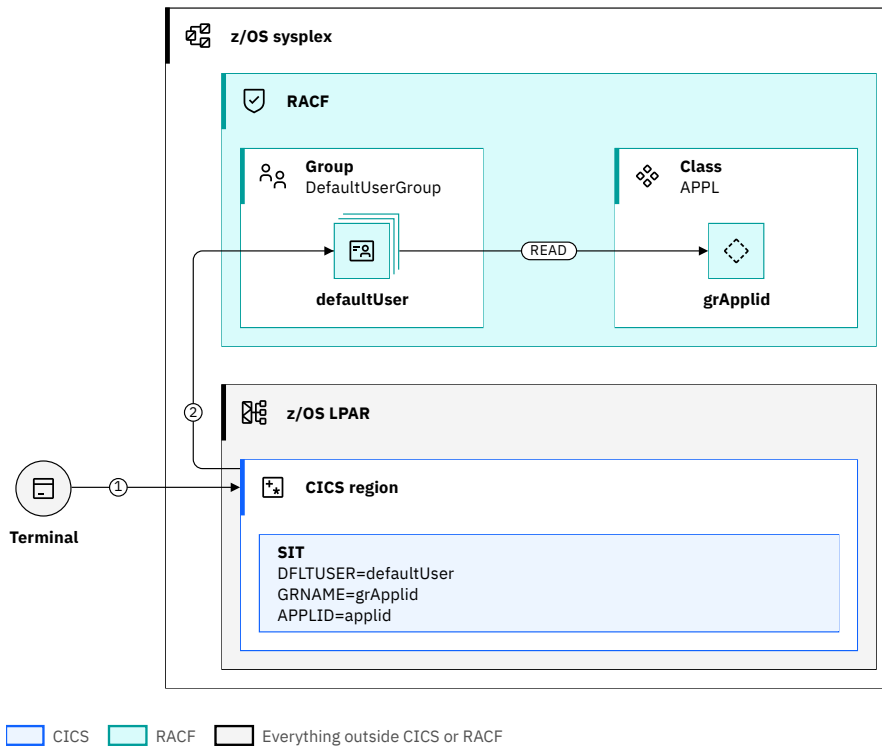


Figure 147. Region startup checks for defaultUser

As part of the CICS start process, shown in [Figure 147 on page 385](#):

1. The default user, *defaultUser* specified by SIT parameter **DFLTUSER**, is signed onto the CICS region.
2. CICS verifies that *defaultUser* is authorized to the CICS region by using VTAM. This check is achieved by calling RACF to verify that *defaultUser* has READ access to profile *grApplid* in class APPL.

**Important:** If a profile is not found in the APPL class that matches this value, CICS assumes that this check is not required.

**Recommendation:** Use different values for *defaultUser* for different types of regions, such as test and production. This configuration gives trust such that the region is running with the appropriate security and is not one that is incorrectly configured to use the wrong *grApplid*.

## 3270 devices that connect to CICS

Three categories of CICS devices use the 3270 protocols. These devices are terminals that are dynamically defined, statically defined, or are consoles.

- “Dynamically defined 3270 terminals” on [page 385](#)
- “Statically defined 3270 terminals” on [page 386](#)
- “3270 consoles” on [page 388](#)

For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

### Dynamically defined 3270 terminals

These terminals are defined by CICS as they connect by using a process that is called *Terminal Autoinstall*. Because they do not usually have a predefined identity, they are assigned *defaultUser*. Afterward, the user authenticates by using a *signOn* transaction.

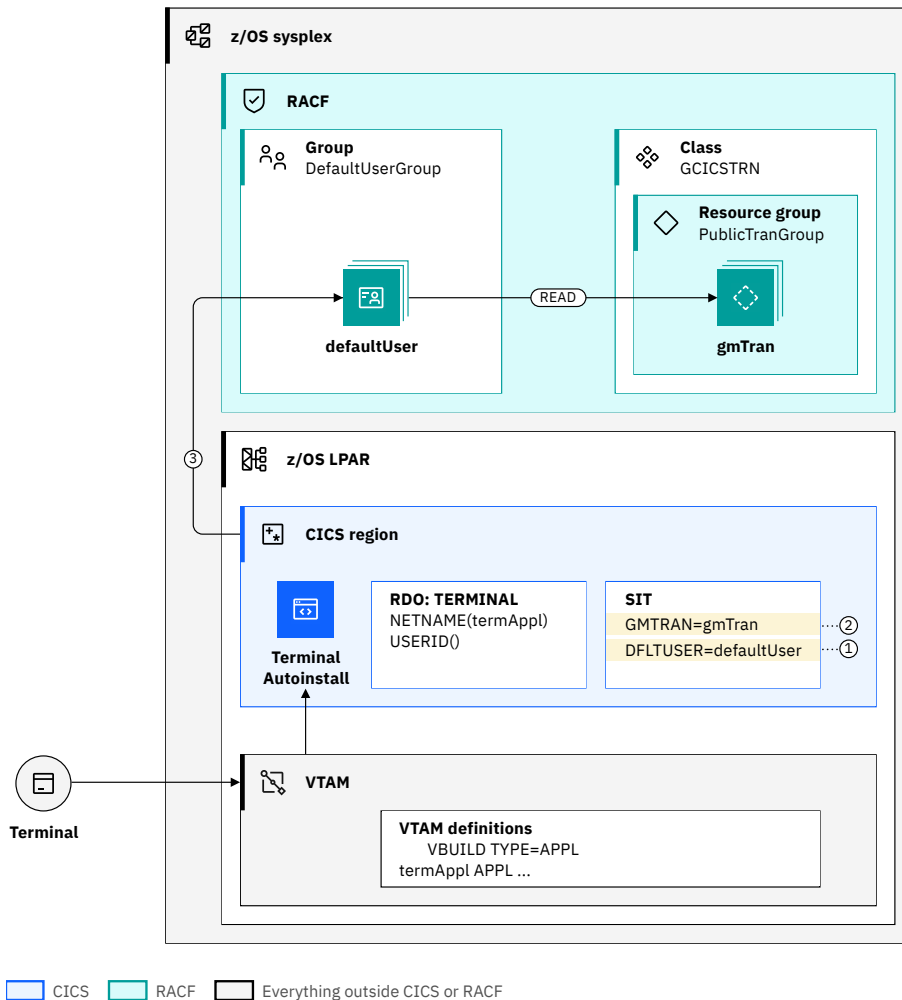


Figure 148. Dynamically defined 3270 connection

Figure 148 on page 386 shows a terminal that uses LU *termAppl* connects to CICS. On receipt of the connection from VTAM, CICS checks the currently defined terminals for one with a NETNAME value of *termAppl*. Because one is not found, CICS Terminal Autoinstall is started to dynamically create a terminal definition.

1. As the USERID value of the terminal definition is blank. This terminal is assigned the default user of the region *defaultUser*.
2. The Good Morning Transaction, *gmTran*, is started.
3. CICS verifies that *defaultUser* is authorized to run *gmTran*. This check is made by calling RACF to verify that *defaultUser* has READ access to resource *gmTran* in resource group class GCICSTRN.

**Important:** If *gmTran* is a CICS Category 3 transaction (for example, CESL, CESN or CSGM), this check is not undertaken. CICS Transaction CSGM is not a Category 3 transaction for releases earlier than CICS Transaction Server for z/OS, Version 6 Release 1.

## Statically defined 3270 terminals

These terminals are defined to CICS in advance. Each definition must have a specific and unique *termAppl* specified along with any other attributes required.

CICS preset security terminals use a fixed user ID that is specified in their CICS terminal definition. Generally, these terminals are physical and use a *presetID* specified in the definition. These terminals are

known as having *preset security* or a *preset user ID*. Any attempts to use a *signOn* transaction to change the credentials fail.

Security checks for terminals with preset security are undertaken in two stages.

1. When the definition is created in CICS. Usually during CICS initialization or on a subsequent manual installation through a tool such as CEDA. This is shown in [Figure 149 on page 387](#).
2. When the device connects to CICS, as shown in [Figure 150 on page 388](#).

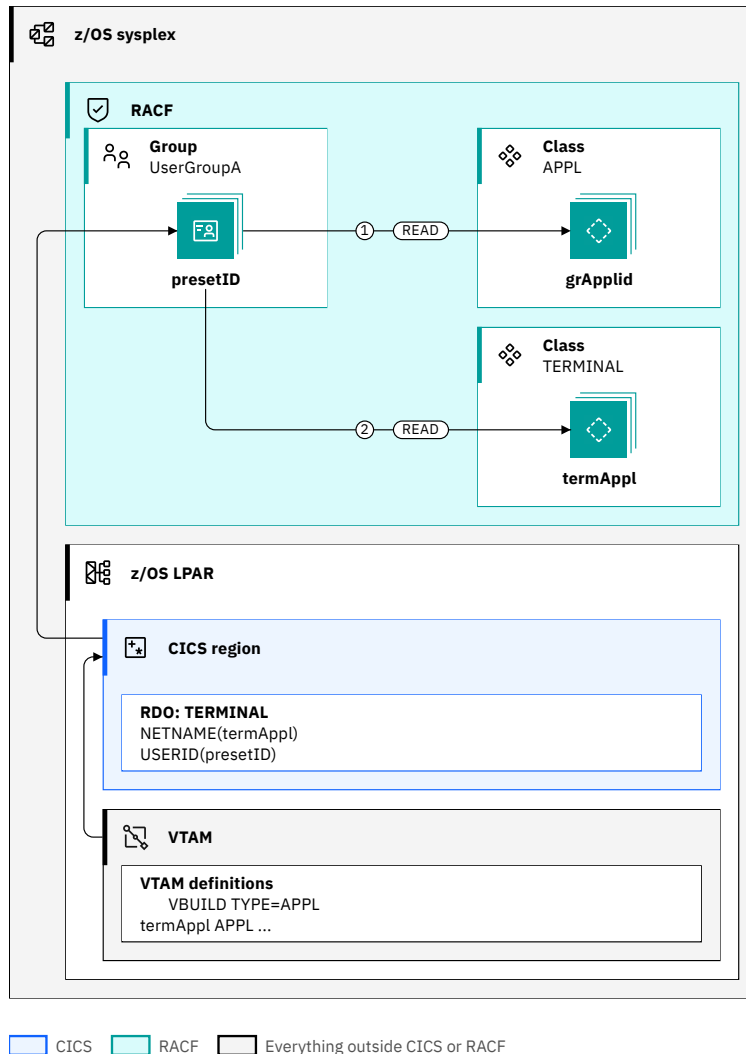


Figure 149. Statically defined 3270 terminal configuration

As shown in [Figure 149 on page 387](#), because the terminal definition has a preset user ID, checks are undertaken when the terminal definition is installed as follows. These checks are to verify that *presetID* is authorized to use the CICS and the specific terminal.

1. CICS verifies that *presetID* is authorized to the CICS region. This check is achieved by calling RACF to verify that *presetID* has READ access to profile *grApplid* in class APPL.

**Important:** If a profile is not found in the APPL class that matches this value, CICS assumes that this check is not required.

2. CICS verifies that *presetID* is authorized to the terminal. This check is achieved by calling RACF to verify that *presetID* has READ access to resource profile *termAppl* in class TERMINAL.

**Important:** If a profile is not found in the TERMINAL class that matches this value, CICS assumes that this check is not required.

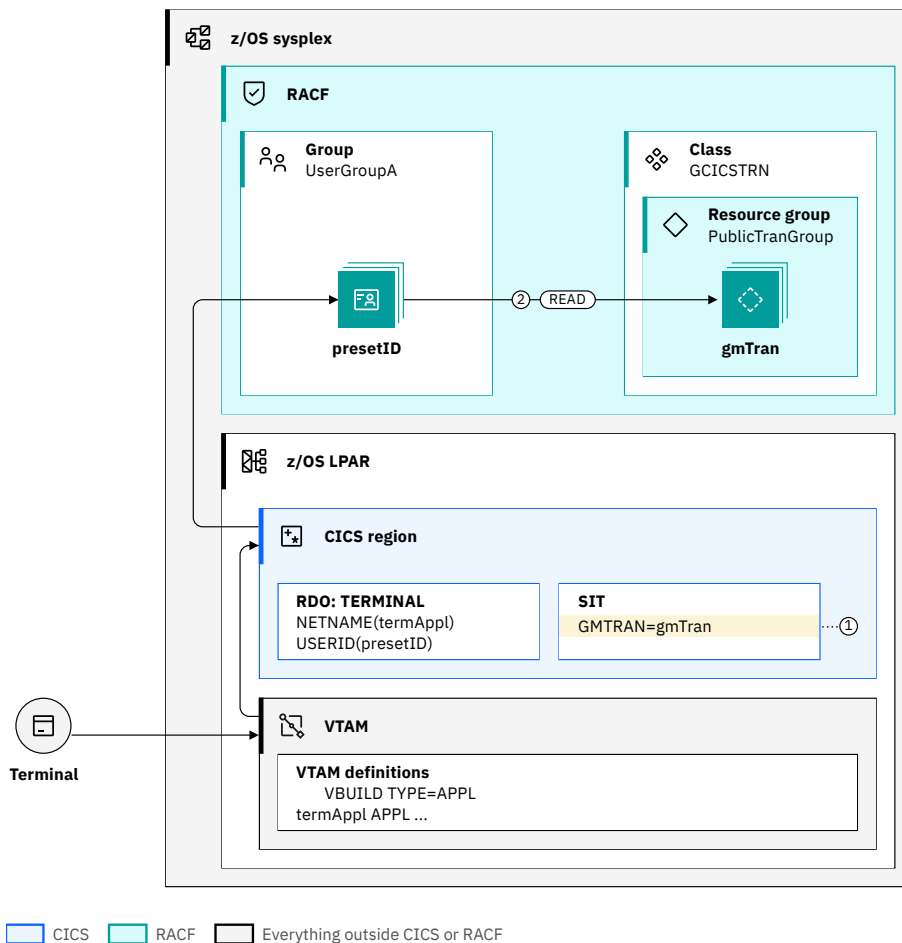


Figure 150. Statically defined 3270 connection

Figure 150 on page 388 shows what happens when a statically defined 3270 terminal connects to CICS:

1. The Good Morning Transaction, *gmTran* specified by the SIT parameter **GMTRAN**, is started on the terminal.
2. CICS verifies that *presetID* is authorized to run *gmTran*. This check is achieved by calling RACF to verify that *presetID* has READ access to resource *gmTran* in resource group class GCICSTRN.

**Important:** If *gmTran* is a CICS Category 3 transaction (for example, CESL, CESL or CSGM) this check is not undertaken. CICS Transaction CSGM is not a Category 3 transaction for releases earlier than CICS Transaction Server for z/OS, Version 6 Release 1.

## 3270 consoles

3270 consoles are devices that are used by operations staff to manage the system. Unlike 3270 terminals, they do not connect to individual CICS regions but instead display messages from multiple jobs that run on the sysplex. Commands from a console are sent to one or more jobs by using a modify command with any returned messages that are displayed on the screen. Console commands routed to a CICS system take the form of a transaction with optional parameters, for example, **CEMT INQUIRE TASK**.

A console can be a physical device that is dedicated to the purpose or a virtual one.



When a console user issues a modify command that specifies the running CICS region job name, z/OS passes the console name, associated user ID, and command to the CICS system. CICS checks the installed console definitions for an entry with the same value in the **CONSNAME** field. If no match is found, CICS installs a console definition that provides console autoinstall in the region.

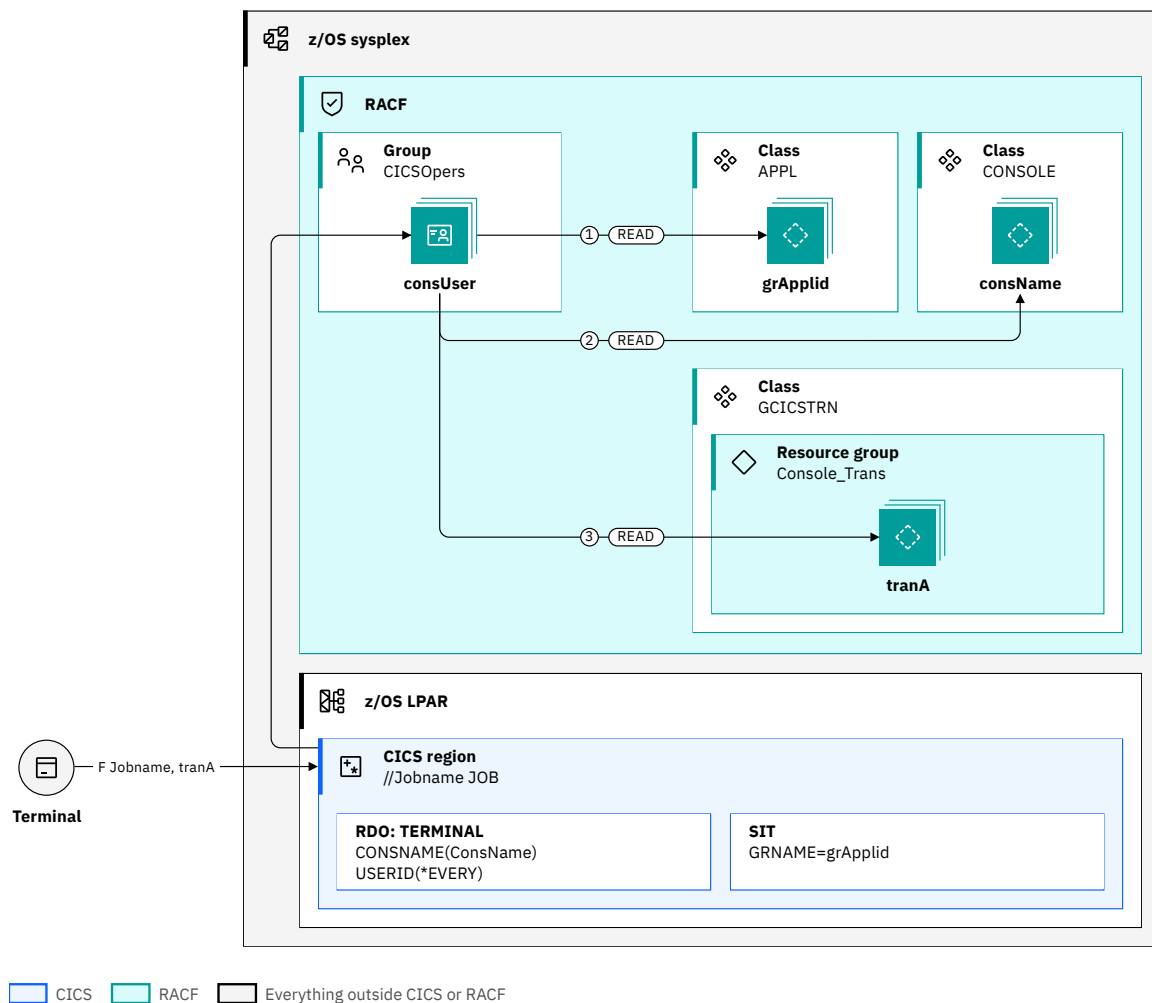


Figure 151. Physical console connection

Usually, the installed console model has the value **\*EVERY** on the **USERID** attribute, so the user ID passed by z/OS is used as the **USERID** of the defined console. Figure 151 on page 389 shows:

1. CICS verifies that **consUser** is authorized to the CICS region. This check is achieved by calling RACF to verify that **consUser** has **READ** access to profile **grApplid** in class **APPL**.
2. CICS verifies that **consUser** is authorized to the console. This check is achieved by calling RACF to verify that **consUser** has **READ** access to resource profile **consName** in class **CONSOLE**.
3. CICS verifies that **consUser** is authorized to run **tranA**, the transaction that the console commands to start. This check is achieved by calling RACF to verify that **consUser** has **READ** access to resource **tranA** in group resource class **GCICSTRN**.

**Important:** Because the console is defined **USERID=\*EVERY**, an **EXEC CICS INQUIRE** command from the terminal shows the last user of the console. The value is updated every time that a command is received from the console.

# 3270 terminal authentication

When a 3270 terminal user authenticates, a set of checks is made to ensure that the connection is allowed.

It is assumed that the user is connected to a 3270 terminal that uses LU *termAppl*. This assumption means that the terminal is assigned the CICS default user ID *defaultUser* as described in [Dynamically defined 3270 Terminals](#). For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

The following diagram shows the security checks conducted during 3270 authentication:

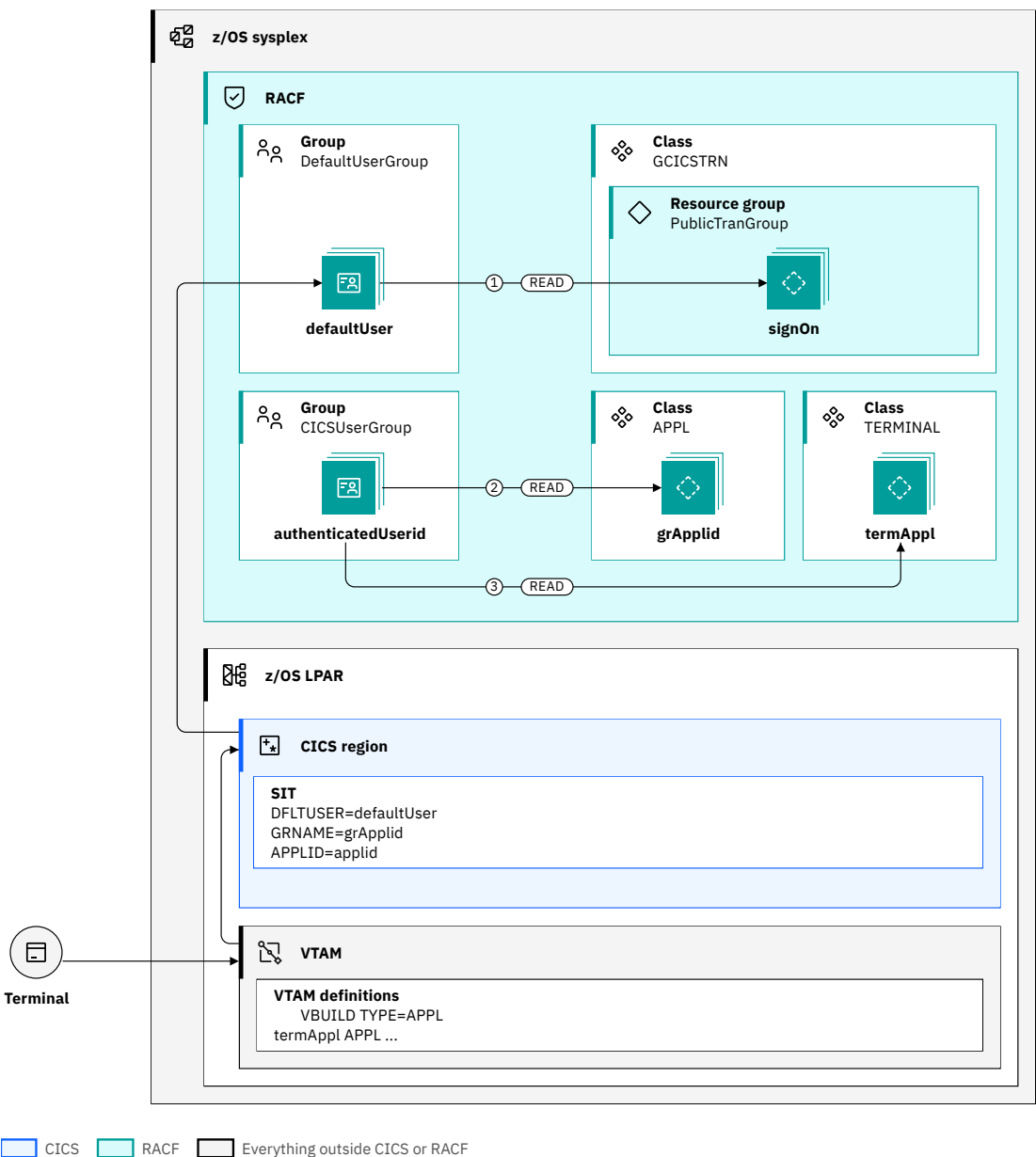


Figure 152. 3270 authentication checks

As shown in [Figure 152 on page 390](#), the security flow is as follows:

1. The user starts transaction *signOn* to sign onto the CICS region. CICS verifies that *defaultUser* is authorized to run *signOn*. This check is made by calling RACF to verify that *defaultUser* has READ access to resource *signOn* in group resource class GCICSTRN.

If *signOn* is a CICS Category 3 transaction (CESL or CESN), this check is not undertaken.

The user supplies credentials for user *authenticatedUserid* and checks are made to ensure that this user is authorized to use both the CICS region and the specific terminal in the following steps.

2. CICS verifies that *authenticatedUserid* is authorized to the CICS region. This check is made by calling RACF to verify that *authenticatedUserid* has READ access to profile *grApplid* in class APPL.

**Important:** If a profile is not found in the APPL class that matches this value, CICS assumes that this check is not required.

3. CICS verifies that *authenticatedUserid* is authorized to the terminal. This check is made by calling RACF to verify that *authenticatedUserid* has READ access to resource profile *termAppl* in class TERMINAL.

**Important:** If a profile is not found in the TERMINAL class that matches this value, CICS assumes that this check is not required.

**Important:** Although transaction *signOn* successfully authenticates the credentials of *authenticatedUserid*, the identity that is used for any security checks does not change until the transaction ends.

## 3270 terminal sign-off

Signing off from a 3270 terminal has security implications.

In the following diagram, a user who has been signed onto a CICS 3270 terminal by using *authenticatedUserid* starts to sign off from the terminal. For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

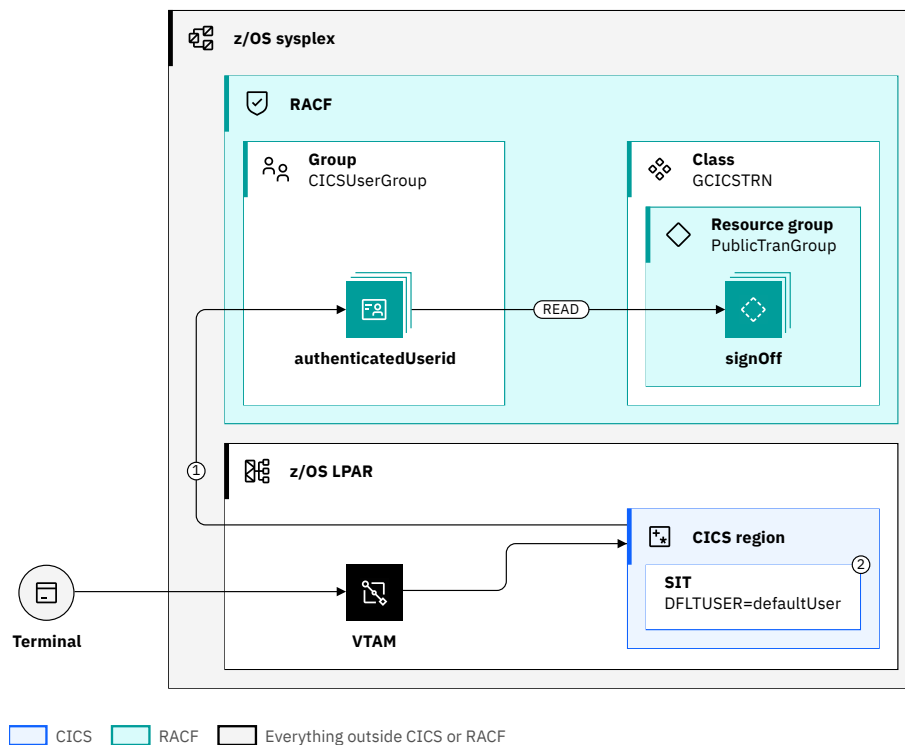


Figure 153. 3270 terminal sign off process

The following security checks are conducted during the sign-off:

1. The user starts transaction *signOff* and CICS verifies that *authenticatedUserid* is authorized to run *signOff*. This check is made by calling RACF to verify that *authenticatedUserid* has READ access to resource *signOff* in group resource class GCICSTRN.

If *signOff* is the CICS Category 3 transaction CESF, this check is not undertaken.

Transaction *signOff* can also disconnect the terminal from CICS, for example, by use of **EXEC CICS  
ISSUE DISCONNECT**.

2. Credentials for *authenticatedUserid* are removed from the terminal so that it reverts to being assigned the region's default user ID *defaultUser*.

**Important:** Although transaction *signOff* successfully removes the credentials of *authenticatedUserid*, the identity that is used for any security checks does not change until the transaction ends.

## Designing 3270 terminal security

---

When you design terminal security for a CICS Region, consider the implications for authentication and identification, authorization, confidentiality and integrity, and audit.

- [Authentication and identification](#)
- [Authorization](#)
- [Confidentiality and integrity](#)
- [Audit](#)

For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

### Authentication and identification

#### Are users of a terminal required to be authenticated?

##### Yes

A user authenticates to the CICS region by use of a *signOn* transaction that verifies their credentials by calling RACF to validate their validity. *DefaultUser* requires authority to start transaction *signOn* and any authenticated user requires authority to start any *signOff* transaction.

##### No

A terminal has a *presetID* specified on the terminal definition that cannot be changed by use of a *signOn* transaction. This option would be used only for physical devices, such as printers. Activity on such a device cannot be attributed to an individual user so the authority of *presetID* must be carefully considered.

### Authorization

#### Do you want to control which users can access the CICS region?

If the region specifies a Generic VTAM application ID, then configure a profile in the RACF APPL class that matches that ID. Otherwise, configure the RACF APPL class to use the VTAM application ID. This configuration gives the ability to restrict users of the region. If RACF indicates that a matching profile does not exist, CICS allows any user to authenticate.

#### Do you want to control which users can access terminals?

To restrict users of individual terminals, configure a profile in the RACF TERMINAL class that matches the terminals' Logical Unit application IDs. If RACF indicates that a matching profile does not exist, CICS allows any user.

For physical devices that do not use preset user IDs, this facility allows you to restrict which users can use which 3270 terminals. An example is where only terminals in a specific location can be used by any user while certain users can use any terminal.

For virtual devices such as TN3270 clients, a dependence exists on the TN3270 services that control allocation of Logical Unit values. For instance, you can permit only a selected range of IP addresses to use a pool of LU values. Without this configuration, controlling access to terminals cannot be effectively implemented.

When you work on the design, it is important to note that profiles in the **TERMINAL** class affect terminal usage in other applications, for example, TSO.

## Confidentiality and integrity

Terminal emulators need to use TLS to encrypt the data that flows between the terminal and the TN3270 servers. Information on configuring the TN3270 security is available at [Connection security](#). Additionally, VTAM can be configured to encrypt data that flows on to CICS. For an introduction to security features in VTAM, see [Security features](#).

## Audit

Various CICS messages are issued reporting terminal authentication attempts and sign-offs.

```
DFHSN1100 03/02/2022 07:16:36 APPLID signOn Signon at netname termAppl TN3270 IP Address tnaddr  
by user userid in group groupid  
is complete after 1 failed attempts. User ID was last accessed on 03/02/2022 at 07:14:23.
```

```
DFHSN1200 03/02/2022 09:33:06 APPLID Signoff at netname portname by user userid is complete. 31  
transactions entered with 2 errors.
```

```
DFHSN1119 03/02/2022 06:55:02 APPLID Signon at netname portname by user userid has failed because  
the user is not authorized to use application applname.
```

```
DFHSN1118 03/02/2022 10:41:56 APPLID Signon at netname portname by user userid has failed because  
the user is not authorized to use the terminal.
```

## Design example: Dynamically defined 3270 terminal connects and authenticates to a CICS region

In this scenario, a currently unknown 3270 device connects to a CICS terminal-owning region. It is dynamically defined by CICS Terminal Autoinstall and the user then authenticates their credentials. It is assumed that *defaultUser* is permitted to the region by reading the access to resource *grApplid* in the APPL class.

For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

For instructions about how to configure for this scenario, see [Configuration example: Dynamically defined 3270 terminal connects and authenticates to a CICS region](#).

The following diagram shows the security checks involved in the scenario:

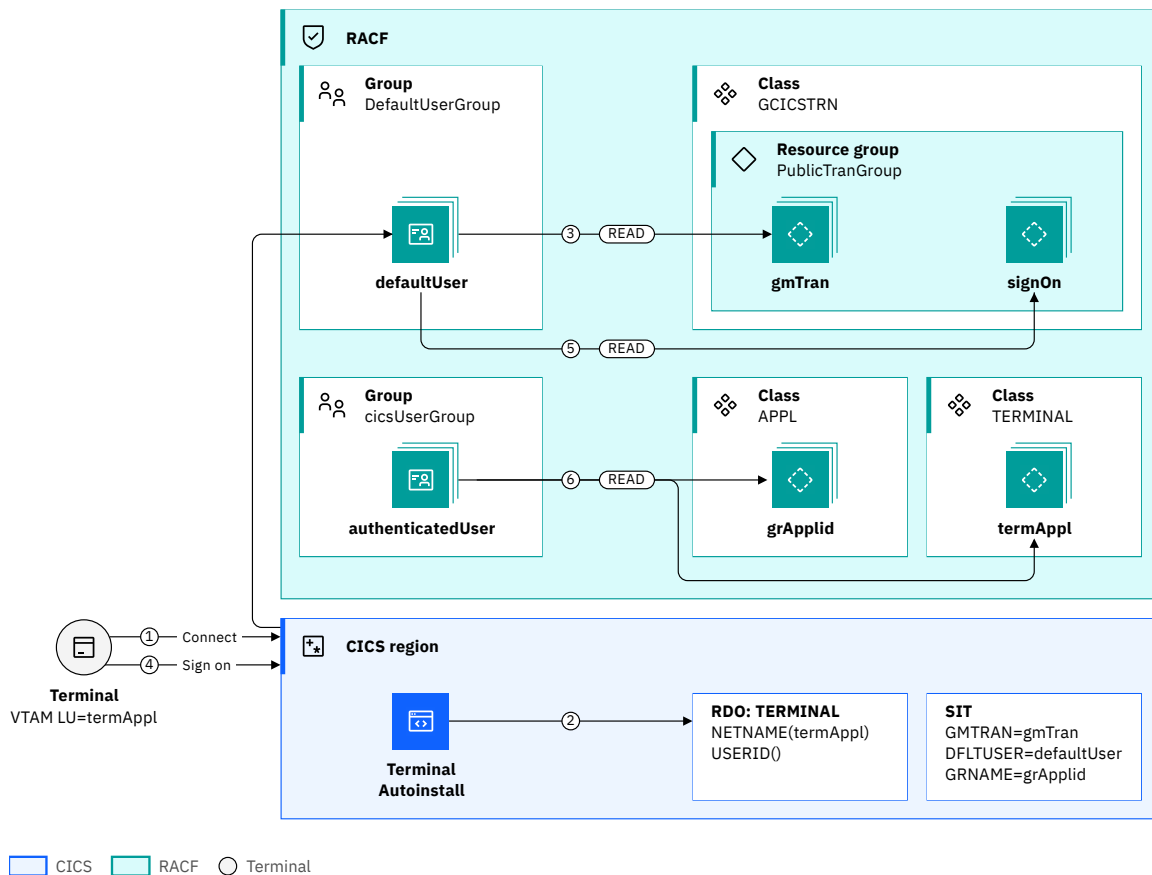


Figure 154. 3270 dynamically defined 3270 terminal connects and authenticates to a CICS region

1. A 3270 terminal user initiates a connection to the CICS region from a terminal that uses LU *termAppl*.
2. Because the terminal is not defined in the region, terminal autoinstall is started to dynamically define the terminal. The definition is created with a blank user ID that causes *defaultUser* to be assigned.
3. With the terminal now connected to CICS, the Good Morning Transaction, *gmTran*, is started on the terminal. CICS then check whether *defaultUser* has READ access to member list *PublicTranGroup* that includes *gmTran*, in class *GCICSTRN*.

**Note:** If *gmTran* is the CICS supplied CSGM transaction or another CICS supplied sign-on transaction (CESN or CESL), no security check is undertaken because they are Category 3 transactions.

4. The terminal user starts transaction *signOn* to authenticate, supplying the credentials for *authenticatedUser*.
5. CICS checks whether *defaultUser* has READ access to member list *PublicTranGroup* that includes *signOn*, in class *GCICSTRN*.

**Note:** If *signOn* is one of the CICS supplied sign-on transactions (CESN or CESL), no security check is undertaken because they are Category 3 transactions.

6. CICS authenticates the supplied credentials and then checks whether *authenticatedUser* has READ access to resource profiles *termAppl* in class *TERMINAL* and *grApplid* in class *APPL*.

**Note:** In the example, the transactions *gmTran* and *signOn* are shown as distinct transactions. However, these transactions are usually the same transaction in a CICS environment when steps 4, 5 and 6 are merged.

### Related information

[Security for 3270](#)

## Configuration example: Dynamically defined 3270 terminal connects and authenticates

Configure the security profiles required for a 3270 terminal to connect to CICS and for the user to then authenticate.

### Before you begin

This configuration task is based on [Design example: Dynamically defined 3270 terminal connects and authenticates to a CICS region](#).

It is assumed that *defaultUser* is permitted access to the region. This access is achieved in that the user is given read access to resource *grApplid* in the APPL class.

You must have authorization to define and manipulate RACF resource access profiles in classes APPL, GCICSTRN, and TERMINAL.

### About this task

In this example, you learn how to define and set the required permissions in RACF to enable a user to connect a 3270 terminal to a CICS region and authenticate.

The task assumes the following definitions:

- *grAppl* is the Generic VTAM application identifier that is used by the CICS region and specified by SIT parameter GRNAME. If the region does not use this option, the region's VTAM application identifier that is specified by SIT parameter APPLID is used.
- *defaultUser* is the default RACF user of the CICS region that is specified by SIT parameter DFLTUSER, which is a member of RACF user group *defaultUserGroup*.
- *publicTranGroup* is the name of a RACF CICS transaction resource group for CICS Transactions usable by any user.
- *gmTran* is the CICS Good Morning Transaction, started by CICS on a terminal when it connects. The *gmTran* is specified by SIT parameter GMTRAN. If this transaction is the default CICS supplied transaction CSGM, CICS automatically allows any user to run it and no access needs to be granted in RACF.
- *signOn* is the transaction that is used by a user to authenticate with the CICS region. If this transaction is either of the default CICS supplied transaction CESN or CESL, CICS automatically allows any user to run it and no access needs to be granted in RACF.
- *termAppl* is the VTAM Logical Unit identifier for the 3270 terminal that connects to CICS.
- *authenticatedUser* is the RACF user ID that is used to authenticate to CICS.
- *cicsUserGroup* is the RACF user group that contains *authenticatedUser*.

### Procedure

1. Define a resource group to protect the Good Morning Transaction allowing all users access and refresh the definitions by refreshing the associated resource class.

**Important:** Any user, including unauthenticated ones, can run transactions in this group. Careful review of a transaction's actions must be undertaken before you add users to this group.

```
REDFINE publicTranGroup GCICSTRN UACC(READ) ADDMEM(gmTran)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

These commands create the definitions to reach step 4 in the [design example](#).

2. Add transaction *signOn* to the *publicTranGroup* and refresh RACF. This group is already available to all users.

```
RALTER GCICSTRN publicTranGroup ADDMEM(signOn)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

These commands create the definitions to reach step 6 in the [design example](#).

3. Protect the terminal *termAppl* by granting the *authenticatedUser* access and refresh RACF.

```
REDFINE TERMINAL termAppl UACC(NONE)
PERMIT termAppl CLASS(TERMINAL) ID(authenticatedUser)
SETROPTS RACLIST(TERMINAL) REFRESH
```

4. Grant the group that is used by authenticated users access to the CICS region and refresh RACF.

```
PERMIT grApplid CLASS(APPL) ID(cicsUserGroup)
SETROPTS RACLIST(APPL) REFRESH
```

## Results

With a correctly configured environment, when a user authenticates, CICS issues the following message:

```
DFHSN1100 30/08/2022 11:59:06 APPLID signOn Signon at netname termAppl TN3270 IP Address tnaddr
by user authenticatedUser in group cicsUserGroup
is complete after 1 failed attempts. User ID was last accessed on 30/08/2022 at 11:57:02.
```

## Design example: 3270 terminal with a preset user ID connects

In this scenario, a 3270 device that is defined to CICS with a preset user ID of *presetID* connects to a CICS terminal-owning region (TOR).

For more information about 3270 terms used in this topic, such as the *signOn* transaction, *defaultUser*, and so on, see [How it works: 3270 security](#).

For instructions about how to configure for this scenario, see [Configuration example: 3270 terminal with a preset user ID connects](#).

The following diagram shows the security checks involved in the scenario:



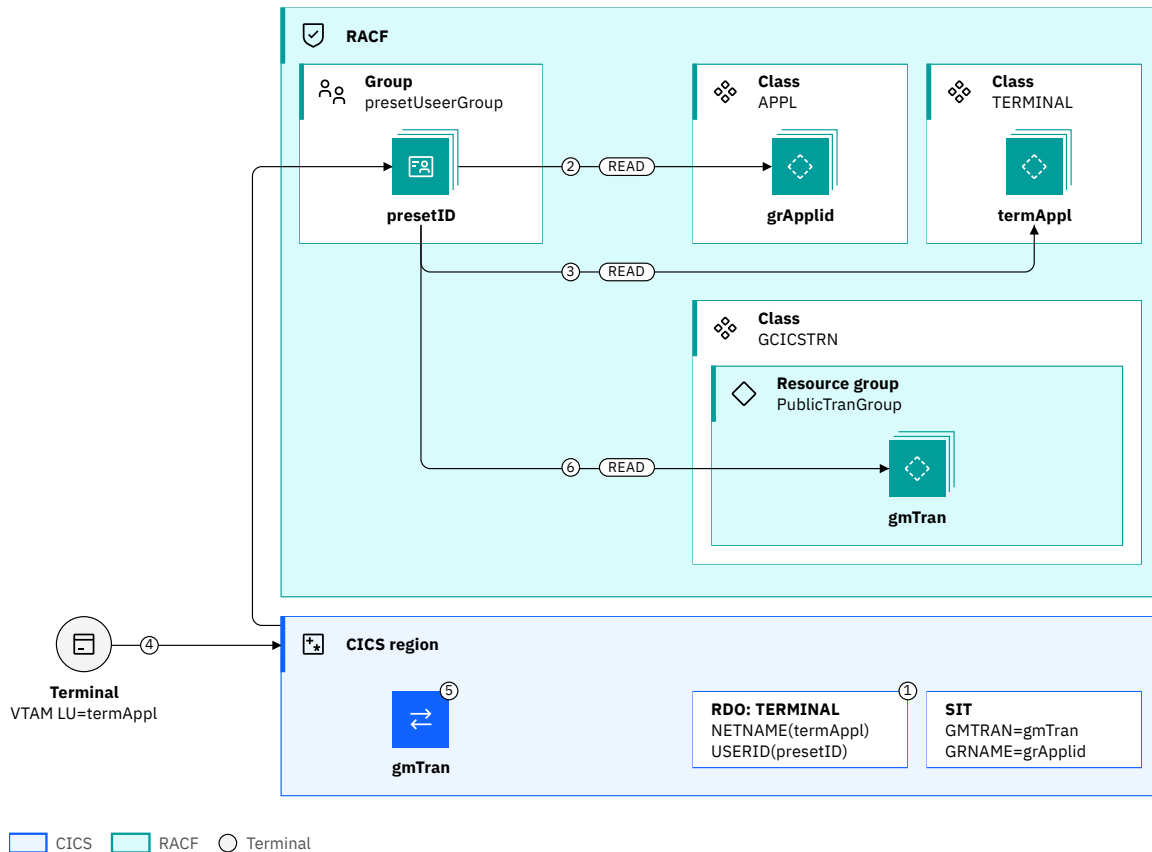


Figure 155. Connection flow for 3270 with a preset user ID

1. The terminal resource is installed in the CICS region, usually during CICS initialization with a fixed LU name of *termAppl* and user ID of *presetID*.
2. CICS checks that *presetID* is permitted to the region by calling RACF to check whether *presetID* has READ access to resource *grApplid* in the APPL class.
3. CICS calls RACF to check that *presetID* has READ access to resource *termAppl* in class TERMINAL.
4. A 3270 terminal user initiates a connection to the CICS region from a terminal that uses LU *termAppl*.
5. CICS matches the terminal to the existing definition and starts the Good Morning transaction, *gmTran*, on the terminal.
6. CICS calls RACF to check that *presetID* has READ access to resource *gmTran* in class GCICSTRN.

**Note:** If *gmTran* is the CICS supplied CSGM transaction, no security check is undertaken because it's a Category 3 transaction.

## Related information

[Security for 3270](#)

## Configuration example: 3270 terminal with a preset user ID connects

Configure the security profiles required for a 3270 terminal to connect to CICS and for the user to then authenticate.

## Before you begin

This configuration task is based on [Design example: 3270 terminal with a preset user ID connects](#).

You must have authorization to define and manipulate RACF resource access profiles in classes APPL, GCICSTRN, and TERMINAL.

## About this task

You learn how to define and set the required permissions in RACF to enable a user to connect a 3270 terminal that is defined with a specific user ID to a CICS region.

The task assumes the following definitions:

- *grAppl* is the Generic VTAM application identifier that is used by the CICS region and specified by SIT parameter GRNAME. If the region does not use this configuration, the region's VTAM application identifier that is specified by SIT parameter APPLID is used.
- *publicTranGroup* is the name of a RACF CICS transaction resource group for CICS Transactions usable by any user.
- *gmTran* is the CICS Good Morning Transaction, which is started by CICS on a terminal when it connects, and specified by SIT parameter GMTRAN. If this transaction is the default CICS supplied transaction CSGM, CICS automatically allows any user to run it and no access needs to be granted in RACF.
- *termAppl* is the VTAM Logical Unit identifier for the 3270 terminal that connects to CICS.
- *presetID* is the RACF user ID that is specified on the CICS terminal.
- *presetUserGroup* is the RACF user group that contains *presetID*.

If you do not want to restrict users access to the CICS region, step 1 is not needed but you must ensure that no generic profiles in the APPL class include *grApplid*.

If you do not want to restrict users access to terminals, step 2 is not needed but you must ensure no generic profiles in the TERMINAL or associated group class GTERMINL include *termAppl*.

## Procedure

1. Grant access to the group that contains *presetID* and refresh the RACF in memory definitions.

```
PERMIT grApplid CLASS(APPL) ID(presetUserGroup)  
SETROPTS RACLIST(APPL) REFRESH
```

These commands create the definitions to reach step 2 in the [design example](#).

2. Protect the terminal *termAppl*, grant *presetID* access, and refresh RACF.

```
REDDEFINE TERMINAL termAppl UACC(NONE)  
PERMIT termAppl CLASS(TERMINAL) ID(presetID)  
SETROPTS RACLIST(TERMINAL) REFRESH
```

These commands create the definitions to reach step 3 in the [design example](#).

3. Define a resource group to protect the Good Morning Transaction allowing all users access and refresh the definitions by refreshing the associated resource class.

```
REDDEFINE publicTranGroup GCICSTRN UACC(READ) ADDMEM(gmTran)  
SETROPTS RACLIST(TCICSTRN) REFRESH
```

These commands create the definitions to reach step 4 in the [design example](#).

## Results

When a terminal with a preset user ID is installed into a CICS region, the following message is issued by CICS. No signon or security-related messages are issued when the terminal connects to the CICS region.

```
DFHSN1800 03/25/2022 15:48:38 applid Signon at netname termAppl by preset user presetID in group presetUserGroup is complete.
```

## Configuring 3270 terminal security

---

The design examples offer options for recommended security configurations for 3270 terminals in CICS. Each design example has a matching configuration example that defines the steps needed to implement the security for that example.

Design examples for 3270 terminals are described here:

- [Design example: Dynamically defined 3270 terminal connects and authenticates to a CICS region](#)

In this scenario, a currently unknown 3270 device connects to a CICS terminal-owning region. It is dynamically defined by CICS Terminal Autoinstall and the user then authenticates their credentials. It is assumed that *defaultUser* is permitted to the region by reading the access to resource *grApplid* in the APPL class.

- [Design example: 3270 terminal with a preset user ID connects](#)

In this scenario, a 3270 device that is defined to CICS with a preset user ID of *presetID* connects to a CICS terminal-owning region (TOR).

The matching configuration examples are described here, as children of the corresponding design examples:

- [Configuration example: Dynamically defined 3270 terminal connects and authenticates to a CICS region](#)
- [Configuration example: 3270 terminal with a preset user ID connects](#)



---

## Chapter 18. Security for CICSplex SM

This part describes how to implement security for CICSplex SM.

### Security planning for CICSplex SM

---

CICSplex SM uses a SAF-compliant external security manager, such as RACF to prevent unauthorized access to CICSplex SM functions and CICS resources, and to control the simulation of CICS command checking and CICS resource checking.

In both cases, security checking is handled by the CMASs managing the CICS systems that are the target of any request to access a resource. For example, if a CICSplex is managed by two CMASs, and a request is made to access a resource in all CICS systems belonging to that CICSplex, the security check is performed in both CMASs.

To activate security checking, you must modify the JCL used to start the CMAS or its managed CICS systems. If security checking is switched off for the CICS system, no checking occurs, regardless of the CMAS setting. However, if security checking is switched off for the CMAS but switched on for the CICS system, the CICS system is not able to connect to the CMAS.

Begin by deciding how much security checking you need. In particular, identify those users who need access to CICSplex SM, and ensure that an individual user has the same user ID across all systems on which a CMAS is installed. The user ID against which the security check is performed is the RACF ID that has been used to sign on to CICSplex SM. Consider also the type of security checking you want to implement.

See [Implementing CICSplex SM security](#) for more information about how to set up CICSplex SM security.

### Protecting access to CICSplex SM functions and CICS resources

To prevent unauthorized access, you create security profiles for combinations of CICSplex SM functions, and CICS resources that are to be protected. In most cases, the security provided by CICSplex SM security profiles is adequate.

RACF is also used to protect CICSplex SM libraries, procedures and Web User Interface resources. Full details of how to protect CICSplex SM libraries and procedures are provided in [Implementing CICSplex SM security](#). In order to protect Web User Interface views, menus, help information and the View Editor, you need to create an appropriate profile in the FACILITY class. See [Controlling access to Web User Interface resources](#) for more information.

### Special considerations for BAS

Take special care in the protection of the BAS views, so that unauthorized users cannot create and administer resources. The equivalent in RDO terms is leaving your CSD unprotected.

Also take care if you use the **EXEC CICS CREATE** command to build new resources. Any definition created with the CICSplex as the context is automatically distributed to all CMASs in the CICSplex. Therefore, giving a user authority to create BAS objects is equivalent to giving authority to install resources on any CICS system in the CICSplex. When the CICS system starts, there is no check on who installed the resource in the system.

### CICS command and resource checking

CICS command and resource checking is simulated by CICSplex SM in the CMASs to which a request is directed. This allows you to protect CICS systems that do not support your external security manager. It also allows for a level of consolidation of your security checking.

Determine where CICS resource and command checking is in effect, and decide whether it needs to be retained along with other CICSplex SM security checking.

## Implementing CICSplex SM security

---

To implement CICSplex SM security using RACF, you must determine who needs access to the various CICSplex SM functions. You must also perform a number of tasks to define CICSplex SM class names and resource names, as well as activating security and refreshing RACF profiles.

### About this task

If you are using a SAF-compliant external security manager (ESM) other than RACF, refer to [“Invoking a user-supplied external security manager”](#) on page 447. The following steps summarize how to set up security for CICSplex SM. Each of the steps are explained in detail in the subsequent topics.

### Procedure

1. Decide who needs access to CICSplex SM.
2. Review the general security requirements for CICSplex SM.
3. Create RACF profiles for the CICSplex SM data sets.
4. Define the CICSplex SM started tasks to RACF.
5. If CICS transaction security is active in a CMAS, define the CICSplex SM transactions to RACF.
6. If CICS transaction security is active in a MAS, define the CICSplex SM transactions to RACF.
7. Create RACF profiles for the CICSplex SM views.
8. Create RACF profiles for the CICSplex SM Web User Interface resources.  
See [Controlling access to Web User Interface resources](#) for more information.
9. Optional: Activate simulated security checking using the CICSSYS, CPLEXDEF, or MAS views.
10. Activate security in the CMASs and MASs using the CICSplex SM and CICS security-related system initialization parameters.

## Determining who requires access to CICSplex SM resources

To determine who requires access to the CICSplex SM resources, answer the questions and complete the matrix. You can then use the results to create the PERMIT statements that are required in RACF to control access to the resources.

### About this task

You can control access to CICSplex SM resources in two ways:

- By restricting access to the objects managed by CICSplex SM views. This restriction does not affect access to the views themselves, but it prevents them from displaying any data.
- By restricting access to Web User Interface view sets, menus and the View Editor. This restriction does not affect access to the objects that are being managed but prevents access to the view sets, menus, and View Editor themselves.

### Procedure

1. Answer the following questions to determine who requires access to the CICSplex SM resources:

#### What groups of users use CICSplex SM?

Your enterprise probably already has several user groups that are defined to RACF. The groups that typically require access to CICSplex SM include systems programming, operations, the help desk, applications programming, and performance monitoring. These groups are used as column headings in the security matrix. You can supply their corresponding RACF group IDs. (If necessary, you can ignore, replace, or add groups to the matrix required for your enterprise.)

#### Which CICSplex SM views do each group need to use?

CICSplex SM manages CICS resources that use views. Views are grouped by function: configuration, topology, workload management, real-time analysis, operations, monitoring, business application services, and CICSplex management. Not all view groups are appropriate

for all users. Certain groups of users will only use a subset of views. For example, the systems programming group might need to work with all views, while the help desk group might only need to use one or two. The view groups are listed vertically on the left side of the matrix, along with the high-level qualifier of their CICSplex SM resource names.

#### What type of access does each RACF group need?

After you decide who needs to use what, stop universal access to all of the objects managed by all of the views. You can then selectively permit read, update, or alter access to specific view groups. To complete the matrix, specify READ, UPDATE, or ALTER access for each RACF group that needs access to a group of views.

- Specify READ access to allow a user to inquire on a resource.
- Specify UPDATE access to allow a user to change a value, by using the SET or UPDATE command, or perform an action. The user can also create or remove a definition, such as a BAS resource object.
- Specify ALTER access to allow a user to discard an installed resource from CICS and allow a user to install a BAS resource object.

**Tip:** For application programmers, if you need to control who is allowed to use the CPSM option on the CICS translator, you can use RACF to control who is allowed to load the DFHSMTAB table at translation time. For a description of RACF program control, see the [z/OS Security Server RACF Security Administrator's Guide](#). DFHSMTAB is the language definition table that defines the CICSplex SM API commands. It is loaded only on demand.

#### Which CICSplex SM Web User Interface views, menus will each group need access to?

Web User Interface views and menus are usually user-defined but like Web User Interface views are most likely grouped by functionality. Not all view sets and menus are appropriate for all users. Certain groups of users require access to a subset of views. For example, the systems programming group might require access to all views and to the View Editor, while the help desk group might not need to use the View Editor or those views that manage the definition of CICSplex SM resources.

2. Fill out the security matrix when you have answered the questions.

Table 27. Security matrix					
RACF group → CICSplex SM view group ↓	System Programming ID( )	Operations ID( )	Help Desk ID( )	Application Programming ID( )	Performance ID( )
Configuration CONFIG					
Topology TOPOLOGY					
Workload Management WORKLOAD					
Real-Time Analysis ANALYSIS					
Operations OPERATE					
Monitor MONITOR					

Table 27. Security matrix (continued)					
RACF group → CICSplex SM view group ↓	System Programming ID( )	Operations ID( )	Help Desk ID( )	Application Programming ID( )	Performance ID( )
Business Application Services BAS					
CSD					

Table 28 on page 404 is a sample of a completed security matrix for a production CICSplex:

Table 28. Sample security matrix					
RACF group → CICSplex SM view group ↓	System Programming ID(SYSPGRP)	Operations ID(OPSGRP)	Help Desk ID(HELPGRP)	Application Programming ID(APPLGRP)	Performance ID(PERFGRP)
Configuration CONFIG	UPDATE				
Topology TOPOLOGY	UPDATE	UPDATE	READ		
Workload Management WORKLOAD	UPDATE			READ	
Real-Time Analysis ANALYSIS	UPDATE	UPDATE	READ		READ
Operations OPERATE	ALTER	UPDATE	READ	READ	READ
Monitor MONITOR	UPDATE	READ			READ
Business Application Services BAS	ALTER	ALTER		UPDATE	
CSD	ALTER	ALTER		UPDATE	

3. Ensure that the CPSMOBJ class is active and that generic profiles can be defined:

```
SETROPTS CLASSACT(CPSMOBJ)
SETROPTS GENERIC(CPSMOBJ)
SETROPTS GENCMD(CPSMOBJ)
```

4. Create a RACF profile to protect all of the views and action commands for all CICSplex SM functions:

```
RDEF CPSMOBJ ** UACC(NONE) OWNER(admin_group) NOTIFY(admin_user)
```



CPSMOBJ is the CICSplex SM member class. The double asterisks indicate that all of the CICSplex SM views are included in this RDEF statement.

5. Using the information in the sample matrix, you can permit access to the specific view groups. For example, the systems programming group requires UPDATE access to all of the view groups and ALTER access to the OPERATE, BAS, and CSD views.

To start, allow UPDATE access to all possible profiles:

```
PERMIT ** CLASS(CPSMOBJ) ID(SYSPGRP) ACCESS(UPDATE)
```

The double asterisks indicate that all of the CICSplex SM views are affected by this PERMIT statement.

The following PERMIT statements grant the appropriate access to all of the topology views for the operations and help desk groups:

```
PERMIT TOPOLOGY.** CLASS(CPSMOBJ) ID(OPSGRP) ACCESS(UPDATE)
PERMIT TOPOLOGY.** CLASS(CPSMOBJ) ID(HELPGRP) ACCESS(READ)
```

For the workload management views:

```
PERMIT WORKLOAD.** CLASS(CPSMOBJ) ID(APPLGRP) ACCESS(READ)
```

For the real-time analysis views:

```
PERMIT ANALYSIS.** CLASS(CPSMOBJ) ID(OPSGRP) ACCESS(UPDATE)
PERMIT ANALYSIS.** CLASS(CPSMOBJ) ID(HELPGRP) ACCESS(READ)
PERMIT ANALYSIS.** CLASS(CPSMOBJ) ID(PERFGRP) ACCESS(READ)
```

For the operations views:

```
PERMIT OPERATE.** CLASS(CPSMOBJ) ID(SYSPGRP) ACCESS(ALTER)
PERMIT OPERATE.** CLASS(CPSMOBJ) ID(OPSGRP) ACCESS(UPDATE)
PERMIT OPERATE.** CLASS(CPSMOBJ) ID(HELPGRP) ACCESS(READ)
PERMIT OPERATE.** CLASS(CPSMOBJ) ID(APPLGRP) ACCESS(READ)
PERMIT OPERATE.** CLASS(CPSMOBJ) ID(PERFGRP) ACCESS(READ)
```

For the monitor views:

```
PERMIT MONITOR.** CLASS(CPSMOBJ) ID(APPLGRP) ACCESS(READ)
PERMIT MONITOR.** CLASS(CPSMOBJ) ID(PERFGRP) ACCESS(READ)
```

For the business application services views:

```
PERMIT BAS.** CLASS(CPSMOBJ) ID(SYSPGRP) ACCESS(ALTER)
PERMIT BAS.** CLASS(CPSMOBJ) ID(OPSGRP) ACCESS(ALTER)
PERMIT BAS.** CLASS(CPSMOBJ) ID(APPLGRP) ACCESS(UPDATE)
```

For the CSD views:

```
PERMIT CSD.** CLASS(CPSMOBJ) ID(SYSPGRP) ACCESS(ALTER)
PERMIT CSD.** CLASS(CPSMOBJ) ID(OPSGRP) ACCESS(ALTER)
PERMIT CSD.** CLASS(CPSMOBJ) ID(APPLGRP) ACCESS(UPDATE)
```

## Results

For simplicity, these PERMIT statements grant access to broad groups of views by using the double asterisks in the resource names. However, if required, you can use more specific resource names in your PERMIT statements. Refer to [“Specifying CICSplex SM resource names in profiles” on page 416](#) for details.

## What to do next

Using your own completed security matrix and the information in the remainder of this section, you can create as many profiles as required for your enterprise. [“Example RACF commands for securing CICSplex SM” on page 448](#) provides detailed profile examples.

# General requirements for CICSplex SM security

Review your RACF configurations to ensure that your systems meet the minimum requirements.

- The IDs for all users expected to use CICSplex SM must be defined to RACF in each MVS system in which there is a CMAS. For each individual user, the ID must be the same for each MVS system.
- User access authority to CICSplex SM definitions and CICS commands and resources must be defined to RACF in a consistent manner in all MVS systems used by CICSplex SM.

In addition, in the CMAS address space, a security environment is created for the user specified in the **DFLTUSER** system initialization parameter associated with the MAS.

## Related concepts

[How it works: Identification in CICS](#)

[How it works: Securing CICS with RACF](#)

# Creating profiles for the CICSplex SM data sets

You should restrict access to CICSplex SM data sets using RACF data set protection.

## Procedure

1. Prohibit universal access by specifying UACC(NONE).
2. Ensure that minimum access to the data sets is authorized for the RACF USERID assigned to each of the following:
  - Every CMAS job or started task.
  - Every MAS.
  - All individuals allowed to use CICSplex SM from the CICSplex SM WUI and API (both system administrators and users).

Table 29 on page 406 lists the CICSplex SM data sets and the minimum access that should be granted to each type of user ID.

Table 29. Access by user ID for CICSplex SM data sets

Data set name	CMAS	MAS	System Admin.	Individual User
<div>6.2</div> SYS1.CICSTS62.CPSM.SEYULPA	NONE	READ	UPDATE	NONE
<div>6.1</div> SYS1.CICSTS61.CPSM.SEYULPA				
<div>6.2</div> SYS1.CICSTS62.CPSM.SEYULINK	READ	NONE	UPDATE	NONE
<div>6.1</div> SYS1.CICSTS61.CPSM.SEYULINK				
<div>6.2</div> SYS1.CICSTS62.CPSM.SEYUAUTH	READ	READ	UPDATE	READ
<div>6.1</div> SYS1.CICSTS61.CPSM.SEYUAUTH				

Table 29. Access by user ID for CICSplex SM data sets (continued)

Data set name	CMAS	MAS	System Admin.	Individual User
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYULOAD SYS1.CICSTS61.CPSM.SEYULOAD	READ	READ	UPDATE	NONE
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUPARM SYS1.CICSTS61.CPSM.SEYUPARM	READ	READ	UPDATE	NONE
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUCOB SYS1.CICSTS61.CPSM.SEYUCOB	NONE	NONE	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUC370 SYS1.CICSTS61.CPSM.SEYUC370	NONE	NONE	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUDEF SYS1.CICSTS61.CPSM.SEYUDEF	READ	READ	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUCLIB SYS1.CICSTS61.CPSM.SEYUCLIB	NONE	NONE	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CICS.SDFHINST SYS1.CICSTS61.CICS.SDFHINST	NONE	NONE	UPDATE	NONE
CICSTS62.CPSM.SEYUMAC	NONE	NONE	UPDATE	READ
CICSTS62.CPSM.SEYUPL1	NONE	NONE	UPDATE	READ

Table 29. Access by user ID for CICSplex SM data sets (continued)

Data set name	CMAS	MAS	System Admin.	Individual User
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUPROC SYS1.CICSTS61.CPSM.SEYUPROC	NONE	NONE	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.SEYUSAMP SYS1.CICSTS61.CPSM.SEYUSAMP	NONE	NONE	UPDATE	READ
<div>6.2</div> <div>6.1</div> SYS1.CICSTS62.CPSM.EYUDREP SYS1.CICSTS61.CPSM.EYUDREP	UPDATE	NONE	UPDATE	NONE

## What to do next

If you require more details about RACF data set protection, see the [z/OS Security Server RACF Security Administrator's Guide](#).

## Determining the agent user ID for CICSplex SM components

You need to know the user ID that is used by components of CICSplex SM, such as the CMAS, MAS, WUI, and SMSS so that you can set up the right authorization in RACF.

### What governs the agent user ID?

The agent user ID depends on the following factors:

- The release of CICS, but irrespective of the release of CICSplex SM
- How the agent was initialized.
- Whether the **PLTPIUSR** system initialization parameter is set (to initialize the agent at the initialization of the CICS region)

The agent user IDs for the CMAS, MAS, or WUI run under the **PLTPIUSR** user ID only when the SIT parameter **SEC=YES** and at least one of the following SIT parameter values are set. Otherwise, the agent user ID continues to run under the CICS region user ID.

- **PLTPI=YES** or **PLTPI=name**
- **MQCONN=YES**
- **DB2CONN=YES**
- **DBCTLCON=YES**

For CMAS and WUI regions, use the **CPSMCONN** SIT parameter instead of the **PLTPI** SIT parameter. Do not specify the **MQCONN=YES**, **DB2CONN=YES**, or **DBCTLCON=YES** SIT parameters in these regions.

For details of SIT parameters, see [System initialization parameter descriptions and summary](#).

Use the tables to determine which ID is used:

- [CMAS agent user ID](#)
- [MAS agent user ID](#)
- [WUI agent user ID](#)

<i>Table 30. CMAS agent user ID</i>			
<b>CICSplex SM release</b>	<b>How was the CMAS agent initialized?</b>	<b>PLTPIUSR in system initialization table?</b>	<b>CMAS agent user ID</b>
Any in-service release	At initialization of the CICS region (CPSMCONN only)	Not applicable (no effect)	CICS region user ID
5.5 or earlier	At initialization of the CICS region (PLT entry only)	No	CICS region user ID
5.5 or earlier	At initialization of the CICS region (PLT entry only)	Yes	PLTPIUSR (see details in <a href="#">“What governs the agent user ID?”</a> on page 408)

<i>Table 31. MAS agent user ID</i>			
<b>CICSplex SM release</b>	<b>How was the MAS agent initialized?</b>	<b>PLTPIUSR in system initialization table?</b>	<b>MAS agent user ID</b>
Any in-service release	At initialization of the CICS region (CPSMCONN only)	Not applicable	CICS region user ID
Any in-service release	Using the COLM transaction	Not applicable	CICS region user ID
5.5 or earlier	At initialization of the CICS region (PLT entry only)	No	CICS region user ID
5.5 or earlier	At initialization of the CICS region (PLT entry only)	Yes	PLTPIUSR (see details in <a href="#">“What governs the agent user ID?”</a> on page 408)

<i>Table 32. WUI agent user ID</i>			
<b>CICSplex SM release</b>	<b>How was the WUI agent initialized?</b>	<b>PLTPIUSR in system initialization table?</b>	<b>WUI agent user ID</b>
Any in-service release	At initialization of the CICS region (CPSMCONN only)	Not applicable	CICS region user ID
Any in-service release	Using the COVC transaction	Not applicable	Signed-on user ID that initiated COVC
5.5 or earlier	At initialization of the CICS region (PLT entry only)	No	CICS region user ID
5.5 or earlier	At initialization of the CICS region (PLT entry only)	Yes	PLTPIUSR (see details in <a href="#">“What governs the agent user ID?”</a> on page 408)

## Defining the CICSplex SM started tasks

When you run a CMAS as a started task, you must associate the appropriate procedure names with a suitably authorized USERID.

### About this task

This is usually achieved using the STARTED general resource class or the RACF ICHRIN03 tables. The names of the associated USERIDs need not match the names of the procedures. Each USERID must have the appropriate level of access to all of the data sets referenced in the cataloged procedures.

For additional information about the STARTED class, see the [z/OS Security Server RACF Security Administrator's Guide](#). For more information about ICHRIN03, see the [z/OS Security Server RACF System Programmer's Guide](#).

**Note:** If the USERID and group name that you assign are not defined to RACF, the started tasks will execute with only the limited authority of an undefined user. In this case, the address space will be able to access protected resources only if the universal access authority (UACC) for the resource is sufficient to allow the requested operation.

## Defining the CICSplex SM transactions in a CMAS

If transaction–attach security is active in a CMAS (that is, SEC=YES and XTRAN=YES|*classname* are specified in the system initialization parameters), you must define to RACF the CICSplex SM transactions that run in a CMAS.

### Procedure

1. You must define the following CICSplex SM transactions to RACF:

BMLT	LPLT	PRLT	WMWC
LCPP	LPRT	PRPR	WMWD
LCMU	LPSC	PSLT	WMWT
LECI	LPSM	TICT	WSCL
LECR	LRLT	TIRT	WSLW
LECS	LSGT	TIST	XDBM
LEEI	LSRT	TSMH	XDNC
LEER	LWTM	TSPD	XDND
LEMI	MCCM	TSSC	XDNE
LEMS	MCTK	TSSJ	XDNR
LENS	MMEI	WMCC	XDNS
LMIR	MMIS	WMGR	XDSR
LNCI	MMST	WMLA	XLEV
LNCS	PEAD	WMQB	XLNX
LNMI	PELT	WMQM	XLST
LNMS	PMLT	WMQS	XMLT
LPDG	PNLT	WMSC	XQST
LPLK	PPLT		XZLT

A list of these transactions is also contained in the EYU\$CDEF member of SEYUSAMP sample library.

- a) Ensure that the CICS region user ID has authority to access these transactions.
  - If the CMAS starts by using the SIT parameter CPSMCONN, the CICS system initialisation parameter [SEC](#) is set to YES, a PLTPI has been specified and a user ID is specified on the [PLTPIUSR](#) system initialization parameter, ensure that the PLTPIUSR user ID has authority to attach these transactions.

- Otherwise ensure that the CICS region user ID has authority to attach these transactions.
  - b) Depending on the security attributes specified for any CMTCMDEF or CMTPMDEF, ensure that any user IDs that might flow from a connected CMAS have the authority to attach these transactions.
2. Define the following debugging transactions to RACF if transaction security is active, regardless of the CICS release that runs as the CMAS:

- CODB
- COD0
- COD1
- COD2
- COLU

These transactions are associated with a terminal and are supplied for debugging purposes under the guidance of IBM support personnel. Authority to initiate these transactions must be restricted to only those users who might become involved in working with IBM to resolve CICSplex SM problems.

3. Give users access to the CESD shutdown-assist transaction.
- Users who can attach CICSplex SM transactions or define debugging transactions need access to CESD in case of CMAS failure.
4. Allow only users who might need to shut down a CMAS to access the COSD transaction.
- The COSD transaction allows a terminal user to shut down a CMAS.

## Defining the transactions in a managed CICS region

For CICS regions that are capable of running with an external security manager, it might be necessary to define the transactions that run in a CICS region that is managed by CICSplex SM to the ESM.

### About this task

If CICSplex SM status probes (STATDEFs) run in a MAS, the region user ID must be given READ access to transaction CORT.

Users who might invoke the debugging transactions must be given READ access to the following transactions:

CODB  
COD0  
COD1  
COD2  
COLU

The security attributes of the CONNECTION/SESSION pair defined for the link to the CMAS define which users are authorized to run these transactions. See [Security for MRO](#) for information on intercommunication security.

The COSH transaction allows a terminal user to stop agent code execution. Access to this transaction must be restricted to those users who might need to stop the agent code in this way.

## Setting up CICSplex SM Web User Interface security

You can set Web User Interface security requirements for CICS security, Secure Sockets Layer (SSL) support, and access to MVS data sets.

### User security access summary

[Table 33 on page 412](#) summarizes the security accesses required by users of the Web User Interface.

Table 33. Security accesses required by users of the Web User Interface

User Roles	CICS Web Support	Administrator	User	View Editor
Transactions	COVP COVE COVU	COVG COVC	COVA	COVA
CICS surrogate user security		Yes		
View Editor profile				Yes
CICSplex SM and CICS security			As appropriate for individual users	As appropriate for individual users

## CICS security in your Web User Interface server region

If your Web User Interface server region is running with CICS security active, you must define the security access required for the CICS Web Support, by the administrator and by the users of the View Editor.

You can use CICS transaction security to limit the users who are allowed to control the Web User Interface server using the COVC transaction.

## Security access for the CICS Web Interface

If CICS transaction security is in use, the CICS DFLTUSER must be given access to the COVP, COVU, and COVE transactions.

## Security access for the administrator

The user ID that starts the Web User Interface (the terminal user of COVC or PLTPIUSR, if started automatically using PLTPI) must have access to the COVC and COVG transactions. If CICS surrogate user security checking is active in the Web User Interface server region, the user ID that started the Web User Interface (the terminal user of COVC or PLTPIUSR, if started automatically using PLTPI) must have READ access to wui-userid.DFHSTART in the SURROGAT class for all Web User Interface users.

## Security access for users of the View Editor

Users of the Web User Interface require access to the COVA transaction and CICSplex SM. Users of the View Editor require access to the COVA transaction, CICSplex SM, and the View Editor profile.

All users who are successfully signed on to the Web User Interface have access to all of the customizable view and menu help pages, if the customizable view and menu help is served by the Web User Interface.

## Secure Sockets Layer support

You can provide secure connections by using the Secure Sockets Layer (SSL) support to provide encryption on the connection. For information about SSL support, see [Web User Interface server initialization parameters](#) for information about the **TCPIPSSL** and **TCPIPSSLCERT** Web User Interface server initialization parameters that you must specify for SSL support and for more guidance on SSL, see [Configuring CICS to use SSL](#).

Web User Interface SSL support uses server authentication only. User authentication is by the external security manager (ESM) user ID and password.



## Defining the CICSplex SM transactions for a WUI

For Web User Interface (WUI) regions that are capable of running with an external security manager (ESM) such as RACF, it is necessary to define the CICSplex SM transactions that run in the region to the ESM.

### Before you begin

This is necessary when transaction attach security is active in the WUI region, where SEC=YES and XTRAN=YES system initialization parameters are specified.

### Procedure

1. Create the same definitions as you would for a MAS region.  
See [“Defining the transactions in a managed CICS region” on page 411](#).
2. Define READ access to the COVG and COVC transactions for the following user IDs:
  - The region user ID
  - All user IDs that are specified on the **PLTPIUSR** system initialization parameter for the region
  - All WUI system administrators.
3. Define READ access to the COVE, COVP, and COVU transactions for the WUI default user ID.
4. Define READ access to the COVA transaction for all WUI users.

### Example

A list of these transactions is also contained in the CSD group EYU\$WDEF.

## Authorizing access to MVS data sets

In addition to standard CICS and CICSplex SM requirements, the CICS region user ID must have the authority to access the data sets associated with the DD names described in the table.

Table 34. Security access required for MVS data sets	
DDnames	Access required
EYUWUI	READ
DFHHTML	READ
EYUCOVI (and clones)	READ
EYUWREP	UPDATE
EYULOG	UPDATE
EYUCOVE (and clones)	UPDATE

## Running your Web User Interface server with security active

If the Web User Interface server is running with the CICS system initialization parameter **SEC** set to YES, you can control who can access the Web User Interface, what resources they can see, what actions they can perform, and the use of the view editor.

If you have already set up CICSplex SM security for use with the CICSplex SM API, users have the same level of access with the Web User Interface as they do with the API.

When you attempt to connect to a Web User Interface server, the CICSplex SM Web User Interface Signon Panel is displayed. The user ID and password entered in this panel are passed to the Web User Interface server, in plain text over the TCP/IP connection, unless you are using SSL support, and then verified by the external security manager. If the external security manager supports mixed case passwords, and this feature is active, an icon is displayed next to the password field when you sign on.

All users who are successfully signed on to the Web User Interface have access to all of the customizable view and menu help pages, if the customizable view and menu help is served by the Web User Interface.

If security is active, messages produced by auditing system programming interface commands contain the user ID used to log on to the WUI. See [Auditing SPI commands](#).

To control who is allowed to sign onto the Web User Interface server, you can protect the Web User Interface CICS application ID by using RACF APPL checking. See [Summary of RACF classes for CICS resources](#).

Access to managed resources uses standard CICSplex SM security using profiles in the CPSMOBJ class. For example, to see a CICS region view, the Web User Interface user needs READ authority through the CPSMOBJ class profile OPERATE.REGION.context.scope.

Access to CICS resources, and actions on resources in a view, use the simulated CICS security checking of CICSplex SM, which uses the normal CICS RACF resource and command security profiles. For example, to issue the shutdown action against a CICS region, if command security is active in the target CICS region, the Web User Interface user would need UPDATE authority to the SHUTDOWN command in the CCICSCMD class.

## Controlling access to Web User Interface resources

You can use your external security manager to control user access to views, menus, editors, and help information and control the import and export of views, menus, maps, user groups, and user objects that use COVC.

The navigation frame is exempt from security checks. To control user access, you need to create the appropriate profiles in the FACILITY class. The following ESM FACILITY profiles are available:

### **EYUWUI.wui\_server\_applid.VIEW.viewsetname**

- used to protect view sets.

### **EYUWUI.wui\_server\_applid.MENU.menuname**

- used to protect menus.

### **EYUWUI.wui\_server\_applid.MAP.mapname**

- used to protect maps.

### **EYUWUI.wui\_server\_applid.HELP.helpmembername**

- used to protect help pages.

### **EYUWUI.wui\_server\_applid.EDITOR**

- used to protect the View Editor.

### **EYUWUI.wui\_server\_applid.USER**

- used to protect the User Editor and user and user group objects.

where `wui_server_applid` is the CICS APPLID of the server.

Users can be given read or update access to WUI resources:

- Read -- to use the views, menus, maps and Help information in the main interface, or to export views, menus, maps, user groups, or user objects that use COVC.
- Update -- to access the editors and create, update, or remove items that use them, or to import views, menus, maps, user groups, or user objects that use COVC.

To support AUTOIMPORT functions, the user ID that runs the COVG transaction must also be given update access to these FACILITY profiles. This is either the region user ID, or the PLTPIUSR value if a PLTPI table is in use.

If the ESM that you are using, neither grants nor refuses access to a profile (for example, if no RACF profile is defined), all users who are successfully signed on to the Web User Interface have access to the resources. You can make not authorized the default by setting up a generic profile.

**Note:** This security is designed to protect the views and menus themselves and not the objects they manage, which is covered by normal CICSplex SM security.

When selecting a view set, map or menu to edit or delete within the view editor, only items for which you have update access are listed. However, when selecting an item to copy, all items for which you have read access are shown. This allows you to copy any object for which you have read only access to a private copy in your updateable namespace .

When browsing for views that are accessible, no security exceptions are logged. Users are presented with a list that has been filtered to remove the views that are not accessible. However, when a user attempts an unauthorized action; for example, creating a view in a denied namespace, the EYULOG security exception message EYUVS1100E is issued.

## Examples of WUI security profiles

The following examples use the RACF TSO command syntax and assume that the default CICS RACF classes and no security prefixing is in use.

This is not the only way that suitable profiles can be defined. These examples can be adapted to suit the installations requirements and standards.

In the examples, lower case strings should be replaced with the appropriate use ID or resource.

- Example 1

Create Web User Interface user groups:

```
ADDGROUP (WUISERV,WUIADM,WUIUSER,WUIEDIT)
```

- Example 2

Define profiles to protect Web User Interface transactions:

```
RDEFINE GCICSTRN WUISYS ADDMEM(COVP,COVU,COVE) UACC(NONE)
RDEFINE GCICSTRN WUIADMIN ADDMEM(COVG,COVC) UACC(NONE)
RDEFINE TCICSTRN COVA UACC(NONE)
```

- Example 3

Authorize user groups to appropriate profiles:

```
PERMIT WUISYS CLASS(GCICSTRN) ID(WUISERV) ACCESS(READ)
PERMIT WUIADMIN CLASS(GCICSTRN) ID(WUIADM) ACCESS(READ)
PERMIT COVA CLASS(TCICSTRN) ID(WUIUSER,WUIEDIT) ACCESS(READ)
```

- Example 4

Refresh transaction security profiles:

```
SETROPTS RACLIST(TCICSTRN) REFRESH
```

- Example 5

Define View Editor profile and give user group appropriate access:

```
RDEFINE FACILITY EYUWUI.wui_server_applid.EDITOR UACC(NONE)
PERMIT EYUWUI.wui_server_applid.EDITOR CLASS(FACILITY) ID(WUIEDIT) ACCESS(UPDATE)
```

- Example 6

Define view set profile and give user group appropriate access:

```
RDEFINE FACILITY EYUWUI.wui_server_applid.VIEW.viewsetname UACC(NONE)
PERMIT EYUWUI.wui_server_applid.VIEW.viewsetname CLASS(FACILITY) ID(WUIUSER)
ACCESS(READ)
```

- Example 7

Connect users to appropriate Web User Interface groups:

```
CONNECT wui_server_dfltuser GROUP(WUISERV)
CONNECT (wui_server_pltplusr,wui_administrator) GROUP(WUIADM)
```

```
CONNECT (wui_administrator,wui_view_designer) GROUP(WUIEDIT)
CONNECT (wui_operator1,wui_operator2...) GROUP(WUIUSER)
```

#### • Example 8

If CICS surrogate user security is active in the Web User Interface region, definitions similar to the following are required:

```
DEFINE SURROGAT wui_administrator.DFHSTART UACC(NONE)
PERMIT wui_administrator.DFHSTART CLASS(SURROGAT) ID(WUIADM) ACCESS(READ)
DEFINE SURROGAT wui_view_designer.DFHSTART UACC(NONE)
PERMIT wui_view_designer.DFHSTART CLASS(SURROGAT) ID(WUIADM) ACCESS(READ)
DEFINE SURROGAT wui_operator1.DFHSTART UACC(NONE)
PERMIT wui_operator1.DFHSTART CLASS(SURROGAT) ID(WUIADM) ACCESS(READ)
DEFINE SURROGAT wui_operator2.DFHSTART UACC(NONE)
PERMIT wui_operator2.DFHSTART CLASS(SURROGAT) ID(WUIADM) ACCESS(READ)

SETROPTS RACLIST(SURROGAT) REFRESH
```

## Running your Web User Interface server with security not active

If the Web User Interface server is running with the CICS system initialization parameter **SEC=NO**, users of the Web User Interface must provide a user ID to identify 'sessions' in the COVC transaction. Users are not required to provide a password.

The user ID does not need to be defined to the external security manager. Access checking of access to views, and CICS resources are based on the **DFLTUSER** for the Web User Interface server CICS region. All Web User Interface users have access to the View Editor and all menus, view sets and help members.

If security is not active, messages produced by auditing system programming interface commands contain the default user ID of either the CMAS or the MAS. See [Changes to the SPI](#) in the Upgrading documentation.

### Related reference

[Coupling facility data table server security parameters](#)

[DFLTUSER system initialization parameter](#)

## Specifying CICSplex SM resource names in profiles

You can create RACF profiles for CICSplex SM views for a specific CICS system, a group of CICS systems, or all systems comprising a CICSplex.

### About this task

The lists contain the resource names for CICSplex SM views to be used in RACF profiles.

CICSplex SM views are divided into groups that reflect the functions they perform. Within each functional group, the views are divided by their type. Functional groups can be even further qualified with the addition of a context and, for some groups, a scope or platform. You can control access to a specific set of views (and their associated action commands) by identifying the set in a profile, using one of the following resource name formats:

```
function.type.context
function.type.context.scope
```

where:

#### ***function***

is the name of the CICSplex SM function to be affected:

#### **ANALYSIS**

Real-time analysis

#### **BAS**

Business application services

**CLOUD**

Platforms and applications

**CONFIG**

CMAS configuration

**CSD**

CICS System Definition

**MONITOR**

Resource monitoring

**OPERATE**

Operations

**TOPOLOGY**

CICSplex configuration

**WORKLOAD**

Workload management

***type***

is the specific or generic name of an area that qualifies the CICSplex SM function to be affected. The specific names are:

**AIMODEL**

CICS AIMODEL

**APPLICTN**

CICS bundle resources and resources only installable by CICS bundles

**APPLICATION**

Applications deployed in platforms

**BRFACIL**

Link3270 Bridge Facility

**CONNECT**

CICS connections

**DB2DBCTL**

Db2/DBCTL resources and subsystems

**DEF**

CICSplex SM definitions

**DOCTEMP**

Document templates

**ENQMODEL**

CICS global enqueue models

**ENTJAVA**

CorbaServers and deployed DJARs

**EXIT**

CICS exits

**FEPI**

CICS FEPI resources

**FILE**

CICS files

**IPCONN**

IPIC connections

**JOURNAL**

Journal models

**PARTNER**

CICS partners

**PLATFORM**

Platforms

**PROCTYPE**

CICS BTS Process types

**PROFILE**

CICS profiles

**PROGRAM**

CICS programs

**REGION**

CICS region data

**RQMODEL**

CICS request models

**TASK**

CICS active tasks

**TCPIPS**

TCP/IP services

**TDQUEUE**

CICS transient data queues

**TERMINAL**

CICS terminals

**TRAN**

CICS transactions

**TSQUEUE**

CICS temporary storage queues

**UOW**

CICS units of work

The type must be valid for the specified function. [Table 35 on page 420](#) lists the valid function.type combinations.

**context**

Context is the specific or generic name of the CMAS or CICSplex to be affected by the designated function and type. If the function is CONFIG, the context can be a CMAS or a CICSplex. For all other functions, the context must be a CICSplex.

**scope**

Scope is the specific or generic name of a CICS system in the CICSplex identified as the context. Do not specify scope when the context is a CMAS, the type is DEF (except when handling CSD resources) or when the function is CLOUD.

**Note:**

1. In this section only, the term scope means CICS systems. It does **not** mean the scope (CICS system groups) you have defined as part of the CICSplex environment, nor does it refer to a BAS logical scope.
2. To include all of the systems comprising a CICS system group when their names do not match a generic system name, you must establish a profile for each system.

**Using asterisks in resource names**

To reduce the number of profiles you need to define, you can use \* (one asterisk) and \*\* (two consecutive asterisks) to represent one or more entries. Use of one or two asterisks is optional.

**Note:** Before using asterisks in profile definitions, ensure that generics have been activated for the relevant class:

The following examples demonstrate how asterisks can be used:

**OPERATE.\*.EYUPLX01.EYUPLX01**

Indicates that all views and action commands associated with any type valid within the OPERATE function are to be recognized when the context and scope are EYUPLX01.

**OPERATE.PROGRAM.\*\***

Indicates that all views and action commands associated with the PROGRAM type within the OPERATE function are to be recognized, regardless of the current context and scope.

**OPERATE.\*\***

Indicates that all views and action commands associated with any type valid within the OPERATE function are to be recognized, regardless of the current context and scope.

**\*\***

Indicates that *all* views and action commands associated with *any* type valid within *any* function are to be recognized, regardless of the current context and scope.

## Valid function and type combinations for CICSplex SM resource profiles

You control access to a specific set of CICSplex SM views and their associated action commands by identifying the set in a RACF resource profile with a defined resource name format, such as *function.type.context*. The *type* in this format must be valid for the specified *function*. This reference summary lists the valid function and type combinations and the resource names that are associated with each combination.

If you are using the WUI, note that the resource name is given at the foot of the views in the corresponding view set.

Table 35. Function and type combinations for resources accessible using CICSplex SM

Function.type combination	Resource Table Name and Usage
ANALYSIS.DEF	<p><b>ACTION</b> Create, display, and maintain action definitions.</p> <p><b>APACTV</b> Display analysis definitions associated with an analysis point specification.</p> <p><b>APSPEC</b> Create, display, and maintain analysis point specifications.</p> <p><b>CMDMPAPS</b> Identify the role of a primary CMAS.</p> <p><b>CMDMSAPS</b> Identify the role of a secondary CMAS.</p> <p><b>EVALDEF</b> Create, display, and maintain evaluation definitions.</p> <p><b>EVENT</b> Display changes in the status of a CICSplex.</p> <p><b>EVENTDTL</b> Display evaluation definitions associated with an analysis definition that caused an event.</p> <p><b>LNKSRSCG</b> Describe the link between a CICS system group and an analysis specification.</p> <p><b>LNKSRSCS</b> Describe the link between a CICS system and an analysis specification.</p> <p><b>RTAACTV</b> Display analysis and status definitions in CICS systems.</p> <p><b>RTADEF</b> Create, display, and maintain analysis definitions.</p> <p><b>RTAGROUP</b> Create, display, and maintain analysis groups.</p> <p><b>RTAINAPS</b> Display analysis groups in analysis point specifications.</p> <p><b>RTAINGRP</b> Display analysis and status definitions in analysis groups.</p> <p><b>RTAINSPC</b> Display analysis groups in analysis specifications.</p> <p><b>RTASPEC</b> Create, display, and maintain analysis specifications.</p> <p><b>STATDEF</b> Create, display, and maintain status definitions.</p> <p><b>STAINGRP</b> Identify the membership relation of a status definition in an RTAGROUP.</p>
BAS.APPLICTN	<p><b>BUNDDDEF</b> Install applications that are deployed as bundles.</p>
BAS.CONNECT	<p><b>CONNDEF</b> Install ISC/MRO connection definitions.</p> <p><b>IPCONDEF</b> Install IPIC connection definitions.</p> <p><b>SESSDEF</b> Install session definitions.</p>
BAS.DB2DBCTL	<p><b>DB2CDEF</b> Install Db2 connection definitions.</p> <p><b>DB2EDEF</b> Install Db2 entry definitions.</p> <p><b>DB2TDEF</b> Install Db2 transaction definitions.</p>



Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF	<p><b>ATOMDEF</b> Create, display, and maintain Atom service definitions.</p> <p><b>BUNDEF</b> Create, display, and maintain bundle definitions.</p> <p><b>CONNDEF</b> Create, display, and maintain MRO and ISC over SNA connection definitions.</p> <p><b>DB2CDEF</b> Create, display, and maintain Db2 connection definitions.</p> <p><b>DB2EDEF</b> Create, display, and maintain Db2 entry definitions.</p> <p><b>DB2TDEF</b> Create, display, and maintain Db2 transaction definitions.</p> <p><b>DOCDEF</b> Create, display, and maintain document template definitions.</p> <p><b>ENQMDEF</b> Create, display, and maintain enqueue models definitions.</p> <p><b>FENODDEF</b> Create, display, and maintain FEPI node definitions.</p> <p><b>FEPODEF</b> Create, display, and maintain FEPI pool definitions.</p> <p><b>FEPRODEF</b> Create, display, and maintain FEPI property set definitions.</p> <p><b>FETRDEF</b> Create, display, and maintain FEPI target definitions.</p> <p><b>FILEDEF</b> Create, display, and maintain file definitions.</p> <p><b>IPCONDEF</b> Create, display, and maintain IPIC connection definitions.</p> <p><b>JRNMDDEF</b> Create, display, and maintain journal model definitions.</p> <p><b>JVMSVDEF</b> Create, display, and maintain JVM server definitions.</p> <p><b>LIBDEF</b> Create, display, and maintain LIBRARY definitions.</p> <p><b>LSRDEF</b> Create, display, and maintain LSR pool definitions.</p> <p><b>MAPDEF</b> Create, display, and maintain mapset definitions.</p> <p><b>MQCONDEF</b> Create, display, and maintain IBM MQ connection definitions.</p> <p><b>MQMONDEF</b> Create, display, and maintain IBM MQ monitor definitions.</p> <p><b>PARTDEF</b> Create, display, and maintain partner definitions.</p> <p><b>PIPEDEF</b> Create, display, and maintain pipeline definitions.</p> <p><b>PROCDEF</b> Create, display, and maintain process type definitions.</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF (continued)	<p><b>PROFDEF</b> Create, display, and maintain profile definitions.</p> <p><b>PROGDEF</b> Create, display, and maintain program definitions.</p> <p><b>PRTNDEF</b> Create, display, and maintain partition set definitions.</p> <p><b>RASGNDEF</b> Create, display, and maintain resource assignments.</p> <p><b>RASINDSC</b> Display resource assignments in descriptions.</p> <p><b>RASPROC</b> Display resource assignment process.</p> <p><b>RDSCPROC</b> Display resource description process.</p> <p><b>RESDESC</b> Create, display, maintain, and install resource descriptions.</p> <p><b>RESGROUP</b> Create, display, maintain, and install resource groups.</p> <p><b>RESINDSC</b> Display resource groups in descriptions.</p> <p><b>RESINGRP</b> Display resource definitions in groups.</p> <p><b>RQMDEF</b> Create, display, and maintain request model definitions.</p> <p><b>SESSDEF</b> Create, display, and maintain session definitions.</p> <p><b>SYSRES</b> Display CICS system resources.</p> <p><b>TCPDEF</b> Create, display, and maintain TCP/IP service definitions.</p> <p><b>TDQDEF</b> Create, display, and maintain transient data queue definitions.</p> <p><b>TERMDEF</b> Create, display, and maintain terminal definitions.</p> <p><b>TRANDEF</b> Create, display, and maintain transaction definitions.</p> <p><b>TRNCLDEF</b> Create, display, and maintain transaction class definitions.</p> <p><b>TYPTMDEF</b> Create, display, and maintain typeterm definitions.</p> <p><b>URIMPDEF</b> Create, display, and maintain URI map definitions.</p> <p><b>WEBSVDEF</b> Create, display, and maintain web services definitions.</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF (continued)	<p><b>ATMINGRP</b> Describe the membership of an Atom service definition in a resource group.</p> <p><b>BUNINGRP</b> Describe the membership of a bundle definition in a resource group.</p> <p><b>CONINGRP</b> Describe the membership of an MRO or ISC over SNA connection definition in a resource group.</p> <p><b>DOCINGRP</b> Describe the membership of a document template definition in a resource group.</p> <p><b>D2CINGRP</b> Describe the membership of a Db2 connection definition in a resource group.</p> <p><b>D2EINGRP</b> Describe the membership of a Db2 entry definition in a resource group.</p> <p><b>D2TINGRP</b> Describe the membership of a Db2 transaction definition in a resource group.</p> <p><b>ENQINGRP</b> Describe the membership of an ENQ/DEQ model definition in a resource group.</p> <p><b>FILINGRP</b> Describe the membership of a file definition in a resource group.</p> <p><b>FNOINGRP</b> Describe the membership of a FEPI node definition in a resource group.</p> <p><b>FPOINGRP</b> Describe the membership of a FEPI pool definition in a resource group.</p> <p><b>FPRINGRP</b> Describe the membership of a FEPI property set definition in a resource group.</p> <p><b>FSGINGRP</b> Describe the membership of a file key segment definition in a resource group.</p> <p><b>FTRINGRP</b> Describe the membership of a FEPI target definition in a resource group.</p> <p><b>IPCINGRP</b> Describe the membership of an IPIC connection definition in a resource group.</p> <p><b>JMSINGRP</b> Describe the membership of a JVM server definition in a resource group.</p> <p><b>JRMINGRP</b> Describe the membership of a journal model definition in a resource group.</p> <p><b>LIBINGRP</b> Describe the membership of a LIBRARY definition in a resource group.</p> <p><b>LSRINGRP</b> Describe the membership of an LSR pool definition in a resource group.</p> <p><b>MAPINGRP</b> Describe the membership of a map set definition in a resource group.</p> <p><b>MQCINGRP</b> Display IBM MQ connection definitions in groups.</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF (continued)	<p><b>PARINGRP</b> Describe the membership of a partner definition in a resource group.</p> <p><b>PGMINGRP</b> Describe the membership of a program definition in a resource group.</p> <p><b>PIPINGRP</b> Describe the membership of a pipeline definition in a resource group.</p> <p><b>PRCINGRP</b> Describe the membership of a process type definition in a resource group.</p> <p><b>PRNINGRP</b> Describe the membership of a partition set definition in a resource group.</p> <p><b>PROINGRP</b> Describe the membership of a profile definition in a resource group.</p> <p><b>SEISINGRP</b> Describe the membership of a session definition in a resource group.</p> <p><b>TCLINGRP</b> Describe the membership of a transaction class definition in a resource group.</p> <p><b>TCPINGRP</b> Describe the membership of a TCPIP Service definition in a resource group.</p> <p><b>TDQINGRP</b> Describe the membership of a transient data queue definition in a resource group.</p> <p><b>TRMINGRP</b> Describe the membership of a terminal definition in a resource group.</p> <p><b>TRNINGRP</b> Describe the membership of a transaction definition in a resource group.</p> <p><b>TSMINGRP</b> Describe the membership of a temporary storage model definition in a resource group.</p> <p><b>TYPINGRP</b> Describe the membership of a typeterm definition in a resource group.</p> <p><b>URIINGRP</b> Describe the membership of a URI map definition in a resource group.</p> <p><b>WEBINGRP</b> Describe the membership of a web service definition in a resource group.</p>
BAS.DOCTEMP	<p><b>DOCTEMP</b> Install document template definitions.</p>
BAS.ENQMODEL	<p><b>ENQMDEF</b> Install enqueue model definitions.</p>
BAS.ENTJAVA	<p><b>JVMSVDEF</b> Install JVM server definitions.</p>
BAS.FILE	<p><b>FILEDEF</b> Install file definitions.</p>
BAS.JOURNAL	<p><b>JRNMDEF</b> Install journal model definitions.</p>
BAS.PARTNER	<p><b>PARTDEF</b> Install partner definitions.</p>
BAS.PROCTYPE	<p><b>PROCDEF</b> Install BTS Process type definitions.</p>
BAS.PROFILE	<p><b>PROFDEF</b> Install profile definitions.</p>
BAS.PROGRAM	<p><b>LIBDEF</b> Install LIBRARY definitions.</p> <p><b>MAPDEF</b> Install map set definitions.</p> <p><b>PROGDEF</b> Install program definitions.</p> <p><b>PRTNDEF</b> Install partition set definitions.</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.REGION	<b>LSRDEF</b> Install LSR pool definitions. <b>MQCONDEF</b> Install IBM MQ connection definitions. <b>MQMONDEF</b> Install IBM MQ monitor definitions. <b>TRNCLDEF</b> Install transaction class definitions.
BAS.TCPIPS	<b>ATOMDEF</b> Install Atom service definitions. <b>PIPEDEF</b> Install pipeline definitions. <b>TCPDEF</b> Install TCPIP service definitions. <b>URIMPDEF</b> Install URI map definitions. <b>WEBSVDEF</b> Install web services definitions.
BAS.TDQUEUE	<b>TDQDEF</b> Install transient data queue definitions.
BAS.TERMINAL	<b>TERMDEF</b> Install terminal definitions. <b>TYPTMDEF</b> Install typeterm definitions.
BAS.TRAN	<b>TRANDEF</b> Install transaction definitions.
CLOUD.APPLICATION	<b>APPLDEF</b> Install application definitions in platforms. <b>APPLCTN</b> Display and manage applications deployed in platforms.
CLOUD.DEF	<b>APPLDEF</b> Create, display, and maintain application definitions in platforms. <b>PLATDEF</b> Create, display, and maintain platform definitions.
CLOUD.PLATFORM	<b>PLATDEF</b> Install platform definitions. <b>PLATFORM</b> Display and manage platforms. <b>MGMTPART</b> Create, display, and maintain management parts for applications and platforms.

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
CONFIG.DEF	<p><b>CICSplex</b> Display and manage CMAS in CICSplex.</p> <p><b>CMAS</b> Display and manage active CMASs.</p> <p><b>CMASLIST</b> Describe a CMAS and its connection to characteristics.</p> <p><b>CMASplex</b> Display CICSplexes for a CMAS.</p> <p><b>CMTCMDEF</b> Create, display, and maintain CMAS links.</p> <p><b>CMTMLNK</b> Display active CMAS links.</p> <p><b>CMTPLNK</b> Display an active CMAS-to-MAS link.</p> <p><b>CplexDEF</b> Create, display, and maintain CICSplex definitions.</p> <p><b>CPLXCMAS</b> Display CMAS to CICSplex.</p>
CSD.DEF	<p><b>ATOMDEF</b> Create, install, display, and maintain Atom service definitions.</p> <p><b>BUNDEF</b> Create, install, display, and maintain bundle definitions.</p> <p><b>CONDEF</b> Create, install, display, and maintain MRO and ISC over SNA connection definitions.</p> <p><b>CSDGROUP</b> Create, install, display, and maintain CSD resource group definitions.</p> <p><b>CSDINGRP</b> Describe the membership of a CSD resource group.</p> <p><b>CSDINLST</b> Describe the membership of a CSD resource group list.</p> <p><b>CSDLIST</b> Create, install, display, and maintain CSD resource group list definitions.</p> <p><b>DB2CDEF</b> Create, install, display, and maintain Db2 connection definitions.</p> <p><b>DB2EDEF</b> Create, install, display, and maintain Db2 entry definitions.</p> <p><b>DB2TDEF</b> Create, install, display, and maintain Db2 transaction definitions.</p> <p><b>DOCDEF</b> Create, install, display, and maintain document template definitions.</p> <p><b>ENQMDEF</b> Create, install, display, and maintain enqueue models definitions.</p> <p><b>FILEDEF</b> Create, install, display, and maintain file definitions.</p> <p><b>IPCONDEF</b> Create, install, display, and maintain IPIC connection definitions.</p> <p><b>JRNMDDEF</b> Create, install, display, and maintain journal model definitions.</p> <p><b>JVMSVDEF</b> Create, install, display, and maintain JVM server definitions.</p> <p><b>LIBDEF</b> Create, install, display, and maintain LIBRARY definitions.</p> <p><b>LSRDEF</b> Create, install, display, and maintain LSR pool definitions.</p> <p><b>MAPDEF</b> Create, install, display, and maintain mapset definitions.</p> <p><b>MQCONDEF</b> Create, install, display, and maintain IBM MQ connection definitions.</p> <p><b>MQMONDEF</b> Create, install, display, and maintain IBM MQ monitor definitions.</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
CSD.DEF (continued)	<p><b>PARTDEF</b> Create, install, display, and maintain partner definitions.</p> <p><b>PIPEDEF</b> Create, install, display, and maintain pipeline definitions.</p> <p><b>PROCDEF</b> Create, install, display, and maintain process type definitions.</p> <p><b>PROFDEF</b> Create, install, display, and maintain profile definitions.</p> <p><b>PROGDEF</b> Create, install, display, and maintain program definitions.</p> <p><b>PRTNDEF</b> Create, install, display, and maintain partition set definitions.</p> <p><b>SESSDEF</b> Create, display, and maintain session definitions.</p> <p><b>TCPDEF</b> Create, install, display, and maintain TCP/IP service definitions.</p> <p><b>TDQDEF</b> Create, install, display, and maintain transient data queue definitions.</p> <p><b>TERMDEF</b> Create, install, display, and maintain terminal definitions.</p> <p><b>TRANDEF</b> Create, install, display, and maintain transaction definitions.</p> <p><b>TRNCLDEF</b> Create, install, display, and maintain transaction class definitions.</p> <p><b>TSMINGRP</b> Describe the membership of a temporary storage model definition in a resource group.</p> <p><b>TYPTMDEF</b> Create, install, display, and maintain typeterm definitions.</p> <p><b>URIMPDEF</b> Create, install, display, and maintain URI map definitions.</p> <p><b>WEBSVDEF</b> Create, install, display, and maintain web services definitions.</p>
MONITOR.CONNECT	<p><b>MCONNECT</b> ISC and MRO connections</p> <p><b>MMODENAME</b> LU 6.2 modenames</p>
MONITOR.DB2DBCTL	<p><b>MDB2THRD</b> Db2 threads</p>
MONITOR.DEF	<p><b>LNKSMSCG</b> Describe the link between a CICS system group and a monitor specification.</p> <p><b>LNKSMSCS</b> Describe the link between a CICS system and a monitor specification.</p> <p><b>MONDEF</b> Create, display, and maintain monitor definitions.</p> <p><b>MONGROUP</b> Create, display, and maintain monitor groups.</p> <p><b>MONINGRP</b> Create, display, and maintain monitor definitions in monitor groups.</p> <p><b>MONINSPC</b> Create, display, and maintain monitor groups in monitor specifications.</p> <p><b>MONSPEC</b> Create, display, and maintain monitor specifications.</p> <p><b>POLMON</b> Describe a monitor definition in a specific CICS system.</p>
MONITOR.FEPI	<p><b>MFEPICON</b> FEPI connections</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
MONITOR.FILE	<b>MCMDT</b> Data tables <b>MLOCFILE</b> Local files <b>MREMFIL</b> Remote files
MONITOR.JOURNAL	<b>MJOURNL</b> Journals <b>MJRNLNAM</b> Journal names
MONITOR.PROGRAM	<b>MPROGRAM</b> Programs
MONITOR.REGION	<b>MCICSDSA</b> Dynamic storage areas <b>MCICSRGN</b> CICS systems <b>MLSRPBUF</b> LSRPOOL buffer pool <b>MLSRPOOL</b> LSRPOOL <b>MTRANCLS</b> Transaction classes
MONITOR.TDQUEUE	<b>MINDTDQ</b> Indirect transient data queues <b>MNTRATDQ</b> Intrapartition transient data queues <b>MREMTDQ</b> Remote transient data queues <b>MTDQGBL</b> Global intrapartition transient data queues <b>MXTRATDQ</b> Extrapartition transient data queues
MONITOR.TERMINAL	<b>MTERMNL</b> Terminals
MONITOR.TRAN	<b>MLOCTRAN</b> Local transactions <b>MREMTTRAN</b> Remote transactions
MONITOR.TSQUEUE	<b>MTSQGBL</b> Global temporary storage queues
OPERATE.AIMODEL	<b>AIMODEL</b> Autoinstall models <b>CRESAIMD</b> Describe an instance of an autoinstalled terminal model in a CICS system.



Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.APPLICTN	<p><b>BUNDLE</b> A CICS resource object that provides information about bundles.</p> <p><b>BUNDPART</b> A CICS resource object that provides information about a bundle part.</p> <p><b>CRESBUND</b> Describes an instance of a bundle in a CICS system.</p> <p><b>CRESXMLT</b> Describes an instance of an XMLTRANSFORM resource in a CICS system.</p> <p><b>EPADAPT</b> A CICS resource object that provides information about EP adapters.</p> <p><b>EPADSET</b> A CICS resource object that provides information about EP adapter sets.</p> <p><b>EPAINSET</b> A CICS resource object that provides the name of event processing adapters in an event processing adapter set.</p> <p><b>EVCSDATA</b> A CICS resource object that provides information about application data predicates.</p> <p><b>EVCSINFO</b> A CICS resource object that provides information about information sources.</p> <p><b>EVCSOPT</b> A CICS resource object that provides information about application option predicates.</p> <p><b>EVCSPEC</b> A CICS resource object that provides information about capture specifications.</p> <p><b>EVNTBIND</b> A CICS resource object that provides information about event bindings.</p> <p><b>EVNTGBL</b> A CICS resource object that provides information about event processing.</p> <p><b>NODEJSAP</b> A CICS resource object that provides information about a Node.js application.</p> <p><b>OSGIBUND</b> A CICS resource object that provides information about an OSGi bundle.</p> <p><b>OSGISERV</b> A CICS resource object that provides information about an OSGi service.</p> <p><b>XMLTRANS</b> XMLTRANSFORM resources.</p>
OPERATE.BRFACIL	<p><b>BRFACIL</b> LINK3270 bridge facility</p>
OPERATE.CONNECT	<p><b>CONNECT</b> MRO and ISC over SNA connections</p> <p><b>CRESCONN</b> Describe an instance of an MRO or ISC over SNA connection in a CICS system.</p> <p><b>CRESIPCN</b> Describe an instance of an IPIC connection in a CICS system.</p> <p><b>CRESMODE</b> Describe an instance of an LU6.2 modename in a CICS system.</p> <p><b>IPCONN</b> IPIC connections</p> <p><b>MODENAME</b> LU 6.2 modenames</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.DB2DBCTL	<p><b>CRESDB2C</b> Describe an instance of a Db2 connection in a CICS system.</p> <p><b>CRESDB2E</b> Describe an instance of a Db2 entry in a CICS system.</p> <p><b>CRESDB2P</b> Describe an instance of a Db2 package set in a CICS region.</p> <p><b>CRESDB2T</b> Describe an instance of a Db2 transaction in a CICS system.</p> <p><b>DB2CONN</b> Db2 connection</p> <p><b>DB2ENTRY</b> Db2 entry</p> <p><b>DB2PKGST</b> Db2 package set</p> <p><b>DB2TRN</b> Db2 transaction</p> <p><b>DBCTLSS</b> DBCTL subsystem</p> <p><b>DB2SS</b> Db2 subsystem</p> <p><b>DB2THRD</b> Db2 threads</p> <p><b>DB2TRAN</b> Db2 transactions</p>
OPERATE.DOCTEMP	<p><b>CRESDOCT</b> Describe an instance of a document template in a CICS system.</p> <p><b>DOCTEMP</b> Document templates</p>
OPERATE.ENQMODEL	<p><b>CRESENQM</b> Describe an instance of an ENQ/DEQ model in a CICS system.</p> <p><b>ENQMODEL</b> Enqueue models</p>
OPERATE.ENTJAVA	<p><b>CLCACHE</b> Shared class caches</p> <p><b>CRESJVMS</b> Describe an instance of a JVM server in a CICS region.</p> <p><b>ENDPOINT</b> JVM Endpoints</p> <p><b>JVM</b> Java virtual machines</p> <p><b>JVMPPOOL</b> JVM pools</p> <p><b>JVMPROF</b> JVM profiles</p> <p><b>JVMSERV</b> JVM servers</p>
OPERATE.EXIT	<p><b>CRESGLUE</b> Describe an instance of a global user exit in a CICS system.</p> <p><b>CRESTRUE</b> Describe an instance of a task-related user exit in a CICS system.</p> <p><b>EXITGLUE</b> Global user exits</p> <p><b>EXITTRUE</b> Task-related user exits</p> <p><b>EXTGLORD</b> Ordered global user exits</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.FEPI	<p><b>CRESFECO</b> Describe an instance of a FEPI connection in a CICS system.</p> <p><b>CRESFEND</b> Describe an instance of a FEPI node in a CICS system.</p> <p><b>CRESFEPO</b> Describe an instance of a FEPI pool in a CICS system.</p> <p><b>CRESFETR</b> Describe an instance of a FEPI target in a CICS system.</p> <p><b>FEPICONN</b> FEPI connections</p> <p><b>FEPINODE</b> FEPI nodes</p> <p><b>FEPIPOOL</b> FEPI pools</p> <p><b>FEPIPROP</b> FEPI property sets</p> <p><b>FEPITRGT</b> FEPI targets</p>
OPERATE.FILE	<p><b>CFDTPOOL</b> Coupling facility data tables pools</p> <p><b>CMDT</b> Data tables</p> <p><b>CRESDSNM</b> Describe an instance of a data set in a CICS system.</p> <p><b>CRESFILE</b> Describe an instance of a file in a CICS system.</p> <p><b>DSNAME</b> Data sets</p> <p><b>LOCFILE</b> Local files</p> <p><b>REMFIL</b> Remote files</p>
OPERATE.JOURNAL	<p><b>CRESJRNL</b> Describe an instance of a journal in a CICS system.</p> <p><b>CRESJRNM</b> Describe an instance of a journal name in a CICS system.</p> <p><b>JRNLMODL</b> Journal models</p> <p><b>JRNLMODL</b> Journal models</p> <p><b>JRNLMODL</b> System logs and general logs</p> <p><b>STREAMNM</b> MVS log streams</p>
OPERATE.PARTNER	<p><b>CRESPART</b> Describe an instance of a partner table in a CICS system.</p> <p><b>PARTNER</b> CICS partners</p>
OPERATE.PROCTYPE	<p><b>CRESPTY</b> Describe an instance of a process type in a CICS system.</p> <p><b>PROCTYP</b> Process types</p>
OPERATE.PROFILE	<p><b>CRESPROF</b> Describe an instance of a profile in a CICS system.</p> <p><b>PROFILE</b> CICS profiles</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.PROGRAM	<p><b>CRESPRGM</b> Describe an instance of a program in a CICS system.</p> <p><b>PROGRAM</b> Programs</p> <p><b>LIBDSN</b> LIBRARY data set names</p> <p><b>LIBRARY</b> LIBRARY data sets</p> <p><b>RPLLIST</b> DFHRPL data sets</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.REGION	<p><b>CICSDSA</b> Dynamic storage areas</p> <p><b>CICSPAGP</b> CICS page pools</p> <p><b>CICSRGN</b> CICS systems</p> <p><b>CICSSTOR</b> All CICS dynamic storage areas</p> <p><b>CRESMQMN</b> Describe an instance of an IBM MQ monitor in a CICS system</p> <p><b>CRESSDMP</b> Describe an instance of a system dump code in a CICS system</p> <p><b>CRESTDMP</b> Describe an instance of a transaction dump code in a CICS system</p> <p><b>DOMSPOOL</b> CICS storage domain subpools</p> <p><b>DSPGBL</b> Global CICS dispatcher</p> <p><b>DSPMODE</b> CICS dispatcher TCB mode</p> <p><b>DSPPOOL</b> CICS Dispatcher TCB pool</p> <p><b>ENQUEUE</b> CICS enqueues</p> <p><b>FEATURE</b> Feature toggles</p> <p><b>LOADACT</b> CICS Loader activity by dynamic storage area</p> <p><b>LOADER</b> CICS loader activity</p> <p><b>LSRPBUF</b> Buffer usage for LSR pools</p> <p><b>LSRPOOL</b> LSR pools</p> <p><b>MASHIST</b> MAS history</p> <p><b>MONITOR</b> CICS monitoring and statistics</p> <p><b>MQCON</b> IBM MQ connection</p> <p><b>MQCONN</b> IBM MQ connection statistics</p> <p><b>MQINI</b> IBM MQ initiation queue</p> <p><b>MQMON</b> IBM MQ monitor</p> <p><b>MVSESTG</b> MVS storage element</p> <p><b>MVSTCB</b> MVS TCB</p> <p><b>MVSTCBGL</b> Global MVS TCB</p> <p><b>MVSWLM</b> z/OS Workload Manager</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.REGION (continued)	<p><b>RECOVERY</b> CICS recovery manager</p> <p><b>REQID</b> Timed requests</p> <p><b>SYSDUMP</b> System dump codes</p> <p><b>SYSPARM</b> A CICS Resource that describes a system initialization parameter or a system initialization parameter override in an active CICS system being managed by CICSplex SM.</p> <p><b>TRANCLAS</b> Transaction classes</p> <p><b>TRANDUMP</b> Transaction dump codes</p>
OPERATE.RQMODEL	<p><b>CRESRQMD</b> Describe an instance of a request in a CICS system.</p>
OPERATE.TASK	<p><b>EXCI</b></p> <p><b>HTASK</b> Completed task</p> <p><b>TASK</b> Active tasks</p> <p><b>TASKESTG</b> Task storage element</p> <p><b>TASKFILE</b> Task file usage</p> <p><b>TASKRMI</b> RMI usage by individual task</p> <p><b>TASKTSQ</b> TSQ usage by individual task</p> <p><b>TSKSPOLS</b> All task subpools</p> <p><b>TSKSPPOOL</b> Task storage subpools</p>
OPERATE.TCPIPS	<p><b>ATOMSERV</b> Atom services</p> <p><b>CRESATOM</b> Describe an instance of an Atom service in a CICS system.</p> <p><b>CRESTCPS</b> Describe an instance of a TCPIP service in a CICS system.</p> <p><b>HOST</b> URI Host</p> <p><b>IPFACIL</b> IPIC connection sessions</p> <p><b>PIPELINE</b> Pipelines</p> <p><b>TCPIPGBL</b> TCP/IP global statistics</p> <p><b>TCPIPS</b> TCP/IP services</p> <p><b>URIMAP</b> URI maps</p> <p><b>URIMPGBL</b> URI map global statistics</p> <p><b>WEBSERV</b> Web services</p>

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.TDQUEUE	<b>CRESTDQ</b> Describe an instance of a transient data queue in a CICS system. <b>EXTRATDQ</b> Extrapartition transient data queues <b>INDTDQ</b> Indirect transient data queues <b>INTRATDQ</b> Intrapartition transient data queues <b>REMTDQ</b> Remote transient data queues <b>TDQGBL</b> Intrapartition transient data queue usage
OPERATE.TERMINAL	<b>CRESTERM</b> Describe an instance of a terminal in a CICS system. <b>TERMNL</b> Terminals
OPERATE.TRAN	<b>CRESTRAN</b> Describe an instance of a transaction in a CICS system. <b>LOCTRAN</b> Local transactions <b>REMTRAN</b> Remote transactions <b>TASKASSC</b> Task association data
OPERATE.TSQUEUE	<b>CRETSMD</b> Describe an instance of a temporary storage queue in a CICS system. <b>TSQUEUE</b> Temporary storage queues <b>TSMODEL</b> Temporary storage models <b>TSPool</b> Temporary storage pools <b>TSQGBL</b> Global temporary storage queues <b>TSQSHR</b> Shared temporary storage queues <b>TSQNAME</b> Long temporary storage queues
OPERATE.UOW	<b>UOWDSNF</b> Display shunted units of work <b>UOWENQ</b> Display enqueues for executing units of work <b>UOWLINK</b> Display links for unit of work <b>UOW</b> Display executing units of work

Table 35. Function and type combinations for resources accessible using CICSplex SM (continued)

Function.type combination	Resource Table Name and Usage
TOPOLOGY.DEF	<p><b>CSYSGRP</b> Create, display, and maintain CICS system groups.</p> <p><b>CSYSDEF</b> Create, display, and maintain CICS systems.</p> <p><b>CSGLCGCG</b> Describe the link of a CICS system group to an outer system group.</p> <p><b>CSGLCGCS</b> Describe the link of a CICS system to a system group.</p> <p><b>MAS</b> Display CICS systems in a CICSplex.</p> <p><b>MASSTAT</b> Display CICS systems in a CICSplex by CMAS.</p> <p><b>PERIODEF</b> Display period definitions.</p> <p><b>SYSLINK</b> Display information about the links that exist between CICS systems.</p>
WORKLOAD.DEF	<p><b>DTRINGRP</b> Display transactions in transaction groups.</p> <p><b>LNKSWSCG</b> Describe the link between a CICS system group and a workload specification.</p> <p><b>LNKSWSCS</b> Describe the link between a CICS system and a workload specification.</p> <p><b>TRANGRP</b> Create, display, and maintain transaction groups.</p> <p><b>WLMATAFF</b> Display and discard active affinities.</p> <p><b>WLMATGRP</b> Display and discard active transaction groups</p> <p><b>WLMATRAN</b> Display and discard transaction directory.</p> <p><b>WLMAWAOR</b> Display active AORs in a workload.</p> <p><b>WLMAWDEF</b> Display and discard active workload definitions.</p> <p><b>WLMAWORK</b> Display active workloads.</p> <p><b>WLMAWTOR</b> Display active TORs in a workload.</p> <p><b>WLMDEF</b> Create, display, and maintain workload definitions.</p> <p><b>WLMGROUP</b> Create, display, and maintain workload groups.</p> <p><b>WLMINGRP</b> Display workload definitions in groups.</p> <p><b>WLMINSPC</b> Display workload groups in workload specifications.</p> <p><b>WLMSPEC</b> Create, display, and maintain workload specifications.</p>

## Security for platforms and applications

You can secure resources for applications that are deployed on platforms by creating RACF security profiles for CICSplex SM to cover platforms and applications in a CICSplex.

Security for platforms and applications is set up in a similar way to security for other CICSplex SM components. You control access to a specific set of views (and their associated action commands) by identifying the set in a security profile. With these security profiles, you can give users authority to install, enable or disable, make available or unavailable, inquire on, or discard platforms and applications, and ensure that unauthorized users cannot create and administer these resources.



When you give a user authority to perform an action on a platform or application, you also give them authority to perform the same action on the dynamically generated resources for the platform or application. For example, a user who has authority to enable an application also has authority to enable the CICS bundles for the application that were installed in CICS regions in all the platforms in the CICSplex. CICS command and resource security checks, and simulated CICS security checking in CICSplex SM, are not carried out when you operate on CICS bundles through an application or platform.

You can secure a platform and its deployed applications by setting up security profiles with the following function and type combinations:

#### **CLOUD.DEF.context**

This security profile covers the PLATDEF and APPLDEF resource tables, which contain the definitions for platforms and applications. *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with UPDATE access for this security profile can create, update, and remove definitions for platforms and applications in the CICSplex SM data repository. Users with READ access can view those definitions in the CICSplex SM data repository.

#### **CLOUD.PLATFORM.context**

This security profile covers the installation of PLATDEF resources and operations on PLATFORM resources. It also allows users to view management parts (MGMTPART resources). *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with ALTER access for this security profile can install platforms in the CICSplex and discard them. (To install a platform, users also need READ access for the CLOUD.DEF profile that covers the PLATDEF resource.) Users with UPDATE access can enable and disable platforms. Users with UPDATE access can also add CICS regions to region types in the platform and remove CICS regions from region types in the platform. Users with READ access can view PLATFORM resources and MGMTPART resources. These permissions apply for all platforms that exist in the CICSplex.

#### **CLOUD.APPLICATION.context**

This security profile covers the installation of APPLDEF resources and operations on APPLCTN resources. *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with ALTER access for this security profile can install applications in the CICSplex and discard them. (To install an application, users also need READ access for the CLOUD.DEF profile that covers the APPLDEF resource.) Users with UPDATE access can enable and disable applications and make them available or unavailable. Users with READ access can view APPLCTN resources. These permissions apply for all applications in all platforms that exist in the CICSplex. If you require different security permissions for certain applications, use a different CICSplex to host the platform where you deploy the application.

**Note:** These security profiles are only checked in the maintenance point CMAS. Security checks are reported by message EYUCR0009I in the EYULOG of the maintenance point CMAS. To receive message EYUCR0009I for violations you must set the CICSplex SM system parameter (EYUPARM) **SECLOGMSG** to YES. For more information about **SECLOGMSG**, see [CICSplex SM system parameters](#).

Although the CLOUD security profiles cover actions on the dynamically generated resources for the platform or application, users may still carry out a limited set of actions directly on individual resources in the CICSplex and CICS regions where they are installed:

- You can modify the CICSplex SM topology definition (CSYSDEF, or CICS system definition) for a CICS region that is part of a platform. Attribute values that you specified at the region type level are locked and cannot be changed, but other attribute values can be changed.
- You can make available or unavailable, enable or disable, or inquire on, a BUNDLE resource that was dynamically created when you installed a platform or application. You cannot discard an individual bundle directly if it was created when you installed a platform or application.
- You can inquire on a dynamically created resource, such as a PROGRAM resource, that was defined inside a CICS bundle and created when you installed a platform or application. You cannot enable,

disable, or discard these resources directly if they were created when you installed a platform or application.

CICS command and resource security checks, and simulated CICS security checking in CICSplex SM, do apply when you perform an action directly on a CICS region that is part of a platform, or on an individual CICS bundle, or a resource defined in a CICS bundle, that was created when you installed a platform or application.

- A `TOPOLOGY.DEF.context` security profile covers actions on the CICSplex SM topology definitions for individual CICS regions that are part of a platform. *context* is the specific or generic name of the CICSplex that is covered by the security profile. Users with UPDATE access can modify the CSYSDEF for a CICS region that is part of a platform, with the exception of attribute values that are locked by the platform itself.
- CICS bundles created when you install a platform or application have a unique generated name beginning with the \$ character. To provide security for actions on individual CICS bundles that were dynamically created in this way, you can set up a security profile specifying the BUNDLE resource type and the resource name \$\*. Users with UPDATE access for BUNDLE.\$\* can make available or unavailable, or enable or disable, BUNDLE resources created for platforms and applications, and users with READ access can inquire on those BUNDLE resources.

If you apply security measures to individual PROGRAM resources, for applications that are deployed on platforms, secure the programs that are declared as application entry points, but do not secure other programs in the applications. The security settings that you specify for a program that is part of an application deployed on a platform apply to both public and private programs, and do not take into account the version of the application. Programs that are declared as an application entry point must have a unique PROGRAM resource name in your environment. However, if you secure programs that run at a lower level in the application, programs with the same names might be running in different applications, which can lead to unforeseen consequences. In this situation, a user might have permission to access a program that is declared as an application entry point, but not have permission to access a program that runs at a lower level in the application, because the security settings from another instance of the program name are in effect. Consider the security measures that you apply to a program that is declared as an application entry point program, as applying to the whole application.

If you used CICS bundles in earlier CICS releases, check the security permissions that you gave to users for those bundles. Depending on the way in which you set up security for CICS bundles, users with authority to take actions on individual CICS bundles might now be able to act on resources that are dynamically created as part of the installation of a bundle. Ensure that the levels of authority for BUNDLE resources are still appropriate.

Table 36 on page 438 summarizes the security checks that apply to actions performed on a platform, an application, an individual CICS bundle that was dynamically created when you installed a platform or application, or a resource defined in a CICS bundle for a platform or application.

Table 36. Security checks for operations on platforms, applications, and generated CICS bundles				
Operation	Platforms, including their CICS bundles	Applications, including their bundles	Dynamically-created CICS bundles	Resources defined in dynamically-created CICS bundles
Define	CLOUD.DEF profile (UPDATE, or READ to view definitions); also TOPOLOGY.DEF profile (UPDATE to modify individual CICS region CSYSDEF after platform install)	CLOUD.DEF profile (UPDATE, or READ to view definitions)	Cannot manage resource definitions individually	Cannot manage resource definitions individually
Install	CLOUD.PLATFORM profile (ALTER) and CLOUD.DEF profile (READ)	CLOUD.APPLICATION profile (ALTER) and CLOUD.DEF profile (READ)	Cannot install individually	Cannot install individually

Table 36. Security checks for operations on platforms, applications, and generated CICS bundles (continued)				
Operation	Platforms, including their CICS bundles	Applications, including their bundles	Dynamically-created CICS bundles	Resources defined in dynamically-created CICS bundles
Enable or disable	CLOUD.PLATFORM profile (UPDATE)	CLOUD.APPLICATION profile (UPDATE)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	Cannot enable or disable individually
Make available or unavailable	Not applicable	CLOUD.APPLICATION profile (UPDATE)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	Cannot make available or unavailable individually
Inquire	CLOUD.PLATFORM profile (READ); also allows viewing of management parts	CLOUD.APPLICATION profile (READ)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	CICS command and resource security, and simulated CICS security checking in CICSplex SM
Discard	CLOUD.PLATFORM profile (ALTER)	CLOUD.APPLICATION profile (ALTER)	Cannot discard individually	Cannot discard individually

For more information on setting up security for CICSplex SM and creating security profiles, see [Implementing CICSplex SM security](#).

## Activating simulated CICS security

When you create RACF profiles using the CICSplex SM resource classes to permit access to the operations and monitoring views, CICSplex SM determines which views a user can access. However, CICSplex SM cannot determine if that user is authorized to access the CICS resources represented within the view.

### About this task

You can enhance the security provided by your CICSplex SM profiles by activating *simulated CICS security checking*. Simulated security uses your existing RACF profiles to control access to CICS resources, CICS commands, or both. It is available only for the operations and monitor views. When using this combination of profiles, your CICSplex SM profiles determine which sets of views can be accessed and your CICS resource profiles determine which resources within the view can be accessed. For example, you can create a CICSplex SM profile that allows a user to issue the file view commands and any associated action commands, and then have CICS simulated security determine which files the user is authorized to access.

#### Note:

1. See [“Activating security for CICSplex SM” on page 441](#) for important information on how the CICSplex SM and CICS security parameters can affect simulated security.
2. Simulated security involves significantly more processing overhead than using only CICSplex SM profiles and will have a negative impact on performance.
3. CICSplex SM simulated CICS security does not include the simulation of CICS surrogate security. If CICS surrogate security checking is required, see [“Considerations for CICS surrogate security checks” on page 441](#) for guidance on how to secure your CICS regions correctly.
4. CICSplex SM does not honour the CMDSEC(NO) and RESSEC(NO) attributes of a CICS transaction issuing CICSplex SM API requests.

### Procedure

- To activate or deactivate simulated security checking, use the CSYSDEF view (for a single CICS system) or CPLEXDEF view (for multiple systems). You can indicate whether you want CICS resource checking,

CICS command checking, or both, to occur. CICS resource checking controls which resources are displayed in a view. CICS command checking controls what commands can be used within the view.

- To activate or deactivate simulated security checking temporarily for an active CICS system, use the MAS view.

## Exempting users and resources from security checking

There might be certain individuals who do not require security checking. There might also be certain CICS resources that are sufficiently protected by CICSplex SM profiles and, therefore, do not need to be involved in security checking.

You can exempt these individuals and resources from simulated CICS security checking using the CICSplex SM CPSMXMP resource class. Exemption bypasses only the simulated CICS security checks, not the basic CICSplex SM resource checks.

For example, if a user does not have RACF authority to issue the CICS command CEMT INQ FILE, you can enable that user to achieve the same result by creating a profile in the exemption class that allows the user to issue the equivalent CICSplex SM command **LOCFILE**.

To create exemption profiles:

1. Decide which resource you want to exempt and specify this on the **PERMIT** command. Use the resource name format described in “Specifying CICSplex SM resource names in profiles” on page 416.
2. Specify the class name CPSMXMP. This RACF class controls exemption from simulated security checking.
3. Specify the type of access that you require.
  - If you do not want to bypass security checking, specify ACCESS(NONE).
  - If you want to bypass security checking of INQUIRE level commands, specify ACCESS(READ).
  - If you want to bypass security checking of INQUIRE, SET, and PERFORM level commands, specify ACCESS(UPDATE).
  - If you want to bypass security checking of all commands, including DISCARD level commands, specify ACCESS(ALTER).
4. Specify the user or the group that you want the exemptions to apply to.

The following example shows how you could define an exemption profile that allows the individuals comprising the group EYUGRP2 to bypass security checking for all views and action commands associated with the TERMINAL type within the MONITOR function, when the context is EYUPLX01 and the scope is EYUMAS1A:

```
PERMIT MONITOR.TERMINAL.EYUPLX01.EYUMAS1A /* Resource name */+
CLASS(CPSMXMP) /* Class name */+
ACCESS(UPDATE) /* Access */+
ID(EYUGRP2) /* User or group */+
/* granted access */
```

## BAS security considerations

Because of the importance of resource definitions to your CICSplex environment, CICSplex SM enables you to define security for the BAS facilities.

Providing security for BAS is handled in the same way as it is for other CICSplex SM components. You can define as narrow or as broad a range of BAS functions as you like and authorize as few or as many people as you like to use them. For security purposes, the BAS functions are divided into the following groups:

### BAS.DEF

This group includes all of the resource definition views and the related BAS administration views. Users with UPDATE access to this group can create, update, and remove definitions in the CICSplex SM data repository. Users with READ access to this group can view definitions in the CICSplex SM data repository.

## **BAS.resource**

These groups are named according to the resource type they represent (such as BAS.CONNECT, for connection-related definitions). Each group includes the resource definition views for a given resource type. For example, BAS.CONNECT includes the **Connection definitions** views (CONNDEF objects) and **Session definitions** views (SESSDEF objects).

The purpose of these security groups is to further restrict a user's ability to install resources in CICS systems. A user must have ALTER access to the appropriate BAS.resource group in order to install the specified resources.

In addition to controlling access by function, you may want to limit the use of these functions to certain resources in certain CICS systems. CICSplex SM also provides simulated CICS security checking, which enables you to control access to CICS resources and commands.

You should be aware of the need to take special care in the adequate protection of the BAS views, so that unauthorized users cannot create and administer resources.

If you are using the EXEC CICS CREATE command to build new resources, any definition created with the CICSplex as the context is automatically distributed to all CMASs in the CICSplex. Therefore, giving a user authority to create BAS objects is equivalent to giving authority to install resources on any CICS system in the CICSplex. When the CICS system starts, there is no check on who installed the resource in the system.

For details on setting up security for CICSplex SM at your enterprise, see [Implementing CICSplex SM security](#).

## **Considerations for CICS surrogate security checks**

If CICS surrogate security checking is required, you need to work out which user ID against which CICS surrogate security checks will be performed and plan for that in the RACF definitions.

In some situations, CICS issues a surrogate security check on a user requesting an action that is required to be run on behalf of another user. To allow this request to be processed, the user requesting the action must have surrogate access for the other user ID.

However, CICSplex SM simulated CICS security does not include the simulation of CICS surrogate security.

For CICS surrogate security, the checks will be performed against the MAS agent user ID rather than the user issuing the request. To determine the MAS agent user ID, follow the instructions in [Determining the CICSplex SM agent user ID](#).

### **Related concepts**

[How it works: Identification in CICS](#)

[Surrogate security](#)

## **Activating security for CICSplex SM**

To activate security in the CMASs and MASs, you must set the CICSplex SM and CICS security-related system initialization parameters.

### **Procedure**

1. Specify the CICSplex SM parameter SEC in the EYUPARM data set or member defined in the JCL used to start the CMAS and MAS.
2. Specify the CICS parameter SEC= in the CICS system initialization parameters used to start the MAS.

### **Results**

Together these parameters determine what security checking is performed. [Table 37 on page 442](#) explains the possible parameter combinations.

Table 37. Parameters controlling security checking

CMAS (CICSplex SM parameter)	MAS (CICS system initialization parameter)	Explanation
SEC(YES)	SEC=YES	Both view selection checking and simulated security checking can occur, depending on the settings in the CICSSYS and CPLEXDEF views. This means that after CICSplex SM determines whether a user can display a particular view, simulated security determines what information can be provided in the view.
SEC(YES)	SEC=NO	View selection occurs; simulated security checking <b>does not</b> occur even if it is requested in the CICSSYS or CPLEXDEF views. This means that after CICSplex SM determines whether a user can display a designated view, no simulated security checking is performed to determine what information is to be provided in that view.
SEC(NO)	SEC=YES	CICSplex SM does not allow the MAS to connect to the CMAS. This prevents a MAS that has requested security from connecting to a CMAS that cannot provide security.

**Note:** CICSplex SM honors any of the system initialization parameters XCMD, XDB2, XDCT, XFCT, XHFS, XJCT, XPCT, XPPT, and XRES; that is, CICSplex SM includes or excludes the designated commands and resources from security checking. For each MAS, you can specify YES, NO, or CLASS NAME for these system initialization parameters. However, for the CMAS, you *must* specify NO for each of these system initialization parameters.

## Refreshing RACF profiles for CICSplex SM

CICSplex SM uses a cached copy of RACF information in order to reduce unnecessary I/O to the RACF database. When you change a RACF definition, you will, in some situations, need to force the CMAS to refresh the cached information.

### About this task

This includes refreshing general resource profiles and user profiles in the cache.

### Procedure

#### Refreshing general resource profiles in the cache

A CMAS requests RACF to create a cached copy of its general resource profiles during initialization:

- During CMAS initialization, global copies of the RACF profiles in the CPSMOBJ (including GCPSMOBJ) and CPSMXMP are created.
- During MAS initialization, the MAS determines which CICS security classes are in use, using the information specified in the XCMD, XDB2, XDCT, XFCT, XJCT, XPCT and XPPT system initialization parameters. This information is passed to the CMAS where it is used to create global copies of the relevant RACF profiles.

Perform the following steps to refresh the general resource profiles in the cache.

1. To identify the profiles for which global copies exist, issue the RACF command **SETROPTS LIST**.  
The "GLOBAL=YES RACLIST ONLY" section of the output shows which general resource classes have been globally copied.
2. Whenever a change is made to one of these classes (for example, with the **RALTER**, **RDEFINE**, **RDELETE** or **PERMIT** commands), you must refresh the cache.

Use the following RACF command:

```
SETR RACLIST(classname) GENERIC(classname) REFRESH
```

In a multi-CMAS environment, this command will refresh the cache only on the z/OS images that share the same RACF database.

3. If other CMASs run on other z/OS images that do not share the same RACF database, repeat the **SETR** command on the other images.

### Refreshing user profiles in the cache

A CMAS stores security information for a user ID in the cache when it performs a security check. By default, the information is retained in the cache until the user has remained inactive in the CMAS for the time specified by the **SECTIMEOUT** CICSplex SM parameter and the CMAS has performed a user ID timeout check.

When a change is made to a user ID (for example, with the **CONNECT**, **REMOVE**, **ALTUSER**, or **DELUSER** commands), the change becomes visible when the CMAS discards security information for the user from the cache.

#### 6.2

If **RACFSYNC=YES** is specified for the CMAS region, the CMAS listens for and processes type 71 ENF events. Security information for the affected user IDs is rebuilt the next time the user IDs are used, irrespective of the setting of the **SECTIMEOUT** parameter.

4. To force a CMAS to discard security information from the cache before timeout processing has occurred, use one of the following actions:

- For a single CMAS, use the **RESET USERID** action on the CMAS object.
- For more than one CMAS, use the **RESET USERID** action on the CMASLIST object.

Both actions are available from the WUI and the API.

Perform the action when a user has previously used the CMAS and you do not want to wait for normal timeout processing to occur.

5. To force a CMAS to immediately check for user IDs that are eligible for timeout, use one of the following actions:

- For a single CMAS, use the **PURGE** action on the CMAS object.
- For more than one CMAS, use the **PURGE** action on the CMASLIST object.

Both actions are available from the WUI and the API.

## CICSplex SM security checking sequence

A user can issue a single CICSplex SM command that causes data to be gathered about or an action to be performed against one or more CICS systems comprising a CICSplex. These CICS systems can reside in different MVS images.

When a user issues a request, the request is directed to the CMAS that manages the target CICS system or systems. [Figure 156 on page 445](#) and [Figure 157 on page 446](#) are flowcharts showing the procedure followed by CICSplex SM to evaluate the security requirements of a request from a user. Here is a description of that procedure:

1. CICSplex SM determines whether CICSplex SM rules allow the request to be processed.
  - If not, CICSplex SM terminates the request and issues an error message.
2. CICSplex SM determines whether simulated CICS security checking is to be performed.
  - If not, processing continues at “9” on page 444.
3. CICSplex SM determines whether the user is exempt from simulated CICS security checking.
  - If so, processing continues at “9” on page 444.
4. CICSplex SM determines whether simulated CICS command checking is to be performed.



- If not, processing continues at [“6” on page 444](#).
  - 5. CICSplex SM determines whether the user is allowed to process the command.
    - If not, CICSplex SM terminates the request and issues an error message.
  - 6. CICSplex SM determines whether the request is an action (not a request for information).
    - If not, processing continues at [“9” on page 444](#).
  - 7. CICSplex SM determines whether simulated CICS resource checking is to be performed.
    - If not, processing continues at [“9” on page 444](#).
  - 8. CICSplex SM determines whether the user is allowed access to information about the resource.
    - If not, CICSplex SM terminates the request and issues an error message.
  - 9. CICSplex SM performs the action or gets the information.
  - 10. CICSplex SM determines whether the request is an action (not a request for information).
    - If so, CICSplex SM returns the results of the action.
  - 11. CICSplex SM determines whether simulated CICS security checking is to be performed.
    - If not, CICSplex SM returns the requested information in the appropriate view.
  - 12. CICSplex SM determines whether the user is exempt from simulated CICS security checking.
    - If so, CICSplex SM returns the requested information in the appropriate view.
  - 13. CICSplex SM determines whether simulated CICS resource checking is to be performed.
    - If not, CICSplex SM returns the requested information in the appropriate view.
  - 14. CICSplex SM determines whether the user is allowed access to information about the resource.
    - If not, CICSplex SM excludes the requested information from the appropriate view.
  - 15. CICSplex SM determines whether information for another resource is requested.
    - If so, processing continues at [“14” on page 444](#).
    - Otherwise, CICSplex SM returns the requested information in the appropriate view.
- No further security checking is required.



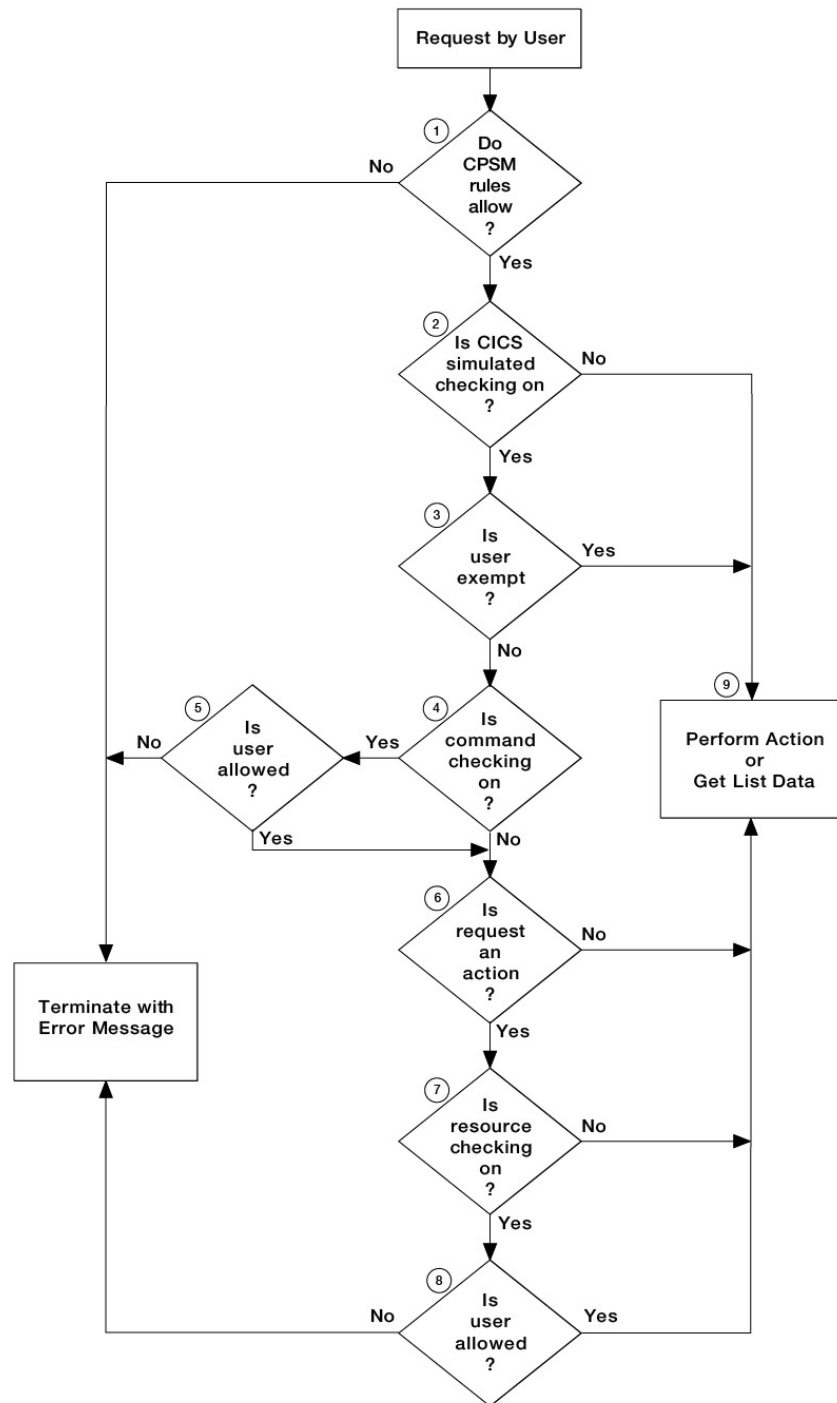


Figure 156. Flowchart of CICSPlex SM security checking sequence - part 1

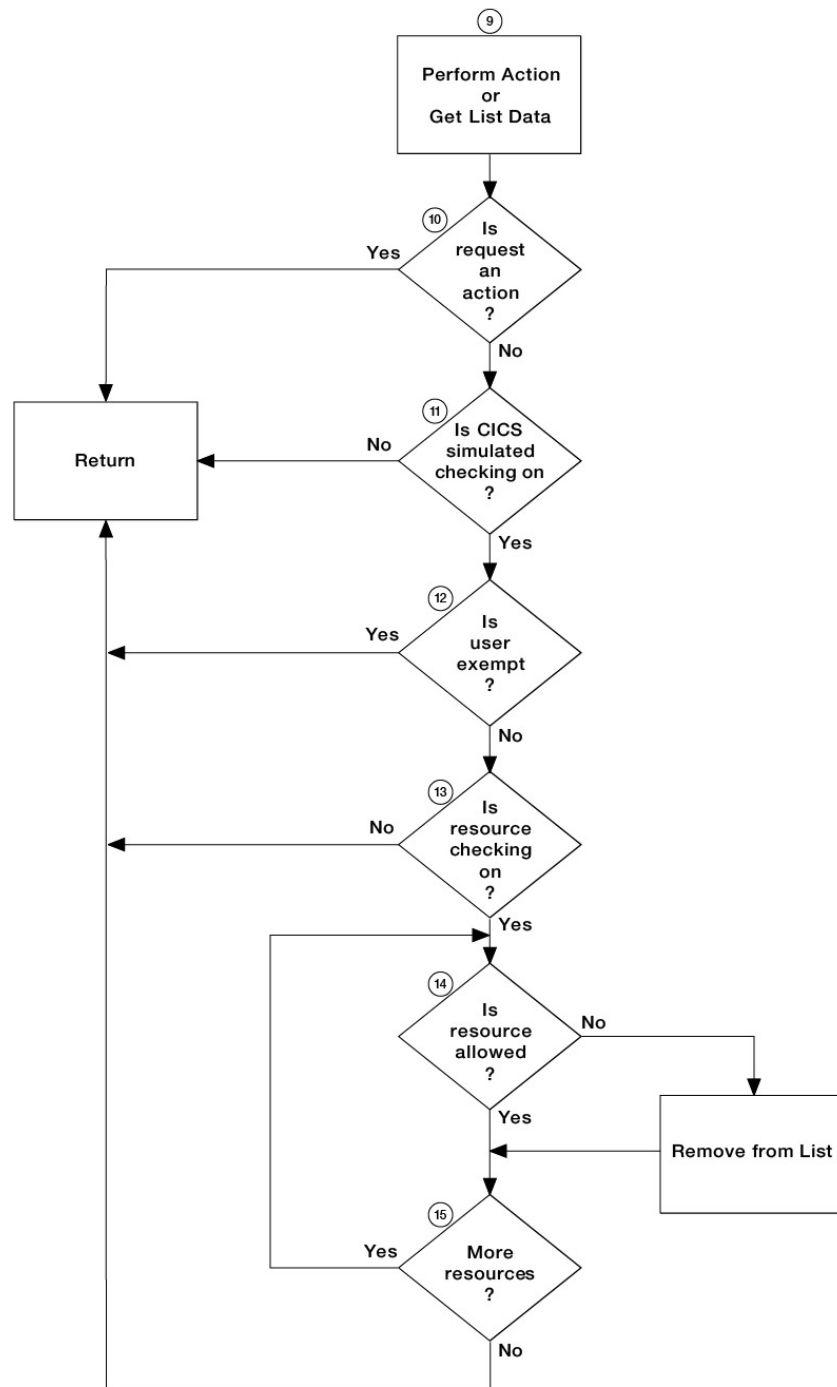


Figure 157. Flowchart of CICSplex SM security checking sequence - part 2

## Invoking a user-supplied external security manager

---

CICSplex SM provides an interface to an external security manager (ESM), which can be user-supplied or can be Resource Access Control Facility (RACF).

On the return from any user-supplied program, CICSplex SM must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general-purpose registers restored (except for those which provide return codes or linkage information). For information about translation modes, see [z/Architecture Principles of Operation](#).

**Note:** This information is intended primarily for non-RACF users. For information about security processing using RACF, refer to [“Implementing CICSplex SM security”](#) on page 402.

### An overview of the CICSplex SM ESM interface

CICSplex SM security uses the RACROUTE macro to get the MVS system authorization facility (SAF) interface to route authorization requests to the ESM.

Usually, if RACF is present, the MVS router passes control to it. However, you can modify the action of the MVS router by invoking the router exit. The router exit can be used, for example, to pass control to a user-supplied or vendor-supplied ESM. If you want to use your own security manager, you must supply an MVS router exit routine.

The control points at which CICSplex SM issues a RACROUTE macro to route authorization requests are described in [“CICSplex SM security control points”](#) on page 448.

## Invoking an external security manager

CICS provides an interface to an external security manager (ESM), which can be RACF, a vendor product, or user-written. CICS security uses a base component of z/OS, the System Authorization Facility (SAF) to route authorization requests to the external security manager (ESM).

For information about RACF, see [How it works: Securing CICS with RACF](#). The information in this section is intended for users of other ESMs.

SAF builds an interface between CICS and the external security manager by using a z/OS system service called the *MVS router*. The System Authorization Facility and the SAF router are present on all z/OS systems, even if no ESM is installed. Resource managers, such as CICS, call the MVS router as part of certain decision-making functions in their processing, for example access control checking and authorization-related checking. These functions are called *control points*. Using the MVS router, SAF routes the questions to the ESM and routes the answer back to the resource manager. The final decision on whether access is granted, is made by the resource manager, not by SAF nor by the ESM.

For more information about the MVS router, see [ICHRTX00 - MVS Router Exit in z/OS MVS Installation Exits](#) and [z/OS Security Server RACF Messages and Codes](#).

Typically, if RACF is present, the MVS router passes control to the RACF router, which in turn invokes the appropriate RACF function. However, you can modify the action of the MVS router by invoking the router exit. The router exit can be used, for example, to pass control to a user-written or vendor-supplied ESM. If you want to use your own security manager, you must supply an MVS router exit routine.

When CICS invokes the ESM, it passes information about the current CICS environment, for use by an ESM exit program, in an *installation data parameter list*. How your exit programs access the installation data parameter list depends on whether or not your ESM is RACF. If you use an ESM exit program that issues a **RACROUTE REQUEST=AUTH** and the exit is running because of a security request from CICS, you must ensure that any CLASS which is referenced is listed in the RACLIST.

For more information, see [Customizing security processing](#).

## CICSplex SM security control points

All RACROUTE macros are issued from a CMAS. Macros required to support simulated CICS security checking are issued from the CMAS to which the target MAS is connected.

The following list summarizes the RACROUTE macros used by CICSplex SM to invoke the ESM, and the control points at which they are issued.

### **RACROUTE**

The "front end" to the macros described, it invokes the MVS router. If RACF is not present on the system, RACROUTE can route to an alternative ESM, via the MVS router exit.

### **RACROUTE REQUEST=VERIFY**

Issued at user signon (with the parameter ENVIR=CREATE), and at user sign-off (with parameter ENVIR=DELETE) to a CMAS. Sign-off calls are made when the window is closed. This macro creates or destroys an access control environment element (ACEE). It is issued at the following CICSplex SM CMAS control points:

- API CONNECT thread creation
- Single system image command routing
- API DISCONNECT thread termination

### **RACROUTE REQUEST=FASTAUTH**

Issued during resource checking, on behalf of a user who is identified by an ACEE. It is the high-performance form of REQUEST=AUTH, using in-storage resource profiles, and is issued at the following CICSplex SM CMAS control points:

- Simulated CICS security checking
- View selection / API security
- May be called to perform logging and auditing

### **RACROUTE REQUEST=AUTH**

This is a higher path length form of resource checking and is issued during PLEXMGR security checking. It is also used by the batched repository-update facility (BATCHREP) to perform access checks against input and output datasets.

### **RACROUTE REQUEST=LIST**

Issued to create and delete the in-storage profile lists needed by REQUEST=FASTAUTH. (One REQUEST=LIST macro is required for each resource class.) It is issued at the following CICSplex SM CMAS control points:

- When CICSplex SM security is being initialized for a MAS
- When the CMAS or CMASD security action command (SEC) is issued.

For a detailed description of these macros, see [z/OS Security Server RACF Macros and Interfaces](#).

## Example RACF commands for securing CICSplex SM

---

The following examples show typical security setup tasks that you can use as a model to secure your CICSplex SM environment. The first example describes creating some RACF profiles to protect all CICSplex SM functions and resources. The other examples show how to further permit access selectively to particular users of particular resources.

Here are some general points that apply to all of the RACF examples:

- Each RACF command shown in these task examples must be issued once against every RACF database in your CICSplex SM configuration. So, if there are two unconnected RACF databases, one on MVS1 and one on MVS2, each RACF command must be issued twice (once on each system).
- In all of the RACF command examples, strings in lowercase must be replaced by values suitable for your own enterprise. For example, you must replace the string `admin_user` with the USERID of the administrator responsible for security of the relevant CICSplex SM resources.

- All of the RACF task examples use the enhanced generic naming facility (\*\*) of RACF. If you do not use this convention at your enterprise, see the RACF documentation for information about creating equivalent profiles.
- Operations and administration RACF groups are used in these examples and are recommended for your configuration too.

## Example: Protecting all CICSplex SM resources

To create the RACF profile that protects all CICSplex SM resources, complete the following steps:

1. Ensure that the CPSMOBJ class is active and that generic profiles can be defined:

```
SETROPTS CLASSACT(CPSMOBJ) GENERIC(CPSMOBJ)
```

2. Create a RACF profile that protects all views and action commands for all CICSplex SM functions:

```
RDEF CPSMOBJ ** UACC(NONE) OWNER(admin_group) NOTIFY(admin_user)
```

This command defines a profile (\*\*) that RACF treats as matching all CPSMOBJ resource entity names, and which therefore protects all CICSplex SM resources. It also specifies that admin\_user is to be notified of any violations.

3. Similar to Step “2” on page 449, define one RACF profile for each CICSplex in the configuration. Each profile will protect all CICSplex SM functions and resources for that CICSplex. The purpose of doing this is to give you more flexibility in granting access to CICSplex-specific resources. In this example, two CICSplexes exist, so two RACF profiles are created:

```
RDEF CPSMOBJ *.*.PLXPROD1.* UACC(NONE) OWNER(admin_group) +  
  NOTIFY(admin_user)  
RDEF CPSMOBJ *.*.PLXPROD2.* UACC(NONE) OWNER(admin_group) +  
  NOTIFY(admin_user)
```

**Note:** You cannot replace Step “2” on page 449 with multiple CICSplex-specific profiles: Such profiles will not necessarily protect CICSplexes that you create later, nor can they protect CICSplex SM functions whose context is the CMAS rather than the CICSplex. For example, the CONFIG views would be unprotected if you did not also perform Step “2” on page 449.

4. In Step “3” on page 449, all CICSplex SM functions and resources are protected at the CICSplex level. In this step, define profiles to control access to the CICSplex SM CONFIG and TOPOLOGY definition functions, so that you can selectively permit any "special" users, such as administrators, the access they need. Anyone who has update access to these two functions can alter the CICSplex configuration, so access must be limited.

```
RDEF CPSMOBJ CONFIG.DEF.** UACC(NONE) OWNER(admin_group)  
RDEF CPSMOBJ TOPOLOGY.DEF.** UACC(NONE) OWNER(admin_group)
```

Now that you've controlled access to CICSplex SM functions and resources, you can begin to grant access to particular users or groups of users.

## Example: Giving CICSplex SM operators appropriate authorizations

CICSplex SM operators need access, at least, to all of the operations views. This example shows how to give CICSplex SM operators update access to all operations views and read access to the monitoring views. This will allow operators to look at monitor data, but not to create or change monitor definitions.

1. Give CICSplex SM operators update access to the OPERATE views:

```
RDEF CPSMOBJ OPERATE.** OWNER(admin_group) UACC(NONE)  
PE OPERATE.** CLASS(CPSMOBJ) ID(ops_group) A(UPDATE)
```

2. Give CICSplex SM operators read access to the MONITOR views:

```
RDEF CPSMOBJ MONITOR.** UACC(NONE) OWNER(admin_group)  
PE MONITOR.** CLASS(CPSMOBJ) ID(ops_group) A(READ)
```

In both steps, you begin by creating a RACF profile to protect the resource, and then grant access to users in group ops\_group.

### **Example: Giving a user read access to all transactions on MVS system A**

This example shows how to give user PAYUSR1 read access to all transactions running on CICS systems on MVS system A.

In the example, three CICS systems (say, CICSAA01, CICSAA02, and CICSAA03) exist, which all belong to CICSplex PLXPROD1.

1. Define the appropriate RACF profile:

```
RDEF CPSMOBJ OPERATE.TRAN.PLXPROD1.CICSAA0* UACC(NONE) +  
OWNER(admin_group)
```

2. Give user PAYUSR1 read access to all transactions on MVS system A:

```
PE OPERATE.TRAN.PLXPROD1.CICSAA0* CLASS(CPSMOBJ) I(PAYUSR1) A(READ)
```

---

## Chapter 19. Security for Node.js applications

Node.js applications typically handle user authentication themselves, and do not often pass on user credentials to the back-end systems they interact with. Therefore, in most cases securing a Node.js application is handled in the same way when it is running in CICS as when it is running in any other environment.

Node.js applications in CICS always run under the CICS region user ID. This is important for Node.js applications that interact with the file system. The CICS region user ID must have the correct permissions for any files that are accessed by the Node.js application.

### Security for invoke requests to CICS

CICS tasks that are started by using a locally optimized invoke request from a Node.js application that is running in CICS will:

1. Run (by default) under the CNJW transaction ID
2. Run (by default) under the CICS region's default user ID (typically CICSUSER)

Both the default transaction ID and the default user ID can be modified, on a per URI basis, by using a URIMAP resource. The URI is passed to the invoke function by the Node.js application. For example, if a URIMAP is installed which indicates that requests for path `/examples/updateAccount` are to be mapped to transaction ID *TEST* and user ID *WORKER* then any work that is started for that URI path will override the default values as requested. In this example, the Node.js application must supply a URI with path `/examples/updateAccount` as a parameter to the invoke function. Therefore, the complete URI passed to the invoke function might be `http://example.org:12345/examples/updateAccount`.

If the URIMAP indicates that the target transaction ID is *CPIH* (which is the default value when *USAGE(PIPELINE)* is specified on a URIMAP), then any tasks for that URI invoked from Node.js will run under the CNJW transaction ID.

There is no mechanism for specifying an alternative user ID for each individual request to the same URI.

For more information on using the invoke function, see [Calling CICS services](#).





---

## Chapter 20. Security for CICS web support

When CICS is connected to the Internet, apply security measures to prevent unauthorized access to CICS applications and data and also to prevent third parties obtaining private information.

Consider security throughout the development process for your CICS web support architecture, as part of the design of your CICS web support applications and utility programs, and when creating resource definitions for the relevant CICS facilities. The subtopics summarize the measures that you can use to enhance the security of your CICS web support implementation.

---

### User IDs for access to document templates and z/OS UNIX files used by CICS Web support

---

When resource security is active for a transaction, the external security manager checks whether the user ID associated with the transaction is authorized to access the required resources. For CICS Web support, the user ID associated with the transaction for a particular Web request can be obtained from different sources. Depending on the level of security that you require, you can arrange your CICS Web support architecture to determine the user IDs that are used for resource security checking against the secured document templates or z/OS UNIX files.

#### Application-generated responses

For CICS Web support, the transaction for application-generated responses is an alias transaction, which can be specified in the URIMAP definition for the request or set by an analyzer program, and defaults to CWBA.

- **6.2** CWBA is defined as CMDSEC(YES) and RESSEC(YES), which means the application security is enabled.
- **6.1** CWBA is defined as CMDSEC(NO) and RESSEC(NO) so, if you require resource security for the alias transaction, you must either copy the CWBA definition to your own group and change its RESSEC attribute, or use a different alias transaction.

If you want to use alternative alias transaction names, see [Creating TRANSACTION resource definitions for CICS web support](#).

When a Web client makes a request to CICS Web support, and the response is provided by an application, CICS selects a userid for the alias transaction in the following order of priority:

1. A user ID that you set using an analyzer program. This user ID can override a user ID obtained from the Web client or supplied by a URIMAP definition.
2. A user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. If authentication is required for the connection but the client does not provide an authenticated user ID, the request is rejected.
3. A user ID that you specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

For application-generated responses, the user ID selected for the Web client applies to the whole alias transaction, and must be authorized to attach the transaction and use the Web application program, as well as to use secured document templates.

Web clients' user IDs do not need specific permissions on z/OS UNIX files for application-generated responses, because applications can only manipulate z/OS UNIX files using EXEC CICS commands when the files are defined as CICS document templates. Security checking is only carried out for the CICS document template, and not again for the z/OS UNIX file.

## Static responses

The transaction for all static responses specified in URIMAP definitions is the default Web attach task CWXN, or any alias transaction that you have specified in place of CWXN using the TRANSACTION attribute on your TCPIP SERVICE definitions.

When a Web client makes a request to CICS Web support, and the response is a static response specified in a URIMAP definition, the user ID used for the Web client is a user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client.

Resource security checking for document templates is controlled by the XRES system initialization parameter and the RESSEC attribute for the transaction (CWXN or its alias). Access control for z/OS UNIX files is controlled by the XHFS system initialization parameter only.

The user ID for the Web client is used only during the process of resource security checking for the document template or z/OS UNIX file that is to be delivered as the static response. The user ID must be authorized to access the document template or z/OS UNIX file.

## CICS as an HTTP server: authentication and identification

---

For CICS as an HTTP server, you specify authentication schemes by the AUTHENTICATE attribute of the TCPIP SERVICE definition. Identification is obtained in connection with the authentication process, or it can be supplied by CICS if authentication is not needed.

Obtaining authentication and identification from web clients is a key step in protecting your CICS system from access by unauthorized users.

Use TCPIP SERVICE resource definitions to specify the security measures that are applied for CICS as an HTTP server. For each port that you use for CICS web support, the TCPIP SERVICE resource definition specifies these attributes:

- Whether or not SSL is used for the port
- The authentication scheme that is used for the port
- The realm for basic authentication

### Authentication

Two authentication schemes are supported by CICS for use with the HTTP protocol:

- **Basic authentication** is part of HTTP that enables a client to authenticate and identify itself to a server by providing a user ID and password or password phrase. This information is encoded using base-64 encoding, which is simple to decode. Therefore, using basic authentication as the sole means of authentication is appropriate only when the password cannot be intercepted. In most environments, use basic authentication with SSL, so that SSL encryption protects the user ID and password information.
- **TLS client certificate authentication** is a more secure method of authenticating a client, using a client certificate that is issued by a trusted third party (or Certificate Authority), and sent using TLS encryption. A client certificate does not contain a user ID that can be used for identification in CICS. To achieve identification, you can associate the client certificate with a user ID in RACF or an equivalent security manager, either before the certificate is used, or automatically (using basic authentication) when the client makes its request. The RACF user ID becomes the client user ID each time the certificate is used, as described in [Associating a RACF user ID with a certificate](#).

[Creating TCPIP SERVICE resource definitions for CICS web support](#) tells you how to set up a TCPIP SERVICE definition for CICS web support that specifies one of these authentication schemes.

When you use basic authentication or client certificate authentication, CICS handles the process of requesting authentication from the user, decoding the authentication information if necessary, checking the supplied authentication against the security manager database, and rejecting the request if the authentication is not acceptable. An analyzer program or user-written application program is called only after the authentication is verified and accepted.

All the user IDs used by web clients must have a user profile in RACF or your equivalent external security manager. Refer to [RACF classes and profiles](#) for resources.

**Note:** CICS uses password verification to verify a user ID during the processes described here. CICS enforces a full verification request once a day for each user ID that is used to log on to the CICS region. The full verification request using the RACROUTE REQUEST=VERIFY macro makes RACF record the date and time of last access for the user ID, and write user statistics.

For basic authentication, if the password or password phrase supplied by the user has expired, CICS prompts the user for a new password or password phrase and helps the user to resubmit the request. The CICS-supplied utility program DFHWBPW is used. You can customize the text on the web pages that CICS displays to the user during this process, as described in [“Password expiry management for HTTP basic authentication”](#) on page 459.

For client certificate authentication, CICS verifies the supplied certificate by checking it against the security manager database, and, optionally, against any certificate revocation list that you have set up. A user-written application can examine information obtained by this process, if this information is useful for determining how to process the request. Use the EXTRACT CERTIFICATE command to retrieve these items:

- Components of the issuer's or the subject's distinguished name.
- The RACF user ID associated with the certificate.

## Identification

Identification takes place when you obtain a user ID for the web client. The ID is obtained from the web client:

- During basic authentication
- By the association of a user ID with a client certificate

For application-generated responses only, CICS can supply a user ID on behalf of the web client:

- In an analyzer program that is used in the processing path for the application-generated response. (This ID can override a user ID obtained for the web client.)
- In the URIMAP definition for the request. (This ID cannot override a user ID obtained for the web client.)
- As the CICS default user ID, if no other can be determined.

Note that, if you supply a user ID on behalf of the web client, the identity of the client is not authenticated. Supply a user ID only when communicating with your own client system, which has already authenticated its users and communicates with the server in a secure environment.

When the client has been identified, the client user ID can be authorized for access to CICS resources like any other user ID, using RACF or an equivalent external security manager. You can choose to apply resource-level security to any or all of the individual resources that the web client is accessing in CICS, such as web pages stored as CICS document templates, or z/OS UNIX files, or CICS commands used by the application that provides the response. [“CICS system and resource security for CICS web support”](#) on page 462 explains how to secure these resources and how to remove resource level security if you do not want it.

## CICS as an HTTP client: authentication and identification

---

When you make an HTTP client request through CICS, a server or proxy might require you to perform basic authentication, proxy authentication, or TLS client certificate authentication.

You can perform basic authentication using the AUTHENTICATE option of your WEB SEND or WEB CONVERSE command. Your user application carries out proxy authentication. You supply a client certificate using a URIMAP definition.

Your client application might be asked to authenticate itself in the following ways:

- **Basic authentication** allows you to provide a user name and password for access to specific information. When you make a request to a server, the server might send you a response with a 401 status code, and a WWW-Authenticate header. The header names the realm for which basic authentication is required. To receive the information you requested, provide the user name and password, and CICS resends the request with an Authorization header, specifying your user name and password, to allow you access to the realm. CICS can also send an Authorization header directly to a server that is expecting it, thus eliminating the need for a 401 response. CICS converts the user name and password to ASCII and applies base-64 encoding, as required by the basic authentication protocol. So you can supply your credentials in normal characters through the WEB SEND or WEB CONVERSE command, or through the XWBAUTH user exit. See [Providing credentials for basic authentication and HTTP basic authentication](#).
- **Proxy authentication** is initiated by a proxy server. For proxy authentication, the status code for the response is 407, the challenge header from the proxy server is Proxy-Authenticate, and the response header is Proxy-Authorization. CICS does not support this protocol.
- **TLS client certificate authentication** uses a client certificate, which is issued by a trusted third party (or Certificate Authority). A server might require you to provide this authentication when you are making an HTTPS request. [Configuring CICS to use SSL](#) tells you how to obtain a certificate and store it in a key ring in the RACF database or equivalent external security manager. If a server does request a client certificate, CICS supplies the certificate label, which is specified in the URIMAP definition that was used on the WEB OPEN command for the connection. Alternatively, you can directly specify the certificate label as an option in the WEB OPEN command. If you use a URIMAP definition but do not specify a certificate label, the default certificate defined in the key ring for the CICS region user ID is used.

Some servers might ask you to provide other types of authentication or identification. If you cannot provide acceptable authentication or identification to a server, your request is rejected. For basic authentication or proxy authentication, the status code used when a server rejects your request is the same as the status code for the challenge (401 for a server or 407 for a proxy). If you respond to a challenge but then receive a further response with one of these status codes, the authorization information that you used is not valid.

## Identifying HTTP users

Identification is the process by which the identity of a user is established. This is how a user's identity is established for the HTTP application protocol.

### About this task

For the HTTP application protocol, you can identify the user in the following ways:

- A user ID can be obtained from the web client using HTTP basic authentication.
- If the web browser sends a client certificate, you can use a user ID that is associated with the certificate.

You can associate a certificate with a RACF user ID in two ways:

- You can use RACF commands to associate a certificate with a user ID.
- CICS can automatically issue the RACF commands to associate a certificate with a user ID (which is obtained from the Web client using HTTP basic authentication).

[“Associating a RACF user ID with a certificate” on page 171](#) tells you how to do this.

For application-generated responses only, it is also possible for CICS to supply a user ID on behalf of the web client:

- In an analyzer program that is used in the processing path for the request.
- In the USERID attribute of the URIMAP definition for a request.
- As the CICS default user ID.

If you use a URIMAP definition or analyzer program to set a user ID that has not been supplied by a client, or allow the CICS default user ID to be used, there is no authentication of the client's identity. Only do

this when communicating with your own client system, which has already authenticated its users, and communicates with the server in a secure environment.

When the HTTP response is to be provided by an application (an application-generated response), the order of precedence of user IDs is:

1. A user ID that you set using an analyzer program. This user ID can override a user ID obtained from the Web client or supplied by a URIMAP definition.
2. A user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. If authentication is required for the connection but the client does not provide an authenticated user ID, the request is rejected.
3. A user ID that you specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

When the HTTP response is to be provided by a URIMAP definition that specifies a CICS document template or z/OS UNIX file (a static response), the user ID used for the Web client is a user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. For static responses, it is not possible to supply a user ID on behalf of the Web client, nor to override an authenticated user ID obtained from a Web client.

For static responses, CICS only makes use of a supplied user ID if you specify resource security checking for the transaction. No default user ID is required for static responses. If the Web client does not supply a user ID, no resource security checking is carried out, even if resource security is active for the transaction.

**Note:** CICS uses password verification to verify a user ID during the processes described here. CICS enforces a full verification request once a day for each user ID that is used to log on to the CICS region. The full verification request using the RACROUTE REQUEST=VERIFY macro makes RACF record the date and time of last access for the user ID, and write user statistics.

The method used to identify the user is determined by the AUTHENTICATE and SSL attributes of the TCPIP SERVICE definition:

<i>Table 38. How the user of an HTTP client is identified</i>		
<b>AUTHENTICATE</b>	<b>SSL</b>	<b>How the user is identified</b>
NO	NO or YES	The client does not supply a user ID. It can be supplied by an analyzer program or URIMAP definition, or allowed to default to the CICS default user ID, if applicable.
NO	CLIENTAUTH, or ATTLSAWARE	<p>If the client sends a certificate that is associated with a user ID, then that user ID applies, unless it is overridden by an analyzer program.</p> <p>If the client sends a certificate that is not associated with a user ID, a user ID can be supplied by an analyzer program or URIMAP definition, or allowed to default to the CICS default user ID, if applicable.</p>
BASIC	all values	<p>If the client sends a certificate that is associated with a user ID, then that user ID applies, unless it is overridden by an analyzer program.</p> <p>If the client sends a certificate that is not associated with a user ID, then the user ID is obtained from the client, using HTTP basic authentication, and the user ID is registered to the certificate.</p> <p>If the client does not send a certificate, then the user ID is obtained from the client, using HTTP basic authentication and can be overridden by an analyzer program.</p>

Table 38. How the user of an HTTP client is identified (continued)

<b>AUTHENTICATE</b>	<b>SSL</b>	<b>How the user is identified</b>
CERTIFICATE	CLIENTAUTH, or ATTLSAWARE	<p>If the client sends a certificate that is associated with a user ID, then that user ID applies, unless it is overridden by an analyzer program.</p> <p>If the client sends a certificate that is not associated with a user ID, or does not send a certificate, then the connection is rejected.</p>
AUTOREGISTER	CLIENTAUTH, or ATTLSAWARE	<p>If the client sends a certificate that is associated with a user ID, then that user ID applies, unless it is overridden by an analyzer program.</p> <p>If the client sends a certificate that is not associated with a user ID, then the user ID is obtained from the client, using HTTP basic authentication, and the user ID is registered to the certificate.</p> <p>If the client does not send a certificate, then the connection is rejected.</p>
AUTOMATIC	NO or YES	A user ID is obtained from the client, using HTTP basic authentication. This can be overridden by an analyzer program.
AUTOMATIC	CLIENTAUTH, or ATTLSAWARE	<p>If the client sends a certificate that is associated with a user ID, then that user ID applies, unless it is overridden by an analyzer program.</p> <p>If the client sends a certificate that is not associated with a user ID, then the user ID is obtained from the client, using HTTP basic authentication, and the user ID is registered to the certificate.</p> <p>If the client does not send a certificate, then the user ID is obtained from the client, using HTTP basic authentication.</p>

**Note:**

1. This table does not list combinations of values for the AUTHENTICATE and SSL attributes that are invalid, and that cannot be specified in the TCPIP SERVICE definition.
2. If HTTP basic authentication is used, CICS verifies the password. If the password is invalid, the connection is rejected.

## Authenticating HTTP users

You can use HTTP basic authentication or TLS client certificate authentication to authenticate HTTP users.

### About this task

The authentication scheme is specified by the AUTHENTICATE and SSL attributes of the TCPIP SERVICE definition:

<b>Authentication scheme</b>	<b>AUTHENTICATE</b>	<b>SSL</b>	<b>Notes</b>
HTTP with no authentication	NO	NO, YES, CLIENTAUTH, or ATTLSAWARE	

Authentication scheme	AUTHENTICATE	SSL	Notes
HTTP with basic authentication	BASIC	NO, YES, CLIENTAUTH, or ATTLISAWARE	
HTTP with basic authentication	AUTOMATIC	NO, YES, CLIENTAUTH, or ATTLISAWARE	If SSL(CLIENTAUTH   ATTLISAWARE) is specified, and the client sends a certificate, then SSL client certificate authentication is used.
HTTP with SSL client certificate authentication	CERTIFICATE or AUTOREGISTER	CLIENTAUTH or ATTLISAWARE	If the client does not send a certificate, the connection is not established.
HTTP with SSL client certificate authentication	AUTOMATIC	CLIENTAUTH or ATTLISAWARE	If the client does not send a certificate, then basic authentication is used.

## Password expiry management for HTTP basic authentication

When basic authentication is used for an HTTP connection, CICS web support checks the user ID and password in the external security manager. If the password has expired, the CICS-supplied utility program DFHWBPW is used to prompt the user to select a new password. You can customize or replace the pages presented to the user by DFHWBPW.

DFHWBPW is used only for password expiry management when the TCIPSERVICE definition that applies to the request is defined with the BASIC, AUTOREGISTER, or AUTOMATIC option for the AUTHENTICATE attribute. Although DFHWBPW has a structure similar to a converter program, it is not part of the normal CICS web support processing path, so you do not need to add code to it for any other purpose. When the user has selected a new password, DFHWBPW restarts the request submission by redirecting the client to the URL for the original request, so that the complete processing path for the request occurs as normal.

DFHWBPW presents two web pages to the user:

1. Password prompt page. This page contains two elements:
  - a. A message about password validity. The initial message displayed to the user states that the password has expired. A user ID can have both a standard password and a password phrase. Passwords between 9 and 100 characters in length are password phrases; passwords of 8 characters or less are standard passwords. Standard passwords and password phrases operate independently of each other. If a standard password has expired, it must be replaced with a new standard password. Similarly, if a password phrase has expired, it must be replaced with a new password phrase. If the user's attempt to change the password fails (for example, the two supplied copies of the new password do not match), further messages are displayed to explain the problem.
  - b. An HTML form for the user to change the password.
2. Confirmation and request refresh page. This page confirms that the expired password has been successfully replaced, and provides a refresh tag and URL link so that the request can be remade automatically or manually.

DFHWBPW builds these web pages using three CICS document templates: DFHWBPW1, DFHWBPW2, and DFHWBPW3. The CICS-supplied definitions for these templates define them as loadable programs; that is, they are of type PROGRAM(DFHWBPW1) and so on. The definitions are in the CICS-supplied resource definition group DFHWEB. You can change these definitions by copying them to another group and using the resource definition ALTER command to change them so that the templates are derived from a different source. Alternatively, you can leave the resource definitions unchanged, and modify the programs that are loaded instead. The three programs DFHWBPW1, DFHWBPW2, and DFHWBPW3 are



assembler language data-only modules, and their source is shipped to you in corresponding members of the CICS sample library, SDFHSAMP. You can modify these samples and reassemble and link-edit them into one of your normal CICS program libraries that are concatenated into the DFHRPL data definition statement.

**Note:** To avoid unnecessary output, ensure that you specify the **NOPROLOG** and **NOEPILOG** parameters when assembling DFHWBPW1, DFHWBPW2 and DFHWBPW3.

The content and function of each of the DFHWBPW templates is as follows:

#### **DFHWBPW1**

Part of the password prompt page. Provides the HTML page heading for the page, and sets symbols for the possible password validity messages (using the server-side include technique for setting symbols). The messages provide the following information:

##### **message.1**

Password has expired.

##### **message.2**

The entered user ID is invalid.

##### **message.3**

The two copies of the proposed new password do not match.

##### **message.4**

The previous password entered (the one that has just expired) is not correct.

##### **message.5**

The proposed new password is not permitted by the external security manager, because of password quality rules.

##### **message.6**

The user ID is now revoked.

The DFHWBPW program selects the appropriate symbol to insert into the document for the password prompt page. You can customize DFHWBPW1 to change the page heading and title, or alter the body tag to change the page colors or background. You can also change the content of the message symbols.

#### **DFHWBPW2**

Part of the password prompt page. Builds an HTML form where the user can input a user ID, the old (expired) password (or password phrase), and two identical copies of a proposed new password. You can customize DFHWBPW2 to change the text used to prompt the user, or otherwise change the layout of the page. However, you must not modify the contents of the form tag, or any of the input tags. If you do, DFHWBPW might not work as intended.

#### **DFHWBPW3**

Confirmation and request refresh page. The text notifies the user that the expired password was successfully replaced, and explains that the user will shortly be prompted by the client to enter the password again. You can customize the text and layout of the page.

DFHWBPW3 restarts the request process. It contains a meta http-equiv="Refresh" tag that causes an automatic redirection after ten seconds to the page that the user had originally requested when the expired password was detected. You can change the time limit on this tag or remove it if you do not want users to be redirected automatically. However, the modified page must always contain a link forward to the originally requested page. The URL for that page is in the symbol &dfhwpw\_target\_url; . Restarting the request process means that, if the web client has cached the old password, it can be replaced with the new password immediately, and also means that the CICS web support processing path is unaffected.

## **Support for HTTP strict transport security (HSTS)**

---

HTTP strict transport security (HSTS) is a mechanism that helps prevent HTTP servers from certain man-in-the-middle attacks. It allows servers to instruct compliant user agents, such as web browsers, to only interact with the server through secure connections (HTTPS).



HSTS is implemented by the server returning a Strict-Transport-Security header in its HTTP responses. This header indicates the period of time that the user agent should only interact securely with the server and whether this behavior extends to any sub-domains.

An example HSTS header is:

```
Strict-Transport-Security: max-age=86400; includeSubDomains
```

It instructs the user agent to interact with the server securely for 24 hours (86400 seconds) and extends that behavior to include all sub-domains. For example, if the initial request was to `example.com`, then future requests to `server1.example.com` and `server2.example.com` are to be made securely too.

HSTS is fully described in [RFC 6797: HTTP Strict Transport Security \(HSTS\)](#).

## Enabling support for HSTS in CICS

Configure CICS to support HSTS only if all inbound connections are secured and the CICS server is part of a fully secure domain. CICS provides a set of feature toggles for HSTS support.

The feature toggle settings apply to all installed TCPIP SERVICES that have `PROTOCOL(HTTP)` and also to the CMCI JVM server.

### Note:

- The feature toggles do not apply to HTTP requests received by Liberty JVM servers within CICS. See [Securing Liberty by using HTTP Strict Transport Security \(HSTS\)](#) instead.
- If a CICS application supplies its own HSTS header using **EXEC CICS WEB WRITE HTTPHEADER**, that header is used. CICS does not override it.

To configure HSTS at the region level, use the following feature toggles:

#### **com.ibm.cics.web.hsts.max-age=seconds**

Activates HSTS for the entire region and sets the max-age time of the HSTS header in seconds (0-99999999). One year is 31536000 seconds.

#### **com.ibm.cics.web.hsts.includesubdomains={true|false}**

Takes effect only when the `com.ibm.cics.web.hsts.max-age` feature toggle is also specified.

It controls whether to extend HSTS to the sub-domains of the CICS server. When `true` is specified, the `includeSubDomains` option is added to the HSTS header.

To gain granularity, you can configure HSTS for individual TCPIP SERVICES to override the region wide settings by using the following feature toggles.

However, the CMCI JVM server can use only the region wide HSTS setting.

#### **com.ibm.cics.web.hsts.max-age.TCIPS={seconds|-1}**

##### **TCIPS**

Specifies the name of an individual TCIPSERVICE to have a different HSTS setting from the region wide setting.

##### **seconds**

Specifies the max-age value for the TCIPSERVICE in its HSTS header.

##### **-1**

Disables HSTS for the specified TCIPSERVICE.

#### **com.ibm.cics.web.hsts.includesubdomains.TCIPS={true|false}**

Takes effect only when the `com.ibm.cics.web.hsts.max-age.TCIPS` feature toggle is also specified.

It controls whether to extend HSTS to the sub-domains of the specified TCIPSERVICE. When `true` is specified, the `includeSubDomains` option is added to the HSTS header of the TCIPSERVICE specified on `TCIPS`.

## CICS system and resource security for CICS web support

---

When CICS is an HTTP server, the CICS system must be protected from access by unauthorized users. If a system is not properly protected, users might be able to access confidential data or obstruct the system to cause denial of service to other users.

To police access to CICS web support in general, you request identification from each user that makes an HTTP client request and then authenticate the identity stated by the user. You use the TCPIP SERVICE definitions for inbound ports to specify these requirements. Refer to [“CICS as an HTTP server: authentication and identification”](#) on page 454.

All the user IDs used by web clients must have a user profile in RACF or your equivalent external security manager. Refer to [RACF classes and profiles for resources](#).

When you have obtained an authenticated user ID for a web client, you can use this ID to implement resource-level security for the resources in the CICS region that you are using to provide the response. The procedure varies for each type of response:

- Application-generated responses
- Static responses, using a URIMAP definition that provides a CICS document template as the response
- Static responses, using a URIMAP definition that provides a z/OS UNIX Systems Services file as the response

For application-generated responses, CICS system defaults specify that no resource security checking is carried out, but transaction security checking is carried out (specifically, transaction security for the alias transaction). Assuming that transaction security is active in your CICS region, you must therefore take some actions relating specifically to security for application-generated responses, even if you do not plan to use web client authenticated user IDs for security checking.

For static responses, transaction security does not apply to web client user IDs. However, CICS system defaults specify that resource-level security checking *is* carried out if a user ID is available for web clients. If you are obtaining authenticated user IDs from web clients, you must therefore either set up resource permissions for these user IDs or take action to disable resource-level security checking.

Whether or not you choose to implement resource-level security using web client user IDs for every response provided by CICS web support, you must provide the following protection:

- Implement measures to protect inbound ports against unauthorized or malicious access.
- Protect CICS system components from modification by unauthorized users, and ensure that authorized users have the correct access to them.

### Security for inbound ports

A TCPIP SERVICE resource definition defines each port used for CICS web support. The TCPIP SERVICE definition specifies security options for the port, including whether SSL is used and the level of authentication that is requested from clients. Ports must be guarded against unauthorized or malicious access.

[Creating TCPIP SERVICE resource definitions for CICS web support](#) explains how to create definitions for ports.

To help keep ports secure:

- Specify the MAXDATALEN attribute on every TCPIP SERVICE definition. This option limits the maximum amount of data that CICS accepts for a single request, and it helps to defend CICS against denial of service attacks involving the transmission of large amounts of data.
- Use Secure Sockets Layer (SSL) wherever you want to ensure that your interaction with the web client remains confidential and cannot be intercepted by a third party. The use of SSL is particularly important when confidential data is being transmitted or when authorization such as a user ID and password are being passed to the server. Refer to [“SSL with CICS web support”](#) on page 467.

If you do experience unusual activity on one or more of your CICS web support ports, use CICS system commands to shut down CICS web support at different levels (a single request, a virtual host, a port, or the whole of CICS web support), without shutting down the CICS system. Refer to [Rejecting HTTP requests](#).

URIMAP resource definitions name either HTTP or HTTPS as the scheme for the request. A URIMAP specifying HTTP accepts web client requests made using either HTTP or the more secure HTTPS. A URIMAP specifying HTTPS accepts only web client requests that are made using HTTPS.

When a URIMAP definition with HTTPS matches a request from a web client, CICS checks that the inbound port used by the request is using SSL. If SSL is not specified for the port, the request is rejected with a 403 (Forbidden) status code. When the URIMAP definition applies to all inbound ports, this check ensures that a web client cannot use an unsecured port to access a secured resource. No check is carried out for a URIMAP definition that specifies HTTP, so web clients can use either unsecured or secured (SSL) ports to access these resources.

## Security for CICS system components

As with any other CICS resource, you must protect CICS system components used in CICS web support from modification by unauthorized users. You must also ensure that authorized users, particularly the CICS region, have the required authority to use these components.

A number of components, such as application programs and resource definitions, are used to control CICS web support. Refer to [Components of CICS web support](#). If you do not secure these components against unauthorized access, the security of your CICS web support architecture might be compromised. For example, a user with access to the TCPIP SERVICE definition for a port might remove the requirement for a web client to use SSL or to provide identification. [How it works: Authorization in CICS](#) explains how to secure CICS transactions, resources, and commands against unauthorized use.

For some CICS system components, you might have to set up additional authorities to allow access to authorized users:

- For [URIMAP](#) resources, additional authority might be required to set a user ID for the web client. If surrogate user checking is enabled in the CICS region (with XUSER=YES specified as a system initialization parameter), CICS checks that the user ID used to install the URIMAP definition is authorized as a surrogate of the user ID specified for the USERID attribute.
- You can use document templates to produce the body of a response from CICS as an HTTP server, or the body of a request from CICS as an HTTP client. You define them by [DOCTEMPLATE](#) resource definitions. If the document templates are stored in partitioned data sets, the CICS region user ID must have READ authority for the data set.
- You can use z/OS UNIX Systems Services files to produce the body of a static response from CICS as an HTTP server. You can specify them under their own names or define them by [DOCTEMPLATE](#) resource definitions. When a z/OS UNIX file is used, the CICS region must have permissions to access z/OS UNIX, and it must have permission to access the z/OS UNIX directory containing the file, and the file itself. Refer to [Giving CICS regions access to z/OS UNIX directories and files](#).

## Resource and transaction security for application-generated responses

If you have obtained an authenticated user ID for a web client, which has a profile in your security manager, this user ID is applied to the alias transaction that is used for the application-generated response.

### About this task

You either give appropriate permissions to the web client user IDs or you supply your own standard user ID as an override. Whether or not you decide to use web client user IDs for resource security checking, you must ensure that the user ID for the alias transaction has the appropriate permissions.

You define alias transactions by TRANSACTION resource definitions. The alias transaction for each application-generated response is specified by the URIMAP definition for the request or by an analyzer

program. The default is the CICS alias transaction CWBA, which applies when either web-aware applications or COMMAREA applications are used to provide the response.

The user ID under which the alias transaction runs must have authority to perform these tasks:

- Attach the alias transaction, if transaction security is specified for the CICS region. Transaction-attach security is controlled by the system initialization parameter XTRAN. The default is YES (transaction-attach security is active).
- Access any CICS resources used by the alias transaction, if resource security is specified for the alias transaction. Resource security is controlled by the RESSEC attribute in the TRANSACTION resource definition for the alias transaction.

6.2 The default is YES (resource security). YES is also the supplied setting for CWBA.

6.1 The default is NO (resource security). NO is also the supplied setting for CWBA.

- Access any CICS system programming commands used by the alias transaction, if command security is specified for the alias transaction. These system programming commands are used in the user-written application program that produces the response. Command security is controlled by the CMDSEC attribute in the TRANSACTION resource definition for the alias transaction.

6.2 The default is YES (resource security). YES is also the supplied setting for CWBA.

6.1 The default is NO (resource security). NO is also the supplied setting for CWBA.

When a web client makes a request to CICS web support, and the response is provided by an application, CICS selects a user ID for the alias transaction in the following order of priority:

1. A user ID that you set using an analyzer program. This user ID can override a user ID obtained from the Web client or supplied by a URIMAP definition.
2. A user ID that you obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. If authentication is required for the connection but the client does not provide an authenticated user ID, the request is rejected.
3. A user ID that you specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

Depending on your CICS web support architecture, you might be using one or several of these types of user ID, for different requests. If you obtain an authenticated user ID for a web client, it is used for the alias transaction unless you take action to override it.

Take the following security actions for application-generated responses.

## Procedure

1. If you are obtaining authenticated user IDs for web clients, but you do *not* want to use these for security checking for your application-generated responses, you use an analyzer program to override web client user IDs with a standard user ID for the relevant alias transactions. (You can use the CICS default user ID.) Place the analyzer program in the processing paths for the requests where you want to supply this override.

See [Analyzer programs](#). Make sure that this user ID has a user profile defined in your security manager.

When you have set up a standard user ID, you can give the required permissions, as described in the remaining steps of this procedure, to the standard user ID.

2. If you are not obtaining authenticated user IDs for web clients, select suitable user IDs to be standard user IDs for your alias transactions. Unless you just want to use the CICS default user ID, specify your chosen user IDs in the URIMAP definitions for the requests, or set up an analyzer program to specify them.

Make sure that the standard user IDs have user profiles defined in your security manager.

3. Assuming that transaction security is specified for the CICS region, ensure that all the possible user IDs for your alias transactions have authority to attach the transaction.

All user IDs might include web client user IDs, if you are obtaining them and are not overriding them, or a standard user ID that you have specified in a URIMAP definition or analyzer program, or just the CICS default user ID. See [Transaction security](#).

4. Optional: Apply resource-level security checking for the resources used by an alias transaction:

- a) Identify all the CICS resources used by the alias transaction, and determine which of them are subject to resource security checking in your CICS region.

Here are some resources that might be used by an application program for CICS web support and the system initialization parameters that control resource security checking for them:

- CICS document templates (XDOC system initialization parameter)
- Other application programs invoked by the main application program to perform business logic (XPPT system initialization parameter)
- Temporary storage queues used to share application state across an HTTP request sequence (XTST system initialization parameter)
- Files managed by CICS file control (XFCT system initialization parameter)

Resource security checking for zFS files (XHFS system initialization parameter) does not apply when zFS files are used by an application program, because the files can be manipulated by an application program only when they are defined as CICS document templates, and CICS document template security controls access to them in this situation.

If you are using an analyzer program, it is the main program for the alias transaction, and so is not subject to resource security checking (only to the transaction security checking). However, note that the user-written web application program itself, and any converter program that you use, is subject to separate resource security checking. Similarly, if you are using a converter program but no analyzer program, the converter program is the main program for the alias transaction, but the application programs called by the converter program are subject to separate resource security checking.

- b) Give all the user IDs that are permitted to attach the alias transaction permission to use the secured resources used by the alias transaction.
  - c) Specify RESSEC(YES) in the TRANSACTION resource definition for the transaction.
5. Optional: To apply command security checking for any CICS system programming commands used by an alias transaction:
    - a) Confirm that command security is active in the CICS region. Command security is activated by the XCMD system initialization parameter.
    - b) Identify the CICS system programming commands used by the application program or programs, analyzer program (if used), and converter program (if used), that are associated with the transaction.

[CICS resources subject to command security checking](#) has a checklist of commands.
    - c) Give all the user IDs that are permitted to attach the alias transaction permission to use the commands used by the alias transaction.
    - d) Specify CMDSEC(YES) in the TRANSACTION resource definition for the alias transaction.

## What to do next

For any security checking to take place in a CICS region, set the SEC=YES system initialization parameter.

## Resource-level security for static responses using document templates

If you have implemented basic authentication or client certificate authentication, and you also want to control users' access to specific web pages, you can use web client authenticated user IDs to control access to individual CICS document templates that you are using to provide static responses.

### About this task

For static responses delivered by CICS web support using a CICS document template specified in a URIMAP definition, resource security checking is enabled by default.

The **XRES** system initialization parameter controls resource security for CICS document templates. The default for this parameter is YES, meaning that resource security is active. If you do not want to use resource security checking for CICS document templates used for any purpose in your CICS region, you can deactivate it by setting this system initialization parameter to NO.

The transaction for all static responses is the default web listener transaction CWXN, or any alternative transaction that you have specified in place of CWXN using the TRANSACTION attribute on your TCIPSERVICE definitions. For CICS document templates, you can also control resource security checking by the RESSEC attribute in the TRANSACTION resource definition. For CWXN, as supplied by CICS, RESSEC(YES) is specified, meaning that resource security is active. If you do not want to use resource security checking for static responses, the best way to deactivate it is to replace CWXN in your TCIPSERVICE definitions with an alternative transaction that specifies the program DFHWPBXN and has RESSEC(NO). This setting deactivates resource security checking for CICS document templates for static responses only. Note that the RESSEC attribute cannot control security checking for z/OS UNIX files specified by the HFSFILE attribute.

You can retrieve document templates from a variety of sources, including partitioned data sets, CICS programs, CICS files, z/OS UNIX System Services files, temporary storage queues, transient data queues, and exit programs. When resource security checking is carried out for a document template, CICS does not perform any additional security checking on the resource that supplies the document template, even if resource security is specified for that type of resource in the CICS region.

To set up resource-level security for static responses using CICS document templates:

### Procedure

1. Identify the authenticated user IDs used by web clients. These IDs must be the basis of your resource security checking. (You cannot supply an override using an analyzer program, as you can with application-generated responses.)

Authenticated user IDs already have a user profile defined in your security manager.

2. Identify all the CICS document templates that you are using to provide static responses.
3. Implement security for CICS document templates in your CICS region, following the instructions in [Resource security](#).

You must define a profile to your security manager for each CICS document template that you are using to provide a static response, and give permissions to access appropriate CICS document templates to each authenticated user ID.

4. Ensure that RESSEC(YES) is specified in the TRANSACTION resource definition of CWXN or in the alternative transaction that you have specified in place of CWXN.

This step activates resource security checking for your static responses, so ensure that, whenever a web client supplies a user ID, you have set up the appropriate permissions.

### What to do next

For any security checking to take place in a CICS region, set the SEC system initialization parameter to YES.



## SSL with CICS web support

---

You can use the Secure Sockets Layer (SSL) with HTTP to enable encryption, message authentication, and client and server authentication using certificates. When you have configured CICS to use SSL, its facilities are available for both CICS as an HTTP server, and CICS as an HTTP client.

[Configuring CICS to use SSL](#) tells you how to make SSL work with CICS.

When CICS is an HTTP server, you can use SSL to protect an interaction with a web client. You specify appropriate security options on the TCPIP SERVICE definition for the port on which CICS receives the client requests.

As well as specifying the use of SSL, you can require basic authentication or require a client certificate. To give more assistance to web clients, you can allow a client to provide a client certificate, and then register itself to the security manager to supply identification for the CICS environment. You can also allow a client to use self-registration or basic authentication as needed to supply identification. CICS handles all these activities, so, if you are providing an application-generated response, your application does not have to handle this registration. Refer to [Creating TCPIP SERVICE resource definitions for CICS web support](#).

When CICS is an HTTP client, a server might require the use of SSL for some connections. If that is the case, you need to perform some or all of these actions:

- Use HTTPS as the scheme for the connection.
- Supply a list of cipher suites that you want to use for the connection. You can specify them in the URIMAP definition that you use on the WEB OPEN command for the connection.
- Supply a client certificate. Client certificates are not a requirement for all SSL transactions, but a server might require one for particular transactions. If a server does request a client certificate, you can specify the label of a suitable certificate in the URIMAP definition that you use on the WEB OPEN command for the connection, or on the WEB OPEN command itself. The client certificate must be stored in your security manager key ring. If you use a URIMAP definition but do not specify a certificate label, the default certificate defined in the key ring for the CICS region user ID is used.

## Introduction to Application Transparent Transport Layer Security (AT-TLS)

Application Transparent Transport Layer Security (AT-TLS) can be used to create secure socket sessions on behalf of CICS. Instead of implementing Transport Layer Security (TLS) in CICS, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the Policy Agent. When AT-TLS is used to secure socket sessions, CICS SSL/TLS start parameters such as KEYRING and MINTLSLEVEL/ENCRYPTION are no longer required as the implementation of TLS is provided by AT-TLS policy statements and all encryption and decryption is done outside of the CICS address space.

### AT-TLS MODES

When AT-TLS is active for a CICS socket connection, CICS sends and receives cleartext (unencrypted data), while AT-TLS encrypts and decrypts data at the TCP transport layer. For more information about AT-TLS and AT-TLS policy setup, see [AT-TLS policy configuration in z/OS Communications Server: IP Configuration Guide](#) and [Policy Agent and policy applications in z/OS Communications Server: IP Configuration Reference](#).

Most address spaces (such as CICS) do not need to be aware of the security negotiations and encryption that is done by TCP/IP on its behalf. However, some address spaces need to be aware of AT-TLS or have control over the security functions that are being performed by TCP/IP. For example, if the address space is a server that requires client authentication, it might want to access the client certificate and the user ID associated with the client certificate. CICS issues an AT-TLS query on a new client connection when a TCPIP SERVICE is defined with SSL(ATTLSAWARE).

Address spaces such as CICS that are taking advantage of AT-TLS can be separated into three different types (AT-TLS Basic mode, AT-TLS Aware mode, and AT-TLS Controlling mode) as described in [Table 39 on page 468](#). The type is based on whether awareness of the service is needed and, if so, the amount of control that the address space is given over the security functions.

AT-TLS Basic mode is where the address space (such as CICS) does not issue any AT-TLS calls to query a socket for its AT-TLS status.

AT-TLS Aware mode is where the address space issues an AT-TLS calls to query a socket for its AT-TLS status. The address space can access items such as the client certificate, and also the certificate User ID.

AT-TLS Controlling mode is where the address space issues AT-TLS calls to control the secure session on a socket.

<i>Table 39. Detailed description of AT-TLS modes and their CICS support</i>			
<b>Mode type</b>	<b>AT-TLS calls issued</b>	<b>ApplicationControlled setting in AT-TLS policy</b>	<b>Supported by CICS TS for z/OS</b>
AT-TLS Basic	Address space does not issue any AT-TLS calls	Off	All CICS releases
AT-TLS Aware	Address space issues query requests	Off	From CICS TS 5.3 For CICS as an HTTP server only
AT-TLS Controlling	Address space issues query and control requests	On	Not supported by CICS TS

For detailed information about AT-TLS modes (types), see [Application Transparent Transport Layer Security data protection in z/OS Communications Server: IP Configuration Guide](#).

## **CICS support for AT-TLS Basic**

In the basic mode, the address space is unaware that AT-TLS is encrypting or decrypting data.

With basic mode, the address space (such as CICS) is oblivious to the fact that TCP/IP is performing a TLS handshake and encryption/decryption of message flows on the socket.

With the basic mode of AT-TLS, a CICS TCPIP SERVICE would be defined with SSL(NO). One drawback of this is that CICS cannot access client certificates as it is unaware of a certificate that is associated with the socket. Also, when CICS uses HTTP redirection with this mode of operation, there are failures because CICS assumes that the client connection is HTTP rather than HTTPS. A problem that might arise (for example) is when you are using AUTHENTICATE(BASIC) on an HTTP TCPIP SERVICE, and CICS discovers that the user's password expired. A dialog is triggered with the user to request a new password. At the end of this dialog, there is an error because the resubmission of the original HTTP request specifies HTTP as the scheme instead of HTTPS.

## **CICS support for AT-TLS Aware**

The only support of AT-TLS aware mode is for CICS as an HTTP server.

CICS supports the AT-TLS Aware mode with the additional option of SSL(ATTLSAWARE) on the TCPIP SERVICE definition. SSL(ATTLSAWARE) is only allowed if the TCPIP SERVICE also specifies PROTOCOL(HTTP).

When a TCPIP SERVICE is defined with SSL(ATTLSAWARE), CICS issues an AT-TLS query to obtain information such as AT-TLS security status, negotiated CIPHER suite, partner certificate, and derived RACF user ID.

When an HTTP TCPIP SERVICE is defined with SSL(ATTLSAWARE), CICS can access client certificates and their associated RACF USERIDs. The result is all the certificate-related TCPIP SERVICE AUTHENTICATE options (CERTIFICATE | AUTOREGISTER | AUTOMATIC) are supported by SSL(ATTLSAWARE).

When a TCPIP SERVICE is defined with SSL(ATTLSAWARE), CICS detects that the client is using an HTTPS connection. This means that redirection failures which can occur when using AT-TLS in basic mode (such as the expired password dialog) are fixed by using SSL(ATTLSAWARE).



## AT-TLS Controlling

CICS does not support AT-TLS Controlling mode.

## CICS AT-TLS query

All new client connections to the TCPIPService defined with SSL(ATTLSAWARE) are queried to extract their AT-TLS attributes. When a client establishes a new connection to a CICS TCPIPService defined with SSL(ATTLSAWARE), it will trigger the query and CICS accepts the new client connection.

- Connection status (AT-TLS secured/not secured).
- Client Authentication type (NONE | PASSTHRU | FULL | REQUIRED | SAFCHECK).
- Client certificate. If present, the client certificate is saved in the CICS certificate repository. Attributes of the certificate can be retrieved later on by applications that issue **EXEC CICS EXTRACT CERTIFICATE**.
- Client certificate USERID can be used with the AUTHENTICATE option of the TCPIPService to establish the security context for the new web task. For example, AUTHENTICATE(CERTIFICATE) requires a CERTIFICATE USERID to be present to allow a web request to be processed.
- Negotiated CIPHER number. This number is registered in a performance monitoring record in the existing field **SO CIPHER**.

## Configurations for AT-TLS and CICS TCPIPService

There are certain combinations of AT-TLS policy and CICS TCPIPService, which are valid. The table here shows these valid combinations and expected results in CICS. It also shows combinations that are invalid.

Table 40. Combinations for AT-TLS and CICS TCPIPService			
AT-TLS policy for the port	CICS TCPIPService	Valid combination	Expected result in CICS
HandShakeRole=Server	SSL(NO   ATTLSAWARE)	Yes	The connection is successfully established.  No client certificate is available.
HandShakeRole=Server or HandShakeRole=ServerWithClientAuth	SSL(YES   ClientAuth)	No	CICS issues DFHWB0732 and rejects the connection.
HandShakeRole=ServerWithClientAuth with ClientAuthType=(REQUIRED   SAFCHECK)	SSL(ATTLSAWARE)	Yes	The connection is successfully established.  Client certificate is available.
HandShakeRole=ServerWithClientAuth with ClientAuthType=(FULL)	SSL(ATTLSAWARE)	Yes	The connection is successfully established.  Client certificate is available if client sends it to the server.

Table 40. Combinations for AT-TLS and CICS TCPIPService (continued)			
AT-TLS policy for the port	CICS TCPIPService	Valid combination	Expected result in CICS
HandShakeRole=ServerWithClientAuth with any of ClientAuthType	SSL(NO)	Yes	The connection is successfully established.  No client certificate is available.
NO AT-TLS policy for the port, or the client is exempted from using an AT-TLS policy.	SSL(ATTLSAWARE)	No	CICS rejects the connection with an HTTP 403 error.  CICS issues DFHSO0147 for the first connection and DFHWB0365 for every non-secure connection.
HandShakeRole=ServerWithClientAuth and ClientAuthType=PASSTHRU	SSL(ATTLSAWARE)	No	CICS rejects the connection because this configuration bypasses client certificate validation.  CICS issues DFHSO0149 and TCPIPService is closed.
NO AT-TLS policy configured.	SSL(YES   ClientAuth)	Yes	CICS processes SSL-connection as documented.

**Note:** For TCPIPService, configuring AT-TLS HandShakeRole=Client is incorrect.

When you are using a TCPIPService that specifies SSL(ATTLSAWARE), CICS expects all connections to be cryptographically secured by AT-TLS. If a client connection arrives which is not secured, it is rejected with an HTTP 403 error. CICS also logs the error with message DFHWB0365.

CICS does not support AT-TLS policies that use ClientAuthType=PassThru. This configuration bypasses client certificate validation, which is not acceptable to CICS. If CICS detects that this type of client authentication is being used when you are receiving a client connection, it closes the connection with the client then closes the TCPIPService. Message DFHSO0149 is written to the console when this error is detected.

## How to refresh the SSL environment and cache when AT-TLS is in use

When you are using a TCPIPService that specifies SSL(ATTLSAWARE), the **PERFORM SSL REBUILD** command does not apply.

So to refresh the SSL environment and the certificate CICS is using, follow these steps:

1. Place the new certificate into the key ring defined in your AT-TLS policy.
2. Refresh the key ring within the security manager.
3. Change or add an **EnvironmentUserInstance** value in the policy rule for this CICS traffic.
4. Issue either of the following modify commands:

F PAGENT, REFRESH

or  
F PAGENT,UPDATE

If you omit step 3 or do not make other changes to the AT-TLS configuration, then the UPDATE option has no effect. UPDATE only refreshes the environment if the AT-TLS configuration has changed. If a certificate in the keyring is the only change that is made, then the REFRESH option can be used.

It is worth noting that a recycle of the CICS region will pick up an AT-TLS certificate change.

## AT-TLS diagnostics

There are a number of tools for diagnosing AT-TLS problems.

For diagnosing AT-TLS problems, see [Diagnosing Application Transparent Transport Layer Security \(AT-TLS\) in z/OS Communications Server: IP Diagnosis Guide](#).

AT-TLS messages contain return codes that are useful in diagnosing problems. Return codes below 5000 come from system SSL. For more information on return codes, see [SSL function return codes in z/OS Cryptographic Services System SSL Programming](#).

## Socket Domain Trace Points for AT-TLS

The socket domain trace points listed, are relevant to AT-TLS. For more information, see, [Socket domain trace points](#).

- SO 0CAC (level 1)
- SO 0CAB (EXC)
- SO 0CA9 (level 2)
- SO 0CAA (level 2)

## Diagnostic Messages

The messages DFHSO0147 and DFHSO0149 are relevant to AT-TLS and detailed here, [DFHSOnnnn messages](#). Also, message DFHWP0365 is detailed here, [DFHWPnnnn messages](#).

CICS provides some diagnostic messages when you encounter errors that are discovered by CICS:

DFHWP0365

*date time applid tranid* A client connects to a TCIPSERVICE defined with SSL(ATTLSAWARE) but the connection is not secured by AT-TLS. Host IP address: *hostaddr*. Client IP address: *clientaddr*. TCIPSERVICE: *tcpipservice*.

DFHSO0147 W

*applid* A non-secure client connection is received for ATTLSAWARE TCIPSERVICE *tcpipservice*. Client IP address: *clientaddr*. TTLS\_IOCTL value *X'ttlsioclt'*.

DFHSO0149 W

*applid* A client connection that uses **CLIENTAUTHTYPE(PASSTHRU)** is detected for ATTLSAWARE TCIPSERVICE *tcpipservice*. TTLS\_IOCTL value *X'ttlsioclt'*. The TCIPSERVICE is closed.

Here are two examples of errors on AT-TLS connections:

1. The following diagnostics are seen when a client connects to an AT-TLS secured port, which is configured by using *HandShakeRole ServerWithClientAuth* and *ClientAuthType Required*. This configuration requires the client to provide a certificate. In this case, the client fails to provide a certificate. Here is the information that is shown in the AT-TLS message log:

```
EZD1287I TTLS Error RC: 403 Initial Handshake 034
LOCAL: ::FFFF:9.20.5.0..25931
REMOTE: ::FFFF:9.174.17.124..50077
JOBNAME: SSYCZCCM RULE: CICS
USERID: HORN GRPID: 0000000D ENVID: 00000013 CONNID: 00395D99
```

The return code of 403 is a system SSL error, and corresponds to error GSK\_ERR\_NO\_CERTIFICATE, which means no certificate received from partner. Nothing is seen in the CICS log. CICS never receives this connection as it is being rejected by AT-TLS.

- The following diagnostics appear when a client connection is made to a TCPIP SERVICE defined with **SSL(ATTLSAWARE)** where the TCPIP SERVICE port is NOT secured by AT-TLS. This time the client is connecting to a port, which is not policed by AT-TLS and means that there are no AT-TLS diagnostics. However, CICS detects that the client connection is not secured by AT-TLS so it issues the following messages:

- DFHS00147 W IY2CZCCM 041 A non-secure client connection is received for ATTLSAWARE TCPIP SERVICE ATTLS2. Client IP address: *clientaddr*. TTLS\_IOCTL value X'0100000102010000'
- DFHWB0365 06/23/2015 10:14:22 IY2CZCCM CWXN A client connects to a TCPIP SERVICE defined with SSL(ATTLSAWARE) but the connection is not secured by AT-TLS. Host IP address: *hostaddr*. Client IP address: *clientaddr*. TCPIP SERVICE: ATTLS2.

The first message is only issued once for any TCPIP SERVICE. The second message is issued every time a client connects and CICS finds that the connection is NOT secured by AT-TLS.

## Migration from CICS SSL to AT-TLS

You can move an existing CICS Transport Layer Security (TLS) (SSL) implementation for an inbound socket connection to Application Transparent Transport Layer Security (AT-TLS).

When CICS is used to establish a TLS (SSL) environment to perform a TLS handshake for an inbound socket connection, the attributes that are used for the handshake are extracted from two sources: region level SIT parameters and TCPIP SERVICE resource attributes.

The following two tables show the CICS SIT parameters and TCPIP SERVICE resource attributes that are used for a TLS handshake and their AT-TLS level equivalents.

Table 41. SIT parameters and AT-TLS equivalents	
SIT Parameter	AT-TLS equivalents
MINTLSLEVEL	TLSv1.1 TLSv1.2 TLSv1.3
KEYRING	TTLSKeyRingParms
CRLPROFILE	TTLSGskLdapParms
SSLDELAY	GSK_V3_SESSION_TIMEOUT
MAXSSLTCBS	Cannot be configured in AT-TLS; TCB numbers grow dynamically.
SSLCACHE=SYSPLEX	GSK_SYSPLEX_SIDCACHE ON
NISTSP800131A=CHECK	FIPS140 ON

Table 42. TCPIP SERVICE resource attributes and AT-TLS equivalents	
TCPIP SERVICE resource attribute	AT-TLS equivalents
SSL=YES	HandShakeRole Server
SSL=CLIENTAUTH	HandShakeRole ServerWithClientAuth with ClientAuthType FULL ClientAuthType REQUIRED and ClientAuthType SAFCHECK are also supported.
CERTIFICATE	CertificateLabel

Table 42. TCIPSERVICE resource attributes and AT-TLS equivalents (continued)	
TCIPSERVICE resource attribute	AT-TLS equivalents
CIPHERS	TTLSCipherParms

## Considerations for using keyrings

The CICS region user ID still requires access to the keyring that is specified in the AT-TLS policy. If you are migrating from CICS SSL to AT-TLS, you can continue to use the existing CICS-owned keyrings and reference them in the AT-TLS policies. If you want to set up new keyrings in TCPIP, the CICS region user ID will require access to this new keyring. The server certificate will remain as either a CICS-owned or SITE certificate.

## Examples

The following examples show how to move an existing CICS TLS implementation to AT-TLS, and then remove the CICS TLS implementation.

- “[Example 1: AT-TLS policy rules for TLS/SSL server authentication](#)” on page 473
- “[Example 2: AT-TLS policy rules for TLS/SSL client authentication](#)” on page 475

### Example 1: AT-TLS policy rules for TLS/SSL server authentication

An example configuration to use CICS to secure inbound HTTP connections might use simple server authentication on the TCIPSERVICE resource (SSL (YES)). This configuration does not support client certificates. [Figure 158 on page 473](#) and [Figure 159 on page 473](#) show the CICS configuration statements that are needed to establish the CICS-TLS environment for simple server authentication.

```
MINTLSLEVEL=TLS11
KEYRING=CICSKeyRing (includes the certificate named CICS-2048-certificate)
SSLDelay=600
MAXSSLTCBS=8
SSLCACHE=CICS
NISTSP800131A=NOCHECK
```

Figure 158. CICS startup parameters

```
TcPipService : HTTPSSL
GRoup       : JULESWEB
DEscription ==> CICS WEB TCIPSERVICE WITH SSL SUPPORT
POrtnumber  ==> 25008
STatus      ==> Open
PROtocol    ==> Http
SSL         ==> Yes
CErtificate ==> CICS-2048-certificate
CIphers     ==> 35363738392F303132330A1613100D15120F0C
AUthenticate ==> Basic
```

Figure 159. SSL-related TCIPSERVICE resource attributes

To use AT-TLS to secure your inbound HTTP connections instead of CICS, you might use the following AT-TLS policy, and then update the TCIPSERVICE resource definition to SSL (NO) or SSL (ATTLSAWARE).

**Note:** The following example AT-TLS policy uses the TLSV1.2 option. Using the TLSV1.2 option helps to achieve optimum performance; also, it is a prerequisite if you need to conform to NIST SP800-131A.

Full details of the NIST standards are available at the [NIST Computer Security Resource Center \(nist.gov\)](https://nist.gov).

[Figure 160 on page 474](#) shows the AT-TLS configuration that replicates the CICS environment for the TCIPSERVICE named HTTPSSL.

```

TTLSRule SIMPLICICS
{
LocalPortRange 25008
Direction Inbound
Priority 256
TTLSGroupActionRef CICSGroupAct1
TTLSEnvironmentActionRef CICSEnvironmentAct1
}
TTLSGroupAction CICSGroupAct1
{
}
TTLSEnabled On
FIPS140 off
}
TTLSEnvironmentAction CICSEnvironmentAct1
{
}
HandShakeRole Server
TTLSKeyRingParmsRef CICSKeyRingParms1
TTLSCipherParmsRef CICS cipherParms1
TTLSEnvironmentAdvancedParmsRef CICSEnvAdvParms1
TTLSGskAdvancedParmsRef CICSgskAdvParms1
}
TTLSKeyRingParms CICSKeyRingParms1
{
}
Keyring CICSKeyRing
}
TTLSCipherParms CICS cipherParms1
{
}
V3CipherSuites TLS_RSA_WITH_AES_256_CBC_SHA
V3CipherSuites TLS_DH_DSS_WITH_AES_256_CBC_SHA
V3CipherSuites TLS_DH_RSA_WITH_AES_256_CBC_SHA
V3CipherSuites TLS_DHE_DSS_WITH_AES_256_CBC_SHA
V3CipherSuites TLS_DHE_RSA_WITH_AES_256_CBC_SHA
V3CipherSuites TLS_RSA_WITH_AES_128_CBC_SHA
V3CipherSuites TLS_DH_DSS_WITH_AES_128_CBC_SHA
V3CipherSuites TLS_DH_RSA_WITH_AES_128_CBC_SHA
V3CipherSuites TLS_DHE_DSS_WITH_AES_128_CBC_SHA
V3CipherSuites TLS_DHE_RSA_WITH_AES_128_CBC_SHA
V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA
V3CipherSuites TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
V3CipherSuites TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
V3CipherSuites TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
V3CipherSuites TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
V3CipherSuites TLS_DHE_RSA_WITH_DES_CBC_SHA
V3CipherSuites TLS_DHE_DSS_WITH_DES_CBC_SHA
V3CipherSuites TLS_DH_RSA_WITH_DES_CBC_SHA
V3CipherSuites TLS_DH_DSS_WITH_DES_CBC_SHA
}
TTLSEnvironmentAdvancedParms CICSEnvAdvParms1
{
}
SSLv3 Off
TL SV1 On
TL SV1.1 On
TL SV1.2 On
CertificateLabel CICS-2048-certificate
}
TTLSGskAdvancedParms CICSgskAdvParms1
{
}
GSK_SYSPLEX_SIDCACHE off
GSK_V3_SESSION_TIMEOUT 600
}
}

```

*Figure 160. AT-TLS configuration*

Before you activate this AT-TLS policy, alter the CICS TCPIP SERVICE resource as follows:

```

TCpipservice : HTTPSSL
GRoup       : JULESWEB
DEscription ==> CICS WEB TCPIP SERVICE WITH AT-TLS SSL SUPPORT
Portnumber  ==> 25008
STatus      ==> Open
PROtocol    ==> Http
SSL         ==> NO|ATTL aware
CErtificate ==>
CIphers     ==>
Authenticate ==> Basic

```

If SSL is set to NO, CICS does not check whether AT-TLS is securing inbound client connections.

If SSL is set to ATTLASWARE, CICS checks whether AT-TLS is securing inbound client connections. If a client connection is not secured by AT-TLS, it is rejected with an HTTP 403 error and message DFHWB0365 is written to the CICS log.

Also, if SSL is set to ATTLASWARE, CICS checks for the presence of a client certificate. The previous example AT-TLS configuration does not support the use of client certificates. Therefore, ensure that the TCIPSERVICE definition does not specify an AUTHENTICATE option that requires client certificates. The previous example TCIPSERVICE resource specifies AUTHENTICATE (BASIC), which does not require a client certificate.

When the AT-TLS policy is active and the TCIPSERVICE resource is redefined to remove the SSL attributes, you can also remove all the related SSL SIT parameters. However, first ensure that nothing else in the CICS region depends on these parameters.

If your CICS-SSL system is started with **NISTSP800131A=CHECK**, CICS sets **MINTLSLEVEL=TLS12** and also sets FIPS140 on. To reflect these settings in the example AT-TLS POLICY configuration, modify it as follows:

```
TTLSTGroupAction CICSGroupAct1
{
  TTLS-enabled On
  FIPS140 on
}

TTLSEnvironmentAdvancedParms CICSEnvAdvParms1
{
  SSLv3 Off
  TLSV1 off
  TLSV1.1 Off
  TLSV1.2 On
  CertificateLabel CICS-2048-certificate
}
```

## Example 2: AT-TLS policy rules for TLS/SSL client authentication

An example configuration to use CICS to secure inbound HTTP connections might use client authentication on the TCIPSERVICE resource (SSL (CLIENTAUTH)).

This configuration supports client certificates. [Figure 161 on page 475](#) and [Figure 162 on page 475](#) show the CICS configuration statements that are needed to establish the CICS-TLS environment for client authentication.

```
MINTLSLEVEL=TLS11
KEYRING=CICSKeyRing (includes the certificate named CICS-2048-certificate)
SSLDelay=600
MAXSSLTCBS=8
SSLCACHE=CICS
NISTSP800131A=NOCHECK
```

*Figure 161. CICS startup parameters*

```
TcPipService : CLAUTH
GRoup : JULESWEB
DEscription ==> CICS Web TCIPSERVICE with SSL CLIENTAUTH support
POrtnumber ==> 25009
STatus ==> Open
PRotocol ==> Http
SSl ==> Clientauth
CErtificate ==> CICS-2048-certificate
CIphers ==> 35363738392F303132330A1613100D15120F0C
AUthenticate ==> Certificate
```

*Figure 162. SSL-related TCIPSERVICE resource attributes*

To use AT-TLS to secure your inbound HTTP connections instead of CICS, you might use the following AT-TLS policy, and then update the TCIPSERVICE resource definition to use SSL (ATTLASWARE).

**Note:** The following example AT-TLS policy uses the TLSV1.2 option. Using the TLSV1.2 option helps to achieve optimum performance; also, it is a prerequisite if you need to conform to NIST SP800-131A.

Full details of the NIST standards are available at the NIST Computer Security Resource Center ([nist.gov](http://nist.gov)).

Figure 163 on page 476 shows the AT-TLS client authentication configuration that replicates the CICS environment for the TCPIP SERVICE named CLAUTH.

```

TTLRule CLIENTAUTHCICS
{
  LocalPortRange 25009
  Direction Inbound
  Priority 256
  TLSGroupActionRef CICSGroupAct2
  TLSEnvironmentActionRef CICSEnvironmentAct2
}
TLSGroupAction CICSGroupAct2
{
  TTLEnabled On
  FIPS140 off
}
TLSEnvironmentAction CICSEnvironmentAct2
{
  HandShakeRole ServerWithClientAuth
  TLSKeyRingParmsRef CICSKeyRingParms2
  TLSCipherParmsRef CICS cipherParms2
  TLSEnvironmentAdvancedParmsRef CICS envAdvParms2
  TLSGskAdvancedParmsRef CICS gskAdvParms2
}
TLSKeyRingParms CICSKeyRingParms2
{
  Keyring CICSKeyRing
}
TLSCipherParms CICS cipherParms2
{
  V3CipherSuites TLS_RSA_WITH_AES_256_CBC_SHA
  V3CipherSuites TLS_DH_DSS_WITH_AES_256_CBC_SHA
  V3CipherSuites TLS_DH_RSA_WITH_AES_256_CBC_SHA
  V3CipherSuites TLS_DHE_DSS_WITH_AES_256_CBC_SHA
  V3CipherSuites TLS_DHE_RSA_WITH_AES_256_CBC_SHA
  V3CipherSuites TLS_RSA_WITH_AES_128_CBC_SHA
  V3CipherSuites TLS_DH_DSS_WITH_AES_128_CBC_SHA
  V3CipherSuites TLS_DH_RSA_WITH_AES_128_CBC_SHA
  V3CipherSuites TLS_DHE_DSS_WITH_AES_128_CBC_SHA
  V3CipherSuites TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  V3CipherSuites TLS_RSA_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites TLS_DHE_RSA_WITH_DES_CBC_SHA
  V3CipherSuites TLS_DHE_DSS_WITH_DES_CBC_SHA
  V3CipherSuites TLS_DH_RSA_WITH_DES_CBC_SHA
  V3CipherSuites TLS_DH_DSS_WITH_DES_CBC_SHA
}
TLSEnvironmentAdvancedParms CICS envAdvParms2
{
  SSLv3 Off
  TLSV1 On
  TLSV1.1 On
  TLSV1.2 On
  CertificateLabel CICS-2048-certificate
  ClientAuthType Full
}
TLSGskAdvancedParms CICS gskAdvParms2
{
  GSK_SYSPLEX_SIDCACHE off
  GSK_V3_SESSION_TIMEOUT 600
}

```

Figure 163. AT-TLS client authentication configuration

Before you activate this example AT-TLS policy, alter the CICS TCPIP SERVICE resource definition as follows:

```

TCpipservice : CLAUTH
GROup       : JULESWEB
DEscription ==> CICS Web TCPIP SERVICE with SSL CLIENTAUTH support

```



```
Portnumber ==> 25009
Status ==> Open
Protocol ==> Http
SSL ==> ATTLISAWARE
Certificate ==>
Ciphers ==>
Authenticate ==> Certificate
```

In this example, SSL must be set to ATTLISAWARE so that CICS retrieves a client certificate from AT-TLS, because AUTHENTICATE is set to CERTIFICATE (a client certificate is required). If a client connection is not secured by AT-TLS, it is rejected with an HTTP 403 error and message DFHWB0365 is written to the CICS log.

With **SSL(ATTLISAWARE)**, CICS checks for a client certificate. If this check maps to a RACF USERID, CICS runs the web user transaction under this USERID.

The previous example AT-TLS policy is defined with `ClientAuthType Full`. This `ClientAuthType` replicates the SSL environment and handshake behavior that occurs when CICS uses SSL. However CICS also supports `ClientAuthType Required` and `ClientAuthType SAFCheck`.

CICS does not support the use of `ClientAuthType PassThru`. If a TCPIP SERVICE port is configured by using `ClientAuthType PassThru` and the TCPIP SERVICE resource is defined with SSL (ATTLISAWARE), when the first client connection arrives, CICS detects the unsupported configuration. CICS then closes the TCPIP SERVICE and issues message DFHSO0149.

When the AT-TLS policy is active and the TCPIP SERVICE resource is redefined to remove the SSL attributes, you can also remove all the related SSL SIT parameters. However, first ensure that nothing else in the CICS region depends on these parameters.

If your CICS-SSL system is started with **NISTSP800131A=CHECK**, CICS sets **MINTLSLEVEL=TLS12** and it also sets FIPS140 on. To reflect these settings in the example AT-TLS POLICY configuration, modify it as follows:

```
TTLSTGroupAction CICSGroupAct2
{
  TTLSSEnabled On
  FIPS140 on
}

TTLSSEnvironmentAdvancedParms CICSSEnvAdvParms2
{
  SSLv3 Off
  TLSV1 Off
  TLSV1.1 Off
  TLSV1.2 On
  CertificateLabel CICS-2048-certificate
  ClientAuthType Full
}
```

## Security for Atom feeds

CICS web support provides a suitable security protocol and authentication method to control web client access to Atom collections and if required, to Atom feeds. You can use CICS resource and command security to protect the resources that you use to deliver the Atom feed or collection.

RFC 5023 recommends that you use authentication to protect Atom collections. When you make Atom feed data available as an editable collection, a web client can insert new entries, modify existing entries, or delete entries. You must therefore ensure that you verify the identity of web clients and permit only trusted clients to have access to the collection, especially if you have included business data in your collection. Ordinary Atom feeds, which web clients cannot edit, are typically made available to any subscribers without security restrictions, although you might need to restrict access to Atom feeds if they include confidential business data or are intended only for certain users.

RFCs 4287 and 5023 discuss the use of digital signatures and encryption for Atom documents. CICS does not provide support for digital signatures and encryption of Atom documents, but, in compliance with RFC 4287, CICS does not reject an Atom document that contains a signature.

CICS web support has the following security functions that you can use to protect Atom feeds or collections from unauthorized access or updates:

**SSL or TLS security protocol**

RFC 5023 recommends the use of the Transport Layer Security (TLS) 1.0 as a minimum level of security protocol for collections. For a list of security protocols supported by CICS, see [TLS](#).

**HTTP basic authentication**

RFC 5023 recommends the use of HTTP basic authentication as a minimum level of authentication for collections.

**Client certificate authentication**

Client certificate authentication is a more secure method of authenticating a client, using a client certificate which is issued by a trusted third party (or Certificate Authority), and sent using SSL encryption.

When you set up these functions in CICS web support, you can apply them to an Atom feed or collection using attributes of the TCPIP SERVICE definition for the port where CICS receives web client requests for the Atom feed or collection. For information on setting up SSL support for CICS web support, see [Configuring CICS to use SSL](#).

---

## Chapter 21. Security for platforms and applications

You can secure resources for applications that are deployed on platforms by creating RACF security profiles for CICSplex SM to cover platforms and applications in a CICSplex.

Security for platforms and applications is set up in a similar way to security for other CICSplex SM components. You control access to a specific set of views (and their associated action commands) by identifying the set in a security profile. With these security profiles, you can give users authority to install, enable or disable, make available or unavailable, inquire on, or discard platforms and applications, and ensure that unauthorized users cannot create and administer these resources.

When you give a user authority to perform an action on a platform or application, you also give them authority to perform the same action on the dynamically generated resources for the platform or application. For example, a user who has authority to enable an application also has authority to enable the CICS bundles for the application that were installed in CICS regions in all the platforms in the CICSplex. CICS command and resource security checks, and simulated CICS security checking in CICSplex SM, are not carried out when you operate on CICS bundles through an application or platform.

You can secure a platform and its deployed applications by setting up security profiles with the following function and type combinations:

### **CLOUD.DEF.context**

This security profile covers the PLATDEF and APPLDEF resource tables, which contain the definitions for platforms and applications. *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with UPDATE access for this security profile can create, update, and remove definitions for platforms and applications in the CICSplex SM data repository. Users with READ access can view those definitions in the CICSplex SM data repository.

### **CLOUD.PLATFORM.context**

This security profile covers the installation of PLATDEF resources and operations on PLATFORM resources. It also allows users to view management parts (MGMTPART resources). *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with ALTER access for this security profile can install platforms in the CICSplex and discard them. (To install a platform, users also need READ access for the CLOUD.DEF profile that covers the PLATDEF resource.) Users with UPDATE access can enable and disable platforms. Users with UPDATE access can also add CICS regions to region types in the platform and remove CICS regions from region types in the platform. Users with READ access can view PLATFORM resources and MGMTPART resources. These permissions apply for all platforms that exist in the CICSplex.

### **CLOUD.APPLICATION.context**

This security profile covers the installation of APPLDEF resources and operations on APPLCTN resources. *context* is the specific or generic name of the CICSplex that is covered by the security profile.

Users with ALTER access for this security profile can install applications in the CICSplex and discard them. (To install an application, users also need READ access for the CLOUD.DEF profile that covers the APPLDEF resource.) Users with UPDATE access can enable and disable applications and make them available or unavailable. Users with READ access can view APPLCTN resources. These permissions apply for all applications in all platforms that exist in the CICSplex. If you require different security permissions for certain applications, use a different CICSplex to host the platform where you deploy the application.

**Note:** These security profiles are only checked in the maintenance point CMAS. Security checks are reported by message EYUCR0009I in the EYULOG of the maintenance point CMAS. To receive message EYUCR0009I for violations you must set the CICSplex SM system parameter (EYUPARM) **SECLOGMSG** to YES. For more information about **SECLOGMSG**, see [CICSplex SM system parameters](#).

Although the CLOUD security profiles cover actions on the dynamically generated resources for the platform or application, users may still carry out a limited set of actions directly on individual resources in the CICSplex and CICS regions where they are installed:

- You can modify the CICSplex SM topology definition (CSYSDEF, or CICS system definition) for a CICS region that is part of a platform. Attribute values that you specified at the region type level are locked and cannot be changed, but other attribute values can be changed.
- You can make available or unavailable, enable or disable, or inquire on, a BUNDLE resource that was dynamically created when you installed a platform or application. You cannot discard an individual bundle directly if it was created when you installed a platform or application.
- You can inquire on a dynamically created resource, such as a PROGRAM resource, that was defined inside a CICS bundle and created when you installed a platform or application. You cannot enable, disable, or discard these resources directly if they were created when you installed a platform or application.

CICS command and resource security checks, and simulated CICS security checking in CICSplex SM, do apply when you perform an action directly on a CICS region that is part of a platform, or on an individual CICS bundle, or a resource defined in a CICS bundle, that was created when you installed a platform or application.

- A `TOPOLOGY.DEF.context` security profile covers actions on the CICSplex SM topology definitions for individual CICS regions that are part of a platform. *context* is the specific or generic name of the CICSplex that is covered by the security profile. Users with UPDATE access can modify the CSYSDEF for a CICS region that is part of a platform, with the exception of attribute values that are locked by the platform itself.
- CICS bundles created when you install a platform or application have a unique generated name beginning with the \$ character. To provide security for actions on individual CICS bundles that were dynamically created in this way, you can set up a security profile specifying the BUNDLE resource type and the resource name \$\*. Users with UPDATE access for BUNDLE.\$\* can make available or unavailable, or enable or disable, BUNDLE resources created for platforms and applications, and users with READ access can inquire on those BUNDLE resources.

If you apply security measures to individual PROGRAM resources, for applications that are deployed on platforms, secure the programs that are declared as application entry points, but do not secure other programs in the applications. The security settings that you specify for a program that is part of an application deployed on a platform apply to both public and private programs, and do not take into account the version of the application. Programs that are declared as an application entry point must have a unique PROGRAM resource name in your environment. However, if you secure programs that run at a lower level in the application, programs with the same names might be running in different applications, which can lead to unforeseen consequences. In this situation, a user might have permission to access a program that is declared as an application entry point, but not have permission to access a program that runs at a lower level in the application, because the security settings from another instance of the program name are in effect. Consider the security measures that you apply to a program that is declared as an application entry point program, as applying to the whole application.

If you used CICS bundles in earlier CICS releases, check the security permissions that you gave to users for those bundles. Depending on the way in which you set up security for CICS bundles, users with authority to take actions on individual CICS bundles might now be able to act on resources that are dynamically created as part of the installation of a bundle. Ensure that the levels of authority for BUNDLE resources are still appropriate.

Table 43 on page 481 summarizes the security checks that apply to actions performed on a platform, an application, an individual CICS bundle that was dynamically created when you installed a platform or application, or a resource defined in a CICS bundle for a platform or application.

Table 43. Security checks for operations on platforms, applications, and generated CICS bundles				
Operation	Platforms, including their CICS bundles	Applications, including their bundles	Dynamically-created CICS bundles	Resources defined in dynamically-created CICS bundles
Define	CLOUD.DEF profile (UPDATE, or READ to view definitions); also TOPOLOGY.DEF profile (UPDATE to modify individual CICS region CSYSDEF after platform install)	CLOUD.DEF profile (UPDATE, or READ to view definitions)	Cannot manage resource definitions individually	Cannot manage resource definitions individually
Install	CLOUD.PLATFORM profile (ALTER) and CLOUD.DEF profile (READ)	CLOUD.APPLICATION profile (ALTER) and CLOUD.DEF profile (READ)	Cannot install individually	Cannot install individually
Enable or disable	CLOUD.PLATFORM profile (UPDATE)	CLOUD.APPLICATION profile (UPDATE)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	Cannot enable or disable individually
Make available or unavailable	Not applicable	CLOUD.APPLICATION profile (UPDATE)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	Cannot make available or unavailable individually
Inquire	CLOUD.PLATFORM profile (READ); also allows viewing of management parts	CLOUD.APPLICATION profile (READ)	CICS command and resource security, and simulated CICS security checking in CICSplex SM; use BUNDLE.\$* profile	CICS command and resource security, and simulated CICS security checking in CICSplex SM
Discard	CLOUD.PLATFORM profile (ALTER)	CLOUD.APPLICATION profile (ALTER)	Cannot discard individually	Cannot discard individually

For more information on setting up security for CICSplex SM and creating security profiles, see [Implementing CICSplex SM security](#).



---

## Chapter 22. Security in BTS

Security considerations for CICS business transaction services are the authority to access BTS resources, the user IDs under which a process and its constituent activities run, the authority to attach the process and its constituent activities and the authority to use BTS system programming commands.

CICS security is described in detail in [Securing CICS](#). Users of external security managers (ESMs) other than the Resource Access Control Facility (RACF) must read this information with the documentation for their own ESM.

### Process and activity user IDs

To understand what user IDs are used by the BTS process or activity, see [Process and activity user IDs](#).

### Resource security in BTS

You can protect BTS resources such as processes, activities, and containers in the same way as resources accessed by CICS file control commands. That is, resource-level security for a process, its activities, and their containers is based on the CICS file definition that specifies the repository data set to which records for processes of this type are written.

Users who run programs that define or acquire processes or activities of a particular process type need UPDATE access to the corresponding CICS file.

**Note:** When a task issues an **ACQUIRE** command, CICS allows the appropriate record to be read from the BTS repository, even if the user ID associated with the request has only READ access. However, when the task issues a sync point, the record is written back to the data set and, if the userid does not have UPDATE access, the task abends.

Users who inquire on or browse processes or activities of a particular process-type need at least READ access to the corresponding CICS file.

### Attach-time security for processes and activities

You can use attach-time security to check whether a transaction has authority to attach (activate) a process or activity. Attach-time security applies only when a process or activity is activated by a **RUN** command, not when it is activated by a **LINK**.

If attach-time security is required for a process, the defined user ID, that is, the user ID obtained from the **DEFINE PROCESS** command, must be given UPDATE access to the CICS file that corresponds to the BTS data set on which details of the process and its constituent activities are stored.

### Command security in BTS

You can use CICS command-level security to protect your BTS system programming commands: EXEC CICS CREATE PROCESSTYPE, [DISCARD PROCESSTYPE](#), [INQUIRE PROCESSTYPE](#), and [SET PROCESSTYPE](#).

---

## Process and activity user IDs

You can use either the **RUN** or **LINK** command to activate a process or activity. The command that you use affects the user ID under which the process or activity runs.

### User IDs for activities activated by RUN commands

When a process or activity is activated by a **RUN** command, it might run using a different user ID than the transaction that issues the **RUN**.

The application programmer can specify under whose authority a process or activity is to run, when it is activated by a **RUN** command, by coding the user ID option of the **DEFINE PROCESS** or **DEFINE**

**ACTIVITY** command. If the user ID option is omitted, its value defaults to the user ID of the transaction that issues the **DEFINE** command.

The user ID obtained from the **DEFINE** command is referred to as the `defined_process_userid` or the `defined_activity_userid`. In the remainder of this section, we use the term “defined user ID” to mean either a defined process user ID or a defined activity user ID.

If the user ID option of **DEFINE PROCESS** or **ACTIVITY** is specified, CICS performs (at define time) a surrogate security check to verify that the user ID of the transaction that issued the **DEFINE** command is authorized to use the defined user ID. The RACF profile used for surrogate checking of a BTS process or activity is `userid.DFHSTART` in the `SURROGAT` class.

The following example RACF commands authorize a user as a surrogate user of a defined process user ID and of a defined activity user ID:

```
RDEFINE SURROGAT defined_process_userid.DFHSTART UACC(NONE)
      OWNER(defined_process_userid)

PERMIT defined_process_userid.DFHSTART CLASS(SURROGAT)
      ID(define_process_command_userid) ACCESS(READ)

RDEFINE SURROGAT defined_activity_userid.DFHSTART UACC(NONE)
      OWNER(defined_activity_userid)

PERMIT defined_activity_userid.DFHSTART CLASS(SURROGAT)
      ID(define_activity_command_userid) ACCESS(READ)
```

#### User IDs for activities activated by **LINK** commands

When a process or activity is activated by a **LINK** command, it runs under the user ID of the transaction that issues the **LINK**.

Resource-level security checking in a process or activity is based on the user ID under whose authority the process or activity runs, that is, the defined user ID or the user ID of the transaction that issues the **LINK** command. This user ID must have UPDATE access to the CICS file that corresponds to the process type.



# Chapter 23. Security for the CICS-MQ adapter

The CICS-MQ adapter provides information to IBM MQ specifically for use in IBM MQ security.

Information provided is as follows:

- Whether CICS resource-level security is active for this transaction. For more information, see [Resource security](#).
- User IDs.
  - For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is one of the following:
    - The default CICS user ID as specified on the CICS **DFLTUSER** system initialization parameter.
    - A preset security user ID specified on the terminal definition.
  - For nonterminal tasks, the adapter acquires the user ID with a call to the user domain.

## Implementing security for CICS-MQ adapter transactions

If you want a user to administer the CICS-MQ adapter, you must grant the user authorization to the appropriate CICS transactions.

If required, you can restrict access to specific functions of the adapter. For example, if you want to allow users to display the current status of the adapter, but nothing else, give them access to CKQC, CKBM, CKRT, and CKDP only.

Define these transactions to CICS with RESSEC (YES) and CMDSEC (YES) . For more details, see [Resource security](#) and [Command security](#).

- **6.2** By default, these transactions are defined with RESSEC (YES) and CMDSEC (YES) .
- **6.1** By default, these transactions are defined with RESSEC (NO) and CMDSEC (NO) .

Transaction	Function
CKAM	Alert monitor
CKBM	Controls the adapter functions
CKCN	Connect
CKDL	Line mode display
CKDP	Full screen display
CKQC	Controls the adapter functions
CKRS	Statistics
CKRT	Controls the adapter functions
CKSD	Disconnect
CKSQ	CKTI START/STOP
CKTI	Trigger monitor

As well as administrators, user IDs connecting to IBM MQ, the user ID set in the **PLTPIUSR** system initialization parameter, and the [CICSplex SM MAS agent user ID](#) must also be authorized to run the CKTI and CKAM transactions.

## CICS-MQ adapter user IDs

---

The user ID associated with the CICS-MQ adapter is the user ID associated with the calling transaction that is accessing IBM MQ.

### User ID checking for IBM MQ resources

If a CICS-MQ resource (MQCONN or MQMONITOR) is used to access IBM MQ, the user ID used by IBM MQ is the user ID of the transaction issuing the MQI command:

- For PLTPI programs, this is the **PLTPIUSR** system initialization parameter.
- For PLTSD programs, this is the user ID associated with the shutdown transaction.
- For other programs, this is the user ID associated with the running transaction.

### User ID of the CICS-MQ trigger monitor

When security checking is active, you are recommended to always use an MQMONITOR to start CKTI instances. If you use an MQMONITOR, do not use CKQC. Using an MQMONITOR ensures that a single user ID, the **MONUSERID** attribute of the MQMONITOR, is always used for the CKTI instance irrespective of how it was started.

### User ID of the CICS-MQ trigger monitor without an MQMONITOR

Starting CKTI without an MQMONITOR is still supported, but not recommended.

If an instance of CKTI is started without using an MQMONITOR, the user ID associated with the CKTI transaction is the user ID of the transaction starting CKTI:

- For PLTPI programs, this is the **PLTPIUSR** system initialization parameter.
- For CKQC users, this is the signed-on user ID running the transaction.
- If a sequential terminal is used to run CKQC, this is the user ID used to run the sequential terminal. The user ID must be a preset user ID rather than the default of the CICS default user ID. See [The DFHTCT TYPE=TERMINAL macro](#).

## Command security for MQCONN and MQMONITOR resources

---

Use CICS command security to control users' ability to issue SPI commands against MQCONN and MQMONITOR resource definitions. For example, you can use it to control which users are allowed to issue CREATE and DISCARD commands against the MQCONN resource definition for the CICS region.

When command security is enabled for a transaction, the external security manager checks that the user ID associated with the transaction is authorized to use the command on the MQCONN or MQMONITOR resource as appropriate. Resource security is not available for MQCONN and MQMONITOR resources.

CICS command security covers the **EXEC CICS CREATE MQCONN, DISCARD MQCONN, SET MQCONN, INQUIRE MQCONN, CREATE MQMONITOR, DISCARD MQMONITOR, SET MQMONITOR, and INQUIRE MQMONITOR** commands. For an explanation of command security and instructions to set up command security for a CICS region, see [Command security](#). For a listing of the level of authority required for each command, see [CICS commands subject to command security checking](#).

When command security is active, the user ID for the running transaction that issues the **EXEC CICS SET MQCONN** command to start the connection to IBM MQ must have the following authority:

1. The authority to use the **EXEC CICS SET MQCONN** command; otherwise, the start of the connection will fail with a response of NOTAUTH and a RESP2 of 100.
2. The authority to use the **EXEC CICS EXTRACT EXIT** command; otherwise, the start of the connection will fail with a response of INVREQ and a RESP2 of 9. In this case, CICS issues messages DFHXS1111 and DFHMQ0302.

In addition, if MQMONITORs are being used, the user ID under which the MQMONITOR is running (specified in the **MONUSERID** parameter on the MQMONITOR definition) requires authorization for command security. This applies to MQMONITORs that are used to control the CICS-MQ trigger monitor, the CICS-MQ bridge, or user-written MQMONITOR programs. The MONUSERID must have the following authority:

1. The authority to use the **EXEC CICS SET MQMONITOR** command to set the status of the MQMONITOR to STARTED or STOPPED; otherwise, the MQMONITOR task will fail, and in the case of the CICS-MQ trigger monitor, CICS issues message DFHMQ0125.
2. In the case of the CICS-MQ trigger monitor, the authority to use the **EXEC CICS START** command with the **TRANSID** option set to the transaction that is specified in the trigger message; otherwise, CICS issues message DFHMQ0102 and the trigger message will be sent to the dead-letter queue.

## Surrogate user security for MQMONITOR resources

---

Use CICS surrogate user security to control which transactions the CICS-MQ trigger monitor is allowed to start. This applies only when a trigger monitor instance has been started by an MQMONITOR.

When security checking is active and **XUSER=YES** is specified as a system initialization parameter, CICS® performs surrogate user checks when an **EXEC CICS START** command with the USERID option is used to start a transaction. If the CICS-MQ trigger monitor has been started by using an MQMONITOR, it uses information from the MQMONITOR definition to start the user transaction. The trigger monitor issues an **EXEC CICS START** command with the USERID option specifying a value obtained from the **USERID** attribute of the MQMONITOR.

CICS requires that the user ID associated with the transaction issuing the **START** request be a surrogate of the user ID associated with the started transaction. The CICS-MQ trigger monitor, when started by an MQMONITOR, always runs under the user ID specified in the **MONUSERID** attribute of the MQMONITOR. Therefore, for CKTI to be able to start the user transaction specified in the trigger message, the MONUSERID must be a surrogate of the user ID associated with the started user task. As the USERID specified in the MQMONITOR definition is used by default to start the user transaction if no suitable user ID is available from any other source, the MONUSERID must be a surrogate of the USERID as well. If the surrogate security check fails, CICS issues message DFHMQ0102 and the trigger message will be sent to the dead-letter queue.

## IBM MQ connection security for the CICS-MQ adapter

---

IBM MQ carries out connection security checking either when an application program tries to connect to a queue manager by issuing an MQCONN or MQCONNX request, or when the channel initiator or the CICS-MQ adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager, but, if you do that, any user can connect to that queue manager.

Only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID.

You can turn IBM MQ connection security checking on or off at either queue manager or queue-sharing group level.

### Related concepts

[How it works: The CICS-MQ adapter](#)

### Related tasks

[Setting up the CICS-MQ adapter](#)

[Administering the CICS-MQ adapter](#)

[Troubleshooting the CICS-MQ adapter](#)

### Related reference

[Troubleshooting data for CICS-MQ adapter and CICS-MQ bridge problems](#)



# Chapter 24. Security for the CICS-MQ bridge

When you start the CICS-MQ bridge, you can specify the level of authentication. If requested, the bridge monitor checks the user ID and password extracted from the IBM MQ request message before running the CICS program named in the request message.

When you run the CICS-MQ bridge monitor transaction (for example, CKBR or your transaction name), you can specify the **AUTH** parameter to select one of the following levels of authentication:

## LOCAL

This level is the default. The bridge monitor starts the bridge task with the CICS default user ID. CICS user programs that the bridge task runs have the authority associated with this user ID. The IBM MQ request message cannot request higher authority because any user IDs or passwords in the message are ignored. If the bridge task runs a CICS program that tries to access protected resources, the CICS program might fail.

## IDENTIFY

If the message descriptor (MQMD) in the request message specifies a user ID, the bridge monitor starts the bridge task with that user ID. CICS user programs that the bridge task runs have the authority associated with that user ID. The user ID is treated as trusted; that is, the bridge monitor does not authenticate the ID by using password or PassTicket information. If the MQMD does not specify a user ID, the bridge monitor starts the bridge task with the CICS default user ID, in the same way as the LOCAL option.

## VERIFY\_UOW

The bridge monitor uses the password or PassTicket to authenticate the user ID if all the following conditions apply:

- The message descriptor (MQMD) in the request message specifies a user ID.
- The request message includes an IBM MQ CICS information header (MQCIH).
- The Authenticator field in the MQCIH contains a password or PassTicket.

If authentication succeeds, the bridge monitor starts the bridge task with that user ID. If authentication fails, the bridge monitor fails the request with a MQCRC\_SECURITY\_ERROR return code.

If any one of the conditions listed earlier is not met, the bridge monitor starts the bridge task with the CICS default user ID, in the same way as the LOCAL option. Only the first request message in the unit of work is checked; the bridge ignores user ID and password or PassTicket information in subsequent messages that are part of the same unit of work.

## VERIFY\_ALL

This level is the same as VERIFY\_UOW, except that the bridge task also checks that the user ID is the same in every request message in the same unit of work, and reauthenticates the user ID for each request message, using the password or PassTicket that the message contains.

If you require different levels of authentication for different applications, use multiple bridge monitors with different transaction IDs. You can use CICS surrogate security to restrict the combinations of transaction and user ID that a bridge monitor transaction and user ID can start.

Table 44 on page 489 shows the user ID under which the bridge monitor is started. The user ID depends on the method that you use to run the bridge monitor transaction, typically CKBR.

Table 44. CICS-MQ bridge monitor security		
Bridge monitor start method	At a signed on terminal	User ID for bridge monitor
From a terminal or <b>EXEC CICS LINK</b> in a program	Yes	Signed-on user ID
From a terminal or <b>EXEC CICS LINK</b> in a program	No	CICS default user ID

Table 44. CICS-MQ bridge monitor security (continued)		
Bridge monitor start method	At a signed on terminal	User ID for bridge monitor
<b>EXEC CICS START</b> with user ID	–	User ID from START
<b>EXEC CICS START</b> without user ID	–	User ID from START
The CICS-MQ trigger monitor CKTI	–	User ID from START
The CICS MQ monitor (MQMONITOR)	–	<ul style="list-style-type: none"> <li>The <b>MONUSERID</b> attribute of the MQMONITOR resource, if security checking is active for the CICS region (that is, the <b>SEC</b> system initialization parameter is set to YES)</li> <li>User ID that started the MQMONITOR resource, if security checking is disabled for the CICS region (that is, <b>SEC</b> is set to NO)</li> </ul>

## User IDs and passwords in request messages

When you use the IDENTIFY, VERIFY\_UOW, or VERIFY\_ALL authentication options, the bridge task and the CICS programs that it runs are started with the user ID that is specified in the message descriptor (MQMD) in the request message. With the VERIFY\_UOW and VERIFY\_ALL options, the bridge monitor also checks the password specified in the IBM MQ CICS information header (MQCIH) in the request message.

To use these levels of authentication, the IBM MQ applications must provide a user ID in the MQMD, and they must provide an MQCIH including the password. You must define these user IDs to RACF. To control the user IDs used, the IBM MQ applications must open the request queue for the bridge monitor using open options that include MQOO\_SET\_IDENTITY\_CONTEXT. The applications must also include a value of MQPMO\_SET\_IDENTITY\_CONTEXT in their put message options.

If the bridge monitor finds a problem with the user ID or the password in a request message, it acts as follows:

- For the IDENTIFY level of authentication, if a message contains a user ID that was revoked, abend AICO might occur. The error reply has return code MQCRC\_BRIDGE\_ERROR with reason MQFB\_CICS\_BRIDGE\_FAILURE.
- For the VERIFY\_UOW or VERIFY\_ALL level of authentication, if the user ID or password is invalid, the request fails with return code MQCRC\_SECURITY\_ERROR.
- If a request message omits either the user ID or the password, the bridge monitor runs the bridge task with the LOCAL level of authentication, even if you started the bridge monitor with one of the other authentication options. In that situation, the CICS programs started by the bridge task run with the user ID under which the bridge monitor was started.

You can use a PassTicket in place of a password to avoid the need to flow passwords in messages.

- The IBM MQ application must provide the PassTicket in the MQCIH in the request message.

- To generate a PassTicket, an application ID is required. The default application ID is the CICS APPLID. You can specify an alternative application ID by using the **PASSTKTA** parameter when you start the CICS-MQ bridge monitor transaction (for example, CKBR or your transaction name).
- If you use multiple bridge monitors for the same request queue, you must use the **PASSTKTA** parameter to specify the same application ID for each bridge monitor.

PassTicket validation is performed by using IBM MQ services, not **EXEC CICS VERIFY**, because the CICS service does not allow you to specify an APPLID. For more information about PassTickets, see [How it works: PassTickets](#) and [Setting up security on z/OS](#).

## Authority

You must give the following permissions to the user IDs that you use with the CICS-MQ bridge. The user IDs include the user ID under which the bridge monitor transaction is started, as listed in [Table 44 on page 489](#), and any user IDs that IBM MQ applications specify in request messages.

- The user ID under which the bridge monitor transaction is started must have authority to start the CKBP and CKBC transactions for CICS DPL programs, and any alternative transactions that IBM MQ applications specify in the TransactionId field in the MQCIH structure in request messages.
- If IBM MQ applications are specifying user IDs in request messages, the user ID under which the bridge monitor transaction is started must be defined to RACF as a surrogate of all the user IDs used in request messages. A surrogate user is one that has the authority to start work on behalf of another user, without knowing the password of the other user. For more information about surrogate user security, see [Surrogate security](#).
- The user IDs for the bridge monitor and for all bridge tasks need authority to get messages from the request queue.
- The user IDs for a bridge task need authority to put messages to its reply-to queue.
- To ensure that any error replies are received, the user ID under which the bridge monitor transaction is started must have authority to put messages to all reply-to queues.
- The user IDs for bridge tasks must have authority to put messages to the dead-letter queue.
- The user ID under which the bridge monitor transaction is started needs authority to put messages to the dead-letter queue, unless you want the bridge to stop if an error occurs.
- The user IDs for the bridge monitor and for all bridge tasks must have authority to put messages to the backout requeue queue, if one is defined.





---

## Chapter 25. Security for data sources

You can configure security for the CICS Db2 environment, the CICS-DBCTL interface, and CICS data sets.

### **Related concepts**

[Access to databases and data sets](#)

---

### Security for Db2

You can secure a CICS Db2 environment by authenticating the user, performing security checks when a CICS user tries to use or modify a Db2-related CICS resource, when the CICS region connects to Db2 and a transaction acquires a thread into Db2, and when a CICS user tries to use a CICS transaction to execute or modify a Db2 resource. You can use RACF, or an equivalent external security manager, to protect the CICS and Db2 components from unauthorized access, such as the Db2 databases, logs, and so on.

To be more specific, you can implement security checking at the following stages:

- When a CICS user signs on to a CICS region. CICS sign-on authenticates users by checking that they supply a valid user ID and password..
- When a CICS user tries to use or modify a CICS resource that is related to Db2. This could be a DB2CONN, DB2ENTRY or DB2TRAN resource definition; or a CICS transaction that accesses Db2 to obtain data; or a CICS transaction that issues commands to the CICS Db2 attachment facility or to Db2 itself. At this stage, you can use CICS security mechanisms, which are managed by RACF or an equivalent external security manager, to control the CICS user's access to the resource.
- When the CICS region connects to Db2, and when a transaction acquires a thread into Db2. Both the CICS region and the transaction must provide authorization IDs to Db2, and these authorization IDs are validated by RACF or an equivalent external security manager.
- When a CICS user tries to use a CICS transaction to execute or modify a Db2 resource. This could be a plan, or a Db2 command, or a resource that is needed to execute dynamic SQL. At this stage, you can use Db2 security checking, which is managed either by Db2 itself, or by RACF or an equivalent external security manager, to control the CICS user's access to the resource.

You can use RACF, or an equivalent external security manager, to protect the CICS and Db2 components from unauthorized access. You can apply this protection to Db2 databases, logs, bootstrap data sets (BSDSs), and libraries outside the scope of Db2, and to CICS data sets and libraries. You can use VSAM password protection as a partial replacement for the protection provided by RACF. For more information, see [CICS system resource security](#).

**Note:** RACF is referred to here as the external security manager used by CICS. Except for the explicit RACF examples, the general discussion applies equally to any functionally equivalent non-IBM external security manager.

[Figure 164 on page 494](#) shows the security mechanisms involved in a CICS Db2 environment.

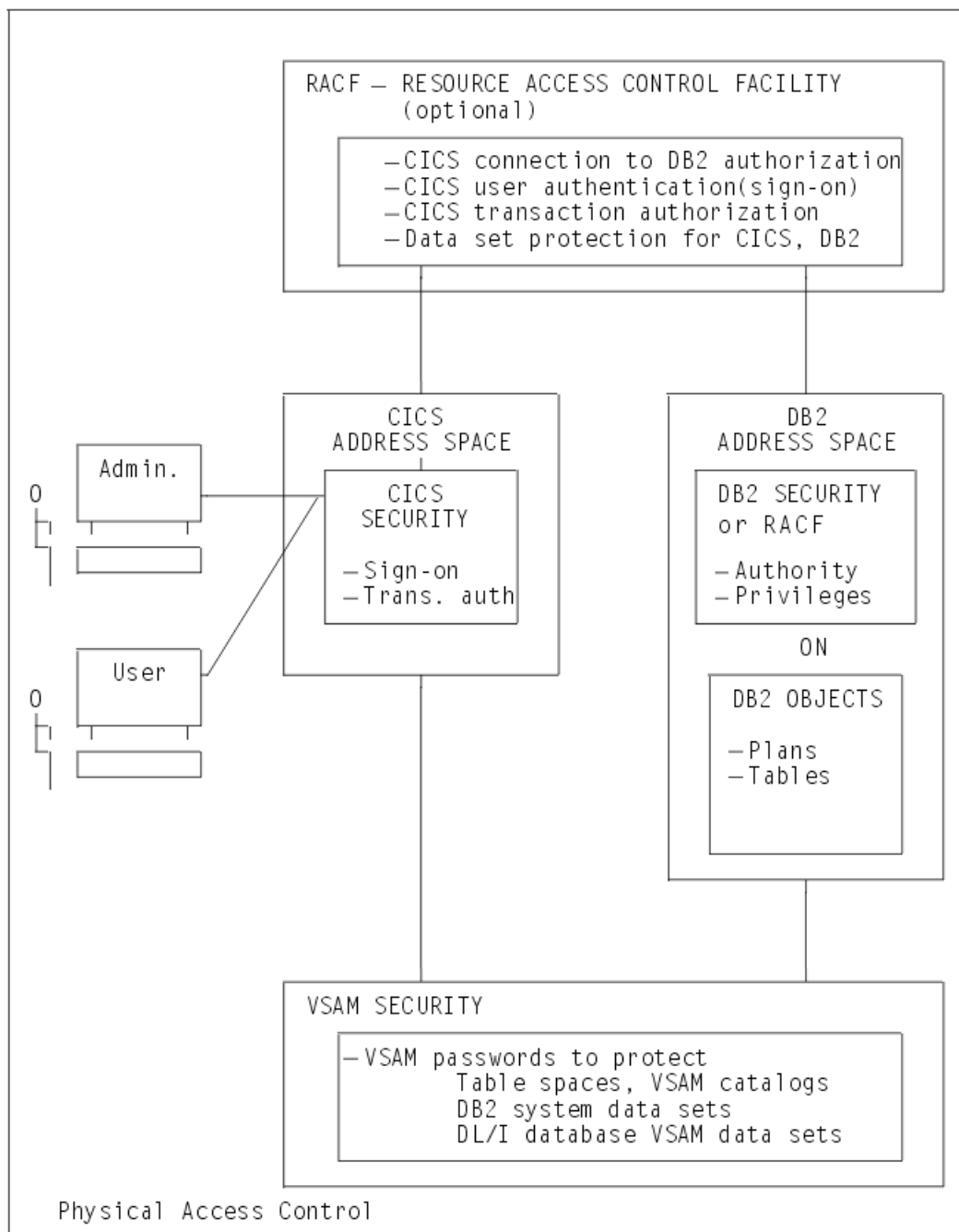


Figure 164. Overview of the CICS Db2 security mechanisms

## Controlling access to Db2-related resources in CICS

You can control access to Db2 resources in your CICS region, and initiate security checking for the resources, by enabling an external security manager and the appropriate CICS security mechanism.

### About this task

CICS users might want to perform the following activities involving Db2:

- Inquire on, modify, create or discard DB2CONN, DB2ENTRY, and DB2TRAN resource definitions.
- Use a transaction that accesses Db2 to obtain data, or issue CICS Db2 attachment facility commands or Db2 commands using the DSNB transaction.

Use RACF, or an equivalent external security manager to perform security checks in your CICS region. When a user tries to access protected resources, CICS calls the external security manager to perform security checking. RACF makes security checks using the CICS user's ID, which is authenticated when the user signs on to CICS. If a user does not sign on to CICS, they are given the default user ID, unless a user ID has been permanently associated with a terminal using preset security.

Enable the appropriate security mechanism for your CICS region: transaction security, resource security, command security, or surrogate security.

For more information about CICS security, see [Securing CICS](#).

## Controlling users' access to DB2CONN, DB2TRAN, and DB2ENTRY resource definitions

You can control users' access to DB2CONN, DB2TRAN, and DB2ENTRY resource definitions by enabling different CICS security mechanisms.

### About this task

The following security mechanisms can be used to control user access to Db2 resource definitions.

- Control users' ability to access particular resources by using the CICS **resource security** mechanism. Resource security is implemented at the transaction level. For example, you could prevent some users from modifying a particular DB2ENTRY definition. [“Using resource security to control access to DB2ENTRY and DB2TRAN resource definitions” on page 495](#) tells you how to use this security mechanism.
- Control users' ability to issue particular SPI commands against Db2-related resources by using the CICS **command security** mechanism. Command security is also implemented at the transaction level. For example, you could permit only certain users to issue CREATE and DISCARD commands against DB2ENTRY resource definitions. [“Using command security to control the issuing of SPI commands against DB2CONN, DB2ENTRY, and DB2TRAN resource definitions” on page 497](#) tells you how to use this security mechanism.
- Control users' ability to modify the authorization IDs that CICS provides to Db2, by using the CICS **surrogate security and AUTHTYPE security** mechanisms. The authorization IDs are used for Db2 security checking, and they are set by the AUTHID, COMAUTHID, AUTHTYPE and COMAUTHTYPE attributes on Db2-related resource definitions, and by the SIGNID attribute on the DB2CONN definition for the CICS region. CICS checks that the user who wants to modify the authorization ID, is permitted to act on behalf of the existing authorization ID that is specified in the resource definition. [“Using surrogate security and AUTHTYPE security to control access to the authorization IDs that CICS provides to Db2” on page 498](#) tells you how to use these security mechanisms.

## Using resource security to control access to DB2ENTRY and DB2TRAN resource definitions

The CICS resource security mechanism controls users' access to named CICS resources. For example, you can use it to protect certain resources, such as a particular DB2ENTRY definition, from being modified by particular users.

### About this task

CICS command security can prevent users from performing certain actions on types of resources, such as "all DB2ENTRY definitions", but it cannot protect individual items within the resource type.

Because your CICS region only has one DB2CONN definition, you do not need to use resource security to protect it; you can control access to the DB2CONN definition using command security. Also, DB2TRAN definitions, for the purpose of resource security, are treated as extensions of the DB2ENTRY definition to which they refer, and are not defined for resource security in their own right. If you give a user permission to access a DB2ENTRY definition, you also give them permission to access the DB2TRAN definitions that refer to it. (In the case where a transaction changes the name of the DB2ENTRY with which a DB2TRAN

definition is associated, a double security check is performed, to verify the user's authority to modify both the old DB2ENTRY to which the definition referred, and the new DB2ENTRY to which it will refer.) For resource security, you therefore only need to define your DB2ENTRY definitions to RACF.

When resource security is enabled for a transaction, the external security manager checks that the user ID associated with the transaction is authorized to modify the resource that is involved. [Resource security](#) has more information about this process.

## Procedure

To protect your Db2-related resources using resource security, complete these steps.

1. To enable RACF, or an equivalent external security manager, and make resource security available for a CICS region, specify SEC=YES as a system initialization parameter for the CICS region.
2. In RACF, create general resource classes to contain your Db2-related resources. You need a member class and a grouping class.

Unlike the RACF default resource class names for CICS, there are no IBM-supplied default class names for DB2ENTRY resources. Create your own installation-defined class names by adding new class descriptors to the installation-defined part (module ICHRRCDE) of the RACF class descriptor table (CDT). For an example of how to do this, see the IBM-supplied sample job, RRCDE, supplied in member DFH\$RACF of CICSTSnn.CICS.SDFHSAMP library, where CICSTSnn is your version of CICS. As an example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is CICSTS62.CICS.SDFHSAMP. This gives an example of a member class called XCICSDB2, and a grouping class called ZCICSDB2. This example uses the same naming convention as the default resource class names for CICS. Do not use existing CICS class names for Db2-related resource definitions; instead, create new class names using a similar naming convention.

3. Define profiles for your DB2ENTRY definitions in the resource classes that you have created.

For example, to add a number of DB2ENTRY names to the XCICSDB2 resource class, use the RDEFINE command as follows:

```
RDEFINE XCICSDB2 (db2ent1, db2ent2, db2ent3.., db2entn) UACC(NONE)
                NOTIFY(sys_admin_userid)
```

Protecting DB2ENTRY resource definitions also protects access to associated DB2TRAN definitions, because a DB2TRAN is considered to be an extension to the DB2ENTRY to which it refers. You do not need to protect your DB2CONN definition using resource security.

4. To activate resource security for your Db2-related resources, specify XDB2=name as a system initialization parameter for the CICS region, where name is the general resource class name that you defined for your Db2-related resources.
5. Specify RESSEC=YES in the resource definition for any transactions involving Db2-related resources for which you want to enable resource security. Now, when a user tries to use one of these transactions to access one of the Db2-related resources that you have protected, RACF checks that the user ID is authorized to access that resource.
6. Give permission to your CICS users, or groups of users, to perform appropriate actions on each Db2-related resource that you have protected.

Remember that if a user has permission to perform actions on a DB2ENTRY definition, they are automatically authorized to perform the same actions on DB2TRAN definitions associated with it. The access that users need to perform certain actions is as follows:

### **INQUIRE command**

Requires READ authority

### **SET command**

Requires UPDATE authority

### **CREATE command**

Requires ALTER authority

## DISCARD command

Requires ALTER authority

For example, you can use the PERMIT command to authorize a group of users to modify a protected DB2ENTRY, db2ent1 in class XCICSDB2, with UPDATE authority, as follows:

```
PERMIT db2ent1 CLASS(XCICSDB2) ID(group1) ACCESS(UPDATE)
```

## Using command security to control the issuing of SPI commands against DB2CONN, DB2ENTRY, and DB2TRAN resource definitions

Use CICS command security mechanisms to protect DB2CONN, DB2ENTRY, and DB2TRAN resource definitions.

### About this task

The CICS command security mechanism controls users' ability to issue particular SPI commands against types of Db2-related resource. For example, you can use it to control which users are allowed to issue CREATE and DISCARD commands against DB2ENTRY resource definitions. Unlike resource security, CICS command security cannot protect individual named resources; it is designed to protect types of resource. You can use command security to protect DB2CONN, DB2ENTRY, and DB2TRAN resource definitions.

When command security is enabled for a transaction, the external security manager checks that the user ID associated with the transaction is authorized to use that command to modify the type of resource that is involved. [Command security](#) has more information about this process.

If you have both resource security and command security enabled for a particular transaction, RACF performs two security checks against the user ID. For example, if a transaction involves the user issuing a DISCARD command against DB2ENTRY definition db2ent1, RACF checks:

1. That the user ID is authorized to issue the DISCARD command (ALTER authority) against the DB2ENTRY resource type.
2. That the user ID is authorized to access the DB2ENTRY definition db2ent1 with ALTER authority.

To protect your Db2-related resources using command security, complete these steps.

### Procedure

1. To enable RACF, or an equivalent external security manager, for a CICS region, specify SEC=YES as a system initialization parameter for the region.
2. Add the Db2 resource names DB2CONN, DB2ENTRY, and DB2TRAN as resource identifiers in one of the IBM-supplied RACF resource classes for CICS commands, CCICSCMD or VCICSCMD.

Alternatively, you can use a user-defined general resource class for your CICS commands. [CICS resources subject to command security checking](#) tells you more about this.

For example, you can use the REDEFINE command to define a profile named CMDSAMP in the default class VCICSCMD, and use the ADDMEM operand to specify that the Db2 resource types are to be protected by this profile, as follows:

```
RDEFINE VCICSCMD CMDSAMP UACC(NONE)
          NOTIFY(sys_admin_userid)
          ADDMEM(DB2CONN, DB2ENTRY, DB2TRAN)
```

3. To make command security available for a CICS region:
  - a) If you have used the IBM-supplied RACF resource classes CCICSCMD or VCICSCMD for CICS command profiles, specify XCMD=YES as a system initialization parameter for the region.  
Specifying YES means that CCICSCMD and VCICSCMD are used to build RACF's in-storage profiles.
  - b) If you have used a user-defined general resource class for CICS commands, specify XCMD=*user\_class* as a system initialization parameter for the region, where *user\_class* is the name of the user-defined general resource class.

4. Specify CMDSEC=YES in the resource definition for any transactions involving Db2-related resources for which you want to enable command security. Now, when a user tries to use one of these transactions to issue a command to modify one of the Db2-related resources that you have protected, RACF checks that the user ID is authorized to issue that command against that type of resource.
5. Give permission to your CICS users, or groups of users, to issue appropriate commands against each type of Db2-related resource. For command security, you need to give separate permissions relating to the DB2TRAN resource type, as well as to the DB2ENTRY resource type. You can also protect the DB2CONN resource type (that is, the CICS region's DB2CONN definition).

The access that users need to issue certain commands is as follows:

**INQUIRE command**

Requires READ authority

**SET command**

Requires UPDATE authority

**CREATE command**

Requires ALTER authority

**DISCARD command**

Requires ALTER authority

For example, if you have defined the Db2 resource types in the CMDSAMP profile as in the example in Step 2, you can use the PERMIT command to authorize a group of users to issue EXEC CICS INQUIRE commands against the Db2 resource types as follows:

```
PERMIT CMDSAMP CLASS(VCICSCMD) ID(operator_group) ACCESS(READ)
```

Within a transaction, you can query whether a user ID has access to Db2 resource types by using the **EXEC CICS QUERY SECURITY RESTYPE(SPCOMMAND)** command, with the RESID parameter specifying DB2CONN, DB2ENTRY, or DB2TRAN.

## Using surrogate security and AUTHTYPE security to control access to the authorization IDs that CICS provides to Db2

The CICS surrogate security and AUTHTYPE security mechanisms control users' ability to modify the authorization IDs that CICS provides to Db2.

### About this task

Use surrogate security and AUTHTYPE security to ensure that only certain users are permitted to change the authorization IDs, which are used for security checking by Db2 own security checking. Surrogate security and AUTHTYPE security are set for the whole CICS region, and any transactions that involve changes to the authorization IDs are subject to them.

[“Providing authorization IDs to Db2 for the CICS region and for CICS transactions” on page 501](#) explains how to select and change these authorization IDs. To summarize, the authorization IDs that CICS provides to Db2 are set by the AUTHID, COMAUTHID, AUTHTYPE, and COMAUTHTYPE attributes on Db2-related resource definitions, and by the SIGNID attribute on the DB2CONN definition for the CICS region. To change the authorization IDs, you first need authority to modify the DB2CONN and DB2ENTRY definitions, which might be protected by command security or resource security. Surrogate security provides an extra layer of protection, because it involves CICS acting on behalf of Db2 to check that the user that is modifying the authorization ID is permitted to act as a surrogate for the existing authorization ID that is specified in the resource definition.

True surrogate security provides security checking when a user attempts to change the SIGNID, AUTHID or COMAUTHID attributes on a DB2CONN or DB2ENTRY definition, all of which specify an authorization ID that is used when a process signs on to Db2. CICS uses the surrogate user facility of RACF to perform this checking. A surrogate user is one who has the authority to do work on behalf of another user, without knowing that other user's password. When a user attempts to change one of the SIGNID, AUTHID

or COMAUTHID attributes, CICS calls RACF to check that the user is authorized as a surrogate of the authorization ID that is presently specified on the SIGNID, AUTHID or COMAUTHID attribute.

For the AUTHTYPE and COMAUTHTYPE attributes, which give a type of authorization ID to be used rather than specifying an exact authorization ID, CICS cannot use true surrogate security. Instead, it uses a mechanism called AUTHTYPE security. When a user attempts to change one of the AUTHTYPE or COMAUTHTYPE attributes, CICS calls RACF to check that the user is authorized through a profile that you have defined for the resource definition in the RACF FACILITY general resource class. Although AUTHTYPE security is not true surrogate security, it is enabled by the same system initialization parameter, and you will probably want to use it in addition to surrogate security, so the instructions in this topic tell you how to set up both types of security.

Note that when DB2CONN and DB2ENTRY resource definitions are installed as part of a cold or initial start of CICS, if surrogate security and AUTHTYPE security are enabled, RACF makes surrogate security and AUTHTYPE security checks for the CICS region user ID. If you install DB2CONN and DB2ENTRY resource definitions in this way, ensure that the CICS region user ID is defined as a surrogate user for any authorization IDs specified in the resource definitions, and also that it is authorized through the correct profiles in the RACF FACILITY general resource class.

To implement surrogate security and AUTHTYPE security to protect the authorization IDs that CICS provides to Db2, complete the following steps:

## Procedure

1. To enable RACF, or an equivalent external security manager, for a CICS region, specify SEC=YES as a system initialization parameter for the region.
2. To activate surrogate security and AUTHTYPE security for a CICS region, specify XUSER=YES as a system initialization parameter for the region. This system initialization parameter enables both the security mechanisms. When the security mechanisms are enabled, CICS calls RACF to perform security checks whenever a transaction involves EXEC CICS SET, CREATE, and INSTALL commands that operate on the SIGNID, AUTHID, COMAUTHID, AUTHTYPE and COMAUTHTYPE attributes of DB2CONN and DB2ENTRY resource definitions. For the SIGNID, AUTHID and COMAUTHID attributes, RACF performs the surrogate security check, and for the AUTHTYPE or COMAUTHTYPE attributes, RACF performs the AUTHTYPE security check.
3. For the purpose of surrogate security, you need to define appropriate CICS users, or groups of users, as surrogates of any authorization IDs that are specified on the SIGNID, AUTHID or COMAUTHID attributes of your DB2CONN and DB2ENTRY definitions. To define a user ID as a surrogate of an authorization ID:
  - a) Create a profile for the authorization ID in the RACF SURROGAT class, with a name of the form *authid.DFHINSTL*, with the authorization ID defined as the owner.  
For example, if you have specified DB2AUTH1 as an authorization ID on a SIGNID, AUTHID or COMAUTHID attribute, use the following command to create a profile:

```
RDEFINE SURROGAT DB2AUTH1.DFHINSTL UACC(NONE) OWNER(DB2AUTH1)
```

- b) Permit appropriate CICS users to act as a surrogate for the authorization ID, by giving them READ authority to the profile you have created.  
For example, to permit a user with the ID CICSUSR1 to act as a surrogate for the authorization ID DB2AUTH1, and therefore to install or modify any SIGNID, AUTHID or COMAUTHID attributes that specify DB2AUTH1 as the existing authorization ID, use the following command:

```
PERMIT DB2AUTH1.DFHINSTL CLASS(SURROGAT) ID(CICSUSR1) ACCESS(READ)
```

Repeat this process for all the authorization IDs that you have specified on SIGNID, AUTHID or COMAUTHID attributes.

- c) If you might need to install DB2CONN and DB2ENTRY resource definitions containing SIGNID, AUTHID or COMAUTHID attributes as part of a cold or initial start of CICS, permit the CICS region user ID to act as a surrogate for any authorization IDs specified by those attributes.

The defaults for DB2CONN and DB2ENTRY resource definitions do not involve the AUTHID and COMAUTHID attributes. The default SIGNID for an installed DB2CONN definition is the applid of the CICS region.

4. For the purpose of AUTHTYPE security, you need to create a profile for each of your DB2CONN or DB2ENTRY resource definitions in the RACF FACILITY general resource class, and give appropriate CICS users, or groups of users, READ authority to the profiles. (This process imitates the true surrogate security mechanism, but does not involve the use of a specific authorization ID; instead, it protects each resource definition.) To do this:

- a) Create a profile for the DB2CONN or DB2ENTRY resource definition in the RACF FACILITY general resource class, with a name of the form DFHDB2.AUTHTYPE.*authname*, where *authname* is the name of the DB2CONN or DB2ENTRY resource definition.

For example, to define a profile for a DB2CONN resource definition named DB2CONN1, use the following command:

```
RDEFINE FACILITY DFHDB2.AUTHTYPE.DB2CONN1 UACC(NONE)
```

- b) Give appropriate CICS users READ authority to the profile you have created.

For example, to permit a user with the ID CICSUSR2 to install or modify the AUTHTYPE or COMAUTHTYPE attributes on a DB2CONN resource definition named DB2CONN1, use the following command:

```
PERMIT DFHDB2.AUTHTYPE.DB2CONN1 CLASS(FACILITY) ID(CICSUSR2) ACCESS(READ)
```

Repeat this process for each of your DB2CONN and DB2ENTRY resource definitions. Also, if you might need to install DB2CONN and DB2ENTRY resource definitions containing AUTHTYPE or COMAUTHTYPE attributes as part of a cold or initial start of CICS, give READ authority to the CICS region user ID on the profiles for those resource definitions.

## Controlling users' access to Db2-related CICS transactions

Use the CICS transaction security mechanism to control users' access to: CICS transactions that access Db2 to obtain data, the DSNB transaction, and to any other transactions that issue CICS Db2 attachment facility commands and Db2 commands.

### About this task

When transaction security is enabled, RACF, or an equivalent external security manager, checks that the CICS user is authorized to run the transaction that they have requested.

To protect Db2-related transactions using transaction security, follow the instructions in [Transaction security](#). The process is the same for all CICS transactions; there are no special considerations for Db2-related transactions as far as the transaction security mechanism is concerned. The instructions tell you how to:

- Set up appropriate system initialization parameters for the CICS region to activate transaction security (see [Transaction security](#)).
- Define transaction profiles to RACF for the transactions that you want to protect (see [RACF classes and profiles for resources](#)).

If you have defined transactions other than DSNB to issue CICS Db2 attachment facility commands and Db2 commands (for example, if you have created a separate transaction to run each command), remember to define these transactions to RACF as well.

You can now control which CICS users can use transactions that access Db2. Add the appropriate users or groups of users to the access list for the transaction profiles, with READ authority. [RACF classes and profiles for resources](#) has some recommendations about this.

For transactions that issue CICS Db2 attachment facility commands and Db2 commands, bear in mind that:



- CICS Db2 attachment facility commands operate on the connection between CICS and Db2, and they run entirely within CICS. Db2 commands operate in Db2 itself, and they are routed to Db2. You can distinguish Db2 commands from CICS Db2 attachment facility commands by the hyphen (-) character, which is entered with Db2 commands.
- If you have access to the DSNB transaction, CICS allows you to issue all of the CICS Db2 attachment facility commands and Db2 commands.
- If you have defined separate transactions to run individual CICS Db2 attachment facility commands and Db2 commands, you can give different CICS users authorization to subsets of these transaction codes, and therefore to subsets of the commands. For example, you could give some users authority to issue CICS Db2 attachment facility commands, but not Db2 commands. CICS-supplied transactions for CICS Db2 has the names of the separate transaction definitions that CICS supplies for the CICS Db2 attachment facility commands and the Db2 commands.

CICS Db2 attachment facility commands do not flow to Db2, so they are not subject to any further security checking. They are only protected by CICS transaction security. However, Db2 commands, and CICS transactions that access Db2 to obtain data, are subject to further stages of security checking by the Db2 security mechanisms, as follows:

- When a transaction signs on to Db2, it must provide valid authorization IDs to Db2. The authorization IDs are checked by RACF or an equivalent external security manager.
- Because the transaction is issuing a Db2 command or accessing Db2 data, the authorization IDs that it has provided must have permission to perform these actions within Db2. In Db2, you can use GRANT statements to give the authorization IDs permission to perform actions.

In addition, the CICS region itself must be authorized to connect to the Db2 subsystem.

“Providing authorization IDs to Db2 for the CICS region and for CICS transactions” on page 501 tells you how to authorize the CICS region to connect to the Db2 subsystem, and how to provide valid authorization IDs for transactions.

“Authorizing users to access resources in Db2” on page 508 tells you how to grant permissions to the authorization IDs that the transactions have provided to Db2.

## Providing authorization IDs to Db2 for the CICS region and for CICS transactions

CICS has two types of process that must provide Db2 with authorization IDs: the overall connection between a CICS region and Db2, and CICS transactions that acquire a thread into Db2.

### About this task

For the purposes of security, Db2 uses the term “process” to represent all forms of access to data, either by users interacting directly with Db2, or by users interacting with Db2 by way of other programs, including CICS. A process that connects to or signs on to Db2 must provide one or more Db2 short identifiers, called authorization IDs, that can be used for security checking in the Db2 address space. Every process must provide a primary authorization ID, and it can optionally provide one or more secondary authorization IDs. Db2 privileges and authority can be granted to either primary or secondary authorization IDs. For example, users can create a table using their secondary authorization ID. The table is then owned by that secondary authorization ID. Any other user that provides Db2 with the same secondary authorization ID has associated privileges over the table. To take privileges away from a user, the administrator can disconnect the user from that authorization ID.

CICS has two types of process that need to provide Db2 with authorization IDs:

- The overall connection between a CICS region and Db2, which is created by the CICS Db2 attachment facility. This process has to go through Db2 connection processing to provide Db2 with authorization IDs.
- CICS transactions that acquire a thread into Db2. These could be, for example, a transaction that is retrieving data from a Db2 database, or the DSNB transaction that is issuing a Db2 command. For each CICS transaction, the actual process that Db2 sees is the thread TCB, which CICS uses to control

a transaction's thread into Db2. These processes have to go through the Db2 sign-on processing to provide Db2 with authorization IDs.

During connection processing and sign-on processing, Db2 sets the primary and secondary authorization IDs for the process to use in the Db2 address space. By default, Db2 uses the authorization IDs that the process has provided. However, both connection processing and sign-on processing involve exit routines, and these exit routines allow you to influence the setting of the primary and secondary authorization IDs. Db2 has a default connection exit routine and a default sign-on exit routine. You can replace these with your own exit routines, and a sample connection exit routine and sign-on exit routine are supplied with Db2 to assist you with this.

## Providing authorization IDs to Db2 for a CICS region

CICS provides a primary authorization ID and one or more secondary authorization IDs to Db2.

### About this task

When the CICS Db2 attachment facility creates the overall connection between a CICS region and Db2, the process goes through the Db2 connection processing. The CICS region can provide:

- A primary authorization ID. The primary authorization ID becomes the CICS region's primary ID in Db2. For the connection between a CICS region and Db2, you cannot choose the primary authorization ID that is initially passed to the Db2 connection processing; it is the user ID for the CICS region. However, it is possible to change the primary ID that Db2 sets during connection processing, by writing your own connection exit routine. If RACF, or an equivalent external security manager, is active, the user ID for the CICS region must be defined to it. [“Providing a primary authorization ID for a CICS region” on page 502](#) tells you about the possible primary authorization IDs for a CICS region.
- One or more secondary authorization IDs. You can use the name of a RACF group, or list of groups, as secondary authorization IDs for the CICS region. If you do this, you need to replace the default Db2 connection exit routine DSN3@ATH, which only passes primary authorization IDs to Db2. The sample Db2 connection exit routine DSN3SATH passes the names of RACF groups to Db2 as secondary authorization IDs. Alternatively, you can write your own connection exit routine that sets secondary IDs for the CICS region. [“Providing secondary authorization IDs for a CICS region” on page 503](#) tells you how to set up secondary authorization IDs for a CICS region.

## Providing a primary authorization ID for a CICS region

The primary authorization ID that is passed to Db2 depends whether CICS is running as a started task, a started job, or a job.

### About this task

The connection type that CICS requests from Db2 is single address space subsystem (SASS). For the connection between a CICS region and Db2, you cannot choose the primary authorization ID that is initially passed to the Db2 connection processing. The ID that is passed to Db2 as the primary authorization ID for the CICS region is one of the following:

- The user ID taken from the RACF started procedures table, ICHRIN03, if CICS is running as a started task.
- The user parameter of the STDATA segment in a STARTED general resource class profile, if CICS is running as a started job.
- The user ID specified on the USER parameter of the JOB card, if CICS is running as a job.

The user ID that a CICS region might use must be defined to RACF, or your equivalent external security manager, if the external security manager is active. Define the user ID to RACF as a USER profile. It is not sufficient to define it as a RESOURCE profile.

Once you have defined the CICS region's user ID to RACF, permit it to access Db2, as follows:

1. Define a profile for the Db2 subsystem with the single address space subsystem (SASS) type of connection, in the RACF class DSNR. For example, the following RACF command creates a profile for SASS connections to Db2 subsystem DB2A in class DSNR:

```
RDEFINE DSNR (DB2A.SASS) OWNER(DB2OWNER)
```

2. Permit the user ID for the CICS region to access the Db2 subsystem. For example, the following RACF command permits a CICS region with a user ID of CICSHA11 to connect to Db2 subsystem DB2A:

```
PERMIT DB2A.SASS CLASS(DSNR) ID(CICSHA11) ACCESS(READ)
```

The Db2 connection exit routine takes the primary authorization ID (the user ID) provided by the CICS region, and sets it as the primary ID for the CICS region in Db2. The default Db2 connection exit routine DSN3@ATH, and the sample Db2 connection exit routine DSN3SATH, both behave in this way. It is possible to change the primary ID that Db2 sets, by writing your own connection exit routine. For more information about the sample connection exit routine and about writing exit routines, see [Securing Db2 in Db2 for z/OS product documentation](#). However, you might find it more straightforward to provide secondary authorization IDs for the CICS region, and grant permissions to the CICS region based on these, rather than on the primary authorization ID.

## Providing secondary authorization IDs for a CICS region

When a CICS region connects to Db2, it can provide one or more secondary authorization IDs to Db2, in addition to the primary authorization ID. You can use the name of the RACF group, or list of groups, to which the CICS region is connected, as secondary authorization IDs. This enables you to grant Db2 privileges and authority to RACF groups, and then connect multiple CICS regions to the same groups, instead of granting Db2 privileges to all the possible primary authorization IDs for each CICS region.

### About this task

To provide the name of the RACF group, or list of groups, to which a CICS region is connected, to Db2 as secondary authorization IDs, complete the following steps.

### Procedure

1. Specify SEC=YES as a system initialization parameter for the CICS region, to ensure that CICS uses RACF.
2. Connect the CICS region to the appropriate RACF group or list of groups.  
See [RACF users and roles](#).
3. Replace the default Db2 connection exit routine DSN3@ATH.

This exit routine is driven during connection processing. The default connection exit routine does not support secondary authorization IDs, so you need to replace it with the sample connection exit routine DSN3SATH, which is provided with Db2, or with your own routine. DSN3SATH is shipped in source form in the Db2 SDSNSAMP library, and you can use it as a basis for your own routine. DSN3SATH passes the names of the RACF groups to which the CICS region is connected, to Db2 as secondary authorization IDs. If the RACF list of groups option is active, DSN3SATH obtains all the group names to which the CICS region is connected, and uses these as secondary authorization IDs. If the RACF list of groups option is not active, DSN3SATH uses the name of the CICS region's current connected group as the only secondary authorization ID.

### Results

When the CICS region connects to Db2, the sample connection exit routine sets the CICS region's primary authorization ID (the region's user ID) as the primary ID, and sets the names of the RACF groups to which the CICS region is connected, as secondary IDs.

## What to do next

As an alternative to providing the names of RACF groups as secondary authorization IDs to Db2, you can write your own connection exit routine that sets secondary IDs for the CICS region. For information about writing connection exit routines, see [Securing Db2 in Db2 for z/OS product documentation](#).

## Providing authorization IDs to Db2 for CICS transactions

So that CICS can pass authorization IDs to Db2, CICS must be using RACF; SEC=YES must be specified in the SIT. This is because CICS needs to pass a RACF access control environment element (ACEE) to Db2.

### About this task

When a CICS transaction's thread TCB signs on to Db2 and goes through the Db2 sign-on processing, it can provide:

- A primary authorization ID. For CICS transactions, you can choose the primary authorization ID. It can be the user ID or operator ID of the CICS user, or a terminal ID, or a transaction ID; or it can be an ID that you have specified. The ID that is used as the primary authorization ID is determined by attributes in the DB2ENTRY definition (for entry threads), or in the DB2CONN definition (for pool threads and command threads). [“Providing a primary authorization ID for CICS transactions” on page 505](#) tells you how to choose the primary authorization ID for a CICS transaction.
- One or more secondary authorization IDs. You can use the name of a RACF group, or list of groups, as secondary authorization IDs. This has the advantage that you can grant Db2 privileges and authority to RACF groups, rather than to each individual CICS user. To use secondary authorization IDs, use the AUTHTYPE attribute in the DB2ENTRY definition (for entry threads), or the AUTHTYPE or COMAUTHTYPE attributes in the DB2CONN definition (for pool threads or command threads), to specify the GROUP option. You also need to replace the default Db2 sign-on exit routine DSN3@SGN, because the default routine does not pass secondary authorization IDs to Db2. When you specify the GROUP option, the primary authorization ID is automatically defined as the user ID of the CICS user associated with the transaction. [“Providing secondary authorization IDs for CICS transactions” on page 507](#) tells you how to set up and use secondary authorization IDs.

A key consideration for choosing the authorization IDs that CICS transactions provide to Db2 is the security mechanism that you have chosen for security checking in the Db2 address space. This security checking covers access to Db2 commands, plans, and dynamic SQL. You can choose to have this security checking carried out by:

- Db2 internal security.
- RACF, or an equivalent external security manager.
- Partly Db2, and partly RACF.

If you are using RACF for some or all of the security checking in your Db2 address space, CICS transactions that sign on to Db2 **must** provide an authorization ID by one of the following methods:

- Specify AUTHTYPE(USERID) or COMAUTHTYPE(USERID) in the appropriate definition for the thread (DB2ENTRY or DB2CONN), to provide the user ID of the CICS user associated with the transaction to Db2 as the primary authorization ID.
- Specify AUTHTYPE(GROUP) or COMAUTHTYPE(GROUP) in the appropriate definition for the thread (DB2ENTRY or DB2CONN), to provide the user ID of the CICS user associated with the transaction to Db2 as the primary authorization ID, and the name of a RACF group or list of groups as the secondary authorization IDs.
- Specify AUTHTYPE(SIGN) in the appropriate definition for the thread (DB2ENTRY or DB2CONN), and specify the CICS region user ID in the SIGNID attribute of the DB2CONN, to provide the CICS region ID to Db2 as the primary authorization ID.

Note that if the RACF access control environment element (ACEE) in the CICS region is changed in a way that affects the CICS Db2 attachment facility, Db2 is not aware of the change until a sign-on occurs. You can use the CEMT or EXEC CICS SET DB2CONN SECURITY(REBUILD) command to cause the CICS Db2

attachment facility to issue a Db2 sign-on the next time a thread is reused, or when a thread is built on an already signed-on TCB. This ensures that Db2 is made aware of the security change.

## Providing a primary authorization ID for CICS transactions

When a CICS transaction's thread TCB signs on to Db2, it must provide a primary authorization ID to Db2. The ID that a transaction uses as its primary authorization ID is determined by an attribute in the resource definition for the thread that the transaction uses to access Db2.

### About this task

This means that all transactions that use the same type of thread (either the same type of entry thread, or pool threads, or command threads) must use the same type of primary authorization ID. In each CICS region, you need to set a primary authorization ID for:

- Each type of entry thread, using your DB2ENTRY definitions.
- The pool threads, using your DB2CONN definition.
- The command threads (used for the DSNB transaction), using your DB2CONN definition.

Before you start to set primary authorization IDs, ensure that you have authority to do so. As well as having authority to change your DB2CONN or DB2ENTRY definitions, if surrogate user checking is in force for the CICS region (that is, the system initialization parameter XUSER is set to YES), you need to obtain special authority to perform operations involving Db2 authorization IDs. These operations are modifying the AUTHID, COMAUTHID, AUTHTYPE, or COMAUTHTYPE attributes on a DB2ENTRY or DB2CONN definition, and modifying the SIGNID attribute on a DB2CONN definition. [“Using surrogate security and AUTHTYPE security to control access to the authorization IDs that CICS provides to Db2” on page 498](#) tells you how to grant users authority to perform these operations.

There are two methods of setting the primary authorization ID for a particular type of thread:

1. Use the AUTHID attribute in the DB2ENTRY definition (for entry threads), or the AUTHID or COMAUTHID attribute in the DB2CONN definition (for pool threads or command threads), to specify a primary authorization ID. For example, you could define AUTHID=test2. In this case, the CICS Db2 attachment facility passes the characters TEST2 to Db2 as the primary authorization ID.

Using AUTHID or COMAUTHID does not permit the use of secondary authorization IDs, and also is not compatible with the use of RACF, or an equivalent external security manager, for security checking in the Db2 address space.

2. Use the AUTHTYPE attribute in the DB2ENTRY definition (for entry threads), or the AUTHTYPE or COMAUTHTYPE attribute in the DB2CONN definition (for pool threads or command threads), to instruct CICS to use an existing ID that is relevant to the transaction as the primary authorization ID. This ID can be a CICS user ID, operator ID, terminal ID, or transaction ID; or it can be an ID that you have specified in the DB2CONN definition for the CICS region.

Using AUTHTYPE or COMAUTHTYPE is compatible with the use of RACF (or an equivalent external security manager) for security checking in the Db2 address space, if you use the USERID or GROUP options, and with the use of secondary authorization IDs, if you use the GROUP option.

The two methods of determining the primary authorization ID are mutually exclusive; you cannot specify both AUTHID and AUTHTYPE, or COMAUTHID and COMAUTHTYPE, in the same resource definition.

Remember that all IDs that you select as primary authorization IDs must be defined to RACF, or your equivalent external security manager, if the security manager is active for the Db2 subsystem. For RACF, the primary authorization IDs must be defined as RACF USER profiles, not just as RESOURCE profiles (for example, as a terminal or transaction).

Follow the instructions in [DB2CONN resources](#) and [DB2ENTRY resources](#) to set up or modify DB2CONN and DB2ENTRY definitions. If you are using the AUTHTYPE or COMAUTHTYPE attributes to determine the primary authorization ID for a type of thread, use [Table 45 on page 506](#) to identify the options that provide the required authorization ID and support the facilities you want. The key points to consider are:

- If you want to provide secondary authorization IDs to Db2 as well as a primary authorization ID, you need to select the GROUP option. When you specify the GROUP option, your primary authorization ID is automatically defined as your CICS user ID, but you can base your security checking on the secondary authorization IDs instead.
- If you are using RACF for security checking in the Db2 address space, you need to select either the GROUP option, or the USERID option. Only these options can pass the RACF access control environment element (ACEE) to Db2, which is required when RACF is used for security checking.
- Think about the performance and maintenance implications of your choice of authorization ID. [Selecting authorization IDs for performance and maintenance](#) outlines these. With the USERID, OPID, TERM, TX or GROUP options, sign-on processing occurs more frequently, and maintenance also takes more time, because you need to grant permissions to a greater number of authorization IDs. With the SIGN option, or using the AUTHID attribute instead of the AUTHTYPE attribute, sign-on processing is decreased, and maintenance is less complicated. However, using standard authorization IDs makes the Db2 security checking less granular.
- Think about the accounting implications of your choice of authorization ID. The authorization ID is used in each Db2 accounting record. From an accounting viewpoint, the most detailed information is obtained if using USERID, OPID, GROUP or TERM. However, depending on the ACCOUNTREC specification, it may not be possible to account at the individual user level in any case. For more information about accounting in a CICS Db2 environment, see [Accounting and monitoring in a CICS Db2 environment](#).

Table 45 on page 506 shows the primary authorization IDs that the CICS Db2 attachment facility passes to Db2 when you select each option for the AUTHTYPE or COMAUTHTYPE attributes.

Table 45. Options available on the AUTHTYPE and COMAUTHTYPE attributes			
Option	Primary authorization ID passed to Db2	Supports RACF checking for Db2?	Supports secondary auth IDs?
USERID	User ID associated with the CICS transaction, as defined to RACF and used in CICS sign-on	Yes	No
OPID	User's CICS operator ID, defined in the CICS segment of the RACF user profile	No	No
SIGN	An ID you specified in the SIGNID attribute of the DB2CONN definition for the CICS region. Defaults to the applid of the CICS region	Yes, when the SIGNID attribute of the DB2CONN resource matches the CICS region userid.	No
TERM	Terminal ID of the terminal associated with the transaction	No	No
TX	Transaction ID	No	No
GROUP	User's CICS RACF user ID used in CICS sign-on	Yes	Yes

If you are not planning to provide secondary authorization IDs for your CICS transactions, you do not need to replace the default Db2 sign-on exit routine DSN3@SGN. The default sign-on exit routine handles primary authorization IDs. However, the Db2 subsystem to which you are connecting might use a different sign-on exit routine for some other reason. If the Db2 subsystem uses the sample sign-on exit routine DSN3SSGN, you might need to make a change to DSN3SSGN, if **all** of the following conditions are true:

- You have chosen an AUTHID or AUTHTYPE option other than GROUP.
- RACF list of groups processing is active.
- You have transactions whose primary authorization ID is not defined to RACF.

If this is the case, see [Securing Db2 in Db2 for z/OS product documentation](#) for the change you need to make to the sample sign-on exit routine.



## Providing secondary authorization IDs for CICS transactions

When a thread TCB that belongs to a CICS transaction signs on to Db2, it can provide one or more secondary authorization IDs to Db2, in addition to the primary authorization ID.

### About this task

You can provide Db2 with the name of a CICS user's RACF group, or list of groups, as secondary authorization IDs. This has the advantage that you can grant Db2 privileges and authority to RACF groups, rather than to each individual CICS user. CICS users can then be connected to, or removed from, RACF groups as required. [RACF users and roles](#) explains how users can be placed into RACF groups.

You can only provide secondary authorization IDs to Db2 for CICS transactions if you specify the GROUP option for the AUTHTYPE attribute in the DB2ENTRY definition (for entry threads), or the AUTHTYPE or COMAUTHTYPE attributes in the DB2CONN definition (for pool threads or command threads). If you specify any other option for AUTHTYPE or COMAUTHTYPE, the secondary authorization ID is set to blanks. When you specify the GROUP option, you cannot choose the primary authorization ID for the thread type; it is automatically defined as the user ID of the CICS user associated with the transaction. You should base your security checking on the secondary authorization IDs instead.

To provide the names of a user's RACF groups to Db2 as secondary authorization IDs, complete the following steps:

1. Specify SEC=YES as a system initialization parameter for the CICS region, to ensure that CICS uses RACF. If your CICS transaction profile names are defined with a prefix, also specify the system initialization parameter SECPRFX=YES or SECPRFX=*prefix*.
2. If the CICS region is using MRO:
  - a. Ensure that each connected CICS region is also using RACF security (SEC=YES).
  - b. Specify ATTACHSEC=IDENTIFY on the CONNECTION definition for the TOR, to ensure that sign-on information is propagated from the TOR to the AORs.
3. Replace the default Db2 sign-on exit routine DSN3@SGN. This sign-on exit routine is driven during sign-on processing. The default sign-on exit routine does not support secondary authorization IDs, so you need to replace it with the sample sign-on exit routine DSN3SSGN, which is provided with Db2, or with your own routine. DSN3SSGN is shipped in source form in the Db2 SDSNSAMP library, and you can use it as a basis for your own routine. For information about sign-on exits and about writing exit routines, see [Securing Db2 in Db2 for z/OS product documentation](#). If the RACF list of groups option is not active, DSN3SSGN passes the name of the current connected group as the secondary authorization ID. If the RACF list of groups is active, DSN3SSGN obtains the names of all the groups to which the user is connected, and passes them to Db2 as secondary authorization IDs.
4. Use the AUTHTYPE attribute in the DB2ENTRY definition (for entry threads), or the AUTHTYPE or COMAUTHTYPE attribute in the DB2CONN definition (for pool threads or command threads), to specify the GROUP option as the authorization type for each type of thread for which you want to provide secondary authorization IDs. If surrogate user checking is in force for the CICS region (that is, the system initialization parameter XUSER is set to YES), you need to obtain special authority to perform operations involving Db2 authorization IDs. [“Using surrogate security and AUTHTYPE security to control access to the authorization IDs that CICS provides to Db2”](#) on page 498 tells you how to do this.

If you have successfully completed all the steps listed above, when a CICS transaction's thread TCB signs on to Db2 with the types of thread that you have defined with the GROUP option, the CICS user's user ID will be passed to Db2 as the primary authorization ID, and the user's RACF group or list of groups will be passed to Db2 as secondary authorization IDs. If you have not successfully completed all the steps, the CICS Db2 attachment facility will issue an authorization failed message.

As an alternative to providing the names of RACF groups as secondary authorization IDs to Db2, you can write your own sign-on exit routine that sets secondary IDs for CICS transactions. See [Securing Db2 in Db2 for z/OS product documentation](#).

## Authorizing users to access resources in Db2

Once the user has accessed the Db2 address space from a CICS transaction they might need permission to issue Db2 commands or execute a plan.

### About this task

Access to the Db2 resources that a CICS user needs to issue Db2 commands or execute a plan is subject to security checking by the Db2 security mechanisms. You can choose to have this security checking carried out by:

- Db2 internal security.
- RACF, or an equivalent external security manager.
- Partly Db2, and partly RACF.

For more information about setting up RACF to perform security checking in the Db2 address space, see [Securing Db2 in Db2 for z/OS product documentation](#).

If you are using RACF for some or all of the security checking in your Db2 address space, remember that CICS transactions that sign on to Db2 must provide an authorization ID. For more information, see [“Providing authorization IDs to Db2 for CICS transactions” on page 504](#). CICS must also be using RACF (SEC=YES must be specified in the SIT). This is because when RACF is used for security checking in the Db2 address space, CICS needs to pass a RACF access control environment element (ACEE) to Db2. CICS can only produce an ACEE if it has RACF active, and only threads defined with the GROUP, SIGN or USERID option can pass the ACEE to Db2.

When the ACEE is passed to Db2, it is used by the Db2 exit DSNX@XAC, which determines whether RACF, or an equivalent non-IBM external security manager, or Db2 internal security is used for security checking. DSNX@XAC is driven when a transaction whose thread has signed on to Db2, issues API requests. You can modify DSNX@XAC. For more information, see [Securing Db2 in Db2 for z/OS product documentation](#).

Db2, or the external security manager, performs security checking using the authorization IDs that the CICS transaction provided to Db2 when the thread that it was using signed on to Db2. The authorization IDs could be related to the individual CICS user (for example, the CICS user's user ID and the RACF groups to which the user is connected), or they could be related to the transaction (for example, the terminal ID or transaction ID), or they could be related to the whole CICS region. For more information, see [“Providing authorization IDs to Db2 for the CICS region and for CICS transactions” on page 501](#).

Db2, or the external security manager, checks that you have given the authorization IDs permission to perform the relevant actions in Db2. You can give the authorization IDs this permission by using GRANT statements in Db2. For information about how to grant, and revoke, Db2 permissions for authorization IDs, see [Securing Db2 in Db2 for z/OS product documentation](#).

### Controlling users' access to Db2 commands

For CICS users, the first security checking related to Db2 commands is performed in the CICS address space, when the user tries to access a CICS transaction that issues Db2 commands. This could be DSNCL, or a user-defined transaction that invokes DFHD2CM1 and runs an individual Db2 command.

### About this task

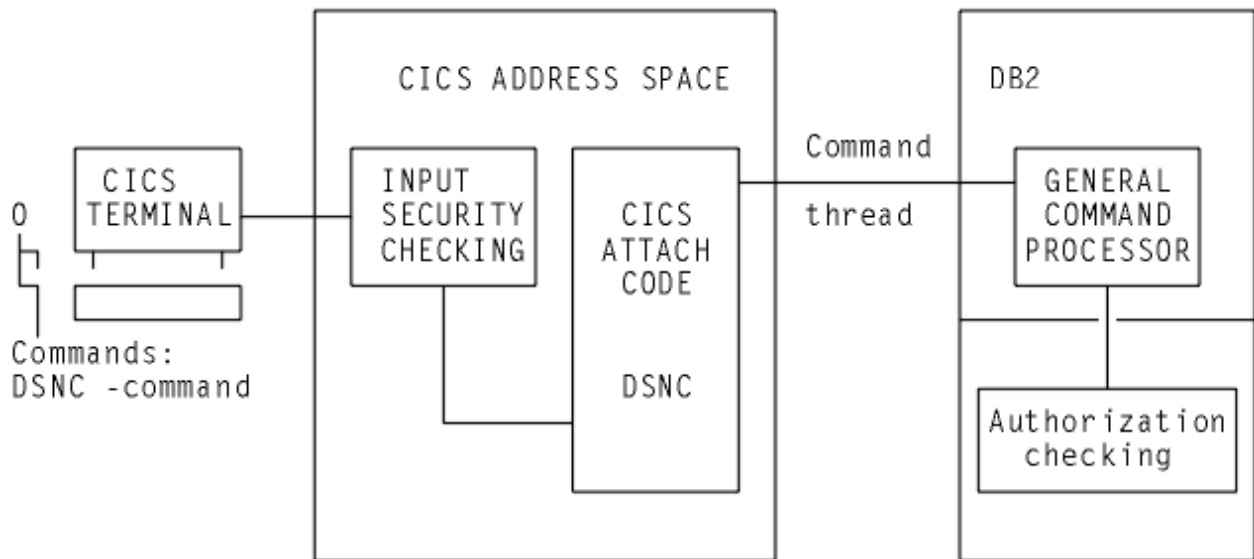
[“Controlling users' access to Db2-related CICS transactions” on page 500](#) describes how to control users' access to transactions that issues Db2 commands in the CICS address space.

When a user issues a Db2 command through a CICS transaction, they are also subject to Db2 security checking, which verifies that they are authorized to Db2 to issue the command. This security checking uses the authorization IDs (primary or secondary) that the transaction has passed from CICS. [“Providing authorization IDs to Db2 for the CICS region and for CICS transactions” on page 501](#) tells you how to choose these authorization IDs and provide them to Db2. For transactions that use DFHD2CM1 to issue Db2 commands, the authorization IDs are set by the COMAUTHID or COMAUTHTYPE attribute of the CICS



region's DB2CONN definition. For other applications that issue Db2 commands, the authorization IDs are set by the AUTHID or AUTHTYPE attribute for the CICS region's resource definition for the type of thread used by the transaction (pool thread or entry thread). These attributes control the authorization ID, or type of authorization ID, that is passed to Db2 by a transaction that is using that type of thread.

Db2 commands are therefore subject to two security checks, one in the CICS address space and one in the Db2 address space. [Figure 165 on page 509](#) illustrates the process.



*Figure 165. Security mechanisms for Db2 commands*

In most cases, only a limited number of users are permitted to execute Db2 commands. A convenient solution can be to specify COMAUTHTYPE(USERID) on the DB2CONN definition, which resolves to the 8-byte CICS user ID as the authorization ID in Db2. Using this method, you can give different Db2 privileges explicitly to CICS user IDs. For example, you can use GRANT DISPLAY to give specific CICS user IDs permission to use only the -DIS command.

To authorize a user to issue a Db2 command, use a GRANT command to grant Db2 command privileges to the authorization ID that the transaction has passed from CICS. For information about how to grant, and revoke, Db2 permissions for authorization IDs, see [Securing Db2 in Db2 for z/OS product documentation](#).

## Controlling users' access to plans

A number of checks take place before a user can access plans in Db2. These checks start in the CICS address space before the request is passed to Db2 for further checks.

### About this task

For Db2 commands, the first security check for users' access to plans takes place in the CICS address space, when CICS verifies that the user is permitted to access the transaction that will execute the plan. The second security check takes place in the Db2 address space, when Db2 verifies that the authorization ID provided by the transaction, is authorized to execute the plan. [Figure 166 on page 510](#) illustrates this process.

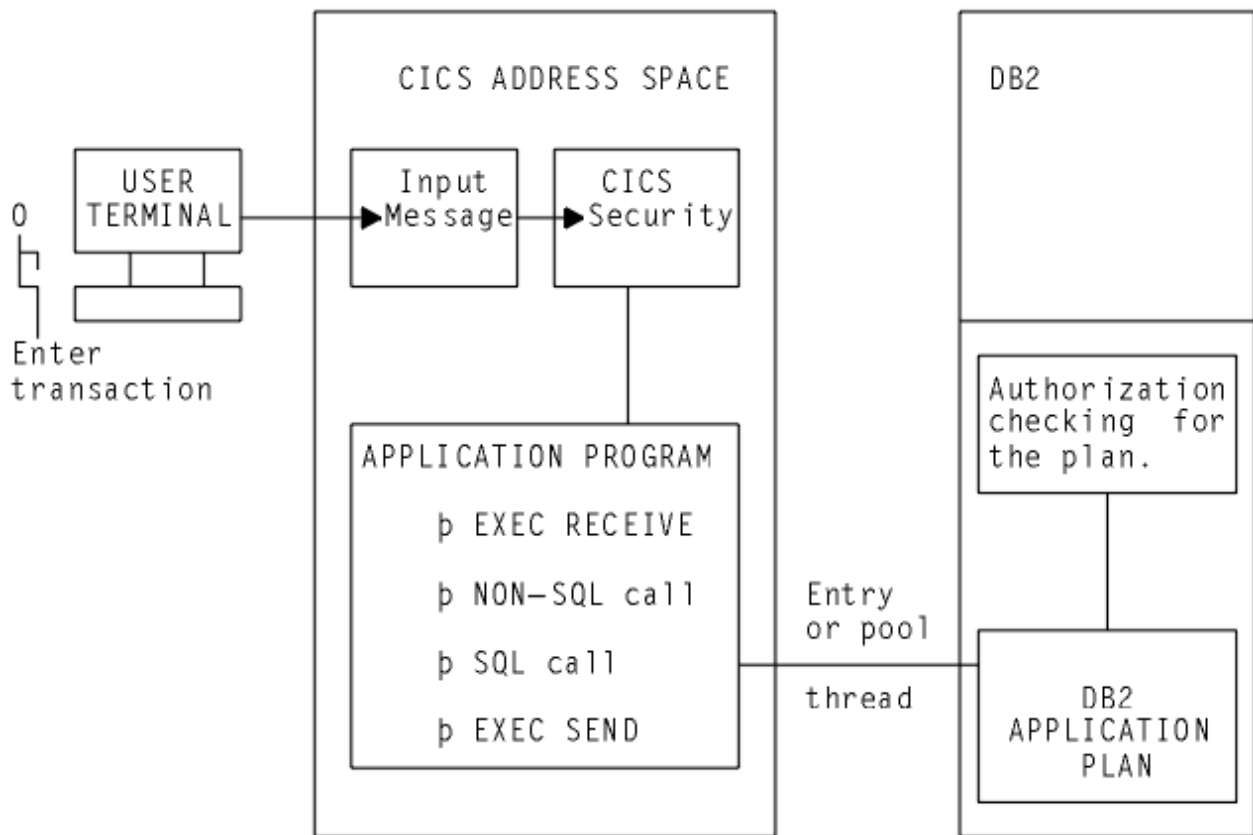


Figure 166. Security mechanisms for executing a plan

To authorize a user to execute a plan, use a GRANT command to grant Db2 command privileges to the authorization ID that the transaction has passed from CICS. For information about how to grant, and revoke, Db2 permissions for authorization IDs, see [Securing Db2 in Db2 for z/OS product documentation](#).

### ***If a plan includes dynamic SQL***

When using static SQL, the binder of the plan must have the privileges needed to access the data, and the authorization ID passed from CICS to Db2 need only have the privileges to execute the plan.

### **About this task**

However, if a plan includes the use of dynamic SQL, the authorization ID passed from CICS to Db2 must possess the privileges required to access all the Db2 resources involved, both the plan and the data. For example, if you specify AUTHTYPE(USERID), the CICS user ID must be granted Db2 privileges to the Db2 resources involved in the dynamic SQL. If this user ID is also a TSO user ID, it has access to the Db2 resources directly from SPUFI, QMF, and other utilities.

If you do not want to spend too much time granting Db2 privileges, where a transaction executes a plan that involves the use of dynamic SQL, consider using one of the following methods of supplying an authorization ID to Db2:

- Use the SIGN option on the AUTHTYPE attribute of the DB2ENTRY definition for the thread used by the transaction. This results in the transaction having the primary authorization ID that you specified in the SIGNID attribute of the DB2CONN definition for the CICS region. (This method is not suitable where RACF is used for security checking in the Db2 address space.)
- Use the AUTHID attribute of the DB2ENTRY definition for the thread used by the transaction, to specify a standard authorization ID. Use the same authorization ID for all the transactions that need to access dynamic SQL. (This method is not suitable where RACF is used for security checking in the Db2 address space.)

- Create a RACF group, and connect your CICS users to this RACF group. Use the GROUP attribute of the DB2ENTRY definition for the thread used by the transaction, so that the RACF group is one of the secondary IDs that is passed to Db2.

In each case, you can then grant Db2 privileges for Db2 resources involved in all dynamic SQL to a single ID, either the standard authorization ID from the DB2CONN definition or the AUTHID attribute, or the name of the RACF group. [“Providing authorization IDs to Db2 for the CICS region and for CICS transactions” on page 501](#) tells you how to provide authorization IDs by all these methods.

## Db2 multilevel security and row-level security

Db2 Version 8 introduced support for multilevel security. CICS does not provide specific support for multilevel security, but you can use CICS in a multilevel-secure environment provided that you take care with the configuration.

For more information about multilevel security, see the following publications:

- [Securing Db2 in Db2 for z/OS product documentation](#)
- [z/OS Planning for Multilevel Security and the Common Criteria](#)
- [IBM Redbooks: Securing DB2 and Implementing MLS on z/OS](#)

When multilevel security is implemented at row level (row-level security) for data in Db2 Version 8 or later, the RACF SECLABEL class is activated, and a set of security labels is defined for users and for the Db2 table rows. The RACF options SETR MLS and MLACTIVE are not required to be active. You can use Db2 row-level security without impact on the rest of the MVS system.

CICS is able to access Db2 rows secured in this way. For CICS, you need to ensure that the RACF user profile for a CICS user that needs access to the Db2 rows is defined in RACF to include a default SECLABEL. For details, see the [z/OS Security Server RACF Security Administrator's Guide](#).

When a CICS user signs on to a CICS region with SEC=YES specified in the SIT, RACF associates the default SECLABEL with the RACF access control environment element (ACEE) for the user. The DB2ENTRY definition (or DB2CONN definition if the pool is being used) needs to specify AUTHTYPE=USERID or AUTHTYPE=GROUP, which ensures the ACEE is passed on to Db2 for further security checking. An individual CICS user can therefore only have one associated SECLABEL.

For non-terminal tasks or programs, such as PLT programs, if the PLTPIUSR system initialization parameter is not specified and the PLTPISEC=NONE system initialization parameter is specified, PLT programs are run under the CICS region userid. In this case, you need to define the CICS region userid with a default SECLABEL. If you need to define different SECLABELS for a transaction, you would need to run each transaction in a separate CICS region which has a different CICS region userid and associated SECLABEL.

## Security for DBCTL

When you use CICS with DBCTL, you can secure DBCTL in CICS using PSB authorization checking by CICS, resource access security checking by DBCTL, DBCTL password security checking, or all those options.

You can use one or more of the following optional security facilities:

- [“PSB authorization checking by CICS” on page 512](#)
- Resource access security checking by DBCTL
- DBCTL password security checking

For details about resource access security checking by DBCTL and DBCTL password security checking, see [System administration in IMS product documentation](#).

Of the resources you can protect by using IMS security, you only need to be concerned about only with PSBs, databases, and commands.

## PSB authorization checking by CICS

At PSB scheduling time, CICS invokes security checking to determine whether the terminal user is authorized to access the PSB. The actual check is carried out by an external security manager, which can be RACF or your own security program.

Although PSB scheduling requests are sent to DBCTL for processing, CICS does PSB authorization checking. For programming information about writing your own security program, see [Invoking an external security manager](#).

## Security for data sets: encryption

You can encrypt any data sets that you use with CICS for which z/OS data set encryption is supported. This includes user data sets that are accessed through CICS File Control APIs, queued sequential access method (QSAM) data sets used for CICS extrapartition transient data, basic sequential access method (BSAM) data sets used with CICS, and CICS system data sets that are appropriate candidates for encryption.

You can use data set encryption with any in-service release of CICS TS for z/OS.

Encrypted data sets must be storage management subsystem (SMS)-managed and extended format. To create an encrypted data set, you assign a key label to a data set when that new data set is allocated. The key label must point to an AES-256 bit encryption key in the integrated cryptographic service facility (ICSF) cryptographic key data set (CKDS) that will be used to encrypt or decrypt the data. The key label is not sensitive information, but the encryption key that it identifies is.

### Encryption support for CICS user data sets

For user data sets defined to CICS, encryption support includes key-sequenced data sets (KSDS), entry-sequenced data sets (ESDS), relative record data sets (RRDS), and variable relative record data sets (VRRDS), accessed through base VSAM and VSAM Record Level Sharing (RLS). Encryption is also supported for the backing VSAM key-sequenced data sets that are used for shared data tables or coupling facility data tables.

### Encryption support for CICS system data sets

You can encrypt any CICS system data sets for which encryption is supported, but some system data sets are good candidates for encryption, whereas others are not.

- Data sets with the *potential* to contain sensitive data are candidates for encryption. You must assess which of your data sets might contain sensitive data and whether to encrypt them.
- Data sets that do not, and are unlikely to, contain sensitive data are not candidates for encryption.

The following table lists the CICS system data sets for which encryption is possible, and whether they are candidates for encryption. If you want to use an "encrypt everything" approach, all the data set types that are listed in this table can be encrypted.

Table 46. Which system data sets are candidates for encryption?		
CICS system data set	Candidate for encryption?	Special considerations
Temporary storage data set (DFHTEMP)	Yes, could contain sensitive data.	DFHTEMP supports extended format, but does <b>not</b> support extended addressing.
Intrapartition Transient Data (DFHINTRA)	Yes, could contain sensitive data.	DFHINTRA supports extended format, but does <b>not</b> support extended addressing.
Extrapartition Transient Data	Yes, could contain sensitive data.	Encryption is not supported for partitioned data sets (PDS).

Table 46. Which system data sets are candidates for encryption? (continued)

CICS system data set	Candidate for encryption?	Special considerations
Auxiliary Trace data sets (DFHAUXT and DFHBUXT)	Yes, can potentially include sensitive data in the diagnostics. Alternatively, use the <a href="#">CONFDATA</a> system initialization parameter, which might provide sufficient protection.	If trace data is encrypted and you need to send it to IBM for diagnostics, use CICS trace formatting or another method to ensure that you send decrypted data.
CICS dump data sets (DFHDMPA and DFHDMPB)	Yes, can potentially include sensitive data in the diagnostics. Alternatively, use the <a href="#">CONFDATA</a> system initialization parameter, which might provide sufficient protection.	If dump data is encrypted and you need to send it to IBM for diagnostics, use CICS dump formatting or another method to ensure that you send decrypted data.  CICS dump data sets support extended format, but do <b>not</b> support extended addressing.
Doctemplate resources	Yes, can potentially contain sensitive data.	Provided that the data set used is of a type for which encryption is supported.
URIMAP resources used for static delivery	Yes, can potentially contain sensitive data.	Provided that the data set used is of a type for which encryption is supported.
BTS repository data sets and BTS local request queue (LRQ) data set	No, contain only control data.	None.
Global and Local catalog data sets (DFHGCD and DFHLCD)	No, contain only configuration data.	Consider only if you believe that CICS configuration data is sensitive information.
CICS system definition data set (DFHCSD)	No, contains only information about resource configuration.	Consider only if you believe that your resource definitions include any sensitive information.
CMAC messages data set (DFHCMACD)	No, contains only message details.	Consider only if you added your own messages that you believe contain sensitive data.
zFS files used for bundle definitions and web services	No, contain only configuration information.	Consider only if you believe that your bundle definitions include any sensitive information.

#### Related information

[Encrypting data sets](#)



---

## Chapter 26. Security for EXCI

CICS applies security checks in a number of ways against requests received from an MVS client program.

### Using MRO logon and bind-time security

---

DFHIRP, the CICS interregion communication program, performs two security checks against users that want to either log on to IRP (specific connections only), or connect to a CICS region (also referred to as bind-time security).

#### About this task

**Generic EXCI connections:** The discussion about logon security checking in this section applies only to EXCI connections that are defined as SPECIFIC. The MRO logon security check is not performed for generic connections.

The MVS client program is treated just the same as another CICS region as far as MRO logon and connect (bind-time) security checking is concerned. This means that when the client program logs on to the interregion communication program, IRP performs logon and bind-time security checks against the user ID under which the client program is running. In the remainder of this information, this user ID is called the user ID of the batch region.

To enable your client program to log on to IRP successfully, and to connect to the target server region, first ensure that you define the user ID of the batch region in a user profile to RACF. After you define the user ID of the batch region to RACF, you can then give the batch region the appropriate logon and bind-time authorizations.

#### Procedure

##### Logon authorization

- Authorize the user ID of the batch region to the DFHAPPL.*user\_name* RACF FACILITY class profile, with UPDATE authority. The *user\_name* part of the profile name is the user name defined on the INITIALIZE\_USER command.
  - For the EXCI CALL interface, the *user\_name* must be the name you specify on the *user\_name* parameter of the INITIALIZE\_USER command.

Define FACILITY class profiles, with appropriate authorizations, for each user name specified in a client program if the program has INITIALIZE\_USER commands for more than one user name.

For example, if the *user\_name* defined on an INITIALIZE\_USER command is DCEUSER1, define the DFHAPPL profile in the FACILITY class as follows:

```
RDEFINE FACILITY (DFHAPPL.DCEUSER1) UACC(NONE)
```

If the batch region's user ID is CLIENTA, authorize the batch region to log on to IRP as follows:

```
PERMIT DFHAPPL.DCEUSER1 CLASS(FACILITY) ID(CLIENTA)  
ACCESS(UPDATE)
```

- For the EXEC CICS **LINK** command, the *user\_name* is preset by the external CICS interface as DFHXCEIP. This does not require authorization for IRP logon because the EXEC CICS LINK interface uses a generic connection to which the logon security check does not apply.

Failure to authorize the user ID of the batch region to the DFHAPPL profile of the specific user ID logging on to IRP causes Allocate\_Pipe processing to fail with RESPONSE(SYSTEM\_ERROR) REASON(IRC\_LOGON\_FAILURE). The subreason field-1 for a logon security check failure returns decimal 204.

##### Bind-time authorization

- Authorize the user ID of the batch region to the DFHAPPL.*applid* RACF FACILITY class profile of the target CICS server region, with READ authority.

Failure to authorize the user ID of the batch region to the DFHAPPL.*applid* profile of the CICS server region causes Open\_Pipe processing to fail with RESPONSE(SYSTEM\_ERROR) REASON(IRC\_CONNECT\_FAILURE). The subreason field-1 for a bind-time security check failure returns decimal 176.

See [Bind security](#) for information about the MRO logon and bind-time security checks, and for examples of how to define the RACF DFHAPPL profiles.

## Link security for EXCI

---

The target CICS server region performs link security checking against requests from the client program.

These security checks cover transaction attach security (when attaching the mirror transaction), and resource and command security checking within the server application program. The link user ID that CICS uses for these security checks is the batch region's user ID.

To ensure these link security checks do not cause security failures, you must ensure that the link user ID is authorized to the following resource profiles, as appropriate:

- The profile for the mirror transaction, either CSMI for the default, or the mirror transaction specified on the *transid* parameter. This is required for transaction attach security checking.
- The profiles for all the resources accessed by the CICS server application program—files, queues (transient data and temporary storage), programs, and so on. This is required for resource security checking.
- The CICS command profiles for the SPI commands issued by the CICS server application program—INQUIRE, SET, DISCARD and so on. This is required for command security checking.

See [CONNECT security checks for AORs](#) for information about MRO link security checking.

## User security for EXCI

---

The target CICS server region performs user security checking against the user ID passed on a DPL\_Request call. User security checking is performed only when connections specify ATTACHSEC(IDENTIFY).

User security is performed in addition to any link security.

For user security, in addition to any authorizations you make for link security, you must also authorize the user ID specified on the DPL\_Request call.

Note that there is no provision for specifying a user ID on the EXEC CICS LINK command. In this case, the external CICS interface passes the batch region's user ID. User security checking is therefore performed against the batch region's user ID if the connection definition specifies ATTACHSEC(IDENTIFY).

**Note:** If your connection resource definitions for the external CICS interface specify ATTACHSEC(IDENTIFY), your server programs will fail with an ATCY abend if you run them in an environment that does not have RACF, or an equivalent external security manager (ESM), installed and active.

If you want to run external CICS interface server programs without any security active, you must specify ATTACHSEC(LOCAL).

## Surrogate user checking for EXCI

---

EXCI client jobs are subject to surrogate user checking (see [Surrogate security](#)). You must authorize the batch region's user ID as a surrogate of the user ID specified on all DPL\_Request calls. This configuration means that the batch region's user ID must have READ access to a profile named *execution-userid*.DFHEXCI in the RACF SURROGAT general resource class (where *execution-userid* is the user ID that is specified on the DPL call).



For example, the following commands define a surrogate profile for a DPL user ID, and grant READ access to the EXCI batch region:

```
RDEFINE SURROGAT execution_userid.DFHEXCI UACC(NONE)
PERMIT execution_userid.DFHEXCI CLASS(SURROGAT) ID(batch_region_userid) ACCESS(READ)
```

If no user ID is specified on the DPL\_Request call, no surrogate user check is performed because the user ID on the DPL\_Request call defaults to the batch region's user ID. For this bypass of surrogate user checking to be successful, ensure that you have correctly omitted the user ID on the DPL\_Request call. See the example of EXCI CALLs with null parameters in [The EXCI CALL interface](#) for the correct way to specify a null pointer when you omit an EXCI call parameter.

If the batch region user ID and the CICS region user ID are different, link security checking is enforced. With link security, a nonauthenticated user ID passed on a DPL\_Request call cannot acquire more authority than that allowed by the link security check. It can acquire only the same, or less, authority than that allowed by the link security check.

For more information about CICS security, see [Securing CICS](#).



---

## Chapter 27. Security for ONC RPC

**Important:** This information contains Product-sensitive Programming Interface and Associated Guidance Information.

Security is an important concern in the provision of ONC RPC support in the CICS environment, because CICS ONC RPC provides an Open Systems communications interface into CICS.

ONC RPC has its own security methods (called authentication in RPC) with dedicated fields in the ONC RPC call and reply message headers. There are three types of RPC authentication:

### **UNIX authentication**

Is used to transmit the client's UNIX user ID, group ID, and other identification information.

### **Data Encryption Standard (DES) authentication**

Not available at ONC RPC Version 3.9, and so cannot be used with CICS ONC RPC.

### **Null authentication**

Offers no security checking.

This section describes how CICS ONC RPC interacts with the security facilities of ONC RPC and CICS.

---

## Security in CICS and its effect on CICS ONC RPC operations

During the operation of CICS ONC RPC, various CICS commands are used to make security checks with an external security manager (ESM).

The checks will always give positive results if SEC=NO is specified as a system initialization parameter. The checks will always give negative results if SEC=YES was specified, but the ESM abended while CICS was operating. The following discussion of the use made of CICS security commands assumes that SEC=YES is specified, and that the ESM is active.

- When a transaction whose user ID is *userid1* issues EXEC CICS START USERID(*userid2*), a surrogate-user check is made with the ESM to see that *userid1* is authorized to use *userid2*. The check is made only if XUSER=YES is specified as a system initialization parameter.

This command is issued when the connection manager starts the server controller, and each time the server controller starts an alias transaction. In the first case, the user ID used is the one supplied to the connection manager as CRPM Userid on panel DFHRP02. In the second case, the user ID used is the one output from **Decode**.

- EXEC CICS VERIFY PASSWORD is issued by the alias before it links to the CICS program that services the client request. A check is made with the ESM that the user ID and password are an acceptable combination.
- EXEC CICS QUERY SECURITY is used by the alias to check that the user ID under which it is executing is authorized to use the CICS program. The check is made only if XPPT=YES is specified as a system initialization parameter.
- During the operation of the CICS program, security checks are made each time the program tries to access a protected resource. The check is made only if RESSEC(YES) is specified in the definition of the alias transaction, and the system initialization parameter controlling security checking for the resource type is set to YES.
- During the operation of the CICS program, security checks are made each time the program tries to use a command from the CICS SPI (system programming interface). The check is made only if CMDSEC(YES) is specified in the definition of the alias transaction, and if XCMD=YES is specified as a system initialization parameter.

[Figure 167 on page 520](#) shows how CICS security interacts with the operation of CICS ONC RPC.

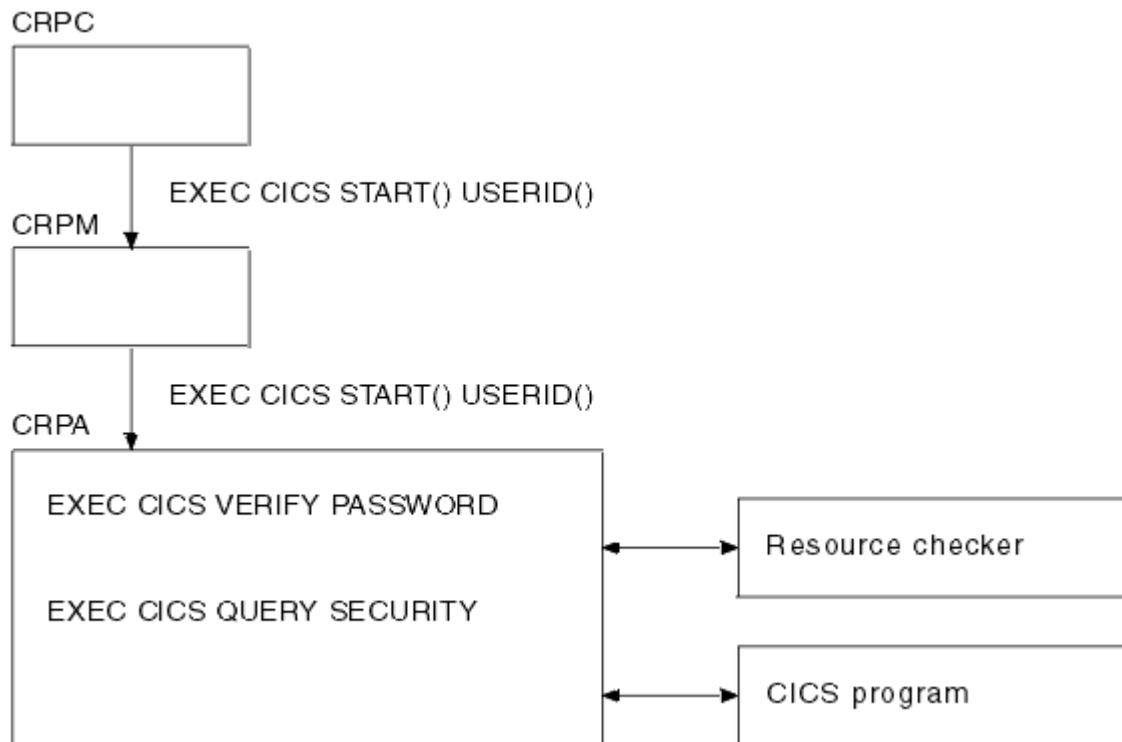


Figure 167. How CICS security interacts with CICS ONC RPC operations

The figure shows that the alias will link to the user-supplied resource checker program if one is configured, but the use of the resource checker program is not recommended. You should use the CICS security facilities, and make the appropriate definitions in the ESM.

## RACF Secured Sign-on for ONC RPC clients

RACF Secured Sign-on support allows RPC clients to gain security access to CICS facilities by sending a PassTicket. This avoids the security hazard of a password being transmitted across the network in clear text.

### PassTicket generation for ONC RPC clients

The algorithm that generates the PassTicket for an ONC RPC client is a function of the following items:

- The CICS user ID of the client.
- The CICS application ID of the CICS region running CICS ONC RPC.
- A secured sign-on application key, known to both sides.
- A time and date stamp.

To generate the PassTicket, the RPC client must:

- Know its CICS user ID, the server CICS application ID, and the application key.
- Synchronize its clock to within ten minutes of the server.
- Have access to the encryption algorithm on its machine. Only the DES algorithm may be used.

For further information, see [z/OS Security Server RACF System Programmer's Guide](#). This includes details of the algorithm that the RPC client must use to generate the PassTicket. This algorithm includes the DES algorithm.

# Writing the resource checker

Your resource checker program must be called DFHRPRSC. There can be only one resource checker in a CICS region.

The resource checker allows you to check the credentials of inbound client requests.

The resource checker can check the client address, passed as an input parameter, against a list of known clients for the host on which the request has been received. The password passed to the resource checker is blank.

## Reference information for the resource checker

The resource checker is optionally invoked by the alias before it attempts to link to the CICS program that is to service the client request. It must say whether the client request is allowed to proceed.

The reference information for the resource checker is presented as follows:

- A summary table of parameters, showing which are for input only, and which for output only.
  - **Input** is for parameters that your resource checker may consult, but not change.
  - **Output** is for parameters that your resource checker must not consult, but may change.
- A description of the processing that the resource checker is expected to do.
- A list of parameters in alphabetical order, with a description of how CICS ONC RPC sets up the inputs, and what use it makes of the outputs.
- A list of the responses and reason codes that the resource checker can return, with a description of the action that CICS ONC RPC takes for each response and reason code.

The descriptions give the names of the program elements as they appear in C. In COBOL the names are all in uppercase, and the underscores are replaced by hyphens.

### Summary of parameters

The format of the communication area containing the resource checker parameters is in the C header file DFHRPRDH, and the COBOL copybook DFHRPRDO. You will also need values defined in the C header file DFHRPUCB, or in the COBOL copybook DFHRPUCO.

Input	Output
<div>res_check_alias_transid res_check_cics_password_ptr res_check_cics_userid res_check_client_ip_address res_check_eyecatcher res_check_host_ip_address res_check_server_program_name</div>	<div>res_check_reason res_check_response</div>

### Parameters

**res\_check\_alias\_transid**  
(Input only)

The 4-character name of the alias transaction that has linked to the resource checker.

**res\_check\_cics\_password\_ptr**  
(Input only)

A pointer to the 8-character password passed from the requesting client or supplied by **Decode**. The value of this field is blank, and it is provided for compatibility with earlier versions of CICS ONC RPC.

**res\_check\_cics\_userid**

(Input only)

The 8-character CICS user ID under which the alias is running.

**res\_check\_client\_ip\_address**

(Input only)

The fullword internet address of the client.

**res\_check\_eyecatcher**

(Input only)

A string of length 8. (Its value is defined in the header file DFHRPUCH and the copybook DFHRPUCO).

**res\_check\_host\_ip\_address**

(Input only)

The fullword internet address of the z/OS Communications Server host with which the server controller is in communication.

**res\_check\_reason**

(Output only)

The reason to be returned to the alias.

**res\_check\_response**

(Output only)

The response to be returned to the alias.

**res\_check\_server\_program\_name**

(Input only)

The 8-character name of the CICS program that is to be invoked to perform the server function requested by the client.

**Response and reason codes**

You must return one of the following values in the **res\_check\_response** field.

**URP\_OK**

The alias will continue to process the client request.

**URP\_EXCEPTION**

The alias writes an exception trace entry (trace point 9F0E), and issues a message that depends on the reason code:

- URP\_AUTH\_BADCRED—message DFHRP0130

An **svcerr\_auth** call with a why-value of AUTH\_BADCRED is used to send a reply to the client.

- URP\_AUTH\_TOOWEAK—message DFHRP0184

An **svcerr\_auth** call with a why-value of AUTH\_TOOWEAK is used to send a reply to the client.

- Any other value—message DFHRP0185

An **svcerr\_systemerr** call is used to send a reply to the client.

**URP\_INVALID**

The alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0186).

An **svcerr\_systemerr** call is used to send a reply to the client.

**URP\_DISASTER**

The alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0187).

An **svcerr\_systemerr** call is used to send a reply to the client.

If you return any other value in **res\_check\_response**, the alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0188). An **svcerr\_systemerr** call is used to send a reply to the client.

You can supply a 32-bit reason code in conjunction with the response value to provide further information in error cases. CICS ONC RPC does not take any action on the reason code returned by the resource checker, except as indicated previously under URP\_EXCEPTION. The reason code is output in any trace or messages that result from the resource checker, and you may use it as a debugging aid.

See [Numeric values of response and reason codes in CICS ONC RPC trace](#) for the numeric values of the response and CICS-defined reason codes in trace output.





---

## Chapter 28. Security for data tables

This topic describes how to provide security for CICS shared data tables and coupling facility data tables.

### Security for CICS shared data tables

---

The CICS shared data tables (SDT) facility reproduces the main characteristics of function-shipping security that operate at the region level, with some differences. For example, SDT does not provide mechanism for the FOR to perform security checks at the transaction level. It does not support any equivalent to preset security on SESSIONS. Neither does it pass any installation parameter list information to the security user exists.

To provide security for a shared data table when **cross-memory services** are used, ensure that:

- The file-owning region (FOR) that is acting as the shared data table server cannot be impersonated. See [“SDT server authorization security check”](#) on page 525 for details of how you ensure this.
- An application-owning region (AOR) cannot gain access to data that it is not meant to access. You can prevent this by checking at CONNECT time that the AOR is allowed access to the FOR and, if file security is in force, that the AOR is allowed access to the requested file.

These security checks are performed through the system authorization facility (SAF), to invoke RACF or an equivalent security manager.

**Note:** A region is still able to use data tables locally even if it does not have authority to act as a shared data table server.

#### How does SDT security work

The CICS shared data tables facility reproduces the main characteristics of function-shipping security that operate at the region level, but note the following differences:

- SDT does not provide any mechanism for the FOR to perform security checks at the transaction level (there is no equivalent of ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY)). Therefore, if you consider that the transaction-level checks performed by the AOR are inadequate for some files, ensure that those files are not associated with data tables in the FOR.
- SDT does not support any equivalent of preset security on SESSIONS, because no sessions are used.
- SDT does not pass any installation parameter list (INSTLN) information to the security user exists.

#### Implications of security checking for SDTs

Consider the implications of the security checks before sharing a file that is associated with a data table.

SDT security makes use of existing CICS file security definitions, but it also relies on treating SDT server APPLIDs as protected resources. An SDT server's APPLID is represented by a DFHAPPL.*applid* profile in the RACF FACILITY resource class.

#### SDT server authorization security check

When a region attempts to be an SDT server, it calls RACF to check whether its user ID has the required access authority to its APPLID.

If the call fails, the region cannot initialize the required SDT support to be a server. This minimizes the risk that an AOR might accept **counterfeit data records** from an FOR that is not properly authorized to act as an SDT server. This check is never bypassed, even when SEC=NO is specified at system initialization.

To act as a server for a protected APPLID, an SDT FOR's userid must have UPDATE (or higher) access to its DFHAPPL.*applid* profile in the FACILITY class. In the following example definitions, the APPLID of the FOR is CICSHF01, and its user ID is CICSSDT1:

```
RDEFINE FACILITY (DFHAPPL.CICSHF01) UACC(NONE)
PERMIT DFHAPPL.CICSHF01 CLASS(FACILITY) ID(CICSSDT1) ACCESS(UPDATE)
```

The first example authorizes one FOR to act as a server with APPLID CICSHF01, running under user ID CICSSDT1. The following example shows how to authorize a group of FORs, with user IDs defined as members of group SDTGRP1, to act as SDT servers using a generic profile in the FACILITY class:

```
RDEFINE FACILITY (DFHAPPL.CICSTST*) UACC(READ)
PERMIT DFHAPPL.CICSTST* CLASS(FACILITY) ID(SDTGRP1) ACCESS(UPDATE)
```

## If SAF neither grants nor refuses an access request

If a security profile for a specified resource is not retrieved, SAF neither grants nor refuses the access request.

In this situation:

- The request fails if a security manager is installed but is either temporarily inactive or inoperative for the duration of this MVS IPL. This decision is made on the grounds that had the security manager been active it might have retrieved a profile that refuses access.
- The request succeeds if:
  - There is no security manager at all.
  - There is an active security manager but the FACILITY class is undefined or inactive.
  - There is no profile covering the APPLID in question.

The request is allowed in these cases because there is no evidence that you want to control access to the particular FOR APPLID.

## CONNECT security checks for AORs

The security checks performed at CONNECT time provide two levels of security, bind security and file security.

- **Bind security** allows an FOR that runs without CICS file security to be able to restrict shared access to selected AORs. (Running without file security minimizes runtime overheads and the number of security definitions.)
- **File security** can be activated in the FOR if you want SDT to implement those checks that apply to the AOR as a whole.

Note that SDT provides no way of implementing those security checks that an FOR makes at the transaction level when ATTACHSEC(IDENTIFY) or ATTACHSEC(VERIFY) is used with function shipping.

### Bind security

To be allowed shared access to any of an FOR's data tables, an AOR's user ID needs READ (or higher) access to the FOR's DFHAPPL.*applid* in the FACILITY class.

This check is never bypassed, even when SEC=NO is specified at system initialization. In the following example definitions, three CICS AORs (user IDs are CICSAOR1, CICSAOR2, and CICSAOR3) all require SDT access to the FOR represented by the DFHAPPL.CICSHF01 profile:

```
PERMIT DFHAPPL.CICSHF01 CLASS(FACILITY) ID(CICSAOR1 CICSAOR2 CICSAOR3)
ACCESS(UPDATE)
```

Cases when SAF neither grants nor refuses access are resolved in the same way as for server LOGON (see [“If SAF neither grants nor refuses an access request” on page 526](#)). If the result is a refusal, CICS does not permit shared access by the AOR to the FOR's APPLID.

Note that controlling SDT server authorization security and bind security by using different (but hierarchical) levels of access to the same resource has the following consequences:

- Any region with the same user ID as a server can always bind to that server.
- It is impossible to control which user IDs can bind to a given APPLID without also controlling which user IDs can log on as servers for that APPLID.

SDT bind-time security uses different definitions from those employed by IPIC, ISC, and (if using preset sessions) MRO. Therefore, unless you make them consistent, SDT access might be granted when function shipping attempts are rejected, or vice versa. Both MRO and SDT use the same class and so, with ISC only, SDT CONNECT security might react to changes in security definitions either earlier or later than function shipping.

If file security is not in force in the FOR (that is, if SEC=NO or XFCT=NO was specified at system initialization), an AOR that is allowed to bind to an FOR is also allowed to access all that FOR's shared data tables.

If file security is in force, an AOR that is allowed to bind is still allowed free access if the user IDs of the AOR and FOR are the same (undefined user IDs are not considered to be the same).

## File security

After the bind-security check, and when file security is in force in the FOR, the FOR checks whether the AOR is authorized to sign on to the FOR.

This security check is optional, and applies only when the user ID of the AOR is different from that of the FOR. It is the equivalent of ATTACHSEC(LOCAL) in an MRO environment (see [MRO user security](#)). The AOR also requires READ authorization to the file it is trying to access in the FOR.

To implement file security checking by the FOR:

- Initialize the FOR with system initialization parameter SEC=YES
- Authorize the AOR with READ access to the FOR's APPLID profile in the APPL general resource class
- Specify the appropriate value on the XFCT system initialization parameter
- Authorize the AOR's region user ID with READ access to the required files in the file resource profiles named on the XFCT system initialization parameter.

For example, define the APPL profile for an FOR with APPLID CICSHF01, and the PERMIT command to enable the AORs with user IDs CICSAOR1 and CICSAOR2 to sign on to CICSHF01, as follows:

```
RDEFINE APPL CICSHF01 UACC(NONE) NOTIFY(sys_admin_userid)
PERMIT CICSHF01 CLASS(APPL) ID(CICSAOR1 CICSAOR2) ACCESS(READ)
```

Cases when SAF neither grants nor refuses the request are resolved in the same way as for server LOGON (see [“If SAF neither grants nor refuses an access request” on page 526](#)).

If the user ID is allowed to sign on to the FOR's application, the CONNECT request succeeds unless the AOR's user ID is not allowed to read the specified file. Otherwise, the CONNECT request is treated in the same way as when the AOR's user ID is undefined.

When file security is in force in an FOR, and the user ID of the AOR is *undefined*, a CONNECT request fails unless the FOR's default user ID (specified by the DFLTUSER system initialization parameter) is allowed to read the specified file.

Function shipping detects that an AOR's access to a file has been revoked when a rebuild of the file control resource class is completed in the FOR. However, if a valid connection exists, SDT continues to allow access until something causes the connection to be broken. See [Caching of RACF classes and their profiles](#).



**CAUTION:** If you use ISC instead of MRO for function shipping, ensure that the value of the SECURITYNAME parameter in the FOR is the same as the user ID of the AOR. If you use IPIC instead of MRO for function shipping, ensure that the value of the IPCONN SECURITYNAME parameter or, if applicable, the AOR's certificate's user ID in the FOR is the same as the user ID of the AOR. Otherwise, the SDT CONNECT, and function shipping security checks are inconsistent.

## Security for coupling facility data tables

You can configure security for coupling facility data tables (CFDT) by authorizing the server to a coupling facility list structure, authorizing the server to act as a server for its specified pool, authorizing a CICS region to a CFDT pool and even a specific CFDT, or configuring file resource security checking.

CICS and MVS use RACF facilities to provide security for coupling facility data tables in the following areas:

1. [“Authorizing server access to a coupling facility list structure” on page 528](#)
2. [“Authorizing the server” on page 528](#)
3. [“Authorizing a CICS region to a CFDT pool” on page 529](#)
4. Optional: [“Authorizing a CICS region to a CFDT” on page 529](#)
5. Optional: [“File resource security checking” on page 529](#)

With the exception of items 4 and 5, which are optional, the other security checks are made automatically and are never bypassed. For items 2 and 3 in this list, in cases when the system authorization facility (SAF) neither grants nor refuses, access is resolved in the same way as the LOGON security check for CICS shared data table support. See [“SDT server authorization security check” on page 525](#) for details.

### Authorizing server access to a coupling facility list structure

Each coupling facility data table (CFDT) server requires access to the coupling facility list structure that contains its pool of coupling facility data tables.

To permit access, the server region user ID requires ALTER access to a FACILITY class general resource profile called IXLSTR.*structure\_name*.

- The server region user ID is the user ID under which the job or started task is running.
- Structure names for coupling facility data tables take the form DFHCFLS\_*poolname*.

For example, if coupling facility data tables are defined in a pool called PRODCFT1, the list structure for this pool is named DFHCFLS\_PRODCFT1 in the CFRM policy. To access this list structure, the server user ID for pool PRODCFT1 requires ALTER access to the IXLSTR profile, defined as follows:

```
RDEFINE FACILITY IXLSTR.DFHCFLS_PRODCFT1 UACC(NONE)
PERMIT IXLSTR.DFHCFLS_PRODCFT1 CLASS(FACILITY) ID(server_userid) ACCESS(ALTER)
```

### Authorizing the server

When a coupling facility data table (CFDT) server starts up for a given coupling facility data table pool, CICS authorized cross-memory (AXM) services call RACF to establish that the server is authorized to act as a server for that pool.

To authorize a CFDT server to act as a server for its specified pool, give the server region user ID CONTROL access to a FACILITY class general resource profile called DFHCF.*poolname*.

For example, if the pool is PRODCFT1, define the profile and the required PERMIT statement as follows:

```
RDEFINE FACILITY DFHCF.PRODCFT1 UACC(NONE)
PERMIT DFHCF.PRODCFT1 CLASS(FACILITY) ID(server_userid) ACCESS(CONTROL)
```

## Authorizing a CICS region to a CFDT pool

Each CICS region requires authorization to connect to a coupling facility data table (CFDT) pool. To authorize a CICS region to connect to a server and its pool, give the CICS region UPDATE access to the server's FACILITY class profile for the pool.

For example, if the pool is PRODCFT1, define the required PERMIT statement as follows:

```
PERMIT DFHCF.PRODCFT1 CLASS(FACILITY) ID(CICS_region_userid) ACCESS(UPDATE)
```

## Authorizing a CICS region to a CFDT

In addition to controlling the access of a CICS region to a coupling facility data table (CFDT) pool, you can optionally control access to each CFDT in the pool.

This security check, if active, is performed by the server each time a CICS region connects to a coupling facility data table for the first time. For more information about CFDT server security parameters, see [Coupling facility data table server security parameters](#).

The resource security check is done as if for a CICS file owned by the coupling facility data table server region, using a profile defined in the general resource class specified on the SECURITYCLASS server initialization parameter. The default for this is the FCICSFCT class. For the profile name, use the table name as defined in the file resource definition.

You can optionally prefix the profile name using the server region user ID as the prefix by specifying SECURITYPREFIX=YES as a server initialization parameter. You can customize the prefix for this security check using the server initialization parameter SECURITYPREFIXID.

The following example shows the security parameters for a pool called PRODCFT1:

```
POOLNAME=PRODCFT1  
SECURITY=YES  
SECURITYCLASS=FCICSFCT  
SECURITYPREFIX=NO
```

For example, if the pool is PRODCFT1, define the profile and the required PERMIT statement as follows:

```
RDEFINE FCICSFCT PRODCFT1 UACC(NONE)  
PERMIT PRODCFT1 CLASS(FCICSFCT) ID(region_userid) ACCESS(UPDATE)
```

The coupling facility data table server performs the table security check by issuing a cross-memory mode FASTAUTH check, which requires the use of global in-storage security profiles. Access fails if a return code other than zero is received by the server in response to the FASTAUTH check. If the external security manager does not support cross-memory mode FASTAUTH or global in-storage profiles, coupling facility data table security checks are not possible and an error message is issued at server initialization time if table security checking is specified. For information about all server initialization parameters that can be specified, see [Coupling facility data table server parameters](#).

## File resource security checking

Normal CICS resource security for files is supported for coupling facility data tables. CICS performs the usual file resource security checks against signed-on users of transactions that access coupling facility data tables, using profiles defined in the general resource class named on the XFCT system initialization parameter.



---

## Chapter 29. Security for CICS resources

This section covers how you use RACF to secure CICS system resources.

For information about how RACF works with CICS, see [How it works: Securing CICS with RACF](#).

---

### CICS system resource security

This topic describes how to protect the system resources that CICS requires.

#### CICS installation requirements for RACF

You can control access to the resources used by your CICS region (or regions) by using RACF facilities. The CICS libraries supplied on the distribution volume include the CICS modules you need to support external security management.

##### CICS-supplied RACF dynamic parse validation routines

To define CICS terminal operator data, use the CICS-supplied RACF dynamic parse validation routines. Install these routines in `SYS1.CICSTSnn.CICS.SDFHLINK`, where `CICSTSnn` is your version of CICS. For example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is `SYS1.CICSTS62.CICS.SDFHLINK`. Make this as an APF-authorized library in your z/OS linklist. For more information, see [Installing CICS-required modules in the MVS linklist](#).

The routines are as follows:

**DFHSNNFY**

CICS segment update notification

**DFHSNPTO**

CICS segment TIMEOUT print formatting

**DFHSNVCL**

CICS segment OPCLASS keyword validation

**DFHSNVID**

CICS segment OPIDENT keyword validation

**DFHSNVPR**

CICS segment OPPRTY keyword validation

**DFHSNVTO**

CICS segment TIMEOUT keyword validation

#### Using RACF support in a multi-z/OS environment

If you are operating a multi-z/OS environment with shared DASD, you are likely to want the active and alternate CICS systems to have access to the same terminal user characteristics.

You can enable this by having the active and alternate CICS systems share the same RACF database.

#### Setting options on the z/OS program properties table

If you have an entry for the DFHSIP program in your z/OS program properties table (PPT), ensure that the NOPASS option is not set for DFHSIP in the PPT statement of the SCHEDxx member of the SYS1.PARMLIB library. Setting the NOPASS option would bypass password and RACF authorization checking on data sets accessed by the CICS region.

For more information about specifying CICS z/OS PPT options, see [Installing CICS-required modules in the MVS linklist](#).

## Protecting CICS load libraries

Although, in general, CICS runs in unauthorized state, the CICS initialization program, DFHSIP, must run in authorized state for part of its execution. For this reason, the version of the DFHSIP module supplied on the distribution tape is link-edited with the “authorized” attribute (using the linkage-editor SETCODE AC(1) control statement), and is installed in SYS1.CICSTSnn.CICS.SDFHAUTH library, where CICSTSnn is your version of CICS. For example, for CICS Transaction Server for z/OS, Version 6 Release 2, this is SYS1.CICSTS62.CICS.SDFHAUTH. This library must be defined to the operating system as APF-authorized.

To prevent unauthorized or accidental modification of SDFHAUTH, make this library RACF-protected. Without such protection, the integrity and security of your z/OS system are at risk. To control the unauthorized startup of a CICS system using DFHSIP, also consider implementing the following:

- If DFHSIP is in a library that has been placed in the z/OS link list, protect DFHSIP with a profile in the PROGRAM resource class. Give READ access to this profile only to those users who are allowed to execute CICS.
- If DFHSIP has been placed in the link pack area (LPA), it cannot be protected by the PROGRAM resource class. Instead, control the start up of CICS by controlling the loading of any suffixed DFHSIT load module. Ensure that no DFHSIT load module is included in the LPA, then control the loading of DFHSIT by creating a generic 'DFHSIT\*' profile in the PROGRAM resource class. Give READ access to this profile only to those users who are allowed to execute CICS.

Also give RACF protection to the following libraries:

- SYS1.CICSTSnn.CICS.SDFHLINK, where CICSTSnn is your version of CICS.
- SYS1.CICSTSnn.CICS.SDFHLPA, where CICSTSnn is your version of CICS.
- The other libraries, including SYS1.CICSTSnn.CICS.SDFHLOAD, that make up the STEPLIB and DFHRPL library concatenations.

See “[Authorizing access to CICS data sets](#)” on page 538 for more information about protecting CICS data sets and creating suitable data set security profiles.

**Note:** The source statements of your application programs are sensitive; consider having RACF protect the data sets containing them.

## Specifying the CICS region user ID

When you start a CICS region (either as a job or as a started task) in an MVS environment that has RACF installed, the job or task is associated with a user ID, referred to as the *CICS region user ID*.

The authority that is associated with this user ID determines which RACF-protected resources the CICS region can access.

Each CICS region, for either production or test use, should be subject to normal RACF data set protection based on the region user ID under which the CICS region runs. You specify the region user ID under which CICS runs in one of three ways:

As a started task:

- In the RACF started procedures table, ICHRIN03, when you start CICS as a started task by using the MVS START command. (See [Authorizing CICS procedures to run under RACF](#).) However, do not assign the “trusted” or “privileged” attributes to CICS entries in the started procedures table. For more information, see the description of associating MVS started procedures with user IDs in the [z/OS Security Server RACF System Programmer's Guide](#).

As a started job:

- In a STARTED general resource class profile, on the user parameter of the STDATA segment.

As a job:

- On the USER parameter of the JOB statement, when you start CICS as a JOB.



**To ensure that the authorizations for different CICS regions are properly differentiated, run each with a unique region user ID.** For example, the user ID under which you run the production CICS regions to process payroll and personnel applications should be the only CICS user ID authorized to access production payroll and personnel data sets.

If you are using intercommunication, it is particularly important to use unique user IDs, unless you want to bypass link security checking. For more information, see [Implementing LU6.2 security](#), [Implementing LU6.1 security](#), or [MRO link security](#), depending on the environment you are using.

## Authorizing CICS to start up under RACF

You can invoke your CICS startup procedure to start CICS as a started task or as a started job. RACF provides the ICHRIN03 procedure table for started tasks, and the STARTED general resource class for started jobs. Both options are discussed here.

### Using the ICHRIN03 table for started tasks

If you run CICS as a started task, you should associate the cataloged procedure name with a suitably authorized RACF user through the RACF table, ICHRIN03.

RACF supplies a default ICHRIN03 table, which you can modify. See the [z/OS Security Server RACF System Programmer's Guide](#) for more information about this table, and how you can add the default entry for the cataloged procedure name for starting CICS.

If your ICHRIN03 table contains the default entry, you need not update the table; but define a RACF user with the same name as the cataloged procedure.

If your ICHRIN03 table does not contain the default entry (or you choose not to set the default entry), update the table with an entry that contains the cataloged procedure name and its associated RACF user. This RACF user need not have the same name as the cataloged procedure.

Whether your ICHRIN03 table contains the default entry or a specific entry you have defined, ensure that the RACF user identified through ICHRIN03 has the necessary access authority to the data sets in the cataloged procedure.

For example, if you associate a cataloged procedure called DFHCICS with the RACF userid CICSR, the userid CICSR needs to have access to the CICS resources accessed by the task started by DFHCICS.

### Using STARTED profiles for started jobs

If you start your CICS regions as started jobs, you can use separate userids for each started region, even though they are all started from the same procedure. Alternatively, you can use generic profiles for groups of CICS regions that are to share the same userid—for example, for all regions of the same type, such as terminal-owning regions.

The support for started jobs is provided by the RACF STARTED general resource class, and its associated STDATA segment. You define profiles in this class for each job, or group of jobs, that needs to run under a unique userid.

Ensure that the userids specified in STDATA segments are defined to RACF. Also ensure that the userids are properly authorized to the data set profiles of the CICS regions that run under them.

For example STARTED profiles, see [Example STARTED profiles for started jobs](#).

### Example STARTED profiles for started jobs

This topic includes examples of defining STARTED profiles in RACF for CICS started jobs.

### Example of a generic profile for multiple AORs

The following example shows how to define a generic profile for jobs that are to be started using a procedure called CICSTASK.

In this example the job names begin with the letters CICSDA for a group of CICS application-owning regions (AORs):

```
RDEFINE STARTED (CICSTASK.CICSDA*) STDATA(USER(CICSDA##))
```

When you issue the START command to start CICSTASK with a job name of, say, CICSDAA1, MVS passes the procedure name (CICSTASK), and the job name (CICSDAA1) in order to obtain the userid under which this CICS application-owning region is to run. In this example the CICS region userid is defined as CICSDA##, for all regions started under the generic profile CICSTASK.CICSDA\*.

## Example of a unique profile for each region

The following example shows how to define a unique profile for jobs that are to be started using a procedure called CICSTASK, and where each started job is to run under a unique CICS region userid:

```
RDEFINE STARTED (CICSTASK.CICSDAA2) STDATA(USER(CICSDAA2))
```

When you issue the START command to start CICSTASK with the job name CICSDAA2, MVS passes the procedure name (CICSTASK) and the job name (CICSDAA2) to obtain the userid under which this CICS application-owning region is to run. In this example the CICS region userid is defined as CICSDAA2, the same as the APPLID.

## Authorization failures in application programs

CICS returns a RESP2 value of 100 for command security failures, a value of 101 for resource security failures, and a value of 102 for surrogate security failures.

If you are running with CICS command security, CICS returns the NOTAUTH condition, RESP value 70, to your application, which is the same condition as for a resource security failure. CICS also issues message DFHXS1111 to the CICS security transient data destination CSCS. To test for this value in your application, it is recommended that you code DFHRESP(NOTAUTH) rather than explicitly coding a value.

To distinguish between a command security failure, a resource security failure, and a surrogate security failure, check the RESP2 value:

- For a command security failure, CICS returns a value of 100 in RESP2.
- For a resource security failure, a value of 101 is returned in RESP2.
- For a surrogate security failure, a value of 102 is returned in RESP2

## Defining user profiles for CICS region user IDs

Before bringing up a CICS region, ensure that the required user IDs are defined - the CICS region user ID and the CICS default user ID.

If you are suitably authorized, you can define a RACF user profile for a CICS region by using the **ADDUSER** command. For example, to define CICSRL as a user ID for a CICS region, enter the following RACF command from TSO:

```
ADDUSER CICSRL NAME(user-name) DFLTGRP(cics_region_group)
```

**Do not assign the OPERATIONS attribute to CICS region user IDs.** Doing so would allow the CICS region to access RACF-protected data sets for which no specific authorization has been performed. CICS region user IDs do not need the **OPERATIONS** attribute if the appropriate **CONNECT** or **PERMIT** commands have been issued. These commands authorize the CICS region user ID for each CICS region to access only the specific data sets required by that region.

## Coding the USER parameter on the CICS JOB statement

If you start CICS from a job, include the parameters USER= and PASSWORD= on the JOB statement.

For example:

```
//CICSA JOB ... ,USER=CICSR,PASSWORD=password
```

When you define a new user to RACF, the password is automatically flagged as expired. For this reason, the first time you start CICS under a new userid, change the PASSWORD parameter on the JOB statement. For example:

```
PASSWORD=(oldpassword,newpassword)
```

If you want to avoid specifying the password on the JOB statement, you can allow a surrogate user to submit the CICS job. A surrogate user is a RACF-defined user who is authorized to submit jobs on behalf of another user (the original user), without having to specify the original user's password. Jobs submitted by a surrogate user execute with the identity of the original user. See [“Surrogate job submission in a CICS environment” on page 547](#) for more information. The region userid must also have surrogate authority to use the default user; see [Surrogate security](#).

## Authorities required for CICS region user IDs

The CICS control program runs under the CICS region user ID.

Therefore, this user ID needs access to all the resources that CICS itself must use. There are two types of these resources:

1. [Resources external to CICS](#), such as data sets, the spool system, and the SNA network.
2. [Resources internal to CICS](#), such as system transactions and auxiliary user IDs.

## Authorizing external resources

Like a batch job, each CICS region must be able to access many external resources. The authority for CICS to access these resources is obtained from the CICS region user ID. It doesn't matter which signed-on user requests CICS to complete the actions that access these resources. The external services are aware only that CICS is requesting them, under the region user ID's authority.

Give access to these resources:

### The z/OS System Logger

CICS needs authority to use log streams defined in the z/OS logger. See [“Authorizing access to z/OS log streams” on page 538](#).

### External data sets used by CICS

CICS needs authority to open all the data sets that it uses. See [“Authorizing access to CICS data sets” on page 538](#).

### External data sets used by application programs

CICS needs authority to open all the data sets that your own application programs need. See [“Authorizing access to user data sets” on page 541](#).

### Temporary storage servers

CICS needs authority to access temporary storage servers if any TS queues are defined as shared. See [“Authorizing access to temporary storage servers” on page 542](#).

### SMSVSAM servers

CICS needs authority to access the SMSVSAM server if you are using VSAM record-level sharing (RLS). See [“Authorizing access to SMSVSAM servers” on page 545](#).

### z/OS Communications Server (for SNA or IP) applications

Consider carefully for each program whether you allow it to become a z/OS Communications Server application. If you do this, CICS needs authority to open its z/OS Communications Server ACB. See [“Controlling the opening of a CICS region's z/OS Communications Server ACB” on page 546](#).

## Jobs that are submitted to the internal reader

If any of your applications submit JCL to the JES internal reader, you can allow them to be submitted without the USERID parameter. See [“Controlling userid propagation”](#) on page 547.

However, you should not usually require your applications to provide a PASSWORD parameter on submitted jobs. So you **should** allow CICS to be a surrogate user of all the possible user IDs that might be submitted. See [“Surrogate job submission in a CICS environment”](#) on page 547.

## System spool data sets

CICS needs authority to access data sets in the JES spool system. See [“JES spool protection in a CICS environment”](#) on page 548.

## Authorizing internal resources

Several conditions exist in which CICS behaves like an application program, but is running housekeeping functions that are not directly for any user. The associated transactions run under control of the CICS region user ID, and because they access CICS internal resources, you must give the CICS region user ID authority to access them.

These conditions are:

### CICS system transactions

CICS needs authority to attach all the internal housekeeping transactions that it uses. See [Transactions in CICS](#).

### Auxiliary user IDs

If CICS surrogate user checking is specified with the **XUSER** system initialization parameter (the default), CICS needs authority to use certain extra user IDs. See [Surrogate security](#). These IDs are:

- The default user ID
- The user ID used for post-initialization processing (PLTPIUSR)
- The user ID that is used for transient data trigger transactions.

### Resources used by PLTPI programs

If the **PLTPIUSR** system initialization parameter is omitted, the CICS region user ID is used for all PLTPI programs. In this case, give the CICS region user ID access to all the CICS resources that these programs use. See [Surrogate security](#).

## Defining the default CICS user ID to RACF

For each CICS region where you specify **SEC=YES**, define a RACF user profile whose user ID matches the value of the **DFLTUSER** system initialization parameter.

You can use the same default user ID on all CICS regions. You can specify this default user ID on the **DFLTUSER** system initialization parameter, or leave **DFLTUSER** set to the default of CICSUSER.

Define a different default CICS user ID for each CICS region if any of the following considerations applies:

- The default CICS user ID requires different security attributes (such as membership in RACF groups).
- The default CICS user ID requires different operator data (CICS segment of the RACF user profile).
- The default CICS user ID requires a different default language (LANGUAGE segment of the RACF user profile).

### Step 1. Define the CICS default user to RACF

Use the **ADDUSER** command with the CICS operand to define a CICS default user to RACF.

Usually, the default CICS user ID should be defined as a protected user ID. This is particularly the case if the CICS region is a started task. Protected user IDs cannot be used to enter the system by any

means that requires a password, and users cannot cause a protected user ID to be revoked. For more information, see [How it works: Identification in CICS](#).

**Example:**

The following command defines CICS default user CICSUSER as a protected user ID to RACF:

```
ADDUSER CICSUSER DFLTGRP(group_id) NAME(user_name)
        OWNER(userid or group)
        NOIDCARD
        NOPASSWORD
```

## Step 2. Authorize the CICS region user ID to be a surrogate user of the default user ID

If you have specified the system initialization parameter **XUSER=YES** (the default), authorize the CICS region user ID to be a surrogate user of the default user ID. For example, the following command authorizes a CICS region user ID to be a surrogate user of CICSUSER:

```
PERMIT CICSUSER.DFHINSTL CLASS(SURROGAT) ID(cics_region_userid)
```

## Sign-on processing of the CICS default user

During startup, CICS signs on the default user ID. If the default user sign-on fails (because, for example, the user ID is not defined to RACF), CICS issues message DFHXS1104 and terminates CICS initialization.

When CICS successfully signs on a valid RACF user ID as the default user, it establishes the terminal user data for the default user from one of the following sources:

- The CICS segment of the default user's RACF user profile
- Built-in CICS system default values

See [“Obtaining CICS-related data for a user” on page 95](#) for details of the sign-on process for obtaining CICS terminal operator data.

## How CICS assigns the security attributes of the default user

CICS assigns the security attributes of the default user ID to all CICS terminals before any terminal user begins to sign on. The security attributes and terminal user data of the default user also apply to any terminals at which users do not sign on (using either the CICS-supplied CESN transaction or a user-written equivalent), unless the security has been explicitly preset by specifying a value for the USERID option in the terminal definition.

CICS also assigns the security attributes of the default user ID to any trigger-level transactions that are initiated for transient data queues without a USERID parameter.

Ensure the default user ID gives at least the minimum authorities that ought to be granted to any other terminal user. In particular:

- Give the default user access to the region's APPLID. See [“Authorizing access to the CICS region” on page 545](#).
- Give the default user access to the CICS-supplied transactions that are intended to be used by everybody. See the definitions in [The CICS segment in user profiles](#), especially those transactions that are recommended for inclusion in the ALLUSER example group of transactions.

## Authorizing access to z/OS log streams

Ensure that you authorize the CICS region userid to write to (and create if necessary) the log streams that are used for its system log and general logs.

You do this by granting the appropriate access authorization to log stream profiles in the LOGSTRM general resource class.

The level of authorization required depends on whether log streams are always explicitly defined to the z/OS System Logger:

- If CICS is expected to create log streams dynamically, give CICS ALTER authority to the relevant log stream profiles, and UPDATE authority to the relevant coupling facility structures.
- If all the log streams to which CICS writes are already defined to z/OS, give CICS only UPDATE authority to the log stream profiles.
- Permit READ access to those users who need to read the CICS log streams.

For example, the generic profile in the following example could be defined to cover all the log streams referenced by the CICS region and identified by its region userid and applid:

```
RDEFINE LOGSTRM region_userid.** UACC(NONE)
```

If, however, you have multiple CICS systems sharing the same region userid, but with differing security requirements, include the applid in the generic profile, as follows:

```
RDEFINE LOGSTRM region_userid.applid.* UACC(NONE)
```

The following example allows the CICS region userid under which CICS is running to write journal and log records to log streams in the named coupling facility structure:

```
PERMIT IXLSTR.structurename CLASS(FACILITY) ACCESS(UPDATE)  
ID(region_userid)
```

The following examples give access to three categories of user:

```
PERMIT region_userid.applid.* CLASS(LOGSTRM) ACCESS(ALTER)  
ID(region_userid)  
PERMIT region_userid.applid.* CLASS(LOGSTRM) ACCESS(READ)  
ID(authorized_browsers)  
PERMIT region_userid.applid.* CLASS(LOGSTRM) ACCESS(UPDATE)  
ID(archive_userid)
```

In these examples, `region_userid` is the CICS region userid under which CICS is running, either as a started task or batch job. The identifier `archive_userid` is the userid under which an application program runs to purge old data from CICS logs when the data is no longer needed. The identifier `authorized_browsers` refers to the userids of users allowed to read log streams, but not purge data.

If several CICS regions share the same CICS region userid, you can make profiles more generic by specifying `*` for the *applid* qualifier.

The number of profiles you define depends on the naming conventions of the logs, and to what extent you can use generic profiling.

## Authorizing access to CICS data sets

When you have defined a region userid for your CICS job (or started task), permit that user id to access the CICS system data sets with the necessary authorization.

When authorizing access to CICS system data sets, choose appropriately from the following levels of access: READ, UPDATE, and CONTROL. Also define data set profiles with UACC(NONE) to ensure that only CICS region user ids can access those data sets. For information about the CICS region user id, see [“Specifying the CICS region user ID” on page 532](#).

For CICS load libraries, only permit READ access.

The following four data sets require CONTROL access.

- The temporary storage data set
- The transient data intrapartition data set
- The CAVM control data set (XRF)
- The CAVM message data set (XRF)

Permit UPDATE access for all the remaining CICS data sets.

Therefore, for CICS system data sets you need at least three generic profiles to restrict access to the appropriate level. See [Table 47 on page 539](#).

Table 47. Summary of generic data set profiles	
Required access level	Type of CICS data sets protected
READ	Load libraries
UPDATE	Auxiliary trace; transaction dump; system definition; global catalog; local catalog; and restart
CONTROL	Temporary storage; intrapartition transient data; XRF message; and XRF control

If you use generic naming of the data set profiles, you can considerably reduce the number of profiles you need for your CICS regions. This policy is illustrated in the examples shown in [Figure 168 on page 539](#) for a number of sample CICS regions.

You can issue the RACF commands shown in the examples from a TSO session, or execute the commands using the TSO terminal monitor program, IKJEFT01, in a batch job as illustrated in [Figure 168 on page 539](#). Alternatively, you can use the RACF-supplied ISPF panels. Any of these methods enables you to create the necessary profiles and authorize each CICS region userid to access the data sets as appropriate for the corresponding CICS region.

[Figure 168 on page 539](#) is an example using CICS TS 6.2. Your installation has its own naming conventions.

```
//RACFDEF JOB 'accounting information',
//          CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
//DEFINE EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=A
//SYSTSPRT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSTSIN DD *
ADDSD 'CICSTS62.CICS.SDFHLOAD' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.SDFHLOAD' ID(cics_id1,...,cics_group1,...,cics_groupn)
ACCESS(READ)
ADDSD 'CICSTS62.CICS.SDFHAUTH' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.SDFHAUTH' ID(cics_id1,...,cics_group1,...,cics_groupn)
ACCESS(READ)
ADDSD 'CICSTS62.CICS.applid.**' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.applid.**' ID(applid_userid) ACCESS(UPDATE)
ADDSD 'CICSTS62.CICS.applid.DFHXR*' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.applid.DFHXR*' ID(applid_userid) ACCESS(CONTROL)
ADDSD 'CICSTS62.CICS.applid.DFHINTRA' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.applid.DFHINTRA' ID(applid_userid) ACCESS(CONTROL)
ADDSD 'CICSTS62.CICS.applid.DFHTEMP' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.applid.DFHTEMP' ID(applid_userid) ACCESS(CONTROL)
ADDSD 'CICSTS62.CICS.DFHCSO' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS62.CICS.DFHCSO' ID(cics_group1,...,cics_groupn) ACCESS(UPDATE)
/*
//
```

Figure 168. Example of a job to authorize access to CICS data sets, using CICS TS 6.2.

**Note:** Data sets that need to be accessed in the same way by all CICS regions (for example, with READ or UPDATE access) should be protected by profiles that do **not** include an APPLID. For example, define the partitioned data sets that contain the CICS load modules with profiles that give all CICS region groups (or userids) READ access.

You could also consider protecting all these data sets with one generic profile. Continuing with the example of using CICS TS 6.2, this generic profile could be 'CICSTS62.CICS.\*\*'. However, you must strictly control who has read access to the SDFHAUTH data set, because it contains APF-authorized programs, and the profile protecting this data set **must** be defined with UACC(NONE). In [Figure 168 on page 539](#) all of the partitioned data sets are defined with UACC(NONE) and have an explicit access list.

Although CICS modules exist in libraries SYS1.CICSTSnn.SDFHLP and SYS1.CICSTSnn.SDFHLINK, no CICS region userid requires access to these libraries.

By establishing a naming convention for the data sets belonging to each region, and one generic profile for each CICS region, with the CICS z/OS Communications Server APPLID as one of the data set qualifiers, you can ensure that only one CICS region has access to the data sets.

CICS needs UPDATE access to all the data sets covered by these profiles. The CICS DDNAMEs for the data sets in this category are as follows:

**DFHGCDD**

Global catalog data set

**DFHLCD**

Local catalog data set

**DFHAUXT**

Auxiliary trace data set, A extent

**DFHBUXT**

Auxiliary trace data set, B extent

**DFHDMPA**

Transaction dump data set, A extent

**DFHDMPB**

Transaction dump data set, B extent

**Note:** The auxiliary trace data set, the transaction dump data set, and the z/OS dump data set may contain sensitive information. Protect them from unauthorized access.

CICS needs CONTROL access for the transient data intrapartition, temporary storage, and CICS availability manager (CAVM) data sets.

The CICS DDNAMEs for the data sets in this category are as follows:

**DFHINTRA**

Transient data intrapartition data set

**DFHTEMP**

Temporary storage data set

**DFHXMSG**

XRF message data set

The CICS system definition data set (CSD) is protected by a discrete profile to which all CICS groups have access. This assumes that all the CICS regions are sharing a common CSD. If your CICS regions do not share a common CSD and each region has its own CSD, or if groups of regions share a CSD, define discrete or generic data set profiles as appropriate.

You must grant the CICS region user ID read access to the VSAM catalog for the DFHCSD file for the CICS system definition data set (CSD).



## Authorizing access with the MVS library lookaside (LLA) facility

If you are using the library lookaside (LLA) facility of MVS, you can control a program's ability to use the LLACOPY macro.

Authorize the CICS region's userid in *one* of the following ways:

- It must have UPDATE authority to the data set that contains the LLA module.
- It must have UPDATE authority in the FACILITY class to the resource CSVLLA. *datasetname*, where *datasetname* is the name of the library that contains the LLA module. For example:

```
RDEFINE FACILITY CSVLLA.datasetname UACC(NONE) NOTIFY  
PERMIT CSVLLA.datasetname CLASS(FACILITY) ID(...) ACCESS(UPDATE)
```

### Related reference

[LLACOPY system initialization parameter](#)

## Authorizing access to user data sets

When you have defined the RACF user ids for your CICS regions and given them access to the CICS system data sets, permit the user IDs to access the CICS **application** data sets with the necessary authority.

You must grant the CICS region user ID read access to each VSAM catalog for files for which CICS has file definitions installed and are to be either opened during CICS startup or at any time after.

The following RACF commands for CICS TS 6.2 permit the userid specified on the ID parameter to access some CICS user application data sets, with READ authority for the first two data sets, and UPDATE authority for the last two:

```
PERMIT 'CICSTS62.CICS.appl1.dataset1' ID(user or group) ACCESS(READ)  
PERMIT 'CICSTS62.CICS.appl1.dataset2' ID(user or group) ACCESS(READ)  
PERMIT 'CICSTS62.CICS.appl2.dataset3' ID(user or group) ACCESS(UPDATE)  
PERMIT 'CICSTS62.CICS.appl2.dataset4' ID(user or group) ACCESS(UPDATE)
```

For another version of CICS, substitute CICSTS62.CICS with your version.

## ACCESS(CONTROL) for VSAM entry-sequenced data sets (ESDS)

CICS file control uses control interval processing when opening a VSAM ESDS (non-RLS mode only). This means that you must specify ACCESS(CONTROL) for all such data sets, otherwise the OPEN command fails with message DFHFC0966.

## ACCESS(ALTER) for VSAM data sets when using BWO

In order to use backup while open (BWO) to back up VSAM data sets that are currently in use and are defined as BACKUPTYPE(DYNAMIC), or BWO(TYPECICS) in the integrated catalog facility (ICF) catalog, give the CICS region userid RACF ALTER authority to the data set or to the ICF catalog in which that data set is defined. If you do not, the OPEN command fails with message DFHFC5803. For guidance on using BWO, see [Backup-while-open \(BWO\)](#).

## ACCESS(ALTER) for VSAM data sets when specifying SMS Data Class attribute Dynamic Volume Count

Dynamic Volume Count

You can use SMS Data Class attribute Dynamic Volume Count to extend your VSAM data to multiple volumes. Depending on your SMS release level, ACCESS(ALTER) may be required by the CICS region userid to update the ICF catalog volume list during EOVS extend processing. To determine the correct

access level for your DFSMS release, see [Required RACF Authorization Tables in z/OS DFSMS Access Method Services Commands](#).

## Authorizing access to the temporary storage pools

You can control access by temporary storage (TS) servers to the TS pools in the coupling facility.

Each TS server can be started as a job or started task. The name of the TS queue pool for a TS server is specified at server startup. For each TS pool there can be only one TS server running on each MVS image in the sysplex.

Two security checks are made against the TS server's userid—that is, the userid under which the job or started task is running. To ensure the server passes these checks, do the following:

- Authorize the TS server region to connect to the coupling facility list structure used for its own TS pool. This requires that the TS server userid has ALTER authority to a coupling facility resource management (CFRM) RACF profile called IXLSTR.*structure\_name* in the FACILITY general resource class.

For example, if the userid of the server is DFHXQTS1, and the list structure is called DFHXQLS\_TSPRODQS, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY IXLSTR.DFHXQLS_TSPRODQS UACC(NONE)
PERMIT IXLSTR.DFHXQLS_TSPRODQS CLASS(FACILITY) ID(DFHXQTS1) ACCESS(ALTER)
```

To reduce security administration, use the same TS server userid to start each TS server that supports the same TS pool.

- Give the TS server's userid CONTROL access to the CICS RACF profile called DFHXQ.*poolname* in the FACILITY general resource class. This authorizes the TS server to act as a server for the named TS pool.

For example, if the userid of the server is DFHXQTS1, and the pool name is TSPRODQS, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY DFHXQ.TSPRODQS UACC(NONE)
PERMIT DFHXQ.TSPRODQS CLASS(FACILITY) ID(DFHXQTS1) ACCESS(CONTROL)
```

See [“System authorization facility \(SAF\) responses to the TS server”](#) on page 543 for information about the responses to the TS server.

## Authorizing access to temporary storage servers

You can control access by CICS regions to the TS servers.

A security check is made against the CICS region userid to verify that the region is authorized to use the services of a TS server. This check is made each time that a CICS region connects to a TS server.

Give each CICS region that connects to a TS server userid UPDATE access to the CICS RACF profile called DFHXQ.*poolname* in the FACILITY general resource class. This authorizes the CICS region to use the services of the TS server for the named TS pool.

For example, if the userid of a CICS region is CICSDA1, and the pool name is TSPRODQS, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY DFHXQ.TSPRODQS UACC(NONE)
PERMIT DFHXQ.TSPRODQS CLASS(FACILITY) ID(CICSDA1) ACCESS(UPDATE)
```

When a CICS region has connected to a TS pool, it can write, read, and delete TS queues without any further security checks being performed by the server. However, the CICS application-owning regions issuing TS API requests can use the existing mechanisms for TS resource security checking

## System authorization facility (SAF) responses to the TS server

If the security profile for a TS pool cannot be retrieved, SAF neither grants nor refuses the access request. In this situation:

Access to the TS pool, either by a CICS region or by the TS server itself, is rejected if:

- A security manager is installed, but is either temporarily inactive or inoperative for the duration of the MVS image. This is a fail-safe action, on the grounds that, if the security manager is active, it might retrieve a profile that does not permit access to the TS pool.

Access to the TS pool, either by a CICS region or by the TS server itself, is accepted if:

- There is no security manager installed, or
- There is an active security manager, but the FACILITY class is inactive, or there is no profile in the FACILITY class. The access request is allowed in this case because there is no evidence that you want to control access to the TS server.

Access is permitted to any TS server without a specific DFHXQ.*poolname* profile, or an applicable generic profile. No messages are issued to indicate this. To avoid any potential security exposures, you can use generic profiles to protect all, or specific groups of, TS servers. For example, specifying:

```
RDEFINE FACILITY (DFHXQ.*) UACC(NONE)
```

ensures that access is allowed only to TS servers with a more specific profile to which a TS server or CICS region is authorized.

## Authorizing access to named counter pools and servers

You can control access by:

- Coupling facility data table (named counter) servers to named counter pools. See [“Access to named counter pools”](#) on page 543.
- CICS regions to the named counter servers. See [“Access to named counter servers”](#) on page 544.

### Access to named counter pools

You can control access by named counter servers to the named counter pools in the coupling facility.

Each named counter server can be started as a job or started task. The name of the named counter pool for a named counter server is specified at server startup. For each named counter pool there can be only one server running on each MVS image in the sysplex.

Two security checks are made against the named counter server's userid—that is, the userid under which the job or started task is running. To ensure the server passes these checks, do the following:

- Authorize the named counter server region to connect to the coupling facility list structure used for its own named counter pool. This requires that the named counter server userid has ALTER authority to a coupling facility resource management (CFRM) RACF profile called IXLSTR.*structure\_name* in the FACILITY general resource class.

For example, if the user ID of the server is DFHNCSV1, and the list structure is called DFHNCLS\_DFHN001, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY IXLSTR.DFHNCCLS_DFHN001 UACC(NONE)
PERMIT IXLSTR.DFHNCCLS_DFHN001 CLASS(FACILITY) ID(DFHNCSV1) ACCESS(ALTER)
```

To reduce security administration, use the same named counter server userid to start each named counter server that supports the same named counter pool.

- Give the named counter server's userid CONTROL access to the CICS RACF profile called DFHNC.*poolname* in the FACILITY general resource class. This authorizes the named counter server to act as a server for the named counter pool.

For example, if the user ID of the server is DFHNCSV1 and the pool name is DFHNC001, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY DFHNC.DFHNC001 UACC(NONE)
PERMIT DFHNC.DFHNC001 CLASS(FACILITY) ID(DFHNCSV1) ACCESS(CONTROL)
```

See [“System authorization facility \(SAF\) responses to the named counter server”](#) on page 544 for information about the responses to the CFDT server.

## Access to named counter servers

You can control access to the named counter servers by CICS regions.

Each time that a CICS region connects to a named counter server, a security check is made against the CICS region userid to verify that the region is authorized to use the services of that named counter server.

Give each CICS region userid that connects to a named counter server UPDATE access to the CICS RACF profile called *DFHNC.poolname* in the FACILITY general resource class. This authorizes the CICS region to use the services of the named counter server for the named counter pool.

For example, if the userid of a CICS region is CICSDA1, and the pool name is DFHNC001, the following RACF commands define the profile and provide the required access:

```
RDEFINE FACILITY DFHNC.DFHNC001 UACC(NONE)
PERMIT DFHNC.DFHNC001 CLASS(FACILITY) ID(CICSDA1) ACCESS(UPDATE)
```

When a CICS region has connected to a named counter pool, it can define, update, delete, get, rewind, and query named counters without any further security checks being performed by the server.

**Note:** Unlike shared temporary storage pools and coupling facility data table pools, named counters can also be accessed by batch application regions. Batch jobs are subject to the same security mechanisms as a CICS region.

## System authorization facility (SAF) responses to the named counter server

If the security profile for a named counter pool cannot be retrieved, SAF neither grants nor refuses the access request.

In this situation:

- Access to the named counter pool, either by a CICS region or by the named counter server itself, is rejected, if a security manager is installed, but is either temporarily inactive or inoperative for the duration of the z/OS image. This is a fail-safe action, on the grounds that, if the security manager is active, it might retrieve a profile that does not permit access to the named counter pool.
- Access to the named counter pool, either by a CICS region or by the named counter server itself, is accepted under either of the following situations:
  - There is no security manager installed.
  - There is an active security manager, but the FACILITY class is inactive, or there is no profile in the FACILITY class. The access request is allowed in this case because there is no evidence that you want to control access to the named counter server.

Access is permitted to any named counter server without a specific *DFHNC.poolname* profile, or an applicable generic profile. No messages are issued to indicate this. To avoid any potential security exposures, you can use generic profiles to protect all, or specific groups of, named counter servers. For example, the following command ensures that access is allowed only to named counter servers with a more specific profile to which a named counter server or a CICS region is authorized.

```
RDEFINE FACILITY (DFHNC.*) UACC(NONE)
```

## Authorizing access to SMSVSAM servers

SMSVSAM is a data-sharing subsystem running on its own address space to provide the RLS support required by CICS. For CICS regions using VSAM record-level sharing (RLS), access to SMSVSAM servers is controlled by RACF security checks made against the CICS region userid to verify that the region is authorized to register with an SMSVSAM server.

In a test environment you might want to use the default action and allow any CICS region using VSAM RLS to connect to an SMSVSAM server. If you want to protect this access, the RACF SUBSYSNM general resource class must be active and you must authorize each CICS region that connects to an SMSVSAM server to have access to that server. This means granting access to the appropriate profile in the RACF SUBSYSNM general resource class.

The general resource class, SUBSYSNM, supports authorizations for subsystems that want to connect to SMSVSAM. The SUBSYSNM profile name is the name by which a given subsystem is known to VSAM. CICS uses its applid as its subsystem name; define a profile for the CICS applid in the SUBSYSNM resource to enable CICS to register the control ACB.

When CICS attempts to register the control ACB during initialization, SMSVSAM calls RACF to check that the CICS region userid is authorized to the CICS profile in the SUBSYSNM class. If the CICS region userid does not have READ authority, the open request fails.

For example, if the applid of a CICS AOR is CICSDA1, and the CICS region userid (shared by a number of AORs) is CICSDA##, define and authorize the profile as follows:

```
RDEFINE SUBSYSNM CICSDA1 UACC(NONE) NOTIFY(userid)
PERMIT CICSDA1 CLASS(SUBSYSNM) ID(CICSDA##) ACCESS(READ)
```

You can use wildcard characters on the applid to specify more than one CICS region, for example:

```
PERMIT CICSDA%% CLASS(SUBSYSNM) ID(CICSDA##) ACCESS(READ)
```

## Authorizing access to the CICS region

You can restrict access by terminal users to specific CICS regions by defining CICS APPLID profiles in the RACF APPL class.

For this purpose, the APPLID of a CICS region is:

- The z/OS Communications Server generic resources name if GRNAME is specified as a system initialization parameter
- The generic APPLID if one is specified on the APPLID system initialization parameter
- The specific APPLID if only one is specified on the system initialization parameter

If you define a profile in the APPL class for a CICS APPLID, or a generic profile that applies to one or more CICS APPLIDs with UACC(NONE), all terminal users trying to sign on to a CICS region must have explicit access to the profile that applies to that region's APPLID, either as an individual profile, or as a member of a group. For example:

```
RDEFINE APPL cics_region_applid UACC(NONE) NOTIFY(sys_admin_userid)
```

You need to define only one APPL profile name in the RACF database for all the CICS regions that are members of the same z/OS Communications Server generic resources name. All sign-on verifications in a CICSplex, where all the terminal-owning regions have the same z/OS Communications Server generic resources name, are made against the same APPL profile.

For MRO only, the APPLID is propagated from the terminal-owning region (TOR) to the other regions that the user accesses — for example, from the TOR to the application-owning region (AOR), and from the AOR to the file-owning region (FOR). As a consequence:

- You do not need to include users of the AOR and FOR in the APPL profiles for those regions.
- You can force users to sign on through a TOR, by denying access to other APPLIDs

Use the RACF PERMIT command to add authorized users to the access list of CICS APPL profiles. For example:

```
PERMIT cics_region_applid CLASS(APPL) ID(group1,...,groupn) ACCESS(READ)
```

permits all users defined in the listed groups to sign on to cics\_region\_applid.

The APPL class must be active for this protection to be in effect:

```
SETROPTS CLASSACT(APPL)
```

Also, for performance reasons, consider activating profiles in the APPL class using RACLIST.

```
SETROPTS RACLIST(APPL)
```

If the APPL class is already active, refresh the in-storage APPL profiles with the SETROPTS command:

```
SETROPTS RACLIST(APPL) REFRESH
```

**Note:**

1. CICS always passes the APPLID to RACF when requesting RACF to perform user sign-on checks, and there is no mechanism within CICS to prevent this.
2. RACF treats undefined CICS APPLIDs as UACC(READ).
3. If the APPL class is active, and a profile exists for a CICS region in the APPL class, ensure that authorized remote CICS regions can sign on to a CICS region protected in this way.

See the [z/OS Security Server RACF Security Administrator's Guide](#) for more information about controlling access to applications.

## Controlling the opening of a CICS region's z/OS Communications Server ACB

You can control which users among those who are running non-APF-authorized programs can OPEN the z/OS Communications Server SNA ACB associated with a CICS address space (CICS region).

This ensures that only authorized CICS regions can present themselves as z/OS Communications Server applications providing services with this APPLID, thus preventing unauthorized users impersonating real CICS regions. The CICS region user ID needs the OPEN access, not the issuer of the **SET VTAM OPEN** command.

For each APPLID, create a VTAMAPPL profile, and give the CICS region user ID READ access. For example:

```
RDEFINE VTAMAPPL applid UACC(NONE) NOTIFY(userid)
PERMIT applid CLASS(VTAMAPPL) ID(cics_region_userid) ACCESS(READ)
```

The correct CICS APPLID to specify in the VTAMAPPL class is the specific APPLID, as specified in the CICS system initialization parameters.

The VTAMAPPL class must be activated using RACLIST for this protection to be in effect:

```
SETROPTS CLASSACT(VTAMAPPL) RACLIST(VTAMAPPL)
```

If the VTAMAPPL class is already active, refresh the in-storage VTAMAPPL profiles with the SETROPTS command:

```
SETROPTS RACLIST(VTAMAPPL) REFRESH
```

**Note:** VTAM is now the z/OS Communications Server (for SNA or IP)

## Controlling userid propagation

Jobs submitted from CICS to the JES internal reader without the USER operand being specified on the JOB statement run under the CICS region user ID. These jobs have the access authorities of the CICS region itself, and so could potentially expose other data sets in the MVS system.

You (or the RACF security administrator) can prevent the CICS region user ID from being propagated to these batch jobs by defining a profile in the PROPCNTL class where the profile name is the CICS regions user ID. For example, if the CICS region userID is CICS1, define a PROPCNTL profile named CICS1:

```
RDEFINE  PROPCNTL  CICS1
```

The PROPCNTL class must be activated using RACLIST for this protection to be in effect:

```
SETROPTS CLASSACT(PROPCNTL) RACLIST(PROPCNTL)
```

If the PROPCNTL class is already active, refresh the in-storage PROPCNTL profiles with the SETROPTS command:

```
SETROPTS RACLIST(PROPCNTL) REFRESH
```

You (or the RACF security administrator) must issue the SETROPTS command to refresh these profiles. Issuing the CICS PERFORM SECURITY REBUILD command does not affect the PROPCNTL class.

## Surrogate job submission in a CICS environment

Batch jobs submitted by CICS can be allowed to run with a USER parameter other than the CICS region's userid, but without specifying the corresponding PASSWORD.

This is called surrogate job submission. These jobs have the access authorities of the USER parameter specified on the JOB statement. If the PASSWORD parameter is specified on the JOB statement, surrogate processing does not occur.

You (or the RACF security administrator) can allow this by defining a profile in the SURROGAT class. For example, if the CICS region's userid is CICS1, and the job is to run for userid JOE, define a SURROGAT profile named JOE.SUBMIT:

```
RDEFINE  SURROGAT  JOE.SUBMIT  UACC(NONE)  
          NOTIFY(JOE)
```

Further, you must permit the CICS region's userid to act as the surrogate to the profile just defined:

```
PERMIT JOE.SUBMIT CLASS(SURROGAT) ID(CICS1) ACCESS(READ)
```

The SURROGAT class must be activated using RACLIST for this protection to be in effect:

```
SETROPTS CLASSACT(SURROGAT) RACLIST(SURROGAT)
```



### Attention:

Any CICS user, whether signed on or not, is able to submit jobs that use the SURROGAT userid, if the CICS userid has authority for SURROGAT. If your installation is using transient data queues to submit jobs, you can control who is allowed to write to the transient data queue that goes to the internal reader. However, if your installation is using EXEC CICS SPOOLOPEN to submit jobs, you cannot control who can submit jobs (without writing an API global user exit program to screen the commands). CICS spool commands do no CICS resource or command checking.

You can use an EXEC CICS ASSIGN USERID command to find the userid of the user who triggered the application code. Application programmers can then provide code that edits a USER operand onto the JOB card destined for the internal reader.

For a complete description of surrogate job submission support, see the [z/OS Security Server RACF Security Administrator's Guide](#).

## JES spool protection in a CICS environment

Your installation can protect JES spool data sets with profiles in the JESSPOOL class.

Spool files created by the **SPOOLOPEN** commands have the user ID of the CICS region in their security tokens, not the user ID of the person who issued the **SPOOLOPEN** command. Thus, the user ID qualifier in the related JESSPOOL profiles is the CICS region's user ID.

When the **SPOOLOPEN INPUT** command is used, CICS checks that the first four characters of the APPLID correspond to the external writer name of the spool file. This checking is independent of any RACF checking that might also be done.

When the **SPOOLWRITE** command is used to write to the internal reader, CICS performs a surrogate user check to verify if the user is authorized to submit a job with the user ID specified on the job card. For more information, see [Security for submitting a JCL job to the internal reader](#).

### Related concepts

[CICS interface to JES](#)

### Related reference

[SPOOLOPEN INPUT](#)

[SPOOLOPEN OUTPUT](#)

[SPOOLWRITE](#)

## Authorizing use of HPO in PARM parameter on an EXEC PGM=DFHSIP statement or in SYSIN

When you specify **HPO** in the **PARM** parameter on an EXEC PGM=DFHSIP statement in the JCL of a CICS region or in the SYSIN data set of the CICS startup job stream, you should use a RACF profile to control the use of **HPO** and give the CICS region user ID necessary access to this profile.

### Creating a RACF profile for HPO

You must create a RACF profile DFHSIT.HPO in the FACILITY class to define the necessary security authorization required for the use of **HPO** in the **PARM** parameter on an EXEC PGM=DFHSIP statement or in SYSIN:

#### Example:

```
RDEFINE FACILITY DFHSIT.HPO UACC(NONE)
```

UACC(NONE) means that the command rules apply, by default, to all users.

### Giving CICS region user ID READ access to profile DFHSIT.HPO

You must give the associated region user ID READ access to the profile by using **PERMIT** command:

#### Example:

```
PERMIT DFHSIT.HPO CLASS(FACILITY) ID(CICS_region_userid) ACCESS(READ)
```

## Setting up PassTickets

For information about PassTickets, see [How it works: PassTickets](#).

### Before you begin

To use PassTickets, the systems involved must meet the following requirements:



- The PassTicket generation and validation algorithm means that the system that generates the PassTicket (the originating system) and the system that authenticates it (the destination system) must both use a level of RACF that supports PassTickets.
- End users must use the same user ID in the destination system as the one that they use in the originating system.
- Because PassTickets are time-stamped, the system clocks for the destination system and the originating system must be synchronized to within the valid time range. A PassTicket is considered to be within the valid time range when the time of generation, with respect to the clock on the generating computer, is within plus or minus 10 minutes of the time of evaluation, with respect to the clock on the evaluating computer. For more information about system time differences and synchronization, see [Using PassTickets in z/OS Security Server RACF Security Administrator's Guide](#).

## Procedure

1. Define Secure Sign-on keys to enable the external security manager to process PassTickets.

To process PassTickets, the external security manager uses Secure Sign-on keys that are shared by the originating system and the destination system. You must define a Secure Sign-on key for each destination system. For information about how to do this with RACF by defining profiles in the PTKTDATA resource class, see [Using PassTickets in z/OS Security Server RACF Security Administrator's Guide](#).

2. Define RACF profiles to allow an originating system to generate a PassTicket.

It is strongly recommended that you limit PassTicket generation to only those regions that require it. The regions should be set with the system initialization parameter **XPTKT=YES**. This is the default.

This is a profile for users on a specific originating system:

```
RDEFINE PTKTDATA IRRPTAUTH.applid.* UACC(NONE)
PERMIT IRRPTAUTH.applid.* CLASS(PTKTDATA) ID(user) ACCESS(UPDATE)
```

*applid* is the generic applid of the originating region. *user* is the user or group of users allowed to generate PassTickets on this region.

3. Define RACF profiles to allow a destination region to accept a PassTicket.

```
RDEF PTKTDATA applid SSIGNON(key-description) UACC(NONE)
```

*applid* is the generic applid of the destination region.

4. If RACLIST is used on PTKTDATA, refresh the definitions.

Issue SETR RACLIST(PTKTDATA) REFRESH.

## Supporting parallel PassTicket requests for the same user ID

PassTickets are one-time use tickets. The generation mechanism has a granularity of 1 second; therefore, if the same command to generate a PassTicket for a user ID is issued twice within the same second, it will generate the same PassTicket. This would limit requests that use PassTickets to one per second for each applid. However, it is possible to support parallel PassTicket requests for the same user ID.

To allow more than one request per second per user for an applid, you can define separate keys for separate groups for a user.

```
RDEF PTKTDATA applid.group1.userid SSIGNON(KEYMASKED(key1)) UACC(NONE)
RDEF PTKTDATA applid.group2.userid SSIGNON(KEYMASKED(key2)) UACC(NONE)
```

You must write custom code that generates PassTickets with the appropriate key.

For the destination region to accept the PassTicket, the region will need to use the group associated with the key when authenticating the password, for example, on the **SIGNON** or **VERIFY PASSWORD** command. If no group is specified, the default group is used.

## Using PassTickets in FEPI applications

A FEPI application uses the PassTicket and user ID to perform a sign-on in the back-end system just as if it were sending a password and user ID.

### Before you begin

To process PassTickets, the external security manager uses keys, known as Secure Signon keys, that are shared by the front-end and back-end systems. You must define a Secure Signon key for each target system with which FEPI communicates. For information about how to do this with RACF by defining profiles in the PTKTDATA resource class, see [Using PassTickets in z/OS Security Server RACF Security Administrator's Guide](#). Users of other ESMs should refer to the documentation for their product.

### Procedure

1. The end user is verified by signing on to the front-end CICS system in the usual way.
2. When the end-user runs a transaction that uses FEPI, your application issues a **FEPI REQUEST PASSTICKET** command to obtain a PassTicket.  
If EDF is being used, the PassTicket is not displayed. The user ID for which the PassTicket is generated is that of the currently signed-on user. Your FEPI application can use an **EXEC CICS ASSIGN** command to check the user ID of the currently signed-on user.
3. Your FEPI application uses the PassTicket and user ID to perform a sign-on in the back-end system. For example:

```
EXEC CICS FEPI SEND FORMATTED
                        CONVID(convid) FROM(CESN userid PassTicket)
                        FROMLENGTH(length_of_data)
```

It is the application's responsibility to provide the sign-on processing, because CICS cannot know either the type of back-end system (CICS or IMS) or the back-end program being used for sign-on processing.

4. The back-end system uses an unchanged interface to perform the sign-on. A CICS system that receives a user ID and a PassTicket can use its existing procedures to sign on the user ID. RACF takes care of the fact that a PassTicket, rather than a password, is passed to it.

### Results

If the PassTicket times out (because, for example, of a session failure), your application must generate another and try to sign on again. If sign-on continues to fail and the front-end and back-end are in different MVS systems, check that the system clocks are suitably synchronized. Too many failed sign-on attempts could result in the user ID being revoked.

### What to do next

For detailed information about PassTickets, see [Using PassTickets in z/OS Security Server RACF Security Administrator's Guide](#).

## The sign-on process

When users log on to CICS through the z/OS Communications Server, but do not sign on, they can use only those transactions that the CICS default user is permitted to use. As these are likely to be strictly limited, users must sign on to obtain authorization to run the transactions that they are permitted to use.

You can explicitly sign on either by using one of the CICS-supplied sign-on transactions CESL or CESN or by using an installation-provided sign-on transaction that uses the **EXEC CICS SIGNON** command.

OIDCARD users can use CESL or CESN to sign on if the card reader supports the DFHOPID identifier (AID). If it does not, use your own installation-provided sign-on transaction. When you sign on to CICS, the sign-on process involves the following phases:

## Scoping

After the sign-on panel is completed and sent, CICS verifies that the user ID you entered does not match a user ID already signed on in the scope of the **SNSCOPE** definition for the CICS system.

## Identification

CICS calls RACF to confirm that a profile has been defined for your user ID.

## Verification

CICS passes information to RACF to verify that the user ID is genuine. For RACF, this information is either a password or an OIDCARD or both. If the password entered is between 9 and 100 characters, RACF verifies it with the password phrase associated with your user ID. If the password is 8 characters or less, RACF verifies it with your standard password. A user ID can have both a standard password and a password phrase. If the password or password phrase has expired, CICS prompts you for a new password or password phrase. When the new password conforms to the RACF password formatting rules for an installation, the new password or password phrase and the date-of-change are recorded in your RACF user profile.

Passwords and password phrases operate independently of each other. You can sign on with a valid password phrase even if your standard password has expired. Similarly, you can sign on with a valid password if your password phrase has expired.

Immediately following the request to RACF for user ID and password verification, CICS clears the internal password field. This minimizes the possibility of the password or password phrase being revealed in any dump of the CICS address space that might be taken.

The CESL and CESN transactions do not warn users when a password or password phrase is about to expire. To display expiry warnings to users, you must replace CESL or CESN with your own transaction that calls a custom sign-on program. This program can use the **EXEC CICS VERIFY PASSWORD** or **EXEC CICS VERIFY PHRASE** commands to return expiry information from RACF, before using the **EXEC CICS SIGNON** command to make a full verification request for the user ID.

## Authorization

RACF performs checks on the application name and the port of entry to verify that you are allowed to use the CICS system. In the application name check, RACF determines whether you are authorized to access the application named in the APPLID or GRNAME system initialization parameter. RACF does this by checking the access list of the CICS application profile defined in the RACF APPL resource class. (See [“Authorizing access to the CICS region” on page 545](#) for information about how to define profiles in the APPL resource class.)

With the port of entry check, RACF verifies that you are authorized to sign on using that port of entry. The use of defined terminals can be restricted to certain times of the day, and to certain days of the week. See [“Controlling access to CICS from specific ports of entry” on page 553](#).

These checks restrict CICS users to signing on only to those CICS regions for which they are authorized, and only from terminals they are authorized to use.

Explicit sign-on, with the CESL or CESN transaction, or the SIGNON command, is performed at the port of entry.

Table 48. Explicit and implicit signons		
Phase	Explicit	Implicit
Scoping	Yes	No
Identification	Yes	Yes
Verification	Yes	No except with ATTACHSEC(IDENTIFY)
Authorization	Yes	Yes

## User attributes

CICS obtains CICS user attributes from the CICS and LANGUAGE segments of the RACF database.

## Implicit sign on

Implicit sign-on means that all other user IDs added to the system by CICS are considered to be implicitly signed on without a password or password phrase.

## The sign-off process

---

The sign-off process dissociates a user from a terminal where the user had been previously signed on. The user can explicitly sign off using the CESF transaction or an installation-provided transaction that uses the SIGNOFF API command. If the attributes of the signed-on user include a non-zero value for TIMEOUT, an implicit sign-off occurs if this interval expires after a transaction terminates at this terminal.

When the timeout period expires, if the default GNTRAN=NO is specified, CICS performs an immediate signoff. If GNTRAN specifies a transaction-id to be scheduled and that transaction performs a signoff, the action CICS takes depends on the SIGNOFF option specified in the terminal's TYPETERM resource definition.

An exceptional case is that the goodnight transaction is not used for the user of a CRTE session. A surrogate user whose time expires is signed off, losing the security capabilities the terminal previously had. Message DFHSN1200 is sent to the CICS log, and indicates what has happened.

For more information about the use of system initialization parameter GNTRAN, see [GNTRAN system initialization parameter](#). The possible signoff options and the associated actions are as follows:

### **SIGNOFF(YES)**

CICS signs off the operator from CICS, but the terminal remains connected.

### **SIGNOFF(LOGOFF)**

CICS signs off the operator from CICS **and** logs off the terminal from z/OS Communications Server.

In addition, if the terminal is auto-installed, the delay period specified by the AILDELAY operand in the system initialization parameters commences, and if the delay period expires before the terminal attempts to log on again, CICS deletes the terminal entry (TCTTE) from the TCT.

### **SIGNOFF(NO)**

CICS leaves the user signed on and the terminal remains logged on, effectively overriding the timeout period.

## Explicit sign off

Explicit sign-off removes the user's scoping. The user must be explicitly signed on before signing off with the CESF transaction or the SIGNOFF command. The user is returned to the default level of security.

**Note:** CESL or CESN will not sign the user off until a valid attempt has been made to use the CESL or CESN panel, even if the sign-on attempt subsequently fails. It is not recommended that CESL or CESN be used for the Goodnight transaction.

## Implicit sign-off

A user is implicitly signed off if the transaction suffers a TERMERR condition while attempting to send data to its principal facility. However, the user is not subject to USRDELAY but is signed off immediately. If SNSCOPE is in use, the scope will be released at the time of sign off. If the transaction handles the ABEND, it continues running as a non-terminal task with the authority of the starting user.

## Goodnight transaction

By specifying your own GNTRAN transaction, you can use the CICS API to control the TIMEOUT operation. For example, your transaction could display a screen that prompts for the password.

Specifying the USERID option on the **ASSIGN** command obtains the userid, and the **VERIFY PASSWORD** command would validate the input. Based on the response, the user could remain signed on or be signed off.

By default CICS uses the CESF transaction to sign-off a user terminal. The goodnight transaction is not available for a surrogate terminal that is timed out during a CRTE session. Sign-off occurs with a loss of the security capabilities the terminal previously had, leaving a DFHSN1200 message in the log.

## Controlling access to CICS from specific ports of entry

---

During sign-on processing, CICS issues a request to RACF to verify the user's password, and to check whether the user is allowed to access that terminal.

This check is also performed for the userid specified for preset security terminal definitions. Autoinstalled consoles that are using automatic sign-on are treated as though they have a preset security definition (see “Preset terminal security” on page 556). If the terminal is not defined to RACF, RACF responds to CICS according to the system-wide RACF option specified by the SETROPTS command. The options are as follows:

### **TERMINAL(READ)**

With this option in force, terminal users can sign on at any terminal covered by a profile to which they have been permitted access, or at any terminal not defined as protected by RACF.

### **TERMINAL(NONE)**

With this option in force, terminal users can sign on at only those terminals with specific terminal profiles defined to RACF, and which they are authorized to use.

**Note:** The TERMINAL class does not control access from MVS consoles. These are controlled by the CONSOLE resource class. See “Console profiles” on page 556.

You can override the system-wide terminal options at the RACF group level by means of the group terminal options, TERMUACC or NOTERMUACC.

See “Universal access authority for undefined terminals” on page 555 for more information about the SETROPTS command for terminals, and about the TERMUACC|NOTERMUACC option on groups.

## Defining port of entry profiles

---

*Port of entry* is the generic term for the device at which the user signs on. For CICS, the port of entry can be either a terminal or a console. You can use associated port of entry profiles to control whether a user can sign on at a particular device.

## Terminal profiles

You can control user access to a terminals using resource profiles in the TERMINAL and GTERMINL resource classes.

This section briefly describes some aspects of terminal profiles that are of interest to CICS users. For more detailed information about defining and protecting terminals on z/OS systems, see the [z/OS Security Server RACF Security Administrator's Guide](#).

- Creating a resource profile in the TERMINAL or GTERMINL class
- Preventing the use of an undefined terminal
- Restricting specific groups of users to specific terminals
- Restricting the days or times when a terminal can be used
- Using a security label to control a terminal.

You can control user access to a terminal by defining it to RACF. (User access is determined at CICS sign-on time.) RACF supports two IBM-supplied resource class names for terminals:

### **TERMINAL**

Defines a resource profile of an **individual** terminal.

### **GTERMINL**

Defines a resource profile of a **group** of terminals.

**Note:** For a GTERMINL profile, RACF always uses an in-storage profile, which must be manually refreshed. Every time you create, change, or delete a GTERMINL profile, you (or the RACF security administrator) must issue a SETROPTS RACLIST(TERMINAL) REFRESH command for the change to take effect.

For CICS, the terminal profiles to define to RACF in the TERMINAL or GTERMINL class are used only for SNA LUs.

The name of the resource profile is the value of the NETNAME that is specified in the RDO terminal definition or autoinstall. It is not possible to use TERMINAL profiles with non-SNA LUs.

For example snippets of defining terminal resource profiles using **TERMINAL** and **GTERMINAL**, see the following topic.

## Example terminal profiles

This topic contains example RACF commands for defining a resource profile for an individual terminal or a group of terminals.

### Defining a profile of an individual terminal

To define terminals with NETNAMEs NETID1, NETID2, and NETID3 in the TERMINAL resource class, use the command:

```
RDEFINE TERMINAL (NETID1, NETID2, NETID3) UACC(NONE)
        NOTIFY(sys_admin_userid)
```

If the terminal IDs start with the same characters, you can create a generic profile to cover a group of terminals with the same initial characters. You must use the SETROPTS GENERIC command before defining generic profiles, as described in [Security administration tasks and the RACF commands to run them](#). This reduces the amount of effort needed to create the access list. For example:

```
RDEFINE TERMINAL NETID* UACC(NONE)
        NOTIFY(sys_admin_userid)
PERMIT netid* CLASS(TERMINAL)
        ID(group1, group2,..., groupn) ACCESS(READ)
```

### Defining a profile of a group of terminals

You can define the same terminals in the resource group class, by including them as members of a suitable terminal group. For example:

```
RDEFINE GTERMINL term_groupid
        ADDMEM(NETID1, NETID2, NETID3) UACC(NONE)
        NOTIFY(sys_admin_userid)
```

To remove a terminal from a resource group profile, specify the DELMEM operand on the RALTER command. For example:

```
RALTER GTERMINL term_groupid
        DELMEM(NETID3)
```

To allow a group of users in a particular department to have access to these terminals, use the PERMIT command as follows:

```
PERMIT term_groupid CLASS(GTERMINL) ID(dept_groupid) ACCESS(READ)
```

## Universal access authority for undefined terminals

RACF supports a universal access facility for undefined terminals, which you can control by means of the SETROPTS TERMINAL command (provided you have the necessary authorization). When SETROPTS TERMINAL(NONE|READ) is in effect, it affects **all** MVS terminal subsystems.

If SETROPTS TERMINAL(READ) is in effect, RACF allows any user to log on at any undefined terminal (that is, a terminal not defined in the TERMINAL or GTERMINL resource classes). If SETROPTS TERMINAL(NONE) is in effect, RACF does not allow anyone to log on at any undefined terminal.

**Note:** Before issuing the SETROPTS TERMINAL(NONE) command, define some TERMINAL or GTERMINL class profiles, with enough authorizations to ensure that at least some of the terminals can be used otherwise no one will be able to access any terminal.

You can override the SETROPTS TERMINAL command at the group level by specifying the TERMUACC or NOTERMUACC option on the ADDGROUP or ALTGROUP command.

The effect of the TERMUACC parameter is to enforce the universal access option. For example, if SETROPTS TERMINAL(READ) is active, the TERMUACC option means that any users in the group can access any undefined terminal. On the other hand, if you specify NOTERMUACC for the group, the SETROPTS TERMINAL command has no effect for that group, and a user in the group needs explicit authorization to use a terminal. To enable a group with the NOTERMUACC option to access terminals, you must add group userid to the access list of the appropriate TERMINAL or GTERMINL profile.

## Conditional access processing

Using RACF, you can permit a user to access resources when that user is signed on a particular terminal or console, but not otherwise. Access that is restricted in this way is known as *conditional access*.

To grant conditional access to a resource, add

```
WHEN (TERMINAL (netname))
```

or

```
WHEN (CONSOLE (console-name))
```

to the PERMIT command.

The following example allows members of the PAYROLL group to read the files in the PAYRAPP member list wherever they are signed on. They would be able to update it only from the terminal with netname PAY001, by issuing the following commands:

```
RDEFINE HCICSFCT PAYRAPP UACC(NONE) ADDMEM(SALARY,STAFF)
PERMIT SALARY CLASS(HCICSFCT) ID(PAYROLL) ACCESS(READ)
PERMIT SALARY CLASS(HCICSFCT) ID(PAYROLL)
(WHEN(TERMINAL(PAY001)) ACCESS(UPDATE))
```

To allow members of the operations group OPS to be able to use the transactions defined in the OPERATOR member list only from the console names MVS1MAST, issue the following command:

```
RDEFINE GCICSTRN OPERATOR UACC(NONE) ADDMEM(CEMT,CETR)
PERMIT OPERATOR CLASS(GCICSTRN) ID(OPS) WHEN(CONSOLE(MVS1MAST)) AC(READ)
```

### Note:

1. The CONSOLE class must be active before CONSOLE conditional access lists can be used.
2. Conditional access lists may only increase authority and not decrease it.

For other considerations on conditional access lists, see the [z/OS Security Server RACF Security Administrator's Guide](#).



## Console profiles

Use the CONSOLE resource class to define profiles that control user access to a console.

If the CONSOLE class has been activated, you can control whether a user is allowed to sign on to a console. Console protection is implemented in a method similar to that for protecting terminals, with the exception of the following:

1. The SETROPTS TERMINAL command does not apply to consoles.
2. The TERMUACC group attribute does not apply to consoles.

Before activating the CONSOLE class, check [z/OS MVS Planning: Operations](#) for the effects of console protection on MVS consoles.

The profile used in the console class is the console name or number. For example:

```
RDEFINE CONSOLE CICSCONS UACC(NONE)
```

The user must have READ access to the console name to sign-on at a console. The following example shows how user CICSOPR would be permitted to sign on to the console named CONCICS1:

```
RDEFINE CONSOLE CONCICS1 UACC(NONE)
PERMIT CONCICS1 CLASS(CONSOLE) ID(CICSOPR) ACCESS(READ)
```

Note that, unlike the case with TERMINAL protection, a sign-on attempt will fail if made at a console that has not been defined in the activated CONSOLE class. The access authority to undefined consoles is NONE.

## Preset terminal security

For some terminals, and MVS consoles when used as CICS terminals, it is appropriate to use preset terminal security as an alternative to terminal user security.

A terminal becomes a preset security terminal when you specify the USERID attribute on the TERMINAL resource definition.

There are two types of preset security for consoles, described below:

1. Normal preset security (the same as preset security for other terminals)
2. Automatic preset security. Automatic preset security applies only to console definitions. CICS automatic preset security allows you to associate the userid, which MVS has already verified through RACF, with the CICS definition for the console.

If you intend to use preset security, you need to be aware of some additional considerations:

- Autoinstall models:

If you are using autoinstall models with preset security, CICS makes the same surrogate authorization check as for ordinary terminals when the model is installed. It does not check surrogate authorization when the autoinstall model is used to perform autoinstall for a device. Also, CICS does not make a surrogate authorization check when it is installing models defined with automatic preset security for consoles.

If an autoinstall model with a preset userid becomes invalid (for example, if the userid is revoked), any attempt to install a terminal with the model fails.

- Sessions with preset security:

A session becomes governed by preset security if you specify the userid operand on the session definition. The same checking is performed if you install preset security sessions.

- Terminals that are defined in the TCT:

For terminals such as BSAM terminals that are defined in the terminal control table (TCT) by using DFHTCT macros, the userid is also defined in the TCT, and, when CICS initializes, it signs on these terminals. If the sign-on fails (for example, if the userid is revoked), the terminal is put out of service. If



the userid later becomes valid (for example, if it is resumed), setting the terminal in service results in a successful sign-on. CICS does not perform a surrogate user check for these terminals.

## Normal preset security

CICS preset terminal security allows you to associate a userid permanently with a terminal, or console, that is defined to CICS.

This means that CICS implicitly signs on the device when it is being installed, instead of a subsequent sign-on of that terminal by a user. Typically, you define preset security for devices without keyboards, such as printers, at which users cannot sign on.

You can also use the normal preset security on ordinary display terminals as an alternative to terminal user security. This permits anyone with physical access to a terminal with preset security to enter the transactions that are authorized for that terminal. The terminal remains signed on as long as it is installed, and no explicit sign-off can be performed against it. If the userid associated with a display terminal with preset-security has been authorized to use any sensitive transactions, ensure that the terminal is in a secure location to which access is restricted. Preset-security might be appropriate, for example, for the terminals physically located within a CICS network control center.

You can use preset-security to assign a userid with **lower** authority than the default, for terminals in unrestricted areas.

For example, to define a terminal with preset-security, use RACF and CICS (CEDA) commands as follows:

```
ADDUSER userid NAME(preset_terminal_user_name) OWNER(owner_userid or group_id)
          DFLTGRP(group_name)
CEDA DEFINE TERMINAL(cics_termid) NETNAME(vtam_termid) USERID(userid)
          TYPETERM(cics_ttypeterm)
```

## Automatic preset security

Instead of specifying an actual user id on the TERMINAL definition, you specify a special value (\*FIRST or \*EVERY), to indicate that CICS is to use the user id passed by MVS on the MODIFY command. This means that CICS implicitly signs on the console when it is being installed, and optionally on each input message, instead of a subsequent sign-on of that console by a user. Particularly in the context of autoinstalled consoles, this allows you to gain the advantage of preset security without having to define the user id/console relationship in the CICS terminal definition. Thus, console users do not have to sign-on with passwords in the clear to each CICS region.

You can use this automatic form of preset security on predefined consoles, autoinstalled consoles, and consoles installed with the **CREATE TERMINAL** command.

For example, to define a console with automatic preset-security, which is checked, and altered (if necessary) on every MODIFY, use CICS (CEDA) commands as follows:

```
CEDA DEFINE TERMINAL(cics_termid)
          CONSNAME(console_name) USERID(*EVERY)
          TYPETERM(cics_ttypeterm)
```

To define a console with automatic preset-security that is defined on the first valid MODIFY command only, use CICS (CEDA) commands as follows:

```
CEDA DEFINE TERMINAL(cics_termid)
          CONSNAME(console_name) USERID(*FIRST)
          TYPETERM(cics_ttypeterm)
```

## Controlling the use of preset security

When a preset security terminal is installed, the specified userid is implicitly signed on at the terminal.

Ensure that only a trusted person is allowed to define and install terminals with preset security, because the userid specified on the terminal might have access to CICS resources not available to the installer. Automatic preset security for consoles does not carry the same risks because the console user is

associated with their true identity (verified by RACF). For this reason, no checking is carried out when a console device is defined to CICS with either USERID(\*EVERY) or USERID(\*FIRST).

Surrogate user checking ensures that a user is authorized to act for another user. Surrogate user checking can be enforced when a user installs a terminal that is preset for a different userid, and is specified by the RACF SURROGAT resource class. The CICS *userid*.DFHINSTL resource can be defined in the SURROGAT resource class for authorization to install terminals that are preset for that specific userid.

When a terminal is installed with a preset userid, the surrogate user is the userid that is performing the installation. For more information, see [Surrogate security](#).

The CEDA command checks the authority of the user to install preset terminals. Consider, therefore, whether to restrict the following functions with a view to controlling who can define and install terminals with preset security:

- The CEDA transaction:

Restricting the use of the CEDA transactions to authorized users gives you control over who can define resources, such as terminals, to CICS. For information about protecting CICS-supplied transactions, see [Transactions in CICS](#)

- The SURROGAT resource class:

If you restrict who can install terminals with preset security, even when such terminals are defined in the CSD, only authorized users can install them on CICS. This authority is in addition to the authority needed to run CEDA; the user must already have authority to run the CEDA transaction.

To define a surrogate profile and authorize a user to install a terminal definition with preset security, use the following commands:

```
RDEFINE userid1.DFHINSTL SURROGAT UACC(NONE)
PERMIT userid1.DFHINSTL CLASS(SURROGAT) ID(userid2) ACCESS(READ)
```

This permits *userid2* to install a terminal preset with *userid1*

- The XUSER system initialization parameter:

To ensure that CICS can perform surrogate user security checks on the use of the CEDA INSTALL command for terminals with preset security, define the **XUSER** system initialization parameter. See [Security-related system initialization parameters](#) for information about defining this parameter.

- The LOCK command for locking CSD definitions

CICS installs resource definitions in the CSD during an initial or cold start, from the list of groups that are defined on the **GRPLIST** system initialization parameter. To control the addition of resource groups to the CICS startup group list, use the CEDA LOCK command to lock the list. This lock protects the group list from unauthorized additions. Also, lock all the groups that are specified in this list.

**Note:** The OPIDENT of the signed-on user is used as the key for the CEDA LOCK and CEDA UNLOCK commands. For more information about the LOCK and UNLOCK commands, see [The CEDA LOCK command](#) and [The CEDA UNLOCK command](#).

**Note:** When CICS installs a GRPLIST that contains preset terminal definitions, no checking is done at initialization time. However, you can still ensure that you control who can define and install terminals and sessions with preset security by using the CEDA LOCK command to control the contents of GRPLIST groups.

## Using a z/OS system console as a CICS terminal

If you intend to use an MVS system console as a CICS terminal, you might need authorization to use the MVS MODIFY command by using the OPERCMDS resource class.

You can specify automatic preset security on the console's CICS terminal definition so that the console user obtains the correct level of authority without explicitly performing a CICS sign-on (which exposes the password).

If preset security is not defined, console users must sign on to get authority different from the default user. In this case, the password or password phrase can generally be seen on the console and system log. However, if CICS is defined as an MVS subsystem in a JES2 system, you can use the `HIDEPASSWORD=YES` option of the `DFHSSIxx` member in `SYS1.PARMLIB`, which enables CICS to intercept the command and overwrite the password or password phrase with asterisks. For more information about defining CICS as an MVS subsystem, see [Defining CICS as an MVS subsystem](#).

You can use the `CESL` or `CESN` commands to sign on to CICS from a console. With `CESL`, you can use either a standard password or a password phrase as authorization. `CESL` treats any password over 8 characters as a password phrase. `CESN` does not support the use of password phrases. The format of the `CESL` and `CESN` sign-on commands, when entered from a console, is as follows:

```
MODIFY jobname,command_name [USERID=userid] [,PS=password]
      [,NEWPS=newpassword] [,GROUPID=groupid]
      [,LANGUAGE=language-code]
```

If a passphrase contains mixed-case characters, blanks, or both, the value of the `PS` parameter must be enclosed in single quotation marks, for example:

```
F JATP3250,CESL USERID=JAT232,PS='MOND July 17th'
```

The maximum length of the **MODIFY** command is 126 characters, including `F taskname`; the maximum length of a passphrase is 91 characters when it is specified in this way:

```
F JATP3250,CESL USERID=JAT284,PS='This is a valid long passphrase of
length 91 chars for CESL JAT testing 9XYZQ, #,.12345678!i'
```

For more information, see [Guidance for issuing console commands in z/OS](#).

With `CESL`, the `PS` (password) and `NEWPS` (new password) parameters must match; they must be either both password phrases or both standard passwords. You cannot authorize a new password phrase that uses a password and you cannot authorize a new password by using a password phrase. The `PS` and `NEWPS` parameters can contain spaces or punctuation characters, including single quotation marks, but each of these characters must be enclosed in two single quotation marks.

If any of the data entered on the command is invalid, or if the password or password phrase is missing or expired, CICS terminates the sign-on attempt.

You can authorize TSO users to use the `TSO CONSOLE` command. (For more information on this command, see [z/OS TSO/E System Programming Command Reference](#).) These users must be defined to CICS as consoles, by using the `CONSNAME` option of the `DEFINE TERMINAL` command, or be supported by autoinstall for consoles. For more information, see [Autoinstalling z/OS consoles](#).

When the `PS` parameter is omitted from the `CESL` or `CESN` command, RACF can produce a security violation message, `ICH408I`. `CESL` and `CESN` cannot distinguish a user who is defined with `OIDCARD`, `NOPASSWORD` from a user who is defined with a `PASSWORD` authenticator who intentionally omits the password. To establish whether to prompt for a password or to reject the sign-on (a user who is defined with `OIDCARD` cannot sign on at a console), the sign-on must be attempted. If the sign-on fails, message `ICH408I` is issued, and CICS interprets the return code from RACF to determine whether the `PASSWORD` or `OIDCARD` authenticator is required.

Users can sign on using `CESL` or `CESN`, or you might prefer to use preset security (the normal preset security for CICS terminals, or automatic preset security for consoles). When the TSO user uses the `CONSOLE` command, that user's user ID, by default, becomes a console name. However, you can change the console name to something else by using the `CONSNAME(name)` option on the `TSO CONSOLE` command. This console name can then be used as a CICS terminal if there is a corresponding `TERMINAL` definition with the `CONSNAME` option in CICS (or if you autoinstall a terminal definition). If another name is specified, that name is the one CICS uses to communicate with the console. For example, it is possible for one TSO user to use a name that is the same as another TSO user's ID.

Furthermore, if the `CONSOLE` command is used to allow TSO operators to sign on to CICS with the `CESL` or `CESN` transaction, their passwords might be exposed on the TSO screen and in the MVS system log. You

can prevent these potential exposures by defining the terminal as having preset security. It is advisable to use automated preset security for several reasons:

- TSO users do not have to sign on, which avoids exposing their IDs and passwords on the log.
- You do not have to define a relationship in a CICS definition between a console name and a user. A relationship might change frequently or become invalid.
- You can define one autoinstall model that covers most of your console definitions and gives each user the correct level of preset security.

To define automatic preset security, specify USERID(\*EVERY) to ensure that the correct user ID is signed on for every command, or USERID(\*FIRST) to sign on the console by using the user ID that first issues an MVSMVS MODIFY command to CICS, and retain this for subsequent commands.

- Choose USERID(\*FIRST) if use of a console is restricted to one or more users who have similar security characteristics to CICS using RACF, and you don't use the user ID as an identifier in applications.
- Use USERID(\*EVERY) if you need to ensure that each input request is tested to be sure that the console user has the correct security level. You should be aware that checking the user ID imposes an overhead on MODIFY, and changing the preset user ID imposes another overhead, which is equivalent to the console user signing on using CESL or CESN.

## Security for transient data

Follow this procedure to implement security for transient data queues.

### About this task

You can read an item from a transient data queue only once, because whenever you read from a transient data queue, CICS deletes the entry (by performing a "destructive read"). Therefore, if you specify security with SEC=YES as a system initialization parameter, CICS requires a minimum authorization level of UPDATE for all TD commands (DELETEQ, WRITEQ, and READQ).

For information about the access authorization levels for system programming commands which apply to transient data queues, see [Resource and command check cross-reference](#).

CICS itself uses a number of transient data queues. These queues are defined in group DFHDCTG, which is part of DFHLIST. If you want to protect access to these definitions from user application programs, define them to RACF with UACC(NONE) and without an access list. In the supplied resource definitions, most of the transient data queues are indirect, pointing to the transient data queues CSSL or CCSO. If you use the definitions as supplied, define to RACF only the queue names CSSL and CCSO, as follows:

```
RDEFINE ECICSDCT CICSQUEUES UACC(NONE)
                ADDMEM(CSSL, CCSO)
                NOTIFY(sys_admin_userid)
```

### Procedure

1. Specify RESSEC(YES) in the resource definition of the appropriate transactions.
2. Define profiles to RACF in the DCICSDCT or ECICSDCT resource classes (or their equivalent if you have user-defined resource class names), with access lists as appropriate.

Transient data queue names are a maximum of 4 characters in length, such as CSMT, L86O, L86P, and so on.

For example, use the following commands to define queues in the DCICSDCT class, and to authorize users to both read from and write to these queues:

```
RDEFINE DCICSDCT (qid1, qid2, ..., qidn) UACC(NONE)
                NOTIFY(sys_admin_userid)
PERMIT qid1 CLASS(DCICSDCT) ID(group1, group2) ACCESS(UPDATE)
PERMIT qid2 CLASS(DCICSDCT) ID(group1, group2) ACCESS(UPDATE)
```

To define transient data queues as members of a profile in the CICS transient data resource group class, with an appropriate access list, use the following commands:

```
RDEFINE ECICSDCT (queue_groupname) UACC(NONE)
                ADDMEM(qida, qidb, ..., qidz) NOTIFY(sys_admin_userid)
PERMIT queue_groupname CLASS(ECICSDCT) ID(group_userid) ACCESS(UPDATE)
```

When you are defining profile names to RACF to control access to transient data queues, define profiles only for:

- intrapartition transient data queues
- extrapartition transient data queues

Do not define profiles for indirect transient data queues; CICS directs all requests for an indirect queues to another queue, which can be extrapartition, intrapartition, or remote. The redirection can also be to another indirect queue.

If you are running CICS with security checking for transient data queues, CICS issues a call to RACF for each command that specifies a queue name. However, the resource name that CICS passes to RACF is the queue name of the final queue, which is not necessarily the name of the queue specified on the command.

For example, if an EXEC CICS command specifies queue QID2, which is defined as indirect to QID1, CICS calls RACF for an authorization check on QID1, not QID2. This is illustrated as follows:

```
TDQ definition: DEFINE TDQUEUE(QID1)
                  TYPE(EXTRA)
                  TYPEFILE(OUTPUT)
                  RECORDSIZE(132)
                  BLOCKSIZE(136)
                  RECORDFORMAT(VARIABLE)
                  BLOCKFORMAT(UNBLOCKED)
                  DDNAME(CICSMMSG)
                  GROUP(DFHDTG)

                  DEFINE TDQUEUE(QID2)
                  TYPE(INDIRECT)
                  INDIRECTNAME(QID1)
                  GROUP(DFHDTG)

CICS transaction: EXEC CICS WRITEQ TD
                  QUEUE(QID2)
                  FROM(data_area)
                  LENGTH(length)

CICS calls RACF:  Does the terminal user of the CICS transaction
                  have UPDATE authorization for QID1?
```

3. Specify SEC=YES as a CICS system initialization parameter (and SECPRFX if you define profiles with a prefix).
4. Specify XDCT=YES for the default resource class names of DCICSDCT and ECICSDCT (or XDCT=class\_name for user-defined resource class names).

## Security for journals and log streams

The CICS log manager provides facilities to write to and read from the CICS system log and CICS general logs. General logs comprise user journals, forward recovery logs, and autojournals. You can implement security for journals and log streams to protect them from unauthorized access.

The system log is used only for recovery purposes; for example, during dynamic transaction backout, or during emergency restart. Do not use it for any other purpose. Do not, therefore, write to it from a user application using the [WRITE JOURNALNAME](#) command.

CICS uses journal identifier **DFHLOG** for its primary system log. Do not permit user transactions to write to this. You can prevent them doing so by using the following command to define the system log in the JCICSJCT class, without any access list:

```
RDEFINE JCICSJCT DFHLOG UACC(NONE) NOTIFY(sys_admin_userid)
```

In addition to the automatic journaling and forward recovery logging that CICS performs for user transactions (depending on the options in the file resource definitions), user applications can also write user journal records using the **WRITE JOURNALNAME** command. The **WRITE JOURNALNUM** command is supported for compatibility with earlier releases but the **WRITE JOURNALNAME** command is preferred for new applications. If resource security applies to a transaction executing **WRITE JOURNALNUM**, the journal number is prefixed with 'DFHJ' before the security check is applied. As a result, writing to journal number 2 requires UPDATE access to the resource DFHJ02.

Users needing to write journal records must have authority to write to the JOURNALNAME (as defined in JCICSJCT). CICS calls RACF to perform a security check only for attempts to access a user journal by a CICS API command, and not for the journaling it performs in response to journaling options in the file resource definition. The CICS API does not provide a READ command for reading journals from a CICS transaction. For this reason, with proper exercise of control over the installation of applications on your CICS systems, you might consider it unnecessary to add RACF protection for journals that cannot be read from within CICS.

If you decide to implement security for CICS journals:

1. Specify RESSEC(YES) in the CSD resource definition of the transactions that write to journals.
2. Define profiles to RACF in the JCICSJCT or KCICSJCT resource classes (or their equivalent if you have user-defined resource class names) using the CICS journal name to identify the profiles.

To define journals as members of a profile in the journal resource group class, with an appropriate access list, use the following commands:

```
RDEFINE  KCICSJCT  userjnl UACC(NONE)
              ADDMEM(JRNL001, JRNL002, ....)
              NOTIFY(sys_admin_userid)
PERMIT   userjnl   CLASS(KCICSJCT) ID(group_userid) ACCESS(UPDATE)
```

3. Specify YES on the **SEC** system initialization parameter, and YES on the **SECPRFX** if you define profiles with a prefix. CICS requires a minimum authorization of UPDATE for journal access. For information about the access authorization levels for system programming commands that apply to journals, see [Resource and command check cross-reference](#).
4. Specify YES on the **XJCT** system initialization parameter for the default resource class names of JCICSJCT and KCICSJCT, or XJCT=*class\_name* for user-defined resource class names. For more information, see [XJCT system initialization parameter](#).

## Security for started transactions

A CICS transaction can start other transactions by means of an **EXEC CICS START** command. Transactions started in this way are known as **started transactions**, and you can use CICS RACF security to control who can start other transactions using the **START** command. CICS requires a minimum authorization of READ for started transactions.

When a transaction issues an **EXEC CICS START TRANSID** command, CICS calls RACF to check that the user of the transaction that issues the command is authorized for the started transaction.

To implement security for started transactions and for transactions checked against the XPCT class:

1. Specify SEC=YES as a CICS system initialization parameter (and SECPRFX if you define profiles with a prefix).
2. Specify RESSEC(YES) in the CSD resource definition of the transactions that issue **START** commands.
3. Specify XPCT=YES for the default resource class names of ACICSPCT and BCICSPCT (or XPCT=*class\_name* for user-defined resource class names).

This ensures that when a transaction is started by a **START** command, CICS calls RACF to check that the userid associated with the transaction is authorized to attach the transaction.

4. Define profiles to RACF in the ACICSPCT or BCICSPCT resource classes (or their equivalent if you have user-defined resource class names) using the name of the started transaction to identify the profiles.

For example, use the following commands to define a transaction in the ACICSPCT class, and to authorize one user only:

```
RDEFINE ACICSPCT (tran1, tran2, ..., trann) UACC(NONE)
          NOTIFY(sys_admin_userid)
PERMIT tran1 CLASS(ACICSPCT) ID(userid) ACCESS(READ)
PERMIT tran2 CLASS(ACICSPCT) ID(userid) ACCESS(READ)
```

To define started transactions as members of a profile in the started transaction resource group class, with an appropriate access list, use the following commands:

```
RDEFINE BCICSPCT started_trans UACC(NONE)
          ADDMEM(trana, tranb, ..., tranx)
          NOTIFY(sys_admin_userid)
PERMIT started_trans CLASS(BCICSPCT) ID(group_userid) ACCESS(READ)
```

5. Specify XTRAN=YES for the default resource class names of TCICSTRN and GCICSTRN (or XTRAN=*class\_name* for user-defined resource class names).
6. Define profiles to RACF in the TCICSTRN or GCICSTRN resource classes (or their equivalent if you have user-defined resource class names) using the name of the started transaction to identify the profiles.

## Transactions started at terminals

The **START** enables a CICS application program to start another transaction associated with a terminal other than the one from which the start command is issued. For example, the following command issued in CICS transaction *tranid1*, invoked at *termid1*, starts another transaction called *tranid2* at *termid2*:

```
EXEC CICS START
  TRANSID(tranid2)
  AT HOURS('18') MINUTES('50')
  TERMID(termid2)
```

When a **TERMID** is specified for the started transaction, CICS performs a transaction security check, using the classes **TCICSTRN** and **GCICSTRN**, on the user ID associated with the terminal (*termid2* in this example). You must therefore ensure that the *userid* associated with the terminal (*termid2*) is authorized to invoke the transaction. This *userid* is that of the signed-on user, or the CICS default *userid* if no user is signed on. If *termid2* is **not** authorized, message **DFHAC2033** is issued to the user of *termid2*. The user of the terminal that issued the **START** command gets a "normal" response. If the started transaction is defined with **RESSEC(YES)**, also ensure that the *userid* associated with the terminal (*termid2* in this example) is suitably authorized to access protected resources.

### Starting tasks at terminals defined with preset security

Typically, started transactions associated with a terminal are printing tasks, where the specified terminal is a printer. In this case, to associate a specific *userid* with the terminal, you define the terminal with preset security. See [“Preset terminal security” on page 556](#) for more information.

## Transactions started by EXEC CICS START

The **EXEC CICS START** command enables a CICS application program to start another transaction that is not associated with any terminal. When no **TERMID** is specified for the started transaction, the *userid* associated with the new transaction depends on whether you also specify the **USERID** option.

### User ID of a non-terminal started transaction

The **USERID** option of the **START** command (or the terminal user if no **TERMID** or **USERID** is included in the **START** command) determines the *userid* for a non-terminal started transaction. Without the **USERID** option, the non-terminal started transaction has the same *userid* as the transaction that executed the **START** command. If the **USERID** option is specified on the **START** command, the specified *userid* is used instead.



When an START command is executed without the TERMID option, CICS performs a surrogate user check to ensure that the transaction is authorized for the userid to be used by the non-terminal started transaction. For information about the link authorization of surrogate users, see [Intercommunication security](#). For information about EDF authorization of surrogate users, see [“Conditional access processing” on page 555](#).

**Note:** Distributed identity information is not preserved when a **START** command is run with any of the parameters: **USERID**, **TERMID**, or **RTERMID**, or is shipped across an LU61 or LU62 connection.

## Access to resources by a non-terminal started transaction

If the USERID option is not specified on the [STARTBROWSE PROCESS](#) command, the non-terminal started transaction does not always inherit all of the security of the transaction that executed the command. Also, it does not inherit resource access determined by link security, or resource access determined by a userid for EDF when used in dual-screen mode. This means:

- If a transaction-routed transaction executes a START command, or if a START command is function shipped, the non-terminal started transaction is not subject to link security.
- If EDF is used in dual-screen mode for a transaction that issues a START command, the non-terminal started transaction is not subject to resource access determined by the userid of the EDF terminal.

If you want the started transaction to have exactly the same security capabilities as the starting transaction, omit the USERID option. Without the USERID option, resource access by the non-terminal started transaction is determined by the sign-on parameters of the terminal transaction. These include the RACF group and the port of entry at which the terminal user signed on; that is, the terminal or console used to sign on, as shown in the following example:

A terminal user signs on using the CESN transaction at a terminal with netname NETNAMEX. For RACF, therefore, the port of entry is NETNAMEX. At the CESN screen the terminal user enters userid USERID1, and groupid GROUPID2. The terminal user then runs a terminal transaction which executes an EXEC CICS START command without the TERMID option or the USERID option specified. The non-terminal started transaction has resource access determined by userid USERID1, groupid GROUPID2, and port of entry NETNAMEX.

If a non-terminal transaction is denied access to a resource by RACF, the error message produced can include the terminal sign-on parameters, userid, and groupid. It can also include a port of entry. The userid, groupid, and port of entry can be those inherited from the terminal transaction that started the non-terminal transaction.

If the USERID option is specified on a START command, the non-terminal started transaction has access to resources determined by the userid specified on the USERID option.

We recommend that you do not specify the current userid of a terminal transaction on the USERID option. The non-terminal started transaction may not have the same resource access as the terminal transaction. The following examples show how the non-terminal started transaction can have different resource access:

### Example 1

RACF conditional access lists can be used by specifying WHEN(TERMINAL( ... )) or WHEN(CONSOLE( ... )) on the RACF PERMIT command to allow a terminal transaction access to certain resources because the specified port of entry is in use. See [“Conditional access processing” on page 555](#).

If a START TRANSID USERID command is executed by a terminal transaction specifying the same userid that the terminal user entered when signing on with CESN, the started transaction has access to resources determined by the specified userid, but not to the resources determined by the port of entry.

The started transaction is not subject to the conditional access list effective for the terminal transaction that executed the EXEC CICS START USERID command.

### Example 2

Using RACF you can grant (or deny) group access to a RACF protected resource.



A terminal user can enter a groupid and a userid when signing on with CESN. When the terminal user runs a terminal transaction, the groupid can determine resource access.

If a **START TRANSID USERID** command is executed by a terminal transaction specifying the same userid as that entered by the terminal user when signing on with CESN, the started transaction has access to resources determined by the specified userid. Resource access is not determined by the groupid that the terminal user entered when signing on with CESN. Resource access for the non-terminal started transaction can be determined by the default groupid for the specified userid.

The started non-terminal transaction is not subject to the group access effective for the terminal transaction that executed the **START USERID** command.

## Security for transactions started with **EXEC CICS RUN TRANSID**

---

A CICS transaction can initiate other transactions by means of an **EXEC CICS RUN TRANSID** command. You can use CICS RACF security to control who can initiate other transactions using the **RUN TRANSID** command.

When a transaction issues an **EXEC CICS RUN TRANSID** command, CICS calls RACF to check that the user of the transaction issuing the command is authorized for the started transaction.

To implement security for asynchronous transactions, you need to do the following:

1. Specify **SEC=YES** as a CICS system initialization parameter (and **SECPRFX** if you define profiles with a prefix).
2. Specify **RESSEC(YES)** in the CSD resource definition of the parent transactions that issue **EXEC CICS RUN TRANSID** commands.

This ensures that when a child transaction is started by an **EXEC CICS RUN TRANSID** command, CICS calls RACF to check that the userid associated with the transaction is authorized to attach the transaction.

3. Specify **XPCT=YES** for the default resource class names of **ACICSPCT** and **BCICSPCT** (or **XPCT=class\_name** for user-defined resource class names).
4. Define profiles to RACF in the **ACICSPCT** or **BCICSPCT** resource classes (or their equivalent if you have user-defined resource class names) using the name of the started child transaction to identify the profiles.
5. Specify **XTRAN=YES** for the default resource class names of **TCICSTRN** and **GCICSTRN** (or **XTRAN=class\_name** for user-defined resource class names).
6. Define profiles to RACF in the **TCICSTRN** or **GCICSTRN** resource classes (or their equivalent if you have user-defined resource class names) using the name of the started child transaction to identify the profiles.

### Userid of a transaction started using **EXEC CICS RUN TRANSID**

A child transaction started by the **EXEC CICS RUN TRANSID** command runs under the **USERID** of the parent transaction which issued the command.

### Access to resources by transactions started using **EXEC CICS RUN TRANSID**

- If a transaction-routed parent transaction executes an **EXEC CICS RUN TRANSID** command, the started child transaction is not subject to link security.
- If EDF is used in dual-screen mode for a transaction that issues an **EXEC CICS RUN TRANSID** command, the started transaction is not subject to resource access determined by the userid of the EDF terminal.

### Access authorization levels

CICS requires a minimum authorization of **READ** for transactions started by an **EXEC CICS RUN TRANSID** command.

## Security for XPCT-checked transactions

---

The **XPCT** system initialization parameter specifies whether CICS performs transaction resource security checking on transactions started by **EXEC CICS** commands, such as **EXEC CICS START** and **EXEC CICS RUN TRANSID**. Transactions defined in the ACICSPCT and BCICSPCT resource class profiles will be subjected to XPCT security checking.

These profiles also control access to transactions specified in certain other **EXEC CICS** commands, if the transaction issuing the command is defined with RESSEC(YES). The **EXEC CICS** commands affected by XPCT-checking are:

- START
- COLLECT STATISTICS TRANSACTION
- DISCARD TRANSACTION
- INQUIRE TRANSACTION
- SET TRANSACTION
- INQUIRE REQID
- CANCEL
- RUN TRANSID

## Security for CICS programs

---

CICS does not perform a security check on the initial program associated with a transaction. Instead, it controls access to the initial program by performing a security check on the associated transaction. You can control access to the initial program by authorizing users to initiate the associated transaction (transaction security). For programs invoked by others, you can perform the resource security check on them by using the **XPPT** system initialization parameter. CICS programs must be readable.

CICS application programs can invoke other programs by means of the LINK, LOAD, and XCTL commands. Also, the load status of programs can be altered by the CICS RELEASE, ENABLE, and DISABLE commands. Note, however, that there is no separate security check on the RELEASE of programs loaded for task lifetime. This is done on the corresponding LOAD.

You control access to programs invoked using these commands by defining profiles in the CICS application program classes, and which you define to CICS on the XPPT system initialization parameter.

To control which users can invoke or change the load status of other programs:

1. Specify RESSEC(YES) in the CSD resource definition of the transactions that use the LINK, LOAD, XCTL, RELEASE, ENABLE or DISABLE commands.
2. Define profiles to RACF in the MCICSPPT or NCICSPPT resource classes (or their equivalent if you have user-defined resource class names) using the name of the program invoked on the LINK, LOAD, or XCTL command to identify the profiles.

For example, use the following commands to define a program in the MCICSPPT class, and to authorize one user only:

```
RDEFINE MCICSPPT (prog1, prog2, ..., progn) UACC(NONE)
                                NOTIFY(sys_admin_userid)
PERMIT prog1 CLASS(MCICSPPT) ID(userid) ACCESS(READ)
PERMIT prog2 CLASS(MCICSPPT) ID(userid) ACCESS(READ)
```

To define programs as members of a profile in the application program resource group class, with an appropriate access list, use the following commands:

```
RDEFINE NCICSPPT cics_programs UACC(NONE)
                                ADDMEM(proga, progb, ..., progx)
                                NOTIFY(sys_admin_userid)
PERMIT cics_programs CLASS(NCICSPPT) ID(group_userid) ACCESS(READ)
```

3. Specify SEC=YES as a CICS system initialization parameter (and SECPRFX if you define profiles with a prefix).
4. Specify the **XPPT** system initialization parameter so that CICS performs the security check on programs that are invoked by **LINK**, **LOAD**, and **XCTL** commands:
  - **6.2** Set XPPT=(YES,ALL) for the default resource class names of MCICSPPT and NCICSPPT (or XPPT=(*class\_name*,ALL) for user-defined resource class names).
  - **6.1** Set XPPT=(YES) for the default resource class names of MCICSPPT and NCICSPPT (or XPPT=(*class\_name*) for user-defined resource class names).

## Exception for distributed program link (DPL) commands

If CICS finds that a program referenced on a **LINK** command is a remote program, it does not perform the security check in the region in which the link command is issued. The security check is performed only in the CICS region in which the linked-to program finally executes.

For example, if CICS A function ships a DPL command to CICS B, where the program then executes, CICS B issues the security check. If the DPL request is function shipped again to CICS C for execution, it is CICS C that issues the security check.

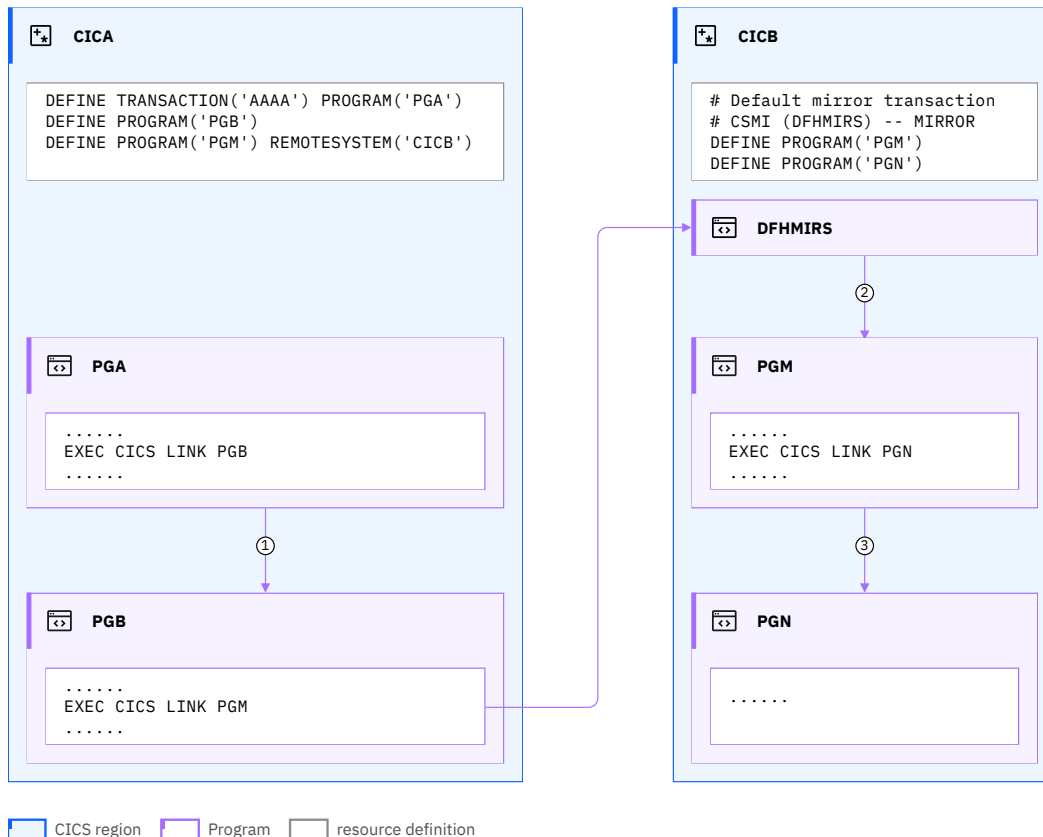
## Performing the security check only on the first linked remote program

**6.2** Applies to 6.2.

Performing the security check on all the invoked programs can be expensive. In case of distributed program link (DPL), you can configure CICS to check only the first program that is linked by the mirror program (DFHMIRS) by specifying DPLONLY on the XPPT system initialization parameter. This way you can reduce the security cost while retaining control over remotely linked programs. For more information about DPL and the mirror program, see [How it works: Illustration of a DPL request](#).

The following diagram shows program linking between two regions using DPL:

1. In the CICA region, the initial program (PGA) of the AAAA transaction issues a **LINK** to the PGB program. The PGB program then issues a **LINK** command to the PGM program in a remote region (CICB).
2. In the CICB region, the DFHMIRS mirror program is run by the CSMI mirror transaction to handle the remote request. The mirror program re-creates the request and links to the PGM program.
3. The PGM program links to the PGN program.



If you want to secure all the programs, specify `XPPT=(YES,ALL)` or `XPPT=(class_name,ALL)` in both regions. In this case, CICA issues the security check on the AAAA transaction and the PGB program while CICB issues the security check on PGM and PGN programs.

If you want to secure only the first program in both regions, specify `XPPT=(YES,DPLONLY)` or `XPPT=(class_name,DPLONLY)` in both regions. In this case, CICA issues the security check on the AAAA transaction while CICB issues the security check on the PGM program.

## Security for temporary storage

Unlike the other resources for which you specify `RESSEC(YES)`, temporary storage queues, for which you require RACF protection, also require the security attribute in a suitable TSMODEL resource definition. CICS requires a level of authorization appropriate to the temporary storage queue access intended, for example, a minimum of `READ` for `READQ TS`, and a minimum of `UPDATE` for `DELETEQ TS` and `WRITEQ TS`.

You specify TSMODEL definitions in the CSD. See [TSMODEL resources](#) for information about TSMODEL resource definitions.

You can define the queue names on the `PREFIX` attribute of the TSMODEL resource definition as follows:

- By specifying a fully identified name that exactly matches the queue name specified on a `READQ TS` or `WRITEQ TS` command. This can be from 1 to 16 alphanumeric characters.
- By specifying a generic name, or prefix, that corresponds to the leading alphanumeric characters of a set of queue names.

It follows that a prefix can only be from 1 to 15 characters, because if you specify the full 16 characters for a queue name, it must be the name of a specific temporary storage queue.

When a CICS application issues a temporary storage command (for example, DELETEQ TS, READQ TS, or WRITEQ TS) and temporary storage security is in effect, CICS searches for a TSMODEL resource definition that corresponds to the leading characters of the queue name.

Note that if you include hexadecimal characters in a temporary storage queue name, unpredictable results may occur. Also, if a temporary storage queue name contains an imbedded blank, RACF truncates the resource name to that blank.

To implement security for temporary storage queues:

1. Specify RESSEC(YES) in the CSD resource definition of the appropriate transactions.
2. Specify the security attribute on suitable TSMODEL resource definitions. CICS does not perform any security checks on temporary storage queues that specify SECURITY=NO on the matching TSMODEL definition.
3. Define profiles to RACF in the SCICSTST or UCICSTST resource classes (or their equivalent if you have user-defined resource class names), with access lists as appropriate. For example, use the following commands to define queues in the SCICSTST class, and to authorize users to both read from and write to these queues:

```
RDEFINE SCICSTST (tsqueue1, tsqueue2, ..., tsqueuen) UACC(NONE)
              NOTIFY(sys_admin_userid)
PERMIT tsqueue1 CLASS(SCICSTST) ID(group1, group2) ACCESS(UPDATE)
PERMIT tsqueue2 CLASS(SCICSTST) ID(group1, group2) ACCESS(UPDATE)
```

To define temporary storage queues as members of a profile in the CICS temporary storage resource group class, with an appropriate access list, use the following commands:

```
RDEFINE UCICSTST tsqueue_group UACC(NONE)
              ADDMEM(tsqueuea, tsqueueb, ..., tsqueuex)
              NOTIFY(sys_admin_userid)
PERMIT tsqueue_group CLASS(UCICSTST) ID(group_userid) ACCESS(UPDATE)
```

For more information about defining temporary storage profiles, see [Security for temporary storage](#).

4. Specify SEC=YES as a CICS system initialization parameter (and SECPRFX if you define profiles with a prefix).
5. Specify XTST=YES as a CICS system initialization parameter for the default resource class names of SCICSTST and UCICSTST (or XTST=class\_name for user-defined resource class names).

## Security for z/OS UNIX files

Files stored in the z/OS UNIX System Services file system can be used to supply Web pages through CICS Web support, as static responses provided by URIMAP definitions. When access control for these files is specified, you can control access to them on the basis of the user IDs for individual Web clients. Access control for z/OS UNIX files is enabled by default.

Access control for z/OS UNIX files is activated by the XHFS system initialization parameter. The default for this parameter is YES, meaning that resource security for z/OS UNIX files is active. If you do **not** want resource security for these files, set this system initialization parameter to NO.

The user IDs of Web clients, and also the CICS region user ID, must have a minimum of **read** access for z/OS UNIX files, and the directory containing them, which are used with CICS Web support as static responses provided by URIMAP definitions (specified by the HFSFILE attribute).

Access control for z/OS UNIX files is based on a user ID that is obtained from the Web client using basic authentication, or a user ID associated with a client certificate sent by the Web client. The user ID is used only during the process of security checking.

Access control for z/OS UNIX files differs from standard resource security for the other resource types controlled by Xname system initialization parameters, in some important ways:

- Access controls for z/OS UNIX files are not managed directly by RACF. They are specified in z/OS UNIX System Services, which makes use of RACF to manage user IDs and groups of user IDs, but keeps control of the permissions set for the files and directories. Because of this, you do not need to define

RACF profiles for individual files, and you cannot use the QUERY SECURITY command to check access to them. You check and specify permissions for z/OS UNIX files and directories in the z/OS UNIX System Services shell environment, using z/OS UNIX commands. RACF is used to manage user profiles, groups and access control lists (ACLs). If you are using ACLs, you need to activate the FSSEC class for these to be checked.

- Security checking for z/OS UNIX files is not affected by the RESSEC attribute in the TRANSACTION resource definition of the transactions that access the files. If XHFS=YES is specified as a system initialization parameter for the CICS region, all z/OS UNIX files used by CICS Web support as static responses (and their directories) are subject to security checking, regardless of the RESSEC attribute for the transaction that is accessing them. (However, the SEC system initialization parameter does affect whether or not security checking is carried out, as for all resources.)
- z/OS UNIX files are not referenced directly by any CICS application programming commands or system programming commands. They can only be referenced by EXEC CICS commands when they are defined as CICS document templates. In this situation, resource security for CICS document templates (specified by the XRES system initialization parameter) controls access to them for users. CICS does **not** perform any additional permissions check on the z/OS UNIX files using the Web client's user ID. This is the case even if access control is specified for z/OS UNIX files in the CICS region, or if resource security is not active for document templates. Where z/OS UNIX files are defined as CICS document templates, you therefore need to set up Web clients' user ID access controls in RACF for the CICS document templates, rather than in z/OS UNIX System Services for the z/OS UNIX files. (However, the CICS region user ID always needs to have **read** permissions on z/OS UNIX files, even if they are defined as document templates.) Note in particular that this situation applies to all application-generated responses from CICS Web support, and to any URIMAP definitions for static responses where the TEMPLATENAME attribute is used, rather than the HFSFILE attribute.
- If z/OS UNIX files are defined as CICS document templates, and the document templates are used either in URIMAP definitions as static responses (specified by the TEMPLATENAME attribute), or by applications, CICS does **not** perform any additional permissions check on the files using the Web client's user ID. However, the CICS region user ID still needs to have read permissions on the files, even when they are defined as document templates.

## Implementing security for z/OS UNIX files

To implement access control for z/OS UNIX files used by CICS web support, when they are specified as static responses in URIMAP definitions that use the HFSFILE attribute, follow the steps listed in this topic.

### Before you begin

The CICS region user ID must always have a minimum of **read** and **execute** permission to all z/OS UNIX files that it uses for CICS web support, and to the directories containing them. The user ID of the web client is only used when accessing z/OS UNIX files as a static response, but the CICS region user ID applies to all other attempts to access the file. If the CICS region user ID does not have permission to access the file, even an authorized web client is unable to view it. This is the case even when the file is defined as a CICS document template.

### About this task

#### Procedure

1. Select an appropriate method to give permissions to web clients to access the z/OS UNIX files and directories.

You might choose to use the group permissions for the files and directories, or access control lists (ACLs).

Even if it is possible for you to use group permissions, the use of ACLs is the preferred solution for giving permissions to web clients' user IDs. With ACLs, you can allow access to multiple user groups, and the access can be set for single files or once for all the files in a directory. Directory permissions can be arranged in the same way. You can also use ACL commands to modify permissions for the files

and directories. Although you work with ACLs in the z/OS UNIX System Services shell environment, they are created and checked by RACF, so if you are using a different security product, check its documentation to see if ACLs are supported.

When you have chosen your preferred method, follow the relevant steps in the remainder of this procedure.

2. Identify the authenticated user IDs used by web clients. These must be the basis of your access control. (You cannot supply an override by using an analyzer program, as you can with application-generated responses.)

Authenticated user IDs already have a user profile defined in your security manager.

3. For each web client user ID, choose and assign a suitable z/OS UNIX user identifier (UID). The UIDs are numbers that can be in the range 0 - 16 777 216. To assign UIDs, specify the UID value in the OMVS segment of the user profile for each user ID.

You can update a RACF user profile by using the ALTUSER command.

For example, if the web client's user ID is WEBUSR1, and the UID you want to assign is 2006, use the command:

```
ALTUSER WEBUSR1 OMVS(UID(2006))
```

All users must have a z/OS UNIX user identifier (UID) in their user profile in order to use z/OS UNIX function, even if you are not assigning permissions based on the UID. [z/OS UNIX System Services Planning](#) explains how to manage the UIDs and GIDs for your z/OS UNIX system.

4. Choose, or create, RACF groups that can be used by groups of web clients with the same permissions. For best performance, even if you are using ACLs, you should assign permissions to groups rather than individual users.
5. For each RACF group, choose a suitable z/OS UNIX group identifier (GID), and assign the GID to the RACF group. To assign a GID, specify the GID value in the OMVS segment of the RACF group profile. For example, if the RACF group is CICSWEB1, and the GID you want to assign is 9, use the command:

```
ALTGROUP CICSWEB1 OMVS(GID(9))
```

6. Make sure that each of your web client user IDs connects to a RACF group to which you assigned a z/OS UNIX group identifier (GID). If your web clients must connect to more than one RACF group, RACF list of groups must be active in your system.
7. Before you modify the permissions for the z/OS UNIX files and directories, ensure that your user ID is either a superuser on z/OS UNIX, or the owner of each z/OS UNIX file and directory you want to work with. Also, if you are working with groups, the owner of the files and directories must be connected to the RACF groups that you are using.
8. Optional: If you have chosen to use ACLs, set up ACLs that apply to all of the z/OS UNIX files and directories used by CICS web support for static responses, by using the `setfac1` command in the z/OS UNIX System Services shell environment.

[z/OS UNIX System Services Planning](#) has information about using ACLs, and examples of how to use the `setfac1` command.

- a) For files, you can set up access ACLs, which apply to an individual file, or file default ACLs, which apply to all the files within a directory and within its subdirectories.
- b) For directories, you can set up access ACLs, which apply to an individual directory, or directory default ACLs, which apply to the subdirectories within a directory.
- c) To minimize the impact to performance, assign group permissions for the files to the RACF groups to which your web clients' user IDs connect, rather than using individual user IDs. (There is also a limit on the number of items that can be specified in an ACL.)
- d) If you must change the permissions granted to the groups (the base permission bits which specify **read**, **write** and **execute** access), you can specify this using the `setfac1` command as well.

Web clients must have **read** access to the z/OS UNIX files and directories.



- e) If you are using ACLs, ensure that the FSSEC class is activated. Use the RACF command `SETROPTS CLASSACT(FSSEC)` to do this.  
You can define ACLs before activating the FSSEC class, but you must activate the FSSEC class before ACLs can be used in access decisions.
9. Optional: If you have chosen to use group permissions without using ACLs, assign the group permissions for each z/OS UNIX file and directory to a group to which your web clients connect, and give the group **read** permissions. Use the UNIX command `chmod` to do this. [z/OS UNIX System Services Command Reference](#) and [z/OS UNIX System Services User's Guide](#) have information about using this command.  
(Note that as group permissions only can be assigned to one group if you are using this method, some of your web clients' user IDs might need to connect to more than one group to acquire all the correct permissions.)
10. Specify `SEC=YES` as a CICS system initialization parameter. (`SECPRFX` is not relevant for z/OS UNIX files, as they do not have RACF profiles).
11. Specify `XHFS=YES` as a CICS system initialization parameter.  
This step activates access control for all z/OS UNIX files in the CICS region.

## Results

When you have completed the setup procedure, from this point onwards:

- All web clients who use a connection with basic authentication or client certificate authentication and attempt to access any HFS files, must have a user profile in the security manager which contains a valid z/OS UNIX UID, and connects to a RACF group with a valid z/OS UNIX GID.
- To be able to view a web page derived from a z/OS UNIX file, web clients who use a connection with basic authentication or client certificate authentication must have **read** permissions to the file and to the directories containing it, either individually or through the RACF groups to which they are connected.

If these conditions are not in place, web clients receive a 403 (Forbidden) status code, and CICS issues message DFHXS1116.

## Security for program specification blocks

DL/I program specification blocks (PSBs) are IMS control blocks that describe databases and logical message destinations used by an application program. PSBs consist of one or more program communication blocks (PCBs), which describe an application program's interface to an IMS database.

To implement security for PSBs scheduled in CICS applications:

1. Define profiles to RACF in the PCICSPSB or QCICSPSB resource classes (or their equivalent if you have user-defined resource class names), with access lists as appropriate. The resource profile names you define to RACF must correspond to the names of PSBs specified in CICS PSB schedule commands. For example, use the following commands to define PSBs in the PCICSPSB class, and to authorize users to access these queues:

```
RDEFINE PCICSPSB (psbname1, psbname2, ..., psbnamen) UACC(NONE)
                NOTIFY(sys_admin_userid)
PERMIT psbname1 CLASS(PCICSPSB) ID(group1, group2) ACCESS(READ)
PERMIT psbname2 CLASS(PCICSPSB) ID(group1, group2) ACCESS(READ)
```

To define PSBs as members of a profile in the CICS PSB resource group class, with an appropriate access list, use the following commands:

```
RDEFINE QCICSPSB psbname_group UACC(NONE)
                ADDMEM(psbnamea, psbnameb, ..., psbnamex)
                NOTIFY(sys_admin_userid)
PERMIT psbname_group CLASS(QCICSPSB) ID(group_userid) ACCESS(UPDATE)
```

2. Specify `SEC=YES` as a CICS system initialization parameter (and `SECPRFX` if you define profiles with a prefix).



3. Specify XPSB=YES as a CICS system initialization parameter for the default resource class names of PCICSPSB and QCICSPSB (or XPSB=class\_name for user-defined resource class names).
4. Specify PSBCHK=YES if you want full security for PSBs that are accessed in transaction-routed transactions. This applies to both types of DL/I interface (remote and DBCTL). If you specify PSBCHK=NO, the authority of the remote user is **not used** in transaction-routed transactions.

**Note:** CICS requires a minimum authorization of READ for PSBs.

If you are using DBCTL, see [Security checking with DBCTL](#) for information on defining security in a CICS-DBCTL environment.

## Security checking of transactions running under CEDF, CEDG, CEDX, or CEDY

---

When a transaction is run under one of the four EDF transactions, CICS checks the security settings for the target transaction.

The IBM-supplied definitions of CEDF, CEDG, CEDX, and CEDY in the DFHEDF group specify RESSEC(YES). Definitions in the IBM-supplied groups cannot be modified, so to change the definition, copy the transaction to another group.

When CEBR and CECI are invoked from within CEDF they are transaction-attach checked.

When one of the four EDF transactions is used to test a transaction, the authority of the user executing the transaction being tested is checked. For each resource accessed by the tested transaction, the user must have access authority, otherwise a NOTAUTH condition is raised. This requirement applies to all resource checks:

- Transaction attach
- CICS resource
- CICS command
- Non-CICS resources accessed through the QUERY SECURITY command
- Surrogate user

## Security for submitting a JCL job to the internal reader

---

To secure who can submit JCL, CICS requires a number of configuration and security definitions.

JCL jobs can be submitted in two ways:

- Using a **WRITEQ TD** command to an extrapartition TDQ defined to the internal reader
- Using the following SPOOL commands:
  - **SPOOLOPEN** with USERID("INTRDR") specified
  - **SPOOLWRITE**

There are three user IDs involved in job submission:

### **CICS region user ID**

This is the region user ID used to start CICS.

### **task user ID**

This is the user ID that the task is running under (the default user ID if not logged on).

### **job user ID**

This is the user ID specified by a USER parameter on the JOB card.

If the JOB statement doesn't contain a USER parameter, the default depends on the security settings as follows:

## JCL submitted through a TDQ

- First choice: The user ID specified in the JOBUSERID option in the TDQ definition
- Second choice: CICS region user ID

For details, see [“Security when using a TDQ to submit JCL” on page 574.](#)

## JCL submitted by SPOOL commands

- **6.2** Set by the **INTRDRJOBUSER** system initialization parameter, which can be either of the following IDs:
  - task user ID
  - CICS region user ID
- **6.1** The job user ID defaults to the region user ID. This can be changed to default to the task user ID by setting `com.ibm.cics.spool.defaultjobuser=TASK`.

**Security tip:** As security best practices, a zero trust strategy is recommended whereby any work should run under the identify of the user doing the work. Therefore,

- **6.2** The default (task user ID) of **INTRDRJOBUSER** is in accordance with this strategy.
- **6.1** Setting `com.ibm.cics.spool.defaultjobuser=TASK` is in accordance with this strategy.

For details, see [“Security when using the SPOOL commands to submit JCL” on page 575.](#)

If the job user ID isn't the CICS region user ID, CICS adds `USER=job_userid` to the JOB card.

### Tip: Specifying the CICS region user ID by using symbolics on the JOB card

If you want a job to be able to run under the CICS region user ID without needing to change the JCL dynamically, you can specify `USER=&SYSUID` on the JOB card. This runs the job under whatever CICS region user ID the job was submitted from. This applies to JCL jobs written using the **SPOOLWRITE** command or using a TDQ. A surrogate security check, if applicable, is made when jobs are submitted using `&SYSUID` to represent the CICS region user ID.

## What security protection applies

Resource security on the TDQ provides protection for JCL jobs that are submitted through the TDQ. Additional protection is provided by CICS surrogate user checking if the `USER` parameter is specified on the JOB card. For details, see [“Security when using a TDQ to submit JCL” on page 574.](#)

Protection for JCL jobs that are submitted by using **SPOOL** commands is provided by surrogate security. For details, see [“Security when using the SPOOL commands to submit JCL” on page 575.](#)

In addition, you must have a profile for the CICS region user ID in the JESSPOOL class to give the CICS region user ID the authority to submit jobs for the job user IDs. See [“z/OS surrogate user checking” on page 575](#) for details of the RACF definitions and error messages if the surrogate check fails.

## Security when using a TDQ to submit JCL

Protection is provided by a resource security check of the TDQ. For details, see [“Security for transient data” on page 560.](#)

Additional security configuration and checking depends on whether a user ID is specified on the job card written to the TDQ by using the **WRITEQ TD** command.

### If a user ID is specified

#### Example:

```
//JOBNAME JOB USER=job_userid
```

CICS can perform an additional surrogate check when writing the job card to the TDQ. This surrogate check verifies whether the task user ID has permission to submit jobs on behalf of the job user ID (*job\_userid* in the above example).

For information on how to enable this additional surrogate check, see [“CICS surrogate user checking” on page 576](#).

If the surrogate check fails, a NOTAUTH response is returned from the **WRITEQ TD** command. See [“CICS surrogate user checking” on page 576](#) for details of the RACF definitions and error messages if the surrogate check fails.

### If a user ID is not specified

#### Example:

```
//JOBNAME JOB
```

CICS adds a USER parameter to the statement written to the TDQ. The job user ID added is set to the value specified by the JOBUSERID option in the TDQ definition.

```
//JOBNAME JOB USER=job_userid
```

If JOBUSERID is not defined in the TDQ definition, the job user ID is set to the CICS region user ID. No additional surrogate checking is done by CICS.

```
//JOBNAME JOB USER=region_userid
```

## Security when using the SPOOL commands to submit JCL

To use spool commands, CICS must be started with the system initialization parameter **SPOOL=YES**.

You can submit a JCL by using a **SPOOLOPEN** command with USERID("INTRDR") specified. **SPOOLWRITE** commands are used to write the JCL statements.

If the JOB card has a USER parameter, a surrogate check is performed to verify whether the task user ID has permission to submit jobs on behalf of the job user ID.

For information on how to enable this surrogate check, see [“CICS surrogate user checking” on page 576](#).

If the surrogate check fails, a NOTAUTH response is returned from the **SPOOLWRITE** command. See [“CICS surrogate user checking” on page 576](#) for details of the RACF definitions and error messages if the surrogate check fails.

If the JOB card doesn't have a USER parameter, the job user ID defaults to a user ID that is set elsewhere:

- **6.2** The job user ID defaults to the user ID that is set by the **INTRDRJOBUSER** system initialization parameter. By the default of **INTRDRJOBUSER**, the task user ID is assumed, and no additional surrogate checking is done by CICS. It is also possible to set **INTRDRJOBUSER** to use the CICS region user ID instead of the task user ID as the default job user ID, in which case it is subject to a surrogate check.
- **6.1** The job user ID defaults to the region user ID. This is subject to a surrogate check. You can change the default for the job user ID to be the task user ID if you set the following feature toggle:  
`com.ibm.cics.spool.defaultjobuser=TASK` .

## z/OS surrogate user checking

Although the **SPOOLWRITE** and **WRITEQ TD** commands are issued by the task user ID, the job is submitted by the CICS region user ID. Therefore, if the job user ID is not the CICS region user ID and no password is supplied, you must grant the CICS region user ID surrogate authority to submit jobs on

behalf of the job user ID. This is required regardless of whether CICS surrogate user checking is active or not.

```
RDEFINE SURROGAT job_userid.SUBMIT UACC(NONE) OWNER(sysadmin)
PERMIT job_userid.SUBMIT CLASS(SURROGAT) ID(region_userid) ACCESS(READ)
```

If you configure your CICS region user IDs to be able to submit jobs on behalf of any user, it is recommended that the CICS region user IDs are not used for any other purpose than a user ID for running CICS.

The check takes place after the job is submitted. If the check fails, the job fails with an ICH408I message issued to the console and job log, but no response is returned to the application.

```
ICH408I USER(job_userid) GROUP(group) NAME(username)
SUBMITTER(region_userid)
LOGON/JOB INITIATION - SUBMITTER IS NOT AUTHORIZED BY USER
```

## CICS surrogate user checking

CICS surrogate user is enabled if:

- **6.2** System initialization parameter **XUSER=YES** is in effect.
- **6.1** System initialization parameter **XUSER=YES** is in effect and the feature toggle for surrogate user checking for spool commands is enabled: `com.ibm.cics.spool.surrogate.check=true`.

If the job user ID is not the task user ID and no password is supplied, you need to grant the task user ID surrogate authority to submit jobs on behalf of the job user ID.

```
RDEFINE SURROGAT job_userid.SUBMIT UACC(NONE) OWNER(sysadmin)
PERMIT job_userid.SUBMIT CLASS(SURROGAT) ID(task_userid) ACCESS(READ)
```

If a surrogate check is applicable, it takes place when the **SPOOLWRITE** or **WRITEQ TD** command writes the JOB card. If the check fails, the command fails with a NOTAUTH response. In addition, DFHXS1111

(**6.2** and DFHXS1117) messages are issued to the CICS log, and an ICH408I message is issued to the console and job log.

```
ICH408I USER(job_userid) GROUP(group) NAME(username) SUBMITTER(task_userid)
LOGON/JOB INITIATION - SUBMITTER IS NOT AUTHORIZED BY USER
```

## Learn more

If you want to migrate to using CICS surrogate user checking, follow the instructions in [Upgrading security](#).

### Related information

[Surrogate security](#)

## Configuring RACF for identity propagation

You must configure RACF® to be able to map distributed identities. You can configure your CICS® system to accept distributed identity information from IPIC connections or from web service requests. Your CICS Liberty server can also use distributed identity mapping.

Use the RACF **RACMAP** command to create, delete, and list a distributed identity filter. Use the RACF **SETR RACLIST (IDIDMAP)** command to refresh the IDIDMAP resource profile that contains the distributed identity filter. You must configure your RACF settings for identity propagation before you update clients and CICS configuration definitions.

If you are using MRO connections, you can accept distributed identity information between CICS systems only. You must ensure that ATTACHSEC(IDENTIFY) is specified, but no other configuration changes are required.

Ensure that you have the required access to update RACF profiles for your z/OS system.

The distributed user information is mapped in RACF. RACF is configured to map the distributed identity to a RACF user ID.

You are now ready to configure your IPIC connections, web service connections or CICS Liberty server to CICS. For more information about the RACF **RACMAP** and **SETR RACLIST (IDIDMAP)** commands, see the information about distributed identity filters in the [z/OS Security Server RACF Security Administrator's Guide](#).



---

## Chapter 30. Auditing CICS

How it works: [Auditing in CICS](#) summarizes the checks that auditors might make on CICS configuration, operations, authorization, and connections. This section tells you how to carry out those checks.

### Compliance data collection

---

The type 1154 record provides compliance evidence data. A different subtype is assigned to each participating z/OS component or product. On receiving an ENF86 signal from the z/OSMF Compliance REST API, participating components and products collect and write compliance data to their associated SMF 1154 subtype records. Each 1154 subtype record includes the SMF extended header, as defined by the IFASMFH macro, and the SMF1154 common area, as defined by the IFAR1154 macro. The remainder of the subtype data in each SMF1154 subtype record is unique to each participating component or product.

The CICS region collects data and writes an SMF 1154 subtype 80 with the information to SMF. No configuration is required to enable this function.

IBM recommendations about how to use the information in the SMF 1154 subtype 80 are documented in the [Center for Internet Security benchmark for CICS TS 6.1](#).

#### Reference

[IBM Z Security and Compliance Center documentation](#)  
[Using the z/OSMF REST services](#)

### Retrieving SMF 1154 subtype 80 records from CICS

The CICS region collects data and writes an SMF 1154 subtype 80 with the information to SMF. Use z/OS compliance data collection to collect compliance data from CICS regions.

#### Before you begin

Customize the JCL sample DFH\$54P in SDFHSAMP.

#### Procedure

1. Identify the LPARs for which you want to collect data. This identification results from discussion with the auditor and might focus on production regions.
2. Issue the restful request for these regions as described in [Using the z/OSMF REST services](#).
3. Wait for approximately three minutes to allow the data collection to complete.
4. Update your customized version of DFH\$54P with the date and time interval of data collection.
5. Submit the JCL.
6. Look at the SUMMARY output of the job to identify for which regions data is collected.
7. Open the CSV file on zFS by using a spreadsheet. If your tools for editing zFS don't have this option, you first need to download the file.

#### Related information

[SMF 1154 subtype 80 record](#)  
[Example of formatted SMF 1154 s](#)

### Example of formatted SMF 1154 subtype 80 records

Example of formatted SMF 1154 type 80 records that uses an LPAR running three CICS regions when the compliance data was requested.

The SUMMARY output is shown in [Figure 169 on page 580](#).

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Rec	RequestID	Jobname	UserID	LPAR	yyyymmdd hh:mm:ss
1	684L7PWWKWM6IX6	CICSDL3D	USERA	MV2I	20220203 11:15:05.20
2	684L7PWWKWM6IX6	CICSDL1D	USERA	MV2I	20220203 11:15:05.20
3	684L7PWWKWM6IX6	CICSDL2D	USERA	MV2I	20220203 11:15:05.20

Figure 169. Example SMF 1154 subtype 80 records

- **1** A number associated with the restful request. All data that is collected at the same time has the same number. If you have multiple requestIDs in the data, you can select just one by specifying the ID in the DFH\$54P JCL.
- **2** The jobname of the CICS regions.
- **3** The region userID of the CICS regions.
- **4** The LPAR on which the CICS jobs are running.
- **5** The date and time at which the data was collected.



Figure 170. Spreadsheet example

Column	Data contained in indicated column or columns
A	A number associated with the restful request.
B-F	Details of the CICS regions and where they are running.
G-K	Basic security-related configuration information.
L-V	CICS classes.
W-Z	CICS tags.
AA	The release.
AB-AK	Includes the five most recent PTFs and their dates in yyyymmdd format.

## Auditing CICS configuration with IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that helps to simplify and automate the identification of potential configuration problems before they impact availability or cause outages. CICS leverages IBM Health Checker for z/OS to provide health checks that define best practices for CICS configuration. You can use these checks to audit that CICS configuration conforms to best practices.



## What are health checks?

To help you validate important configuration settings in CICS, CICS provides health checks by leveraging IBM Health Checker for z/OS.

The ability to review key settings is also of value when CICS is audited, for example, providing feedback to external auditors.

IBM Health Checker for z/OS issues messages that indicate the severity of any issues identified by CICS health checks. Messages might be marked as information, warning, or exceptions. When a message is displayed in a production or other important region, it is required that the system programmer review the message and actions required. Compliance and regulatory requirements are your responsibility.

## Setting up for health checks

No configuration is necessary within CICS to run the health checks.

See [IBM Health Checker for z/OS User's Guide](#) for further details on how to use and configure IBM Health Checker for z/OS.

CICS region tagging can be used to manage the CICS health checks that are excluded from running. For example, you might have a development or sandbox region that you do not want checked to prevent false positives. You can identify such regions based on the assigned APPLID, region user name, or job name and exclude them from all CICS health checks. You might also want to exclude specific health checks from running for all regions. For more information, see [Classifying CICS regions with region tagging](#).

## Running the health checks

CICS health checks run automatically if IBM Health Checker for z/OS is enabled.

The CICS health checks start running in the CICS address space 3–4 minutes after the region initializes. After the initial run of the checks, they run every 30 minutes to check various configurations and resources in use in the CICS region. If any configurations or resources which do not conform with the best practices are detected, they are identified as non-compliant. The interval at which the CICS health checks run is fixed at 30 minutes.

By default, IBM Health Checker for z/OS runs every 30 minutes in the z/OS address space to issue messages about the non-compliant results returned by CICS health checks, for CICS regions that ran within the previous 30 minutes. You can change the interval at which IBM Health Checker for z/OS issues messages as described in [Cheat sheet: examples of MODIFY hzsproc commands](#). However, this does not affect the interval at which CICS health checks run. To review the details of the Health Check output, use the TSO SDSF CK command. For more information, see [Health Check panel \(CK\) in z/OS documentation](#). For more information about how to review messages, see [IBM Health Checker for z/OS User's Guide](#).

Health checks might fail during the region shutdown if the domain services that the checks need are unavailable.

## Related information

- [Reference: Available CICS health checks supported by IBM Health Checker for z/OS](#)

Find out what health checks are available and how IBM Health Checker for z/OS works in CICS.

- [Reference: IBM Health Checker for z/OS messages related to CICS](#)

Reference information explaining IBM Health Checker for z/OS messages that report on non-compliant issues found in CICS regions.

- [DFHHCnnnn messages](#)

Reference information explaining additional IBM Health Checker for z/OS messages that are issued by CICS.

- [IBM Health Checker for z/OS User's Guide](#)

## Identifying changes to resources (resource signatures)

---

The *resource signature* is the combination of the *definition signature* and the *installation signature*. It can be used to audit and manage resources by capturing when the resource is defined, installed, and last changed.

### Definition signature

The definition signature captures details about when, how, and by whom each resource is defined or changed in the CSD file or in the CICSplex SM EYUDREP data repository. The definition signature is updated each time a change is made to the resource.

The definition signature fields are DEFINESOURCE, DEFINETIME, CHANGEAGENT, CHANGEAGREL, CHANGETIME, and CHANGEUSRID.

Resources defined in CICS releases before CICS TS 4.1 do not have information displayed for the definition signature until they are modified in this CICS release or later. When the resource is modified, the DEFINETIME field remains blank.

### Installation signature

The installation signature shows when, how, and by whom each resource is installed. The installation signature fields are INSTALLAGENT, INSTALLTIME, and INSTALLUSRID.

See [Summary of the resource signature field values](#) for details about the fields and the values that are returned, by resource definition method.

The following resources support resource signatures:

ATOMSERVICE, BUNDLE, CONNECTION, DB2CONN, DB2ENTRY, DB2TRAN, DOCTEMPLATE, DUMPCODE, ENQMODEL, EPADAPTER, EPADAPTERSET, EVENTBINDING, FILE, IPCONN, JOURNALMODEL, JVMSERVER, LIBRARY, MQCONN, MQINI, NODEJSAPP, OSGIBUNDLE, PIPELINE, PROFILE, PROCESSTYPE, PROGRAM, TCPIPService, TDQUEUE, TRANCLASS, TRANSACTION, TSMODEL, URIMAP, WEBSERVICE, and XMLTRANSFORM.

## Finding the audit information

Resource signatures can be displayed in the CICS Explorer views, CICSplex SM views and resource table records, CEDA panels, CEMT **INQUIRE** commands, or by using EXEC CICS **INQUIRE** SPI commands.

The resource statistics field names for the resource signature end in CHANGE\_AGENT, CHANGE\_TIME, CHANGE\_USERID, DEFINE\_SOURCE, INSTALL\_AGENT, INSTALL\_TIME, and INSTALL\_USERID. For detailed information about the content of the resource signature fields, see [Summary of the resource signature field values](#).

To display a summary of the definition signatures for all the specified resources, add the **SIGSUMM** parameter to the **DFHCSDUP LIST** command. The definition signature fields are displayed with the resource attributes when you use the **OBJECTS** option on the command. The **DFHCSDUP EXTRACT** command also extracts the definition signature fields from the CSD file. For more information, see [Resource management utility DFHCSDUP commands](#).

## Resource signature field values

The resource signature fields capture when the resource is defined, installed, and last changed.

“Resource signature fields” on page 582 lists the fields and the possible values for each one. Table 49 on page 584 shows the combination of values that are returned, depending how the resource is defined.

### Resource signature fields

#### DEFINESOURCE

The source of the resource definition. The DEFINESOURCE value depends on the CHANGEAGENT.

#### DEFINETIME

The time when the resource definition was created using the **DEFINE**, **USERDEFINE**, **COPY**, **MOVE**, or **RENAME** commands. When you alter an existing resource by using the **ALTER** command, the value

specified by DEFINETIME does not change. On CEDA panels, the date is displayed in the format that you specified in the **DATFORM** system initialization parameter.

#### **CHANGEAGENT**

How the resource was defined or last modified:

##### **AUTOINSTALL**

From autoinstall

##### **CSDAPI**

From CEDA, the programmable interface to DFHEDAP, or EXEC CICS CSD command

##### **CSDBATCH**

DFHCSDUP

##### **DREPAPI**

From CICSplex SM **BAS** API command

##### **DYNAMIC**

- A PIPELINE scan (URIMAP or WEBSERVICE).
- CICS web template management, using DFHWBTL or DFHWBBMS (DOCTEMPLATE).
- The installation of a DB2ENTRY resource definition with transaction ID specified (DB2TRAN).
- The installation of an ATOMSERVICE resource definition with XSDBIND specified (XMLTRANSFORM).
- The installation of an MQCONN resource definition with INITQNAME specified (MQINI).

##### **OVERRIDE**

From a resource overrides file during installation. The CSD file or the CICSplex SM EYUDREP data repository are unchanged.

##### **SYSTEM**

From CICS or CICSplex SM system

##### **TABLE**

From a table definition

#### **CHANGEAGREL**

The level of the CICS system used for the definition of, or last modification to, the resource definition. This is a 4-digit identifier: for example, 0750 is the identifier for CICS TS for z/OS, Version 6.2

#### **CHANGETIME**

The time when the resource definition was last modified. When the resource is first defined, the CHANGETIME value is identical to the DEFINETIME value. On CEDA panels, the date is displayed in the format that you specified in the **DATFORM** system initialization parameter.

#### **CHANGEUSRID**

The ID of the user who defined or last modified the resource definition. For the CICSplex SM BAS resource definitions in the EYUDREP data repository, when CICS security is active the CHANGEUSRID field contains the user ID that made the last modification to the resource definition. When CICS security is not active, the CHANGEUSRID field contains blanks.

#### **INSTALLAGENT**

How the resource was installed:

##### **AUTOINSTALL**

From autoinstall

##### **BUNDLE**

From bundle deployment

##### **CREATESPI**

From EXEC CICS CREATE command

##### **CSDAPI**

From CEDA, the programmable interface to DFHEDAP, or EXEC CICS CSD command

## DYNAMIC

- A PIPELINE scan (URIMAP or WEBSERVICE).
- CICS web template management, using DFHWBTL or DFHWBBMS (DOCTEMPLATE).
- The installation of a DB2ENTRY resource definition with transaction ID specified (DB2TRAN).
- The installation of an ATOMSERVICE resource definition with XSDBIND specified (XMLTRANSFORM).
- The installation of an MQCONN resource definition with INITQNAME specified (MQINI).

## GRPLIST

From a GRPLIST INSTALL

## SYSTEM

From CICS or CICSplex SM system

## TABLE

From a table definition

## INSTALLTIME

The time when the resource was installed.

## INSTALLUSRID

The ID of the user who installed the resource.

Table 49. Summary of resource signature field values, by resource definition method									
How the resource is defined	DEFINE SOURCE	DEFINE TIME	CHANG EAGENT	CHANG EAGREL	CHANG ETIME	CHANG EUSRID	INSTALL AGENT	INSTALL TIME	INSTALL USRID
GRPLIST INSTALL	CSD GROUP	Time stamp of CSD record creation	CSDAPI, CSDBAT CH, or OVERRI DE	CICS release of CHANGE AGENT system	Time stamp of CSD record change	User ID that ran the CHANGE AGENT	GRPLIST	Time of the last cold start	Jobstep user ID
CEDA INSTALL or EXEC CICS CSD INSTALL	CSD GROUP	Time stamp of CSD record creation	CSDAPI, CSDBAT CH, or OVERRI DE	CICS release of CHANGE AGENT system	Time stamp of CSD record change	User ID that ran the CHANGE AGENT	CSDAPI	Time of the installati on	User ID that ran the installati on
EXEC CICS CREATE	Name of the program issuing the <b>CREATE</b> command	Time that the resource is created	CREATE SPI or OVERRI DE	CICS release of CHANGE AGENT system	Time that the resource is created	User ID that ran the EXEC <b>CICS CREATE</b> command	CREATE SPI	Time that the resource is created (from earlier run if warm-started)	User ID that ran the EXEC CICS <b>CREATE</b> command

Table 49. Summary of resource signature field values, by resource definition method (continued)

How the resource is defined	DEFINE SOURCE	DEFINE TIME	CHANG EAGENT	CHANG EAGREL	CHANG ETIME	CHANG EUSRID	INSTALL AGENT	INSTALL TIME	INSTALL USRID
Autoinst all	Autoinst all user program name	Time of the autoinst all	AUTOIN STALL or OVERRI DE	CICS release of CHANGE AGENT system	Time of the autoinst all	User ID that ran the autoinst all	AUTOIN STALL	Time of the autoinst all (from earlier run if warm-started)	User ID that ran the autoinst all
Table definitio n	Table name	Time of the last cold start	TABLE	CICS release of CHANGE AGENT system	Time of the last cold start	Jobstep user ID	TABLE	Time of the last cold start	Jobstep user ID
System defined	SYSTEM	Time of CICS startup	SYSTEM	CICS release of CHANGE AGENT system	Time of CICS startup	Jobstep user ID	SYSTEM	Time of CICS startup	Jobstep user ID
Dynamic ally created	<a href="#">Table 50 on page 586</a>	Time that the resource is generate d	DYNAMI C	CICS release of CHANGE AGENT system	Time that the resource is generate d	User ID that generate d the resource	DYNAMI C	Time that the installed resource is generate d	User ID that generate d the installed resource
Created by BUNDLE	BUNDLE name	Inherite d from BUNDLE	Inherite d from BUNDLE or OVERRI DE <b>2</b>	Inherite d from BUNDLE	Inherite d from BUNDLE	Inherite d from BUNDLE	BUNDLE	Inherite d from BUNDLE	Inherite d from BUNDLE
CICSPl e x SM (EXEC CICS CREATE)	CPSMV $n$ where $nn$ is the version of the CICSPl e x SM BAS resource definitio n (the VER attribute )	CREATE TIME from EYUDRE P	DREPAP I, OVERRI DE or SYSTEM	CICS release of CHANGE AGENT system	CHANGE TIME from EYUDRE P	CHANGE USRID from EYUDRE P	CREATE SPI	Time that the resource is created (from earlier run if warm-started)	User ID that ran the create or passed from CICSPl e x SM

Table 49. Summary of resource signature field values, by resource definition method (continued)

How the resource is defined	DEFINE SOURCE	DEFINE TIME	CHANGEAGENT	CHANGEAGREL	CHANGETIME	CHANGEUSRID	INSTALL AGENT	INSTALL TIME	INSTALL USRID
Created by CICSplex SM for platform	Management part name	Time that the resource is created	CREATE SPI or OVERRIDE	CICS release of CHANGE AGENT system	Time that the resource is created	User ID that ran the create or passed from CICSplex SM	CLOUD	Time that the resource is created	User ID that ran the create or passed from CICSplex SM
CICSplex SM SYSLINK	SYSLINK	Time that the resource is installed	DREPAP I or OVERRIDE	CICS release of CHANGE AGENT system	Time that the resource is installed	User ID that requested the SYSLINK installation <sup>1</sup>	CREATE SPI	Time that the resource is installed	User ID that requested the SYSLINK installation

**Note:**

1. For the CICSplex SM BAS resource definitions in the EYUDREP data repository, when CICS security is active the CHANGEUSRID field contains the user ID that made the last modification to the resource definition. When CICS security is not active, the CHANGEUSRID field contains blanks.
2. If the BUNDLE resource definition is overridden then the CHANGEAGENT for any resource created by the BUNDLE is not inherited but will be set to what it would have been had the BUNDLE not been overridden, e.g. CSDAPI, CSDBATCH e.t.c.

Table 50. The contents of the CHANGEUSRID, DEFINESOURCE, and INSTALLUSRID fields for dynamic resources

Dynamic resource	Generated by	CHANGEUSRID	DEFINESOURCE	INSTALLUSRID
DB2TRAN	The installation of a DB2ENTRY resource definition with TRANSID specified	User ID that installed the DB2ENTRY	DB2ENTRY name	User ID that installed the DB2ENTRY
DOCTEMPLATE	CICS Web template management, using DFHWBTL or DFHWBMS	User ID that ran DFHWBMS or DFHWBTL	DFHWBMS or DFHWBTL	User ID that ran DFHWBMS or DFHWBTL
DUMPCODE	The installation of a DUMPCODE resource definition by a <b>SET SYSDUMPCODE ADD</b> command or a <b>SET TRANDUMPCODE ADD</b> command	User ID that issued the <b>SET SYSDUMPCODE ADD</b> or <b>SET TRANDUMPCODE ADD</b> command	Name of the program that issued the <b>SET SYSDUMPCODE ADD</b> or <b>SET TRANDUMPCODE ADD</b> command	User ID that issued the <b>SET SYSDUMPCODE ADD</b> or <b>SET TRANDUMPCODE ADD</b> command
MQINI	The installation of an MQCONN resource definition with INITQNAME specified	User ID that installed the MQCONN	MQCONN name	User ID that installed the MQCONN
PROGRAM	The installation of a Liberty application containing an @CICSProgram annotation	User ID that installed the Link to Liberty application	BUNDLE name or \$WLPAPP if the application is not installed in a CICS bundle	User ID that installed the Link to Liberty application
URIMAP	A PIPELINE scan	User ID that ran the PIPELINE scan	PIPELINE name	User ID that ran the PIPELINE scan

Table 50. The contents of the <i>CHANGEUSRID</i> , <i>DEFINESOURCE</i> , and <i>INSTALLUSRID</i> fields for dynamic resources (continued)				
Dynamic resource	Generated by	CHANGEUSRID	DEFINESOURCE	INSTALLUSRID
WEBSERVICE	A PIPELINE scan	User ID that ran the PIPELINE scan	PIPELINE name	User ID that ran the PIPELINE scan
XMLTRANSFORM	The installation of an ATOMSERVICE resource definition with XSDBIND specified	User ID that installed the ATOMSERVICE	ATOMSERVICE name	User ID that installed the ATOMSERVICE

## Classifying CICS regions with region tagging

CICS region tags allow you to classify regions based on the assigned APPLID, region user ID, and job name for the region. You can then use these region tag definitions to aid with auditing, running CICS health checks, or providing categorization information for operators. Region tagging is optional for use in CICS.

### Why use region tagging?

You might use region tagging for the following purposes:

#### Displaying information for operators

Operators or other users with valid access can use [INQUIRE TAG](#) to identify the region tags in use, which you configure based on your existing naming conventions.

#### Auditing

Auditors can more easily determine the region usage.

#### Configuring CICS health check exclusions

You can define the following exclusions in the region tagging file. The two types of exclusion are independent of each other.

- Excluding regions from running all health checks. For example, you might exclude regions that are defined as Development or Sandpit.
- Excluding specific health checks from running for all regions. For example, you might want to disable the health check that detects whether the SIT parameter **XPCT** is set to No.

Checks from the following sets of health checks can be excluded:

- [CICS\\_REGION\\_CONFIGURATION](#)
- [CICS\\_RESOURCE\\_CONFIGURATION](#)
- [CICS\\_RESOURCE\\_SECURITY](#)
- [CICS\\_STABILIZED\\_FUNCTIONS](#)
- [CICS\\_USS\\_CONFIGURATION](#)

For more information about health checks in CICS, see [Reference: Available CICS health checks supported by IBM Health Checker for z/OS](#).

### How to configure region tags?

You specify region tags in a region tagging file named `cicstags.yaml` or `cicstags.yml`. CICS provides three types of tags for you to classify regions: region type (e.g. TOR, AOR), region usage (e.g. Development, Sandpit, Production), and application (e.g. Banking, Travel). You can also define custom tags to meet your needs. Each tag contains matching criteria that identify a group of regions based on their APPLID, region user ID, or region job name.

CICS then identifies regions that match the criteria using the [matching algorithm](#) and tag them accordingly. You can then define health check exclusions (in the same file) or manage the regions based on the defined tags.

For instructions on how to configure the region tagging file, see [Setting up CICS region tags](#).

## Auditing installed resource

Messages about resource installation are written to a number of destinations, depending on how the resources are installed.

Table 51 on page 588 describes these destinations, followed by examples of the messages.

Table 51. Finding messages about installed resources		
Method of resource installation	Message	Destination
Install group list during installation	DFHSI1511I for each list that is installed	JESMSG LG
	Message(s) for each resource	MSGUSR
	DFHAM4893 for giving the group and the list for the preceding resource messages	TDQ CADL
Manually install group using CEDA (see “Example: messages from a CSD install of a transaction and a program by using CEDA install” on page 589)	DFHRDnnnn for the install of the resource	TDQ CRDI
	INSTALL ALL GROUP(group) . These messages do not have a message number.	TDQ CSDL
	Message(s) for each resource	MSGUSR
	DFHAM4893 giving the group	TDQ CADL
Manually install single resource using CEDA	DFHRDnnnn for the install of the resource	TDQ CRDI
	INSTALL resource GROUP(group). These messages do not have a message number.	TDQ CSDL
	Message(s) for each resource	MSGUSR
Application issues <b>EXEC CICS CREATE</b> resource	CREATE resourcetype(resource) options	TDQ CSDL
	Message for each resource	MSGUSR
Application issues <b>EXEC CICS CREATE</b> resource with option NOLOG	Message for each resource	MSGUSR
Install using CICSplex SM during installation		EYULOG
Manually install group using CICSplex SM		EYULOG
Manually install single resource using CICSplex SM		EYULOG

### Example: messages from a CSD install of a transaction and a program during CICS initialization

```
DFHXM0101 01/12/2020 09:57:33 IYK2ZDL1  USERA CSSY TRANSACTION definition entry for BR14 has
been added.
DFHPG0101 01/12/2020 09:57:33 IYK2ZDL1  USERA CSSY Resource definition for BR14 has been added.
```



## Example: messages from a CSD install of a transaction and a program by using CEDA install

```
DFHRD0101 01/12/2020 10:44:20 IYK2ZDL1 IYCUTC02 CRPTST3 CEDA INSTALL PROGRAM(BC14)
DFHRD0104 01/12/2020 10:44:20 IYK2ZDL1 IYCUTC02 CRPTST3 CEDA INSTALL TRANSACTION(BC14)
TC02      CEDA CRPTST3 01/12/20 10:44:20 INSTALL ALL GROUP(BR14CICS)
DFHXM0101 01/12/2020 10:44:20 IYK2ZDL1 IYCUTC02 CRPTST3 CEDA TRANSACTION definition entry for
BC14 has been added.
DFHPG0101 01/12/2020 10:44:20 IYK2ZDL1 IYCUTC02 CRPTST3 CEDA Resource definition for BC14 has
been added.
```

## Auditing SPI commands

Configuration defines the initial state of a CICS region but it can be changed by system programming interface (SPI) commands. RACF SMF type 80 records can be used to audit the commands that are issued by an operator but they do not identify what options were used on these commands and what was changed as a result.

### The DFHAP1900 message

The system programming interface commands SET, PERFORM, ENABLE, DISABLE, RESYNC, which can change resource definitions dynamically, are audited. When these commands are issued, message DFHAP1900 is written to the CADS transient data queue (TDQ) and contains information about the parameters that are used in the command.

All SPI commands are audited with message , except as follows:

- SET TERMINAL
- FEPI SET commands
- PERFORM SHUTDOWN (already recorded by message DFHTM1715)
- CREATE (already recorded by existing messages, unless the NOLOG option is used on this command)

CICS starts to audit the SPI commands after message DFHSI1517 is issued, indicating that control is given to CICS. When SPI auditing becomes available in the region, message DFHAP1901 is issued and written to the CADS TDQ. This configuration means that during system initialization, SPI commands that are issued during PLT processing are not audited.

When you use CEMT or CECI commands, some options might be added or changed. The audit message shows the command that was issued, which might be different from the command you entered.

When you issue operator commands with generic parameters by using CEMT, CICS Explorer, CICSplex SM WUI, or EXEC CPSM commands, each command is audited as if it was entered separately. For example, if you have 2000 programs and enter the command **CEMT SET PROGRAM(\*) ENABLE**, 2000 separate messages are logged.

If you are not interested in the messages, disable them by directing TDQ CADS to a dummy TDQ. See [Using dummy transient data queues](#).

### SPI commands that can be audited

The system programming interface commands **SET, PERFORM, ENABLE, DISABLE, RESYNC** can change resource definitions dynamically. An incorrect entry can cause the CICS system to fail. When diagnosing a problem, it is important to know whether resources were changed. System administrators and anyone who manages audit records can audit certain system programming interface commands which dynamically change system resources.

### Audit messages

When a system resource is changed by one of the audited system programming interface commands, a new message DFHAP1900 is written to a transient data queue CADS. The CADS transient data queue is an

indirect queue that is defined in the DFHDCTG group, which is part of DFHLIST. The messages are written in a human readable form.

The messages contain the following information:

- Time
- Application ID
- Netname
- Transaction identification
- User ID
- Details of the command, including attribute name and value
- RESP response code
- RESP2 response code

#### Example 1

The command **CEMT SET SYSTEM MAXTASKS(250)** is entered from terminal TC99. For a normal response, the following message is written to the CADS queue:

```
DFHAP1900 I 11/11/2011 11:11:11 IYK3ZC76 IYCWTC99 CNTEST7  
CEMT SET SYSTEM MAXTASKS(250) RESP(NORMAL) RESP2(0)
```

#### Example 2

The command **CECI SET FILE(TEMP) OPEN** is entered from terminal TC99. The response is: Open/close failed EIBRESP=+0000000012 EIBRESP2=+0000000018. The audit message is written as:

```
DFHAP1900 I 11/11/2011 11:11:11 IYK3ZC76 IYCWTC99 CNTEST7  
CECI SET FILE(TEMP) OPEN RESP(FILENOTFOUND) RESP2(18)
```

Where possible, the CVDA value is used in the message instead of the code to improve the readability of the audit messages.

#### Example 3

The command **CECI SET FILE(TEMP) ENABLESTATUS(ENABLED)**. The audit message is written as:

```
DFHAP1900 I 11/11/2011 11:11:11 IYK3ZC76 IYCWTC99 CNTEST7  
CECI SET FILE(TEMP) ENABLESTATUS(ENABLED) RESP(FILENOTFOUND) RESP2(18)
```

When you use CEMT, WUI or Explorer operator commands with generic parameters, each command is audited as if it was entered separately. For example, if you have 2000 programs and enter the command **CEMT SET PROGRAM(\*) ENABLE**, 2000 separate messages are logged. Similarly, if you enter the command **CEMT SET PROGRAM(\*) NEWCOPY** when you are not authorized for **SET PROGRAM**, 2000 RACF failure messages are logged. So many messages might flood the CSSL queue so audit messages should be redirected to another queue. As each command is logged as if it were entered separately, you can search the log for a single program name to aid problem determination.

When you use CEMT or CECI commands, some options can be added or changed. The audit message shows the command that was issued, which might be different than the command you entered.

**Note:** The audit message can be disabled by directing the messages to a dummy transient data queue. See [Using dummy transient data queues](#).

## User IDs in audit messages

The user ID displayed in audit messages is dependent upon the security that is active within the context in which the command is issued. If the command is issued under the control of CICSplex SM, you have several settings that affect which user ID is used, as illustrated in the following table:

Table 52. User IDs in audit messages		
HEYUPARM in CMAS	SIT parm in CICS region where request is initiated	User ID in audit message
SEC(YES)	SEC=YES	Authenticated user ID  <b>Note:</b> The authenticated user ID depends on how, and where, the request to issue the command was initiated, as illustrated in Table 53 on page 591.
SEC(YES)	SEC=NO	Default user ID for CMAS
SEC(NO)	SEC=NO	Default user ID for CICS Region where command is issued
SEC(NO)	SEC=YES	Invalid combination

Table 53. Authenticated user ID	
Where request is initiated	Authenticated user ID
WUI	User ID used to sign on to the WUI
CICSplex SM API Batch Job	User ID under which the Job connects to CICSplex SM, by default this is the user under which the job is run.
CICSplex SM API Application	User ID under which the task connects to CICSplex SM, by default this is the user under which the task is running in the CICS.
Region Explorer (CMCI)	User ID specified in the Connection Credentials.

**Note:** If security is not active in the WUI, users can log on through the Web User Interface by using any string value for a user ID. The default user ID of MAS or CMAS is displayed in the audit message, and therefore cannot be used to identify the user that entered the command.

For more information on CICS user security, see [How it works: Identification in CICS](#).

## SPI commands that are not audited

Some SPI commands are not audited:

- SET TERMINAL
- FEPI SET commands
- PERFORM SHUTDOWN (already handled by message DFHTM1715)
- CREATE (already recorded by existing messages)
- SET ASSOCIATION USERCORRDATA

## When CICS starts auditing SPI commands

Auditing of the SPI commands starts after message DFHSI1517 is issued, indicating that control is given to CICS. When SPI auditing becomes available in the region, message DFHAP1901 is issued, indicating that it is active.

This means that during system initialization, SPI commands that are issued during PLT processing are not audited.

## Auditing sign-on and sign-off

A user signs onto a CICS region through interfaces, such as a 3270 terminal, the CICS Client Management Interface (CMCI) or the (stabilized) CICSplex SM Web User Interface (WUI). RACF can log all sign-on and sign-off activity to SMF, including any invalid or unsuccessful sign-on attempts. You can use this information as an audit trail, to identify possible attempts to breach security, and to help with capacity planning. Recording the successful sign-on and sign-off activities establishes an audit trail of the access to particular systems by the terminal user population. This may also be useful for systems capacity planning, and generally constitutes a very modest portion of the information recorded to SMF.

- [“Finding information about sign-on and sign-off through a 3270 terminal” on page 592](#)
- [“Finding information about sign-on and sign-off through CMCI” on page 592](#)
- [“Finding information about sign-on and sign-off through CICSplex SM WUI” on page 592](#)
- [“Finding information about sign-on and sign-off through Liberty” on page 593](#)

### Finding information about sign-on and sign-off through a 3270 terminal

CICS uses its CSCS Transient Data Queue (TDQ) for security messages. Messages of interest to the security administrator for the CICS region are directed to this destination. In some instances, when security-related messages are directed to terminal users, corresponding messages are written to the CSCS TDQ: for example, when DFHCE3544 and DFHCE3545 messages are sent to terminal users, the corresponding messages DFHSN1118 and DFHSN1119 are sent to CSCS. The DFHSNnnnn messages include reason codes that indicate the exact nature of the invalid sign-on attempt.

This is an example of messages in the CSCS TDQ as a result of a successful user sign-on and sign-off from a 3270 terminal:

```
DFHSN1100 27/11/2020 16:15:34 IYK2ZDL1 CESN Signon at netname IYCWT126 by user CRPTST3 in group
TSOUSER is complete.

DFHSN1200 27/11/2020 16:55:58 IYK2ZDL1 Signoff at netname IYCWT126 by user CRPTST3 is complete.
25 transactions entered with 1
errors.
```

This is an example of messages in the CSCS TDQ as a result of a failed user sign-on from a 3270 terminal:

```
DFHXS1201 27/11/2020 16:40:11 IYK2ZDL1 The password supplied in the verification request for
userid CRPTST2 was invalid. This
occurred in transaction CESN when userid CICSUSER was signed on at netname
IYCWT126.

DFHSN1102 27/11/2020 16:40:11 IYK2ZDL1 Signon at netname IYCWT126 by user CRPTST2 has failed.
Password not
recognized.
```

This is an example of messages in the z/OS Security Console and job log as a result of a failed user sign-on from a terminal (3270):

```
16.40.11 JOB39558 ICH408I USER(CRPTST2 ) GROUP(TSOUSER ) NAME(USERA ) 624
624 LOGON/JOB INITIATION - INVALID PASSWORD ENTERED AT TERMINAL IYCWT126
```

### Finding information about sign-on and sign-off through CMCI

For information about CMCI, see [CICS management client interface \(CMCI\)](#).

### Finding information about sign-on and sign-off through CICSplex SM WUI

CICSplex SM WUI is stabilized.

## Finding information about sign-on and sign-off through Liberty

For more information, see [HTTP access logging](#).

### Finding authentication issues in CICS statistics

You can only properly interpret the logging of unsuccessful sign-on attempts and authentications by also recording successful sign-on and authentication instances. For example, if a user makes one or two unsuccessful attempts followed immediately by a successful sign-on, the unsuccessful attempts to sign on can be interpreted as the result of a mistake. However, several unsuccessful attempts for a variety of user IDs that occur within a short space of time, and without any subsequent successful activity being recorded, might be cause for a security concern and you should investigate further.

This is an example of messages

This information is summarized in CICS statistics.

```
Successful fastpath authentications :  
651  
Successful fullpath authentications :      212      Failed fullpath  
authentications . . :      4  
Successful kerberos authentications :      0      Failed kerberos  
authentications . . :      0  
Successful JWT creations . . . . :      0      Failed JWT  
creations . . . . . :      0  
Successful JWT authentications . . :      0      Failed JWT  
authentications . . . . :      0
```

## Auditing bind time security (LU6.2)

You can configure CICS to audit bind time security by setting the system initialization parameter XAPPC=YES and the BINDSECURITY(YES) option on the CONNECTION resource definition that defines the remote system.

For information about bind time security, see [Intercommunication security](#).

The following conditions are considered bind failures so they cause RACF® to write an SMF record and to issue a message:

- Session key does not match the partner's session key.
- Session segment is locked.
- Session segment expired.
- Session key is null.
- Session segment does not exist.
- Session segment retrieval was unsuccessful.
- Session bind was unsuccessful.

The following conditions are considered bind successes so they cause RACF to write an SMF record, but not to issue a message:

- Session is successfully bound.
- Session key expires in less than six days.

An SMF record is written if either of the following conditions is true:

- In RACF, the resource profile's audit option is set (AUDIT(ALL(READ))).
- In RACF, SETROPTS LOGOPTIONS(ALWAYS(APPCLU)) is set.

Two things happen when an SMF audit record is written:

- Message ICH700051 is sent to the user ID that is specified in the profile's NOTIFY option. (Specify the TSO user ID of a RACF administrator who is responsible for the APPCLU class.)
- The security console receives message ICH415I, which contains text similar to message ICH70005I.

## Auditing authorization

When a user requires access to a protected CICS resource and RACF denies the requested access, CICS provides messages.

### Category 1 transactions

Category 1 transactions are part of CICS and the CICS region user ID is the only ID configured to run Category 1 transactions. If any other user attempts to run a Category 1 transaction directly, the transaction abends with type AXS1. For these reasons, Category 1 transactions do not need to be authorized to run making them different from other transactions. Category 1 transaction security is secure as only the CICS region user ID needs to be defined and the risk of misconfiguration is minimized.

Category 1 and other transaction types are described in [Transactions in CICS](#).

The IBM Health Checker for z/OS validates key production region configuration parameters, including the CICS region user ID. For more information about how this is checked, see [Reference: Available CICS health checks supported by IBM Health Checker for z/OS](#).

### Information returned to the user about authorization issues

CICS issues an authorization message, such as DFHAC2033, to a terminal user or returns a “not authorized” return code to an application.

### CICS messages about authorization issues

CICS issues an authorization message DFHXS1111 to the CICS Transient Data Queue (TDQ).

```
DFHXS1111 26/09/95 15:34:01 CICS SYS1 Security violation by user JONES
at netname D2D1 for resource FLA32 in class TCICSTRN.
SAF codes are (X'000000008',X'000000008'). ESM codes are
(X'000000008',X'000000008').
```

This message reports that user ID JONES, signed on at VTAM terminal with netname D2D1, caused a security violation in requesting access to resource FLA32, which is in class TCICSTRN. There are reason and response codes from the System Authorization Facility (SAF) and from RACF.

Most CICS authorization messages also go to the CICS TDQ, except DFHIR and DFHZC messages that go to the CSMT TDQ.

### RACF messages about authorization issues

RACF® sends an ICH408I message to the CICS region's job log and to the z/OS security console .

```
ICH408I USER(JONES ) GROUP(DEPT60 ) NAME(M.M.JONES )
ICH408I FLA32 CL(FCICSFCT)
ICH408I INSUFFICIENT ACCESS AUTHORITY
ICH408I FROM F%* (G)
ICH408I ACCESS INTENT(UPDATE ) ACCESS
ALLOWED(READ )
```

This message reports that user ID JONES, a member of group DEPT60, whose name is M.M.JONES, had INSUFFICIENT ACCESS AUTHORITY to resource FLA32, which is in class FCICSFCT. The RACF profile protecting the resource is F%\* . (G) indicates that F%\* is a generic profile. The access that is attempted by user JONES was UPDATE, but the access allowed by RACF was READ. Therefore, user ID JONES was denied access.

If the transaction is defined to RACF with LOG(NONE), no ICH408I message is issued.

For a complete description of RACF message ICH408I, see [z/OS Security Server RACF Messages and Codes](#).

## CICSplex SM messages about authorization issues

CICSplex SM sends messages to the EYULOG.

## Liberty messages about authorization issues

If Liberty accesses CICS resources, through the JCICS or JCICSX APIs, messages are output to the standard CICS message logs. If the Liberty server itself denies the requested access, messages are output to the configured Liberty messages .log file location. Further RACF messages about authorization issues may also be issued to the region's job log and z/OS security console if SAF security is in use and the attribute **racRouteLog="ASIS"** is specified on the <saAuthorization/> element in server.xml.

## Impact of SECPRFX on authorization checks

If you specify the SECPRFX system initialization parameter, the resource profile that is checked is prefixed by the SECPRFX value. For example, if SECPRFX=DEV and the FILE PAYROLL is accessed, the check is against the profile DEV.PAYROLL.

## Information in CICS statistics about authorization issues

A summary of authorization attempts and failures is provided in CICS statistics

Successful resource authorizations . . :	11	867	Failed resource
Successful command authorizations . . :	567		Failed command
Successful surrogate authorizations . . :	51		Failed surrogate
Successful non-CICS authorizations . . :	0		Failed non-CICS
authorizations . . :	0		

## Auditing RACF

Auditing RACF is checking that RACF meets the installation's needs for access control and accountability. RACF provides functions for you to specify the information that it records and tools to help you format and analyze the logged security events. For information about the auditing capabilities that RACF provides, see [z/OS Security Server RACF Auditor's Guide](#).

## Auditing RACF profiles

**Recommendation:** When you define RACF profiles for CICS resources, you should use at least the default option of AUDIT(FAILURES). For high-value resources, consider using AUDIT(ALL) to record everyone who accessed this resource.

## Auditing RACF activity

The RACF SMF data unload utility (IRRADU00) helps you to monitor RACF-related activity during system operation. It creates a sequential file from the security relevant audit data. The sequential file can be used in several ways: viewed directly, used as input for installation-written programs, manipulated with sort/merge utilities, output to an XML-formatted file for viewing on a web browser, or uploaded to a database manager (for example, Db2) to process complex inquiries and create installation-tailored reports.

For more information, see [The RACF SMF data unload utility](#) in the z/OS product documentation.





---

## Chapter 31. Developing SAML applications

CICS supports Security Assertion Markup Language (SAML). You can develop CICS applications to process SAML tokens.

For example, you can write programs to:

- Extract a SAML token from an incoming message.
- Place a SAML token in an outbound request.
- Add attributes to a SAML token and re-sign the token.
- Create a SAML token.

For an overview of CICS support for SAML, see [How it works: SAML](#).

### Related information

[How it works: Authentication in CICS](#)

---

## Developing a SAML-aware initial program

You can write a program to extract a SAML token and link to DFHSAML to process the token.

As an alternative to using the web services support (see [Configuring a provider pipeline to use SAML tokens](#)), you can write your own SAML-aware initial program as a security front end to validate SAML tokens. Such a program can be useful when you are using messages that use HTTP, IBM MQ, or any other protocol, rather than a SOAP message.

The SAML-aware initial program extracts the SAML token from the message. It then puts the token into a character container, DFHSAML-TOKEN, in a channel. The program then links to the program DFHSAML with the channel to validate the token and extract the containers. For details of the containers, see [SAML support containers](#).

You can use either a user-defined channel or the transaction channel (DFHTRANSACTION). If you are using a user-defined channel, the containers are passed on LINK requests that explicitly pass that channel. If you are using the transaction channel, the containers are available throughout the transaction.

The application can use information in the containers, such as SAML attributes, by using **GET CONTAINER** commands. For a specific example of how you might use this information, see [“Pattern: developing a SAML-aware initial program” on page 597](#).

### Pattern: developing a SAML-aware initial program

SAML-aware programs can conform to common patterns. A typical pattern for an application is for the initial program to be SAML-aware. In this pattern, the program uses the information in the SAML assertion to make decisions before it runs appropriate parts of the application.

The information in the SAML assertion is in read-only containers. These containers are either in the channel that is passed to the program from the SOAP pipeline or, if the program is doing its own message handling and SAML validation, returned from the DFHSAML program.

An example of the processing that the SAML-aware program does is obtaining information from the attribute containers. It looks for an attribute name container (for example, DFHSAML-ATTRN001) and an attribute value in that container (for example, DFHSAML-A001V001). This attribute might be used by the program to represent the role or authority of the user and so allow the application to select which parts of the application are available to the caller.

The application might have to pass on information to other programs in the application. Because the containers are read-only, information can be passed securely by passing the channel to the next program, or transaction that uses the CHANNEL interface, on the LINK, XCTL, RETURN, or START commands.

The application might also have to audit the request so that the transaction can be associated with the original SAML token, and hence the user. To make this association, write a customer logging program, which writes the validated SAML token (in container DFHSAML-OUTTOKEN) or selected containers to a journal.

### **Pattern: reusing a validated SAML token**

You might want to validate a SAML token and later in the same transaction call a web service from a requester program and use the same token.

A validated SAML token is held in the DFHSAML-OUTTOKEN container. As this container is read-only, it cannot be moved between channels. To avoid having to reissue the validation request, and thus to improve performance, you can use the transaction channel, DFHTRANSACTION.

When you validate a SAML token from an incoming web service, code the **tran\_channel="yes"** attribute in the <sts\_authentication> element in the configuration file for your provider pipeline. This attribute specifies that the SAML assertions are copied from the output containers into containers in the DFHTRANSACTION channel.

To reuse the validated SAML token in a web service, code the **tran\_channel="yes"** attribute in the <sts\_authentication> element in the configuration file for the requester pipeline that is used by the web service.

## **Developing a program that uses a validated SAML token in an outbound request**

---

You might want your application to use the SAML token in an outbound request. You can configure web services to add the SAML token to an outbound request.

For detailed instructions and an example, see [Configuring a requester pipeline to use SAML tokens](#).

For an application to insert a SAML token in a web service request SOAP message, the token must first be validated. Optionally, the application can add attributes to the token after validating it and before invoking the web service. For more information, see [“Developing a program that creates or augments SAML tokens”](#) on page 598.

The SAML token is stored in the read-only container DFHSAML-OUTTOKEN.

### **Related concepts**

[How it works: SAML](#)

### **Related tasks**

[“Developing a program that creates or augments SAML tokens”](#) on page 598

You can develop a CICS application or a web service application to create a SAML token or to add attributes to a SAML token and to re-sign it.

[Configuring CICS for SAML](#)

### **Related information**

[“Developing a SAML-aware initial program”](#) on page 597

You can write a program to extract a SAML token and link to DFHSAML to process the token.

## **Developing a program that creates or augments SAML tokens**

---

You can develop a CICS application or a web service application to create a SAML token or to add attributes to a SAML token and to re-sign it.

### **About this task**

You can use CICS SAML support to add attributes to SAML tokens and re-sign the request with the certificate specified in the CICS STS configuration file. The SAML token might be received from an external

sender or created from a template. The application that you develop can be either a CICS application or a web service application.



**Attention:** Create, augment, and re-sign tokens only on a region where all application code that participates in the augmenting of the token is trusted by other members of the federation.

## Procedure

1. Add the attributes by creating the following containers in the same channel that was used to validate the original SAML token.

**Note:** You must have validated the token before you can modify it. The validated token is contained in the DFHSAML-OUTTOKEN container.

- a) Put the attribute name into container DFHSAML-ATTRN *aaa* , where *aaa* are three uppercase alphanumeric characters.  
For example:

```
EXEC CICS PUT CONTAINER('DFHSAML-ATTRNORG')  
CHANNEL('SAML-CHANNEL') FROM('title')
```

- b) Optional: Put the attribute name space into container DFHSAML-ATTRS *aaa* , where *aaa* are the same characters as you used for the attribute name container.

This step is not required for SAML version 2.0.

- c) Optional: Put the attribute friendly name into container DFHSAML-ATTRY *aaa* , where *aaa* are the same characters as you used for the attribute name container.

For example:

```
EXEC CICS PUT CONTAINER('DFHSAML-ATTRYORG')  
CHANNEL('SAML-CHANNEL') FROM('eduPersonAffiliation')
```

- d) Optional: Put the attribute format into container DFHSAML-ATTRF *aaa* , where *aaa* are the same characters as you used for the attribute name container.

For example:

```
EXEC CICS PUT CONTAINER('DFHSAML-ATTRNORG')  
CHANNEL('SAML-CHANNEL')  
FROM('urn:oasis:names:tc:SAML:2.0:attrname-format:uri')
```

- e) Optional: Put one or more attribute values into containers DFHSAML-A *aaa* N *bbb* , where *aaa* are the same characters as you used for the attribute name container and *bbb* are three uppercase alphanumeric characters.

For example:

```
EXEC CICS PUT CONTAINER('DFHSAML-AORGV001')  
CHANNEL('SAML-CHANNEL') FROM('staff')  
EXEC CICS PUT CONTAINER('DFHSAML-AORGV002') CHANNEL('SAML-CHANNEL')  
FROM('employee')
```

2. Create the token in either of the following ways:

- Put the SAML -ISSUE value in the DFHSAML-FUNCTION container and link to the linkable interface DFHSAML, which creates the new token. By default, the token is re-signed using the signature options that are specified in the STS configuration file. If no signature is required, the application can create a DFHSAML-SIGNED container with the SAML-IGNORED option specified before it calls DFHSAML. If an <issuer> is specified in the STS configuration file, its value is used in the new SAML token.
- Invoke a web service. If the requester pipeline associated with the web service is configured for SAML, it automatically adds attributes to the original token and creates a new one. By default, the pipeline re-signs the SAML token by using the signature options that are specified in the STS configuration file. If no signature is required, set the requester pipeline configuration option token\_signature to no. If an <issuer> is specified in the STS configuration file, its value is used in

the new SAML token. In addition to creating a new SAML token, the requester pipeline also inserts the SAML token in the outbound web service request.

---

## Chapter 32. Troubleshooting security

These topics explain how to determine the cause of security-related problems in CICS.

### **Understanding the basics of CICS security**

[How it works: Identification in CICS](#)

[How it works: Authentication in CICS](#)

[How it works: Authorization in CICS](#)

[How it works: Confidentiality and integrity in CICS](#)

[How it works: Auditing in CICS](#)

[How it works: Securing CICS with RACF](#)

---

## Problem determination in a CICS-RACF security environment

This topic provides information to help you find the causes of access authority problems.

### **What's your problem?**

- [“Access is denied incorrectly” on page 601](#)
- [“Access is allowed incorrectly” on page 606](#)

### **Access is denied incorrectly**

When a user requires access to a protected resource (such as a CICS transaction) and RACF denies the requested access, you will often have to analyze the problem before deciding what action to take.

The basic points to ensure are that:

- CICS is using RACF for this particular kind of resource.
- You know which profile RACF is using to check the user's authority.
- You know which user ID CICS is using for the authorization check.

For each security violation, up to three messages are issued:

- CICS issues an authority message to the terminal user (or returns a "not authorized" return code to an application).
- CICS sends a message DFHXS1111 to the CICS transient data destination.
- RACF sends an ICH408I message to the CICS region's job log and to the security console.

For a brief description of message ICH4081, see [“RACF message ICH408I: an explanation with example” on page 605](#). For complete descriptions of this and all other RACF messages, see [z/OS Security Server RACF Messages and Codes](#).

- CICS sends a message DFHXS1117 in addition to DFHXS1111 or ICH4081. It provides information about the originator of the request that had the security definition.

If message ICH408I is issued for an authorization failure, RACF is active. The message text itself indicates the user ID for which the authorization check was done and the name of the RACF profile that was used for the check.

When message ICH408I is issued because of a CICS-originated authorization check, the RACF sends the ICH408I message to the CICS region's job log. Most CICS authorization messages also go to the CICS transient data queue, except DFHIR and DFHZC messages, which go to the CSMT transient data queue.

**Note:** You can use the CICS-supplied message domain global user exit, XMEOUT, to reroute CICS-issued authorization messages. (For example, you can send them to the same console as the ICH408I messages.) For programming information about using XMEOUT, see [Global user exit points \(by function\)](#).

If no profile exists for a particular resource, RACF returns a "profile not found" indication to CICS. CICS issues message DFHXS1111 with a SAF return code of X'00000004' and an ESM code of X'00000000'. **No**

**ICH408I message is issued in this case.** The RLIST command issues a message stating that no profile was found.

**Note:**

- The RLIST command shows the profile as it exists in the RACF database, which might not necessarily be the same as the in-storage copy that CICS uses.
- When you have determined which RACF profile is denying access, see the [z/OS Security Server RACF Security Administrator's Guide](#) for a description of authorization checking for RACF-protected resources. The following describe some further steps to take in resolving "access denied" problems.

Alternatively, you might wish to use the security request recording facility. For more information, see [Diagnosing access issues with security request recording \(SRR\)](#).

## Diagnosis checklist

- [“Is CICS using RACF for this particular kind of resource?” on page 602](#)
- [“Which profile is RACF using?” on page 602](#)
- [“Which user ID did CICS supply for the authorization check?” on page 603](#)
- [“Which profile is used to protect the resource?” on page 603](#)

## Is CICS using RACF for this particular kind of resource?

Make sure that CICS is using an external security manager (ESM) such as RACF. If it is not using an ESM, it issues message DFHXS1102.

- Is CICS using RACF?
- Is security checking done for the particular general resource class? Message DFHXS1105 tells you if the class named on an *Xname* parameter has been initialized.

**Note:** If message DFHXS1105 is not there, ensure that the **SEC=YES** system initialization parameter is specified for the region.

Check the appropriate CICS initialization parameter for the resource. For example, for transactions this is the **XTRAN** parameter.

## Which profile is RACF using?

Check the RACF message ICH408I for the name of the profile that RACF used.

If CICS prefixing is in effect for the CICS region involved, the prefix specified is used as the first qualifier of RACF resource profiles (or member names).

- Make sure that you have specified the correct prefix as part of resource profile names (on the RDEFINE command) and as member names on the ADDMEM operand.
- Make sure the CICS job is running under the correct prefix if **SECPRFX=prefix** or **SECPRFX=YES** is specified. The name of the resource in message ICH408I includes the prefix if one of these parameter values is specified in the system initialization parameters.
- Make sure that an installation-written SAF exit is not changing the effective userid under which the CICS region is running.

## Is CICS using current copies of the RACF resource profiles?

If you have created, changed, or deleted a resource profile, the in-storage profile does not reflect the change until one of the following is completed:

- SETROPTS GENERIC(*classname*) REFRESH (a generic profile has been changed).
- SETROPTS RACLIST(*classname*) REFRESH

For more information about refreshing resource profiles, see [Caching of RACF classes and their profiles](#).

You can use the TSO RLIST command to determine which profile (or profiles) protect the resource. See [“Which profile is used to protect the resource?” on page 603](#)

## Which user ID did CICS supply for the authorization check?

Check to see if the user reporting the problem has signed on to CICS.

If the user has not signed on to CICS, one of the following could be occurring:

- If you are using preset-terminal security, the authorization could be related to that terminal's user ID.
- The user could be trying to operate as the CICS default user (without signing on to CICS).
- If the transaction was initiated by a START command, the userid could be inherited from the transaction issuing the START, or specified on the START command itself.
- If the transaction was initiated as the trigger transaction associated with a transient data queue, the userid could have been specified in the transient data destination definition for the queue.
- If the program is running as a PLTPI program, the userid could be specified in the **PLTPIUSR** system initialization parameter.

**Note:** RACF message ICH408I identifies the userid, as supplied by CICS to RACF, for which the authorization failed.

For help in identifying the user, see [How it works: Identification in CICS](#).

## Which profile is used to protect the resource?

If you are using generic profiles (and you are **not** using resource group profiles, that is, member lists), only the most specific profile is used.

### Example:

Assume that the following profiles exist:

```
**  
C*  
CE*  
CEDA
```

CEDA is the profile that is used to control access to the CEDA transaction. If you delete profile CEDA and refresh the in-storage copies, CE\* is used.

Note that this assumes CICS prefixing is not used and generic profile checking **is** used. (That is, that the RACF command SETROPTS GENERIC(*class\_name*) has been issued for the class.)

If resource group profiles have been defined in the relevant class (for example, profiles in the GCICSTRN class), it is possible that more than one profile is used in determining a user's access. To determine which profiles protect the resource, enter the RLIST command with the RESGROUP operand. Be sure to specify the **member class** on the RLIST command. For example:

```
RLIST TCICSTRN transaction-name RESGROUP
```

If prefixing is in effect for this CICS region, include the prefix on the resource name specified on the RLIST command:

```
RLIST TCICSTRN prefix.transaction-name RESGROUP
```

Note that these examples use the member class TCICSTRN, not the resource group class GCICSTRN.

The AUDITOR attribute enables users to list all profiles that are defined, but does not authorize them to change those profiles. It is recommended that you give AUDITOR access to system programmers who need to see all profiles (for example, those who are doing problem determination) instead of system-SPECIAL access.

As a result of issuing RLIST with RESGROUP, you might see the following information:

- A brief listing of the resource group profile that protects the resource. See [Figure 171](#) on page 604.
- A slightly longer listing showing the member profile as well as the resource group profile. See [Figure 172](#) on page 604.
- A “profile not found” message, if no profile is found that protects the resource. See [Figure 173](#) on page 604.
- A “not authorized” message, if If a profile exists, but you are not authorized to list the profile. See [Figure 174](#) on page 605.

```
rlist tcicstrn cemt resgroup
CLASS      NAME
-----
TCICSTRN   CEMT
GROUP CLASS NAME
-----
GCICSTRN
RESOURCE GROUPS
-----
CAT2
```

*Figure 171. Output of RLIST command with RESGROUP: resource group profile*

```
rlist tcicstrn cemt resgroup
CLASS      NAME
-----
TCICSTRN   CEMT
GROUP CLASS NAME
-----
GCICSTRN
RESOURCE GROUPS
-----
CICSCAT2A
LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----
  00    PUB01      NONE              ALTER        NO
INSTALLATION DATA
-----
NONE
APPLICATION DATA
-----
NONE
AUDITING
-----
FAILURES(READ)
NOTIFY
-----
NO USER TO BE NOTIFIED
```

*Figure 172. Output of RLIST command with RESGROUP: several profiles*

**Note:** When you are using resource group profiles, more than one profile might be used at the same time. If the resource is protected by more than one profile, you are strongly urged to delete all other occurrences of the resource name. Use the DELMEM operand on the RALTER command to remove the resource name from existing resource group profiles. Use the RDELETE command **with care** to delete profiles from the member class.

```
rlist tcicstrn dfhcicsm.cemt resgroup
ICH13003I DFHCICSM.CEMT NOT FOUND
```

*Figure 173. Output of RLIST command with RESGROUP: profile not found*



**Note:** If you get the “profile not found” message, make sure that generic profile processing is in effect for the specified class. (SETOPTS LIST will show this.) If, indeed, no profile exists, create a suitable profile and ensure that the appropriate users and groups have access.

```
rlist tcicstrn dfhcicsm.cemt resgroup
ICH13002I NOT AUTHORIZED TO LIST DFH*
```

*Figure 174. Output of RLIST command with RESGROUP: authorization message*

The “not authorized” message identifies the name of the profile preventing you from having access. You can ask the RACF security administrator (who has the system-SPECIAL attribute and can therefore list the profile) to investigate the problem.

Some possible solutions are as follows:

- The profile is not needed and can be deleted.
- You can be made OWNER of the profile.
- A more specific generic profile can be created, and you or your group can be made OWNER of the new profile.
- If the profile is a discrete profile, you can be given ALTER access to the profile.
- You can be assigned the AUDITOR attribute.

For a description of the authority needed to list a general resource profile, see the description of the RLIST command in the [z/OS Security Server RACF Command Language Reference](#).

## RACF message ICH408I: an explanation with example

The first line of message ICH408I identifies a user who had an authorization problem. The other lines of the message describe the request the user was issuing and the reason for the failure.

Consider the following example:

```
ICH408I  USER(JONES  ) GROUP(DEPT60  ) NAME(M.M.JONES  )
ICH408I  FLA32 CL(FCICSFCT)
ICH408I  INSUFFICIENT ACCESS AUTHORITY
ICH408I  FROM F%* (G)
ICH408I  ACCESS INTENT(UPDATE)  ACCESS ALLOWED(READ)
```

This message can be interpreted as follows:

User JONES, a member of group DEPT60, whose name is M.M.JONES, had INSUFFICIENT ACCESS AUTHORITY to resource FLA32, which is in class FCICSFCT.

The RACF profile protecting the resource is F%\*. "(G)" indicates that F%\* is a generic profile.

The access attempted by user JONES was UPDATE, but the access allowed by RACF was READ. Therefore, user JONES was denied access.

**Note:** If the class shown is a resource group class, the profile might be in the class shown or in the related member class. For example, if GCICSTRN appears, check TCICSTRN also. If HCICSFCT appears, check FCICSFCT also. For a list of all the IBM-supplied class names, see [Summary of RACF classes for CICS resources](#). For a list of the installation-defined class names that are in use on your installation, see your RACF system programmer, or issue the SETOPTS LIST command.

A DFHXS1111 message would also be issued for this access attempt:

```
DFHXS1111 26/09/95 15:34:01 CICS SYS1 Security violation by user JONES
           at netname D2D1 for resource FLA32 in class
           TCICSTRN. SAF codes are (X'00000008',X'00000000'). ESM codes
           are (X'00000008',X'00000000').
```

The SAF and ESM codes come from RACROUTE REQUEST=AUTH.

To find the cause of the violation, issue the RLIST command with AUTHUSER specified to list the profile indicated in the ICH408I message. The AUTHUSER operand displays the access list, as shown in [Figure 175 on page 606](#).

```

rlist fcicsfct f%a* authuser
CLASS      NAME
-----
FCICSFCT   F%a* (G)
GROUP CLASS NAME
-----
HCICSFCT
LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----
00     CICSADM      NONE              ALTER        NO

USER      ACCESS
-----
DEPTA     UPDATE
JONES     READ
GROUPX    NONE
SYSPROG   ALTER

```

Figure 175. Requesting a display of the access list

In this profile, user JONES has an explicit entry in the access list. If the userid itself does not appear in the access list, check for one of JONES's connect groups. To list the groups to which JONES is connected, issue LISTUSER JONES Other specifications in the profile (such as security level or security category) might cause access to be denied. For a complete description, see the [z/OS Security Server RACF Security Administrator's Guide](#).

**Note:** If NOTIFY(CICSADM) were specified in this profile, TSO userid CICSADM would receive immediate notification of failed attempts to access resources protected by the profile.

## Related information

For a complete description of RACF message ICH408I, see [z/OS Security Server RACF Messages and Codes](#).

## Access is allowed incorrectly

There could be many reasons why a user might have access to a protected resource, even when you think that the user should **not** have that access. Here are some checks that you can make to investigate this kind of situation.

- Confirm which userid the user is signed on as. (Make sure the user has signed on and is not acting as the CICS default user.) You can ask the user to sign off, then sign on again. You can also ask the user to issue **EXEC CICS ASSIGN** or **EXEC CICS INQUIRE TERMINAL**, which can be issued with the CECI transaction (or a user-written transaction).
- Make sure that the SEC system initialization parameter is SEC=YES for the CICS region the user is signed on to.  
If SEC=NO is specified, users can access any resource.
- If the user is running a transaction that communicates with other regions such as application-owning regions (AORs) or file-owning regions (FORs), make sure that the SEC system initialization parameter is SEC=YES for those regions.
- Is prefixing correct?
  - Has the CICS JOB been submitted by the correct USER?
  - Is SECPRFX set correctly?
  - Has an installation-written SAF exit been used to return a different CICS region userid when RACROUTE=EXTRACT has been specified?

- Depending on the resource, make sure that RESSEC(YES) is specified for each transaction that might access that resource.
- Is the appropriate *Xname* CICS system initialization parameter correctly set?  
For example, if it is a file control request, is XFCT=YES or XFCT=*value* specified? If the *Xname* parameter specifies a value other than YES or NO, does the value show the correct installation-defined class name?
- Is the transaction exempt from transaction security (that is, defined as a *Category 3* transaction)? (For information on transactions that are defined in this way, see [Transactions in CICS](#).)
- Does the transaction have the correct RESSEC and CMDSEC options?
- Check that the RESSEC setting on the MIRROR transaction is correct.
- If the resource is temporary storage, check that the SECURITY attribute on the TSMODEL resource definition is set to YES.
- If intersystem communication is involved, check the following:
  - Is a SECURITY REBUILD required (on this or on the remote system)?
  - If ATTACHSEC=LOCAL is specified, does the SECURITYNAME userid have access to the resource?
  - Is ATTACHSEC=IDENTIFY specified?
  - Is link security being bypassed because the link userid matches the local region userid?
  - Is the remote system using the same RACF database?
- Do you have any RACF installation exits?
- To check the profile that you think protects the resource, use the checklist provided in the [z/OS Security Server RACF Security Administrator's Guide](#).
- To identify the profile that is being used, use the RACF SMF 80 record. This record includes the user ID, the resource, and the profile. For more information, see [Record type 80: RACF processing record](#).

## Diagnosing access issues with security request recording (SRR)

Security request recording (SRR) collects trace data about security settings in CICS regions by recording security checks conducted by one or more requests. You can use it to diagnose complex security problems.

### Before you begin

You must have set up SRR. For more information, see [Configuring security request recording \(SRR\)](#).

**Important:** Security request recordings do not persist over a CICS restart. If the CICS region is restarted, you need to rerun any recordings that you require.

### About this task

Use SRR to resolve security issues with the following complexities:

- The application that is being investigated might traverse multiple regions, multiple LPARs, or even multiple sysplexes.
- There might not be any messages that are associated with the reason that causes a security failure.
- The application might grant access to something when it mustn't.

For more information about SRR, see [Diagnostic tools and information](#).

### Procedure

1. Identify as much information about how the request came into CICS.
  - a. Do you know the APPLID for the origin region (ODAPPLID)?
  - b. Do you know the client IP address (ODCLNTIPADDR)?

- c. Do you know the term ID (ODFACILNAME)?
  - d. Do you know the LU name (ODLUNAME)?
  - e. Do you know the server port of the request (ODSERVERPORT)?
  - f. Do you know the TCPIP SERVICE handling the request (ODTCPIPS)?
  - g. Do you know the transaction ID (ODTRANSID)?
  - h. Do you know the user ID (ODUSERID)?
2. Identify the CICS region, or possible CICS regions that the request came into. There might be more than one region if you are port sharing, or by using some other form of load balancing.
  3. You can use either CICS Explorer or the SPI command to invoke SRR and control the types of security requests that are to be recorded.
    - To use CICS Explorer, see [Checking that a CICS security configuration example is working by using the SRR](#)
    - Alternatively, you can write a program that uses the **SET SECRECORDING** SPI command.
  4. Edit the DFH\$SRRP JCL.

You need to copy and customize the JCL to print the log stream. The JCL is member DFH\$SRRP in the SDFHSAMP library. You set the following variables:

- a. HLQ to the high-level qualifier of your CICS libraries.
- b. CSV is the file name to which the CSV output is written.
- c. LOGSTRM to the name of the common log stream used by all of the CICS regions. If you are using multiple log streams, such as one per LPAR, then you need to modify the JCL. Modify the JCL by concatenating DD statements for SYSUT1 in the JUPCOPY step, and naming the log streams explicitly on these DD statements.
- d. You can use parameters for the DFHSRR program in the SYSIN data set to restrict how much data is printed from the log stream. All records are printed when you do not specify any parameters in the SYSIN data set. Parameters for the DFHSRR program in the SYSIN data set must start in column one. Wildcard characters are not supported in parameters.

Specify one of the following parameters to restrict what is printed:

**MATCHID=\***

Prints all records and is the default when nothing is specified.

**MATCHID=matchid**

Where *matchid* is the value of *matchid* that is used on a SET or INQUIRE SECRECORDING(*matchid*) command. Printed output is limited to records for the specified *matchid*.

**ODSTARTTIME=odstarttime**

When *odstarttime* is copy and pasted from a previous run, the value of a start time that matches *odstarttime* for a single request that is based on its start time. Use the same format as *odstarttime* in a SUMMARY report as shown in [Example output of security request recording \(SRR\)](#). An example value is 2022-04-05 16:04:51.818905. If you have a CSV file from a previous run of DFHSRR and you can open the CSV file on zFS as a spreadsheet, you can select an *odstarttime* cell for a record, then copy and paste into the SYSIN data set. Printed output is limited to the single request based on its start time.

Only one value is allowed. Any line that starts with a \* is ignored.

For this example, update the SYSIN with the *matchid* and then submit the job.

5. Look at the SUMMARY output from the job, which shows the request or requests that were recorded.
6. If you have multiple requests that matched, see whether you are able to identify the specific problem request. If you can, then copy the *origin time*, update the SYSIN with this value, and rerun the DFH\$SRRP JCL.
7. Open the CSV file on zFS by using a spreadsheet. If your tools for editing zFS don't have this option, you first need to download the file.

8. Look through the requests to identify where the security checks did not work as expected. The information in the spreadsheet provides you all the identities for all security checks for a request, and why the requests worked or failed.

**Related information**

[Security request offline utility program \(DFHSRP\)](#)

**Example output of security request recording (SRR)**

This example shows output of security request recording (SRR) used to trace a transaction that uses distributed program links (DPL) to run transactions in other regions.

For this example, a user runs transaction SRR0 in region IYK2ZDL1. The transaction uses distributed program links to regions IYK2ZDL2 and IYK2ZDL3. The linked programs then issue various start requests. For more information about distributed program links, see [CICS distributed program link](#).

The example SUMMARY output is generated by issuing this command:

```
SET SECRECORDING(DEMO) ADD MAX(9) ODTRANSID(SRR0)
```

**Example SUMMARY output**

This is the resulting SUMMARY output:

```

*****
* Match ID      ODtranid  ODTaskid  ODApplid  ODStartTime  *
* DEMO 1       SRR0      00250    IYK2ZDL1  2021-12-20 12:41:36.902298 * 2
*****

Applid IYK2ZDL1
-----

Transaction ID.....: SRR0      Transaction number.....: 00250
Successful security requests.....: 2      Failed security requests.....: 1
Complete record of requests.....: Yes

Region settings 3
-----

Security active.....: Yes      Region user id.....: USERA
Default user id.....: CICSUSER  PLT user id.....: USERA
Kerberos user id.....:          XUSER.....: Yes
Resource security.....: Asis    Command security.....: Asis
Security prefix.....:          EMS exits.....: Noinstln
XHFS.....: Yes      XPTKT.....: Yes
PSBCHK.....: No

Region classes 4
-----

XAPPC.....:          XCMD.....: CCICSCMD
XDB2ENT.....:          XDCT.....: DCICSDCT
XFCT.....: FCICSFCT  XJCT.....: JCICSJCT
XPCT.....: ACICSPCT  XPPT.....: MCICSPPT
XTST.....: SCICSTST  XTRAN.....: TCICSTRN
XRES.....: RCICSRES

Applid IYK2ZDL2
-----

Transaction ID.....: CSMI      Transaction number.....: 00244
Successful security requests.....: 9      Failed security requests.....: 1
Complete record of requests.....: Yes

Transaction ID.....: SRR1      Transaction number.....: 00245
Successful security requests.....: 2      Failed security requests.....: 0
Complete record of requests.....: Yes

Transaction ID.....: SRR2      Transaction number.....: 00246
Successful security requests.....: 2      Failed security requests.....: 0
Complete record of requests.....: Yes

Region settings 3
-----

Security active.....: Yes      Region user id.....: USERA
Default user id.....: CICSUSER  PLT user id.....: USERA
Kerberos user id.....:          XUSER.....: Yes
Resource security.....: Asis    Command security.....: Asis
Security prefix.....:          EMS exits.....: Noinstln
XHFS.....: Yes      XPTKT.....: Yes
PSBCHK.....: No

Region classes 4
-----

XAPPC.....:          XCMD.....: CCICSCMD
XDB2ENT.....:          XDCT.....: DCICSDCT
XFCT.....: FCICSFCT  XJCT.....: JCICSJCT
XPCT.....: ACICSPCT  XPPT.....: MCICSPPT
XTST.....: SCICSTST  XTRAN.....: TCICSTRN
XRES.....: RCICSRES

Applid IYK2ZDL3
-----

Transaction ID.....: CSMI      Transaction number.....: 00244
Successful security requests.....: 9      Failed security requests.....: 1
Complete record of requests.....: Yes

Transaction ID.....: SRR1      Transaction number.....: 00245
Successful security requests.....: 2      Failed security requests.....: 0
Complete record of requests.....: Yes

Transaction ID.....: SRR2      Transaction number.....: 00246
Successful security requests.....: 2      Failed security requests.....: 0
Complete record of requests.....: Yes

Transaction ID.....: SRR3      Transaction number.....: 00247
Successful security requests.....: 2      Failed security requests.....: 0
Complete record of requests.....: Yes

Region settings 3
-----

Security active.....: Yes      Region user id.....: USERA
Default user id.....: CICSUSER  PLT user id.....: USERA
Kerberos user id.....:          XUSER.....: No
Resource security.....: Asis    Command security.....: Asis
Security prefix.....:          EMS exits.....: Noinstln
XHFS.....: Yes      XPTKT.....: Yes
PSBCHK.....: No

Region classes 4
-----

XAPPC.....:          XCMD.....: CCICSCMD
XDB2ENT.....:          XDCT.....: DCICSDCT
XFCT.....: FCICSFCT  XJCT.....: JCICSJCT
XPCT.....: ACICSPCT  XPPT.....: M1DEV PPT
XTST.....: SCICSTST  XTRAN.....: TCICSTRN
XRES.....: RCICSRES

*****
* Record counts *
*****

SRR records read.....: 64
SRR records selected.....: 8

```

**1** This request was run by specifying the *matchid* of DEMO in the SYSIN parameters for program DFHSRR on the DFH\$SRRP JCL.

```
//SYSIN DD *
MATCHID=DEMO
/*
```

**2** One request matched. The line shows the origin data for the task.

**3** The request ran in three regions. Shown are the SIT parameters for security and other security information for these regions.

**4** Shown are the RACF classes that are used by these regions. Most options have generic and member classes, only one is listed.

## Example SRR spreadsheet

Figure 176 on page 611 shows the spreadsheet (CSV file) that the process creates:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	matchid	odapplid	odtransid	odtaskid	odstarttime	applid	transid	transnum	userid	link_userid	resource_type	profile	class	access	response	reason	cal_resp	cal_rtn	ecm_resp	ecm_rtn	failing_userid	diagnostics	
2	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSATTACH	SRR0	TCICSTRN	READ	OK								X'00800000'
3	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000C	MCICSPPT	READ	OK	RESSEC(N)	5	0	0	0	0		X'00800000'
4	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Query Security	DUMIMY	FACILITY	NONE	EXCEPTION	NOTAUTHNOLOG	1	8	0	8	0		X'00010000'
5	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSATTACH	C5MI	TCICSTRN	READ	OK								X'00800000'
6	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000B		READ	OK	XFPTRNO							
7	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000C		READ	OK	XFPTRNO							
8	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR1	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
9	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	OK			0	0	0	0		X'00800000'
10	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR3	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
11	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	OK			0	0	0	0		X'00800000'
12	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR3	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
13	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	EXCEPTION	NOTAUTH	2	8	0	8	0	CRPTST2	X'00E10000'
14	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000B		READ	OK	XFPTRNO							
15	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR1	245	CRFTST2		TRANSATTACH	SRR1	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
16	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR1	245	CRFTST2		PROGRAM	SRR000C		READ	OK	RESSEC(N)							
17	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR2	246	CRFTST2		TRANSATTACH	SRR2	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
18	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR2	246	CRFTST2		PROGRAM	SRR000C		READ	OK	RESSEC(N)							
19	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSATTACH	C5MI	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
20	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000B		READ	OK			0	0	0	0		X'00800000'
21	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000C		READ	OK			0	0	0	0		X'00800000'
22	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR1	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
23	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	OK	XUSER-NO	3	0	0	0	0		X'00800000'
24	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR2	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
25	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	OK	XUSER-NO			0	0	0		X'00800000'
26	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		TRANSACTION	SRR3	ACICSPCT	READ	OK			0	0	0	0		X'00800000'
27	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		Surrogate Check	CRPTST2.DFH\$START	SURROGAT	READ	OK	XUSER-NO			0	0	0		X'00800000'
28	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR0	250	CRFTST2		PROGRAM	SRR000B		READ	EXCEPTION	NOTAUTH	8	0	8	0	CRPTST2		X'00E10000'
29	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR1	245	CRFTST2		TRANSATTACH	SRR1	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
30	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR1	245	CRFTST2		PROGRAM	SRR000C		READ	OK	RESSEC(N)							X'00800000'
31	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR2	246	CRFTST2		TRANSATTACH	SRR2	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
32	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR2	246	CRFTST2		PROGRAM	SRR000C		READ	OK	RESSEC(N)							X'00800000'
33	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR3	247	CRFTST3		TRANSATTACH	SRR3	TCICSTRN	READ	OK			0	0	0	0		X'00800000'
34	DEMO	IFYZ2DL1	SRR0	250	2011-12-20 12:41:36.902298	IFYZ2DL1	SRR3	247	CRFTST3		PROGRAM	SRR000C		READ	OK	RESSEC(N)							X'00800000'

Figure 176. Example SRR spreadsheet

The columns in the example SRR spreadsheet are explained as follows:

Table 54. Description of columns in SRR spreadsheet		
Column number	Column name	Description
A	matchid	If you generate the output without using any <b>SYSIN</b> parameters, the <i>matchid</i> column can be used to identify trace entries generated from a specific <b>SET SECRECORDING</b> command. In this example, the operator issued <b>SET SECRECORDING( ' DEMO ' )</b> so that all entries are associated with that <i>matchid</i> .
B	odapplid	The APPLID of the CICS region where the entry point of the request was.
C	odtransid	The initial transaction of the entry point of the request.
D	odtaskid	The initial task number of the entry point of the request.
E	odstarttime	The <i>odstarttime</i> column can be used to link together all the tasks from a specific request. In this example, all the entries are associated with a single <i>odstarttime</i> even though you have multiple tasks on multiple regions. All originated from the original task identified in the <i>odtaskid</i> column (Column D) on region <i>odapplid</i> (Column B). In the CSV file, the <i>odstarttime</i> column data is suffixed with a space so that it formats correctly in a spreadsheet.
F	applid	The APPLID of the CICS region in which the security request was issued.

Table 54. Description of columns in SRR spreadsheet (continued)

Column number	Column name	Description
G	transid	The transaction issuing the security request.
H	trannum	The task number issuing the security request.
I	userid	The user ID running the task that issued the security request.
J	link_userid	The link user ID that was associated with the task issuing the security request, if applicable.
L	resource_type	The resource type being checked or the security request type.
M	profile	The profile of the security that was checked. This is the name of the resource.
N	class	The security class that was being checked. This is the generic class name for CICS classes.
O	access	The access that was required in the security check.
P	response	The XSRC domain call response to the security check. This is usually OK or EXCEPTION.
Q	reason	SRR reason codes that indicate authorization status with reasons. For a complete list of SRR reason codes, see <a href="#">SRR reason codes</a> .
R	saf_resp	The SAF response code. See the RACF documentation for details.
S	saf_rsn	The SAF reason code. See the RACF documentation for details.
T	esm_resp	The External Security Manager (ESM) response code. See the ESM documentation for details.
U	esm_rsn	The ESM reason code. See the ESM documentation for details.
V	failing_userid	Checks are made against the task user ID and the link user ID if applicable. This is the user ID that failed the security check.
W	diagnostics	Diagnostics are for IBM Service use.

The spreadsheet data is initially sorted in order by using the fields odstarttime, odapplid, odtransid, odtaskid, applid, transid, taskid. This ordering groups the requests together and also you can see where security errors occur. If you prefer another order, such as listing just the exception responses, you can use the spreadsheet program to change the sorting to your preferred sequence.

For this example:

- ❶ The request failed EXCEPTION with a NOTAUTH for an **EXEC CICS QUERY SECURITY** request on the profile DUMMY in the FACILITY class. The user ID CRPTST2 had ACCESS of NONE. However, the request would not issue any messages as NOLOG was specified.
- ❷ A surrogate check failed in region IYK2ZDL2. User ID CRPTST2 was trying to issue an **EXEC CICS START** user ID(CRPTST3). However, CRPTST2 did not have READ access to the profile CRPTST3.DFHSTART in class SURROGAT. In column V you can see the user ID that failed.
- ❸ The same surrogate check worked in region IYK2ZDL3 because XUSER=NO was specified on the SIT to disable surrogate checking.
- ❹ In region IYK2ZDL3 two user IDs are shown, the task user ID CRPTST2 and the link user ID CRPTST3. Checks would be made on both user IDs.
- ❺ Resource security checks worked because the transaction definition is specified as RESSEC(NO).



## Checking that a CICS security configuration example is working by using the SRR

Configuration examples for various scenarios exist in the security documentation. This task explains how to use security request recording (SRR) to check that one of security documentation configuration examples uses the expected security.

### Before you begin

Before you begin this task, you must complete several other tasks:

- Installed the CICS Explorer or CECI
- [Configuring security request recording \(SRR\)](#)
- Completed the configuration example

You must have:

- Authority to run the CICS Explorer or issue CECI commands.
- Authority to issue the SET SECRECORDING command.
- Authority to run program DFHSRR to obtain a report for SRR.
- Authority to read the CSV output file produced by the DFHSRR program.
- An application that uses the configuration example.

### About this task

In this task, you configure the SRR to record the security requests for an application.

This task assumes that you know the entry point of the application by using the configuration example. This information can include the user ID of the person that is testing the configuration.

### Procedure

1. Select the Regions view in the CICS Explorer and select the region that represents the entry point of the applications. You can select more than one region when the entry point might be in more than one region.
2. Right-click to select the actions menu and then select **Add Security Request Recording**.
3. Identify and select the tab that is most appropriate for the type of security problem you are trying to diagnose.
4. Specify the *matchid* for your request. The *matchid* is used in step 8 to identify this specific recording when you want to look at the output.
5. Specify the maximum number of requests that you want to record.
6. Specify in the tab as much information about the entry point that you know. Reference [Figure 177 on page 614](#) that shows where the only information known was the user ID.

Figure 177. SRRADD window in CICS Explorer

7. Run your application.
8. Run the SRR report by using the *matchid* used in step 4.
9. Open the CSV file generated by the report.

For more information about using the SRR, see [Diagnosing access issues with security request recording \(SRR\)](#).

## Results

The CSV file shows all of the authorization requests for resources that are used by the application, and the CICS region in which these resources are used. It also shows the task user ID and any link user ID used when access is authorized to these resources.

Check that the user IDs used are what you would expect based on the configuration examples that you are testing. Also, check that the resources used by the application and the access that is granted is what you are expecting.

In the following simple example, signed on user ID PENFOLD uses CECI to LINK to a program called BR14 on a remote region over an IPIC link as described in [Configuration example: Securing CICS-to-CICS with an IPIC connection within a sysplex](#). The region user ID of the local region is CRPTST1. The program BR14 just issues an EXEC CICS RETURN statement.

You can see the user ID that is used in [Figure 178 on page 615](#). The task user ID is PENFOLD both in the local and remote regions. Security checking on the remote region is also done against the link user ID CRPTST1.

A	B	C	D	E	F	G
matchid	odapplid	odtransid	odtaskid	odstarttime	applid	transid
IPICEX1	IYK2ZDL1	CECI	60	2022-02-22 13:40:30.369069	IYK2ZDL1	CECI
IPICEX1	IYK2ZDL1	CECI	60	2022-02-22 13:40:30.369069	IYK2ZDL1	CECI
IPICEX1	IYK2ZDL1	CECI	60	2022-02-22 13:40:30.369069	IYK2ZDL1	CECI
IPICEX1	IYK2ZDL1	CECI	60	2022-02-22 13:40:30.369069	IYK2ZDL2	CSMI
IPICEX1	IYK2ZDL1	CECI	60	2022-02-22 13:40:59.785414	IYK2ZDL2	CSMI

H	I	J	K	L	M	N	O
taskid	userid	link_userid	edf_userid	resource_type	profile	class	access
60	PENFOLD			TRANSATTACH	CECI	TCICSTRN	READ
60	PENFOLD			PROGRAM	DFHEITAB	MCICSPPT	READ
60	PENFOLD			PROGRAM	DFHECID	MCICSPPT	READ
57	PENFOLD	CRPTST1		TRANSATTACH	CSMI	TCICSTRN	READ
57	PENFOLD	CRPTST1		PROGRAM	BR14	MCICSPPT	READ

P	Q	R	S	T	U	V	W
response	reason	saf_resp	saf_rsn	esm_resp	esm_rsn	failing_userid	diagnostics
OK		0	0	0	0		X'00800000'
OK	EXEMPT=PROGRAM						X'00040000'
OK	EXEMPT=PROGRAM						X'00040000'
OK		0	0	0	0		X'00800000'
OK		0	0	0	0		X'00800000'

Figure 178. Sample SRR spreadsheet view of the user ID details

## CICS initialization failures related to security

If SEC=YES is specified, external security is *required*. If external security cannot be provided, CICS cannot be initialized.

Figure 179 on page 616 shows an example of a failure to initialize.

If security initialization fails:

- Examine the DFHXS1106 message return codes. In the example shown in Figure 179 on page 616, SAF return code X'00000004' and reason code X'00000000' were issued:

A return code of X'00000004' indicates that an error occurred in the MVS security router (RACROUTE). See the RACROUTE macro reference in [CICS security control points](#).

- Check the CICS startup options, in particular the *Xname* system initialization parameters. Make sure that:
  - The class is defined to RACF and is active (use the SETROPTS LIST command to check this).
  - The class is defined in the router table. To do this, examine the installation source for ICHRF01 for any installation-defined classes. (The description of the ICHFRFB macro in [z/OS Security Server RACF Macros and Interfaces](#) includes a listing of the IBM-supplied module, ICHRF0X.)
- If a RACF abend occurs, see [z/OS Security Server RACF Messages and Codes](#) and [z/OS Security Server RACF Diagnosis Guide](#) for further guidance.
- Check if any SAF or RACF installation exits are causing initialization RACLIST requests to fail.

Figure 179 on page 616 shows that XPPT=UNKNOWN has been specified. This causes CICS to try to use a class called MUNKNOWN. MUNKNOWN has not been defined to the z/OS router, or to the RACF CDT.

## CICS default user fails to sign on

The following example shows a CICS job log when the DFLTUSER fails to sign on. CICS is started with SEC=YES and DFLTUSER=ORMAN. User profile ORMAN has not been defined to RACF.

This CICS region cannot be initialized because, with SEC=YES specified, external security is required and the default user must be defined to RACF.

```
DFHPA1927 IYCTZCCA AKPFREQ=0
DFHPA1927 IYCTZCCA APPLID=IYCTZCCA
DFHPA1927 IYCTZCCA CSDFRLOG=NO
DFHPA1927 IYCTZCCA CSDRECOV=NONE
DFHPA1927 IYCTZCCA FCT=NO
DFHPA1927 IYCTZCCA SIT=6$
DFHPA1927 IYCTZCCA START=INITIAL
DFHPA1927 IYCTZCCA GMTEXT='SSYS NOOR CIC5100M SYSTEM with RACF'
DFHPA1927 IYCTZCCA GRPLIST=USERLIST
DFHPA1927 IYCTZCCA PLTPI=NO
DFHPA1927 IYCTZCCA SEC=YES
DFHPA1927 IYCTZCCA XCMD=NO
DFHPA1927 IYCTZCCA XFCT=1CVFFCT
DFHPA1927 IYCTZCCA XJCT=1CVFJCT
DFHPA1927 IYCTZCCA XPCT=1CVFPCT
DFHPA1927 IYCTZCCA XPPT=UNKNOWN
DFHPA1927 IYCTZCCA XPSB=NO
DFHPA1927 IYCTZCCA XTST=1CVFTST
DFHPA1927 IYCTZCCA XTRAN=1CVFTRN
DFHPA1927 IYCTZCCA CICSSVC=212
DFHPA1927 IYCTZCCA SRBSVC=211
DFHPA1927 IYCTZCCA .END
DFHPA1103 IYCTZCCA END OF FILE ON SYSIN.
+DFHTR0103 TRACE TABLE SIZE IS 64K
+DFHSM0122I IYCTZCCA Limit of DSA storage below 16MB is 5,120K.
+DFHSM0123I IYCTZCCA Limit of DSA storage above 16MB is 48M.
+DFHSM0113I IYCTZCCA Storage protection is not active.
+DFHSM0126I IYCTZCCA Transaction isolation is not active.
+DFHDM0101I IYCTZCCA CICS is initializing.
+DFHLG0101I IYCTZCCA Log manager domain initialization has started.
+DFHSI1500 IYCTZCCA CICS startup is in progress for CICS Transaction Server.
+DFHXS1100I IYCTZCCA Security initialization has started.
+DFH DU0304I IYCTZCCA Transaction Dump Data set DFHDMPA opened.
+DFHSI1501I IYCTZCCA Loading CICS nucleus.
+DFHXS1105 IYCTZCCA Resource profiles for class A1CVFPCT have been built.
+DFH DU0304I IYCTZCCA Transaction Dump Data set DFHDMPA opened.
+DFHXS1105 IYCTZCCA Resource profiles for class F1CVFFCT have been built.
+DFHXS1105 IYCTZCCA Resource profiles for class J1CVFJCT have been built.
+DFHXS1106 IYCTZCCA Resource profiles could not be built for class MUNKOWN.
CICS is terminated. SAF codes are (X'00000004',X'00000000'). ESM codes are
(X'00000000',X'00000000').
+DFH DU0303I IYCTZCCA Transaction Dump Data set DFHDMPA closed.
+DFHKE1800 IYCTZCCA ABNORMAL TERMINATION OF CICS IS COMPLETE.
IEF450I SSYTZCCA CICS - ABEND=S000 U1800 REASON=00000000
TIME=13.55.26
$HASP395 SSYTZCCA ENDED
```

Figure 179. Security initialization failure

```

DFHHPA1927 IYCTZCCE SEC=YES
DFHHPA1927 IYCTZCCE XUSER=YES
DFHHPA1927 IYCTZCCE DFLTUSER=ORMAN
DFHHPA1927 IYCTZCCE XCMD=NO
DFHHPA1927 IYCTZCCE XFCT=1CVFFCT
DFHHPA1927 IYCTZCCE XJCT=1CVFJCT
DFHHPA1927 IYCTZCCE XPCT=1CVFPCT
DFHHPA1927 IYCTZCCE XFCT=1CVFPPT
DFHHPA1927 IYCTZCCE XPSB=NO
DFHHPA1927 IYCTZCCE XTST=1CVFTST
DFHHPA1927 IYCTZCCE XTRAN=1CVFTRN
DFHHPA1927 IYCTZCCE CICSSVC=212
DFHHPA1927 IYCTZCCE SRBSVC=211
DFHHPA1927 IYCTZCCE .END
DFHHPA1103 IYCTZCCE END OF FILE ON SYSIN.
+DFHTR0103 TRACE TABLE SIZE IS 64K
+DFHSM0122I IYCTZCCE Limit of DSA storage below 16MB is 5,120K.
+DFHSM0123I IYCTZCCE Limit of DSA storage above 16MB is 48M.
+DFHSM0113I IYCTZCCE Storage protection is not active.
+DFHSM0126I IYCTZCCE Transaction isolation is not active.
+DFHDM0101I IYCTZCCE CICS is initializing.
+DFHGL0101I IYCTZCCE Log manager domain initialization has started.
+DFHSI1500 IYCTZCCE CICS startup is in progress for CICS Transaction
Server
+DFHXS1100I IYCTZCCE Security initialization has started.
+DFHDU0304I IYCTZCCE Transaction Dump Data set DFHDMPA opened.
+DFHSI1501I IYCTZCCE Loading CICS nucleus.
+DFHXS1105 IYCTZCCE Resource profiles for class A1CVFPCT have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class F1CVFFCT have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class J1CVFJCT have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class M1CVFPPT have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class S1CVFTST have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class T1CVFTRN have been built.
+DFHXS1105 IYCTZCCE Resource profiles for class SURROGAT have been built.
ICH4081 USER(ORMAN ) GROUP( ) NAME(???)
LOGON/JOB INITATION - USER AT TERMINAL NOT RACF-DEFINED
IRR012I VERIFICATION FAILED. USER PROFILE NOT FOUND.
+DFHXS1104 IYCTZCCE Default security could not be established for userid ORMAN.
The security domain cannot continue, so CICS is terminated. SAF codes are
(X'00000004',X'00000000'). ESM codes are (X'00000004',X'00000000').
+DFHDU0303I IYCTZCCE Transaction Dump Data set DFHDMPA closed.
+DFHKE1800 IYCTZCCE ABNORMAL TERMINATION OF CICS IS COMPLETE.
IEF450I SSSYZCCE CICS - ABEND=S000 U1800 REASON=00000000
TIME=13.55.26
$HASP395 SSSYZCCE ENDED

```

Figure 180. Example of CICS job log if DFLTUSER fails to sign on

## Revoked user attempting to sign on

When a user attempts to sign on with a revoked user ID, the sign-on is terminated with a message to inform the user of the user ID revocation. RACF and CICS messages are issued to log this sign-on failure.

The following example sequence illustrates what happens when a revoked user attempts to sign on:

1. User USR001 attempts to sign on using CESN. However, the user is revoked. The user sees the following on the terminal:

```
DFHCE3546 Your signon userid has been revoked. Signon is terminated.
```

2. A RACF ICH408I message is sent to the CICS region's job log:

```
ICH408I USER(USR001 ) GROUP(GRP001 ) NAME(AUSER )
LOGON/JOB INITATION - REVOKED USER ACCESS ATTEMPT
```

This message indicates that user USR001, whose name as recorded in the RACF user profile is AUSER, and whose current RACF connect group is GRP001, attempted to sign on.

3. A CICS message is sent to the CICS transient data queue:

```
DFHSN1120 26/09/95 12:20:24 CICSSYS1 Signon at netname D2D1
with userid USR001 failed because the userid has been revoked.
```

## Related information

[“RACF message ICH408I: an explanation with example” on page 605](#)

The first line of message ICH408I identifies a user who had an authorization problem. The other lines of the message describe the request the user was issuing and the reason for the failure.

[How it works: Authentication in CICS](#)

[Auditing sign-on and sign-off](#)

## User has insufficient authority to access a resource

Now consider user USR001, who has signed on successfully with current connect group GRP001. User USR001 attempts unsuccessfully to use transaction CEMT, which is protected by profile CAT2 in class GCICSTRN (the resource group class for CICS transactions), because **XTRAN=YES** is specified in the CICS system initialization parameters.

1. The terminal user received the following CICS message:

```
DFHAC2033 26/09/95 15:18:44 CICSSYS1 You are not authorized to use
transaction CEMT. Check that the transaction name is correct.
```

2. A RACF ICH408I message is sent to the CICS region's job log:

```
ICH408I USER(USR001 ) GROUP(GRP001 ) NAME(AUSER )
ICH408I CEMT CL(TCICSTRN)
ICH408I INSUFFICIENT ACCESS AUTHORITY
ICH408I ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

This message indicates that user USR001, whose name as recorded in the RACF user profile is AUSER, and whose current RACF connect group is GRP001, attempted to use the CEMT transaction. To do this, AUSER needs to have at least READ access to the profile protecting the CEMT transaction. However, RACF determined that AUSER had **no** access authority.

3. A CICS message is sent to the CICS transient data queue:

```
DFHXS1111 26/09/95 13:30:41 CICSSYS1 CEMT Security violation
by user USR001 at netname D2D1 for resource CEMT in class
TCICSTRN. SAF codes are (X'00000008',X'00000000'). ESM codes
are (X'00000008',X'00000000').
```

The following message is also sent to the CSMT transient data queue:

```
DFHAC2003 26/09/95 15:18:44 CICSSYS1 Security violation has been
detected term id = D2D1, trans id = CEMT, userid = USR001.
```

4. Which profile protects CEMT?

It appears from the ICH408I message that profile CEMT in class TCICSTRN protects CEMT. However, this is not necessarily the case. A resource group profile (in class GCICSTRN) might protect CEMT. In fact, in this case, there is no profile named CEMT. If a system-SPECIAL or AUDITOR user issues the SEARCH command with CLASS(TCICSTRN) specified, no profile named CEMT would appear.

To determine which profile was used, you must issue the **RLIST** command with the RESGROUP operand as follows:

```
RLIST member-class resource-name RESGROUP
```

In this case, issue the following:

```
RLIST TCICSTRN CEMT RESGROUP
```

**Note:** If prefixing is used for this CICS region, specify the prefix on the resource-name in the RLIST command.

RACF displays the following:

```

CLASS      NAME
-----
TCICSTRN   CEMT
GROUP CLASS NAME
-----
GCICSTRN
RESOURCE GROUPS
-----
CAT2

```

The profiles in class GCICSTRN that protect CEMT are shown under RESOURCE GROUPS in the command output. In this case, only one profile (CAT2) protects profile CEMT.

**Note:** If a profile in class TCICSTRN protected CEMT, that profile's contents would be added to the output of RLIST.

5. To determine how profile CAT2 protects CEMT, list that profile with the AUTHUSER operand specified on the RLIST command:

```
RLIST GCICSTRN CAT2 AUTHUSER
```

RACF displays the following:

```

CLASS      NAME
-----
GCICSTRN   CAT2
MEMBER CLASS NAME
-----
TCICSTRN
RESOURCES IN GROUP
-----
CDBC
CDBI
CBRC
CEDA
CEMT
CETR
LEVEL      OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----
:
NOTIFY
-----
NO USER TO BE NOTIFIED
USER      ACCESS      ACCESS COUNT
-----
DEPTA     ALTER          000000
USR001    NONE          000000
-----
NO ENTRIES IN CONDITIONAL ACCESS LIST

```

## Authorization failures and error messages

If a terminal user tries to initiate an unauthorized transaction, CICS issues a security violation message (DFHAC2033) to the terminal.

CICS then sends a corresponding message (DFHAC2003) to the CSMT transient data destination, and messages DFHXS1111 and DFHXS1117 are sent to CSCS. RACF normally issues an ICH408I message to the CICS region's job log and to the security console (the console defined for routing code 9 messages). However, if the transaction has been defined to RACF with AUDIT(NONE), no ICH408I message is issued. For a description of the ICH408I message, see [z/OS Security Server RACF Messages and Codes](#).

For more information on resolving authorization problems, see [Problem determination in a CICS-RACF security environment](#).

If auditing (such as that requested by the AUDIT operand) is requested for this access, RACF writes an SMF type 80 log record. Your RACF auditor can use the RACF report writer to generate reports based on these records. For more information, see the [z/OS Security Server RACF Auditor's Guide](#).

## Password expiry management problems

If you are running a CICS-APPC PEM environment, and you are not receiving the expected responses, you can perform checks to identify the cause of the errors.

The following possible errors might have occurred in the sign-on transaction program:

- The function management header (FMH) might be in error. Check that the following requirements are satisfied:
  - The conversation type being used is `basic`.
  - The XTRANID in the CICS TRANSACTION definition for CLS4 is `X'06F3F0F1'`.
  - The CICS PEM server sign-on transaction is running as a `synclevel 0` transaction.
- The user data might be in error. Check that the following requirements are satisfied:
  - Valid lengths are being sent.
  - User IDs and passwords are sent in uppercase EBCDIC.
  - GDS variables (required in basic conversations) are being used.

**Note:** If the CICS PEM server receives an error in the FMH or user data, it sends an `ISSUE ERROR` to the PEM requester, and terminates without an `abend`. In this situation, it is likely that there is an error in the flow.

For security reasons, you cannot use the execution diagnostic facility (EDF) to check DFHCLS4 because user passwords would be displayed on the EDF screens.

## Troubleshooting Kerberos

---

Sources of diagnostic information and detail of some specific exceptions

### Where to look for more information

In general, if you experience an error, check for information in the following trace files:

#### CICS trace

##### XS Exception Trace

Look for calls to `R_TICKETSERV`. A specimen trace record follows:

```
XS FE04 XSSK *EXC* FUNCTION(INQUIRE_CLIENT_PRINCIPAL) RESPONSE(EXCEPTION)
REASON(INVALID_TOKEN)
      SAF_RESPONSE(8) SAF_REASON(0) ESM_RESPONSE(10) ESM_REASON(861B6D0B)
      ESM_SERVICE_NAME(R_TICKETSERV) PRINCIPAL_NAME(000704DA , 000000F4 ,
      000000F4) USERID_LENGTH(0) USERID()
```

Identify the response code and look it up in [z/OS Security Server RACF Callable Services](#) or [RACF and z/OS Integrated Security Services Network Authentication Service](#).

Full trace for XS trace points `x'0A01'`, `x'0A02'`, `x'FE01'`, `x'FE02'`, and `x'FE04'` can show additional information to aid diagnosis. For example, it can show the values of a principal name and of a Kerberos token, before and after base-64 conversion.

#### SKRKBDC logs

The Key Distribution Centre task SKRKBDC can produce useful messages in its output. For example, the following message might be written to SYSOUT:

```
EUVF04039W Kerberos login failed for ROBOT@CLOUD.IBM.COM at
9.20.141.141:33063:
      KDC status 0x96c73a06 - Client principal is not found in user
      registry.
```

To obtain more information from SKRKBDC, turn on all trace by issuing the following commands:



```
F SKRKBDC,DEBUG *.9
F SKRKBDC,DEBUG ON
```

For more information, see [RACF and z/OS Integrated Security Services Network Authentication Service](#).

### SKRKBDC spool files

z/OS Integrated Security Services Network Authentication provides environment variables that can be used as an aid to diagnosis. For more information see [RACF and z/OS Integrated Security Services Network Authentication Service](#). Note that some of these variables are intended to be used only under the direction of IBM Service Personnel.

## Calls by CICS to z/OS Callable Services and an External Security Manager

Error messages can be written to destination CSCS if errors occur while you are using Kerberos. For example, such messages can be produced if a web service that uses Kerberos fails because of a Kerberos related security failure, or if an **EXEC CICS VERIFY TOKEN** command does not execute as expected. In the case of an **EXEC CICS VERIFY TOKEN** command, these messages are in addition to the conditions and EIBRESP2 values produced.

As an aid to showing Kerberos-related identifiers in use by CICS TS, the following messages can be produced during CICS TS initialization:

```
DFHXS1400 03/04/2014 15:31:36 IYKZZZZ Kerberos realm is MVS2.IBM.COM
DFHXS1401 03/04/2014 15:31:36 IYKZZZZ Kerberos principal name is cloud_cics512
```

## Information about specific messages

### DFHXS1402 with reason = R\_TICKETSERV service responded invalid server principal name.

The server principal name is the Kerberos principal name specified in message DFHXS1401. If you use Tivoli® Federated Identity Manager, set the Kerberos Service Name in the Kerberos Module Configuration to this value.

### DFHXS1402 with reason = R\_TICKETSERV service responded invalid kerberos token.

A possible reason for this error is that you used an unsupported Kerberos format type, such as `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`. For a list of supported format types, see [The <authentication> element](#).

### DFHXS1402 with reason = 'Kerberos not configured'

The region has not been configured to support the Kerberos service. For the region to use Kerberos, you must set the **KERBEROSUSER** system initialization parameter, which specifies a user ID to be associated with the Kerberos service principal.

## Information about specific exceptions

### INVREQ exception condition with RESP2=40: The key distribution center is not started or is terminating

You must have a key distribution center (KDC) running on the same LPAR as the CICS region that issued the VERIFY TOKEN command. The KDC is the SKRKBDC Started Task that supplies the Network Authentication Service for z/OS.

### NOTAUTH exception condition with RESP2=20: The user is not authorized to use this service

RACF errors do not cause a RACF violation in CICS, or messages in the KDC. The "user" in this case is likely to be the default user of the CICS region, rather than the CICS region user ID or the ID of the Kerberos Ticket.

## Troubleshooting Passtickets

---

Sources of diagnostic information and detail of a specific exception

### Where to look for more information

In general, if you experience an error, check for information in the following trace files:

#### CICS trace

##### XS Exception Trace

Look for calls to CREATE\_PASSTICKET. A specimen trace record follows:

```
XS FE04 XSSD *EXC* FUNCTION(CREATE_PASSTICKET) RESPONSE(EXCEPTION)
REASON(PASSTICKET_NOT_CREATED) ESM_RESPONSE(8) ESM_REASON(0)
PASSTICKET_LENGTH(0) PASSTICKET() USERID_LENGTH(6) USERID(JAT271)
RCVTPTGN_REASON(24000000)
```

Identify the RCVTPTGN\_REASON code and look it up in [Using the RCVTPTGN service to generate a PassTicket](#).

#### Related information

[How it works: PassTickets](#)

[Using CICS trace for problem determination](#)

---

## Chapter 33. Security reference

The security reference information provides details about security rules, RACF classes, commands and resources, SIT parameters, and standards.

### Available CICS health checks supported by IBM Health Checker for z/OS

---

Checking and reporting of key security configuration settings in CICS regions identifies configuration issues, potential compliance exposures, and enables simpler auditing. This section provides details about each health checker rule.

**Security best practice (validated by IBM Health Checker for z/OS):** In the CICS documentation, configuration best practices that are validated by IBM Health Checker for z/OS are highlighted in boxes, like the one that surrounds this statement.

For an overview of this capability, including information on excluding certain health checks, see [Auditing CICS configuration with IBM Health Checker for z/OS](#).

#### Related information

Reference: [IBM Health Checker for z/OS messages related to CICS](#)

[DFHHCnnnn messages](#)

[IBM Health Checker for z/OS User's Guide](#)

### CICS\_CAT3\_CONFIGURATION

CICS\_CAT3\_CONFIGURATION checks the configuration of the CICS category 3 transactions.

#### Description

This check reviews a subset of the CICS category 3 transaction definitions to ensure that they invoke the expected initial programs.

#### Reason for check:

Validate that category 3 transactions are used as intended and have not been altered, which could lead to security exposures.

View this [table of transactions checked](#) for details.

#### z/OS releases the check applies to:

Any z/OS release that supports CICS TS 6.1 or later.

#### Minimum CICS TS release required:

CICS TS 6.1 or later.

#### Type of check (local, remote, or Rexx):

Local.

#### User override of IBM values:

No.

#### Debug support:

- **6.2** Yes.
- **6.1** No.

#### Verbose support:

No.

#### Parameters accepted:

None.

**Reference:**

For more information about applicable guidance, see the [table of transactions checked](#).

**Messages:**

[This table](#) displays the list of messages for this check.

See Reference: [IBM Health Checker for z/OS messages related to CICS](#).

The rule checks these transactions:

<i>Table 55. Transactions checked</i>		
<b>Transaction checked</b>	<b>Related risk or concern and applicable guidance</b>	<b>Message displayed</b>
CEGN	The transaction should run program DFHCEGN. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0801 (Exception)
CESF	The transaction should run program DFHSFP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0802 (Exception)
CESL	The transaction should run program DFH SNP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0803 (Exception)
CESN	The transaction should run program DFH SNP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0804 (Exception)
CLQ2	The transaction should run program DFHLUP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0805 (Exception)
CLR1	The transaction should run program DFHZLS1. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0806 (Exception)
CLR2	The transaction should run program DFHLUP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0807 (Exception)

Table 55. Transactions checked (continued)

Transaction checked	Related risk or concern and applicable guidance	Message displayed
CLS1	The transaction should run program DFHZLS1. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0808 (Exception)
CLS2	The transaction should run program DFHLUP. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0809 (Exception)
CSGM	The transaction should run program DFHGMM. If it does not, it is possible that the transaction is being used inappropriately in a manner which could represent a security exposure to the region.	DFHH0810 (Exception)

## CICS\_CEDA\_ACCESS

CICS\_CEDA\_ACCESS checks whether the CICS default user has access to the CEDA transaction, or whether CICS security is turned off by the SIT parameter SEC=NO. If either is detected, it means that anyone who uses a 3270 device or terminal emulator can use the CEDA transaction to change the configuration of CICS. This could compromise the security of this CICS region - and other regions if the CSD is shared.

### Description

Define regions with **SEC**=YES and secure the CEDA transaction so that only authorized, signed-on users can make changes to system configuration.

### Reason for check:

Avoid compromising the security of CICS where any user who accesses the system by using a 3270 device or terminal emulator can use the CEDA transaction to change the configuration of CICS.

In general, regions should be defined with **SEC**=YES and CEDA should be secured so that only authorized, signed-on users can make changes to system configuration.

If you do need to run CICS with **SEC**=NO, either disable CEDA access for this region or configure the region to run in an isolated LPAR.

### z/OS releases the check applies to:

Any z/OS release that supports CICS TS 5.4 or later.

### Minimum CICS TS release required:

CICS TS 5.4 or later.

### Type of check (local, remote, or Rexx):

Local.

### User override of IBM values:

No.

### Debug support:

- **6.2** Yes.

- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

For more information: [CEDA - resource definition online](#)

**Messages:**

This check issues the following messages:

- DFHH0001E - The CEDA transaction is accessible to unauthenticated users.
- DFHH0301I - All CEDA transactions are protected from unauthenticated users.

See [Reference: IBM Health Checker for z/OS messages related to CICS](#).

## CICS\_JOBSUB\_SPOOL

CICS\_JOBSUB\_SPOOL checks whether the system spooling interface is enabled (that is, the **SPOOL** SIT parameter is set to SPOOL=YES, and, if so, whether the CECI transaction is accessible to the default user, or whether CICS security is turned off in this region (that is, the **SEC** SIT parameter is **SEC=NO**).

**Description**

Use the SIT parameter **SPOOL=YES** only on regions that have applications that need to write to the spool by using the API command **SPOOLWRITE**.

**Reason for check:**

When the system spooling interface is enabled, either if CECI is accessible to the default user or if CICS security is turned off, anyone who can connect to the IP address and port number of the CICS region can submit jobs to run on the z/OS system remotely under the region user ID without authentication.

In general, defined regions with **SEC=YES** and CECI so that only authorized, signed-on users can issue commands by using the CECI transaction.

If you do need to run CICS with **SEC=NO**, disable CECI access for this region, or configure the region to run in an isolated LPAR.

**z/OS releases the check applies to:**

Any z/OS release that supports CICS TS 5.4 or later.

**Minimum CICS TS release required:**

CICS TS 5.4 or later.

**Type of check (local, remote, or Rexx):**

Local.

**User override of IBM values:**

No.

**Debug support:**

- **6.2** Yes.
- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

You can check which programs use the **SPOOLWRITE** command by using the [load module scanner DFHEISUP](#).

**Messages:**

This check issues the following messages:

- DFHH0002E - The spool is accessible to unauthenticated users.
- DFHH0302I - The spool is protected from unauthenticated users.

See [Reference: IBM Health Checker for z/OS messages related to CICS](#).

## CICS\_JOBSUB\_TDQINTRDR

CICS\_JOBSUB\_TDQINTRDR checks whether any transient data queue that writes to the internal reader can either be written to by the default user or is available in a region where CICS security is turned off (that is, the SIT parameter is **SEC=NO**).

**Description**

Secure transient data queues that write to the internal reader so that only authorized, signed-on users can write to them.

**Reason for check:**

When transient data queues are defined to write to the internal reader (that is, the transient data queues are defined as extrapartition and specify a DD name that has SYSOUT referencing INTRDR), any of the following conditions, if detected, mean that anyone who can connect to the IP address and port number of the CICS regions can submit jobs to run on the z/OS system remotely under the region user ID without authentication:

- Transient data queues that write to the internal reader can be written to by the default user.
- The IBM-supplied transaction CECI is accessible to the default user.
- CICS security is turned off.

Define regions with **SEC=YES** and CECI so that only authorized, signed-on users can issue commands using the CECI transaction.

If you do need to run CICS with **SEC=NO**, disable CECI access for this region, or configure the region to run in an isolated LPAR.

**z/OS releases the check applies to:**

Any z/OS release that supports CICS TS 5.4 or later.

**Minimum CICS TS release required:**

CICS TS 5.4 or later.

**Type of check (local, remote, or Rexx):**

Local

**User override of IBM values:**

No

**Debug support:**

- **6.2** Yes.
- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

[Installing TDQUEUE resource definitions](#)

**Messages:**

This check issues the following messages:

- DFHH0003E
- DFHH0303I

See [Reference: IBM Health Checker for z/OS messages related to CICS](#).

## CICS\_REGION\_CONFIGURATION

CICS\_REGION\_CONFIGURATION checks that key production region configuration parameters align to best practice recommendations in CICS.

**Description**

This check reviews a set of CICS region parameters to ensure that the region configuration aligns to best practice.

**Reason for check:**

Validate that access to key resources aligns to best practice recommendations for production regions.

View this [table of conditions checked](#) for details.

**z/OS releases the check applies to:**

Any z/OS release that supports CICS TS 6.1 or later.

**Minimum CICS TS release required:**

CICS TS 6.1 or later.

**Type of check (local, remote, or Rexx):**

Local.

**User override of IBM values:**

No.

**Debug support:**

- **6.2** Yes.
- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

For more information about applicable guidance, see the [table of conditions checked](#).

**Messages:**

This [table](#) displays the list of messages for this check.

See [Reference: IBM Health Checker for z/OS messages related to CICS](#).

This table lists the conditions that are checked, the related risk or concern with the setting if discovered in a production region, and the message displayed for production regions.

Table 56. Conditions checked		
Condition checked	Related risk or concern and applicable guidance	Message displayed
SEC=NO	This setting prevents the running of security checks. <a href="#">Learn more about SEC system initialization parameter.</a>	DFHH0401 (Exception)



Table 56. Conditions checked (continued)

Condition checked	Related risk or concern and applicable guidance	Message displayed
<b>XTRAN=NO</b>	Any user can run any transaction.  Learn more about <a href="#">XTRAN system initialization parameter</a> .	DFHH0402 (Exception)
<b>XDB2=NO</b>	No surrogate user checking and AUTHTYPE checking for IBM Db2 occurs.  Learn more about <a href="#">XDB2 system initialization parameter</a> .	DFHH0403 (Exception)
<b>CONFDATA=SHOW</b>	Prevents redaction of user IDs or passwords at selected trace points, which might expose user data inappropriately.  Learn more about <a href="#">CONFDATA system initialization parameter</a> .	DFHH0404 (Warning)
<b>MINTLSLEVEL</b> set to a value less than TLS12	TLS 1.2 is the minimum recommended TLS level. Setting <b>MINTLSLEVEL</b> to a value less than TLS12 increases the risk of interception of TCP/IP transactions that use TLS.  Learn more about <a href="#">MINTLSLEVEL system initialization parameter</a> .	DFHH0405 (Warning)
<b>HTTPSERVERHDR=YES</b> or <b>HTTPSUSRAGENTHDR=YES</b>	These two settings provide a default header response that might be used to identify possible penetration vectors and is often a key focus for audit penetration testing.  For more information: <ul style="list-style-type: none"> <li>• <a href="#">HTTPSERVERHDR system initialization parameter</a></li> <li>• <a href="#">HTTPOSAGENTHDR system initialization parameter</a></li> </ul>	DFHH0406 (Exception)
<b>STGPROT=YES</b> and <b>CWAKEY=USER</b>	When set with these values, write access to the CWA is possible.  Learn more about <a href="#">Sharing data across transactions</a> .	DFHH0407 (Warning)
<b>RESSEC=ASIS</b>	Any user might create a new resource definition.  See <a href="#">Resource security</a> for more details.	DFHH0409 (Information)

Table 56. Conditions checked (continued)

Condition checked	Related risk or concern and applicable guidance	Message displayed
<b>DFLTUSER</b> or <b>KERBEROSUSER</b> match the CICS region user ID	The CICS region user ID is recommended to be used only for the region.  For more information: <ul style="list-style-type: none"> <li>• <a href="#">DFLTUSER system initialization parameter</a></li> <li>• <a href="#">KERBEROSUSER system initialization parameter</a></li> </ul>	DFHH0411 (Information)
<b>RACFSYNC=NO</b>	This setting prevents CICS listening for events of type 71 ENF, which means cached tokens are not invalidated when a 71 ENF event occurs. <b>RACFSYNC</b> is an important parameter for production regions and must be set to <b>RACFSYNC=NO</b> only under guidance from IBM.  <a href="#">Learn more about RACFSYNC system initialization parameter.</a>	DFHH0410 (Information)
<b>GMTRAN=DISCONNECT</b> and the good morning transaction is set to either CESL or CESN	3270 users either logon directly to a default user screen or can reach that screen when they complete sign-on or sign-off transactions.  <a href="#">Learn more about GMTRAN system initialization parameter.</a>	DFHH0412 (Information)
CICS region user ID has a revocable password	If the CICS region user ID password is revoked, the region is inaccessible.  <a href="#">Learn more about Specifying the CICS region user ID.</a>	DFHH0413 (Exception)
<b>XCMD=NO</b>	This setting disables command security.  <a href="#">Learn more about XCMD system initialization parameter.</a>	DFHH0414 (Warning)

## CICS\_RESOURCE\_CONFIGURATION

CICS\_RESOURCE\_CONFIGURATION checks that configuration of the CICS resources aligns to best practice recommendations for production regions.

### Description

This check reviews a set of CICS resource configuration settings to ensure alignment to best practice.

### Reason for check:

Validate that access to key resources aligns to best practice recommendations for production regions.

See [Table 57 on page 631](#) below for details.

**z/OS releases the check applies to:**

Any z/OS release that supports CICS TS 6.1 or later.

**Minimum CICS TS release required:**

CICS TS 6.1 or later.

**Type of check (local, remote, or Rexx):**

Local.

**User override of IBM values:**

No.

**Debug support:**

- **6.2** Yes.
- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

For more information about applicable guidance, see [Table 57 on page 631](#).

**Messages:**

[Table 57 on page 631](#) displays the list of messages for this check. See also [Reference: IBM Health Checker for z/OS messages related to CICS](#).

The rule checks for these conditions:

<i>Table 57. Conditions checked</i>		
Condition checked	Related risk or concern and applicable guidance	Message displayed
CEDF, CECI, or CEMT transactions are installed and the default user can access.	It is a potential security risk if the default user can run any of the checked transactions in a production region.  You are advised to remove access for the default user.	DFHH0501 (Exception)
Review TCPIP SERVICE definitions for use of the IPIC protocol with a user replaceable module (URM) of DHISAPI.	Using this URM allows any client to connect and automatically install a connection.  Learn more about <a href="#">Defining IPIC connections</a>	DFHH0502 (Exception)

Table 57. Conditions checked (continued)

Condition checked	Related risk or concern and applicable guidance	Message displayed
Validate if clones of sensitive transactions exist. The list of transactions that are checked for clones is: <ul style="list-style-type: none"> <li>• CEMT</li> <li>• CEDA</li> <li>• CECI</li> <li>• CEDF</li> <li>• CEDX</li> <li>• CETR</li> <li>• CLER</li> </ul>	Unauthorized users might run the cloned transaction, which represents a security risk to the region.  You are advised to review access to the cloned transactions or remove them.	DFHH0503 (Exception)
Evaluate if any TCPIP SERVICES exist that do not use SSL and defined with AUTHENTICATE=NONE or AUTHENTICATE=BASIC.	Any TCPIP SERVICE not configured with TLS and set up with either AUTHENTICATION(NO) or AUTHENTICATE(BASIC) is not secure.  Learn more about <a href="#">TCPIP SERVICE resource</a>	DFHH0504 (Exception)
<b>6.2</b> If XCMD=YES, a check is made for transactions that specify CMDSEC(NO).	XCMD=YES and a transaction is defined with CMDSEC(NO), security checking is unavailable for those transactions. You are advised to review access and secure transactions.	DFHH0505 (Warning)

## CICS\_RESOURCE\_SECURITY

CICS\_RESOURCE\_SECURITY checks a range of SIT parameters that align to best practice recommendations for production regions in CICS.

### Description:

This check reviews a range of SIT parameters for alignment to best practice recommendations in production regions.

### Reason for check:

Validate that key SIT parameters align to best practice recommendations in production regions.

View [Table 58](#) on page 633 for details.

### z/OS releases the check applies to:

Any z/OS release that supports CICS TS 6.1 or later.

### Minimum CICS TS release required:

CICS TS 6.1 or later.

### Type of check (local, remote, or Rexx):

Local.

### User override of IBM values:

No.

**Debug support:**

- **6.2** Yes.
- **6.1** No.

**Verbose support:**

No.

**Parameters accepted:**

None.

**Reference:**

For more information about applicable guidance, see [Table 58 on page 633](#).

**Messages:**

[Table 58 on page 633](#) displays the list of messages for this check.

See Reference: [IBM Health Checker for z/OS messages related to CICS](#).

The list of SIT parameter conditions checked by this rule:

<i>Table 58. Conditions checked</i>		
Condition checked	Related risk or concern and applicable guidance	Message displayed
<b>XPCT=NO</b>	This setting disables transaction resource security checks.  Learn more about <a href="#">Security for PCT-checked transactions</a> .	DFHH0701 (Warning)
<b>XDCT=NO</b>	This setting allows any user to access a transient data queue.  Learn more about <a href="#">Security for transient data</a> .	DFHH0702 (Warning)
<b>XDB2=NO</b> and IBM Db2 is in use	You are advised to set <b>XDB2</b> to improve security.  Learn more about <a href="#">Resource classes for DB2ENTRY resources</a> .	DFHH0703 (Warning)
<b>XJCT=NO</b>	Any user is able to access any CICS journal.  Learn more about <a href="#">XJCT system initialization parameter</a> .	DFHH0704 (Information)
<b>XPPT=NO</b>	Any user is able to use LINK, LOAD, or XCTL commands to start other programs.  Learn more about <a href="#">XPPT system initialization parameter</a> .	DFHH0705 (Information)
<b>XTST=NO</b>	Any user can access any temporary storage queue.  Learn more about <a href="#">XTST system initialization parameter</a> .	DFHH0706 (Information)

Table 58. Conditions checked (continued)

Condition checked	Related risk or concern and applicable guidance	Message displayed
<b>XFCT=NO</b>	Any user can access any file on a CICS production system.  Learn more about <a href="#">XFCT system initialization parameter</a>	DFHH0707 (Warning)
<b>XHFS=NO</b>	For systems that use z/OS UNIX files, this setting disables client user access validation.  Learn more about <a href="#">XHFS system initialization parameter</a>	DFHH0708 (Information)

## CICS\_STABILIZED\_FUNCTIONS

6.2 Applies to 6.2.

CICS\_STABILIZED\_FUNCTIONS checks whether any outdated, stabilized technology, capabilities, or functions are used in the CICS region.

### Description

Out-of-date technology inside CICS TS for z/OS is often stabilized and might be reduced in capability or discontinued in a future release. This check includes a suite of tests and alerts the use of stabilized functions in the CICS region.

### Reason for check:

Stabilized functions that are still in use might expose CICS to potential risks, reduction of performance, or constraints in capability. It is highly recommended to migrate to current technology and functions and make the most of CICS capabilities.

### z/OS releases the check applies to:

Any z/OS release that supports CICS TS 6.2 or later.

### Minimum CICS TS release required:

CICS TS 6.2 or later.

### Type of check (local, remote, or Rexx):

Local.

### User override of IBM values:

No.

### Debug support:

Yes.

### Verbose support:

No.

### Parameters accepted:

None.

### Reference:

[Table 59 on page 635](#) lists the tests that are executed.

See also [Stabilization notices and discontinued functions](#).

### Messages:

[Table 59 on page 635](#) displays a list of messages that can be issued by this check for violations.

See [Reference: IBM Health Checker for z/OS messages related to CICS](#) for details about each message.

Table 59. Stabilized functions checked	
Function	Message displayed
APPC password expiration management (PEM)	DFHH0952
CICS Application Debugging Profile Manager	DFHH0965
CICS debugging tools sockets interface	DFHH0960
CICS Service Flow Runtime	DFHH0953
CICS system events	DFHH0959
CICSplex SM RTA MRM function	DFHH0962
CICSplex SM RTA SAM function	DFHH0963
DFHWBCLI web client interface	DFHH0954
Enterprise Bundle Archive (EBA)	DFHH0961
Extended recovery facility (XRF)	DFHH0951
ONC RPC	DFHH0958
Installation of pipelines that are configured to use a JVMSERVER	DFHH0964
SAML using the CICS Security Token Service	DFHH0955

## CICS\_USS\_CONFIGURATION

CICS\_USS\_CONFIGURATION checks that access to key USS (z/OS UNIX System Services) files and directories align to best practice recommendations for production regions in CICS.

### Description

This check reviews access to key USS (z/OS UNIX System Services) files and directories in production regions.

### Reason for check:

Validate that access to USS (z/OS UNIX System Services) files and directories aligns to best practice recommendations for production regions.

View this table of [conditions checked](#) for details.

### z/OS releases the check applies to:

Any z/OS release that supports CICS TS 6.1 or later.

### Minimum CICS TS release required:

CICS TS 6.1 or later.

### Type of check (local, remote, or Rexx):

Local.

### User override of IBM values:

No.

### Debug support:

- **6.2** Yes.
- **6.1** No.

### Verbose support:

No,

**Parameters accepted:**

None.

**Reference:**

For more information about applicable guidance, see the table of [conditions checked](#).

**Messages:**

[Table 60 on page 636](#) displays the list of messages for this check.

See [Reference: IBM Health Checker for z/OS messages related to CICS](#).

The rule checks for these conditions:

<i>Table 60. Conditions checked</i>		
Condition checked	Related risk or concern and applicable guidance	Message displayed
<b>USSCONFIG</b>	Setting the access flags on <b>USSCONFIG</b> to allow any user access to the directory or files within <b>USSCONFIG</b> creates a security risk.  Learn more about <a href="#">Handling security for your files</a> .	DFHH0601 (Exception)
<b>JVMPROFILEDIR</b>	Setting the access flags on <b>JVMPROFILEDIR</b> to allow any user access to the directory or files within <b>JVMPROFILEDIR</b> creates a security risk.  Learn more about <a href="#">JVMPROFILEDIR system initialization parameter</a> .	DFHH0602 (Exception)
Access to Bundle files.	Setting the access flags on Bundle files that allow any user to access the directory or files within a bundle creates a security risk.  Learn more about <a href="#">Bundle files</a> .	DFHH0603 (Exception)

## Summary of RACF classes for CICS resources

RACF provides classes for CICS resources, CICSplex SM resources, and the system resources that CICS uses.

[Supplied resource classes for z/OS systems](#) in the RACF documentation is a list of all the classes that RACF provides. The tables that are shown summarize the classes that are relevant to CICS.

No default resource class names for DB2ENTRY resources exist. You must define your own resource classes for these resources. For more information, see [Resource classes for DB2ENTRY resources](#).

The initial character of the class names is significant when you define your own classes. For example, if you define your own classes for CICS programs, the names you choose must start with T for the member class and G for the resource grouping class.



Table 61. RACF-provided resource classes for CICS resources

General resource class (member class)	Resource group class	Description
ACICSPCT	BCICSPCT	CICS-started transactions and the following <b>EXEC CICS</b> commands:  COLLECT STATISTICS TRANSACTION DISCARD TRANSACTION INQUIRE TRANSACTION SET TRANSACTION
CCICSCMD	VCICSCMD	CICS and FEPI system commands.
DCICSDCT	ECICSDCT	CICS transient data queues.
FCICSFCT	HCICSFCT	CICS files.
JCICSJCT	KCICSJCT	CICS journals.
MCICSPPT	NCICSPPT	CICS programs.
PCICSPSB	QCICSPSB	CICS PSBs.
RCICSRES	WCICSRES	Miscellaneous CICS resources (see Table 62 on page 637).
SCICSTST	UCICSTST	CICS temporary storage queues.
TCICSTRN	GCICSTRN	CICS transactions, normal attach security.

The user transactions for CICS are included in a sample called DFH\$CAT2 CLIST that uses GCICSTRN.

Table 62. Resources supported by the miscellaneous CICS resource class

Resource type	Prefix
Atom services	ATOMSERVICE
Bundle	BUNDLE
Document template	DOCTEMPLATE
EP adapters	EPADAPTER
EP adapter sets	EPADAPTERSET
Event bindings	EVENTBINDING
JVM Endpoint	JVMENDPOINT
JVM server	JVMSERVER
Pipeline	PIPELINE
URI map	URIMAP
Web Service	WEBSERVICE
XML transform	XMLTRANSFORM

Profile names in the RCICSRES class must contain a prefix that specifies the CICS resource type to which they apply: for example, BUNDLE.profile-name for bundles.

Table 63. RACF Provided Resource Classes for CICSplex SM

General resource class (member class)	Resource group class	Description
CPSMOBJ	GCPSMOBJ	Controls access to CICSplex SM resources.
CPSMXMP	Not applicable	Controls exemption from simulated security checking in CICSplex SM.

Table 64. RACF-provided resource classes for system resources used by CICS

General resource class (member class)	Resource group class	Description	Useful links
APPCLU		Verifies the identity of APPC partner logical units (LU type 6.2) during z/OS Communications Server session establishment.	
APPL		Controls terminal users' access to z/OS Communications Server applications, including CICS.	<a href="#">Defining specific APPL definitions and APPL parameters to SNA</a> <a href="#">Authorizing access to the CICS region</a>
CONSOLE		Controls user access to consoles.	<a href="#">Console profiles</a>
DIGTCERT	-	Contains digital certificates and information that is related to them.	<a href="#">Creating new RACF certificates</a>
FACILITY		Protect several different system resources.	See <a href="#">Table 65 on page 640</a>
FIELD		Controls access to fields in RACF profiles.	<a href="#">Controlling access to fields in RACF profiles</a>
IDIDMAP		The IDIDMAP resource profile contains the distributed identity filters. These filters map a RACF user ID and one or more distributed identities.	<a href="#">Configuring RACF for identity propagation</a> <a href="#">Configuring Liberty to use an LDAP user registry</a>
JESSPOOL		Protects JES spool data sets.	<a href="#">JES spool protection in a CICS environment</a>
LOGSTRM		Controls access to the MVS logstreams that CICS uses for its system logs and general logs.	<a href="#">Authorizing access to MVS log streams</a> <a href="#">Authorizations for CICS regions</a>

Table 64. RACF-provided resource classes for system resources used by CICS (continued)

General resource class (member class)	Resource group class	Description	Useful links
OPERCMD5		<ul style="list-style-type: none"> <li>Controls which console users are allowed to issue MODIFY commands that are directed to specific CICS regions.</li> <li>Controls which operator commands CICS can issue; for example, commands in the command list table (CLT), and MODIFY network commands.</li> </ul>	<a href="#">Using an MVS system console as a CICS terminal</a>
PROGRAM		Controls which users can start CICS.	<a href="#">Protecting CICS load libraries</a> <a href="#">Protecting CICS load module data sets</a>
PROPCNTL		Prevents the CICS region user ID being propagated to jobs that are submitted from CICS to the JES internal reader, and that do not specify the USER operand.	<a href="#">Controlling userid propagation</a>
PTKTDATA		Contains the encryption keys that are used for generating and validating PassTickets.	<a href="#">Implementing PassTickets for secure sign-on</a> <a href="#">Supporting parallel PassTicket requests for the same user ID</a>
STARTED		Contains profiles that provide the user IDs for MVS started jobs.	<a href="#">Example STARTED profiles for started jobs</a>
SUBSYSNM		Authorizes subsystems (such as instances of CICS) to open a VSAM ACB and use VSAM Record Level Sharing (RLS) functions.	<a href="#">Authorizing access to an SMSVSAM server</a> <a href="#">Authorizing access to SMSVSAM servers</a> <a href="#">Authorizing CICS user IDs for VSAM RLS support</a>
SURROGAT		Specifies which user IDs can act as surrogates for other user IDs.	<a href="#">Surrogate job submission in a CICS environment</a>
TERMINAL	GTERMINL	Controls the ability of users to sign on at individual terminals.	<a href="#">Example terminal profiles</a> <a href="#">Universal access authority for undefined terminals</a> <a href="#">Defining port of entry profiles</a> <a href="#">Terminal profiles</a>

Table 64. RACF-provided resource classes for system resources used by CICS (continued)

General resource class (member class)	Resource group class	Description	Useful links
VTAMAPPL		Controls the ability of users to open an SNA ACB. .VTAM is now z/OS Communications Server (for SNA or IP).	<a href="#">Controlling the opening of a CICS region's z/OS Communications Server ACB</a> <a href="#">Authorizing access to a CICS region z/OS Communications Server ACB</a>

If a profile is not defined, the default access is NONE. The exceptions are the following classes:

Table 65. Exception classes

Class	Reference
APPL	For more information, see <a href="#">.RACF users and roles</a>
CONSOLE	For more information, see <a href="#">Controlling access to CICS from specific ports of entry</a> .
TERMINAL	For more information, see <a href="#">Universal access authority for undefined terminals</a> .

Table 66. Resources protected by the FACILITY general resource class

CICS resource	What the FACILITY class controls
API and SPI restrictions from the DFHAPIR parmlib member	Whether users can access the DFHAPIR parmlib member, which contains rules that identify restricted CICS API and SPI commands. For more information, see <a href="#">API and SPI restrictions DFHAPIR parmlib member</a> .
CICSplex SM resources	Access to many CICSplex SM resources. For more information, see <a href="#">Security planning for CICSplex SM</a> .
Coupling facility data tables	<ul style="list-style-type: none"> <li>Whether a CICS region can connect to a coupling facility data table (CFDT).</li> <li>Whether the CFDT server can act as a server for a CFDT pool</li> <li>Whether the CFDT server region can connect to the coupling facility list structure for its CFDT pool.</li> </ul> For more information, see <a href="#">Security for coupling facility data tables</a> .
EXCI	Whether a client program can connect to a CICS region. For more information, see <a href="#">Using MRO logon and bind-time security</a> .
HPO system initialization parameter overrides	Whether the HPO can be specified as a SIT override.. For more information, see <a href="#">Authorizing use of HPO in PARM parameter on EXEC PGM=DFHSIP statement or in SYSIN data set</a> .

Table 66. Resources protected by the FACILITY general resource class (continued)

CICS resource	What the FACILITY class controls
Library lookaside (LLA) libraries	Whether a program can use the LLACOPY macro. For more information, see <a href="#">Authorizing access with the MVS library lookaside (LLA) facility</a> .
Log streams	Whether a CICS region can connect to the coupling facility list structures used for MVS log streams.
Multi Region Operation (MRO)	Whether a CICS region can use MRO and which CICS regions can connect to each other. For more information, see <a href="#">Configuring security for MRO</a> .
Named counter servers	<ul style="list-style-type: none"> <li>Whether a CICS region can connect to a named counter server.</li> <li>Whether the named counter server can act as server for a named counter pool.</li> <li>Whether the named counter server region can connect to the coupling facility list structure for its named counter pool.</li> </ul> For more information, see <a href="#">Authorizing access to named counter pools and servers</a> .
Shared data tables (SDT)	Whether a file-owning region can act as a shared data table server. For more information, see <a href="#">Security for CICS shared data tables</a> .
Shared temporary storage	<ul style="list-style-type: none"> <li>Whether a CICS region can connect to a shared temporary storage (TS) server.</li> <li>Whether the shared TS server can act as server for a shared TS pool.</li> <li>Whether the shared TS server region can connect to the coupling facility list structure for its shared TS pool.</li> </ul>
Transport Level Security (TLS)	Security administrator access to build a Keyring. For more information, see <a href="#">Setting up profiles in RACF</a> .

## CICS transactions subject to security checking

6.2 Applies to 6.2.

To conform with a zero trust strategy and various compliance regulations such as PCI-DSS, as of CICS TS 6.2, all CICS transactions, excluding CJXA and CICSplex SM transactions (CO\*\*), are defined with CMDSEC (YES) and RESSEC (YES). This might impact your security definitions.

All CICS transactions, excluding CJXA and CICSplex SM transactions (CO\*\*), are defined with CMDSEC (YES) and RESSEC (YES) to make it easier to identify transactions that do not conform to the recommendations, and thereby make it easier to audit CICS. In practice:

- Category 1 transaction are exempt from security checking because they are part of CICS and can only run as part of the CICS system under the CICS region user ID.
- For Category 2 transactions, some require resource and command security in addition to transaction security. Follow advice in [Table 68 on page 642](#) to configure security for them. The remaining Category 2 transactions require only transaction security because they only use CICS code to perform their function.

- Category 3 transactions are available to all users, whether signed-on or not. These transactions are not subject to security checking.
- CJXA is usually used for testing purposes and CICSplex SM WUI and CMAS regions do not run user code. Therefore, CJXA and CICSplex SM transactions (CO\*\*) stay unchanged.

The following table lists CICS Category 1 transactions that require extra security to be configured.

<i>Table 67. CICS Category 1 transactions requiring extra security</i>	
<b>Transactions</b>	<b>Recommended security configuration</b>
CPLT	Security for PLT programs. See <a href="#">PLTPIUSR</a> .
CWXN	Security for static responses. See <a href="#">Resource-level security for static responses using document templates</a> .

The following table lists CICS Category 2 transactions that require extra security to be configured in addition to transaction security.

<i>Table 68. CICS Category 2 transactions requiring extra security in addition to transaction security</i>	
<b>Transactions</b>	<b>Recommended security configuration</b>
CEBR	Users need access to any TS queue that they browse.
CEDA	Users need surrogate authority to install definitions that contain user IDs.
CESD	See <a href="#">Security for CICS shutdown</a> .
CKAM	Users need to define transaction security by specifying the transaction attribute for the MQMONITOR resource. See <a href="#">Defining and installing MQMONITOR resources</a> .  The specified transaction can be the default CKTI, user MQ adapter transaction, or CKBR. See <a href="#">Setting up an MQMONITOR resource for the CICS-MQ bridge</a> .  Learn more: <a href="#">Security for the CICS-MQ bridge</a> , <a href="#">Security for the CICS-MQ adapter</a>
CKBC	Users need to define security on transactions specified in the request message body. See <a href="#">DPL message structure for the CICS-MQ bridge</a> .
CKBP	Users need to define security on transactions specified in the request message body. See <a href="#">DPL message structure for the CICS-MQ bridge</a> .
CKBR	Users need permission to run transaction CKBC or CKBP.  Learn more: <a href="#">Security for the CICS-MQ bridge</a>
CKCN	Users need permission to run transaction CKRT.
CKDP	Users need permission to run transaction CKRT.
CKQC	Users need permission to run transactions CKCN, CKDL, CKRS, CKSD, and CKSQ.  Users might need permission to run transaction CKAM when starting or stopping the MQ connection. See <a href="#">Starting a CICS-MQ connection</a> and <a href="#">Stopping a CICS-MQ connection</a> .  Learn more: <a href="#">Security for the CICS-MQ adapter</a>
CKRS	Users need permission to run transaction CKRT.
CKRT	Users need permission to run transactions CKBM and CKDB.
CKSD	Users need permission to run transaction CKRT.
CKSQ	Users need permission to run transactions CKTI and CKRT.

Table 68. CICS Category 2 transactions requiring extra security in addition to transaction security (continued)

Transacti ons	Recommended security configuration
CKTI	Users need to define transaction security by specifying the transaction attribute for the MQMONITOR resource. See <a href="#">Defining and installing MQMONITOR resources</a> .
CLER	Users need READ and UPDATE access to the temporary storage queues (TS QUEUES) whose names begin with "TR1".
CPIL	Users need to configure permission of transactions driven by CPIL. <b>Note:</b> CPIL drives CPIH or CIPQ by default, but you can specify a different transaction or user in the URIMAP resource definition. URIMAP routing determines which user ID and transaction will be used. For more information, see <a href="#">URIMAP resources</a> and <a href="#">Routing provider mode CICS Web Services</a> .
CPMI	Configure security for the intercommunication method in use. See <a href="#">Implementing LU6.2 security</a> .
CRPA	See CRPC.
CRPC	Users need to configure security for commands issued by this transaction, such as SYSTEM for the <b>INQUIRE SYSTEM</b> command, and any other related resources, including any programs that are called (see the DFHRPC row in Table 1 in <a href="#">CICS-supplied groups not in DFHLIST</a> ).
CRPM	See CRPC.
CSHR	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , and <a href="#">Implementing LU6.1 security</a> .
CSMI	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , <a href="#">Implementing LU6.2 security</a> , and <a href="#">Implementing LU6.1 security</a> .
CSM1	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , <a href="#">Implementing LU6.2 security</a> , and <a href="#">Implementing LU6.1 security</a> .
CSM2	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , <a href="#">Implementing LU6.2 security</a> , and <a href="#">Implementing LU6.1 security</a> .
CSM3	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , <a href="#">Implementing LU6.2 security</a> , and <a href="#">Implementing LU6.1 security</a> .
CSM5	Configure security for the intercommunication method in use. See <a href="#">Intercommunication security</a> , <a href="#">Security for MRO</a> , <a href="#">Security for IPIC (IP interconnectivity)</a> , <a href="#">Implementing LU6.2 security</a> , and <a href="#">Implementing LU6.1 security</a> .
CWBA	See <a href="#">Resource and transaction security for application-generated responses</a> .
CVMI	Configure security for the intercommunication method in use. See <a href="#">Implementing LU6.2 security</a> .

These remaining changed CICS Category 2 transactions only require transaction security. No additional security definitions are required to run these transactions:

- CADP
- CDBC
- CDBF
- CDBI
- CDBM
- CDBQ
- CDBT
- CDFS
- CECS
- CEDB
- CEDC
- CEOT
- CETR
- CIDP
- CKBM
- CKDL
- CLDM
- CMSG
- CPIA
- CPIH
- CPIQ
- CPIW
- CRTE
- CRTX
- CSFE
- CWTO
- CXSD
- DSNC

Obsolete or stabilized transactions also have command security and resource security enabled. Use them at your own risk. For a complete list of CICS transactions, see [List of CICS transactions](#).

## Commands subject to command security

---

These commands are subject to command security. If you enable command security, security checks are performed for these commands, when they are issued from a CICS application and for the equivalent commands that you can issue with the CEMT main terminal transaction or CECI. CICS calls RACF to check that the user that initiated the transaction has authority to use the command for the specified resource.

If a user is authorized to initiate the CEMT transaction, but is not authorized for the resources on which the commands depend, CICS returns a NOTAUTH condition. To allow use of the CEMT command in a command security environment, give users UPDATE access to the group profile that protects commands on which you want them to issue the PERFORM, SET, and DISCARD command and provide READ access to the group profile that protects the commands on which you want them to issue only INQUIRE and COLLECT commands.



Table 69. Access required for system programming commands

Command name	Access required
COLLECT CSD DISCONNECT CSD ENDBRGROUP CSD ENDBRLIST CSD ENDBRRSRCE CSD GETNEXTGROUP CSD GETNEXTLIST CSD GETNEXTRSRCE CSD INQUIREGROUP CSD INQUIRELIST CSD INQUIRERSRCE CSD STARTBRGROUP CSD STARTBRLIST CSD STARTBRRSRCE EXTRACT STATISTICS INQUIRE	READ
DISABLE CSD ADD CSD ALTER CSD APPEND CSD COPY CSD DEFINE CSD DELETE CSD LOCK CSD REMOVE CSD RENAME CSD UNLOCK CSD USERDEFINE ENABLE EXTRACT (but not EXTRACT STATISTICS) PERFORM RESYNC SET	UPDATE
CREATE CSD INSTALL DISCARD	ALTER

## Resources subject to command security checking

For transaction and resource security checking, you identify the resources to RACF by using the identifiers that you assigned to them; for example, file names, queue names, and transaction names. However, for command security, the resource identifiers are all predefined by CICS, and you use these predefined names when you define resource profiles to RACF or as the RESID values on the **QUERY SECURITY** command.

The full list of resource identifiers that are subject to command security checking with the associated commands is shown in [Table 70 on page 646](#). Most of these commands are common to both the CEMT and EXEC CICS interfaces; commands that are specific to CEMT have the CEMT prefix.

If you use prefixing, the value that is specified by the **SECPRFX** SIT parameter must be prefixed to the command resource name.

If you use `QUERY SECURITY` to query a user's access to resources, use identifiers from the **Resource identifier** column as the RESID values when issuing the `EXEC CICS QUERY SECURITY RESTYPE('SPCOMMAND')` command.

<i>Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking</i>	
<b>Resource identifier (RESID)</b>	<b>Related CICS commands</b>
ASSOCIATION	<b>INQUIRE ASSOCIATION SET ASSOCIATION USERCORRDATA</b>
ATOMSERVICE	<b>CREATE ATOMSERVICE DISCARD ATOMSERVICE INQUIRE ATOMSERVICE SET ATOMSERVICE</b>
AUTINSTMODEL	<b>DISCARD AUTINSTMODEL INQUIRE AUTINSTMODEL</b>
AUTOINSTALL	<b>INQUIRE AUTOINSTALL SET AUTOINSTALL</b>
BRFACILITY	<b>INQUIRE BRFACILITY SET BRFACILITY</b>
BUNDLE	<b>CREATE BUNDLE DISCARD BUNDLE INQUIRE BUNDLE SET BUNDLE</b>
BUNDLEPART	<b>INQUIRE BUNDLEPART</b>
CAPDATAPRED	<b>INQUIRE CAPDATAPRED</b>
CAPINFOSRCE	<b>INQUIRE CAPINFOSRCE</b>
CAPOPTPRED	<b>INQUIRE CAPOPTPRED</b>
CAPTURESPEC	<b>INQUIRE CAPTURESPEC</b>
CFDTPOOL	<b>INQUIRE CFDTPOOL</b>
CONNECTION	<b>CREATE CONNECTION DISCARD CONNECTION INQUIRE CONNECTION SET CONNECTION</b>

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
CSD	CSD ADD CSD ALTER CSD APPEND CSD COPY CSD DEFINE CSD DELETE CSD DISCONNECT CSD ENDBRGROUP CSD ENDBRLIST CSD ENDBRRSRCE CSD GETNEXTGROUP CSD GETNEXTLIST CSD GETNEXTSRCE CSD INQUIREGROUP CSD INQUIRELIST CSD INQUIRERSRCE CSD INSTALL CSD LOCK CSD REMOVE CSD RENAME CSD STARTBRGROUP CSD STARTBRLIST CSD STARTBRRSRCE CSD UNLOCK CSD USERDEFINE
DB2CONN	CREATE DB2CONN DISCARD DB2CONN INQUIRE DB2CONN SET DB2CONN
DB2ENTRY	CREATE DB2ENTRY DISCARD DB2ENTRY INQUIRE DB2ENTRY SET DB2ENTRY
DB2TRAN	CREATE DB2TRAN DISCARD DB2TRAN INQUIRE DB2TRAN SET DB2TRAN
DELETSHIPED	INQUIRE DELETSHIPED PERFORM DELETSHIPED SET DELETSHIPED
DISPATCHER	INQUIRE DISPATCHER SET DISPATCHER

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
DOCTEMPLATE	CREATE DOCTEMPLATE DISCARD DOCTEMPLATE INQUIRE DOCTEMPLATE SET DOCTEMPLATE
DSNAME	INQUIRE DSNAME SET DSNAME
DUMP	CEMT PERFORM SNAP PERFORM DUMP
DUMPCODE	CREATE DUMPCODE
DUMPDS	INQUIRE DUMPDS SET DUMPDS
ENQMODEL	CREATE ENQMODEL INQUIRE ENQMODEL SET ENQMODEL
EPADAPTER	INQUIRE EPADAPTER SET EPADAPTER <b>1</b>
EPADAPTERSET	INQUIRE EPADAPTERSET SET EPADAPTERSET <b>1</b>
EPADAPTINSET	INQUIRE EPADAPTINSET <b>1</b>
EVENTBINDING	INQUIRE EVENTBINDING SET EVENTBINDING <b>1</b>
EVENTPROCESS	INQUIRE EVENTPROCESS SET EVENTPROCESS
EXCI	INQUIRE EXCI
EXITPROGRAM	DISABLE PROGRAM ENABLE PROGRAM EXTRACT EXIT RESYNC ENTRYNAME INQUIRE EXITPROGRAM
FEPIRESOURCE	Certain <b>FEPI</b> commands

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
FILE	CREATE FILE DISCARD FILE INQUIRE FILE SET FILE
HOST	INQUIRE HOST SET HOST
IPCONN	CREATE IPCONN DISCARD IPCONN INQUIRE IPCONN SET IPCONN
IRC	INQUIRE IRC SET IRC
JOURNALMODEL	CEMT INQUIRE JMODEL CREATE JOURNALMODEL DISCARD JOURNALMODEL INQUIRE JOURNALMODEL
JOURNALNAME	INQUIRE JOURNALNAME SET JOURNALNAME
JVMENDPOINT	INQUIRE JVMENDPOINT SET JVMENDPOINT
JVMSERVER	CREATE JVMSERVER DISCARD JVMSERVER INQUIRE JVMSERVER PERFORM JVMSERVER SET JVMSERVER
LIBRARY	CREATE LIBRARY DISCARD LIBRARY INQUIRE LIBRARY SET LIBRARY  1
LINE	CEMT INQUIRE LINE CEMT SET LINE
LSRPOOL	CREATE LSRPOOL
MAPSET	CREATE MAPSET

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
MODENAME	INQUIRE MODENAME SET MODENAME
MONITOR	INQUIRE MONITOR SET MONITOR
MQCONN	CREATE MQCONN DISCARD MQCONN INQUIRE MQCONN SET MQCONN
MQMON	CREATE MQMONITOR DISCARD MQMONITOR INQUIRE MQMONITOR SET MQMONITOR
MVSTCB	COLLECT STATISTICS INQUIRE MVSTCB
NODEJSAPP	INQUIRE NODEJSAPP <sup>1</sup>
OSGIBUNDLE	INQUIRE OSGIBUNDLE <sup>1</sup>
OSGISERVICE	INQUIRE OSGISERVICE <sup>1</sup>
PARTITIONSET	CREATE PARTITIONSET
PARTNER	CREATE PARTNER DISCARD PARTNER INQUIRE PARTNER
PIPELINE	CREATE PIPELINE DISCARD PIPELINE INQUIRE PIPELINE PERFORM PIPELINE SET PIPELINE
POLICY	INQUIRE POLICY <sup>1</sup>
POLICYRULE	INQUIRE POLICYRULE <sup>1</sup>
PROCESSTYPE	CEMT INQUIRE PROCESSTYPE CEMT SET PROCESSTYPE CREATE PROCESSTYPE DISCARD PROCESSTYPE

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
PROFILE	<b>CREATE PROFILE</b> <b>DISCARD PROFILE</b> <b>INQUIRE PROFILE</b>
PROGRAM	<b>CREATE PROGRAM</b> <b>DISCARD PROGRAM</b> <b>INQUIRE PROGRAM</b> <b>SET PROGRAM</b> <b>SET PROGRAM REPLICATION</b> <b>1</b> <b>SET PROGRAM REPLICATION</b> has extra command security checking beyond SET PROGRAM.
REQID	<b>INQUIRE REQID</b>
RESETTIME	<b>PERFORM RESETTIME</b>
RRMS	<b>INQUIRE RRMS</b>
SECURITY	<b>6.2 INQUIRE SECDISCOVERY</b> <b>INQUIRE SECRECORDING</b> <b>6.2 PERFORM SECDISCOVERY</b> <b>PERFORM SECURITY REBUILD</b> <b>PERFORM SSL REBUILD</b> <b>6.2 SET SECDISCOVERY</b> <b>SET SECRECORDING</b>
SESSIONS	<b>CREATE SESSIONS</b>
SHUTDOWN	<b>PERFORM SHUTDOWN</b> Be particularly cautious when you authorize access to these and any other CICS commands that include a SHUTDOWN option.
STATISTICS	<b>COLLECT STATISTICS</b> <b>EXTRACT STATISTICS</b> <b>PERFORM STATISTICS RECORD</b> <b>INQUIRE STATISTICS</b> <b>SET STATISTICS</b>
STORAGE	<b>INQUIRE STORAGE</b> <b>INQUIRE STORAGE64</b>
STREAMNAME	<b>INQUIRE STREAMNAME</b>
SUBPOOL	<b>INQUIRE SUBPOOL</b>

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
SYSDUMPCODE	INQUIRE SYSDUMPCODE SET SYSDUMPCODE
SYSTEM	INQUIRE SYSTEM SET SYSTEM INQUIRE FEATUREKEY
TASK	INQUIRE TASK SET TASK
TCLASS	CREATE TRANCLASS DISCARD TRANCLASS INQUIRE TRANCLASS SET TRANCLASS INQUIRE TCLASS SET TCLASS
TCPIP	INQUIRE TCPIP SET TCPIP
TCPIPSERVICE	CREATE TCPIPSERVICE DISCARD TCPIPSERVICE INQUIRE TCPIPSERVICE SET TCPIPSERVICE
TDQUEUE	CREATE TDQUEUE DISCARD TDQUEUE INQUIRE TDQUEUE SET TDQUEUE
TEMPSTORAGE	INQUIRE TEMPSTORAGE SET TEMPSTORAGE
TERMINAL	INQUIRE NETNAME SET NETNAME CREATE TERMINAL DISCARD TERMINAL INQUIRE TERMINAL SET TERMINAL  <b>2</b>
TRACEDEST	INQUIRE TRACEDEST SET TRACEDEST
TRACEFLAG	INQUIRE TRACEFLAG SET TRACEFLAG



Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
TRACETYPE	INQUIRE TRACETYPE SET TRACETYPE
TRANDUMPCODE	INQUIRE TRANDUMPCODE SET TRANDUMPCODE
TRANSACTION	CREATE TRANSACTION DISCARD TRANSACTION INQUIRE TRANSACTION SET TRANSACTION  1
TSMODEL	CREATE TSMODEL DISCARD TSMODEL INQUIRE TSMODEL
TSPool	INQUIRE TSPool
TSQUEUE	INQUIRE TSQUEUE
TSQNAME	INQUIRE TSQNAME SET TSQNAME
TYPETERM	CREATE TYPETERM
UOW	INQUIRE UOW SET UOW
UOWDSNFAIL	INQUIRE UOWDSNFAIL
UOWENQ	INQUIRE UOWENQ
UOWLINK	SET UOWLINK INQUIRE UOWLINK
URIMAP	CREATE URIMAP DISCARD URIMAP INQUIRE URIMAP SET URIMAP  1
VTAM	INQUIRE VTAM SET VTAM
WEB	INQUIRE WEB SET WEB

Table 70. Resource identifiers and associated commands for CICS resources subject to command security checking (continued)

Resource identifier (RESID)	Related CICS commands
WEBSERVICE	<b>CREATE WEBSERVICE</b> <b>DISCARD WEBSERVICE</b> <b>INQUIRE WEBSERVICE</b> <b>SET WEBSERVICE</b>
WLMHEALTH	<b>INQUIRE WLMHEALTH</b> <b>SET WLMHEALTH</b>
XMLTRANSFORM	<b>INQUIRE XMLTRANSFORM</b> <b>SET XMLTRANSFORM</b>

#### Notes:

1. Bundle command security applies when you use SPI commands to perform an action on a BUNDLE resource, and in that process you install, enable, disable, or discard a dynamically generated resource of this type that was defined in the CICS bundle. No CICS command security applies when you install, enable, disable, or discard a dynamically generated resource of this type through an application or platform. For more information, see [Security for bundles](#).
2. **6.2** For **INQUIRE TERMINAL**, **INQUIRE NETNAME**, and **SET TERMINAL**, command security checking is not performed if the task or program that issues the command was started or attached to the same terminal that is being inquired or modified by the command. This is because resource security checking was already performed on the terminal when the program or task was started or attached to the terminal. The following options are exceptions, where the command security checking is still performed:
  - Browsing options (START, NEXT, and END) on **INQUIRE TERMINAL** or **INQUIRE NETNAME**
  - Tracing options (EXITTRACING, TRACING, and ZCPTRACING), the naming option (OPERID), and the purging option (PURGETYPE) on **SET TERMINAL**

#### Resource profile examples: command security checking

Define resource profiles to RACF, with access lists as required, by using the resource names in [Table 70 on page 646](#) as the profile names. Alternatively, you can create resource group profiles in the VCICSCMD class.

In the following example, the **RDEFINE** command defines a profile that is named CMDSAMP. The commands that are protected by this profile are specified on the ADDMEM operand. The PERMIT command allows a group of users to issue the commands for INQUIRE:

```
RDEFINE VCICSCMD CMDSAMP UACC(NONE)
          NOTIFY(sys_admin_userid)
          ADDMEM(AUTINSTMODEL, AUTOINSTALL, CONNECTION,
                DSNAME, TRANSACTION, TRANDUMPCODE, VTAM)
PERMIT CMDSAMP CLASS(VCICSCMD) ID(operator_group) ACCESS(READ)
```

The second example defines a profile that is called CMDSAMP1 with the same commands in the ADDMEM operand, as in the previous example. The **PERMIT** command allows a group of users to issue PERFORM, SET, and DISCARD against these commands:

```
RDEFINE VCICSCMD CMDSAMP1 UACC(NONE)
          NOTIFY(sys_admin_userid)
          ADDMEM(AUTINSTMODEL, AUTOINSTALL, CONNECTION,
```

DSNAME, TRANSACTION, TRANDUMPCODE, VTAM)  
 PERMIT CMLSAMP1 CLASS(VCICSCMD) ID(op\_group\_2) ACCESS(UPDATE)

Users require the access levels that are shown in [Resource and command check cross-reference](#).

## Resource and command check cross-reference

Use this cross-reference to check commands against resources.

Table 71. Resource and command check cross-reference					
EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
ABEND					
ACQUIRE				UPDATE	TERMINAL
ACQUIRE FOR BTS See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
ADDRESS					
ALLOCATE					
ASKTIME					
BIF DEEDIT					
BIF DIGEST					
BUILD ATTACH					
CANCEL See note “1” on page 671	XPCT	READ	TRANSID		
CANCEL FOR BTS	XFCT	UPDATE	BTS REPOSITORY FILE		
CHANGE PASSWORD					
CHANGE TASK					
COLLECT STATISTICS				READ	STATISTICS
COLLECT STATISTICS FILE	XFCT	READ	FILE	READ	STATISTICS
COLLECT STATISTICS JOURNALNAME	XJCT	READ	JOURNAL	READ	STATISTICS
COLLECT STATISTICS JOURNALNUM	XJCT	READ	JOURNAL	READ	STATISTICS
COLLECT STATISTICS PROGRAM	XPPT	READ	PROGRAM	READ	STATISTICS
COLLECT STATISTICS TDQUEUE	XDCT	READ	TDQUEUE	READ	STATISTICS
COLLECT STATISTICS TRANSACTION	XPCT	READ	TRANSID	READ	STATISTICS

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
CONNECT PROCESS					
CONVERSE					
CREATE ATOMSERVICE	XRES	ALTER	ATOMSERVICE.resource_name	ALTER	ATOMSERVICE
CREATE BUNDLE	XRES	ALTER	BUNDLE.resource_name	ALTER	BUNDLE
CREATE CONNECTION See note “2” on page 672				ALTER	CONNECTION
CREATE DB2CONN See note “3” on page 672		ALTER			DB2CONN
CREATE DB2ENTRY See note “3” on page 672	XDB2	ALTER	DB2ENTRY	ALTER	DB2ENTRY
CREATE DB2TRAN See note “3” on page 672	XDB2	ALTER	DB2TRAN	ALTER	DB2TRAN
CREATE DOCTEMPLATE	XRES	ALTER	DOCTEMPLATE.resource_name	ALTER	DOCTEMPLATE
CREATE ENQMODEL				ALTER	ENQMODEL
CREATE FILE	XFCT	ALTER	FILE	ALTER	FILE
CREATE IPCONN				ALTER	IPCONN
CREATE JOURNALMODEL				ALTER	JOURNALMODEL
CREATE JVMSERVER	XRES	ALTER	JVMSERVER.resource_name	ALTER	JVMSERVER
CREATE LIBRARY				ALTER	LIBRARY
CREATE LSRPOOL				ALTER	LSRPOOL
CREATE MAPSET	XPPT	ALTER	MAPSET	ALTER	MAPSET
CREATE MQCONN				ALTER	MQCONN
CREATE MQMONITOR				ALTER	MQMON
CREATE PARTITIONSET	XPPT	ALTER	PARTITIONSET	ALTER	PARTITIONSET
CREATE PARTNER				ALTER	PARTNER
CREATE PIPELINE				ALTER	PIPELINE

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
CREATE PROCESSTYPE See note “10” on page 672				ALTER	PROCESSTYPE
CREATE PROFILE				ALTER	PROFILE
CREATE PROGRAM	XPPT	ALTER	PROGRAM	ALTER	PROGRAM
CREATE SESSIONS See note “3” on page 672				ALTER	SESSIONS
CREATE TCPIP SERVICE				ALTER	TCPIP SERVICE
CREATE TDQUEUE See note “3” on page 672	XDCT	ALTER	TDQUEUE	ALTER	TDQUEUE
CREATE TERMINAL See note “3” on page 672				ALTER	TERMINAL
CREATE TRANCLASS				ALTER	TCLASS
CREATE TRANSACTION	XPCT	ALTER	TRANSID	ALTER	TRANSACTION
CREATE TSMODEL				ALTER	TSMODEL
CREATE TYPETERM				ALTER	TYPETERM
CREATE URIMAP				ALTER	URIMAP
CREATE WEBSERVICE				ALTER	WEBSERVICE
CSD ADD				UPDATE	CSD
CSD ALTER				UPDATE	CSD
CSD APPEND				UPDATE	CSD
CSD COPY				UPDATE	CSD
CSD DEFINE				UPDATE	CSD
CSD DELETE				UPDATE	CSD
CSD DISCONNECT				READ	CSD
CSD ENDBRGROUP				READ	CSD
CSD ENDBRLIST				READ	CSD

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
CSD ENDBRRSRCE				READ	CSD
CSD GETNEXTGROUP				READ	CSD
CSD GETNEXTLIST				READ	CSD
CSD GETNEXTRSRCE				READ	CSD
CSD INQUIREGROUP				READ	CSD
CSD INQUIRELIST				READ	CSD
CSD INQUIRERSRCE				READ	CSD
CSD INSTALL				ALTER	CSD
CSD LOCK				UPDATE	CSD
CSD REMOVE				UPDATE	CSD
CSD RENAME				UPDATE	CSD
CSD STARTBRGROUP				READ	CSD
CSD STARTBRLIST				READ	CSD
CSD STARTBRRSRCE				READ	CSD
CSD UNLOCK				UPDATE	CSD
CSD USERDEFINE				UPDATE	CSD
DEFINE ACTIVITY See notes “7” on page 672 and “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
DEFINE PROCESS See notes “7” on page 672 and “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
DELAY					
DELETE	XFCT	UPDATE	FILE		
DELETE ACTIVITY See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
DELETEQ TD	XDCT	UPDATE	TDQUEUE		
DELETEQ TS See note “4” on page 672	XTST	UPDATE	TSQUEUE		
DEQ					

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
DISABLE PROGRAM	XPPT	UPDATE	PROGRAM	UPDATE	EXITPROGRAM
DISCARD ATOMSERVICE	XRES	ALTER	ATOMSERVICE. <i>resource_name</i>	ALTER	ATOMSERVICE
DISCARD AUTINSTMODEL				ALTER	AUTINSTMODEL
DISCARD BUNDLE	XRES	ALTER	BUNDLE. <i>resource_name</i>	ALTER	BUNDLE
DISCARD CONNECTION					
DISCARD DB2CONN				ALTER	DB2CONN
DISCARD DB2ENTRY	XDB2	ALTER	DB2ENTRY	ALTER	DB2ENTRY
DISCARD DB2TRAN	XDB2	ALTER	DB2TRAN	ALTER	DB2TRAN
DISCARD DOCTEMPLATE	XRES	ALTER	DOCTEMPLATE. <i>resource_name</i>	ALTER	DOCTEMPLATE
DISCARD ENQMODEL				ALTER	ENQMODEL
DISCARD FILE	XFCT	ALTER	FILE	ALTER	FILE
DISCARD IPCONN				ALTER	IPCONN
DISCARD JOURNALMODEL				ALTER	JOURNALMODEL
DISCARD JOURNALNAME	XJCT	ALTER	JOURNAL	ALTER	JOURNALNAME
DISCARD JVMSERVER	XRES	ALTER	JVMSERVER. <i>resource_name</i>	ALTER	JVMSERVER
DISCARD LIBRARY				ALTER	LIBRARY
DISCARD MQCONN				ALTER	MQCONN
DISCARD MQMONITOR				ALTER	MQMON
DISCARD PARTNER				ALTER	PARTNER
DISCARD PIPELINE				ALTER	PIPELINE
DISCARD PROCESSTYPE <a href="#">See note “10” on page 672</a>				ALTER	PROCESSTYPE
DISCARD PROFILE				ALTER	PROFILE
DISCARD PROGRAM	XPPT	ALTER	PROGRAM	ALTER	PROGRAM
DISCARD TCPIPSERVICE				ALTER	TCPIPSERVICE

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
DISCARD TDQUEUE	XDCT	ALTER	TDQUEUE	ALTER	TDQUEUE
DISCARD TERMINAL				ALTER	TERMINAL
DISCARD TRANCLASS				ALTER	TCLASS
DISCARD TRANSACTION	XPCT	ALTER	TRANSID	ALTER	TRANSACTION
DISCARD TSMODEL				ALTER	TSMODEL
DISCARD URIMAP				ALTER	URIMAP
DISCARD WEBSERVICE				ALTER	WEBSERVICE
DOCUMENT CREATE See note <a href="#">14</a>	XRES	READ	DOCTEMPLATE.resource_name		
DOCUMENT INSERT See note <a href="#">14</a>	XRES	READ	DOCTEMPLATE.resource_name See note <a href="#">13</a>		
DUMP TRANSACTION					
ENABLE PROGRAM	XPPT	UPDATE	PROGRAM	UPDATE	EXITPROGRAM
ENDBR See note “5” on page <a href="#">672</a>					
ENQ					
ENTER TRACENUM					
EXTRACT					
EXTRACT EXIT	XPPT	READ	PROGRAM	UPDATE	EXITPROGRAM
EXTRACT STATISTICS				READ	STATISTICS
FEPI					FEPI
FORMATTIME					
FREE					
FREEMAIN					
GDS					
GETMAIN					
HANDLE ABEND PROGRAM	XPPT	READ	PROGRAM		
HANDLE AID					



Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
HANDLE CONDITION					
IGNORE CONDITION					
INQUIRE ACTIVITYID See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
INQUIRE ASSOCIATION				READ	ASSOCIATION
INQUIRE ATOMSERVICE	XRES	READ	ATOMSERVICE.resource_name	READ	ATOMSERVICE
INQUIRE AUTINSTMODEL				READ	AUTINSTMODEL
INQUIRE AUTOINSTALL				READ	AUTOINSTALL
INQUIRE BRFACILITY				READ	BRFACILITY
INQUIRE BUNDLE	XRES	READ	BUNDLE.resource_name	READ	BUNDLE
INQUIRE BUNDLEPART	XRES	READ	BUNDLE.resource_name	READ	BUNDLEPART
INQUIRE CAPDATAPRED	XRES	READ	EVENTBINDING.resource_name	READ	CAPDATAPRED
INQUIRE CAPINFOSRCE	XRES	READ	EVENTBINDING.resource_name	READ	CAPINFOSRCE
INQUIRE CAOPTPRED	XRES	READ	EVENTBINDING.resource_name	READ	CAOPTPRED
INQUIRE CAPTURESPEC	XRES	READ	EVENTBINDING.resource_name	READ	CAPTURESPEC
INQUIRE CFDTPOOL				READ	CFDTPOOL
INQUIRE CONNECTION				READ	CONNECTION
INQUIRE CONTAINER See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
INQUIRE DB2CONN				READ	DB2CONN
INQUIRE DB2ENTRY	XDB2	READ	DB2ENTRY	READ	DB2ENTRY
INQUIRE DB2TRAN	XDB2	READ	DB2TRAN	READ	DB2TRAN
INQUIRE DELETSHIPED				READ	DELETSHIPED

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
INQUIRE DOCTEMPLATE	XRES	READ	DOCTEMPLATE.resource_name	READ	DOCTEMPLATE
INQUIRE DSNAME				READ	DSNAME
INQUIRE DUMPDS				READ	DUMPDS
INQUIRE ENQMODEL				READ	ENQMODEL
INQUIRE EPADAPTER	XRES	READ	EPADAPTER.resource_name	READ	EPADAPTER
INQUIRE EPADAPTERSET	XRES	READ	EPADAPTERSET.resource_name	READ	EPADAPTERSET
INQUIRE EPADAPTINSET	XRES	READ	EPADAPTERSET.resource_name	READ	EPADAPTINSET
INQUIRE EVENT See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
INQUIRE EVENTBINDING	XRES	READ	EVENTBINDING.resource_name	READ	EVENTBINDING
INQUIRE EVENTPROCESS				READ	EVENTPROCESS
INQUIRE EXCI				READ	EXCI
INQUIRE EXITPROGRAM	XPPT	READ	PROGRAM	READ	EXITPROGRAM
INQUIRE FEATUREKEY				READ	SYSTEM
INQUIRE FILE	XFCT	READ	FILE	READ	FILE
INQUIRE HOST				READ	HOST
INQUIRE IPCONN				READ	IPCONN
INQUIRE IRC				READ	IRC
INQUIRE JOURNALMODEL				READ	JOURNALMODEL
INQUIRE JOURNALNAME	XJCT	READ	JOURNAL	READ	JOURNAL
INQUIRE JVMENDPOINT				READ	JVMENDPOINT
INQUIRE JVMSERVER	XRES	READ	JVMSERVER.resource_name	READ	JVMSERVER
INQUIRE LIBRARY				READ	LIBRARY
INQUIRE MODENAME				READ	MODENAME
INQUIRE MONITOR				READ	MONITOR

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
INQUIRE MQCONN				READ	MQCONN
INQUIRE MQINI				READ	MQMON
INQUIRE MQMONITOR				READ	MQMON
INQUIRE MVSTCB				READ	MVSTCB
INQUIRE NETNAME  6.2 See note “18” on page 672.				READ	TERMINAL
INQUIRE NODEJSAPP	XRES	READ	BUNDLE.resource_name	READ	NODEJSAPP
INQUIRE OSGIBUNDLE	XRES	READ	BUNDLE.resource_name	READ	OSGIBUNDLE
INQUIRE OSGISERVICE	XRES	READ	BUNDLE.resource_name	READ	OSGISERVICE
INQUIRE PARTNER				READ	PARTNER
INQUIRE PIPELINE				READ	PIPELINE
INQUIRE POLICY	XRES	READ	POLICY.resource_name	READ	POLICY
INQUIRE POLICYRULE	XRES	READ	POLICY.resource_name	READ	POLICYRULE
INQUIRE PROCESS See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
INQUIRE PROCESSTYPE See note “10” on page 672	XPTT			READ	PROCESSTYPE
INQUIRE PROFILE				READ	PROFILE
INQUIRE PROGRAM	XPPT	READ	PROGRAM	READ	PROGRAM
INQUIRE REQID See note “8” on page 672	XPCT	READ	TRANSID	READ	REQID
INQUIRE RRMS				READ	RRMS
6.2 INQUIRE SECDISCOVERY				READ	SECURITY

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
INQUIRE SECRECORDING				READ	SECURITY
INQUIRE STATISTICS				READ	STATISTICS
INQUIRE STORAGE INQUIRE STORAGE64				READ	STORAGE
INQUIRE STREAMNAME				READ	STREAMNAME
INQUIRE SUBPOOL				READ	SUBPOOL
INQUIRE SYSDUMPCODE				READ	SYSDUMPCODE
INQUIRE SYSTEM				READ	SYSTEM
INQUIRE TASK				READ	TASK
INQUIRE TCLASS				READ	TCLASS
INQUIRE TCPIP				READ	TCPIP
INQUIRE TCPIPSERVICE				READ	TCPIPSERVICE
INQUIRE TDQUEUE	XDCT	READ	TDQUEUE	READ	TDQUEUE
INQUIRE TEMPSTORAGE				READ	TEMPSTORAGE
INQUIRE TERMINAL See note “18” on page 672.				READ	TERMINAL
INQUIRE TIMER See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
INQUIRE TRACEDEST				READ	TRACEDEST
INQUIRE TRACEFLAG				READ	TRACEFLAG
INQUIRE TRACETYPE				READ	TRACETYPE
INQUIRE TRANCLASS				READ	TCLASS
INQUIRE TRANDUMPCODE				READ	TRANDUMPCODE
INQUIRE TRANSACTION	XPCT	READ	TRANSID	READ	TRANSACTION
INQUIRE TSMODEL				READ	TSMODEL

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
INQUIRE TSPOOL				READ	TSPOOL
INQUIRE TSQNAME See note “4” on page 672	XTST	READ	TSQNAME	READ	TSQUEUE
INQUIRE TSQUEUE See note “4” on page 672	XTST	READ	TSQUEUE	READ	TSQUEUE
INQUIRE UOW				READ	UOW
INQUIRE UOWDSNFAIL				READ	UOWDSNFAIL
INQUIRE UOWENQ				READ	UOWENQ
INQUIRE UOWLINK				READ	UOWLINK
INQUIRE URIMAP				READ	URIMAP
INQUIRE VTAM				READ	VTAM (now the z/OS Communications Server)
INQUIRE WEB				READ	WEB
INQUIRE WEBSERVICE				READ	WEBSERVICE
INQUIRE WLMHEALTH				READ	WLMHEALTH
INQUIRE XMLTRANSFORM	XRES	READ	XMLTRANSFORM.resource_name	READ	XMLTRANSFORM
ISSUE					
LINK	XPPT	READ	PROGRAM		
LINK ACQPROCESS See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
LINK ACTIVITY / ACQACTIVITY See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
LOAD	XPPT	READ	PROGRAM		
MONITOR					
PERFORM DELETSHIPED				UPDATE	DELETSHIPED

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
PERFORM DUMP				UPDATE	DUMP
PERFORM JVMSEVER				UPDATE	JVMSEVER
PERFORM PIPELINE				UPDATE	PIPELINE
PERFORM RESETTIME				UPDATE	RESETTIME
6.2 PERFORM SECDISCOVERY				UPDATE	SECURITY
PERFORM SECURITY REBUILD				UPDATE	SECURITY
PERFORM SHUTDOWN				UPDATE	SHUTDOWN
PERFORM SSL REBUILD				UPDATE	SECURITY
PERFORM STATISTICS				UPDATE	STATISTICS
POINT					
POP HANDLE					
POST					
PURGE MESSAGE					
PUSH HANDLE					
QUERY SECURITY See note “6” on page 672					
QUERY SECURITY USERID See note “17” on page 672					
READ	XFCT	READ	FILE		
READ NEXT See note “5” on page 672					
READ PREV See note “5” on page 672					

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
READQ TD	XDCT	UPDATE	TDQUEUE		
READQ TS See note “4” on page 672	XTST	READ	TSQUEUE		
RECEIVE					
RELEASE	XPPT	READ	PROGRAM		
RESET ACTIVITY See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
RESET ACQPROCESS See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
RESETBR See note “5” on page 672					
RESYNC ENTRYNAME				UPDATE	EXITPROGRAM
RETRIEVE					
RETURN / RETURN ENDACTIVITY See note “9” on page 672 and “11” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
REWRITE	XFCT	UPDATE	FILE		
ROUTE					
RUN See note “9” on page 672 and “11” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
RUN / ASYNCH SYNC See note “9” on page 672	XFCT	UPDATE	DFHLRQ FILE		
SEND					
SET ASSOCIATION USERCORRDATA				UPDATE	ASSOCIATION
SET ATOMSERVICE	XRES	UPDATE	ATOMSERVICE.resource_name	UPDATE	ATOMSERVICE

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
SET AUTOINSTALL				UPDATE	AUTOINSTALL
SET BRFACILITY				UPDATE	BRFACILITY
SET BUNDLE	XRES	UPDATE	BUNDLE.resource_name	UPDATE	BUNDLE
SET CONNECTION				UPDATE	CONNECTION
SET DB2CONN				UPDATE	DB2CONN
SET DB2ENTRY	XDB2	UPDATE	DB2ENTRY	UPDATE	DB2ENTRY
SET DB2TRAN	XDB2	UPDATE	DB2TRAN	UPDATE	DB2TRAN
SET DELETSHIPED				UPDATE	DELETSHIPED
SET DOCTEMPLATE	XRES	UPDATE	DOCTEMPLATE.resource_name	UPDATE	DOCTEMPLATE
SET DSNAME				UPDATE	DSNAME
SET DUMPDS				UPDATE	DUMPDS
SET ENQMODEL				UPDATE	ENQMODEL
SET EPADAPTER	XRES	UPDATE	EPADAPTER.resource_name	UPDATE	EPADAPTER
SET EPADAPTERSET	XRES	UPDATE	EPADAPTERSET.resource_name	UPDATE	EPADAPTERSET
SET EVENTBINDING	XRES	UPDATE	EVENTBINDING.resource_name	UPDATE	EVENTBINDING
SET EVENTPROCESS	XRES	UPDATE	EVENTPROCESS.resource_name	UPDATE	EVENTPROCESS
SET FILE	XFCT	UPDATE	FILE	UPDATE	FILE
SET HOST				UPDATE	HOST
SET IPCONN				UPDATE	IPCONN
SET IRC				UPDATE	IRC
SET JOURNALNAME	XJCT	UPDATE	JOURNAL	UPDATE	JOURNAL
SET JVMENDPOINT				UPDATE	JVMENDPOINT
SET JVMSERVER	XRES	UPDATE	JVMSERVER.resource_name	UPDATE	JVMSERVER
SET LIBRARY				UPDATE	LIBRARY
SET MODENAME				UPDATE	MODENAME
SET MONITOR				UPDATE	MONITOR
SET MQCONN See note “13” on page 672				UPDATE	MQCONN
SET MQMONITOR				UPDATE	MQMON
SET NETNAME				UPDATE	TERMINAL
SET PIPELINE				UPDATE	PIPELINE



Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
SET PROCESSTYPE See note “10” on page 672	XPTT			UPDATE	PROCESSTYPE
SET PROGRAM	XPPT	UPDATE	PROGRAM	UPDATE	PROGRAM
SET PROGRAM REPLICATION See note “16” on page 672				ALTER	REPLICATION
6.2 SET SECDISCOVERY				UPDATE	SECURITY
SET SECRECORDING				UPDATE	SECURITY
SET STATISTICS				UPDATE	STATISTICS
SET SYSDUMPCODE See note 15				UPDATE	SYSDUMPCODE
SET SYSTEM				UPDATE	SYSTEM
SET TASK				UPDATE	TASK
SET TCLASS				UPDATE	TCLASS
SET TCPIP				UPDATE	TCPIP
SET TCPIPSERVICE				UPDATE	TCPIPSERVICE
SET TDQUEUE See note “3” on page 672	XDCT	UPDATE	TDQUEUE	UPDATE	TDQUEUE
SET TEMPSTORAGE				UPDATE	TEMPSTORAGE
SET TERMINAL 6.2 See note “18” on page 672.				UPDATE	TERMINAL
SET TRACEDEST				UPDATE	TRACEDEST
SET TRACEFLAG				UPDATE	TRACEFLAG
SET TRACETYPE				UPDATE	TRACETYPE
SET TRANCLASS				UPDATE	TCLASS
SET TRANDUMPCODE				UPDATE	TRANDUMPCODE
SET TRANSACTION	XPCT	UPDATE	TRANSID	UPDATE	TRANSACTION

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
SET TSQNAME See note “4” on page 672	XTST	UPDATE	TSQUEUE	UPDATE	TSQUEUE
SET TSQUEUE See note “4” on page 672	XTST	UPDATE	TSQNAME	UPDATE	TSQUEUE
SET UOW				UPDATE	UOW
SET UOWLINK				UPDATE	UOWLINK
SET URIMAP				UPDATE	URIMAP
SET VTAM				UPDATE	VTAM
SET WEB				UPDATE	WEB
SET WEBSERVICE				UPDATE	WEBSERVICE
SET WLMHEALTH				UPDATE	WLMHEALTH
SET XMLTRANSFORM	XRES	UPDATE	XMLTRANSFORM.resource_name	UPDATE	XMLTRANSFORM
SIGNOFF					
SIGNON					
SPOOLCLOSE					
SPOOLOPEN					
SPOOLREAD					
SPOOLWRITE					
START See note “7” on page 672	XPCT	READ	TRANSID		
STARTBR	XFCT	READ	FILE		
STARTBROWSE ACTIVITY See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
STARTBROWSE CONTAINER (BTS) See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		

Table 71. Resource and command check cross-reference (continued)

EXEC CICS COMMAND	RESOURCE CHECK			CHECK CLASS=XCMD	
	CLAS S	ACCESS	RESOURCE	ACCESS	RESOURCE
STARTBROWSE EVENT See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
STARTBROWSE PROCESS See note “9” on page 672	XFCT	READ	BTS REPOSITORY FILE		
SUSPEND See note “9” on page 672	XFCT	UPDATE	BTS REPOSITORY FILE		
SYNCPPOINT					
TCPIP					
UNLOCK					
VERIFY PASSWORD					
WAIT					
WAIT JOURNALNAME	XJCT	READ	JOURNAL		
WAIT JOURNALNUM	XJCT	READ	JOURNAL		
WAITCICS					
WEB					
WRITE	XFCT	UPDATE	FILE		
WRITE JOURNALNAME	XJCT	UPDATE	JOURNAL		
WRITE JOURNALNUM	XJCT	UPDATE	DFHJNN		
WRITE OPERATOR					
WRITEQ TD	XDCT	UPDATE	TDQUEUE		
WRITEQ TS See note “4” on page 672	XTST	UPDATE	TSQUEUE TSQNAME		
XCTL	XPPT	READ	PROGRAM		

**Notes:**

1. The **CANCEL** command can perform two checks. One check is performed against the transaction specified on the **CANCEL** command. The other check is performed against the transaction associated with the REQID that you are canceling where applicable.

2. The **CREATE CONNECTION** command is subject to command security checking when you define a connection. For example, **CREATE CONNECTION(CON1) attribute(...)**. However, when you use the **CREATE CONNECTION COMPLETE** or **CREATE CONNECTION DISCARD** command, no command security checking is performed unless you have been authorized to use **COMPLETE** and **DISCARD** commands. The **COMPLETE** and **DISCARD** commands can be used only by those authorized to perform **CREATE CONNECTION(CON1)** and **CREATE SESSIONS(SES1)** commands. Otherwise, ILLOGIC is returned.
3. An install surrogate user check can also occur. For more information, see [Surrogate security](#).
4. A security check is performed for temporary storage queues when SECURITY(YES) is specified in the relevant TSMODEL resource definition, or, if you use a temporary storage table (TST), when a DFHTST TYPE=SECURITY macro is coded for the temporary storage queue.
5. No security check is performed, because the **STARTBR** command must be issued before this command. A security check is issued on the **STARTBR** command.
6. The **QUERY SECURITY** command is not controlled by resource or command checks, but it can cause them to be issued.
7. A start surrogate user check can also occur.
8. The resource check for the transid is performed only if the REQID is associated with a transaction.
9. CICS BTS API commands.
10. CICS BTS commands that are subject to command security. All other CICS BTS commands are not subject to command-level security.
11. Any BTS commands that use timing operands access the BTS LRQ file.
12. The **EXEC CICS DOCUMENT CREATE** and **INSERT** commands reference document templates using the 48-character name of the template as specified in the TEMPLATENAME attribute of the DOCTEMPLATE resource definition. However, security checking for these commands uses the name of the DOCTEMPLATE resource definition that corresponds to the TEMPLATENAME attribute.
13. When command security is active for your CICS region, to use the **EXEC CICS SET MQCONN** command to start or stop the connection to IBM MQ, users must have authority to use not only the **EXEC CICS SET MQCONN** command, but also the **EXEC CICS EXTRACT EXIT** command. If a user attempts to start or stop the connection without the authority to use the **EXEC CICS EXTRACT EXIT** command, CICS issues messages DFHXS1111 and DFHMQ0302. When resource security for programs (XPPT) is active for your CICS region, when using SET to set the Connection Status to Connected, the user must have sufficient authority to run the supplied CICS MQ programs.
14. A minimum of READ access is required for CICS document templates used as static Web pages, document templates used by application programs as part of application-generated responses, and document templates used with all **EXEC CICS DOCUMENT INSERT** and **EXEC CICS DOCUMENT CREATE** commands with the TEMPLATE option.
15. For CHECK CLASS=XCMD, you must have at least CONTROL access to update the **JOBLIST** or **DSPLIST** option on **SET SYSDUMPCODE**.
16. If command security checking is active, besides UPDATE access already required by **SET PROGRAM**, to issue **SET PROGRAM REPLICATION**, you must also have ALTER access to the RACF REPLICATION resource. REPLICATION is treated as a CICS resource in the RACF definitions.
17. A surrogate check for userid.DFHQUERY. Task user must have at least READ access.
18. **6.2** For **INQUIRE TERMINAL**, **INQUIRE NETNAME**, and **SET TERMINAL**, command security checking is not performed if the task or program that issues the command was started or attached to the same terminal that is being inquired or modified by the command. This is because resource security checking was already performed on the terminal when the program or task was started or attached to the terminal. The following options are exceptions, where the command security checking is still performed:
  - Browsing options (START, NEXT, and END) on **INQUIRE TERMINAL** or **INQUIRE NETNAME**

- Tracing options (EXITTRACING, TRACING, and ZCPTRACING), the naming option (OPERID), and the purging option (PURGETYPE) on **SET TERMINAL**

## Responses to SDC HTTP calls

**6.2** Applies to 6.2.

SDC captures the security definitions required by applications during development. You can interact with security definition capture (SDC) through an HTTP REST interface. The response schema and example responses are documented.

For more information about the structure of HTTP responses, see [HTTP responses](#).

### Status codes and reason phrases

Responses to the SDC HTTP calls contain the following status codes, by request type.

The following table shows responses of the SDC HTTP calls:

<i>Table 72. Status codes of SDC calls' responses</i>			
Status code	Reason phrases	Applicable request type	Description
200	OK	GET, DELETE	The request completed.
201	CREATED	POST	SDC started.
202	ACCEPTED	DELETE	The job to stop SDC and generate required definitions was submitted.
400	BAD REQUEST	DELETE	The parameters or the syntax in the request were invalid.
401	UNAUTHORIZED	DELETE	The requester did not have required authority.
404	NOT FOUND	GET, DELETE	Couldn't stop SDC because the path was wrong or it was already off.
409	CONFLICT	POST	SDC was already on.
500	INTERNAL SERVER ERROR	POST, DELETE	An error occurred. Details can be found in the reason message.

### Response scheme

In each response, the following parameters are present:

#### **srr\_id**

Displays the ID generated by SDC. If SDC is not active, null is returned.

#### **message**

Displays a reason message that explains the result of the client HTTP request.

## Example responses to GET requests

GET requests query the status of SDC.

Table 73. Responses to GET request	
Example response	Description
<pre>HTTP/1.1 404 NOT FOUND {   "srr_id": null,   "message": "SDC IS NOT SWITCHED ON" }</pre>	SDC is off.
<pre>HTTP/1.1 200 OK {   "srr_id": "_1047462",   "message": "SDC IS ALREADY SWITCHED ON" }</pre>	SDC is on.

## Example responses to POST requests

POST requests start SDC.

Table 74. Responses to POST requests	
Example response	Description
<pre>HTTP/1.1 201 Created {   "srr_id": "_1053208",   "message": "SDC HAS BEEN SWITCHED ON" }</pre>	SDC started successfully.
<pre>HTTP/1.1 409 Conflict {   "srr_id": "_1053208",   "message": "SDC IS ALREADY SWITCHED ON" }</pre>	SDC was already on.
<pre>HTTP/1.1 500 Internal Server Error {   "srr_id": "null",   "message": "(This message can be a number of reasons as to why the request has failed)" }</pre>	An error occurred. Troubleshoot the error as instructed in the message section of the response.

## Example responses to DELETE requests

DELETE requests stop SDC.

Table 75. Responses to DELETE requests	
Example response	Description
<pre>HTTP/1.1 202 Accepted {   "srr_id": "_1053208",   "message": "SDC JOB SUBMITTED" }</pre>	SDC was stopped and a job was submitted to output the required security definitions.

Table 75. Responses to DELETE requests (continued)

Example response	Description
<pre>HTTP/1.1 200 OK {   "srr_id": "_1058733",   "message": "SDC TURNED OFF, NO RECORDS CAPTURED" }</pre>	SDC was stopped but no records were captured while SDC was on. Try extending the period when SDC is on.
<pre>HTTP/1.1 200 OK {   "srr_id": "_1058700",   "message": "SDC HAS BEEN TURNED OFF" }</pre>	SDC was stopped without submitting a job because "submit": false was specified in the DELETE request. See <a href="#">Capturing security definitions during development using security definition capture</a> .
<pre>HTTP/1.1 404 Not Found {   "srr_id": null,   "message": "SDC IS NOT SWITCHED ON" }</pre>	SDC was already off. If you want to capture security definitions, start it before testing. See <a href="#">Capturing security definitions during development using security definition capture</a> .
<pre>HTTP/1.1 500 Internal Server Error {   "srr_id": "null",   "message": "(This message can be a number of reasons as to why the request has failed)" }</pre>	An error occurred. Troubleshoot the error as instructed in the message section of the response.

## Data captured by CICS security discovery

### 6.2

Applies to 6.2.

Security discovery data (SDD) is the information that is collected about unique CICS resource accesses for every user while CICS security discovery is active.

Resource access requests are recorded when a [SET SECDICSOVERY](#) call is made that sets the associated resource class to DISCOVER.

### Data captured by security discovery

The information that is recorded for each unique resource access contains:

- User ID
- A set of indicators that show:
  - The user ID was the task's principal user ID
  - The user ID was the task's session user ID
  - The user ID was the region user ID
  - The user ID was the default user
- The complete SECPRFX, resource class, and profile name
- The level of access that was requested

### Data not captured by security discovery

Security discovery data does not contain any information that is related to the time or frequency of resource access.

The following resource accesses are not captured:

- Failed resource access requests
- Category 1 transactions and the resources within them
- Category 3 transactions and the resources within them
- Exempt internal programs
- Resources that were accessed from within the following Category 2 system management transactions:
  - CBAM
  - CCRL
  - CDBC, CDBF, CDBI, CDBQ, CDEA
  - CEDC, CEMN, CEMT, CEOT, CESD, CETR
  - CEDF and its variants CEDG, CEDX and CEDY
  - CIDP, CIND
  - CLDM, CLER
  - CMSG
  - COLM, CORM, COSH, COVC
  - CRTE
  - CSFE
  - CWGQ, CWTO, CWWU
  - DSNC
- TSQUEUE requests that did not match a TSMODEL or TST entry
- PassTicket generation controlled by XPTKT
- Non-CICS resources

#### **Related information**

How it works: [CICS security discovery](#)

[Capturing security discovery data \(SDD\)](#)

## **Security discovery statistics**

---

6.2

Applies to 6.2.

CICS security discovery statistics show the number of security calls that have, or would have, been made while resource security is active.

The following list summarizes fields in the security discovery statistics output produced by the DFH\$SDDP job.

#### **Security discovery records**

The number of unique resource security checks that were recorded by security discovery.

#### **Transactions executed**

The number of transactions that were executed.

#### **Successful resource checks**

The number of successful resource security checks that were made.

#### **Checks that bypassed DFHXSRC**

The number of resource security checks that bypassed DFHXSRC. DFHXSRC is bypassed if CMDSEC(NO) or RESSEC(NO) is specified. DFHXSRC is also bypassed for TS queues that use a model or TST entry but don't have security active.



**Checks that called DFHXSRC**

The number of successful resource security checks that called DFHXSRC.

**Exempt resources**

The number of successful resource security checks that were made against exempt resources. Category 1 and 3 transactions and certain internal programs are exempt from resource security checks. Surrogate checks where the surrogate user ID is the same as the current user ID are also exempt.

**Resource class not active**

The number of resource security checks that were successful because the associated security class was not active, for example, an attempt to access file when XFCT=N0.

**FASTAUTH calls**

The number of successful resource security checks that made a RACROUTE REQUEST=FASTAUTH call. The FASTAUTH call is made only if the associated resource class is active.

**AUTH calls**

The number of successful resource security checks that made a RACROUTE REQUEST=AUTH call. The AUTH call is made for **XHFS** checks and when logging has been requested for the associated resource class.

**XCMD checks**

The number of successful resource security checks that are controlled by the [XCMD](#) system initialization parameter.

**XDB2 checks**

The number of successful resource security checks that are controlled by the [XDB2](#) system initialization parameter.

**XDCT checks**

The number of successful resource security checks that are controlled by the [XDCT](#) system initialization parameter.

**XFCT checks**

The number of successful resource security checks that are controlled by the [XFCT](#) system initialization parameter.

**XHFS checks**

The number of successful resource security checks that are controlled by the [XHFS](#) system initialization parameter.

**XJCT checks**

The number of successful resource security checks that are controlled by the [XJCT](#) system initialization parameter.

**XPCT checks**

The number of successful resource security checks that are controlled by the [XPCT](#) system initialization parameter.

**XPPT checks**

The number of successful resource security checks that are controlled by the [XPPT](#) system initialization parameter.

**XPSB checks**

The number of successful resource security checks that are controlled by the [XPSB](#) system initialization parameter.

**XRES checks**

The number of successful resource security checks that are controlled by the [XRES](#) system initialization parameter.

**XTRAN checks**

The number of successful resource security checks that are controlled by the [XTRAN](#) system initialization parameter.

### XTST checks

The number of successful resource security checks that are controlled by the [XTST](#) system initialization parameter.

### XUSER checks

The number of successful resource security checks that are controlled by the [XUSER](#) system initialization parameter.

### Related information

[Estimating performance cost of additional resource security using security discovery statistics](#)

## Security-related system initialization parameters

These SIT parameters are used by CICS security processing.

Table 76. SIT parameters used in security processing	
SIT parameter	Specifies
<a href="#">SEC</a>	Overall level of resource security management for the region.
<a href="#">CMDSEC</a>	Whether you want CICS to use the CMDSEC option specified on a transaction's resource definition
<a href="#">CRLPROFILE</a>	The name of the profile that authorizes CICS to access certificate revocation lists that are stored in an LDAP server
<a href="#">DFLTUSER</a>	Identify to CICS the name you have defined to RACF as the default user ID.
<a href="#">ESMEXITS</a>	Whether you want CICS to pass installation data for use by the RACF installation exits.
<a href="#">GMTRAN</a>	The GMTRAN system initialization parameter specifies the ID of a transaction.
<a href="#">GNTRAN</a>	The GNTRAN system initialization parameter specifies the transaction that you want CICS® to invoke when a user's terminal-timeout period expires, and instructs CICS whether to keep a pseudo-conversation in use at a terminal that is the subject of a timeout sign-off.
<a href="#">KERBEROSUSER</a>	The KERBEROSUSER system initialization parameter specifies the user ID that is associated with the Kerberos service principal for the CICS region.
<a href="#">KEYRING</a>	The name of a keyring in the RACF database that contains keys and certificates used by CICS. (Not used by the Liberty JVM server SSL support.)
<a href="#">MAXSSLTCBS</a>	The maximum number of S8 TCBs that are available to CICS to process secure sockets layer connections and requests to LDAP using the DFHDDAPX XPI interface.
<a href="#">MAXTLSLEVEL</a>	Specifies the maximum TLS protocol that CICS uses for secure TCP/IP connections.
<a href="#">MINTLSLEVEL</a>	Specifies the minimum TLS protocol that CICS uses for secure TCP/IP connections.
<a href="#">NISTSP800131A</a>	The NISTSP800131A system initialization parameter specifies whether the CICS region is to check for conformance to the NIST SP800-131A standard.
<a href="#">PSBCHK</a>	Whether you want CICS to perform command security or resource security checking for PLT programs that run during CICS initialization.
<a href="#">PLTPISEC</a>	The userid that CICS is to use for security checking for PLT programs that run during CICS initialization

Table 76. SIT parameters used in security processing (continued)

<b>SIT parameter</b>	<b>Specifies</b>
<a href="#">PSBCHK</a>	Whether you want CICS to perform PSB authorization checks for remote terminal users who use transaction routing to initiate a transaction in this CICS region (to access an attached IMS system)
<a href="#">RACFSYNC</a>	The RACFSYNC system initialization parameter specifies whether CICS listens for type 71 ENF events.
<a href="#">RESSEC</a>	Whether you want CICS to use the RESSEC option specified on a transaction's resource definition
<a href="#">SECPREFX</a>	Whether you want CICS to prefix the resource names that it passes to RACF® for authorization. Effective only if you also specify SEC=YES.
<a href="#">SNSCOPE</a>	Whether a userid can be signed on to CICS more than once, within the scope of a single CICS region, a single z/OS image, and a sysplex.
<a href="#">SSLCACHE</a>	Specifies whether CICS should use a local cache of SSL sessions for the CICS region, or share the cache across multiple CICS regions by using the coupling facility.
<a href="#">SSLDELAY</a>	Specifies the length of time in seconds for which CICS retains session IDs for secure socket connections in a local CICS region.
<a href="#">USRDELAY</a>	The USRDELAY system initialization parameter specifies the maximum time that an eligible user ID and its associated attributes are cached in the CICS region after use.
<a href="#">XAPPC</a>	Whether you want CICS to perform resource security checking for APPC partner-LU verification (that is, RACF session security can be used when establishing APPC sessions). Effective only if you also specify SEC=YES.
<a href="#">XCMD</a>	Whether you want CICS to perform resource security checking for the subset of CICS application programming commands that are subject to command security checking. EXEC CICS FEPI system commands are also controlled by this parameter. For resource security for commands related to started transactions, see <a href="#">XPCT</a> . No authorization processing is done for BMS commands. Effective only if you also specify SEC=YES.
<a href="#">XDB2</a>	Whether you want CICS to perform resource security checking for CICS Db2 resources. Effective only if you also specify SEC=YES.
<a href="#">XDCT</a>	Whether you want CICS to perform resource security checking for transient data queues. Effective only if you also specify SEC=YES.
<a href="#">XFCT</a>	Whether you want CICS to perform resource security checking for files. Effective only if you also specify SEC=YES.
<a href="#">XHFS</a>	Whether CICS is to check the transaction user's ability to access files in the z/OS® UNIX System Services file system . Effective only if you also specify SEC=YES.
<a href="#">XJCT</a>	Whether you want CICS to perform resource security checking for journals and logs. Effective only if you also specify SEC=YES.
<a href="#">XPCT</a>	Whether you want CICS to perform resource security checking for started transactions and the following EXEC CICS commands: COLLECT STATISTICS TRANSACTION, DISCARD TRANSACTION, INQUIRE TRANSACTION, SET TRANSACTION. Effective only if you also specify SEC=YES.

Table 76. SIT parameters used in security processing (continued)

SIT parameter	Specifies
<a href="#">XPPT</a>	Whether you want CICS to perform resource security checking on application programs. Effective only if you also specify SEC=YES.
<a href="#">XPSB</a>	Whether you want CICS to perform resource security checking on DL/I program specification blocks (PSBs). Effective only if you also specify SEC=YES.
<a href="#">XPTKT</a>	Whether you want CICS to check if a user can generate a PassTicket for the user's userid using the EXEC CICS REQUEST PASSTICKET command or the EXEC FEPI REQUEST PASSTICKET command. Effective only if you also specify SEC=YES.
<a href="#">XRES</a>	Whether you want CICS to perform resource security checking for particular resources that are subject to XRES security checks. For a list of resources subject to XRES security checks, see <a href="#">Resource security</a> . Effective only if you also specify SEC=YES.
<a href="#">XTRAN</a>	Whether you want CICS to perform resource security checking for attached ransactions. Effective only if you also specify SEC=YES.
<a href="#">XTST</a>	Whether you want CICS to perform resource security checking for temporary storage queues. Effective only if you also specify SEC=YES.
<a href="#">XUSER</a>	Whether you want CICS to perform surrogate user checks and AUTHTYPE checking for Db2. Effective only if you also specify SEC=YES.

## Security standards

CICS complies with the following standards for security processing.

Table 77. Security standards

Standard	CICS support
<a href="#">SAML</a>	SAMLCORE1.1 and SAMLCORE2.0 standards. CICS does not support the protocols that are described in those standards. It does not support SAMLCORE 1.0.
<a href="#">Web Services Security: SOAP Message Security</a>	Conditionally complies with: <ul style="list-style-type: none"> <li>• SOAP Message Security</li> <li>• UsernameToken Profile 1.0</li> <li>• X.509 Certificate Token Profile 1.0</li> </ul>
<a href="#">Web Services Trust Language</a>	CICS supports only WS-Trust 1.2, which is the February 2005 version of the specification..
<a href="#">National Institute of Standards and Technology (NIST) SP800-131A</a>	You can make your system SP800-131A conformant by configuring CICS to use suitable cipher suites and certificates.
<a href="#">FIPS 140-2</a>	CICS supports the FIPS140-2 standard for cryptographic modules.
<a href="#">TLS support</a>	CICS supports the Transport Layer Security (TLS) protocol that supersedes the Secure Sockets Layer (SSL) protocol that provides support in CICS for TLS 1.1 to TLS 1.3.

Table 77. Security standards (continued)	
Standard	CICS support
Compliance data collection (PCI-DSS)	CICS supports PCI-DSS (Payment Card Industry Data Security Standard).

## SMF 1154 subtype 80 record

The CICS SMF 1154 subtype 80 compliance evidence record is documented in [XSCC – SMF 1154 Subtype 80](#).

SMF 1154 subtype 80 DSECTs are available in a copybook for each programming language, as listed in [Table 78 on page 681](#).

Table 78. Copybooks by language		
Language	Copybook name	Data set name
COBOL	DFHXSCCO	SDFHCOB
PL/I	DFHXSCCL	SDFHPL1
C	DFHXSCCH	SDFHC370
Assembler	DFHXSCCD	SDFHMAC

An SMF 1154 record has a common header. This information is described in [SMF records](#).

The CICS SMF 1154 subtype 80 records contain the following sections:

### Header

The header identifies the number of sections, their location, length, and the number.

### Section 1 - CICS Region Definition

This part of the record contains the [CICS SIT options](#) relevant for security for the regions.

### Section 2 - CICS Region Usage Tags

This part of the record contains the region tags for this region. See [CICS region tagging file](#).

The cicsts-RegionUsage tag can be useful in identifying production regions.

### Section 3 CICS Service Level

This part of the record lists the most recent PTFS or APARs that are in the instance of the running code. This information can include the base function name of there are too few Ftps.

### Section 4 CICS Resources

This optional part of the record is a list of active resources that are not conformant.

CICS provides a JCL sample DFH\$54P in SDFHSAMP to print SMF 1154 subtype 80 records from an SMF logstream. It writes a summary of the data that is collected to the job output and to a CSV file on zFS. It also outputs a field by field description of the data that is collected and a dump of the information for diagnostic purposes.

This sample uses a Rexx program DFH\$1154 in SDFHSAMP.

### Related links:

[SMF 1154 subtype 80 record](#)

[Example of formatted SMF 1154 s](#)

## SRR reason codes

The security request recording (SRR) reason codes are returned by SRR to indicate why the security check succeeded or failed.

The SRR reason codes are in the reason column of the SRR spreadsheet. For more information about the SRR spreadsheet, see [Example output of security request recording \(SRR\)](#).

For information about using SRR, see [Diagnosing access issues with security request recording \(SRR\)](#).

The following reason codes are returned by SRR to indicate the authorization status with reasons. When the authorization is successful because of internal CICS processing, the request is not passed to RACF.

Table 79. SRR reason codes		
Reason code	Authorization status	Description
CMDSEC(NO)	Successful	The request was not processed because the transaction was defined to ignore command security.
RESSEC(NO)	Successful	The request was not processed because the transaction was defined to ignore resource security.
<div>6.2</div> EDF_SET_CMDSEC(NO) <div>6.2</div> CICS_SET_CMDSEC(NO) <div>6.2</div> EDF_SET_RESSEC(NO) <div>6.2</div> CICS_SET_RESSEC(NO)	Successful	CICS switched off security internally to execute this command.
XCMD=NO XDB2=NO XDCT=NO XFCT=NO XHFS=NO XJCT=NO XPCT=NO XPPT=NO XPSB=NO XRES=NO XTRAN=NO XTST=NO XUSER=NO	Successful	Security for this class was disabled by the option that is specified on the corresponding SIT parameter.
XPPT=...DPLONLY	Successful	Security was switched off because DPLONLY was specified on the <b>XPPT</b> SIT parameter and this was not the first program that is linked by the mirror program during distributed program link (DPL).
EXEMPT=CAT1	Successful	Category 1 programs can only be run by the region user ID. Therefore, no external security checking is required.

Table 79. SRR reason codes (continued)		
Reason code	Authorization status	Description
EXEMPT=CAT3	Successful	Category 3 programs can be run by all users. Therefore, no external security checking is required.
EXEMPT=PROGRAM	Successful	This is a CICS program that can be accessed by all users. Therefore, no external security checking is required.
NOTAUTH	Failed	Authorization not successful. The code will be passed to RACF.

Other codes might be returned if the CICS XSRG domain gate returns a failure. These are the same as the values in the CICS trace.

## The SAML linkable interface, DFHSAML

This topic describes the input and output containers, copybooks, DSECTs, and response codes that are used by CICS SAML support.

Container names and DSECTs are available in a copybook for each programming language, as listed in the following table.

Table 80. Copybooks by language		
Language	Copybook name	Data set name
COBOL	DFHSAMLO	SDFHCOB
PL/I	DFHSAMLPL	SDFHPL1
C	DFHSAMLH	SDFHC370
Assembler	DFHSAMLD	SDFHMAC

### Input containers

Table 81 on page 683 lists the input containers. In this table, *nnn* means that there might be more than one container. Containers are distinguished by the string *nnn*. This string contains an alphanumeric value, unlike the corresponding input containers, where *nnn* is always numeric.

Table 81. Input containers		
Name	Type	Description
DFHSAML-TOKEN	CHAR	
DFHSAML-FUNCTION	CHAR	
DFHSAML-JVMSEVR	CHAR	
DFHSAML-FILTER	CHAR	
DFHSAML-SIGNED	CHAR	

Table 81. Input containers (continued)

Name	Type	Description
DFHSAML-OUTTOKEN	CHAR	Read-only container. Contains the previously obtained from tokenvalidated token, which is validationbeing routed to another service provider, or extended and then routed.
DFHSAML-ATTRNnnn	CHAR	Attribute Name for attribute <i>nnn</i> , where <i>nnn</i> is a string of three alphanumeric characters. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.
DFHSAML-ATTRSnnn	CHAR	Attribute Name Space for attribute <i>nnn</i> , where <i>nnn</i> is a string of three alphanumeric characters. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.
DFHSAML-ATTRYnnn	CHAR	Attribute Friendly name for attribute <i>nnn</i> , where <i>nnn</i> is a string of three alphanumeric characters. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.
DFHSAML-ATTRFnnn	CHAR	Attribute name format for attribute <i>nnn</i> , where <i>nnn</i> is a string of three alphanumeric characters. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.
DFHSAML-ATTRAnnn	BIT	BIN(31) field, containing the number of values for attribute <i>nnn</i> . The maximum number of values is 999. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.
DFHSAML-AnnnVmmm	CHAR	Attribute Value <i>mmm</i> for attribute <i>nnn</i> , where <i>nnn</i> and <i>mmm</i> are strings of three alphanumeric characters. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container. The number of values for this attribute is in DFHSAML-ATTRAnnn.

## SAML-FILTER DSECT

The filter DSECT has a series of fields for each of the containers.

The name of the field is based on the name of the container, for example: SAMLI-CONFMETH to include container DFHSAML-CONFMETH.

Set the field to SAML - YES for each container that you want to be returned and SAML - NO for the containers that you do not want.

DFHSAML response container is always returned.



Table 82. SAML-FILTER DSECT containers		
Variable	Type	Description
SAMLI-COUNTS	CHAR (1)	DFHSAML-COUNTS container
SAMLI-TIMES	CHAR (1)	DFHSAML-TIMES container
SAMLI-FLAGS	CHAR (1)	DFHSAML-FLAGS container
SAMLI-ASSQNAME	CHAR (1)	DFHSAML-ASSQNAME container
SAMLI-AUDNR	CHAR (1)	DFHSAML-AUDNR* containers
SAMLI-AUTHMETH	CHAR (1)	DFHSAML-AUTHMETH container
SAMLI-CONFMETH	CHAR (1)	DFHSAML-CONFMETH container
SAMLI-CERT	CHAR (1)	DFHSAML-CERT* containers
SAMLI-PROXY	CHAR (1)	DFHSAML-PROXY* container
SAMLI-ATTR	CHAR (1)	All attributes and values
SAMLI-ID	CHAR (1)	DFHSAML-ID container
SAMLI-ISSUER	CHAR (1)	DFHSAML-ISSUER container
SAMLI-NAMEID	CHAR (1)	DFHSAML-NAMEID* containers
SAMLI-CERTIDN	CHAR (1)	DFHSAML-CERTIDN container
SAMLI-CERTSDN	CHAR (1)	DFHSAML-CERTSDN container
SAMLI-SUBJLOC	CHAR (1)	DFHSAML-SUBJ* containers

## Output containers

The following containers are output and are all read only.

The WebSphere Application Server methods are described in [Package com.ibm.websphere.wssecurity.wssapi.token](#).

This list is a subset of the full SAML assertion schema.

In the following table, *nnn* means that there might be more than one container. Containers are numbered 001 to *nnn* (the number of containers of this type returned). More than 999 containers of a particular type are not supported and the data in the SAML assertion that relates to them is ignored. Containers that are not mapped to a DSECT are variable length.

Table 83. Output containers		
Name	Type	Description
DFHSAML-RESPONSE	BIT	<a href="#">Response Code</a>
DFHSAML-OUTTOKEN	CHAR	SAML token output by DFHSAML processing. Depending on the processing performed, this can be a validated, extracted, or modified and resigned token.
DFHSAML-COUNTS	BIT	Number of variable length containers <a href="#">DFHSAML-COUNTS DSECT</a>
DFHSAML-TIMES	CHAR	Time values <a href="#">DFHSAML-TIMES DSECT</a>
DFHSAML-FLAGS	CHAR	Flags bytes <a href="#">DFHSAML-FLAGS DSECT</a>

Table 83. Output containers (continued)

Name	Type	Description
DFHSAML-ASSQNAME	CHAR	<p>SAML Assertion namespace:</p> <p><b>SAML 1.1</b> urn:oasis:names:tc:SAML:1.0:assertion</p> <p><b>SAML 2.0</b> urn:oasis:names:tc:SAML:2.0:assertion</p> <p>This assertion must be a URI. If the assertion is more complex, it extracts into the 3 parts.</p>
DFHSAML-AUDNRnnn	CHAR	<p>AudienceRestriction name.</p> <p>Number of containers returns is AUDNRNUM.</p>
DFHSAML-AUTHMETH	CHAR	<p>The method that is used to authenticate the token holder. For example, password, Kerberos, ltpa.</p>
DFHSAML-CONFMETH	CHAR	<p>SubjectConfirmation Method that is used in this SAML token. Valid methods are holder-of-key, bearer, or sender-vouches. The returned string is based on the OASIS SAML token profile 1.1 and 2.0.</p> <p><b>Note:</b> SAML tokens that have more than one confirmation method are not supported. If there is more than one confirmation method, the results are unpredictable.</p>
DFHSAML-PROXYnnn	CHAR	<p>ProxyRestriction Audience name</p>
DFHSAML-ATTRSnnn	CHAR	<p>Attribute Name Space for attribute <i>nnn</i>, where <i>nnn</i> is a 3-digit number.</p> <p>Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.</p>
DFHSAML-ATTRYnnn	CHAR	<p>Attribute Friendly name for attribute <i>nnn</i>, where <i>nnn</i> is a 3-digit number.</p> <p>Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.</p>
DFHSAML-ATTRNnnn	CHAR	<p>Attribute Name for attribute <i>nnn</i>, where <i>nnn</i> is a 3-digit number.</p> <p>Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.</p>
DFHSAML-ATTRFnnn	CHAR	<p>Attribute name format for attribute <i>nnn</i>, where <i>nnn</i> is a 3-digit number.</p> <p>Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.</p>
DFHSAML-ATTRAnnn	BIT	<p>BIN(31) field with number of values for attribute <i>nnn</i> (the maximum number of values is 999).</p> <p>Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container.</p>

Table 83. Output containers (continued)		
Name	Type	Description
DFHSAML-AnnnVmmm	CHAR	Attribute Value <i>mmm</i> for attribute <i>nnn</i> , where <i>nnn</i> and <i>mmm</i> are 3-digit numbers. Number of attributes is SAMLC-ATTRNUM in DFHSAML-COUNTS container. Number of values for this attribute is in DFHSAML-ATTRAnnn.
DFHSAML-SAMLID	CHAR	A string that represents the ID for SAML 2.0, or AssertionID for SAML 1.1.
DFHSAML-ISSUER	CHAR	Name of issuer
DFHSAML-NAMIDF	CHAR	URI reference that represents the classification of string-based identifier information
DFHSAML-NAMIDQ	CHAR	Security or administrative domain that qualifies the name.
DFHSAML-NAMIDSPQ	CHAR	Name of a service provider or affiliation of providers.
DFHSAML-NAMIDSP	CHAR	Name identifier that is established by a service provider or affiliation of providers of the entity.
DFHSAML-NAMID	CHAR	Value of the name format property
DFHSAML-CERTSNUM	CHAR	An eight-character field that contains the SAML signer's X.509 Certificate serial number
DFHSAML-CERTIDN	CHAR	Issuer's distinguished name of SAML signer's X.509 Certificate
DFHSAML-CERTSDN	CHAR	Subject's distinguished name of SAML signer's X.509 Certificate
DFHSAML-SUBJDNS	CHAR	DNSAddress in SubjectLocality
DFHSAML-SUBJADDR	CHAR	IP address in SubjectLocality. <b>Restriction:</b> This container is not returned for SAML 2.0.

Table 84. Response codes	
Value	Description
0	OK
1	Invalid token
2	Container error
3	Missing required input container
6	JVM server is not enabled
7	JVM server is not found
9	DFHSAML-FUNCTION container is not CHAR
10	DFHSAML-TOKEN container is not found
11	DFHSAML-TOKEN container is not CHAR
12	DFHSAML-JVM container is not CHAR

*Table 84. Response codes (continued)*

<b>Value</b>	<b>Description</b>
13	DFHSAML-FILTER container is not CHAR
14	DFHSAML-SIGNED container is not CHAR
15	An error occurred in parsing the token. The DFHSJ-ERROR container contains further details. See <a href="#">DFHSJ-ERROR container</a> .
16	The DFHSAML-FILTER container has invalid data
17	The DFHSAML-FUNCTION container has invalid data
18	The DFHSAML-SIGNED container has invalid data
19	The DFHSAML-OUTTOKEN container is not found
21	The certificate in the token has expired
22	The token is no longer valid
23	The certificate in the token is not trusted
24	A container is not read-only
25	A signature is not configured in the STS configuration file
26	Add attribute error
27	Attribute input containers are not CHAR
28	An attribute value is missing or empty
29	An attribute name is missing or empty
30	No key ring is specified
31	The certificate was not found in the keyring
33	The JVM server is not configured for SAML

*Table 85. SAML-COUNTS DSECT*

<b>Name</b>	<b>Type</b>	<b>Description</b>
SAMLC-AUDRNUM	Signed fullword binary value	Number of audience restriction values
SAMLC-ATTRNUM	Signed fullword binary value	Number of attributes
SAMLC-PROXYNUM	Signed fullword binary value	Number of ProxyRestriction counts

*Table 86. SAML-TIMES DSECT*

<b>Name</b>	<b>Type</b>	<b>Description</b>
SAMLT-NOTBEFORE	CHAR(20)	Not before time
SAMLT-EXPIRES	CHAR(20)	Not on or after time
SAMLT-AUTHTIME	CHAR(20)	Time SAML token authorized
SAMLT-CERTNAFT	CHAR(20)	Certificate not valid after
SAMLT-CERTNBEF	CHAR(20)	Certificate not valid before

All times are in Coordinated Universal Time (UTC) format. If they are not available, the field is blank.

<i>Table 87. SAML-FLAGS DSECT</i>		
<b>Name</b>	<b>Type</b>	<b>Description</b>
SAMLB-PROXYRST	CHAR(1)	'Y'/'N' flag to indicate ProxyRestriction
SAMLB-ONETIME	CHAR(1)	'Y'/'N' flag to indicate OneTimeUse



## Notices

---

This information was developed for products and services offered in the United States of America. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119 Armonk,  
NY 10504-1785  
United States of America*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Client Relationship Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

#### **COPYRIGHT LICENSE:**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## **Programming interface information**

IBM CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 6 are included in the following sections of the online product documentation:

- [Developing applications](#)
- [Developing system programs](#)
- [Securing CICS](#)
- [Developing for external interfaces](#)
- [Application development reference](#)
- [Reference: system programming](#)
- [Reference: connectivity](#)

Information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 6, but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- [Troubleshooting and support](#)
- [CICS TS diagnostics reference](#)

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 6 are included in the following manuals:

- Application Programming Guide and Application Programming Reference



- Business Transaction Services
- Customization Guide
- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- CICS Transactions
- CICSplex System Manager (CICSplex SM) Managing Workloads
- CICSplex SM Managing Resource Usage
- CICSplex SM Application Programming Guide and Application Programming Reference
- Java Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 6, but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSplex SM Problem Determination Guide

## Trademarks

IBM, the IBM logo, and [ibm.com](http://ibm.com)® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and trademark information at www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

The Stack Exchange name and logos are trademarks of Stack Exchange Inc.

Red Hat, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms, RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

## **Terms and conditions for product documentation**

Permissions for the use of these publications are granted subject to the following terms and conditions.

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

### **Rights**

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## **IBM online privacy statement**

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information (PII) is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect PII. If this Software Offering uses cookies to collect PII, specific information about this offering's use of cookies is set forth below:

### **For the CICSplex SM Web User Interface (main interface):**

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other PII for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

**For the CICSplex SM Web User Interface (data interface):**

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other PII for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

**For the CICSplex SM Web User Interface ("hello world" page):**

Depending upon the configurations deployed, this Software Offering may use session cookies that do not collect PII. These cookies cannot be disabled.

**For CICS Explorer:**

Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management, authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect PII from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see [IBM Privacy Policy](#) and [IBM Online Privacy Statement](#), the section entitled *Cookies, Web Beacons and Other Technologies* and the [IBM Software Products and Software-as-a-Service Privacy Statement](#).



---

# Index

## Special Characters

- ESM interface
  - RACROUTE macros [448](#)

## A

- access allowed incorrectly
  - resolving [606](#)
- access control
  - HFS files [570](#)
  - z/OS UNIX files [569](#)
- accounting [506](#)
- ACICSPCT general resource class [562](#), [565](#)
- activating security parameters [441](#)
- activities
  - security of [483](#)
- ADDMEM operand [497](#)
- ADDUSER command
  - defining the userid for CICS to RACF [534](#)
- administration
  - security [483](#)
- ALLOCATE\_PIPE command
  - security check failure [515](#)
- APPCLU general resource class [638](#)
- APPL general resource class
  - controlling access to CICS region [545](#)
- application
  - security [416](#), [436](#), [479](#)
- AT-TLS
  - Application Transparent Transport Layer Security [467](#), [471](#), [472](#)
  - PROTOCOL(HTTP) [467](#), [471](#), [472](#)
  - SSL(ATTLSAWARE) [467](#), [471](#), [472](#)
  - SSL(NO) [467](#), [471](#), [472](#)
- AT-TLS Basic [467](#)
- AT-TLS Controlling [467](#)
- Atom collection
  - security [477](#)
- Atom feed
  - security [477](#)
- ATTLS [471](#), [472](#)
- ATTLS Aware [471](#), [472](#)
- ATTLS Basic [471](#), [472](#)
- ATTLS Controlling [471](#), [472](#)
- ATTLSAWARE [467](#)
- auditing
  - second request to RACF to write log data [619](#)
  - SMF type 80 log records [619](#)
- AUTHENTICATE(AUTOREGISTER) [472](#)
- AUTHENTICATE(CERTIFICATE) [472](#)
- authentication [454](#), [455](#), [459](#)
- authentication, RPC [519](#)
- authorization failures
  - access is denied incorrectly [601](#)
  - command security [534](#)

- authorization failures (*continued*)
  - error messages [619](#)
  - ICH408I, RACF message [619](#)
- authorization IDs, primary [505](#)
- authorization IDs, secondary [507](#)
- AUTHTYPE security [498](#)
- AUTHTYPE values for security
  - authorization ID [501](#)
  - group ID [501](#)
  - sign ID from DB2CONN [501](#)
  - terminal ID [501](#)
  - transaction ID [501](#)
  - user ID [501](#)

## B

- backup while open (BWO) [541](#)
- basic authentication [454](#), [455](#), [459](#)
- batch access to CSD, restricting [557](#)
- BCICSPCT general resource class [562](#), [565](#)
- bind-time security [515](#)
- building a keyring [167](#)
- BWO (backup while open) [541](#)

## C

- cataloged procedures
  - authorizing CICS as a started task [533](#)
- CCRL
  - running from a START command [184](#)
  - running from a terminal [184](#)
- CEDA LOCK command [557](#)
- CEDA transaction [557](#)
- CEDF [573](#)
- CEDF transaction [573](#)
- CEDX [573](#)
- CEDX transaction [573](#)
- certificate revocation list [182](#), [183](#)
- certificates, creating new [170](#)
- CFRM policy [528](#)
- CICS as an HTTP client
  - security [455](#)
  - SSL [467](#)
- CICS as an HTTP server
  - security [454](#), [459](#), [462](#)
- CICS business transaction services
  - administration
    - security [483](#)
- CICS command security
  - VCICSCMD general resource class [497](#)
- CICS commands and resources
  - creating security profiles with RACF
    - controlling access to resources [416](#)
- CICS JOB statement, PASSWORD parameter [534](#)
- CICS JOB statement, USER parameter [534](#)
- CICS Liberty [257](#), [260](#), [266](#), [269](#), [276](#)
- CICS region

- CICS region (*continued*)
  - access to [545](#)
  - access to APPL class profiles [546](#)
  - remote [546](#)
  - userid as security token [548](#)
- CICS region userid [532](#)
- CICS resources
  - protecting [636](#)
- CICS security [494](#)
- CICS security, controlling [439](#)
- CICS segment [93](#)
- CICS SIT parameters
  - security-related [441](#)
- CICS system definition file (CSD), restricting batch access to [557](#)
- CICS system initialization parameters
  - SEC [519](#)
  - XCMD [519](#)
  - XPPT [519](#)
  - XUSER [519](#)
- CICS web support
  - security [453](#)
- CICSplex SM
  - authorizing
    - procedures [410](#)
  - protecting
    - with another ESM [447](#)
    - with RACF [402](#)
  - resource names [416](#)
- CICSplex SM definitions, protecting
  - adding SAF resource classes [416](#)
- CICSplex SM resource classes
  - controlling access to [416](#)
- cipher suites
  - identifying which are used [179](#)
  - restricting with CIPHERS attribute [179](#)
- CIPHERS attribute
  - CORBASERVER resource [179](#)
  - TCIPSERVICE resource [179](#)
  - URIMAP resource [179](#)
- classification of data and users [103](#)
- CLAUTH (class authority) attribute
  - installation-defined classes [99](#)
- client certificate [454](#), [455](#)
- client program
  - MRO logon and bind-time security [515](#)
- ClientAuthType [467](#), [471](#)
- clock skew
  - changing [45](#)
  - setting [45](#)
- clock\_skew [45](#)
- CLOUD.APPLICATION security profile [436](#), [479](#)
- CLOUD.DEF security profile [436](#), [479](#)
- CLOUD.PLATFORM security profile [436](#), [479](#)
- CMDSEC [519](#)
- command authorization
  - Db2 [508](#)
- command security
  - authorization failures [534](#)
  - CICS resources subject to [645](#)
  - VCICSCMD general resource class [654](#)
  - XUSER parameter [557](#)
- conditional access processing [555](#)
- configuration files (*continued*)
  - Security Token Service [46](#)
- configuring [41](#)
- connection authorization [502](#)
- connection exit routine [503](#)
- connection security [487](#)
- CONSOLE general resource class
  - description [556](#)
- containers
  - SAML [683](#)
- coupling facility data tables security [528](#)
- COVA [412](#)
- COVC [412](#)
- COVE [412](#)
- COVG [412](#)
- COVP [412](#)
- COVU [412](#)
- CPLT transaction [60](#)
- CRL (certificate revocation list) [182](#)
- CSD (CICS system definition file), restricting batch access to [557](#)
- CSD definitions, locking [558](#)
- CSMI (CICS-supplied mirror transaction)
  - authorizing the link user ID [516](#)
  - security [516](#)
- custom security handler [380](#)

## D

- data set
  - encryption [512](#)
- data set security
  - access to user data sets [541](#)
  - CICS system [532](#)
  - MVS library lookaside (LLA) facility [541](#)
- data tables
  - CONNECT security checks [526](#)
  - coupling facility [528](#)
  - server authorization security check [525](#)
- Db2 security
  - accounting [506](#)
  - authorization to execute a plan [509](#)
  - establishing authorization IDs [505](#), [507](#)
  - primary authorization ID [502](#)
  - secondary authorization ID [502](#)
  - security mechanisms [495](#)
- DCICSDCT general resource class
  - defining profiles [560](#)
- default user, CICS
  - defining [536](#)
- defined activity userid [483](#)
- defined process userid [483](#)
- definitions, protecting
  - controlling access to resources [416](#)
- DELMEM operand [554](#)
- deploying [40](#)
- DES (data encryption standard) [164](#)
- DES authentication [519](#)
- developing
  - applications
    - Kerberos [37](#)
- DFHHTML [413](#)
- DFHIRP (interregion communication program)
  - security checks performed by [515](#)

- DFHWPBW, password expiry management program [454](#), [459](#)
- DFLTUSER parameter
  - obtaining user data [95](#)
- DFLTUSER SIT parameter
  - creating a security environment [406](#)
- diagnostic messages [471](#)
- DIGTCERT general resource class [638](#)
- document template security
  - XRES parameter [466](#)
- document templates
  - security [462](#), [463](#)
- domain [471](#)
- DSN3SATH sample [503](#)
- DSN3SSGN sample [507](#)

## E

- ECICSDCT general resource class
  - defining profiles [560](#)
- encryption
  - data sets [512](#)
- ENCRYPTION parameter
  - and SP800-131A [180](#)
- ESDs, VSAM, access to [541](#)
- ESM (external security manager)
  - invoking another
    - RACROUTE macros [448](#)
  - PassTickets [550](#)
  - using RACF
    - controlling CICS security [439](#)
    - creating profiles [416](#)
- evaluation sequence, security [443](#)
- example tasks
  - security [448](#)
- examples
  - RACF commands [483](#)
  - surrogate security checking [483](#)
- EXEC CICS LINK command
  - security checking [516](#)
- EXEC CICS QUERY SECURITY [519](#)
- EXEC CICS START USERID [519](#)
- EXEC CICS VERIFY PASSWORD [519](#)
- external CICS interface (EXCI)
  - security [515](#)
- external security manager (ESM)
  - invoking another
    - RACROUTE macros [448](#)
  - using RACF
    - controlling CICS security [439](#)
    - creating profiles [416](#)
- EXTRACT CERTIFICATE command [454](#)
- EYUCOVE [413](#)
- EYUCOVI [413](#)
- EYULOG [413](#)
- EYUWREP [413](#)
- EYUWUI [413](#)

## F

- FIELD general resource class [638](#)
- file security
  - data set profiles [101](#)
- FIPS compliance [180](#)

- FMH (function management header)
  - possible errors in [620](#)

## G

- general resource classes
  - ACICSPCT [562](#), [565](#)
  - APPLU [638](#)
  - APPL [638](#)
  - BCICSPCT [562](#), [565](#)
  - CONSOLE [638](#)
  - DIGTCERT [638](#)
  - FIELD [638](#)
  - for protecting CICS resources [636](#)
  - JCICSJCT [561](#)
  - JESSPOOL [548](#), [638](#)
  - KCICSJCT [561](#)
  - LOGSTRM [638](#)
  - OPERCMDS [558](#), [639](#)
  - PCICSPSB [572](#)
  - PROGRAM [639](#)
  - PROPCNTL [547](#), [639](#)
  - PTKTDATA [639](#)
  - QCICSPSB [572](#)
  - STARTED [639](#)
  - SUBSYSNM [639](#)
  - SURROGAT [547](#), [639](#)
  - TERMINAL [639](#)
  - VCICSCMD [654](#)
  - VTAMAPPL [546](#), [640](#)
- general resource profiles
  - refreshing [442](#)
- generating and using RACF PassTickets [21](#)
- generic connection
  - note about lack of security checks [515](#)
- generic profiles
  - SETROPTS GENERIC [101](#), [554](#)
- generic resource names (z/OS Communications Server)
  - z/OS Communications Server generic resource [545](#)
- generic resource profiles [98](#)
- global security parameters [442](#)
- GRANT command [508](#)
- group profiles [92](#)
- group-SPECIAL attribute [92](#)
- GTERMINAL definition [553](#)
- GTERMINL definition [553](#)
- GTERMINL resource group class [639](#)

## H

- handshakerole=serverwithclientauth [472](#)
- HFS file security
  - XHFS parameter [570](#)
- HTTP client requests
  - security [455](#)

## I

- ICH408I, RACF message [619](#)
- ICHRIN03 started procedures table [410](#)
- ICHRIN03, RACF started task table [533](#)
- in-storage profiles

in-storage profiles (*continued*)  
GTERMINL profiles [554](#)  
invoking the trust client [380](#)  
IRR.DIGTCERT.ADD profile [171](#)

## J

JCICSJCT general resource class [561](#)  
JES spool protection [548](#)  
JESSPOOL general resource class [548](#), [638](#)  
job submission, surrogate [547](#)  
journal security  
    access authorization levels [561](#)  
    defining resource classes [561](#)  
    XJCT parameter [561](#)  
journals and log streams  
    journal access authorization levels [561](#)

## K

KCICSJCT general resource class [561](#)  
Kerberos  
    configuring web services [36](#)  
    developing applications [37](#)  
key rings [167](#)  
keyring  
    building [167](#)  
KEYRING [467](#), [471](#)

## L

labels, RACF security [103](#)  
language segment  
    system defaults [94](#)  
    user profile [94](#)  
LDAP server  
    configuring [183](#)  
levels, RACF security [103](#)  
libraries, CICSplex SM  
    protecting with RACF [402](#)  
LOCK command, CEDA [557](#)  
log records  
    SMF type 80 [619](#)  
log streams  
    authorizing access to [538](#)  
    LOGSTRM general resource class [538](#)  
logon security [515](#)  
LOGSTRM general resource class [638](#)

## M

maps  
    security considerations [414](#)  
marking a certificate untrusted [172](#)  
members, group  
    ADDMEM operand to add [554](#)  
    DELMEM operand to remove [554](#)  
message handler  
    invoking trust client [380](#)  
messages  
    authorization failures [619](#)  
    destination of ICH408I message [619](#)  
    ICH408I, RACF [619](#)

messages (*continued*)  
    RLIST command [602](#)  
MINTLSLEVEL/ENCRYPTION [467](#), [471](#)  
MQCONN  
    security [486](#)  
MQMONITOR  
    security [486](#), [487](#)  
multiple volumes [541](#)  
multiregion operation (MRO)  
    logon and bind time security [515](#)  
MVS  
    library lookaside (LLA) facility [541](#)

## N

National Language Support [94](#)  
NIST conformance [180](#)  
Null authentication [519](#)

## O

OPCLASS [94](#)  
operator, CICS terminal  
    obtaining data for [95](#)  
operator, terminal  
    data for default user [95](#)  
OPERCMDS general resource class [639](#)  
OPIDENT [94](#)  
OPPRY [94](#)

## P

parameter  
    authorizing access to CICS region [545](#)  
parameters  
    security  
        activating [441](#)  
        checking [441](#)  
        global [442](#)  
PassTickets, for signon security [21](#), [550](#)  
password expiry management program DFHWBPW [454](#), [459](#)  
patterns  
    SAML programs [597](#)  
PCICSPB general resource class [572](#)  
PEM problem determination [620](#)  
PERMIT command  
    WHEN operand [555](#)  
persistent sessions  
    XRFSOFF operand [94](#)  
pipeline  
    provider  
        configuring for SAML tokens [42](#)  
    requester  
        configuring for SAML [44](#)  
PKCS (public key cryptography standard) [164](#)  
platform  
    security [416](#), [436](#), [479](#)  
PLT  
    post-initialization processing [60](#)  
PLTPIUSR system initialization parameter [60](#)  
POSIT numbers  
    installation-defined general resource classes [99](#)  
post-initialization processing, surrogate security [60](#)



- PREFIX attribute definition [568](#)
- preset terminal security
  - CEDA LOCK command [557](#)
  - CEDA transaction [557](#)
  - controlling definition and installation [557](#)
  - restricting batch access to CSD [557](#)
  - SURROGAT transaction [557](#)
  - using MVS system console as CICS terminal [558](#)
- problem determination
  - access is allowed incorrectly [606](#)
  - access is denied incorrectly [601](#)
  - error messages for authorization failures [619](#)
  - errors, common causes [620](#)
  - FMH in error [620](#)
  - ICH408I, RACF message [619](#)
  - RLIST command [602](#)
  - security-related CICS initialization failures [615](#)
  - user data in error [620](#)
- process
  - security of [483](#)
- PRODCFT1 [528](#)
- profiles
  - ACICSPCT general resource class [562](#), [565](#)
  - BCICSPCT general resource class [562](#), [565](#)
  - data set [101](#)
  - DCICSDCT general resource class [560](#)
  - ECICSDCT general resource class [560](#)
  - JCICSJCT general resource class [561](#)
  - JESSPOOL [548](#)
  - KCICSJCT general resource class [561](#)
  - not found [602](#)
  - PCICSPSB general resource class [572](#)
  - PROPCNTL [547](#)
  - QCICSPSB general resource class [572](#)
  - refreshing in main storage [98](#)
  - resource and WARNING option [52](#)
  - resources, defining generic [98](#)
  - SETROPTS command [553](#)
  - SURROGAT general resource class [547](#)
  - terminal (PoE), defining [553](#)
  - USER parameter on CICS JOB statement [534](#)
  - VCICSCMD general resource class [654](#)
  - VTAMAPPL [546](#)
- PROGRAM general resource class [639](#)
- propagation of userid, controlling [547](#)
- PROPCNTL general resource class
  - defining profiles [547](#)
- protecting
  - CICS resources [636](#)
- provider pipeline
  - configuring
    - for SAML tokens [42](#)
- proxy authentication [455](#)
- PSB security
  - access authorization levels [573](#)
  - defining resource classes [572](#)
  - PCICSPSB general resource class [572](#)
  - QCICSPSB general resource class [572](#)
  - XPSB parameter [573](#)
- PSBCHK parameter [573](#)
- PTKTDATA general resource class [639](#)
- public key encryption [164](#)

## Q

- QCICSPSB general resource class [572](#)

## R

- RACDCERT command [167](#), [171](#)
- RACF
  - example commands [483](#)
  - external security manager [493](#)
- RACF (resource access control facility)
  - administration [92](#)
  - CICS segment [93](#)
  - data set profiles [101](#)
  - defining default CICS userid [536](#)
  - defining port of entry profiles [553](#)
  - defining your own resource class names [99](#)
  - generic resource profiles [98](#)
  - group profile [92](#)
  - group profiles [92](#)
  - language segment [94](#)
  - refreshing resource profiles in main storage [98](#)
  - security labels [103](#)
  - security levels [103](#)
  - terminal profiles [553](#)
  - undefined terminals [555](#)
- RACF (Resource Access Control Facility)
  - controlling access to resources [416](#)
  - defining transactions [410](#), [411](#)
  - exempting items from security checking [440](#)
- RACF class
  - DSNR [503](#)
- RACF commands
  - ADDUSER, example for default CICS userid [537](#)
  - DELMEM operand [554](#)
  - RALTER [554](#)
  - RLIST [602](#)
- RACF default resource profiles
  - VCICSCMD general resource class [497](#)
- RACF list of groups option [507](#)
- RACF PassTickets [21](#)
- RACF profiles
  - refreshing [442](#)
- RACF Secured Sign-on [520](#)
- RACF SPECIAL authority [171](#)
- RACROUTE macros, for security [448](#)
- RDO
  - restricting use of transaction [557](#)
- refreshing RACF profiles [442](#)
- requester pipeline
  - configuring
    - for SAML [44](#)
- resource access control facility (RACF) [515](#)
- Resource Access Control Facility (RACF)
  - controlling access to resources [416](#)
  - defining transactions [410](#), [411](#)
  - exempting items from security checking [440](#)
- Resource and command check cross-reference [655](#)
- resource checker (CICS ONC RPC)
  - writing [521](#)
- resource classes, CICSplex SM
  - controlling access to [416](#)
- resource definition parameters
  - RESSEC [573](#)

- resource group
  - DELMEM operand to remove [554](#)
- resource group classes
  - GTERMINL [639](#)
- resource names, CICSplex SM [416](#)
- resource security
  - ACICSPCT general resource class [562](#), [565](#)
  - BCICSPCT general resource class [562](#), [565](#)
  - DCICSDCT general resource class [560](#)
  - defining generic profiles [98](#)
  - defining your own resource class names [99](#)
  - document templates [466](#)
  - ECICSDCT general resource class [560](#)
  - HFS files [570](#)
  - JCICSJCT general resource class [561](#)
  - journals and log streams [561](#)
  - KCICSJCT general resource class [561](#)
  - PCICSPSB general resource class [572](#)
  - profiles and WARNING option [52](#)
  - program specification blocks [572](#)
  - QCICSPSB general resource class [572](#)
  - refreshing profiles in main storage [98](#)
  - temporary storage [568](#)
  - transient data queues [560](#)
  - XHFS parameter [569](#), [570](#)
  - XJCT parameter [561](#)
  - XPCT parameter [562](#), [565](#), [566](#)
  - XPSB parameter [573](#)
  - XRES parameter [466](#)
  - XTST parameter [569](#)
  - z/OS UNIX files [569](#)
- RESSEC [519](#)

## S

- SAML
  - configuring CICS [40](#)
  - containers [683](#)
  - copybooks [683](#)
  - developing applications [597](#)
  - DSECTs [683](#)
  - response codes [683](#)
- SAML tokens
  - expired [45](#)
  - modifying [45](#)
  - not yet valid [45](#)
  - re-signing [45](#)
  - signing [45](#)
- SAML-aware programs
  - augmenting tokens [598](#)
  - outbound request [598](#)
- sample connection exit routine (DSN3SATH) [503](#)
- sample sign-on exit routine (DSN3SSGN) [507](#)
- SASS (single address space) [502](#)
- scoping sign-on definition [551](#)
- SEC system initialization parameter [519](#)
- Secured Sign-on [520](#)
- security
  - access to WUI resources [414](#)
  - authentication [454](#), [455](#), [462](#)
  - AUTHTYPE [498](#)
  - basic authentication [454](#), [455](#), [459](#)
  - CICS system [463](#)
  - command security [495](#)

- security (*continued*)
  - COVA [412](#)
  - COVC [412](#)
  - COVE [412](#)
  - COVG [412](#)
  - COVP [412](#)
  - COVU [412](#)
  - DFHHTML [413](#)
  - document templates [462](#), [463](#)
  - EYUCOVE [413](#)
  - EYUCOVI [413](#)
  - EYULOG [413](#)
  - EYUWREP [413](#)
  - EYUWUI [413](#)
  - identification [454](#), [455](#)
  - inbound ports [462](#)
  - of activities
    - defined activity userid [483](#)
  - of processes
    - defined process userid [483](#)
  - password expiry management [459](#)
  - password phrase expiry management [459](#)
  - planning [401](#)
  - port number
    - security [462](#)
  - proxy authentication [455](#)
  - RACF [493](#)
  - RACF class, DSNR [503](#)
  - resource level [462](#)
  - SAML [40](#)
  - SASS [502](#)
  - sign-on [21](#), [550](#)
  - SSL [467](#)
  - surrogate user checking [498](#)
  - the CICESM interface [104](#), [447](#)
  - using PassTickets [21](#), [550](#)
  - z/OS UNIX files [462](#), [463](#)
  - z/OS UNIX System Services [462](#), [463](#)
- security categories [103](#)
- security checking
  - controlling CICS [439](#)
  - ESM interface [447](#)
  - evaluation sequence [443](#)
  - exempting items [440](#)
  - parameters [441](#)
  - with another ESM [447](#)
  - with RACF [402](#)
- security classification of data and users [103](#)
- security considerations [440](#)
- security handler
  - writing your own [380](#)
- security labels [103](#)
- security levels [103](#)
- security profiles
  - refreshing [442](#)
- security profiles, RACF
  - views protected by [418](#)
- security tasks, example [448](#)
- security token extensions
  - configuring provider pipeline [42](#)
  - configuring requester pipeline [44](#)
- security token of JES spool files [548](#)
- Security Token Service configuration file [46](#)
- security, surrogate [498](#)

- SECURITYPREFIXID [529](#)
- segment
  - CICS [93](#)
  - LANGUAGE [94](#)
- server program
  - security considerations [515](#)
- SETOPTS command
  - generic terminal profiles [554](#)
  - generic user profiles [101](#)
- shared data tables
  - CONNECT security checks [526](#)
  - server authorization security check [525](#)
- sign-off
  - after persistent sessions restart [94](#)
  - after XRF takeover [94](#)
  - process [552](#)
- sign-off process [552](#)
- sign-on
  - after persistent sessions restart [94](#)
  - after XRF takeover [94](#)
- sign-on exit routine [507](#)
- sign-on security [21](#), [550](#)
- signature verification
  - SAML [45](#)
- simulated CICS security [439](#)
- single address space (SASS) [502](#)
- SIOCTLCTL ioctl [467](#), [471](#)
- SIT parameters, CICS
  - security-related [441](#)
- skew time
  - changing [45](#)
  - setting [45](#)
- SMF (System Management Facility) [89](#)
- SNA LU
  - SNA ACB access [546](#)
- SP800-131A [180](#)
- specific connection
  - MRO logon security checks [515](#)
- spool files, security token [548](#)
- SPOOLOPEN commands [548](#)
- SQL
  - dynamic [510](#)
  - static [510](#)
- SSL
  - client certificate authentication [454](#), [455](#)
  - for CICS as an HTTP client [467](#)
- SSL(ATLSAWARE) [472](#)
- STARTED general resource class [639](#)
- started task
  - authorizing CICS procedures [533](#)
- started transaction security [562](#), [565](#), [566](#)
- STS configuration file [46](#)
- sts-config.xml [46](#)
- sts.xml [46](#)
- sts.xsd [46](#)
- SUBSYSNM general resource class [639](#)
- SURROGAT general resource class [547](#), [558](#), [639](#)
- SURROGAT transaction [557](#)
- surrogate job submission
  - to JES internal reader [547](#)
- surrogate security checking [483](#)
- surrogate user security
  - post-initialization processing [60](#)
- symmetric encryption [164](#)

- system data set
  - protecting [532](#)
- system initialization parameters, CICS
  - PSBCHK [573](#)
  - XDCT [560](#)
  - XHFS [569](#), [570](#)
  - XJCT [561](#)
  - Xname parameters [607](#), [615](#)
  - XPCT [562](#), [565](#), [566](#)
  - XPSB [573](#)
  - XRES [466](#)
  - XTST [569](#)
- System Management Facility (SMF) [89](#)

## T

- tasks, example
  - security [448](#)
- TCIPSERVICE [472](#)
- TCIPSERVICE resource definition
  - security [462](#)
  - SSL
    - URIMAP resource definition [462](#)
- temporary storage
  - authorizing access to the named counter pools [543](#)
  - authorizing access to the TS pools [542](#)
  - authorizing access to TS servers [542](#)
  - defining resource classes [568](#)
- terminal
  - emulators [37](#)
- TERMINAL definition [553](#)
- TERMINAL general resource class [639](#)
- terminal security
  - CEDA LOCK command [557](#)
  - controlling access [553](#)
  - obtaining CICS-related data for a user [95](#)
  - preset [556](#)
  - sign-on [550](#)
  - terminal profiles [553](#)
  - undefined terminals [555](#)
  - universal access authority [555](#)
  - using MVS system console as CICS terminal [558](#)
- TIMEOUT [94](#)
- TLS12 [180](#)
- trace points [471](#)
- transaction security
  - started transactions [562](#), [565](#), [566](#)
  - XJCT parameter [562](#), [565](#), [566](#)
  - XPCT-checked transactions [562](#), [565](#), [566](#)
- transactions
  - in a CMAS
    - defining to RACF [410](#)
  - in a MAS
    - defining to RACF [411](#)
  - Kerberos [37](#)
- transient data
  - security considerations [560](#)
- trigger level transactions
  - default security for [537](#)
- troubleshooting
  - Kerberos [620](#)
  - Passtickets [622](#)
- trust client
  - invoking [380](#)

TSO command  
refreshing using TSO command [98](#)

## U

UNIX authentication [519](#)  
untrusted certificate, marking [172](#)  
URP\_DISASTER response  
to resource checker [522](#)  
URP\_EXCEPTION response  
to resource checker [522](#)  
URP\_INVALID response  
to resource checker [522](#)  
URP\_OK response  
to resource checker [522](#)  
user ID  
security [462](#), [463](#)  
USER parameter on CICS JOB statement [534](#)  
user profiles  
refreshing [442](#)  
user security [516](#)  
userid  
ADDUSER to add default CICS userid [537](#)  
controlling propagation of [547](#)  
defining CICS default user [536](#)  
defining for CICS [534](#)  
of CICS region as security token [548](#)  
surrogate job submission [547](#)

## V

validate [41](#)  
validating [41](#)  
VCICSCMD general resource class [497](#), [654](#)  
verifying  
signatures [45](#)  
views  
security considerations [414](#)  
views protected by security profiles [418](#)  
VSAM data sets, and BWO [541](#)  
VSAM data sets, and Dynamic Volume Count [541](#)  
VSAM ESDSs, access to [541](#)  
VTAMAPPL general resource class  
defining profiles [546](#)

## W

WARNING option [52](#)  
web application [257](#), [260](#), [266](#), [269](#), [276](#)  
web services  
configuring  
Kerberos [36](#)  
WHEN operand of PERMIT  
WHEN operand [555](#)

## X

XCMD system initialization parameter [519](#)  
XDCT, system initialization parameter [560](#)  
XHFS, system initialization parameter [569](#), [570](#)  
XJCT, system initialization parameter [561](#)  
Xname, system initialization parameters [607](#), [615](#)  
XPCT-checked transaction security [562](#), [565](#), [566](#)

XPCT, system initialization parameter [562](#), [565](#), [566](#)  
XPPT system initialization parameter [519](#)  
XPSB, system initialization parameter [573](#)  
XRES, system initialization parameter [466](#)  
XRF (extended recovery facility)  
XRFSOFF operand [94](#)  
XTST, system initialization parameter [569](#)  
XUSER system initialization parameter [519](#)  
XUSER, system initialization parameter [557](#)

## Z

z/OS Communications Server  
generic resource names [545](#)  
z/OS UNIX file security  
XHFS parameter [569](#)  
z/OS UNIX files  
security [462](#), [463](#)  
z/OS UNIX Systems  
Services  
security [462](#), [463](#)



