CICS Transaction Server for z/OS
6.1

CICS Security Guide

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 Release 1 (product number 5655-YA15655-BTA) and to all subsequent releases and modifications until otherwise indicated in new editions.

The IBM CICS Transaction Server for z/OS, Version 6 Release 1 may be referred to in the product and documentation as CICS Transaction Server for z/OS, 6.1 .

Contents

About this PDF xi

Chapter 1. What does security mean for CICS? 1

Chapter 2. CICS security is a team sport 5

Chapter 3. How it works: Identification in CICS 7
- Identity propagation 14

Chapter 4. How it works: Authentication in CICS 17
- 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 29
- OAuth 2.0 29
- OpenID Connect 30
- Kerberos 31
  - Configuring RACF for Kerberos 33
- Lightweight Third-Party Authentication (LTPA) 36
- SAML 37
  - Configuring CICS for SAML 38
- User registries 45

Chapter 5. How it works: Authorization in CICS 47
- Transaction security 47
- Resource security 48
  - Use of the WARNING option 50
- Command security 50
- Intercommunication security 52
- Role authorization 55
- Application-specific security (QUERY SECURITY) 55
- Post-initialization processing 57
- Surrogate security 57

Chapter 6. How it works: Integrity and confidentiality in CICS 61
- Data in motion 61
  - Transport Layer Security (TLS) for IP connections 61
  - TLS cipher suites 65
  - Implementation options for TLS 65
  - Performance considerations for TLS connections 69
  - SOAP messages 71
  - 3270 sessions 71
- Data in memory 72
- Data at rest 73

Chapter 7. How it works: Auditing in CICS 75

Chapter 8. How it works: Securing CICS with RACF 79
RACF administration..... 80
RACF users and roles..... 81
RACF user profiles..... 81
RACF group profiles..... 83
RACF classes and profiles..... 84
RACF classes..... 84
RACF profiles for CICS classes..... 87
Security administration tasks and the RACF commands to run them..... 89
Security classification of data and users..... 90
Invoking an external security manager..... 91

Chapter 9. Security through the network layers..... 93
Transport Layer Security (TLS)..... 93
Application Transparent Transport Layer Security (AT-TLS)..... 94
Changing TLS protocol level or ciphers safely..... 94
Providing support to update to TLS 1.3..... 98
Security for TCP/IP clients..... 99
Authenticating ECI users..... 99
SSL encryption..... 100
The SSL cache..... 100
The SSL pool..... 101
Configuring CICS to use SSL..... 101
Setting up profiles in RACF..... 102
Requesting a certificate from a certificate authority..... 103
Building a key ring manually..... 104
Building a key ring with certificates using DFH$RING..... 105
Creating new RACF certificates..... 107
Associating a RACF user ID with a certificate..... 108
Using an existing certificate that is not owned by the CICS region user ID..... 108
Configuring a RACF site certificate for use with CICS TS..... 109
Making a certificate untrusted..... 109
System initialization parameters for SSL..... 110
TCPIPSERVICE attributes for SSL..... 110
Creating a TLS cipher suite specification file..... 112
Customizing encryption negotiations..... 113
Making your CICS TS system conformant to NIST SP800-131A..... 114
Configuring LDAP for CICS use..... 115
Using certificate revocation lists (CRLs)..... 116
Intrusion Detection Services..... 119

Chapter 10. Security for MRO..... 121
How it works: CICS MRO security..... 121
MRO connection (bind-time) security..... 121
MRO link security..... 124
MRO user security..... 126
Designing security for MRO..... 127
Design example: MRO connection – using only the task user ID..... 128
Design example: MRO connection – using only the link user ID..... 131
Design example: MRO connection – using both the task ID and the link user ID..... 133
Configuring security for MRO..... 136
Allowing the regions to use MRO..... 136
Allowing regions to connect to each other using MRO..... 136

Chapter 11. Security for IPIC (IP interconnectivity)..... 139
How it works: CICS IPIC security.....139
IPIC connection security 139
IPIC transport security 142
IPIC link security 143
IPIC user security.....143
Designing security for IPIC.....145
Design example: Securing CICS-to-CICS with an IPIC connection within a sysplex.....147
Design example: Securing CICS-to-CICS with an IPIC connection that uses TLS.....151
Design example: Securing client to CICS with a trusted IPIC connection..... 156
Configuring security for IPIC..... 159
Installing a resource with a user ID attribute.....159

Chapter 12. Security for LU6.2 connections..... 161

Chapter 13. Security for LU6.1 connections..... 163

Chapter 14. Security for CICS Liberty..... 165
How it works: CICS Liberty security.....165
How it works: Securing Liberty web applications..... 169
How it works: Securing Link to Liberty applications..... 178
How it works: Liberty angel process.....180
Designing security for CICS Liberty applications.....180
Designing application security for Liberty with the angel process..... 180
Designing security for Liberty web applications..... 181
Designing security for Link to Liberty applications..... 199
Configuring security for CICS Liberty..... 203
Configuring the angel process..... 204
Configuring authentication in CICS Liberty..... 209
Configuration authorization in CICS Liberty..... 224
Configuring confidentiality and integrity..... 230

Chapter 15. Security for Java applications.....243
Configuring security for OSGi applications..... 243
Configuring security for a Liberty JVM server..... 243
The Liberty angel process..... 246
Authenticating users in a Liberty JVM server.....250
Authorizing users to run applications in a Liberty JVM server.....252
Authorization using SAF role mapping..... 253
Configuring security for a Liberty JVM server with the Enterprise Java security API..... 255
Configuring security for a Liberty JVM server by using an LDAP registry..... 260
Configuring security for remote JCICSX API development..... 263
Configuring SSL (TLS) for a Liberty JVM server using a Java keystore..... 267
Configuring SSL (TLS) for remote JCICSX API development.....268
Using the syncToOSThread function 270
Authorizing applications by using OAuth 2.0..... 271
Enabling a Java security manager.....274

Chapter 16. Security for SOAP web services..... 277
How it works: SOAP message security 278
How it works: Authentication for CICS with SOAP message security..... 281
How it works: Signing SOAP messages 282
How it works: SOAP message encryption 284
Designing security for CICS web service providers..... 286
Design example: Securing a direct request with TLS client authentication..... 289

Design example: Asserting an identity to the CICS web service provider....293
Designing security for CICS web service requesters....297
Design example: Securing a web service request with TLS client authentication....299
Design example: Securing a web service request with identity assertion....302
Configuring SOAP message security for CICS web services....305
Installing the prerequisites for WS-Security support....306
Configuring the pipeline for WS-Security....307
Configuring provider mode web services for identity propagation....310
Configuring RACF for WS-Security....312
Invoking the Trust client from a message handler....313
Writing a custom security handler....314

Chapter 17. Security for 3270....317
How it works: 3270 security....317
3270 connections to z/OS....317
CICS connection to VTAM to enable 3270 terminals....318
3270 devices that connect to CICS....319
3270 terminal authentication....324
3270 terminal sign-off....325
Designing 3270 terminal security....326
Design example: Dynamically defined 3270 terminal connects and authenticates to a CICS region....327
Design example: 3270 terminal with a preset user ID connects....330
Configuring 3270 terminal security....333

Chapter 18. Security for CICSPlex SM....335
Security planning for CICSPlex SM....335
Implementing CICSPlex SM security....336
Determining who requires access to CICSPlex SM resources....336
General requirements for CICSPlex SM security....340
Creating profiles for the CICSPlex SM data sets....340
Determining the agent user ID for CICSPlex SM components....341
Defining the CICSPlex SM started tasks....342
Defining the CICSPlex SM transactions in a CMAS....343
Defining the transactions in a managed CICS region....344
Setting up CICSPlex SM Web User Interface security....344
Specifying CICSPlex SM resource names in profiles....349
Security for platforms and applications....374
Activating simulated CICS security....377
BAS security considerations....378
Considerations for CICS surrogate security checks....379
Activating security for CICSPlex SM....379
Refreshing RACF profiles for CICSPlex SM....380
CICSPlex SM security checking sequence....381
Invoking a user-supplied external security manager....385
An overview of the CICSPlex SM ESM interface....385
Invoking an external security manager....385
CICSPlex SM security control points....386
Example tasks: security....386
Example: Protecting all CICSPlex SM resources....387
Example: Giving CICSPlex SM operators appropriate authorizations....387
Example: Giving a user read access to all transactions on MVS system A....388

Chapter 19. Security for Node.js applications....389

Chapter 20. Security for CICS web support....391
User IDs for access to document templates and z/OS UNIX files used by CICS Web support....391

CICS as an HTTP server: authentication and identification ... 392
CICS as an HTTP client: authentication and identification ... 393
Identifying HTTP users ... 394
Authenticating HTTP users ... 396
Password expiry management for HTTP basic authentication ... 397
CICS system and resource security for CICS web support ... 399
Security for inbound ports ... 399
Security for CICS system components ... 400
Resource and transaction security for application-generated responses ... 400
Resource-level security for static responses using document templates ... 402
SSL with CICS web support ... 403
Introduction to Application Transparent Transport Layer Security (AT-TLS) ... 404
Security for Atom feeds ... 414

Chapter 21. Security for platforms and applications ... 417

Chapter 22. Security in BTS ... 421
Resource security in BTS ... 421
Process and activity user IDs ... 421
Attach-time security for processes and activities ... 422
Command security in BTS ... 422

Chapter 23. Security for the CICS-MQ adapter ... 423
Implementing security for CICS-MQ adapter transactions ... 423
CICS-MQ adapter user IDs ... 424
Command security for MQCONN and MQMONITOR resources ... 424
Surrogate user security for MQMONITOR resources ... 425
IBM MQ connection security for the CICS-MQ adapter ... 425

Chapter 24. Security for the CICS-MQ bridge ... 427

Chapter 25. Security for data sources ... 431
Security for Db2 ... 431
Controlling access to Db2-related resources in CICS ... 432
Providing authorization IDs to Db2 for the CICS region and for CICS transactions ... 439
Authorizing users to access resources in Db2 ... 446
Db2 multilevel security and row-level security ... 449
Security for DBCTL ... 449
PSB authorization checking by CICS ... 450
Security for data sets: encryption ... 450

Chapter 26. Security for EXCI ... 453
Using MRO logon and bind-time security ... 453
Defining DFHAPPL FACILITY class profiles for an EXCI region ... 453
Link security for EXCI ... 454
User security for EXCI ... 454
Surrogate user checking for EXCI ... 455

Chapter 27. Security for ONC RPC ... 457
Security in ONC RPC ... 457
Security in CICS and its effect on CICS ONC RPC operations ... 457
RACF Secured Sign-on for ONC RPC clients ... 458
Writing the resource checker ... 459
Reference information for the resource checker ... 459

Chapter 28. Security for data tables ... 463

Security for CICS shared data tables..... 463
Security checking for data tables..... 463
SDT server authorization security check..... 463
CONNECT security checks for AORs..... 464
Security for coupling facility data tables..... 465
Authorizing server access to a list structure..... 466
Authorizing the server..... 466
Authorizing a CICS region to a CFDT pool..... 466
Authorizing a CICS region to a coupling facility data table..... 467
File resource security checking..... 467

Chapter 29. Security for CICS resources..... 469
CICS system resource security..... 469
CICS installation requirements for RACF..... 469
Specifying the CICS region user ID..... 470
Authorizing CICS procedures to run under RACF..... 471
Using the ICHRIN03 table for started tasks..... 471
Using STARTED profiles for started jobs..... 471
Defining user profiles for CICS region user IDs..... 472
Defining the default CICS user ID to RACF..... 474
Authorizing access to z/OS log streams..... 475
Authorizing access to CICS data sets..... 476
Authorizing access to the temporary storage pools..... 479
Authorizing access to temporary storage servers..... 480
Authorizing access to named counter pools and servers..... 481
Authorizing access to SMSVSAM servers..... 482
Authorizing access to the CICS region..... 483
Controlling the opening of a CICS region's z/OS Communications Server ACB..... 484
Controlling userid propagation..... 484
Surrogate job submission in a CICS environment..... 485
JES spool protection in a CICS environment..... 485
Authorizing use of **HPO** in **PARM** parameter on an EXEC PGM=DFHSIP statement or in SYSIN..... 486
Setting up PassTickets..... 486
Supporting parallel PassTicket requests for the same user ID..... 487
Using PassTickets in FEPI applications..... 487
The sign-on process..... 488
The sign-off process..... 489
Controlling access to CICS from specific ports of entry..... 490
Defining port of entry profiles..... 491
Terminal profiles..... 491
Defining a profile of an individual terminal..... 492
Defining a profile of a group of terminals..... 492
Universal access authority for undefined terminals..... 492
Conditional access processing..... 493
Console profiles..... 493
Preset terminal security..... 494
Controlling the use of preset security..... 495
Using a z/OS system console as a CICS terminal..... 496
Security for transient data..... 498
Security for journals and log streams..... 499
Security for started transactions..... 500
Transactions started at terminals..... 501
Transactions started by EXEC CICS START..... 501
Security for transactions started with EXEC CICS RUN TRANSID..... 503
Security for XPCT-checked transactions..... 503
Security for applications..... 504
Security for temporary storage..... 505

Security for z/OS UNIX files....506
Implementing security for z/OS UNIX files....507
Security for program specification blocks....509
Security checking of transactions running under CEDF, CEDG, CEDX, or CEDY....509
Defining generic profiles for resources....510
Security for submitting a JCL job to the internal reader....511
Configuring RACF for identity propagation....513

Chapter 30. Auditing CICS....515
Compliance data collection....515
Retrieving SMF 1154 subtype 80 records from CICS....515
Example of formatted SMF 1154 subtype 80 records....515
Auditing CICS configuration with IBM Health Checker for z/OS....516
Identifying changes to resources (resource signatures)....517
Resource signature field values....518
Classifying CICS regions with region tagging....522
Auditing installed resource....523
Auditing SPI commands....524
Auditing sign-on and sign-off....527
Auditing bind time security (LU6.2)....529
Auditing authorization....529
Auditing RACF....531

Chapter 31. Developing SAML applications....533
Developing a SAML-aware initial program....533
Pattern: developing a SAML-aware initial program....533
Pattern: reusing a validated SAML token....534
Developing a program that uses a validated SAML token in an outbound request....534
Developing a program that creates or augments SAML tokens....534

Notices....537

Index....543

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 2024-05-21 (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 security 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.

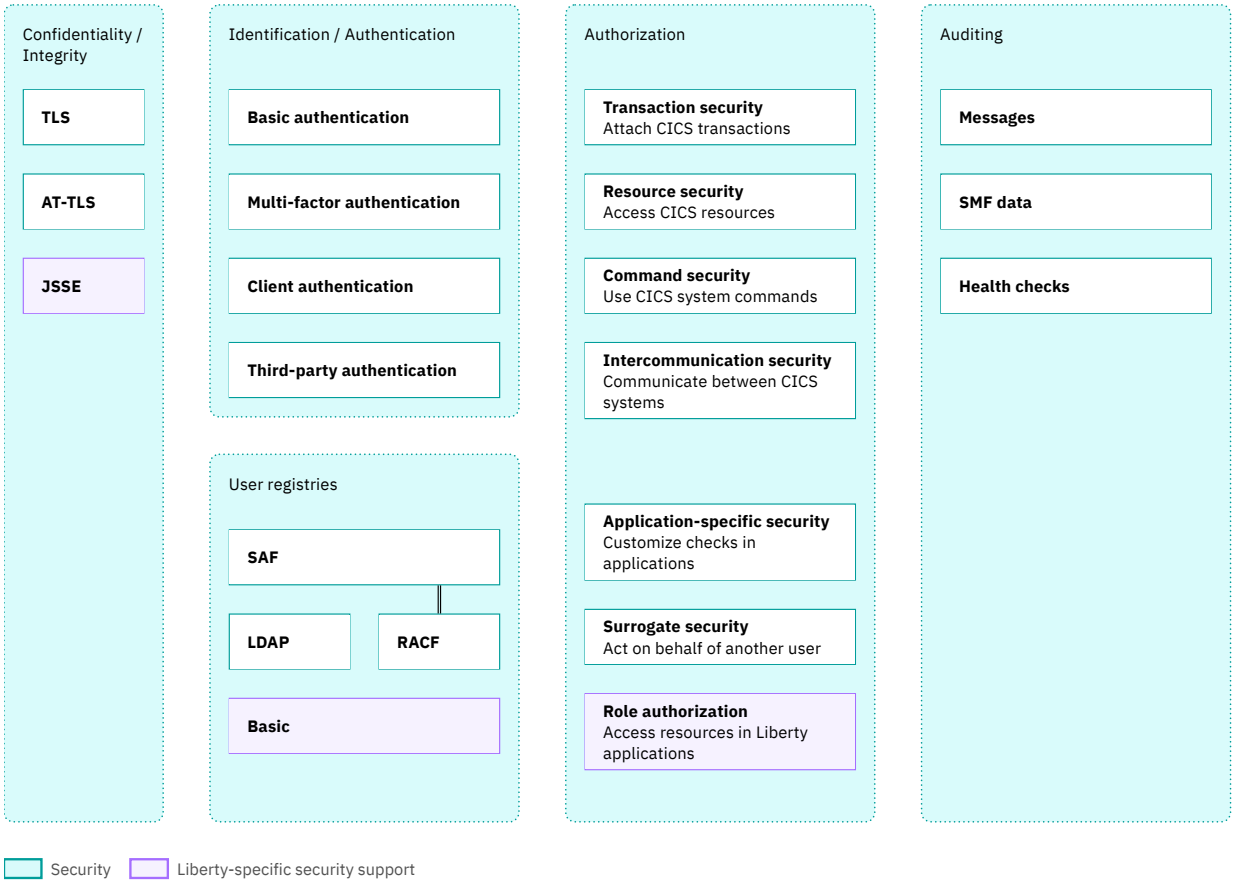


Figure 1. Security options that are available in CICS

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

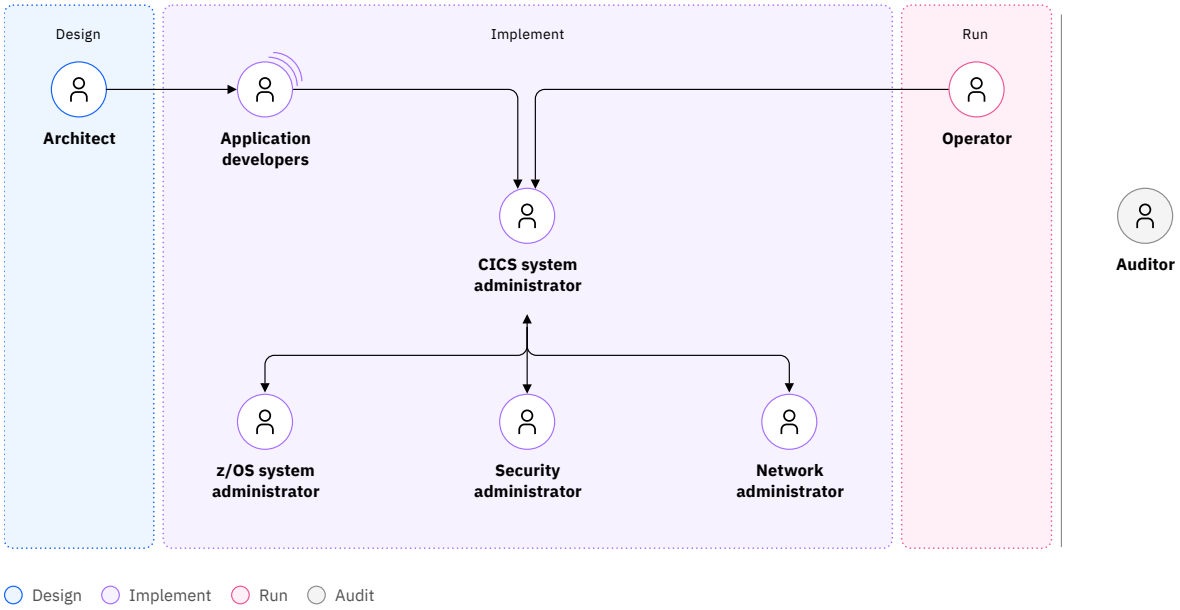
CICS does not protect your system from application programs that use undocumented or unsupported interfaces to bypass CICS security. You are responsible for ensuring that such programs are not installed on your system.

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 systems 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
Asserted user ID	User ID flowed from a trusted remote system.
Authenticated user ID	User ID shared from a remote system that includes credentials that CICS uses to authenticate the user.
“Client user ID” on page 11	An identity that comes from outside of CICS with some credentials to be authenticated.
EDF user ID	The user ID under which the Execution Diagnostic Facility is run to test a transaction.
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.
Link user ID	A user ID passed as part of the bind process or defined in a resource definition that is associated with the connection.
Resource user ID	A user ID defined in a resource definition and used to run a task or used to authorize a request.
Session user ID	A user ID that can be associated with a session, such as a 3270 terminal session.
Task user ID	The task user ID is obtained from whoever or whatever started the task.
Technical user ID	Alternative name for functional user ID.

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
Batch region user ID	A batch region user ID, is the user ID under which a request will run. It is typically a functional user ID
“Execution user ID” on page 12	An execution user ID, typically a functional user ID, is paired with a surrogate user ID.
“Surrogate user ID” on page 12	A surrogate user has the authority to start work on behalf of another user (the execution user).

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**
Distributed identity	A distributed identity represents user identity information from a different security realm that is flowed into CICS.
Java™ security subject	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**
CICS default user ID	One of the user IDs that is associated with the CICS region to protect resources in the absence or other, more specific, user identification.
CICS region user ID	One of the user IDs that is associated with the CICS region that is associated with the tasks started when CICS starts.
Kerberos service principal user ID	One of the user IDs that is associated with the CICS region that is related to Kerberos use.
PLT user ID	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.

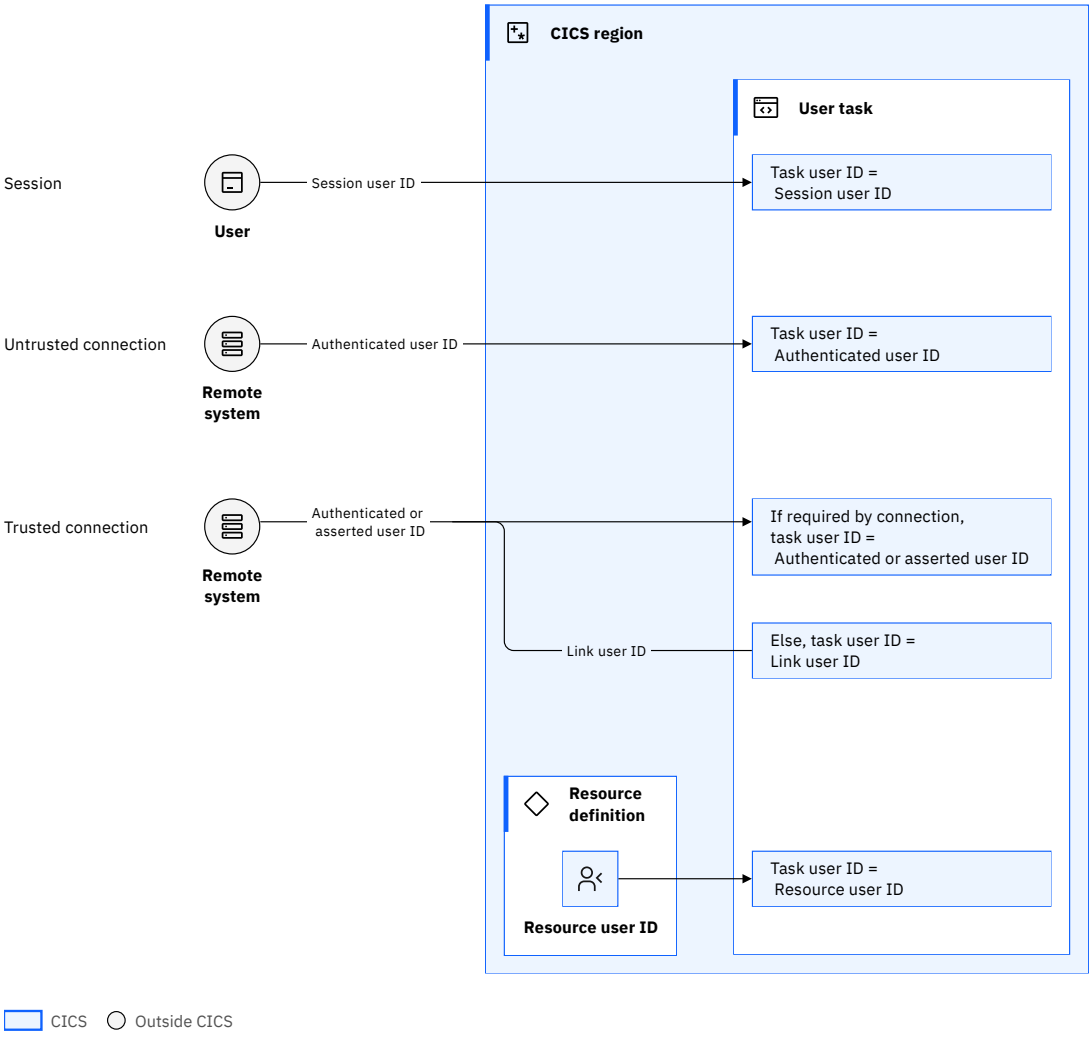


Figure 2. How users are identified in CICS

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 for:

- Submitting jobs to the JES internal reader
- 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 security 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 transaction start EP adapter.	User ID under which the transaction is started
DB2CONN	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
DB2ENTRY	AUTHID (used in API requests), AUTHTYPE	For DB2ENTRY authorization (not RACF)
MQMONITOR	MONUSERID	Task user ID of the MQ monitor
	USERID	Task user ID associated with messages
TDQUEUE	JOBUSERID	Job user ID for JCL that is submitted by using transient data queues
	USERID	Task user ID of triggered task
URIMAP	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**
CONNECTION (APPC)	SECURITYNAME	Link user ID
IPCONN	SECURITYNAME	Link user ID
SESSIONS	USERID	Link user ID
TERMINAL	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.

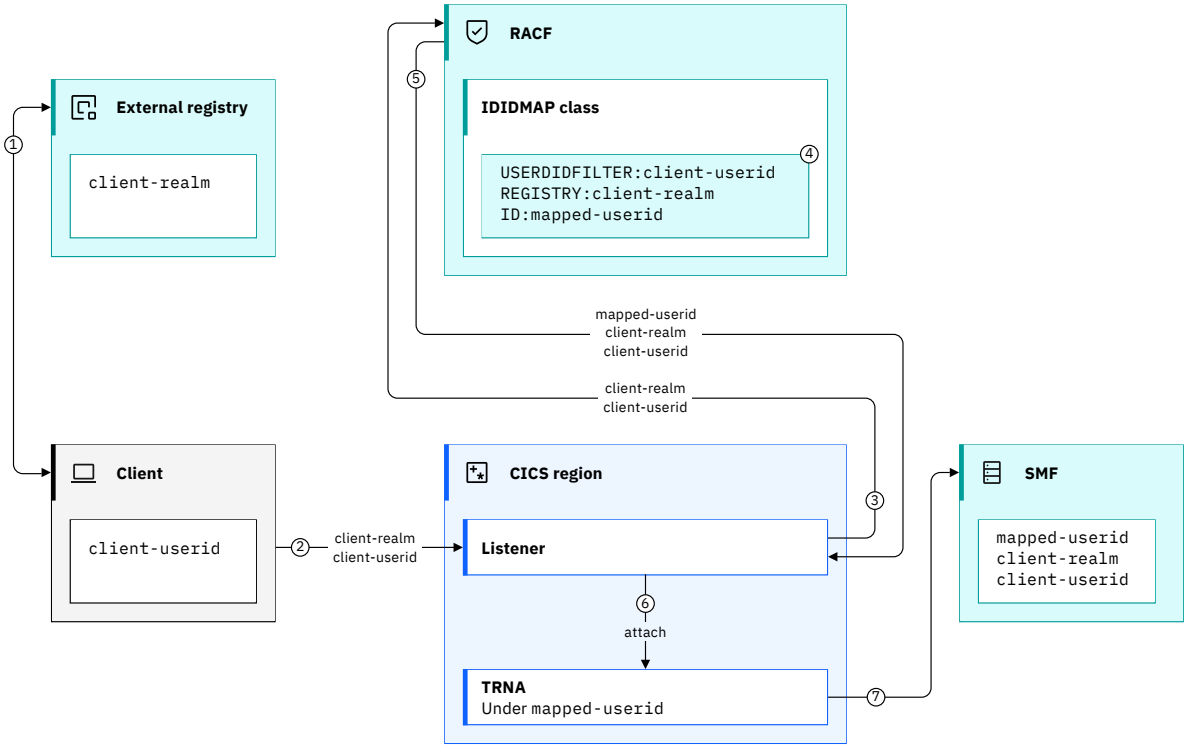


Figure 3. How identity propagation works with CICS

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 security for a Liberty JVM server by using an LDAP registry.

Chapter 4. How it works: Authentication in CICS

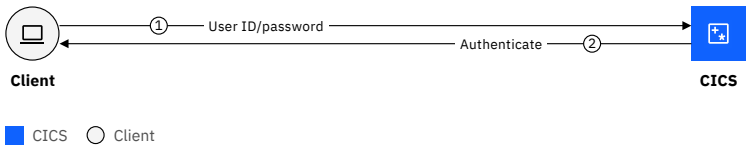
CICS handles the authentication process. It requests credentials from a user, decodes the authentication information if necessary, calls RACF or a third-party authentication server to authenticate the supplied credentials, and rejects the request if the authentication fails. It supports different forms 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

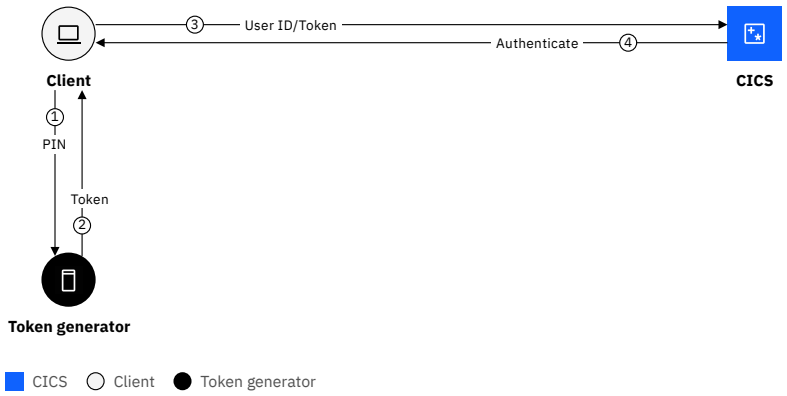
This form of authentication uses credentials in the form of a user ID and password, a passphrase, or a PassTicket.



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.



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

Client authentication

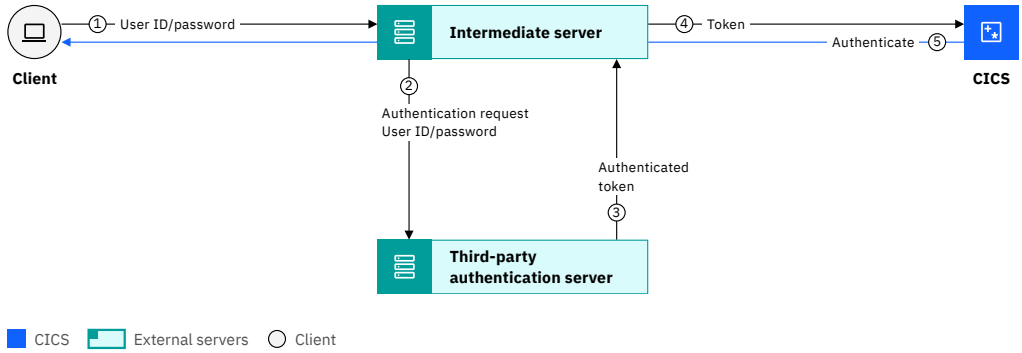
This form of authentication uses a TLS certificate to identify the client. Either CICS Liberty, CICS TLS support or Application Transparent Transport Layer Security (AT-TLS) can be used.

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)



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.

	How it works: Passwords and passphrases	**How it works: PassTickets ("1" on page 20)**	**MFA**	**How it works: X.509 Certificates**	**JWT**	**OAuth 2.0 or How it works: OpenID Connect**	**How it works: Kerberos**	**SAML**	**LTPA**	**ICRX**
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)*

	How it works: Passwords and passphrases	**How it works: PassTickets ("1" on page 20)**	**MFA**	**How it works: X.509 Certificates**	**JWT**	**OAuth 2.0 or How it works: OpenID Connect**	**How it works: Kerberos**	**SAML**	**LTPA**	**ICRX**
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 4 on page 22 shows how one CICS region creates a PassTicket for a signed-on user to access another system.

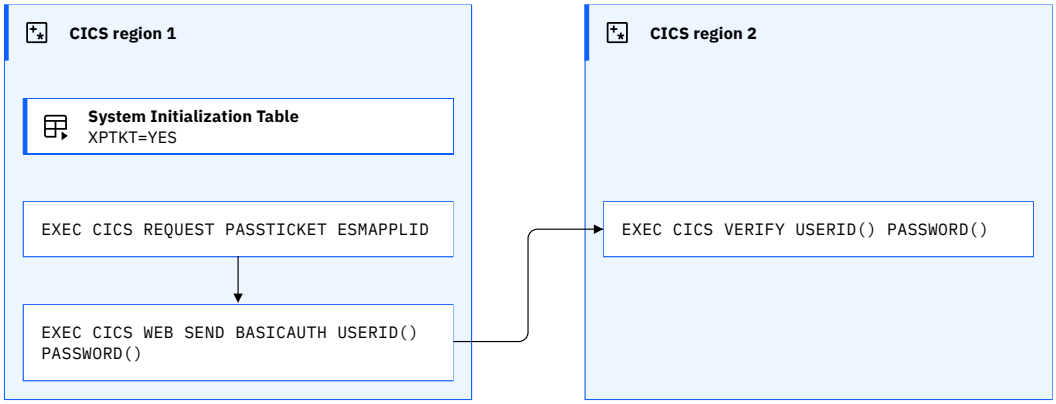


Figure 4. PassTickets

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 23
- "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 5 on page 23 illustrates in-band authentication with an MFA solution, such as RSA SecurID:

- The user logs on with a user ID and an RSA SecurID token and PIN.
- When RACF determines that the user is an MFA user, RACF calls the MFA Server.
- MFA Server calls RACF to retrieve the user's MFA factor details.
- MFA Server validates the user's authentication factors by calling the RSA server.
- RACF uses the return codes from the MFA Server to allow or deny the logon.

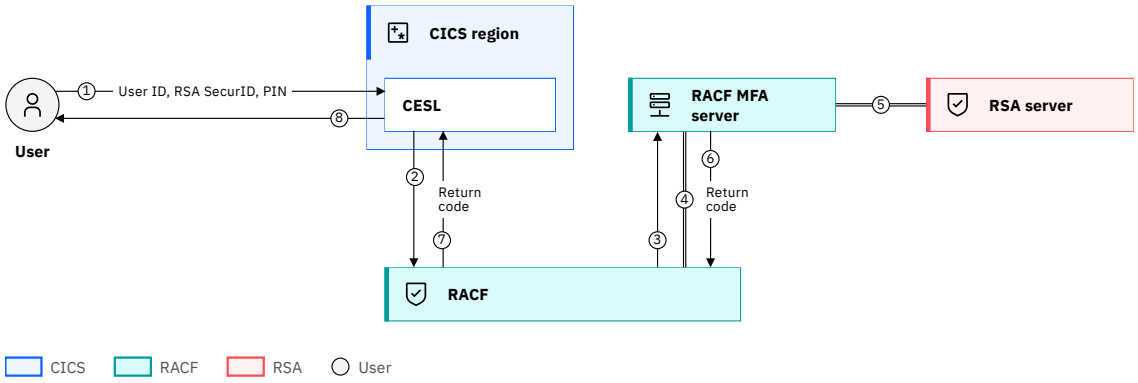


Figure 5. MFA in-band authentication

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 6 on page 24 illustrates out-of-band authentication:

- The user pre-authenticates to the MFA Server through a smartphone app.
- MFA Server stores the pre-authentication record in a session cache.
- The user logs on with a user ID and with the token-returned One Time Passcode (OTP).
- When RACF determines that the user is an MFA user, RACF calls the MFA Server.
- MFA Server calls RACF to retrieve the user's MFA factor details.
- MFA Server checks the session cache for valid pre-authentication and OTP.
- RACF uses the return codes from the MFA Server to allow or deny the logon.

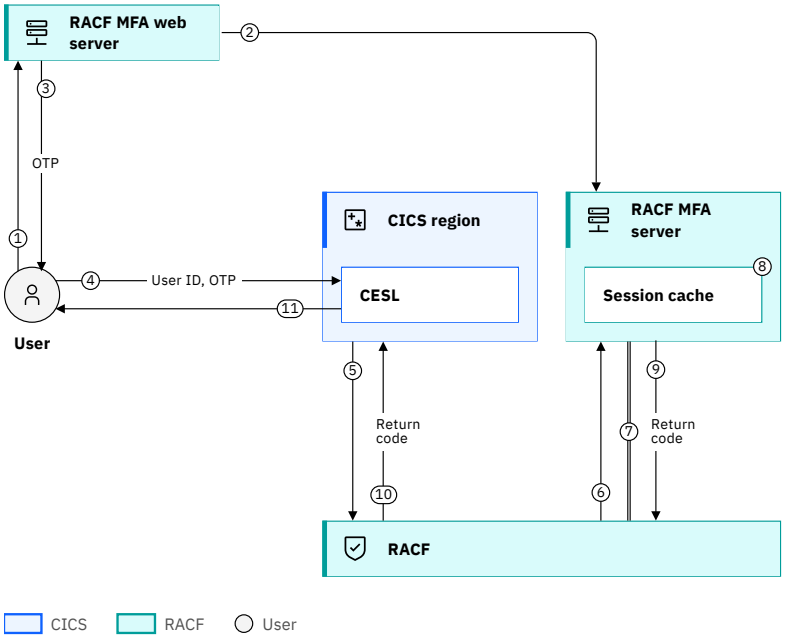


Figure 6. MFA out-of-band authentication

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?.

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 7 on page 25 illustrates the exchange.

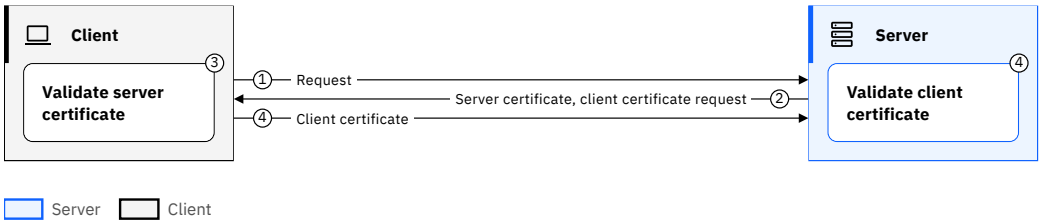


Figure 7. TLS client authentication flow

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 it's associations: for example, that the user, who's 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 8 on page 27 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.

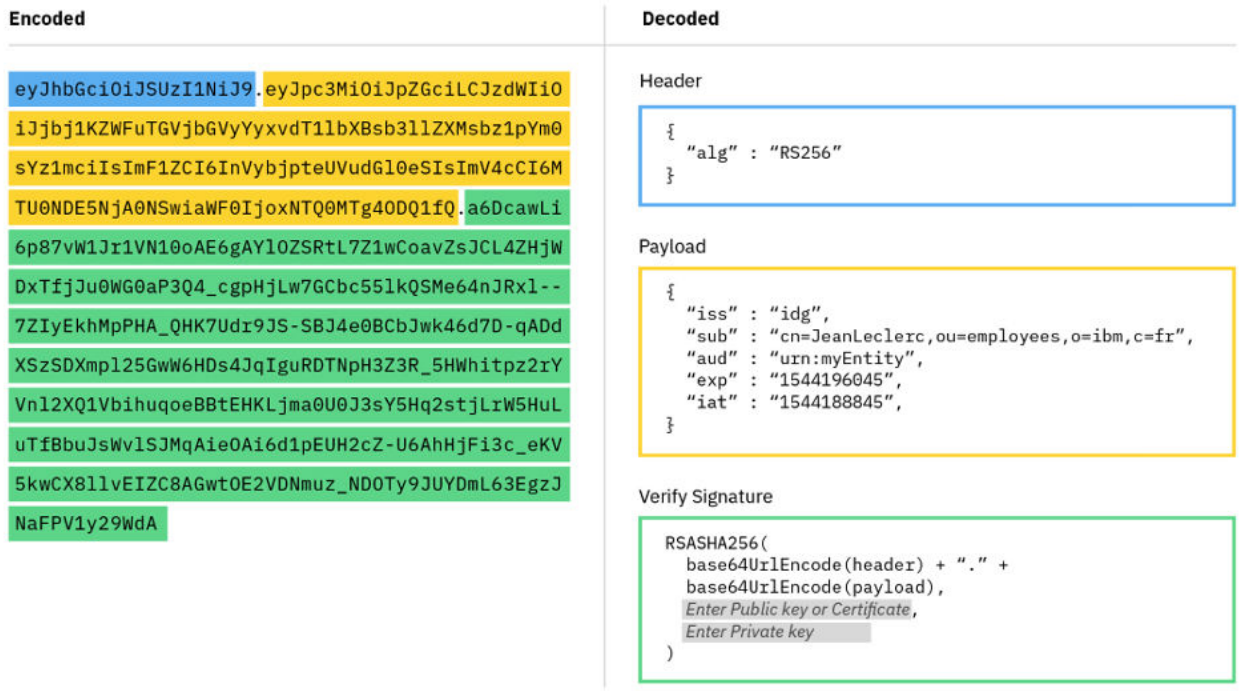


Figure 8. Anatomy of a JWT

Figure 9 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.

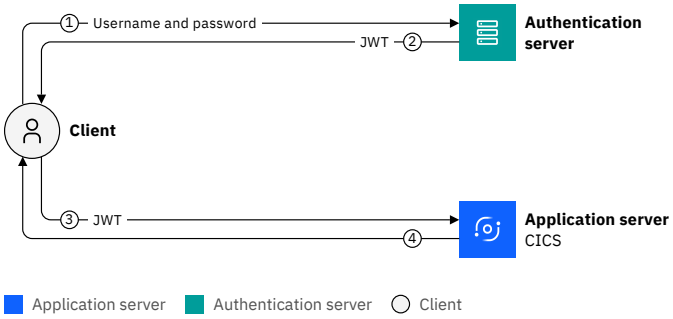


Figure 9. JWT authentication flow

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.

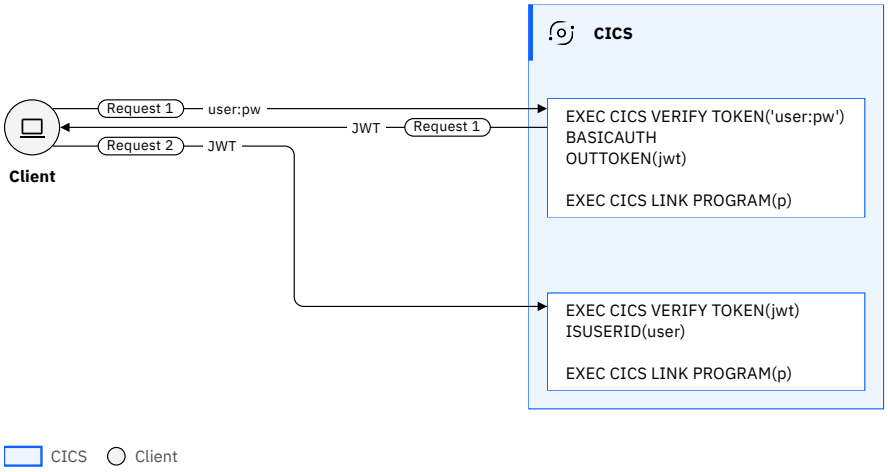


Figure 10. Converting basic authentication credentials to a JWT

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 11 on page 29 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 11. Example RDEFINE statements to create profiles for JWT support

OAuth 2.0

OAuth 2.0 is an open standard for delegated authorization that is defined in RFC 6749 and RFC 6750. The OAuth 2.0 authorization framework enables a user to grant a third-party application access to information

that is stored with another HTTP service without sharing their access permissions or the full extent of their data.

To see which CICS access methods support OAuth 2.0, see Which authentication method can I use with which access method?.

The OAuth 2.0 protocol facilitates the authorization of one site to access and use information that is related to the user's account on another site.

CICS Liberty supports OAuth 2.0, and can be used as an OAuth 2.0 service provider endpoint and an OAuth 2.0 protected resource enforcement endpoint. Liberty supports persistent OAuth 2.0 services.

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 12 on page 31 show the steps in a typical OpenID Connect flow for a grant type of *authorization code*.

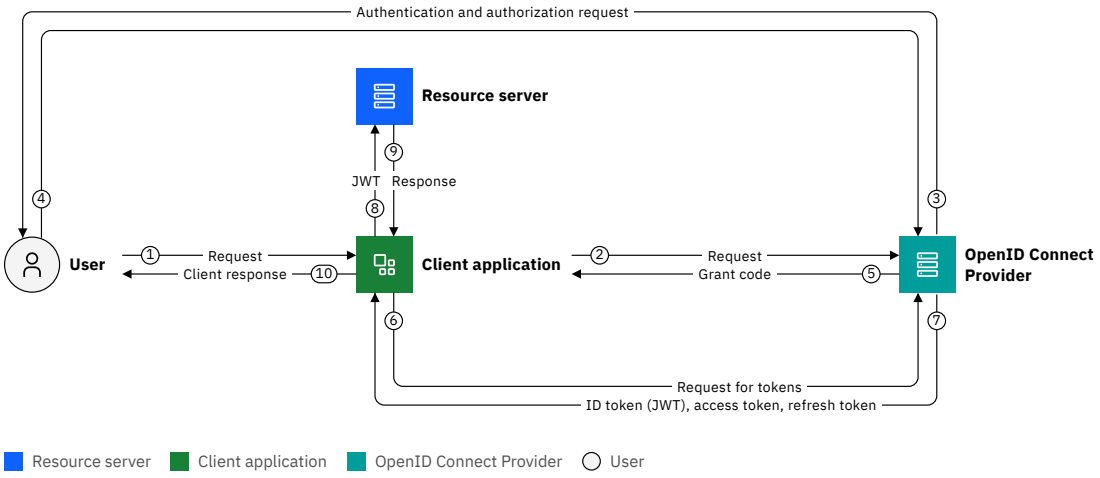


Figure 12. OpenID Connect flow

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.

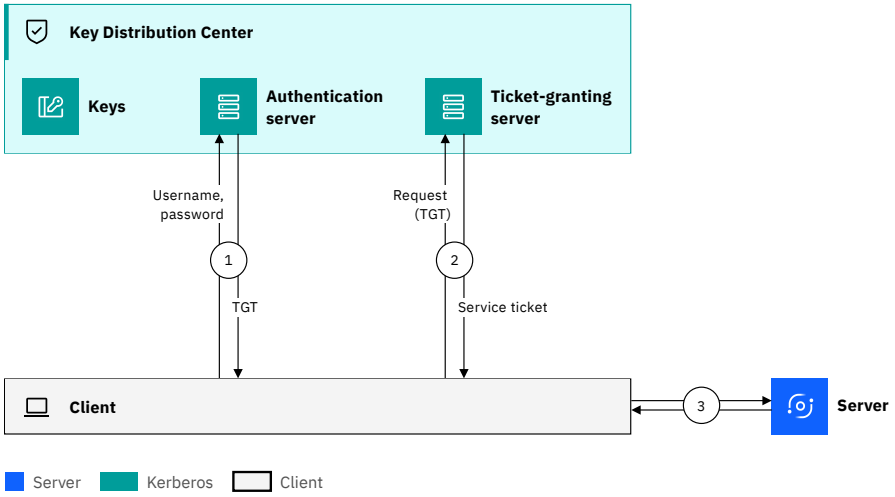


Figure 13. How Kerberos works

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.

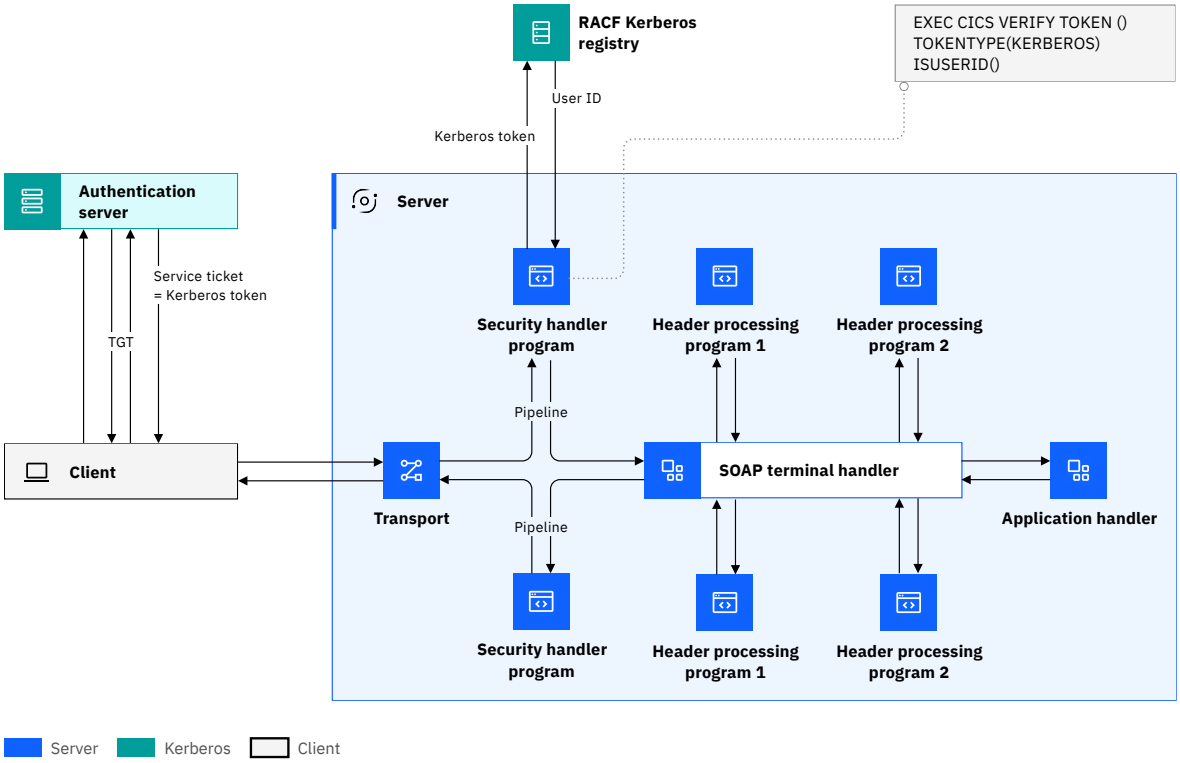


Figure 14. CICS support for Kerberos

Configuring RACF for Kerberos

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

Note: These instructions apply only to the use of local principals.

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:

   ```
   SETROPTS CLASSACT(KERBLINK)
   ```

   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 opintion 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.

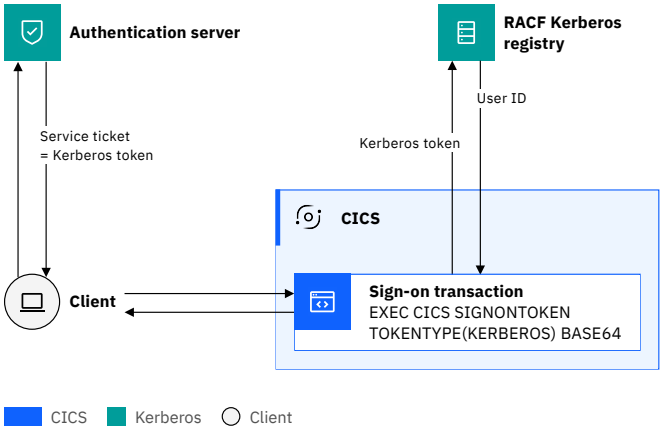


Figure 15. Flow of requests between the Client terminal emulator, the authentication server, and CICS TS

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 security 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

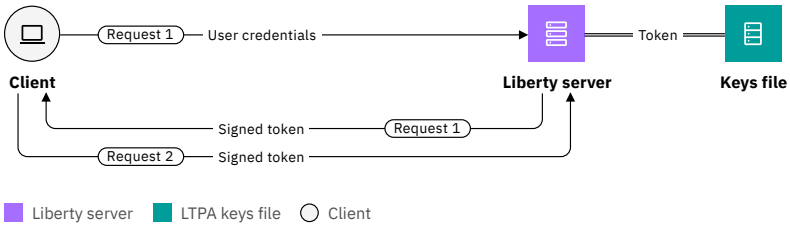


Figure 16. Creating and validating an LTPA token

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 `ltpaToken2`. 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 logon, 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.ouput.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.

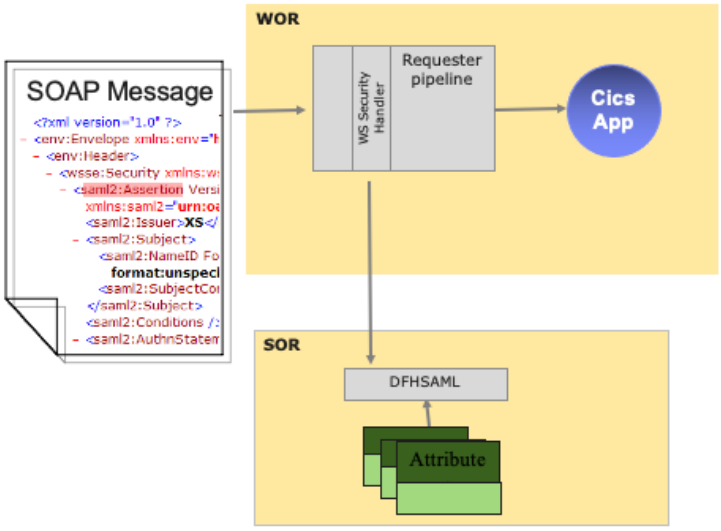


Figure 17. CICS support for SAML

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 39.

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 38.

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/htp/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/htp/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>
          <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/JVMSERVER/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 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.

- RACF is 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 Java 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 SAFCRED 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 server in CICS.

- LDAP

  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.

If you use CICS TS with Liberty, an extra option is available with the basic user registry. The basic user 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*). 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 achieve 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 18 on page 48 shows how transaction security works.

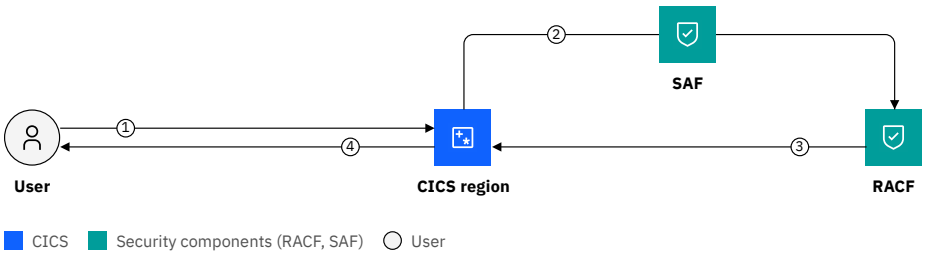


Figure 18. Transaction security

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 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.

.

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 19 on page 49 shows how resource security works, where the system administrator has defined the transaction AP01 to require resource security checking.

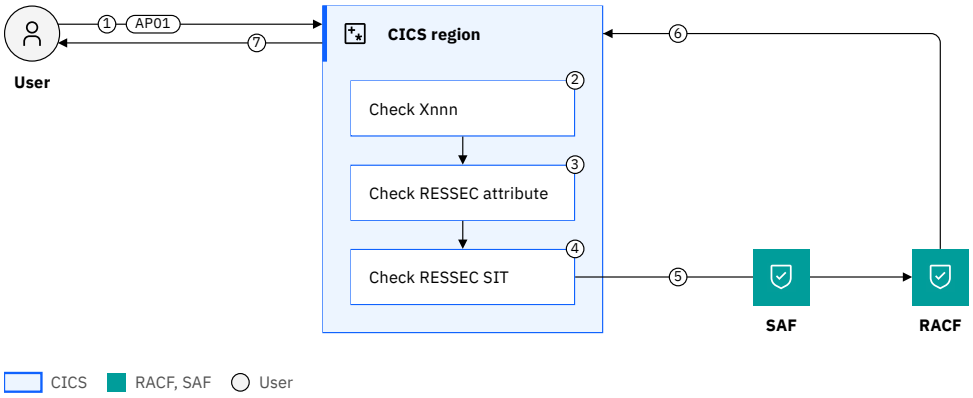


Figure 19. Resource security

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 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: WARNING should not be used in a production environment.

If you do not have a suitable test environment, you should also specify the NOTIFY option so that you can take action immediately.

You should 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.

Note: Specify the NOTIFY option, if you want to be notified at once when access is denied to a user.

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 20 on page 51 shows how command security works, where the system administrator has defined the transaction CEMT to require command security checking.

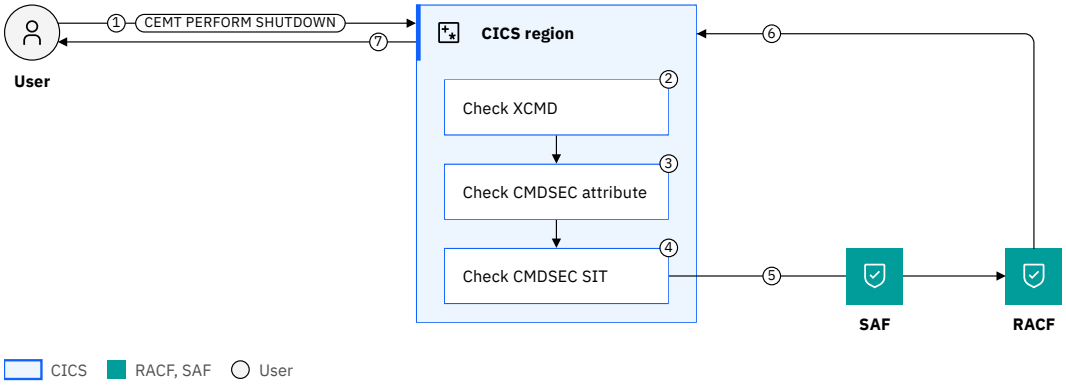


Figure 20. Command security

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 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 CMDSSEC 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.

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 now 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.

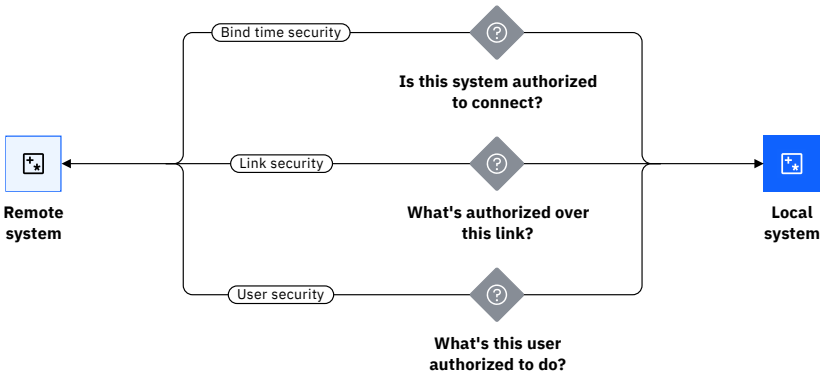
You are recommended to connect systems through IPIC instead of ISC. ISC is listed here only for legacy purposes and no further documentation about it is provided in this section.

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.

Table 9. How link security is enforced		
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.

Table 10. How remote user security is enforced		
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.

Table 11.		
	Link user ID	**No link user ID**
Authenticated or asserted user ID	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. For example, a user can access a resource if they issue the request at work but not if they make the request from home.	Use this where none of the security checks need to take account of where a request came from. Security is based only on the authenticated or asserted user ID.

Table 11. (continued)		
	Link user ID	**No link user ID**
No authenticated or asserted user ID	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. Security is based only on the link user ID.	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.

Role authorization

To restrict user access to applications that run in a CICS Liberty JVM server, you can use security roles to authorize access to application URIs or Java methods. Users or groups of users are associated with security roles. These roles are specified within the Java application, either in the code or within its deployment descriptor. Role authorization 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 3 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. Using the role to security 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.

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 21 on page 56 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.

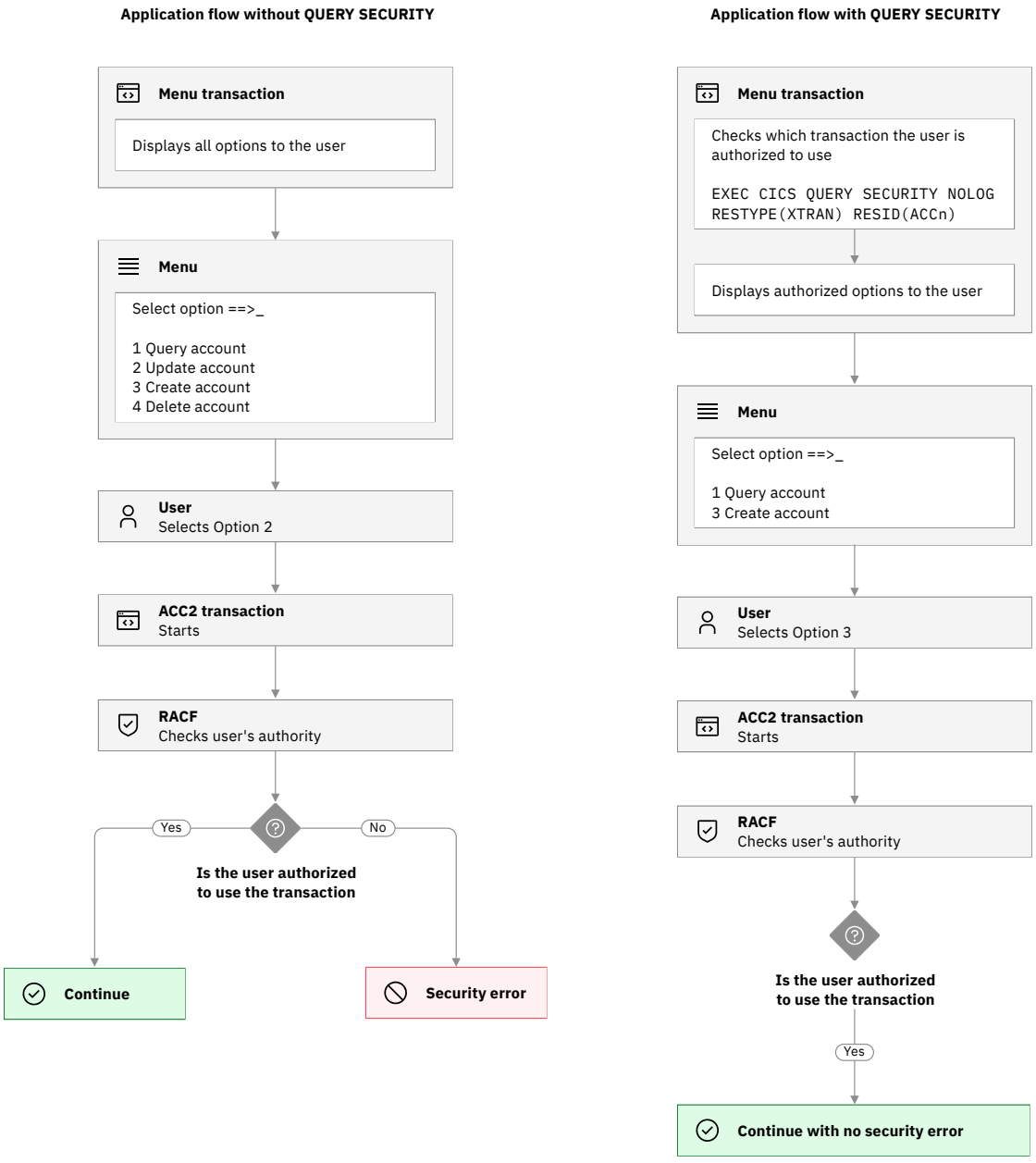


Figure 21. Application flows without and with QUERY SECURITY

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 58).

Figure 22 on page 58 shows how surrogate security works for starting transactions.

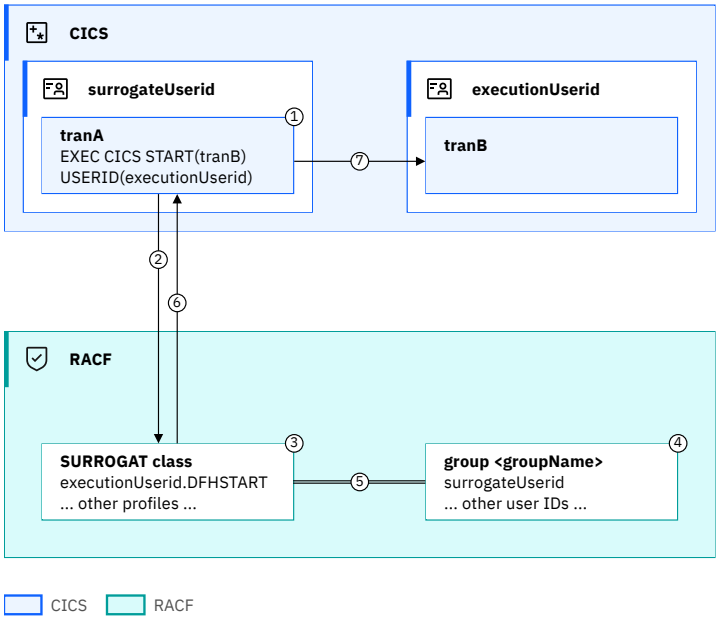


Figure 22. Surrogate security

1. Task that runs Transaction tranA under USERID of *surrogateUserid* issues an **EXEC CICS START** command to run transaction tranB under the user ID *executionUserid*.
2. CICS asks RACF if *surrogateUserid* has surrogate authority to start a transaction that runs under user ID *executionUserid*.
3. RACF finds a profile in the RACF SURROGAT class to match *executionUserid*.DFHSTART. See RACF classes for more information about the RACF resource classes.
4. RACF finds a group or groups permitted to use this profile and checks if the *surrogateUserid* is in this group.
5. RACF checks that the group has READ access 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 59 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**
execution-userID.DFHSTART	Task user ID	Start a transaction running under the *execution-userID*. This applies to the API commands START, DEFINE PROCESS, and DEFINE ACTIVITY.
execution-userID.DFHINSTL	At startup, the CICS region user ID, or the task user ID that issues a DEFINE, ALTER, or INSTALL command, or issue a SET command that changes the specific user ID.	Install or change a resource that has the ability to perform function under the *execution-userID*. See How it works: Identification in CICS.
default-userID.DFHINSTL	CICS region user ID	Use the CICS default user ID. See Defining the default CICS user ID to RACF.
plt-userID.DFHINSTL	CICS region user ID	Allows the PLTPIUSR user ID to run PLT processing. See Surrogate security.
execution-userID.DFHEXCI	The batch job's user ID	Run an EXCI call in CICS by using the *execution-userID*.
execution-userID.DFHQUERY	Task user ID	Issue the QUERY SECURITY command on behalf of the *execution-userID*.
execution-userID.SUBMIT	Task user ID	Submit JCL to run a job under the execution-userID. See Security for submitting a JCL job to the internal reader.

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. Figure 23 on page 62 and Figure 24 on page 64 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.

Flow of the TLS 1.2 handshake

Figure 23 on page 62 is a swimlane diagram that shows the flow of the TLS 1.2 handshake.

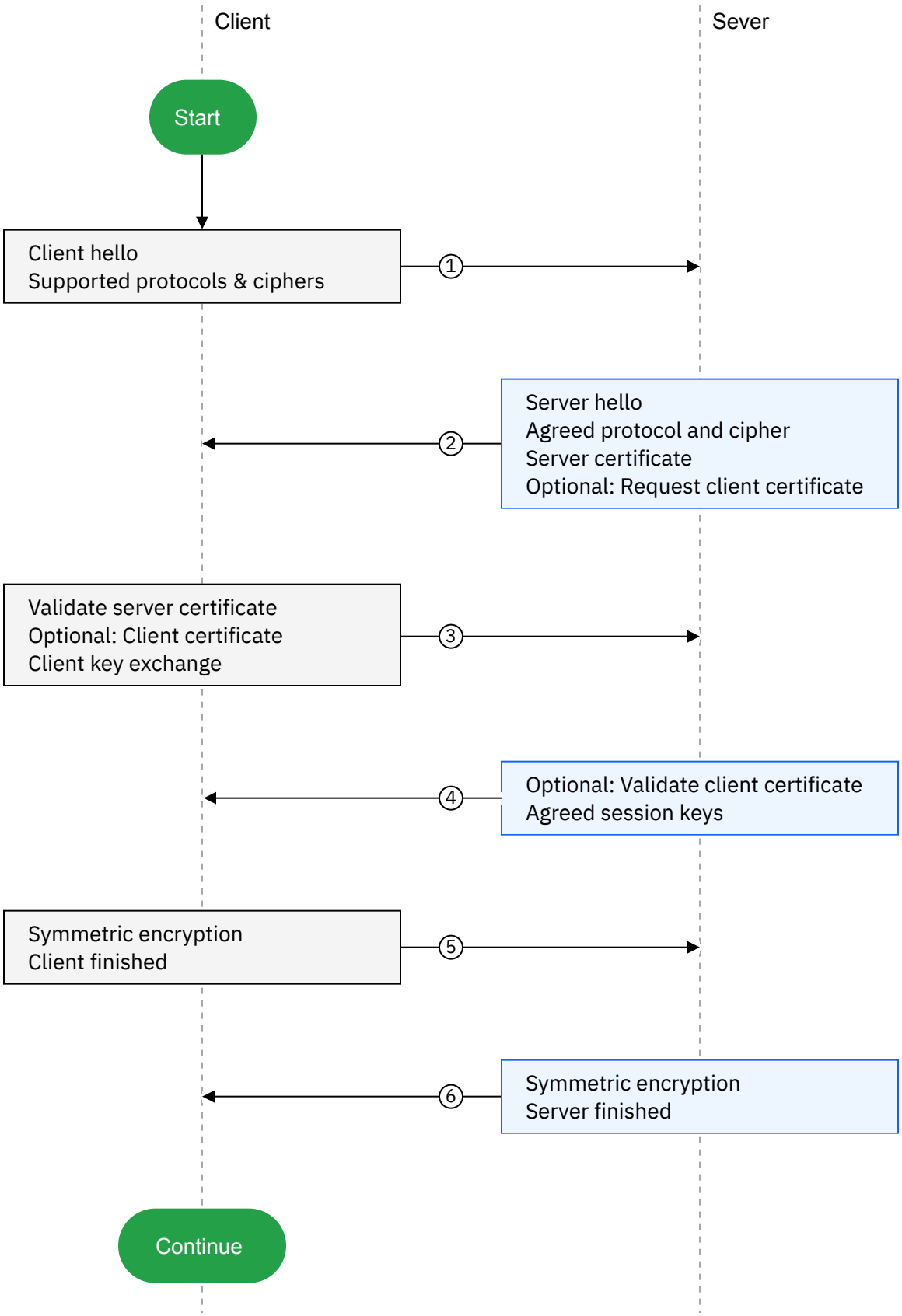


Figure 23. The TLS 1.2 handshake

An explanation of each step in Figure 23 on page 62 follows:

1. The client:

- Sends the highest version of TLS that it supports.
- Sends a list of the cipher suites that it supports.

2. The server:
   - Chooses the highest version of TLS that both the client and server support.
   - Chooses the *best cipher suite* that both the client and server support.
   - Sends agreed TLS version and cipher suite.
   - Sends a server certificate.

     Optionally, requests a client certificate.

     Important: Part of the handshake is the exchange of digital certificates. A digital certificate for the server is passed to the client. The client knows that the server can be trusted according to which certificate authority (CA) signed the server certificate. If the CA is trusted, the server certificate is trusted and the TLS session is set up. The server might use a certificate from the client to identify the client. Certificates are part of the authentication process rather than confidentiality so are covered in Certificates.
3. The client validates the server certificate, and optionally, sends the client certificate.
4. The server:
   - Validates the client certificate, if sent.
   - Sends a session setup message.
5. Start of secure communication of data.
6. Communication continues securely.

Flow of the TLS 1.3 handshake

Figure 24 on page 64 a swimlane diagram that shows the flow of the TLS 1.3 handshake.

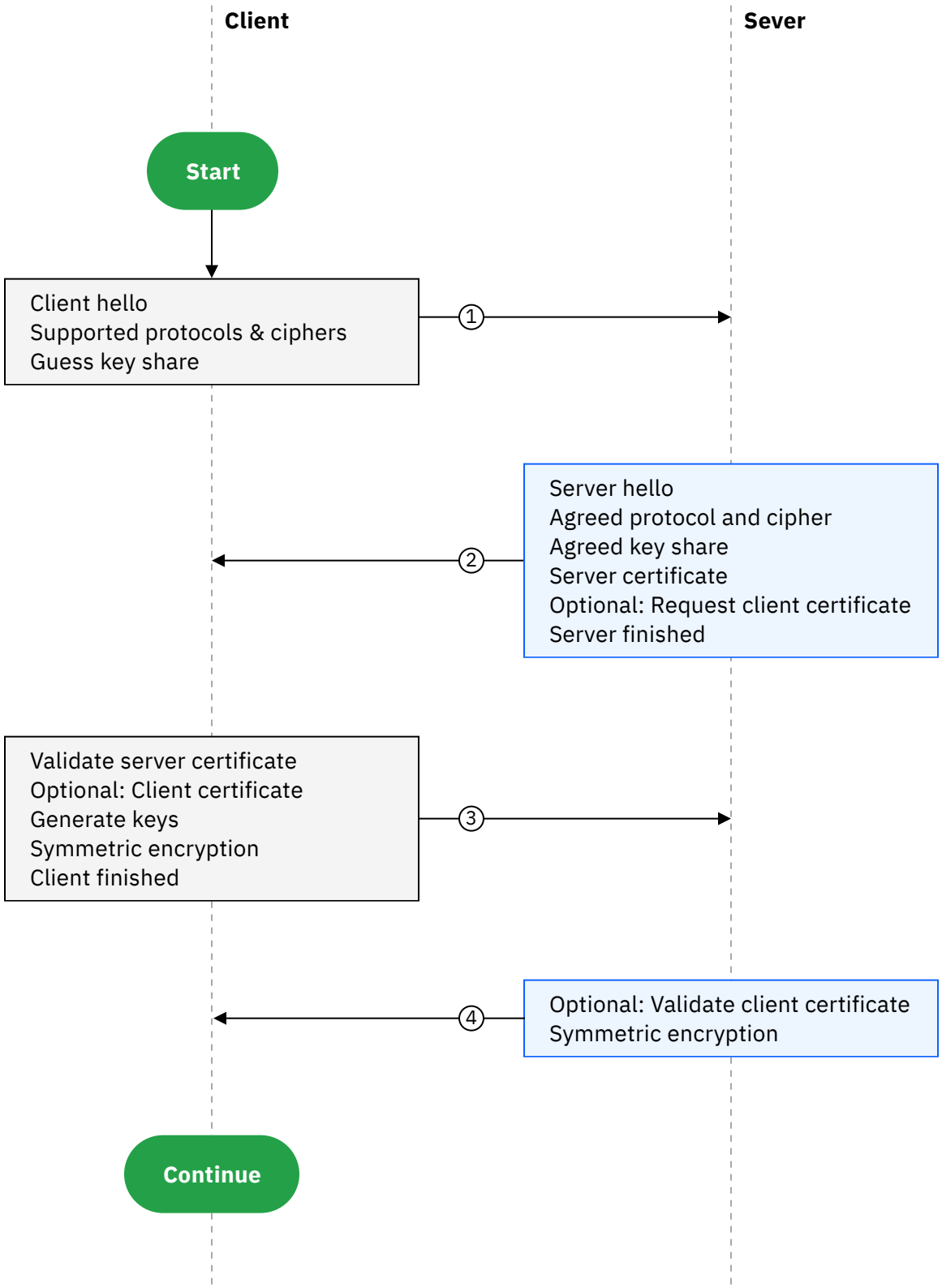


Figure 24. The TLS 1.3 handshake

An explanation of each step in Figure 24 on page 64 follows:

1. The client:
   - Sends the highest version of TLS that it supports.
   - Sends a list of the cipher suites that it supports.
   - Sends a list of key share groups it supports.
2. The server:
   - Chooses the highest version of TLS that both the client and server support.
   - Chooses the *best cipher suite* that both the client and server support.
   - Chooses the key share groups that both the client and the server support based on the client's preference settings.
   - Sends agreed TLS version and cipher suite.
   - Sends a server certificate.

     Optionally, requests a client certificate.

     Important: Part of the handshake is the exchange of digital certificates. A digital certificate for the server is passed to the client. The client knows that the server can be trusted according to which certificate authority (CA) signed the server certificate. If the CA is trusted, the server certificate is trusted and the TLS session is set up. The server might use a certificate from the client to identify the client. Certificates are part of the authentication process rather than confidentiality so are covered in Certificates.
3. The client:
   - Validates the server certificate.

     Optionally, sends the client certificate.
   - Start of secure communication of data.
4. The server:
   - Validates the client certificate, if sent.
   - Communication continues securely.

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 .

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 25 on page 66 shows an example for an inbound connection that uses a TCPIPSERVICE definition. Figure 26 on page 67 shows an example for an outbound connection that uses a URIMAP definition.

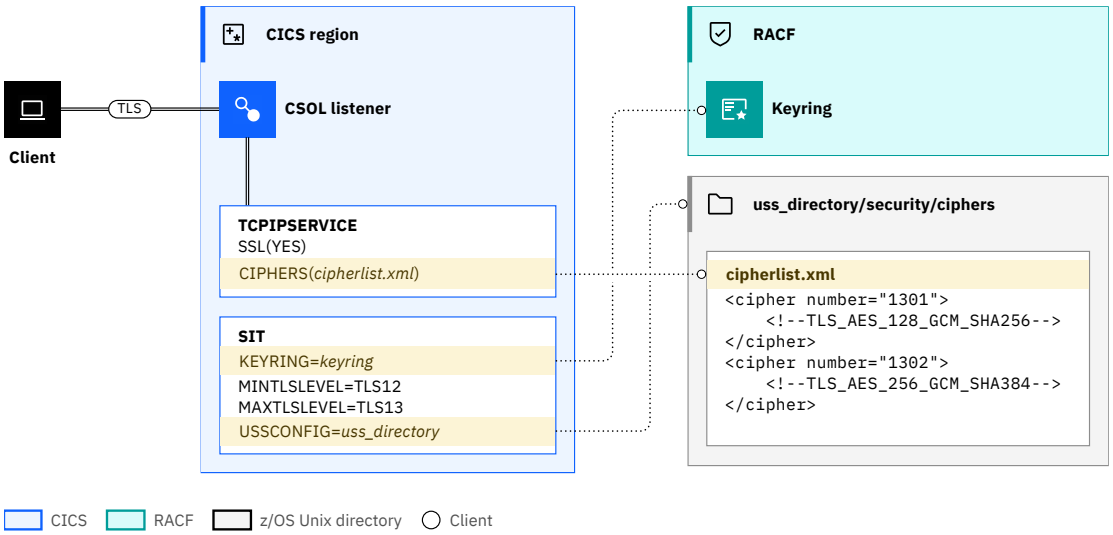


Figure 25. Inbound TLS support in CICS

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.

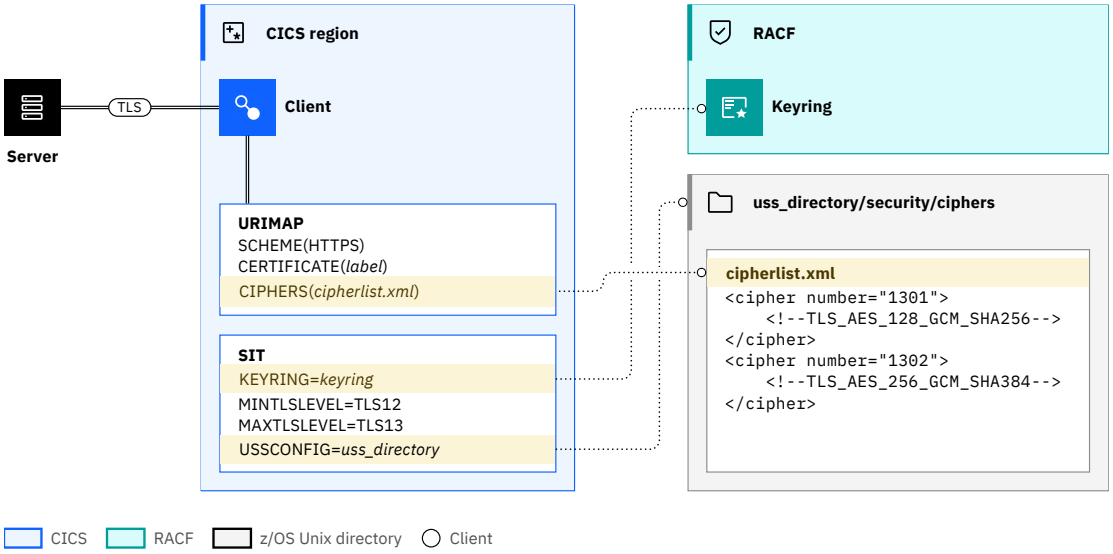


Figure 26. Outbound TLS support in CICS

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 27 on page 67.

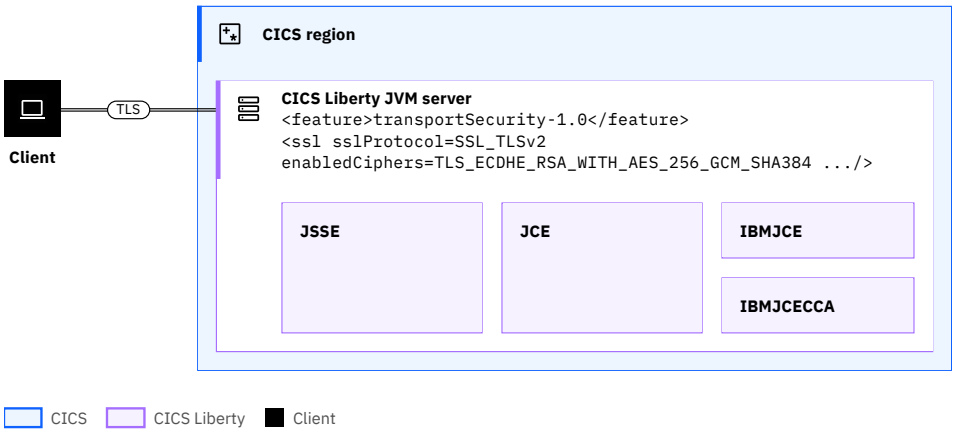


Figure 27. TLS in Liberty

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

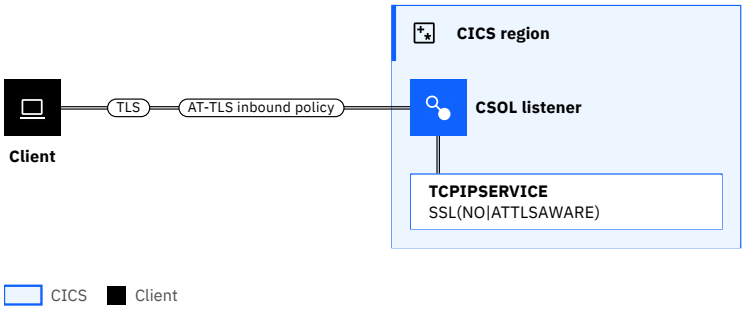


Figure 28. Inbound TLS support in CICS by using AT-TLS

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 TCPIPSERVICE 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(ATTLSAWARE) option on the TCPIPSERVICE 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 29 on page 69 shows an example outbound from CICS.

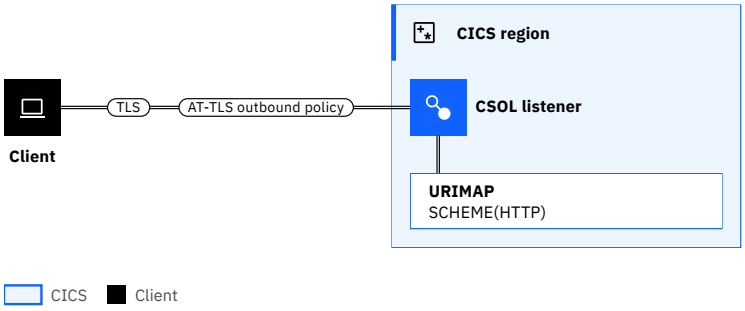


Figure 29. Outbound support for TLS in CICS by using AT-TLS

Figure 30 on page 69 shows an example in CICS Liberty.

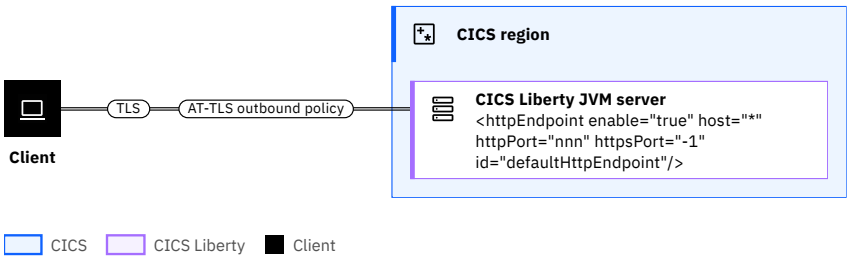


Figure 30. TLS in CICS Liberty by using AT-TLS

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 Encryption algorithms.

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.

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 31 on page 72 illustrates the flow of data in a 3270 session and the capabilities for integrity and confidentiality at each stage.

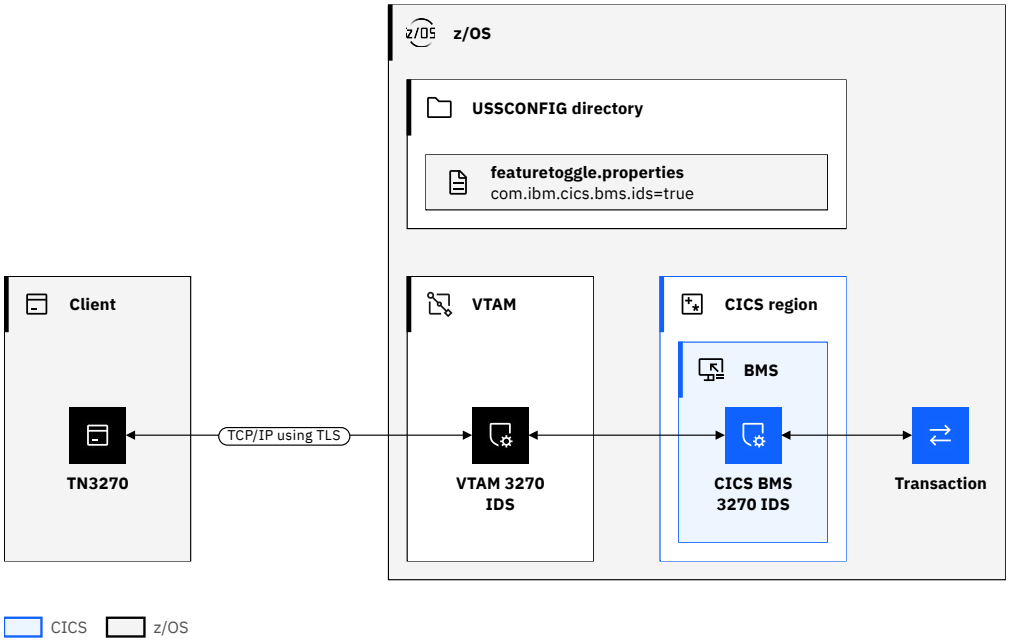


Figure 31. Integrity and confidentiality for 3270 sessions

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 if 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 DFHBMSX 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.

Table 13 on page 75 outlines 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.

Table 13. Finding audit information for CICS

What you want to audit	Where to find audit information	How to use the audit information
Configuration		
CICS configuration	IBM Health Checker for z/OS	Checking CICS configuration with IBM Health Checker for z/OS
Who last changed a CICS resource	• CICS Explorer, WUI, or CEDA displays • DFHCSDUP output	Identifying changes to resources (resource signatures)
Changes that are made to CICS resources	• Transient Data Queue CSDL • EYULOG of the CICSPlex SM maintenance point CMAS	Change (before or after)
Resource installation	• Transient Data Queue CSDL or resource-specific TDQs, • CICSPlex SM MAS MSGUSR	Auditing installed resources
Operation		
Changes that are made to CICS by operator commands	Transient Data Queue CADS	Auditing SPI commands
Authentication		

Table 13. Finding audit information for CICS (continued)		
What you want to audit	**Where to find audit information**	**How to use the audit information**
User sign-on and sign-off	• z/OS security console • SMF type 80 records • Transient Data Queue CSCS • EYULOG for the CICSPlex SM WUI • Liberty messages log	Auditing sign-on and sign-off
Failed authentication attempts	• z/OS security console • SMF type 80 records • CICS statistics • Liberty messages log	Auditing sign-on and sign-off
Connection attempts	• z/OS security console • SMF type 80 records	
Authorization		
Failed access attempts	• z/OS security console • SMF type 80 records • Transient Data Queue CSCS • CICSPlex SM WUI EYULOG	Auditing authorization
User changes to CICS resources	Log streams	
SAF requests	SMF type 80 records	
WARNING option used	RACF profiles	Listing the profiles in a class

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

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. See z/OS MVS System Management Facilities (SMF).

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: Securing CICS with RACF

z/OS Security Server (RACF) is IBM's 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.

RACF provides the following facilities:

- The necessary functions to record information that identifies individual *users* of system resources, and information that identifies the resources that require protection.
- The information that you define to RACF about users is stored in a *user profile*. Sets of users with the same role can be stored in *groups*.
- The information that you define to RACF about resources is stored in resource *profiles*. Sets of profiles that are usually for the same resource, are stored in *classes*.
- 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 *profile* alone. Defines a resource to be protected. For example, a transaction name.

Class
: Also called a class profile, is a set of resource profiles for similar resources. For example, transactions.

User
: Also called a user profile, defines characteristics about a user.

Group
: A set of users with the same role. For example, bank tellers.

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

When CICS tries to perform an operation that is subject to a security check, it calls RACF. RACF checks to see 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 results in ICH408I security failure messages.

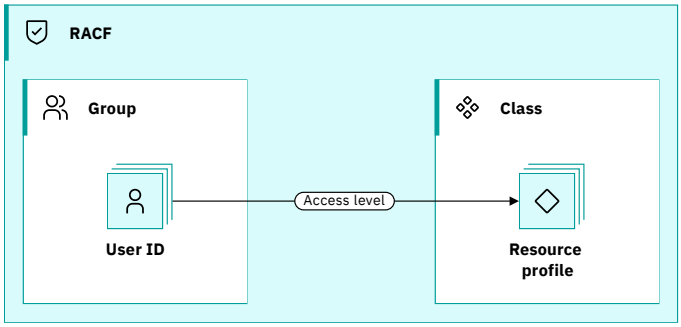


Figure 32. RACF security elements

When CICS tries to perform an operation that is subject to a security check, it calls RACF. RACF checks to see 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 results in ICH408I security failure messages.

More details about these terms and concepts are described in the various sections of the how it works documentation.

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.

Roles and 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 provide by the CICS region. User IDs have profiles which describe the user. User IDs will be associated with one or more roles that can be described in RACF group profiles.

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.

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 Giving access to control CICS fields in the user profile.

The contents of the segments relevant to CICS are described in the following sections:

The base segment in user profiles

You identify a RACF user by an alphanumeric user ID, which RACF associates with the user profile for that user.

For more information about the base segment in user profiles, see the z/OS documentation about The base segment in user profiles.

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).

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.

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.

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:

For more information about national language, see National language and non-terminal transactions.

LANGUAGE segment in z/OS Security Server RACF Security Administrator's Guide
The LANGUAGE segment in user profiles

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.

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 group profiles

A group profile 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. Groups 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 groups 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 roles.

In a CICS environment, group profiles offer a number of advantages:

- Easier control of access to resources, since you define only the group or groups that have access to the resource.
- Easier management of role-based security since membership of a group defines to what a user has 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 are signed onto 3270 terminals, it is necessary to sign off and sign back on again for this condition to take effect.

Recommendation: Access lists in profiles must be groups rather than individual user IDs. When someone leaves a department, or stops doing a role, removing the user ID from the group revokes all privileges. 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.

RACF classes and profiles

RACF classes and 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 these profiles.

RACF classes

A RACF class is a set of RACF profiles for protecting similar resources. For example, transactions.

Two types of class exist, *general resource classes* and *resource group classes*.

General resource classes (also known as member classes) can contain either discrete profiles or generic profiles. Generic profiles are defined by wild cards, such as AB* to represent resources with names that start with the letters AB.

Resource group classes (also known as grouping classes) include named sets of profiles. The named sets of resources would be associated with resources that would be associated with a user role. For example, the CICS supplied DFH$CAT2 CLIST has sets of associated transactions that are called SYSADM that would be used by users with a system administrator role. For a basic example of defining role access to classes in RACF, see How it works: Securing CICS with RACF.

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 name apart from the initial character that is used to identify the type of class.

If duplicate resources exist in the resource groups, these are merged according to the algorithm defined in Resolving conflicts among grouping profiles in the z/OS Security Server RACF Security Administrator's Guide.

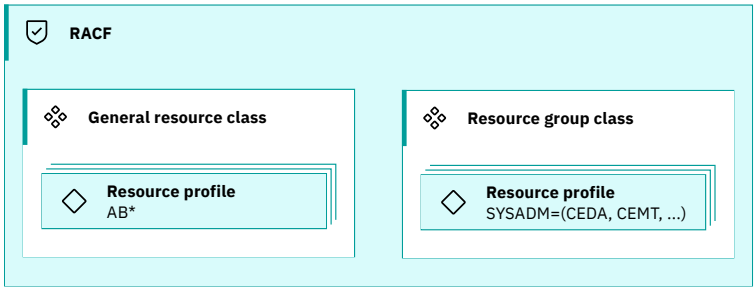


Figure 33. RACF classes

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.

General resource classes are sometime 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.

For more information, see Summary of RACF classes for CICS resources.

Reference

Creating resource group profiles
z/OS Security Server RACF Security Administrator's Guide

Caching of RACF classes and their profiles

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. This change is done by issuing the following command:

```
SETROPTS RACLIST (TCICSTRN) REFRESH
```

Where *class* is the generic resource class. This class name must be specified even if a profile is changed in the associated resource group class. The resource group class is not specifically RACLISTed.

Related information

Summary of RACF classes for CICS resources

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.

Use the DFH$RACF sample job as an example of how to define Db2 resource class names for CICS use.

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 an example of how to add 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 ICHRFR01, 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 IBM-supplied sample, DFH$RACF, which is in CICSTS61.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 and 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 X*name* 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 say 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 classname must be a national or numeric character.

RACF profiles for CICS classes

In RACF, a *profile* describes the security characteristics of a user, a group of users, or one or more computer resources. Profiles that are associated with the CICS classes are known in RACF as *general resource profiles*.

The information in general resource profiles includes the following information:

Profile name
A description of the resources protected.

Profile owner
The security administrator 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.

The profile name depends on the type of class that it relates to.

In general resource classes the profile is either discrete or generic.

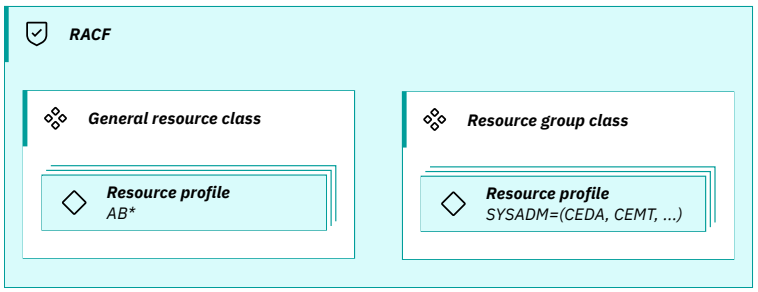
- A *discrete profile* protects a single resource, for example a single transaction.
- A *generic profile* protects several resources with similar names, for example all transactions that start with AB. Wildcard characters * and % are used to define these profiles. See Other rules for generic profile names in z/OS Security Server RACF Security Administrator's Guide for details.

In resource group classes, you associate a set of resources. This set is given a name associate with this set. For example, in the DFH$CAT2 CLIST the name SYSADM is given to the set of CICS transactions likely to be of use to a CICS system administrator.

Figure 34 on page 87 gives an example of these profiles.

For more information about RACF classes, see Summary of RACF classes for CICS resources.

Figure 34. RACF profiles



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.

If there are duplicate resource in the resource groups, these resources are merged according to the algorithm that is defined in Resolving conflicts among grouping profiles in z/OS Security Server RACF Security Administrator's Guide.

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.

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 group profiles” on page 83 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 89.

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 14. Types of RACF administration

Table	Example tasks
Defining authorities required by security administrators	Delegation of RACF administrative responsibility, controlling access to fields in RACF profiles
Managing users	Specifying default values in the CICS segment, creating or updating segment data for a CICS user, defining terminal users and user groups to RACF
Managing groups	Creating groups, creating subgroups, adding a user to a group, removing a user from a group
Managing profiles and classes	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`	LISTUSER
Delegate authority to administer a class.	`ALTUSER userid CLAUTH(class)`	ALTUSER
Delegate authority to administer users.	`ALTUSER userid CLAUTH(USER)`	ALTUSER
Delegate authority to administer a group.	`CONNECT userid GROUP(group) SPECIAL`	CONNECT
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)`	RDEFINE PERMIT Field-level access checking
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)`	RDEFINE PERMIT Field-level access checking
Installing a resource with user ID attributes.	`DEFINE SURROGAT *.DFHINSTAL UACC(NONE)` `PERMIT *.DFHINSTAL CLASS(SURROGAT) GROUP(group) ACCESS(UPDATE)`	RDEFINE PERMIT Surrogate Security

Managing users

Task	RACF command	Reference
Adding a CICS User with the default CICS options.	`ADDUSER userid DFLTGRP(group) CICS`	ADDUSER The CICS segment in user profiles

Task	RACF command	Reference
Changing CICS options for a user.	`ALTUSER userid CICS(field1 field2 .. fieldN)`	ALTUSER The CICS segment in user profiles
Removing CICS options for a user.	`ALTUSER userid NOCICS`	ALTUSER
Listing the CICS options for a user.	`LISTUSER userid CICS`	LISTUSER
Specifying the language in the LANGUAGE segment.	`ALTUSER userid LANGUAGE(PRIMARY(language_code) SECONDARY(language_code))`	ALTUSER The LANGUAGE segment in user profiles
Listing the LANGUAGE segment for a user.	`LISTUSER userid LANGUAGE`	LISTUSER

Managing groups

Task	RACF command	Reference
Creating groups .	`ADDGROUP group OWNER(owner)`	ADDGROUP
Creating subgroups.	`ADDGROUP subgroup OWNER(subowner) SUPGROUP(group)`	ADDGROUP
Adding a user to a group.	`CONNECT userid GROUP(group)`	CONNECT
Removing a user from a group.	`REMOVE userid GROUP(group)`	REMOVE

Managing profiles and classes

Task	RACF command	Reference
Listing the profiles in a class.	`RLIST class * ALL`	RLIST
Defining a resource by using a generic profile.	`REDEFINE class profile UACC(NONE)`	RDEFINE RACF profiles for CICS classes
Defining a resource by using a member profile.	`REDEFINE class profile ADDMEM(memberList) UACC(NONE)`	RDEFINE RACF profiles for CICS classes
Giving access to a group of users to a profile.	`PERMIT profile CLASS(class) ID(group) ACCESS(access)`	PERMIT
Defining resource definitions when you use SECPRFX.	`REDEFINE class secprfx.profile UACC(NONE)`	RDEFINE Prefixing RACF Profiles with SECPRFX
Refreshing resource profiles in main storage.	`SETROPTS RACLIST(class) REFRESH`	SETROPTS Caching of RACF classes and their profiles
Activating the CICS classes.	`SETROPTS CLASSACT(class)`	SETROPTS Caching of RACF classes and their profiles
Making an installation defined class support generic resources.	`SETROPTS GENERIC(class)`	SETROPTS

When you have a pair of classes, class is the generic 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 9. 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.

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 TCPIPSERVICE, 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 TCPIPSERVICE 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 TCPIPSERVICE are queried to extract their AT-TLS attributes. When a client establishes a new connection to a CICS TCPIPSERVICE 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)** TCPIPSERVICE are queried to extract their AT-TLS attributes. When a client establishes a new connection to a CICS TCPIPSERVICE 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 DFHSOCK.

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 (TCPIPSERVICE, 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 35 on page 97 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 36 on page 97 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 (TCPIPSERVE, 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 35 on page 97 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 36 on page 97 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  Last Reset 16:10:34
Applid IYK4ZON1  Jobname D1
--------------------------------------------------------------------------------------------------------------
--------------------------------
TLS CIPHER STATISTICS
---------------------
                                                                      TLS       TLS    AT-TLS
AT-TLS
  Number  Cipher Name                                               Inbound  Outbound   Inbound
Outbound

--------------------------------------------------------------------------------------------------------------
--
    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 35. 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  9.20.5.0
                                   +X0014   40404040 40404040 40404040 40404040 40404040
         DFHCICS  A369     OCLIPORT         0000FA66                                       641020
         DFHCICS  A370     OTRNFLAG         8000804009800000
         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
            ...
         DFHSOCK  A458     SOFLAG           40000000
         DFHSOCK  C457     SOTLSLVL         E3D3E2E5 F14BF240                             TLSV1.2
         DFHSOCK  A320     SOCIPHER         00000039 F14BF240                                57
         DFHTASK  C430     CECMCHTP         F3F9F0F6                                      3906
```

Figure 36. Monitoring data (formatted output)

Providing support to update to TLS 1.3

TLS 1.3 differs from prior versions of the protocol. 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.

Before you begin

Recommended: Your migration to TLS 1.3 must be separate from your upgrade to CICS TS 6.1.

Hardware and Software Prereqs required for upgrading to TLS 1.3:

- TLS 1.3 requires a minimum z/OS level of z/OS 2.4.

About this task

TLS 1.3 is a step change to the TLS protocol. Because of this change, you need to complete the following steps to implement TLS 1.3 in CICS TS.

A number of factors impact this task:

- No ciphers in common between TLS 1.3 and earlier versions of the TLS protocol.
- The ciphers are 4-digit ciphers that can be defined only in CICS by using XML files.
- Software dependencies higher than the minimum level for CICS TS 6.1.
- Several hardware and software dependencies that 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.

To move from TLS 1.2 to TLS 1.3, you need to:

1. Complete the upgrade to CICS Transaction Server 6.1.
2. Prepare RDO definitions that include configuring cipher file definitions and updating your WEB OPEN commands.
3. Upgrade your certificates.
4. Enable TLS 1.3.
5. Disable TLS 1.2.

The SIT parameter **MAXTLSLEVEL** defaults to TLS12. This value enables an upgrade to CICS TS 6.1 to take place without affecting any existing TLS connections or programs that open TLS connections.

Learn more about the RFC 8446 The Transport Layer Security (TLS) Protocol Version 1.3.

Procedure

1. Upgrade to CICS Transaction Server 6.1.

   Ensure your CICS upgrade to version 6.1 is complete. This step covers the background information that you need to ensure is configured before you prepare RDO definitions.

   The SIT parameter **USSCONFIG** must specify a directory in which you create definitions. This directory must include a security subdirectory. The security subdirectory must include a ciphers subdirectory. XML cipher definitions exist in the ciphers subdirectory. **USSCONFIG** cannot be set to the same value as **USSHOME**.

   You must 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.

   This file is used as the default value for new definitions that require the **CIPHERS** options; in previous releases of CICS this parameter defaults to two-digit ciphers.
2. Preparing RDO definitions before you upgrade to TLS 1.3.

To use TLS 1.3, you must specify at least one of the following new TLS 1.3 ciphers in the CIPHERS option on all IPCONN, TCPIPSERVICE, and URIMAP resource definitions, regardless of whether TLS 1.3 is used for the connection. Because the TLS 1.3 ciphers can be defined only in an XML file, it is necessary to update any definitions that use ciphers to use an XML file in the CIPHERS option.

```
<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>
```

You can blank out the ciphers in a resource definition by using CEDA. CEDA replaces these ciphers with `defaultciphers.xml`.

If you require specific ciphers for a selected IPCONN, TCPIPSERVICE, and URIMAP resource definition, you need to create an XML file in the ciphers subdirectory and change the CIPHERS option of the selected definition to that name.

Ensure that all of the XML cipher files that you are using, including `defaultciphers.xml` have at least one of the TLS 1.3 ciphers, in addition to your existing TLS 1.2 ciphers.

If you use applications with the WEB OPEN commands, you might still have some CIPHERS defined and see a message DFHWB0767 to indicate that the CIPHERS option is deprecated. The CIPHERS option is ignored for TLS 1.3. The cipher that you use for the WEB OPEN connection is either obtained from the URIMAP defined in the URIMAP option on the WEB OPEN command, or if that option is not defined, from `defaultciphers.xml`. Ensure that the XML cipher file used contains ciphers appropriate for this connection.

3. Upgrade your certificates.

   TLS 1.3 requires certificates to have a minimum key size of 2048. Check that any certificates in your keyrings are at least SIZE(2048). If they are smaller, they need to be reissued.

   The RACDCERT command can be used to list keyrings and certificates. For more information about RACDCERT, see Using the RACDCERT command.

4. Enable TLS 1.3.

   Set the SIT option **MAXTLSLEVEL**=TLS13.

   Check that each of your connections is established. The connection uses the highest level of TLS supported by both sides. If any of the connections do not use TLS 1.3, you need to reconfigure or upgrade that client or server.

5. Disable TLS 1.2.

   When all connections have been checked to use TLS 1.3, you can disable lower versions of TLS by setting **MINTLSLEVEL**=TLS13.

   After these changes are complete for all of your regions, change all XML cipher files to remove non-TLS 1.3 ciphers.

Security for TCP/IP clients

This part 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 Encryption techniques

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.

The SSL cache

The SSL cache is used to store session IDs for SSL sessions between clients and CICS. Reusing these session IDs allows CICS to perform partial handshakes with clients that it has previously authenticated. The SSL cache can be local to a CICS regions or shared between CICS regions on a sysplex. This is configured by the system initialization parameter **SSLCACHE**. For optimal performance, it is important that you select the correct option.

Local caching, SSLCACHE=CICS

In a local CICS region, by default, the SSL cache is stored in the enclave for the S8 TCBs. It is managed by z/OS System SSL as part of the SSL environment, which exists within this enclave.

When you issue the **PERFORM SSL REBUILD** command for the CICS region, a new cache is created. The new cache is populated by new SSL sessions that are established in the CICS region. The old cache is removed when the last connection using it is dropped.

If you use the **SSLCACHE=CICS** option and use port sharing to enable HTTP connection requests to the same host and port to resolve to different CICS regions, a full SSL handshake is required every time a connection request from a client resolves to a different region, and the benefits of the caching are lost.

Sysplex caching, SSLCACHE=SYSPLEX

Sharing SSL session IDs across different CICS regions on a sysplex is particularly useful when HTTP requests are being routed across a set of CICS regions by using TCP/IP connection workload balancing techniques, such as TCP/IP port sharing or Sysplex Distributor. You should use sysplex caching if you have multiple CICS socket-owning regions that accept SSL connections at the same IP address. If appropriate for your CICS systems, using sysplex caching can significantly reduce the number of full SSL handshakes.

To enable sysplex caching, activate the z/OS System SSL started task GSKSRVR and specify the system initialization parameter **SSLCACHE=SYSPLEX** for the CICS regions. For details of the SSL started task GSKSRVR and its configuration, see The SSL started task GSKSRVR.

To use the sysplex session cache, each system in the sysplex must be using the same external security manager, and a user ID on one system in the sysplex must represent the same user on all other systems in the sysplex.

The **PERFORM SSL REBUILD** command does not affect the sysplex cache.

The SSL pool

CICS uses the open transaction environment (OTE) to manage SSL connections and requests to LDAP using the DFHDDAPX XPI interface.

To improve the number and performance of SSL connections in CICS, each SSL connection uses an S8 TCB from the SSL pool. All CICS processing for an SSL connection occurs on the S8 TCB. The web server HTTP attach task (CWXN by default) will stay on the S8 TCB until all the data have been sent or received. If the messages being sent or received are very large, the task could hit the runaway limit and be terminated. To avoid the task being abended, you might need to increase the transaction **RUNAWAY** value for the web server HTTP attach transaction (CWXN by default), the web server alias transactions, and any transactions that issue the web client API commands SEND, RECEIVE, and CONVERSE.

The S8 TCBs are contained in an SSL pool, which is managed by the CICS dispatcher. The S8 TCBS are allocated from the new SSL pool, but are only locked to a transaction for the period that it needs to perform SSL or LDAP functions. After the SSL or LDAP request is complete, the TCB is released back into the SSL pool to be reused. The **MAXSSLTCBS** system initialization parameter specifies the maximum number of S8 open TCBs in the SSL pool. The default value is 8, but you can specify up to 1024.

You can monitor the performance of the SSL pool and the S8 TCBs using the dispatcher reports from DFH0STAT and DFHSTUP. The statistics include information on how often the maximum number of S8 TCBs are reached, the delay before a TCB is allocated and the actual number of TCBs in the SSL pool.

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 TCPIPSERVICE 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. Your user ID must have CONTROL access to create the signing certificate the first time your run the program. If you run the program again, you require only UPDATE access.

About this task

DFH$RING is in library CICSTS61.CICS.SDFHSAMP. Edit the values in DFH$RING to create a suitable key ring and certificates:

Procedure

1. Enter values for the *firstname*, *lastname*, and *hostname* variables.

   The *firstname* and *lastname* values are concatenated together to form the name of the key ring. Enter the host name of your Web server for the *hostname* variable.
2. Optional: Enter a value for the *FORUSER* variable if you are building a key ring for a different user ID, such as a CICS region user ID.
3. If you have a signing certificate (certificate authority certificate), enter the label in the *certifier* variable.
4. 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.

5. Edit the variables for the **RACDCERT GENCERT** RACF commands to create appropriate certificates to add to your key ring.

   DFH$RING has four examples that you can edit, add to, or remove. Ensure that the *certifier* variable on the **SIGNWITH** parameter matches the label of your signing certificate.
6. Edit the labels for the **RACDCERT CONNECT** RACF commands to match your certificates. Ensure that the signing certificate is added to the key ring first, because it signs all the other certificates.
7. Run DFH$RING to create the key ring and certificates as follows:

```
EXEC 'CICSTS61.CICS.SDFHSAMP(DFH$RING)' 'firstname lastname webservername  [ FORUSER(userid) ] '
```

where *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 certificates with the following labels:

***lastname*-Web-Server**
: This certificate can be used in the CERTIFICATE attribute of TCPIPSERVICEs with PROTOCOL(HTTP). The distinguished name within the certificate has a common name of *webservername*, which must be the same as the host name associated with the connection. Web browsers usually check that the common name in the certificate matches the host name of the server from which it is received.

***lastname*-IP-CONNECTION**
: This certificate can be used for IP interconnectivity (IPIC). It can be used in CERTIFICATE attributes of resource definitions that are required for a CICS region to use IPIC. This sample certificate is for a CICS region to use as a client certificate and as a server certificate during an SSL handshake that occurs when an IPCONN is acquired. It can be used in the CERTIFICATE attribute of an IPCONN definition for a client certificate and the CERTIFICATE attribute of a TCPIPSERVICE definition with PROTOCOL(IPIC) for a server certificate.

***lastname*-2048-Certificate**
: This certificate can be used for CICS systems that require high-strength certificates. It can be used in CERTIFICATE attributes of TCPIPSERVICE, IPCONN, and URIMAP definitions, and EXEC CICS WEB OPEN commands.

***lastname*-Default-Certificate**
: This certificate is marked as the default certificate for the key ring and is the one that is used for all TCPIPSERVICE resources that do not specify a CERTIFICATE attribute. This certificate also contains a common name of *webservername*.

Verisign Class 1 Primary CA

Verisign Class 2 Primary CA

IBM World Registry CA

These certificates are required to validate client certificates that you might receive that have been signed by these Certificate Authorities. If you intend to accept client certificates signed by other Certificate Authorities, or certificates that you have created yourself, you will have to add their certificates to the key ring manually, using the **RACDCERT CONNECT** command. When you add a certificate to the key ring in this way, you must specify USAGE(PERSONAL).

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 TCPIPSERVICE 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 TCPIPSERVICE definition. Users connecting to CICS through such a TCPIPSERVICE 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 RDATALIB 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 RDATALIB class.
3. Activate the RDATALIB 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 104 or "Building a key ring with certificates using DFH$RING" on page 105.

The key ring must be completely configured, that is, it must contain not only the certificate pointed to by your TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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.

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 116.

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 in CICSTS61.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 TCPIPSERVICE** 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 TCPIPSERVICE, 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 stricture and where it is used by CICS and z/OS, see Customizing encryption negotiations.

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

“Providing support to update to TLS 1.3” on page 98

TLS 1.3 differs from prior versions of the protocol. 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.

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).

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 TCPIPSERVICE, 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 TCPIPSERVICE, 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 **TCPIPSSLCIPHERS** 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 **TCPIPSSLCERT** 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 TCPIPSERVICE defined with SSL(ATTLSAWARE) is opened, CICS issues message DFHSO0148 to warn you that the AT-TLS policy used to secure this TCPIPSERVICE 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 117.

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 116 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 118 .

- using a command. See “Running CCRL from a START command” on page 118.

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 117 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/ATTClass1Individual.crl http://crl.verisign.com/ATTClass2Individual.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 116

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 118.

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 117 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/ATTClass1Individual.crl
        http://crl.verisign.com/ATTClass2Individual.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 Communications Server. If configured, IBM Communication 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 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 10. 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 37 on page 121 defines the terms local CICS region and remote CICS region.

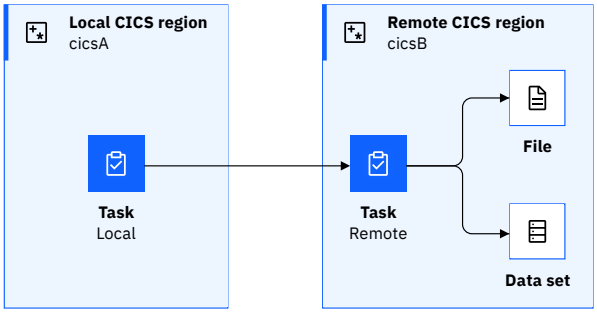


Figure 37. Definition of local and remote CICS regions

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.

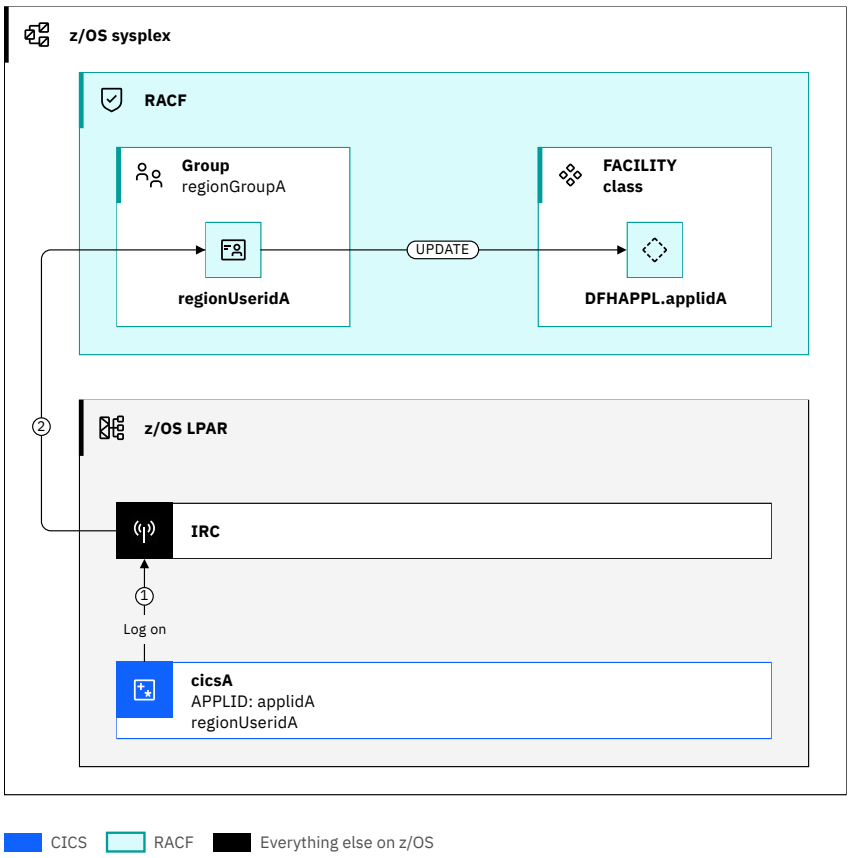


Figure 38. CICS region cicsA connects to IRC

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 39 on page 123 shows the flow of a connect request from the local region *cicsA* to the remote CICS region *cicsB*.

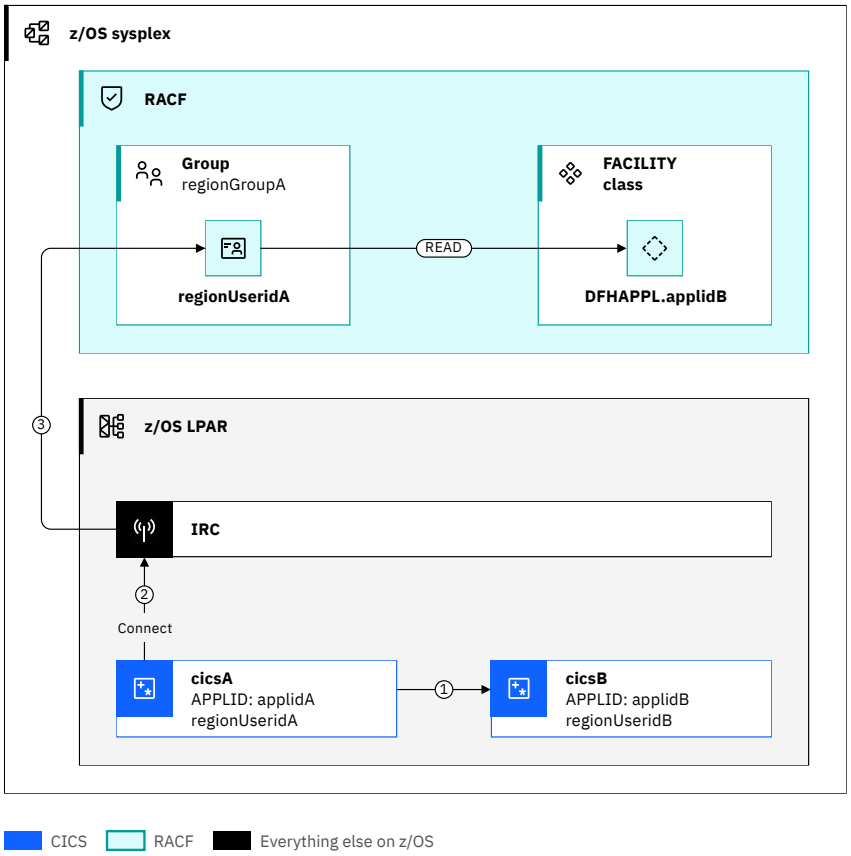


Figure 39. CICS region cicsA connects to CICS region cicsB

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 40 on page 124 shows the flow of a connect request from the remote CICS region *cicsB* to the local CICS region *cicsA*.

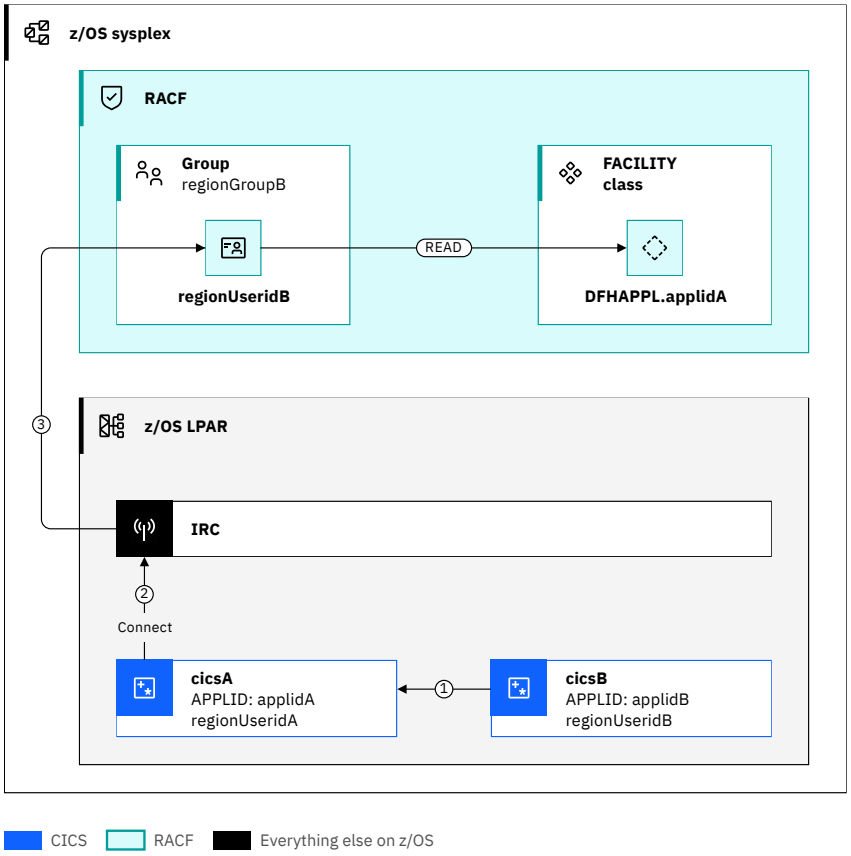


Figure 40. CICS region cicsB connects to CICS region cicsA

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
- 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.

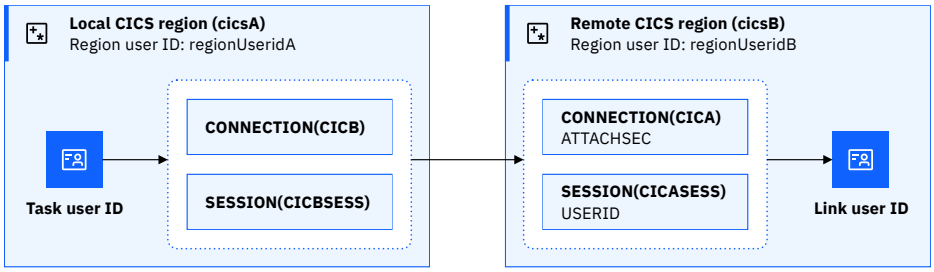


Figure 41. Resource definition attributes used for link security

How these values can be combined is shown in Table 15 on page 125.

Where a user ID is specified by the USERID value on 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*.

Table 15. How the link user ID is determined

Local CICS region cicsA	**Remote CICS region cicsB Default User: *defaultUseridB***				**Notes or examples**
Region user ID	**Region user ID**	**Resource definition**		**Link user ID**	
		ATTACHSEC attribute on CONNECTION	**USERID attribute on SESSION**		
regionUseridA	Same as *regionUseridA*	LOCAL		*defaultUseridB*	1 Not recommended.
regionUseridA	*regionUseridB*	LOCAL	Same as *regionUseridA*	*defaultUseridB*	1 Not recommended.
regionUseridA	*regionUseridB*	LOCAL		*regionUseridB*	2 Not recommended.
regionUseridA	*regionUseridB*	LOCAL	*functionalUseridB*	*functionalUseridB*	Design example 2
regionUseridA	Same as *regionUseridA*	IDENTIFY		No link user ID	
regionUseridA	*regionUseridB*	IDENTIFY	Same as *regionUseridA*	No link user ID	Design example 1
regionUseridA	*regionUseridB*	IDENTIFY		*regionUseridA*	2 Not recommended.

Table 15. How the link user ID is determined (continued)					
Local CICS region cicsA	**Remote CICS region cicsB Default User: *defaultUseridB***				**Notes or examples**
regionUseridA	*regionUseridB*	IDENTIFY	*functionalUseridB*	*functionalUseridB*	Design example 3

Recommended:

1 The default user ID must not be used as a link 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)`.

Figure 42 on page 126 shows the resource definitions in use for MRO user security.

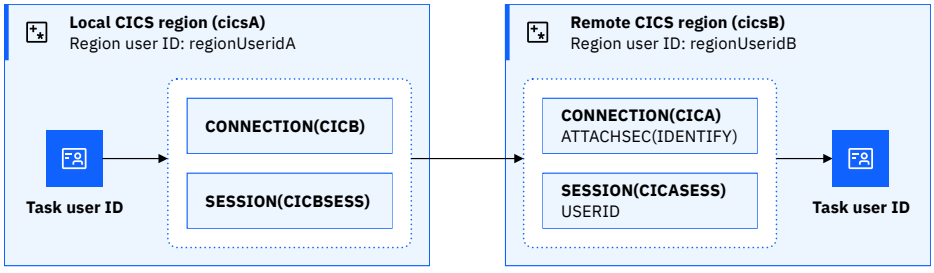


Figure 42. Resource definition attributes used for user security

User security authorization checks are made in addition to any link user ID checks.

The task user ID is flowed across the connection if IDENTIFY is specified. The task user ID is not flowed if LOCAL is specified. In this case, the task user ID in the remote region is set to the link user ID. This information is summarized in the following table.

Table 16. 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** [1]		
regionUseridA	*TaskUseridA*	Same as *regionUseridA*	IDENTIFY		*TaskUseridA*	
regionUseridA	*TaskUseridA*	*regionUseridB*	IDENTIFY		*TaskUseridA*	
regionUseridA	*TaskUseridA*	*regionUseridB*	IDENTIFY	*regionUseridA*	*TaskUseridA*	Design example 1
regionUseridA	*TaskUseridA*	*regionUseridB*	IDENTIFY	*functionalUseridB*	*TaskUseridA*	Design example 3
regionUseridA	*TaskUseridA*	*regionUseridB*	LOCAL		*Link user ID*	Not recommended, see MRO link security.

[1] The USERID specified on the CONNECTION definition's SESSION attribute is used for MRO link security.

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 127
- “Authorization” on page 128
- “Confidentiality and integrity” on page 128
- “Trust” on page 128
- “Audit” on page 128

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.

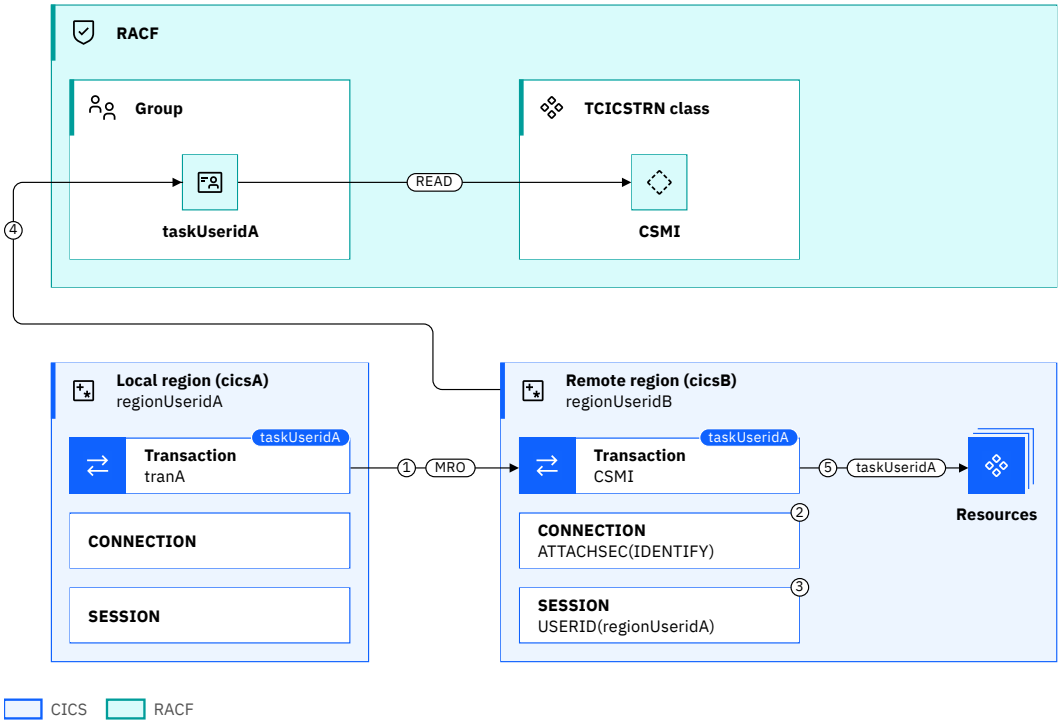


Figure 43. MRO connections - using only the task user ID

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. You don't need a *link user ID* because the **SESSION** definition in *cicsB* is defined with a `USERID` value, which is the same as *regionUseridA*.
4. *cicsB* checks that the *taskUseridA* is authorized to attach the transaction by calling RACF.
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. For more information, see Allowing the regions to use MRO .

- Allow each region to connect to the other region. For more information, see Allowing regions to connect to each other using 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 UACC(NONE) ADDMEM(CSMI)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup)
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.

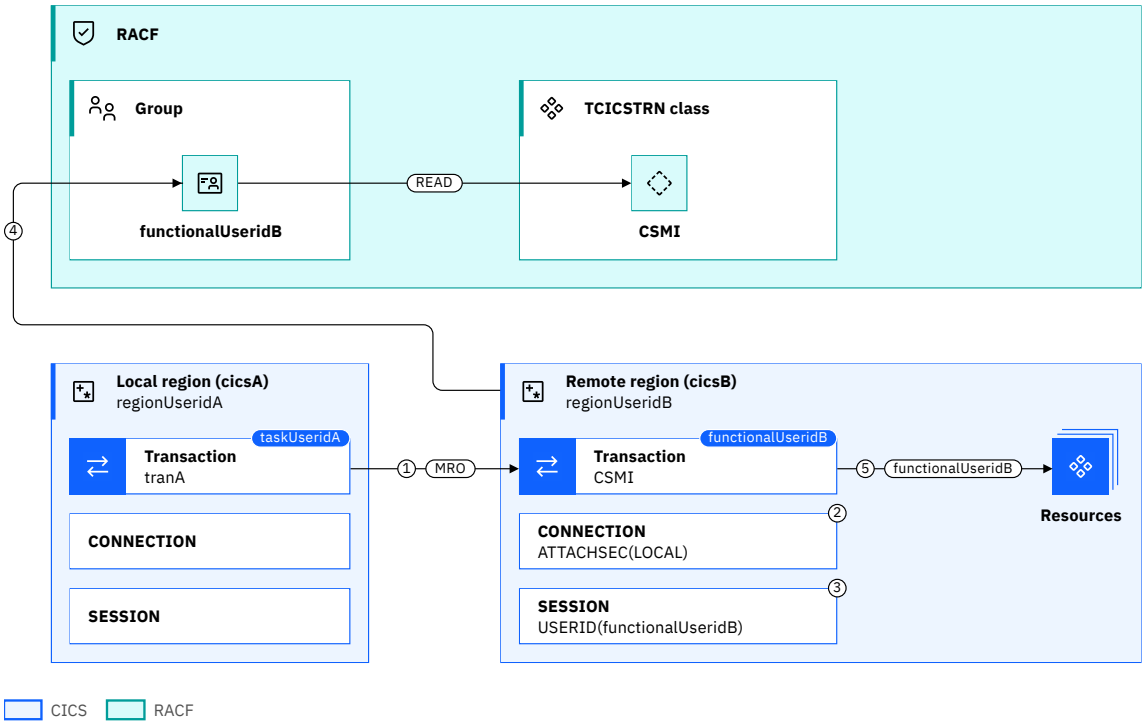


Figure 44. MRO connection – using only the link user ID

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 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.

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. For more information, see Allowing the regions to use MRO .
- Allow each region to connect to the other region. For more information, see Allowing regions to connect to each other using 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*.
- *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(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 TCICSTRN INTERCOM UACC(NONE) ADDMEM(CSMI)
PERMIT INTERCOM CLASS(TCICSTRN) ID(functionalUseridB) 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(installUserid) 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.

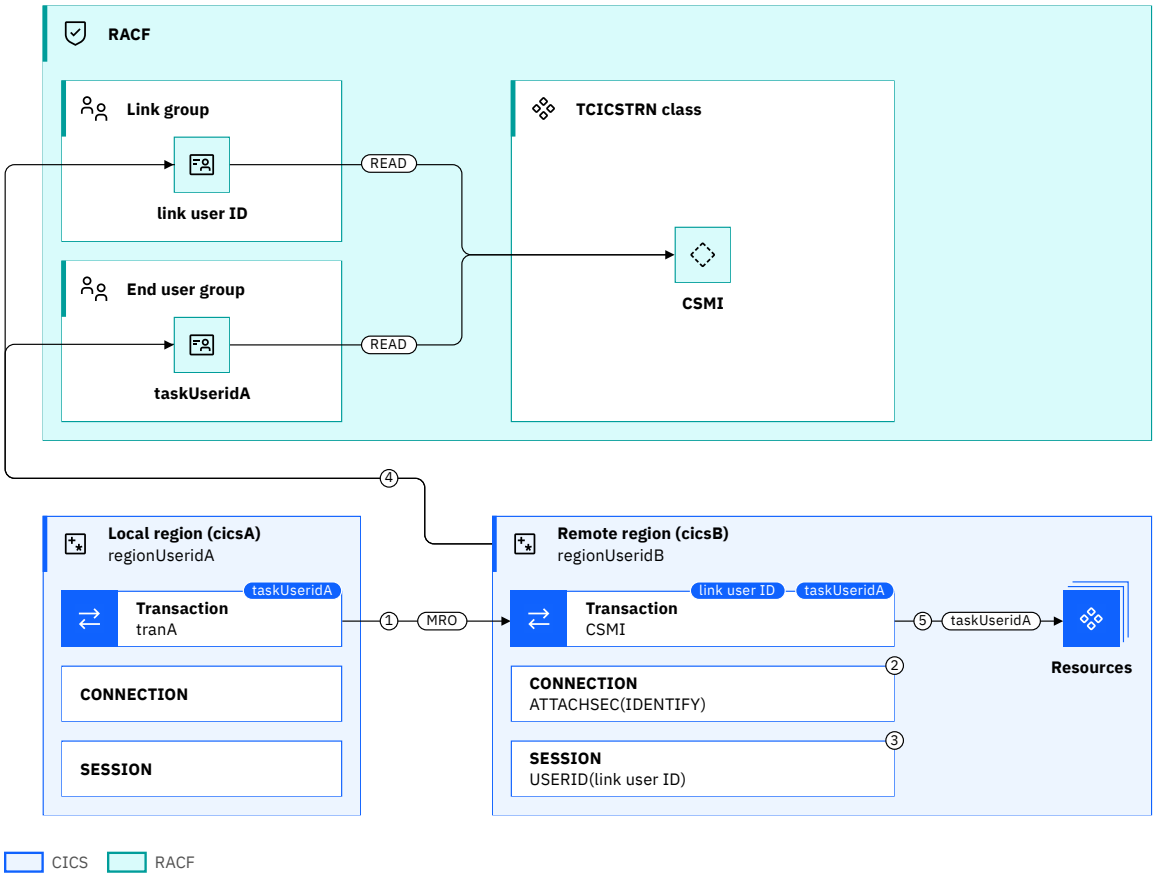


Figure 45. MRO connection – using both the task ID and the link user ID

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.
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. For more information, see Allowing the regions to use MRO .
- Allow each region to connect to the other region. For more information, see Allowing regions to connect to each other using 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 TCICSTRN INTERCOM UACC(NONE) ADDMEM(CSMI)
PERMIT INTERCOM CLASS(TCICSTRN) ID(linkUserGroup) ACCESS(READ)
PERMIT INTERCOM CLASS(TCICSTRN) 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

For common configuration tasks and additional information, refer to the following topics.

Allowing the regions to use MRO

Learn how to allow regions to use MRO.

For each CICS region with APPLID *applid* and Region User ID *region user ID*.

```
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

Learn how to allow two regions to connection to each other using MRO.

Two CICS regions with APPLIDs *applidA* and *applidB* and region user IDs *regionUseridA* and *regionUseridB* .

```
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 that have applids that 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 would be.

```
RDEFINE FACILITY DFHAPPL.PROD* UACC(NONE)
PERMIT DFHAPPL.PROD* CLASS(FACILITY) ID(productionRegionGroup) ACCESS(READ)
```

Which regions connect is still subject to connection definitions.

Recommendation: Do not mix definitions for production and test or development regions.

Recommendation: Where possible, development regions must use a separate RACF database. This configuration avoids any possibility of development regions that connect to production regions.

Chapter 11. 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 140 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 141 describes how connection security is implemented at this stage.

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 141 describes how connection security is implemented at this stage.

These flows are shown in Figure 46 on page 140. Connection security can be implemented at each of these flows.

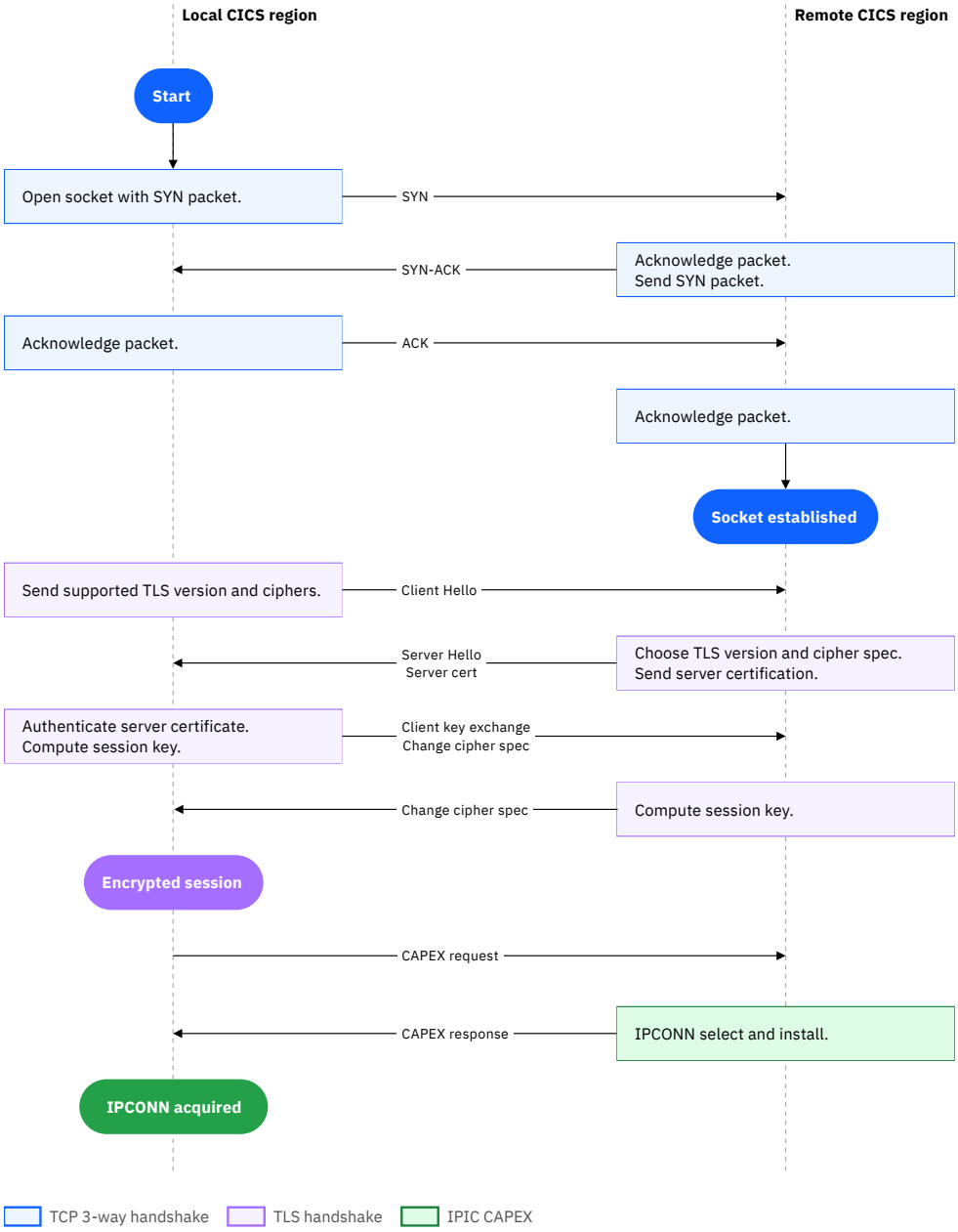


Figure 46. Network flows to establish an IPIC socket

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 z/OS Connect Enterprise Edition, 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 EE 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 TCPIPSERVICE 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 46 on page 140 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, TCPIPSERVICE 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 IBM zEnterprise® Unified Resource Manager (URM) attribute in the relevant TCPIPSERVICE 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, which 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 confidentially 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 TCPIPSERVICE resource definition, along with equivalent settings in the client. For more information, see TCPIPSERVICE attributes.

Figure 47 on page 142 summarizes the resources that are used to define TLS connectivity between two CICS regions.

Figure 47. TLS support for inbound and outbound IPCONN resource definitions

The TCPIPSERVICE 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 TCPIPSERVICE resource definition. If SSL(CLIENTAUTH) is specified in the TCPIPSERVICE 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 48 on page 142 summarizes the resources that are used to define TLS with a one-way IPCONN between an IPIC client and a CICS region.

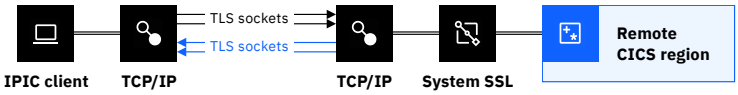


Figure 48. TLS support for IPIC client connections

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 TCPIPSERVICE 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: *cicsA*	**Remote CICS Region: *cicsB* Default User: *DefaultUseridB***			
Region user ID	**Region user ID**	**IPCONN resource definition**		**Link user ID**
		LINKAUTH attribute	**SECURITYNAME attribute**	
regionUseridA	*regionUseridB*	CERTUSER		*certUseridA*
regionUseridA	*regionUseridB*	SECUSER	*regionUseridB*	No link user ID
regionUseridA	*regionUseridB*	SECUSER	*functionalUseridB*	*functionalUseridB*
regionUseridA	*regionUseridB*	SECUSER		*defaultUseridB*

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 on the IPCONN resource definition in the remote CICS region.

Figure 49 on page 144 shows the security attributes defined in the resource definitions for user security.

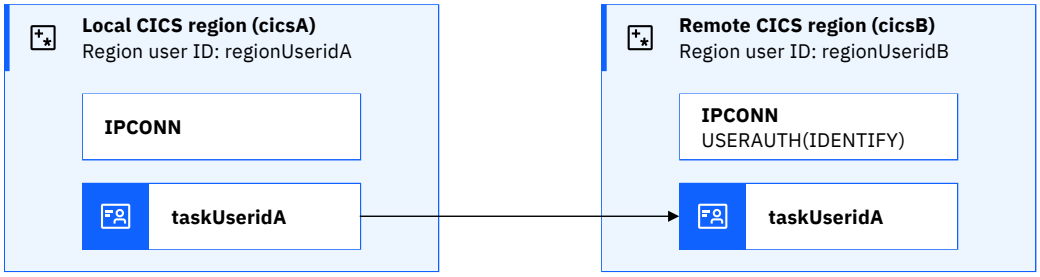


Figure 49. Resource definition attributes used for user security

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 EE.
- 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 security 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 17 on page 144.

Table 17. Identification and authentication

USERAUTH attribute	Description
IDENTIFY (see 1)	The *asserted user ID* 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 17. 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, a secondary user ID is associated with the task. This secondary user ID is used in addition to the task user ID for all checks for transaction security and resource security.

Table 18. How to identify the secondary ID when a link user ID is provided					
Local CICS region: *cicsA*		**Remote CICS region: *cicsB*** **Default User ID: *defaultUseridB***			
Region user ID	**Task user ID**	**Region user ID**	**USERAUTH** on IPCONN	**Link user ID** *	**Task user ID**
regionUseridA	*taskUseridA*	*regionUseridB*	LOCAL	*linkUseridB*	*linkUseridB*
regionUseridA	*taskUseridA*	*regionUseridB*	LOCAL	*regionUseridA*	*defaultUseridB* **
regionUseridA	*taskUseridA*	*regionUseridB*	IDENTIFY or VERIFY	*linkUseridB*	*taskUseridA*
regionUseridA	*taskUseridA*	*regionUseridB*	IDENTIFY or VERIFY	*regionUseridA*	*taskUseridA*
regionUseridA	*taskUseridA*	*regionUseridB*	DEFAULTUSER	*linkUseridB*	*defaultUseridB* **
regionUseridA	*taskUseridA*	*regionUseridB*	DEFAULTUSER	*regionUseridA*	*defaultUseridB* **

* 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 147
- 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 TCPIPSERVICE 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 TCPIPSERVICE resource definition is set to CLIENTAUTH. The TCPIPSERVICE 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 TCPIPSERVICE resource definition that has the attribute of PROTOCOL(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 50 on page 148 shows an overview of the scenario.

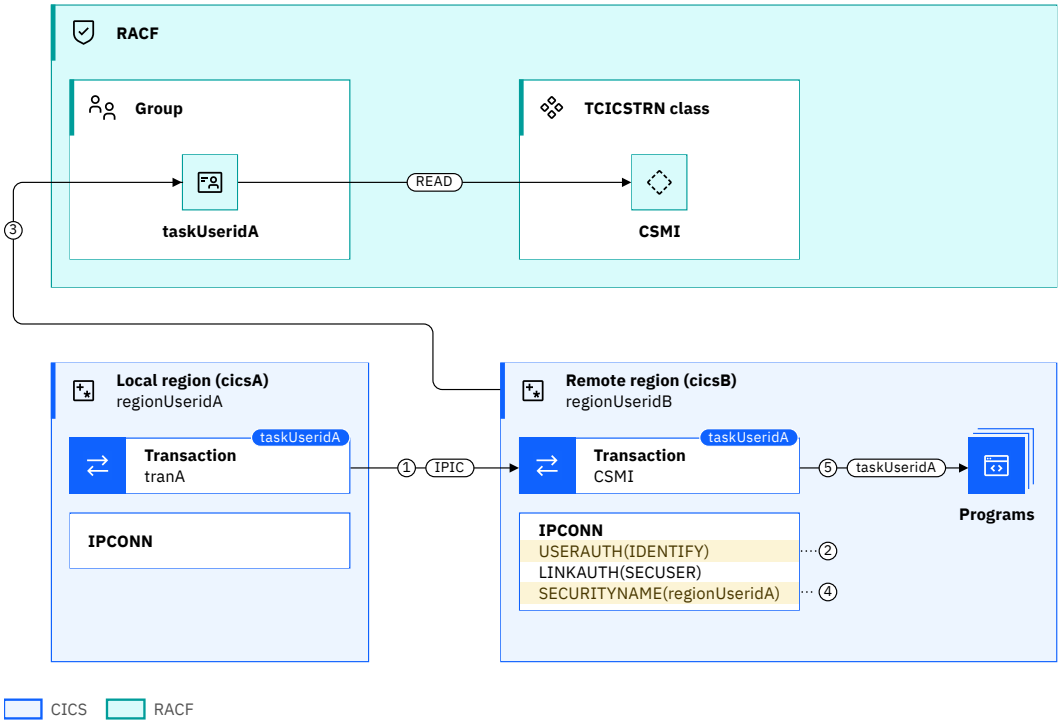


Figure 50. IPIC trusted connection between systems in the same sysplex

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 the profile called CSMI in the TCICSTRN class.
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 TCPIPSERVICE 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)
       TCPIPSERVICE(tcpipB)
       SENDCOUNT(nSess) RECEIVECOUNT(nSess)
DEFINE TCPIPSERVICE(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)
       TCPIPSERVICE(tcpipA)
       LINKAUTH(SECUSER) SECURITYNAME(regionUseridA)
       USERAUTH(IDENTIFY)
       SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)
DEFINE TCPIPSERVICE(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 UACC(NONE) ADDMEM(CSMI)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
PERMIT INTERCOM CLASS(GCICSTRN) ID(regionUseridA) ACCESS(READ)
SETROPTS 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.

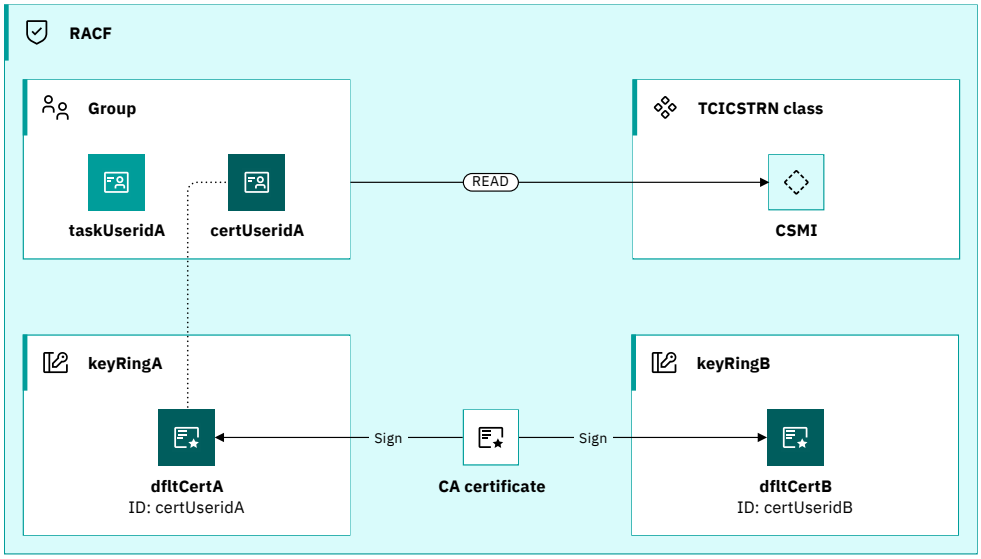


Figure 51. RACF configuration to connect two CICS regions that use TLS

In Figure 51 on page 151, 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, which allows cicsA to DPL to cicsB. certUserB'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.

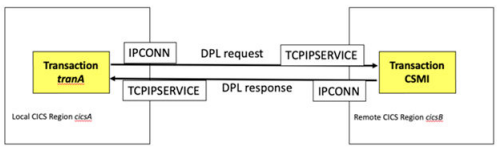


Figure 52. Overview of IPCONN and TCPIP setup to connect two CICS regions that use TLS

Shown in Figure 52 on page 151 are a pair of definitions with an IPCONN in one region that connects to a TCPIPSERVICE 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 51 on page 151.

The flow of a DPL request from cicsA to cicsB, and the definitions that are required to achieve this configuration are shown in Figure 53 on page 152. The DPL response from cicsB to cicsA and the definitions required to achieve this flow are shown in Figure 54 on page 153.

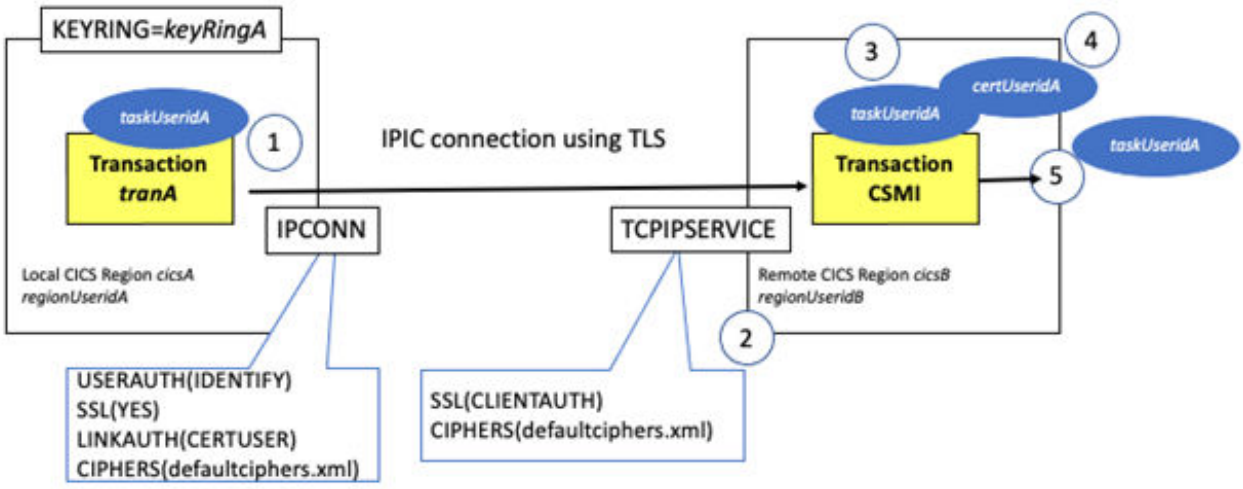


Figure 53. IPCONN and TCPIP setup for a DPL request

Figure 53 on page 152 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 check whether *taskUseridA* belongs to a group that has READ authority to the profile called CSMI in the TCICSTRN class.
4. Link security is also in force as 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 check whether *certUseridA* belongs to a group that has READ authority to the profile called CSMI in the TCICSTRN 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).

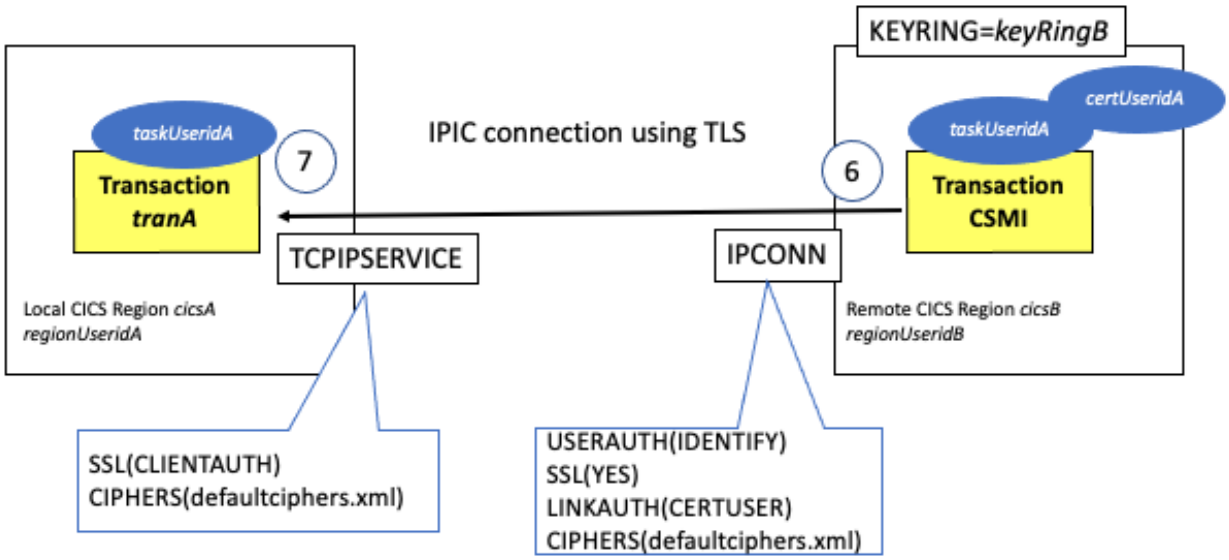


Figure 54. IPCONN and TCPIP setup for the DPL response

Figure 54 on page 153 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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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)
       TCPIPSERVICE(tcpipB) SSL(YES)
       LINKAUTH(CERTUSER) USERAUTH(IDENTIFY)
       SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)
DEFINE TCPIPSERVICE(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)
       TCPIPSERVICE(tcpipA) SSL(YES)
       LINKAUTH(CERTUSER) USERAUTH(IDENTIFY)
       SENDCOUNT(nSess) RECEIVECOUNT(nSess) AUTOCONNECT(YES)
DEFINE TCPIPSERVICE(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 TCICSTRN INTERCOM UACC(NONE) ADDMEM(CSMI)
PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
PERMIT INTERCOM CLASS(GCICSTRN) ID(certUseridA) ACCESS(READ)
SETROPTS 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,in 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 z/OS Connect Enterprise Edition colocated in the same sysplex as the remote CICS region. This example demonstrates how the client system can assert identities in CICS based on authentication that is performed in the z/OS Connect EE system. This scenario applies to any client system that supports IPIC connections to CICS TS, including both CICS TG and z/OS Connect EE.

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 a remote CICS region.

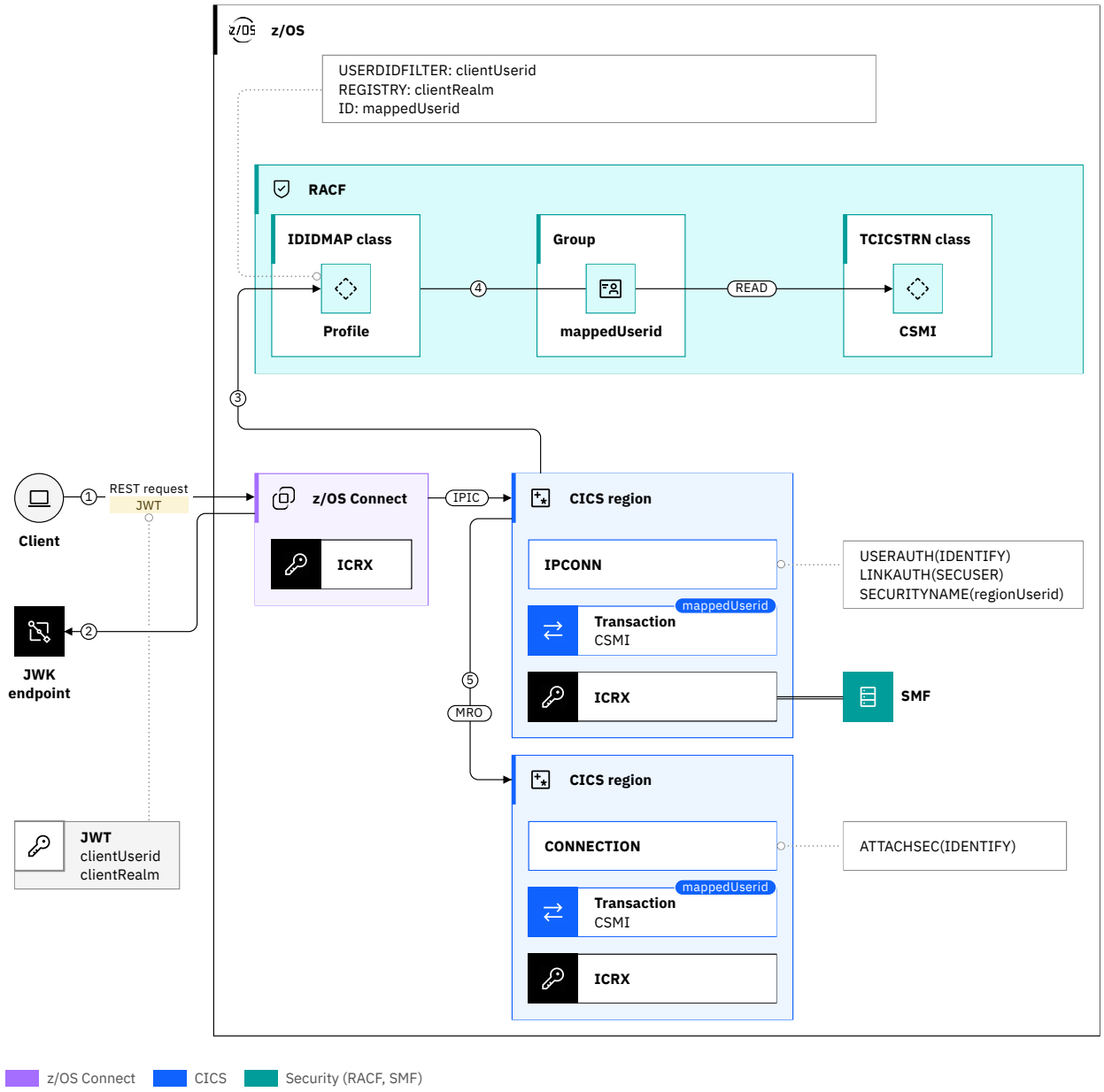


Figure 55. A trusted IPIC client connection to a remote CICS region

The following security flows occur at the points shown:

1. The client sends a JWT to z/OS Connect EE. The distributed identity (*clientUserid* and *clientRealm*) is represented by claims in the JWT (normally the subject and issuer claims)
2. z/OS Connect EE 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 AOR is performed by checking that the *mappedUserid* has READ authority to the TCICSTRN profile for the transaction CSMI. The CICS region does this check by calling RACF to validate that the *mappedUserid* belongs to a group that has READ authority to the profile called CSMI in the TCICSTRN class.
5. *mappedUserid* 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).

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 AOR.

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 z/OS Connect Enterprise Edition (EE) 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 EE to perform JWT authentication and use the identity in the JWT to authorize access to z/OS Connect EE. For more information, see How to configure JWT authentication.
- Configure z/OS Connect EE 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 EE 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 EE.
- *zceeAppl* is the APPLID specified in the z/OS Connect EE CICS service provider on the zosConnectApplid attribute within the zosconnect_cicsIpicConnection element.

- *zceeNetid* is the network ID specified in the z/OS Connect EE zosConnectNetworkid attribute within the zosconnect_cicsIpicConnection element.
- *nSess* is the number of sessions defined.
- The distributed identity flowed from z/OS Connect EE 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 TCPIPSERVICE that listens for IPIC connection requests from z/OS Connect EE. The value URM(NO) is used to disable connection autoinstall.

      ```
      DEFINE TCPIPSERVICE(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 EE, naming the TCPIPSERVICE defined in step 1a and the APPLID and network ID used by the z/OS Connect EE 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 DIGTNMAP class to allow certificate name filters to be created or changed.

      Enter the following RACF command:

      ```
      SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
      ```

   b. Create a RACMAP to map the distributed identity flowed from z/OS Connect EE to a RACF user ID.

      Enter the following RACF command:

      ```
      RACDCERT ID(mappedUserid) MAP
      USERDIDFILTER(NAME('CN=clientUserid,O=IBM,C=US'))
      REGISTRY(NAME('*')) WITHLABEL('ZCEE-LABEL')
      ```

   c. Refresh the DIGTNMAP RACF class for the changes to take effect.

   ```
   SETROPTS RACLIST(DIGTNMAP) 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 UACC(NONE) ADDMEM(CSMI)
   PERMIT INTERCOM CLASS(GCICSTRN) ID(taskUserGroup) ACCESS(READ)
   SETROPTS RACLIST(TCICSTRN) REFRESH
   ```

4. Send a request from the z/OS Connect EE 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 EE 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, you select the **Add Security Request Recording** pop-up menu option. On that window, select the **IPIC** tab and set the **Appl ID** to the *zceeAppl* from the z/OS Connect EE 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:

- CICS Explorer Task Association view.
- DNAME and REALM returned by the INQUIRE ASSOCATION 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 options of 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 doing an install
- *installerGroup* is the Group of people allowed to do an install

- *resourceUserid* is the userID 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 12. Security for LU6.2 connections

Information about security for LU6.2 connections is not included in this version of the documentation. It is available in previous versions: Implementing LU6.2 security.

Chapter 13. Security for LU6.1 connections

Information about security for LU6.2 connections is not included in this version of the documentation. It is available in previous versions: Implementing LU6.1 security.

Chapter 14. Security for CICS Liberty

A Liberty JVM server can run Java applications that are accessed in different ways, for example, a web application that uses HTTP or a JMS application that uses a message driven bean (MDB). Since Liberty has its own security model, security is mostly configured directly in the Liberty server and in the Enterprise Java application deployment descriptor. Additional security layers can be configured through CICS security by using transaction and resource security.

When deciding how to secure an application running in a Liberty JVM server you need to consider the following security principles: authentication, identification, authorization, integrity, confidentiality, and audit. For a description of these terms see What does security mean in CICS?.

Different options, or combinations of capabilities, can be used depending on the types of application to be run in a Liberty JVM server.

- For help with designing security for web applications, see Designing security for Liberty web applications.
- For help with designing security for Link to Liberty applications, see Designing security for Link to Liberty applications.

For information on specific configuration tasks, see Configuring security for 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 in CICS Liberty security.

Components of Liberty security

The key components of security in a Liberty JVM server are:

- The user registry, which provides authentication and authorization.
- The CICS Liberty security feature, which integrates Liberty and CICS security.
- The angel process, which provides access to authorized SAF registry services.
- The Java HTTP listener, which provides TLS support.

These components are shown in: Figure 56 on page 166

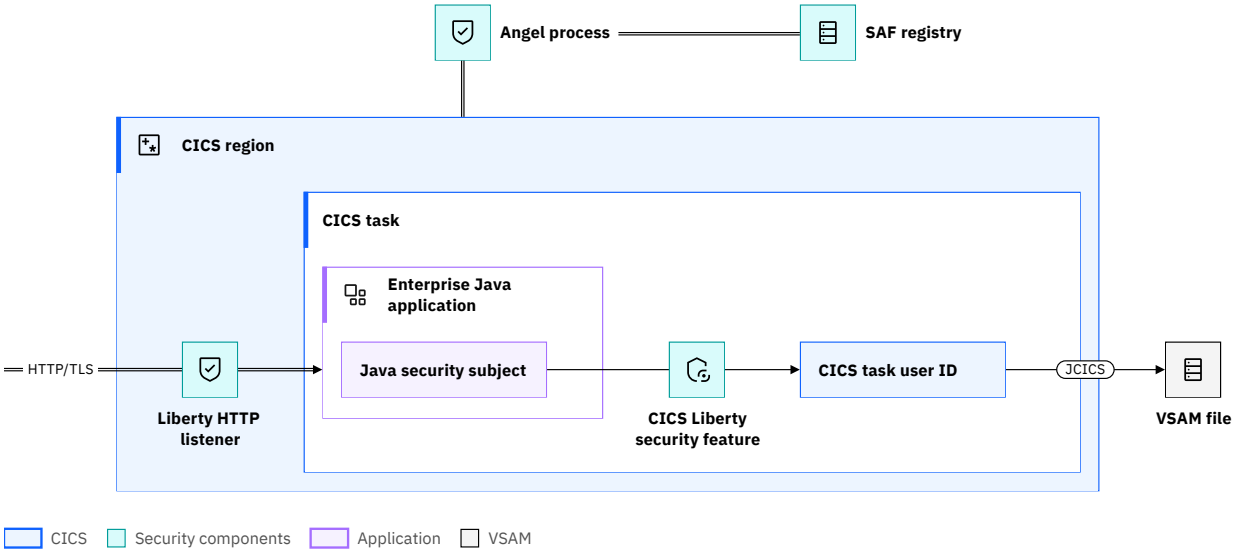


Figure 56.

User registry

User registries store information about users and groups that can be used for authentication and authorization. CICS Liberty supports the following types of user registry.

- System Authorization Facility (SAF) registry

  This is the RACF database. The angel process is used by Liberty to handle security calls to RACF. 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) user registry

  LDAP security registries are widely used outside of z/OS to store enterprise security information. It is possible for an LDAP security 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 user 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 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 like SAF (see Figure 57 on page 167).

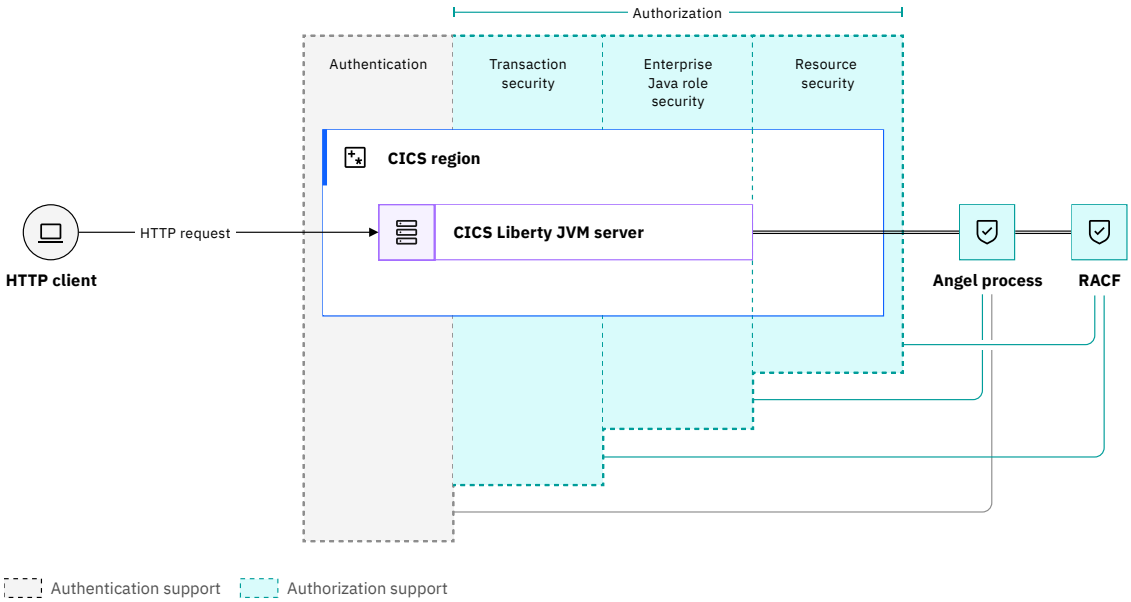


Figure 57. How Liberty servers call the angel process

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.

For more information about using the Liberty angel process with CICS Liberty, see Using the Liberty angel to access authorized z/OS services.

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 JVM server.

Using named angels allows multiple uniquely named angel processes to run on a single z/OS system, in addition to the default unnamed angel process. With this configuration, you can isolate Liberty servers from one another so that they can run different service levels or be managed independently.

For more information about named angel processes, see Named angel.

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 than the one bundled with CICS TS. In this case, make sure that you are using the latest version of the angel process, regardless of which product it is bundled with. The latest version might be bundled with other IBM software, and might supersede the version that is bundled with CICS TS. If this configuration is not what is happening, and CICS TS provides the latest version of the angel process, use the angel process that is bundled with CICS TS.

You can identify the Liberty version of the angel process and the Liberty JVM server that's running in CICS as shown in The Liberty server 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.

For more information about configuring the Liberty angel process, see the 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 a Liberty JVM server.

Transport Layer Security

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 usage 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 58 on page 168. This support can be integrated with the RACF security registry to control access to digital certificates that are stored in RACF.

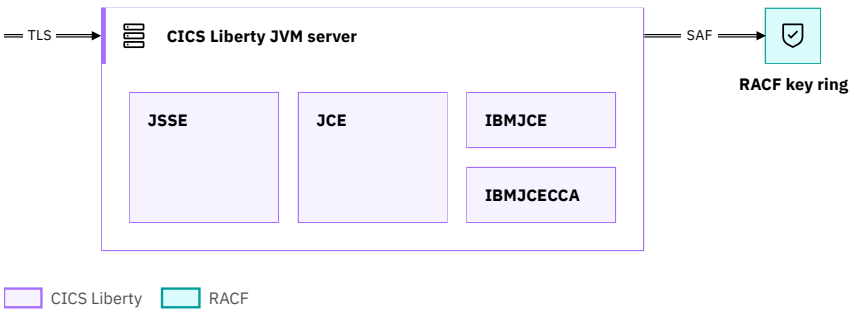


Figure 58. TLS support in Java that uses JCE and JSSE

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 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). 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. The use of JCE provider 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 SSL (TLS) for a Liberty JVM server using a Java keystore.

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: 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.

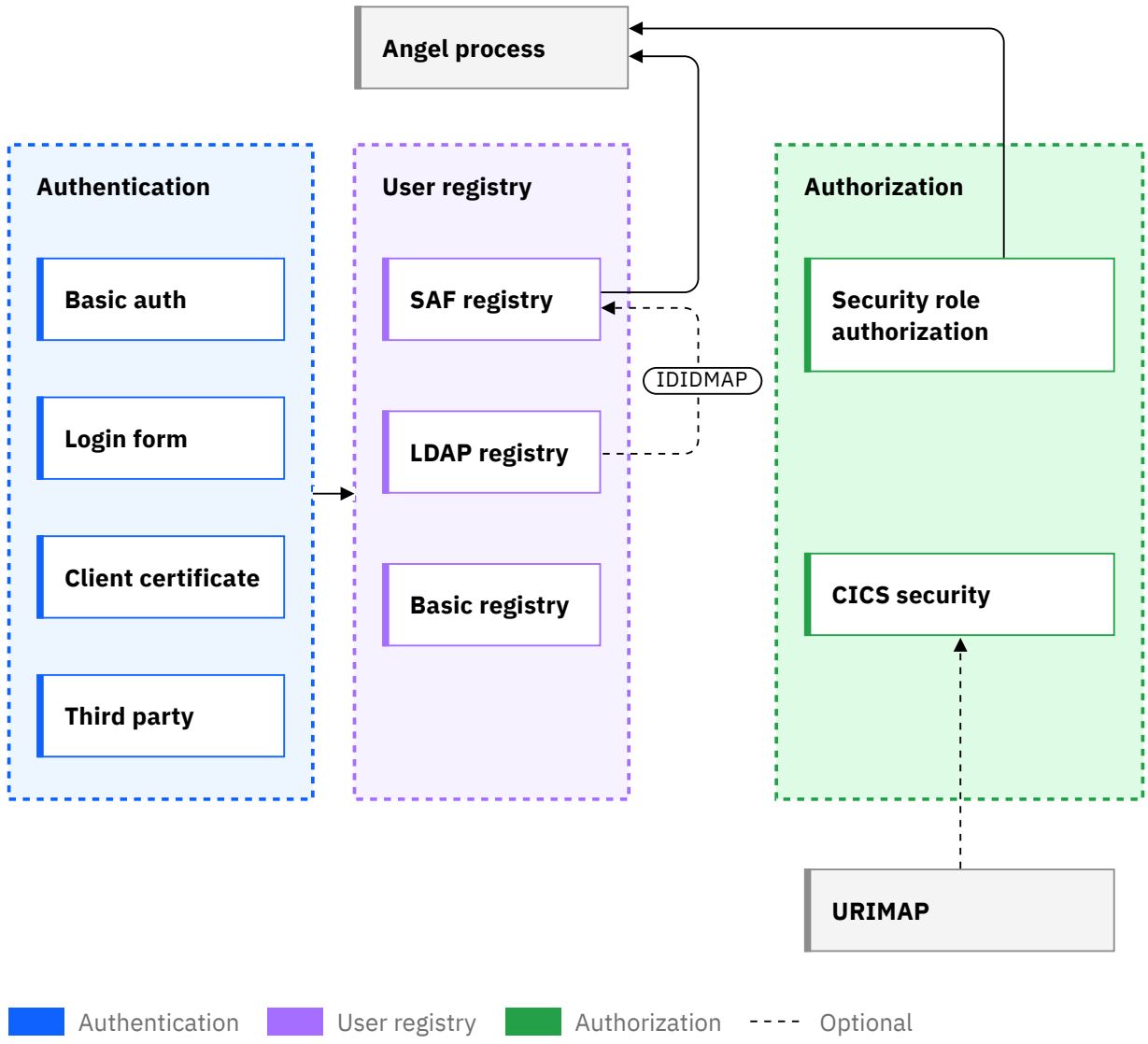


Figure 59. Web authentication and authorization options for CICS Liberty

Options in the diagram are also available with related links in the following table.

Authentication	User registry	Authorization	External components
• "Basic authentication" on page 171 • "Form-based authentication" on page 172	• Configuring Liberty to use a SAF user registry • Configuring Liberty to use an LDAP user registry	• "Role authorization" on page 177 • "CICS transaction security" on page 176	• Angel process • URIMAP

Authentication	User registry	Authorization	External components
• "Client authentication" on page 172 • "Third-party authentication" on page 172	• Configuring basic authentication for Liberty web applications		

Authenticating users to run CICS Liberty web applications

CICS Liberty supports authentication by providing a selection of options that include basic, form-based, client, and third-party authentication.

- Basic authentication specifies that the web client must provide a user ID and password.
- 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.
- Client authentication specifies that the web client must provide a certificate that is then validated and associated with a user ID.
- Third-party authentication specifies that the user authenticates with a trusted third party that then sends an authentication token to CICS Liberty.

Single Sign-On (SSO) can be implemented with each of these options (see Single Sign-on). It is also possible to write your own custom authentication solution (see Custom authentication options).

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`). Third-party authentication is configured in the Liberty server configuration file `server.xml`.

Note: You can either override the authentication method to be used for all applications or you can specify which authentication mechanism must be used if client authentication fails. You can do this by configuring 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.

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 user registry.
- An LDAP distinguished name and password, which is defined in an LDAP user 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 use a custom login page for users to access your application from a browser. The custom login page uses a form-based login, which is a customized form of basic authentication.

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 authentication

Certificate-based client authentication uses information that is provided in the client's TLS certificate to map to an associated user ID. It also provides all the normal benefits that are associated with a secure TLS connection.

When client 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 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 How it works: JSON Web Token (JWT)JSON Web Token (JWT).

Figure 60 on page 173 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.

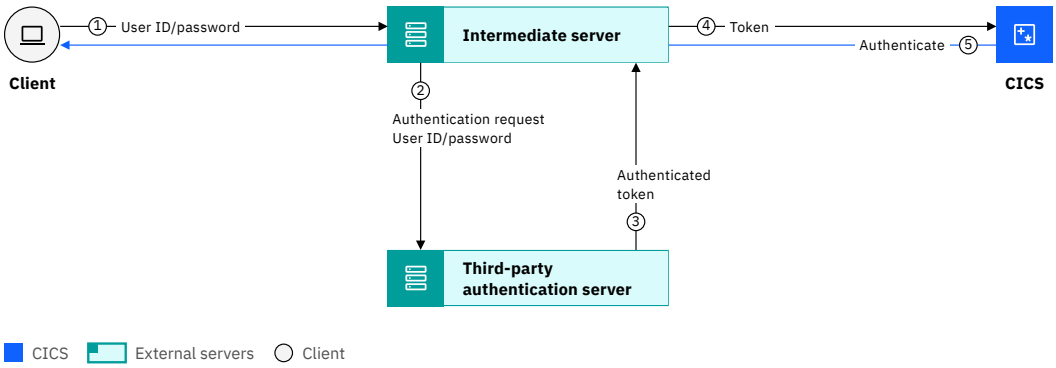


Figure 60. Third-party authentication

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 message is not changed between being created by the authentication server and received by 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 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

Third-party authentication can be implemented by using OpenID Connect, which allows applications to verify the identity of the user based on the authentication that is implemented by an OpenID Connect Provider.

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, then 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.

OpenID Connect is based on the OAuth 2.0 protocol, which is an open standard for access delegation. For more information about using OAuth 2.0 with CICS Liberty, see Authorizing applications by using OAuth 2.0.

Single Sign-On

Single Sign-On (SSO) enables users to log in to one server and then 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 Lightweight Third Party Authentication (LTPA) token, although it is also possible to use a JSON Web Token (JWT) as an SSO token.

If you use the `cicts: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 more information or to change the default behavior, see Configuring LTPA in Liberty in the IBM Documentation for Liberty z/OS.

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:

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.

For more information about the JWT feature, see Building and consuming JSON Web Token (JWT) tokens.

For an example on using the JWT feature with CICS Liberty, see the GitHub sample CICS Developer Center: Using the JWT feature with CICS Liberty.

Enterprise Java Security API

The Enterprise Java security API provides a programmatic way to enable HTTP basic authentication or form-based authentication. Authentication can use either an LDAP registry, a database identity store, or a custom identity store. To use the Enterprise Java security API in a Liberty JVM server, you must enable the Liberty appSecurity-3.0 feature.

The Enterprise Java Security API specification extends self-contained application security capabilities to port across different application servers and uses modern programming concepts, such as Expression Language (EL) and Contexts and Dependency Injection (CDI). It covers three principles:

1. An authentication mechanism provided by the **`HttpAuthenticationMechanism`** interface for the servlet container.
2. An identity store, which standardizes the JAAS **`LoginModule`**.
3. A security context, which acts as an access point for programmatic security.

For more information about the Enterprise Java security API, see Enterprise Java Security API in the Liberty documentation.

For more information about using the Enterprise Java security API for authenticating users with a CICS Liberty JVM server, see Configuring security for a Liberty JVM server with the Enterprise Java security API.

Java Authentication Service Provider Interface for Containers (JASPIC)

Java Authentication Service Provider Interface for Containers (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 the GitHub sample CICS Developer Center: Using a JASPIC implementation with CICS Liberty.

For more information, see Configuring a Java Authentication SPI for Containers (JASPIC) User Feature.

Trust Association Interceptor (TAI)

A Trust Association Interceptor (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 the GitHub sample CICS Developer Center: Using a Trust Association Interceptor to validate a JWT with CICS Liberty.

For more information, see Configuring TAI for Liberty.

Authentication mechanism with Enterprise Java Security API

An authentication mechanism is a way that is used to obtain a username and password from the user to be processed later by the Java Security API. You have two standard options for authentication, both take advantage of the annotations that are introduced by the Enterprise Java Security API.

HTTP basic authentication

Basic authentication displays the browser's built-in login dialog before the user can access the protected resource.

```
@BasicAuthenticationMechanismDefinition(realmName="user-realm")
@WebServlet("/home") @DeclareRoles({"user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class HomeServlet extends HttpServlet {
    ...
}
```

Form-based authentication

You can use form-based authentication to replace the browser’s built-in dialog with your own custom HTML form. You can create an application config class with annotations as shown.

```
@FormAuthenticationMechanismDefinition(
    loginToContinue = @LoginToContinue(
        loginPage = "/login",
        errorPage = "/error"
    )
)
@ApplicationScoped
public class ApplicationConfig {
    ...
}
```

Identity store with Enterprise Java Security API

An identity store acts as a DAO (Data Access Object) for accessing user information, including their usernames, passwords, and associated roles.

A number of identity store types are introduced by the Enterprise Java Security API, including:

Database identity store

A database identity store is used to retrieve user information from a relational database. The following `@DatabaseIdentityStoreDefinition` annotation configures an identity store with the parameters necessary to connect to an external database and validate user credentials.

```
@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"
    }
)
```

Lightweight Directory Access Protocol (LDAP) identity store

The following annotation `@LdapIdentityStoreDefinition` configures an identity store with the necessary options to communicate with an external LDAP server and validate user credentials.

```
@LdapIdentityStoreDefinition(
    url = "ldap://localhost:33389/",
    callerBaseDn = "ou=user,dc=jsr375,dc=net",
    groupSearchBase = "ou=group,dc=jsr375,dc=net"
)
public class HomeServlet extends HttpServlet{
```

```
    ...
}
```

url
The URL of the LDAP server to use for authentication.

callerBaseDn
Base distinguished name for callers in the LDAP store.

groupSearchBase
Search base for looking up groups.

Custom identity store
In addition to the built-in identity stores found in Enterprise Java Security API, a user can implement their own identity store and control exactly where to obtain user information. This configuration can be achieved by creating a custom identity store class, then creating an HTTP authentication mechanism associated with this custom identity store.

Security context with Enterprise Java Security API

The security context object is used to check a user's authority to access a specific resource by using a program. This function is useful when you need to perform custom behavior.

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, res);
        } else {
            req.getRequestDispatcher("/logout").forward(req, res);
        }
    }
}
```

Authorizing users to run CICS Liberty web applications

To authorize user access to applications that run in a Liberty JVM server, you can use standard CICS transaction and resource security profiles. These profiles restrict access to the transaction and its resources that run as part of the web request received by the Liberty server. Additionally you can restrict access to the Java application by using the security roles function provided by Liberty. You can also control the thread identity that is used for accessing operating system resources by using the Liberty syncToOSThread function.

CICS transaction security

Java applications that are running in a Liberty JVM server are subject to CICS transaction and resource security. Any user IDs that are authenticated in Liberty must also include the correct access permissions to run transactions, access files, or call programs.

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, it is recommended to 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 having to configure Enterprise Java role authorization by granting access to all Java components 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. Authorization to the application is then controlled entirely by using URIMAPs and CICS transaction

security. If you deploy a Liberty application in a CICS bundle, CICS automatically configures this mechanism for you in the `installedApps.xml` file in Figure 61 on page 177.

```
<server>
  <application 1 id="example.app"
               type="war"
               name="Example application"
               location="${server.config.dir}/apps/example.app.war" >
    <application-bnd>
      <security-role name="cicsAllAuthenticated"> 2
        <special-subject type="ALL_AUTHENTICATED_USERS" /> 3
      </security-role>
    </application-bnd>
  </application>
</server>
```

1 Application element added by CICS bundle installation process.

2 Role name must match `web.xml` role.

3 Predefined as all authenticated subjects.

Figure 61.

For more information, see Authorizing users to run applications in a Liberty JVM server.

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 component. 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 description by using an **<auth-constraint>** element. This method maps specific URL patterns within the web application to a defined security role.
2. Using the rolesAllowed modifier of the servlet `@HttpConstraint` annotation in a web application. This option is an alternative to option one.
3. On an individual Java method by using the `@RolesAllowed` annotation to define the permissible roles for this specific component. This option is only supported for EJB components, which can either be called from a servlet or linked to using Link to Liberty if the EJB is also a CDI managed bean.

Note: If no authorization constraint is present in the Java application's deployment descriptor, then the Liberty server does not enforce authentication of the Java security subject.

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 and mapping takes place outside the SAF registry. The mappings are defined either in the **<application>** element of the `server.xml` or alternatively if you are using an EAR file the mapping can be configured in the `ibm-application-bnd.xml` or `ibm-application-bnd.xmi` files. 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 is mapped to the special subject ALL_AUTHENTICATED_USERS as shown.

```
<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>
```

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 and authorization to the transaction must be controlled by using CICS transaction security. If applications are deployed directly into Liberty without a CICS bundle, by using an **<application>** element, then more complex role mappings can be configured.

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 then the **<application-bnd>** no longer acts as the source of user ID to role mapping. Instead, user or group role membership is determined by the Liberty SAF role mapper and corresponding RACF security profiles in the EJBROLE class. Authenticated users are authorized to access an application by giving them access to the EJBROLE that is referenced in the `profilePattern` attribute of 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. Any security-role information in application elements in `server.xml` is ignored.

For more information, see Configuring SAF authorization with an EJBROLE .

Default role to group mapping

If SAF authorization is not in use and local role mapping binding information is not provided in an **<application-bnd>** element, Liberty provides a default role to group mapping by mapping the role directly to a group in the security registry. So for instance, if the role in the **<auth-constraint>** protecting the application is defined as *MANAGER*. Then, any user ID that belongs to the group *MANAGER* in the local security registry has access to the role and thus to the application.

Operating system thread authorization

Within a CICS region, the operating system thread identity of any UNIX System Services thread defaults to be 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 use 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, authenticated by Liberty, to be synchronized with the operating system thread identity. With syncToOSThread in effect, the user's subject is used to access operating system resources such as zFS files.

For more information, see Using 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 62 on page 179 shows a view of the security options available with Link to Liberty.

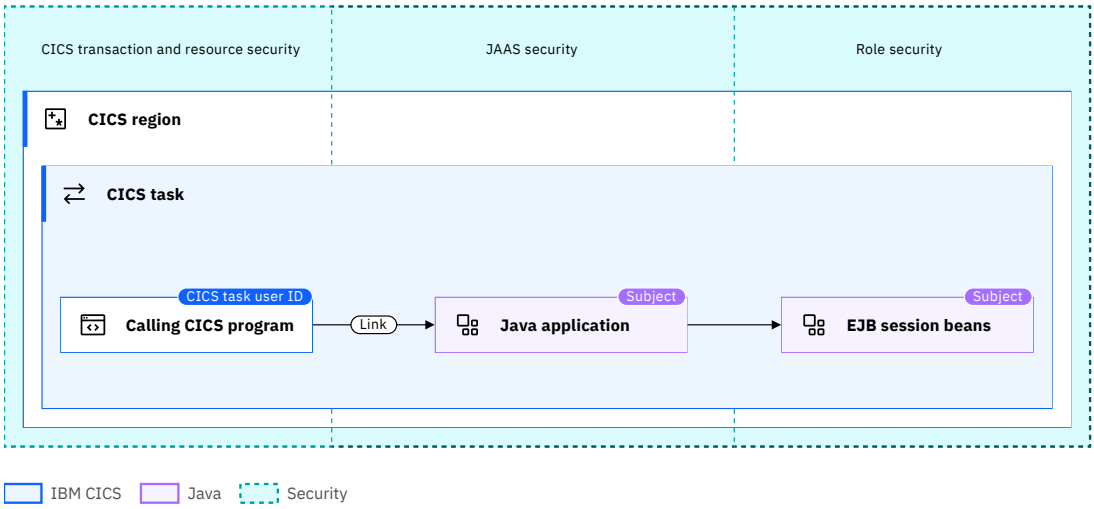


Figure 62. Security options for Link to Liberty

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: Liberty angel process

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.

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 as 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. When the angel process is run in a CICS region, the angel process is the only way for Liberty servers to authenticate users with the SAF registry. The access to its services is controlled by the SAFCRED resource profiles.

Any Liberty server can connect only to a single angel process at server startup. However, multiple Liberty servers can share the angel process, independently of the level of code that the servers are running. 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.

An angel process can service only Liberty servers that run on the same LPAR. Each LPAR can have multiple named angel processes but only one default angel process.

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 SAFCRED, 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 SAFCRED 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 would typically access a CICS resource such as a VSAM file or a Db2 connection, or link to a CICS program. Figure 63 on page 181 shows a high-level view of how the different security principles are supported for web applications.

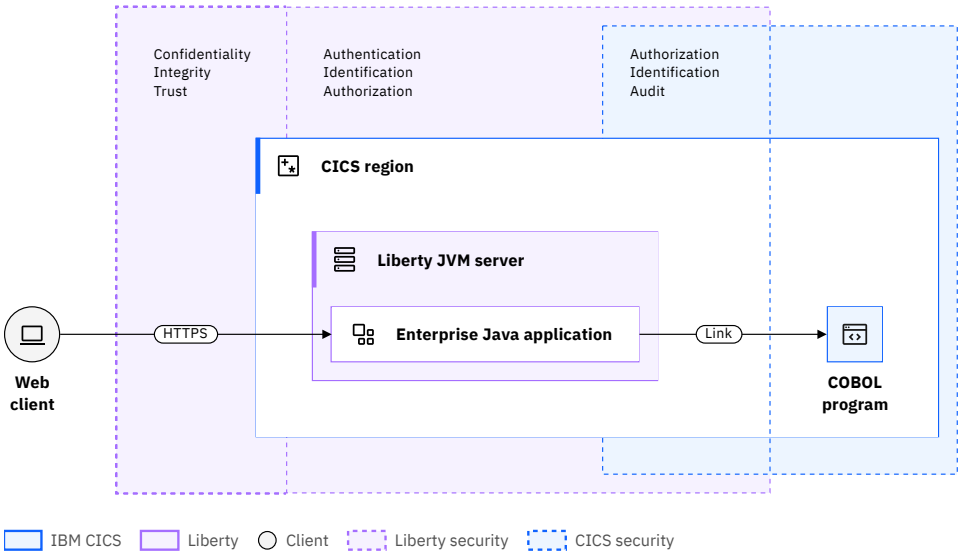


Figure 63.

Figure 63 on page 181 shows how:

- HTTPS is used for confidentiality, integrity, and trust.

- Java Platform, Enterprise Edition roles (Enterprise Java), and Liberty security is used for authentication, identification, and, authorization. You can authenticate users by using various different authentication mechanisms and authorize access to web applications through Enterprise Java roles.
- CICS security is used to perform extra 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 are designing 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

What authentication mechanism to use?

When you are using form-based authentication or basic authentication, it is recommended to use HTTPS because the credentials are not signed or encrypted.

TLS client authentication can be used to establish trust between a remote server and a Liberty JVM server. It is most often used when what's being authenticated is not a person, but another server or application. TLS client authentication might not be practical when many clients need to connect to a JVM server. This lack of practicality 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 the client certificate can be imported into the RACF registry as a trusted certificate. When many clients are authenticated by using certificates, it is recommended to use certificate name filtering to avoid having to store and administer too many client certificates.

Basic authentication, form-based authentication, or client authentication might be suitable when the web client authenticates directly with the Enterprise Java application that is running in a Liberty JVM server. For some scenarios, it is recommended to use third-party authentication. For example, if a web client (for example, a REST client) authenticates with an intermediate server and then the intermediate server connects to the Liberty JVM server.

Which user registry to use?

The recommended user registry is a SAF registry (for example RACF), although using 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.

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. Normally, the identity in the token requires to be mapped to a RACF user ID.

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.

What 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 by using a REST or SOAP request)

You can enable third-party authentication by using a JWT or an LTPA token by configuration only. Custom tokens need custom authentication code, for example, a Trust Association Interceptor (TAI). If possible, you are recommended to use a configuration-only solution.

The JWT is the most commonly used third-party authentication token, especially for REST API requests.

What JWT authentication option to use?

You have several options for implementing JWT authentication with CICS Liberty. You are recommended to use the OpenID Connect Client feature, which provides a flexible configuration-only way of implementing JWT authentication. Consider whether to use other JWT authentication options when it is not possible to use the OpenID Connect Client feature. For example, when you need to programmatically access the claims in the JWT.

What if different authentication mechanisms for different services or different types of client is 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 using the OpenID Connect Client feature, you can use authentication filters to identify which types of requests that authenticate by using a JWT.

When to enable Single Sign-On?

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.

You can enable users to log in to one Liberty JVM server and then access applications on other Liberty servers without being prompted to log in again. LTPA SSO is enabled for a Liberty server by default. However, if used 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 recommended 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).

What RACF user ID is used for running the CICS programs and accessing CICS resources?

- If the user is authenticated by using one of the methods that are described in Authenticating users to run CICS Liberty web applications, the RACF user ID used for the CICS task is set from the authenticated Java security subject.
- If an unauthenticated subject is supplied from Liberty, then the USERID defined in the URIMAP is used.
- Otherwise, the CICS default user ID is used, although running requests with the CICS default user ID is not recommended.

Authorization

What option to use for authorizing web requests?

You can use Enterprise Java application security roles to authorize access to web applications. You can also use CICS transaction and resource security.

Using role-based security requires authorization constraints to be defined in application deployment descriptors or in source code annotations.

By using CICS transaction and resource security, you can reuse existing CICS authorization procedures. Using this option requires that individual web applications are accessed from different URIMAPs so that different transaction IDs can be assigned.

What option to use for role-based authorization?

You can use SAF role authorization, local role authorization, or role to group mapping.

SAF role authorization is recommended as it allows security administrators to authorize access to Enterprise Java applications based on RACF profiles. In addition, with syncToOSThread in effect, RACF can be used to authorize access to resources that are not controlled by CICS and instead use the operating system thread identity, such as zFS files.

Local role authorization can be a useful method if you are porting an application directly from a third-party Enterprise Java application server that already uses this method. However, with local 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 SAF role and local role authorizations are not in use. It associates a role to the group of the same name.

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.

Confidentiality and Integrity

What 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.

What 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 as they benefit from the additional protection of SAF (for example, RACF).

What cypher suite to use?

It is recommended to 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 like support for stronger cipher suites. To control the TLS version that is supported by the JVM server, the `sslProtocol` attribute on the `ssl` element can be updated in `server.xml`. To modify the list of ciphers that are supported by the server, a list of ciphers can be specified 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?
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 servers 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 using a private key. The signature can then be verified by using 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?
Due to the way that security processing for Liberty transactions is deferred during CICS transaction attach processing, the user ID audited in the CICS Monitoring Facility (CMF) records might not always match the CICS task user ID. The user ID in the CMF record is determined as follows:

- The user ID in the HTTP security header.
- If no user ID is in the HTTP security header, the user ID taken from the matching URIMAP.
- If neither exist, the CICS default user ID is used.

Review these design examples that offer different configurations.

Design example: Securing a 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 64 on page 186 shows an overview of the scenario.

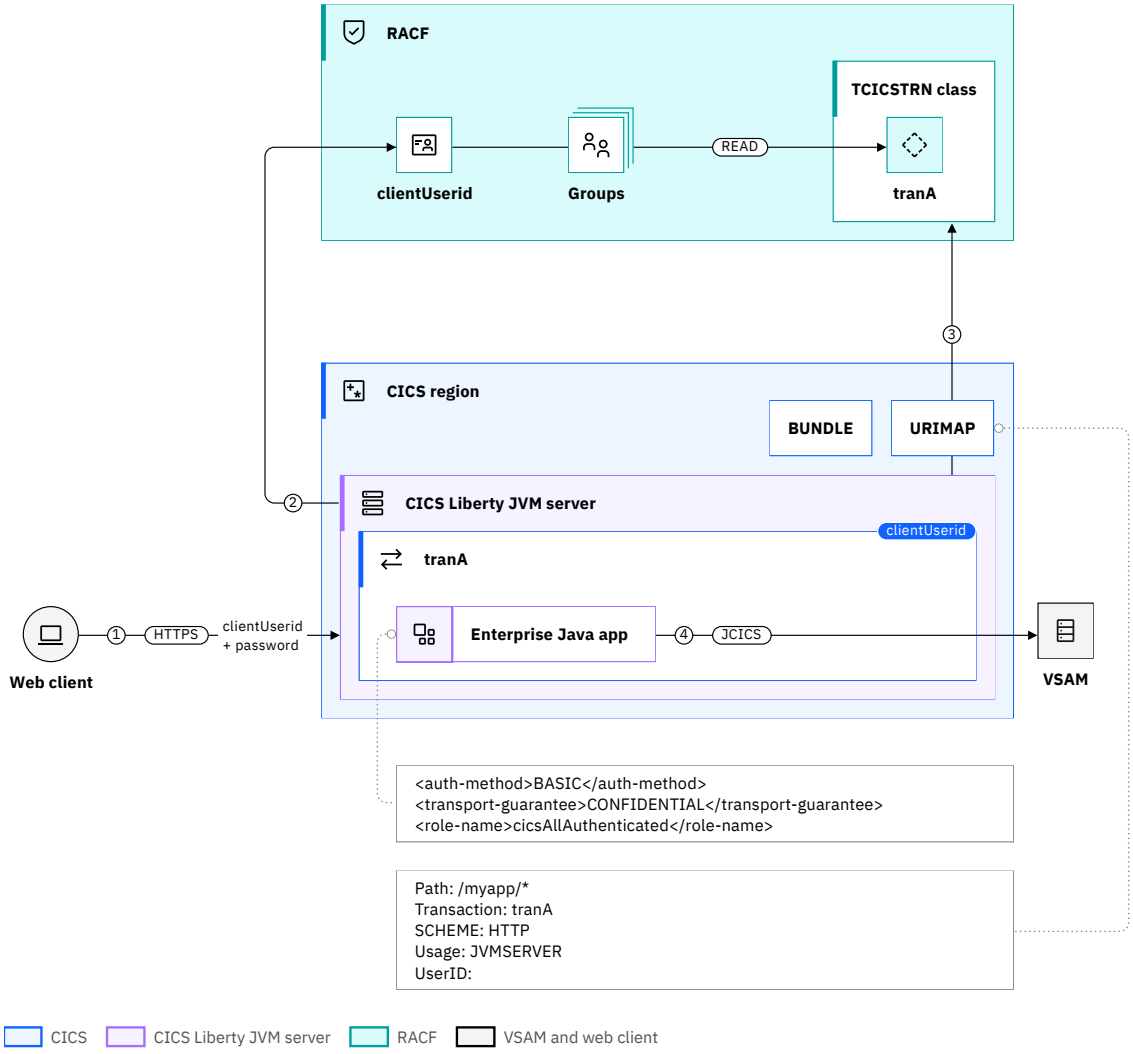


Figure 64. Securing a Liberty web application with basic authentication and CICS transaction security

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. CICS calls RACF to verify that the *clientUserid* has READ authority to the profile *tranA* in the TCICSTRN class or its corresponding resource group 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 CICS Liberty

Configuration example: Securing a 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 user 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 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 role *myRole* 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 65 on page 189 shows an overview of the scenario.

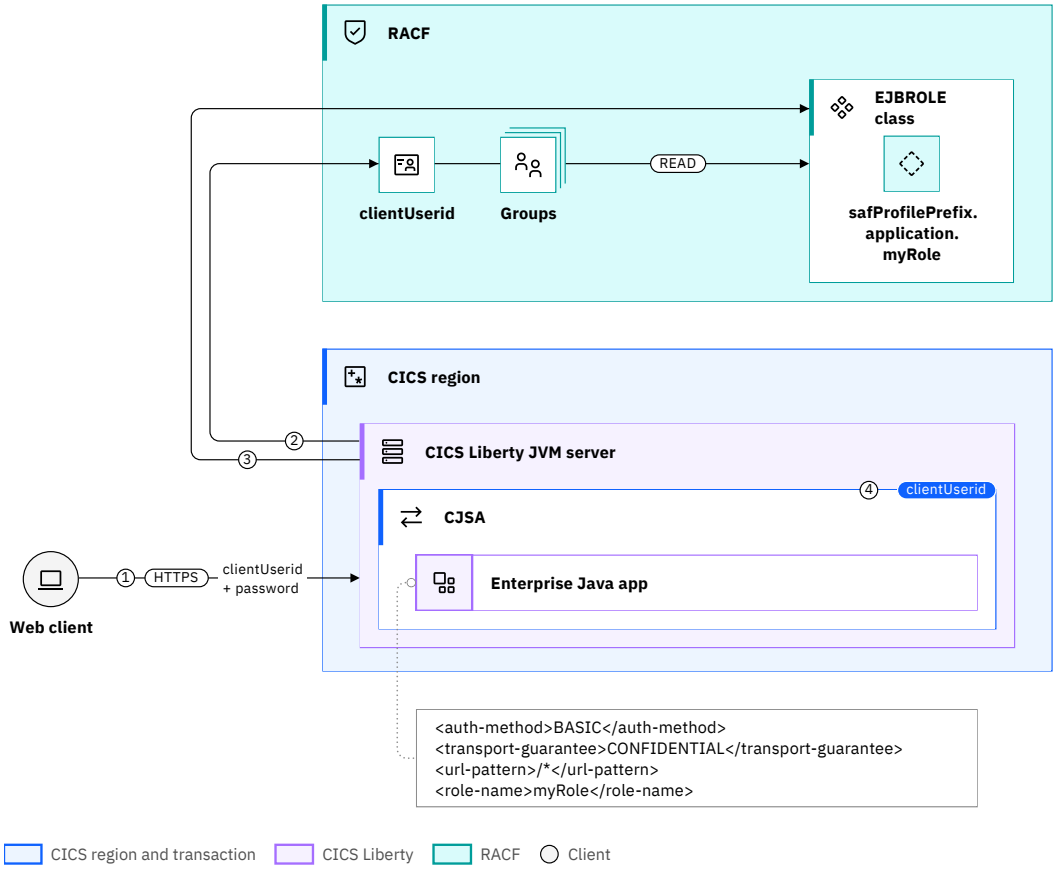


Figure 65. Securing a Liberty web application with basic authentication and SAF role authorization

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 CICS Liberty

Configuration example: Securing a 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 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 user 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 authorization to check that the *clientUserid* is authorized to run the web application.

This task assumes the following definitions:

- *clientUserid* is the user ID that is used to authenticate.
- *myRole* is the Java 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 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 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 a JWT and CICS transaction security.

Figure 66 on page 192 shows an overview of the scenario.

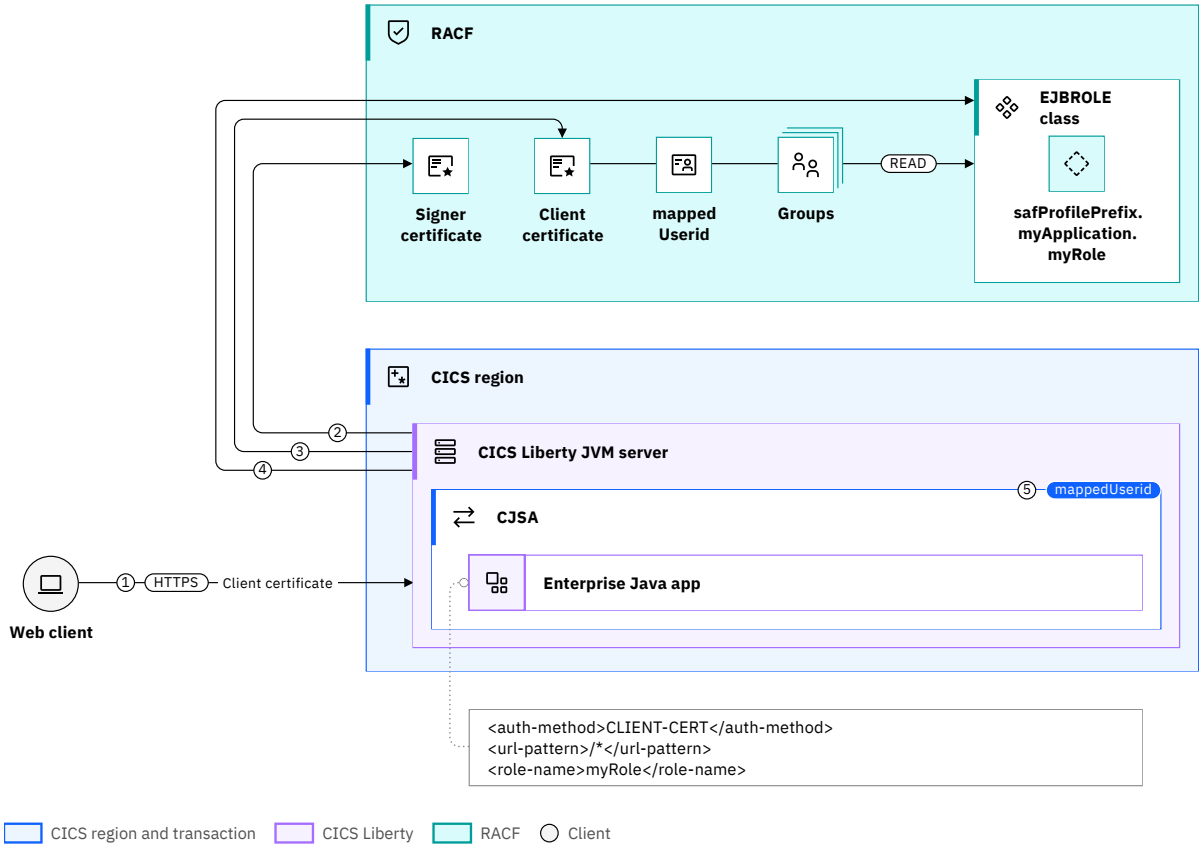


Figure 66. Securing a Liberty web application with TLS client authentication and SAF role authorization

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 CICS Liberty

Configuration example: Securing a 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:

   ```
   SETROPTS 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:

   ```
   SETROPTS 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 CJSA 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 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 67 on page 195 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.

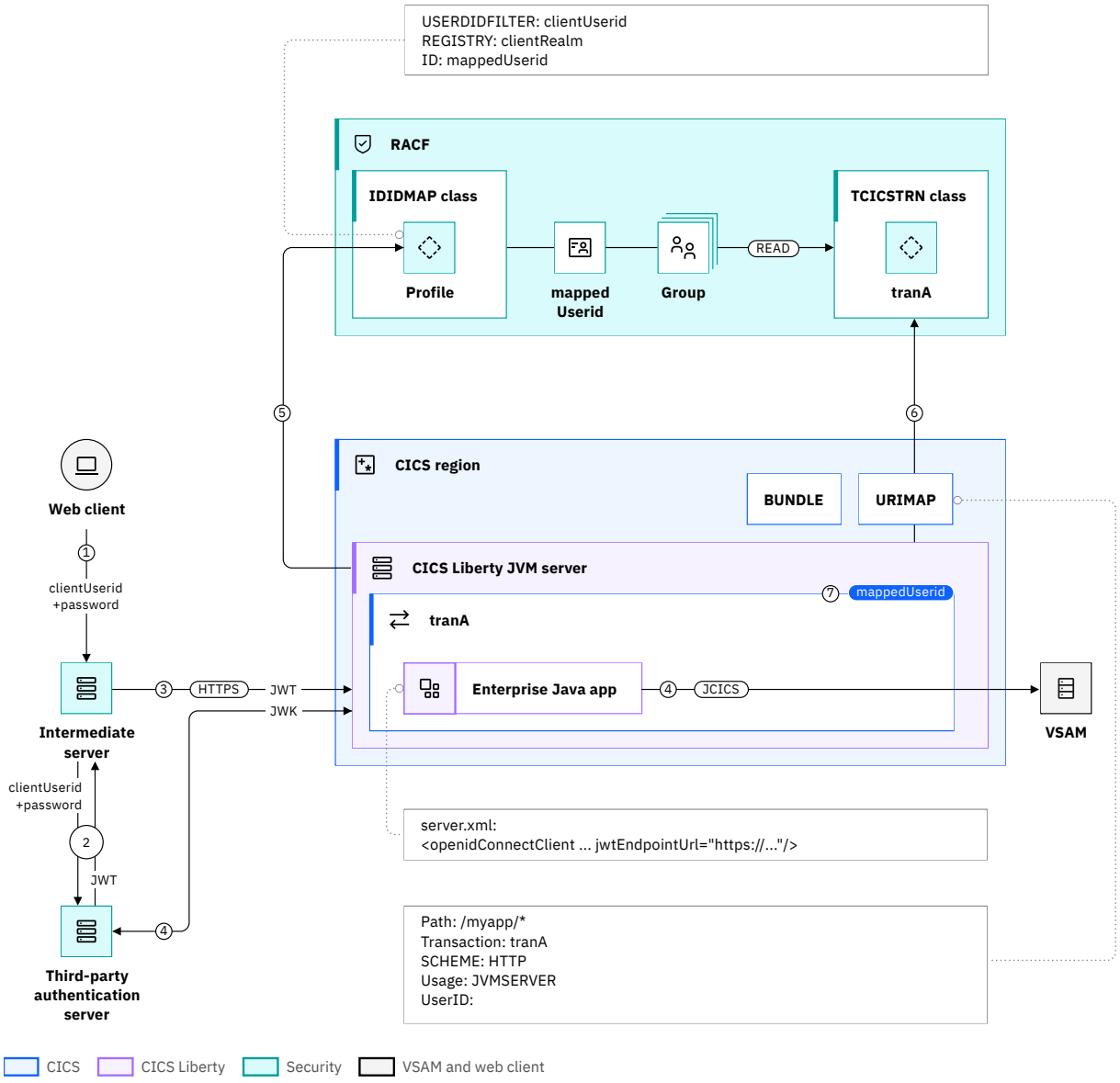


Figure 67. Securing a Liberty web application with a JWT and CICS transaction security

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* has READ access to profile *tranA* in the TCICSTRN class or its corresponding resource group 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 CICS Liberty

Configuration example: Securing a 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 199.

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 USERDIDFILTER(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:

   ```
   SETROPTS 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="JWTTrustStore"
                  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 68 on page 200 shows a high-level view of how CICS and Liberty security mechanisms work together to address the security principles for this scenario.

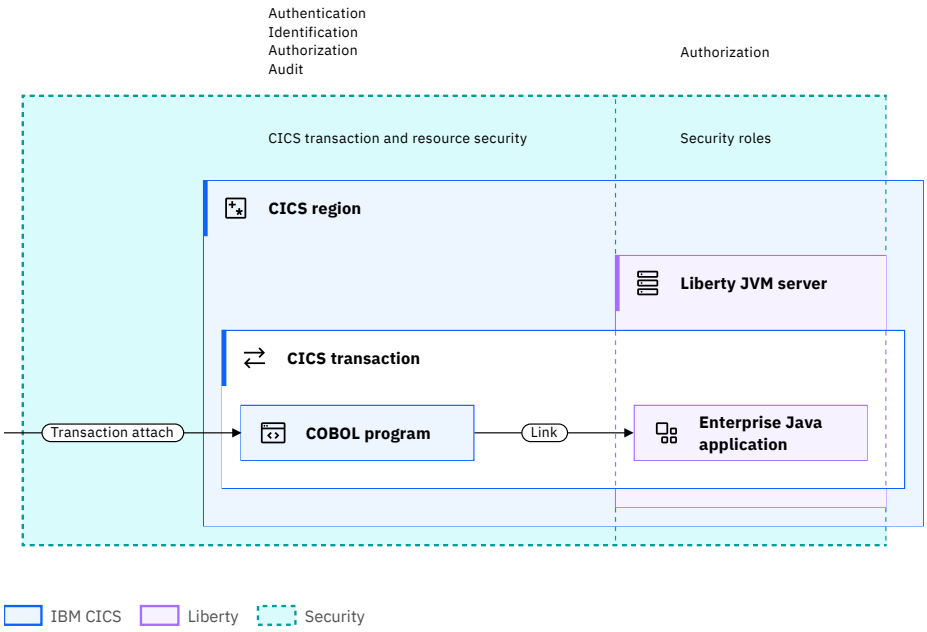


Figure 68. Security for Link to Liberty applications

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 security 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.

Design example: Securing a Link to Liberty Java application with CICS transaction security

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.

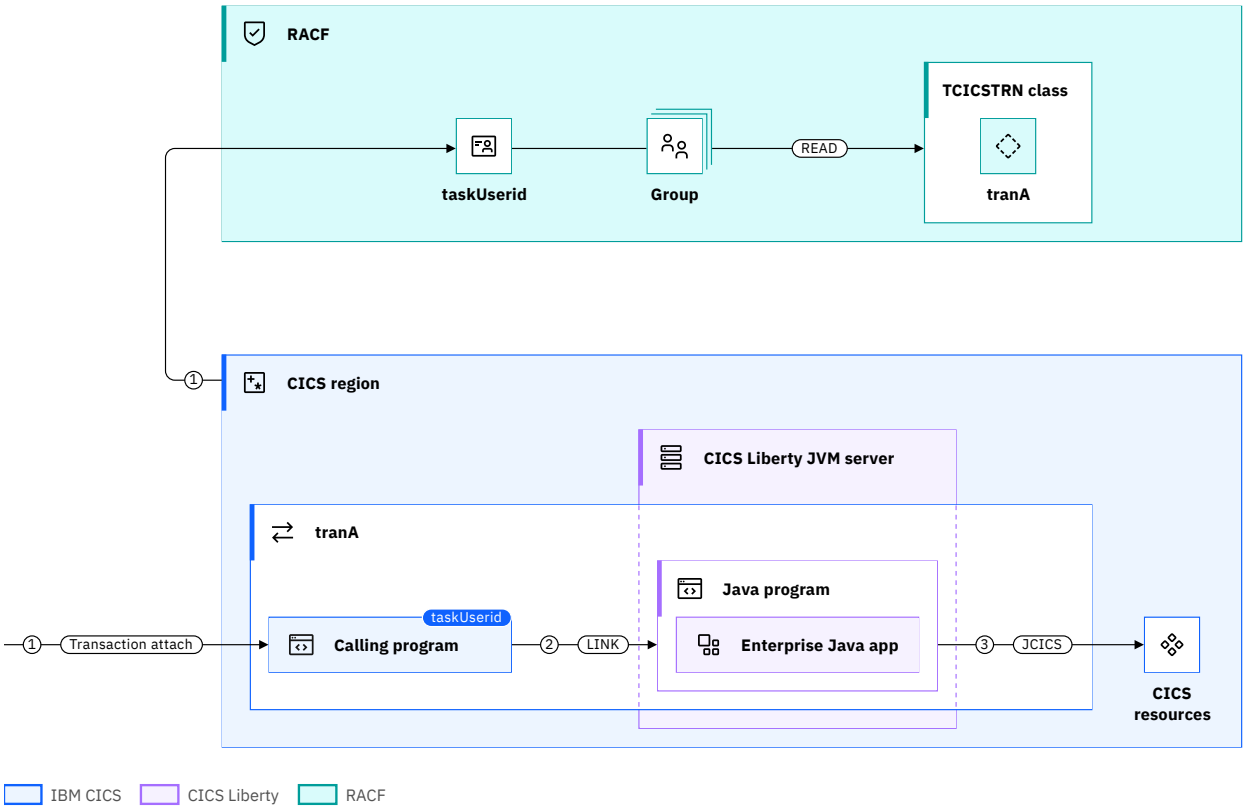


Figure 69. Securing a Link to Liberty Java application with CICS transaction security

For more information about configuring this scenario, see the configuration task Configuration example: Securing a Link to Liberty Java application with CICS transaction security.

Figure 69 on page 201 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 *tranA* profile in the TCICSTRN class that protects the transaction ID.
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.

Important: In this scenario, the Java security subject is set to the Liberty unauthenticated user ID since the JVM property **com.ibm.cics.jvmserver.wlp.security.subject.create** is set to **false**. The Java security subject is not used in this scenario for any authorization checks

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 CICS Liberty

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 RACF profile for the *tranA* transaction and grant the authenticated user ID read access to the profile.

```
RDEFINE TCICSTRN tranA UACC(NONE)
PERMIT tranA CLASS(TCICSTRN) ID(taskUserid) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

2. 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 TCICSTRN profile names need to be prefixed with this value.
3. 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
   ```
4. Restart the JVM server.
5. 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
   ```
6. Deploy the Java application by using a CICS bundle resource (see Deploying a CICS bundle in the CICS Explorer product documentation).
7. 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.

Configuring security for CICS Liberty

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 the example.

Design examples for CICS Liberty are described here:

- Design example: Securing a CICS Liberty web application with basic authentication and CICS transaction security
- Design example: Securing a web application with basic authentication and SAF role authorization
- Design example: Securing a web application with a TLS client certificate and SAF role authorization
- Design example: Securing a web application with a JWT and CICS transaction security

The Design Example for a Java application invoked using Link to Liberty is described here: Design example: Securing a Link to Liberty Java application with CICS transaction security

For common configuration tasks and additional information refer to the following topics.

Configuring the angel process

You can use z/OS authorized services from CICS Liberty JVM servers. These services include the SAF user registry and SAF 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 your 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.
- 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 a named angel process.

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 user registry and SAF 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.ZOSAIO 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 the 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 through MODIFY commands by looking at MODIFY commands that are supported by the 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 (ZOSAIO) 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.ZOSAIO 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 ZOSAIO 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.

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 19 on page 208.

Table 19. RACF profile table for CICS Liberty security

Class	Profile	Required for	Access for CICS region user ID 1	Access for unauthenticated user ID 2	Access for authenticated user ID 3
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		

Table 19. RACF profile table for CICS Liberty security (continued)

Class	Profile	Required for	Access for CICS region user ID 1	Access for unauthenticated user ID 2	Access for authenticated user ID 3
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		
SERVER	BBG.AUTHMOD.BBGZSAFM.ZOSAIO	Angel process registration at Liberty server startup	READ		
SERVER	BBG.SECPFX.BBGZDFLT 4	Authentication or authorization	READ		
APPL	BBGZDFLT 4	Authentication or authorization		READ	READ
EJBROLE	BBGZDFLT.<resource>.<role> 4 5	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 user registry follow the steps in Configuring Liberty to use a SAF user registry.
- LDAP, to configure the LDAP user 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 SAFCRED 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 SAFCRED 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 user 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 user 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 user registry and to authenticate users with their LDAP credentials. Once the user is authenticated, a Java security subject is created with the LDAP credentials. Additionally, the LDAP credentials can be mapped to a RACF user ID with the use of RACF distributed identity filters. You can use the CICS distributed identity mapping feature to set up distributed identity mapping. The mapped RACF user ID can then be used to authorize access to web applications that use CICS transaction security or that use application role security.

Before you begin

You must complete several other tasks:

- Enable an LDAP registry.
- Complete the task Configuring and starting the Liberty angel process for distributed identity mapping.
- Complete the task Connecting a Liberty JVM server to the angel process for distributed identity mapping.

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.
- The RACF user ID that the distributed identity is mapped to. This user ID must exist in the RACF registry and have an OMVS segment. The RACF user ID that is used to authenticate must exist and have an OMVS segment.

You must have:

- Authorization to issue RACF RACMAP command.
- 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 an LDAP user registry and optionally a distributed identity filter. In this example, the common name of the distributed identity *JeanLeclerc* is used to authenticate with the Liberty server. The distributed identity is mapped to the RACF user ID *mappedUserid* that is used for running the CICS task.

This task assumes the following definitions:

- The distinguish name of the distributed identity is *cn=JeanLeclerc,ou=employees,o=ibm,c=fr.*
- The distributed identity is mapped to the RACF user ID *mappedUserid*.
- The distributed user registry realm is *registryRealm*.
- The profile prefix that is used by the CICS Liberty server is *safProfilePrefix*.
- The CICS region user ID is *regionUserid*.

Procedure

1. Add the **`ldapRegistry-3.0`** and **`cicsts:security-1.0`** features to the `server.xml` file.
2. Configure the Liberty server to authenticate the user with an LDAP user registry by defining a connection to the LDAP server in the `server.xml`, 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>
```

Full details on configuring LDAP user registries with Liberty are available in Configuring LDAP user registries in Liberty.

The following steps are optional. They indicate how to configure distributed identity mapping by configuring the server, defining the distributed identity filter in RACF and allowing the mapped RACF user ID to authenticate with the server.

These steps are necessary for:

- Associating a mapped RACF user ID to the CICS task.
- Role authorization is enforced with RACF EJBROLE.

If the distributed identity mapping is not used, then the CICS task runs with the RACF user ID defined in the URIMAP (or the CICS default user ID) and the Enterprise Java role authorization is done based on application bindings with LDAP entries.

3. Add the cicsts:distributedIdentity-1.0 feature to the `featureManager`. This feature enables `safAuthorization` and sets mapDistributedIdentities to true, without having you to add the configuration elements.

   Alternatively, if you don't want to use SAF authorization, or if you need to specify the profile prefix, manually add the **<safCredentials>** element to the `server.xml` instead:

   ```
   <safCredentials profilePrefix="safProfilePrefix" mapDistributedIdentities="true"/>
   ```

   This configuration sets the profile prefix to *safProfilePrefix* (default value is BBGZDFLT) and configures the use of distributed identity filters by enabling mapDistributedIdentities.

   You can 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.

   Note that if you want to use SAF authorization and the **cicsts:distributedIdentity-1.0** feature has not been added, you need to manually add the **safAuthorization** configuration element.
4. Activate the RACF IDIDMAP class. Enter the following RACF command:

   ```
   SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)
   ```
5. Define a distributed identity filter in RACF to map the distributed user ID from the basic authentication to a RACF user ID.

   For example, enter the following command:

   ```
   RACMAP ID(mappedUserid) MAP USERDIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))
   REGISTRY(NAME('registryRealm')) WITHLABEL('Mapping Jean Leclerc')
   ```

   In this example, the following values are used:

   - cn=JeanLeclerc,ou=employees,o=ibm,c=fr is the full distinguished name of the distributed identity.
   - mappedUserid is the RACF user ID to which the distributed identity is to be mapped.
   - REGISTRY(NAME('*registryRealm*') must match the LDAP registry realm name. Alternatively, to match any realm, specify the value *.

   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 a subset of 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.
6. Refresh the RACF IDIDMAP class.

   For the changes to take effect, enter the following RACF command:

   ```
   SETROPTS RACLIST(IDIDMAP) REFRESH
   ```

7. 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 *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)
   ```

8. 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 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 as the CICS task user ID.

In the case that the distributed identity filter is not configured, the RACF user ID running the CICS task is the user ID set in the matching URIMAP resource, or if that ID is not set, the CICS default user ID.

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.

You might want to configure authorization with CICS transaction security, see Configuring CICS transaction security for a Liberty JVM server.

If the application uses role-based authorization, it can be enforced by using either RACF EJBROLE, see Configuring SAF authorization with an EJBRole, or application bindings in the server configuration file, see Configuring authorization for applications in Liberty.

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 CJSA 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.

You need to know the claims that are present in the JWT.

You must have:

- An X.509 certificate that contains the public key for JWT signature validation. Consult the security administrator of the JWT issuer for how to obtain a suitable X.509 certificate.
- 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:

- The JWT is sent to CICS in an HTTP Authorization request header field.
- The RS256 public/private algorithm is used to sign the JWT.
- The identity in the JWT **`subject`** claim is a distributed identity.
- RACF is used for the user registry, storing certificates and for authorizing access to the CICS transaction.
- The 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.

Figure 70 on page 217 shows an example JWT that is used for authentication.

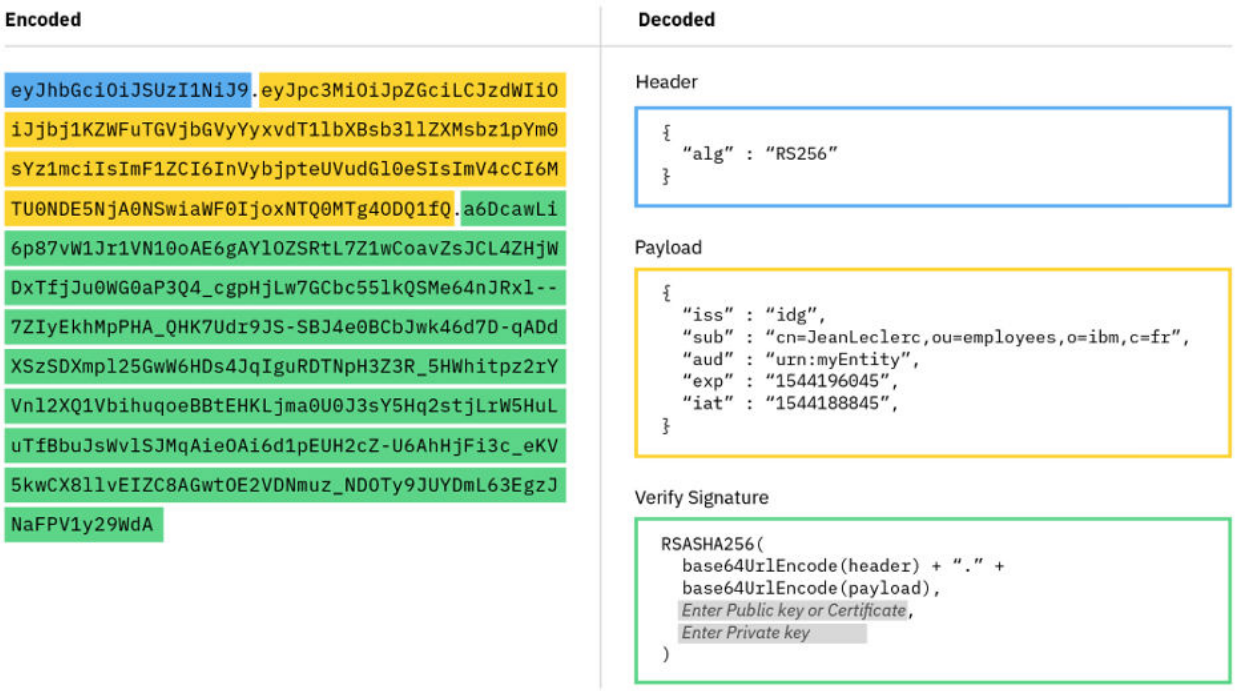


Figure 70. Example JWT used for authentication

The *header* contains the algorithm that is used, such as HMAC SHA256 or RSA SHA256, and is Base64Url encoded to form the first part of the JWT.

The *payload* contains the claims. The payload contains a set of predefined 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 sub claim is used to create the OpenID Connect user subject.

In the previous example:

- 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 220.

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 USERDIDFILTER(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:

   ```
   SETROPTS 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).

Use the Enterprise Java Security API with a database identity store

Authenticate a user with a database identity store by using the Enterprise Java Security API then map the database user ID to a RACF user ID. Finally, use the RACF user ID to run the CICS task.

Before you begin

You must be familiar with the information in Enterprise Java security API.

You need to know:

- The user ID and password that is used to authenticate.
- The RACF user ID to which the database user ID is mapped. The RACF user ID must exist and have an OMVS segment.

You must have write access to the `server.xml` configuration file.

About this task

In this task, you

- Configure a database identity store.
- Use basic authentication with the identity store.
- Map the database user ID to a RACF user ID.
- Use the RACF user ID to run the CICS task.

Procedure

1. Add the `appSecurity-3.0` feature to `server.xml` before you start the server.
2. Configure the application to be an *explicit bean archive*, by adding an empty `beans.xml` file into the `WEB-INF/` or `META-INF/` directory in the application.

   Alternatively, Liberty can be configured to process applications as *implicit bean archives*. For more information about CDI and bean archives, see Context and Dependency Injection (CDI).
3. Create a table in the database and set up `server.xml`.

   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 here: Database setup.

a. a. Add the jdbc-4.2 feature in `server.xml`:

```
<feature>jdbc-4.2</feature>
```

b. b. Configure `jndiName` in `server.xml`, for example:

```
<dataSource id="DefaultDataSource" jndiName="jdbc/sec">
    <jdbcDriver libraryRef="<xxx>"/>
    ...
</dataSource>
```

4. Annotate the web application to use basic authentication by using the Enterprise Java Security API, for example:

```
@BasicAuthenticationMechanismDefinition(realmName="user-realm")
@WebServlet("/home") @DeclareRoles({"user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class HomeServlet extends HttpServlet {
    ...
}
```

5. Determine the RACF user ID to be used to run the CICS task.

   To run the CICS task with a RACF user ID mapped from the database identity store, take the following steps:

   a. Configure SAF in `server.xml` by setting the following SAF elements.

```
<safCredentials mapDistributedIdentities="true" profilePrefix="<xxx>"/>
<safAuthorization id="saf"/>
<safRoleMapperprofilePattern="<xxx>.%resource%.%role%" toUpperCase="false"/>
```

   b. Issue the RACMAP command. The general RACMAP command of mapping a distributed user ID to a SAF user ID is in the format of:

```
RACMAP ID(userid) MAP
   WITHLABEL('label-name')
   USERDIDFILTER(NAME('distributed-identity-user-name'))
   REGISTRY(NAME('distributed-identity-registry-name'))
```

   Use `"defaultRealm"` in `REGISTRY(NAME('<nnn>'))`, and use `"<username_in_DBIS>"` in `USERDIDFILTER(NAME('<nnn>'))`, for example:

```
RACMAP ID(userid) MAP
   WITHLABEL('authorisedUser:userid')
   USERDIDFILTER(NAME('authorisedUser'))
   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>
```

The security role `"cicsAllAuthenticated"` takes precedence over the group name that is stored in the database identity store and an HTTP 403 error occurs. In this instance, you have two options:

- Deploy your database identity store application with a direct `<application>` element in `server.xml`.

- Deploy within a CICS bundle, but use safAuthorization to bypass the CICS-generated `<application-bnd>`, which overrides the group information that is stored in the Custom Identity Store.

Results

You successfully configured the database identity store and used the Enterprise Java Security API to perform basic authentication.

Alternative configuration that uses CICS default user ID

If you do not want to push the database identity onto the CICS task, you can remove the default **`safRegistry`** setting in `server.xml`. The CICS task then runs under the default CICS user ID.

Use the Enterprise Java Security API with a custom identity store

Authenticate a user with a custom identity store by using the Enterprise Java security API then map the database user ID to a RACF user ID. Finally, use the RACF user ID to run the CICS task.

Before you begin

You must be familiar with the information in Enterprise Java security API.

You need to know:

- The user ID and password that is used to authenticate.
- The RACF user ID to which the database user ID is mapped. The RACF user ID must exist and have an OMVS segment.

You must have write access to the `server.xml` configuration file.

About this task

In this task, you

- Configure your own identity store.
- Obtain user information.
- Map the custom user ID to a RACF user ID.
- Use the RACF user ID to run the CICS task.

Procedure

1. Add the `appSecurity-3.0` feature to `server.xml` before you start the server.
2. Ensure that CDI annotation file scanning is enabled. CICS® disables it by default in `server.xml`.

   To ensure that CDI annotation file scanning is enabled, you can check that the following line is not present in `server.xml`: `<cdi12 enableImplicitBeanArchives="false"/>`.
3. 3. Create Java™ classes to process the custom identity store logic and build them into a WAR file.

   a. 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("authorisedUser", "tomtom")) {
            return new CredentialValidationResult("authorisedUser",
                new HashSet<String>(asList("user")));
        }
        return INVALID_RESULT;
    }
}
```

b. 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, HttpMessageContext 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();
        }
    }
}
```

c. Create a servlet.

```
@WebServlet("/home")
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
    public class Servlet extends HttpServlet {...}
```

4. Determine the RACF user ID to be used to run the CICS task.

   To run the CICS task with a RACF user ID mapped from the database identity store, take the following steps:

   a. Configure SAF in `server.xml` by setting the following SAF elements.

```
<safCredentials mapDistributedIdentities="true" profilePrefix="<xxx>"/>
<safAuthorization id="saf"/>
<safRoleMapperprofilePattern="<xxx>.%resource%.%role%" toUpperCase="false"/>
```

   b. Issue the RACMAP command. The general RACMAP command of mapping a distributed userid to a SAF userid is in this format:

```
RACMAP ID(userid)
MAP
WITHLABEL('label-name')
USERDIDFILTER(NAME('distributed-identity-user-name'))
REGISTRY(NAME('distributed-identity-registry-name'))
```

   Use "`defaultRealm`" in `REGISTRY(NAME('<nnn>'))`, and use "`<username_in_DBIS>`" in `USERDIDFILTER(NAME('<nnn>'))`, for example:

```
RACMAP ID(JATM12) MAP WITHLABEL('authorisedUser:JATM12')
USERDIDFILTER(NAME('authorisedUser'))
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>
```

The security role `"cicsAllAuthenticated"` takes precedence over the group name that is stored in the database identity store and an HTTP 403 error occurs. In this instance, you have two options:

- Deploy your database identity store application with a direct `<application>` element in `server.xml`.

- Deploy within a CICS bundle, but use safAuthorization to bypass the CICS-generated <application-bnd>, which overrides the group information that is stored in the Custom Identity Store.

Results

You successfully configured a custom identity store and used the Enterprise Java Security API to perform a custom authentication.

Alternative configuration that uses CICS default user ID

If you do not want to push the custom identity onto the CICS task, you can remove the default **safRegistry** setting in server.xml. The CICS task then runs under the default CICS userid.

Configuration 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 CICS transaction ID for running the web application request:

   ```
   DEFINE TRANSACTION(tranA) GROUP(groupA) PROGRAM(DFHSJTHP)
   ```

   You receive a warning when you define this transaction because the value of the PROGRAM attribute started 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)
   ```

   In this example, `/mycompany/myapp/*` is the path component of the URI to which the URIMAP definition applies.

   Important: URIMAPs that specify SCHEME(HTTP) are applied to HTTP and HTTPS requests. If you specify SCHEME(HTTPS), the URIMAP is applied 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 *taskUserid* read access:

   ```
   RDEFINE TCICSTRN tranA UACC(NONE)
   PERMIT tranA CLASS(TCICSTRN) 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 TCICSTRN profile names need to be prefixed with this value.
5. Optional: If you are using SAF 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 change 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 Enterprise Java 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 change 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.

When 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 CJSA by default). For more information, see Checking that a CICS security configuration example is working by using the SRR.

Configuring SAF authorization with an EJBROLE

Authorize access to an application that runs in a Liberty JVM server by using 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 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 authorization, you can continue to use CICS bundles to lifecycle your web applications. The `<application-bnd>` is ignored by Liberty and the SAF authorization takes precedence.

Important: When SAF 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 229.

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 authorization 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"/>
```

   The `<safRoleMapper>` determines the name of the RACF EJBROLE profiles that are used to validate the user authorization.

   The EJBROLE profile name is case-sensitive by default. If you don't want it to be case-sensitive configure `toUpperCase="true"`.

   The default `profilePattern` 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 is started.
   - `%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 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`).
- Liberty authorized SAF services must be available, the angel process must be up and running, and the Liberty server must be connected to it.

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 user that is accessing the application has a RACF user ID *clientUserid*.
- 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.*clientUserid* profiles in the SURROGATE class:

     ```
     PERMIT BBG.SYNC.safProfilePrefix ID(regionUserid) ACCESS(READ) CLASS(FACILITY)
     ```

     ```
     PERMIT BBG.SYNC.safProfilePrefix ID(regionUserid) ACCESS(READ) CLASS(SURROGAT)
     ```

     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 *clientUserid* is synchronized with the operating system (OS) thread identity and used to authorize access to resources outside of CICS control.

Configuring confidentiality and integrity

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 these RACDCERT commands used to create a RACF key ring and certificates:
  - ADD

- ADDRING
- CONNECT
- EXPORT
- GENCERT
- LIST
- LISTRING

For more information about the RACDCERT commands and the authorizations that are required, see RACDCERT (Manage RACF digital certificates).

You must have:

- 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:

- The CA certificate that is used to sign the server certificate used by the Liberty JVM server.
- A personal RACF certificate signed by the CA certificate, which is used by the Liberty JVM server to identify itself over TLS connections.
- A RACF key ring to act as the server's keystore.
- A Java KeyStore (JKS) for a client to use as its truststore so that it can trust the certificate that is presented by the Liberty JVM server on the TLS connection.

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, keyrings, and JKS files that are used in this task are described in Figure 71 on page 231.

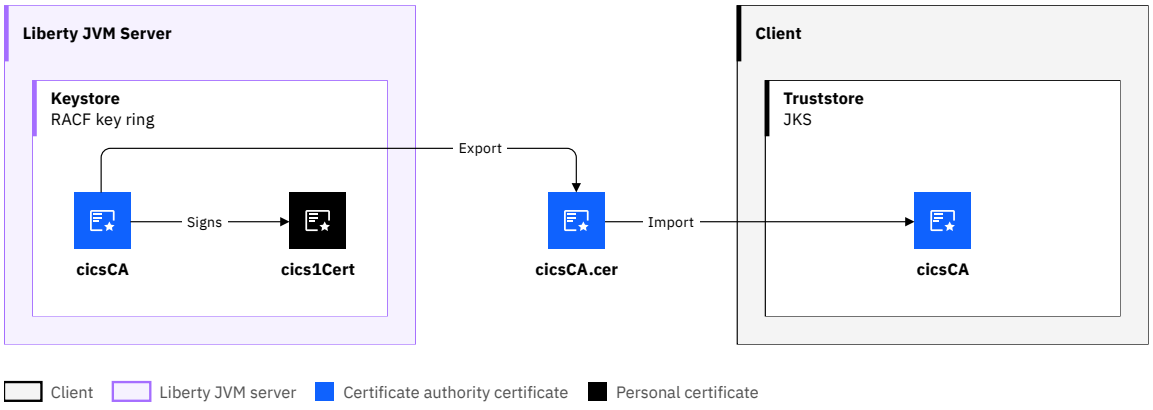


Figure 71. Certificates that are used for TLS connection

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 that is signed by a certificate authority (CA) certificate.
- 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

In the following steps, you configure the TLS connection between the Liberty JVM server (steps 1 - 5) and the client (steps 6 - 9). Finally, you configure the Liberty JVM server to use the RACF key ring (steps 10 - 14).

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(DATE(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.

   b. 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.

3. Create a personal certificate, signed by the CA certificate, for the Liberty JVM server.

   a. 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.

   b. 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.

4. Confirm that the key ring and certificates were created correctly.

   a. 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
      ```

   b. 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.

In the following steps, you configure the TLS connection on 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.

In the following steps, you 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="defaultHttpEndpoint" host="*"
        httpPort="9080" httpsPort="9443"/>
    ```

12. Configure the SSL configuration to be used by the Liberty JVM server.

    Create an `ssl` repertoire element in the `server.xml` configuration file, with default values for the `id` and `keyStoreRef` attributes. For example,

    ```
    <ssl id="defaultSSLConfig"
        keyStoreRef="defaultKeyStore" />
    ```

    This example uses the default id and keyStoreRef attributes. If you use different id and keyStoreRef attributes, you also need to associate the customized ssl element with the appropriate httpEndpoint element by configuring the sslOptions subelement. For more information about the `ssl` repertoire 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. For example,

    ```
    <keyStore id="defaultKeyStore"
        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:

    - `defaultKeyStore` 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:

    ```
    CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now listening for requests on host * (IPv6) port 9443
    ```

    The message uses the following values:

    - `defaultHttpEndpoint-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 also be familiar with the information in How it works: Integrity and confidentiality in CICS.

Before you begin this task, you must complete the prerequisite task Configuring TLS for a Liberty JVM server by using RACF.

You must have authorization to Issue these RACDCERT commands used to create a RACF key ring and certificates:

- ADD

- ADDRING
- CONNECT
- EXPORT
- GENCERT
- LIST
- LISTRING

For more information about the RACDCERT commands and the authorizations that are required, see RACDCERT (Manage RACF digital certificates).

You must have:

- 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.
- Write access to the `server.xml` configuration file.

About this task

This task describes extra 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 additional configuration that is required to use the client's personal certificate to authenticate with a Liberty JVM server, but is a prerequisite to that task. For more information about configuring a client certificate to authenticate with a Liberty JVM server, see Configuration example: Securing a web application with a JWT and CICS transaction security.

This configuration creates these artifacts:

- A certificate authority (CA) certificate that is used to sign the client certificate.
- A CA signed personal certificate for the client to identify itself on TLS connections.
- A JKS to act as the client's keystore.

The following diagram shows certificates that are used in this task:

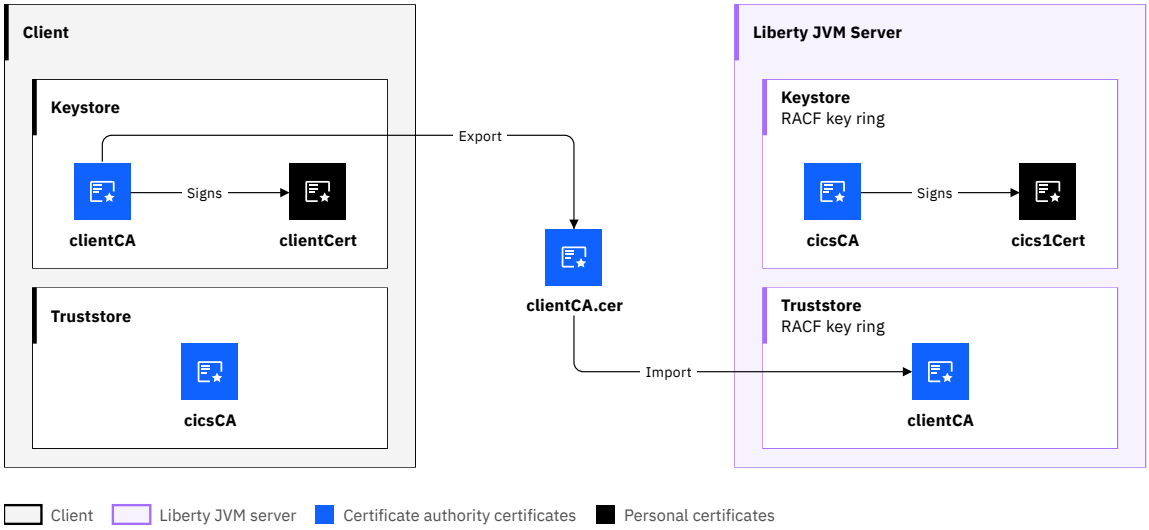


Figure 72. Certificates used for a client authenticated TLS connection

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.

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.

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 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

In the next steps, you configure TLS client authentication for the client.

1. Create a self-signed certificate for the client.

   Generate a self-signed RSA key pair for the client and add them to a JKS file, which acts as the client's keystore.

   Enter this command from the USS directory that you created in step six of How to configure TLS with RACF key rings:

   ```
   keytool -genkeypair
       -alias clientCert
       -dname "CN=myClient.host.com, O=IBM, C=US"
       -keyalg RSA
       -keypass passw0rd
       -keysize 2048
       -keystore clientKey.jks
       -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.jks` is the name of the JKS file to be dynamically created to act as the client's keystore.

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 CA root certificate and optionally, an intermediate certificate.

   - Create a certificate signing request (CSR) for the personal certificate.

Enter this command from the USS directory that you created in step six of How to configure TLS with RACF key rings:

```
keytool -certreq
    -alias clientCert
    -keystore clientKey.jks
    -file clientCert.csr
    -storepass passw0rd
```

The command uses these values:

- `clientCert` is the alias of the personal certificate to be signed.
- `clientKey.jks` is the name of the JKS 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 CA root certificate, the signed client's personal certificate, and optionally, an intermediate certificate.
- Import the CA root certificate and if present, the intermediate certificate, into the client's JKS keystore.

  Enter this command from the USS directory that you created in step six of How to configure TLS with RACF key rings:

```
keytool -importcert
    -file clientCA.cer
    -alias clientCA
    -keypass passw0rd
    -keystore clientKey.jks
    -storepass passw0rd
    -storetype jks
```

  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 CA root certificate to be imported.
  - `clientCA.cer` is the name of the certificate file that is returned by the certificate authority. It contains the CA root certificate to be imported.
  - `clientKey.jks` is the name of the JKS file into which the certificate is to be imported. This file is the client's keystore.
- Import the signed client's personal certificate.

  Enter this command from the USS directory that you created in Configuring TLS for a Liberty JVM server using RACF.

```
keytool -importcert
    -file clientCertSigned.cer
    -alias clientCert
    -keypass passw0rd
    -keystore clientTrust.jks
    -storepass passw0rd
    -storetype jks
```

  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 is returned by the certificate authority, and contains the signed personal certificate to be imported.

`clientKey.jks` is the name of the JKS 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 the USS directory that you created in step six of Configuring TLS for a Liberty JVM server using RACF to list the contents of the client's keystore.

   ```
   keytool -list -v -keystore clientKey.jks
   ```

   The expected response contains entries for the signed personal certificate with an `Entry type` of `PrivateKeyEntry` and for the CA root certificate, an entry with `Entry type` of `trustedCertEntry`.

4. Export the client's CA certificate (public key) to a certificate file.

   Enter this command from the USS directory that you created in Configuring TLS for a Liberty JVM server using RACF:

   ```
   keytool -exportcert
       -alias clientCA
       -file clientCA.cer
       -keystore clientKey.jks
       -storepass passw0rd
       -storetype jks
   ```

   The command uses these values:

   - `clientCA` is the alias of the CA certificate to be exported.
   - `clientCA.cer` is the name of the certificate file to be created.

   `clientKey.jks` is the name of the JKS file from which the certificate is to be exported.

5. Make the client's keystore JKS file available to the client.

   The client's keystore that is created in this task is called `clientKey.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.

In these steps, you configure TLS client authentication for the Liberty JVM server.

6. Connect the client's CA certificate to the Liberty JVM server's RACF key ring.

   a. Transfer the exported client's CA certificate in binary format to the z/OS LPAR where the Liberty JVM server is hosted, as a z/OS sequential file.

      Name the created z/OS sequential file as `CICS.CERTS.CLIENTCA`.

      The transfer method that you use depends on the operating system on which you ran the `keytool` command. You might be able to FTP to the z/OS LPAR. You can then put the z/OS sequential file directly. Alternatively you might need to first FTP the file to a USS directory and then copy that USS file to be a z/OS sequential file.

      ```
      cp /u/myuser/cicsJKS/clientCA.cer "//'CICS.CERTS.CLIENTCA'"
      ```

   b. Import the client's CA certificate into RACF.

      Enter this command:

      ```
      RACDCERT ID(CICS1) ADD('CICS.CERTS.CLIENTCA') WITHLABEL('clientCA') TRUST
      ```

   c. Connect (add) the client's CA certificate as a trusted (CERTAUTH) certificate to the RACF key ring used by the Liberty JVM server:

Enter this command:

```
RACDCERT ID(CICS1) CONNECT(RING(Keyring.CICS1) LABEL('clientCA') USAGE(CERTAUTH))
```

7. Confirm that the client's CA certificate is 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                            ID(CICS1)      CERTAUTH     NO
   ```

8. Edit the SSL configuration to be used by the Liberty JVM server.

   Edit the existing `ssl` repertoire element in the `server.xml` configuration file to add a `trustStoreRef` attribute with the default value. Set client authentication by using `clientAuthentication="true"`.

   ```
   <ssl id="defaultSSLConfig"
       keyStoreRef="defaultKeyStore"
       trustStoreRef="defaultTrustStore"
       clientAuthentication="true" />
   ```

   In cases where some clients have a client certificate, and SSL client authentication is not mandatory for every https connection, then the `clientAuthenticationSupported` attribute can be set to true. If `clientAuthentication` is set to true, the Liberty JVM server ignores the setting of `clientAuthenticationSupported`.

9. Create a `keyStore` element in the `server.xml` configuration file for the Liberty JVM server's truststore.

   The `keyStore` element is also used for truststores:

   ```
   <keyStore id="defaultTrustStore"
       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 these values:

   - `defaultTrustStore` 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.
   - `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.

10. 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. Additionally, 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

You might also configure authentication by using the client certificate, and run a request with a RACF user ID mapped from the client certificate, by following the steps in Configuration example: Securing a web application with a JWT and CICS transaction security.

Chapter 15. Security for Java applications

You can secure Java applications to ensure that only authorized users can deploy and install applications, and access those applications from the web or through CICS. You can also use a Java security manager to protect the Java application from performing potentially unsafe actions.

You can add security at different points in the Java application lifecycle:

- Implement security checking for defining and installing Java application resources. Java applications are packaged in CICS bundles, so you must ensure that users who are allowed to install applications in the JVM server can install this type of resource.
- Implement security checking for application users to ensure that only authorized users can access an application.
- Implement security checking for CICS Java tasks that are started using the CICSExecutorService. All such CICS tasks run under the CJSA transaction and the default user ID.
- Implement security restrictions on the Java API by using a Java security manager.

Java applications can run in an OSGi framework or a Liberty server. Liberty is designed to host web applications and includes an OSGi framework. The security configuration for a Liberty server is different, because Liberty has its own security model.

To configure security for OSGi applications, use CICS resource security to authorize which users can manage the lifecycle of the JVMSERVER and the Java applications. Use CICS transaction security to determine who can access the application.

Configuring security for OSGi applications

Use CICS resource security to authorize which users can manage the lifecycle of the JVMSERVER and the Java applications. Use CICS transaction security to determine who can access the application.

Procedure

- Authorize application developers and system administrators to create, view, update, and remove JVMSERVER and BUNDLE resources as appropriate. The JVMSERVER resource controls the availability of the JVM server. The BUNDLE resource is a unit of deployment for the Java application and controls the availability of the application.
- Authorize users to run the application by ensuring the relevant user ID is allowed to attach the transaction under which the application will run.

Results

You have successfully configured security for Java applications that run in an OSGi framework.

Configuring security for a Liberty JVM server

You can use the CICS Liberty security feature to authenticate users and authorize access to web applications through Java Platform, Enterprise Edition roles (Java security roles), providing integration with CICS transaction and resource security. You can also use CICS resource security to authorize the appropriate users to manage the lifecycle of both the JVMSERVER resource and Java web applications that are deployed in a CICS BUNDLE resource. In this topic, authentication verifies the identity of a given user, typically by requiring the user to enter a username and password. Authorization then grants access control permissions based on the identity of the authenticated user.

Before you begin

1. Ensure that the CICS region is configured to use SAF security and is defined with SEC=YES as a system initialization parameter. If CICS security is turned off (SEC=NO), you can still use Liberty security by manually configuring the `server.xml` file as described in "6" on page 245.
2. Authorize application developers and system administrators to create, view, update, and remove JVMSERVER and BUNDLE resources to deploy web applications into a Liberty JVM server.

The JVMSERVER resource controls the availability of the JVM server, and the BUNDLE resource is a unit of deployment for the Java applications and controls the availability of the applications. The default behavior of the CICS TS security feature, `cicsts:security-1.0`, is to use the SAF registry. If you use an LDAP registry, a SAF registry is not created. For more information, see Configuring security for a Liberty JVM server by using an LDAP registry. The basic user registry (which is also used by `quickStartSecurity`) is only suitable for simple security testing. Be aware that if you configure and run with basic user registry and you need to switch to `cicsts:security-1.0`, you need to delete the session tokens.

About this task

This task explains how to configure security for a Liberty JVM server and integrate Liberty security with CICS security. For information about how to configure security for Link to Liberty, see Linking to a Enterprise Java or Spring Boot application from a CICS program. For guidance on configuring security for the JCICSX remoting server, see "Configuring security for remote JCICSX API development" on page 263.

The default transaction ID for running web requests is CJSA. However, you can configure CICS to run web requests under a different transaction ID by using a URIMAP of type JVMSERVER. Typically, you might specify a URIMAP to match the generic context root (URI) of a web application to scope the transaction ID to the set of servlets that make up the application. Or you might choose to run each individual servlet under a different transaction with a more precise URI.

Calls to the JCICSX Liberty JVM server are run under transaction CJXA.

The default user ID for running web requests is the CICS default user ID. If a URIMAP is available and contains a static user ID, it is used in preference to the default user ID. If the web request contains a user ID in its security header, it takes precedence over all other mechanisms.

Tasks starting from Liberty that are not classified as web requests run under the CJSU transaction by default. Although there is no URIMAP style mechanism for these types of tasks, you can override the default transaction ID by using the JVM profile property of `com.ibm.cics.jvmserver.unclassified.tranid` and the default user ID by using the JVM profile property `com.ibm.cics.jvmserver.unclassified.userid`.

Note: The user ID requires permission to attach the specified transaction. For more information, see Transaction security.

Procedure

1. Configure the Liberty angel process to provide authentication and authorization services to the Liberty JVM server, see The Liberty server angel process.

   Tip: If you have a named angel process, you need to configure your Liberty JVM server to connect to it by adding the following line to your JVM profile.

   ```
   -Dcom.ibm.ws.zos.core.angelName=<named_angel>
   ```

2. Optional: Enforce the requirement to connect to the Liberty angel process when the Liberty JVM server is being enabled by adding the following line to your JVM profile:

   ```
   -Dcom.ibm.ws.zos.core.angelRequired=true
   ```

   This option prevents the Liberty JVM server from starting if the angel process is unavailable.

   It instructs CICS to call the Liberty angel check API to verify whether an angel process is available for Liberty JVM server startup.

If the angel process is unavailable, CICS reacts as follows:

- If the Liberty JVM server is being enabled through the CEMT transaction, a message is issued, and the Liberty JVM server is disabled.
- If the Liberty JVM server is being enabled by the **SET JVMSERVER** SPI command or by using the CMCI through the CICS Explorer, a message is issued, and the Liberty JVM server is disabled.
- If the Liberty JVM server is being enabled by the CICS CREATE SPI, by BAS, or from GRPLIST, a message is issued, and CICS will wait 30 seconds before retrying the Liberty angel check API call. If the angel process is unavailable on the fifth attempt, a WTOR message is issued, giving the operator the option to continue waiting or to disable the JVMSERVER resource.

3. Add the `cicsts:security-1.0` feature to the `featureManager` list in the `server.xml`,

```
<featureManager>
   ...
   <feature>cicsts:security-1.0</feature>
</featureManager>
...
```

4. Add the System Authorization Facility (SAF) registry to `server.xml` by using the following example:

```
<safRegistry id="saf" enableFailover="false"/>
```

5. Save the changes to `server.xml`.
6. Optional: Alternatively, if you are autoconfiguring the Liberty JVM server and the **SEC** system initialization parameter is set to YES in the CICS region, the Liberty JVM server is dynamically configured to support Liberty JVM security when the JVM server is restarted. For more information, see Configuring a Liberty JVM server.

   If the **SEC** system initialization parameter is set to NO, you can still use Liberty security for authentication or SSL support. If CICS security is turned off, and you want to use a Liberty security, you must configure the `server.xml` file manually:

   a. Add the `appSecurity-2.0` feature to the `featuremanager` list.
   b. Add a user registry to authenticate users. Liberty security supports SAF, LDAP, and basic user registries. For more information, see Configuring a user registry in Liberty.
   c. Add security-role definitions to authorize access to application resources, see “Authorizing users to run applications in a Liberty JVM server” on page 252.

Results

The web container is automatically configured to use the z/OS Security feature of Liberty. A SAF registry is used for authentication, and Java security roles are respected for authorization. Authorization constraints and security roles govern who can access the application. These are usually defined in the deployment descriptor (`web.xml`) of the application, but might also be defined as security annotations in the source-code. Typically, users and groups are mapped to roles by the applications *<application-bnd>* element in `server.xml`. Alternatively, if the*<safAuthorization>* element is configured in `server.xml`, the mappings are held in SAF (as EJBROLEs in RACF).

What to do next

Note: You can also delegate authentication to another identity by configuring the RunAs specification for Liberty, see Configuring RunAs authentication in Liberty.

- Configure Liberty application security authentication rules; see “Authenticating users in a Liberty JVM server” on page 250.
- Define authorization rules for web applications; see “Authorizing users to run applications in a Liberty JVM server” on page 252 and “Authorization using SAF role mapping” on page 253.
- Modify the Liberty authentication cache.

For more information about using Secure Sockets Layer (SSL), see “Configuring SSL (TLS) for a Liberty JVM server using a Java keystore” on page 267.

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 your multiple Liberty servers. When you include the `cicsts:security-1.0` feature, the CICS Liberty JVM server uses the angel process to call z/OS authorized services such as System Authorization Facility (SAF).

Named angels

A Liberty server can only connect to one angel process at server startup. However, all Liberty servers that are running on a z/OS image can share a single angel process. This is regardless of the level of code that the servers are running or whether they are running in a CICS JVM server. To achieve this, you need to use named angels.

If an angel process is not given a name, it becomes the default angel process. You can have only one default angel process. If you try to create another, it fails to start.

Optionally, you can name an angel process. Named angels allow 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 provides the ability to isolate servers from one another, so that they can run different service levels or be managed independently.

For more information about named angel processes, see Named angel.

Angel version interoperability

All 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. It's recommended that the angel process be upgraded before the Liberty servers that use its services, because it provides back-level support for earlier versions of Liberty servers. This ensures support is available for all authorized services potentially required by the Liberty servers.

Important: Install the latest version of the angel process, regardless of which product it is bundled with. The latest version might be bundled with other IBM software, and might supersede the version that is bundled with CICS.

You can identify the version of Liberty for the angel process and the Liberty JVM server that's running in CICS as shown in “Examples of identifying Liberty versions” on page 249.

Running the angel process started task

1. Locate the JCL procedure for the **started** task in the USSHOME directory, for example: `/usr/lpp/cicsts61/wlp/templates/zos/procs/bbgzangl.jcl`
2. Modify and copy the JCL procedure to a JES procedure library. You can set **ROOT** to the value of USSHOME/`wlp`, for example: `ROOT=/usr/lpp/cicsts61/wlp`
3. Start the angel process. In the following examples, `[.identifier]` indicates an optional identifier that can be up to 8 characters.
   a. To start the angel process without naming it, use the following command:

   ```
   START BBGZANGL[.identifier]
   ```

   b. To start the angel process as a named angel process, code the `NAME` parameter on the operator START command. For example:

   ```
   START BBGZANGL[.identifier],NAME=<named_angel>
   ```

   The angel process name is 1 - 54 characters inclusive, and must use only the following characters: `A-Z 0-9 ! # $ + - / : < > = ? @ [ ] ^ _ ` { } | ~`

Note: A Liberty server can use its own named angel process. One benefit of this isolation is that the angel process can be serviced without affecting any other Liberty server instances on the LPAR. The angel process must be running before the Liberty JVM server starts.

4. Start the Liberty JVM server. By default, the server connects to the unnamed angel process if one is available. To connect to a specific angel process, set the `com.ibm.ws.zos.core.angelName` property, for example:

   ```
   -Dcom.ibm.ws.zos.core.angelName=named_angel
   ```

5. You can specify that CICS checks for the presence of a running angel process before enabling, by setting the `com.ibm.ws.zos.core.angelRequired` property to true. For example:

   ```
   -Dcom.ibm.ws.zos.core.angelRequired=true
   ```

   The server fails if the angel process is not available during startup. Use of this property allows a quicker and cleaner failure.

6. You can specify which z/OS authorized services will be available to the Liberty server by setting the `com.ibm.ws.zos.core.angelRequiredServices` property. The value for this property must be a comma-separated list of valid angel process services. If set then `com.ibm.ws.zos.core.angelRequired=true` must also be set. For example to allow only SAFCRED, PRODMGR and ZOSAIO authorized services specify:

   ```
   -Dcom.ibm.ws.zos.core.angelRequiredServices=SAFCRED,PRODMGR,ZOSAIO
   ```

   This property must be specified with `com.ibm.ws.zos.core.angelRequired property set to true`.

Interacting with the angel process started task

In the following examples, `[.identifier]` indicates an optional identifier that can be up to eight characters.

- Display the Liberty JVM servers that are connected to the angel process use the following console command:

  ```
  MODIFY BBGZANGL[.identifier],DISPLAY,SERVERS,PID
  ```

  A list of job names and process identifiers (PID) 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`.

- Stop the angel process.

  ```
  STOP BBGZANGL[.identifier]
  ```

Note: The Liberty JVM server must be stopped before restarting or applying maintenance to the angel process.

SAF profiles used by the angel process

When a Liberty Server connects to an angel process during server startup, it checks that the server has access to the z/OS authorized services. By default, access checks are performed for all authorized services. You can restrict the Liberty server to check and use only the authorized services it requires, which then makes other authorized services unavailable. You can specify the required authorized services by setting `-Dcom.ibm.ws.zos.core.angelRequiredServices` in your JVM profile. The value for this property must be a comma-separated list of valid angel process services. All service

names must be 8 characters or less and symbols are not valid. This property must be specified with `com.ibm.ws.zos.core.angelRequired` property set to true.

This section describes the SAF profiles to which access is required for CICS processing. For information on the full set of SAF profiles defined by Liberty, refer to Enabling z/OS authorized services on Liberty for z/OS.

- The Liberty JVM server runs under the authority of the CICS region user ID. This user ID must be able to connect to the angel process to use authorized services. The user ID that the angel process runs under needs access to the SAF STARTED profile, for example:

```
RDEFINE STARTED BBGZANGL.* UACC(NONE) STDATA(USER(WLPUSER))
SETROPTS RACLIST(STARTED) REFRESH
```

- For the Liberty JVM server to connect to an angel process, create a profile for the angel (**BBG.ANGEL**, or **BBG.ANGEL.<namedAngelName>** if you are using a named angel process) in the **SERVER** class. Give the CICS region user ID (*cics_region_user*) authority to access it, for example, in RACF:

```
RDEFINE SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

- For a Liberty server to use the z/OS authorized services, create a **SERVER** profile for the authorized module **BBGZSAFM** and give the CICS region user ID (*cics_region_user*) to the profile:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

- Give the Liberty JVM server, under the authority of the CICS region user ID (*cics_region_user*), access to the SAF user registry and SAF authorization services (**SAFCRED**) in the **SERVER** class. For example, in RACF:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

- Create a **SERVER** profile for the **IFAUSAGE** services (**PRODMGR**) and allow the CICS region user ID access to it. This allows the Liberty JVM server to register and unregister from **IFAUSAGE** when the CICS JVM server is enabled and disabled:

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

- Refresh the **SERVER** resource:

```
SETROPTS RACLIST(SERVER) REFRESH
```

The following table summarizes the SAF security profiles that are used by a Liberty server running in a CICS JVM server.

Table 20. SAF profile table for CICS Liberty security

Class	Profile	Required for	CICS region user ID 1	Unauthenticated user ID 2	Authenticated user ID 3
SERVER	BBG.ANGEL	Angel process registration at Liberty server startup	READ		
SERVER	BBG.ANGEL.<namedAngelName>	Angel process registration at Liberty server startup	READ		

Table 20. SAF profile table for CICS Liberty security (continued)

Class	Profile	Required for	CICS region user ID 1	Unauthenticated user ID 2	Authenticated user ID 3
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		
SERVER	BBG.SECPFX.BBGZDFLT 4	Authentication or authorization	READ		
APPL	BBGZDFLT 4	Authentication or authorization		READ	READ
EJBROLE	BBGZDFLT.<resource>.<role> 5	Authentication or 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%"/>`.

For more information, see Process types on z/OS.

Examples of identifying Liberty versions

Example: 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:

```
CWWKB0079I THE ANGEL BUILD LEVEL IS 18.0.0.2 20180619-0654 2018.7.0.0 20180619-0654
```

Example: 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 following message:

```
DFHSJ1405I 08/22/2018 17:04:39 IYK3ZDRI JVMSERVER EYUCMCIJ is running WebSphere Application Server
                        Version 18.0.0.2 Liberty - (18.0.0.2-cl180220180619-0403) process ID
67174497.
```

Example: Identifying both Liberty versions by running scripts

Suppose that the angel JCL specifies the following ROOT parameter:

```
// SET ROOT='/usr/lpp/zosmf/wlp'
```

To find out what the version of Liberty for the angel process is, run the following script:

```
/usr/lpp/zosmf/wlp/bin/productInfo version --verbose
```

For a Liberty JVM server running in CICS, run the following script:

```
/usr/lpp/cicsts56/wlp/bin/productInfo version --verbose
```

```
WebSphereApplicationServer.properties:
        com.ibm.websphere.productId=com.ibm.websphere.appserver
        com.ibm.websphere.productOwner=IBM
        com.ibm.websphere.productVersion=16.0.0.3
        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=16.0.0.3             <== Liberty Version
        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

zOSMF.properties:
        com.ibm.websphere.productId=com.ibm.zoszmf
        com.ibm.websphere.productOwner=IBM
        com.ibm.websphere.productVersion=2.2.0
        com.ibm.websphere.productName=z/OSMF
        com.ibm.websphere.productPID=5650-ZOS
        com.ibm.websphere.productQualifier=z/OSMF
        com.ibm.websphere.productReplaces=com.ibm.websphere.appserver.zos
        com.ibm.websphere.productEdition=N/A
```

Figure 73. Example output

Authenticating users in a Liberty JVM server

Although you can configure CICS security for all web applications that run in a Liberty JVM server, the web application will only authenticate users if it includes a security constraint. The security constraint is defined by an application developer in the deployment descriptor (web.xml) of the Dynamic Web Project or OSGi Application Project. The security constraint defines what is to be protected (URL) and by which roles.

A `<login-config>` element defines the way a user gains access to web container and the method used for authentication. The supported methods are either HTTP basic authentication, form based authentication or TLS client authentication. Here is an example of those elements in web.xml:

```
<!-- Secure the application -->
<security-constraint>
    <display-name>com.ibm.cics.server.examples.wlp.tsq.web_SecurityConstraint</display-name>
        <web-resource-name>com.ibm.cics.server.examples.wlp.tsq.web</web-resource-name>
        <description>Protection area for com.ibm.cics.server.examples.wlp.tsq.web</description>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description>Only SuperUser can access this application</description>
        <role-name>SuperUser</role-name>
    </auth-constraint>
    <user-data-constraint>
        <!-- Force the use of SSL -->
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

```
<!-- Declare the roles referenced in this deployment descriptor -->
<security-role>
    <description>The SuperUser role</description>
    <role-name>SuperUser</role-name>
</security-role>

<!--Determine the authentication method -->
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

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.

Tasks that are authenticated in CICS using Liberty security can use the user ID derived from any of the Liberty application security mechanisms to authorize transaction and resource security checks in CICS. The CICS user ID is determined according to the following criteria:

1. Liberty application security authentication.

   Integration with the SAF registry is part of the CICS Liberty security feature, unless distributed identity mapping is used. Any of the application security mechanisms supported by Liberty are supported in CICS, this includes HTTP basic authentication, form login, TLS client certificate authentication, identity assertion using a custom login module, JACC, JASPIC, or a Trust Association Interceptor (TAI). All SAF user IDs authenticated by Liberty must be granted read access to the Liberty JVM server APPL class profile. The name of this is determined by the profilePrefix setting in the safCredentials element in the Liberty server configuration file server.xml.

   ```
   <safCredentials profilePrefix="BBGZDFLT"/>
   ```

   The APPL class is also used by CICS terminal users to control access to specific CICS regions and your Liberty JVM server can use the same profile as the CICS APPLID depending upon your security requirements. If you do not specify this element, then the default profilePrefix of BBGZDFLT is used.

   You must define the APPLID and users must have access to the it. To configure and activate the BBGZDFLT profile in the APPL class:

   ```
   RDEFINE APPL BBGZDFLT UACC(NONE)
   SETROPTS CLASSACT(APPL)
   ```

   The users must be given read access to the BBGZDFLT profile in the APPL class in order to authenticate. To allow user AUSER to authenticate against the BBGZDFLT APPL class profile:

   ```
   PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(AUSER)
   ```

   The Liberty SAF unauthenticated user id must be given read access to the APPL class profile. The SAF unauthenticated user id can be specified in the safCredentials element in the Liberty server configuration file server.xml.

   ```
   <safCredentials unauthenticatedUser="WSGUEST"/>
   ```

   If you do not specify the element, then the default unauthenticatedUser is WSGUEST. To allow the SAF unauthenticated user id WSGUEST read access to the BBGZDFLT profile in the APPL class:

   ```
   PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST)
   ```

   If you use WSGUEST, then you should follow the steps to configure the SAF user registry as described in Setting up the System Authorization Facility (SAF) unauthenticated user.

   The WLP z/OS System Security Access Domain (WZSSAD) refers to the permissions granted to the Liberty server. These permissions control which System Authorization Facility (SAF) application domains and resource profiles the server is permitted to query when authenticating and authorizing users. The CICS region user ID must be granted permission within the WZSSAD domain to make

authentication calls. To grant permission to authenticate, the CICS region ID must be granted READ access to the BBG.SECPFX.<APPL> profile in the SERVER class:

```
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

For more details refer to Accessing z/OS security resources using WZSSAD.

2. If an unauthenticated subject is supplied from Liberty, then the USERID defined in the URIMAP will be used.
3. If no USERID is defined in the URIMAP the request will run under the CICS default user ID.

Note:

Due to the way that security processing for Liberty transactions is deferred during CICS transaction attach processing, the user ID used in the CICS Monitoring Facility (CMF) records, the z/OS Workload Manager (WLM) classification, and the task association data and the UEPUSID global user exits field for the XAPADMGR exit, will be determined as follows; the user ID in the HTTP security header, or if there isn't one, the user ID taken from matching URIMAP. If neither exist, the CICS default user ID will be used.

Be aware that Liberty caches authenticated user IDs and, unlike CICS, 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. Please see Configuring the authentication cache in Liberty.

Authorizing users to run applications in a Liberty JVM server

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.

About this task

Your application is secured by providing an authorization constraint, the `<auth_constraint>` element, in the deployment descriptor (`web.xml`). If present, this ensures that access to your application is achieved only by a user that is a member of an authorized role. User or group membership to an Enterprise Java role is determined in one of two ways:

- Use an `<application-bnd>` element in the `<application>` element of your `server.xml` to describe the user/group to role mappings directly in XML.
- Use `<safAuthorization>` in your `server.xml` to allow users/groups role membership to be mapped by SAF (typically using EJBROLEs).

For more information, see Authorization using SAF role mapping.

Using CICS security allows you to re-use existing security procedures but requires that individual web applications are accessed from different URIMAPs. Using role-based security allows you to use existing standard Enterprise Java security definitions from another Enterprise Java application server. For more information, see “Authenticating users in a Liberty JVM server” on page 250.

If you want to use CICS transaction and resource authorization exclusively, or prefer to use finer-grained annotation-based role checking in code, you can defer the authorization decision to those components by using the special subject ALL_AUTHENTICATED_USERS role, as shown in the following example. If you deploy a Liberty application in a CICS bundle, CICS automatically configures this for you.

Note: Access checks are performed for the declarative security annotations and CICS transaction and resource security only after the configured constraints (`web.xml`) are verified

```
<application id="com.ibm.cics.server.examples.wlp.tsq.app"
    name="com.ibm.cics.server.examples.wlp.tsq.app" type="eba"
    location="${server.output.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>
```

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 using any authenticated user ID and authorization to the transaction must be controlled using CICS transaction security. If you deploy your application directly to the dropins directory, it is not configured to use CICS security as dropins does not support security.

If you are using safAuthorization then the `<application-bnd>` no longer acts as the source of user ID to role mapping. Instead, EJBROLEs in SAF determine which SAF users are in which roles (EJBROLEs). With safAuthorization the `<application-bnd>` is ignored. To achieve the same effect and allow all authenticated users to be authorized to run your application, the `<auth-constraint>` in `web.xml` must use the special role `**`, for example:

```
<auth-constraint>
        <description>special role for all authenticated users</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 defined in the deployment descriptor.

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 additional `<security-role>` declaration in `web.xml`.

To use CICS transaction or resource security you should follow the following steps:

Procedure

1. Define a URIMAP of type JVMSERVER for each web application. Typically, you might specify a URIMAP to match the generic context root (URI) of a web application to scope the transaction ID to the set of servlets that make up the application. Or you may choose to run each individual servlet under a different transaction with a more precise URI.
2. Authorize all users of the web application to use the transaction specified in the URIMAP by using a CICS transaction security profile.

Authorization using SAF role mapping

Mapping Java security roles to users and groups can be achieved in different ways. In distributed systems, a basic registry or LDAP registry would typically be used in conjunction with an application specific `<application-bnd>` element, to map users from those registries into *roles*. The deployment descriptor of the application determines which roles can access which parts of the application.

About this task

On z/OS, there is an additional registry type, the System Authorization Facility (SAF) registry. A Liberty JVM server implicitly uses this type for authentication when the `cicsts:security-1.0` feature is installed unless configured to use LDAP. You can choose to make use of SAF authorization. When using SAF authorization, user to role mappings are used to map roles to EJBROLE resource profiles using the SAF role mapper. The server queries SAF to determine if the user has the required READ access to the EJBROLE resource profile.

In a Liberty JVM server, if you want to use Java security roles without SAF authorization, you cannot use CICS bundles to install your applications. This 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. Instead, you must create an `<application>` element in `server.xml` directly and configure the `<application-bnd>` with the roles and users you require.

If, however, you choose to use Java security roles and SAF authorization, you can continue to use CICS bundles to lifecycle your web applications. The <application-bnd> is ignored by Liberty in favor of using the role mappings determined by the SAF registry. Role mappings are determined by virtue of a user belonging to an EJB role.

Tip: When SAF authorization is enabled, EJB roles in RACF are used for role mapping instead of the roles in server.xml. Therefore, special subjects such as ALL_AUTHENTICATED_USERS and EVERYONE, or users can not be defined in server.xml in this case.

Tip: It is advisable to create or update your EJB roles before starting the CICS region. Liberty issues a RACROUTE REQUEST=LIST with GLOBAL=NO in order to support a minimum version of z/OS. The address space will not see updates until it is restarted (or started).

Procedure

1. Add the <safAuthorization id="saf"/> element to your server.xml. If you are using the cicsts:distributedIdentity-1.0 feature, this is defined for you.
2. Optional: You can add racRouteLog="ASIS" to the element in the previous step.

   This allows you to see the RACF EJBROLE logging from Liberty.
3. Create the EJB roles in RACF, with reference to the prefix scheme described.
4. Add users to those EJB roles.

   By default, if SAF authorization is used, the application uses the pattern <profile_prefix>.<resource>.<role> to determine if a user is in a role. The profile_prefix defaults to BBGZDFLT but can be modified using the <safCredentials> element. For example, you can set it to the APPL_ID of a region. If you want multiple regions to share identical security configuration, you can set <profile_prefix> to the same value for those regions. For more information, see Accessing z/OS security resources using WZSSAD.

   The role mapping preferences can be modified using the <safRoleMapper> element in the server.xml, for example:

   ```
   <safRoleMapper profilePattern="myprofile.%resource%.%role%" toUpperCase="true"/>
   ```

   Users can then be authorized to a particular EJB role using the following RACF commands, where WEBUSER is the authenticated user ID.

   ```
   RDEFINE EJBROLE BBGZDFLT.MYAPP.ROLE UACC(NONE)
   PERMIT BBGZDFLT.MYAPP.ROLE CLASS(EJBROLE) ACCESS(READ) ID(WEBUSER)
   ```
5. Optional: If you are deploying the CICS servlet examples and want to use the Java security role with SAF authorization, create a SAF EJBROLE for each servlet that you have deployed. For example, if you use the default APPL class of BBGZDFLT, define the following EJBROLE security definitions using RACF commands:

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.hello.war.cicsAllAuthenticated UACC(NONE)
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.tsq.app.cicsAllAuthenticated UACC(NONE)
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.jdbc.app.cicsAllAuthenticated UACC(NONE)
SETROPTS RACLIST(EJBROLE) REFRESH
```

   Give read access to the defined roles for each web user ID that requires authorization:

```
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.hello.war.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.tsq.app.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.jdbc.app.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
SETROPTS RACLIST(EJBROLE) REFRESH
```

Results

You can authorize access to web applications using CICS Security, Java security role, or both by defining the roles and the users in the roles.

Configuring security for a Liberty JVM server with the Enterprise Java security API

Java EE 8 introduces a portable, flexible, and standardized security model with the Java EE security API 1.0. A Liberty JVM server can be configured to respect the new security configuration through the inclusion of the Liberty appSecurity-3.0 feature.

The Java EE security API 1.0 specification covers three principles:

1. Authentication mechanism: provided by the HttpAuthenticationMechanism interface for the servlet container
2. Identity store: an attempt to standardize the JAAS LoginModule
3. Security context: an access point for programmatic security

Authentication mechanism

An authentication mechanism is a way that is used to obtain a username and password from the user to be processed later by the Java Security API. There are two standard options for authentication, both take advantage of the annotations that are introduced by the Java EE security 1.0 API.

HTTP basic authentication

Basic authentication displays the browser's native login dialog before the user can access the protected resource.

```
@BasicAuthenticationMechanismDefinition(realmName="user-realm")
@WebServlet("/home") @DeclareRoles({"user"})
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class HomeServlet extends HttpServlet {
    ...
}
```

Form-based authentication

You can use form-based authentication to replace the browser’s built-in dialog with your own custom HTML form. You can create an application config class with annotations as follows:

```
@FormAuthenticationMechanismDefinition(
    loginToContinue = @LoginToContinue(
        loginPage = "/login",
        errorPage = "/error"
    )
)
@ApplicationScoped
public class ApplicationConfig {
    ...
}
```

Identity store

A component acts as a DAO (Data Access Object) for accessing user information, including their usernames, passwords, and associated roles. A number of identity store types are introduced by the Java EE security API 1.0, including:

Database identity store

A database identity store is used to retrieve user information from a relation database.

```
@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"
    }
)
```

Lightweight Directory Access Protocol (LDAP) identity store

LDAP is a common way of organizing a user's access to different systems across a single organization. LDAP realizes the idea of Single-Sign On, where a user has a single username and password, and then uses it across all different systems that are used to perform the everyday business of a specific organization.

```
@WebServlet("/home")
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
@LdapIdentityStoreDefinition(
    url = "ldap://localhost:33389/",
    callerBaseDn = "ou=user,dc=jsr375,dc=net",
    groupSearchBase = "ou=group,dc=jsr375,dc=net"
)
public class HomeServlet extends HttpServlet{
    ...
}
```

URL: The URL of the LDAP server to use for authentication.

callerBaseDn: Base distinguished name for callers in the LDAP store.

groupSearchBase: Search base for looking up groups.

Custom identity store

In addition to the built-in identity stores found in Java EE security API 1.0, a user can implement their own identity store and control exactly where to obtain user information. This can be achieved by creating a custom identity store class, then creating an HTTP authentication mechanism associated with this custom identity store.

Security context

The security context object is used to programmatically check a user's authority to access a specific resource. This is useful when you need to perform custom behavior. 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, res);
        } else {
            req.getRequestDispatcher("/logout").forward(req, res);
        }
    }
}
```

For more information about the Java EE 8 security API, see Java EE Security API in the Liberty documentation.

Authenticating by using a database identity store

You can use the @DatabaseIdentityStoreDefinition interface to retrieve user credentials from a database for authentication.

About this task

Follow these steps to authenticate by using a database identity store.

Procedure

1. Add the `appSecurity-3.0` feature to `server.xml` before you start the server.
2. Ensure that CDI annotation file scanning is enabled. CICS disables it by default in `server.xml`.

   You can ensure CDI annotation file scanning is enabled by checking the following line is not present in `server.xml`: `<cdi12 enableImplicitBeanArchives="false"/>`.
3. Create a table in the database and set up `server.xml`.
   For example, to create a Db2 table 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 here: Database Setup

   a) Add the jdbc-4.2 feature in `server.xml`:

   ```
   <feature>jdbc-4.2</feature>
   ```

   b) Set `jndiName` in `server.xml`, for example:

   ```
   <dataSource id="DefaultDataSource" jndiName="jdbc/sec">
       <jdbcDriver libraryRef=<xxx>"/>
       ...
   </dataSource>
   ```
4. Determine whether to use SAF for the CICS task userid.

   a) If you do not want to push the database identity onto the CICS task, you can remove the default safRegistry setting in `server.xml`. This makes the CICS task run under the default CICS userid.

   b) If you want CICS tasks to run under specific SAF users mapped from your database identity store, you need to take the following steps:

      i) Configure SAF in `server.xml` by setting the following SAF elements.

      ```
      <safCredentials mapDistributedIdentities="true" profilePrefix=lt;xxx>"/;
      <safAuthorization id="saf"/>
      <safRoleMapperprofilePattern=“<xxx>.%resource%.%role%" toUpperCase="false;
      ```

      ii) Issue the RACMAP command. The general RACMAP command of mapping a distributed userid to a SAF userid is in the format of:

      ```
      RACMAP ID(userid)
      MAP
      WITHLABEL('label-name')
      USERDIDFILTER(NAME('distributed-identity-user-name'))
      REGISTRY(NAME('distributed-identity-registry-name'))
      ```

Use `defaultRealm` in `REGISTRY(NAME(<nnn>))`, and use `<username_in_DBIS>` in `USERDIDFILTER(NAME(<nnn>))`, for example:

```
RACMAP ID(JATM12) MAP WITHLABEL('authorisedUser:JATM12')
USERDIDFILTER(NAME('authorisedUser')) REGISTRY(NAME('defaultRealm'))
```

Note: 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>
```

The security role `"cicsAllAuthenticated"` takes precedence over the group name that is stored in the database identity store and an HTTP 403 error occurs. There are two options you take:

i) Deploy your database identity store application with a direct `<application>` element in `server.xml`.

ii) Deploy within a CICS bundle, but use safAuthorization to bypass the CICS-generated `<application-bnd>` which overrides the group information stored in the Custom Identity Store.

Results

You have successfully configured the database identity store.

Authenticating by using a custom identity store

You can use a custom identity store to implement your own identity store and control exactly where to obtain user information.

About this task

Follow these steps to authenticate by using a custom identity store.

Procedure

1. Add the `appSecurity-3.0` feature to `server.xml` before you start the server.
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 by checking the following line is not present in `server.xml`: `<cdi12 enableImplicitBeanArchives="false"/>`.
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("authorisedUser", "tomtom")) {
            return new CredentialValidationResult("authorisedUser",
                new HashSet<String>(asList("user")));
        }
        return 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, HttpMessageContext 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();
        }
    }
}
```

c) Create a servlet.

```
@WebServlet("/home")
@ServletSecurity(@HttpConstraint(rolesAllowed = "user"))
public class Servlet extends HttpServlet {...}
```

4. Determine whether to use SAF for the CICS task userid.

a) If you do not want to push the custom identity onto the CICS task, you can remove the default safRegistry setting in `server.xml`. This makes the CICS task run under the default CICS userid.

b) If you want CICS tasks to run under specific SAF users mapped from your custom identity store, you need to take the following steps:

i) Configure SAF in `server.xml` by setting the following SAF elements.

```
<safCredentials mapDistributedIdentities="true" profilePrefix="<xxx>"/>
<safAuthorization id="saf"/>
<safRoleMapperprofilePattern="<xxx>.%resource%.%role%" toUpperCase="false"/>
```

ii) Issue the RACMAP command. The general RACMAP command of mapping a distributed userid to a SAF userid is in the format of:

```
RACMAP ID(userid)
MAP
WITHLABEL('label-name')
USERDIDFILTER(NAME('distributed-identity-user-name'))
REGISTRY(NAME('distributed-identity-registry-name'))
```

Use "`defaultRealm`" in `REGISTRY(NAME('<nnn>'))`, and use "`<username_in_CIS>`" in `USERDIDFILTER(NAME('<nnn>'))`, for example:

```
RACMAP ID(JATM12) MAP WITHLABEL('authorisedUser:JATM12')
USERDIDFILTER(NAME('authorisedUser')) REGISTRY(NAME('defaultRealm'))
```

Note: If you deploy the application within a CICS bundle, the security role `"cicsAllAuthenticated"` is automatically set in `installedApps.xml` as follows:

```
<application ...>
    <application-bnd>
        <security-role name="cicsAllAuthenticated">
            <special-subject type="ALL_AUTHENTICATED_USERS"/>
        </security-role>
    </application-bnd>
</application>
```

It takes precedence over the group name that is stored in the custom identity store and an HTTP 403 error occurs. There are two options you can take:

i) Deploy your custom identity store application with a direct `<application>` element in `server.xml`.

ii) Deploy within a CICS bundle, but use safAuthorization to bypass the CICS-generated <application-bnd> which overrides the group information stored in the custom identity store.

Results

You have successfully configured the custom identity store.

Configuring security for a Liberty JVM server by using an LDAP registry

Liberty uses a user registry to authenticate a user and retrieve information about users and groups to perform security-related operations, including authentication and authorization. Default CICS Liberty security uses the SAF registry. However, many transactions that run on CICS are initiated by users who authenticate their identities on distributed application servers, so CICS also supports the use of a Lightweight Directory Access Protocol (LDAP) registry in Liberty. To use LDAP, it is necessary to manually configure the `server.xml`.

Before you begin

- Ensure that the CICS region is configured to use SAF security and is defined with SEC=YES as a system initialization parameter.
- Authorize application developers and system administrators to create, view, update, and remove JVMSERVER and BUNDLE resources to deploy web applications into a Liberty JVM server. The JVMSERVER resource controls the availability of the JVM server, and the BUNDLE resource is a unit of deployment for the Java applications and controls the availability of the applications.

About this task

This task explains how to configure LDAP security for a Liberty JVM server, and integrate Liberty security with CICS security. Distributed identity mapping can be used to associate a SAF user ID with a distributed identity. You can use the CICS distributed identity mapping feature to set up distributed identity mapping. A user can then log on to a CICS web application with their distributed identity, as authenticated by an LDAP server. Filters that are defined in the z/OS security product (RACMAP) determine the mapping of this identity to a SAF user ID. This SAF user ID can then be used to authorize access to web applications through JEE application role security, providing integration with CICS transaction and resource security. You can map a SAF user ID to one or more distributed identities.

The default transaction ID for running any web request is CJSA. You can configure CICS to run web requests under a different transaction ID by using a URIMAP of type JVMSERVER. You can specify a URIMAP to match the generic context root (URI) of a web application to scope the transaction ID to the set of servlets that make up the application. Or you can choose to run each individual servlet under a different transaction with a more precise URI.

There are three scenarios for this task:

- Scenario 1 – Distributed identity mapping with SAF authorization
- Scenario 2 – Distributed identity mapping without SAF authorization
- Scenario 3 – LDAP for authentication and authorization

Procedure

1. **Distributed identity mapping with SAF authorization**

   You can use the CICS distributed identity mapping feature, `cicsts:distributedIdentity-1.0` to enable LDAP distributed identities to be mapped to SAF user IDs. When used with the CICS security feature `cicsts:security-1.0`, Liberty LDAP security is used for authentication and JEE application role security from EJB role mappings are respected for authorization. CICS transactions run under the mapped SAF user ID providing integration with CICS transaction and resource security.

a. Configure the WebSphere Liberty angel process to provide authentication and authorization services to the Liberty JVM server, for more information see The Liberty server angel process.

b. Add the `cicsts:security-1.0` and the `cicsts:distributedIdentity-1.0` feature to the featureManager list in the `server.xml`.

```
<featureManager>
   ...
   <feature>cicsts:security-1.0</feature>
   <feature>cicsts:distributedIdentity-1.0</feature>
</featureManager>
...
```

c. Configure Liberty to use LDAP authentication by defining the LDAP server in the `server.xml`, for example:

```
<ldapRegistry id="ldap"
    host="host.domain.com" port="389"
    ldapType="IBM Tivoli Directory Server"
    baseDN="ou=users,dc=domain,dc=com"
    ignoreCase="true">
</ldapRegistry>
```

Full details on configuring LDAP user registries with Liberty are available in Configuring LDAP user registries in Liberty.

d. Remove the safRegistry element, if present. Save the changes to the `server.xml`.

e. Make the necessary RACF definitions, including setting up the RACMAPs to map distributed identities to SAF user IDs as which are described in Configuring LDAP user registries in Liberty and providing access for these user IDs to the appropriate EJBROLEs as described in “Authorization using SAF role mapping” on page 253. CICS configures SAF authorization and the mapDistributedIdentities attributes in the safCredentials configuration element for you.

When the `cicsts:distributedIdentity-1.0` feature is used with the `cicsts:security-1.0` feature, Liberty LDAP security is used for authentication, and JEE application role security from EJB role mappings are respected for authorization. CICS transactions run under the RACMAP mapped user ID providing integration with CICS transaction and resource security.

What to do next

Back to top

2. **Distributed identity mapping without SAF authorization**

It is possible to allow CICS transactions to run under a RACMAP mapped user ID while respecting the roles configured in the application's <application-bnd> element. This might be useful when migrating work from distributed Liberty to CICS Liberty. Be aware that if CICS bundles are used, a user-defined <application-bnd> is overwritten by the CICS-generated <application-bnd>. SAF authorization using role mapping is preferred, for more information see “Authorization using SAF role mapping” on page 253 for more details.

a. Configure the WebSphere Liberty angel process to provide authentication and authorization services to the Liberty JVM server, for more information, see The Liberty server angel process.

b. Add the `cicsts:security-1.0` and the `ldapRegistry-3.0` feature to the featureManager list in the `server.xml`.

```
<featureManager>
   ...
   <feature>cicsts:security-1.0</feature>
   <feature>ldapRegistry-3.0</feature>
</featureManager>
...
```

c. Configure Liberty to use LDAP authentication by defining the LDAP server in the `server.xml`, for example:

```
<ldapRegistry id="ldap"
    host="host.domain.com" port="389"
```

```
    ldapType="IBM Tivoli Directory Server"
    baseDN="ou=users,dc=domain,dc=com"
    ignoreCase="true">
</ldapRegistry>
```

Full details on configuring LDAP user registries with the Liberty are available in Configuring LDAP user registries in Liberty.

d. Configure Liberty to use distributed identity filters to map the distributed identities to SAF user IDs by setting the `mapDistributedIdentities` attribute in the `safCredentials` configuration element to `true` in the `server.xml`.

e. Remove the safRegistry element, if present. Save the changes to the `server.xml`.

f. Make the necessary RACF definitions, including setting up the RACMAPs to map distributed identities to SAF user IDs as which are described in Configuring LDAP user registries in Liberty.

g. If JEE application role security from EJB roles is required for authorization then refer to the topic "Authorization using SAF role mapping" on page 253.

Applications use Liberty LDAP security for authentication, and JEE application role security in an `<application-bnd>` element are respected for authorization of the distributed identity. In CICS, transactions run under the RACMAP mapped user ID, providing integration with CICS transaction and resource security.

What to do next

Back to top

3. **LDAP for authentication and authorization**

LDAP security can be used in a CICS Liberty JVM server for both authentication and authorization using JEE application role security. URIMAP definitions can then be used to set the user ID under which transactions run. The `mapDistributedIdentities` attribute is not set in this scenario.

This scenario might be useful if migrating a distributed application into a CICS Liberty JVM server, without requiring any significant security resource changes.

a. Add the `cicsts:security-1.0` and the `ldapRegistry-3.0` feature to the featureManager list in the `server.xml`.

```
<featureManager>
   ...
   <feature>cicsts:security-1.0</feature>
   <feature>ldapRegistry-3.0</feature>
</featureManager>
...
```

b. Configure Liberty to use LDAP authentication by defining the LDAP server in the `server.xml`, for example:

```
<ldapRegistry id="ldap"
    host="host.domain.com" port="389"
    ldapType="IBM Tivoli Directory Server"
    baseDN="ou=users,dc=domain,dc=com"
    ignoreCase="true">
</ldapRegistry>
```

Full details on configuring LDAP user registries with Liberty are available in Configuring LDAP user registries in Liberty.

c. Remove the safRegistry element, if present. Save the changes to the `server.xml`.

d. If JEE application role security from EJB roles is required for authorization then refer to the topic "Authorization using SAF role mapping" on page 253.

Applications use Liberty LDAP security for authentication, and JEE application role security in an `<application-bnd>`element are respected for authorization. In CICS transactions run under the URIMAP or CICS DFLTUSER user ID as appropriate.

What to do next

Back to top

What to do next

This applies to all three scenarios:

- Modify the Liberty authentication cache.
- Set up URIMAP definitions to map web application URIs to transaction IDs.

This applies to scenarios 1 and 2:

- Set up CICS transaction security definitions to authorize access to URIs based on the mapped user ID.

Back to top

Configuring security for remote JCICSX API development

The JCICSX server is a remote Liberty JVM server in a CICS region. With the JCICSX API, it allows developers to run Java applications on their local workstation as if they were run in CICS, without deploying the applications to CICS. When remote connection is established from a JCICSX development client in the developer's local JVM to a JCICSX server, the remote server can authenticate users and authorize them with access based on their identities to ensure security. It also prevents users from interfering with remote tasks started by other users.

Table of contents

"What authentication and authorization options are available?" on page 263
"What options to choose?" on page 264
"Typical scenarios and procedures" on page 265

What authentication and authorization options are available?

The JCICSX server *authenticates* users to verify their identity. When planning how to authenticate JCICSX users, you have a few options to consider. First, you need to decide which user registry is used to store the user identity information. This topic covers the configuration of two registry options: using user identities from SAF (`safRegistry`) and embedding user identities directly in your `server.xml` (`basicRegistry`).

After configuring where the server is to find user identities, those users can be *authorized* to use the JCICSX server application. By default, the JCICSX server allows all users who are able to be authenticated with the server to access its services. However, this is customizable. The JCICSX server defines the Enterprise Java role JCICSXUSER, which can be used to customize access. This is achieved by mapping a role, which provides access to the application, to a group of users. Also, this role mapping can either be recorded in SAF (`safAuthorization`) or embedded directly in your `server.xml`.

Note: You must have a SAF registry to use SAF authorization.

Therefore, as shown in Figure 74 on page 264, the following authentication and authorization options are available for your remote JCICSX server:

- Using a basic user registry for authentication and `server.xml` for role mapping.
- Using a SAF registry for authentication and `server.xml` for role mapping.
- Using a SAF registry for authentication and SAF authorization for role mapping.

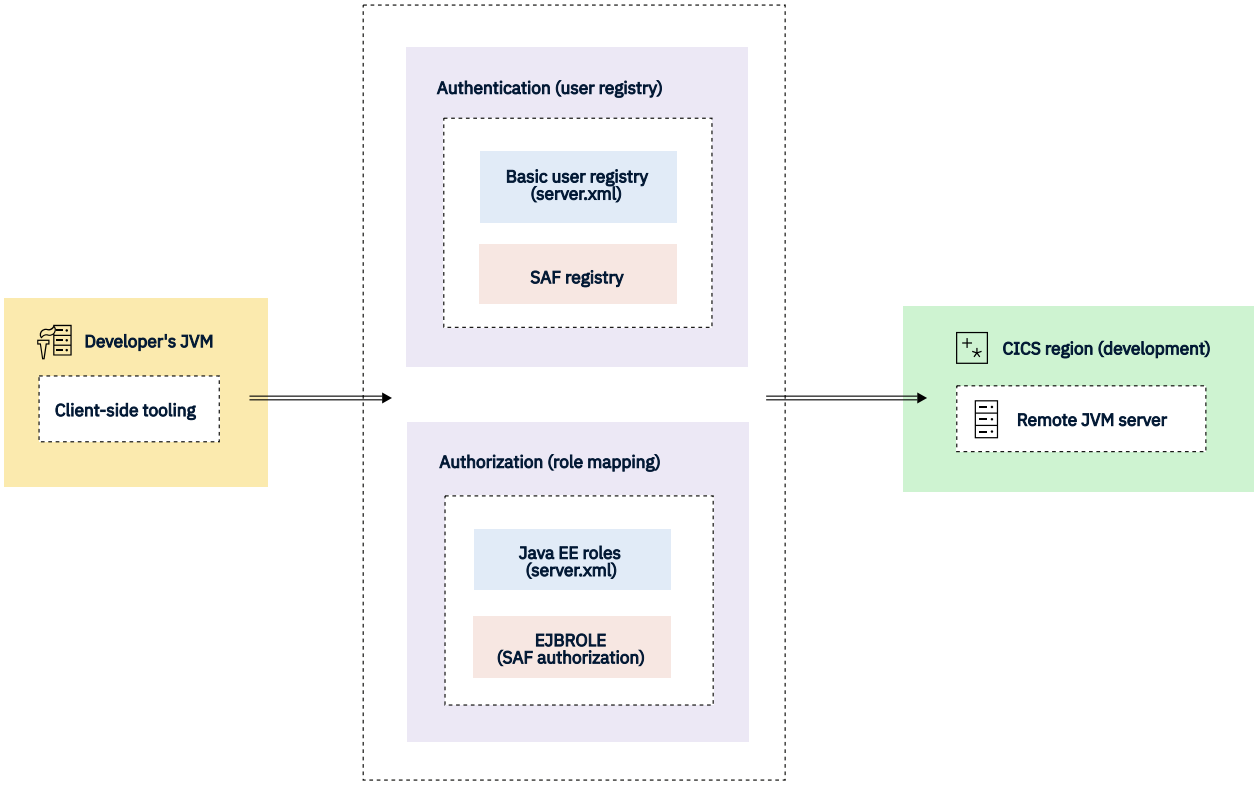


Figure 74. Authentication and authorization of remote JCICSX server

What options to choose?

From a logistical standpoint, it's simple to configure authentication and authorization directly in your `server.xml`. Therefore, it can be a convenient option if you are setting up the JCICSX server in a private development region. However, it has limitations at larger scales because the `server.xml` configuration is difficult to share. While it's more complicated to set up SAF for authentication and authorization, which involves the creation of EJB roles (EJBROLEs) in RACF, you can take advantage of existing information in your SAF database if you already have one. For example, you can authorize existing groups that are defined in SAF to use JCICSX, without having to specify them again. You can also share that information across multiple instances of JCICSX server running in different CICS regions, without having to configure each region independently.

Note: Any security configuration that you specify in the JCICSX server's `server.xml` must co-exist with the security requirements of other applications you have deployed into that server. For example, if you have an application that requires SAF authorization be enabled, you cannot specifically disable it for the JCICSX server and enable SAF authorization in the same Liberty server. In this case, you can create another JVM server that's dedicated to running the JCICSX server in your CICS region to work around this. If you don't create a separate server for JCICSX, you must follow instructions in “Scenario 3: Set up security in all my CICS regions, granting access to specific people” on page 266 to set up SAF authorization for the server.

In addition, when the client starts a session with the JCICSX server, a new CICS task is created to run under transaction CJXA and URI map DFHJXSU. Subsequent JCICSX requests for that session will run under the same task, and must be issued by the same user. Transaction CJXA is a category 2 transaction. If you have transaction attach security turned on, you also need to permit users to run transaction CJXA.

See "Typical scenarios and procedures" on page 265 for a discussion on how you might configure authentication and authorization for a number of typical scenarios.

Typical scenarios and procedures

Three scenarios are provided to cover the options described before.

- "Scenario 1: Allow all users to try it out in a single region with no authentication" on page 265
- "Scenario 2: Allow users to sign on using SAF identities, granting access to authenticated or specific users" on page 266
- "Scenario 3: Set up security in all my CICS regions, granting access to specific people" on page 266

Scenario 1: Allow all users to try it out in a single region with no authentication

To allow developers to test applications quickly in a development region, you can configure the remote JCICSX server to use no authentication. This task configures all security settings in `server.xml`.

Before you begin

Ensure that you have set up a Liberty JVM server to serve as the JCICSX server, by adding the JCICSX server feature (`cicsts:jcicsxServer-1.0`) to the `server.xml` file of your Liberty JVM server:

```
<featureManager>
    <feature>cicsts:jcicsxServer-1.0</feature>
</featureManager>
```

For more information, see Java development using JCICSX.

Procedure

1. If your server is not configured with the `appSecurity-2.0` feature to use Liberty security, no further configuration is needed. Any user can access the server with no credentials provided. For more information about `appSecurity`, see "Configuring security for a Liberty JVM server" on page 243.
2. If your server is configured with the `appSecurity-2.0` feature to use Liberty security:
   a. By default, the 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"/>
      ```

      Otherwise, users get a 403 error when they access the server with a username and password.
   b. Create a basic user registry to authenticate users in `server.xml`. For instructions, see Configuring a basic user registry for Liberty.
   c. By default, the server allows all the authenticated users defined in your user registry to access the servlet. Override the default setting to allow all users by adding the following snippet to `server.xml`:

      ```
      <authorization-roles id="com.ibm.cics.wlp.jcicsxserver">
        <security-role name="JCICSXUSER">
          <special-subject type="EVERYONE"/>
        </security-role>
      </authorization-roles>
      ```

      It changes the `special-subject` type from `ALL_AUTHENTICATED_USERS` to `EVERYONE` so that any user can access the servlet no matter what usernames or passwords they provide. If you don't specify the `EVERYONE` special subject, unauthenticated users who access the server get a 401 error.

3. If you are using transaction attach security, grant the CICS default user ID access to run the CJXA transaction.

Result

You have now configured remote JCICSX server to use no authentication for a CICS region.

Scenario 2: Allow users to sign on using SAF identities, granting access to authenticated or specific users

This is convenient if you already have a SAF registry to manage user identities. In this scenario, you use the SAF registry for authentication and configure role mapping in `server.xml` so that you don't need to configure new SAF EJBROLEs.

Before you begin

Ensure that you have set up a Liberty JVM server to serve as the JCICSX server, by adding the JCICSX server feature (`cicsts:jcicsxServer-1.0`) to the `server.xml` file of your Liberty JVM server:

```
<featureManager>
    <feature>cicsts:jcicsxServer-1.0</feature>
</featureManager>
```

For more information, see Java development using JCICSX.

Procedure

1. Enable CICS security, which integrates Liberty security, for the JCICSX server and configure it to use the SAF registry. For instructions, see “Configuring security for a Liberty JVM server” on page 243.
2. By default, the 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"/>
   ```

   Otherwise, users get a 403 error when they access the server with a username and password.
3. After users are authenticated, all the authenticated users are allowed to use the application by default. If you want to restrict the application to specific users, bind users to the `JCICSXUSER` security role in `server.xml`:

   ```
   <authorization-roles id="com.ibm.cics.wlp.jcicsxserver">
     <security-role name="JCICSXUSER">
       <user name="USER"/>
     </security-role>
   </authorization-roles>
   ```
4. If you are using transaction attach security, grant the CICS default user ID access to run the CJXA transaction.

Result

You now have configured the remote JCICSX server to authenticate users using the SAF registry and to authorize them access to the service by using role mapping in Enterprise Java.

Scenario 3: Set up security in all my CICS regions, granting access to specific people

In this scenario, you configure the JCICSX servers in all CICS regions that run under the default profile prefix to use SAF for authentication and authorization to grant specific user or user groups access. This is because SAF authorization makes it easy to share security settings across multiple CICS regions. When using SAF authorization, user to role mappings are used to map roles to EJBROLE resource profiles using the SAF role mapper. The server queries SAF to determine if the user has the required READ access to the EJBROLE resource profile. It's also convenient in that you can authorize an existing user group to use the JCICSX server by creating an EJBROLE.

Before you begin

Ensure that you have set up a Liberty JVM server to serve as the JCICSX server, by adding the JCICSX server feature (`cicsts:jcicsxServer-1.0`) to the `server.xml` file of your Liberty JVM server:

```
<featureManager>
    <feature>cicsts:jcicsxServer-1.0</feature>
</featureManager>
```

For more information, see Java development using JCICSX.

Procedure

1. Enable CICS security, which integrates Liberty security, for the JCICSX server and configure it to use the SAF registry. For instructions, see "Configuring security for a Liberty JVM server" on page 243.
2. By default, the 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"/>
   ```

   Otherwise, users get a 403 error when they access the server with a username and password.
3. Add the`<safAuthorization>` element to `server.xml`, to use SAF authorization for role mapping:

   ```
   <safAuthorization id="saf"/>
   ```

4. Create the EJBROLE in RACF using the following RACF command:

   ```
   RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER UACC(NONE)
   ```

   where BBGZDFLT is the default profile prefix, so this security configuration applies to all CICS regions that run under the default profile prefix. The profile prefix can be modified, making it easy for regions with the same profile prefix to share security settings. For more information, see "Authorization using SAF role mapping" on page 253.
5. Grant users READ access to those EJBROLE:

   ```
   PERMIT BBGZDFLT.com.ibm.cics.wlp.jcicsxserver.JCICSXUSER CLASS(EJBROLE) ACCESS(READ)
   ID(<user|group>)
   ```

6. If you are using transaction attach security, grant the CICS default user ID access to run the CJXA transaction.

Result

You have configured the remote JCICSX servers in all CICS regions running under the default profile prefix to authenticate users using the SAF registry and to authorize specific users or user groups with access to the services.

Configuring SSL (TLS) for a Liberty JVM server using a Java keystore

You can configure a Liberty JVM server to use SSL for data encryption and, optionally, authenticate with the server by using a client certificate. Certificates can be stored in a Java keystore or in a SAF key ring, such as a RACF key ring.

About this task

Enabling SSL in a Liberty JVM server requires adding the **`transportSecurity-1.0`** Liberty feature, a keystore, and an HTTPS port. CICS automatically creates and updates the `server.xml` file. Autoconfiguring always results in the creation of a Java keystore.

It is important to understand that any web request to a Liberty JVM server uses the JVM support for TCP/IP sockets and SSL processing, not CICS sockets domain.

Procedure

- To use autoconfigure to configure SSL, complete the following steps:
  a) Ensure autoconfigure is enabled in the JVM profile by using the JVM system property **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true**.
  b) Set the SSL port by setting the JVM system property **-Dcom.ibm.cics.jvmserver.wlp.server.https.port** in the JVM profile.
  c) Restart the JVM server to add the necessary configuration elements to `server.xml`.

Results

SSL for a Liberty JVM server is successfully configured.

Configuring SSL (TLS) for remote JCICSX API development

When configuring a Liberty JVM server in CICS for remote JCICSX API development, you can configure it to use SSL for data encryption.

About this task

To enable the remote development functionality of the JCICSX API, a Liberty JVM server is required in CICS to receive requests from the developer's local Liberty JVM server. To enable SSL communication between the remote Liberty JVM server in CICS and the local Liberty JVM server on the developer's machine, the remote server must be configured to use SSL and its certificate must be trusted by the local Liberty server.

Before you begin

- Configure the remote Liberty JVM server for user authentication and authorization. For instructions, see "Configuring security for remote JCICSX API development" on page 263
- Enable SSL in the Liberty JVM server in CICS. For instructions, see "Configuring SSL (TLS) for a Liberty JVM server using a Java keystore" on page 267.
- Ensure that a local Liberty JVM server is configured to make remote JCICSX requests. This is usually done by the application developer on their local machine. For instructions, see **Extra configuration for remote development (local workstation)** in Java development using JCICSX.

Procedure

- **For system programmers, configure the remote Liberty JVM server as follows:**
  As a system programmer, you need to generate a certificate for the remote Liberty JVM server with its host name registered.
  a) **Register the host name of the remote server in the `server.xml` file.**
  By default the host name of the Liberty server is `localhost`. In this case your SSL connection will fail because the certificate will be registered to `localhost` while your client will be trying to connect to the host name of your CICS region. To override the default `localhost` host name, add the `defaultHostName` variable to your `server.xml` file:

```
<variable name="defaultHostName" value="your-hostname"/>
```

  where *your-hostname* is the host name of the remote Liberty JVM server. For more information, see Setting the default host name of a Liberty server.
  b) **Generate a new copy of the remote Liberty JVM server's public certificate.**
  After setting the default host name of your server, you must regenerate the certificate for it.
  a. Stop the Liberty JVM server.

b. Delete the Java keystore that stores the certificates. It is the `key.p12` file located in `{server.config.dir}/resources/security`.

c. Start the liberty server.

d. Verify that a new `key.p12` file is regenerated.

c) **Verify that the host name is correct in the certificate using OpenSSL or the Java keytool utility:**

– If you're using OpenSSL, input this command to show the certificates of the remote Liberty JVM server:

```
$ openssl s_client -showcerts -connect remotejcicsxserver.com:portNo
```

– If you're using the Java keytool utility:

a. Navigate to the folder of the keystore on the remote Liberty server at: `{server.config.dir}/resources/security`.

b. If the local Liberty server is at 19.0.0.3 or later, which is the minimum version required to use the client-side tooling of remote JCICSX development, and that autoconfigure is enabled for the remote Liberty server to use SSL, the remote Liberty server will have created a keystore using default values. In this case, use this command to show the certificates stored in the auto-created Java keystore:

```
keytool -list -keystore key.p12 -storepass defaultPassword -storetype PKCS12 -v
```

Otherwise, substitute values in for `storepass` and `storetype` according to your custom configuration.

c. In the output, verify that `CN = your-hostname` shows the host name of the remote JVM server, instead of the default `localhost` value. Otherwise, repeat Step "2" on page 268.

- **For application developers, configure the local Liberty JVM server to trust the remote server's public certificate as follows:**

  By default, the local Liberty JVM server might refuse to connect to the remote Liberty JVM server because it does not trust the public certificate that the remote Liberty provides during the SSL handshake. Therefore, the application developer must download a copy of the certificate from the remote Liberty server and add it to the truststore used by the local Liberty server.

  a) Download a copy of the public certificate from the remote Liberty JVM server:

  To use OpenSSL:

  a. Run the following command to show the certificates of the remote server, where *your-hostname* and *your-port* are the host name and port number of your remote Liberty JVM server:

  ```
  openssl s_client -showcerts -connect your-hostname:your-port
  ```

  b. From the output, copy the first certificate. Include the following lines and the information between these lines:

  ```
  "-----BEGIN CERTIFICATE-----"
  "-----END CERTIFICATE-----"
  ```

  c. Paste the certificate into a new file with a `.cer` extension, for example, `publicKey.cer`.

  Note: Note: Be sure not to include additional lines in this file; otherwise the certificate won't be added to you local Liberty truststore successfully.

  If you have access to the remote Liberty server, you can also use the Java keytool utility to download the certificate:

  a. Navigate to the keystore on your remote Liberty JVM server. The file path is `{server.config.dir}/resources/security/key.p12`.

b. Use the Java keytool utility to create a public certificate:

```
keytool -rfc -export -keystore key.p12 -alias default -file /your/save/location/
public-remote.cer -v -storepass yourKeyStorePassword -storeType yourType
```

where *yourType* is the keystore type, which defaults to PKCS12. For more information, see Liberty default keystore type changed to PKCS12.

b) Navigate to the folder of the keystore on the local Liberty JVM server: `{server.config.dir}/resources/security`.

c) Import the public certificate that the system programmer downloaded into the truststore of the local Liberty, using the following command:

```
keytool -importcert -file /cert/location/public-remote.cer -keystore key.p12 -storepass
localPassword -storetype yourType -trustcacerts -v
```

where *yourType* is the keystore type, which defaults to PKCS12. For more information, see Liberty default keystore type changed to PKCS12.

d) Restart the local Liberty JVM server to pick up the new certificate.

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.

Using the syncToOSThread function

You can use the syncToOSThread function of Liberty in a CICS Liberty JVM server. SyncToOSThread enables 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 is the identity 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.

About this task

Enabling syncToOSThread requires the Liberty appSecurity-1.0 and zosSecurity-1.0 features. These features are included with the cicsts:security-1.0 feature. You must also define the syncToOSThread configuration element in the Liberty `server.xml` and add a special `<env-entry/>` to the application's deployment descriptor (`web.xml`). In addition, the SAF registry must be used for authentication, the angel process must be up and running, and the server must be connected to the angel process. For more information about the angel process, see Process types on z/OS.

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.*<profilePrefix>* profile in the FACILITY class, where *<profilePrefix>* is specified on the `<safCredentials />` element. This allows the Liberty server to synchronize any Java security subject with the OS thread identity:

     ```
     PERMIT BBG.SYNC.<profilePrefix> ID(<serverUserId>) ACCESS(CONTROL) CLASS(FACILITY)
     ```

   - Grant the CICS region user ID READ access to the BBG.SYNC.*<profilePrefix>* profile in the FACILITY class. Additionally, grant the CICS region user ID READ access to one or more

BBG.SYNC.*<AuthUserid/>* profiles in the SURROGATE class, one for each authenticated user ID to be synchronized with the OS identity:

```
PERMIT BBG.SYNC.<profilePrefix> ID(<serverUserId>) ACCESS(READ) CLASS(FACILITY)
PERMIT BBG.SYNC.<AuthUserid> ID(<serverUserId>) ACCESS(READ) CLASS(SURROGAT)
```

Restriction: A servlet configured as the welcome page in `web.xml`, does not support the syncToOSThread function.

Authorizing applications by using OAuth 2.0

OAuth 2.0 is an open standard for delegated authorization. The OAuth authorization framework enables a user to grant a third-party application access to information that is stored with another HTTP service without sharing their access permissions or the full extent of their data.

WebSphere Liberty supports OAuth 2.0, and can be used as an OAuth service provider endpoint and an OAuth protected resource enforcement endpoint. Liberty supports persistent OAuth 2.0 services. See Configuring persistent OAuth 2.0 services. Clients can be defined locally with the localStore and client elements. The following procedure uses local clients to enable OAuth 2.0 authorization.

Before you begin

SAF security is a common use-case in CICS, and this procedure uses SAF in the examples.

Ensure that the CICS region is configured to use SAF security and is defined with SEC=YES as a system initialization parameter.

Optionally, you can grant an administrator user access to the SAF EJBROLE `BBGZDFLT.com.ibm.ws.security.oauth20.clientManager`. The security role `clientManager` controls access to the management interfaces, allowing local clients to be queried, and persistent local clients to be created. The administrator user controls OAuth 2.0 local clients.

Configure the Liberty angel process to provide authentication and authorization services to the Liberty JVM server. See The Liberty server angel process.

For more information about OAuth, see oauth-2.0.

About this task

The following procedure covers how to:

- Create an OAuth 2.0 service provided in a Liberty JVM server.
- Create a locally configured client.
- Use this local client to grant an OAuth 2.0 token to a relying party application, also known as a third-party web application.
- Use this token to access protected resources in an application.

Restriction: Db2 JDBC type 2 connectivity is not supported for persistent OAuth 2.0 services.

Procedure

1. Configure an OAuth 2.0 service provider.
   a) 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>
...
```

   b) Configure an OAuth 2.0 provider in `server.xml`.

```
<oauthProvider id="myProvider">
</oauthProvider>
```

2. Configure a local client for the relying party application. Local clients define the details of the relying party application, including the name, secret password, and redirect URI of the application.
   a) Define a meaningful local client name and create a secret password that is used by the server for authorization. The local client application listens on a URI, and the server supplies authorization codes.
   b) Configure an OAuth 2.0 local client in the `oauthProvider` element of `server.xml`, supplying the local client ID, secret password, and the redirect URI.

```
<oauthProvider id="myProvider">
    <localStore>
        <client id="myClient" redirect="https://client.example.ibm.com/webApp/redirect"
secret="mySecret" />
    </localStore>
</oauthProvider>
```

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.

Note: More than one local client can be configured in the `localStore` element.

3. When the relying party application requires access to protected resources on the server, the user must authorize access to these resources first.
   a) The relying party application requires the user to authenticate with the server, and select the type of access for the relying party application by linking or redirecting the user to the authorization endpoint:

```
https://hostname:port/oauth2/endpoint/provider_name/authorize
```

or

```
https://hostname:port/oauth2/declarativeEndpoint/provider_name/authorize
```

Additional parameters are required in the query parameters of the URL. For the local client that was configured in step 2, the following GET request is required (all one line):

```
https://zos.example.ibm.com/oauth2/endpoint/myProvider/authorize?response_type=code
   &client_id=myClient&client_secret=mySecret&redirect_uri=https://client.example.ibm.com/webApp/redirect
```

After the user selects the access for the relying party application, they are redirected back to the relying party application using the redirect URI:

```
https://client.example.ibm.com/webApp/redirect?code=access_code
```

The relying party application must store this access code to request an OAuth token.

Note: For local clients, the user must exist in a user register in the Liberty JVM server. For more information about authenticating users in Liberty JVM servers, see Authenticating users in a Liberty JVM server.

   b) The relying party application requests an OAuth 2.0 token by sending a POST request to the server:

```
https://hostname:port/oauth2/endpoint/provider_name/token
```

The relying party application sends the authorization code that is received from the authorization endpoint, the local client ID, and the secret password in the POST data (`grant_type` is all one line):

```
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=myClient
   &client_secret=mySecret&redirect_url=https://client.example.ibm.com/webApp/redirect
```

This returns a JSON document that contains the token.

4. Use the token to access protected resources.
   a) Add the token to the `Authorization` header on the HTTP request.

   ```
   Authorization: Bearer <token>
   ```

Results

Users are able to authorize third-party applications to access their protected resources in a Liberty JVM server through OAuth 2.0 authorizations flows. The Liberty JVM server can configure the provider of these tokens and create locally configured clients.

Several methods to grant tokens are available. For more information, see OAuth 2.0 service invocation.

Configuring persistent OAuth 2.0 services

WebSphere Liberty supports persisting OAuth 2.0 local clients and tokens to a database. With persistent OAuth 2.0, an authorized local client can continue to access OAuth 2.0 services after a restart.

Before you begin

SAF security is a common use-case in CICS, and this procedure uses SAF in the examples.

- Gain the necessary access to create tables and read/write to these tables in a database and configure it in the Liberty `server.xml`.
- Grant access to the SAF EJBROLE BBGZDFLT`.com.ibm.ws.security.oauth20.clientManager` to an administrator user to control OAuth 2.0 local clients.
- Create an OAuth 2.0 provider in the Liberty `server.xml`. For more information, see Authorization using OAuth 2.0.

About this task

The following steps create a persistent OAuth 2.0 local client. This local client is used to grant OAuth 2.0 tokens.

Restriction: Db2 JDBC type 2 connectivity is not supported for persistent OAuth 2.0 services.

Procedure

1. Create the necessary tables using IBM Db2 for persistent OAuth services as a guide.
2. Create a persistent local client by sending a POST request to the URL:

   ```
   https://hostname:port/oauth2/endpoint/provider_name/registration
   ```

   Use the JSON document which is described in the first table in Configuring an OpenID Connect Provider to accept client registration requests; for example:

   ```
   {
       "client_id": "client_id",
       "client_secret": "client_secret",
       "grant_types": [ "authorization_code", "refresh_token" ],
       "redirect_uris": [ "https://client.example.ibm.com/webApp/redirect" ]
   }
   ```

Results

A persistent OAuth 2.0 local client is created. When this local client is used to produce tokens, the tokens are persisted to the database. If the server restarts, the persistent local client and tokens remain valid.

Enabling a Java security manager

By default, Java applications have no security restrictions placed on activities requested of the Java API. To use Java security to protect a Java application from performing potentially unsafe actions, you can enable a security manager for the JVM in which the application runs.

About this task

The security manager enforces a security policy, which is a set of permissions (system access privileges) that are assigned to code sources. A default policy file is supplied with the Java platform. However, to enable Java applications to run successfully in CICS when Java security is active, you must specify an additional policy file that gives CICS the permissions it requires to run the application.

You must specify this additional policy file for each kind of JVM that has a security manager enabled. CICS provides some examples that you can use to create your own policies.

Notes: Enabling a Java security manager is not supported in a Liberty JVM server.

- The OSGi security agent example creates an OSGi middleware bundle called `com.ibm.cics.server.examples.security` in your project that contains a security profile. This profile applies to all OSGi bundles in the framework in which it is installed.
- The `example.permissions` file contains permissions that are specific to running applications in a JVM server, including a check to ensure that applications do not use the `System.exit()` method.
- CICS must have read and execute access to the directory in zFS where you deploy the OSGi bundle.

For applications that run in the OSGi framework of a JVM server:

Procedure

1. Create a plug-in project in the IBM CICS SDK for Java and select the supplied OSGi security agent example.
2. In the project, select the `example.permissions` file to edit the permissions for your security policy.
   a) Validate that the CICS zFS and Db2 installation directories are correctly specified.
   b) Add other permissions as necessary.
3. Deploy the OSGi bundle to a suitable directory in zFS such as `/u/bundles`.
4. Edit the JVM profile for the JVM server to add the OSGi bundle to the `OSGI_BUNDLES` option before any other bundles:

   ```
   OSGI_BUNDLES=/u/bundles/com.ibm.cics.server.examples.security_1.0.0.jar
   ```
5. Add the following Java property to the JVM profile to enable security.

   ```
   -Djava.security.policy=all.policy
   ```
6. Add the following Java environment variable to the JVM profile to enable security in the OSGi framework:

   ```
   org.osgi.framework.security=osgi
   ```
7. To allow the OSGi framework to start with Java 2 security, add the following policy:

   ```
   grant { permission java.security.AllPermission; };
   ```
8. Save your changes and enable the JVMSERVER resource to install the middleware bundle in the JVM server.
9. Optional: Activate Java 2 security.
   a) To activate a Java 2 security policy mechanism, add it to the appropriate JVM profile. You must also edit your Java 2 security policy to grant appropriate permissions.

b) To use JDBC or SQLJ from a Java application with a Java 2 security policy mechanism active, use the IBM Data Server Driver for JDBC and SQLJ.

c) To activate a Java 2 security policy mechanism, edit the JVM profile.

d) Edit the Java 2 security policy to grant permissions to the JDBC driver, by adding the lines that are shown in Example 1. In place of db2*xxx*, specify a directory below which all your Db2 libraries are located. The permissions are applied to all the directories and files below this level. This enables you to use JDBC and SQLJ.

e) Edit the Java 2 security policy to grant read permissions, by adding the lines that are shown in Example 2. If you do not add read permission, running a Java program produces AccessControlExceptions and unpredictable results. You can use JDBC and SQLJ with a Java 2 security policy.

Example 1:

```
grant codeBase "file:/usr/lpp/db2xxx/-" {
  permission java.security.AllPermission;
};
```

Example 2:

```
grant {

// allows anyone to read properties
permission java.util.PropertyPermission "*", "read";

};
```

Results

When the Java application is called, the JVM determines the code source for the class and consults the security policy before granting the class the appropriate permissions.

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.

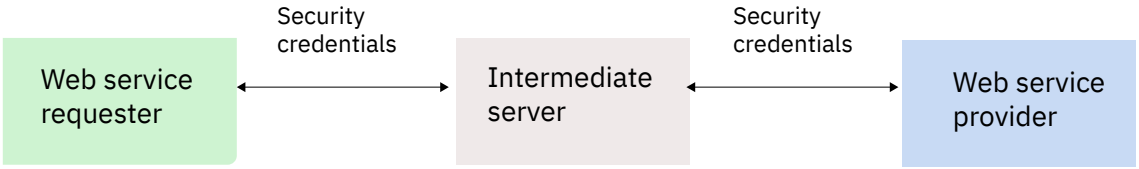


Figure 75. Transport-based security for SOAP messages

SOAP message security

When you use transport-based security (as illustrated in Figure 75 on page 277), 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.

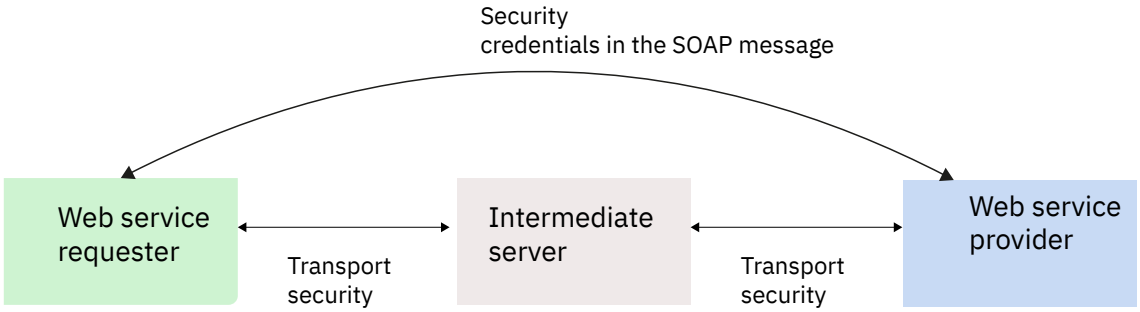


Figure 76. SOAP message security

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 278
- "What SOAP message security offers" on page 278
- "Components of SOAP message security" on page 279.

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 77 on page 279 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.

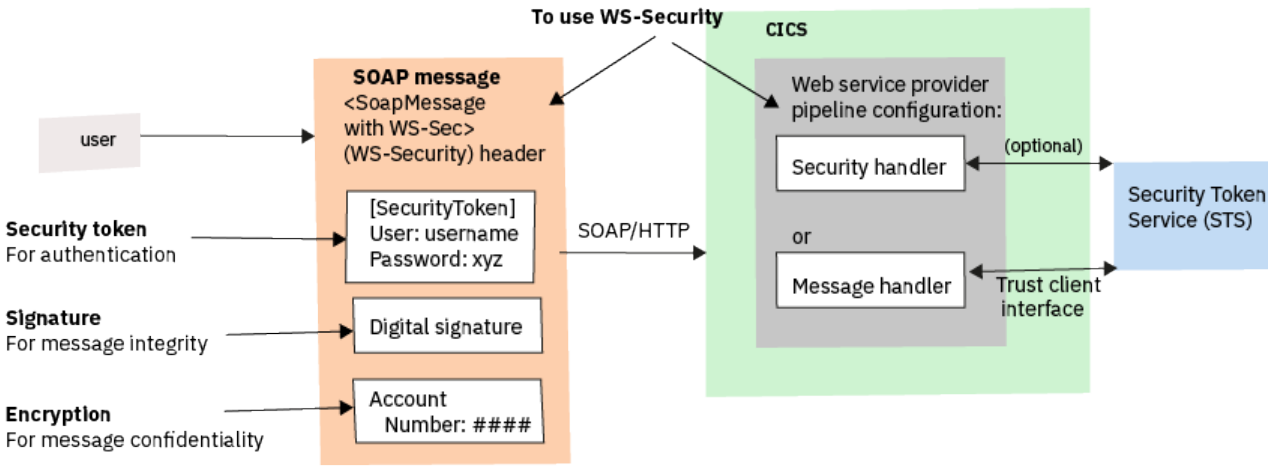


Figure 77. How CICS supports SOAP message security

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 281.
- 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 282 .
- 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 284.
- 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 one of 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 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 reauthenticate.

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 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 with SHA1) Supported on inbound SOAP messages only.	http://www.w3.org/2000/09/xmldsig#dsa-sha1
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/xmlenc#" 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">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                   SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                   MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                   hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                   6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                   msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                   AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                   ZQVRFU0BVSy5JQk0uQ09ggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
 </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/xmlenc#">
   <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>QORZEA+gpafluShspHxhrjaFlXE=</ds:DigestValue> 3
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>drDH0XESiyN6YJm27mfK1ZMG4Q4IsZqQ9N9V6kEnw2lk7aM3if77XNFnyKS4deglbC3ga11kkaFJ 4
                     p4jLOmYRqqycDPpqPm+UEu7mzfHRQGe7H0EnFqZpikNqZK5FF6fvYlv2JgTDPwrOSYXmhzwegUDT
                     lTVjOvuUgXYrFyaO3pw=</ds:SignatureValue>
   <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"/
> 5
    </wsse:SecurityTokenReference>
   </ds:KeyInfo>
  </ds:Signature>
 </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.wssecfvt.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 safely be 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 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).

Algorithm	URI
Triple Data Encryption Standard algorithm (Triple DES)	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 128 bits	http://www.w3.org/2001/04/xmlenc#aes128-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 192 bits	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Advanced Encryption Standard (AES) algorithm with a key length of 256 bits	http://www.w3.org/2001/04/xmlenc#aes256-cbc

Example of an encrypted message

This example shows a SOAP message that is encrypted 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/xmlenc#" SOAP-ENV:mustUnderstand="1">

  <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary" 1
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"
                wsu:Id="x509cert00">MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJHQjEMMAoGA1UEChMD
                                    SUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEzMTAwMDAwMFoXDTA3MDEzMTIzNTk1OVow
                                    MDELMAkGA1UEBhMCR0IxDDAKBgNVBAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkq
                                    hkiG9w0BAQEFAAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiogePsR
                                    6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8JXPFzA+BqV63JqRgJyxN6
                                    msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhC
                                    AQ0EMhMwR2VuZXJhdGVkIGJ5IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgG
                                    A1UdEQQxMC+BEVdZQVRFU0BVSy5JQk0uQ09NggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO
                                    BgNVHQ8BAf8EBAMCAfYwHQYDVR0OBBYEFMiPX6VZKP5+mSOY1TLNQGVvJzu+MA0GCSqGSIb3DQEB
                                    BQUAA4GBAHdrS409Jhoe67pHL2gs7x4SpV/NOuJnn/w25sjjop3RLgJ2bKtK6RiEevhCDim6tnYW
                                    NyjBL1VdN7u5M6kTfd+HutR/HnIrQ3qPkXZK4ipgC0RWDJ+8APLySCxtFL+J0LN9Eo6yjiHL68mq
                                    uZbTH2LvzFMy4PqEbmVKbmA87alF
  </wsse:BinarySecurityToken>
  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
   <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/> 2
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
```

```
    <wsse:SecurityTokenReference>
     <wsse:Reference URI="#x509cert00"
                    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509"/> 3
    </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   <xenc:CipherData>
    <xenc:CipherValue>M6bDQtJrvX0pEjAEIcf6bq6MP3ySmB4TQOa/B5UlQj1vWjD56V+GRJbF7ZCES5ojwCJHRVKW1ZB5 4
                      Mb+aUzSWlsoHzHQixc1JchgwCiyIn+E2TbG3R9m0zHD3XQsKTyVaOTlR7VPoMBd1ZLNDIomxjZn2
                      p7JfxywXkObcSLhdZnc=</xenc:CipherValue>
   </xenc:CipherData>
   <xenc:ReferenceList>
    <xenc:DataReference URI="#Enc1"/>
   </xenc:ReferenceList>
  </xenc:EncryptedKey>
 </wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
 <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="Enc1" Type="http://www.w3.org/2001/04/
xmlenc#Content">
  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/> 5
  <xenc:CipherData>
   <xenc:CipherValue>kgvqKnMcgIUn7rl1vkFXF0g4SodEd3dxAJo/mVN6ef211B1MZelg7OyjEHf4ZXwlCdtOFebIdlnK 6
                     rrksql1Mpw6So7ID8zav+KPQUKGm4+E=</xenc:CipherValue>
  </xenc:CipherData>
 </xenc:EncryptedData>
</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 was used to encrypt the symmetric key.

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 286
- “Security design considerations for CICS web service providers” on page 287

For more information, see Configuring security for CICS web services.

Scenarios for CICS web service providers

In Figure 78 on page 286, the web service requester sends a request directly to a CICS web service provider application.

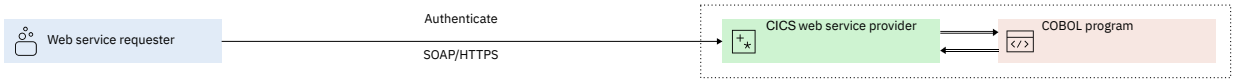


Figure 78. Direct request to CICS web service provider

In Figure 79 on page 287, 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.

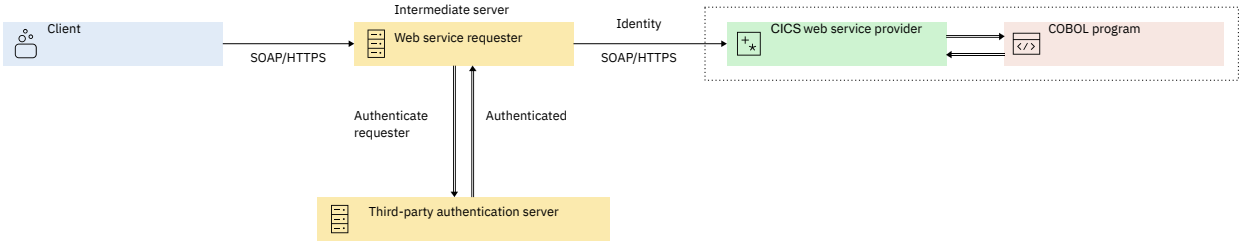


Figure 79. Request from an intermediate server

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 79 on page 287.

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 can be 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.

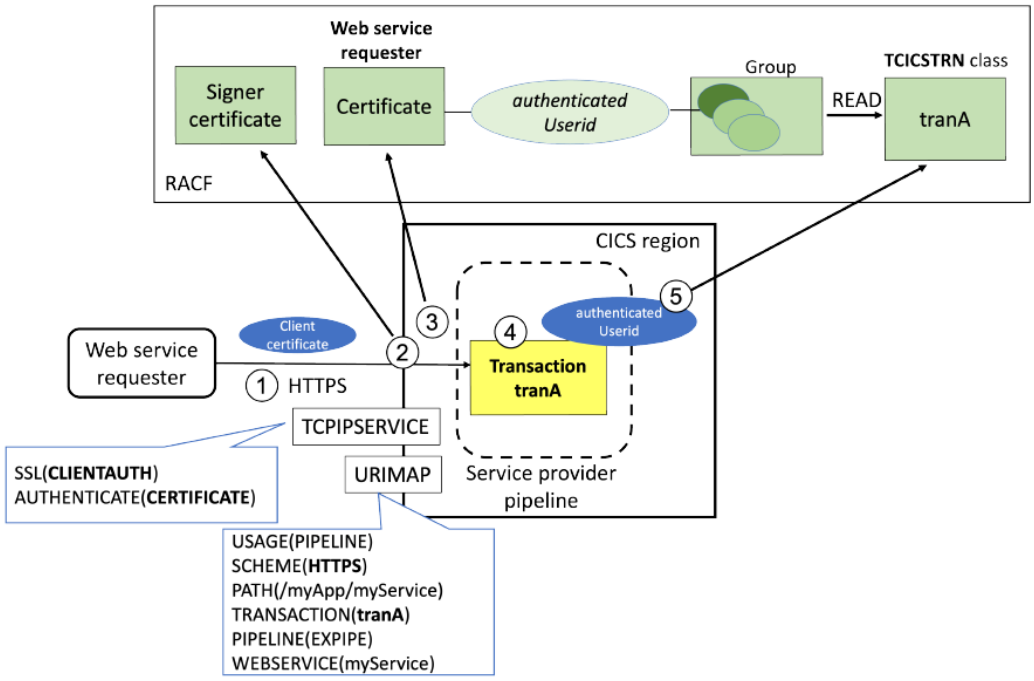


Figure 80. TLS authentication in a web service provider scenario

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 authenticated-userid is authorized to run transaction *tranA*. This validation is achieved by calling RACF to check that a group to which the *authenticatedUserid* belongs has READ authority to the profile called *tranA* in the TCICSTRN class. The resource group class GCICSTRN can also be used for transaction security checks.

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 Creating a SOAP web service.

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 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.
- *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 keying:

   ```
   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:

      ```
      SETROPTS 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:

      ```
      SETROPTS 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 TCPIPSERVICE.

   A TCPIPSERVICE 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 TCPIPSERVICE(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.

   ```
   RDEFINE GCICSTRN PIPEUSER UACC(NONE) +
            ADDMEM(CPIA,CPIH,CPIL,CPIQ) +
   PERMIT GCICSTRN PIPEUSER ID(taskUserGroup) ACCESS(READ)
   CONNECT clientUserid 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 *clientUserid* to *taskUserGroup*.

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:

```
DFHSO0107 02/07/2022 14:27:00 IYK2ZGV1 TCPIPSERVICE 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 TCPIPSERVICE 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.

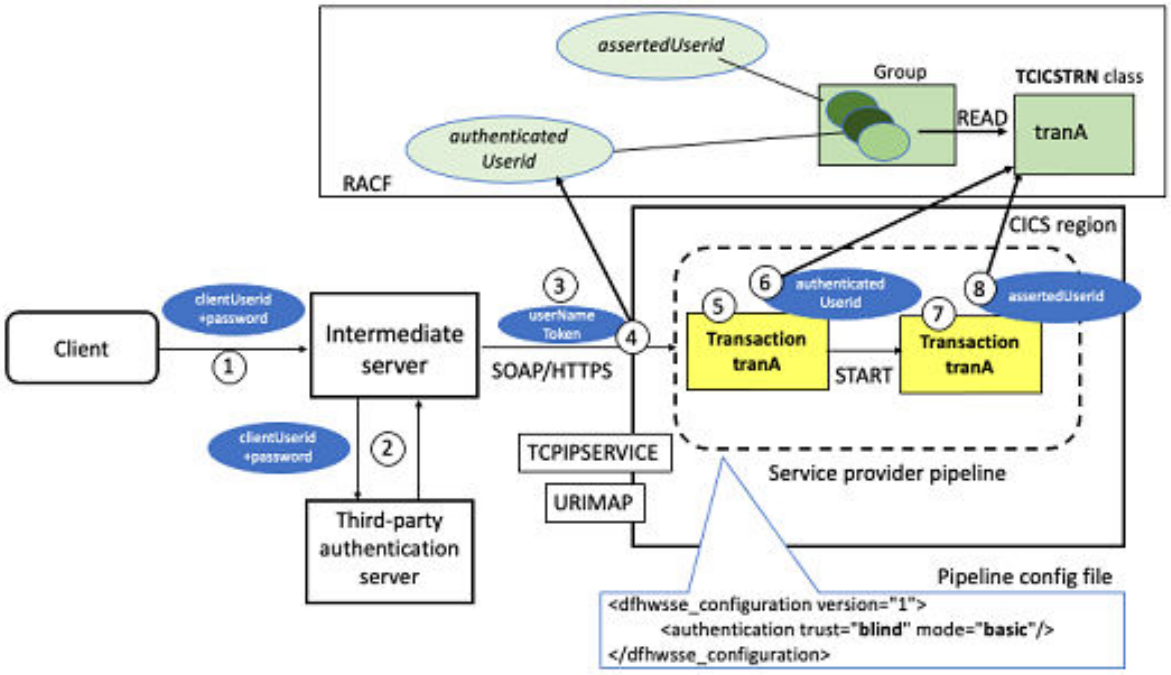


Figure 81. Identity assertion in a web service provider scenario

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.
5. The web service requester application validates the CICS server certificate and sends its TLS client certificate.
6. The request is sent over HTTPS. Therefore, encryption and integrity are enabled at the transport level.
7. CICS validates the client certificate and maps the client certificate to a RACF user ID
8. The web attach transaction runs with the user ID mapped from TLS client certificate.

9. 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.
10. The pipeline alias transaction runs under the asserted user ID 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 Creating a SOAP web service.

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 keying:

   ```
   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:

      ```
      SETROPTS 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:

      ```
      SETROPTS 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 TCPIPSERVICE.

A TCPIPSERVICE 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 TCPIPSERVICE(tcpipService) GROUP(groupA)
       PORT(tcpipPort)
       SSL(CLIENTAUTH) AUTHENTICATE(CERTIFICATE)
       URM(NONE) PROTOCOL(HTTP) TRANSACTION(CWXN)
```

5. Define the PIPELINE.

   The PIPELINE proves 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/htp/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:

   ```
   RDEFINE GCICSTRN PIPEUSER UACC(NONE) +
            ADDMEM(CPIA,CPIH,CPIL,CPIQ) +
   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 *taskUserGroup*.

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:

```
DFHSO0107 02/07/2022 13:27:00 IYK2ZGV1 TCPIPSERVICE 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.
DFHWB1560 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 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 TCPIPSERVICE 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 297
- "Security design considerations for CICS web service requesters" on page 297

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 82 on page 297, the CICS web service requester sends a request to a web service provider.

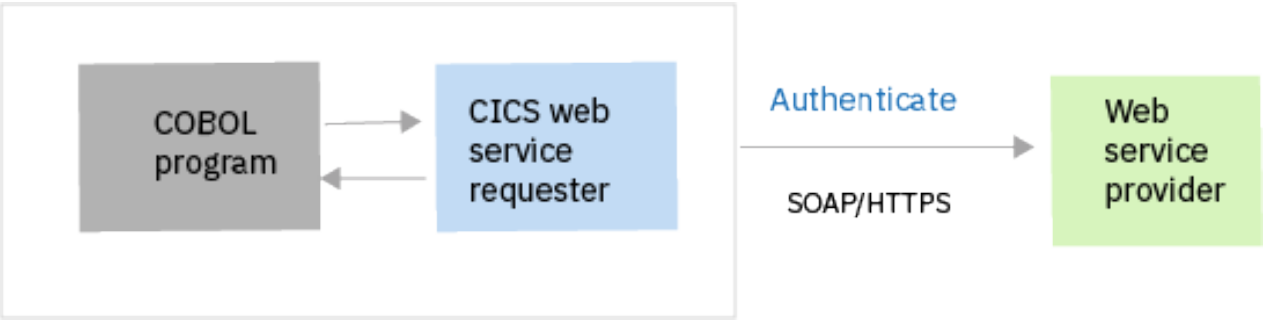


Figure 82. Direct request to the web service provider

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 Example: Designing a secure 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 Example: Designing to assert an identity to the CICS web service provider).

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.

Recommended: 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 more information about configuring this scenario, see the configuration task Configuration example: Securing a web service request with TLS client authentication.

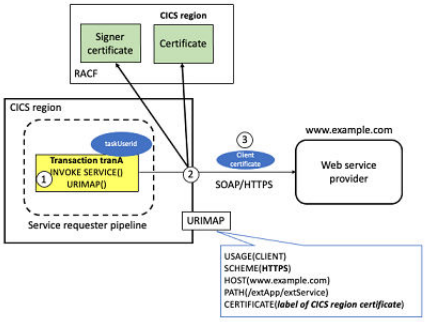


Figure 83. TLS authentication in a web service requester scenario

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 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 Example: Designing a secure 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(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 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA URIMAP urimapA has been
created.
DFHPI0701 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA PIPELINE pipelineA has been
created.
DFHPI0204 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA New WEBSERVICE webserviceRequesterA is
being created during a scan against
pipelineA.

DFHPI0910 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA WEBSERVICE webserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 03/25/2022 21:21:18 IYK2ZGV2 regionUseridA WEBSERVICE webserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 03/25/2022 21:21:18 IYK2ZGV2 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 more information about configuring this scenario, see the configuration task Configuration example: Securing a web service request with identity assertion.

Figure 84 on page 302 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.

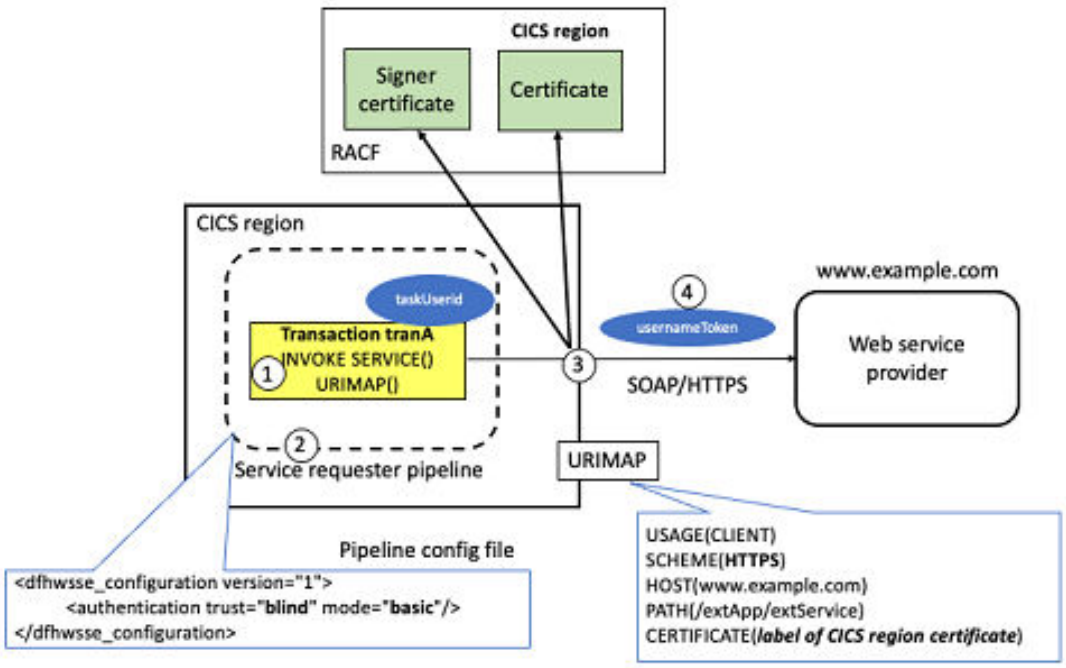


Figure 84. Identity assertion in a web service requester scenario

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 84 on page 302:

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 (see TLS client authentication).
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.
- *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.
- *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/htp/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 IYK2ZGV2 regionUseridA PIPELINE pipelineA has been
created.
DFHPI0204 I 04/05/2022 15:26:10 IYK2ZGV2 regionUseridA PIPELINE pipelineA is now ENABLED and is
ready for use.
DFHPI0703 I 04/05/2022 15:26:10 IYK2ZGV2 regionUseridA PIPELINE pipelineA is about to scan the
WSDIR directory.
DFHPI0901 I 04/05/2022 15:26:10 IYK2ZGV2 regionUseridA New WEBSERVICE webserviceRequesterA is
being created during a scan against PIPELINE
pipelineA.

DFHPI0910 I 04/05/2022 15:26:10 IYK2ZGV2 regionUseridA WEBSERVICE webserviceRequesterA within
PIPELINE pipelineA has been created.
DFHPI0915 I 04/05/2022 15:26:10 IYK2ZGV2 regionUseridA WEBSERVICE webserviceRequesterA is now
INSERVICE and is ready for use.
DFHPI0704 I 04/05/2022 15:26:10 IYK2ZGV2 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 21 on page 306.

Table 21. Configuration activities to set up security for CICS web services

Activity	For web service providers	For web service requesters	Instructions
Review prerequisites for implementing WS-Security	✓	✓	Installing the prerequisites for WS-Security support
Configure a pipeline for SOAP message security	✓	✓	Configuring the pipeline for Web Services Security
Configure ICRX-based identity propagation	✓	n/a	Configuring provider mode web services for identity propagation
Sign or encrypt SOAP messages	✓	✓	Configuring RACF for Web Services Security and the instructions for signing or encrypting SOAP messages in Configuring the pipeline for Web Services Security
Use the Trust client interface to invoke a Security Token Service (STS)	✓	✓	Invoking the Trust client from a message handler
Write a custom security handler to secure SOAP messages	✓	✓	Writing a custom security handler

Installing the prerequisites for WS-Security support

To implement WS-Security support in CICS, you must install the IBM XML Toolkit for z/OS v1.10 and add 3 libraries to the DFHRPL concatenation.

Procedure

1. Install the free IBM XML Toolkit for z/OS v1.10.

   You can download it from the following site: https://www.ibm.com/servers/eserver/zseries/software/xml/. You must install version 1.10. Later versions 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 CICSTS61.

   The first two libraries contain DLLs that are required at run time by the CICS security handler. IXM4C57 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/xmldsig#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>
```

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

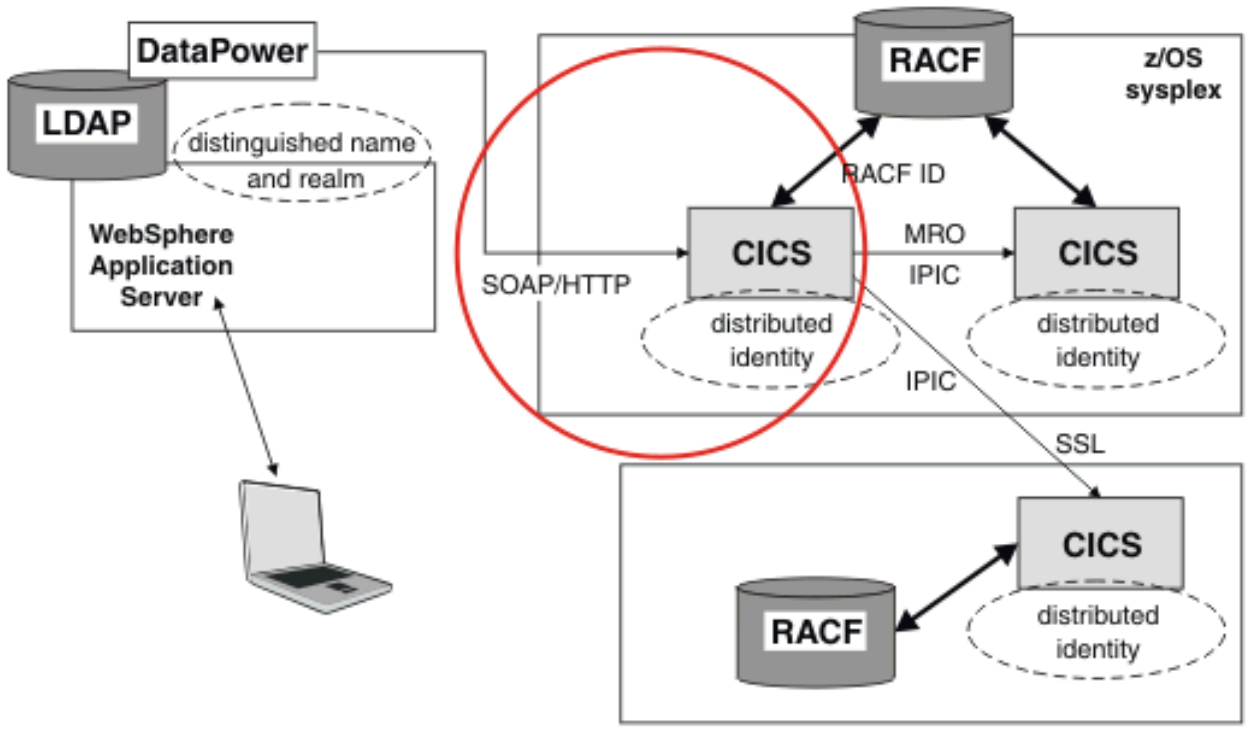
Configure your RACF RACMAP settings before you configure your web service connections. Otherwise, you receive the RACF ICH408I message for every unmapped request that is sent to RACF. For more information about configuring the RACF **RACMAP** command, see Configuring RACF for identity propagation.

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 85. 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/htp/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-STSURI, 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 313 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.

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.
- Confidentially 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.

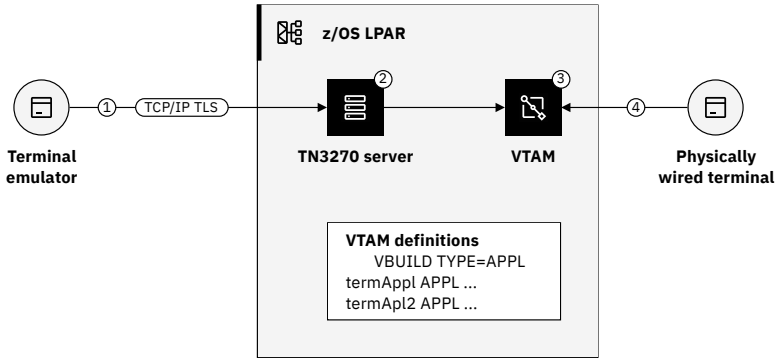


Figure 86. 3270 connection for physical terminals and emulators

Figure 86 on page 318 shows how a terminal emulator or physically wired terminal 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 *termApl2*.

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 87 on page 319. For more information about 3270 terms used in this topic, see How it works: 3270 security.

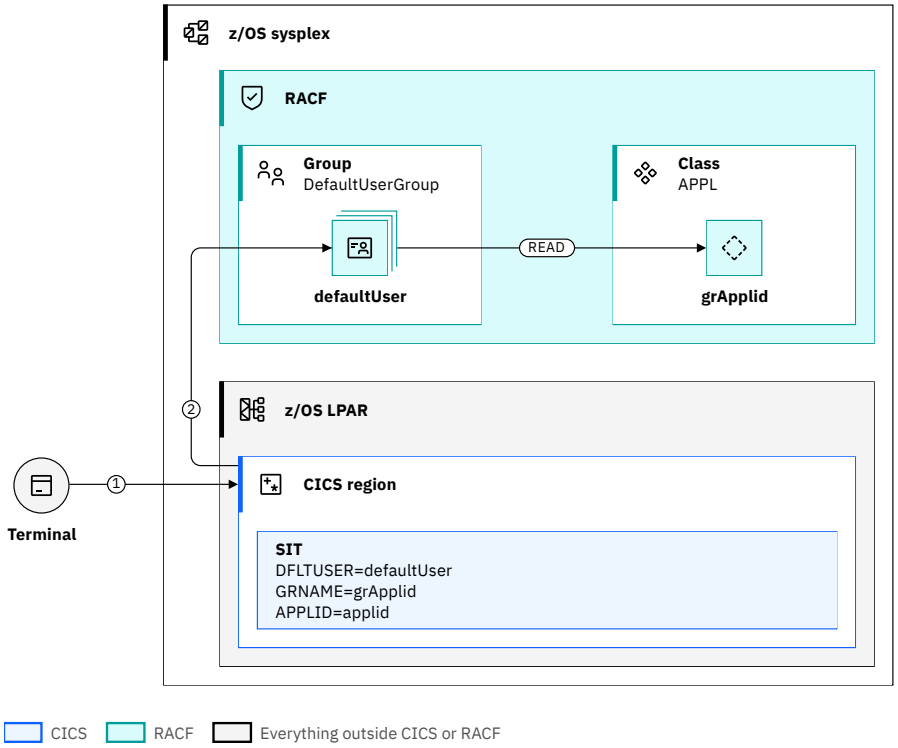


Figure 87. Region startup checks for defaultUser

As part of the CICS start process, shown in Figure 87 on page 319:

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 319
- “Statically defined 3270 terminals” on page 320
- “3270 consoles” on page 322

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.

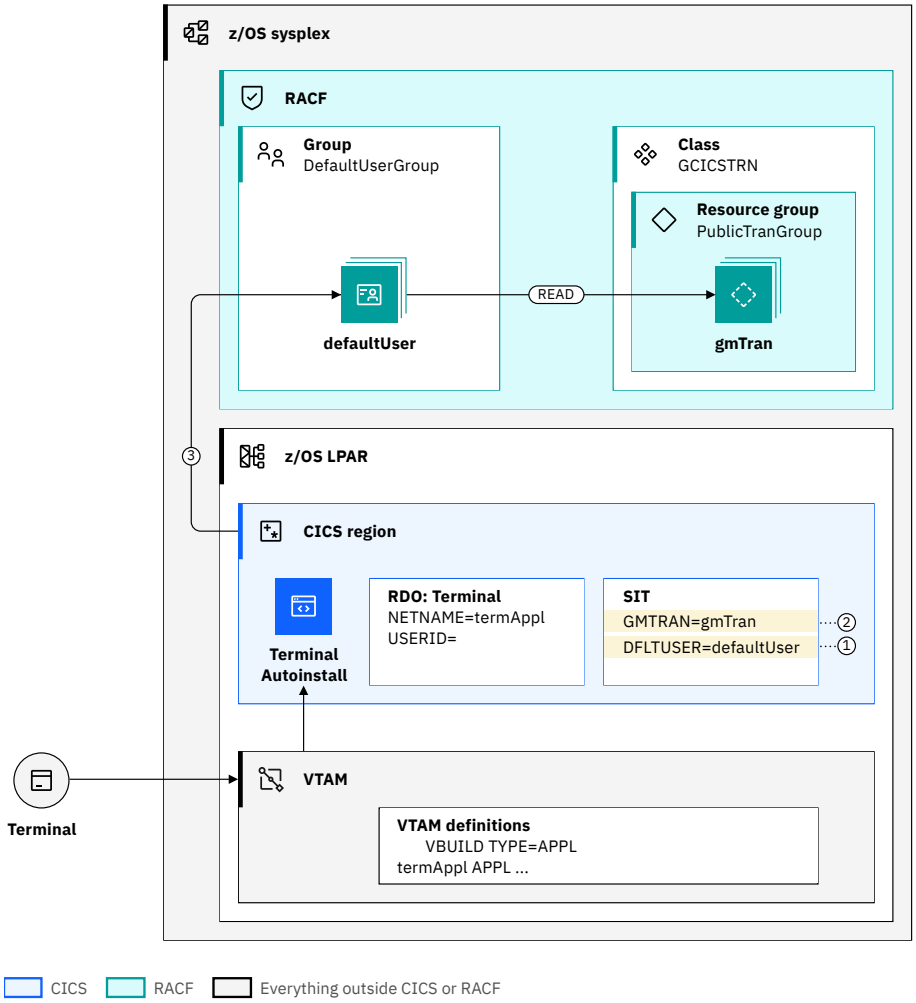


Figure 88. Dynamically defined 3270 connection

Figure 88 on page 320 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 89 on page 321.
2. When the device connects to CICS, as shown in Figure 90 on page 322.

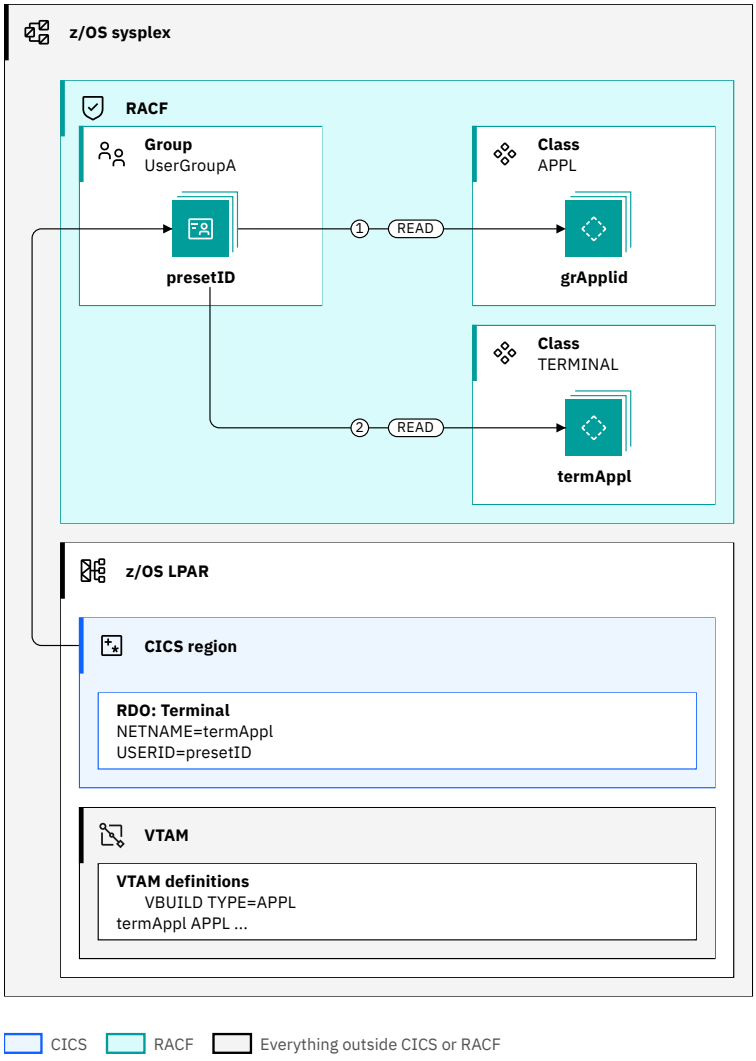


Figure 89. Statically defined 3270 terminal configuration

As shown in Figure 89 on page 321, 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.

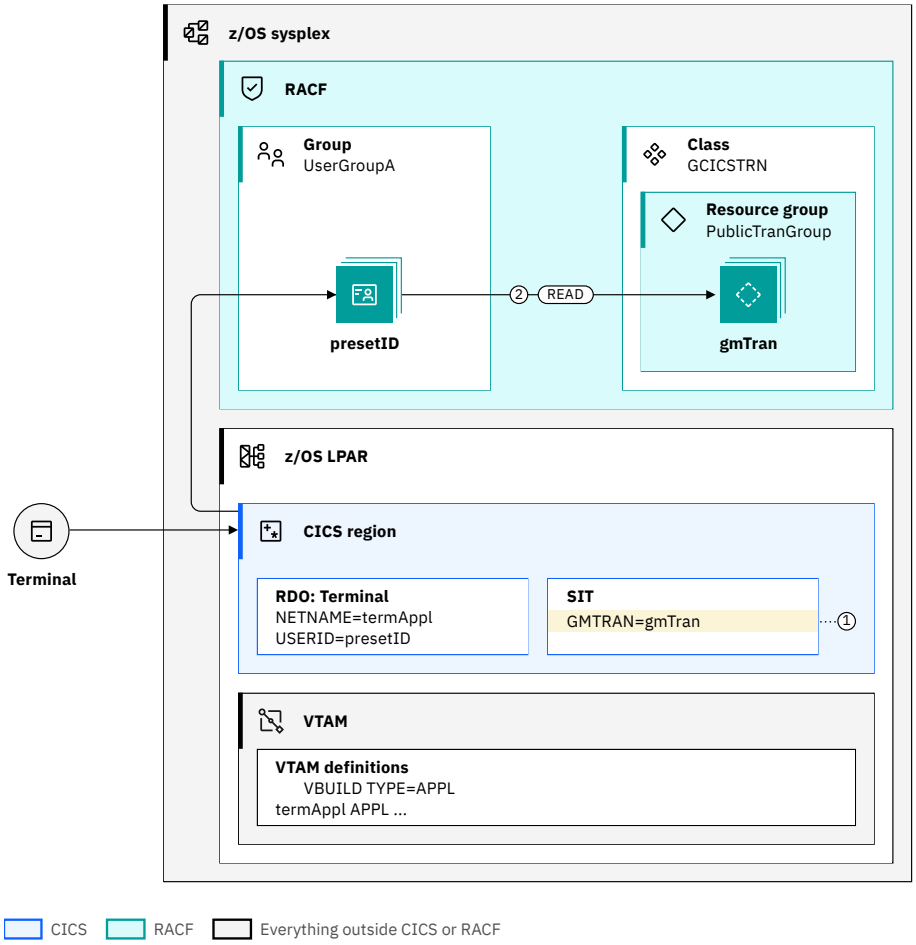


Figure 90. Statically defined 3270 connection

Figure 90 on page 322 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.

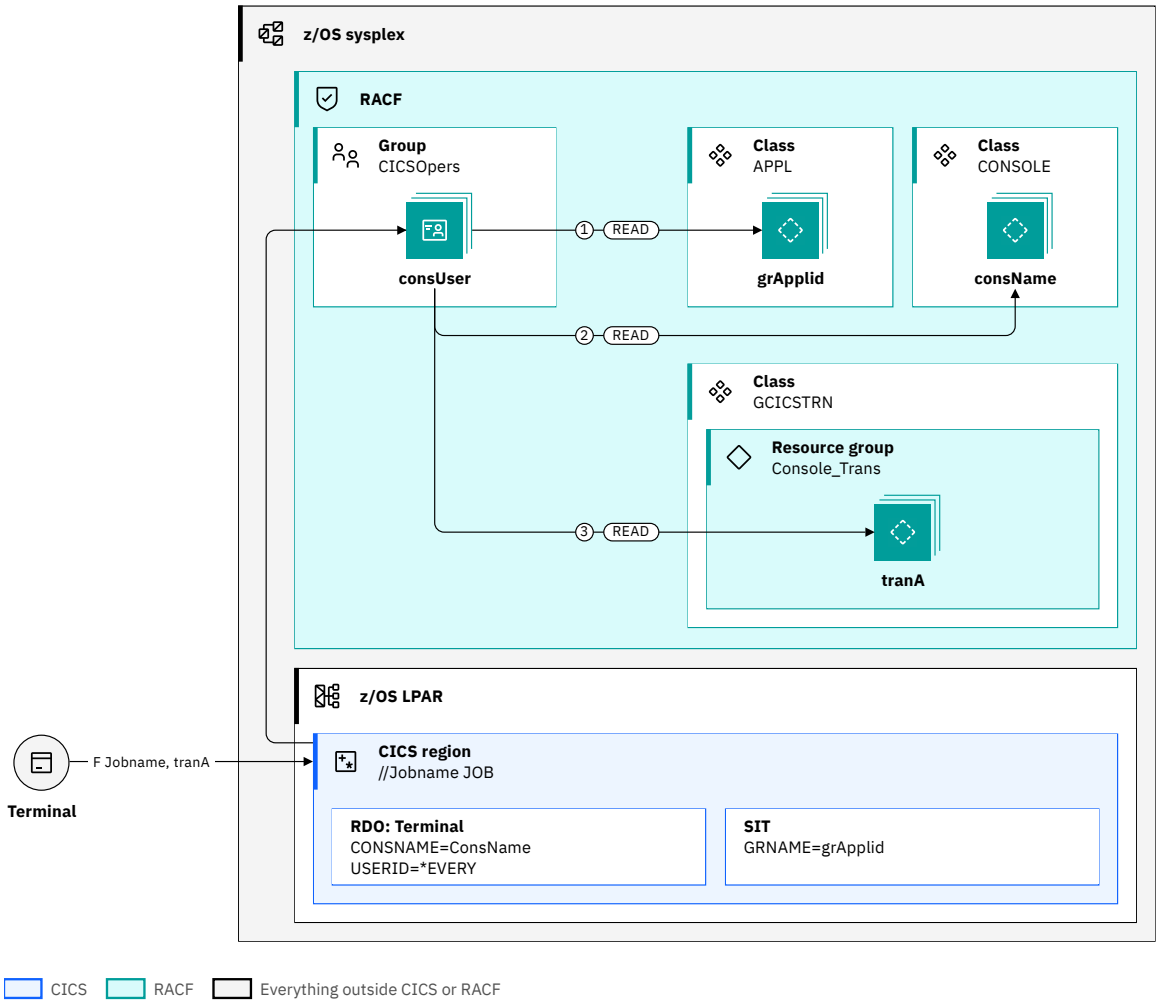


Figure 91. Physical console connection

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 91 on page 323 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:

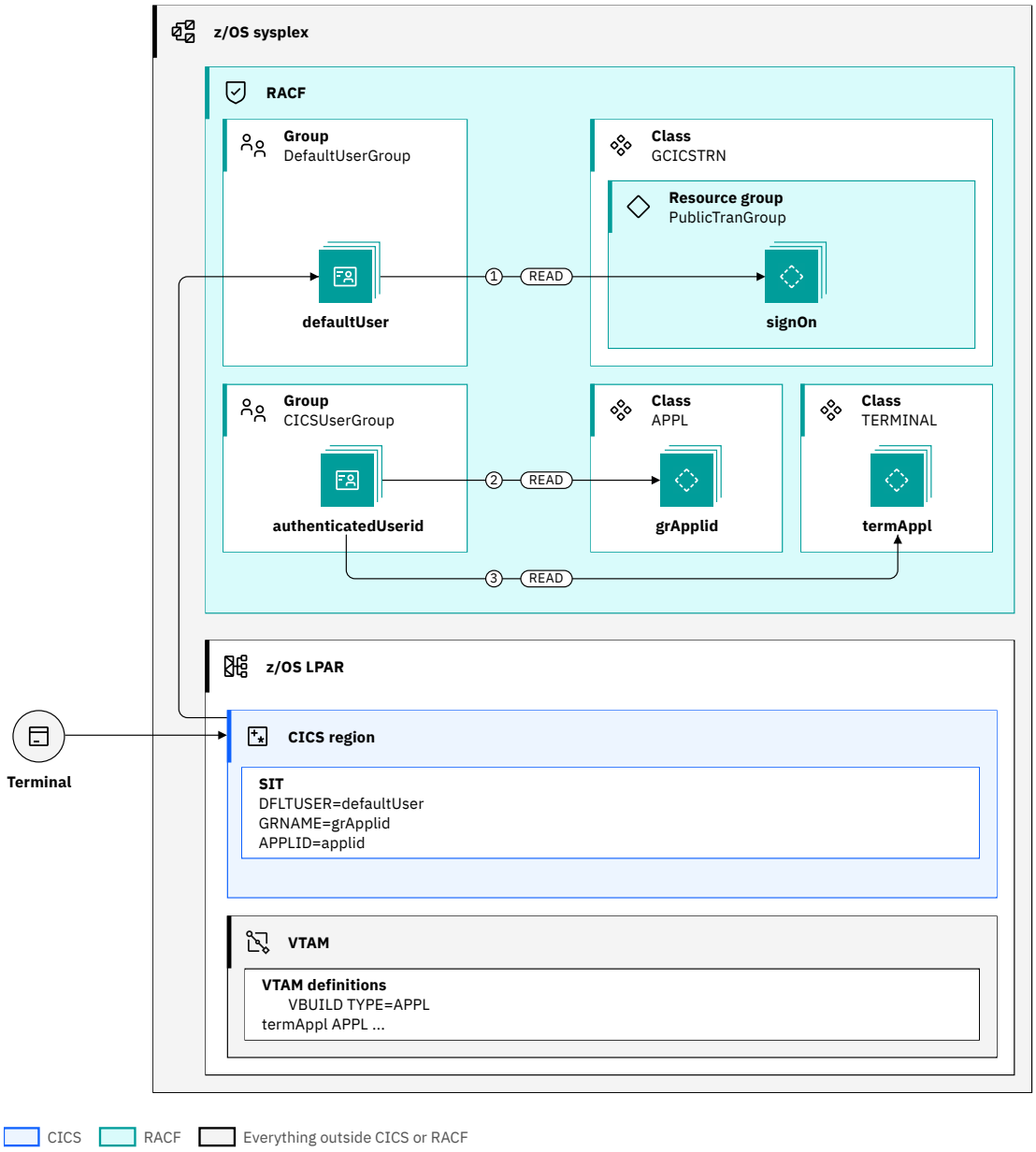


Figure 92. 3270 authentication checks

As shown in Figure 92 on page 324, 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.

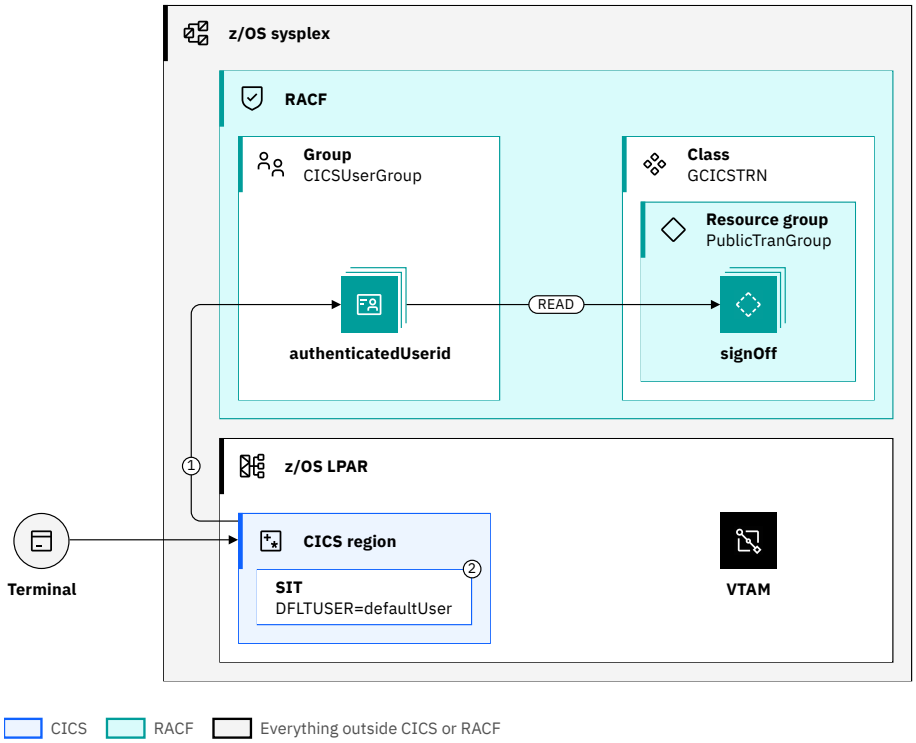


Figure 93. 3270 terminal sign off process

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*.

Transaction *signOff* can also disconnect the terminal from CICS, for example, by use of **EXEC CICS ISSUE DISCONNECT**.

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:

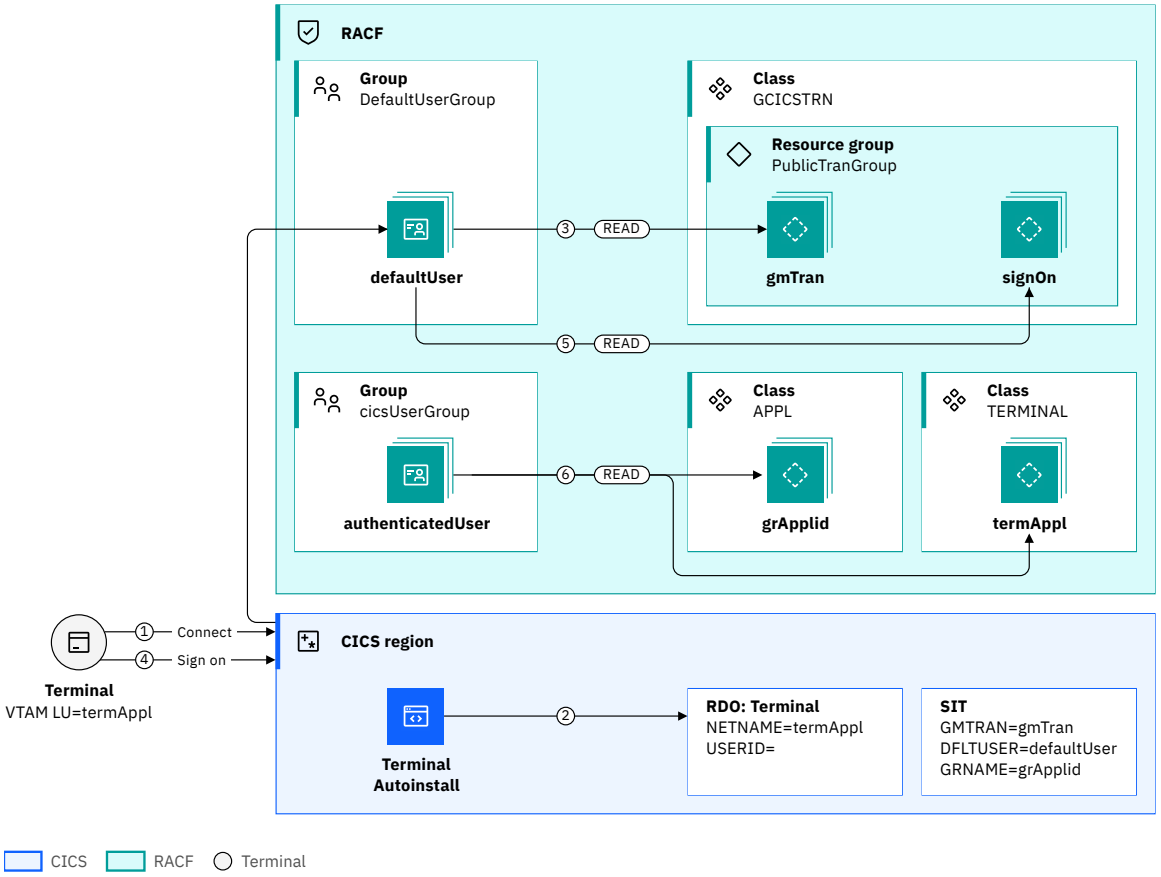


Figure 94. 3270 dynamically defined 3270 terminal connects and authenticates to a CICS region

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 access to resource *gmTran* in class GCICSTRN.
4. The terminal user starts transaction *signOn* to authenticate, supplying the credentials for *authenticatedUser*.
5. CICS checks whether *defaultUser* has access to resource *signOn* in class GCICSTRN.
6. CICS authenticates the supplied credentials and then checks whether *authenticatedUser* has access to resource *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, TCICSTRN, 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 DFTUSER, 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 GCICSTRAN UACC(READ) ADDMEM(gmTran)
   SETROPTS RACLIST(TCICSTRAN) REFESH
   ```

   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 GCICSTRAN publicTranGroup ADDMEM(signOn)
SETROPTS RACLIST(TCICSTRAN) REFESH
```

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) REFESH
```

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:

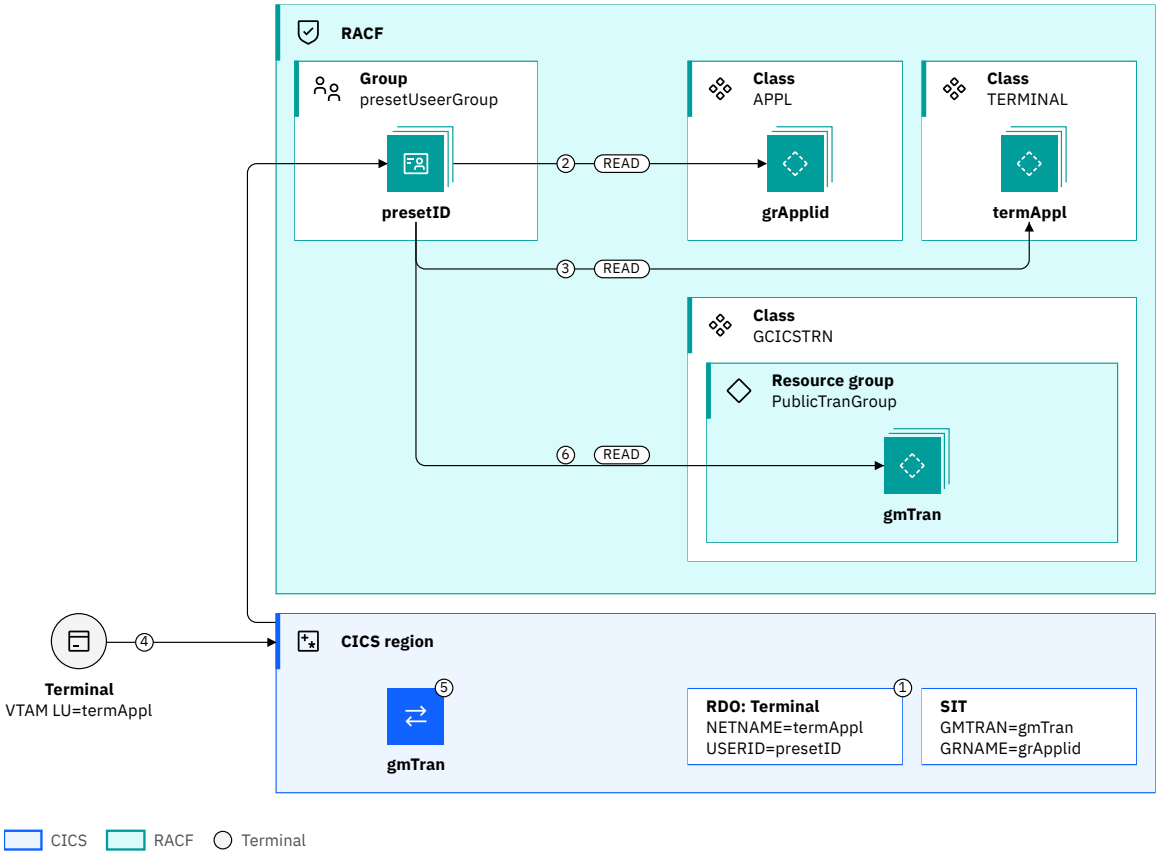


Figure 95. Connection flow for 3270 with a preset user ID

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.

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, TCICSTRN, 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.
- *presetUsewrGroup* 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.

   ```
   REDFINE 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.

   ```
   REDFINE publicTranGroup GCICSTRAN UACC(READ) ADDMEM(gmTran)
   SETROPTS RACLIST(TCICSTRAN) 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 had 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
- Design example: 3270 terminal with a preset user ID connects

The matching configuration examples are described here:

- 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 385. 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 22. 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 22. 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 23 on page 338 is a sample of a completed security matrix for a production CICSplex:

Table 23. 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 349 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 tasks: security” on page 386 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.

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 24 on page 340 lists the CICSPlex SM data sets and the minimum access that should be granted to each type of user ID.

Table 24. Access by user ID for CICSPlex SM data sets

Data set name	CMAS	MAS	System Admin.	Individual User
SYS1.CICSTS61.CPSM.SEYULPA	NONE	READ	UPDATE	NONE
SYS1.CICSTS61.CPSM.SEYULINK	READ	NONE	UPDATE	NONE
CICSTS61.CPSM.SEYUAUTH	READ	READ	UPDATE	READ
CICSTS61.CPSM.SEYULOAD	READ	READ	UPDATE	NONE
CICSTS61.CPSM.SEYUPARM	READ	READ	UPDATE	NONE
CICSTS61.CPSM.SEYUCOB	NONE	NONE	UPDATE	READ
CICSTS61.CPSM.SEYUC370	NONE	NONE	UPDATE	READ
CICSTS61.CPSM.SEYUDEF	READ	READ	UPDATE	READ
CICSTS61.CPSM.SEYUCLIB	NONE	NONE	UPDATE	READ
CICSTS61.CICS.SDFHINST	NONE	NONE	UPDATE	NONE
CICSTS61.CPSM.SEYUMAC	NONE	NONE	UPDATE	READ
CICSTS61.CPSM.SEYUPL1	NONE	NONE	UPDATE	READ
CICSTS61.CPSM.SEYUPROC	NONE	NONE	UPDATE	READ
CICSTS61.CPSM.SEYUSAMP	NONE	NONE	UPDATE	READ
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

Table 25. CMAS agent user ID

CICSPlex SM release	How was the CMAS agent initialized?	PLTPIUSR in system initialization table?	CMAS agent user ID
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 “What governs the agent user ID?” on page 341)

Table 26. MAS agent user ID			
CICSPlex SM release	**How was the MAS agent initialized?**	**PLTPIUSR in system initialization table?**	**MAS agent user ID**
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 “What governs the agent user ID?” on page 341)

Table 27. WUI agent user ID			
CICSPlex SM release	**How was the WUI agent initialized?**	**PLTPIUSR in system initialization table?**	**WUI agent user ID**
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 “What governs the agent user ID?” on page 341)

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 28 on page 344 summarizes the security accesses required by users of the Web User Interface.

Table 28. 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 344.
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 29. 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 UACC(NONE) ADDMEM(COVP,COVU,COVE)
RDEFINE GCICSTRN WUIADMIN UACC(NONE) ADDMEM(COVG,COVC)
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_pltpiusr,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.

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 30 on page 352 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:

```
SETROPTS GENERIC(CPSMOBJ,CPSMXMP)
```

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 30. Function and type combinations for resources accessible using CICSPlex SM	
Function.type combination	**Resource Table Name and Usage**
ANALYSIS.DEF	**ACTION** Create, display, and maintain action definitions. **APACTV** Display analysis definitions associated with an analysis point specification. **APSPEC** Create, display, and maintain analysis point specifications. **CMDMPAPS** Identify the role of a primary CMAS. **CMDMSAPS** Identify the role of a secondary CMAS. **EVALDEF** Create, display, and maintain evaluation definitions. **EVENT** Display changes in the status of a CICSplex. **EVENTDTL** Display evaluation definitions associated with an analysis definition that caused an event. **LNKSRSCG** Describe the link between a CICS system group and an analysis specification. **LNKSRSCS** Describe the link between a CICS system and an analysis specification. **RTAACTV** Display analysis and status definitions in CICS systems. **RTADEF** Create, display, and maintain analysis definitions. **RTAGROUP** Create, display, and maintain analysis groups. **RTAINAPS** Display analysis groups in analysis point specifications. **RTAINGRP** Display analysis and status definitions in analysis groups. **RTAINSPC** Display analysis groups in analysis specifications. **RTASPEC** Create, display, and maintain analysis specifications. **STATDEF** Create, display, and maintain status definitions. **STAINGRP** Identify the membership relation of a status definition in an RTAGROUP.
BAS.APPLICTN	**BUNDDEF** Install applications that are deployed as bundles.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.CONNECT	**CONNDEF** Install ISC/MRO connection definitions. **IPCONDEF** Install IPIC connection definitions. **SESSDEF** Install session definitions.
BAS.DB2DBCTL	**DB2CDEF** Install Db2 connection definitions. **DB2EDEF** Install Db2 entry definitions. **DB2TDEF** Install Db2 transaction definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF	**ATOMDEF** Create, display, and maintain Atom service definitions. **BUNDDEF** Create, display, and maintain bundle definitions. **CONNDEF** Create, display, and maintain MRO and ISC over SNA connection definitions. **DB2CDEF** Create, display, and maintain Db2 connection definitions. **DB2EDEF** Create, display, and maintain Db2 entry definitions. **DB2TDEF** Create, display, and maintain Db2 transaction definitions. **DOCDEF** Create, display, and maintain document template definitions. **ENQMDEF** Create, display, and maintain enqueue models definitions. **FENODDEF** Create, display, and maintain FEPI node definitions. **FEPOODEF** Create, display, and maintain FEPI pool definitions. **FEPRODEF** Create, display, and maintain FEPI property set definitions. **FETRGDEF** Create, display, and maintain FEPI target definitions. **FILEDEF** Create, display, and maintain file definitions. **IPCONDEF** Create, display, and maintain IPIC connection definitions. **JRNMDEF** Create, display, and maintain journal model definitions. **JVMSVDEF** Create, display, and maintain JVM server definitions. **LIBDEF** Create, display, and maintain LIBRARY definitions. **LSRDEF** Create, display, and maintain LSR pool definitions. **MAPDEF** Create, display, and maintain mapset definitions. **MQCONDEF** Create, display, and maintain IBM MQ connection definitions. **MQMONDEF** Create, display, and maintain IBM MQ monitor definitions. **PARTDEF** Create, display, and maintain partner definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
BAS.DEF (continued)	**PIPEDEF** Create, display, and maintain pipeline definitions. **PROCDEF** Create, display, and maintain process type definitions. **PROFDEF** Create, display, and maintain profile definitions. **PROGDEF** Create, display, and maintain program definitions. **PRTNDEF** Create, display, and maintain partition set definitions. **RASGNDEF** Create, display, and maintain resource assignments. **RASINDSC** Display resource assignments in descriptions. **RASPROC** Display resource assignment process. **RDSCPROC** Display resource description process. **RESDESC** Create, display, maintain, and install resource descriptions. **RESGROUP** Create, display, maintain, and install resource groups. **RESINDSC** Display resource groups in descriptions. **RESINGRP** Display resource definitions in groups. **RQMDEF** Create, display, and maintain request model definitions. **SESSDEF** Create, display, and maintain session definitions. **SYSRES** Display CICS system resources. **TCPDEF** Create, display, and maintain TCP/IP service definitions. **TDQDEF** Create, display, and maintain transient data queue definitions. **TERMDEF** Create, display, and maintain terminal definitions. **TRANDEF** Create, display, and maintain transaction definitions. **TRNCLDEF** Create, display, and maintain transaction class definitions. **TYPTMDEF** Create, display, and maintain typeterm definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
BAS.DEF (continued)	**URIMPDEF** Create, display, and maintain URI map definitions. **WEBSVDEF** Create, display, and maintain web services definitions. **ATMINGRP** Describe the membership of an Atom service definition in a resource group. **BUNINGRP** Describe the membership of a bundle definition in a resource group. **CONINGRP** Describe the membership of an MRO or ISC over SNA connection definition in a resource group. **DOCINGRP** Describe the membership of a document template definition in a resource group. **D2CINGRP** Describe the membership of a Db2 connection definition in a resource group. **D2EINGRP** Describe the membership of a Db2 entry definition in a resource group. **D2TINGRP** Describe the membership of a Db2 transaction definition in a resource group. **ENQINGRP** Describe the membership of an ENQ/DEQ model definition in a resource group. **FILINGRP** Describe the membership of a file definition in a resource group. **FNOINGRP** Describe the membership of a FEPI node definition in a resource group. **FPOINGRP** Describe the membership of a FEPI pool definition in a resource group. **FPRINGRP** Describe the membership of a FEPI property set definition in a resource group. **FSGINGRP** Describe the membership of a file key segment definition in a resource group. **FTRINGRP** Describe the membership of a FEPI target definition in a resource group. **IPCINGRP** Describe the membership of an IPIC connection definition in a resource group. **JMSINGRP** Describe the membership of a JVM server definition in a resource group. **JRMINGRP** Describe the membership of a journal model definition in a resource group. **LIBINGRP** Describe the membership of a LIBRARY definition in a resource group.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.DEF (continued)	**LSRINGRP** Describe the membership of an LSR pool definition in a resource group. **MAPINGRP** Describe the membership of a map set definition in a resource group. **MQCINGRP** Display IBM MQ connection definitions in groups. **PARINGRP** Describe the membership of a partner definition in a resource group. **PGMINGRP** Describe the membership of a program definition in a resource group. **PIPINGRP** Describe the membership of a pipeline definition in a resource group. **PRCINGRP** Describe the membership of a process type definition in a resource group. **PRNINGRP** Describe the membership of a partition set definition in a resource group. **PROINGRP** Describe the membership of a profile definition in a resource group. **SESINGRP** Describe the membership of a session definition in a resource group. **TCLINGRP** Describe the membership of a transaction class definition in a resource group. **TCPINGRP** Describe the membership of a TCPIP Service definition in a resource group. **TDQINGRP** Describe the membership of a transient data queue definition in a resource group. **TRMINGRP** Describe the membership of a terminal definition in a resource group. **TRNINGRP** Describe the membership of a transaction definition in a resource group. **TSMINGRP** Describe the membership of a temporary storage model definition in a resource group. **TYPINGRP** Describe the membership of a typeterm definition in a resource group. **URIINGRP** Describe the membership of a URI map definition in a resource group. **WEBINGRP** Describe the membership of a web service definition in a resource group.
BAS.DOCTEMP	**DOCTEMP** Install document template definitions.
BAS.ENQMODEL	**ENQMDEF** Install enqueue model definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
BAS.ENTJAVA	**JVMSVDEF** Install JVM server definitions.
BAS.FILE	**FILEDEF** Install file definitions.
BAS.JOURNAL	**JRNMDEF** Install journal model definitions.
BAS.PARTNER	**PARTDEF** Install partner definitions.
BAS.PROCTYPE	**PROCDEF** Install BTS Process type definitions.
BAS.PROFILE	**PROFDEF** Install profile definitions.
BAS.PROGRAM	**LIBDEF** Install LIBRARY definitions. **MAPDEF** Install map set definitions. **PROGDEF** Install program definitions. **PRTNDEF** Install partition set definitions.
BAS.REGION	**LSRDEF** Install LSR pool definitions. **MQCONDEF** Install IBM MQ connection definitions. **MQMONDEF** Install IBM MQ monitor definitions. **TRNCLDEF** Install transaction class definitions.
BAS.TCPIPS	**ATOMDEF** Install Atom service definitions. **PIPEDEF** Install pipeline definitions. **TCPDEF** Install TCPIP service definitions. **URIMPDEF** Install URI map definitions. **WEBSVDEF** Install web services definitions.
BAS.TDQUEUE	**TDQDEF** Install transient data queue definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
BAS.TERMINAL	**TERMDEF** Install terminal definitions. **TYPTMDEF** Install typeterm definitions.
BAS.TRAN	**TRANDEF** Install transaction definitions.
CLOUD.APPLICATION	**APPLDEF** Install application definitions in platforms. **APPLCTN** Display and manage applications deployed in platforms.
CLOUD.DEF	**APPLDEF** Create, display, and maintain application definitions in platforms. **PLATDEF** Create, display, and maintain platform definitions.
CLOUD.PLATFORM	**PLATDEF** Install platform definitions. **PLATFORM** Display and manage platforms. **MGMTPART** Create, display, and maintain management parts for applications and platforms.
CONFIG.DEF	**CICSPLEX** Display and manage CMAS in CICSplex. **CMAS** Display and manage active CMASs. **CMASLIST** Describe a CMAS and its connection to characteristics. **CMASPLEX** Display CICSplexes for a CMAS. **CMTCMDEF** Create, display, and maintain CMAS links. **CMTCMLNK** Display active CMAS links. **CMTPMLNK** Display an active CMAS-to-MAS link. **CPLEXDEF** Create, display, and maintain CICSplex definitions. **CPLXCMAS** Display CMAS to CICSplex.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
CSD.DEF	**ATOMDEF** Create, install, display, and maintain Atom service definitions. **BUNDDEF** Create, install, display, and maintain bundle definitions. **CONNDEF** Create, install, display, and maintain MRO and ISC over SNA connection definitions. **CSDGROUP** Create, install, display, and maintain CSD resource group definitions. **CSDINGRP** Describe the membership of a CSD resource group. **CSDINLST** Describe the membership of a CSD resource group list. **CSDLIST** Create, install, display, and maintain CSD resource group list definitions. **DB2CDEF** Create, install, display, and maintain Db2 connection definitions. **DB2EDEF** Create, install, display, and maintain Db2 entry definitions. **DB2TDEF** Create, install, display, and maintain Db2 transaction definitions. **DOCDEF** Create, install, display, and maintain document template definitions. **ENQMDEF** Create, install, display, and maintain enqueue models definitions. **FILEDEF** Create, install, display, and maintain file definitions. **IPCONDEF** Create, install, display, and maintain IPIC connection definitions. **JRNMDEF** Create, install, display, and maintain journal model definitions. **JVMSVDEF** Create, install, display, and maintain JVM server definitions. **LIBDEF** Create, install, display, and maintain LIBRARY definitions. **LSRDEF** Create, install, display, and maintain LSR pool definitions. **MAPDEF** Create, install, display, and maintain mapset definitions. **MQCONDEF** Create, install, display, and maintain IBM MQ connection definitions. **MQMONDEF** Create, install, display, and maintain IBM MQ monitor definitions.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
CSD.DEF (continued)	**PARTDEF** Create, install, display, and maintain partner definitions. **PIPEDEF** Create, install, display, and maintain pipeline definitions. **PROCDEF** Create, install, display, and maintain process type definitions. **PROFDEF** Create, install, display, and maintain profile definitions. **PROGDEF** Create, install, display, and maintain program definitions. **PRTNDEF** Create, install, display, and maintain partition set definitions. **SESSDEF** Create, display, and maintain session definitions. **TCPDEF** Create, install, display, and maintain TCP/IP service definitions. **TDQDEF** Create, install, display, and maintain transient data queue definitions. **TERMDEF** Create, install, display, and maintain terminal definitions. **TRANDEF** Create, install, display, and maintain transaction definitions. **TRNCLDEF** Create, install, display, and maintain transaction class definitions. **TSMINGRP** Describe the membership of a temporary storage model definition in a resource group. **TYPTMDEF** Create, install, display, and maintain typeterm definitions. **URIMPDEF** Create, install, display, and maintain URI map definitions. **WEBSVDEF** Create, install, display, and maintain web services definitions.
MONITOR.CONNECT	**MCONNECT** ISC and MRO connections **MMODENAME** LU 6.2 modenames
MONITOR.DB2DBCTL	**MDB2THRD** Db2 threads

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
MONITOR.DEF	**LNKSMSCG** Describe the link between a CICS system group and a monitor specification. **LNKSMSCS** Describe the link between a CICS system and a monitor specification. **MONDEF** Create, display, and maintain monitor definitions. **MONGROUP** Create, display, and maintain monitor groups. **MONINGRP** Create, display, and maintain monitor definitions in monitor groups. **MONINSPC** Create, display, and maintain monitor groups in monitor specifications. **MONSPEC** Create, display, and maintain monitor specifications. **POLMON** Describe a monitor definition in a specific CICS system.
MONITOR.FEPI	**MFEPICON** FEPI connections
MONITOR.FILE	**MCMDT** Data tables **MLOCFILE** Local files **MREMFILE** Remote files
MONITOR.JOURNAL	**MJOURNL** Journals **MJRNLNAM** Journal names
MONITOR.PROGRAM	**MPROGRAM** Programs
MONITOR.REGION	**MCICSDSA** Dynamic storage areas **MCICSRGN** CICS systems **MLSRPBUF** LSRPOOL buffer pool **MLSRPOOL** LSRPOOL **MTRANCLS** Transaction classes

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
MONITOR.TDQUEUE	**MINDTDQ** Indirect transient data queues **MNTRATDQ** Intrapartition transient data queues **MREMTDQ** Remote transient data queues **MTDQGBL** Global intrapartition transient data queues **MXTRATDQ** Extrapartition transient data queues
MONITOR.TERMINAL	**MTERMNL** Terminals
MONITOR.TRAN	**MLOCTRAN** Local transactions **MREMTRAN** Remote transactions
MONITOR.TSQUEUE	**MTSQGBL** Global temporary storage queues
OPERATE.AIMODEL	**AIMODEL** Autoinstall models **CRESAIMD** Describe an instance of an autoinstalled terminal model in a CICS system.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.APPLICTN	**BUNDLE** A CICS resource object that provides information about bundles. **BUNDPART** A CICS resource object that provides information about a bundle part. **CRESBUND** Describes an instance of a bundle in a CICS system. **CRESXMLT** Describes an instance of an XMLTRANSFORM resource in a CICS system. **EPADAPT** A CICS resource object that provides information about EP adapters. **EPADSET** A CICS resource object that provides information about EP adapter sets. **EPAINSET** A CICS resource object that provides the name of event processing adapters in an event processing adapter set. **EVCSDATA** A CICS resource object that provides information about application data predicates. **EVCSINFO** A CICS resource object that provides information about information sources. **EVCSOPT** A CICS resource object that provides information about application option predicates. **EVCSPEC** A CICS resource object that provides information about capture specifications. **EVNTBIND** A CICS resource object that provides information about event bindings. **EVNTGBL** A CICS resource object that provides information about event processing. **NODEJSAP** A CICS resource object that provides information about a Node.js application. **OSGIBUND** A CICS resource object that provides information about an OSGi bundle. **OSGISERV** A CICS resource object that provides information about an OSGi service. **XMLTRANS** XMLTRANSFORM resources.
OPERATE.BRFACIL	**BRFACIL** LINK3270 bridge facility

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
OPERATE.CONNECT	**CONNECT** MRO and ISC over SNA connections **CRESCONN** Describe an instance of an MRO or ISC over SNA connection in a CICS system. **CRESIPCN** Describe an instance of an IPIC connection in a CICS system. **CRESMODE** Describe an instance of an LU6.2 modename in a CICS system. **IPCONN** IPIC connections **MODENAME** LU 6.2 modenames
OPERATE.DB2DBCTL	**CRESDB2C** Describe an instance of a Db2 connection in a CICS system. **CRESDB2E** Describe an instance of a Db2 entry in a CICS system. **CRESDB2P** Describe an instance of a Db2 package set in a CICS region. **CRESDB2T** Describe an instance of a Db2 transaction in a CICS system. **DB2CONN** Db2 connection **DB2ENTRY** Db2 entry **DB2PKGST** Db2 package set **DB2TRN** Db2 transaction **DBCTLSS** DBCTL subsystem **DB2SS** Db2 subsystem **DB2THRD** Db2 threads **DB2TRAN** Db2 transactions
OPERATE.DOCTEMP	**CRESDOCT** Describe an instance of a document template in a CICS system. **DOCTEMP** Document templates
OPERATE.ENQMODEL	**CRESENQM** Describe an instance of an ENQ/DEQ model in a CICS system. **ENQMODEL** Enqueue models

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.ENTJAVA	**CLCACHE** Shared class caches **CRESJVMS** Describe an instance of a JVM server in a CICS region. **ENDPOINT** JVM Endpoints **JVM** Java virtual machines **JVMPOOL** JVM pools **JVMPROF** JVM profiles **JVMSERV** JVM servers
OPERATE.EXIT	**CRESGLUE** Describe an instance of a global user exit in a CICS system. **CRESTRUE** Describe an instance of a task-related user exit in a CICS system. **EXITGLUE** Global user exits **EXITTRUE** Task-related user exits **EXTGLORD** Ordered global user exits
OPERATE.FEPI	**CRESFECO** Describe an instance of a FEPI connection in a CICS system. **CRESFEND** Describe an instance of a FEPI node in a CICS system. **CRESFEPO** Describe an instance of a FEPI pool in a CICS system. **CRESFETR** Describe an instance of a FEPI target in a CICS system. **FEPICONN** FEPI connections **FEPINODE** FEPI nodes **FEPIPOOL** FEPI pools **FEPIPROP** FEPI property sets **FEPITRGT** FEPI targets

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.FILE	**CFDTPOOL** Coupling facility data tables pools **CMDT** Data tables **CRESDSNM** Describe an instance of a data set in a CICS system. **CRESFILE** Describe an instance of a file in a CICS system. **DSNAME** Data sets **LOCFILE** Local files **REMFILE** Remote files
OPERATE.JOURNAL	**CRESJRNL** Describe an instance of a journal in a CICS system. **CRESJRNM** Describe an instance of a journal name in a CICS system. **JRNLMODL** Journal models **JRNLNAME** System logs and general logs **STREAMNM** MVS log streams
OPERATE.PARTNER	**CRESPART** Describe an instance of a partner table in a CICS system. **PARTNER** CICS partners
OPERATE.PROCTYPE	**CRESPRTY** Describe an instance of a process type in a CICS system. **PROCTYP** Process types
OPERATE.PROFILE	**CRESPROF** Describe an instance of a profile in a CICS system. **PROFILE** CICS profiles

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.PROGRAM	**CRESPRGM** Describe an instance of a program in a CICS system. **PROGRAM** Programs **LIBDSN** LIBRARY data set names **LIBRARY** LIBRARY data sets **RPLLIST** DFHRPL data sets

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
OPERATE.REGION	**CICSDSA** Dynamic storage areas **CICSPAGP** CICS page pools **CICSRGN** CICS systems **CICSSTOR** All CICS dynamic storage areas **CRESMQMN** Describe an instance of an IBM MQ monitor in a CICS system **CRESSDMP** Describe an instance of a system dump code in a CICS system **CRESTDMP** Describe an instance of a transaction dump code in a CICS system **DOMSPOOL** CICS storage domain subpools **DSPGBL** Global CICS dispatcher **DSPMODE** CICS dispatcher TCB mode **DSPPOOL** CICS Dispatcher TCB pool **ENQUEUE** CICS enqueues **FEATURE** Feature toggles **LOADACT** CICS Loader activity by dynamic storage area **LOADER** CICS loader activity **LSRPBUF** Buffer usage for LSR pools **LSRPOOL** LSR pools **MASHIST** MAS history **MONITOR** CICS monitoring and statistics **MQCON** IBM MQ connection **MQCONN** IBM MQ connection statistics **MQINI** IBM MQ initiation queue

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.REGION (continued)	**MQMON** IBM MQ monitor **MVSESTG** MVS storage element **MVSTCB** MVS TCB **MVSTCBGL** Global MVS TCB **MVSWLM** z/OS Workload Manager **RECOVERY** CICS recovery manager **REQID** Timed requests **SYSDUMP** System dump codes **SYSPARM** 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. **TRANCLAS** Transaction classes **TRANDUMP** Transaction dump codes
OPERATE.RQMODEL	**CRESRQMD** Describe an instance of a request in a CICS system.
OPERATE.TASK	**EXCI** **HTASK** Completed task **TASK** Active tasks **TASKESTG** Task storage element **TASKFILE** Task file usage **TASKRMI** RMI usage by individual task **TASKTSQ** TSQ usage by individual task **TSKSPOLS** All task subpools **TSKSPOOL** Task storage subpools

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
OPERATE.TCPIPS	**ATOMSERV** Atom services **CRESATOM** Describe an instance of an Atom service in a CICS system. **CRESTCPS** Describe an instance of a TCPIP service in a CICS system. **HOST** URI Host **IPFACIL** IPIC connection sessions **PIPELINE** Pipelines **TCPIPGBL** TCP/IP global statistics **TCPIPS** TCP/IP services **URIMAP** URI maps **URIMPGBL** URI map global statistics **WEBSERV** Web services
OPERATE.TDQUEUE	**CRESTDQ** Describe an instance of a transient data queue in a CICS system. **EXTRATDQ** Extrapartition transient data queues **INDTDQ** Indirect transient data queues **INTRATDQ** Intrapartition transient data queues **REMTDQ** Remote transient data queues **TDQGBL** Intrapartition transient data queue usage
OPERATE.TERMINAL	**CRESTERM** Describe an instance of a terminal in a CICS system. **TERMNL** Terminals

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
OPERATE.TRAN	**CRESTRAN** Describe an instance of a transaction in a CICS system. **LOCTRAN** Local transactions **REMTRAN** Remote transactions **TASKASSC** Task association data
OPERATE.TSQUEUE	**CRESTSMD** Describe an instance of a temporary storage queue in a CICS system. **TSQUEUE** Temporary storage queues **TSMODEL** Temporary storage models **TSPOOL** Temporary storage pools **TSQGBL** Global temporary storage queues **TSQSHR** Shared temporary storage queues **TSQNAME** Long temporary storage queues
OPERATE.UOW	**UOWDSNF** Display shunted units of work **UOWENQ** Display enqueues for executing units of work **UOWLINK** Display links for unit of work **UOW** Display executing units of work

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)

Function.type combination	Resource Table Name and Usage
TOPOLOGY.DEF	**CSYSGRP** Create, display, and maintain CICS system groups. **CSYSDEF** Create, display, and maintain CICS systems. **CSGLCGCG** Describe the link of a CICS system group to an outer system group. **CSGLCGCS** Describe the link of a CICS system to a system group. **MAS** Display CICS systems in a CICSplex. **MASSTAT** Display CICS systems in a CICSplex by CMAS. **PERIODEF** Display period definitions. **SYSLINK** Display information about the links that exist between CICS systems.

Table 30. Function and type combinations for resources accessible using CICSPlex SM (continued)	
Function.type combination	**Resource Table Name and Usage**
WORKLOAD.DEF	**DTRINGRP** Display transactions in transaction groups. **LNKSWSCG** Describe the link between a CICS system group and a workload specification. **LNKSWSCS** Describe the link between a CICS system and a workload specification. **TRANGRP** Create, display, and maintain transaction groups. **WLMATAFF** Display and discard active affinities. **WLMATGRP** Display and discard active transaction groups **WLMATRAN** Display and discard transaction directory. **WLMAWAOR** Display active AORs in a workload. **WLMAWDEF** Display and discard active workload definitions. **WLMAWORK** Display active workloads. **WLMAWTOR** Display active TORs in a workload. **WLMDEF** Create, display, and maintain workload definitions. **WLMGROUP** Create, display, and maintain workload groups. **WLMINGRP** Display workload definitions in groups. **WLMINSPC** Display workload groups in workload specifications. **WLMSPEC** Create, display, and maintain workload specifications.

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 31 on page 376 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 31. 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

Table 31. 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**
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 379 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 379 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 349.
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.

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 32 on page 380 explains the possible parameter combinations.

Table 32. 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 **does not** 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 MVS images that share the same RACF database.

3. If other CMASes run on other MVS 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 userid 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 userid timeout check.

Whenever a change is made to a userid (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.

4. To force a CMAS to discard security information from the cache before timeout processing has occurred, use one 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 userids 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 96 on page 383 and Figure 97 on page 384 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 382.
3. CICSPlex SM determines whether the user is exempt from simulated CICS security checking.
   - If so, processing continues at “9” on page 382.
4. CICSPlex SM determines whether simulated CICS command checking is to be performed.
   - If not, processing continues at “6” on page 382.
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 382.
7. CICSPlex SM determines whether simulated CICS resource checking is to be performed.
   - If not, processing continues at “9” on page 382.
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 382.
    - Otherwise, CICSPlex SM returns the requested information in the appropriate view.

No further security checking is required.

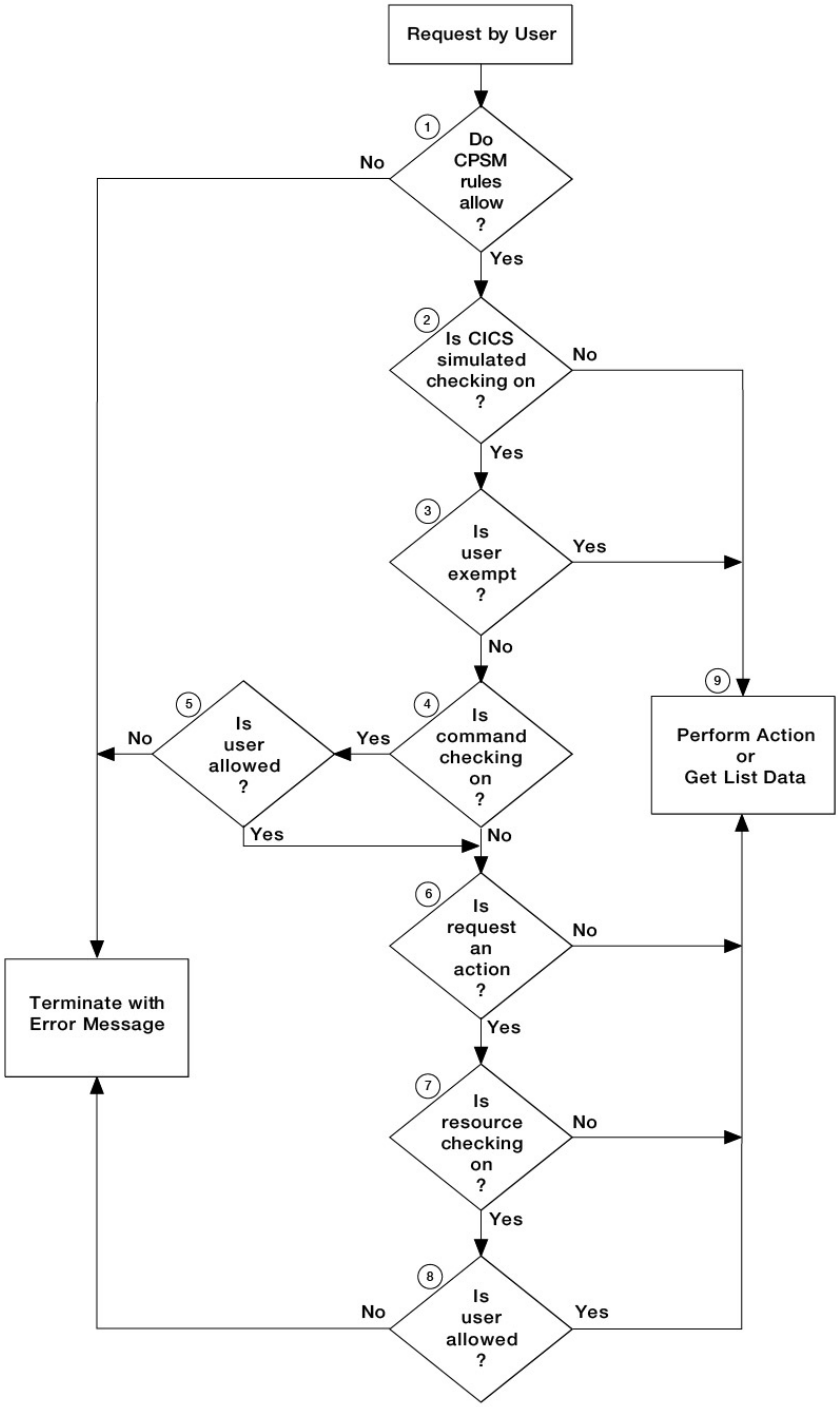


Figure 96. Flowchart of CICSPlex SM security checking sequence - part 1

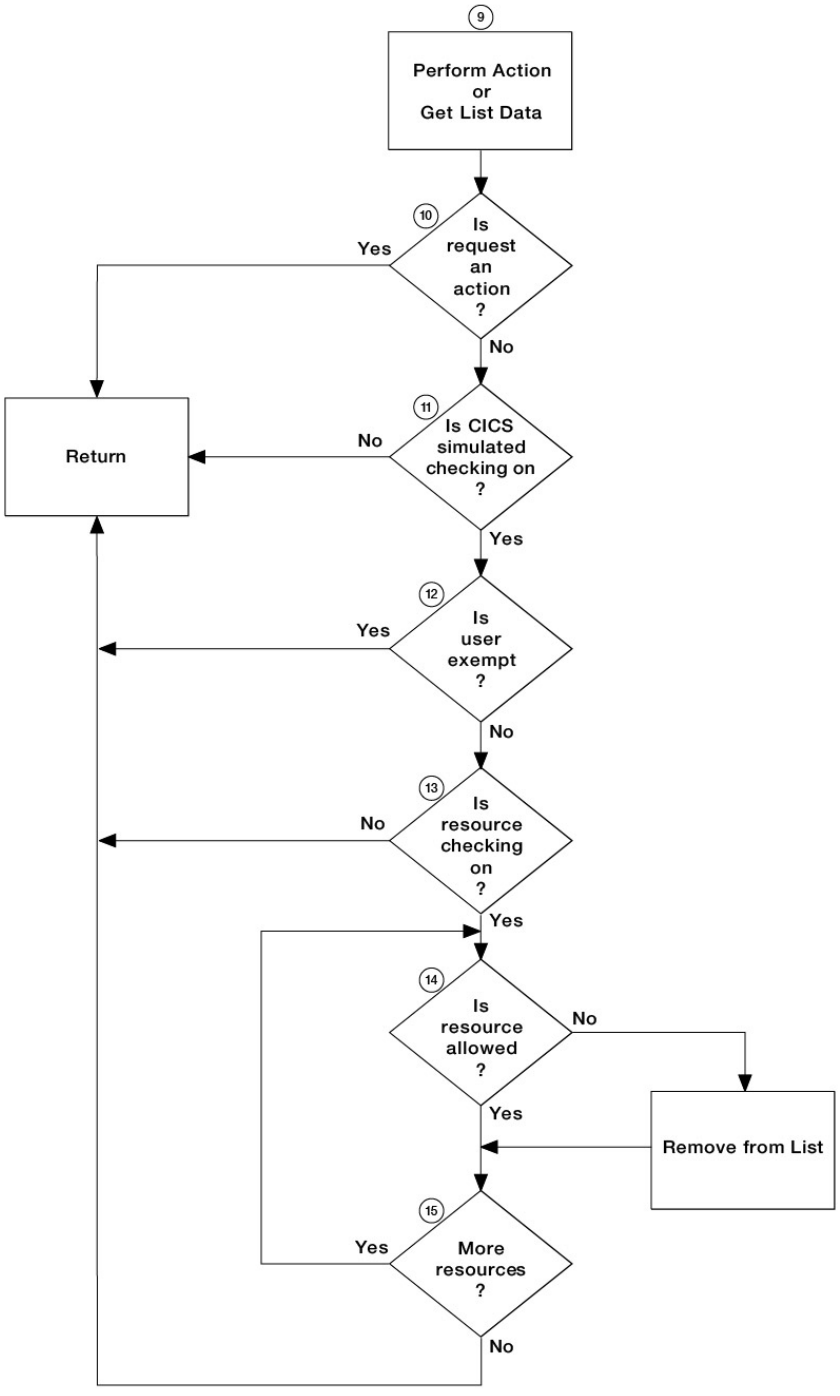


Figure 97. Flowchart of CICSPlex SM security checking sequence - part 2

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 336.

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 386.

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 tasks: security

The following information provides examples of typical security setup tasks that you can use as a model for your own.

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 have been used in these examples: we recommend that you create such groups.

The first example describes creating some RACF profiles to protect all CICSPlex SM functions and resources. The next step is to permit access selectively to particular users of particular resources.

Example: Protecting all CICSPlex SM resources

To create the RACF profile to protect all CICSPlex SM resources, do the following:

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 to protect 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. The next step is very similar to Step “2” on page 387: we 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, we have two CICSplexes, and so create two RACF profiles:

   ```
   RDEF CPSMOBJ *.*.PLXPROD1.* UACC(NONE) OWNER(admin_group) +
        NOTIFY(admin_user)
   RDEF CPSMOBJ *.*.PLXPROD2.* UACC(NONE) OWNER(admin_group) +
        NOTIFY(admin_user)
   ```

   Note that you cannot replace Step “2” on page 387 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 387.

4. In Step “3” on page 387 we protected all CICSPlex SM functions and resources at the CICSplex level. In this step, we're going to define profiles to control access to the CICSPlex SM CONFIG and TOPOLOGY definition functions, so that we 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, and 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 we've controlled access to CICSPlex SM functions and resources, we 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. In this example, we'll show you 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 can see that we 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

In this example, we show you how to give user PAYUSR1 read access to all transactions running on CICS systems on MVS system A.

In the example, we have three CICS systems (say, CICSAA01, CICSAA02, and CICSAA03) 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. CWBA is defined as 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.

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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE resource definitions for CICS web support tells you how to set up a TCPIPSERVICE 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 profiles for CICS classes.

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 397.

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 399 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 108 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 TCPIPSERVICE definition:

Table 33. How the user of an HTTP client is identified

AUTHENTICATE	**SSL**	**How the user is identified**
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	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. 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.
BASIC	all values	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. 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. 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.

Table 33. How the user of an HTTP client is identified (continued)		
AUTHENTICATE	**SSL**	**How the user is identified**
CERTIFICATE	CLIENTAUTH, or ATTLSAWARE	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. 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.
AUTOREGISTER	CLIENTAUTH, or ATTLSAWARE	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. 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. If the client does not send a certificate, then the connection is rejected.
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	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. 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. If the client does not send a certificate, then the user ID is obtained from the client, using HTTP basic authentication.

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 TCPIPSERVICE 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 TCPIPSERVICE definition:

Authentication scheme	AUTHENTICATE	SSL	Notes
HTTP with no authentication	NO	NO, YES, CLIENTAUTH, or ATTLSAWARE	

Authentication scheme	AUTHENTICATE	SSL	Notes
HTTP with basic authentication	BASIC	NO, YES, CLIENTAUTH, or ATTLSAWARE	
HTTP with basic authentication	AUTOMATIC	NO, YES, CLIENTAUTH, or ATTLSAWARE	If SSL(CLIENTAUTH | ATTLSAWARE) 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 ATTLSAWARE	If the client does not send a certificate, the connection is not established.
HTTP with SSL client certificate authentication	AUTOMATIC	CLIENTAUTH or ATTLSAWARE	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 TCPIPSERVICE 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 `&dfhwbpw_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.

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 TCPIPSERVICE definitions for inbound ports to specify these requirements. Refer to "CICS as an HTTP server: authentication and identification" on page 392.

All the user IDs used by web clients must have a user profile in RACF or your equivalent external security manager. Refer to RACF profiles for CICS classes.

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 TCPIPSERVICE resource definition defines each port used for CICS web support. The TCPIPSERVICE 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 TCPIPSERVICE resource definitions for CICS web support explains how to create definitions for ports.

To help keep ports secure:

- Specify the MAXDATALEN attribute on every TCPIPSERVICE 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 403.

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 in Administering.

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 TCPIPSERVICE 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-supplied 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. The default is NO (no resource security), and 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. The default is NO (no command security), and 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 in Developing system 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 TCPIPSERVICE 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 TCPIPSERVICE definitions with an alternative transaction that specifies the program DFHWBXN 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.

   RESSEC(YES) is specified in CWXN as supplied by CICS, but, for TRANSACTION resource definitions in general, the default is RESSEC(NO).

   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 TCPIPSERVICE 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 TCPIPSERVICE 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 TCPIPSERVICE 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 34 on page 405. 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.

Table 34. Detailed description of AT-TLS modes and their CICS support			
Mode type	**AT-TLS calls issued**	**`ApplicationControlled` setting in AT-TLS policy**	**Supported by CICS TS for z/OS**
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 TCPIPSERVICE 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 TCPIPSERVICE, 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 TCPIPSERVICE definition. SSL(ATTLSAWARE) is only allowed if the TCPIPSERVICE also specifies PROTOCOL(HTTP).

When a TCPIPSERVICE 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 TCPIPSERVICE is defined with SSL(ATTLSAWARE), CICS can access client certificates and their associated RACF USERIDs. The result is all the certificate-related TCPIPSERVICE AUTHENTICATE options (CERTIFICATE | AUTOREGISTER | AUTOMATIC) are supported by SSL(ATTLSAWARE).

When a TCPIPSERVICE 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 **SOCIPHER**.

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 35. 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.
`HandShakeRole=ServerWithClientAuth` with any of `ClientAuthType`	SSL(NO)	Yes	The connection is successfully established. No client certificate is available.

Table 35. Combinations for AT-TLS and CICS TCPIPService (continued)			
AT-TLS policy for the port	**CICS TCPIPSERVICE**	**Valid combination**	**Expected result in CICS**
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, DFHSO messages. Also, message DFHWB0365 is detailed here, DFHWB messages.

CICS provides some diagnostic messages when you encounter errors that are discovered by CICS:

DFHWB0365

date time applid tranid A client connects to a TCPIPSERVICE defined with SSL(ATTLSAWARE) but the connection is not secured by AT-TLS. Host IP address: *hostaddr*. Client IP address: *clientaddr*. TCPIPSERVICE: *tcpipservice*.

DFHSO0147 W

applid A non-secure client connection is received for ATTLSAWARE TCPIPSERVICE *tcpipservice*. Client IP address: *clientaddr*. TTLS_IOCTL *value X'ttlsioctl'*.

DFHSO0149 W

applid A client connection that uses **CLIENTAUTHTYPE(PASSTHRU)** is detected for ATTLSAWARE TCPIPSERVICE *tcpipservice*. TTLS_IOCTL *value X'ttlsioctl'*. The TCPIPSERVICE 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: CICSD
   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.
2. The following diagnostics appear when a client connection is made to a TCPIPSERVICE defined with **SSL(ATTLSAWARE)** where the TCPIPSERVICE 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:

- DFHSO0147 W IY2CZCCM 041 A non-secure client connection is received for ATTLSAWARE TCPIPSERVICE ATTLS2. Client IP address: 9.174.17.124. TTLS_IOCTL value X'0100000102010000'
- DFHWB0365 06/23/2015 10:14:22 IY2CZCCM CWXN A client connects to a TCPIPSERVICE defined with SSL(ATTLSAWARE) but the connection is not secured by AT-TLS. Host IP address: 9.20.5.0. Client IP address: 9.174.17.124. TCPIPSERVICE: ATTLS2.

The first message is only issued once for any TCPIPSERVICE. 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 TCPIPSERVICE resource attributes.

The following two tables show the CICS SIT parameters and TCPIPSERVICE resource attributes that are used for a TLS handshake and their AT-TLS level equivalents.

Table 36. 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 37. TCPIPSERVICE resource attributes and AT-TLS equivalents

TCPIPSERVICE 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
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 410
- "Example 2: AT-TLS policy rules for TLS/SSL client authentication" on page 412

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 TCPIPSERVICE resource (SSL(YES)). This configuration does not support client certificates. Figure 98 on page 410 and Figure 99 on page 410 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 98. CICS startup parameters

```
TCpipservice   : HTTPSSL
GROup          : JULESWEB
DEScription  ==> CICS WEB TCPIPSERVICE WITH SSL SUPPORT
POrtnumber   ==> 25008
STatus       ==> Open
PROtocol     ==> Http
SSl          ==> Yes
CErtificate  ==> CICS-2048-certificate
CIphers      ==> 35363738392F303132330A1613100D15120F0C
AUthenticate ==> Basic
```

Figure 99. SSL-related TCPIPSERVICE 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 TCPIPSERVICE 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).

Figure 100 on page 411 shows the AT-TLS configuration that replicates the CICS environment for the TCPIPSERVICE named HTTPSSL.

```
TTLSRule SIMPLECICS
{
LocalPortRange 25008
Direction Inbound
Priority 256
TTLSGroupActionRef CICSGroupAct1
TTLSEnvironmentActionRef CICSEnvironmentAct1
}
TTLSGroupAction CICSGroupAct1
{
TTLSEnabled On
FIPS140 off
}
TTLSEnvironmentAction CICSEnvironmentAct1
{
HandShakeRole Server
TTLSKeyRingParmsRef CICSKeyRingParms1
TTLSCipherParmsRef CICSCipherParms1
TTLSEnvironmentAdvancedParmsRef CICSEnvAdvParms1
TTLSGskAdvancedParmsRef CICSGskAdvParms1
}
TTLSKeyRingParms CICSKeyRingParms1
{
Keyring CICSKeyRing
}
TTLSCipherParms CICSCipherParms1
{
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
TLSV1 On
TLSV1.1 On
TLSV1.2 On
CertificateLabel CICS-2048-certificate
}
TTLSGskAdvancedParms CICSGskAdvParms1
{
GSK_SYSPLEX_SIDCACHE off
GSK_V3_SESSION_TIMEOUT 600

}
```

Figure 100. AT-TLS configuration

Before you activate this AT-TLS policy, alter the CICS TCPIPSERVICE resource as follows:

```
TCpipservice   : HTTPSSL
GROup          : JULESWEB
DEScription  ==> CICS WEB TCPIPSERVICE WITH AT-TLS SSL SUPPORT
POrtnumber   ==> 25008
STatus       ==> Open
PROtocol     ==> Http
SSl          ==> NO|ATTLSAWARE
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 ATTLSAWARE, 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 ATTLSAWARE, 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 TCPIPSERVICE definition does not specify an AUTHENTICATE option that requires client certificates. The previous example TCPIPSERVICE resource specifies AUTHENTICATE(BASIC), which does not require a client certificate.

When the AT-TLS policy is active and the TCPIPSERVICE 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:

```
TTLSGroupAction CICSGroupAct1
{
TTLSEnabled 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 TCPIPSERVICE resource (SSL(CLIENTAUTH)).

This configuration supports client certificates. Figure 101 on page 412 and Figure 102 on page 412 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 101. CICS startup parameters

```
TCpipservice   : CLAUTH
GROup          : JULESWEB
DEScription  ==> CICS Web TCPIPSERVICE with SSL CLIENTAUTH support
POrtnumber   ==> 25009
STatus       ==> Open
PROtocol     ==> Http
SSl          ==> Clientauth
CErtificate  ==> CICS-2048-certificate
CIphers      ==> 35363738392F303132330A1613100D15120F0C
AUthenticate ==> Certificate
```

Figure 102. SSL-related TCPIPSERVICE 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 TCPIPSERVICE resource definition to use 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).

Figure 103 on page 413 shows the AT-TLS client authentication configuration that replicates the CICS environment for the TCPIPSERVICE named CLAUTH.

```
TTLSRule CLIENTAUTHCICS
{
LocalPortRange 25009
Direction Inbound
Priority 256
TTLSGroupActionRef CICSGroupAct2
TTLSEnvironmentActionRef CICSEnvironmentAct2
}
TTLSGroupAction CICSGroupAct2
{
TTLSEnabled On
FIPS140 off
}
TTLSEnvironmentAction CICSEnvironmentAct2
{
HandShakeRole ServerWithClientAuth
TTLSKeyRingParmsRef CICSKeyRingParms2
TTLSCipherParmsRef CICSCipherParms2
TTLSEnvironmentAdvancedParmsRef CICSEnvAdvParms2
TTLSGskAdvancedParmsRef CICSGskAdvParms2
}
TTLSKeyRingParms CICSKeyRingParms2
{
Keyring CICSKeyRing
}
TTLSCipherParms CICSCipherParms2
{
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 CICSEnvAdvParms2
{
SSLv3 Off
TLSV1 On
TLSV1.1 On
TLSV1.2 On
CertificateLabel CICS-2048-certificate
ClientAuthType Full
}
TTLSGskAdvancedParms CICSGskAdvParms2
{
GSK_SYSPLEX_SIDCACHE off
GSK_V3_SESSION_TIMEOUT 600
}
```

Figure 103. AT-TLS client authentication configuration

Before you activate this example AT-TLS policy, alter the CICS TCPIPSERVICE resource definition as follows:

```
TCpipservice   : CLAUTH
GROup          : JULESWEB
DEScription  ==> CICS Web TCPIPSERVICE with SSL CLIENTAUTH support
```

```
POrtnumber   ==> 25009
STatus       ==> Open
PROtocol     ==> Http
SSl          ==> ATTLSAWARE
CErtificate  ==>
CIphers      ==>
AUthenticate ==> Certificate
```

In this example, SSL must be set to ATTLSAWARE 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(ATTLSAWARE)**, 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 TCPIPSERVICE port is configured by using `ClientAuthType PassThru` and the TCPIPSERVICE resource is defined with SSL(ATTLSAWARE), when the first client connection arrives, CICS detects the unsupported configuration. CICS then closes the TCPIPSERVICE and issues message DFHSO0149.

When the AT-TLS policy is active and the TCPIPSERVICE 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:

```
TTLSGroupAction CICSGroupAct2
{
TTLSEnabled On
FIPS140 on
}

TTLSEnvironmentAdvancedParms CICSEnvAdvParms2
{
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 TCPIPSERVICE 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 38 on page 419 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 38. 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 CICS TS security. 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.

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 userid 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.

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 userid 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 userid 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 userid and of a defined activity userid:

```
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 userid 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 userid or the userid 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.

Attach-time security for processes and activities

You can use Attach-time security to check if 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 userid - that is, the userid 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**.

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(NO) and CMDSEC(NO). For more details, see Resource security and Command security.

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.

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 39 on page 427 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 39. CICS-MQ bridge monitor security

Bridge monitor start method	At a signed on terminal	User ID for bridge monitor
From a terminal or **EXEC CICS LINK** in a program	Yes	Signed-on user ID
From a terminal or **EXEC CICS LINK** in a program	No	CICS default user ID

Table 39. CICS-MQ bridge monitor security (continued)		
Bridge monitor start method	**At a signed on terminal**	**User ID for bridge monitor**
EXEC CICS START with user ID	–	User ID from START
EXEC CICS START without user ID	–	User ID from START
The CICS-MQ trigger monitor CKTI	–	User ID from START
The CICS MQ monitor (MQMONITOR)	–	• The **MONUSERID** attribute of the MQMONITOR resource, if security checking is active for the CICS region (that is, the **SEC** system initialization parameter is set to YES) • User ID that started the MQMONITOR resource, if security checking is disabled for the CICS region (that is, **SEC** is set to NO)

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 in IBM MQ product documentation.

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 39 on page 427, 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

Security for Db2

In the CICS Db2 environment, there are four main stages at which you can implement security checking.

The four stages are:

- 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 also use RACF, or an equivalent external security manager, to protect the components that make up CICS and Db2 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 104 on page 432 shows the security mechanisms involved in a CICS Db2 environment.

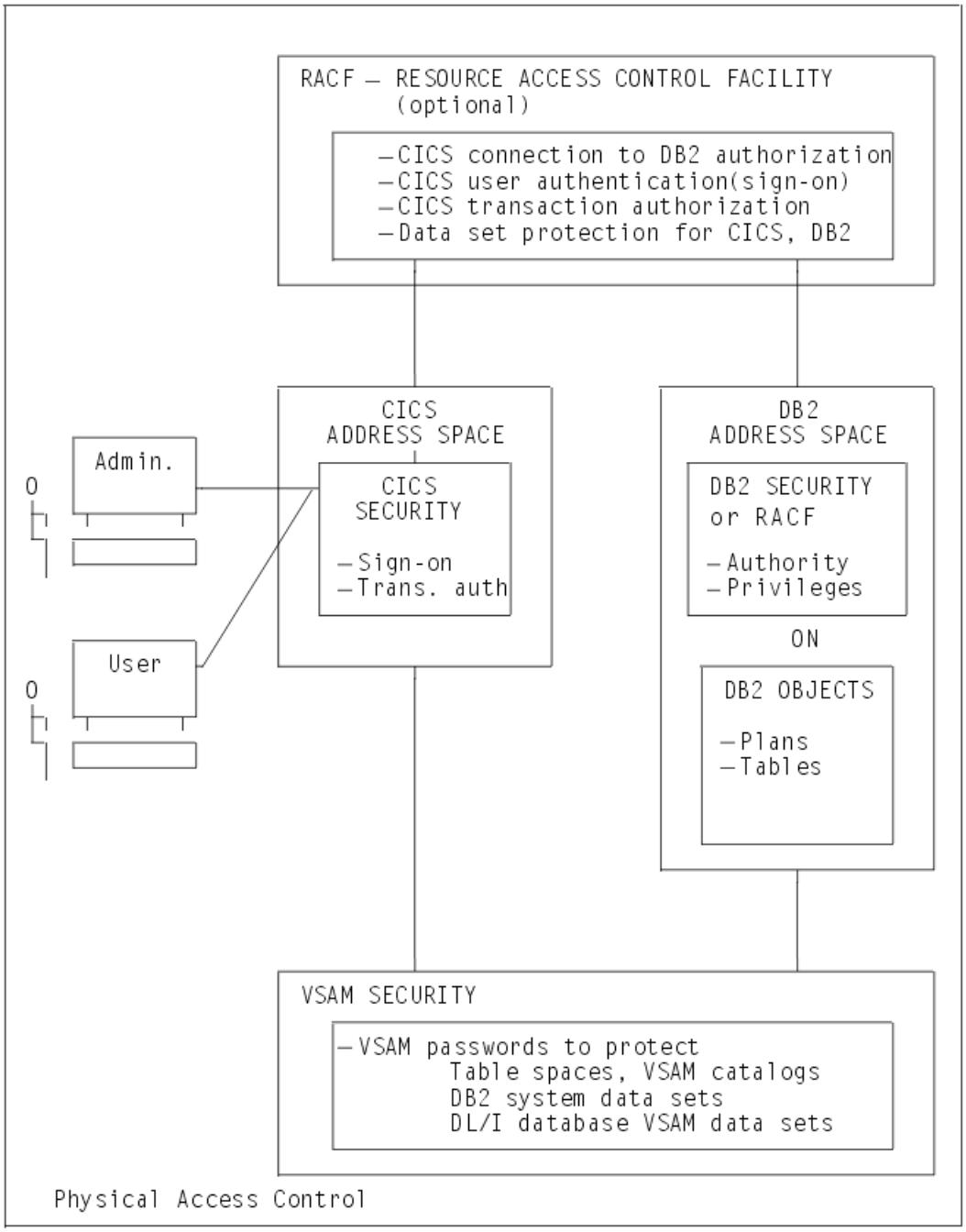


Figure 104. Overview of the CICS Db2 security mechanisms

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 DSNC 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 CICS TS security.

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 433 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 435 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 436 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.

To protect your Db2-related resources using resource security, complete these steps.

Procedure

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, RRCDTE, supplied in member DFH$RACF of CICSTS61.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 439 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 DSNC 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 profiles for CICS classes).

If you have defined transactions other than DSNC 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 profiles for CICS classes 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 DSNC 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 439 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 446 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 DSNC 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 440 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 441 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 group profiles.
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 443 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 445 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 DSNC 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 436 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 40 on page 444 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 in Monitoring.

Table 40 on page 444 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 40. 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 documentationfor 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 group profiles 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 436 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 442. 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 439.

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 DSNC, 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 438 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 439 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 105 on page 447 illustrates the process.

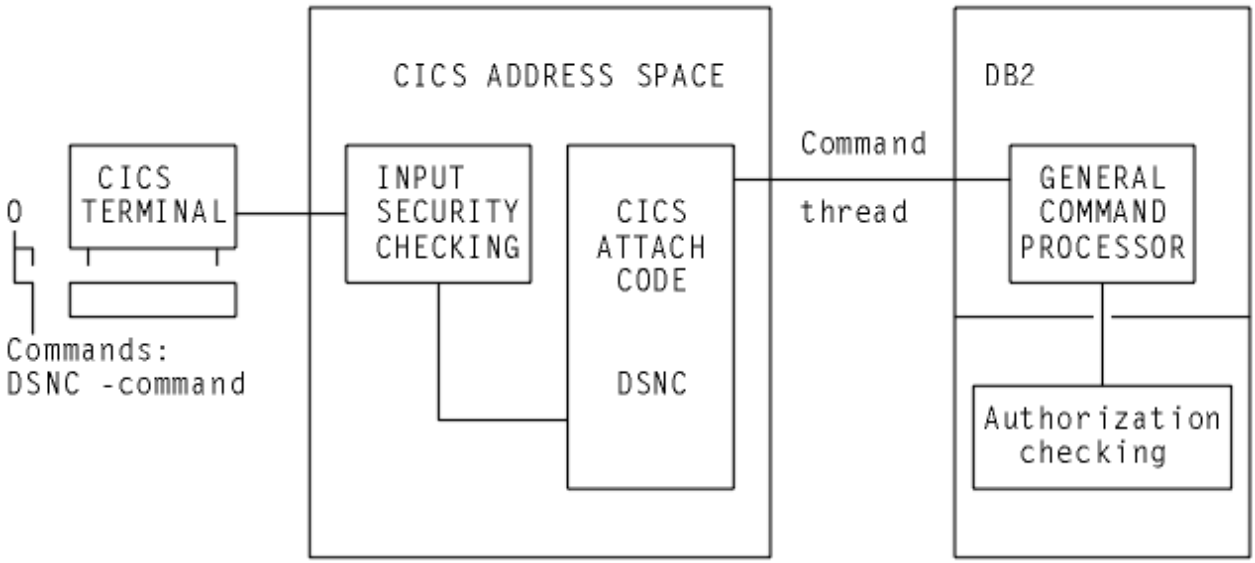


Figure 105. Security mechanisms for Db2 commands

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 106 on page 448 illustrates this process.

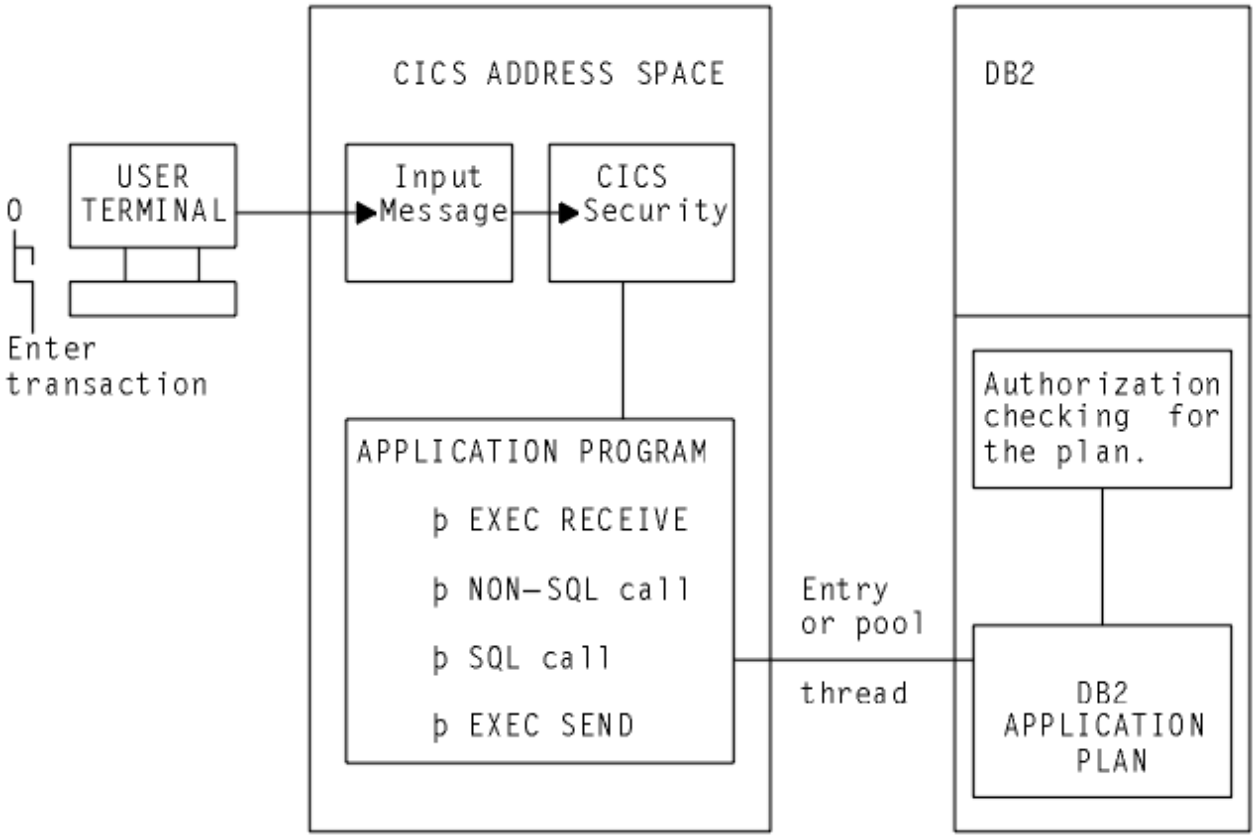


Figure 106. Security mechanisms for executing a plan

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 439 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, several security facilities are available.

You can use one or more of the following optional security facilities:

- "PSB authorization checking by CICS" on page 450
- 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 41. 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 **not** support extended addressing.
Intrapartition Transient Data (DFHINTRA)	Yes, could contain sensitive data.	DFHINTRA supports extended format, but does **not** support extended addressing.
Extrapartition Transient Data	Yes, could contain sensitive data.	Encryption is not supported for partitioned data sets (PDS).

Table 41. 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 CONFDATA 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 CONFDATA 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 **not** 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.

  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.

  See “Defining DFHAPPL FACILITY class profiles for an EXCI region” on page 453 for information about FACILITY class profiles for an EXCI client program.

- 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.

Defining DFHAPPL FACILITY class profiles for an EXCI region

Define the *user_name* part of the DFHAPPL profile name as follows:

- 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.

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 Bind security 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 ATTACHCSEC(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 userid, 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 CICS TS security.

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**, which is used to transmit the client's UNIX user ID, group ID, and other identification information.
- **Data Encryption Standard (DES) authentication**, which is not available at ONC RPC Version 3.9, and so cannot be used with CICS ONC RPC.
- **Null authentication**, which offers no security checking.

This section describes how CICS ONC RPC interacts with the security facilities of ONC RPC and CICS.

Security in ONC RPC

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**, which is used to transmit the client’s UNIX user ID, group ID, and other identification information.
- **Data Encryption Standard (DES) authentication**, which is not available at ONC RPC Version 3.9, and so cannot be used with CICS ONC RPC.
- **Null authentication**, which offers no security checking.

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 107 on page 458 shows how CICS security interacts with the operation of CICS ONC RPC.

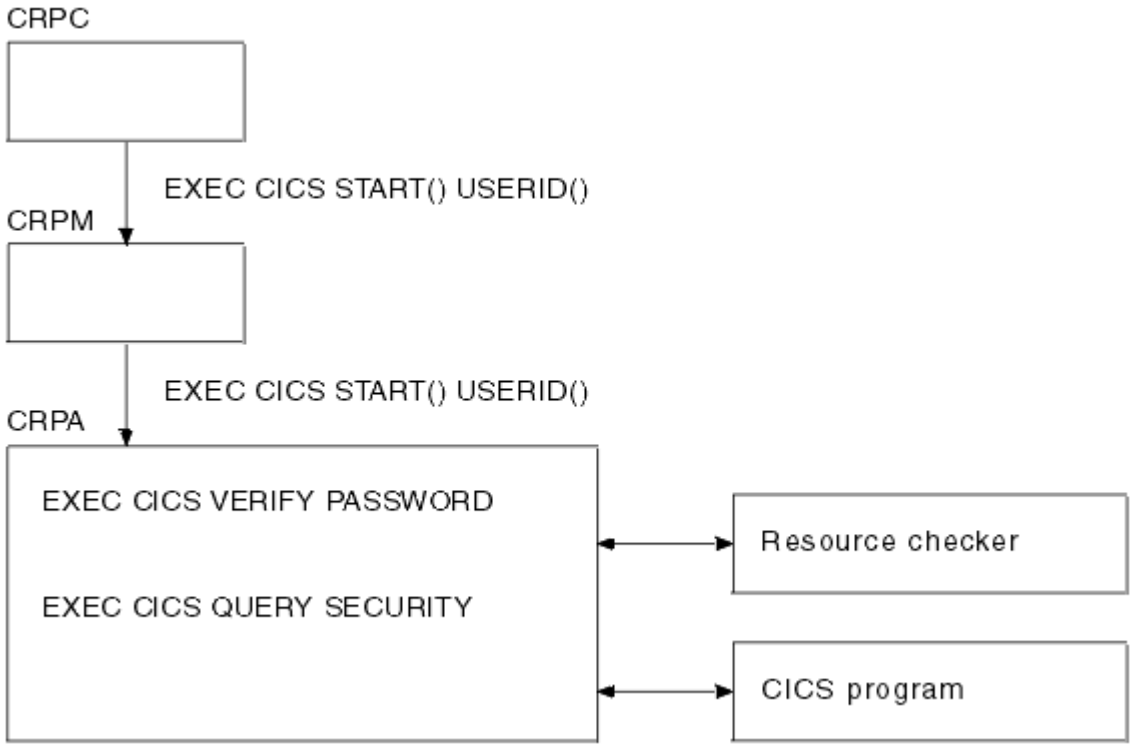


Figure 107. How CICS security interacts with CICS ONC RPC operations

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.

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.

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.

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 DFHRPUCH, or in the COBOL copybook DFHRPUCO.

Input	Output
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**	**res_check_reason** **res_check_response**

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 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

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 463 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.

The CICS shared data tables (SDT) 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 exits.

Security checking for data tables

You should 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 464). 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 464).

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

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
2. Authorizing the server
3. Authorizing a CICS region's access to a coupling facility data table pool
4. Authorizing a CICS region to a CFDT
5. File resource security checking.

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 are resolved in the same way as the LOGON security check for CICS shared data table support (see "SDT server authorization security check" on page 463 for details).

An optional security check, which is controlled by server startup parameters, is provided for controlling access to specific tables within a coupling facility data table pool. This is described further in "Authorizing a CICS region to a coupling facility data table " on page 467.

Authorizing server access to a list structure

Each coupling facility data table 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 userid 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 tablepool. 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 coupling facility data table

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 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.CICSTS61.CICS.SDFHLINK, which should be made an APF-authorized library in your MVS 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-MVS environment

If you are operating a multi-MVS 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 MVS program properties table

If you have an entry for the DFHSIP program in your MVS 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 MVS 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 CICSTS61.CICS.SDFHAUTH. This library must be defined to the operating system as APF-authorized.

To prevent unauthorized or accidental modification of CICSTS61.CICS.SDFHAUTH, make this library RACF-protected. Without such protection, the integrity and security of your MVS 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 MVS 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 startup 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 SYS1.CICSTS61.CICS.SDFHLINK and to SYS1.CICSTS61.CICS.SDFHLPA; and the other libraries (including CICSTS61.CICS.SDFHLOAD) that make up the STEPLIB and DFHRPL library concatenations.

See “Authorizing access to CICS data sets” on page 476 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 procedures to run 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.

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

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.

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.

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 CICSR as a user ID for a CICS region, enter the following RACF command from TSO:

```
ADDUSER CICSR  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 485 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 475.
- 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 476.
- 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 479.
- 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 480.
- 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 482.
- 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 484.
- 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 484.

  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 485.
- 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 485.

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 use 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)
        NOOIDCARD
        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 83 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 483.
- 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 470.

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 42 on page 476.

Table 42. Summary of generic data set profiles

Required access level	Type of CICS data sets protected
READ	Load libraries

Table 42. Summary of generic data set profiles (continued)	
Required access level	**Type of CICS data sets protected**
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 108 on page 477 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 108 on page 477. 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.

```
//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  'CICSTS61.CICS.SDFHLOAD'  NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.SDFHLOAD' ID(cics_id1,...,cics_group1,..,cics_groupn)
                         ACCESS(READ)
ADDSD  'CICSTS61.CICS.SDFHAUTH'  NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.SDFHAUTH' ID(cics_id1,...,cics_group1,..,cics_groupn)
                         ACCESS(READ)
ADDSD  'CICSTS61.CICS.applid.**'  NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.applid.**'  ID(applid_userid) ACCESS(UPDATE)
ADDSD  'CICSTS61.CICS.applid.DFHXR*'  NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.applid.DFHXR*'  ID(applid_userid) ACCESS(CONTROL)
ADDSD  'CICSTS61.CICS.applid.DFHINTRA' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.applid.DFHINTRA'  ID(applid_userid) ACCESS(CONTROL)
ADDSD  'CICSTS61.CICS.applid.DFHTEMP' NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.applid.DFHTEMP'  ID(applid_userid) ACCESS(CONTROL)
ADDSD  'CICSTS61.CICS.DFHCSD'          NOTIFY(cics_sys_admin_id) UACC(NONE)
PERMIT 'CICSTS61.CICS.DFHCSD' ID(cics_group1,..,cics_groupn) ACCESS(UPDATE)
/*
//
```

Figure 108. Example of a job to authorize access to CICS data sets

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 called 'CICSTS61.CICS.**'. However, you must strictly control who has read access to CICSTS61.CICS.SDFHAUTH, because it contains APF-authorized programs, and the profile protecting this data set **must** be defined with UACC(NONE). In Figure 108 on page 477 all of the partitioned data sets are defined with UACC(NONE) and have an explicit access list.

Although CICS modules exist in libraries SYS1.CICSTS61.CICS.SDFHLPA and SYS1.CICSTS61.CICS.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. In the examples shown in Figure

108 on page 477, all the names have a high-level qualifier of CICSTS61.CICS, but your installation will have its own naming conventions for you to follow.

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:

DFHGCD
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 MVS 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

DFHXRMSG
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)
```

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 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 'CICSTS61.CICS.appl1.dataset1'  ID(user or group)  ACCESS(READ)
PERMIT 'CICSTS61.CICS.appl1.dataset2'  ID(user or group)  ACCESS(READ)
PERMIT 'CICSTS61.CICS.appl2.dataset3'  ID(user or group)  ACCESS(UPDATE)
PERMIT 'CICSTS61.CICS.appl2.dataset4'  ID(user or group)  ACCESS(UPDATE)
```

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 Back-up-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 EOV 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 480 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 CICSDAA1, 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(CICSDAA1) 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 481)
- CICS regions to the named counter servers (see “Access to named counter servers” on page 481).

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_DFHNC001, the following RACF commands define the profile and provide the required access:

  ```
  RDEFINE FACILITY IXLSTR.DFHNCLS_DFHNC001 UACC(NONE)
  PERMIT IXLSTR.DFHNCLS_DFHNC001 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 482 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 CICSDAA1, 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(CICSDAA1) 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 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 named counter pool.

Access to the named counter pool, either by a CICS region or by the named counter 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 named counter server.

Access is permitted to any named counter server without a specific DFHCF.*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, specifying:

```
RDEFINE FACILITY (DFHNC.*) UACC(NONE)
```

ensures that access is allowed only to named counter servers with a more specific profile to which a named counter server or CICS region is authorized.

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 CICSDAA1, and the CICS region userid (shared by a number of AORs) is CICSDA##, define and authorize the profile as follows:

```
RDEFINE SUBSYSNM  CICSDAA1  UACC(NONE) NOTIFY(userid)
PERMIT  CICSDAA1  CLASS(SUBSYSNM)  ID(CICSDA##)  ACCESS(READ)
```

You can use wildcard characters on the applid to specify more than one CICS region, for example:

```
PERMIT  CICSD%%%  CLASS(SUBSYSNM)  ID(CICSDGRP) 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 userid of the CICS region in their security tokens, not the userid of the person who issued the **SPOOLOPEN** command. Thus, the userid qualifier in the related JESSPOOL profiles is the CICS region's userid.

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.

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 483 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 490.

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 43. 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 CSCS 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 494). 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 493.

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 492 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 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 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
For defining a profile of an **individual** terminal.

GTERMINL
For defining a 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 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.

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 SALARY file 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 FCICSFCT SALARY UACC(NONE)
PERMIT SALARY CLASS(FCICSFCT) ID(PAYROLL) ACCESS(READ)
PERMIT SALARY CLASS(FCICSFCT) ID(PAYROLL)
       (WHEN(TERMINAL(PAY001)) ACCESS(UPDATE)
```

To allow members of the operations group OPS to be able to use the CEMT transaction only from the console names MVS1MAST, issue the following command:

```
RDEFINE TCICSTRN CEMT UACC(NONE)
PERMIT CEMT CLASS(TCICSTRN) 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_typeterm)
```

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(consolename) USERID(*EVERY)
            TYPETERM(cics_typeterm)
```

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(consolename) USERID(*FIRST)
            TYPETERM(cics_typeterm)
```

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 DFHSSI*xx* 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 9XYZ@,#,.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 MVS 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(CICSMSGS)
                       GROUP(DFHDCTG)

                DEFINE TDQUEUE(QID2)
                       TYPE(INDIRECT)
                       INDIRECTNAME(QID1)
                       GROUP(DFHDCTG)

 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 in CICS Transaction Server for z/OS, Version 6 Release 1 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  userjnls UACC(NONE)
                   ADDMEM(JRNL001, JRNL002, ....)
                   NOTIFY(sys_admin_userid)
PERMIT   userjnls  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 userid 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 494 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 493.

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 493.

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 applications

You control access to the initial program specified in the transaction resource definition by authorizing the user to initiate the transaction (transaction security). CICS requires a minimum authorization of READ for programs.

However, 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 XPPT=YES as a CICS system initialization parameter 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 CICSA function ships a DPL command to CICSB, where the program then executes, CICSB issues the security check. If the DPL request is function shipped again to CICSC for execution, it is CICSC that issues the security check.

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 X*name* 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.

   “RACF user profiles” on page 81 tells you how to 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 `setfacl` 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 `setfacl` 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 `setfacl` 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

Defining generic profiles for resources

If you control access to CICS transactions by means of transaction security, there is probably only a very small subset of other resource types for which you need a further level of RACF protection.

For example, there may 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.

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)

may 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?

f 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.

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 **WRITEQ TD** commands to an extrapartition TDQ defined to the internal reader
- Using **SPOOLWRITE** commands when a `USERID("INTRDR")` was specified on the **SPOOLOPEN** command

There are three user IDs involved in job submission:

- Region user ID.
- Signed-on 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 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. If this isn't the region user ID, CICS adds USER=*job_userid* to the JOB card. In releases earlier than CICS TS 5.5, the default is always the region user ID.

Resource security on the TDQ provides protection for JCL jobs that are submitted through the TDQ. Additional protection is provided by 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 511.

Protection for JCL jobs that are submitted by using spool commands is provided by surrogate user checking. For details, see "Security when using the SPOOLWRITE commands to submit JCL" on page 512.

In addition, you must have a profile for the region user ID in the JESSPOOL class to give the region user ID the authority to submit jobs for the job user IDs. See "z/OS surrogate user checking" on page 512 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 498.

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=JOBUSER
```

CICS performs 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 (JOBUSER in the above example).

To enable this security check, you must set the following options:

- CICS surrogate user checking is enabled by the system initialization parameter **XUSER=YES**.
- The feature toggle for surrogate user checking for spool commands is enabled:

```
com.ibm.cics.spool.surrogate.check=true
```

If the surrogate check fails, a NOTAUTH response is returned from the **WRITEQ TD** command. See "CICS surrogate user checking" on page 513 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=jobuserid
```

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=regionuserid
```

Security when using the SPOOLWRITE commands to submit JCL

To use spool commands, CICS must be started with the system initialization parameter **SPOOL=YES**.

If a JOB card that is written using a **SPOOLWRITE** command has a USER parameter, protection is provided by a surrogate check is made if the following options are defined:

- CICS surrogate user checking is enabled by the system initialization parameter **XUSER=YES**.
- The feature toggle for surrogate user checking for spool commands is enabled:

```
com.ibm.cics.spool.surrogate.check=true
```

If the surrogate check fails, a NOTAUTH response is returned from the **SPOOLWRITE** command. See "CICS surrogate user checking" on page 513 for details of the RACF definitions and error messages if the surrogate check fails.

If a JOB card that is written using a **SPOOLWRITE** command doesn't have a USER parameter, the job user ID defaults to the region user ID. This is subject to a surrogate check. You can change the default to job user ID to be the signed-on user ID if you set the following feature toggle:

```
com.ibm.cics.spool.defaultjobuser=TASK
```

Specifying the region user ID on the JOB card

If you want a job to be able to run under the 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 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 region user ID.

z/OS surrogate user checking

Although the **SPOOLWRITE** and **WRITEQ TD** commands are issued by the signed-on user ID, the job is submitted by the region user ID. Therefore, if the job user ID is not the region user ID and no password is

supplied, you must grant the 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 region user IDs to be able to submit jobs on behalf of any user, it is recommended that the 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 checking is made if the following options are defined:

- CICS surrogate user checking is enabled by the system initialization parameter **XUSER=YES**.
- 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 signed-on user ID and no password is supplied, it is necessary to grant the signed-on 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(signed_on_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, a DFHXS1111 message is 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(signed_on_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.

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

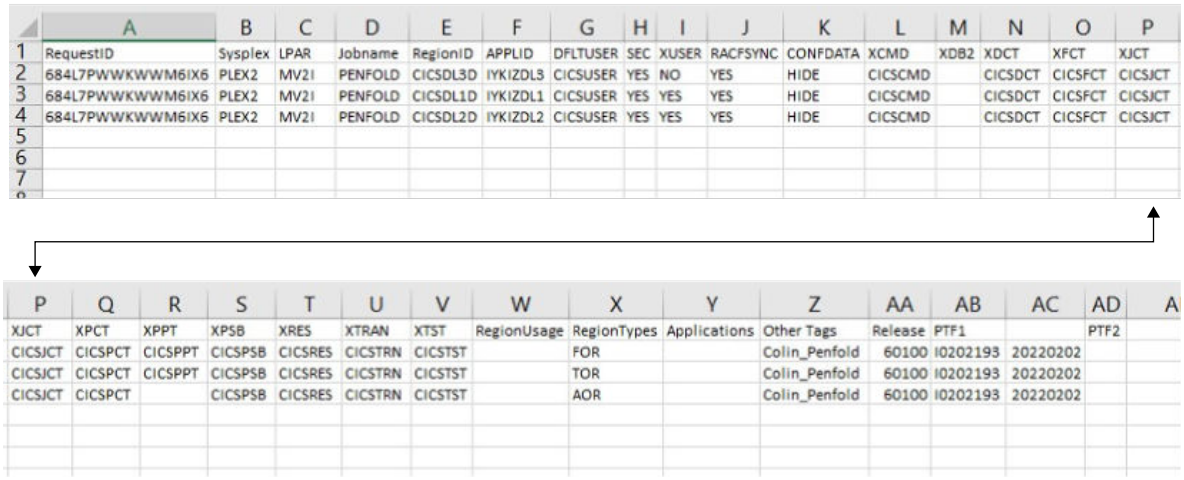
Example of formatted SMF 1154 type 80 records that uses an LPAR running three CICS regions when the compliance data was requested.

The SUMMARY output is shown in Figure 109 on page 516.

```
        1              2        3      4          5
Rec RequestID        Jobname  UserID   LPAR     yyyymmdd hh:mm:ss
  1 684L7PWWKWWM6IX6 CICSDL3D USERA    MV2I     20220203 11:15:05.20
  2 684L7PWWKWWM6IX6 CICSDL1D USERA    MV2I     20220203 11:15:05.20
  3 684L7PWWKWWM6IX6 CICSDL2D USERA    MV2I     20220203 11:15:05.20
```

Figure 109. 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.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	RequestID	Sysplex	LPAR	Jobname	RegionID	APPLID	DFLTUSER	SEC	XUSER	RACFSYNC	CONFDATA	XCMD	XDB2	XDCT	XFCT	XJCT
2	684L7PWWKWWM6IX6	PLEX2	MV2I	PENFOLD	CICSDL3D	IYKIZDL3	CICSUSER	YES	NO	YES	HIDE	CICSCMD		CICSDCT	CICSFCT	CICSJCT
3	684L7PWWKWWM6IX6	PLEX2	MV2I	PENFOLD	CICSDL1D	IYKIZDL1	CICSUSER	YES	YES	YES	HIDE	CICSCMD		CICSDCT	CICSFCT	CICSJCT
4	684L7PWWKWWM6IX6	PLEX2	MV2I	PENFOLD	CICSDL2D	IYKIZDL2	CICSUSER	YES	YES	YES	HIDE	CICSCMD		CICSDCT	CICSFCT	CICSJCT
5																
6																
7																

	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	A
1	XJCT	XPCT	XPPT	XPSB	XRES	XTRAN	XTST	RegionUsage	RegionTypes	Applications	Other Tags	Release	PTF1		PTF2	
2	CICSJCT	CICSPCT	CICSPPT	CICSPSB	CICSRES	CICSTRN	CICSTST		FOR		Colin_Penfold	60100	I0202193	20220202		
3	CICSJCT	CICSPCT	CICSPPT	CICSPSB	CICSRES	CICSTRN	CICSTST		TOR		Colin_Penfold	60100	I0202193	20220202		
4	CICSJCT	CICSPCT		CICSPSB	CICSRES	CICSTRN	CICSTST		AOR		Colin_Penfold	60100	I0202193	20220202		

Figure 110. 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 (see IBM Health Checker for z/OS User's Guide for details.) CICS supports IBM Health Checker for z/OS checks that define best practices for security-related CICS configuration. You can use these checks to audit that CICS configuration conforms to best practices.

What are health checks?

CICS provides health checks by using the IBM Health Checker for z/OS to help you validate important configuration settings in CICS.

The ability to review key settings is also of value when CICS is audited. For example, providing feedback to external auditors.

The health checks provide messages that indicate the severity of any issues identified. Message 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 reviews the message and actions required. Compliance and regulatory requirements are your responsibility.

Set 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 the 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.

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.

Running the health checks

CICS health checks run automatically if the IBM Health Checker for z/OS is enabled. Health checks are run in the local address space. For more information, see the IBM Health Checker for z/OS User's Guide.

The CICS health checker rules run every 30 minutes in the IBM Health Checker for z/OS address space. IBM Health Checker for z/OS reports on all CICS regions that ran within the previous 30 minutes. If any regions on an LPAR are identified as noncompliant with any of the best practices, IBM Health Checker for z/OS issues messages that warn of the noncompliant regions. For more information about how to review messages, see IBM Health Checker for z/OS User's Guide.

See IBM Health Checker for z/OS messages related to CICS for a list of the health check messages.

For more information about how IBM Health Checker for z/OS works in CICS, see IBM Health Checker for z/OS for CICS security.

For more information about the available health checks, see IBM Health Checker for z/OS for CICS security.

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 518 lists the fields and the possible values for each one. Table 44 on page 520 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, 0740 is the identifier for CICS TS for z/OS, Version 6.1

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 44. Summary of resource signature field values, by resource definition method

How the resource is defined	DEFINE SOURCE	DEFINE TIME	CHANGEAGENT	CHANGEAGREL	CHANGETIME	CHANGEUSRID	INSTALLAGENT	INSTALLTIME	INSTALLUSRID
GRPLIST INSTALL	CSD GROUP	Time stamp of CSD record creation	CSDAPI, CSDBATCH, or OVERRIDE	CICS release of CHANGEAGENT system	Time stamp of CSD record change	User ID that ran the CHANGEAGENT	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, CSDBATCH, or OVERRIDE	CICS release of CHANGEAGENT system	Time stamp of CSD record change	User ID that ran the CHANGEAGENT	CSDAPI	Time of the installation	User ID that ran the installation
EXEC CICS CREATE	Name of the program issuing the **CREATE** command	Time that the resource is created	CREATESPI or OVERRIDE	CICS release of CHANGEAGENT system	Time that the resource is created	User ID that ran the EXEC **CICS CREATE** command	CREATESPI	Time that the resource is created (from earlier run if warm-started)	User ID that ran the EXEC CICS **CREATE** command
Autoinstall	Autoinstall user program name	Time of the autoinstall	AUTOINSTALL or OVERRIDE	CICS release of CHANGEAGENT system	Time of the autoinstall	User ID that ran the autoinstall	AUTOINSTALL	Time of the autoinstall (from earlier run if warm-started)	User ID that ran the autoinstall
Table definition	Table name	Time of the last cold start	TABLE	CICS release of CHANGEAGENT system	Time of the last cold start	Jobstep user ID	TABLE	Time of the last cold start	Jobstep user ID

Table 44. Summary of resource signature field values, by resource definition method (continued)

How the resource is defined	DEFINE SOURCE	DEFINE TIME	CHANGEAGENT	CHANGEAGREL	CHANGETIME	CHANGEUSRID	INSTALLAGENT	INSTALLTIME	INSTALLUSRID
System defined	SYSTEM	Time of CICS startup	SYSTEM	CICS release of CHANGEAGENT system	Time of CICS startup	Jobstep user ID	SYSTEM	Time of CICS startup	Jobstep user ID
Dynamically created	Table 45 on page 522	Time that the resource is generated	DYNAMIC	CICS release of CHANGEAGENT system	Time that the resource is generated	User ID that generated the resource	DYNAMIC	Time that the installed resource is generated	User ID that generated the installed resource
Created by BUNDLE	BUNDLE name	Inherited from BUNDLE	Inherited from BUNDLE or OVERRIDE 2	Inherited from BUNDLE	Inherited from BUNDLE	Inherited from BUNDLE	BUNDLE	Inherited from BUNDLE	Inherited from BUNDLE
CICSPlex SM (EXEC CICS CREATE)	CPSMV*nn* where *nn* is the version of the CICSPlex SM BAS resource definition (the VER attribute)	CREATETIME from EYUDREP	DREPAPI, OVERRIDE or SYSTEM	CICS release of CHANGEAGENT system	CHANGETIME from EYUDREP	CHANGEUSRID from EYUDREP	CREATESPI	Time that the resource is created (from earlier run if warm-started)	User ID that ran the create or passed from CICSPlex SM
Created by CICSPlex SM for platform	Management part name	Time that the resource is created	CREATESPI	CICS release of CHANGEAGENT 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	DREPAPI	CICS release of CHANGEAGENT system	Time that the resource is installed	User ID that requested the SYSLINK installation 1	CREATESPI	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 45. 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 DFHWBBMS	User ID that ran DFHWBBMS or DFHWBTL	DFHWBBMS or DFHWBTL	User ID that ran DFHWBBMS or DFHWBTL
DUMPCODE	The installation of a DUMPCODE resource definition by a **SET SYSDUMPCODE ADD** command or a **SET TRANDUMPCODE ADD** command	User ID that issued the **SET SYSDUMPCODE ADD** or **SET TRANDUMPCODE ADD** command	Name of the program that issued the **SET SYSDUMPCODE ADD** or **SET TRANDUMPCODE ADD** command	User ID that issued the **SET SYSDUMPCODE ADD** or **SET TRANDUMPCODE ADD** 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
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_USS_CONFIGURATION

For more information about health checks in CICS, see IBM Health Checker for z/OS for CICS security.

How to configure region tags?

You specify region tags in a region tagging file named `cicstags.yaml`. 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 46 on page 523 describes these destinations, followed by examples of the messages.

Table 46. Finding messages about installed resources

Method of resource installation	Message	Destination
Install group list during installation	DFHSI1511I for each list that is installed	JESMSGLG
	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 524)	DFHRD*nnnn* 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

Table 46. Finding messages about installed resources (continued)		
Method of resource installation	**Message**	**Destination**
Manually install single resource using CEDA	DFHRD*nnnn* 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 **EXEC CICS CREATE** resource	CREATE resourcetype(resource) options	TDQ CSDL
	Message for each resource	MSGUSR
Application issues **EXEC CICS CREATE** 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 IYCWTC02 CRPTST3 CEDA INSTALL PROGRAM(BC14)
DFHRD0104 01/12/2020 10:44:20 IYK2ZDL1 IYCWTC02 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 IYCWTC02 CRPTST3 CEDA TRANSACTION definition entry for
BC14 has been added.
DFHPG0101 01/12/2020 10:44:20 IYK2ZDL1 IYCWTC02 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 47. User IDs in audit messages

EYUPARM in CMAS	SIT parm in CICS region where request is initiated	User ID in audit message
SEC(YES)	SEC=YES	Authenticated user ID **Note:** The authenticated user ID depends on how, and where, the request to issue the command was initiated, as illustrated in Table 48 on page 526.
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 48. 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.

Table 48. Authenticated user ID (continued)	
Where request is initiated	**Authenticated user ID**
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 527
- “Finding information about sign-on and sign-off through CMCI” on page 528
- “Finding information about sign-on and sign-off through CICSPlex SM WUI” on page 528
- “Finding information about sign-on and sign-off through Liberty” on page 528

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 DFHSN*nnnn* 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 IBM Health Checker for z/OS for CICS security.

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 CSCS Transient Data Queue (TDQ).

```
DFHXS1111 26/09/95 15:34:01 CICSSYS1 Security violation by user JONES
at netname D2D1 for resource FLA32 in class TCICSTRN.
SAF codes are (X'00000008',X'00000000'). ESM codesare
(X'00000008',X'00000000').
```

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 CSCS 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%A* (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%A* . (G) indicates that F%A* 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 `<safAuthorization/>` 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  :         867        Failed resource
authorizations  . . :          11
```

```
Successful command authorizations . :  567                     Failed command
authorizations . . . :           3
Successful surrogate authorizations :  51                      Failed surrogate
authorizations . . :           0
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 SAML; see How it works: 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.

Developing a SAML-aware initial program

You can write a program to extract a SAML token and link to DFHSAML to process the token.

About this task

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 533.

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

Configure web services to add a verified SAML token to an outbound request.

About this task

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 instructions, 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 534.

The SAML token is stored in the read-only container DFHSAML-OUTTOKEN.

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.

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 Release 1 (CICS TS 6.1) are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS TS security
- 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 TS 6.1, 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 TS 6.1 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 TS 6.1, 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® 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.

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.

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
  overview 385
  RACROUTE macros 386

A

access control
  HFS files 507
  z/OS UNIX files
  506
accounting 444
ACICSPCT general resource class 500, 503
activating security parameters 379
activities
  security of 421, 422
ADDMEM operand 435
ADDUSER command
  defining the userid for CICS to RACF 472
administration
  security 421
ALLOCATE_PIPE command
  security check failure 453
APPL general resource class
  controlling access to CICS region 483
application
  security 349, 374, 417
application program security
  defining resource classes 504
  MCICSPPT general resource class 504
  NCICSPPT general resource class 504
applying a security policy 274
AT-TLS
  Application Transparent Transport Layer Security 404,
  408, 409
  PROTOCOL(HTTP) 404, 408, 409
  SSL(ATTLSAWARE) 404, 408, 409
  SSL(NO) 404, 408, 409
AT-TLS Basic 404
AT-TLS Controlling 404
Atom collection
  security 414
Atom feed
  security 414
attach-time security 422
ATTLS 408, 409
ATTLS Aware 408, 409
ATTLS Basic 408, 409
ATTLS Controlling 408, 409
ATTLSAWARE 404
AUTHENTICATE(AUTOREGISTER) 409
AUTHENTICATE(CERTIFICATE) 409
authentication 392, 393, 397
authentication, RPC 457
authorization failures
  command security 471
authorization IDs, primary 443
authorization IDs, secondary 445
authorizing SYS1.PARMLIB libraries
  data sets 340
AUTHTYPE security 436
AUTHTYPE values for security
  authorization ID 439
  group ID 439
  sign ID from DB2CONN 439
  terminal ID 439
  transaction ID 439
  user ID 439

B

backup while open (BWO) 479
basic authentication 392, 393, 397
batch access to CSD, restricting 495
BCICSPCT general resource class 500, 503
bind-time security 453
building a keyring 104
BWO (backup while open) 479

C

cataloged procedures
  authorizing CICS as a started task 471
CCRL
  running from a START command 118
  running from a terminal 118
CDT (class descriptor table)
  IBM-supplied default classes 86
  setting up installation-defined classes 86
CEDA LOCK command 495
CEDA transaction 495
CEDF 509
CEDF transaction 509
CEDX 509
CEDX transaction 509
certificate revocation list 116
certificates, creating new 107
CFDT server authorization 466
CFRM policy 466
CICS as an HTTP client
  security 393
  SSL 403
CICS as an HTTP server
  security 392, 397, 399
CICS business transaction services
  administration
    security 421
  security
    attach-time 422
    command-level 422
    resource-level 421
CICS command security
  VCICSCMD general resource class 435

CICS commands and resources
creating security profiles with RACF
controlling access to resources 349
CICS JOB statement, PASSWORD parameter 472
CICS JOB statement, USER parameter 472
CICS Liberty 185, 188, 191, 195, 201
CICS load libraries, protecting 470
CICS region
access to 483
access to APPL class profiles 484
remote 484
userid as security token 485
CICS region userid
in started jobs 471
CICS resource security
XCICSDB2 general resource class 434
CICS security 432
CICS security, controlling 377
CICS segment 81
CICS SIT parameters
security-related 379
CICS source libraries, protecting 470
CICS system definition file (CSD), restricting batch access to 495
CICS system initialization parameters
SEC 457
XCMD 458
XPPT 457
XUSER 457
CICS web support
security 391
CICS-supplied RACF dynamic parse validation routines 469
CICSPlex SM
authorizing
libraries 340
procedures 342
protecting
with another ESM 385
with RACF 336
resource names 349
CICSPlex SM definitions, protecting
adding SAF resource classes 349
CICSPlex SM resource classes
controlling access to 349
CICSPlex SM security profiles
creating 340
cipher suites
identifying which are used 113
restricting with CIPHERS attribute 113
CIPHERS attribute
CORBASERVER resource 113
TCPIPSERVICE resource 113
URIMAP resource 113
CKBM security 423
CKCN security 423
CKDL security 423
CKDP security 423
CKQC transaction
security 423
CKRS security 423
CKRT security 423
CKSD security 423
CKSQ security 423
CKTI transaction
CKTI transaction *(continued)*
security 423
classification of data and users 90
CLAUTH (class authority) attribute
installation-defined classes 86
client certificate 392, 393
client program
MRO logon and bind-time security 453
ClientAuthType 404, 408
clock skew
changing 43
setting 43
clock_skew 43
CLOUD.APPLICATION security profile 374, 417
CLOUD.DEF security profile 374, 417
CLOUD.PLATFORM security profile 374, 417
CMDSEC 458
command authorization
Db2 446
command security
authorization failures 471
XUSER parameter 495
command-level security 422
conditional access processing 493
configuration files
Security Token Service 44
configuring 39
connection authorization 440
connection exit routine 441
connection security 425
CONSOLE general resource class
description 493
coupling facility data table pool 466
coupling facility data tables security 465
COVA 344
COVC 344
COVE 344
COVG 344
COVP 344
COVU 344
CPLT transaction 57
CRL (certificate revocation list) 116
CSD (CICS system definition file), restricting batch access to 495
CSD definitions, locking 496
CSMI (CICS-supplied mirror transaction)
authorizing the link user ID 454
security 454
custom security handler 314

D

data set
encryption 450
data set security
access to CICS data sets 476
access to user data sets 479
APPLID parameter 477
CICS installation requirements 469
CICS system 470
MVS library lookaside (LLA) facility 478
data tables
bind security 464
CONNECT security checks 464

data tables *(continued)*
coupling facility 465
file security 465
security checking 463
server authorization security check 463
Db2 security
accounting 444
authorization to execute a plan 447
establishing authorization IDs 443, 445
primary authorization ID 440
secondary authorization ID 440
security mechanisms 433
DB2ENTRY resource classes 85
DCICSDCT general resource class
defining profiles 498
default user, CICS
defining 474
defined activity userid 421
defined process userid 421
definitions, protecting
controlling access to resources 349
DELMEM operand 492
deploying 38
DES (data encryption standard) 100
DES authentication 457
developing
applications
Kerberos 35
DFH$RACF 86
DFHAPPL FACILITY class profiles, defining 453
DFHHTML 346
DFHIRP (interregion communication program)
security checks performed by 453
DFHWBPW, password expiry management program 392, 397
DFLTUSER parameter
obtaining user data 83
DFLTUSER SIT parameter
creating a security environment 340
diagnostic messages 408
document template security
XRES parameter 402
document templates
security 399, 400
domain 408
DSN3SATH sample 441
DSN3SSGN sample 445
dynamic parse validation routines 469

E

ECICSDCT general resource class
defining profiles 498
enabling a security policy 274
encryption
data sets 450
ENCRYPTION parameter
and SP800-131A 114
ESDSs, VSAM, access to 479
ESM (external security manager)
invoking another
overview of interface 385
RACROUTE macros 386
PassTickets 487
using RACF
ESM (external security manager) *(continued)*
using RACF *(continued)*
controlling CICS security 377
creating profiles 340, 349
evaluation sequence, security 381
example tasks
security 386
examples
attach-time security 422
RACF commands 421, 422
surrogate security checking 421
EXEC CICS LINK command
DFHAPPL profile definition 454
security checking 454
EXEC CICS QUERY SECURITY 457
EXEC CICS START USERID 457
EXEC CICS VERIFY PASSWORD 457
external CICS interface (EXCI)
CALL interface
DFHAPPL profile definition 454
security 453
external security manager (ESM)
invoking another
overview of interface 385
RACROUTE macros 386
using RACF
controlling CICS security 377
creating profiles 340, 349
EXTRACT CERTIFICATE command 392
EYUCOVE 346
EYUCOVI 346
EYULOG 346
EYUWREP 346
EYUWUI 346

F

FACILITY class profiles, defining 453
file resource security checking 467
file security
data set profiles 88
FIPS compliance 114

G

GCICSTRN general resource class 501
general resource classes
ACICSPCT 500, 503
BCICSPCT 500, 503
JCICSJCT 499
JESSPOOL 485
KCICSJCT 499
MCICSPPT 504
NCICSPPT 504
OPERCMDS 496
PCICSPSB 509
PROPCNTL 484
QCICSPSB 509
SURROGAT 485
VTAMAPPL 484
general resource profiles
refreshing 380
generating and using RACF PassTickets 21

generic connection
  note about lack of security checks 453
generic profiles
  SETROPTS GENERIC 88, 492
generic resource names (z/OS Communications Server)
  z/OS Communications Server generic resource 483
generic resource profiles 510
global security parameters 380
GRANT command 446
group profiles 83
group-SPECIAL attribute 80
GTERMINAL definition 491
GTERMINL definition 491

H

handshakerole=serverwithclientauth 409
HFS file security
  XHFS parameter 507
HTTP client requests
  security 393

I

ICHRIN03 started procedures table 342
ICHRIN03, RACF started task table 471
in-storage profiles
  GTERMINL profiles 491
invoking the trust client 313
IRR.DIGTCERT.ADD profile 108

J

Java security manager 274
java.security.policy 274
JCICSJCT general resource class 499
JES spool protection 485
JESSPOOL general resource class 485
job submission, surrogate 485
journal security
  access authorization levels 499
  defining resource classes 499
  XJCT parameter 499
journals and log streams
  journal access authorization levels 499

K

KCICSJCT general resource class 499
Kerberos
  configuring web services 34
  developing applications 35
key rings 104
keyring
  building 104
KEYRING 404, 408

L

labels, RACF security 90
language segment
  system defaults 82
language segment *(continued)*
  user profile 82
LDAP server
  configuring 116
levels, RACF security 90
libraries, CICSPlex SM
  protecting with RACF 336
load libraries, protecting 470
LOCK command, CEDA 495
log streams
  authorizing access to 475
  LOGSTRM general resource class 475
logon security 453

M

maps
  security considerations 346
marking a certificate untrusted 109
MCICSPPT general resource class 504
members, group
  ADDMEM operand to add 492
  DELMEM operand to remove 492
message handler
  invoking trust client 313
MINTLSLEVEL/ENCRYPTION 404, 408
MQCONN
  security 424
MQMONITOR
  security 424, 425
multiple volumes 479
multiregion operation (MRO)
  logon and bind time security 453
MVS
  library lookaside (LLA) facility 478
  password and RACF authorization checking 469
  program properties table (PPT) 469

N

National Language Support 82
NATLANG and non-terminal transactions 82
NCICSPPT general resource class 504
NETNAME terminal definition 492
NIST conformance 114
Null authentication 457

O

OPCLASS 81
operator, CICS terminal
  obtaining data for 83
operator, terminal
  data for default user 83
OPIDENT 81
OPPRTY 81
OSGi security 274

P

parameter
  authorizing access to CICS region 483
  protecting CICS data sets 477

parameters
security
activating 379
checking 379
global 380
PassTicket 458
PassTickets, for signon security 21, 487
password expiry management program DFHWBPW 392, 397
patterns
SAML programs 533
Patterns
SAML programs 534
PCICSPSB general resource class 509
PERMIT command
WHEN operand 493
persistent sessions
XRFSOFF operand 82
pipeline
provider
configuring for SAML tokens 40
requester
configuring for SAML 42
PKCS (public key cryptography standard) 100
platform
security 349, 374, 417
PLT
post-initialization processing 57
PLTPIUSR system initialization parameter 57
POSIT numbers
installation-defined general resource classes 86
post-initialization processing, surrogate security 57
PREFIX attribute definition 505
preset terminal NATLANG 82
preset terminal security
CEDA LOCK command 495
CEDA transaction 495
controlling definition and installation 495
restricting batch access to CSD 495
SURROGAT transaction 495
using MVS system console as CICS terminal 496
process
security of 421, 422
PRODCFT1 466
profile 87
profiles
ACICSPCT general resource class 500, 503
BCICSPCT general resource class 500, 503
data set 88
DCICSDCT general resource class 498
ECICSDCT general resource class 498
GCICSTRN general resource class 501
JCICSJCT general resource class 499
JESSPOOL 485
KCICSJCT general resource class 499
MCICSPPT general resource class 504
NCICSPPT general resource class 504
PCICSPSB general resource class 509
PROPCNTL 484
QCICSPSB general resource class 509
refreshing in main storage 85
resource and WARNING option 50
resources, defining generic 510
SETROPTS command 491
SURROGAT general resource class 485
profiles *(continued)*
TCICSTRN general resource class 501
terminal (PoE), defining 491
USER parameter on CICS JOB statement 472
VTAMAPPL 484
program properties table (PPT), MVS 469
program security
XPPT parameter 504
propagation of userid, controlling 484
PROPCNTL general resource class
defining profiles 484
provider pipeline
configuring
for SAML tokens 40
proxy authentication 393
PSB security
access authorization levels 509
defining resource classes 509
PCICSPSB general resource class 509
QCICSPSB general resource class 509
XPSB parameter 509
PSBCHK parameter 509
pthreads 101
public key encryption 100

Q

QCICSPSB general resource class 509

R

RACDCERT command 104, 108
RACF
example commands 421, 422
external security manager 431
profile 87
RACF (resource access control facility)
administration 80
CICS installation requirements 469
CICS segment 81
data set profiles 88
defining default CICS userid 474
defining port of entry profiles 491
defining your own resource class names 85
generic resource profiles 510
group profile 83
group profiles 83
language segment 82
RACF segment 81
refreshing resource profiles in main storage 85
security labels 90
security levels 90
terminal profiles 491
undefined terminals 492
user profiles 81
with multiple MVS images 469
RACF (Resource Access Control Facility)
controlling access to resources 349
defining transactions 343, 344
exempting items from security checking 378
RACF class
DSNR 441
RACF commands

RACF commands *(continued)*
ADDUSER, example for default CICS userid 474
DELMEM operand 492
RALTER 492
RACF default resource profiles
VCICSCMD general resource class 435
RACF list of groups option 445
RACF PassTickets 21
RACF profiles
refreshing 380
RACF Secured Sign-on 458
RACF SPECIAL authority 108
RACROUTE macros, for security 386
RCICSRES 86
RDO
restricting use of transaction 495
refreshing RACF profiles 380
requester pipeline
configuring
for SAML 42
resource access control facility (RACF) 453
Resource Access Control Facility (RACF)
controlling access to resources 349
defining transactions 343, 344
exempting items from security checking 378
resource checker (CICS ONC RPC)
writing 459
resource classes, CICSPlex SM
controlling access to 349
resource definition parameters
RESSEC 509
resource group
DELMEM operand to remove 492
resource names, CICSPlex SM 349
resource security
ACICSPCT general resource class 500, 503
application programs 504
BCICSPCT general resource class 500, 503
DCICSDCT general resource class 498
defining generic profiles 510
defining your own resource class names 85
document templates 402
ECICSDCT general resource class 498
GCICSTRN general resource class 501
HFS files 507
JCICSJCT general resource class 499
journals and log streams 499
KCICSJCT general resource class 499
MCICSPPT general resource class 504
NCICSPPT general resource class 504
PCICSPSB general resource class 509
profiles and WARNING option 50
program specification blocks 509
QCICSPSB general resource class 509
refreshing profiles in main storage 85
TCICSTRN general resource class 501
temporary storage 505
transient data queues 498
XHFS parameter 506, 507
XJCT parameter 499
XPCT parameter 500, 503
XPPT parameter 504
XPSB parameter 509
XRES parameter 402
resource security *(continued)*
XTST parameter 505
z/OS UNIX files 506
resource-level security 421
RESSEC 458
RRCDTE sample job 434

S

SAML
configuring CICS 38
developing applications 533
SAML tokens
expired 43
modifying 43
not yet valid 43
re-signing 43
signing 43
SAML-aware programs
augmenting tokens 534
outbound request 534
sample connection exit routine (DSN3SATH) 441
sample sign-on exit routine (DSN3SSGN) 445
SASS (single address space) 440
scoping sign-on definition 488
SEC system initialization parameter 457
Secured Sign-on 458
security
access to WUI resources 346
alias transaction 400
application-generated responses 400
authentication 392, 393, 399
AUTHTYPE 436
basic authentication 392, 393, 397
CICS system 400
command security 433
COVA 344
COVC 344
COVE 344
COVG 344
COVP 344
COVU 344
DB2TRAN resource security 434
defining RACF profiles 434
DFHHTML 346
document templates 399, 400
EYUCOVE 346
EYUCOVI 346
EYULOG 346
EYUWREP 346
EYUWUI 346
identification 392, 393
inbound ports 399
of activities
attach-time 422
defined activity userid 421
resource-level 421
of BTS commands 422
of processes
attach-time 422
defined process userid 421
resource-level 421
password expiry management 397
password phrase expiry management 397

security *(continued)*
planning 335
port number
security 399
proxy authentication 393
RACF 431
RACF class, DSNR 441
resource level 399
resource security 433
SAML 38
SASS 440
sign-on 21, 487
SSL 403
surrogate user checking 436
the CICSESM interface 91, 385
using PassTickets 21, 487
z/OS UNIX files 399, 400
z/OS UNIX System Services 399, 400
security categories 90
security checking
controlling CICS 377
ESM interface 385
evaluation sequence 381
exempting items 378
parameters 379
with another ESM 385
with RACF 336
security classification of data and users 90
security considerations 378
security handler
writing your own 314
security labels 90
security levels 90
security manager
applying a security policy 274
enabling a security policy 274
security profiles
refreshing 380
security profiles, RACF
creating 340
views protected by 351
security tasks, example 386
security token extensions
configuring provider pipeline 40
configuring requester pipeline 42
security token of JES spool files 485
Security Token Service configuration file 44
security, DBCTL
PSB authorization checking by CICS 450
security, surrogate 436
SECURITYPREFIXID 467
segment
CICS 81
LANGUAGE 82
RACF 81
server program
security considerations 453
SETROPTS command
generic terminal profiles 492
generic user profiles 88
shared data tables
bind security 464
CONNECT security checks 464
file security 465
shared data tables *(continued)*
security checking 463
server authorization security check 463
sign-off
after persistent sessions restart 82
after XRF takeover 82
process 489
sign-off process 489
sign-on
after persistent sessions restart 82
after XRF takeover 82
sign-on exit routine 445
sign-on security 21, 487
signature verification
SAML 43
simulated CICS security 377
single address space (SASS) 440
SIOCTTLSCTL ioctl 404, 408
SIT parameters, CICS
security-related 379
skew time
changing 43
setting 43
SMF (System Management Facility) 79
SNA LU
SNA ACB access 484
source libraries, protecting 470
SP800-131A 114
specific connection
MRO logon security checks 453
spool files, security token 485
SPOOLOPEN commands 485
SQL
dynamic 448
static 448
SSL
client certificate authentication 392, 393
for CICS as an HTTP client 403
SSL connection improvements 101
SSL pool 101
SSL(ATTLSAWARE) 409
started jobs
defining CICS region userid 471
started task
and RACF userid 81
authorizing CICS procedures 471
started transaction security 500, 503
STS configuration file 44
sts-config.xml 44
sts.xml 44
sts.xsd 44
SURROGAT general resource class 485, 495
SURROGAT transaction 495
surrogate job submission
to JES internal reader 485
surrogate security checking 421
surrogate user security
post-initialization processing 57
symmetric encryption 100
system data set
authorizing access to 476
generic profiles needed 476
levels of access to 476
protecting 470

system initialization parameters, CICS
PSBCHK 509
XDCT 498
XHFS 506, 507
XJCT 499
XPCT 500, 503
XPPT 504
XPSB 509
XRES 402
XTST 505
System Management Facility (SMF) 79
system security
CICS installation requirements 469

T

tasks, example
security 386
TCICSTRN general resource class 501
TCPIPSERVICE 409
TCPIPSERVICE resource definition
security 399
SSL
URIMAP resource definition 399
temporary storage
authorizing access to named counter servers 481
authorizing access to the named counter pools 481
authorizing access to the TS pools 479
authorizing access to TS servers 480
defining resource classes 505
terminal
emulators 35
TERMINAL definition 491
terminal security
CEDA LOCK command 495
controlling access 490
obtaining CICS-related data for a user 83
preset 494
sign-on 488
terminal profiles 491
undefined terminals 492
universal access authority 492
using MVS system console as CICS terminal 496
Terminals, defining individual profiles 492
TIMEOUT 81
TLS12 114
trace points 408
transaction security
started transactions 500, 503
XJCT parameter 500, 503
XPCT-checked transactions 500, 503
transactions
in a CMAS
defining to RACF 343
in a MAS
defining to RACF 344
Kerberos 35
transient data
security considerations 498
trigger level transactions
default security for 475
trust client
invoking 313
TSO command
TSO command *(continued)*
refreshing using TSO command 85

U

UNIX authentication 457
untrusted certificate, marking 109
URP_DISASTER response
to resource checker 461
URP_EXCEPTION response
to resource checker 460
URP_INVALID response
to resource checker 460
URP_OK response
to resource checker 460
user ID
security 399, 400
USER parameter on CICS JOB statement 472
user profiles
in RACF 81
refreshing 380
user security
user profiles 81
userid
ADDUSER to add default CICS userid 474
controlling propagation of 484
defining CICS default user 474
defining for CICS 472
of CICS region as security token 485
surrogate job submission 485

V

validate 39
validating 39
VCICSCMD general resource class 435
verifying
signatures 43
views
security considerations 346
views protected by security profiles 351
VSAM data sets, and BWO 479
VSAM data sets, and Dynamic Volume Count 479
VSAM ESDSs, access to 479
VTAMAPPL general resource class
defining profiles 484

W

WARNING option 50
WCICSRES 86
web application 185, 188, 191, 195, 201
web services
configuring
Kerberos 34
WHEN operand of PERMIT
WHEN operand 493

X

XCICSDB2 general resource class 434
XCICSDB2 member class 434
XCMD system initialization parameter 458

XDCT, system initialization parameter 498
XHFS, system initialization parameter 506, 507
XJCT, system initialization parameter 499
XPCT-checked transaction security 500, 503
XPCT, system initialization parameter 500, 503
XPPT system initialization parameter 457
XPPT, system initialization parameter 504
XPSB, system initialization parameter 509
XRES, system initialization parameter 402
XRF (extended recovery facility)
    XRFSOFF operand 82
XTST, system initialization parameter 505
XUSER system initialization parameter 457
XUSER, system initialization parameter 495

Z

z/OS Communications Server
    generic resource names 483
z/OS UNIX file security
    XHFS parameter 506
z/OS UNIX files
    security 399, 400
z/OS UNIX Systems
        Services
    security 399, 400
ZCICSDB2 grouping class 434