

IBM Security Access Manager
Version 9.0.7
June 2019

*Advanced Access Control Configuration
topics*



IBM Security Access Manager
Version 9.0.7
June 2019

*Advanced Access Control Configuration
topics*



Contents

Figures	vii
----------------	------------

Tables	ix
---------------	-----------

Chapter 1. Upgrading configuration . . . 1

Upgrading external databases with the dbupdate tool (for appliance at version 9.0.0.0 and later)	2
Upgrading a SolidDB external database (for appliance versions earlier than 9.0.0.0)	3
Upgrading a DB2 external runtime database (for appliance versions earlier than 9.0.0.0)	4
Upgrading an Oracle external runtime database (for appliance versions earlier than 9.0.0.0)	5
Setting backward compatibility mode for one-time password	6
Updating template files	6
Updating PreTokenGeneration to limit OAuth tokens	7
Reviewing existing Web Reverse Proxy instance point of contact settings	8
Upgrading the signing algorithms of existing policy servers	9

Chapter 2. Getting started with Advanced Access Control . . . 13

Chapter 3. Managing application interfaces . . . 15

Chapter 4. Managing the runtime component . . . 17

Chapter 5. Managing user registries . . 19

Chapter 6. Runtime security services external authorization service . . . 21

Configuring runtime security services for client certificate authentication	21
Permitting access decisions when runtime security services cannot be contacted	23
Retaining the version 7.0 attribute IDs in existing policies	24

Chapter 7. Adding runtime listening interfaces . . . 25

Chapter 8. Support for compliance with NIST SP800-131a . . . 27

Chapter 9. Authentication. . . 31

Authentication Service configuration overview	35
Authentication configuration scenarios	37
Configuring step-up authentication	37

Configuring authentication	39
Configuring an HOTP one-time password mechanism	40
Configuring a TOTP one-time password mechanism	42
Configuring a MAC one-time password mechanism	45
Configuring an RSA one-time password mechanism	46
Configuring one-time password delivery methods	50
Configuring username and password authentication	54
Configuring an HTTP redirect authentication mechanism	56
Configuring consent to device registration	57
Configuring an End-User License Agreement authentication mechanism	59
Configuring an Email Message mechanism	60
HTML format for OTP email messages	62
Configuring the reCAPTCHA Verification authentication mechanism	62
Configuring an Info Map authentication mechanism	64
Embedding reCAPTCHA verification in an Info Map mechanism	66
Available parameters in Info Map	66
Embedded Cloud Identity API calls in an Info Map mechanism	67
Configuring a Knowledge Questions authentication mechanism	68
Configuring a FIDO Universal 2nd Factor authentication mechanism	71
Configuring a FIDO2/WebAuthn authentication mechanism	73
Configuring a QR Code authentication mechanism	74
Enabling or disabling authentication policies	75
Managing mapping rules	75
Authentication Service Credential mapping rule	77
OTPGetMethods mapping rule	78
OTPGenerate mapping rule	79
OTPDeliver mapping rule	79
OTPVerify mapping rule	80
Customizing one-time password mapping rules to use access control context data	81
One-time password and authentication template files	83
Push notification registration	83
Obtaining the required authentication credentials to configure push notification for IBM Verify	84
Cloud Identity API Integration	85
Cloud Identity JavaScript	86
Authentication flow	86
User Self Care flow	87
Configuring the authentication and access module for cookieless operation	88
Reverse Proxy Configuration with Authentication Services	89
Configuring advanced access control authentication on a reverse proxy	89
Using the isamcfg tool	90

Chapter 10. OAuth 2.0 and OIDC

support 105

OAuth and OpenID Connect concepts	105
OAuth 2.0 concepts	105
OpenID Connect concepts	106
OAuth 2.0 endpoints	108
OAuth 2.0 and OIDC workflows	111
Client authentication considerations at the	
OAuth 2.0 token endpoint	120
Configuring an authenticated token endpoint	
with WebSEAL as the point of contact	121
State management	122
Trusted clients management	123
Proof Key for Code Exchange support	124
Reverse proxy configuration for OAuth and OIDC	
provider	125
Configuring a reverse proxy for OAuth and an	
OIDC Connect provider	125
Viewing a reverse proxy log for an automated	
configuration	127
Example reverse proxy log for OAuth and	
OIDC configuration	128
Removing reverse proxy configuration for	
OAuth and OIDC provider	130
Configuring API protection	131
Creating an API protection definition	131
Managing API protection definitions	132
API Protection token management properties	133
API Protection OpenID Connect Provider	
properties	135
PIN policy	137
Registering an API protection client	137
Managing registered API protection clients	139
Managing policy attachments	140
Using oauthScope attributes in an access control	
policy	143
Uploading OAuth response files	144
OAuth introspection	144
OAuth revocation endpoint	146
OIDC Claims customization	147
Client authentication to /token through an	
incoming JSON Web Token	153
Passing parameters through JWT in a request to	
/authorize	154
Mapping rules for OAuth and OIDC	156
Managing OAuth 2.0 and OIDC mapping rules	156
OAuth 2.0 and OIDC mapping rule methods	157
OAuth and OIDC mapping rules files	157
OAuth and OIDC mapping rules actions	158
Customizing OAuth tokens by updating the	
sample PreTokenGeneration mapping rule	163
OpenID Connect mapping rules	164
Device flows verification uri	166
OAuth 2.0 template files	166
OAuth 2.0 template page for consent to authorize	167
Error responses	170
User self-administration tasks for OAuth	170
Managing OAuth 2.0 authorization grants	170
APIs for managing OAuth 2.0 authorization	
grants	171

OAuth STS Interface for Authorization	
Enforcement Points	173
API Protection form post response mode	180
Access policy for OAuth or OIDC	181
Making an OAuth or OIDC consent decision	
using access policy	181
OIDC Dynamic Clients	182
OIDC Dynamic Clients- Authentication and	
deployment	182
OIDC Dynamic Clients- Register a client	183
OIDC Dynamic Clients- Retrieve a dynamic	
client	183
OIDC Dynamic Clients- Custom Identifiers	184
OIDC Dynamic Clients- Update a client	184
OIDC Dynamic Clients- Delete a client	185

Chapter 11. Mobile Multi-Factor

Authentication 187

Authenticator registration	187
Authentication method enrollment	188
Configuring Mobile Multi-Factor Authentication	188
Configuring a Mobile Multi-Factor Authentication	
(MMFA) Authenticator Mechanism	189
MMFA mapping rule methods	190

Chapter 12. FIDO and WebAuthn

Support 193

FIDO2 Server Endpoints	193
Concepts	193
WebAuthn Ceremonies	194
Attestation	195
Public Key Algorithms	197
Metadata	197
Registration	197
U2F Migration	199
FIDO2 Configuration	199
Create a FIDO2 Relying Party	199
Authentication Service Mechanism	201
Limitations	202
FIDO2 Mediation	202

Chapter 13. Access control policies 205

Defining a custom application for policy	
attachments	205
Invoking the RTSS XACML engine	206
ContextId JSON example	206
ApplicationId JSON example	207
resource-id JSON example	208

Chapter 14. Defining a custom domain for policy attachments 211

Chapter 15. Deploying pending changes 213

Chapter 16. Options for handling session failover events 215

Option 1: No handling of failover events	215
--	-----

Option 2: The distributed session cache.	215
--	-----

Chapter 17. Global settings 217

Managing advanced configuration	218
Advanced configuration properties	219
Managing user registries	242
Tuning runtime application parameters and tracing specifications	243
Template files	249
Managing template files	249
Customizing the consent page.	251
Template page scripting	251
Template files reference	254
Template file macros	281
Mapping rules	284
Managing JavaScript mapping rules	284
Managing mapping rules	285
Managing OAuth 2.0 mapping rules.	292
Actions to be performed in mapping rules.	294
MMFA mapping rule methods.	295
JavaScript whitelist	297
Managing JavaScript mapping rules	300

Customizing SAML 2.0 identity mapping	301
STSRequest and STSResponse access using a JavaScript mapping rule.	303
OpenID Connect mapping rules	310
Import a mapping rule from another mapping rule.	311
Auditing from Mapping Rules.	312
Managing Distributed Session Cache	315
Managing server connections	315
Server connection properties	317
Point of contact profiles	320
Creating a point of contact profile	320
Updating or viewing a point of contact profile	321
Deleting a point of contact profile	321
Setting a current point of contact profile	322
Callback parameters and values	322

Chapter 18. Choose a synchronization mode 325

Index 327

Figures

- | | | |
|----|---|-----|
| 1. | WebSEAL client in an environment with multiple IBM Security Access Manager servers. | 94 |
| 2. | OAuth 2.0 JavaScript sample code with state management. | 123 |
| 3. | Template for user_consent.html | 169 |
| 4. | OAuth STS trust chain workflow | 173 |
| 5. | OAuth authorization enforcement point workflow | 180 |

Tables

1. Advanced Access Control configuration upgrade tasks	1
2. Scenario 1	10
3. Scenario 2	11
4. Runtime security services EAS access decisions	21
5. OAuth 2.0 endpoint definitions and URLs	109
6. Response type values for each flow	115
7. Configurations supported	121
8. OAuth modes	126
9. Reverse proxy instance	126
10. Reuse configuration	127
11. Auto-configuration log files	128
12. Mapping rule variable for OAuth revocation	147
13. Claims types	148
14. Example configuration of Attribute Sources	150
15. Attribute Mapping	150
16. LDAP Attribute Source example	150
17. Chain modules for JWT format	153
18. Mapping Rules	158
19. New request types	165
20. FIDO2 Server Endpoints	193
21. Configuration data types	219
22. Filter by Category	219
23. HTTP proxy properties	246
24. Valid trace levels	248
25. Example JavaScript	252
26. Default template files in the ac/ directory	255
27. Default template files in the mga/ directory	255
28. Default template files in the authsvc/ directory	257
29. Default template files in the otp/ directory	257
30. Default template files in the authsvc/authenticator/password/ directory	260
31. Default template files in the authsvc/authenticator/http_redirect/ directory	260
32. Default template files in the authsvc/authenticator/macotp/ directory	261
33. Default template files in the authsvc/authenticator/rsa/ directory	262
34. Default template files in the authsvc/authenticator/totp/ directory	262
35. Default template files in the authsvc/authenticator/hotp/ directory	263
36. Default template files in the authsvc/authenticator/consent_register_device/ directory	263
37. Default template files in the authsvc/authenticator/eula/ directory	264
38. Default template files in the authsvc/authenticator/knowledge_questions/ directory	265
39. Default files in the proper/ directory	266
40. Default files in the oauth20/ directory	268
41. SAML 2.0 HTML page identifiers and macros	271
42. Supported consent values for SAML 2.0 response	280
43. JSON to SAML2 module chain values	305
44. SAML2 to JSON module chain values	308
45. Server Connection properties	317
46. Tuning properties	319

Chapter 1. Upgrading configuration

After you install the upgrade software on a Security Access Manager appliance, you must complete several configuration tasks.

Review the following tasks, and perform the ones that are appropriate for your installation.

Table 1. Advanced Access Control configuration upgrade tasks

Upgrade task	See
Run the isamcfg tool to obtain the correct configuration settings for WebSEAL and Security Access Manager.	"Using the isamcfg tool" on page 90
If you use solidDB as an external runtime database, upgrade solidDB.	"Upgrading a SolidDB external database (for appliance versions earlier than 9.0.0.0)" on page 3 or "Upgrading external databases with the dbupdate tool (for appliance at version 9.0.0.0 and later)" on page 2
If you use DB2® as an external runtime database, upgrade DB2.	"Upgrading a DB2 external runtime database (for appliance versions earlier than 9.0.0.0)" on page 4 or "Upgrading external databases with the dbupdate tool (for appliance at version 9.0.0.0 and later)" on page 2
If you use Oracle as an external runtime database, upgrade Oracle.	"Upgrading an Oracle external runtime database (for appliance versions earlier than 9.0.0.0)" on page 5 or "Upgrading external databases with the dbupdate tool (for appliance at version 9.0.0.0 and later)" on page 2
After an upgrade, the setting for one-time password authentication is set to run in backward compatibility mode. Disable this mode.	"Setting backward compatibility mode for one-time password" on page 6
If you customized any template files, upgrade them.	"Updating template files" on page 6
The new role membership features in later versions of the appliance are granted the permission of None by default. Configure the permissions for these new role membership features, if necessary.	Managing roles of users and groups
To limit the number of OAuth tokens per user per definition, update the mapping rule.	"Updating PreTokenGeneration to limit OAuth tokens" on page 7
Review and update some existing Web Reverse Proxy instance point of contact settings for the Advanced Access Control runtime.	"Reviewing existing Web Reverse Proxy instance point of contact settings" on page 8

After you complete the appropriate configuration tasks, go to Chapter 2, "Getting started with Advanced Access Control," on page 13.

Upgrading external databases with the dbupdate tool (for appliance at version 9.0.0.0 and later)

Use the **dbupdate** tool that is provided by the appliance to upgrade your external runtime databases, such as DB2, Oracle, PostgreSQL, and solidDB.

About this task

Use this task if your Security Access Manager appliance version is 9.0.0.0 or later. If your appliance version is earlier than 9.0.0.0, you must first update the appliance and the external database to version 9.0.0.0 by using the following methods:

- “Upgrading a SolidDB external database (for appliance versions earlier than 9.0.0.0)” on page 3
- “Upgrading a DB2 external runtime database (for appliance versions earlier than 9.0.0.0)” on page 4
- “Upgrading an Oracle external runtime database (for appliance versions earlier than 9.0.0.0)” on page 5

You must have Java version 8 or later to run the database update tool.

Procedure

1. In the local management interface, go to **Manage System Settings > Secure Settings > File Downloads**.
2. Expand **access_control > database**.
3. Select the dbupdate9.zip file. This file contains the database update tool (dbupdate.jar), the README file, and update files for all databases and SQL types.
4. Click **Export**.
5. Save the file.
6. Extract the dbupdate9.zip file and run the dbupdate.jar tool for your environment.

- The usage of the tool is as follows:

```
${JAVA_HOME}/bin/java -jar dbupdate.jar [-t] [-n]  
<dbType> <sqlType> <connectString> <user> [<password>]
```

-t Enable debug trace output.

-n Do not perform any updates, but instead output the update operations that would have been executed.

dbType The database type. Valid values are config or runtime.

sqlType

The database server type. Valid values are db2, oracle, postgresql, or soliddb.

connectString

The string that is used when establishing a connection to the database server.

db2 Typically the database name, which is used in "db2 CONNECT TO <connectString> USER <user> USING <password>;".

oracle If the string is not empty, it corresponds to the value used for

sqlplus "CONNECT <user>/<password>@<connectString>". If the string is empty (""), the ORACLE_SID environment variable is used.

postgresql

The connectString for PostgreSQL contains the psql command line options that need to be supplied to connect to the database. This usually consists of the hostname, port and database name. For example:

```
'--host=<hostname> --port=<port> --dbname=<db name>'
```

soliddb

Typically "tcp <host> <port>", which is used in the "sqlsql <options> <connectString> <user> <password>" command.

user Database user to update the database with.

password

Database user's password. If not provided, the value is read from console.

Note: For DB2 external databases, the **db2** command must be in the path. For Oracle external databases, the **sqlplus** command must be in the path. For PostgreSQL external database, the **psql** command must be in the path. For SolidDB external databases, the **sqlsql** command must be in the path.

- Java 8 or later is required for running the tool.
- The following command shows an example of using the tool to update a DB2 database:

```
/opt/ibm/java-x86_64-80/jre/bin/java -jar dbupdate.jar runtime db2  
"HVDB" db2inst1 passw0rd
```

The following command shows an example of using the tool to update an Oracle database:

```
/opt/ibm/java-x86_64-80/jre/bin/java -jar dbupdate.jar runtime oracle  
"" SYSTEM passw0rd
```

- If the tool is not flexible enough for your environment, you can see which commands the tool runs without executing them and manually run those commands to suit your environment. To see which commands the update tool runs against the database, run the tool with the -n option. This option still must be able to read from the database to get the current update versions in order to determine which updates to apply. However, it will not execute any update operations.
- See the README file that is included in the dbupdate9.zip file for more details about the update tool.

Upgrading a SolidDB external database (for appliance versions earlier than 9.0.0.0)

If SolidDB is the external runtime database, upgrade it after you install the appliance upgrade so that you are using the correct .sql file version.

About this task

Attention: Use this task only if you have not installed v9000.

The updates to the .sql file for each release are not cumulative. Therefore, depending on which version of the isam_access_control_soliddb_update_v*.sql file you installed, make the following updates:

- v8003.sql is not installed: First upgrade to v8003, then to v8004, then to v8005, and finally to v9000.
- v8003.sql is installed: First upgrade to v8004, then to v8005, and finally to v9000.
- v8004.sql is installed: First upgrade to v8005, and then to v9000.
- v8005.sql is installed: Upgrade to v9000.

Use this task to install each version you require, replacing references to isam_access_control_soliddb_update_v9000.sql with the version number you are upgrading to each time, and starting with the earliest version you require.

Procedure

1. Log in to the local management interface.
2. Click **Manage System Settings > File Downloads**.
3. Expand **access_control > database > soliddb > runtime**.
4. Select isam_access_control_soliddb_update_v9000.sql
5. Click **Export**.
6. Save the file.
7. Log in to the **solsql** utility.

```
/opt/solidDB/soliddb-7.0/bin/solsql network_name username password  
network_name
```

The network name of the solidDB server.

username

The user name for the database administrator.

password

The password for the database administrator.

8. Run the following command in the SolidDB SQL Editor:
@fully_qualified_path_to_script

The following example shows the fully qualified path to the script:

```
@/tmp/isam_access_control_soliddb_update_v9000.sql
```

9. Validate that the tables were successfully updated.
10. Ensure that no errors were returned during the update and log in to the database to manually check that the tables exist.

Upgrading a DB2 external runtime database (for appliance versions earlier than 9.0.0.0)

If DB2 is the external runtime database, upgrade it after you install the appliance upgrade so that you are using the correct .sql file version.

About this task

Attention: Use this task only if you have not installed v9000.

The updates to the .sql file for each release are not cumulative. Therefore, depending on which version of the isam_access_control_db2_update_v*.sql file you installed, make the following updates:

- v8004.sql not installed: First upgrade to v8004, then to v8005, and finally to v9000.
- v8004.sql is installed: Upgrade to v8005 and then to v9000.
- v8005.sql is installed: Upgrade to v9000.

Use this task to install each version you require, replacing references to isam_access_control_db2_update_v9000.sql with the version number you are upgrading to each time, and starting with the earliest version you require.

Procedure

1. Log in to the local management interface.
2. Click **Manage System Settings > File Downloads**.
3. Expand **access_control > database > db2 > runtime**.
4. Select isam_access_control_db2_update_v9000.sql
5. Click **Export**.
6. Save the file.
7. Open the isam_access_control_db2_update_v9000.sql file in an editor on the DB2 server.
8. Replace the following macros with the values specific to your environment:

&DBINSTANCE

The name of the DB2 instance.

&DBUSER

The name of the DB2 administrator.

&DBPASSWORD

The password for the DB2 administrator.

9. Save the changes.
10. Log in to the DB2 Command utility (Windows) or DB2 host (UNIX) as the DB2 administrator.
11. Run the following command:

```
db2 -tsvf fully_qualified_path_to_script
```

The following example shows the fully qualified path to the script:

```
db2 -tsvf tmp/isam_access_control_db2_update_v9000.sql
```

12. Validate that the tables were successfully updated.
13. Ensure that no errors were returned during the update and log in to the database to manually check that the tables exist.

Upgrading an Oracle external runtime database (for appliance versions earlier than 9.0.0.0)

If Oracle is the external runtime database, upgrade it after you install the appliance upgrade so that you are using the correct .sql file version.

About this task

Attention: Use this task only if you have not installed v9000.

The updates to the .sql file for each release are not cumulative. Therefore, depending on which version of the isam_access_control_oracle_update_v*.sql file you installed, make the following updates:

- v8011.sql not installed: First upgrade to v8011 and then to v9000.
- v8011.sql is installed: Upgrade to v9000.

Use this task to install each version you require, replacing references to isam_access_control_oracle_update_v9000.sql with the version number you are upgrading to each time, and starting with the earliest version you require.

Procedure

1. Log in to the local management interface.
2. Click **Manage System Settings > File Downloads**.
3. Expand **access_control > database > oracle > runtime**.
4. Select isam_access_control_oracle_update_v9000.sql
5. Click **Export**.
6. Save the file.
7. Copy the downloaded isam_access_control_oracle_update_v9000.sql file into the Oracle home directory. For example: ORACLE_HOME=/opt/oracle/app/oracle/product/11.2.0/dbhome_1
8. Log in to SQL*Plus.
9. At the SQL prompt, run **START isam_access_control_oracle_update_v9000.sql**.
10. Validate that the tables were successfully updated.
11. Ensure that no errors were returned during the update and log in to the database to manually check that the tables exist.

Setting backward compatibility mode for one-time password

When you upgrade your installation, the setting for one-time password authentication is set to run in backward compatibility mode. Complete this task if you need to disable this mode.

About this task

The default value for a new installation is **false**. For an upgrade, the value is set to **true**.

Procedure

1. Select **Secure Access Control > Global Settings > Advanced Configuration**.
2. Click the edit icon for poc.otp.backwardCompatibilityEnabled.
3. Change the value from **true**, which enables backward compatibility mode, to **false**.
4. Click **OK**.
5. Deploy the changes when prompted.

Updating template files

If you want to update your template files to the latest version, complete this task. Otherwise, the template files from the previous version are ready for use after the upgrade.

Procedure

1. Download the new template file.
2. Log in to the local management interface.
3. Select **Manage System Settings > File Downloads**.
4. Expand **access_control > pages** and select `mga_template_files.zip`.
5. Click **Export**.
6. Extract the file.
7. Edit and customize the template files, as necessary.
8. Compress the template files into a `.zip` file.
9. In the local management interface, select **Secure Access Control > Global Settings > Template Files**.
10. Select **Manage > Import Zip**.
11. Click **Browse** and locate the `.zip` file you compressed in step 8.
12. Click **Open**.
13. Click **Import**.
14. Deploy the changes.

Updating PreTokenGeneration to limit OAuth tokens

Update the latest version of the PreTokenGeneration mapping rule to limit the number of OAuth tokens per user per definition.

About this task

Follow this procedure so that any API protection definitions you created in a previous version can take advantage of limiting OAuth tokens. If you create new definitions, the latest PreTokenGeneration mapping rule is used.

Procedure

1. Log in to the local management interface.
2. Download the updated mapping rule:
 - a. Click **Manage System Settings > File Downloads**.
 - b. Expand **access_control > examples > mapping_rules**.
 - c. Select `oauth_20_pre_mapping.js`.
 - d. Click **Export**.
3. Optional: Edit and customize the mapping rule if you customized it in the previous version.
4. Replace the PreTokenGeneration mapping rule for the existing API protection definitions.
 - a. Click **Secure Access Control > API Protection**.
 - b. Click **Advanced**.
 - c. Select the appropriate PreTokenGeneration mapping rule.
 - d. Click **Replace**.
 - e. Browse for the updated mapping rule and click **OK**.
5. Deploy the changes.

Reviewing existing Web Reverse Proxy instance point of contact settings

After you upgrade from appliance v8.n.n.n to v9.n.n.n, it might be necessary to review and update some existing Web Reverse Proxy instance point of contact settings for the Advanced Access Control runtime.

Reviewing ACL settings for Authentication Services REST endpoint

A new REST endpoint for the Authentication Services Framework was introduced in v9.0.0.0. The default URL for this endpoint is `"/mga/sps/apiauthsvc"`. After an upgrade from a Security Access Manager appliance at v8.n.n.n, if you want to use the `"/mga/sps/apiauthsvc"` endpoint with an existing web reverse proxy, it might be necessary to create an ACL named `"isam_mobile_rest_unauth"` and attach it to the `"/mga/sps/apiauthsvc"` endpoint. You can use the following Security Access Manager policy administration commands to enable this setting.

```
acl create "isam_mobile_rest_unauth"
acl modify "isam_mobile_rest_unauth" set user "sec_master" TcmdbsvaBRrxl
acl modify "isam_mobile_rest_unauth" set group iv-admin TcmdbsvaBRrxl
acl modify "isam_mobile_rest_unauth" set group webseal-servers Tgmdbsrxl
acl modify "isam_mobile_rest_unauth" set any-other Tmdrxl
acl modify "isam_mobile_rest_unauth" set unauth Tmdrxl

acl attach "/WebSEAL/<web reverse proxy>/mga/sps/apiauthsvc" "isam_mobile_rest_unauth"
```

Reviewing EAI point of contact settings

Some of the default settings that are related to Advanced Access Control point of contact and EAI headers changed in v9.0.0.0. After an upgrade from v8.n.n.n where an existing Web Reverse Proxy instance has been configured with Advanced Access Control, review the following settings and correct the settings if required.

In the Web Reverse Proxy configuration file, check the **[eai]** stanza settings:

```
# EAI HEADER NAMES

# EAI PAC header names
eai-pac-header = am-eai-pac
eai-pac-svc-header = am-eai-pac-svc

# EAI USER ID header names
eai-user-id-header = am-eai-user-id
eai-auth-level-header = am-eai-auth-level
eai-xattrs-header = am-eai-xattrs

# EAI external USER ID header names
eai-ext-user-id-header = am-eai-ext-user-id
eai-ext-user-groups-header = am-eai-ext-user-groups

# EAI COMMON header names
eai-redirect-url-header = am-eai-redirect-url
```

The names of the headers must match the point of contact settings for the Advanced Access Control runtime. You can manage these settings with the local management interface by going to **Secure Access Control > Global Settings > Point of Contact**. Review the parameter value settings for the active point of contact profile.

AAC point of contact parameter	Reverse Proxy header name
fim.user.response.header.name	am-eai-ext-user-id
fim.target.response.header.name	am-eai-redir-url
fim.attributes.response.header.name	am-eai-xattrs
fim.groups.response.header.name	am-eai-ext
fim.user.request.header.name	iv-user
fim.cred.request.header.name	iv-creds
fim.groups.request.header.name	iv-groups
fim.cred.response.header.name	am-eai-pac

Upgrading the signing algorithms of existing policy servers

Internally Security Access Manager uses X509 certificates to handle authentication of the communication channel between the various Security Access Manager servers. You can run certain CLI commands to force the generation of new X509 certificates.

Before you begin

Make sure that your environment meets the following conditions.

- All appliances and any external PD.jar API applications must be updated to version 9.0.2.1 before you use this upgrade feature. Older versions are not supported and they will permanently lose contact with the policy server after it has been updated with a new certificate.
- The time and date on all servers should be synchronized so that the “not before” and “not after” attributes of the new certificates are accurate.

Procedure

1. Use the following CLI commands in order.

- a. `isam -> ca -> new`

This command must be the first command to run. Use this command on the appliance that is running the policy server. It creates a new CA certificate in the policy server key file. The policy server keeps the original CA certificate in its key file and continues to use its original server certificate to accept connections from Security Access Manager servers that have not been updated.

The policy server will be automatically restarted by this command.

- b. `isam -> ca -> update`

This command imports the new CA certificate to all Security Access Manager related key files on the appliance. It also forces all Security Access Manager servers to request a new server certificate from the policy server. If the policy server is on the same appliance, this command also updates the server certificate of the policy server. Each Security Access Manager server will start to use the new server certificate, but will also retain the original CA certificate in its key file so that it can continue to connect to the policy server until the server certificate of the policy server is also updated.

This command must be used after you execute the command in the previous step to generate the new CA certificate and the command must be run on all appliances that are configured to use the policy server. The appliance that is running the policy server must be the last appliance on

which this command is run. This is because when the policy server has a new personal certificate, clients that do not have the new CA certificate will fail to connect to it. The policy server must be running for this command to work.

In a scenario where a policy server is configured on a machine that is not in a cluster and a cluster is using that non-cluster policy server, then the cluster primary node must be the first node among the cluster nodes to run the "update" command.

All Security Access Manager servers on the appliance on which this command is run will be automatically restarted.

c. `isam -> ca -> deprecate`

This command must be used after you execute the command from the previous step to update the Security Access Manager CA certificates of all appliances. This command removes the original CA certificate from all Security Access Manager key files on the appliance. This command is to be used after all appliances have had their Security Access Manager CA certificate updated.

All Security Access Manager servers on the appliance on which this command is run will be automatically restarted.

If a cluster of appliances is configured to replicate the runtime, then after step a, the subsequent step b might have to be delayed until the update on the primary node is replicated to the node that is running the "update" command. A new command `isam -> ca -> replicated` is provided for administrators to detect when the replication has occurred.

2. If a system missed running the "update" operation before the "update" and "deprecate" operation were run on the policy server, use the following commands to allow the system to reconnect with the policy server.

a. `isam -> ca -> add-old`

This command adds the original CA certificate into the policy server key file so that Security Access Manager servers using server certificates that are signed by the original CA certificate can be accepted by the policy server. The policy server will be restarted by this command.

b. `isam -> ca -> remove-old`

This command removes and saves the original CA certificate from the policy server key file. The policy server will be restarted by this command.

Use these commands in the following sequence:

- a. On the policy server appliance, run the command `isam -> ca -> add-old`.
- b. On the appliances that need to be recovered, run the `isam -> ca -> update` and `isam -> ca -> deprecate` commands.
- c. On the policy server appliance, run the command `isam -> ca -> remove-old`.

Example scenarios:

- a. The environment consists of a cluster of two appliances. The policy server is running on the primary master and the runtime data is being replicated to the cluster. An additional appliance, which is not a member of the cluster, also has a Web Reverse Proxy instance configured against the policy server.

Table 2. Scenario 1

Step	Primary (policy server) node	Secondary node	Non-cluster node
1	new		

Table 2. Scenario 1 (continued)

Step	Primary (policy server) node	Secondary node	Non-cluster node
2		update	update
3	update		
4		deprecate	deprecate
5	deprecate		

- b. The environment consists of a cluster of two appliance that are configured to replicate the runtime. The Web Reverse Proxy instances on these two appliances are configured against a policy server that is running on an appliance that is external to the cluster.

Table 3. Scenario 2

Step	Primary node	Secondary node	Non-cluster node (policy server)
1			new
2	update (before other cluster nodes)		
3		update	
4			update
5		deprecate	
6	deprecate (after other cluster nodes)		
7			deprecate

Chapter 2. Getting started with Advanced Access Control

Several configuration tasks must be completed in order to start using Advanced Access Control.

After setup of IBM Security Access Manager Appliance, complete an initial configuration of Advanced Access Control.

Note: If you have not already set up your appliance, see Getting Started.

Complete the following tasks:

1. Activate the product. You must activate both the IBM Security Access Manager Platform offering and the Advanced Access Control Module.
2. Manage application interfaces.
3. Configure the runtime environment.
4. “Managing user registries” on page 242.
5. Configure the services that you require:
 - “Reverse Proxy Configuration with Authentication Services” on page 89
 - “Reverse proxy configuration for OAuth and OIDC provider” on page 125
 - “Configuring Mobile Multi-Factor Authentication” on page 188
6. Review optional runtime security services EAS configuration tasks.

You also might need to complete the following task to enable communication between appliances. See Chapter 7, “Adding runtime listening interfaces,” on page 25.

Chapter 3. Managing application interfaces

To create or edit your management and application interfaces, see [Configuring Interfaces](#).

Chapter 4. Managing the runtime component

To manage configuration files with the local management interface, use the Runtime Component management page.

About this task

When you first install the appliance, you must configure the runtime component. At any time after configuration, you can either edit the configuration settings, or unconfigure the runtime component.

Procedure

1. From the top menu, select **Secure Web Settings > Manage > Runtime Component**.
2. Select one of the following actions.

- **Configure**

- a. On the Main tab, select the **User Registry** type of LDAP.

Note: The runtime component does not communicate with the user registry, but you must select a registry type. It does not matter what user registry your system uses. Selection of the user registry type has no effect on the runtime component. Select LDAP in order to minimize the configuration steps.

- b. On the Policy Server tab, provide settings.
 - **Host name:** The name of the host that hosts the IBM® Security Access Manager policy server.
 - **Port:** The port over which communication with the IBM Security Access Manager policy server takes place.
 - **Management Domain:** The IBM Security Access Manager domain name.
- c. Provide settings on the LDAP tab. Enter a **Host name** and accept the default value for **Port**.

Note: The host name and port values are just placeholders. The runtime component does not use the values, but the configuration wizard requires that values must be set.

- d. Click **Finish** to save the settings.

- **Unconfigure** the remote policy server

- a. Select the **Force** check box if you want the unconfigure operation to forcefully remove all of the configuration data. By default, this check box is not selected.

Note: Select the **Force** check box only if the unconfiguration fails repeatedly. Use this option only as a last resort.

- b. Click **Submit** to confirm operation.

- **Edit**

- a. Select the runtime configuration file of interest.

- b. Edit the configuration file and then click **Save** to save the changes. If you do not want to save the changes, click **Cancel**. If you want to revert to the previous version of the configuration file, click **Revert**.

Note: For the changes to take effect, they must be deployed.

Related information:

Deploy pending changes

Some configuration and administration changes require an extra deployment step.

Chapter 5. Managing user registries

The appliance runtime profile has a user registry associated. Use the User Registry management page to administer the users and group memberships. The user registry in discussion here is the one used by the runtime applications, not the one used by the management interface.

Procedure

1. From the top menu, select the user interface panel for your licensing level.

- **Secure Access Control > Manage > User Registry**
- **Secure Federation > Manage > User Registry**

A list of all the current users in the registry is displayed. You can filter and reorder the list of users.

2. Perform one or more of the following actions as needed:

Create a user in the registry

- a. Click **New**.
- b. In the Create User window, enter the user name and password for the new user.
- c. Click **OK**.

Delete a user from the registry

- a. Select the user to delete.
- b. Click **Delete**.
- c. In the Delete User window, click **Yes** to confirm the delete operation.

Change the password of a user in the registry

- a. Select the user for which you want to change password.
- b. Click **Set Password**.
- c. In the Set Password window, enter the password in the **New Password** and **Confirm Password** fields.
- d. Click **OK**.

Manage group memberships of a user

- a. Select the user of interest. The group memberships that are associated with this user are displayed under the **Group Membership** section.
- b. You can add the user to a group or delete the user from a group in the registry.

Add the user to a group

- 1) In the **Group Membership** section, click **Add**.
- 2) In the Add to Group window, select the group to add this user to.

Note: Only a single group can be selected.

- 3) Click **OK**.

Remove the user from a group

- 1) In the **Group Membership** section, select the group to remove the user from.
- 2) Click **Delete**.
- 3) In the Remove from Group window, click **Yes** to confirm the removal.

Chapter 6. Runtime security services external authorization service

The runtime security services external authorization service (EAS) provides the policy enforcement point function for context-based access.

You can configure the runtime security services EAS to include context-based access decisions as part of the standard authorization on WebSEAL requests. WebSEAL becomes the authorization enforcement point for access to resources that context-based access protects.

The runtime security services EAS constructs a request that it sends to the policy decision point (PDP). Based on the policy decision that is received from the PDP, the EAS takes one of the actions listed in the following table.

Table 4. Runtime security services EAS access decisions.

Action	Description
Permit	Grants access to the protected resource.
Deny	Denies access to the protected resource.
Permit with Authentication	Grants access to the protected resource, after a specific authentication action successfully takes place.
Permit with Obligation	Grants access to the protected resource, after the user successfully authenticates with a secondary challenge.
Deny with Obligation	Denies access to the protected resource, after the user unsuccessfully responds to a secondary challenge.

The following steps set up the initial integration with Advanced Access Control:

1. Configure runtime security services for client certificate authentication.
2. Run the **isamcfg** tool to automatically update the WebSEAL configuration file and to complete other configuration setup.
3. (Optional) Update the WebSEAL configuration file to:
 - Retain the version 7.0 attribute IDs.
 - Define custom attributes for the authorization service.
 - Map an obligation to a URL.
 - Permit access decisions when runtime security services cannot be contacted.

For information about WebSEAL, see Web Reverse Proxy configuration.

Configuring runtime security services for client certificate authentication

Configure runtime security services for client certificate authentication used for authentication between WebSEAL and Advanced Access Control.

About this task

Before selecting the client certificate authentication option provided in the **isamcfg** tool, you must perform the following general steps for the client certificate:

1. Generate a certificate that represents the user who will be authenticating from WebSEAL, or the Web Reverse Proxy, to Advanced Access Control. For example, use **easuser**.
2. Import that certificate into the WebSEAL or Web Reverse Proxy key database as a personal certificate.
3. Import the signer of this certificate as a trusted certificate in the Advanced Access Control keystore.
4. Set **Accept Client Certificates** to True on the appliance.

Procedure

1. Create a client certificate for user **easusercert**.
 - a. In the local management interface, go to **Manage System Settings > Secure Settings > SSL Certificates**.
 - b. Select the **pdsrv** certificate database.
 - c. Click **Manage > Edit SSL Certificate Database**.
 - d. Click **Personal Certificates**.
 - e. Click **New** to create a new personal certificate.
 - f. Provide the following information:
 - Certificate Label: **easusercert**
 - Certificate Distinguished Name: **cn=easuser**
 - Key Size: 2048
 - Expiration Time (in days): 365
 - g. Click **Save**.
2. Deploy pending changes. See Chapter 15, "Deploying pending changes," on page 213.
3. Restart your reverse proxy instances.
4. Export the client certificate.
 - a. Select the **pdsrv** certificate database.
 - b. Click **Manage > Edit SSL Certificate Database**.
 - c. Click **Personal Certificates**.
 - d. Select the **easusercert** certificate you created.
 - e. Click **Manage > Export**.
 - f. Save the file.
5. Import the exported personal certificate as a signer certificate on the appliance. The signer of the client certificate needs to be trusted. The certificate is self-signed. Importing the **easusercert** as a signer certificate into the appliances allows that trust.
 - a. Click **Manage System Settings > Secure Settings > SSL Certificates**.
 - b. Select the **rt_profiles_keys** certificate database.
 - c. Click **Manage > Edit SSL Certificate Database**.
 - d. Click **Signer Certificates**.
 - e. Click **Manage > Import**.
 - f. Click **Browse**.
 - g. Browse to the directory that contains the file to be imported and select the file. Click **Open**.
 - h. Click **Import**. A message that indicates successful import is displayed.

6. Deploy pending changes. See Chapter 15, “Deploying pending changes,” on page 213.
7. Configure the appliance for client certificate authentication.
 - a. In the local management interface, go to **Secure Access Control > Global Settings > Runtime Parameters**.
 - b. Select **Accept Client Certificates**.
 - c. Click **Edit** and set the value as True.
8. Restart the runtime.

What to do next

Run the **isamcfg** tool. Ensure that you respond to the following **isamcfg** prompts appropriately:

- When answering the question Select the method for authentication between WebSEAL and the Advanced Access Control runtime listening interface in the **isamcfg** tool, select Certificate Authentication.
- When prompted to enter the Advanced Access Control runtime listening interface SSL keyfile label, enter the label of the certificate that represents the user who will be authenticating from WebSEAL or Web Reverse Proxy to Advanced Access Control.

For more information, see “isamcfg Security Access Manager appliance configuration worksheet” on page 97.

Permitting access decisions when runtime security services cannot be contacted

Update the WebSEAL configuration file if you want to change the behavior when runtime security services servers cannot be contacted by the EAS.

About this task

By default, if the EAS cannot contact a runtime security services server, the EAS removes the server from the pool of servers. If all of the servers are removed from the pool, WebSEAL returns an error. To prevent the error, you can permit access decisions even if no servers can be contacted. The following instructions show you how to update the WebSEAL configuration file to make this change.

Important: When you perform this task, every single request will be permitted only when none of the runtime security services servers are available. This includes access that might not be permitted if the server was available.

Procedure

1. Open the WebSEAL configuration file.
2. Add the following entry to the [rtss-eas] stanza:
 permit-when-no-rtss-available = true
 The default value for this entry is false.
3. Save the file.
4. Restart the WebSEAL server for the change to take effect.

Results

If none of the servers are available, the user is always be permitted to access a resource. The access is granted even when the runtime security services server would normally deny access if it was available.

Retaining the version 7.0 attribute IDs in existing policies

If your existing policies contain any of the changed attribute IDs, you can update your WebSEAL configuration file to continue using risk-based access version 7.0 IDs.

Before you begin

. Determine whether it is necessary for you to continue using the risk-based access version 7.0 IDs.

About this task

Use the following procedure to continue to use the version 7.0 attribute IDs.

Procedure

1. Open the WebSEAL configuration file, and locate the [rtss-eas] stanza.
2. Add the provide_700_attribute_ids flag and set it to true. This flag enables your existing policies to use the version 7.0 attribute IDs. For example:

```
[rtss-eas]
provide_700_attribute_ids = true
```
3. Save the file.
4. Restart the WebSEAL server for the changes to take effect.

Chapter 7. Adding runtime listening interfaces

Add your interfaces to the list of runtime listening interfaces. This procedure enables communication between an appliance with Advanced Access Control and another Security Access Manager appliance.

Before you begin

When you run the **isamcfg** tool, ensure that you specify the runtime listening interface host name and port when prompted.

Define your interfaces.

About this task

If you want to configure a specific IP other than using the Local Host of the ISAM Appliance, you must configure this as an Interface.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Global Settings**, click **Runtime Parameters**.
4. On the **Runtime Tuning Parameters** page, click **Add** to add a runtime listening interface on port 443:
 - a. For **Interface**, select your interface.
 - b. For **Port**, select or type 443.
 - c. Ensure **SSL** is checked.
 - d. Click **OK**.
5. Click **Add** to add a runtime listening interface on port 80:
 - a. For **Interface**, select your interface.
 - b. For **Port**, select or type 80.
 - c. Clear the **SSL** checkbox.
 - d. Click **OK**.
6. Deploy these changes.

Chapter 8. Support for compliance with NIST SP800-131a

Advanced Access Control supports the requirements that are defined by the National Institute of Standards and Technology (NIST) Special Publications 800-131a.

SP 800-131a strengthens security by defining stronger cryptographic keys and more robust algorithms. The standard defines a period to allow customers time to make the transition to the new requirements. The transition period closes at the end of 2013. See the NIST publication *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths* for the new standards that are defined by Special Publication 800-131, and details about allowed protocols, cipher suites, and key strength.

You can run the appliance that provides Advanced Access Control in either of the two modes that are supported by NIST SP800-131a:

- Transition mode
- Strict mode

When configured in transition mode, server components support the transition mode Transport Layer Security (TLS) protocols, which include TLS 1.0 and TLS 1.1. Client components, such as the HTTPS client that performs one-time password (OTP) delivery and the syslog auditing client, support TLS 1.2 only.

When configured in strict mode, both the server components and the client components of Advanced Access Control support TLS 1.2 only.

To deploy in transition mode, you need to select only the mode during initial configuration of the appliance. To run in strict mode, you must also set an extra configuration option.

If your deployment uses client certificate authentication, and you want to use strict mode, you must complete more configuration steps for the point of contact server. The point of contact server can be either IBM Security Access Manager WebSEAL or IBM Security Web Gateway appliance 7.0.

Transition mode

When you install the appliance, select the option to enable FIPS 140-2 mode. This selection turns on compliance for NIST SP800-131a.

When enabled, NIST SP800-131a compliance is run in transition mode. You do not have to complete any further configuration steps in order to run in transition mode.

Note:

- Enable FIPS 140-2 mode only if you must comply with the NIST SP800-131a requirements. There is no advantage to enabling FIPS 140-2 mode if your installation does not require this compliance.

Important: The setting of the FIPS 140-2 Mode option is permanent and cannot be turned off after it is enabled. To disable the option, you must reinstall the appliance.

- If you enable FIPS 140-2 mode, the appliance is automatically restarted before it continues with the rest of the setup.
- FIPS Limitation: For Advanced Access Control, the FIPS 140-2 mode option in the appliance setup wizard does not turn on compliance for FIPS 140-2. It turns on compliance for NIST SP800-131a only.

Strict mode

Overview of configuration tasks:

1. Enable FIPS 140-2 mode during appliance configuration.
2. Set a tuning parameter to enable strict mode.
3. (Optional) If your deployment uses client certificate authentication, configure TLS v1.2.

Instructions:

1. Install the appliance and choose to enable FIPS 140-2 mode. This selection turns on compliance for NIST SP800-131a.
2. Use the appliance local management interface (LMI) to modify the advanced tuning parameter `nist.sp800-131a.strict`. This parameter is set by default to false. Complete the following steps:
 - a. Verify that your browser supports TLS 1.2.
CAUTION:
Strict mode requires the use of TLS 1.2. Some browsers support TLS 1.2 but have the support disabled by default. If you set the value of the `nist.sp800-131a.strict` parameter to true, and your browser is not configured to support TLS 1.2, you lose access to the appliance LMI.
 - b. On the LMI, select **Manage System Settings > System Settings > Advanced Tuning Parameters**.
 - c. Select `nist.sp800-131a.strict`. Select **Edit**. Change the value to true.
3. Determine whether your deployment uses basic authentication or client certificate authentication, for communication between Advanced Access Control and the point of contact server.
 - If you use basic authentication, the configuration is complete.
 - If you use client certificate authentication, continue with the next section.

Client certificate configuration for strict mode

If you use client certificate authentication on the point of contact server, you must configure it to be in compliance with NIST SP800-131a strict mode.

To comply with strict mode, configure the point of contact server to use TLS v1.2 for client certificate authentication.

You must create a self-signed certificate, and configure the point of contact server to use TLS v1.2 with the runtime security services external authorization service (EAS). Complete each of the following tasks:

1. Create a self-signed certificate.

- Review the “Before you begin” section of “Configuring runtime security services for client certificate authentication” on page 21. Select one of the following actions, as fits your deployment:
 - If your deployment uses Web Reverse Proxy, follow the instructions in “Configuring runtime security services for client certificate authentication” on page 21. In Step 1 “Create a client certificate for user easusercert”, specify:


```
Signature Algorithm: SHA2withRSA
```
 - If your deployment uses WebSEAL:

Manually create a self-signed certificate. To specify a NIST-compliant algorithm, use an external utility such as **gsk7ikm**. Open the pd.srv certificate database, and create a self-signed certificate with these credentials:

```
Certificate Label: easusercert
Certificate Distinguished Name: cn=easuser
Key Size: 2048
Expiration Time (in days): 365
Signature Algorithm: SHA2withRSA
```

Note:

- The user cn=easuser is the built-in user, but any user with sufficient permissions (as created by the Advanced Access Control administrator) can be used instead.
- It is not mandatory that WebSEAL has FIPS 140-2 mode configured in order to communicate with the Advanced Access Control server. However, to comply with NIST SP800-131a strict mode, client certificate authentication between WebSEAL and the server must be over TLS v1.2.
- See the WebSEAL information in the IBM Knowledge Center for complete information on configuring client certificate authentication.

2. Configure the point of contact server to use TLS v1.2 with the Runtime Security Services External Authorization Service (EAS)

The point of contact server uses the EAS to process authorization requests. The default EAS setting for communication specifies Secure Sockets Layer (SSL) v2, which is not supported by Advanced Access Control when it operates in NIST SP800-131a strict mode. If you do not adjust the configuration setting for the EAS, the authorization request (and the regular ping call) does not succeed.

Select the action that fits your deployment:

- If you deploy your point of contact server on the same computer as the appliance:
 - a. In the Advanced Access Control local management interface, select **Reverse Proxy Settings > your_instance_name > Manage > Configuration > Edit** to open the configuration file. Add the following parameter to the existing stanza:


```
[rtss-cluster:cluster1]
gsk-attr-name = enum:438:1
```
 - b. Click **Save**. Deploy the changes. Restart the instance.
- If you deploy your point of contact server on a different computer from the appliance:
 - a. Open the WebSEAL instance configuration file for editing. For example:


```
/opt/pdweb/etc/webseald-appliance-default.conf.
```
 - b. Add the following parameter to the existing stanza:


```
[rtss-cluster:cluster1]
gsk-attr-name = enum:438:1
```

Chapter 9. Authentication

Security Access Manager provides user authentication functions that allow for simple and complex authentication scenarios.

The users who want to access your protected resources can be challenged to provide credentials to authenticate with the various authentication technologies that are supported by Security Access Manager. The component responsible for this capability is called the Authentication Service. The Authentication Service consists of a framework you can use to enforce the execution of various supported authentication mechanisms to authenticate users.

Authentication mechanisms are modules that authenticate the user with a specific challenge or authentication technology, such as user name and password and one-time password. The order on which the authentication mechanisms are run is controlled by an authentication policy. An authentication policy is an XML document that you create with the authentication policy editor. The authentication policy dictates the order of authentication mechanisms to execute.

During an authentication event, the Authentication Service manages the execution of the authentication policy that is required for the event. Each authentication mechanism is included on the authentication policy workflow. This workflow is started by the Authentication Service authentication policy. After the user successfully authenticates to all of the authentication mechanisms that are required by the authentication policy, the Authentication Service generates a user credential. This user credential creates an authenticated session for the user at the point of contact.

The administrator can determine what information is included in the credential by configuring the authentication policy. The authentication policy editor provides a credential editor that an administrator can use to specify the attributes to be included in the resulting credential.

The generated credential contains:

- **authenticationTypes** and **authenticationMechanismTypes** attributes to indicate the authentication policies.
- Authentication mechanisms that are completed during the authenticated session.

The administrator can use **authenticationTypes** and **authenticationMechanismTypes** attributes to author an access control policy to require:

- The user to authenticate through an authentication policy or mechanism once per policy.
- The user to authenticate every time they access the protected resource.

The Authentication Service supports two flows of execution:

- Enforcement of an authentication event as a result of an access control evaluation.
- Request the user to authenticate as a result of either:
 - A point of contact access control list (ACL).
 - A protected object policy (POP) evaluation.

Authentication mechanisms

Security Access Manager provides the following authentication mechanisms:

One-time password authentication mechanisms

A one-time password is a password that is generated for an authentication event and is valid for one use. The one-time password authentication capability in Security Access Manager provides the following features:

- One-time password generation and validation with support for various implementations as provided.
- One-time password delivery with email and short message service (SMS) implementation.
- Time-based, counter-based, and RSA one-time password generation and validation that requires no delivery mechanism.

The one-time password authentication mechanisms are described in the following table.

Mechanism	Description
One-time password authentication	Users provide a one-use password that is generated for an authentication event and is typically communicated between the client and the server through a secure channel. The OTP mechanism groups all the supported one-time passwords methods in a single flow and ask the user to select which one-time password method to use to login. The user can select from the supported one-time password authentication methods: • HOTP • TOTP • RSA OTP • MAC OTP with SMS delivery • MAC OTP with email delivery

Mechanism	Description
MAC One-time password authentication	Users provide a one-use password that is generated for an authentication event and is typically communicated between the client and the server through a secure channel. The MAC mechanism generates one-time passwords by randomly drawing one character at a time from the configured character set until the configured number of characters are drawn. The MAC mechanism also stores the generated one-time passwords in the configured one-time password store plug-in. Each one-time password is salted and hashed before it is stored in the configured one-time password store plug-in. The user can select from the supported MAC one-time password authentication methods: • MAC OTP with email delivery. The email delivery sends the email address of the user and the one-time password in a message, whose MIME type is text/plain, to the configured SMTP Server. The SMTP Server then sends the one-time password to the user by email. The product does not provide an SMTP Server. You must configure your own SMTP Server. • MAC OTP with SMS delivery. The SMS Delivery first sends the phone number of the user and the one-time password in an HTTP POST request, whose content type is application/x-www-form-urlencoded, to the configured SMS Gateway. The SMS Gateway then sends the one-time password to the user through SMS. The product does not provide an SMS Gateway. You must configure your own SMS Gateway. This mechanism also supports the one-time password mapping rules.
HOTP One-time password authentication	Users provide a one-use password that is generated for an authentication event. The one-time password is generated by the HOTP method.
TOTP One-time password authentication	Users provide a one-use password that is generated for an authentication event. The TOTP mechanism generates one-time passwords by using a specified algorithm with a time-based one-time password application. Passwords are not communicated or stored, but are verified as a match between server and client as they are regenerated at regular intervals.

Mechanism	Description
RSA One-time password authentication	Users provide a one-use password that is generated for an authentication event. The RSA mechanism works with an RSA SecurID Authentication Manager and passcode generator. You must own the RSA Authentication Manager product to use RSA as a mechanism. The RSA Authentication Manager and passcode generator generates a passcode every 30 - 60 seconds. The user name and passcode are supplied by the user and passed to the RSA Authentication Manager. The RSA Authentication Manager makes a decision and returns it to Security Access Manager, which relays the decision back to the user.

Username and Password mechanism

Users provide a user name and password.

HTTP Redirect mechanism

Use this mechanism to integrate a custom authentication mechanism into the workflow of an authentication policy. Users provide credentials that are required by the custom authentication mechanism.

Consent to device registration mechanism

Users provide consent to allow their device to be registered.

FIDO Universal 2nd Factor mechanism

Users authenticate through the use of registered FIDO Universal 2nd Factor tokens.

FIDO2/WebAuthn mechanism

Users authenticate through the use of registered FIDO2/WebAuthn authenticators.

Authentication policies

By grouping the provided authentication mechanisms into the workflow of an authentication policy, you can achieve several types of authentication:

- **Simple authentication**
Users provide basic identifying information such as a user name and password.
- **Step-up authentication**
Users provide a specific type of credential usually to access sensitive resources. The users might be challenged to authenticate and provide an extra set of credentials to prove that they are allowed to access sensitive resources.
- **Multi-factor authentication**
Users provide more than one type of credential to access a protected resource.

Each authentication policy has a unique identifier that you can use with an access policy or to start the authentication service directly without any prior access policy invocation.

Web-based authentication

For web-based authentication (using HTML template files and responses)

```
/mga/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:<Unique PolicyId>
/mga/sps/authsvc/policy/<Unique PolicyId>
```

For example, the TOTP can be started like this:

```
/mga/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:totp  
/mga/sps/authsvc/policy/totp
```

When the policy runs, it cycles based on a rolling StateId parameter, on either the policy URL or the query string of the authsvc URL, depending on how it is started.

For example, /mga/sps/authsvc/policy/totp?StateId=cf845efa-52d0-4296-a524-9028acba4108 or /mga/sps/authsvc?StateId=cf845efa-52d0-4296-a524-9028acba4108.

REST API-based authentication

For REST based authentication (Using JSON Template files and responses). This is suitable for use from Single Page Applications making ajax requests or other REST clients. For more information, see Authentication Service Framework for REST API clients.

```
/mga/sps/apiauthsvc?PolicyId=urn:ibm:security:authentication:asf:<Unique PolicyId>  
/mga/sps/apiauthsvc/policy/<Unique PolicyId>
```

For example, the TOTP policy can be started like this:

```
/mga/sps/apiauthsvc?PolicyId=urn:ibm:security:authentication:asf:totp  
/mga/sps/apiauthsvc/policy/totp
```

When the policy runs, it cycles based on a rolling StateId parameter, on either the policy URL or the query string of the authsvc URL, depending on how it is started.

For example, /mga/sps/apiauthsvc/policy/totp?StateId=cf845efa-52d0-4296-a524-9028acba4108 or /mga/sps/apiauthsvc?StateId=cf845efa-52d0-4296-a524-9028acba4108.

Note: Query string invocation can be disabled through the advanced configuration parameter: `sps.authService.allowKickoffViaQueryStringPolicyId`.

This allows administrators to set or use ACLs or POPs and CBA policy to prevent access to certain Authentication policies where necessary.

Authentication Service configuration overview

Most of the configuration that is associated with the authentication service and the supported authentication mechanisms is pre-configured on the appliance. In most scenarios, this configuration is adequate. However, some scenarios require customization to meet your requirements.

You can configure the following components to customize the authentication support:

- Configure the point of contact settings.
- Customize the authentication mechanism settings.
- Modify the template pages to customize how you interact with your users.

Point of contact settings

You can configure the point of contact in the **Advanced Configuration** settings of the local management interface. For more information, see the configuration settings that begin with `poc.` in Managing Advanced Configuration.

This version of the Security Access Manager simplified the configuration that is required for the authentication service. Previous versions relied on a list of preconfigured authentication callbacks to determine the authentication flow. The addition of the new authentication policy format eliminated the need to rely on the authentication level value to determine the order of execution of the authentication mechanisms. The execution of an authentication event now depends on the content of the authentication policy. You can configure the Authentication Service to allow reauthentication. If enabled, the Authentication Service runs all the authentication mechanisms included on the authentication policy regardless of a pre-existing authentication session.

Access policy scenario configuration

This scenario is almost fully configured when you complete deployment and run activation and **isamcfg**. To enable this scenario:

1. Create an access policy that references any of the authentication policies that are provided.
2. Attach the access policy to the resource that you want to protect.

No further configuration is needed.

Web Gateway Appliance step-up authentication scenario

This scenario requires a set of manual steps to enable it when you complete deployment and run activation and **isamcfg**. This scenario relies on an ACL or POP on the point of contact configuration to initiate the policy execution. The user must complete an authentication policy flow when the policy requires that the user step up to a higher authentication. This setup is specific to and dependent on the point of contact technology you are using in your environment. To configure the Web Gateway Appliance to enable this scenario, see “Configuring step-up authentication” on page 37.

Web Gateway Appliance authentication scenario

This scenario requires a set of manual steps to enable it when you complete deployment and run activation and **isamcfg**. This scenario relies on an ACL or POP on the point of contact configuration to initiate the policy execution. The user must complete an authentication policy flow when the policy requires that the user authenticate. This setup is specific to and dependent on the point of contact technology you are using in your environment. To configure the Web Gateway Appliance to enable this scenario, see “Configuring authentication” on page 39.

Authentication mechanism settings

You can modify authentication mechanism settings through the local management interface. See the configuring topics for the authentication mechanisms you want to use:

- “Configuring a TOTP one-time password mechanism” on page 42
- “Configuring an HOTP one-time password mechanism” on page 40
- “Configuring an RSA one-time password mechanism” on page 46
- “Configuring a MAC one-time password mechanism” on page 45
- “Configuring consent to device registration” on page 57
- “Configuring an HTTP redirect authentication mechanism” on page 56
- “Configuring username and password authentication” on page 54

- “Configuring an End-User License Agreement authentication mechanism” on page 59
- “Configuring a Knowledge Questions authentication mechanism” on page 68

For advanced customization of the authentication service or the one-time password generation, delivery, and verification, you can customize the mapping rules. See “Managing mapping rules” on page 75.

Template configuration

Many HTML pages and XML documents are provided to interact with your users. The pages prompt users for authentication information, provide them with one-time passwords, or notify them of errors during authentication. For information about customizing the template pages, see Managing template files.

Authentication configuration scenarios

Use the configuration scenarios to create a custom configuration for your environment.

Configuring step-up authentication

The appliance reverse proxy server can be configured to use the authentication service for step-up authentication. The user is required to complete an authentication policy flow when the policy (ACL or POP) dictates that the user steps up to a higher authentication level.

About this task

This task applies to both the Web Gateway appliance and the Security Access Manager appliance.

Procedure

1. Configure the appliance with the **isamcfg** tool. See “Using the isamcfg tool” on page 90.
2. Modify the appliance `stepuplogin.html` file so that it redirects the authentication request to the Security Access Manager Authentication Service.
 - a. Locate the `stepuplogin.html` file. For information about working with reverse proxy pages, see HTML server response page modification.
 - b. Edit the file to insert the following code in the JavaScript section of the file. Optionally, to indicate where to send the user agent after successful authentication, pass the **Target** query string parameters, which is the default. For example:

```
var authnlevel="%AUTHNLEVEL%";
if (authnlevel == "2"){
  window.location = "https://<HOST>:<PORT>/<JUNCTION>
    /sps/authsvc?Target=%HTTPS_BASE%URL_ENCODED%&PolicyId=<POLICY_ID>";
}
```

Where:

HOST

The host name for the reverse proxy instance.

PORT

The port number for the reverse proxy instance.

JUNCTION

The Advanced Access Control junction name. For example: mga.

POLICY_ID

The authentication policy identifier to execute when the user is requested to step up.

The following example uses one-time password as the step-up mechanism:

```
var authnlevel="%AUTHNLEVEL%";
if (authnlevel == "2"){
    window.location = "https://example.com/mga/
    sps/authsvc?Target=%HTTPS_BASE%URL_ENCODED%&PolicyId=urn:ibm:security:
    authentication:asf:otp";
}
```

3. Restart the appliance.
4. Verify the configuration:
 - a. Create a test user account. For example:

```
pdadmin> user create john cn=john, dc=iswga John Doe password
```
 - b. Activate the account. For example:

```
pdadmin> user modify john account-valid yes
```
 - c. Create a test resource that is protected with level 2 authentication and place it in the document root of the appliance reverse proxy server. For example:

```
/junction-root/test.html
```

For information about working with reverse proxy pages, see HTML server response page modification

- d. Try accessing that resource through the appliance reverse proxy server. For example:

```
https://mga.example.com/test.html
```

A web form is displayed and prompts you to enter the user name and password.

- e. Enter the credential that you created in step 4a. The contents of the resource is displayed.
- f. Create a Protected Object Policy (POP) with a level 2 authentication. For example:

```
pdadmin> pop create level2only
pdadmin> pop modify level2only set ipauth anyothernw 2
```
- g. Attach the POP to the protected resource that you created in step 4c. For example:

```
pdadmin> pop attach /WebSEAL/mga.example.com-default/
test.html level2only
```
- h. Open a new browser session and try accessing the test resource again. A web form is displayed and prompts you to enter the user name and password.
- i. Enter the credential for the test user. You are forwarded to the extended authentication endpoint. You are now starting the authentication policy.
- j. Enter the required credentials to complete the authentication policy. If you authenticate successfully, you are redirected to back to the test resource and you can access the contents of the resource.

Configuring authentication

The appliance reverse proxy server can be configured to use the authentication service for authentication. The user will be required to complete an authentication policy flow when the Security Access Manager policy (ACL or POP) dictates that the user authenticates.

About this task

This task applies to the Web Gateway, Security Access Manager for Web, and Security Access Manager appliances.

Procedure

1. Configure the appliance using the **isamcfg** tool. See “Using the isamcfg tool” on page 90.
2. Modify the appliance `login.html` so that it redirects the authentication request to the Security Access Manager Authentication Service.
 - a. Locate the `login.html` file on the appliance. For information about working with reverse proxy pages, see HTML server response page modification.
 - b. Open the file in a text editor and insert a meta-tag refresh tag to send the request to the authentication service. Optionally, to indicate where to send the user agent after successful authentication, pass the **Target** query string parameters, which is the default. For example:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta http-equiv="refresh" content="2;url=https://<HOST>:<PORT>/<JUNCTION>/
    sps/authsvc?Target=%HTTPS_BASE%URL_ENCODED&PolicyId=<POLICY_ID>">
<TITLE>Access Manager for Web Login</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
</BODY>
</HTML>
```

Where:

HOST

The host name for the reverse proxy instance.

PORT

The port number for the reverse proxy instance.

JUNCTION

The Advanced Access Control junction name. For example: `mga`.

POLICY_ID

The authentication policy identifier to execute when the user is requested to step up.

The following example uses user name and password as the login mechanism:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta http-equiv="refresh" content="2;url=https://example.com/mga/
    sps/authsvc?Target=%HTTPS_BASE%URL_ENCODED&PolicyId=urn:ibm:security:authentication:asf:pa
<TITLE>Access Manager for Web Login</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#000000">
</BODY>
</HTML>
```

3. Restart the appliance.
4. Verify the configuration:
 - a. Create a test user account. For example:

```
pdadmin> user create john cn=john,dc=iswga John Doe password
```
 - b. Activate the account. For example:

```
pdadmin> user modify john account-valid yes
```
 - c. Create a test resource that is protected with the `isam_mobile_anyauth` ACL and place it in the document root of WebSEAL. For example:

```
junction-root/test.html
```

For information about working with reverse proxy pages, see [HTML server response page modification](#)
 - d. Attach the `isam_mobile_anyauth` ACL to the protected resource. For example:

```
pdadmin> acl attach /WebSEAL/mga.example.com-default/test.html  
isam_mobile_anyauth
```
 - e. Open a new browser session and try accessing the test resource. You are forwarded to the authentication service endpoint. You are now starting the authentication policy.
 - f. Enter the required credentials to complete the authentication policy. If you authenticate successfully, you are redirected to the test resource and you can access the contents of that resource.

Configuring an HOTP one-time password mechanism

The HOTP one-password mechanism relies on a public algorithm to generate the one-time password.

About this task

The HOTP client solution and Security Access Manager use the same algorithm to generate the one-time password value. No interaction is required between the client software and the Security Access Manager solution. The algorithm uses a shared secret key and a counter to generate the one-time password value. Every time a new one-time password is generated, the counter value increments on both server and client solutions. No delivery of the one-time password is required.

This task describes the steps and properties for configuring a HOTP mechanism. For information about configuring other providers, see:

- “Configuring a MAC one-time password mechanism” on page 45
- “Configuring a TOTP one-time password mechanism” on page 42
- “Configuring an RSA one-time password mechanism” on page 46

Note: When users attempt to log in using HOTP or TOTP and submit an incorrect one-time password, they receive one strike against their account. This strike remains on their account for a configurable duration. By default, the duration is 10 minutes. After that duration, the strike is removed from their account. When users submit multiple incorrect one-time passwords, they can reach a maximum and are then prevented from making another attempt until one of their strikes expires. By default, the maximum is 5. If the users log in successfully, any strikes on their account are cleared. Strikes are shared between TOTP and HOTP. For example, if the users made two incorrect attempts using TOTP, those strikes count against them on HOTP as well. Because user retries affect only TOTP and HOTP logins,

users who exceeded password attempt using those logins can still use other OTP provider logins or basic username/password authentication. You can modify the password retry settings through the Advanced Configuration settings in the local management interface. For more information, see “Managing advanced configuration” on page 218.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **HOTP One-time Password**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

HOTP

Max Counter Lookahead

The number of times to increment the counter to see whether the one-time password is valid before stopping. Any non-negative number is valid.

The default is 25.

Password Length

The length of the generated one-time passwords, which can be 6 - 9 characters or numbers.

The default is 6.

Generation Algorithm

The algorithm that is used to generate the one-time password. Valid options include the following algorithms:

- HmacSHA1
- HmacSHA256
- HmacSHA512

The default is HmacSHA1.

Secret key URL

The URL that is used to deliver the secret key. The QR code is also generated using this URL. The URL format might include information specific to your environment, such as your company name.

The default URL is:

```
otpauth://hotp/Example:@USER_NAME?secret=@SECRET_KEY
&issuer=Example&counter=0
```

The URL supports the following macros and may be positioned wherever their corresponding values belong.

@SECRET_KEY@

The secret key.

@USER_NAME@

The user name of the authorized user who logs in.

@ALGORITHM@

The one-time password generation algorithm.

@DIGITS@

The one-time password length.

A secret key URL example to utilize all macros is:

```
otpauth://hotp/Example:@USER_NAME@?secret=@SECRET_KEY@&issuer=Example
&counter=0&algorithm=@ALGORITHM@&digits=@DIGITS@
```

Secret key attribute name

The attribute name that is used for storage of the HOTP secret key in the database.

Data type: String

Example: otp.hmac.hotp.secret.key

Secret key attribute namespace

The attribute namespace of the HOTP secret key. The namespace in combination with the attribute name constitutes the unique identifier for the attribute in the database.

Data type: String

Example: urn:ibm:security:otp:hmac

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring a TOTP one-time password mechanism

The TOTP one-password mechanism relies on a public algorithm to generate the one-time password.

About this task

The TOTP client solution and the Security Access Manager use the same algorithm to generate the one-time password value. No interaction is required between the client software and the Security Access Manager solution. The algorithm uses a shared secret key and the time to generate the one-time password value. No delivery of the one-time password is required.

This task describes the steps and properties for configuring a TOTP mechanism. For information about configuring other one-time password providers, see:

- “Configuring a MAC one-time password mechanism” on page 45
- “Configuring an HOTP one-time password mechanism” on page 40

- “Configuring an RSA one-time password mechanism” on page 46

Note: When users attempt to log in using HOTP or TOTP and submit an incorrect one-time password, they receive one strike against their account. This strike remains on their account for a configurable duration. By default, the duration is 10 minutes. After that duration, the strike is removed from their account. When users submit multiple incorrect one-time passwords, they can reach a maximum and are then prevented from making another attempt until one of their strikes expires. By default, the maximum is 5. If the users log in successfully, any strikes on their account are cleared. Strikes are shared between TOTP and HOTP. For example, if the users made two incorrect attempts using TOTP, those strikes count against them on HOTP as well. Because user retries affect only TOTP and HOTP logins, users who exceeded password attempt using those logins can still use other OTP provider logins or basic username/password authentication. You can modify the password retry settings through the Advanced Configuration settings in the local management interface. For more information, see “Managing advanced configuration” on page 218.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **TOTP One-time Password**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

TOTP

Generation Interval (seconds)

The number of seconds an interval lasts. This number determines how long a one-time password is active before the next one-time password generates.

The default is 30.

Password Length

The length of the generated one-time passwords, which can be 6 - 9 characters or numbers.

The default is 6.

Skew Intervals

The skew intervals of the algorithm. The skew intervals consider any possible synchronization delay between the server and the client that generates the one-time password. For example, a skew interval of 2 means a one-time password in up to two intervals in the past, or two in the future are valid. For example, if it is interval 563, and intervals are 30 seconds, then

one-time passwords for intervals 561-565 are computed and checked against within a range of 2.5 minutes.

The default is 1.

One Time Use

Whether to cache one-time passwords if they are used to successfully log in. If set to true, then the reuse of a one-time password is prevented while it is in cache.

The default is true.

Generation Algorithm

The algorithm that is used to generate the one-time password. Valid options include the following algorithms:

- HmacSHA1
- HmacSHA256
- HmacSHA512

The default is HmacSHA1.

Secret key URL

The URL that is used to deliver the secret key. The QR code is also generated using this URL. The URL format might include information specific to your environment, such as your company name.

The default URL is:

otpauth://totp/Example:@USER_NAME@?secret=@SECRET_KEY@&issuer=Example

The URL supports the following macros and may be positioned wherever their corresponding values belong.

@SECRET_KEY@

The secret key.

@USER_NAME@

The user name of the authorized user who logs in.

@ALGORITHM@

The one-time password generation algorithm.

@DIGITS@

The one-time password length.

@PERIOD@

The one-time password generation interval.

A secret key URL example to utilize all macros is:

otpauth://totp/Example:@USER_NAME@?secret=@SECRET_KEY@&issuer=Example
&algorithm=@ALGORITHM@&digits=@DIGITS@&period=@PERIOD@

Secret key attribute name

The attribute name that is used for storage of the TOTP secret key in the database.

Data type: String

Example: otp.hmac.totp.secret.key

Secret key attribute namespace

The attribute namespace of the TOTP secret key. The

namespace in combination with the attribute name constitutes the unique identifier for the attribute in the database.

Data type: String

Example: urn:ibm:security:otp:hmac

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring a MAC one-time password mechanism

A one-time password is valid for one session or login. The MAC password is generated by Security Access Manager and can be delivered to the user through Short Message Service (SMS) or e-mail.

About this task

This task describes the steps and properties for configuring a MAC mechanism. For information about configuring other providers, see:

- “Configuring an HOTP one-time password mechanism” on page 40
- “Configuring a TOTP one-time password mechanism” on page 42
- “Configuring an RSA one-time password mechanism” on page 46

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **MAC One-time Password**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

MAC

Password Character Set

The character set from which the characters in the one-time password are generated.

The default is 0123456789.

Password Length

The length of the characters in the one-time password.

The default is 6.

Store Entry Hash Algorithm

The hash algorithm that is used for hashing the one-time password before it is stored in the one-time password store plug-in. The supported algorithms are:

- SHA1
- SHA-256
- SHA-512

The default is SHA-256.

Store Entry Lifetime (seconds)

The length of time that the one-time password is stored. The lifetime is in seconds.

The default is 300.

9. Click **Save**.

What to do next

When you configure one-time password providers, a message indicates that changes have not been deployed. If you have finished making changes, deploy them. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Next, consider configuring the delivery methods for the one-time password. Both SMS and Email delivery are enabled but you will want to configure the delivery properties, such as SMTP server or connection URL, for your environment. See “Configuring one-time password delivery methods” on page 50.

Configuring an RSA one-time password mechanism

A one-time password is valid for one session or login. To use RSA as a mechanism, you must own RSA Authentication Manager. The server and the client generate the passwords with the same algorithm.

Before you begin

Complete the following steps.

1. On your RSA server, generate the following files:

sdconf.rec

The configuration file for connecting to the RSA Authentication server.

sdopts.rec

The configuration properties file that contains optional configurations for load balancing.

2. See your RSA Authentication server documentation for details on creating these files and use the following guidelines:

- On the appliance, you must specify an Agent Network Interface. See Agent Network Interface in step 8 on page 47. If you connect the RSA server to the appliance by using an application network interface with multiple IP addresses, list all the IP addresses in the **Alternate IPs** box on the RSA server.
- For **Agent type**, choose **Standard**.

- **Agent Auto-Registration** must be enabled when the first RSA one-time password authentication is performed. You can disable it after the first successful authentication is completed.

Note: The RSA one-time password mechanism does not support replication of the RSA session information through the cluster environment. The session information is local to each cluster node and the environment must be configured to enforce session affinity between the client and the cluster node.

3. Move or copy the generated files from the RSA server to the appliance.

About this task

This task describes the steps and properties for configuring an RSA mechanism. For information about configuring other providers, see:

- “Configuring an HOTP one-time password mechanism” on page 40
- “Configuring a MAC one-time password mechanism” on page 45
- “Configuring a TOTP one-time password mechanism” on page 42

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **RSA One-time Password**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Agent Network Interface

The name of the network interface that the RSA Agent is using to connect to the RSA server.

Required: Yes

Data type: String

Valid interface values:

- 1.1
- 1.2
- 1.3
- 1.4

Note: If you are using the RSA mechanism in a cluster environment and use an application interface with multiple IP addresses defined for that interface, use the RSA console to add all of those IP addresses to the whitelist. See the RSA documentation for information about adding IP addresses to the whitelist.

Example: 1.1

Server Exchange Initial Timeout

The initial timeout coefficient in milliseconds used to calculate the timeout of the request.

Required: No

Data type: Integer

Example: 1000

Server Exchange Timeout Offset

The offset timeout coefficient in milliseconds used to calculate the timeout of the request.

Required: No

Data type: Integer

Example: 200

Server Exchange Timeout Increment

The increment coefficient in milliseconds used to calculate the timeout of the request.

Required: No

Data type: Integer

Example: 100

Event Log Level

The minimum event level to be logged. Events below the level that is specified in this property are not logged.

The events in order from lowest level to highest are:

- a. OFF
- b. DEBUG
- c. INFO
- d. WARN
- e. ERROR
- f. FATAL

Required:

Data type: String

Example: INFO. If this property is set to INFO, the DEBUG errors are not logged.

Enable Debug Tracing

The property that enables debug tracing.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, debug tracing is not enabled.

Trace Function Entries

The property that enables tracing of function entries.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, function entries are not traced.

Trace Function Exits

The property that enables tracing of exits.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, exits are not traced.

Trace Flow Statements

The property that enables tracing of flow statements.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, flow statements are not traced.

Trace Regular Statements

The property that enables tracing of regular statements.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, regular statements are not traced.

Trace Location

The property that enables the class name and line number to be displayed in the trace.

Required: No

Data type: Boolean

Example: FALSE. If set to FALSE, class name and line number are not displayed.

Session Timeout

The length of time, in seconds, that a connection to the RSA Authentication Manager server remains open before it times out when a user attempts to authenticate.

Required: No

Data type: Integer

Example: 1800

9. Click the **Agent Files** tab.
10. Select a file in the table that corresponds to the file you generated on the RSA server.
11. Click **Upload** to upload the file or **Clear** to remove the contents of the selected file. The status area indicates one of three statuses:

Not uploaded

Upload is not completed.

Last upload date

Upload was completed on date indicated.

Auto-generated

The SecurID was automatically generated instead of uploaded.

Repeat this step until all of your files have been uploaded to the appliance.

12. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring one-time password delivery methods

Passwords can be delivered to the user through Short Message Service (SMS) or email.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click the delivery type.
 - **SMS One-time Password**
 - **Email One-time Password**
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the delivery method.

SMS

Basic Authentication User Name

The user name that is used in HTTP Basic authentication.

SMS Delivery does not perform the HTTP basic authentication if this configuration is not specified.

Required: False

Multi-value: No

Example: username

Basic Authentication Password

The password that is used in HTTP basic authentication.

SMS Delivery does not perform HTTP Basic authentication if this configuration is not specified.

Required: False

Multi-value: No

Example: password

Connection URL

The URL of the SMS Gateway where the phone number of the user and the one-time password is sent.

Required: True

Multi-value: No

Example: `https://msgateway.tfim.example.com/`

HTTP Request Parameters

The list of name and value pairs that is included in the body of the **HTTP POST** request to the SMS Gateway. In each pair, the name and the value must be separated by equal sign.

Two macros, **\$DEST_NO\$** and **\$MSG\$**, are replaced by the phone number of the user and the content of the SMS. These two macros can be used only as value in the name and value pair.

Required: True

Multi-value: Yes

Example:

- `From=+0123456789`
- `To= $DEST_NO$`
- `Body= $MSG$`

Success HTTP Response Body Regex Pattern

This parameter defines the Java™ regular-expression pattern that matches the HTTP response body that is returned by the SMS Gateway. When the match is successful, the SMS delivery is successful.

The default value is empty.

The default behavior is that the HTTP response body is not going to be matched against any Java regular-expression and the success or failure decision is going to be based on the `SuccessHTTP` `ReturnCode` value only.

Note: If the HTTP response from the SMS Gateway does not contain a body, this matching is not performed.

Required: False

Multi-value: No

Example:

- When the body of all responses by the SMS Gateway contains either `Success` or `Failure` followed by no newline character, the sample `SuccessHTTP` `Response` `BodyRegex` `Pattern` value is `Success`
- When the body of all responses by the SMS Gateway contains the following text:
`MGDID=TTTT`
`TTTTTTTTT`
`RESPONSE`

```
CODE=NNN
SMS=TTTTTTT
TTTTTTT
TTTTTTT
DATE=NNNNNNNN
```

where each line ends with the \n character without any preceding \r character, and the RESPONSECODE is defined such that a three-digit number from 0 to 199 indicates success, the sample SuccessHTTP

ResponseBody
RegexPattern value is

```
(?s).*
RESPONSE
CODE=(\d{1,2}
|[0-1]{1}
\d{2})\n.*
```

Success HTTP Return Code

The response code from the SMS Gateway that is an acknowledgment from the SMS Gateway that the request is successfully processed.

The default SuccessHTTP ReturnCode, which is 200, is used when this configuration is not specified.

Note: The SuccessHTTP ReturnCode match must be successful before this matching is done.

Required: False

Multi-value: No

Example: 200

HTTPS Trust Store

The keystore that validates the SMS Gateway SSL certificate.

This configuration must be specified only when SMS Delivery communicates with the SMS Gateway by using HTTPS.

Required: False

Multi-value: No

Example: rt_profile_keys

Client Authentication Key

The certificate that is used as client certificate in SSL Client authentication.

SMS Delivery does not perform SSL Client authentication if this configuration is not specified.

Required: False

Multi-value: No

Example: rt_profile_keys

Email

Sender Email

The email address that is used as the sender of the email that is sent to the user.

Required: True

Multi-value: No

Example: otp_emailer@example.com

SMTP Host Name

The host name of the SMTP Server.

Required: True

Multi-value: No

Example: smtpserver.tfim.example.com

SMTP User Name

The user name that is used in SMTP authentication.

Required: False

Multi-value: No

Example: username

SMTP Password

The password that is used in SMTP authentication.

Required: False

Multi-value: No

Example: password

Use SSL

Use SSL for the connection to the SMTP server.

Required: True

Multi-value: No

Example: false

Enable STARTTLS

Defines whether STARTTLS will be used to negotiate TLS to the SMTP server.

Required: True

Default: false

Type: Boolean

TLS protocol

TLS protocol to be used when connecting to the SMTP server.

Required: True

Default: TLS

Type: String

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring username and password authentication

The user name and password authentication mechanism authenticates users with their user name and password credentials that are stored in the Access Manager user repository.

Before you begin

This authentication mechanism uses the user registry that is configured as part of the runtime component settings. Ensure that you configured this registry before you use the mechanism. See Chapter 4, “Managing the runtime component,” on page 17.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **Username Password**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

LDAP Bind DN

An LDAP account with sufficient rights to update the user registry entries. For example: `cn=SecurityMaster,secAuthority=Default`

One method for creating such an account is using the **pdadmin** command. For example:

```
user create no-password-policy testapi cn=testapi,secAuthority=Default  
testapi api passwd (SecurityGroup ivacld-servers remote-acl-users)
```

Data type: String

LDAP Bind Password

The LDAP bind password.

Data type: String

LDAP Host Name

The host name of the LDAP server.

Data type: String

LDAP Port

The port number of the LDAP server.

Data type: String

Default: 389

Management Domain

The Security Access Manager Management Domain name. This name is used to determine the location of subdomain in the registry. Subdomains are located relative to the Management Domain LDAP location.

Data type: String

Default: Default.

SSL Enabled

Set this option to true to enable SSL to the LDAP server.

Data type: Boolean

Default: False.

SSL Trust Store

The keystore that contains the trusted CA signers for the LDAP server certificate.

Specify an SSL trust store if you use one of the following LDAP registry scenarios for user name and password authentication:

- You configure one primary LDAP registry which uses SSL.
- You configure federated directories, where at least one of the directories uses SSL. In this scenario, the **Use Federated Directories Configuration** property must be set to true.

The trust store you specify must be configured to work with any and all of the LDAP registries that use SSL.

Data type: String

Use Federated Directories Configuration

Set this option to true to use the configured federated directories when authenticating a user name and password.

If you specify true:

- The **LDAP Host Name** and **LDAP Port** properties must define a Security Access Manager user registry. This is typically the user registry of the runtime component.
- The users in any of the additional federated directories you configure must exist in the user registry of the runtime component. Therefore, import these users, if necessary.

Data type: Boolean

Default: false.

User Search Filter

An LDAP search filter that selects any native user entry.

Data type: String

Default: `(|(objectclass=ePerson)(objectclass=Person))`.

Maximum Server Connections

The maximum number of connections that can exist on the LDAP server. Valid values are 2 through 4096.

Data type: Integer

Default: 16.

Login Failures Persistent

Login failures are used with the three-strikes policy. If you set this option to false, each process that uses this API stores the number of login failures in memory. If you use multiple appliances in a cluster, the total number of login failures to trigger a strike-out might vary.

If you set this option to true, the strike count is stored in LDAP and shared across all servers. An accurate count can be kept in a multiserver environment.

Data type: Boolean

Default: False.

9. Click the **Attributes** tab.

10. Complete any of the following tasks.



Add an attribute. Complete the **Registry Attribute**, **Context Name**, **Context Namespace** fields for the attribute.



Modify an attribute. Modify the **Registry Attribute**, **Context Name**, **Context Namespace** fields for the attribute.



Delete an attribute. Select an attribute and click delete.

By default, this mechanism uses the following attributes. These registry attributes are retrieved from the user account in the user registry and are stored in the Session context with the context name and name space.

Registry Attribute	Context Name	Context Namespace
mail	emailAddress	urn:ibm:security:authentication: :asf:mechanism:password
mobile	mobileNumber	urn:ibm:security:authentication: asf:mechanism:password

11. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring an HTTP redirect authentication mechanism

The HTTP redirect authentication mechanism integrates an external application that you provide to use for authenticating users.

Before you begin

The external application that you plan to use must exist before you begin this task. The application must meet the following requirements:

- To indicate a successful authentication, the application must generate a credential with an attribute that matches the **Success Credential Attribute Name** and **Success Credential Attribute Value** properties.
- When the authentication is complete, the application must return control to Security Access Manager by redirecting the browser to the location provided on the ReturnURL query string parameter when the application was invoked.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **HTTP Redirect Authentication**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Redirect URL

The URL to the external authentication application.

Data type: String

Success Credential Attribute Name

The credential attribute name that verifies successful authentication.

Data type: String

Default: httpRedirectAuthCompleted.

Success Credential Attribute Value

The credential attribute value that verifies successful authentication.

Data type: String

Default: true.

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. When you finish the changes, deploy them. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Configuring consent to device registration

Consent-based device registration is the process of registering the device fingerprint only after the user consents to the device registration.

About this task

The settings of the consent to device registration mechanism specify:

- Whether to set the authentication level of the user's credential when the consent to device registration is completed.
- The value to be used to set the authentication level. The authentication level on the credential is used to represent the strength of the authentication that is used to generate the credential.

By default, the authentication level is not set by the consent to device registration operation. Use this task to enable setting the authentication level on the user credential. When the authentication level is set, it can be evaluated as part of an access control policy or by the policy enforcement point to grant access to a resource that requires a specific authentication level .

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **Consent to device registration**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Set Authentication Level Credential Attribute

Enables the consent to device registration authentication to set the authentication level on the session.

Data type: Boolean

Default: False.

Authentication Level Credential Attribute Value

The authentication level value to be used when the consent to device registration is configured to set the authentication level.

Data type: Integer.

Default: 2.

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy changes when you are finished. For more information, see Chapter 15, "Deploying pending changes," on page 213.

Configuring an End-User License Agreement authentication mechanism

The End-User License Agreement authentication mechanism prompts the user to accept an End-User License Agreement (EULA) during an authentication flow.

About this task

Configure the End-User License Agreement and the corresponding properties to determine when the mechanism will show the license agreement.

Note: When you accept the license, the date that you last accepted the license file is stored.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **End-User License Agreement**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Accept If Last Accepted Date Before

If the date the user last accepted the license is before this date, the mechanism requires the user to accept the license again.

Data type: Date

There is not a default value.

Valid values: A date in the following format: YYYY-MM-DD

Always Show License

Set this option to true so that the mechanism always prompts the user to accept the license file.

Data type: Boolean

Default value: false

License File

Specify the path to the license template file to display for the End-User License Agreement. The path to the license file is relative to the locale in the template tree. For more information about how to update the license and add additional license files, see [Template files and Template file macros](#)

Data type: String

Default value: /authsvc/authenticator/eula/license.txt

License Renewal Term

Specify the number of days until the user must accept the license again. When you specify a value that is less than 1, there is not a renewal term. This property compares the date that the user last accepted the license to the current date. The software then determines the number of days since the user last accepted the license.

Data type: String

Default value: 0

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy them. See Chapter 15, “Deploying pending changes,” on page 213.

Related reference:

Authentication policy parameters and credentials

When you add or modify an authentication policy, you specify parameters for the authentication mechanism and the attributes that you want in the credential. The credentials are evaluated as part of the access control decision.

Configuring an Email Message mechanism

The Email Message mechanism provides arbitrary information about a user via either email, webpage, or JSON for consumption by users or applications.

Before you begin

Before using the Email Message mechanism, an SMTP server connection must first be configured. For more information about how to configure the SMTP server connection, see Managing server connections.

About this task

This mechanism can be used in conjunction with the Info Map mechanism. The Info Map mechanism populates some session info and potentially enriches the session further through user mapping. The Email Message mechanism then provides this information to the user via email.

For example, for a forgotten username:

- The user initiates the forgot username flow.
- The user is prompted to enter his or her email and date of birth.
- The user provides the details.
- The Info Map mechanism performs a lookup based on the information and enriches the session with the user name.
- The Email Message mechanism sends an email that provides the user name to the user.

If this mechanism is not used in conjunction with the Info Map mechanism, only information from the Access Manager credential will be made available.

To use values in the Access Manager credential or session information added by the Info Map mechanism, add wrapping @ signs to the attribute identifier in the same way as they are used in macros. For example, to make use of a user's credential that contains the attribute "**firstName**" in the template page:

```
...
This is the welcome page for @firstName@
...
```

Note: The attribute identifier is case sensitive. For example, **@firstname@** cannot be used to reference the attribute **firstName**.

You can use the Email Message mechanism to send messages in HTML format. See "HTML format for OTP email messages" on page 62.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **Email Message**.
6. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click **Modify Property**.
 - c. Enter the value for that property.
 - d. Click **OK**.
7. Take note of the properties for the mechanism.

Email Attribute Identifier

The name of the attribute that contains the email address to be used.

If this attribute is not set, the system always displays the template HTML page to the user.

Default value: emailAddress

Email Sender Value

The value to use in the sender field of an email.

Email Template

The path to the template XML file to be used when sending an email to the user.

Default value: /authsvc/authenticator/sessionattributeresponse/email_message.xml

Note: The default value omits the locale portion of the path, which you can see in the templates page view.

Error Template

The path to the template HTML file to be used when displaying an error message to the user.

Default value: /authsvc/authenticator/sessionattributeresponse/error.html

Server Connection

This field defines the SMTP connection that is used to send the email. You can select the SMTP server from the drop-down list.

8. Click **Save**.

What to do next

After you have configured the mechanism, a message that indicates the changes are not deployed will be displayed. Deploy changes when you are finished. For more information, see Chapter 15, "Deploying pending changes," on page 213.

After deploying the changes, you can create policies that include this mechanism. For more information, see Creating an authentication policy.

HTML format for OTP email messages

The HTML format for One-Time Password (OTP) email messages includes an identifying header of the Security Access Manager host that sent the message.

The email message mechanism supports sending an HTML formatted message to the configured SMTP server. The server receives a message that contains a Message-ID with information that identifies the Security Access Manager LMI host that sent the message. When your deployment is a Security Access Manager cluster, this information is useful for determining which host sent the email.

For example, in the email below, the LMI host `acme.com` is included in the Message-ID.

```
Date: Thu, 5 Oct 2017 01:55:29 -0400 (EDT)
From: support@mycompany.com
To: testuser@example.company.com
Message-ID: <1932638797.7.1507182929009.JavaMail.www-data@acme.com>
Subject: Email Subject
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: 7bit
```

Configuring the reCAPTCHA Verification authentication mechanism

The reCAPTCHA Verification authentication mechanism provides anti-robot protection.

Before you begin

The appliance uses the Google reCAPTCHA service to provide such verification. For more information, see www.google.com/recaptcha.

Note: The appliance supports only Google reCAPTCHA V2.

Before configuring a reCAPTCHA Verification mechanism, you must first complete the following steps.

- Ensure that the appliance can connect to `www.google.com`. You can test the connection in the CLI, for example:

```
myappliance.example.ibm.com:tools>
myappliance.example.ibm.com:tools> connect www.google.com:443
Test: www.google.com (address: 216.58.197.68) on port 443
Status: connection was successful
```

- Add the issuer of the Google CA certificate to the HTTP client default trust store, which is set by the value of the `util.httpClient.defaultTrustStore` advanced tuning parameter. The default value of the `util.httpClient.defaultTrustStore` parameter is `rt_profile_keys`.
 1. From the top menu, select **Manage System Settings > Secure Settings > SSL Certificates**.
 2. Select the `rt_profile_keys` key database.
 3. Select **Manage > Edit SSL Certificate Database**.
 4. Select the **Signer Certificates** tab.
 5. Select **Manage > Load**.
 6. Specify the following fields.

Server: `www.google.com`
 Port: `443`
 Certificate Label: `Google`
 7. Click **Load**.

About this task

The reCAPTCHA Verification mechanism can provide protection against spam or abuse caused by robots. With this mechanism, the user is presented with a web page that contains a simple Turing test provided by the Google reCAPTCHA API. These tests can distinguish a human user from a robot. You can add this mechanism to your policy to prevent robots from accessing your applications.

The following HTML snippet shows an example of embedding the reCAPTCHA mechanism in the template page:

```
<form method="POST" action="@ACTION@">
  <input type="hidden" name="operation" value="verify"></input>
  <div class="g-recaptcha" data-sitekey="@SITE_KEY@"></div>
  <br>
  <div class="controls">
    <input class="submitButton" id="Submit" name="Submit"
      type="submit" value="Submit"></input>
  </div>
</form>
```

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **reCAPTCHA Verification**.
6. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click **Modify Property**.
 - c. Enter the value for that property.
 - d. Click **OK**.
7. Take note of the properties for the mechanism.

Site Key

This property is embedded in the HTML template and used to generate the CAPTCHA in the client browser.

Default value: 6LeIxAcTAAAAAJcZVRqyHh71UMIEGNQ_MXjiZKhl

Secret Key

This property is used on the server side by the appliance to verify reCAPTCHA responses with Google.

Default value: 6LeIxAcTAAAAAGG-vFI1TnRWxMZNFuojJ4WifJWe

Note: The default Site Key and Secret Key values are designated Google test credentials. When these default values are used, all verification requests will pass.

Template Page

The path to the template HTML page to be displayed to the user.

Default value: /authsvc/authenticator/recaptcha/standalone.html

8. Click **Save**.

What to do next

After you have configured the mechanism, a message that indicates the changes are not deployed will be displayed. Deploy changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

After deploying the changes, you can create policies that include this mechanism. For more information, see [Creating an authentication policy](#).

Configuring an Info Map authentication mechanism

Use this mechanism in your policy to return a template form and perform validation on the responding POST data. This mechanism is intended to work in conjunction with the Email Message mechanism.

About this task

The Info Map mechanism can be used to implement JavaScript authentication mechanisms. When this mechanism is invoked, the configured JavaScript mapping rule will be run.

- If the rule returns FALSE, then a page will be returned to the user. The JavaScript can also define which page to return or it can use a preconfigured page. The JavaScript can also populate any macros on the page and modify what is displayed to the user.
- If the rule returns TRUE, then the mechanism will return success and the policy will continue.

The following parameters are available in an Info Map mapping rule:

Context

This is an authentication service context. It is identical to what is provided in the Authentication Service Credential mapping rule. For more information about how to use the context, see the context attributes section of [Authentication policy parameters and credentials](#).

Use the context to make changes to the credential and the values that the Email Message mechanism will display.

State

A state map that is used for the lifetime of this mechanism invocation.

Note: Each instance of this mechanism will have a new state map created per invocation of the policy. If the user invokes the policy again, the state map will be empty because the state map is discarded when the rule returns TRUE.

Page

The path to the page to be returned. By default, this parameter is set to the value that is configured in the mechanism properties. It can be modified to return a different page.

Macros

A map of macros that will be populated on the returned page.

Success

Indicates whether the rule execution was successful. This parameter is set to TRUE if the rule was successful and the policy will continue. It is set to FALSE if the rule was not successful and a page will be returned to a user.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **Info Map Authentication**.
6. In the New Authentication Mechanism window, set the name and identifier of the mechanism on the **General** tab. If you are modifying an existing Info Map authentication mechanism instead of creating a new instance, values on the **General** tab cannot be changed.
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click **Modify Property**.
 - c. Enter the value for that property.

Template Page

This property defines the HTML template page.

Mapping Rule

Select a mapping rule from the list. Only JavaScript mapping rules in the **InfoMap** category are displayed in the list for selection.

- d. Click **OK**.
 - e. Repeat the previous steps as needed.
8. Click **Save**.

What to do next

After you have configured the mechanism, a message that indicates the changes are not deployed will be displayed. Deploy the changes when you are finished. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

After deploying the changes, you can create policies that include this mechanism. For more information, see Creating an authentication policy.

Embedding reCAPTCHA verification in an Info Map mechanism

You can embed reCAPTCHA verification in Info Map mechanism instances.

The Site Key and Secret Key configured in the reCAPTCHA Verification mechanism are made available to the Info Map mechanism via two default macros: @RECAPTCHASITEKEY@ and @RECAPTCHASECRETKEY@.

For more information about configuring reCAPTCHA support, see “Configuring the reCAPTCHA Verification authentication mechanism” on page 62.

In the HTML template:

- Include the reCAPTCHA JavaScript in the head of your template
- Include the reCAPTCHA element in your form

For example:

```
...
<SCRIPT SRC="https://www.google.com/recaptcha/api.js" ASYNC DEFER></SCRIPT>
...
<DIV CLASS="g-recaptcha" DATA-SITEKEY="@RECAPTCHASITEKEY@"></DIV>
...
```

In the JavaScript mapping rule:

- The reCAPTCHA response is received in a request parameter named **g-recaptcha-response**
- Verification can be performed in the JavaScript mapping rule

For example:

```
importClass(Packages.com.ibm.security.access.recaptcha.RecaptchaClient);
...
// Retrieve the reCAPTCHA response from the request.
var captchaResponse = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:parameter",
    "g-recaptcha-response");
// Verify captchaResponse with Google, using @RECAPTCHASECRETKEY@. The third parameter
allows you to overload Google's verify endpoint URL, passing null will use the default.
var captchaVerified = (captchaResponse != null) && RecaptchaClient.verify(captchaResponse,
    macros.get("@RECAPTCHASECRETKEY@"), null);
```

Available parameters in Info Map

The following parameters are available in a mapping rule that is invoked by the Info Map authentication mechanism.

Input:

- **var:context** - type:Context - The same session context that is passed into the authsvc_credential mapping rule. For more information, see Table 2 in Authentication policy parameters and credentials.
- **var:state** - type:Map - Any values placed in the user's state by prior invocations of this instance of the Info Map authentication mechanism.

Output:

- **var:page** - type:InfoMapString - The page template to be displayed if this rule returns false. The following methods can be used to get and set the string:
 - java.lang.String getValue()** - Returns the internal String value.
 - void setValue(java.lang.String value)** - Sets the internal String value.

For more information, see the Javadoc on the appliance.

- `var:macros` - type: `Map<String, String>` - Values to populate on the returned template page.
- `var:result` - type: `InfoMapResult` - An object that captures the status of this Info Map invocation. Set it to true to signal success or false to signal failure. The following method is used to set success:

`void setValue(boolean value)`

For more information, see the Javadoc on the appliance.

Embedded Cloud Identity API calls in an Info Map mechanism

You can embed Cloud Identity (CI) API calls in Info Map mechanism instances with a new client, CI Client. Configure a CI Server connection to make calls with the CI Client.

The client ID and client secret that is configured in the CI Server Connection are made available to the CI Client by using the Server Connection Factory. The CI Client then automatically manages the client credentials token.

For more information about configuring the CI Server Connection, see Server connection properties.

For example:

```
importPackage(Packages.com.ibm.security.access.ciclient);
importPackage(Packages.com.ibm.security.access.server_connections);
importClass(Packages.com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils);

IDMappingExtUtils.traceString("entry Cloud Identity Mapping Rule");

var connection = ServerConnectionFactory.getCiConnectionByName("milano");

var id = CiClient.getUserId(connection, "testuser@ibm.com");
if (id != null) {
    IDMappingExtUtils.traceString("CI User ID: " + id);
} else {
    IDMappingExtUtils.traceString("CI User does not exist.");
}
```

Available CI Client methods include:

`getUser(CiServerConnection connection, String username)`
Retrieve a user object (via SCIM) by username

`getId(CiServerConnection connection, String username)`
Retrieve the user's IUI (via SCIM) by username

`registerAuthenticator(CiServerConnection connection, String json)`
Initiate device registration (to be completed by the user's authenticator app)

`getAuthenticators(CiServerConnection connection, String username)`
Get all authenticators for the given user

`getAuthenticator(CiServerConnection connection, String id)`
Get a specific authenticator based on ID

`updateAuthenticator(CiServerConnection connection, String id, String json)`
Update a specific authenticator based on ID

`deleteAuthenticator(CiServerConnection connection, String id)`
Delete a specific authenticator based on ID

`getAuthMethods(CiServerConnection connection, String username)`

Get all auth methods for the given user

`getAuthMethod(CiServerConnection connection, String id)`
Get a specific auth method based on ID

`updateAuthMethod(CiServerConnection connection, String id, String json)`
Update a specific auth method based on ID

`deleteAuthMethod(CiServerConnection connection, String id)`
Delete a specific auth method based on ID

`createTransaction(CiServerConnection connection, String authenticatorId, String json)`
Create a new transaction for the given authenticator ID

`getTransactions(CiServerConnection connection, String authenticatorId)`
Get all transactions for the given authenticator ID

`getTransaction(CiServerConnection connection, String authenticatorId, String id)`
Get a specific transaction for the given authenticator ID

`getRequest(CiServerConnection connection, String url)`
Generic GET request on the given URL

`postRequest(CiServerConnection connection, String url, String json)`
Generic POST request on the given URL

`putRequest(CiServerConnection connection, String url, String json)`
Generic PUT request on the given URL

`deleteRequest(CiServerConnection connection, String url)`
Generic DELETE request on the given URL

Configuring a Knowledge Questions authentication mechanism

The Knowledge Questions authentication mechanism is an extra step-up authentication measure that uses knowledge questions and answers to authenticate the user.

Before you begin

The user must register answers to the knowledge questions that the mechanism uses during authentication.

About this task

The mechanism requires users to provide personal information to successfully authenticate. You can use the Knowledge Questions authentication mechanism:

- With user ID and password authentication to provide two-factor authentication.
- As a step-up authentication method when the user accesses a high-value resource or performs a high-value transaction.

The administrator can configure the mechanism to provide a predetermined list of knowledge questions, or the user can specify and register their own knowledge questions. Typical knowledge questions about the user might include:

- Mother's maiden name.
- Name of first grade teacher.
- Name of favorite pet.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **Knowledge Questions**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Allow User Provided Questions

Specify true to specify custom questions as opposed to pre-configured questions.

Default value: true

Valid values: Boolean

Answer Hashing Algorithm

Specify this property to indicate the hashing algorithm that the appliance uses to store the knowledge questions for each user.

Default value: SHA-256

Valid values include the following string values:

- SHA-1
- SHA-256
- SHA-512

Answer Hashing Enabled

The mechanism uses a hashing algorithm to store hash values of the answers to the knowledge questions provided by the user instead of storing the actual answers to the knowledge questions. This prevents the administrator from reading the knowledge question answers for the user. Specify **False** so that the mechanism does not hash the question answer before it stores it.

Default value: true

Valid values: Boolean

Correct Answers Required

Specify the number of correct answers that is required for the authentication to be successful.

Default value: 1

Valid values: Any positive integer that does not exceed the number of questions that are stored per user.

Retry Count Attribute Name

Specify the number of times that a user can submit invalid answers to the knowledge questions. When the user reaches this number, they are unable to authenticate.

Default value: `user:knowledge:questions:retry:count`

Valid values: String

Grace Period Authentication Count Attribute Name

Specify the name of the attribute that is used to record the number of times the user has authenticated during the grace period. The number of times that the user has authenticated during the grace period is stored in the user information database. The mechanism does not require the user to authenticate during the grace period.

Default value: `user:knowledge:questions:grace:period:count`

Valid values: String

Maximum Amount of Answers Stored

Specify the maximum number of question and answer combinations that the mechanism can store for each user.

Default value: 3

Valid values: Any positive integer.

Maximum Amount of Grace Period Authentications

Specify the maximum number of user authentications that the mechanism permits during the grace period. The mechanism does not require the user to configure knowledge questions during the grace period.

Default value: 0

Valid values: Any positive integer.

Presentation Mode

Specify Individual so that the mechanism presents one knowledge question at a time. When you specify Group, the mechanism presents all of the knowledge questions in one form.

Default value: Group

Presentation Order

Specify Sequential so that the mechanism presents the questions in the order that they are stored. When you specify Random, the mechanism presents the questions in random order.

Default value: Random

Questions Attribute Name

Specify the name of the attribute that is used to store the user knowledge questions in the user information database.

Default value: `user:knowledge:questions`

Valid values: String

Retry Protection Enabled

Specify false to disable retry protection.

Default value: true

Valid values: Boolean

Retry Protection Max Number Of Attempts

Specify the maximum number of times that a user can supply incorrect answers before the mechanism prohibits the user from logging in.

Default value: 5

Valid values: Integer

Retry Timeout

Specify the number of seconds that a user must wait before trying to log in again after the user reaches the maximum number of login attempts.

Note: If a value of -1 is entered the user is locked out indefinitely until an administrator explicitly unlocks the user with the SCIM API.

Default value: 600

Valid values: Integer

Use Exact Answer Matching

Specify true so that the mechanism performs an exact match when it validates the submitted answer.

Default value: false

Valid values: Boolean

User Attributes Namespace

Specify the namespace to be used to store all of the user attributes that are related to the Knowledge Questions authentication mechanism that are stored in the user information database.

Default value:

urn:ibm:security:authentication:asf:mechanism:knowledge_questions

Valid values: String

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy them. See Chapter 15, “Deploying pending changes,” on page 213.

Related reference:

Authentication policy parameters and credentials

When you add or modify an authentication policy, you specify parameters for the authentication mechanism and the attributes that you want in the credential. The credentials are evaluated as part of the access control decision.

Configuring a FIDO Universal 2nd Factor authentication mechanism

The FIDO Universal 2nd Factor authentication mechanism prompts the user to sign a random challenge string with a FIDO Universal 2nd Factor token provided during the authentication flow.

Before you begin

The user must register a compatible FIDO Universal 2nd Factor token.

About this task

Configure the FIDO Universal 2nd Factor and the corresponding properties to determine the operation of the mechanism.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **FIDO Universal 2nd Factor**.
6. Click **Modify**.
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click **Modify**.
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Application ID

The protocol, hostname, and port that the user will use to attempt authentication.

Default value: `https://webseal.com`

Valid values: String, valid URL

Attestation Type

The type of certificate attestation validation to perform. Specify **None** to not perform certificate attestation validation. Specify **Keystore** to perform certificate attestation validation using the keystore configured in **attestationSource**. Specify **JWKS** to perform certificate attestation validation using the JSON Web Key Set configured in **attestationSource**.

Default value: **None**

Valid values: **None**, **Keystore**, **JWKS**

Attestation Source

The keystore or key set to use for certificate attestation validation. Either the name of the keystore on the appliance, or the URL for a JSON Web Key Set.

Default value: No default value

Valid values: String

Attestation Enforcement

The level of enforcement of certificate attestation validation. When you specify **Required**, certificate attestation validation is required, and requests that fail validation will return a validation error. When you specify **Optional**, certificate attestation validation is performed, but requests that fail validation will not return an error.

Default value: **Required**

Valid values: Required, Optional

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy them. See Chapter 15, “Deploying pending changes,” on page 213.

Related information:

Authentication policy parameters and credentials

Configuring a FIDO2/WebAuthn authentication mechanism

The FIDO2/WebAuthn authentication mechanism prompts the user to sign a random challenge string with a FIDO2/WebAuthn authenticator provided during the authentication flow.

Before you begin

The user must register a compatible FIDO2/WebAuthn authenticator.

About this task

Configure the FIDO2/WebAuthn mechanism and the corresponding properties to determine the operation of the mechanism.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **FIDO2/WebAuthn Authentication**.
6. Click **Modify**.
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click **Modify**.
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Relying Party ID

The relying party ID to use with this mechanism. The relying party ID is a domain string that identifies the Relying Party and the Relying Party-specific configuration to use. See “FIDO2 Configuration” on page 199.

The Relying Party ID must be based on the origin that the user accesses, that is the Relying Party ID must be the registrable domain suffix of or equal to the origin's effective domain.

Default value: Empty string

Example: webseal.com

Valid values: String, valid URI

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that changes are not deployed. Deploy them. See Chapter 15, “Deploying pending changes,” on page 213.

More advanced configuration can be configured for each Relying Party ID. See “FIDO2 Configuration” on page 199.

Configuring a QR Code authentication mechanism

The QR Code authentication mechanism is an authentication capability that permits a registered device to scan a QR Code to authenticate the user. It provides a completely alternative-to-password method of authenticating a user.

About this task

The mechanism requires users to scan a generated QR code to successfully authenticate using a previous registered application such as IBM Verify or an equivalent built on the IBM Verify SDK. The QR Code authentication mechanism operates in one of the following modes:

Initiate

In this mode the mechanism generates a QR code and displays it to the user. It then waits for the code to be scanned or a timeout period to be reached. The waiting process consists of polling the authentication policy using a `device_session_index` until it is associated with an authenticated user. Scanning the code results in the IBM Verify mobile application contacting a companion authentication policy. This policy uses the same mechanism in **Response** mode. After successful login with the QR Code scan, there are three attributes that are made available in the session context for downstream policies:

- **urn:ibm:security:asf:qrcode.prompt**- This is a confirmation message that might be used by other mechanisms to ensure that the QR code login operation is what the user intended.
- **urn:ibm:security:asf:qrcode.qr_login_session_index**- This is analogous to the `user_code` from the OAuth device flow.
- **urn:ibm:security:asf:qrcode.qr_device_session_index**- This is analogous to the `device_code` from the OAuth device flow.

Response

In this mode the mechanism associates the `login_session_index` with the authenticated username from the request. Any associated policy using the QR code mechanism in **Initiate** mode that is polling on the `device_session_index` is unblocked and completed.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **QR Code**.

6. Click  .
7. Click the **Properties** tab.
 - a. Select a property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the properties for the mechanism.

Timeout

This is the period in seconds that the QR code remains valid.

Enable Browser Testing

This is a flag that can be set such that if a registered device is not available to scan the QR Code, the user can simulate the back channel flow with another (authenticated) browser. This is only relevant when the mechanism is configured in **Response** mode and should only be used for testing the mechanism.

- a. Login to IBM Security Access Manager by using a protected page. For example, `<https://<reverseproxy>>`.
- b. Navigate to the "backchannel" URL with a browser, where you are able to enter the login session index (LSI) to authenticate. The LSI is shown on the QR code login page in clear text for this reason:
`<https://<reverseproxy>/mga/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:qrcode_response>`

9. Click **Save**.

Enabling or disabling authentication policies

You can selectively enable or disable authentication policies to control exactly which authentication policies are enabled in your environment.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Policies**.
 - To enable or disable a specific authentication policy, select the authentication policy and then click **Enable** or **Disable**. Follow the prompts.
 - To enable or disable all authentication policies, click **Enable all** or **Disable all**. Follow the prompts.

Managing mapping rules

The mapping rules are JavaScript code that run during the authentication flow. Use the rules to customize the authentication service and the one-time password generation, delivery, and verification.

Before you begin

Attention: Use extreme care when you replace mapping rules. Any change that you make to a mapping rule can affect the entire runtime environment. Always export a copy of the original rule you plan to replace so that you have a backup copy.

About this task

You can customize several components through JavaScript code. For example, you can customize the Authentication Service to modify the content of user credential by modifying the AuthSvcCredential mapping rule.

The JavaScript code is run by the Rhino JavaScript engine. Your JavaScript code must conform to JavaScript 1.7. Your JavaScript code is not run under a browser environment. Therefore, you cannot use objects and functions that are available only in a browser environment. You can, however, use standard JavaScript objects (such as Math) and functions (such as parseInt). In addition, your JavaScript code can use white-listed Java classes, which you might need so that you can use operations that are not supported by standard JavaScript functions. You can find the list of these Java classes at “JavaScript whitelist” on page 297. To find out more about using Java classes in JavaScript, see the Rhino documentation <https://developer.mozilla.org/en/docs/Rhino>.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Advanced**.
5. Take one of the following actions:

View a mapping rule:

- a. Select a mapping rule.
- b. Click . The View Mapping Rule panel opens. The content of the mapping rule is displayed.
- c. Click **OK** to close the panel.

Export a mapping rule:

- a. Select a mapping rule.
- b. Click .
- c. Choose a location and save the file.

Replace a mapping rule:

Use an existing mapping rule as the basis for the updated mapping rule.

Attention: When you replace this file, an error in the JavaScript source might be found immediately after it is replaced or it might not be found until the file is run.

- a. Select a mapping rule that you want to replace.
- b. Click . The Replace Mapping Rule panel opens.
- c. Click the field or the **Browse** button and select a file.

Attention: The name of the mapping rule cannot be replaced. The name of the uploaded file is ignored.

- d. Click **OK** to upload the mapping rule.

What to do next

When you replace a mapping rule, the appliance displays a message that there are undeployed changes. Deploy the changes when you are done. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Authentication Service Credential mapping rule

The Authentication Service Credential mapping rule is JavaScript code that you can use to customize the information that is contained in the user credential.

During authentication, the Authentication Service gathers information about the authenticated user, including attributes associated with the user ID. After successful authentication, the Authentication Service provides this information to the Authentication Service Credential mapping rule. The main task of the mapping rule is to modify or add attributes to the user information before it is used to generate a credential.

Customizing the mapping rule is an advanced way to customize the credential. To specify basic credential attributes, use an authentication policy and the Credentials panel in the local management interface instead of creating a custom mapping rule. See *Creating an authentication policy*.

If you write your own mapping rule and use it to replace the existing rule, be aware of the following considerations:

- Credential attributes are string values. For example, user names and lists of groups are string arrays.
- Do not use spaces, commas, or colons in credential attribute names. Use alphanumeric characters.

The sample mapping rule provides more descriptions about considerations for writing your own mapping rule.

A default `AuthSvcCredential` mapping rule is provided. To review the rule:

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Advanced**.
5. Select `AuthSvcCredential`.
6. Click  .
7. Choose a location and save the file.

To review an example of a customized credential mapping rule:

1. Log in to the local management interface.
2. Click **Manage System Settings**.
3. Click **File Downloads**.
4. Click `access_control > examples > mapping_rules`.
5. Select `authsvc_credential.js`.

6. Click **Export** to download the file.

If you create your own rule, use it to replace the existing rule. See the replacement instructions in “Managing mapping rules” on page 75.

OTPGetMethods mapping rule

OTPGetMethods specifies the methods for delivering the one-time password to the user.

This sample mapping rule sets password delivery conditions for the following delivery methods:

- By email
- By SMS
- No delivery

Each delivery method includes the following attributes and their corresponding value:

id Specifies a unique delivery method ID. This value replaces the @OTP_METHOD_ID@ macro in the OTP Method Selection page. Use a unique value across different methods. For example, sms.

deliveryType

Specifies the delivery plug-in that delivers the one-time password. The value must match one of the types in the DeliveryTypesToOTPDeliveryModuleIds parameter of the OTP response file. For example, sms_delivery.

deliveryAttribute

Specifies an attribute that is associated with the delivery type. The value depends on the one-time password provider plug-in for the delivery type. For example:

- For SMS delivery, the value is the mobile number of the user. For example, mobileNumber.
- For email delivery, the value is the email address of the user. For example, emailAddress.
- For no delivery, the value is an empty string.

label Specifies the unique delivery method to the user. For time-based and counter-based one-time password, use this attribute to specify the secret key of the user. If label is not specified, the time-based and counter-based one-time password code retrieves the key by invoking the user information provider plug-in. This parameter replaces the @OTP_METHOD_LABEL@ macro in the OTP Method Selection page.

otpType

Specifies the one-time password provider plug-in that generates and verifies the password. The value must match one of the types in the OTPTypesToOTPProviderModuleIds parameter of the OTP response file. For example, mac_otp.

userInfoType

Specifies which user information provider plug-in to use to retrieve user information that is required to calculate the one-time password. This parameter is only required if user information is used for calculation of the one-time password.

To customize one-time password delivery, you can do one of the following actions:

- Create your own mapping rules that are based on the sample `OTPGetMethods` mapping rule.
- Modify the sample `OTPGetMethods` mapping rule.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPGenerate mapping rule

`OTPGenerate` mapping rule specifies the generation of the one-time password for the user.

You can use the `OTPGenerate` mapping rule in the following configuration:

Modify the one-time password type of the selected method to generate the one-time password

Indicates the one-time password type to determine the one-time password Provider plug-in that generates the one-time password for the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPDeliver mapping rule

The `OTPDeliver` mapping rule specifies the delivery method of the one-time password to the user.

Use the following `OTPDeliver` mapping rules:

Generate the one-time password hint

The one-time password hint is a sequence of characters that is associated with the one-time password. The one-time password hint is displayed in the One-Time Password Login page. It is also sent to the user together with the one-time password.

You can customize the way the one-time password hint is generated by modifying the following section in the default `OTPDeliver` mapping rule:

```
var otpHint = Math.floor(1000 + (Math.random() * 9000));
```

Note: See the comments in the mapping rule for more details.

Generate the formatted one-time password

The formatted one-time password is the formatted version of the one-time password. The formatted one-time password, instead of the actual one-time password, is sent to the user. For example, for one-time password hint `abcd`, and one-time password `12345678`, you can set the formatted one-time password as `abcd-12345678`. For one-time password hint `efgh`, and one-time password `87654321`, you can set the one-time password as `efgh#8765#4321`.

You can customize the way that the one-time password is generated by modifying the following section in the sample `OTPDeliver` mapping rule:

```
var otpFormatted = otpHint + "-" + otp;
```

Note: See the comments in the mapping rule for more details.

Modify the delivery type of the selected method for delivering the one-time password

The delivery type specifies the one-time password Delivery plug-in that delivers the one-time password to the user.

Modify the delivery attribute of the selected method to deliver

The delivery attribute is an attribute that is associated with delivery type. The meaning of the delivery attribute depends on the one-time password provider plug-in for the delivery type. For example, for SMS delivery type, the delivery attribute is the mobile number of the user. For email delivery type, the delivery attribute is the email address of the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPVerify mapping rule

OTPVerify specifies the verification of the one-time password that is submitted by the user.

You can customize the sample OTPVerify mapping rule to modify the following verification rules:

Modify the one-time password type of the user

Indicates the one-time password type to determine the one-time Provider plug-in that verifies the one-time password submitted by the user.

Set the authentication level of the user

After one-time password authentication completes, a credential is issued that contains the authentication level of the user. You can customize the authentication level by modifying the following section in the mapping rule:

```
var authenticationLevel = contextAttributesAttributeContainer.getAttributeValueByNameAndType("otp.otp-callback.authentication-level", "otp.otp-callback.type");
var attributeAuthenticationLevel = new Attribute("AUTHENTICATION_LEVEL",
    "urn:ibm:names:ITFIM:5.1:accessmanager", authenticationLevel);
attributeContainer.setAttribute(attributeAuthenticationLevel);
```

Enforce the number of times the user can submit the one-time password in the one-time password login page

If a user exceeds the permitted number of times to submit a one-time password, an error message displays. You can customize the number of times that the user can submit the one-time password in the one-time password login page by modifying the following section in the mapping rule:

```
var retryLimit = 5;
```

By default, this option is set to false.

Note: This setting applies only to MAC OTP.

Identify the secret key of a user

When a user registers with a time-based one-time password application, they are assigned a secret key. Store the secret key in this mapping rule for verification of the user by modifying the following code:

```
var secretStr = new java.lang.String(SECRET_KEY_GOES_HERE);
```

By default, this option is set to false.

Override the one-time password target URL

By default, a user is redirected to a target URL upon completion of an one-time password flow. That target URL was either the initial cached request at the WebSEAL or reverse proxy instance or was specified as part of the one-time password invocation using the **Target** query string parameter.

You can use the OTPVerify mapping rule to override this target URL by adding an attribute called **itfim_override_targeturl_attr**. This attribute ensures that at the completion of a successful one-time password flow, the user is redirected to the override target instead of the initial target.

Example code:

```
var targetUrl = new java.lang.String("http://www.example.com/url");
var targetUrlAttr = new Attribute("itfim_override_targeturl_attr",
    "urn:ibm:names:ITFIM:5.1:accessmanager", targetUrl);
attributeContainer.setAttribute(targetUrlAttr);
```

To customize one-time password verification, you can do one of the following actions:

- Create your own verification rules that are based on the sample OTPVerify mapping rule.
- Modify the sample OTPVerify mapping rule.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data.”

Customizing one-time password mapping rules to use access control context data

Some authentication scenarios require that context data used in making an access control decision be available during authentication. You can configure Security Access Manager to capture the content data and make it available to the one-time password mapping rules.

About this task

You can configure Security Access Manager to perform access control policy evaluation when a resource is accessed. The access control policy evaluation can result on a permit with authentication. The required authentication is determined by the access control policy. Some scenarios require that the context data used to perform the access control decision be available during the authentication. In order to provide access to the access control context data, you can persist the context information for the predefined authentication obligations that perform one-time password authentication.

Note: The context data available is limited to the attributes referenced by the access control policy and the request attributes provided by the policy enforcement point. If the policy relies on the risk score to perform access control, the context data available also includes the risk-profile attributes.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control > Global Settings > Advanced Configuration**.

3. Select **attributeCollection.authenticationContextAttributes**.

4. Click  for the property.

5. In the text field, enter a list of comma separated attribute names to be collected during the authorization policy evaluation. For example, if your scenario requires the authentication level and host of the request the configuration property, enter `authenticationLevel, http:host`. The access control context data is provided to the one-time password mapping rules as context attributes values. The following format is used:

```
<stsuser:Attribute name="AttributeName-AttributeURI"
  type="authn.service.context.attribute.type.AttributeDatatype">
<stsuser:Value>AttributeValue</stsuser:Value>
</stsuser:Attribute>
```

Where:

- name is the attribute name and attribute identifier separated by a dash (-).
- type is the attribute data type prefixed by `authn.service.context.attribute.type`.

For example the `authenticationLevel` attribute value is added as:

```
<stsuser:Attribute name="authenticationlevel-urn-ibm:
  security:subject:authenticationlevel"
  type="authn.service.context.attribute.type.Integer">
<stsuser:Value>1</stsuser:Value>
</stsuser:Attribute>
```

6. Click **OK**.

7. When you edit a property, a message indicates that there are undeployed changes. If you have finished making changes, deploy them.

For more information, see Chapter 15, “Deploying pending changes,” on page 213.

8. Configure the mapping rule to use the information collected by this property as the context attribute.

a. Click **Secure Access Control**.

b. Under **Policy**, click **Authentication**.

c. Click **Advanced**.

d. Select and export the mapping rule.

e. Use a text editor and modify the rule to access the attributes collected during the access control policy evaluation in the following format:

```
var accessControlAttribute =
contextAttributesAttributeContainer.getAttributeValueByNameAndType
("AttributeName-AttributeURI",
"authn.service.context.attribute.type.AttributeDatatype");
```

Where:

- name is the attribute name and attribute identifier separated by a dash (-).
- type is the attribute data type prefixed by `authn.service.context.attribute.type`.

For example, the `authenticationLevel` attribute can be obtained using the following information:

```
var accessControlAuthenticationLevel =
contextAttributesAttributeContainer.getAttributeValueByNameAndType
("authenticationlevel-urn-ibm:security:subject:authenticationlevel",
"authn.service.context.attribute.type.Integer");
```

f. Save the mapping rule and take note of its location.

g. In the local management interface, click **Secure Access Control**.

- h. Under **Policy**, click **Authentication**.
- i. Click **Advanced**.
- j. Select the mapping rule you want to replace.
- k. Click **Replace**. The Replace Mapping Rule panel opens.
- l. Click the field or the **Browse** button and select the file for your saved mapping rule.
Attention: The name of the mapping rule cannot be replaced. The name of the uploaded file is ignored.
- m. Click **OK** to upload the mapping rule.

One-time password and authentication template files

The one-time password and authentication methods rely on HTML pages to interact with users, such as displaying errors or prompting users to provide a password or pin or to indicate the method by which they want to receive the password. You can customize these pages using the one-time password and authentication template files.

For information about one-time password and authentication template files, see [Template files](#).

Push notification registration

Security Access Manager supports push notifications on both iOS and Android platforms. It can also be configured to send push notifications to the IBM Verify application.

About this task

To issue a notification to a client device, a specific payload must be generated and sent to the push notification service of the device's platform (Apple Push Notification Service, Firebase Cloud Messaging, or Push for IBM Verify). This notification request requires a form of authentication and authorization. To establish a trusted connection, Apple Push Notification Service requires a provider certificate, Firebase Cloud Messaging requires a server (API) key, and Push for IBM Verify requires configuration of authentication credentials.

As an administrator, you must register such forms of authentication for your authenticator applications to successfully deliver push notifications to clients on demand. Such registration can be done through either the local management interface or the RESTful API. For details about how to register push notification endpoints through the RESTful API, see the RESTful API documentation.

Note: For certificate-based push notification registration, use a specific SSL certificate database for this purpose and import all required certificates to the SSL certificate database before registration.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Manage**, click **Push Notification Providers**.

Adding a push notification provider

- a. Click **Add**.

- b. Provide values for the displayed fields.

Mobile Platform

Specifies whether the push notification is for iOS or Android platform.

Application ID

Identifier of the application.

Push Provider Host

Host name to be used to connect to the push service provider. The value can include port number, for example, fcm.googleapis.com:443.

Push Provider

Select the provider for your push notifications. The available options are **Firestore** (Google's push notification provider), **Apple** (Apple's push notification service), or **Push for IBM Verify**.

Server Key

If **Android Application** is selected in the **Mobile Platform** field or **Firestore** is selected as the **Push Provider** when **iOS Application** is selected in the **Mobile Platform** field, then this text field becomes available to enter the Server API Key to be used for authentication.

Certificate Store

If **iOS Application** is selected in the **Mobile Platform** field and **Apple** is selected as the **Push Provider**, then this field becomes available to select the certificate store on the appliance that contains the certificate to be used to authenticate to the Apple push notification service.

Certificate Label

If **iOS Application** is selected in the **Mobile Platform** field and **Apple** is selected as the **Push Provider**, then this field becomes available to select the certificate to be used to authenticate to the Apple push notification service.

- c. Click **Save**.
- d. Deploy the changes.

Modifying a push notification provider

- a. Select the push notification provider to be modified.
- b. Click **Edit**.
- c. Change the settings as needed.
- d. Click **Save**.
- e. Deploy the changes.

Obtaining the required authentication credentials to configure push notification for IBM Verify

IBM Verify can receive push notifications from Security Access Manager to alert users about pending transactions. But first Security Access Manager must be configured to have the required IBM Verify authentication credentials before it can send push notifications to IBM Verify.

To configure your Security Access Manager appliance to send push notifications to IBM Verify, you must create a push notification application in Security Access Manager. This process requires you to have a push notification account including a unique client ID, client secret, and refresh token for push notification.

1. Send an email to the IBM Verify Administrator, `verify@au1.ibm.com`, with the subject line **“IBM Verify Push Notification Registration”**.

In the email contents, include your Name, IBM ID, company name, and site number (as provided in your Passport Advantage account).

For example:

Name: Joe Smith
IBM ID: jblack@thecompany.com
Company Name: The Company Pty Ltd
Site Number: 1768

2. You will receive an email reply within two business days with the following details:

Mobile platform: iOS Application
Application ID: com.ibm.security.verifyapp
Push Provider Host: api8.silverpop.com
Push Provider: Push for IBM Verify
Client ID: xxx
Client Secret: xxx
Refresh Token: xxx
App Key: apdREtdPxG

Mobile platform: Android Application
Application ID: com.ibm.security.verifyapp
Push Provider Host: api8.silverpop.com
Push Provider: Push for IBM Verify
Client ID: xxx
Client Secret: xxx
Refresh Token: xxx
App Key: gcGgrbKtJZ

3. Configure a new push notification application on your appliance. Select **Push for IBM Verify** as the **Push Provider** during the configuration. For more information, see “Push notification registration” on page 83.

Cloud Identity API Integration

Cloud Identity supports several multi-factor authentication types including IBM Verify. One advantage of leveraging authentication methods from the cloud is that the methods can be updated with newer technology more rapidly, and new methods can be adopted without the need for a Security Access Manager update.

A second advantage is that Cloud Identity supplies both an email gateway and an SMS gateway, for SMS and Email OTP methods.

Instead of redirecting users to Cloud Identity to perform authentication the Cloud Identity API integration within ISAM can be used. This allows for complete control over the look and feel of the authentication experience.

The API Integration is achieved through a series of Info Map rules as well as a new Authentication Mechanism type - Cloud Identity JavaScript. The new mechanism type is very similar to an Info Map mechanism, with a few extra properties.

Cloud Identity JavaScript

The Cloud Identity JavaScript mechanism can be used to implement authentication and user self care flows between Security Access Manager, Cloud Identity, and the end user.

This mechanism has several properties:

Mapping Rule

The configured Info Map mapping rule to be run

Server Connection

The Cloud Identity server connection to use to perform all operations

Verify Client ID

The client ID configured for IBM Verify in Cloud Identity

Bypass if not enrolled

A boolean indicating whether to return success without attempting authentication if no multi-factor authentication methods are enrolled.

Similar to Info Map mechanisms, if the configured mapping rule returns FALSE, then a page will be returned to the user. The JavaScript must define which page to return. The JavaScript can also populate any macros on the page and modify what is displayed to the user. If the rule returns TRUE, then the mechanism will return success and the policy will continue.

The following parameters are available in the mapping rule: "Available parameters in Info Map" on page 66

After you have configured the mechanism, you can create policies that include this mechanism. For more information, see [Creating an authentication policy](#)

Authentication flow

One of the Cloud Identity JavaScript mapping rules provided out of the box is the Authentication rule, which operates at a high level as follows.

Action	Result
Empty or "initiate"	Produce a landing page with all authentication methods listed such that the user can choose which method they would like to perform authentication with.
"chooseMethod"	Create a transaction (if required) and return a page relevant to the chosen method. Waiting page for IBM Verify, OTP input page for SMS/Email/Time-Based OTP, and OTP delivery detail input page for Transient Email/SMS.
"submitTransient"	Create a transient transaction with the given OTP delivery detail. Returns a OTP input page.
"verifyOTP"	Send the OTP to Cloud Identity for verification. If the verification succeeds, progress to the next step in the policy. If verification fails, display an error to the user.

Action	Result
"checkVerify"	Check the status of the IBM Verify transaction. If the transaction was successful progress to the next step in the policy, otherwise display an error to the user.

Several parameters can be modified at the beginning of the mapping rule to control different behavior:

Variable	Affect	Affect
otpCorrelation	The correlation to use in SMS and Email OTP transactions.	"ISAM verification"
enabledMethods	The type of methods to display to a user, if enrolled.	["Verify", "SMSOTP", "EmailOTP", "TOTP", "TransientEmail", "TransientSMS"]
verifyTransactionMessage	The transaction message to send when creating Verify transactions.	"You have a pending authentication challenge."
expandVerifyMethods	A boolean indicating whether all available Verify methods should be displayed to the user, or only one (which is the highest priority in verifyMethodPriority).	false
verifyMethodPriority	The priority of Verify methods to display if expandVerifyMethods is false.	["face", "iris", "retina", "eye", "fingerprint", "userpresence"]
jitMethodEnrollment	A boolean indicating whether to redirect to the USC flow if no enrollments are found.	false

User Self Care flow

One of the Cloud Identity Javascript mapping rules provided out of the box is the User Self Care rule, which operates at a high level as follows:

Action	Result
Empty or "initiate"	Produce a landing page with all authentication methods listed and an Add new button.
"register"	Either register a new authenticator, or enroll a new method type.

Several parameters can be modified at the beginning of the mapping rule to control different behavior:

Variable	Affect	Default
enabledMethods	The type of methods to display to a user, and to allow to be enrolled.	["Verify", "SMSOTP", "EmailOTP", "TOTP"]

Configuring the authentication and access module for cookieless operation

To allow the Authentication and access module to function in like an API, use of a client side cookie can be avoided with an advanced configuration option.

Before you begin

Configure the appliance to use Authentication-based and Content-based access with one of the following methods:

- Use the internal High Volume Database (HVDB)
- Set up an external HVDB
- Set up a Distributed Session Cache (DSC)

About this task

When the cookieless operation is enabled, several configuration options are available to suit a range of deployment configurations and use cases.

In a high availability or clustered environments it is recommended that session affinity is enforced for a sufficient period of time to allow session replication between nodes. The length of time that sticky session is enforced depends on the deployment.

During normal operation a **jsession** cookie is still returned. However if this sessions cookie is returned in subsequent requests, it is ignored by the authentication service.

Note: This configuration option only removes the reliance on session cookies for the authentication service (**/sps/authsvc** and **/spsapiauthsvc**) endpoints. Users still require a webseal session cookie to maintain state.

Configure the Authentication-based and Content-based access module to not rely on client side cookies to store authentication information.

Administrators can choose to store this information in either the DSC, Memory, or the HVDB, depending on deployment requirements.

Procedure

1. In the local management interface, click **Secure Access Control > Advanced Configuration**.
2. To enable cookies operation, toggle the **authsvc.stateMgmt.cookieless** key to **Enabled**.
3. Select session store by using the **authsvc.stateMgmt.store** key (either DSC for the Distributed Session Cache, HVDB for the High volume Database or Memory for JVM memory caching):
 - Distributed Session Cache (DSC)
 - a. Enable the **distributedSessionCache.enabled** key.
 - b. Set DSC parameters:
 - **distributedSessionCache.localCacheEnabled**
 - **distributedSessionCache.localCacheSize**
 - **distributedSessionCache.externalServers**

- High Volume Database (HVDB) or Memory
 - a. Set the **authsvc.stateMgmt.HVDB.cleanupOnlyOnPrimaryMaster** for only the removing session by the primary master in clustered environments.
 - b. Set **authsvc.stateMgmt.HVDB.cleanupThread.batchSize** if a maximum cleanup batch size is required

Note: Setting this parameter as 0 disables this option.

- c. Set **authsvc.stateMgmt.HVDB.cleanupWait** to control the cleanup thread run frequency.

Note: Setting this parameter to -1 disables the cleanup thread.

- d. Set **authsvc.stateMgmt.HVDB.lifetime** for the maximum lifetime of a session in the HVDB.
- e. Set **authsvc.stateMgmt.maxSessions** to control the maximum number of sessions to cache. When this value is exceeded, IBM Security Access Manager removes the oldest sessions in the case.

Reverse Proxy Configuration with Authentication Services

In order to use the authentication service and advanced access control features, the reverse proxy protecting resources must be configured first.

Configuring advanced access control authentication on a reverse proxy

Configure the reverse proxy protecting resources to use the authentication service and access control features.

About this task

There is a tool in the Local Management Interface that configures a reverse proxy to work with these services. This tool can be invoked by the User Interface or with REST APIs.

When you invoke the REST API to configure the reverse proxy from the Local Management Interface, the following changes are made:

- The reverse proxy configuration is updated.
- Access control lists (ACLs) are created and attached.
- The rba-pop is created.
- The Runtime Server certificate is loaded into the WebSEAL Truststore.

All the changes that this API performs are documented in the REST API documentation, under *Authentication and Context based access configuration for a reverse proxy*.

This API does not perform any configuration which cannot be performed with public interfaces.

After this API is invoked, the `autocfg_authsvc.log` file in the reverse proxy logs is available to view the configuration that is performed.

Note: This topic only covers Authentication and Context based access. For OAuth, MMFA, or Federation, see one of the following topics:

- “Reverse proxy configuration for OAuth and OIDC provider” on page 125

- Configuring Mobile Multi-Factor Authentication
- Configuring a reverse proxy point of contact server

To configure a reverse proxy for use with Advanced Access Control, follow the procedure below:

Procedure

1. From the local management interface, select **Secure Web Settings > Manage > Reverse Proxy**. A list of reverse proxy instances displays.
2. Select the reverse proxy instance name from the list.
3. Select **Manage > AAC and Federation Configuration > Authentication and Context Based Access Configuration**. A window opens where you can add the configuration information. Populate or port the host as appropriate.

The username and password which are required are that of the RTSS and STS service user. It is, by default easuser.

For more information on managing these users, see “Managing user registries” on page 242.

Using the isamcfg tool

The isamcfg tool is deprecated. The configuration API invoked on a reverse proxy is used instead.

Various features of Advanced Access Control can be configured with the **isamcfg** tool. The **isamcfg** tool helps automate configuration of the following software and appliances:

- WebSEAL
- Security Access Manager appliance
- Security Access Manager for Web appliance
- Security Access Manager for Web software version
- Security Web Gateway appliance

The tools helps with:

- Creating a junction that points to the Advanced Access Control runtime endpoint.
- Creating a Security Access Manager POP used by Advanced Access Control to attach a policy.
- Configuring the instance of the appliance that supplies the web function, such as WebSEAL, with the instance of the appliance that provides the authorization server for Advanced Access Control.
- Configuring SSL, sets up key stores, trust stores, and authentication configuration between Advanced Access Control and the web function.
- Configuring a set of default obligation-to-URL mappings for the authentication service.
- Modifying the Security Access Manager appliance authentication configuration to support the authentication service.

Note: If you are upgrading, see the installation and configuration instructions in the IBM Knowledge Center.

If you are upgrading from a previous version of the product, run the **isamcfg** tool to unconfigure the current version. Then, run the **isamcfg** tool to configure the new version.

Configuring an appliance reverse proxy instance from the appliance

Use the **isamcfg** tool to configure an appliance reverse proxy instance from a local or remote appliance.

About this task

Run the command from the appliance command-line interface.

You must use this method to configure a reverse proxy instance on an appliance that is a restricted node in a cluster.

Note: Your appliance can be one of the following product versions:

- Security Access Manager 9.*
- Security Access Manager for Web 8.*
- Security Web Gateway 7.*

Procedure

1. Connect to the appliance with SSH.
2. Enter the administrator user ID and password.
3. Navigate to **isam > aac > config**. The **isamcfg** tool starts.
4. Use **isamcfg** to complete the configuration. For configuration details, see “isamcfg Security Access Manager appliance configuration worksheet” on page 97.

Results

When you complete the configuration, a summary screen displays indicating that the configuration is complete.

Configuring an appliance reverse proxy instance from an external machine

Use the **isamcfg** tool to configure an appliance reverse proxy instance from a remote machine.

Before you begin

Your appliance server and Advanced Access Control servers must be listening for connections on the appropriate management IP addresses and port numbers.

To use the **isamcfg** tool, you must meet the following conditions:

- Obtain an IBM® JRE v6.0 Update 10 or later.
- At least one reverse proxy instance exists on the appliance.
- Configure the *com.ibm.security.cmskeystore.CMSProvider* in the *java.security* file, which is in *\$JAVA_HOME/lib/security*, of the IBM® JRE. The **isamcfg** tool uses the **ikeycmd** command to manipulate key database files. This requires the JRE to have the CMS provider that is configured in the *java.security* file.
- Ensure that the **ikeycmd** tool in the *\$JAVA_HOME/bin* is on the system path.

- The reverse proxy instance that you are configuring cannot be on an appliance that is a restricted node in a cluster. See “Configuring an appliance reverse proxy instance from the appliance” on page 91.

About this task

Use this procedure if you want to configure an appliance that is on a machine that is separate from the Advanced Access Control appliance. Once you download the tool from the Advanced Access Control appliance, you can then use the command shell to configure an existing remote appliance.

The appliance you configure can be one of the following product versions:

- Security Access Manager 9.*
- Security Access Manager for Web 8.*
- Security Web Gateway 7.*

Procedure

1. Download the **isamcfg.jar** from the IBM Security Access Manager appliance with Advanced Access Control activated.
2. From the command line, type:

```
java -jar isamcfg -action config -cfgurl http://isam-appliance-host-url/
```

The *isam-appliance-host-url/* refers to the URL of the activated Security Access Manager appliance base.
3. Use the **isamcfg** tool to complete the configuration. For configuration details, see “isamcfg Security Access Manager appliance configuration worksheet” on page 97.

Results

When you complete the configuration, a summary screen displays indicating that the configuration is complete.

Configuring a WebSEAL instance

Use the **isamcfg** tool to configure WebSEAL as a point of contact and policy enforcement point for an appliance that has Advanced Access Control activated.

Before you begin

Make sure that your WebSEAL server is listening for connections on the appropriate IP addresses and port numbers. You can control the IP address and port number by using the WebSEAL configuration file. The IP address is controlled by the [server] network-interface configuration option, and the port numbers are controlled by the [server] https-port and [server] http-port options.

To use the **isamcfg** tool, you must:

- Obtain an IBM JRE, version 8.0 or later that is supported by the version of PDJRte installed.
- Ensure that the Java Runtime used to start the **isamcfg** tool is configured into the Security Access Manager domain in full mode that uses the PDJRTE. An error is displayed if this condition is not met. For more information about using the PDJRTE, see <http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/security/60/iKeyman.8.User.Guide.pdf>.

- Ensure that the **isamcfg** tool is able to access the application interface for Advanced Access Control.
- Run the command from the appliance that hosts the reverse proxy instance, if the instance is a restricted node in a cluster. Also, you must use the command-line interface to run the command.

For IBM Security Access Manager WebSEAL, version 7.0 or later, you must also meet the following conditions:

- Configure the *com.ibm.security.cmskeystore.CMSProvider* in the `java.security` file, which is in `$JAVA_HOME/lib/security`, of the IBM JRE. The **isamcfg** tool uses the **ikeycmd** command to manipulate key database files. This requires the JRE to have the CMS provider that is configured in the `java.security` file.
- Ensure that the **ikeycmd** tool in the `$JAVA_HOME/bin` is on the system path.

For Tivoli Access Manager for e-business WebSEAL versions 6.1.1 or prior, ensure that **gsk7ikm** tool is on the system path.

Run the tool on the same system where WebSEAL is located.

About this task

This procedure connects the WebSEAL software version 7.* to Security Access Manager.

Note: This procedure is not intended for deployments that have a Security Access Manager appliance with the WebSEAL function.

Procedure

1. Download the **isamcfg.jar** from the Security Access Manager appliance with Advanced Access Control.
2. On the WebSEAL machine, set up a `JAVA_HOME` environment variable for the JRE: For example:

```
export JAVA_HOME=/opt/ibm/java-x86_64-60/jre, or
export JAVA_HOME=/opt/IBM/WebSphere/AppServer/java/jre
```
3. Add `$JAVA_HOME/bin` to the path `export PATH=$JAVA_HOME/bin:$PATH`.
4. From the command line, type:

```
java -jar isamcfg.jar -action config -cfgfile /path/to/webseald.conf
```
5. Use the **isamcfg** tool to complete the configuration. For configuration details, see “isamcfg WebSEAL configuration worksheet” on page 99.

Results

When you complete the configuration, a summary screen displays indicating that the configuration is complete.

Configuring WebSEAL in a highly available environment

When you are working in an environment with multiple Security Access Manager with Advanced Access Control servers, you can configure WebSEAL for failover and high availability.

About this task

You can configure the WebSEAL junction and Runtime Security Services External Authorization Service (RTSS EAS) to take advantage of high availability.

Figure 1 depicts an environment where WebSEAL is configured to use two Security Access Manager servers, AAC_1 and AAC_2. AAC_1 and AAC_2 are appliances with Advanced Access Control activated. For high availability, you can configure a stateful junction to each available appliance. You can also include each server in the RTSS EAS configuration.

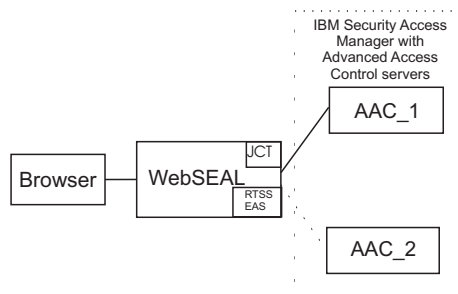


Figure 1. WebSEAL client in an environment with multiple IBM Security Access Manager servers

Advanced Access Control provides the **isamcfg** tool which configures each WebSEAL instance. This tool sets up a single junction server and configures the RTSS EAS to point to a single appliance.

If you have more than one appliance with Advanced Access Control activated, you need to manually configure the additional servers.

Procedure

1. For each Advanced Access Control appliance, include a server entry in the **[rtss-cluster:<cluster>]** stanza in the WebSEAL configuration file (for example, `webseald-default.conf`).

```
[rtss-cluster:cluster1]
server = 9,https://192.0.2.0:443/rtss/authz/services/AuthzService
server = 9,https://192.0.2.5:443/rtss/authz/services/AuthzService
```

Note:

- The first parameter in each entry is the priority of the server in the cluster. Set the priority of your servers as appropriate to your environment. Using a priority of 9 for all servers evenly distributes the load and switches between the available appliances.
 - The second parameter is a well-formed Uniform Resource Locator (URL) for the runtime security services on the appliance. Use the IP address of the application interface on the Advanced Access Control-activated appliance.
2. Use the **pdadmin** utility to add extra servers to the junction.

```
pdadmin sec_master> server task default-webseald-test.example.com add -h
192.0.2.0 -p 443 /mga
pdadmin sec_master> server task default-webseald-test.example.com add -h
192.0.2.5 -p 443 /mga
```

Note:

- You must replace all example values in these commands with values that are appropriate to your environment.
 - The first parameter in this server task command is the fully qualified name of the WebSEAL server. For example, `default-webseald-test.example.com`.
 - The `-h` option specifies the appliance that you want to add to the junction. Use the IP address of the application interface on the target appliance.
 - The **isamcfg** tool creates an SSL junction by default. Therefore, when you are adding servers to this junction, use the SSL port number 443.
 - By default, the **isamcfg** tool creates a junction that is called `/mga`. This default value is used in the example commands.
3. For secure communication between WebSEAL and the appliance, use trusted certificates. WebSEAL must trust the certificates that are presented by the appliance. To establish this trust, you can use a common certificate authority (CA) that is trusted in your environment or you can configure WebSEAL to trust each individual certificate.
- Similarly, for client certificate authentication, the Advanced Access Control appliance must trust the certificates that are presented by WebSEAL.
4. To configure failover between junctioned servers, set the **use-new-stateful-on-error** stanza entry to `yes` for the stateful junction to the appliance. That is, update the **use-new-stateful-on-error** entry in the **[junction:/mga]** stanza in the WebSEAL configuration file. Where `/mga` is the name of the junction. The **isamcfg** tool creates a junction that is called `/mga` by default, but this name is configurable.
- If a stateful junction becomes unavailable when this value is set to `yes`, WebSEAL fails over to a different server. For example, if the stateful junction to `AAC_1` in Figure 1 on page 94 becomes unavailable, WebSEAL fails over to `AAC_2`.

isamcfg reference

Use the **isamcfg** tool to configure WebSEAL and Security Access Manager appliance servers. You configure a point of contact and policy enforcement point.

The appliance you configure can be one of the following product versions:

- Security Access Manager 9.*
- Security Access Manager for Web 8.*
- Security Web Gateway 7.*

isamcfg command line reference:

Use the command line options described in this section to configure and unconfigure WebSEAL and Security Access Manager appliance servers.

Syntax

```
java -jar isamcfg.jar -action mode options
```

Description

The configuration tool has two modes of operation:

- **config**
- **unconfig**

Each mode uses different command line options.

Options

-action config options

This command configures a WebSEAL or Secure Access Manager appliance server. This mode uses different command line options:

-cfgfile *file*

Specifies which WebSEAL configuration file to use. This option is required when configuring a WebSEAL server.

-cfgrl *URL*

Specifies the appliance configuration URL to use. This option is required if configuring a Security Access Manager appliance.

-rspfile *file*

Specifies the response file for a configuration that is not interactive.

Default value: Interactive configuration.

-record

Generates a response file and make some necessary changes to WebSEAL configuration file but does not modify any other Security Access Manager appliance configuration.

-sslfactory

Specifies the secure socket connection factory to use.

When the Security Access Manager environment is enabled for NIST SP800-131a Strict mode, the only supported factory type is TLSv1.2.

If the parameter is not specified, the factory default is TLS.

-action unconfig options

This command unconfigures a WebSEAL or Security Access Manager appliance server. This mode uses different command line options:

-cfgfile *file*

Specifies which WebSEAL configuration file to use. This option is required when unconfiguring a WebSEAL server.

-cfgrl *URL*

Specifies the Security Access Manager appliance configuration URL to use. This option is required when unconfiguring a Security Access Manager appliance.

-rspfile *file*

Specifies the response file for a configuration that is not interactive.

Default value: Interactive configuration.

-record

Generates a response file and make some necessary changes to WebSEAL configuration file but does not modify any other Security Access Manager appliance configuration.

-sslfactory

Specifies the secure socket connection factory to use.

When the IBM Security Access Manager environment is enabled for NIST SP800-131a Strict mode, the only supported factory type is TLSv1.2.

If the parameter is not specified, the factory default is TLS.

Example

```
java -jar isamcfg.jar -action -config -cfgfile webseald.conf
```

The log files for the **isamcfg** tool are written to the system temporary directory. The system temporary file directory is specified by the system property **java.io.tmpdir**.

isamcfg Security Access Manager appliance configuration worksheet:

Use the worksheet for the isamcfg command-line tool to collect the information you need about the configuration properties before you run the tool.

Description of properties

Note: If you are upgrading the Advanced Access Control module, see the installation and configuration instructions.

Select/deselect the capabilities you would like to configure by typing its number.

By default, the tool selects context-based authorization, authentication service, and API protection. You can configure all of them at the same time. If you do not want to configure them all, clear the capability that you do not want to configure by selecting its corresponding number.

Context-based Authorization

Configure this capability if your environment requires the use of behavioral and contextual data analytics to calculate the risk of a transaction.

Authentication service

Configure this capability if your environment requires the use of a step-up authentication type of authentication.

API Protection

Configure this capability if your environment requires the use of an OAuth authentication type to protect your Application Programming Interface (API).

Advanced Access Control Local Management Interface hostname

Enter the Local Management Interface hostname or IP address.

Advanced Access Control Local Management Interface port

Specify the port number of the Local Management Interface. The tool displays a port number.

Example value: 443

Press Enter to use the displayed port or enter your preferred port.

Advanced Access Control administrator user ID

Press Enter to use the displayed user ID or enter your preferred user ID.

Advanced Access Control administrator password

Enter the corresponding administrator password.

SSL certificate data valid (y/n)

Press **y** to validate that the displayed SSL certificate values are valid otherwise, press **n**.

Security Access Manager Appliance Local Management Interface hostname

Enter the Security Access Manager Appliance Local Management Interface hostname or IP address. The tool might display a value. Press Enter to use the displayed value or enter your preferred hostname or IP address.

Security Access Manager Appliance Local Management Interface port

Specify the port number of the Local Management Interface port. The tool displays a port number.

Example value: 443

Press Enter to use the port or enter your preferred port.

Security Access Manager Appliance administrator user ID

Press Enter to use the user ID or enter your preferred user ID.

Security Access Manager Appliance administrator password

Enter the corresponding administrator password.

SSL certificate data valid (y/n)

Press **y** to validated that the displayed SSL certificate values are valid otherwise, press **n**.

Instance to configure

The tool displays the available instances that you can configure in a list. Select the instance that you would like to configure.

Security Access Manager administrator user ID

Press Enter to use the displayed user ID or enter your preferred user ID.

Security Access Manager administrator password

Enter the corresponding administrator password.

Security Access Manager domain name [Default]:

Enter the corresponding domain name.

Advanced Access Control runtime listening interface hostname

Enter the hostname or IP address of the runtime listening interface for the appliance that has Advanced Access Control activated.

Example value: 172.16.229.10

Advanced Access Control runtime listening interface port

Specify the port number of the runtime listening interface for the appliance that has Advanced Access Control activated.

Example value: 443

Select the method for authentication between WebSEAL and the Advanced Access Control runtime listening interface

Certificate authentication

Use a certificate to authenticate between WebSEAL and the Advanced Access Control runtime listening interface.

User ID and password authentication

Use credentials to authenticate between WebSEAL and the Advanced Access Control runtime listening interface.

The default username is easuser and the default password is passwd.

Advanced Access Control runtime listening interface user ID:

Press Enter to use the displayed user ID or enter your preferred user ID.

Advanced Access Control runtime listening interface password:

Enter the corresponding Advanced Access Control runtime listening interface password.

SSL certificate data valid (y/n):

Press **y** to validated that the displayed SSL certificate values are valid otherwise, press **n**.

Automatically add CA certificate to the key database (y/n)

Press **y** if you want to automatically add the CA certificate to the key database, otherwise press **n**.

Note: Web Reverse Proxy instance restarts if **y** is selected.

The CA certificate already exists in the key database. Replace the CA certificate? (y/n)

Press **y** if you want to automatically replace the CA certificate to the key database, otherwise press **n**.

The following files are available on the Security Access Manager Appliance.

Choose one file for the following pages:

- The **400 Bad Request** response page. The default page is `oauth_template_rsp_400_bad_request.html`.
- The **401 Unauthorized** response page. The default page is `oauth_template_rsp_401_unauthorized.html`.
- The **502 Bad Gateway** response page. The default page is `oauth_template_rsp_502_bad_gateway.html`.

If you are not running the `isamcfg` tool on the appliance, you can choose **Cancel** to upload a local file.

If you are running the `isamcfg` tool on the appliance, you must upload your custom response file. Upload the file to the Security Access Manager appliance first before you run the `isamcfg` tool so that the file is displayed as an option. See “Uploading OAuth response files” on page 144.

The junction mga contains endpoints that require Authorization HTTP header to be forwarded to the backend server. Do you want to enable this feature? [y|n]?

Press **y** to allow endpoints that require Authorization HTTP header to be forwarded to the backend server. Otherwise, press **n**.

isamcfg WebSEAL configuration worksheet:

Use the worksheet for the `isamcfg` command-line tool to collect the information you need about the configuration properties before you run the tool.

Description of properties

Note: If you are upgrading the Advanced Access Control module, see the installation and configuration instructions.

Select/deselect the capabilities you would like to configure by typing its number.

By default, the tool selects context-based authorization, authentication service, and API protection. You can configure all of them at the same time. If you do not want to configure them all, clear the capability that you do not want to configure by selecting its corresponding number.

Context-based Authorization

Configure this capability if your environment requires the use of behavioral and contextual data analytics to calculate the risk of a transaction.

Authentication service

Configure this capability if your environment requires the use of a step-up authentication type of authentication.

API Protection

Configure this capability if your environment requires the use of an OAuth authentication type to protect your Application Programming Interface (API).

Advanced Access Control Local Management Interface hostname

Enter the Local Management Interface hostname or IP address.

Advanced Access Control Local Management Interface port

Specify the port number of the Local Management Interface. The tool displays a port number.

Example value: 443

Press Enter to use the displayed port or enter your preferred port.

Advanced Access Control Appliance administrator user ID

Press Enter to use the displayed user ID or enter your preferred user ID.

Advanced Access Control Appliance administrator password

Enter the corresponding administrator password.

Security Access Manager Domain name

Enter the Security Access Manager domain name. Press Enter to use the default domain name or enter your preferred domain name.

Security Access Manager administrator user ID

Enter a valid Security Access Manager administrator user ID. Press Enter to use the user ID or enter your preferred user ID.

Security Access Manager administrator password

Enter the corresponding Security Access Manager administrator password.

Advanced Access Control runtime listening interface hostname

Enter the hostname or IP address of the runtime listening interface for the appliance that has Advanced Access Control activated.

Example value: 172.16.229.10

Advanced Access Control runtime listening interface port

Specify the port number of the runtime listening interface for the appliance that has Advanced Access Control activated.

Example value: 443

Advanced Access Control runtime listening interface SSL key file

Specify the path to the keystore that contains the SSL keys that are required to connect to the Advanced Access Control runtime listening interface. Press Enter to use the default key file.

Advanced Access Control runtime listening interface SSL stash file

Specify the path to the stash file that contains the password to the Advanced Access Control runtime listening interface SSL keyfile. Press Enter to use the default stash file.

Select the method for authentication between WebSEAL and the Advanced Access Control runtime listening interface

Certificate authentication

Use a certificate to authenticate between WebSEAL and the Advanced Access Control runtime listening interface.

Note: On Windows operating systems, you must use certificate authentication for WebSEAL from IBM Security Access Manager for Web 7.0.0.2.

User ID and password authentication

Use credentials to authenticate between WebSEAL and the Advanced Access Control runtime listening interface.

The default username is easuser and the default password is passw0rd.

Advanced Access Control runtime listening interface SSL key file label

Specify the key label of the certificate to present to Advanced Access Control at run time.

SSL certificate data valid (y/n)

Press **y** to validated that the displayed SSL certificate values are valid otherwise, press **n**.

Automatically add CA certificate to the key database (y/n)

Press **y** if you want to automatically add the CA certificate to the key database, otherwise press **n**.

Note: Web Reverse Proxy instance restarts if **y** is selected.

The CA certificate already exists in the key database. Replace the CA certificate? (y/n)

Press **y** if you want to automatically replace the CA certificate to the key database, otherwise press **n**.

Runtime security service external authorization service library

By default, the tool displays the available library. Press Enter to use the available library or enter your preferred library.

The 400 Bad Request response page:

Choose one for the **400 Bad Request** response page. The default page is `oauth_template_rsp_400_bad_request.html`.

The following files are available on the Secure Access Manager Appliance.

Choose one file for the following pages:

- The **400 Bad Request** response page. The default page is `oauth_template_rsp_400_bad_request.html`.

- The **401 Unauthorized** response page. The default page is `oauth_template_rsp_401_unauthorized.html`.
- The **502 Bad Gateway** response page. The default page is `oauth_template_rsp_502_bad_gateway.html`.

Using a response file

Create and use a response file with the **isamcfg** tool to configure WebSEAL and IBM Security Access Manager for Web 7.0.

A response file records the actions to be taken by the **isamcfg** tool.

Use the **-record** command to create a response file without changing the configuration. For example:

```
/usr/lib/jvm/jre-1.7.0-ibm.x86_64/bin/java -jar
/opt/IBM/FIM/tools/isamcfg/isamcfg.jar -action config
-cfgurl https://1.1.1.1/ -record
```

You can then use the response file to run the **isamcfg** tool non-interactively. Use the **-rspfile** command to run the **isamcfg** tool with a response file. For example:

```
/usr/lib/jvm/jre-1.7.0-ibm.x86_64/bin/java -jar
/opt/IBM/FIM/tools/isamcfg/isamcfg.jar -action config
-cfgurl https://1.1.1.1/ -rspfile /tmp/isamcfg-access-control.properties
```

The contents of a response varies depending on your configuration, for example:

```
#Fri Sep 06 12:45:28 EST 2013
webseal.addcacert=y
tam.admin=sec_master
wga.pass=
pop.replace=pop.reuse
mga.admin.ssl.shalfingerprint=CD\1C\F0\D9\A6\3A\7A\11\
:16\CC\18\CB\56\02\08\E2\53\99\83\3B
mga.runtime.auth.mode=mga.runtime.auth.ba
mga.runtime.ssl.md5fingerprint=E4\65\16\A9\D8\B2\97\
:3C\F6\13\19\77\25\8B\B0\0A
mga.admin.ssl.subjectdn=CN\=amapp800
wga.ssl.md5fingerprint=B2\9F\87\8A\D1\49\D9\A1\BA\
:03\4B\41\E9\DF\44\C7
rtss.password=
wga.ssl.shalfingerprint=CD\1C\F0\D9\A6\3A\7A\11\16\
:CC\18\CB\56\02\08\E2\53\99\83\3B
wga.instance=mobile
mga.admin.user=admin
wga.port=443
isam.mode=context_based_authorization,
authentication_service,
mga.runtime.port=443
mga.admin.pass=
jct.replace=reuse
wga.host=1.1.1.1
mga.runtime.host=1.1.1.1
mga.admin.ssl.md5fingerprint=B2\9F\87\8A\D1\49\D9\
:A1\BA\03\4B\41\E9\DF\44\C7
wga.ssl.subjectdn=CN\=amapp800
mga.runtime.ssl.subjectdn=CN\=isam, O\=ibm, C\=us
mga.admin.ssl.issuerdn=CN\=amapp800
rtss.user=admin
mga.runtime.ssl.shalfingerprint=F9\38\5A\53\0A\DA\1A\
:FF\67\46\C9\58\3F\F1\2B\00\B0\6C\83\32
mga.admin.port=443
tam.password=
```

```
wga.ssl.issuerdn=CN\=amapp800  
mga.runtime.ssl.issuerdn=CN\=isam, O\=ibm, C\=us  
wga.user=admin  
mga.admin.host=1.1.1.1
```

Chapter 10. OAuth 2.0 and OIDC support

Security Access Manager supports the OAuth 2.0 protocol, including OpenID Connect.

The support is provided at both the Advanced Access Control and the Federation licensing levels.

- OAuth is an HTTP-based authorization protocol. It gives third-party applications scoped access to a protected resource on behalf of the resource owner. It gives scoped access by creating an approval interaction between the resource owner, client, and the resource server. It gives users the ability to share their private resources between sites without providing user names and passwords. Private resources can be anything, but common examples include photos, videos, and contact lists.

The implementation of OAuth 2.0 in Advanced Access Control strictly follows the OAuth 2.0 standards. For a complete description of the OAuth 2.0 specifications, see the OAuth website <http://www.oauth.net>.

The OAuth 2.0 implementation of Advanced Access Control also integrates with WebSphere DataPower. For more information, see DataPower Integration.

- OpenID Connect is an extension of the OAuth protocol to better support identity and authentication. For a complete description of the OpenID Connect specifications, see the OpenID website: <http://openid.net/specs/>

Note: Prior versions of Security Access Manager supported OIDC through federation support. Security Access Manager now supports OIDC through API Protection. Existing deployments of Security Access Manager OIDC federations are fully supported, but new OIDC deployments should use API Protection. For documentation on managing existing OIDC federations, see Legacy support for OpenID Connect federations

OAuth and OpenID Connect concepts

You can use the following topics to review the main concepts for the OAuth 2.0 protocol and for the OpenID Connect extensions to the protocol.

OAuth 2.0 concepts

This topic introduces the main concepts of OAuth 2.0.

The following concepts are generally used in OAuth 2.0.

Resource owner

An entity capable of authorizing access to a protected resource. When the resource owner is a person, it is called an *user*.

OAuth client

A third-party application that wants access to the private resources of the resource owner. The OAuth client can make protected resource requests on behalf of the resource owner after the resource owner grants it authorization. OAuth 2.0 introduces two types of clients: confidential and public. Confidential clients are registered with a client secret, while public clients are not.

OAuth server

Known as the **Authorization server** in OAuth 2.0. The server that gives OAuth clients scoped access to a protected resource on behalf of the resource owner. The server issues an access token to the OAuth client after it successfully does the following actions:

- Authenticates the resource owner.
- Validates a request or an authorization grant.
- Obtains resource owner authorization.

An authorization server can also be the resource server.

Scope A property requested by the OAuth client, to specify the scope of the access request. The scope is used by the caller to tag the intended use of the token. The authorization server can use the scope response parameter to tell the client the scope of the access token that was issued. Scopes are usually shown on the consent page, so that a user can understand the client's intended use of the token. Common scopes include profile and email.

Access token

A string that represents authorization granted to the OAuth client by the resource owner. This string represents specific scopes and durations of access. It is granted by the resource owner and enforced by the OAuth server.

Bearer token

Token issued from the token endpoint. This includes an access token and potentially a refresh token. See <http://tools.ietf.org/html/rfc6750> for more information on bearer tokens.

Protected resource

A restricted resource that can be accessed from the OAuth server using authenticated requests.

Resource server

The server that hosts the protected resources. It can use access tokens to accept and respond to protected resource requests. The resource server might be the same server as the authorization server.

Authorization grant

A grant that represents the resource owner authorization to access its protected resources. OAuth clients use an authorization grant to obtain an access token. There are four authorization grant types: authorization code, implicit, resource owner password credentials, and client credentials.

Authorization code

A code that the Authorization server generates when the resource owner authorizes a request.

Refresh token

A string that is used to obtain a new access token.

A refresh token is optionally issued by the authorization server to the OAuth client together with an access token. The OAuth client can use the refresh token to request another access token that is based on the same authorization, without involving the resource owner again.

OpenID Connect concepts

OpenID Connect extends OAuth 2.0 function. The OpenID Connect concepts include the OAuth 2.0 concepts.

OpenID Connect Provider (OP)

OAuth 2.0 Authorization Server that can authenticate the user and providing claims to a Relying Party about the authentication event and the user.

Relying Party (RP)

OAuth 2.0 Client application that requires user authentication and claims from an OpenID Connect Provider.

Claim

Piece of information asserted about an entity that is included in the ID token. An OpenID Connect Provider must document which claims it includes in its ID tokens.

The following claims are required claims about the authentication event:

- **aud** (Audience): Must contain the client identifier of the RP registered at the issuer.
- **iss** (Issuer): The issuer identifier of the OP.
- **exp** (Expiration time): The RP must validate the ID token before this time.
- **iat** (Issued at): The time at which the ID token was issued.

The following claims are required claims about the user:

- **sub** (Subject): A locally unique and permanent (never reassigned) identifier of the user at the issuer.

Optional claims about the user can include `first_name`, `last_name`, `picture`, `gender`, etc.

Scope A property that is requested by the Relying Party, which can be consented to by the user, that requests certain claims be included in the ID token. In addition to the definition of scope in OAuth, OpenID Connect adds some well-defined scopes. It requires the `openid` scope to identify a request to an OpenID Connect flow. It also includes the common scopes `profile` and `email`, which pertain to a specific set of claims.

Bearer token

In addition to the types of tokens that are listed in the description of Bearer token for OAuth 2.0 support, for OpenID Connect the token can be an ID token.

ID token

JSON Web Token (JWT) that contains claims about the authentication event and the user.

JWTs are Base64 encoded JSON objects with three sections: Header, Claims Set, and JSON Web Signature (JWS). The sections are separated in the JWT by a period ('.'). The Header must at least contain the algorithm that is used to sign the JWT (the `alg` claim).

The Claims Set includes claims about the authentication event and the user.

The JSON Web Signature (JWS) is used to verify the signing of the JWT. For more information, see RFC7515.

For more information about JWTs, see RFC7519.

Issuer Entity that issues a set of claims.

Issuer identifier

Verifiable identifier for an issuer. An issuer identifier is a case-sensitive

URL that uses the HTTPS scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.

Hybrid flow

The OpenID Connect hybrid flow is a request to /authorize, where both an authorization code and either an access token or id_token, or both, are returned. The value of response_type for a hybrid flow is any of the following values.

- code id_token
- code id_token token
- code token

Some tokens are returned by the authorization endpoint, and others are returned by the token endpoint.

Note: Hybrid flow is supported in OpenID Connect but not in OAuth. See http://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth.

Metadata

Metadata is the discovery information that the OpenID Provider (OP) exposes. If metadata is configured, the Relying Party (RP) uses it as the source of the /authorize, /token, /jwks, and /userinfo URLs for the RP. See http://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata.

Userinfo

The Userinfo endpoint is an OAuth 2.0 protected resource that returns claims about the authenticated user. These claims are normally represented by a JSON object that contains a collection of name and value pairs for each claim. For more information, see http://openid.net/specs/openid-connect-core-1_0.html#UserInfo.

OAuth 2.0 endpoints

Endpoints provide OAuth clients the ability to communicate with the OAuth server or authorization server within a definition.

All endpoints can be accessed through URLs. The syntax of the URLs is specific to the purpose of the access.

If you are responsible for installing and configuring the appliance, you might find it helpful to be familiar with these endpoints and URLs.

API protection definitions

The API protection definitions naming follows the standard Advanced Access Control naming convention. The syntax is:

`https://<hostname:port>/<junction>/sps/oauth/oauth20`

For example:

`https://server.oauth.com/mga/sps/oauth/oauth20`

The following table describes the endpoints that are used in an API protection definition.

Notes:

- There is only a single set of endpoints.
- Not all authorization grant types use all three endpoints in a single OAuth 2.0 flow.

Table 5. OAuth 2.0 endpoint definitions and URLs

Endpoint name	Description	Example
Authorization endpoint	An authorization URL where the resource owner grants authorization to the OAuth client to access the protected resource.	https://server.oauth.com/mga/sps/oauth/oauth20/authorize
Token endpoint	A token request URL where the OAuth client exchanges an authorization grant for an access token and an optional refresh token.	https://server.oauth.com/mga/sps/oauth/oauth20/token
Clients manager endpoint	A URL for resource owners to manage their trusted clients. The resource owner can use the clients manager endpoint to access and modify the list of clients that are authorized to access the protected resource. The trusted clients manager shows the client name and permitted scope of an authorized client. Note: The list does not show clients that are disabled or deleted from the definition. The resource owner can optionally remove trusted client information from the list. In doing so, the resource owner is prompted for consent to authorize the next time the OAuth client attempts to access the protected resource.	https://server.oauth.com/mga/sps/oauth/oauth20/clients
Session endpoint	A URL where an access_token can be exchanged for a web session. The client uses the endpoint to obtain an authenticated web session for the resource owner that is typically used in hybrid mobile application scenarios. Note: The session endpoint is disabled by default and can be enabled by using advanced configuration. The client must send a POST request with the access_token in the body. POST /mga/sps/oauth/oauth20/session HTTP/1.1Host: server.oauth.com Content-Type: application/x-www-form-urlencoded access_token=abc123...	https://server.oauth.com/mga/sps/oauth/oauth20/session
Authorization grant management endpoint	A URL where you can view your authorization grants and the tokens and attributes of each authorization grant.	http://server.oauth.com/mga/sps/mga/user/mgmt/html/device/device_selection.html
Logout endpoint	A URL where you can end a session by revoking an access_token. The token must be provided in the Authorization header or a session cookie must be used.	http://server.oauth.com/mga/sps/oauth/oauth20/logout

Table 5. OAuth 2.0 endpoint definitions and URLs (continued)

Endpoint name	Description	Example
Introspect endpoint	A URL where an access_token can be inspected by an oauth_client. For more details, see "OAuth introspection" on page 144. Note: The introspect endpoint is disabled by default and can be enabled by using the advanced configuration.	https://server.oauth.com/mga/sps/oauth/oauth20/introspect
Revocation endpoint	A URL where you can revoke OAuth tokens issued to a client. For more details, see "OAuth revocation endpoint" on page 146.	https://server.oauth.com/mga/sps/oauth/oauth20/revoke
Metadata endpoint	Final portion of URL is a path parameter that is the name of your API Protection definition. Template file available: <locale>/oauth20/metadata.json If a custom template is needed per definition use: <Locale>/oauth20/<Your_API_Definition_Name>/metadata.json Example: { "issuer": "https://mywebseal.com", "authorization_endpoint": "https://mywebseal.com/sps/oauth/oauth20/authorize", "token_endpoint": "https://mywebseal.com/sps/oauth/oauth20/token", "userinfo_endpoint": "https://mywebseal.com/sps/oauth/oauth20/userinfo", "jwks_uri": "http://mywebseal.com/sps/oauth/oauth20/jwks/testDef", "response_types_supported": ["token", "id_token", "token id_token", "code"], "response_modes_supported": ["fragment", "form_post"], "grant_types_supported": ["implicit", "password", "authorization_code"], "id_token_signing_alg_values_supported": ["RS256"], "introspect_endpoint": "https://mywebseal.com/sps/oauth/oauth20/introspect", "revocation_endpoint": "https://mywebseal.com/sps/oauth/oauth20/revoke"} }	https://server.oauth.com/mga/sps/oauth/oauth20/metadata/<Definition_Name>
Userinfo Endpoint	The Userinfo endpoint is an OAuth 2.0 protected resource that returns claims about the authenticated end-user. These claims are normally represented by a JSON object that contains a collection of name and value pairs for each claim. For more info, see http://openid.net/specs/openid-connect-core-1_0.html#UserInfo	https://server.oauth.com/mga/sps/oauth/oauth20/userinfo
JWKS Uri	The URL of the JSON Web Key (JWK) Set document for the OpenID Provider. This data contains the signing key (or keys) that the Relying Party uses to validate signatures from the OpenID Provider. Optionally, the JWK Set can contain the Server's encryption key (or keys), which Relying Parties use to encrypt requests to the Server.	https://server.oauth.com/mga/sps/oauth/oauth20/jwks/<Definition_Name>
Client Registration Endpoint	The Client Registration Endpoint where an application can request a clientId in order to make OAuth/OIDC requests. This is also the endpoint to retrieve a registered client's definition, or delete it.	https://server.oauth.com/mga/sps/oauth/oauth20/register/<Definition_Name>

Table 5. OAuth 2.0 endpoint definitions and URLs (continued)

Endpoint name	Description	Example
Device Authorize Endpoint	Endpoint initially visited by the device client to obtain a device code and user code.	<code>https://server.oauth.com/mga/sps/oauth/oauth20/device_authorize</code>
User Authorize Endpoint	Endpoint visited by a user to verify a user_code so a device client may obtain an authorization grant for the user.	<code>https://server.oauth.com/mga/sps/oauth/oauth20/user_authorize</code>
Authorization grants management API endpoint	An API to list all of a user's grants.	<code>https://server.oauth.com/mga/sps/mga/user/mgmt/grant</code>
Authorization grant management API endpoint	An API to retrieve a specific grant based on a grant ID. This API can also be used to delete a grant.	<code>https://server.oauth.com/mga/sps/mga/user/mgmt/grant/{grantId}</code>

OAuth 2.0 and OIDC workflows

The OAuth 2.0 support in IBM Security Access Manager provides four different ways for an OAuth client to obtain access the protected resource.

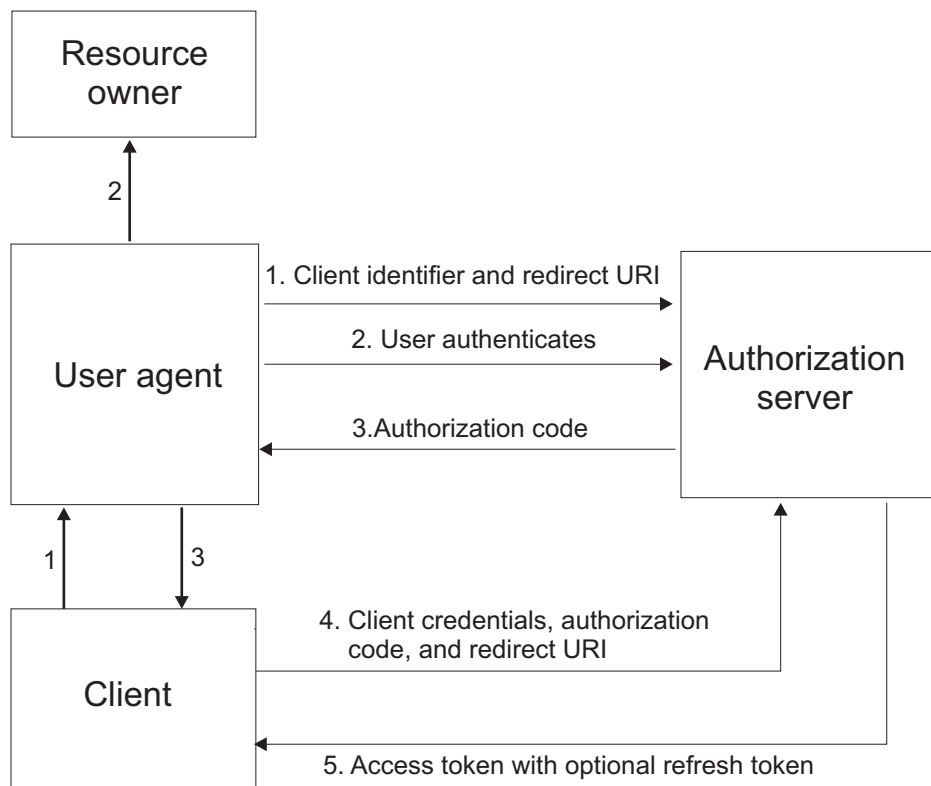
OAuth 2.0 workflow

Advanced Access Control supports the following OAuth 2.0 workflows.

Authorization code flow

The authorization code grant type is suitable for OAuth clients that can keep their client credentials confidential when authenticating with the authorization server. For example, a client implemented on a secure server. As a redirection-based flow, the OAuth client must be able to interact with the user agent of the resource owner. It also must be able to receive incoming requests through redirection from the authorization server.

Note: For OIDC, a Relying Party is an OAuth Client, and an OIDC Provider is an OAuth Authorization server. For OIDC, the authorization code flow returns an authorization code to the Relying Party, which can then directly exchange it for an ID token and access token. This mechanism provides the benefit of not exposing any tokens to the browser or end-user. The OpenID Connect Provider also authenticates the Relying Party before exchanging the authorization code for an access token. The authorization code flow is suitable for Relying Parties that can securely maintain a client secret between themselves and the OpenID Connect Provider.

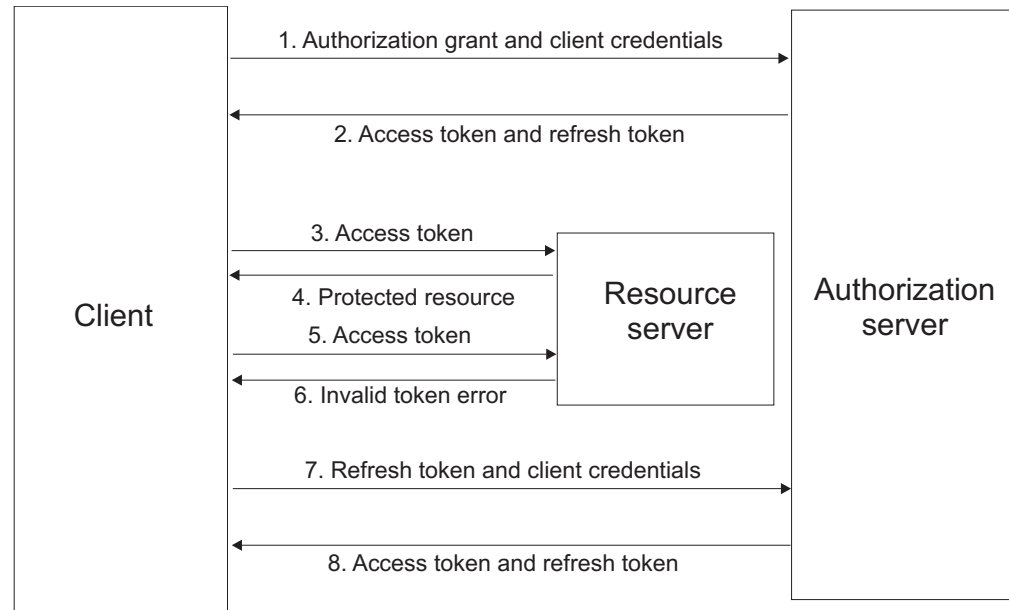


1. The OAuth client initiates the flow when it directs the user agent of the resource owner to the authorization endpoint. The OAuth client includes its client identifier, requested scope, local state, and a redirection URI. The authorization server sends the user agent back to the redirection URI after access is granted or denied.
For OIDC, the scope must include openid. The state parameter must also be included.
2. The authorization server authenticates the resource owner through the user agent and establishes whether the resource owner grants or denies the access request.
3. If the resource owner grants access, the OAuth client uses the redirection URI provided earlier to redirect the user agent back to the OAuth client. The redirection URI includes an authorization code and any local state previously provided by the OAuth client.
If this is an OpenID Connect request, then the redirection URI must be present. For OAuth, by contrast, it can be sourced from the client configuration. This requirement exists because OpenID Connect is stricter on request validation.
4. The OAuth client requests an access token from the authorization server through the token endpoint. The OAuth client authenticates with its client credentials and includes the authorization code received in the previous step. The OAuth client also includes the redirection URI used to obtain the authorization code for verification.
For OIDC, the Relying Party requests an access token and, in addition, an ID token from the OpenID Connect Provider through the token endpoint.
5. The authorization server validates the client credentials and the authorization code. The server also ensures that the redirection URI received matches the URI used to redirect the client in Step 3. If valid, the authorization server responds back with an access token. If OIDC is configured, an id token is returned.

For OIDC, if the redirection URI is valid, the OpenID Connect Provider responds back with an access token and an ID token.

The authorization server can be the same server as the resource server or a separate entity. A single authorization server can issue access tokens accepted by multiple resource servers.

Authorization code flow with refresh token



The authorization code workflow with refresh token diagram involves the following steps:

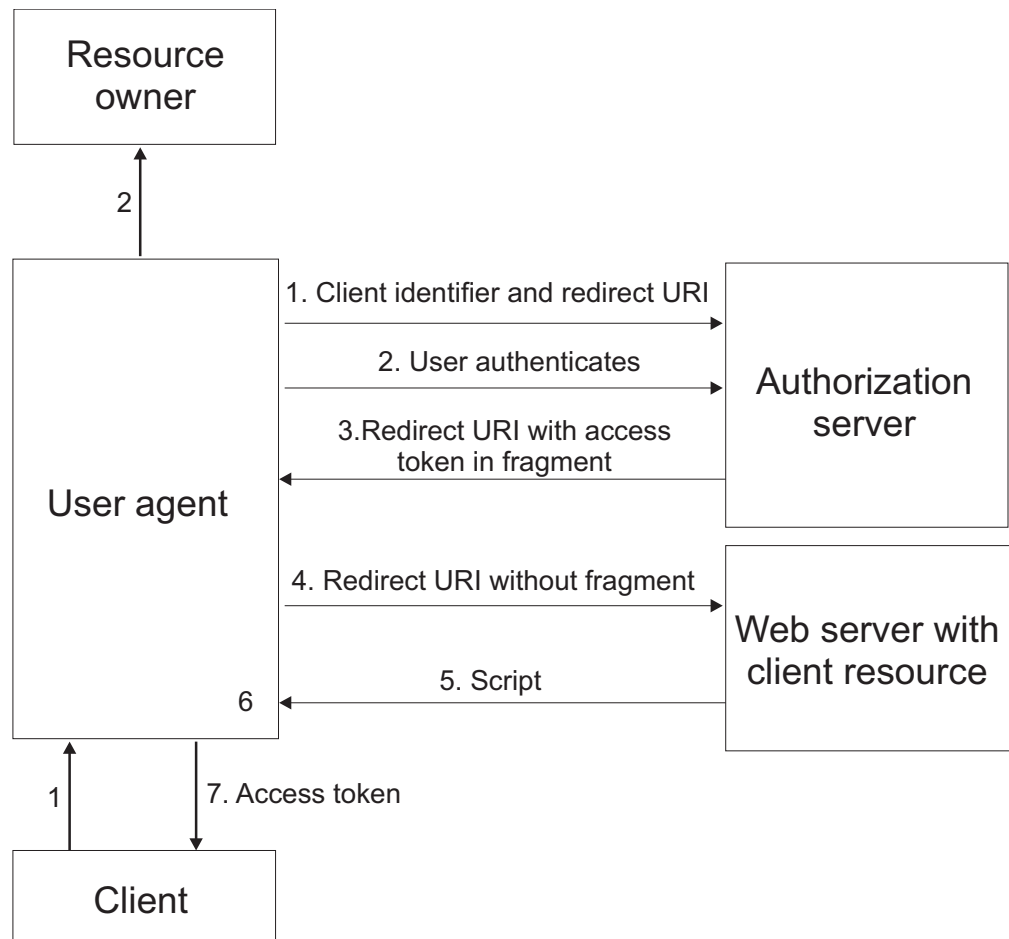
1. The OAuth client requests an access token by authenticating with the authorization server with its client credentials, and presenting an authorization grant.
2. The authorization server validates the client credentials and the authorization grant. If valid, the authorization server issues an access token and a refresh token.
3. The OAuth client makes a protected resource request to the resource server by presenting the access token.
4. The resource server validates the access token. If the access token is valid, the resource owner serves the request.
5. Repeat steps 3 and 4 until the access token expires. If the OAuth client knows that the access token has expired, skip to Step 7. Otherwise, the OAuth client makes another protected resource request.
6. If access token is not valid, the resource server returns an error.
7. The OAuth client requests a new access token by authenticating with the authorization server with its client credentials, and presenting the refresh token.
8. The authorization server validates the client credentials and the refresh token, and if valid, issues a new access token and a new refresh token. For OpenID Connect, an ID token is returned in addition to the new access token and refresh token.

Implicit grant flow

The implicit grant type is suitable for clients that are not capable of maintaining their client credentials confidential for authenticating with the authorization server. An example can be in the form of client applications that are in a user agent, typically implemented in a browser using a scripting language such as JavaScript.

As a redirection-based flow, the OAuth client must be able to interact with the user agent of the resource owner, typically a web browser. The OAuth client must also be able to receive incoming requests through redirection from the authorization server.

Note: For OIDC, a Relying Party is an OAuth Client, and an OIDC Provider is an OAuth Authorization server. For OIDC, the implicit flow can be used by Relying Parties with an in-browser scripting language component. The access token and ID token are returned directly to the Relying Party, which may expose them to the end-user and applications that have access to the end-user's browser. The token endpoint is not used and the OpenID Connect Provider does not perform authentication on the Relying Party in this flow. The Relying Party does not have to directly communicate with the OpenID Connect Provider as all interactions can be performed through the browser.



The implicit grant workflow diagram involves the following steps:

1. The OAuth client initiates the flow by directing the user agent of the resource owner to the authorization endpoint. The OAuth client includes its client

identifier, requested scope, local state, and a redirection URI. The authorization server sends the user agent back to the redirection URI after access is granted or denied.

For OIDC, the requested scope must include `openid`, and the parameters `nonce` and `state` must also be included.

2. The authorization server authenticates the resource owner through the user agent and establishes whether the resource owner grants or denies the access request.
3. If the resource owner grants access, the authorization server redirects the user agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

For OIDC, the redirection URI includes both the access token and the ID token in the URI fragment.

4. The user agent follows the redirection instructions by making a request to the web server without the fragment. The user agent retains the fragment information locally.
5. The web server returns a web page, which is typically an HTML document with an embedded script. The web page accesses the full redirection URI including the fragment retained by the user agent. It can also extract the access token and other parameters contained in the fragment.
6. For OAuth 2.0, the user agent runs the script provided by the web server locally, which extracts the access token and passes it to the client.

For OIDC, the script extracts both the access token and the ID token. If `response_mode=form_post` is specified, the OpenID Provider returns a self-posting form.

Note: When the response type includes token or ID token, the parameter `response_mode=form_post` is set by default. You can use advanced mapping to change or remove this parameter if necessary. However, by default most authorization servers ignore this parameter if they do not support it.

Hybrid flow

OpenID Connect supports a hybrid flow. In the OAuth 2.0 hybrid flow, an authorization code (`response_type = code`) or an access token (`response_type = token`) is returned by the authorization endpoint. Some tokens are returned by the authorization endpoint, and others are returned by the token endpoint.

The hybrid flow is similar to authorization code flow in allowing clients to be authenticated, and in supporting refresh tokens. The hybrid flow is similar to implicit grant flow in allowing tokens to be revealed to the user agent.

The hybrid flow supports multiple `response_type` values.

Table 6. Response type values for each flow

response_type value	Flow
code	Authorization Code Flow
id_token	Implicit Flow
id_token token	Implicit Flow
code id_token	Hybrid Flow
code token	Hybrid Flow
code id_token token	Hybrid Flow

The hybrid flow uses the steps shown in the diagram for authorization code.

1. The OAuth client initiates the flow when it directs the user agent of the resource owner to the authorization endpoint. The OAuth client includes its client identifier, requested scope, local state, and a redirection URI. The authorization server sends the user agent back to the redirection URI after access is granted or denied.

For OIDC, the scope must include openid. The state parameter must also be included.

For hybrid flow, the response type is more than just code. For example, `response_type = code + id_token` will result in the return of both an `id_token` and code in step 3.

2. The authorization server authenticates the resource owner through the user agent and establishes whether the resource owner grants or denies the access request.
3. If the resource owner grants access, the OAuth client uses the redirection URI provided earlier to redirect the user agent back to the OAuth client. The redirection URI includes an authorization code and any local state previously provided by the OAuth client.

If this is an OpenID Connect request, then the redirection URI must be present. For OAuth, by contrast, it can be sourced from the client configuration. This requirement exists because OpenID Connect is stricter on request validation.

For hybrid flow, when the authorization code is returned, an `id_token` and (or) an access token may also be returned.

4. The OAuth client requests an access token from the authorization server through the token endpoint. The OAuth client authenticates with its client credentials and includes the authorization code received in the previous step. The OAuth client also includes the redirection URI used to obtain the authorization code for verification.

For OIDC, the Relying Party requests an access token and, in addition, an ID token from the OpenID Connect Provider through the token endpoint.

5. The authorization server validates the client credentials and the authorization code. The server also ensures that the redirection URI received matches the URI used to redirect the client in Step 3. If valid, the authorization server responds back with an access token. If OIDC is configured, an ID token is returned.

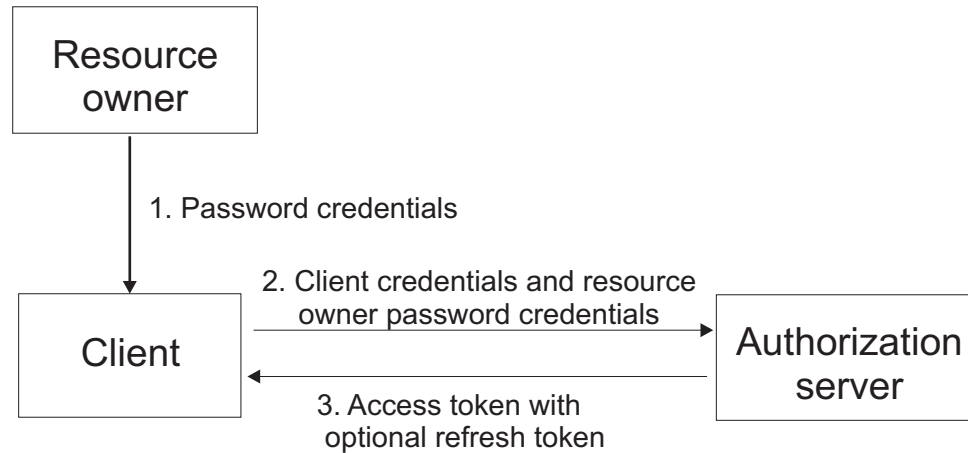
For OIDC, if the redirection URI is valid, the OpenID Connect Provider responds back with an access token and an ID token.

For more information on the hybrid flow, see http://openid.net/specs/openid-connect-core-1_0.html#HybridFlowSteps.

Resource owner password credentials flow

The resource owner password credentials grant type is suitable in cases where the resource owner has a trust relationship with the client. For example, the resource owner can be a computer operating system of the OAuth client or a highly privileged application.

You can only use this grant type when the OAuth client has obtained the credentials of the resource owner. It is also used to migrate existing clients using direct authentication schemes by converting the stored credentials to an access token.



The resource owner password credentials workflow diagram involves the following steps:

1. The resource owner provides the client with its user name and password.
2. The OAuth client requests an access token from the authorization server through the token endpoint. The OAuth client authenticates with its client credentials and includes the credentials received from the resource owner.
3. After the authorization server validates the resource owner credentials and the client credentials, it issues an access token and optionally a refresh token.

OIDC requests can return, from a request to `/authorize`, any combination of the following:

- An access token
- An `id_token`
- A code

If a code is returned, then the following can be returned to `/token`:

- An access token
- An ID token
- A refresh token, if enabled

If a refresh token is presented to `/token`, then following are returned:

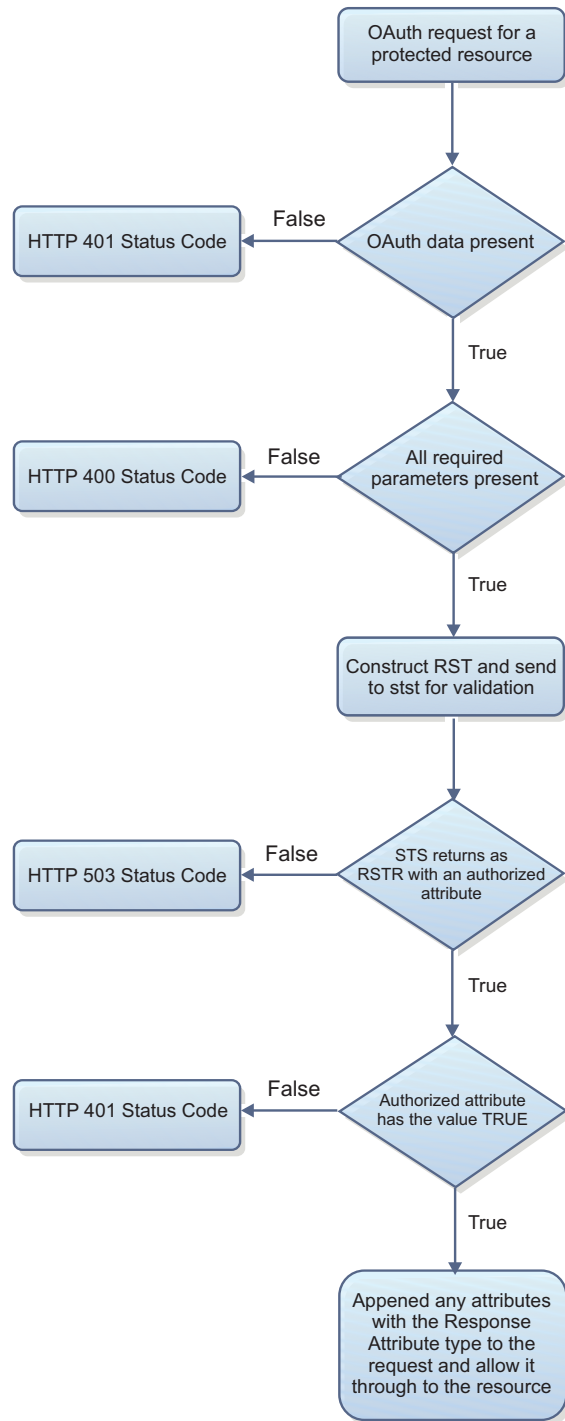
- An access token
- An ID token
- A refresh token

JWT and SAML bearer grant type flows

The assertion bearer grant types are an extension to the OAuth 2.0 framework. In such flows, a client presents a JWT or SAML assertion to the token endpoint in exchange for tokens. The assertion that is presented must represent the resource owner for whom tokens will be issued to. See RFC 7522 and RFC 7523 for further details.

The assertions must be validated in the pre-token mapping rule. A callout to the STS is one way to validate a presented assertion.

The following diagram describes the steps in the assertion bearer grant type flows:

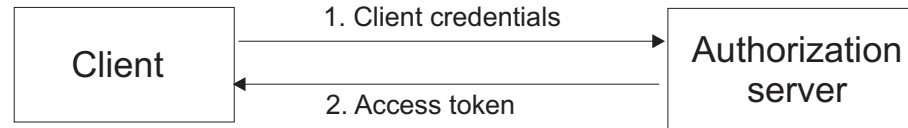


1. The client obtains a JWT or SAML assertion from the resource owner.
2. The client presents the assertion to the authorization server.
3. The authorization server validates the presented assertion through signature validation, decryption, or both. The issuer and subject are extracted and validated. If the issuer is trusted, tokens are issued to the subject that is captured in the assertion.

Client credentials flow

The client credentials flow is used when the OAuth client requests an access token using only its client credentials. This flow is applicable in one of the following situations:

- The OAuth client is requesting access to the protected resources under its control.
- The OAuth client is requesting access to a different protected resource, where authorization has been previously arranged with the authorization server.



The client credentials workflow diagram involves the following steps:

1. The OAuth client requests an access token from the token endpoint by authenticating with its client credentials.
2. After the authorization server validates the client credentials, it issues an access token.

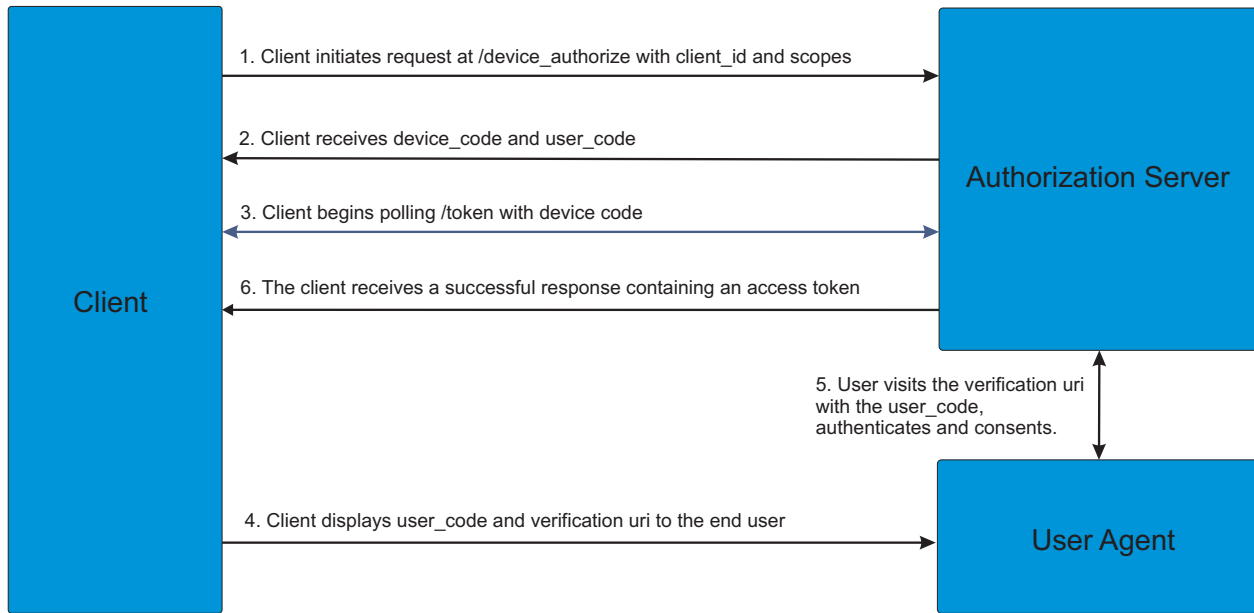
Device flow

The OAuth device flow is a draft RFC (Currently version 9, see: <https://tools.ietf.org/html/draft-ietf-oauth-device-flow-09>). The OAuth device flow is intended for use where the OAuth client is unable to provide any input mechanism to the user, and is only able to broadcast information. Such applications would be smart devices which can display (for example, a smart device plugged into a TV) content, but not provide a user-agent. This means the flow of information is one way from the client to the resource owner. There are three endpoints used in this flow, the device_authorize endpoint, a JSON endpoint used by the client to get the initial device and user codes. The user authorize endpoint, where the user visits to authenticate and authorize the client, and the token endpoint, where the client will poll with the device code.

The steps of a device flow are:

1. The client makes a request to device authorize and receives a device_code, a user_code and a verification_uri.
2. The client shows the user_code on the screen, along with the verification uri. The client may also choose to show an alternative method of consuming this user_code and verification code, such as a QR code to be scanned by a resource owner with a mobile device.
3. The client begins polling the token endpoint with the device_code, it will receive errors of 'authorization_pending' or 'slow_down' while it waits for a user to verify the user code.
4. The user visits the verification uri presenting the user_code. The user will then be prompted to authenticate and consent. After the user has authenticated and consented, a success page is shown.
5. The client, which has been continuing to poll will now receive a bearer token rather than an 'authorization_pending' error.

The following diagram describes the steps in the device flows:



Client authentication considerations at the OAuth 2.0 token endpoint

The OAuth 2.0 token endpoint is used for direct communications between an OAuth client and the authorization server. The token endpoint is used to obtain an OAuth token.

The client type, whether public or confidential, determines the authentication requirements of the OAuth 2.0 token endpoint. The Advanced Access Control runtime is responsible for authenticating the client by using the `client_id` and `client_secret` in sending the request.

OAuth 2.0 workflows for confidential clients that require client authentication at the token endpoint, can be configured in one of the following ways:

1. The Advanced Access Control point of contact requires authentication at the token endpoint:
 - The point of contact is responsible for authenticating the client.
 - The **Confidential** check box from the client instance panel is not relevant. A `client_secret` parameter must *not* be sent in the token endpoint request.
 - If a `client_id` parameter is sent in the request, it must match the identity of the client that is authenticated by the point of contact.
2. The Advanced Access Control point of contact permits unauthenticated access to the token endpoint:
 - The `client_id` parameter in the token endpoint request is used to identify the client.
 - The **Confidential** check box from the client instance panel determines whether a `client_secret` parameter is required in the token endpoint request. A client secret is required for confidential clients only.
3. Basic Authentication can be performed by the runtime instead of by the point of contact server.

Note: When enforcing client authentication at the token endpoint, the point of contact must contain the client ID and client secret within its user registry. The point of contact must be able to map the authenticated user credential to the `client_id` parameter sent in the OAuth 2.0 token endpoint request.

Based on this information, the following configurations are supported:

Table 7. Configurations supported

Client types	Configurations	WebSEAL point of contact token endpoint URI considerations	Check box setting for the Confidential parameter
Confidential clients	The point of contact performs client authentication.	- Authenticated ACL on token endpoint is required. - Token endpoint port must match WebSEAL port.	N/A
Confidential clients	Basic Authentication is performed by the runtime.	The point of contact configuration does not need to make any change to the Authorization header.	N/A
Confidential clients	The <code>client_id</code> and <code>client_secret</code> parameters in the token endpoint request are used to perform client authentication.	- Unauthenticated ACL on token endpoint is required. - Token endpoint port must match WebSEAL port.	Must be cleared.
Public clients	The <code>client_id</code> parameter is used to perform client validation.	- Unauthenticated ACL on token endpoint is required. - Token endpoint port must match the WebSEAL port.	Must be selected.

Configuring an authenticated token endpoint with WebSEAL as the point of contact

Configure an authenticated token endpoint with WebSEAL as the point of contact to delegate authentication of the client to WebSEAL. Note that basic authentication can be completed without configuration of WebSEAL as the point of contact.

Before you begin

When you want WebSEAL to do client authentication, you must attach an authenticated ACL on the token endpoint. You can use the `isam_mobile_anyauth` ACL that you can create by using the `oauth_config` REST API. See “Configuring a reverse proxy for OAuth and an OIDC Connect provider” on page 125.

You must also know how to enable Basic Authentication and Certificate Authentication. For more information, see Basic authentication and Client-side certificate authentication .

About this task

Use separate WebSEAL instances for the token and authorization endpoints to enforce authentication at WebSEAL for the token endpoint. Clients can authenticate with authentication mechanisms, such as Basic Authentication and Client Certificates. At the same time, users can authenticate by using forms authentication at the authorize and clients manager endpoints.

Procedure

1. Log in to the pdadmin utility with the sec_master account.
2. Attach the isam_mobile_anyauth ACL to the token endpoint /WebSEAL/<WebSEAL_instance_name>/mga/sps/oauth/oauth20/token. For example,

```
acl attach /WebSEAL/server-default/mga/sps/oauth/oauth20/token isam_mobile_anyauth
```
3. Enable Basic Authentication, Certificate Authentication, or both.
4. Ensure that the point of contact contains the client ID and client secret within its user registry by running the following command:

```
user list * 0
```
5. Verify the configuration:
 - a. Ensure that the token endpoint is protected. For example, run the following command and verify that you get a login form:

```
curl -kv https://server:445/mga/sps/oauth/oauth20/token
```
 - b. If you enabled Basic Authentication or Certificate authentication, ensure that you can authenticate to the point of contact with the Basic Authentication header or Client Certificate. For example, run the following commands and ensure that you can reach the token endpoint:

Basic Authentication

```
curl -kv https://server:445/mga/sps/oauth/oauth20/token  
--basic -u jHTzyi19lQAcFsJu9Dw3:CDrQllexadocQ6FwTzEUG
```

Certificate Authentication

```
curl -kv https://server:445/mga/sps/oauth/oauth20/token  
--cert /path/to/cert.pem
```

State management

The state_id parameter in the STSUniversalUser module is used as a key to store or retrieve state information for each invocation of the trust chain of an OAuth flow.

Advance Access Control provides sample mapping rules. These sample mapping rules use state management API and are applicable to OAuth 2.0 protocols. You can get the sample mapping rules from the File downloads section.

OAuth 2.0

OAuth 2.0 tokens, such as grants, access tokens, and refresh tokens, have a state_id parameter that is used in Security Token Service mapping rules. The state_id parameter maintains state between associated Security Token Service calls in an OAuth 2.0 flow.

The OAuth 2.0 mapping rule uses the state_id as the key to issue an authorization grant. The key is used to add the token storage time to a cache. The storage time is then retrieved from the cache during a request for a protected resource.

Figure 2 on page 123 shows a section of the sample JavaScript mapping rule for OAuth 2.0.

```

...
var request_type = null;
var grant_type = null;

// The request type - if none available assume 'resource'
temp_attr = stsuu.getContextAttributes().getAttributeValuesByNameAndType("request_type", "urn:ibm:names:ITFIM:oauth:request");
if (temp_attr != null && temp_attr.length > 0) {
    request_type = temp_attr[0];
} else {
    request_type = "resource";
}

// The grant type
temp_attr = stsuu.getContextAttributes().getAttributeValuesByNameAndType("grant_type", "urn:ibm:names:ITFIM:oauth:body:param");
if (temp_attr != null && temp_attr.length > 0) {
    grant_type = temp_attr[0];
}

/* The following demonstrates the use of the state management API.
 *
 * request_type = 'authorization' ==> Store the UTC time of the request into a cache
 *                                with state_id as key [authorization_code, implicit]
 * request_type = 'access_token' && grant_type = 'client_credentials' ==> Store the UTC time of the request
 *                                into a cache with state_id as key [client_credentials]
 * request_type = 'access_token' && grant_type = 'password' ==> Store the UTC time of the request into a cache
 *                                with state_id as key [password]
 * request_type = 'resource' ==> Retrieve the stored time and put it into an attribute named recovered_state
 *
 * It also stores the flow type we are in be used later to detect if this is a client_credentials two-legged flow or not.
 */
if (request_type == "authorization" || (request_type == "access_token" &&
    ( grant_type == "client_credentials" || grant_type == "password" ))) {
    var curr_utc_time = "State storage time was: " + IDMappingExtUtils.getCurrentTimeStringUTC();
    IDMappingExtUtils.getIDMappingExtCache().put(state_id, curr_utc_time, 1000);
} else if (request_type == "resource") {
    var recovered_state = IDMappingExtUtils.getIDMappingExtCache().get(state_id);

    if (recovered_state != null) {
        var state_arr = java.lang.reflect.Array.newInstance(java.lang.String, 1);
        state_arr[0] = recovered_state;
        stsuu.addContextAttribute(new Attribute("recovered_state",
            "urn:ibm:names:ITFIM:oauth:response:attribute", state_arr));
    }
}
...

```

Figure 2. OAuth 2.0 JavaScript sample code with state management

Trusted clients management

Advanced Access Control stores trusted client information that is based on the decisions of a resource owner on which clients to trust.

In an OAuth 2.0 flow, the resource owner is asked to provide consent on the scopes that are requested by a client to access the protected resource. The resource owner can either grant permission or deny the client from its access request.

The OAuth server or authorization server uses the trusted clients manager to manage information about trusted clients.

Administrators can configure the behavior of the trusted clients manager in the API protection page. They can configure whether a resource owner is prompted for consent in the Authorization code flow or the Implicit grant flow.

The following configuration options are available:

- Never prompt a resource owner for consent - Resource owners are never prompted for consent and the authorization decision defaults to allow access to the resource.

- Always prompt a resource owner for consent - Resource owners are always prompted for consent even if the client was previously allowed to access the resource.
- Prompt the resource owner once and remember consent - Resource owners are prompted once for consent and later allows access to the resource.

Note: For the Prompt once and remember configuration options, the trusted client manager verifies whether the resource owner previously provided consent on the scopes that are requested by a client.

Proof Key for Code Exchange support

You can configure support for Proof Key for Code Exchange for OAuth clients.

Proof Key for Code Exchange (PKCE) support is a capability (defined in RFC 7636) that adds security when performing the authorization code flow on a mobile device. It addresses a possible security problem that can occur when the following conditions are true:

- There is no client secret.
- The browser or operating system is being used to perform the authentication request.
- A native mobile application is consuming the redirect from the authentication request, and performing an exchange of code for access tokens at the token endpoint.

PKCE support aims to mitigate the risk of a bad actor on the mobile device intercepting the redirect back to native app, and maliciously using the authorization code and the returned access tokens. For a detailed explanation of the scenario, see the Internet Engineering Task Force (IETF) Request for Comments (RFC) 7636: <https://tools.ietf.org/html/rfc7636>

PKCE requires the OAuth client to generate a random string and perform a hash (SHA256 + BASE64URL) on this string. The initial string must be persisted for use at /token, and both the hash and the hash method are presented at /authorize. The authorization server, upon receiving the hash and the method, persists this value against the issued authorization code. When the authorization code is presented at /token, along with the initially generated string, the hash method is applied to the presented string and checked against the string presented at /authorize. If the two match, the request to /token is successful. If they do not match, the request is rejected.

The processing flow is as follows:

1. Client generates a code_verifier, and computes code_challenge using code_challenge_method.
2. Client makes request to /authorize.
3. Authorization server performs standard OAuth request validation for /authorize.
4. Authorization server checks for presence of code_challenge and code_challenge_method.
5. Authorization server stores code_challenge and code_challenge_method against authorization code.
6. Authorization server returns authorization code response.

7. Client presents authorization code to /token including the additional code_verifier.
8. Authorization server performs standard OAuth request validation for /token.
9. Authorization server generates its own code_challenge, using the presented code_verifier, and the stored code_challenge_method.
10. Authorization server compares its generated code_challenge, to the value which was presented in the initial request to /authorize (and stored against the authorization code).
11. If the two match, then an access_token is issued. If the two do not, the request is rejected.

Note: The IETF specification contains a diagram of the above flow. See <https://tools.ietf.org/html/rfc7636#section-1.1>.

To use the IBM Security Access Manager support for PKCE, you must configure the OAuth client to set the **requirePkce** property to true. When this property of the OAuth client is set to true, the following conditions apply:

- A new parameter is required in the request to /token.

code verifier

A cryptographic string of sufficient entropy such that an attacker cannot predict or guess its value, as specified in section 4.1 of the RFC. This value is used with the code_challenge_method presented in the /authorize request, to produce the value to check against the code_challenge, which is also presented at /authorize.

- Two new parameters are required in the request to /authorize.

code_challenge

The product of the code_verifier and the code_challenge_method. Must be the product of the SHA256 + BASE64URL encode, as specified in section 4.2 of the RFC.

code_challenge_method

The method applied to the code_verifier as presented at /token, which is used to check the value of code_challenge. The value of code_challenge_method must be S256, as specified in section 4.2 of the RFC.

Reverse proxy configuration for OAuth and OIDC provider

You can run an automated configuration of a reverse proxy for OAuth and OIDC provider, and view a log of the configuration steps. To remove the reverse proxy configuration, follow the instructions in this section.

Configuring a reverse proxy for OAuth and an OIDC Connect provider

Use a wizard to perform automated configuration of a reverse proxy appliance for OAuth and an OIDC Connect provider.

Before you begin

The reverse proxy server that you want to use for your OAuth or OIDC Connect provider must already be configured. See [Configuring an instance](#).

Procedure

1. From the local management interface, select **Secure Web Settings > Manage > Reverse Proxy**. A list of reverse proxy instances displays.
2. Select the reverse proxy instance name from the list.
3. Select **Manage > OAuth and OIDC Connect provider Configuration**. A window opens where you can add the configuration information.
4. Enter the configuration details.

The **OAuth modes** section lists supported modes. You can select more than one mode.

The modes are options that extend a basic OAuth configuration. A basic configuration sets up the junction, loads the runtime certificate, and provides access to the API Protection endpoints: /token, /userinfo, /introspect, /revoke, /metadata, and /jwks. The base configuration is sufficient if you are doing only a resource or password credentials flow. In this case, you cannot do any API enforcement, but you can get tokens issued. In this scenario, you do not need to select either of the OAuth modes.

If you want to use of the authorization code flow, or implicit flows, which go via a user agent, or if you want to get a user session using the /session endpoint, then you must select **Configure for browser interaction**. If you want this reverse proxy to protect resources with access tokens you must select **Configure for API protection**. The two options are not mutually exclusive; you can select both.

Table 8. OAuth modes

Mode	Description
Configure for browser interaction	When configured for browser interaction, the /authorize and /session endpoints are accessible. Also, EAI authentication is enabled for /session. This configuration option is required for the authorize or implicit code flows.
Configure for API Protection	When this option is selected, an access token can be presented to WebSEAL, and an authenticated session retrieved. The use of cookies is not required; the authorization header is used as the session index. Selecting this option configures oauth-auth and oauth-cluster in the [oauth] stanza in the WebSEAL configuration file. Note: If you select Configure for API protection and do not select Configure for browser interaction , the configuration parameter forms-auth is disabled.

Table 9. Reverse proxy instance

Parameter	Description
Host name	The host name or IP address of the runtime server. This field is required.
Port	The SSL port number of the runtime server. This field is required.
User name	The user name that is used to authenticate with the runtime server. This field is required.

Table 9. Reverse proxy instance (continued)

Parameter	Description
Password	The password that is used to authenticate with the runtime server. This field is required.
Junction	The junction for the reverse proxy instance. The default is /mga.

The **Reuse Actions** section indicates reuse of existing access control lists (ACLs) and certificates.

Table 10. Reuse configuration

Parameter	Description
Reuse Certificates	Select to reuse the SSL certificate if it was already saved. If this check box is not selected, the certificate is overwritten.
Reuse ACLs	Select to reuse any existing ACLs with the same name. If this check box is not selected, the ACLs are replaced.

5. Click **Finish**.
6. When prompted, deploy the pending changes.
7. Restart the reverse proxy.

What to do next

You can examine a log file to view the results of the auto-configuration. See “Viewing a reverse proxy log for an automated configuration”

Viewing a reverse proxy log for an automated configuration

You can view a log file to see the steps taken during the automated configuration of a reverse proxy, such as changes to the reverse proxy configuration file and execution of Security Access Manager **pdadmin** administration commands.

About this task

Use the LMI to view the log that was created when you ran the automated configuration of the reverse proxy for one of the following uses:

- OAuth and OIDC Connect Provider
- Mobile Multi-Factor Authentication (MMFA)
- Federation

The log files typically show configuration of the junction, such as /mga, and creation of the required ACLs, plus additional steps as required.

Procedure

1. Select **Secure Web Settings > Reverse Proxy > Manage > Logging**.
2. Select the log file you want to view. The following log files are supported:

Table 11. Auto-configuration log files

Log file	Description
autoconfig__oauth.log	Automated configuration of the reverse proxy instance with OAuth or an OIDC Connect provider.
autoconfig__mmfa.log	Automated configuration of the reverse proxy instance with MMFA, including configuration of the junction and creation of the required ACLs.
autoconfig__federation.log	Automated configuration of a federation on a reverse proxy server, to set up access between the federation and reverse proxy appliances.

3. Click **View**.

The log file displays in a new window. You can take the following actions:

- View log output by scrolling through the display.
- Select **Number of lines to view** at one time.
- Select **Starting from line** to narrow the range of log file entries to view at one time.
- Click **Reload** to update the display after completion of another configuration.
- Click **Export** to save the log output to a file.

Example

To view example log file output, see “Example reverse proxy log for OAuth and OIDC configuration.”

Example reverse proxy log for OAuth and OIDC configuration

The log file for the automated configuration of a reverse proxy instance lists the configuration actions taken.

Sample output:

- Junction creation

Performing pdadmin cmd:

```
server task default-webseald-server create -t ssl -h localhost -p 443
-b ignore -c all -j -J inhead -k -r -e utf8_uri -f /mga
```

Created junction at /mga

- Reverse proxy configuration file changes

setting stanza value:

```
[server] http-method-disabled-remote = TRACE,CONNECT
```

setting stanza value:

```
[eai] eai-auth = https
```

setting stanza value:

```
[eai] retain-eai-session = yes
```

setting stanza value:

```
[eai] eai-redir-url-priority = yes
```

adding stanza value:

```
[eai-trigger-urls] trigger = /mga/sps/oauth/oauth20/session*
```

adding stanza value:

```
[eai-trigger-urls] trigger = /mga/sps/auth*
```

adding stanza value:

```
[eai-trigger-urls] trigger = /mga/sps/authservice/authentication*
```

setting stanza value:

```

[azn-decision-info] HTTP_HOST_HDR = header:host
setting stanza value:
[azn-decision-info] HTTP_REQUEST_SCHEME = scheme
setting stanza value:
[azn-decision-info] HTTP_REQUEST_METHOD = method
setting stanza value:
[azn-decision-info] HTTP_REQUEST_URI = uri
setting stanza value:
[azn-decision-info] HTTP_AZN_HDR = header:authorization
setting stanza value:
[azn-decision-info] HTTP_CONTENT_TYPE_HDR = header:content-type
setting stanza value:
[azn-decision-info] HTTP_TRANSFER_ENCODING_HDR = header:transfer-encoding
setting stanza value:
[oauth] oauth-auth = https
setting stanza value:
[oauth] default-fed-id = https://localhost/sps/oauth/oauth20
setting stanza value:
[oauth] fed-id-param = FederationId
setting stanza value:
[oauth] cluster-name = oauth-cluster
setting stanza value:
[oauth] user-identity-attribute = username
setting stanza value:
[tfim-cluster:oauth-cluster] handle-pool-size = 10
setting stanza value:
[tfim-cluster:oauth-cluster] handle-idle-timeout = 240
setting stanza value:
[tfim-cluster:oauth-cluster] timeout = 240

setting stanza value:
[tfim-cluster:oauth-cluster] server = 9,
https://localhost:443/TrustServerWS/SecurityTokenServiceWST13
setting stanza value:
[tfim-cluster:oauth-cluster] basic-auth-user = easuser
setting stanza value:
[tfim-cluster:oauth-cluster] basic-auth-passwd = ###
setting stanza value:
[tfim-cluster:oauth-cluster] ssl-keyfile = /var/pdweb/shared/keytab/pdsrv.kdb
setting stanza value:
[tfim-cluster:oauth-cluster] ssl-keyfile-stash = /var/pdweb/shared/keytab/pdsrv.sth
setting stanza value:
[session] require-mpa = no
setting stanza value:
[session] user-session-ids = yes
setting stanza value:
[session-http-headers] Authorization = https

```

- Creating or modifying an ACL

```

Performing pdadmin cmd:
acl create isam_mobile_anyauth
Performing pdadmin cmd:
acl modify isam_mobile_anyauth description OAuth_Auto_Configuration
Performing pdadmin cmd:
acl modify isam_mobile_anyauth set user sec_master TcmdsbsvaBRrx1
Performing pdadmin cmd:
acl modify isam_mobile_anyauth set group iv-admin TcmdsbsvaBRrx1
Performing pdadmin cmd:
acl modify isam_mobile_anyauth set group webseal-servers Tgmdbsrx1
Performing pdadmin cmd:
acl modify isam_mobile_anyauth set any-other Tr
Performing pdadmin cmd:
acl modify isam_mobile_anyauth set unauth T
Performing pdadmin cmd:
acl create isam_mobile_nobody
Performing pdadmin cmd:

```

```

    acl modify isam_mobile_nobody description OAuth_Auto_Configuration
Performing pdadmin cmd:
    acl modify isam_mobile_nobody set user sec_master TcmdbsvaBRrx1
Performing pdadmin cmd:
    acl modify isam_mobile_nobody set group iv-admin TcmdbsvaBRrx1
Performing pdadmin cmd:
    acl modify isam_mobile_nobody set group webseal-servers Tgmdbsrx1
Performing pdadmin cmd:
    acl modify isam_mobile_nobody set any-other T
Performing pdadmin cmd:
    acl modify isam_mobile_nobody set unauth T

```

- **Attaching an ACL**

```

Performing pdadmin cmd:
    acl attach /WebSEAL/isam-default/mga/sps/oauth/oauth20/session isam_mobile_unauth
Performing pdadmin cmd:
    acl attach /WebSEAL/isam-default/mga/sps/oauth/oauth20/token isam_mobile_unauth
Performing pdadmin cmd:
    acl attach /WebSEAL/isam-default/mga/sps/static isam_mobile_unauth
Performing pdadmin cmd:
    acl attach /WebSEAL/isam-default/mga/sps/wsoi isam_mobile_anyauth
Performing pdadmin cmd:
    acl attach /WebSEAL/isam-default/mga/sps/xauth isam_mobile_anyauth

```

Removing reverse proxy configuration for OAuth and OIDC provider

You must manually remove the configuration of OAuth and OIDC provider from a reverse proxy instance.

About this task

You can accomplish the manual steps by using the **pdadmin** command and by editing the WebSEAL configuration file.

You can use the appliance Local Management Interface (LMI) to edit the WebSEAL configuration file. On the Reverse Proxy management page, select the appropriate WebSEAL instance and click **Manage > Configuration > Edit Configuration File** to open the Advanced Configuration File Editor. You can use this editor to directly edit the WebSEAL configuration file.

For information on **pdadmin**, see pdadmin commands.

Procedure

1. Remove the following ACLs:

- isam_oauth_anyauth
- isam_oauth_unauth
- isam_oauth_nobody
- isam_oauth_rest

You can use the **pdadmin** command to remove ACLs. See `acl detach` and `acl delete`.

2. If your deployment has no further need for the junction, delete it.

Note: Ensure that the junction is not used for any function other than configuration of OAuth and OIDC provider. If you are not certain about whether junction is used for other configurations, you can skip this step.

The junction name is the value you specified when you created the junction.
The default junction name is `/mga`.

- You can use the **pdadmin** command to delete junctions. See server task delete.
3. If OAuth was configured for API protection, disable oauth-auth in the [oauth] stanza.
To disable the property, edit the WebSEAL configuration file. See oauth-auth and [oauth] stanza.
 4. If OAuth was configured for Browser flows:
 - a. Remove the trigger URI /<jct>/sps/oauth/oauth20/session

Note: In the URI, /<jct> refers to the WebSEAL junction that you configured. You assigned it a unique name or accepted the default name of /mga.
 - b. If no other ISAM services are configured, remove the following trigger URIs:
 - /<jct>/sps/auth*
 - /<jct>/sps/authservice/authentication
 - c. If all triggers are removed, disable eai-auth in the [eai] stanza.
To remove trigger URLs, edit the WebSEAL configuration file. See [eai-trigger-urls] stanza and [eai] stanza.
 5. If OAuth was configured for API protection but OAuth was **not** configured for browser flows, re-enable forms-auth in the [forms] stanza.
To modify the property, edit the WebSEAL configuration file. See [forms] stanza and forms-auth.

Configuring API protection

The API protection uses the OAuth 2.0 protocol. To configure the API protection, you must create a definition and a client.

You must then attach the API protection definition to a resource.

Creating an API protection definition

Create API protection definitions to configure the settings that dictate the behavior of how resources are accessed. The configuration settings protect the resources from unauthorized access.

Procedure

1. Log in to the local management interface.
2. Click either **Secure Access Control > Policy > OpenID Connect and API Protection** or **Secure Federation > Manage > OpenID Connect and API Protection**.
3. Click **Definitions**, and click .
4. In the **Name** field, type a unique name for the definition.

Note: The name must begin with an alphabetic character. Do not use control characters, leading and trailing blanks, and the following special characters ~ ! @ # \$ % ^ & * () + | ` = \ ; : " ' < > ? , [] { } / anywhere in the name.

5. In the **Description** field, provide a brief description about the definition.
6. If you want to enforce an access policy, select the policy from the menu for the **Access Policy** field.

Note: The menu shows Access Policies that are currently defined. To use an access policy with **OpenID Connect and API Protection**, you must define the policy prior to running the configuration wizard. See Access policies.

7. Click **Grant Types** and select at least one grant type.

The grant type **Authorization code** is enabled by default. For information on grant types, see “OAuth 2.0 and OIDC workflows” on page 111.

8. Click **Token Management**.

Specify values for the token properties. For descriptions of each property, see “API Protection token management properties” on page 133.

9. Click **Trusted Clients and Consent** and select when you want the user to be prompted to consent to an authorization grant.

10. If you want to protect an OpenID Connect Provider, click **OpenID Connect Provider** and select **Enable OpenID Connect**.

Specify OpenID Connect Provider settings as needed for your deployment. For descriptions of each property, see “API Protection OpenID Connect Provider properties” on page 135

11. Click **Save**.

What to do next

- Register an API protection client.
- Deploy the pending changes. See Chapter 15, “Deploying pending changes,” on page 213

Managing API protection definitions

An API protection definition is a set of configurations that define how resources are accessed.

About this task

You can add, modify, and delete definitions.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **OpenID Connect and API Protection**.
4. Click **Definitions**.
5. Perform one or more of the following actions:

Add definitions

Click . See “Creating an API protection definition” on page 131 for details.

Modify definitions

- a. Select a definition in the list of definitions.
- b. Click .
- c. Complete the properties for the definition.

Note:

You cannot modify the definition name and grant types. See “Creating an API protection definition” on page 131 for details.

- d. Click **Save**.

Delete definitions

- a. Select a definition or press Ctrl and select multiple definitions in the definition list.
- b. Click . Confirm the deletion. Click **OK** to continue or click **Cancel**.

Note: A definition cannot be deleted if there are clients associated with it or it is attached to a resource.

6. Click **Save**.
7. When you add, modify or delete a definition, a message indicates that there are changes to deploy. If you are finished with the changes, deploy them.
For more information, see Chapter 15, “Deploying pending changes,” on page 213.

API Protection token management properties

When you configure API Protection for OAuth and OpenID Connect, you must specify properties for token management.

The local management interface (LMI) page **OpenID Conect and API Protection** has a section that prompts for settings for token management. Refer to the following list of properties to determine the appropriate value, for your deployment, for each property.

For configuration task instructions, see “Creating an API protection definition” on page 131.

Access token lifetime (seconds)

Specifies the number of seconds an access token is valid. When the access token becomes invalid, the client cannot use it to access the protected resource.

Default value: 3600 seconds.

Minimum value: 1 second.

Access token length

Specifies the number of characters in an access token.

Default value: 20 characters.

Minimum value: one character.

Maximum value: 500 characters.

Enforce single-use authorization grant

If enabled, all the authorization grant tokens are revoked after an access token is validated. If enabled, resource requests that involve redirects fail because the access token is validated multiple times.

Default value: disabled

Authorization code lifetime (seconds)

Specifies the number of seconds that an authorization code is valid.

This option applies only to an authorization code grant type. The authorization server generates an authorization code and sends it to the client. The client uses the authorization code in exchange for an access token.

Default value: 300 seconds.

Minimum value: 1 second.

Authorization code length

Specifies the number of characters in an authorization code.

Default value: 30 characters.

Minimum value: one character.

Maximum value: 500 characters.

Issue refresh token

Specifies whether a refresh token is sent to the client. A refresh token obtains a new pair of access and refresh tokens. This option is only applicable to the Authorization code and Resource owner password credentials grant types.

Maximum authorization grant lifetime (seconds)

Specifies the maximum number of seconds that the resource owner authorizes the client to access the protected resource.

This option is available only if you enable the **Issue refresh token** option.

The value for this lifetime must be greater than the values specified for the authorization code and access token lifetimes.

When this lifetime expires, the resource owner must reauthorize the client to obtain an authorization grant to access the protected resource.

Default value: 604800 seconds.

Minimum value: 1 second.

Refresh token length

Specifies the number of characters in a refresh token. This option is available only if you enable the **Issue refresh token** option.

Default value: 40 characters.

Minimum value: 1 characters.

Maximum value: 500 characters.

Enforce single access token per authorization grant

If enabled, all previously granted access tokens are revoked after a new access token is generated presenting the refresh token to the authorization server.

This option is available only if you enable the **Issue refresh token** option.

Default value: enabled

Enable multiple refresh tokens for fault tolerance

Specifies how refresh tokens are handled. When this option is enabled, and a refresh request is made, the initially-used refresh token remains active (assuming it was initially active), even after a successful refresh request is made and a new token pair (access token and refresh token) is returned. Only upon the subsequent use of the new access token or new refresh token will the initially presented refresh token be invalidated. If the initially used refresh token is presented again, the tokens issued on the first refresh request (Pair 1) are revoked, and another token pair (access token and refresh token) is issued. This new pair (Pair 2) is valid, and Pair 1 is invalid.

This option is available only if you enable the **Issue refresh token** option.

Default value: disabled

Enable PIN policy

Provides more protection during the exchange of a refresh token for a new pair of access and refresh tokens.

This option is available only if you enable the **Issue refresh token** option. If enabled, you must configure the PIN length.

PIN Length

Specifies the number of characters in a PIN. This option is available only if you enable the **Enable PIN policy** option. You can use the **runtime.hashAlgorithm** runtime parameter to configure the algorithm that is used to hash the PIN before it is stored. For more information, see Advanced configuration properties.

Default value: 4 characters.

Minimum value: 3 characters.

Maximum value: 12 characters.

Token character set

By default, a set of alphanumeric characters is displayed. You can specify the set of characters used to generate tokens in the following methods:

- Manually enter characters
- Select from a pre-defined character set from the drop-down list
- Edit the characters in the field after selecting from a set from the drop-down list

The configured token character set is applicable for all token types. If this parameter is left blank, all available alphanumeric characters are used to generate the token.

Maximum number for characters allowed: 200

API Protection OpenID Connect Provider properties

When you configure API Protection for OAuth and OpenID Connect, and you enable OpenID Connect, you must specify properties for the OIDC Provider.

The local management interface (LMI) page **OpenID Connect and API Protection** has a section that prompts for settings for OpenID Connect Provider. Refer to the following list of properties to determine the appropriate value for each property.

For configuration task instructions, see “Creating an API protection definition” on page 131.

Issuer Identifier

This entry identifies the issuing entity. It must be a valid URL with the protocol prefix `https://`. For example, `https://ibm.com` or `https://accounts.google.com`. It must not include fragment or query portions. The Issuer Identifier is defined by the OIDC specification. See http://openid.net/specs/openid-connect-core-1_0.html#IssuerIdentifier

Point of Contact Prefix

The Point of Contact Prefix is used to correctly populate the URLs on the metadata page. It must include the host, port, and path information of the

reverse proxy junction to the runtime. For example: `https://isam.myidp.ibm.com:443/mga/`. Note that is not a field from the OIDC standard.

Metadata URI

A location where you can view your metadata. Metadata is useful to discover the capabilities of an OP. The metadata includes all other URIs. This field is read-only.

id_token Lifetime

Time in seconds for which the id_token is valid. The value is the difference between the values in the iat and exp claims of the issued JSON Web Token (JWT). You can use a pre-token mapping rule to overload this value at runtime.

Default: 3600 seconds.

Signing Algorithm

The algorithm that is used to sign the JWT. This setting is the alg claim in the JWT. Use the menu to select the appropriate value. You can use a pre-token mapping rule to overload this value at runtime.

Default: RS256.

Key Database for Signing

The Key database that is used to source the private key for signing the ES/RS signature algorithms. You can use a pre-token mapping rule to overload this value at runtime.

Default: rt_profile_keys

Certificate Label for Signing

The label of the key in the selected keystore that is used as the private key for ES/RS signing. You can use a pre-token mapping rule to overload this value at runtime.

Default: server

Encrypt ID token

Boolean value to indicate whether this JWT must be encrypted. Select the check box to encrypt the token and configure encryption settings. You can use a pre-token mapping rule to overload this value at runtime.

Key Agreement Algorithm

The encryption algorithm that is used for JWT key agreement. This setting is the alg claim in the encrypted JWT. You can use a pre-token mapping rule to overload this value at runtime.

Default: RSA-OAEP-256

Encryption Algorithm

The encryption algorithm that is used for JWT payload encryption. This setting is the enc claim in the encrypted JWT. You can use a pre-token mapping rule to overload this value.

Default: A128CBC-HS256

Attribute Mapping

You can use the Attribute Mapping section to define attributes that can be used to customize claims from attribute sources. Attribute sources can be: Fixed, Credential, or LDAP.

When you select **Enable OpenID Connect**, the **New** and **Delete** icons are activated for attribute mapping. To create, select **New** and enter **Attribute Name**. Select **Attribute Source** type.

To remove an existing **Attribute Name**, select the attribute and click **Delete**.

If you do not select **Enable OpenID Connect**, you cannot create new attribute mappings.

Enable client registration

Check this check box to allow users to register dynamic clients.

Issue Client Secret

If dynamic clients are enabled, check this check box if you want them to be confidential clients.

PIN policy

Advanced Access Control extends OAuth 2.0 capabilities with a PIN policy.

The PIN policy provides the capability of protecting a refresh token with a PIN provided by the API protection client. An administrator can configure the API protection definition to enable the PIN policy for the grant types that issue a refresh token. The two grant types that issue a refresh token are Authorization code and the Resource owner password credentials.

When enabled, the client is required to send a PIN as a parameter in the first access token request. The parameter name is **pin**. The parameter value consists of digits of the length that is configured in the API protection definition. The client must submit the same PIN on subsequent requests when exchanging a refresh token for a new access token.

The PIN policy can be configured to use various hash algorithms to hash and store the PIN. Use the **runtime.hashAlgorithm** configuration parameter to specify the hash algorithm. For more information about configuring the hash algorithm, see *Runtime properties* in Advanced configuration properties

Registering an API protection client

Register OAuth API protection clients in the Clients panel. Clients are the entities against which OAuth access and refresh tokens are granted at runtime.

About this task

API Protection clients now have a dynamic data field when they are configured. This allows storage of arbitrary data against the client which can be accessed at runtime (for example, in the consent page or in mapping rules).

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **OpenID Connect and API Protection**.
4. Click **Clients**.
5. Click  .
6. Specify the following information:

Client name

Specify a meaningful client identifier for each client registration. You can use this value to search for client registrations.

API definition

Specifies the related Definition, which owns and defines the client. A Definition can own many client registrations but a client registration can belong to only one Definition. When you create a client, a list of available Definitions are available. When a client is created, this value cannot be modified.

Confidential

Specify whether the client type is confidential. A confidential client type requires a client secret. Enable this feature if you want the client to require a client secret.

Client secret

This field is enabled only if the client is indicated as confidential. Specify a client secret that is used to authenticate an OAuth client at runtime. It is mandatory for all clients that belong to API protection definitions where the client type is Confidential and the client credentials grant type is enabled. Click **Generate** to have a client secret that is generated for you or specify your own secret.

Redirect URI (Optional)

Click **New** to specify the redirect URI to use for the client. You can create multiple redirect URI entries. Each URL must be unique.

Company name

Specify the name of the company for this client.

Company URL (Optional)

Specify the URL of the company website.

Contact name (Optional)

Specify a name of the contact person for this client.

Email address (Optional)

Specify the email address of the contact person for this client.

Telephone number (Optional)

Specify the telephone number of the contact person for this client.

Contact type (Optional)

Select the contact type from the list:

- Administrative
- Support
- Technical
- Billing
- Other

Other information (Optional)

Specify extra information about the client contact.

Require PKCE (RFC 7636)

Requires Proof Key for Code Exchange, which adds security when performing the authorization code flow on a mobile device. See “Proof Key for Code Exchange support” on page 124.

JWKS endpoint

This endpoint allows retrieval for a client's public key when encryption is used.

JWT Encryption keystore

The database that is used in key agreement when using an asymmetric JWT encryption algorithm. You can use a pre-token mapping rule to overload this value at runtime.

This field is enabled if OIDC is enabled for the selected API Definition.

JWT Encryption certificate

The label of the key in the keystore that is used in key agreement, when you use an asymmetric JWT encryption algorithm. You can use a pre-token mapping rule to overload this value at runtime.

This field is enabled only if a valid encryption keystore is selected from the drop-down list for **JWT Encryption keystore**.

7. Enter any dynamic data on the **Extension Properties** tab.
8. Click **OK**.

Managing registered API protection clients

Manage registered OAuth API protection clients.

About this task

You can search and delete clients. You can search for API protection clients based on the following values:

- Client name
- Client ID
- API protection definition

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **OpenID Connect and API Protection**.
4. Click **Clients**.
5. Perform one or more of the following actions:

View and filter clients

You can filter for client name, client ID, and API protection definition.

Take any of the following actions to filter your view:

- Select the  **Details View** to view client name, client ID, and API protection definition.
- Select the  **List View** to view only the name of the client.
- Type a term, such as a client name, client ID, and API protection definition in the **Filter** field to list clients that use that term. Any part of the values for client name, client ID, or API protection definition that match is applied by the filter and is displayed in the search results. Click x to clear the **Filter** field.
- Sort the client list by column with the up or down arrow on each column. For example, you can view the list of clients that are sorted by the **Clients** column in ascending order by clicking the up arrow.

Modify clients

Attention: Ensure that the modification does not affect a current policy or configuration. If you modify a client that is in-use, the policy or configuration that uses the client might stop working.

- a. Select the client you want to modify.
- b. Click .
- c. Complete the properties for the clients.
- d. Click **OK**.

Delete clients

- a. Select a client or press and hold the Ctrl key and select multiple clients to remove.
 - b. Click . Confirm the deletion. Click **OK** to continue or click **Cancel**.
When you delete a client:
 - The client registration is removed from the database.
 - All tokens issued against that client is removed.
6. When you add, modify or delete a client, a message indicates that there are changes to deploy. If you are finished with the changes, deploy them.
For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Managing policy attachments

Attach policies or API protection definitions to resources so that the policies and definitions can be enforced.

Before you begin

You must create policies, policy sets, or API protection definitions.

When you create policies, policy sets, or API protection definitions you cannot use them until you publish them to resources. Once policies, policy sets, or API protection definitions are published, they are enforced during the evaluation of access requests.

About this task

You can:

- Add a resource
- Add a policy or API protection definition attachment to a resource
- Remove a policy or API protection definition attachment from a resource
- Delete a resource
- Publish a policy or API protection definition attachment

When a deployment is fully configured, the Resources panel displays three levels of entries. The top-level entry is the web container that contains the protected object space for a server instance. The second level shows the resources in the protected object space. The third level lists the policies and API protection definitions that are attached to each resource.

Tip: The user interface provides a quick filter feature for use on the top-level entry. Use the quick filter to search for a specific top-level entry. Enter the first few characters of the web container, and the list displays only the entries that contain the specified characters.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Access Control**.
4. Click **Resources**.
5. Perform one or more of the following actions:

Add a resource

- a. Click  .

Note: When you add a resource for the first time, the system prompts you to enter the user name, password, and domain for the Security Access Manager policy server. The entered information is cached and used by default when you add a resource again. If you want to change this domain, click **Change Domain** and then enter the new user name, password, and domain information. This new information replaces the old cached values.

- b. Select the resource type in the **Type** field.
 - If you select the **Reverse Proxy** type:
 - 1) In the **Proxy Instance** field, click the down arrow icon to display a list of proxy instances. Select an entry.
For example, the list of proxy instances is the WebSEAL protected object space that is defined directly under /WebSEAL.
 - 2) Specify a resource by entering its name or browsing for it. When you browse, you can expand the list of resources. The list hierarchy is based on the structure of the WebSEAL protected object space.
 - In some cases, not all resources are displayed because the WebSEAL protected object space is a sparse tree. For example, you might see only the resource /myserver-jct/benefits. You can select this resource and click **OK** to add it to the **Protected Path**. You can then add /myserver-jct/benefits/medical.
 - In some cases, you cannot view the object space for the web server junction. For example, if the administrator did not install the IBM Security Access Manager **querycontents** script on the application server, you cannot see the junction contents. In these cases, you can enter the resource path manually.
 - If you select the **Application** type:
 - 1) Select an application ID from the list or click **Add New** to add a new application ID.
 - 2) Enter the resource ID.
- c. Click **Save**.
- d. Attach a policy to the resource.

Attach a policy or API protection definition to a resource

- a. Select a resource node and click  Attach.
- b. In the Attach Policies panel, select **Policies** or **Policy Sets** or **API Protection**.
- c. From the displayed list, select one or more policies or policy sets or API protection definitions.

Tip: You can type the name of the applicable policy or policy set or API protection definition in the quick filter.

Notes:

- You can attach both individual policies, policy sets, or API protection definitions.
 - You cannot attach policies or policy sets to a resource where that resource already has API protection definitions attached.
 - You cannot attach API protection definitions to a resource where that resource already has policies and policy sets attached.
- d. Click **OK** to save your changes.

Note: The policy or API protection definition remains inactive until you publish it.

Remove a policy or API protection definition attachment

- a. To remove a policy or API protection definition attachment from a resource, select the policy node and click .
- b. When prompted, confirm the deletion.

Note: You must publish the change.

Delete a resource

- a. To delete a resource and all attached policies or API protection definitions, select the resource node and click .
 - b. When prompted, confirm the deletion.
- When you delete a resource:
- You cannot delete the server node.
 - You do not have to manually publish the change. The deletion is automatically published.

Publish a policy or API protection definition

Select a resource in the resource hierarchy and click  Publish. When the publication completes, the status column for the resource indicates the status and time of the publication.

Note: Activation of the published policy or API protection definition could take up to a minute to complete.

Modify Resource

Note: You can only use this function if policy or policy sets are attached to the given resource.

- a. Select a resource node and click .

- b. In the Modify Resource panel, you can modify the **Policy Combining Algorithm**. Choose the preferred algorithm:
 - **Deny access if any attached policy returns deny**
This algorithm means that if multiple policies or API protection definitions are attached to a resource, and any one of those policies or API protection definitions returns Deny, then the access request is denied.
 - **Permit access if any attached policy returns permit**
This algorithm means that if multiple policies or API protection definitions are attached to a resource, and any one of those policies or API protection definitions returns Permit, then the access request is permitted.

Using `oauthScope` attributes in an access control policy

You can use the subject and resource `oauthScope` attributes as part of an access control decision for a resource.

Before you begin

1. Create a reverse proxy instance.
2. Run the `isamcfg` tool. You must configure access control policies and API protection capabilities.
3. Determine the access control resources that your policies must be attached to. If the resources do not exist, add them.

About this task

To use the OAuth attributes in an access control decision, you must attach the access control policy and API protection definition in the proper locations in the protected object space.

Procedure

1. Create an access control policy. Specify the `oauthScopeResource` attribute, the `oauthScopeSubject` attribute, or both, in one or more rules for this policy. See [Creating an access control policy](#).
2. Attach the access control policy to an object in the protected object space. See [“Managing policy attachments”](#) on page 140.
3. Create an API protection definition. See [“Creating an API protection definition”](#) on page 131.
4. Register an API client that uses the API protection definition you created in step 3. See [“Registering an API protection client”](#) on page 137.
5. Attach the API protection definition to an object in the protected object space. See [“Managing policy attachments”](#) on page 140.

When you attach the definition to a resource, the resource must be at a level lower than where the access control policy is attached in step 2. The term *lower* means farther away from the root of the protected object space.

For example, in the resource tree `jct/dir1/dir2/protected_resource`, you can attach the access control policy to `/jct`. Then, attach the API protection definition to `/jct/dir1`.

6. Deploy the pending changes.

Results

The access decision for a resource at or below the API protection definition involves the **oauthScope** attributes that were defined in the access control policy.

Uploading OAuth response files

Use the local management interface to upload your own custom OAuth response files.

Procedure

1. From the top menu, select **Secure Reverse Proxy Settings > Manage > Reverse Proxy**.
2. Select a reverse proxy.
3. Select **Manage > Management Root**.
4. Select the **oauth** folder.
5. Select **Manage > Import**.
6. Click **Browse**.
7. Browse to the file you want to import.
8. Click **Open**.
9. Click **Import**.

OAuth introspection

An Introspection URL implemented to the spec of RFC 7662 allows for information about an access token to be returned. This allows OAuth clients to query a token to identify if the token exists and is valid. Extensions to this endpoint have been made to also include some information about the token, beyond whether the token is valid.

The introspection URL is disabled by default. To enable it, set the advanced configuration **oauth20.introspectEndpointEnabled** to true.

A usual introspection response for a valid token includes the following values:

```
{
  "active":true,
  "username":"jessica",
  "client_id":"yb981a1",
  "scope":"email profile",
  "iat": 1487744340,
  "exp": 1487747940
}
```

active Signifies this token is valid.

scope The scope of the access token.

username

The username of the user token was granted by.

client_id

The client this token was granted to.

iat The UNIX timestamp for when this token was granted.

exp The UNIX timestamp for when this token will expire.

A usual introspection response for an invalid token includes the following values:

```
{
  "active":false
}
```

active Signifies this token is invalid.

The RFC articulates that the introspection endpoint must be authenticated with client credentials. These credentials can be provided as post parameters 'client_id' and 'client_secret', or they can be provided as a Basic Authentication (BA) header. The authentication using BA can occur at the point of contact (reverse proxy) or by the introspection endpoints itself (similar to the token endpoint). The client may also authenticate using an access token issued to this client.

This client authentication is performed in the pre-token mapping rule (with the default rule). This out of the box rule does not allow non-confidential clients to introspect tokens. However, by modification of the rule, non-confidential clients may be able to make use of this endpoint. The RFC articulates security concerns when allowing non-confidential clients to introspect tokens (DOS, crawling the possible token space. See section 4. <https://tools.ietf.org/html/rfc7662#section-4>). A non-confidential client can provide client credentials using BA or post parameters. When using BA, the credentials should present a password of empty string ("").

The introspection endpoint can allow clients to introspect the tokens of each other. By default, this is not allowed. However, a change to the POST token rule can be made to enable it. See the out of the box rule for details.

URL:

`https://<Reverse proxy host/port/junction> /sps/oauth/oauth20/introspect`

HTTP Request Example:

```
POST /mga/sps/oauth/oauth20/introspect HTTP/1.1
Host: server.oauth.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded

client_id=yb981a1&client_secret=4531959525657&token=2YotnFZFEjrlzCsicMwPAA
```

The introspection endpoint supports use of the '**token_type_hint**' as per section 2.1 (<https://tools.ietf.org/html/rfc7662#section-2.1>). This allows an optimization in the time of the token lookup. This does not limit the breadth of the search for the token in the token cache. Any token type will still be found even when its type is not the same as the hint.

For example:

```
POST /sps/oauth/oauth20/introspect HTTP/1.1
Content-Type: application/x-www-form-urlencoded

token=&client_id=aClient&client_secret=aSecret&token_type_hint=access_token
```

Valid values for **token_type_hint** are '**access_token**' and '**refresh_token**'.

If using custom tokens that are not stored in the token cache, the pre-mapping rule can be used to inform the runtime that the token provided was valid. To do this, add a context attribute with the name '**active**' and the type '**urn:ibm:names:ITFIM:oauth:rule:decision**'. The response attribute type can also be used. If this parameter is provided, the runtime will do no further work, and the post-rule will be invoked. For example:

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("active",
"urn:ibm:names:ITFIM:oauth:rule:decision", "true"));
```

OAuth revocation endpoint

You can use a revocation endpoint to ensure that tokens are revoked.

Security Access Manager supports use of an OAuth revocation endpoint. This endpoint enables clients to inform an authorization server that a specified token is no longer used, and must be revoked. The support is compliant with RFC 7009.

The revocation URL is enabled by default and cannot be disabled.

A typical revocation response returns a 200 response, with an empty body. You can modify a mapping rule to add response attributes.

The RFC states that the revocation endpoint must be authenticated with client credentials. You can provide these credentials as post parameters *client_id* and *client_secret*, or provide them as a Basic Authentication (BA) header. The authentication that uses BA can occur at the point of contact (reverse proxy) or by the revocation endpoint itself (similar to the OAuth token endpoint). The client can also authenticate by using an access token that was issued to this client.

The RFC states that the revocation endpoint can revoke only tokens that were generated by the client that is requesting the revocation.

URL

`https://<Reverse proxy host/port/junction> /sps/oauth/oauth20/revoke`

HTTP Request Example

```
POST /mga/sps/oauth/oauth20/revoke HTTP/1.1
Host: server.oauth.com
Content-Type: application/x-www-form-urlencoded
client_id=yb981a1&client_secret=4531959525657&token=2YotnFZFEjr1zCsicMWpAA
```

token_type_hint

The revocation endpoint supports use of the *token_type_hint*. Use of the hint optimizes the lookup time for the token. Use of the hint does not limit the breadth of the search for the token in the token cache. Token types are found even if a token's type is not the same as the hint.

For example:

```
POST /sps/oauth/oauth20/revoke HTTP/1.1
Content-Type: application/x-www-form-urlencoded

token=&client_id=aClient&client_secret=aSecret&token_type_hint=access_token
```

Valid values for *token_type_hint* are *access_token* and *refresh_token*.

Mapping rule variables

Table 12. Mapping rule variable for OAuth revocation

Variable	Description
<code>only_allow_conf_client_revoke</code>	You can use the pre-mapping rule to specify whether non-confidential clients can revoke tokens. By default, only confidential clients can revoke tokens. To enable non-confidential clients to revoke tokens, set this parameter to false. Default: <pre>var only_allow_conf_client_revoke = true;</pre>

OIDC Claims customization

You can customize the OIDC claims that contain information about the user and about the authentication event.

The OIDC specification describes standards for creating and processing claims. OpenID defines a standard set of basic profile Claims. You can use specific scope values to access pre-defined sets of Claims. You can also request individual Claims by using the `claims request` parameter.

The specification outlines the following structure for defining and handling claims:

- There are about twenty standard claims, such as name, picture, gender, and so on.
- You can use the `scope` parameter to request Claims. There are well defined scopes, each of which translates to a collection of standard claims such as profile, email, phone, and address.
- You can also request claims individually by using the `claims request` parameter.
- When an access token exists, claims are returned in `UserInfo`. When there is no access token, claims are returned in ID Tokens.
- `UserInfo` is returned at the `/userinfo` endpoint. ID Tokens are returned at either the `/authorize` or `/token` endpoints, depending on which OIDC flow is executed.
- There are additional request parameters that can affect the claims. Examples of these include `max_age`, `acr_values`, and `claims_locales`.

OIDC does not perform authentication itself, but relies on an authentication module to authenticate the user. This means that the OIDC protocol cannot provide claims values, because it has no knowledge of what user information is available, and how the data is stored. For example, perhaps only the user name information is available, but is stored in LDAP as the attribute `cn`.

As a result, Security Access Manager supports the customization of claims values. For ID Tokens, you can customize claims values through a pre-token mapping rule. For `UserInfo`, you can customize claims in a post-mapping rule.

In Security Access Manager, when you create a new OIDC Definition, default pre-token and post-token mapping rules are created. These mapping rules contain examples of how the claims can be resolved.

Note: To review the OIDC specification, see http://openid.net/specs/openid-connect-core-1_0.html#Claims

UserInfo endpoint

The OIDC specification recommends the use of the UserInfo endpoint. The UserInfo endpoint is useful, for example, when a Relying Party cannot parse a JWT Token to obtain information about the authenticated user.

Without the use of customization, the /userinfo endpoint contains only the field sub. This is the subject identifier for the end-user at the issuer.

The specification mandates to output the claims as much as possible at the /userinfo endpoint, except when an access token is not generated. Also, in some cases the ID Token is generated at both /authorize and /token endpoint. The Security Access Manager will not impose any restriction of which claims must be output to ID Token or User Info. Likewise, there are no restrictions that the same claims must be returned at /authorize and /token. Based on the needs of your deployment, you can choose not to output some claims in certain flows even though the claims are requested.

The /userinfo endpoint is protected by an access token. You can access it by an HTTP GET or POST. See the following links:

- http://openid.net/specs/openid-connect-core-1_0.html#UserInfoRequest
- <https://tools.ietf.org/html/rfc6750#section-2>

An unauth ACL is attached for this endpoint.

Claims List

Many parameters can contribute to a claims list. For example, parameters specified by a scope are often used to build a list. By parsing the claims parameter, you can also retrieve individual claims and determine which claims are for UserInfo and which claims are for ID Tokens.

Security Access Manager builds a claims list and makes it available in the Security Token Service Universal User (STSUU) as a list of voluntary and essential claims. This list is produced only for OIDC requests that have scope openid against an OIDC-enabled provider.

The processing sequence is:

1. The scope parameter is processed first, since all claims requested through it are voluntary claims. The specification defines only profile, address, email, and phone scopes as well-defined scopes. These translate into a list of standard claims. However, if the scope contains other scopes that are not well-defined, it is treated as an individual claim.
2. Next the claims parameter is processed. Either UserInfo or ID Token is processed, depending on the endpoint.

Table 13. Claims types

Claim	Type
Voluntary claims	urn:ibm:names:ITFIM:oidc:claim:voluntary
Essential claims	urn:ibm:names:ITFIM:oidc:claim:essential

For example, given a request with scope="openid phone organization" and a claims parameter that is the described in the following JavaScript Object Notation (JSON):

```
{
  "userinfo": {
    "given_name": {"essential": true},
    "email": {"essential": true},
    "email_verified": {"essential": true},
    "http://example.info/claims/groups": null
  },
  "id_token": {
    "nickname": null,
    "auth_time": {"essential": true},
    "acr": {"values": ["urn:mace:silver"]}
  }
}
```

At the /authorize or /token endpoint where the ID Token can be generated, the following is available in the STSUU context:

```
<stsuser:ContextAttributes>
  // from scope: phone and organization
<stsuser:Attribute name="organization" type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  <stsuser:Attribute name="phone_number" type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  <stsuser:Attribute name="phone_number_verified"
    type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  // from id_token claims parameter
  <stsuser:Attribute name="nickname" type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  <stsuser:Attribute name="auth_time" type="urn:ibm:names:ITFIM:oidc:claim:essential"/>
  <stsuser:Attribute name="acr" type="urn:ibm:names:ITFIM:oidc:claim:voluntary">
    <stsuser:Value>urn:mace:silver</stsuser:Value>
  </stsuser:Attribute>
</stsuser:ContextAttributes>
```

At the /userinfo endpoint, the following is available in the STSUU context:

```
<stsuser:ContextAttributes>
  // from scope: phone and organization
  <stsuser:Attribute name="organization" type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  <stsuser:Attribute name="phone_number" type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  <stsuser:Attribute name="phone_number_verified"
    type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
  // from userinfo claims parameter
  <stsuser:Attribute name="given_name" type="urn:ibm:names:ITFIM:oidc:claim:essential"/>
  <stsuser:Attribute name="email" type="urn:ibm:names:ITFIM:oidc:claim:essential"/>
  <stsuser:Attribute name="email_verified"
    type="urn:ibm:names:ITFIM:oidc:claim:essential"/>
  <stsuser:Attribute name="http://example.info/claims/groups"
    type="urn:ibm:names:ITFIM:oidc:claim:voluntary"/>
</stsuser:ContextAttributes>
```

Retrieving Claim Values

You can use the function `associate()` and **UserLookupHelp** to retrieve claim values. Security Access Manager allows you to also retrieve claim values by using attribute sources. The Federation support in Security Access Manager includes management of attribute sources, in the LMI under Advanced Access Control.

You can provide mappings between a specific attribute name and an attribute source, as part of configuring an OIDC Definition. During runtime, this mapping is resolved and made available in the STSUU.

For example, assume the attribute sources in the following table are configured.

Table 14. Example configuration of Attribute Sources

Attribute Name	Value	Type
CredentialNickName	AZN_CRED_PRINCIPAL_NAME	Credential
FixedOrganization	www.ibm.com	Fixed
LDAPMail	mail	LDAP

Assume also that you have created the following attribute mapping in the OIDC Definition configuration.

Table 15. Attribute Mapping

Attribute Name	Attribute Source
email	LDAPMail
nickname	CredentialNickName
organization	FixedOrganization

Then, during runtime, in the STSUU the following attributes can be found:

```
<stsuser:AttributeList>
  <stsuser:Attribute name="nickname" type="urn:ibm:names:ITFIM:5.1:accessmanager">
    <stsuser:Value>test1</stsuser:Value>
  </stsuser:Attribute>
  <stsuser:Attribute name="organization" type="urn:ibm:names:ITFIM:5.1:accessmanager">
    <stsuser:Value>www.ibm.com</stsuser:Value>
  </stsuser:Attribute>
  <stsuser:Attribute name="email" type="urn:ibm:names:ITFIM:5.1:accessmanager">
    <stsuser:Value>test1@iswga.com</stsuser:Value>
  </stsuser:Attribute>
</stsuser:AttributeList>
```

The resolution of attribute sources during runtime is dependent on the context. The context is the list of attributes that are available in the STSUU at that moment. The following conditions apply:

- For fixed attribute source, there is no restriction on attribute source resolution.
- For a credential type attribute source, resolution occurs best at the /authorize endpoint. This is because since the entire authentication information (and credential) is available at that time.
- For LDAP type attribute source, if you are using a macro for the search filter or baseDN, resolution also depends on the context. The information might be available in STSUU, but in a different context it might be available under a different name. This means it can be difficult to create one LDAP attribute source that fits all contexts.

The common use of a macro is to retrieve information based on the username that is being authenticated. Security Access Manager includes an attribute `oidc_username` which contains the authenticated username in all contexts: whether at /authorize, /token or /userinfo endpoints. For example, you can create a common LDAP Attribute Source, as shown in the following table.

Table 16. LDAP Attribute Source example

Type	LDAP
Attribute Name	LDAPMail
LDAP Attribute	mail
Server Connection	TestLDAP

Table 16. LDAP Attribute Source example (continued)

Type	LDAP
Scope	Subtree
Selector	uid
Search filter:	(uid={oidc_username})
Base DN:	dc=iswga

Saving Values or Parameters

Security Access Manager enables you to save values or parameters that are related to the claims at the /authorize endpoint.

For example, some request parameters can be specified at the /authorize endpoint, but the claims might be produced at the /token or /userinfo endpoint. The claims need to be saved so that they can be available at the /token or /userinfo endpoint. An example of this is the `claims_locales` request parameter.

Another example is the existence of values, such as credential information, that are available at the /authorize endpoint, but need to be output at the /token or /userinfo endpoint. An example of this is the values that are used in the credential type attribute sources. You can save these values. This overcomes the limitation of the attribute source such that for the credential-type attribute source, the context exists only at the /authorize endpoint, and if the values are not saved, the context is not available at any other endpoint.

You can do this by creating a context attribute in the STSUA of the type `urn:ibm:names:ITFIM:oidc:claim:parameter` for parameter or `urn:ibm:names:ITFIM:oidc:claim:value` for value.

The difference between these two types is that, during runtime, the claim value is put back into STSUA Attribute List under type `urn:ibm:names:ITFIM:5.1:accessmanager`, but the claim parameter is put in the STSUA context under type `urn:ibm:names:ITFIM:oidc:claim:parameter`.

Note that these saved values or parameters exist as long as the grant exists. When the grant is removed or expires, the saved values and parameters are removed.

For example, we can modify the pre-token mapping rule and add the following code to save the `AZN_CRED_PRINCIPAL_NAME` attribute.

```
var saveValue = stsuu.getAttributeContainer()
    .getAttributeValueByNameAndType("AZN_CRED_PRINCIPAL_NAME",
    "urn:ibm:names:ITFIM:5.1:accessmanager");
if (saveValue != null) {
    var attro = new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("AZN_CRED_PRINCIPAL_NAME",
    "urn:ibm:names:ITFIM:oidc:claim:value", saveValue);
    stsuu.getContextAttributes().setAttribute(attro);
}
```

By doing this, the `CredentialNickname` attribute source (as shown in the example earlier in this article) can be resolved in the /token or /userinfo endpoint.

Customizing claims

Security Access Manager provides, in pre-token and post-token mapping rule, examples of how to customize the claims.

For ID Tokens, customization is done in a pre-token mapping rule. You must enable the sample by setting the `customize_id_token` variable to `true`. For `UserInfo`, customization is done in a post-token mapping rule.

For the example, the algorithm used is the same. The first step is to retrieve the claim list. You might want to process essential claims and voluntary claims separately.

To resolve claims, the `produceClaim()` method is called. If an expected value, as specified by the claim, is found, use it. Next, try to find an attribute that has been resolved using Attribute Source mapping, and has the same name as the claim. Next, if still there is no value, for essential claims, you can set some value or optionally throw an STS exception. Note, however, that the OIDC specification does not recommend throwing an error even if an essential claim cannot be fulfilled.

For the final step, if the claim value exists, it is put into the STSUU. The type must be set correctly, in order to be included as part of ID Token or `UserInfo`. For ID Token, the type must be `urn:ibm:jwt:claim`. For `UserInfo`, the type must be `urn:ibm:names:ITFIM:oauth:response:attribute`.

Special parameter for UserInfo

You can customize `UserInfo` by using a special attribute. You can create an attribute in the STSUU Context, with name `userinfo` and type `urn:ibm:names:ITFIM:oauth:rule:userinfo`, with the value as a JSON string.

The purpose of this special parameter is to allow you to provide a complex value for `UserInfo`, in situations where the limitations of the STSUU structure prevent support for the value. For example, the following JSON defines an address that consists of two parts:

```
{ "firstname": "John",  
  "lastname": "Doe",  
  "address": {  
    "zipcode": "34234"  
    "city": "Newark"  
  }  
}
```

The JSON string provided in this parameter serves as the base JSON structure, and is added with other customization attributes, if they exist, to the output of the `/userinfo` endpoint.

Another use of the special parameter is when you want to generate `UserInfo` in JWT format. By default, the Security Access Manager implementation produces `UserInfo` in JSON format. If you need to produce `UserInfo` in JWT format, there is an example in the post-mapping rule.

To use the example in the post-mapping rule, complete the following steps:

1. Create an STS chain that includes the modules in the table below.

Table 17. Chain modules for JWT format

Module	Mode
Default STSOU Module Instance	Validate
Default Jwt Module	Issue
Default STSOU Module Instance	Issue

For more information on creating STS templates and chains, see Managing trust chains.

2. In the post mapping-rule, set `produce_jwt_userinfo` to `true`:
`var produce_jwt_userinfo = true;`

Client authentication to /token through an incoming JSON Web Token

Security Access Manager OIDC Providers support client authentication to /token through an incoming JSON Web Token (JWT).

Some deployment scenarios, such as Open Banking, require the use of a signed assertion as a method to replace `client_id` and `client_secret`. To view the implementer requirements for client authentication, see <https://www.openbanking.org.uk/read-write-apis/security-profile/id1-0-1/>.

Security Access Manager OIDC Providers support client authentication to /token through an incoming JSON Web Token (JWT). Security Access Manager support of client assertions satisfies RFC 7523. See <https://tools.ietf.org/html/rfc7523>.

Note: Support for authentication to /token with a JWT is different from support for request JWTs that are presented to /authorize. For /authorize, see “Passing parameters through JWT in a request to /authorize” on page 154.

When an incoming client assertion is detected by the presence of the parameters `client_assertion_type` (of a valid value) and `client_assertion`, the OAuth delegate invokes a token exchange. This token exchange is to a well-known (predictable) set of `issuer` and `appliesTo` values.

The JWT must contain the following claims:

iss

The issuer of the JWT. This value must be that of the security entity that created this JWT. Its presence is validated, but explicit validation of the value must be completed in the STS chain.

sub

Subject Identifier. This value must be the `client_id` you want to authenticate.

aud

Intended audiences for the ID Token. It must be a value that represents this entity, such as the API Protection definition name.

exp

Expiration time on or after which the JWT is not accepted for processing.

The parameter **nbf**, if present, is validated. This parameter is the "not before" claim that identifies the time before which the JWT is not accepted for processing.

You can create a Secure Token Service (STS) chain with modules to handle client assertions through incoming JWTs during authentication. To configure an STS chain that is compatible with incoming client assertions, the chain must ensure:

1. No token type is set.
2. RequestType of Validate is accepted.

Examples of ISSUER and APPLIESTO fields that handle all presented client assertions are as follows:

```
ISSUER="REGEXP:(urn:ietf:params:oauth:client-assertion-type:jwt-bearer:.*)"
APPLIESTO=https://localhost/sps/oauth/oauth20
```

Note: In the example above, all clients match with this chain. (Note the `.*` value in the regexp for the Issuer.) If a particular chain is needed, then use the `issuer:urn:ietf:params:oauth:client-assertion-type:jwt-bearer:myClient` where `myClient` is the `client_id` of the interested client.

- The issuer is a combination of the assertion type plus the client identifier.
- The appliesto is the federation name. For Security Access Manager, the federation name is always:

```
https://localhost/sps/oauth/oauth20
```

The client configured secret and `jwt_keys_uris` are included in the request to the STS through WS-Trust claims. To view how the JWT module supports validation, see Validate mode.

After the JWT is validated, OAuth expects a Secure Token Service Universal User (STSUU) in return, as follows:

- The sub claim is populated with the `client_id` of the incoming request.
- The aud field is checked against the configured issuer identifier of the API definition.

Note: The values of `iss` and `aud` must be validated. This validation can be done through the STS chain configuration, or in a map module in the STS chain, or in the pre-token mapping rule. Validation within the chain is easiest when the values of `iss` and `aud` are static.

All the claims in the JWT are mapped into the STSUU attribute list, with the type similar to `urn:ibm:oauth20:client:assertion`.

To implement this set of features, you must configure an STS chain with the following modules:

- JWT module in Validate mode >
- An optional JavaScript mapping rule >
- STSUU module in Issue mode

Following is an example of code for retrieving the values:

```
var sub = stsuu.getAttributeContainer().getAttributeValueByName("sub");
var aud = stsuu.getAttributeContainer().getAttributeValueByName("aud");
var iss = stsuu.getAttributeContainer().getAttributeValueByName("iss");
IDMappingExtUtils.traceString("sub: " + sub + " aud: " + aud + " iss: " + iss);
```

Passing parameters through JWT in a request to /authorize

Security Access Manager OIDC Providers support passing request parameters by way of a JWT in a request to `/authorize`.

The support satisfies the requirements in Section 6.1 of the OpenID Connect Core specification https://openid.net/specs/openid-connect-core-1_0.html.

Deployments such as web banking applications require integration with strong authentication as offered by a third party. This scenario requires that clients avoid providing claims directly in a query string. By sending the claims in a JSON Web Token (JWT), the client proves that it signed them with an established secret. Security Access Manager supports this function by providing decryption of OIDC ID tokens that are received with the authorization code grant flow from a third-party OpenID Provider. The received tokens are signed (for example, by presenting a JWT that uses the JWS algorithm RS256) and encrypted (for example, by use of the content encryption algorithm AES128CBC-HS256 and key agreement algorithm of RSA-OEAP). See <https://tools.ietf.org/html/rfc7518>.

A request JWT can be used to provide incoming request parameters. The specification requires that the OAuth request parameters must still contain `client_id` and `response_type`, but all other parameters, such as `redirect_uri`, and `scope`, can be provided only in the request JWT. The `client_id` and `response_type` can also be presented in the request JWT, but the values of each must match those that are provided in the OAuth request parameters.

To configure an STS chain that is compatible with incoming JWT request parameters, the chain must meet the same requirements as required for handling client assertions:

1. No token type is set.
2. RequestType of Validate is accepted.

Examples of ISSUER and APPLIESTO fields that handle all presented client assertions are as follows:

```
ISSUER="REGEXP:(urn:ibm:ITFIM:oauth20:client_request:.*)"
APPLIESTO=https://localhost/sps/oauth/oauth20
```

Note: In the example above, all clients match with this chain. (Note the `.*` value in the regexp for the Issuer.) If a particular chain is needed, then use the `issuer:urn:ietf:params:oauth:client-assertion-type:jwt-bearer:myClient` where `myClient` is the `client_id` of the interested client.

- The APPLIESTO is the federation ID.
- The ISSUER must be the string "urn:ibm:ITFIM:oauth20:client_request:", and the `clientId` that is included in the request.

The claims are mapped into the request. Validation occurs on `request_type` and `client_id`, as required by the specification:

- Validating JWT-based Requests

When the request authorization parameter is used, the JWT is passed to the auxiliary chain, and the returned claims are mapped back into the request STSUU. The response type of the auxiliary chain must be STSUU. If signature validation fails, the request is rejected.

- Request Parameter Assembly and Validation

The Authorization Server must assemble the set of Authorization Request parameters to be used from the Request Object value and the OAuth 2.0 Authorization Request parameters (minus the request parameters). If the same parameter exists both in the Request Object and the OAuth Authorization Request parameters, the parameter in the Request Object is used. Using the

assembled set of Authorization Request parameters, the Authorization Server then validates the request in the normal manner for the flow that is being used.

Mapping rules for OAuth and OIDC

Security Access Manager provides default mapping rules that you can use and customize for your OAuth or OIDC deployment.

Managing OAuth 2.0 and OIDC mapping rules

Use the mapping rules to customize the methods for the OAuth 2.0 or OIDC flow.

About this task

The OAuth 2.0 and OIDC mapping rules are JavaScript code that run during the OAuth 2.0 or OIDC flow. You can view, export, and replace OAuth or OIDC mapping rules.

View the mapping rule if you want to see the content and structure of the mapping rule. Export the mapping rule if you want to save a copy of the mapping rule. You can also edit this copy. Replace a mapping rule if you want to use a new mapping rule.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control > Policy > OpenID Connect and API Protection** or **Secure Federation > Manage > OpenID Connect and API Protection**.
3. Click **Mapping Rules**.
4. Perform one or more of the following actions:

View a mapping rule

- a. Select a mapping rule.
- b. Click . The View Mapping Rule panel opens. The content of the mapping rule is displayed.
- c. Click **OK** to close the panel.

Export a mapping rule

- a. Select a mapping rule.
- b. Click .
- c. Choose a location and save the file.

Replace a mapping rule:

Note: Use an existing mapping rule as the basis for the updated mapping rule.

- a. Select a mapping rule that you want to replace.
 - b. Click . The Replace Mapping Rule panel opens.
 - c. Click the field or **Browse** and select a file.
 - d. Click **OK** to upload the mapping rule.
5. When you replace a mapping rule, the appliance displays a message that there are undeployed changes. If you are finished with the changes, deploy them.

For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Related reference:

“OAuth 2.0 and OIDC mapping rule methods”

You can use Java methods to customize the `PreTokenGeneration` and `PostTokenGeneration` mapping rules.

OAuth 2.0 and OIDC mapping rule methods

You can use Java methods to customize the `PreTokenGeneration` and `PostTokenGeneration` mapping rules.

The sample mapping rules are `oauth_20_pre_mapping.js` and `oauth_20_post_mapping.js`.

You can access the sample mapping rules from the LMI. Navigate to **Manage System Settings > Secure Settings > File Downloads**. Continue to either of the following locations:

- **access_control > examples > mapping rules**
- **federation > examples > mapping rules**

The following limitations affect the attribute keys and values that are associated with the `state_id` by using the `OAuthMappingExtUtils` class:

- Keys cannot be null or empty.
- Values cannot be null but can be empty.
- Associated key-value pairs are read and write-allowed and not-sensitive.
- Some keys are reserved for system use and cannot be modified by this utility. For example, the keys and values for the API PIN protection.

For more information, see the Javadoc. In the LMI, navigate to **Manage System Settings > Secure Settings > File Downloads**. Continue to either **access_control > doc** or **federation > doc**.

See also “JavaScript whitelist” on page 297.

OAuth and OIDC mapping rules files

In OAuth and OpenID Connect deployments, you can use mapping rules to customize your use of Security Access Manager features.

Security Access Manager provides template mapping rules that you can use when configuring OAuth and OpenID Connect deployments. For OIDC, the rules are automatically included when you create an OIDC API Protection definition. One mapping rule is used pre-token generation. The other mapping rule is used post-token generation.

Note: If you created API definitions in a prior release of Security Access Manager, and updated to Version 9.0.4, you have the option to enable OIDC. However, enabling OIDC and saving the definition does not update the mapping rules. You can manually update the mapping rules by following the instructions in “Updating mapping rules when enabling OIDC” on page 165.

Table 18. Mapping Rules

Mapping Rule	Supported Actions
oauth_20_pre_mapping.js	• Use a user registry for verification of the username and password for the ROPC scenario. Optionally, force sourcing the ROPC password validation config from ldap.conf. • Show an example of the ROPC scenario using an external service for verification of the username and password. • Limit the number of tokens per user per client, and specify the algorithm to use. • Customize ID Token • Specify whether to only allow confidential clients to introspect or revoke tokens • Discover the request_type and the grant type. • Limit the number of grants per user per client. • Enable a token lookup example. • Enable custom tokens • Enable assertion grants • Calling additional STS chains
oauth_20_post_mapping.js	• Associate attributes • Deletetokens • Makean HTTP(S) callout • Update a token • Register an Authenticator for MFA • Enforce that clients are only introspecting their own tokens • UserInfo Customization • Produce JWT UserInfo • Call additional STS chains • Return additional attributes to the user via response attributes.

OAuth and OIDC mapping rules actions

You can modify mapping rules to perform actions that you need for your OAuth or OIDC deployment.

This topic provides details for some of the actions you can take to modify the mapping rules. For information on more actions, see the comments in the code for each mapping rule.

Note that for certain grant types, some actions must be performed in the pre-token mapping rule.

See “Managing OAuth 2.0 and OIDC mapping rules” on page 156 for instructions on replacing a mapping rule after you make the updates to the file.

Resource owner password credentials (ROPC) grant type flow

For the ROPC flow, the pre-token mapping rule is responsible for performing validation of the user name and password. This validation can be performed in various ways. The pre-defined rule that is included with the appliance provides the following examples:

- Use `UserLookupHelper` to validate a user name and password against a configured LDAP. You can also use the java class **PluginUtils** but it is limited.

To configure the LDAP to be used, see “Configuring username and password authentication” on page 54.

- Validate the user name and password through an HTTP callout. The mapping rule sends the user name and password to a web service. As the format of the messages is not fixed, various services (for example, REST, SOAP, SCIM) can be used for this purpose. Javadoc on the HTTP client and all other exposed Java classes available in mapping rules can be downloaded from the appliance **File Downloads** page under the path **access_control > doc > ISAM-javadoc.zip**.

JWT and SAML bearer grant type flow

For the JWT or SAML assertion bearer grant type flows, the pre-token mapping rule must perform the following actions:

- Validate the assertion, including but not limited to:
 - Validate the signature (if signed).
 - Decrypt the assertion (if encrypted).
 - Check the expiry and "not before" value of the assertion.
 - Ensure that the issuer is a trusted party.
- Extract the subject from the assertion and set the **USERNAME** field of the STSUU.

The **USERNAME** field of the STSUU can be set via a call, for example:

```
// username is a variable containing the subject of the assertion
```

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute(
  ("username","urn:ibm:names:ITFIM:oauth:rule:decision", username));
```

The validation of the assertion can be performed in various ways:

- HTTP callout to a web service. Use the HTTP client to perform this.
- WS-Trust request to the Secure Token Service (STS).
 - A chain must be configured to consume the assertion and return the required information.
 - The **STSClientHelper** will be called to invoke the STS via HTTP. For more information about this class, see the Javadoc that is embedded in the appliance.

Any attributes of the assertion can be extracted and associated to the OAuth grant to be used later. For more information about associating attributes, see “OAuth 2.0 and OIDC mapping rule methods” on page 157.

- The type of the username attribute added must be "urn:ibm:names:ITFIM:oauth:rule:decision" to ensure that only a value populated from the rule is used.

OAuth 2.0 token limits

You can define limits on the number of OAuth tokens per user per definition so that the high-volume database does not go beyond capacity.

Security Access Manager for Mobile has a thread that runs at a specified interval defined by the advanced configuration property, `oauth20.tokenCache.cleanupWait`. This property defines the amount of time, in seconds, to wait before it performs another cleanup of expired grants and tokens in the OAuth 2.0 token cache.

Depending on the interval and use of OAuth grants and tokens, there is a possibility that the capacity of the high-volume database can be reached before the cleanup process runs. If this happens, the appliance can be negatively impacted.

To prevent issues such as this, the OAuth PreTokenGeneration mapping rule, by default, limits the number of OAuth tokens per user per client definition. When a user requests an OAuth token, the current number of tokens for that user and the specified client definition will be compared to the maximum allowed. If the maximum is exceeded, an error message is returned to the user.

An additional algorithm implements least recently used (LRU) and, if the maximum is exceeded, the least recently used token determined by the date last used for that user and client definition will be removed from the high-volume database.

You can set the limit and algorithm to use, which are controlled by variables in the mapping rule file. Two algorithms are implemented in this mapping rule:

- Strictly enforce the limit.
- When the limit is reached, remove the least recently used tokens for the user per client.

Update the OAuth PreTokenGeneration mapping rule, `oauth_20_pre_mapping.js` to modify the algorithms. See the comments in the code for an explanation of the values you can modify.

See “Updating PreTokenGeneration to limit OAuth tokens” on page 7 so that any API protection definitions you created in versions prior to 8.0.1.2 can take advantage of these limits.

Customizing OAuth tokens by updating the sample PreTokenGeneration mapping rule

You can customize the format of the tokens that are issued by your OAuth definition.

The OAuth tokens can be customized by modifying the sample PreTokenGeneration mapping rule. Enable the PreTokenGeneration mapping rule on the appliance by setting the variable `enable_custom_tokens` to true.

When custom token formats are used, the tokens must remain unique. Otherwise, users might become authenticated with another user's credential. Thus, it is recommended that custom tokens always contain a nonce of reasonable entropy.

To customize the authorization code, insert a context attribute into the STSUA with the type `"urn:ibm:ITFIM:oauth20:custom:token"` and the name `"urn:ibm:ITFIM:oauth20:custom:token:authorization_code"`. The provided value will be used as the authorization code if an authorization code would have been issued in this request.

To customize the access token, insert a context attribute into the STSUA with the type `"urn:ibm:ITFIM:oauth20:custom:token"` and the name `"urn:ibm:ITFIM:oauth20:custom:token:access_token"`. The provided value will be used as the access token if an access token would have been issued as part of this request.

To customize the refresh token, insert a context attribute into the STSUA with the type `"urn:ibm:ITFIM:oauth20:custom:token"` and the

name="urn:ibm:ITFIM:oauth20:custom:token:refresh_token". The provided value will be used as the refresh token if a refresh token would have been issued as part of this request.

Returning additional attributes in responses

If you want to return additional attributes to an OAuth response, modify the pre or post token mapping rule to add an attribute with the name and value of the desired response attribute, and specify the well-known type urn:ibm:names:ITFIM:oauth:response:attribute. You can also use the post token mapping rule to modify any response attributes which are already included.

Here is an example of adding a new response attribute in JavaScript.

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("myAdditionalAttribute", "urn:ibm:names:ITFIM:oauth:response:attribute", "myValue"));
```

If you want to return a multi-valued attribute, provide a Java array as the value. Here is an example of constructing and returning a multi-valued response attribute.

```
var javaArray = java.lang.reflect.Array.newInstance(java.lang.String, 1);
javaArray[0] = 'myValue';
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("myAdditionalAttribute", "urn:ibm:names:ITFIM:oauth:response:attribute", javaArray));
```

Here is an example of an access_token response using the same examples:

```
{
  "access_token": "afcddfegedffdeeabb",
  "refresh_token": "cbbdegfcgcfgfddcbdeafadaccegaabeebeaagd",
  "scope": "scope1",
  "myAdditionalAttribute": "myValue",
  "token_type": "bearer",
  "expires_in": 3589
}

{
  "access_token": "gdbafbcgffcgfggeccb",
  "refresh_token": "dfgagedgbeddeebbedcdacegdacccccfabcee",
  "scope": "scope1",
  "myAdditionalAttribute": [
    "myValue",
    "mySecondValue"
  ],
  "token_type": "bearer",
  "expires_in": 3589
}
```

This action is commonly performed for the response types:

- authorize
- access_token
- introspect
- userinfo
- revoke

Note: when used for revoke, JSON will be returned rather than a 200 with no body.

Customize ID Tokens

For OIDC deployments, the post-mapping rule `oauth_20_pre_mapping.js` provides examples for how to customize ID tokens. When populating an

ID token, the rule processes essential claims and voluntary claims separately so that they are treated appropriately if they have no value. In the STSUU, the attribute's name is the claim name and the attribute's value(s) are the expected value of the claim. See the mapping rule for more information.

Customize UserInfo

For OIDC deployments, the post-mapping rule `oauth_20_post_mapping.js` provides examples for how to customize UserInfo based on OIDC scope and claims request parameters. In the STSUU context, the claims are listed in terms of essential and voluntary claims. When `AttributeSources` are configured in the definition, they too are resolved and available in the STSUU. This is one way of doing customization.

Produce JWT UserInfo

For OIDC deployments, the post-mapping rule `oauth_20_post_mapping.js` provides examples for how to produce JWT UserInfo. In the STSUU context, the signing and encryption data (based on the OP Definition) are available. To create JWT, you can call an STS Chain which has 2 modules: 1) Default STSUU validation module, 2) Default JWT issuer module. You must create this chain, it is not supplied by default. The chain passes the signature and encryption data and all the JWT claims. The JWT token result then needs to be set back in the STSUU under specific name and type. See the mapping rule file for more information.

Modifying JWT signing and encryption parameters in the pre-token mapping rule

This action applies only when OIDC is enabled.

When using OIDC, the JSON Web Token (JWT) configuration is sourced from the OAuth definition. However, an administrator may want to write logic which at runtime augments how a JWT is formed. For example, specifying a different certificate when signing JWTs for a specific client or specific type of client. You can take this action within the pre-token mapping rule for the definition.

The JWT STS module is used to build the JWT that is returned in the OIDC flow. This module allows a well-known set of claims to be provided at runtime. For documentation on those values, see Issue mode.

When making use of the issue context attributes defined in the JWT STS module, the values must be provided by using a different and specific attribute type. When calling the STS, the STS Universal User (STSUU) that is provided is just for a JWT. For OIDC, the STSUU is an encapsulation of the HTTP request. To provide a custom property for building a JWT, set a context attribute with the name and attribute as described in the "Issue mode" link, but use the attribute type `urn:ibm:oidc10:jwt:create`. If you want to set a JWT claim or header claim (not how the actual JWT is formed), use the types `urn:ibm:jwt:claim` or `urn:ibm:JWT:header:claim`, respectively.

Note: The Security Token Service Universal User (STSUU) document is an XML representation of a request that passes through a trust module chain in the STS. The three elements in the STS Universal User document are Principal, AttributeList, and RequestSecurityToken. For more information on the role of the STSUU in identity mapping, see Security Token Service Universal User document.

The following example ensures that the JWT that is issued uses HS256 signing with the provided key. This action overrides the configuration in the OAuth definition.

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute(
    "signing.alg", "urn:ibm:oidc10:jwt:create", "HS256"));
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute(
    "signing.symmetricKey", "urn:ibm:oidc10:jwt:create", "myKey"));
```

The following code adds an extra header to the JWT (in this case, the cty field).

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("cty",
    "urn:ibm:JWT:header:claim", "JWT"));
```

The following code adds a static claim to the JWT claims.

```
stsuu.addAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("myClaim" ,
    "urn:ibm:jwt:claim", "claimValue"));
```

Here is a raw JWT produced with all 4 of the above examples enabled.

```
eyJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJub25jZSI6InNvbWV0b25jZTE0
NzUxIiwiaWF0IjoxNTA2NDkyNTAyLCJpc3MiOiJodHRwczovL3Rlc3REZWYuY29tI
iwic3ViIjoiaGVzZXIiLCJleHAiOjE1MDY0OTYxMDIsIm15Q2xhaWoiOiJjbG
FpbVZhbHVlIiwiaXVkiOiJoaXBxX10ZXN0Q2xpZW50In0.ZJdBmJtVw_Ti3QjnpI21HWl
Yk-asu72UnosYBZXeRn4
```

The decoded header:

```
{
  "cty": "JWT",
  "alg": "HS256"
}
```

The claims:

```
{
  "nonce": "someNonce14751",
  "iat": 1506492502,
  "iss": "https://testDef.com",
  "sub": "testuser",
  "exp": 1506496102,
  "myClaim": "claimValue",
  "aud": "mytestClient"
}
```

Customizing OAuth tokens by updating the sample PreTokenGeneration mapping rule

You can customize the format of the tokens that are issued by your OAuth definition.

The OAuth tokens can be customized by modifying the sample PreTokenGeneration mapping rule. Enable the PreTokenGeneration mapping rule on the appliance by setting the variable *enable_custom_tokens* to true.

When custom token formats are used, the tokens must remain unique. Otherwise, users might become authenticated with another user's credential. Thus, it is recommended that custom tokens always contain a nonce of reasonable entropy.

To customize the authorization code, insert a context attribute into the STSUU with the type "urn:ibm:ITFIM:oauth20:custom:token" and the name

"urn:ibm:ITFIM:oauth20:custom:token:authorization_code". The provided value will be used as the authorization code if an authorization code would have been issued in this request.

To customize the access token, insert a context attribute into the STSUU with the type "urn:ibm:ITFIM:oauth20:custom:token" and the name "urn:ibm:ITFIM:oauth20:custom:token:access_token". The provided value will be used as the access token if an access token would have been issued as part of this request.

To customize the refresh token, insert a context attribute into the STSUU with the type "urn:ibm:ITFIM:oauth20:custom:token" and the name "urn:ibm:ITFIM:oauth20:custom:token:refresh_token". The provided value will be used as the refresh token if a refresh token would have been issued as part of this request.

To customize the device code, insert a context attribute into the STSUU with the type "urn:ibm:ITFIM:oauth20:custom:token" and the name "urn:ibm:ITFIM:oauth20:custom:token:device_code". The provided value will be used as the device_code in the device flow.

To customize the user code, insert a context attribute into the STSUU with the type "urn:ibm:ITFIM:oauth20:custom:token" and the name "urn:ibm:ITFIM:oauth20:custom:token:user_code". The provided value will be used as the user_code in the device flow. Customizing the user_code is particularly important, as the end user will be required to key this into a user agent –potentially on a mobile device, where entering long strings can be cumbersome. There is an out of the box example of this configured by default to make the format of the user code xxxx-xxxx.

OpenID Connect mapping rules

Mapping rules allow users to customize the information that is propagated from an OpenID Connect Provider or what is consumed by a Relying Party.

These mapping rules can either be JavaScript, which is invoked internally via the STS, or the mapping can be performed externally via a HTTP request.

OpenID Connect Provider mapping rules

When you write mapping rules for a provider, the primary goal is to augment the claims that are included in the ID token.

After mapping rule execution, all attributes in the STSUU will be added to the id_token as a claim, where the attribute key is the key in the id_token, and the value is the value of the attribute. If there are several attributes with the same key, then an array containing each attribute will be added to the claim. Some context information is made available to the user when writing mapping rules; the context attributes of the passed in STSUU will contain attributes with the type "urn:ibm:ITFIM:oidc:provider:context", which can be used to make decisions on what claims are added, or if any other actions are performed.

These context attributes include:

- The client ID of the client making the request.
- The federation name of the provider servicing the request.
- The redirect URI sent in the request.
- The response type of the request.

- The state parameter of the request.
- The user-consented scopes for the request.

OpenID Connect Relying Party mapping rules

When you write mapping rules for a Relying Party, the resulting STSUI is turned into a PAC that is used to authenticate the user to a Reverse Proxy via EAI.

The attributes that are included in that PAC will be the attributes of the STSUI, and the principal will be the first principal which was in the STSUI. When writing mapping rules for a Relying Party, the values of the `id_token` will be made available as Attributes in the STSUI. Some additional context is made available to the user via the STSUI's context attributes. These attributes will have the types `"urn:ibm:ITFIM:oidc:client:token:param"` and `"urn:ibm:ITFIM:oidc:client:token:param"`.

These context attributes include:

- All of the claims inside the `id_token`.
- The raw JWT.
- Any issued access or refresh tokens.
- All of the properties of the issued bearer token if an authorization code flow is used.
- All of the parameters issued in the response if an implicit flow is used.

Attribute sources

Both OpenID Connect Providers and Relying Parties can be configured to use an attribute source.

For an OpenID Connect Provider, this can be used instead of a mapping rule. However for an OpenID Connect Relying Party a mapping rule must still be present, this mapping rule is required to construct the principal used in the `iv-cred`.

For more information about attribute sources, see [Managing attribute sources](#).

Updating mapping rules when enabling OIDC

You can update the default mapping rules for OIDC to enable and customize mapping actions.

About this task

Security Access Manager provides mapping rules for use with OAuth 2.0 and OIDC deployments. You can access these files from the File Downloads section of the LMI. You can then update these files as appropriate for your deployment.

For Version 9.0.4, Security Access Manager supports new OIDC request types, as described in the following table.

Table 19. New request types

Request type	Associated endpoint	Description
userinfo	https://server.oauth.com/mga/sps/oauth/oauth20/userinfo	See "OAuth 2.0 endpoints" on page 108 and "OIDC Claims customization" on page 147

Table 19. New request types (continued)

Request type	Associated endpoint	Description
revoke	https://server.oauth.com/mga/sps/oauth/oauth20/revoke	See “OAuth 2.0 endpoints” on page 108 and “OAuth revocation endpoint” on page 146
introspect	https://server.oauth.com/mga/sps/oauth/oauth20/introspect	Support for introspect was added in Version 9.0.3. See “OAuth 2.0 endpoints” on page 108 and “OAuth introspection” on page 144

Procedure

1. In the LMI, go to **Manage System Settings > File Downloads**
2. Expand either **federation > examples > mapping rules** or **access_control > examples > mapping rules**
3. Select one or more of the following files and click **Export**.
 - oauth_20_pre_mapping.js
 - oauth_20_post_mapping.js
4. Edit the mapping rule as appropriate for your deployment.
5. Import the revised mapping rule into your OIDC API Protection definition.
 - a. Select either **Secure Access Control > Policy > OpenID Connect and API Protection** or **Secure Federation > Manage > OpenID Connect and API Protection**.
 - b. Select the **Mapping Rules** sub-menu.
 - c. Click **Import** to add a new mapping rule. Or, to use an edited mapping rule to replace an existing mapping rule, highlight the existing mapping rule and click **Replace**.

Device flows verification uri

The device flow requires the client be presented with a verification_uri. Because the exact details of the point of contact are not known by the authorization server and the response is written back in JSON not HTML, this value needs to be changed per deployment.

In order to update the verification_uri shown in the response from device_authorize, update the variable webseal_portion to be the new protocol, hostname, port, and junction to user in the URI.

OAuth 2.0 template files

The OAuth process relies on HTML pages to interact with users, such as displaying errors or prompting users to provide information. You can customize these pages through the OAuth template files.

For information about OAuth template files, see Template files.

OAuth 2.0 template page for consent to authorize

The authorization server uses this page to determine and store user consent information about which OAuth clients are authorized to access the protected resource. This page also indicates scopes that the OAuth client requests.

The Security Access Manager for Mobile provides an HTML page template called `user_consent.html`. The macros in the template are specifically for an OAuth 2.0 flow.

Note: You can use a separate template for each API definition. To add a template for a specific definition, create a directory with the same name as the definition under `oauth20` and add the `user_consent.html` template there.

Security Access Manager for Mobile stores the decisions made by the resource owner about which OAuth clients to trust. The resource owner is not prompted every time the same OAuth client requests authorization to access the protected resource.

The authorization request from the OAuth client shows a list of approved scopes, and a list of scopes to be approved. These lists are shown in the consent page and can be of indeterminate length. The template supports multiple copies of stanzas that are repeated once for each scope in either list.

This template file provides several replacement macros:

@OAUTH_AUTHORIZE_URI@

This macro is replaced with the URI for the authorization endpoint.

@OAUTH_CLIENT_COMPANY_NAME@

This macro is replaced with the name of the company that is requesting access the protected resource.

@CLIENT_ID@

This macro is replaced with the `client_id` parameter specified in the authorization request.

@REDIRECT_URI@

This macro is replaced with the redirect URI that the authorization server uses to send the authorization code to. The value depends on the following items:

- Redirect URI that is entered during partner registration
- `oauth_redirect` parameter specified in the authorization request

@STATE@

This macro is replaced with the state parameter specified in the authorization request.

@RESPONSE_TYPE@

This macro is replaced with the `response_type` parameter specified in the authorization request.

@OAUTH_CLIENT_DATA_MACRO@

This macro is replaced with the client data in JSON format, which contains values that are entered at configuration time such as:

- Company name
- Company URL
- Contact name

- Email address
- Telephone number
- Contact type
- Other information

This macro is also the dynamic data of the client. This includes any statistically configured client values such as **Company name**, and any dynamic values, regardless of whether they are from a dynamically registered client or from an extended client portion. For example, `tos_uri`

The fields are sanitized through a filter list. To populate or filter a specific value, change the advanced configuration **oauth20.clientDataToInclude**.

@USERNAME@

This macro is replaced with the Security Access Manager for Mobile user name.

@OAUTH_OTHER_PARAM_REPEAT@

A multi-valued macro that belongs inside a [RPT `oauthOtherParamsRepeatable`] repeatable replacement list. The values show the list of extra parameter names.

@OAUTH_OTHER_PARAM_VALUE_REPEAT@

A multi-valued macro that belongs inside a [RPT `oauthOtherParamsRepeatable`] repeatable replacement list. The values show the list of extra parameter values.

@OAUTH_TOKEN_SCOPE_REPEAT@

A multi-valued macro that belongs either inside [RPT `oauthTokenScopePreapprovedRepeatable`] or [RPT `oauthTokenScopeNewApprovalRepeatable`] repeatable replacement lists. The values inside the [RPT `oauthTokenScopePreapprovedRepeatable`] show the list of token scopes that have been previously approved by the resource owner. Alternatively, the values inside the [RPT `oauthTokenScopeNewApprovalRepeatable`] show the list of token scopes that have *not* yet been approved by the resource owner.

@CONSENT_FORM_VERIFIER@

This macro is replaced with a unique identifier for the `consent_form_verifier` parameter value. The `consent_form_verifier` parameter value is automatically generated by the authorization server. The parameter name and value must not be modified.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>OAuth 2.0 - Consent to Authorize</title>
<link rel="stylesheet" type="text/css" href="/sps/static/styles.css" />
</head>
<body>
<div class="header">
<div class="brandingLogo"></div>
</div>
<div class="content">
<div class="contentHeader">
<h1 class="pageTitle">OAuth 2.0 - Consent to Authorize</h1>
<div class="instructions"></div>
</div>

<div class="pageContent">
<form action="@OAUTH_AUTHORIZE_URI@" method="post">

<p>The following site is requesting access to an OAuth 2.0 protected resource:</p>
<div class="sectionTitle">
<p><b>@OAUTH_CLIENT_COMPANY_NAME@</b></p>
</div>

<p>The client type is: @CLIENT_TYPE@</p>
<br/>
<p>The client provided the following OAuth 2.0 request parameters:</p>
<br/>
<ul style="margin-left: 20px">
<li>Client Id: @CLIENT_ID@</li>
<li>Redirect URI: @REDIRECT_URI@</li>
<li>State: @STATE@</li>
<li>Response Type: @RESPONSE_TYPE@</li>
</ul>
<br/>
<p>By approving this request you will be providing delegated authorization
on behalf of:</p>
<p><b>@USERNAME@</b></p>
<br/>

<p>The client provided the following extra request parameters:</p>
<!-- START NON-TRANSLATABLE -->
<ul style="margin-left: 20px">
[RPT oauthOtherParamsRepeatable]
<li>@OAUTH_OTHER_PARAM_REPEAT@=@OAUTH_OTHER_PARAM_VALUE_REPEAT@</li>
<input type="hidden" name="@OAUTH_OTHER_PARAM_REPEAT@"
value="@OAUTH_OTHER_PARAM_VALUE_REPEAT@" />
[ERPT oauthOtherParamsRepeatable]
</ul>
<!-- END NON-TRANSLATABLE -->
<br/>
<p>The client requested the following token scopes that have been previously approved:</p>
<!-- START NON-TRANSLATABLE -->
<ul style="margin-left: 20px">
[RPT oauthTokenScopePreapprovedRepeatable]
<li>@OAUTH_TOKEN_SCOPE_REPEAT@</li>
<input type="hidden" name="scope" value="@OAUTH_TOKEN_SCOPE_REPEAT@" />
[ERPT oauthTokenScopePreapprovedRepeatable]
</ul>
<!-- END NON-TRANSLATABLE -->
<br/>
<p>The client requested the following token scopes that have not yet been approved:</p>
<!-- START NON-TRANSLATABLE -->
[RPT oauthTokenScopeNewApprovalRepeatable]
<input type="checkbox" name="scope" value="@OAUTH_TOKEN_SCOPE_REPEAT@" checked="checked" />
<label>@OAUTH_TOKEN_SCOPE_REPEAT@</label><br />
[ERPT oauthTokenScopeNewApprovalRepeatable]
<!-- END NON-TRANSLATABLE -->
</p>
<br />
<p>Would you like to approve access to this scope?</p>
<br/>
<input type="hidden" name="consent_form_verifier" value="@CONSENT_FORM_VERIFIER@" />

<!--
The scope parameters can be:
1. Requested as part of the redirect for authorization by the client
   by appending them to the authorize URL as query string parameters, and/or
2. If not requested by the client, and you know what authorization options
   are valid for the protected resources being requested, you may
   also manually prompt for them in this page template as demonstrated
   by the following example scope's
-->
<!--
<table>
<tr>
<td>Scopes to be authorized:&nbsp;</td>
<td>Scope 1</td><td><input type="checkbox" name="scope" value="token_scope_1" /></td>
<td>:: Scope 2</td><td><input type="checkbox" name="scope" value="token_scope_2" /></td>
<td>:: Scope 3</td><td><input type="checkbox" name="scope" value="token_scope_3" /></td>
</tr>
</table>
-->

<table>
<tr>
<td>Permit&nbsp;</td>
<td><input type="radio" name="trust_level" value="permit" checked="checked" /></td>
</tr>
<tr>
<td>Deny&nbsp;</td>
<td><input type="radio" name="trust_level" value="deny" /></td>
</tr>
</table>
<br />

<div class="controls">
<input class="submitButton" type="submit" name="submit" value="Submit" style="width: 80px" />
</div>
</form>
</div>
</div>
</body>
</html>

```

Figure 3. Template for user_consent.html

Error responses

An HTTP response indicates the type of error that has occurred when an action in an authorization process fails. The error responses described here are only applicable to Policy Enforcement Point (PEP) error responses.

For more information about OAuth 2.0 error responses for other endpoints, see the OAuth website: <http://www.oauth.net>.

In some circumstances, the following HTTP error responses must be returned to the client:

- 400 Bad Request
- 401 Unauthorized
- 502 Bad Gateway

For the 401 response, an additional WWW-Authenticate header is added to the response in the following format:

WWW-Authenticate: OAuth realm = <realm-name>

The HTML component of the responses is preinstalled from files that have been specified in the EAS configuration.

For details on how to configure the response template files for OAuth EAS, see *Configuring WebSEAL to include OAuth decisions*.

User self-administration tasks for OAuth

Administrators can configure OAuth to enable users to perform certain self-management tasks.

A common user task is to manage authorization grants. For example, users can view the attributes of an authorization grant. A user can also enable an authorization grant.

Managing OAuth 2.0 authorization grants

You can view your authorization grants and the tokens and attributes of each authorization grant.

About this task

You can complete the following tasks:

- View a list of your OAuth 2.0 authorization grants.
- View the OAuth 2.0 tokens and attributes of an authorization grant.
- Remove an OAuth 2.0 authorization grant.
- Enable an OAuth 2.0 authorization grant.
- Disable an OAuth 2.0 authorization grant.

Procedure

Take one of the following actions:

View your OAuth 2.0 authorization grants and the tokens and attributes of each authorization grant

1. Log in to http://hostname/mga/sps/mga/user/mgmt/html/device/device_selection.html.
2. Click the ID from the table to view the tokens and attributes of that authorization grant.

Note: You can also use the following URL to go directly to the tokens and attributes of a specific authorization grant: http://hostname/mga/sps/mga/user/mgmt/html/device/grant_attributes.html?id=x.

The query string, `id=x`, indicates the authorization grant that you are trying to access. The `x` represents the ID of the authorization grant.

Remove an OAuth 2.0 authorization grant

1. Log in to http://hostname/mga/sps/mga/user/mgmt/html/device/device_selection.html.
2. Click **Remove** next to the authorization grant that you want to remove.

Enable an OAuth 2.0 authorization grant

1. Log in to http://hostname/mga/sps/mga/user/mgmt/html/device/device_selection.html.
2. Select the **Enabled** box next to the authorization grant that you want to enable.

Disable an OAuth 2.0 authorization grant

1. Log in to http://hostname/mga/sps/mga/user/mgmt/html/device/device_selection.html.
2. Clear the **Enabled** box next to the authorization grant that you want to disable.

Note: Authorization grants can be enabled, disabled, or removed in the authorization grant attribute page too.

APIs for managing OAuth 2.0 authorization grants

There are two API endpoints that are available to manage a user's grants. These endpoints are useful for building an SPA or custom USC.

The first endpoint allows listing all of a user's grants.

Issue a HTTP GET to: <http://server.oauth.com/mga/sps/mga/user/mgmt/grant>. The API responds with:

```
{
  "grants": [
    {
      "id": "uuid8f63b7ee-0169-1c05-a78c-af253b6a2308",
      "isEnabled": true,
      "clientId": "client1",
      "tokens": [
        {
          "type": "authorization_grant",
          "subType": "refresh_token",
          "dateCreated": "2019-03-18T06:01:11Z",
          "lifetime": 604799,
          "lastUsed": "2019-03-18T06:01:11Z",
          "scope": "openid,email"
        }
      ]
    }
  ],
}
```

```

    "attributes": [
      {
        "name": "attribute1",
        "readonly": false,
        "sensitive": false,
        "value": "123"
      },
      {
        "name": "attribute2",
        "readonly": false,
        "sensitive": false,
        "value": "456"
      }
    ],
    "clientName": "client1"
  },
  ...
],
"username": "testuser"
}

```

The second endpoint allows operations on a per grant basis. This endpoint requires the grant-id to be known, the API documented above includes the grantId.

To use this endpoint, issue a HTTP GET to: <http://server.oauth.com/mga/sps/mga/user/mgmt/grant/{grantId}>. The API responds with the grant:

```

{
  "id": "uuid8f63b7ee-0169-1c05-a78c-af253b6a2308",
  "isEnabled": true,
  "clientId": "client1",
  "tokens": [
    {
      "type": "authorization_grant",
      "subType": "refresh_token",
      "dateCreated": "2019-03-18T06:01:11Z",
      "lifetime": 604799,
      "lastUsed": "2019-03-18T06:01:11Z",
      "scope": "openid,email"
    }
  ],
  "attributes": [
    {
      "name": "attribute1",
      "readonly": false,
      "sensitive": false,
      "value": "123"
    },
    {
      "name": "attribute2",
      "readonly": false,
      "sensitive": false,
      "value": "456"
    }
  ],
  "clientName": "client1"
}

```

This endpoint also supports a HTTP DELETE to remove a grant. Issue a HTTP delete to <http://server.oauth.com/mga/sps/mga/user/mgmt/grant/{grantId}>.

The attributes can also be updated when they are not read-only. Issue a HTTP PUT to <http://server.oauth.com/mga/sps/mga/user/mgmt/grant/{grantId}> , with the body:


```

{
  "isEnabled": true,
  "attributes": [
    {
      "name": "attribute1",
      "value": "newvalue1"
    },
    {
      "name": "attribute2",
      "value": "newvalue2"
    }
  ]
}

```

OAuth STS Interface for Authorization Enforcement Points

Use the WS-Trust interface to directly contact an OAuth Security Token Service (STS) trust chain in Security Access Manager to validate a request for an OAuth protected resource. An OAuth enforcement point intercepts requests for OAuth protected resources. The OAuth enforcement point also validates the request with Security Access Manager, and passes the request through, if it is valid. If the request is not valid, the enforcement point denies access to the protected resource.

OAuth STS overview

You can develop your own customized policy enforcement point to work with the Security Token Service (STS) trust chain through the STS interface. Some examples of existing customized policy enforcement points are WebSphere® Servlet Filter, Trust Association Interceptor (TAI), and a reverse proxy such as WebSEAL. As Security Access Manager supports OAuth 2.0 federations, you can develop customized policy enforcement points to work with OAuth 2.0 federations. The following diagram illustrates the relationship between the OAuth STS trust chain and other OAuth components.

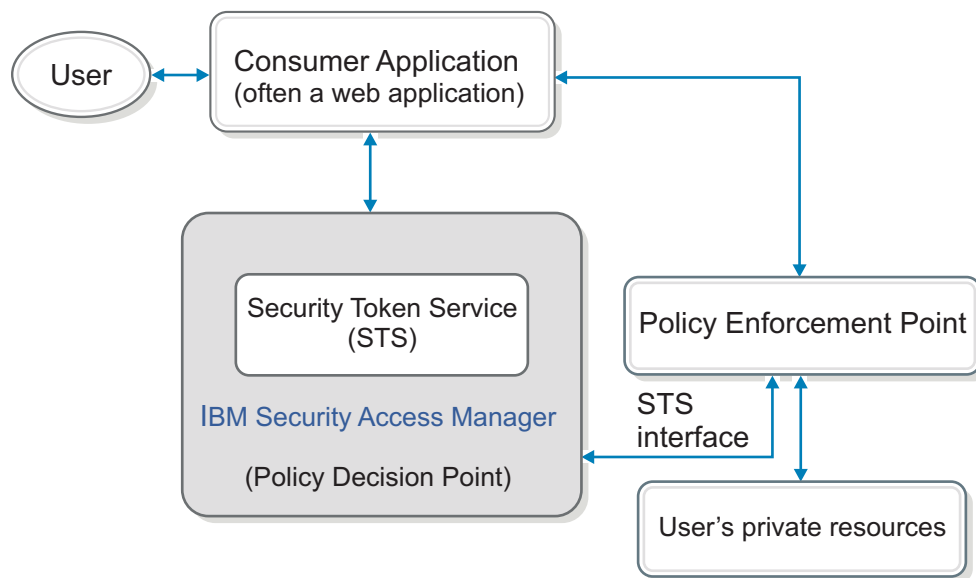


Figure 4. OAuth STS trust chain workflow

This section describes the process an OAuth enforcement point undertakes to transform an HTTP request for an OAuth protected resource into a WS-Trust message.

The transformation makes it possible for the STS to validate the request. It also describes the possible responses an enforcement point can receive from the STS and how to deal with them.

The following information about the policy decision point in Security Access Manager must be made available to the enforcement point:

- The absolute URL of the Security Access Manager STS trust service endpoint. (For example: <https://isam.com/TrustServer/SecurityTokenService>)
- The basic authentication user name and password for the Security Access Manager STS trust service (if required).
- The ProviderID of the Security Access Manager federation the client belongs to, which is used as the AppliesTo address for WS-Trust requests. Optionally, the enforcement point accepts a provider ID from the OAuth client as a request parameter to serve more than one federation concurrently.

Authorization decision request

Configuration

For OAuth 2.0 requests, the enforcement point must know the Security Access Manager OAuth 2.0 issuer address prefix (`urn:ibm:ITFIM:oauth20:token:`).

HTTP request

When an OAuth 2.0 client retrieves a protected resource with its access token, it constructs a request similar to any of the following examples. Each of these three examples is logically the same request. All that differs is the transmission mechanism (HTTP header, query string, post body) for sending the OAuth 2.0 bearer access token:

OAuth 2.0 Example 1 (Access token in authorization header)

```
POST /oauth/protectedresource.jsp
Host: isam.com
Authorization: Bearer YPxa78JggdW7hvcFRJph
Content-Type: application/x-www-form-urlencoded

username=steve
```

OAuth 2.0 Example 2 (Access token in post body)

```
POST /oauth/protectedresource.jsp
Host: isam.com
Content-Type: application/x-www-form-urlencoded

username=steve&access_token=YPxa78JggdW7hvcFRJph
```

OAuth 2.0 Example 3 (Access token in query string)

```
POST /oauth/protectedresource.jsp?access_token=YPxa78JggdW7hvcFRJph
Host: isam.com
Content-Type: application/x-www-form-urlencoded

username=steve
```

Authorization decision request

The OAuth 2.0 enforcement point is responsible for the following actions:

- Transform HTTP requests into a WS-Trust SOAP message.
- Send the WS-Trust SOAP message to the Security Access Manager STS for request validation.

The HTTP request is transformed into the following WS-Trust SOAP message:

OAuth 2.0 Token Validate Request (Request Security Token)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <wst:RequestSecurityToken xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
      <wst:RequestType xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        http://schemas.xmlsoap.org/ws/2005/02/trust/Validate
      </wst:RequestType>
      <wst:Issuer xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          urn:ibm:ITFIM:oauth20:token:bearer
        </wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          <wsa:Address>https://localhost/sps/oauth/oauth20/</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:Base xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
        <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
          <stsuuser:Principal/>
          <stsuuser:AttributeList/>
          <stsuuser:ContextAttributes>
            <stsuuser:Attribute name="access_token"
              type="urn:ibm:names:ITFIM:oauth:param">
              <stsuuser:Value>YPxa78JggdW7hvcFRJph</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="username"
              type="urn:ibm:names:ITFIM:oauth:body:param">
              <stsuuser:Value>steve</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="port"
              type="urn:ibm:names:ITFIM:oauth:request">
              <stsuuser:Value>9443</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="method"
              type="urn:ibm:names:ITFIM:oauth:request">
              <stsuuser:Value>POST</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="path"
              type="urn:ibm:names:ITFIM:oauth:request">
              <stsuuser:Value>/oauth/protectedresource.jsp</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="scheme"
              type="urn:ibm:names:ITFIM:oauth:request">
              <stsuuser:Value>https</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="host"
              type="urn:ibm:names:ITFIM:oauth:request">
              <stsuuser:Value>isam.com</stsuuser:Value>
            </stsuuser:Attribute>
          </stsuuser:ContextAttributes>
        </stsuuser:STSUniversalUser>
      </wst:Base>
    </wst:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following attributes are defined by the WS-Trust specification. They are used by Security Access Manager to identify the federation that is associated with this request and to identify the type of OAuth 2.0 access token being used.

- The Issuer address element (highlighted in **bold**) must be set to the Security Access Manager OAuth 2.0 issuer address prefix (urn:ibm:ITFIM:oauth20:token:). The token type must be appended at the end and separated by a colon. Currently, the only token supported type is *bearer*, which means the issuer address must be set to urn:ibm:ITFIM:oauth20:token:bearer.

- The AppliesTo address element (highlighted in *italics*) must match the Provider ID of the API Protection Definition within Security Access Manager. The general form is `https://localhost/sps/oauth/oauth20/{id}`.

The **access_token** attribute with type `urn:ibm:names:ITFIM:oauth:param` is mandatory in the WS-Trust message sent to Security Access Manager. It must be appended to the **ContextAttributes** section of the **STSUniversalUser** within the WS-Trust Request Security Token.

If **access_token** attribute is missing from the request from the OAuth 2.0 client, the enforcement point does not validate the request with Security Access Manager STS. It can instantly return an HTTP 400 Bad Request status code and optionally can include a description of the error in the body.

Note: If the access token is included in the authorization header in the Authorization: Bearer <token> format, the token must still be added to the **ContextAttributes** section of the STSUU. The same format must be used as if the access token was sent through a query string or post body.

The following attributes are not mandatory in the WS-Trust message that is sent to Security Access Manager STS for OAuth 2.0. However, they might be useful to a custom mapping rule that is executed by Security Access Manager.

It is recommended to append the following attributes to the **ContextAttributes** section of the **STSUniversalUser** within the WS-Trust Request Security Token and set the attribute type to `urn:ibm:names:ITFIM:oauth:request`.

- **method** - the HTTP method of the request (GET/POST)
- **scheme** - (http/https)
- **host** - host header from the request
- **port** - the port number on the host (only if it is a non-standard port. For example, not 80 if the method is HTTP or not 443 if the method is HTTPS)
- **path** - the requested path

Append any additional parameters that the OAuth 2.0 enforcement point finds in the request, such as query or post body parameters that are not of OAuth 2.0, to the **Context Attribute** section of the **STSUniversalUser** within the WS-Trust Request Security Token. The type value is determined by the following table.

In OAuth 2.0 requests, these parameters are not required. However, they might be useful to a custom mapping rule that is executed by Security Access Manager. So it is recommended that you append them.

HTTP Parameter Location	Attribute Type Value
URL Query String Parameters	<code>urn:ibm:names:ITFIM:oauth:query:param</code>
HTTP Request Body Parameters	<code>urn:ibm:names:ITFIM:oauth:body:param</code>

Post body parameters must be included only if the following conditions are met:

- The entity-body is single-part.
- The entity-body follows the encoding requirements of the “application/x-www-form-urlencoded” content-type as defined by [W3C.REC-html40-19980424].
- The HTTP request entity-header includes the “Content-Type” header field set to “application/x-www-form-urlencoded”.

Authorization decision response

The SOAP message response from Security Access Manager (regardless of OAuth version) echoes all the context attributes sent in the original request and some extra response context attributes.

OAuth Token Validate Response (RSTR)

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <wst:RequestSecurityTokenResponse wsu:
      Id="uuid56a54e7c-012f-1207-9133-c24cad886d75"
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wss-wssecurity-utility-1.0.xsd">
      <wsp:AppliesTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2004
        /08/addressing"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference>
          <wsa:Address>https://localhost/sps/oauth/oauth20/</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:RequestedSecurityToken>
        <stsuser:STSUniversalUser xmlns:stsuser="urn:ibm:names
          :ITFIM:1.0:stsuser">
          <stsuser:Principal/>
          <stsuser:AttributeList/>
          <stsuser:ContextAttributes>
            <stsuser:Attribute name="authorized"
              type="urn:ibm:names:ITFIM:oauth:response:decision">
              <stsuser:Value>TRUE</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="expires" type="urn:ibm
              :names:ITFIM:oauth:response:decision">
              <stsuser:Value>2011-04-22T00:52:18Z</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="scope" type="urn:ibm
              :names:ITFIM:oauth:response:attribute">
              <stsuser:Value>email</stsuser:Value>
              <stsuser:Value>first</stsuser:Value>
              <stsuser:Value>last</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="username" type="urn:ibm
              :names:ITFIM:oauth:response:attribute">
              <stsuser:Value>wasadmin</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="username_is_self"
              type="urn:ibm:names:ITFIM:oauth:response:attribute">
              <stsuser:Value>FALSE</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="oauth_token" type="urn:ibm
              :names:ITFIM:oauth:response:attribute">
              <stsuser:Value>YPxa78JggdW7hvcFRJph</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="recovered_state" type="urn:ibm
              :names:ITFIM:oauth:response:attribute">
              <stsuser:Value>State storage time was:
                2011-04-15T00:52:18Z</stsuser:Value>
            </stsuser:Attribute>
            <stsuser:Attribute name="state_id" type="urn:ibm
              :names:ITFIM:oauth:state">
              <stsuser:Value>2cJsZ3QhXV5rDVZHNePp</stsuser:Value>
            </stsuser:Attribute>
          </stsuser:ContextAttributes>
          <stsuser:AdditionalAttributeStatement id=""/>
        </stsuser:STSUniversalUser>
      </wst:RequestedSecurityToken>
      <wst:Status>
        <wst:Code>http://schemas.xmlsoap.org/ws/2005/02/trust/status
          /valid</wst:Code>
      </wst:Status>
    </wst:RequestSecurityTokenResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```

        </wst:Status>
    </wst:RequestSecurityTokenResponse>
</soapenv:Body>
</soapenv:Envelope>

```

The following context attributes returned to the enforcement point by Security Access Manager relate to the authorization decision. It also has the attribute type `urn:ibm:names:ITFIM:oauth:response:decision` highlighted in *italics* in the previous RSTR example. It is up to the enforcement point to decide whether to down-stream these attributes to the OAuth protected resource.

These attributes are primarily for the use of the enforcement point itself to determine the authorization status.

Context attributes	Description
authorized	The value is set to TRUE if the OAuth request is valid and authorized; FALSE if otherwise.
expires	The UTC time when the access token that is used in the request is no longer valid.

The following context attributes returned to the enforcement point by Security Access Manager must be down-streamed from the enforcement point to the OAuth protected resource. They might be appended to the original HTTP request in any way deemed suitable by the enforcement point and the protected resource. This way, the protected resource can retrieve them (for example, as additional HTTP headers).

These context attributes have the attribute type `urn:ibm:names:ITFIM:oauth:response:attribute` (highlighted in **bold** in the previous RSTR example).

Custom mapping rules that are executed after the OAuth trust chain might also append attributes with this type. Therefore, any attribute with this type must be down-streamed to the requested protected resource.

Context attributes	Description
access_token	The OAuth access token that is used in the protected resource request.
client_type	The type of client that this token was issued to, can be either public or confidential. Public clients are clients that do not have client credentials and therefore cannot authenticate to the authorization server.
oauth_token_client_id	The unique identifier of the client to which the current access token was issued.
scope	A list of strings that represents the resource scope that is authorized by the user at the OAuth resource owner authorization step. The OAuth protected resource can use this attribute to determine which resources to return in the response. This attribute is only present for OAuth flows that include a user authorization step.
username	The name of the user who authorized the OAuth token to access their protected resources on their behalf. With OAuth flows that do not involve a separate resource owner, this value is the client identifier.

Additional attributes with the type `urn:ibm:names:ITFIM:oauth:response:attribute` are sometimes appended by a custom mapping rule, such as the case with `recovered_state` and `username_is_self` in the example.

The `state_id` context attribute returned to the enforcement point by Security Access Manager is used by a custom mapping rule that is executed after the OAuth trust chain. It has the attribute type `urn:ibm:names:ITFIM:oauth:state` (highlighted with an underline) and can be ignored by the enforcement point.

The `state_id` attribute is a unique identifier for the current OAuth token that is used to store state information.

If the `state_id` attribute is required by the OAuth protected resource, a custom mapping rule can be implemented to make a copy of this attribute. The type can be changed to `urn:ibm:names:ITFIM:oauth:response:attribute` from the custom mapping rule to ensure that it is down-streamed to the resource.

Error responses

You can customize the amount of OAuth request validation that the enforcement point performs. Any validation it performs is repeated by Security Access Manager. Doing some validation before sending an authorization request to Security Access Manager might improve performance. The following validation must be performed by the enforcement point before sending a request to Security Access Manager.

- Validate that some OAuth data is present. If not, return an HTTP 401 Unauthorized status code.
- Validate that none of the required OAuth parameters are missing. If any of them are not present in the request, return an HTTP 400 Bad Request status code.
- Validate that none of the required OAuth parameters occur more than once in the request. They must also occur only in the one component of the request; for example, the query string or the authorization header. If the validation fails, return an HTTP 400 Bad Request status code.

The enforcement point must return an HTTP 401 Unauthorized status code to the OAuth client if the following scenario occurs:

- The enforcement point receives a SOAP message with an authorized context attribute that has a value of FALSE.

The enforcement point must return an HTTP 503 Service Unavailable status code to the OAuth client if the following scenarios occur:

- Security Access Manager encounters an error.
- Security Access Manager does not return a constructed SOAP message or the SOAP message does not contain an authorized context attribute.

The enforcement point might also optionally return a WWW-Authenticate HTTP header to indicate its support for OAuth.

Flow chart

The following chart shows the expected workflow of an OAuth authorization enforcement point.

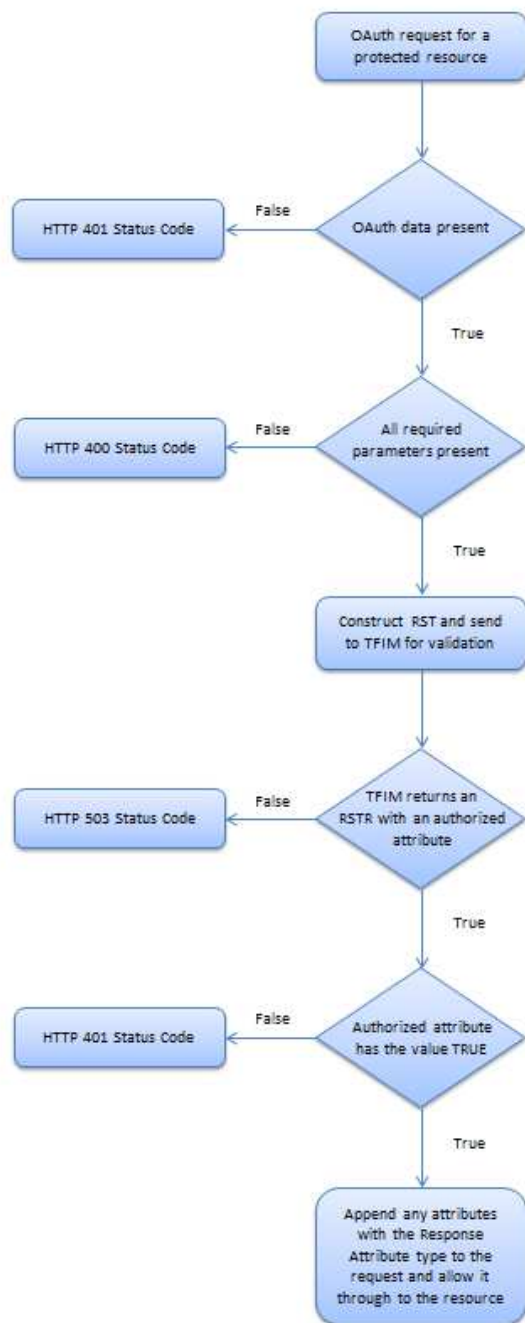


Figure 5. OAuth authorization enforcement point workflow

API Protection form post response mode

With the form post response mode, a client can make an OAuth authorization request and receive a self-posting form rather than a 302 response.

For more information about the form post response mode, see https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html.

The form post template page contains a form that is populated with the action URI as the redirect URI presented on the authorization request. There is also a repeating macro inside the form, containing name and value macros.

Macros:

@ACTION@

The validated redirect URI presented in the authorization request.

Repeating macros:

The following macros must be used inside the repeating macro block 'oauth_form_post'. For example:

```
[RPT oauth_form_post]
<input type="hidden" name="@OAUTH_HIDDEN_NAME@" value="@OAUTH_HIDDEN_VALUE@" />
[ERPT oauth_form_post]
```

@OAUTH_HIDDEN_NAME@

Parameter name of the form post body. Default values include (depending on response_type) :

scope, state, expires_in, access_token, token_type, code

@OAUTH_HIDDEN_VALUE@

Parameter value in the form post body. This value corresponds to the @OAUTH_HIDDEN_NAME@ macro value.

Access policy for OAuth or OIDC

Access policy is used to enable advanced authentication scenarios when performing an OAuth flow.

For details on access policy, see Access policies.

When using access policy for OAuth or OIDC, the consent decision can be made by the access policy.

Making an OAuth or OIDC consent decision using access policy

You can use an access policy to prompt the user to enter further information via a web page or redirect the user to another website. This logic could be used to perform the consent step when advanced logic beyond "prompt once", "always prompt", or "never prompt" is required.

This advanced logic is undefined. But it is assumed that as a result of it, the author of the policy will be able to decide whether the user has consented, and if they have consented, which scopes the user has granted the client.

The following snippet can be used to set the list of scopes consented:

```
// Get the protocol Context:
var pctx = context.getProtocolContext();
// Construct our array of scopes
var scopes = java.lang.reflect.Array.newInstance(java.lang.String,2);
// Set the values
scopes[0] = "scope1";
scopes[1] = "scope2";
// Add this to the context
pctx.setConsentDecision(scopes);
```

If consent has been performed but no scope was granted, then the follow snippet can be used:

```
// Get the protocol Context:
var pctx = context.getProtocolContext();
var scopes = java.lang.reflect.Array.newInstance(java.lang.String,1);
scopes[0] = "";
// Add this to the context
pctx.setConsentDecision(scopes);
```

OIDC Dynamic Clients

OpenID Connect (OIDC) publishes a specification that allows registration of a client to an OpenID Connect Provider.

This enables someone to onboard their application to an OpenID Connect provider through a standard well-formed API. See the specification https://openid.net/specs/openid-connect-registration-1_0.html.

The primary information that an application administrator is required to provide is the redirect URI that the application uses when requesting an identity.

To use dynamic client registration, you must be using an OIDC-enabled definition and have the option **Enable Client Registration** set to true. See “Creating an API protection definition” on page 131.

OIDC Dynamic Clients- Authentication and deployment

There are considerations to take when you are deploying a definition that allows the registration of clients through a public API.

Consider the following factors:

Do you require authentication to register a client?

If you do not require authentication for when a client is registered, there is no way of identifying who owns a client application.

CAUTION:

This is a higher risk deployment pattern than if you require authentication.

You can control the access to the registration endpoint with an Access Control List (ACL). The reverse proxy OAuth configuration API then configures the ACLs with the **Require authentication to register a client** option. Group based requirements might also be added to ensure that only administrators or trusted users can register clients. The authenticated users credential information is available in the STSUniversalUser attribute list during the registration. This information can be associated with the registered client for use during the consent step, informing the end user who the application administrator is.

What consent challenges are sent to the resource owner?

Since dynamic clients are inherently less trusted than an administrator-registered client, the emphasis on the users consent in the delegated authorization is increased. When and how consent might be performed depends on who is able to register a client. Dynamic client parameters such as **client_uri** are available on the consent page as a macro. you can use this parameter and other values (For example, **tos_uri**, **log_uri**, **policy_uri**) to allow the user to identify, discover, and verify who a client is before granting them access.

Will you issue a client secret?

A client secret allows access to the client centric API endpoints such as **/token** and **/introspect**, as well as allows the client to perform HMAC signing of JWTs. Without a client secret, an authorization code flow cannot be performed. When the configuration property **issue client secret** is enabled, a client secret is issued when the registration is made by an authenticated party.

OIDC Dynamic Clients- Register a client

To register a client, issue a HTTP POST to the Client Registration Endpoint.

See “OAuth 2.0 endpoints” on page 108.

Any values which are posted in the JSON body are stored such that both standard values and custom values can be kept. These values are available in mapping rules and in a macro on the consent page.

The following example is an example request to register a client:

```
POST_DATA='{ "redirect_uris": [ "https://app.com"],
  "tos_uri": "https://app.com/tos",
  "company_name": "Applications Inc" }'
```

```
curl https://myisam.com/mga/sps/oauth/oauth20/register/mydefinition-d "$POST_DATA" -H "Accept: application/json" -H "Authorization: Bearer myAccessToken" -H "Content-type: application/json"
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "client_secret_expires_at": 0,
  "owner_username": "testuser",
  "company_name": "Applications Inc",
  "registration_client_uri": "https://myisam.com/mga/sps/oauth/oauth20/register/testDef?client_id=myClient",
  "client_secret": "mySecret",
  "tos_uri": "https://app.com/tos",
  "client_id_issued_at": 1522139359,
  "redirect_uris": "https://app.com",
  "registration_access_token": "myClientAccessToken",
  "client_id": "myClientId"
}
```

OIDC Dynamic Clients- Retrieve a dynamic client

When a client is registered, a `registration_client_uri` is returned in the payload.

This endpoint can be used to perform the following actions:

- Retrieve the registered clients definition
- Delete the client

Note: You must authenticate as either the client, or the user who was authenticated at the time of client registration in order to view or delete the client.

To view the client, issue an HTTP GET request and include the **client_id** parameter. For example:

```
$ curl https://myisam.com/mga/sps/oauth/oauth20/register/mydefinition?client_id=myClient -H
Accept:application/json -H "Authorization: Bearer myClientAccessToken"
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "client_secret_expires_at": 0,
  "owner_username": "testuser",
  "company_name": "Application Inc",
  "registration_client_uri": "https://myisam.com/mga/sps/oauth/oauth20/
register/myDefinition?client_id=myClient",
}
```

```
"client_secret": "hunter2",
"tos_uri": "https://app.com/tos",
"client_id_issued_at": 1522137286,
"redirect_uris": "https://app.com",
"client_id": "myClient"}
```

OIDC Dynamic Clients- Custom Identifiers

You can customize the value of the `client_id`, `client_secret` and `registration_access_token` issued to the application.

There is an example of this in the out-of-the-box PreToken mapping rule. Look for the variable `custom_client_id_secret`.

If a custom *client_secret* is set then it is issued, regardless of whether it is enabled in the definition configuration and whether or not the request is made by an authenticated party

To customize the *registration_access_token*, see “Customizing OAuth tokens by updating the sample PreTokenGeneration mapping rule” on page 163.

API Protection clients now have a dynamic data field when they are configured. This allows storage of arbitrary data against the client which can be accessed at runtime (For example, from the consent page and in mapping rules).

OIDC Dynamic Clients- Update a client

To update a client, issue a HTTP PUT to the clients management endpoint.

About this task

When you are updating a dynamic client, the client must authenticate as the OAuth client or the owner of the OAuth client. The following attributes in the payload are ignored when you are updating a dynamic client:

- client_id
- client_id_issued_at
- registration_client_uri
- owner_username
- registration access token

Any user or administrator provided values in the client metadata that are not presented in the update request is removed from the client metadata.

If the client has a secret, the `client_secret` **must** be presented and match the current secret.

When an update occurs, a new `client_secret` and `registration_access_token` are issued to the client.

Example of updating a dynamic client:

```
$ curl https://www.myidp.ibm.com/mga/sps/oauth/oauth20/register/testDef?
client_id=VWM3W8zxlRrgsnmFGd -H "Accept:application/json" -H "content-type: application/json" -d
'{
  "registration_client_uri": "https://www.myidp.ibm.com/mga/sps/oauth/oauth20/register/testDef?client
  "client_id": "VWM3W8zxlRrgsnmFGd",
  "client_secret": "as9r83nfo312o",
  "client_id_issued_at": 1537328443,
  "client_name": "A dynamic client",

```

```

    "grant_types": [
      "authorization_code"
    ],
    "redirect_uris": [
      "https://myapp.com"
    ],
    "new_property": "new_value",
    "company_name": "ORG"
  }' -X PUT -H "Authorization: Bearer registrationAccessToken"

```

HTTP/1.1 200 OK
Content-Type: application/json

```

{
  "registration_client_uri": "https://www.myidp.ibm.com/mga/sps/oauth/oauth20/register/testDef?
client_id=VWM3W8zxlagRrgsnmFGd",
  "registration_access_token": "newRegistrationAccessToken",
  "client_id": "VWM3W8zxlagRrgsnmFGd",
  "client_id_issued_at": 1537328443,
  "client_name": "A dynamic client",
  "client_secret": "newClientSecret",
  "grant_types": [
    "authorization_code"
  ],
  "redirect_uris": [
    "https://myapp.com"
  ],
  "new_property": "new_value",
  "company_name": "ORG"
}

```

OIDC Dynamic Clients- Delete a client

To delete a dynamic client, issue a HTTP DELETE to the client management endpoint.

About this task

OAuth dynamic clients can only be deleted by themselves or their owner.

For example,

```

$ curl -k myisam.com/mga/sps/oauth/oauth20/register/mydefinition?
client_id=myClient -H Accept:application/json -H 'Authorization:
Bearer registrationAccessTokenValue' -X DELETE
HTTP/1.1 204 OK

```

The default response is a 204. Response attributes can be added in the post-mapping rule. If they are added the status code is 200 and the JSON payload.

Dynamic clients can also be managed from the LMI. See the web services documentation on how to do so. See REST API documentation.

Chapter 11. Mobile Multi-Factor Authentication

The IBM Security Access Manager Advanced Access Control component supports authenticator applications. Such support is built around the OAuth 2.0 protocol.

Authenticator applications are mobile-based applications that enable users to authenticate with minimal reliance on passwords. Mobile devices and biometric characteristics are used to support authentication and reduce the threat of unauthorized access to sensitive resources.

The IBM Verify application, which is available for download in major mobile application stores, is natively supported by the Advanced Access Control component. The authenticator application is built on the Mobile Access SDK, which is available for download from Fix Central. The Mobile Access SDK can also be used to create custom applications.

For instructions on configuring and using the IBM Verify application, see the IBM Verify User Guide.

The authenticators framework can be integrated with context-based access and authentication policies. Security Access Manager provides several pre-defined authentication policies to enable combinations of mobile and biometric mechanisms.

The authenticator application registration is built around an OAuth grant that is issued to the mobile device. The grant is used to identify the authenticator in future requests.

Authenticator registration

The IBM Verify application uses the OAuth authorization grant flow to perform registration, which is launched by the user in a browser.

A button to initiate the registration flow is available in the `device_selection.html` page, which is an example page that demonstrates how the registration might be initiated. This page is available from the appliance **File Downloads** area at the path `access_control/pages/C/mga/user/mgmt/device/device_selection.html`.

When the button is clicked, it calls the OAuth authorize endpoint to obtain an authorization code. The user is then presented with a QR code that can be scanned by the IBM Verify application to complete registration. The following steps illustrate a typical authenticator registration flow.

1. The user downloads and installs the IBM Verify application.
2. The user logs in to the Security Access Manager User Self Care (USC) with a desktop browser and clicks a button on the page that is presented by USC to initiate the registration flow.
3. The browser starts the OAuth authorization code flow.
4. Security Access Manager responds with a QR code.
5. The user scans the QR code with the IBM Verify application.
6. The IBM Verify application completes the registration automatically.

The Security Access Manager SDK supports registration without the need for browser initiation in custom applications and also supports the OAuth ROPC flow.

Authentication registration is completed when an OAuth flow is completed with the scope set to “**mmfaAuthn**”. Other attributes that can be included and saved via the OAuth mapping rule are:

- Push token ID
- Application ID
- Device Name
- Device Type
- OS Version
- Fingerprint support included
- Front camera support included
- Tenant ID

Authentication method enrollment

After an authenticator is registered, the user is prompted to enroll authentication methods.

Supported authentication methods include fingerprint and simple user presence.

Enrollment is performed through the System for Cross-Domain Identity Management (SCIM) API. For more information, see SCIM configuration.

Configuring Mobile Multi-Factor Authentication

Follow these steps to configure Mobile Multi-Factor Authentication.

Before you begin

The following pre-requisites must be met:

- The IBM Security Access Manager Platform and Advanced Access Control Module are activated.
- The runtime component and a reverse proxy instance are configured.
- Basic User support is enabled on the local LDAP.
- Transparent path junction to **/scim** on localhost is configured.
 - BA with easuser enabled
 - **isam_mobile_rest** ACL attached to **/scim** (ACL won't exist until step 2)
- Username Password Mechanism is configured.
- Server connection to local LDAP is set up.
- SCIM is configured with local LDAP server connection **dc=iswga** suffix.

Procedure

1. Create an API Protection definition and client with:
 - Authorization code and ROPC enabled
 - Redirect URI: `https://<webseal_hostname>:<port>/mga/sps/mmfa/user/mgmt/html/mmfa/qr_code.html?client_id=<client_ID>`

Note:

The redirect URI is essential so that when a user clicks the **Register Authenticator** button in the USC UI, the user is correctly redirected to the QR Code page.

2. Run the Reverse Proxy MMFA Config API.

This step configures the /mga junction and creates the required ACLs.

```
curl -ki -H 'Accept: application/json' -H
'Content-type:application/json' --user 'admin:XXXX' -X POST https://
192.168.124.130/wga/reverseproxy/default/mmfa_config -d
'{"lmi":{"hostname":"192.168.124.130", "port":443, "username":"admin",
"password":"XXXX"}, "runtime":{"hostname":"localhost", "port":443,
"username":"easuser", "password":"XXXX"}, "reuse_certs":false,
"reuse_acls":false, "reuse_pops":false}'
```

3. Run the AAC MMFA Config API.

This step configures the reverse proxy details into a location where the AAC code can access it.

```
curl -ki -H 'Accept: application/json' -H
'Content-type: application/json' --user 'admin:XXXX' -X POST
https://192.168.124.130/iam/access/v8/mmfa-config -d
'{"client_id":"AuthenticatorClient",
"hostname":"192.168.124.140",
"port":443, "junction":"/mga"}'
```

Configuring a Mobile Multi-Factor Authentication (MMFA) Authenticator Mechanism

This authentication mechanism can be used to initiate and correlate results for a range of mobile multi-factor authentication techniques.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Mechanisms**.
5. Click **MMFA Authenticator**.
6. Click  .
7. Click the **Properties** tab.
 - a. Select the property that you want to configure.
 - b. Click  .
 - c. Enter the value for that property.
 - d. Click **OK**.
8. Take note of the following property for the mechanism:

Signing Attributes

This property defines a comma separated list of context attributes that is added to a new JSON value attribute that is passed as a new pending attribute to the target mobile device. If supported by the device, this JSON value is used to extract the various messages that is displayed to the end user. The MMFA server also uses this JSON value during signature validation.

Note: The value that is set here can be overridden when the **signAttributeList** property is set in a specific MMFA authentication policy. See Authentication Policy Parameters and Credentials

9. Click **Save**.

What to do next

When you configure the mechanism, a message indicates that the changes are not deployed. Deploy the changes when you are finished. See Deploying Pending Changes.

MMFA mapping rule methods

Customize the OAuth **PreTokenGeneration** and **PostTokenGeneration** mapping rules by using these methods.

Sample mapping rules are available from **Manage System Settings > Secure Settings > File Downloads** under the **access_control > examples > mapping rules** directory.

The following limitations affect the attribute keys and values that are associated with the **state_id** by using the **MMFAMappingExtUtils** class:

- Keys cannot be null or empty.
- Values can only be null or empty when specified.
- Associated key-value pairs are read-only and not case sensitive.
- The push token is read-only and case sensitive.

registerAuthenticator

```
public static String registerAuthenticator(  
    String stateId  
)
```

This method performs the final steps of registering an authenticator. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

These responses come from the runtime after registration.

- The new authenticator's ID if successful.
- Null if not successful.

savePushToken

```
public static boolean savePushToken(  
    String stateId,  
    String pushToken,  
    String applicationID  
)
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

pushToken

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

applicationID

The application ID of the authenticator application. This parameter can be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

savePushToken

```
public static boolean savePushToken(  
    String stateId,  
    String pushToken  
)
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

pushToken

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

saveDeviceAttributes

```
public static boolean saveDeviceAttributes(  
    String stateId,  
    String deviceName,  
    String deviceType,  
    String osVersion,  
    String fingerprintSupport,  
    String frontCameraSupport,  
    String tenantId  
)
```

This method saves various device attributes with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

deviceName

The name of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

deviceType

The type of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

osVersion

The OS version of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

fingerprintSupport

The type of fingerprint sensor that is supported by the device. This parameter can be null or empty. If empty, the value is cleared.

frontCameraSupport

flag that indicates if the device has a front facing camera. This parameter can be null or empty. If empty, the value is cleared.

tenantId

The tenant ID for this registration, if the authenticator application is multi-tenant. This parameter can be null or empty. If empty, the value is cleared.

These responses come from the runtime.

- True if successful.
- False if not successful.

Chapter 12. FIDO and WebAuthn Support

The IBM Security Access Manager Advanced Access Control component provides support for FIDO and WebAuthn.

FIDO2 Server Endpoints

Endpoints provide FIDO2 clients the ability to communicate with the FIDO2 server.

Table 20. FIDO2 Server Endpoints

Endpoint Name	Description	Example
Credential Creation Options	The attestation options request is the first step of registration. The options returned by the server are intended to be used with WebAuthn's <code>navigator.credentials.create()</code> .	<code>https://server.com/mga/sps/fido2/<relying_party>/attestation/options</code>
Authenticator Attestation Response	The attestation result request is the second step of registration. The result of the <code>navigator.credentials.create()</code> request is sent to the server, which validates the challenges, origins, signatures, and the rest of the request. If validation passes, the registration is saved.	<code>https://server.com/mga/sps/fido2/<relying_party>/attestation/result</code>
Credential Get Options	The assertion options request is the first step of authentication. The options returned by the server are intended to be used with WebAuthn's <code>navigator.credentials.get()</code> .	<code>https://server.com/mga/sps/fido2/<relying_party>/assertion/options</code>
Authenticator Assertion Response	The assertion result request is the second step of authentication. The result of the <code>navigator.credentials.get()</code> request is sent to the server, which validates the assertion. If validation passes, the authentication was successful	<code>https://server.com/mga/sps/fido2/<relying_party>/assertion/result</code>

Concepts

The FIDO alliance is an open industry association that produces authentication standards and certifications for password-less scenarios.

There are currently three sets of specifications that have been published:

- FIDO Universal Second Factor (FIDO U2F). See “Configuring a FIDO Universal 2nd Factor authentication mechanism” on page 71.
- FIDO Universal Authentication Framework (FIDO UAF)
- FIDO2

The FIDO2 specification includes the W3C's Web Authentication (WebAuthn) specification and FIDO Client to Authenticator Protocol (CTAP). The WebAuthn specification defines the interactions between a Client and a Relying Party, whereas the CTAP protocol is between an Authenticator and a Client.

The Relying Party is the service that is requesting authentication.

A Client is a web browser or operating system.

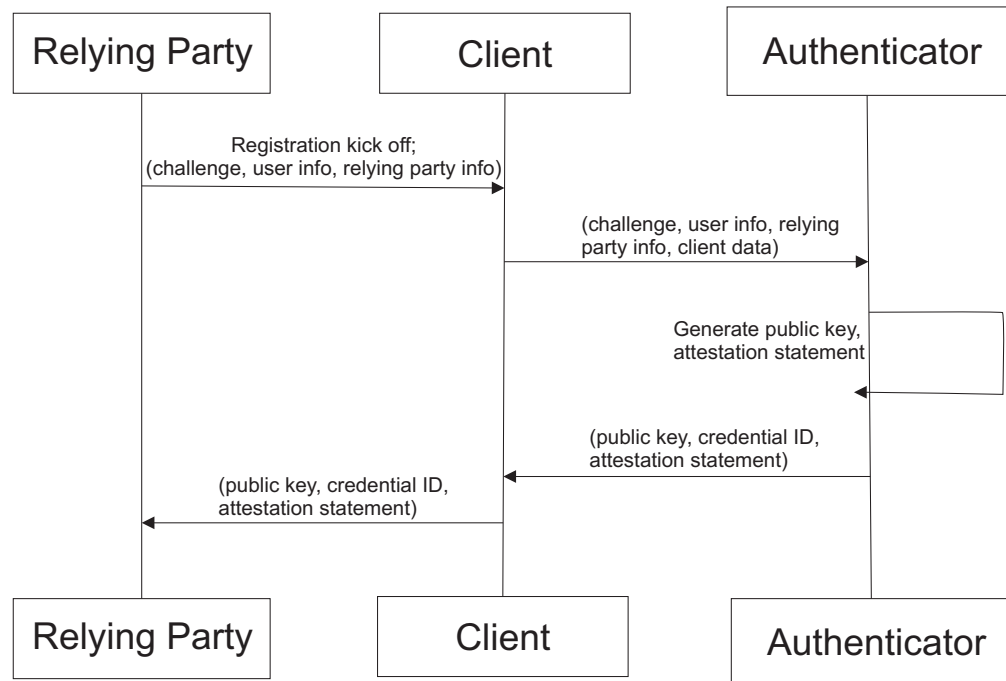
An Authenticator can be a device or program that performs cryptographic operations to provide verification for a user.

WebAuthn Ceremonies

The WebAuthn spec defines two ceremonies that are performed between the user, authenticator, client and relying party: Registration and Authentication

Registration Ceremony

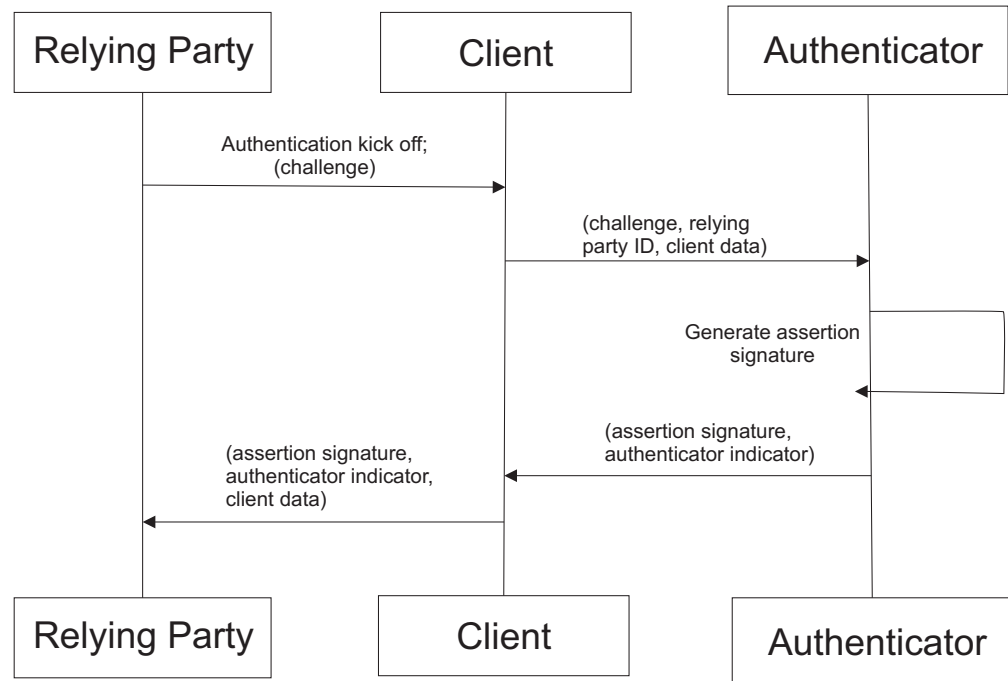
The registration ceremony consists of up of five steps:



1. When registration is requested, the Relying Party provides a challenge and other information to the Client.
2. The Client sends the information from step 1 along with extra client data to the Authenticator.
3. The Authenticator generates a public key, optionally performs user presence or verification, and produces an attestation statement based on the given challenge.
4. The public key and attestation statement sent back to the Client.
5. The Client sends the information to the Relying Party, which validates the attestation statement against the original challenge, and if successful, saves the public key to the user's account.

Authentication Ceremony

The authentication ceremony consists of five steps:



1. When authentication is requested, the Relying Party provides a challenge to the Client.
2. The Client sends the information from step 1 along with extra client data to the Authenticator.
3. The Authenticator generates an assertion signature using the stored private key and optionally performs user presence or verification.
4. The assertion signature and extra authenticator data is sent back to the Client.
5. The Client sends the information to the Relying Party, which validates the assertion signature against the original challenge, and if successful, marks the user as authenticated.

Attestation

Attestation is a step within the registration flow that allows a relying party to establish whether an authenticator is authentic and can be trusted.

This is achieved through the attestation statement sent by the authenticator back through to the relying party. The attestation statement contains a signature which is the combination of the credential public key and the provided challenge, and optionally a certificate which contains the attestation public key.

There are several types of attestations which indicate how the signature was generated:

Basic Attestation

The attestation key pair is specific to the authenticator model, and authenticators of the same model may share the same key pair.

Self Attestation

The authenticator may not have an attestation key pair, so the credential private key is used.

Attestation CA

The authenticator can generate multiple attestation key pairs from an Attestation CA, a trusted third party.

Elliptic Curve based Direct Anonymous Attestation (ECDAA)

The authenticator receives direct anonymous attestation (DAA) credentials from a single DAA-Issuer. The DAA credentials are used with blinding to sign the attested credential data.

No attestation statement (None)

No attestation information is made available.

During registration the relying party can indicate a preference regarding how the attestation statement is generated.

There are attestation statement formats which indicate the syntax of the statement:

Packed Attestation Statement Format

Attestation Type supported: Basic, Self, AttCA

This is a WebAuthn optimized attestation format. Packed attestation statement format uses a very compact but still extensible encoding method.

TPM Attestation Statement Format

Attestation Types supported: AttCA

This attestation statement format is generally used by authenticators that use a Trusted Platform Module as their cryptographic engine.

Android Key Attestation Statement Format

Attestation Types Supported: Basic

This attestation statement format is for when the authenticator is provided by the Android platform, version “N” and later. In this case the attestation statement is produced by a component in a secure operating environment, but the authenticator data for the attestation is produced outside this environment.

Android SafetyNet Attestation Statement Format

Attestation Types Supported: Basic

This attestation statement format is for when the authenticator is provided by certain Android platforms and is based on the SafetyNet API. With this statement format the authenticator data is completely controlled by the Android application which invokes the SafetyNet API.

FIDO U2F Attestation Statement Format

Attestation Types Supported: Basic, AttCA

This attestation statement format is user with FIDO U2F authenticators using formats defined in the FIDO U2F specification.

None Attestation Statement Format

Attestation Types Supported: None

The none attestation statement format is used to replace any authenticator-provided attestation statement when a Relying Party indicates it does not wish to receive attestation information.

Public Key Algorithms

The list of preferred algorithms is provided to an authenticator when you are creating a credential, and an authenticator makes a best-effort to create the most preferred credential that it can.

There are several cryptographic algorithms for which a Relying Party might specify a preference.

The following algorithms are currently supported in IBM Security Access Manager:

Description	COSE Identifier
SHA256 with ECDSA	-7
SHA384 with ECDSA	-35
SHA512 with ECDSA	-36
SHA1 with RSA	-65535
SHA256 with RSA	-257
SHA384 with RSA	-258
SHA512 with RSA	-259

Metadata

Vendor Metadata files contain characteristics of authenticators as a Metadata Statement by using a specific syntax.

The following formats and their corresponding syntax are supported:

FIDO Metadata Statement

This metadata statement is JSON and follows the format as outlined by the FIDO Alliance.

Yubico Metadata

Yubico have established their own Metadata format (file extension .yubico).

PEM Certificate

PEM certificates (file extension .pem) do not provide any of the user experience advantages that other Metadata statements do. However they can be used to verify an attestation statement.

Upon registration, a device provides the server with its attestation certificate during the attestation ceremony. This certificate can be (optionally with Metadata Enforcement) used to verify the authenticity of the device against a Metadata file.

Registration

There are several user self care REST services available for managing FIDO2/WebAuthn registered authenticators.

Note: The user must authenticate to use the REST services capability.

REST services usage scenarios

Note: None of the REST services below require a JSON payload in the request.

Method	URL	Response	Response type
GET	https://hostname/mga/sps/fido2/authenticators	<pre>[{ "credentialId":credentialId, "rpId":rpId, "username":username, "version": 1 or 2, "present":true or false, "verified":true or false, "usageCount":usageCount, "nickname":nickname, "lastUsed":dateLastUsed, "created":dateCreated, "enabled": true or false, "metadata":{ "description": descriptionFromMetadata, "icon": icon } }]</pre> If the request completes successfully, the HTTP response code is 200. If the request does not complete successfully, the HTTP response is 500.	application/json
GET	https://hostname/mga/sps/fido2/authenticators/{credentialId}	<pre>{ "credentialId":credentialId, "rpId":rpId, "username":username, "version":1 or 2, "present":true or false, "verified":true or false, "usageCount":usageCount, "nickname":nickname, "lastUsed":dateLastUsed, "created":dateCreated, "enabled": true or false, "metadata":{ "description": descriptionFromMetadata, "icon": icon } }</pre> If the request completes successfully, the HTTP response code is 200. If the request does not complete successfully, the HTTP response is 500.	application/json
DELETE	https://hostname/mga/sps/fido2/authenticators/{credentialId}	If the request completes successfully, the HTTP response code is 204. If the request does not complete successfully, the HTTP response is 500.	application/json

U2F Migration

The WebAuthn specification includes backwards compatibility support for FIDO U2F.

To allow previously registered U2F tokens to authenticate with the FIDO2/WebAuthn mechanisms and delegates, the U2F registration data must be migrated.

Note: Not all WebAuthn scenarios are supported as FIDO U2F authenticators are unable to store a user handle.

IBM Security Access Manager offers two ways to migrate data by using the U2F Migration section of the FIDO2 Configuration screen. Select **Secure Access Control > Manage > FIDO2 Configuration > U2F Migration**.

Manual migration in batches

In this section, the number of unmigrated U2F registrations is displayed. An IBM Security Access Manager administrator can choose the batch size, and whether to migrate a single batch or all batches that are available.

Auto-migration on use

U2F registrations will be migrated when WebAuthn authentication is attempted.

When auto-migration is enabled and a WebAuthn authentication flow is attempted, the server checks if a user has any WebAuthn registrations. If a user does not have WebAuthn registrations, the server checks if a user has any U2F registrations, and migrates any that it finds.

The server then resumes the authentication flow.

FIDO2 Configuration

This topic provides the description of the parameters each option sets.

Name The name of the defined Relying Party (RP).

Relying Party ID

The hostname associated with the defined Relying Party. Needs to be a valid domain string signifying on whose behalf a given registration or authentication ceremony is being performed.

Create a FIDO2 Relying Party

When you are creating a FIDO2 Relying Party there are two distinct configuration modes, each resulting in a FIDO2 Relying Party being configured and functional.

FIDO2 Relying Party Configuration is accessed through **Secure Access Control > Manage > FIDO2 Configuration**.

Simple Configuration

Creating a FIDO2 Relying Party using the Simple Configuration will result in a relying party with most out of the box configuration defaults selected

Values need to be specified for the following parameters

Name this is the display name for the FIDO2 Relying Party.

Relying Party ID

The Relying Party ID is a valid domain string that identifies the WebAuthn Relying Party. When an authenticator is registered to a Relying Party, that registration is only valid for authenticating to that Relying Party. An example Relying Party ID is “example.com”.

The following Relying Party ID examples are invalid:

- https://example.com- protocol is not included in the Relying Party ID
- example.com/example_path- path is not included in the Relying Party ID

:

Advanced Configuration

Creating a FIDO2 Relying Party by using the Advanced Configuration results in a fully customized relying party using the options specified.

Values need to be specified for the following parameters:

Name this is the display name for the FIDO2 Relying Party.

Relying Party ID

The Relying Party ID is a valid domain string that identifies the WebAuthn Relying Party. When an authenticator is registered to a Relying Party, that registration is only valid for authenticating to that Relying Party. An example Relying Party ID is “example.com”.

The following Relying Party ID examples are invalid:

- https://example.com- protocol is not included in the Relying Party ID
- example.com/example_path- path is not included in the Relying Party ID

Administrative Group

Requests made to the FIDO2 server attestation and assertion endpoints usually contain username information in the message payload.

IBM Security Access Manager enforces that this message payload user information matches the authenticated session user (e.g. currently logged in user via ISAM session cookie) **UNLESS** the currently authenticated ISAM user is a member of this administrative group. Membership of this administrative group is intended for application service identities acting on behalf of users they have authenticated locally.

Attestation Types

See “Attestation” on page 195.

Attestation Statement Format

See “Attestation” on page 195.

Public Key Algorithms

See “Public Key Algorithms” on page 197.

Android SafetyNet Options

When Android SafetyNet is selected in **Attestation Statement Formats**, there are several options to specify values for.

Attestation Max Age

The maximum age in milliseconds of an attestation that is using the Android SafetyNet Statement Format.

Clock Skew

The amount of allowed variance in milliseconds when validating an attestation statement on the appliance.

Metadata

See “Attestation” on page 195.

The following formats are valid FIDO2 Metadata file formats:

- FIDO MDS document (.json file extension)
- Yubico Metadata (.yubico file extension)
- PEM Certificate (.pem file extension)

More than one Metadata file can be selected for this FIDO2 Relying Party.

Metadata Enforcement

When Metadata Enforcement is enabled for a FIDO2 Relying Party, the authenticator metadata is validated against the set of Metadata files enabled for this Relying Party, and the registration fails if this validation fails.

When Metadata Enforcement is disabled for a FIDO2 Relying Party, the authenticator metadata is still validated, however if this validation fails, the registration could still be allowed to succeed.

Mediator Mapping Rules

See “FIDO2 Mediation” on page 202.

Leaving the selection as **None** will disable FIDO2 Mediation from occurring on this FIDO2 Relying Party.

Authentication Service Mechanism

The WebAuthn authentication ceremony (“WebAuthn Ceremonies” on page 194) is integrated to the Authentication Service by using an authentication mechanism.

This provides out of the box template pages for the authentication flow, and the ability to be integrated into Context Based Access Policies.

Both step up authentication and username-less authentication are supported by WebAuthn. However username-less is highly dependent on the authenticator and browser being used.

Prerequisites

Before the FIDO2/WebAuthn Authentication Mechanism can be configured, a relying party must be defined. See “FIDO2 Configuration” on page 199.

WebAuthn includes support for authentication with existing U2F registrations. Before U2F registrations can be used with the FIDO2/WebAuthn authentication mechanism, the registration data must be migrated. See “U2F Migration” on page 199.

Request Flow

When the FIDO2/WebAuthn mechanism is triggered, IBM Security Access Manager returns a pending page to the user. This allows IBM Security Access Manager to provide the options required to trigger the browser to prompt the user for their authenticator. Once the user completes the required user presence and optional user verification steps, the browser sends the assertion result back to IBM Security Access Manager with the following request:

```

POST /mga/sps/authsvc
{
  "StateId": "...",
  "operation": "verify",
  "id": "LFdoCFJTyB82ZzSJUHc-c72yraRc_1mPvGX8ToE8su39xX26Jcqd31LUkKOS36FIAWgWl6itMKqmDvruha6ywA",
  "rawId": "LFdoCFJTyB82ZzSJUHc-c72yraRc_1mPvGX8ToE8su39xX26Jcqd31LUkKOS36FIAWgWl6itMKqmDvruha6ywA",
  "authenticatorData": "SZYN5Yg0jGh0NBcPZHgW4_krrmihjLHmVzzuoMd12MBAAAAA",
  "signature": "MEYCIQCv7EqsBRtf2E4o_BjzZfBwNpP8fLjd5y6TU0LWt519DQIhANiYig9newAJZYTzG1i51wP-YQk9uXF",
  "userHandle": "",
  "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJ4ZGowQ0JmWDY5MnFzQVRweTBrTmM4NTMzMmR2ZExVcHFZUDh3RFRYX1pFIiwiaW50IjoiImhhc2hBbGdvcml0aG0iOiJTSEEtMjU2Iiwib3JpZ21uIjoiaHR0cDovL2xvY2FsaG9zdDozMDAwIiwidHlwZSI6IndlYXN0IiwiaWF0Ijoi",
  "type": "public-key"
}

```

IBM Security Access Manager validates the assertion result and either allows the user to continue as expected, or returns an error.

Note: Unlike the FIDO2 Server Endpoints, the relying party ID is not consumed from the address path, but instead is obtained from mechanism or policy level configuration.

Limitations

There is no out-of-the-box support for registration of authenticators by using SCIM or a registration mechanism.

Registrations can only be performed by using the attestation endpoint. See “FIDO2 Server Endpoints” on page 193.

FIDO2 Mediation

When you are deploying FIDO2/WebAuthn it might be necessary to implement specific business controls during authentication and registration ceremonies. FIDO2 Mediators can be used to add this extra validation, as well as insert additional data into the server responses.

Mediators take the form of a JavaScript mapping rule which runs in four places

1. As part of the initiation of a registration attempt (a call to the attestation options endpoint).
2. As part of an completion of a registration attempt, after all FIDO2 and WebAuthn checks have passed, but before the registration is considered validated (a call to attestation result).
3. As part of the initiation of an authentication attempt (a call to the assertion options endpoint).
4. As part of the completion of an authentication attempt (a call to the assertion result endpoint).

Mediation also occurs as part of an authentication service flow by using FIDO2/WebAuthn as an authentication mechanism.

The capabilities of mediation are as follows:

1. During any mediation, FIDO2 processing can be halted with a custom message and status. To do this, populate the error field:


```

error.put("status", "authenticator_denied");
error.put("message", "An administrator has disabled your authenticator type from being used");

```

2. During mediation of an attestation result request, additional variables can be saved in the “attributes” parameter of the new registration. Add to the “attributes” parameter with:

```
attributes.put("example_attribute", "example_value");
```

Note: Only attributes of type String can be stored.

3. During mediation of an attestation result or assertion result call, additional values can be returned to the browser:

```
responseData.put("did_user_verify", context.requestData.registration.userVerified);
```

The response is returned to the browser to let it act based on the authenticator and its registration. For example, this could be used to return the model name of the authenticator.

4. During mediation of an attestation result or assertion result call, additional credential attributes can be added to the users session:

```
credentialData.put("authenticator_friendly_name", context.requestData.registration.friendlyName);
```

Credential data returned indicates the users identity. Credential data is returned to the relying party in an API call to /attestation/result or /assertion/result, or when the authentication service is used to perform the assertion request, the user's ISAM credential is enriched.

The mediator context

The context variable available in mediation contains several essential pieces of information about the request.

1. `context.requestType`- Identifies which one of the following mediation is occurring:
 - `attestation_options`
 - `attestation_result`
 - `assertion_options`
 - `assertion_result`
2. If enabled, HTTP request context (such as headers) are made available under:
 - `context.requestData.headers`
 - `context.requestData.cookies`

Note: HTTP context is not enabled by default, enable the advanced configuration entry `sps.httpRequestClaims` to include headers and cookies in the context. See “Advanced configuration properties” on page 219.

3. In the case of an assertion options or attestation options request, this property contains the properties to be returned to the browser. In the case of an assertion result or attestation result request, this property indicates the options which were provided to the client as part of this FIDO2/WebAuthn ceremony.
4. The registration that is used (in the case of an assertion request), or the registration about to be saved (in the case of an attestation request) is under `context.requestData.registration`

Other FIDO2 values such as the `attestationStatement`, `authData`, and `clientData` are also be available under `context.requestData`. The context object is native JavaScript, and can be accessed as an associative array.

When developing mediators the following line can be used to print the entire context variable:

```
IDMappingExtUtils.traceString(JSON.stringify(context));
```

Note: Ensure `com.tivoli.am.fim.trustserver.sts.utilities.*=ALL` trace is enabled.

A complete list of variables available in the mediator context are:

1. `stsuu`- The users current identity. Read only.
2. `context`- All of the context data. Read only. Native JSON.
3. `responseData`- Java HashMap, writable. Available only in attestation result and assertion result requests. Response are returned to the end user.
4. `credentialData`- Java HashMap, writable. Available only in attestation result and assertion result requests. Values to go into the credential (in API usage are returned to the user caller same as `responseData`).
5. `attributes`- Java HashMap, writable. Values are stored in the registration. Only available in attestation result requests. Values are available in assertion result requests in `context.requestData.registration.attributes`
6. `error`- Java HashMap, writable, Available in all options or results requests. If the message and status keys are populated they are returned to the caller as an error.

The traditional ISAM Java helpers are available in the mediator context, such as `UserLookupHelper`, `LocalSTSCient`, `IDMappingExtUtils`, and `OAuthMappingExtUtils`.

Chapter 13. Access control policies

An access control policy is a set of conditions that, after they have been evaluated, determine access decisions.

Defining a custom application for policy attachments

There are two types of Access Control Resources, Reverse Proxy or Application.

A Reverse Proxy resource defines a server instance with a protected object space, and a specific resource in that protected object space. An application resource describes an application server and resource that you would like to protect that is not in a Reverse Proxy object space.

Ensure that the application ID is unique. The application ID is case-sensitive; for example "ClaimApplication" and "claimapplication" are considered to be unique names.

Note: Avoid control characters, leading and trailing blanks, and special characters such as ! @ # \$ % ^ & * [] ; , < >

Application IDs and resources are used as either URL paths or URI scheme names and therefore must consist of a sequence of any combination of lowercase letters, numbers, or any of the following special characters: plus ("+"), period ("."), or hyphen ("-").

If a URL path is used, the ID must begin with a forward slash ("/"). If a URI scheme name is used, the ID must begin with a lowercase letter.

For an application resource with an Application ID /myapp and Resource ID /myresource, the corresponding XACML JSON would be:

```
{
  "Request": {
    "Action": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:action:action-id",
          "DataType": "string", "Value": "GET"
        }
      ]
    },
    "Resource": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "DataType": "string", "Value": "/myresource"
        }
      ]
    }
  },
  "Environment": {
    "Attribute": [
      {
        "AttributeId": "ApplicationId", "DataType": "string", "Value":
          "/myapp", "Issuer": "http://security.tivoli.ibm.com/policy/distribution",
      }
    ]
  }
}
```

```

    }
  }
}

```

Invoking the RTSS XACML engine

The RTSS XACML engine can be invoked directly to retrieve policy decisions.

Both Reverse Proxy or Application resources can be used in an RTSS request. A JSON endpoint that roughly adheres to the XACML JSON specification can be accessed via:

```
https://{runtime_hostname}/rtss/rest/authz/json
```

To determine which policy to evaluate, the engine will lookup configured policy attachments via a policy key. The key corresponds to the concatenation of the resource server and resourceUri. For example, a policy attachment with server `isam.ibm.com-default` and resourceUri `/protected` will be referenced by the key `isam.ibm.com-default/protected`. This key is required when sending the JSON request.

The engine will attempt to find the policy key via the `Request.Environment` attributes `ContextId` or `ApplicationId`, and if neither are set then the `Request.Resource resource-id` attribute will be used.

ContextId JSON example

The `ContextId` attribute must contain the full policy key, that is the server and the resourceUri.

For a reverse proxy resource with the server `isam.ibm.com-default` and resourceUri `/protected`, the corresponding XACML JSON request would be:

```

{
  "Request": {
    "Action": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:action:action-id",
          "DataType": "string", "Value": "GET"
        }
      ]
    },
    "Resource": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "DataType": "string", "Value": "/protected"
        }
      ]
    },
    "Environment": {
      "Attribute": [
        {
          "AttributeId": "ContextId", "DataType": "string", "Value":
            "/WebSEAL/isam.ibm.com-default/protected", "Issuer":
            "http://security.tivoli.ibm.com/policy/distribution",
          "Value": ""
        }
      ]
    }
  }
}

```

If the policy attached to `isam.ibm.com-default/protected` results in a Permit decision, the XACML JSON response would be:

```
{
  "Response": [
    {
      "Status": {
        "StatusCode": {
          "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
        }
      },
      "Decision": "Permit"
    }
  ]
}
```

ApplicationId JSON example

The `ApplicationId` attribute must only contain the server part of the full policy key. The `resourceUri` is then retrieved from the `Request.Resource resource-id` attribute and concatenated on the `ApplicationId`.

For an application resource with an Application ID `/myapp` and two resources, `/myresource1` and `/myresource2`, two policy keys would be generated, `/myapp/myresource1` and `/myapp/myresource2`.

This allows two separate policies to be evaluated within the one JSON request.

The corresponding XACML JSON would be:

```
{
  "Request": {
    "Action": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:action:action-id",
          "DataType": "string", "Value": "GET"
        }
      ]
    },
    "Resource": [
      {
        "Attribute": [
          {
            "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
            "DataType": "string", "Value": "/myresource1"
          }
        ]
      },
      {
        "Attribute": [
          {
            "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
            "DataType": "string", "Value": "/myresource2"
          }
        ]
      }
    ]
  },
  "Environment": {
    "Attribute": [
      {
        "AttributeId": "ApplicationId", "DataType": "string",
        "Value": "/myapp", "Issuer":
          "http://security.tivoli.ibm.com/policy/distribution",
      }
    ]
  }
}
```

```

    }
  }
}

```

If the policy attached to /myapp/myresouce1 results in a Permit decision and the policy attached to /myapp/myresouce2 results in a Deny decision, the XACML JSON response would be:

```

{
  "Response": [
    {
      "Status": {
        "StatusCode": {
          "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
        }
      },
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "Value": "\myresource1"
        }
      ],
      "Decision": "Permit"
    },
    {
      "Status": {
        "StatusCode": {
          "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
        }
      },
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "Value": "\myresource2"
        }
      ],
      "Decision": "Deny"
    }
  ]
}

```

resource-id JSON example

When neither the ContextId or ApplicationId attributes are set, the Request.Resource resource-id attribute is used as the policy key.

This allows two separate policies to be evaluated within the one JSON request.

The corresponding XACML JSON would be:

```

{
  "Request": {
    "Action": {
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:action:action-id",
          "DataType": "string", "Value": "GET"
        }
      ]
    },
    "Resource": [
      {
        "Attribute": [
          {
            "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",

```

```

        "DataType": "string", "Value": "/WebSEAL/isam.ibm.com-default/protected"
    }
  ],
  {
    "Attribute": [
      {
        "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
        "DataType": "string", "Value": "/myapp/myresource1"
      }
    ]
  }
],
"Environment": {
  "Attribute": [
  ]
}
}
}

```

If the policy attached to isam.ibm.com-default/protected results in a Permit with Obligation decision and the policy attached to /myapp/myresource1 results in a NotApplicable decision, the XACML JSON response would be:

```

{
  "Response": [
    {
      "Status": {
        "StatusCode": {
          "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
        }
      },
      "Obligations": [
        {
          "Id": "ObligationId"
        }
      ],
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "Value": "/WebSEAL/isam.ibm.com-default/protected"
        }
      ],
      "Decision": "Permit"
    },
    {
      "Status": {
        "StatusCode": {
          "Value": "urn:oasis:names:tc:xacml:1.0:status:ok"
        }
      },
      "Attribute": [
        {
          "AttributeId": "urn:oasis:names:tc:xacml:1.0:resource:resource-id",
          "Value": "/myapp/myresource1"
        }
      ],
      "Decision": "NotApplicable"
    }
  ]
}

```

Chapter 14. Defining a custom domain for policy attachments

The administrator can specify a custom domain to separate metadata in a registry. For example, your company might possess metadata that belongs to several companies, and it is a security demand that the data does not overlap.

About this task

The policy attachment credential automatically selects the default management domain in all supported versions of IBM Tivoli® Access Manager when you integrate it with the IBM Security Access Manager local management interface. You must choose one domain to use for policy attachments.

Procedure

1. Log in to the local management interface.
2. Specify the Tivoli Access Manager administrator credentials when you create a new reverse proxy instance:
 - a. Select **Secure Web Settings > Manage > Reverse Proxy**  **New**.
 - b. Select the **IBM Security Access Manager** tab.
 - c. Specify the following administrator credentials. These credentials must be the same as the ones that you use to attach a policy to a domain other than the default.
 - **Administrator Name**
 - **Administrator Password**
 - **Domain**
3. Select **Secure Access Control > Policy > Access Control > Resources**.
4. Click  .
5. Enter the information that you specified in 2c at **Policy Server Login**.

What to do next

You can reset the credentials that you just defined with the **setCredential** parameter under the following conditions:

- You upgrade to IBM Security Access Manager, version 8.0.0.4 or later.
- You want to manage a domain name other than the default.

Before you reset the **setCredential** parameter, remove all current resources and their corresponding policy attachments. For more information about this command, go to the REST API documentation and select **Policy Attachments > Resources > Authenticate with Security Access Manager**.

Chapter 15. Deploying pending changes

Some configuration and administration changes require an extra deployment step.

About this task

When you use the graphical user interface on the appliance to specify changes, some configuration and administration tasks take effect immediately. Other tasks require a deployment step to take effect. For these tasks, the appliance gives you a choice of deploying immediately or deploying later. When you must make multiple changes, you can wait until all changes are complete, and then deploy all of them at one time.

When a deployment step is required, the user interface presents a message that says that there is an undeployed change. The number of pending changes is displayed in the message, and increments for each change you make.

Note: If any of the changes require the runtime server to be restarted, the restart occurs automatically when you select **Deploy**. The runtime server will then be unavailable for a period of time until the restart completes.

Procedure

1. When you finish making configuration changes, select **Click here to review the changes or apply them to the system**.

The Deploy Pending Changes window is displayed.

2. Select one of the following options:

Option	Description
Cancel	Do not deploy the changes now. Retain the undeployed configuration changes. The appliance user interface returns to the previous panel.
Roll Back	Abandon configuration changes. A message is displayed, stating that the pending changes were reverted. The appliance user interface returns to the previous panel.
Deploy	Deploy all configuration changes. When you select Deploy , a system message is displayed, stating that the changes were deployed. If any of the changes require the runtime server to be restarted, the restart occurs automatically when you select Deploy . The runtime server will then be unavailable for a period of time until the restart completes.

Chapter 16. Options for handling session failover events

Advanced Access Control offers several solutions to the challenge of providing sharing of session state across multiple servers in a clustered environment.

The following sections describe the options available for handling failover events in clustered environments:

- No handling of failover events
- The Distributed Session Cache

Option 1: No handling of failover events

Failover events are rare when WebSEAL or Web Reverse Proxy instance is configured to maintain session affinity in a stateful junction to Advanced Access Control.

This scenario is applicable in the case of using the **isamcfg** tool to configure the junction.

When failover events do occur, session state is lost and clients might be required to restart their current transaction.

This option is configured by default. However, there is a risk of a poor user experience when:

- The server containing Advanced Access Control becomes unavailable
- The WebSEAL or Web Reverse Proxy cannot maintain session affinity

Option 2: The distributed session cache

The distributed session cache (DSC) can be used for session storage by all Security Access Manager appliances in a cluster.

When a fail over event occurs, Security Access Manager appliance retrieves the session data of the user from the DSC. It therefore maintains the existing session state.

Within Security Access Manager, the DSC is part of the cluster configuration. For more information about turning on or turning off this feature, see the Distributed Session Cache section in Advanced configuration properties and Managing Distributed Session Cache.

Chapter 17. Global settings

You can use the LMI to access an administrative menu to configure global settings that are used by both Federation and Advanced Access Control.

The Local Management Interface (LMI) has a user interface page for administering each major feature in IBM Security Access Manager. Since some features are used by multiple licensing levels for the product, the administration page for these features can be accessed through multiple user interface menu paths.

You can use either of the following LMI menus to access the global settings:

- **Secure Access Control > Global Settings**
- **Secure Federation > Global Settings**

You can use the global settings menus to configure the following features:

- Advanced Configuration

Some of the advanced configuration properties are common to Advanced Access Control and Federation. Others are specific to one of the licensing levels.

- User Registry

Use these settings to administer users and group memberships for the user registry that is used by the runtime applications. Management tasks are common to Advanced Access Control and Federation.

- Runtime Parameters

You can use the Runtime Parameters menu to view runtime status, tune runtime parameters, and set tracing on the runtime. These functions are common to Advanced Access Control and to Federation.

In addition, the runtime tracing feature can be set in the LMI through **Monitoring Analysis and Diagnostics > Logs > Runtime Tracing > ..**

The topic for Runtime Parameters is also included in the appliance troubleshooting section of the IBM Knowledge Center. See Tuning runtime application parameters and tracing specifications

- Template Files

Template files are HTML pages that are presented to your users. You can customize the content of the pages for your deployment by setting supported macros, or by adding JavaScript scripting. Template pages are used in multiple scenarios.

- Customizing the authentication process, such as error messages
- Specifying settings for the supported authentication mechanisms
- Customizing error messages for authentication attempts
- Obtaining consent for registering devices
- Specifying authorization parameters for OAuth 2.0
- Configuring user self-care tasks

- Mapping Rules

Mapping rules are JavaScript code that runs during the authentication flow for Advanced Access Control and Federation. Mapping rules can be used for multiple purposes. For Advanced Access Control, you can modify rules for the Authentication Service, OTP, and OAuth 2.0. For Federation, you can modify mapping rules to manage identities for OIDC and SAML 2.0.

- Distributed Session Cache

The Distributed Session Cache is supplied by the Web Reverse Proxy and is used with all activation levels. The management windows in the LMI can also be accessed through **Secure Web Settings > Manage > Distributed Session Cache**.

For an overview of the Distributed Session Cache, and a review of advanced configuration options, see: Distributed session cache.

- Server Connections

Advanced Access Control and Federations both use the IBM Security Access Manager appliance to connect to external data sources. For Advanced Access Control, you can use the server connections menus to configure LDAP or database server connections so that you can set up policy information points. For Federation, you can configure an LDAP server as an attribute source for attribute mapping.

- Point of Contact

IBM Security Access Manager provides servers, such as WebSEAL, that function as point of contact servers for handling external requests for authentication and authorization. You can configure a point of contact profile to specify the information that is needed for the runtime to communicate with a specific point of contact server. Security Access Manager provides three Point of Contact profiles that are ready for use. You can specify callback parameters and values for these profiles.

- Access Policies

You can use access policies to perform step-up and reauthentication during a single sign-on flow based on contextual information. Access policies can be enforced at a federation or at API Protection for OAuth and OpenID Connect.

Note: The LMI mega-menu for the **Secure Web** licensing level also presents a set of tasks under a Global Settings heading. These tasks are different from the tasks under the **Global Settings** menu for **Secure Access** and **Secure Federation**. The **Secure Web > Global Settings** LMI menus are not used with **Secure Access** and **Secure Federation**.

Managing advanced configuration

Adjust configuration settings in supported configuration files.

About this task

The advanced configuration panel displays a table of configuration settings. Some can be modified and some are read-only. Each setting is displayed as a row in the table. The name of the setting is listed in the *key* column. The current value of the key is listed in the *value* column. You can locate a setting by using one of the following methods:

- Scroll through the list until you see the setting.
By default, all configuration settings are included in the list.
- Filter the list by entering a string in the **Filter** field.
When you enter a string, the list is modified to show only the settings that contain the specified string.
- Filter the list by selecting a category from the **Filter by Category** menu.

For descriptions of the categories and properties, see “Advanced configuration properties” on page 219.

Procedure

1. Select the menu entry for your licensing level:
 - If using an Advanced Access Control license, select **Secure Access Control > Global Settings > Advanced Configuration**.
 - If using a Federation license, select **Secure Federation > Global Settings > Advanced Configuration**

2. To edit a key, select the edit icon  for the key.

Note: You cannot edit keys that are marked with the read-only icon: .

When you choose to edit a key, a new window displays the name of the key and the current value.

3. Edit the value for your deployment.

Table 21. Configuration data types

Data type	Action
Integers	Use the arrow icons to increment or decrement the value. Alternatively, you can type a new value in the field.
Strings	Enter a string value in the field. Note: This field must have a value. You cannot specify an empty field. To clear values from a field, enter NULL.
Booleans	Select the check box to set the value to true. Clear the check box to set the value to false.

4. Click **OK**.
5. Deploy the changes.

Advanced configuration properties

Modify the advanced configurations for Advanced Access Control or Federation to meet the requirements of your organization.

Category filter

The category filter displays names of grouping of configuration settings. The groupings correspond to functional areas. When you select a category, the user interface displays only the settings for the category.

Table 22. Filter by Category

Category	Displays values for:
All	All keys
poc.websealAuth	"WebSEAL Authenticate Callback" on page 220
poc.otpAuth	"One-time password Authenticate Callback" on page 221
poc.authPolicy	"Authentication-Policy Callback" on page 221
sps.httpRequestClaims	"SPS HTTP request claims" on page 221
distributedMap	"Distributed shared data storage" on page 221
userBehavior	"Attribute matcher properties" on page 222

Table 22. Filter by Category (continued)

Category	Displays values for:
ipReputation	"IP reputation PIP properties" on page 222
attributeCollection	"Attribute collector properties" on page 223
deviceRegistration	"Device registration properties" on page 224
runtime	"Runtime properties" on page 225
sps.page	"SPS page" on page 227
sps	"Single sign-on protocol service" on page 226
riskEngine	"Risk engine properties" on page 227
sps.authService	"Authentication service properties" on page 228
authsvc.stateMgmt	"Authentication service session store properties" on page 229
session	"Session" on page 230
distributedSessionCache	"Distributed session cache" on page 230
otp.retry	"TOTP and HOTP retry properties" on page 231
oauth20	"OAuth20" on page 232
util.httpClient	"HTTP client" on page 234
demo	"Demo" on page 235
knowledge.questions	"Knowledge questions properties" on page 236
kess	"Key encryption and signing service (KESS)" on page 236
jwks	"JSON Web Key" on page 238
pip	"Policy information point (PIP)" on page 238
sts	"Security token service (STS)" on page 238
mmfa	Mobile Multi-Factor Authentication (MMFA)
wsfed	"WS-Federation" on page 240
saml20	"SAML 2.0" on page 241
demo	"Demo" on page 235
saml11	"SAML 1.1" on page 240
oidc	"OIDC" on page 242

WebSEAL Authenticate Callback

`poc.websealAuth.authLevel`

The authentication level of the callback.

Data type: Integer

Example: 1

One-time password Authenticate Callback

poc.otp.authLevel

The authentication level of the callback.

Data type: Integer

Example: 2

poc.otp.backwardCompatibilityEnabled

Indicates whether the one-time password authentication mechanism should run in backward compatibility mode. The default value is false if it is a new installation. The default value is true if the installation is an upgrade.

Data type: Boolean

Example: true

Authentication-Policy Callback

poc.authPolicy.allowRequestOverride

Whether the authentication level, the authentication mode, and the authentication type of the callback can be overwritten by query string parameters.

Data type: Boolean

Example: true

poc.authPolicy.authLevel

The authentication level of the callback.

Data type: Integer

Example: 1

poc.authPolicy.authType

The authentication type of the callback.

Data type: String

Example: COMPLEMENTARY, HIERARCHICAL

SPS HTTP request claims

sps.httpRequestClaims.enabled

Whether HTTP request information is sent to STS as HTTPRequestClaims.

Data type: Boolean

Example: false

sps.httpRequestClaims.filterSpec

The filter that specifies the HTTP request information that is sent to STS as HTTPRequestClaims.

Data type: String

Example: cookies=:headers=*

Distributed shared data storage

distributedMap.cleanupWait

The amount of time, in milliseconds, to wait before it performs another cleanup against the distributed map.

Distributed map clean up can be disabled by setting the cleanupWait to 0.

Data type: Integer

Example: 10000

distributedMap.defaultTTL

The amount of time, in seconds, that the entries in the distributed map must live when no lifetime is specified for an entry.

Data type: Integer

Example: 3600

distributedMap.getRetryDelay

The amount of time, in milliseconds, to wait before it performs another retrieval against the distributed map. The default is 0.

Data type: Integer

Example: 500

distributedMap.getRetryLimit

The number of retrievals that is done against the distributed map before it returns that the retrieved data is not in the distributed map. The default is 0.

Data type: Integer

Example: 10

Attribute matcher properties

userBehavior.minimumUsageHistoryRequired

Minimum usage data records required for any usage data analysis; used by LoginTimeMatcher.

Data type: Integer

Example: 8

userBehavior.ipAddressRequestAttribute

The XACML request attribute to read from the IP address.

Data type: String

Example: urn:ibm:security:subject:ipAddress

IP reputation PIP properties

ip.reputation.ipAddressAdverseReputationThreshold

The value that an IP classification score must be at or above for an IP address to be considered as that classification.

Data type: Integer

Example: 50

ipReputation.dbConnectionTimeout

Indicates the number of seconds that the IP reputation policy information point (PIP) waits for a connection to the IP reputation database. The ipReputation.dbConnectionTimeout property defaults to 120.

Data type: Integer

Example: 60

Attribute collector properties

attributeCollection.cookieName

Correlation ID used by the attribute collector.

Data type: String

Example: ac.uuid

attributeCollection.requestServer

Request server for attribute collector. A list of the allowable hosts where the ajaxRequest can be sent from.

Data type: String List

Example: https://rbademo.example.com,https://rbaemo2.example.com

attributeCollection.serviceLocation

Location of the attribute collector.

Data type: String List

Example: http://rbademo.example.com/mga

attributeCollection.sessionTimeout

Number of seconds in which sessions stored in context-based access will automatically expire, unless updated. If any attribute in the session is updated, the session expiry is extended by the specified number of seconds configured in this property. The default is 1800 seconds.

Data type: Integer

Example: 1800 seconds

attributeCollection.enableGetAttributes

Enables the REST GET method to return attributes.

Data type: Boolean

Example: false

attributeCollection.getAttributesAllowedClients

A comma-separated list of clients that are allowed to access the ACS REST GET method.

If this property is not set and attributeCollection.enableGetAttributes is set to true, anyone can access the GET method. If this property is set but attributeCollection.enableGetAttributes is set to false, this property is ignored.

Data type: String List

Example: hostname1, hostname2

attributeCollection.hashAlgorithm

The algorithm that is used to create the hash.

Data type: String

Example: SHA256

attributeCollection.attributesHashEnabled

A comma-separated list of attribute URI values configured for hashing.

Attention: Do not hash the following attributes:

- ipAddress
- geoLocation
- accessTime

Data type: String List

Example:

```
urn:ibm:security:environment:http:userAgent,  
urn:ibm:security:environment:deviceFonts,  
urn:ibm:security:environment:browserPlugins
```

attributeCollection.authenticationContextAttributes

Comma-separated lists of attribute names to be collected during an authentication service obligation. The maximum number of characters for this property is 200.

Data type: String List

Example: authenticationLevel, http:host

Device registration properties

deviceRegistration.allowIncompleteFingerprints

Specifies to allow the device registration obligation to store fingerprints where all the fingerprint attributes are not available on the session information.

Data type: Boolean

Example: false

deviceRegistration.checkForExpiredDevices

Determines whether registered devices are inactive or expired. If the deviceRegistration.checkForExpiredDevices property is set to true, the risk engine checks whether a device is inactive or expired. The deviceRegistration.checkForExpiredDevices property defaults to false, which means that users can use any of the devices that are registered.

Data type: Boolean

Example: true

deviceRegistration.cleanupThread.batchSize

Specifies if batch delete is enabled for expired devices and how many records are deleted per batch.

If the value is defined as 0 or is blank, batch delete is not enabled and all expired devices are deleted using one SQL delete statement.

If the value is defined as an integer greater than 0, batch delete is enabled. The number that you specify determines how many records are deleted in each batch. The batch delete continues until all of the expired devices are deleted. The batch process is useful for deleting a large quantity of expired devices.

Data type: Integer

Example: 1000 (Batch delete is enabled, with a batch size of 1000 records.)

deviceRegistration.deviceMatchThreshold

The risk score threshold where an existing fingerprint is considered to match the incoming device fingerprint.

Data type: Integer

Example: 20

deviceRegistration.inactiveExpirationTime

Specifies the number of days that a device must be inactive for it to expire. The deviceRegistration.inactiveExpirationTime property defaults to 90.

Data type: Integer

Example: 100

deviceRegistration.maxRegisteredDevices

Maximum device fingerprint count. The default is 10. Valid values are 1 to 100.

Data type: Integer

Example: 10

deviceRegistration.maxUsageDataPerUser

Maximum number of historical usage attribute records stored per user. The default is 200. Valid values are 1 to 5000.

Data type: Integer

Example: 1000

deviceRegistration.permitOnIncompleteFingerprints

Specifies to permit access to the resource if the fingerprint collected by the device registration obligation does not include all fingerprint attributes.

Data type: Boolean

Example: false

Runtime properties

runtime.dbLoggingEnabled

Enables fine-grained logging for database SQL statements.

Data type: Boolean

Example: false

runtime.hashAlgorithm

The algorithm that is used for hashing. The supported algorithms are:

- SHA-1
- SHA-256
- SHA-384
- SHA-512

The runtime.hashAlgorithm property defaults to SHA-256.

Data type: String

Example: SHA-256

runtime.verifcationHashAlgorithms

Defines the hashing algorithms that are used to verify a hashed value. The value is typically a comma separated list of hashing algorithms.

Data type: String

Example: SHA-256, SHA-1

Single sign-on protocol service

sps.setCookiesAsSecure

Determine whether to flag the cookies set by Security Access Manager as secure.

The default value is false.

Data type: Boolean

Example: false

sps.targetURLWhitelist

Specifies a list of allowed target URLs for SAML 2.0, OpenID Connect, and the authentication service. Use this property to prevent an attacker from redirecting a user to malicious target URLs.

The value of this advanced configuration property is a comma-separated string, where each string is a target URL in the form of a regular expression. The regular expression must not contain commas, and spaces between regular expressions are ignored.

- For SAML 2.0 SSO flows, you can specify a Target URL when you configure the initial URL in flows that are initiated by either the Identity Provider or the Service Provider. For more information, see SAML 2.0 profile initial URLs.
- For Open ID Connect flows, you can specify a Target URL when you configure the initial URL for Relying Party initiated single sign-on. For more information, see Relying Party SSO initiation endpoint.
- For the authentication service, you can specify a Target URL when you configure the authentication service trigger URL. For more information, see “Configuring authentication” on page 39.

The default value is “.*”.

Data type String

Example

(http|https)://www.app.ibm.com/.*, (http|https)://www.myidp.ibm.com/.*

sps.illegalUrlSubstrings

A comma-separated list of strings, the single sign-on service stops processing the request if the request URL query parameters contain any of the strings.

The default value is "".

Data type: String

Example:

"<script"

sps.doNotSendXFrameOptionsHeader

Specifies whether an X-Frame-Options header with value SAMEORIGIN must be returned from the SPS endpoints for browser based flows. When this property is set to true, no X-Frame-Options header is sent.

Note: The sps.doNotSendXFrameOptionsHeader property defaults to false.

Data type: Boolean

Example: False

SPS page

sps.page.htmlEscapedMacros

A comma-separated list of macros that is HTML-escaped when it is rendered in pages that are sent to the browser.

Data type: String

Example:

```
@REQ_ADDR@,  
@DETAIL@,  
@EXCEPTION_STACK@,  
@EXCEPTION_MSG@,  
@OTP_METHOD_ID@,  
@OTP_METHOD_LABEL@,  
@OTP_HINT@,  
@ERROR_MESSAGE@,  
@MAPPING_RULE_DATA@
```

sps.page.exceptionMacros

A comma-separated list of classname:macro pairs. Classname is the fully qualified name of the exception class. Macro is the name of the macro to which the class maps.

Data type: String

Example:

```
com.tivoli.am.fim.otp.deliveries.OTPDeliveryException =  
    @OTP_DELIVERY_EXCEPTION@,  
com.tivoli.am.fim.otp.providers.OTPProviderException =  
    @OTP_PROVIDER_EXCEPTION@
```

sps.page.notEscapedMacros

A comma-separated list of macros that are **not** HTML-escaped when they are rendered in pages that are sent to the browser. Macros that do not appear in this list or the Macros in the **htmlEscapedMacros** list are HTML-escaped.

Data type: String

Example:

```
@COOKIE_NAME@,  
@SERVER_NAME@,  
@JUNCTION@
```

sps.page.hiddenMacros

A comma-separated list of macros that are not rendered in the pages that are sent to the browser. The default value is @EXCEPTION_STACK@.

Data type: String

Example: @EXCEPTION_STACK@

Risk engine properties

riskEngine.reportsEnabled

Enables the generation of risk calculation reports.

Data type: Boolean

Example: false

riskEngine.reportsMaxStored

Specifies the maximum number of reports to store.

Data type: Integer

Example: 5

Authentication service properties

sps.authService.reauthenticationEnabled

Specifies that the authentication service performs authentication even if the user already has an authenticated session at the required authentication level.

Data type: Boolean

Example: true

sps.authService.policyKickoffMethod

Specifies whether the URLs /sps/authsvc and /sps/apiauthsvc can be invoked with the **policyId** query string parameter.

If set to query, the authentication service endpoints continue to accept **policyId** as a query or post parameter.

If set to path, authentication service endpoints are changed to:

- /sps/apiauthsvc/policy/<shortPolicyId>
- /sps/authsvc/policy/<shortPolicyId>

Where <shortPolicyId> is the value that comes after the prefix urn:ibm:security:authentication:asf:

By default, the value is set to both.

When set to both, either the path or query parameter can be used to initiate an authentication service flow.

sps.authService.stateIdSource.authsvc

Specifies whether the URL /sps/authsvc can be invoked with the **StateId** query string parameter.

If set to **Body and Query**, the authentication service endpoint continues to accept StateId as a query or body parameter.

If set to **Body Only**, the authentication service endpoint only accepts the StateId as a body parameter (POST or PUT).

Data type: String

Default: Body and Query

Example: Body only

sps.authService.stateIdSource.apiauthsvc

Specifies whether the URL /sps/apiauthsvc can be invoked with the **StateId** query string parameter.

If set to **Body and Query**, the API authentication service endpoint continues to accept StateId as a query or body parameter.

If set to **Body Only**, the API authentication service endpoint only accepts the StateId as a body parameter (POST or PUT).

Data type: String

Default: Body and Query

Example: Body Only

Authentication service session store properties

authsvc.stateMgmt.cookieless

Enables the server side storage of session data for the authentication service. If enabled, this removes the need for the JSESSIONID cookie.

Data type: Boolean

Example: true

Default value: true

authsvc.stateMgmt.store

Specifies the storage type that is used by the Authentication service to cache user session data. The authentication service can be supported by the DSC, the HVDB, or stored in Memory.

Note: For clustered environments, storage in Memory does not replicate between nodes.

Data type: String

Example: Memory

Default value: HVDB

authsvc.stateMgmt.HVDB.lifetime

Length of time in seconds that a session is cached for. Once this time period is exceeded, the user's session is removed from the session store. If this value is less than 0, the default lifetime of 3600 seconds (1 hour) is enforced. This configuration option applies only to session stores supported by the HVDB or Memory.

Data type: Integer

Example: 60 (1 minute)

Default value: 3600

authsvc.stateMgmt.HVDB.maxSessions

Maximum number of user sessions to be cached at any point in time. If the number of sessions in the store exceeds this value, the oldest session is invalidated. This configuration option only applies to session stores that are supported by the HVDB or Memory.

Data type: Integer

Example: 10000

Default value: 1000

authsvc.stateMgmt.HVDB.cleanupWait

Frequency (in seconds) that expired or excess sessions are removed from the session store. Setting this entry to -1 disables the cleanup thread. This configuration option only applies to session stores backed by the HVDB or Memory.

Data type: Integer

Example: 30

Default value: 120

authsvc.stateMgmt.HVDB.cleanupThread.batchSize

Maximum number of expired sessions which are removed in a single cleanup operation. If the value is defined as 0 or is blank, batch delete is

not enabled. All expired sessions are deleted by using one SLQ delete statement. If the value is defined as an integer greater than 0, batch delete is enabled. The number that you specify determines how many sessions are deleted in each batch. The batch delete continues until all of the expired sessions are deleted. This configuration option only applies to sessions that are stored in the HVDB or Memory.

Data type: Integer

Example: 1000

Default value: 0

authsvc.stateMgmt.HVDB.cleanupOnlyOnPrimaryMaster

Prevent the cleanup thread from running on non-primary master nodes in a clustered environment. This configuration option only applies to sessions that are stored in the HVDB or Memory.

Data type: Boolean

Example: true

Default value: true

Session

distributedSessionCache.enabled

A switch that dictates if the distributed session cache is used for session failover. If this setting is not enabled, the distributed session cache server still runs as a service, but the client does not use it.

Data type: Boolean

Example: false

distributedSessionCache.localCacheSize

The number of sessions to be stored on the client as a local cache. A value of 0 or less means that any number of sessions can be cached by the client. A low number requires more connections to the distributed session cache if there are many active sessions. A high number runs the risk of running out of memory if many sessions are locally cached. All sessions are still stored on the distributed session cache when it is enabled.

Data type: Integer

Example: 4096

session.dbCleanupInterval

Specifies the interval, in seconds, that the database cleanup thread runs to remove expired data in the runtime database. The default is 86400. The minimum value for this property is 3600. For more information, see Runtime database tuning parameters

Session database clean up can be disabled by setting the dbCleanupInterval to 0. This is not overridden by the minimum value.

Data type: Integer

Example: 90000

Distributed session cache

distributedSessionCache.enabled

A switch that dictates if the distributed session cache is used for session

failover. If this setting is not enabled, the distributed session cache server still runs as a service, but the client does not use it.

Data type: Boolean

Example: false

distributedSessionCache.localCacheSize

The number of sessions to be stored on the client as a local cache. A value of 0 or less means that any number of sessions can be cached by the client. A low number requires more connections to the distributed session cache if there are many active sessions. A high number runs the risk of running out of memory if many sessions are locally cached. All sessions are still stored on the distributed session cache when it is enabled.

Data type: Integer

Example: 4096

distributedSessionCache.externalServers

A list of locations of the distributed session cache servers in weighted order.

Syntax:

`<primary_address>:<port>[:<ssl>];<secondary_address>:<port>[:<ssl>],...`

<address>

The IP address of the distributed session cache server. For example, 10.150.21.80.

<port>

The port for the distributed session cache. For example, 2126.

<ssl>

Whether SSL communication with the distributed session cache is required. The default value is false.

Data type: String

Example:

10.150.21.80:2126:true;10.150.21.81:2126:false,10.150.21.82:2126

distributedSessionCache.localCacheEnabled

A switch that dictates whether a local cache of distributed sessions is maintained. If this setting is disabled a higher load is placed on the distributed session cache server. The local cache should only be enabled if all requests from the same client is guaranteed to be sent to the same runtime server (otherwise known as stickiness). Session inconsistencies might occur if the local cache is enabled and stickiness is not maintained. All sessions are still stored in the distributed session cache when it is enabled.

Data type: Boolean

Example: False

TOTP and HOTP retry properties

otp.retry.enabled

Whether the retry protection is enabled.

Data type: Boolean

Example: true

otp.retry.maxNumberOfAttempts

The maximum number of strikes the users can have before they are prevented from logging in.

Data type: Integer

Example: 5

otp.retry.otpRetryTimeout

The number in seconds a strike lasts.

Data type: Integer

Example: 600

OAuth20

oauth20.cleanupThread.batchSize

Specifies if batch delete is enabled for expired OAuth 2.0 tokens and how many records are deleted per batch.

If the value is defined as 0 or is blank, batch delete is not enabled and all expired OAuth tokens are deleted using one SQL delete statement.

If the value is defined as an integer greater than 0, batch delete is enabled. The number that you specify determines how many records are deleted in each batch. The batch delete continues until all of the expired OAuth tokens are deleted. The batch process is useful for deleting a large quantity of expired tokens.

Data type: Integer

Example: 1000 (Batch delete is enabled, with a batch size of 1000 records.)

oauth20.clientDataToInclude

Specifies the OAuth client information to be returned as JSON data. This property is a comma-separated list of the JSON Keys. Valid values are:

contact_type
email_address
contact_person
company_name
company_url
phone_number
other_info

You can specify one or more of these keys for this property.

Note: The `oauth20.clientDataToInclude` property defaults to `contact_type, email_address, contact_person, company_name, company_url, phone_number, other_info`.

Data type: String

Example: `contact_type, email_address, company_name`

oauth20.doNotSendXFrameOptionsHeader

Specifies whether an X-Frame-Options header with value SAMEORIGIN must be returned from the OAuth 2.0 endpoints. When set to true, no X-Frame-Options header is sent.

Note: The `oauth20.doNotSendXFrameOptionsHeader` property defaults to false.

Data type: Boolean

Example: false

oauth20 hashedTokenStorageEnabled

Enables hashed storage when set to true. The Security Access Manager appliance can persist OAuth 2.0 tokens in the clear text form or in the more secure hashed form.

The hashing algorithm set in the runtime.hashAlgorithm property will be used. When verifying hashed tokens, the runtime.verificationHashAlgorithms property will be used. The algorithms listed in the runtime.verificationHashAlgorithms property will be tried in the specified order. This mechanism allows for upgrading of the hashing algorithm while continuing to support old tokens.

Note: The oauth20.hashedTokenStorageEnabled property defaults to false, and the OAuth 2.0 tokens will be stored as-is.

Data type: Boolean

Example: false

oauth20.sessionEndpointEnabled

Enables the ability to return an authenticated session at the point-of-contact when the oauth20.sessionEndpointEnabled property is set to true.

Note: The oauth20.sessionEndpointEnabled property defaults to false.

Data type: Boolean

Example: false

oauth20.tokenCache.cleanupWait

The amount of time, in seconds, to wait before it performs another cleanup of expired tokens in the OAuth 2.0 token cache.

Note: The oauth20.tokenCache.cleanupWait property defaults to 120.

OAuth token clean up can be disabled by setting the cleanupWait value to 0.

Data type: Integer

Example: 120

oauth20.legacyAttributeHandling

Changes how associated attributes function across the API Protection and OpenID Connect solution. This includes:

- OAuthMappingExtUtils.retrieveAllAssociations()
OAuthMappingExtUtils.getAssociation() calls in mapping rules
 - When it is set to **True**, it does not return READONLY or SENSITIVE attributes.
 - When it is set to **False**, it returns READONLY or SENSITIVE attributes.
- The user self care endpoint /mga/sps/mga/user/mgmt/grant/
 - When it is set to **True**, attributes that are both READONLY and SENSITIVE are returned
 - When it is set to **False**, attributes that are both READONLY and SENSITIVE are not returned.

- Attributes which are saved from attribute sources when performing identity enrichment.
 - When it is set to **True**, attributes are saved against the grant as neither READONLY or SENSITIVE.
 - When it is set to **False**, attributes are saved against the grant as READONLY. The post token rule can be used to update this value if necessary.

HTTP client

util.httpClient.defaultTrustStore

Stores the default truststore that HTTPS connections in HTTP client uses.

Note: The util.httpClient.TrustStore property defaults to rt_profile_keys.

Data type: String

Example: rt_profile_keys

util.httpClient.defaultSSLProtocol

Stores the default SSL protocol configuration that HTTPS connections in HTTP client uses.

Note: The util.httpClient.defaultSSLProtocol property defaults to TLS.

Data type: String

Example: TLS

util.httpClient.maxActiveConnections

Specifies the maximum number of HTTP and HTTPS connections, per host, between the appliance runtime and other modules. In a multiple host environment, the runtime might need to establish many HTTP/HTTPS connections at the same time. By specifying this property, you can limit the number of active connections for each host. This setting ensures that each host can obtain their fair share of HTTP/HTTPS connections without being forced to wait for other hosts to release connections.

- Data type: String
- Default: An unlimited number of HTTP/HTTPS connections are permitted

You can specify the maximum number of active connections in one of two ways:

- Specify a maximum number to apply to every host. Syntax:
"*=<count>"
- Specify a maximum number on a per host basis. Syntax:
"<host1>:<port1>=<count>,<host2>:<port2>=<count>,*=<count>"

<host>

The host value can be either an IP address, a hostname or domain name as specified in the Endpoint URL. Specify the host value based on the URL format. For example:

- IP Address: 192.168.102.192
- Hostname or domain name: www.server1.com

<port>=<count>

The communication port on the host. For example, to limit port 80 to only 100 connections, enter 80=100.

***=<count>**

The count limit for servers that are not specified by a *<host>* value in this property. When set to zero (**=0*) there is no limit on the number of HTTP/HTTPS connections that can be created to other servers. When set to an integer greater than zero, the integer specifies the maximum number of HTTP/HTTPS connections that can be created to each of the other servers.

Note: Ensure that *<count>* is specified as a value of type *integer*. Do not use values of type *string* for *<count>*.

Example 1: Specifying a maximum number to apply to every host

For example, your deployment must establish connections to two servers. You want to limit the number of connections to 100 per server. You also want to ensure that when you add additional servers, the number of connections to each additional server is limited to 100. Use the syntax "**=<count>*". For this example:

```
"*=100"
```

Example 2: Specifying maximum numbers on a per host basis

For example, your deployment must establish connections to two servers. You want to limit the number of connections for one server to 100, but allow the other server to have 200 connections. In addition, you do not want to limit the number of connections for any additional servers.

Use the syntax:

```
"<host1>:<port1>=<count>,<host2>:<port2>=<count>,*=<count>"
```

For example, the runtime might need to establish the connections to the following URLs, for an SMS OTP flow and an OIDC flow:

- http://www.server1.com/isam/sms_otp
- https://192.168.102.192/isam/oidc_sts

Example configuration entry:

```
"www.server1.com:80=100,192.168.102.192:443=200,*=0"
```

The example configuration entry specifies:

- The maximum number of HTTP/HTTPS connections that can be created to www.server1.com at a time (on port 80) is 100.
- The maximum number of HTTP/HTTPS connections that can be created to 192.168.102.192 at a time (on port 443) is 200.
- There is no limit on the number of HTTP/HTTPS connections that can be created to other hosts.

Demo**live.demos.enabled**

Enables the mobile demonstration application.

Data type: Boolean

Example: False

live.demos.settings

This setting can be used to pre-populate the settings of the mobile demo. This is a comma separated set of key, value pairs that match what is submitted on the settings form.

Data type: String

Example: lmiHostAndPort=lmi.host.com, lmiAdminId=admin,
lmiAdminPwd=admin, acHostAndPort=127.0.0.1,
websealHostNameAndPort=webseal.host.com

Knowledge questions properties

knowledge.questions.AnswerValidationRegex

Specifies the regular expression used to validate the knowledge question answer value provided during a knowledge question management operation. The assigned value is the list of invalid characters to match against to determine if the supplied value is valid.

Note: At a minimum, this property must include the following characters:
<>:"

Data type: Regex

Example: [\[()\<>,;:\\"\\\"\\"]=]

knowledge.questions.QuestionValidationRegex

Specifies the regular expression used to validate the knowledge question text value provided during a knowledge question management operation. The assigned value is the list of invalid characters to match against to determine if the supplied value is valid.

Note: At a minimum, this property must include the following characters:
<>:"

Data type: Regex

Example: [\[()\<>,;:\\"\\\"\\"]=]

Key encryption and signing service (KESS)

kess.crlEnabled

Checks the certificate revocation list. Checking is done by the key encryption and signature service (KESS) for all functions that use an external certificate, except for the audit syslog. If your configuration does not require CRL checking, you can disable it. For example, if you use if an internal certificate authority (CA), you might want to disable CRL checking. The kess.crlEnabled property defaults to true.

CRL site unavailability scenario

If you have kess.crlEnabled set to true and a CRL site becomes unavailable, you cannot determine the revocation status of the certificate. In this situation, the single sign-on flow will fail.

Confirm a CRL site unavailability issue by looking for the message "FBTKJK056E The CRL site could not be determined." in the runtime trace.log file.

As a temporary workaround, set the CRL checking to false to keep the single sign-on flow running. As soon as the CRL site is working again, set kess.crlEnabled to true so that the single sign-on flow contains the CRL check.

CAUTION:

If you do stop CRL checking as a temporary workaround, be aware that the certificate might have already been revoked by the CA. If this type of certificate is allowed to pass the validation, it creates security issues. Therefore, ensure that you enable CRL checking to avoid potential security issues such as this.

Data type: Boolean

Example: true

kess.crlInterval

The amount of time, in seconds, between successive CRL checks. Using an interval of time between CRL checks reduces the performance impact of doing the checks every time a certificate needs to be validated.

A value less than or equal to zero means that the runtime performs a CRL check every time it wants to use a certificate. The default is 0 seconds.

If kess.crlEnabled is set to false, this value is ignored.

Data type: Integer

Example: 86400

This value means that a CRL check on a certificate is performed once per day.

kess.hostnameValidationDisabled

Determine whether to disable host name verification when establishing an SSL connection. Host name verification is performed when the host name of the server does not match the CN of the certificate of the server.

In a test environment, you might want to disable the validation. In a production environment, you might want to enable validation.

The default value is False.

Data type: Boolean

Example: False

kess.keySelectionCriteria

Specify which key or certificate to use for signing, validating, encrypting, or decrypting various messages. If there are multiple keys or certificates with the same Subject DN as the key or certificate with the specified alias, this setting determines which one to use. Use one of the following selection methods:

only.alias

Select the key or certificate with the specified alias. This is the default.

longest.lifetime

Select the key or certificate with the longest lifetime.

shortest.lifetime

Select the key or certificate with the shortest lifetime.

Data type: String

Example: only.alias

kessjkservice.exclude.inclusive.namespace.prefixes

Specifies a comma-separated list of prefix names. When this is set, the prefixes in the list are not added to the InclusiveNamespaces list that is in the Signature Element.

Data type: String

Example: ds

JSON Web Key**jwks.encryption.keystore**

Defines the name of the encryption keystore to be used by the jwks endpoint for the runtime. These certificates will have their public keys exposed, with the 'use' value 'enc'.

Default value: rt_profile_keys

jwks.signing.keystore

Defines the name of the signing keystore to be used by the jwks endpoint for the runtime. These certificates will have their public keys exposed, with the 'use' value 'sig'.

Default value: rt_profile_keys

Policy information point (PIP)**pip.uncachedAttributes**

Defines a comma-separated list of attributes that are generated by a policy information point (PIP) that you do not want to be cached.

Data type: String list

Example: urn:ibm:security:jdbc:city,
urn:ibm:security:ldap:privilegeUser

Security token service (STS)**sts.ivcred.unauthenticated.user.name**

Set to a special user account for unauthenticated user tokens when using IVCRED STS module in validate mode. The Default value is "".

Data type: String

Example: guest

sts.ivcred.unauthenticated.user.registry.id

In addition to the user name set in sts.ivcred.unauthenticated.user.name, a user's registry id can also be added when using IVCRED STS module in validate mode. The Default value is "".

This parameter is optional.

Data type: String

Example: cn=guest,o=ibm,c=us

sts.ivcred.unauthenticated.user.uuid

In addition to the user name set in

`sts.ivcred.unauthenticated.user.name`, a user's UUID can also be added when using IVCRED STS module in validate mode. The Default value is "".

This parameter is optional.

Data type: String

Example: 81a2a65e-0018-0150-8080-3f83b0f74f4c

`sts.ldapAttributeCache.TTL`

Specifies a time-to-live (TTL) value, in seconds, for the amount of time to keep an LDAP attribute in the cache. Specify 0 to disable.

The default value is 60.

Data type: Integer

Example: 60

Mobile Multi-Factor Authentication (MMFA)

`mmfa.authenticator.cleanupWait`

The amount of time, in seconds, to wait before another cleanup of expired authenticators is performed.

MMFA authenticator clean up can be disabled by setting `cleanupWait` to 0.

The default value is 3600.

Data type: Integer

Example: 3600

`mmfa.transactionArchival.maxCompletedPerUser`

The number of historical transactions in a completed state to keep in the HVDB before archival to the audit log. The oldest transactions will be removed first. A value of -1 will indicate that no archival should be performed.

The default value is 50.

Data type: Integer

Example: 50

`mmfa.transactionArchival.maxPendingPerUser`

The number of transactions to keep in a pending state. Transactions over this number will have their status set to "fail". The oldest transactions will be aborted first. A value of -1 will indicate that no archival should be performed.

The default value is 1.

Data type: Integer

Example: 1

`mmfa.transactionPending.minAgeBeforeAbort`

The minimum number of seconds a transaction is in the pending state before being aborted via a cleanup thread. Due to the cleanup thread interval, the total time a transaction can be in the pending state can be between `minAgeBeforeAbort` and `(minAgeBeforeAbort + cleanupInterval) - 1`

The default value is 300.
Data type: Integer
Example: 300

mmfa.transactionPending.cleanupInterval

The number of seconds between each run of the pending transactions cleanup thread.

The default value is 150.
Data type: Integer
Example: 150

mmfa.transaction.cleanupOnlyOnPrimaryMaster

Indicates whether transaction cleanup should be run on all nodes in a cluster, or only on the primary master. This applies to pending transaction cleanup as well as transaction archival.

The default value is false.
Data type: Boolean
Example: false

mmfa.devicePrompt.skipIfOneDevice

Indicates whether to skip the device selection page in an MMFA flow if the user only has one device or authenticator registered.

The default value is false.
Data type: Boolean
Example: true

WS-Federation

wsfed.idp.rstr.excluded.elements

Specifies a comma-separated list of elements to exclude from the WS-Federation request security token response. Can optionally contain a federation realm and federation partner realm, to indicate the federation or federation partner that uses the property values.

The default value is default=Forwardable,Delegatable,Status,Renewing.

The syntax for specifying federation and federation partner is:

default=<comma_separated_list_of_elements>:<federation_realm>=<comma_separated_list_of_elements>:
<federation_realm>%<partner_realm>=<comma_separated_list_of_elements>

Data type: String

Example:

default=Forwardable,Delegatable,Status,Renewing:fed1-REALM=Forwardable,Delegatable:
fed1-REALM%partner1-REALM=Status

SAML 1.1

saml.use.legacy.clockskew.default

IBM Security Access Manager can add a clock skew of 60 seconds when validating the SAML assertion timestamps. To enable the 60 second clock skew, add the custom property:

saml.use.legacy.clockskew.default = true

Default value = False

- Value type: Boolean
- Example value: True

Note: This custom property is also applicable for SAML 2.0

saml.allowDebugMessages

When specified as true, and a SAML artifact resolution failure occurs, the SystemOut.log and SystemErr.log contains an informational message. In addition, the message contains extra debug information about the request that contained the failed artifact and provides a reason for the event.

Note: This message is only available in English.

Default value: False

- Value type: Boolean
- Example value: SAML.allowDebugMessage = True

saml.allowNoRecipient

Use this custom property if a SAML 1.x service provider needs to accept a samlp:Response that does not contain a Recipient attribute.

Default value: False

saml.assertion.IncludeNSPrefixList.DS

When this custom property is specified as true, ds is included in the Prefix List attribute of the InclusiveNameSpaces in the SAML assertion.

Default value: False

- Value type: Boolean
- Example value: True

Note: This custom property is also applicable for SAML 2.0

saml.allowSpecificInvalidArtifactMessages

When this custom property is specified as true, and a SAML artifact resolution failure occurs, *identity provider sends a SAML Response with specific invalid message to tell the service provider that there is no assertion available. The specific invalid message isFBTSML276E*. If not specified, by default it is false, and the invalid message send back to service provider is **FBTSML013E**.

Default value: False

- Value type: Boolean
- Example value: True

SAML 2.0

saml20.enableSubjectInAuthnRequest

Set to true if the Subject element is required for the SAML 2.0 AuthnRequest. The Subject element is set to the userid of the existing authenticated session. The Default value is false.

Data type: Boolean

Example: true

saml20.idp.acsurlpattern

IBM Security Access Manager uses an exact string comparison between the AssertionConsumerService URL in the AuthnRequest message and the protocol endpoint specified in metadata.

This custom property allows regular expression matching for the AssertionConsumerService URL and the protocol endpoint, so that a dynamic AssertionConsumerService URL that matches the regular expression can be provided in the AuthnRequest.

Data type: String

Format:

```
<FederationId>%<PartnerId>  
%<Binding>=<RegularExpression>,<FederationId2>%<PartnerId2>  
%<Binding2>=<RegularExpression2>
```

Example:

```
https://www.myidp.ibm.com/isam/sps/saml20idp/saml20%https://www.mysp.ibm.com  
/isam/sps/saml20sp/saml20%urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST=https://*.ibm.com/
```

OIDC

oidc.rp.idToken.validationSkew

The number of seconds of skew allowed on the 'nbf' and 'exp' claims of an idToken when it is being processed by an OpenID Connect relying party. For instances where the clocks of two systems are not perfectly synchronized.

Note: This advanced configuration does not apply to legacy OpenID Connect relying parties or Reverse Proxy Relying parties.

Default value: 0

Managing user registries

The appliance runtime profile has a user registry associated. Use the User Registry management page to administer the users and group memberships. The user registry in discussion here is the one used by the runtime applications, not the one used by the management interface.

Procedure

1. From the top menu, select the user interface panel for your licensing level.

- **Secure Access Control > Manage > User Registry**
- **Secure Federation > Manage > User Registry**

A list of all the current users in the registry is displayed. You can filter and reorder the list of users.

2. Perform one or more of the following actions as needed:

Create a user in the registry

- a. Click **New**.
- b. In the Create User window, enter the user name and password for the new user.
- c. Click **OK**.

Delete a user from the registry

- a. Select the user to delete.
- b. Click **Delete**.
- c. In the Delete User window, click **Yes** to confirm the delete operation.

Change the password of a user in the registry

- a. Select the user for which you want to change password.
- b. Click **Set Password**.
- c. In the Set Password window, enter the password in the **New Password** and **Confirm Password** fields.
- d. Click **OK**.

Manage group memberships of a user

- a. Select the user of interest. The group memberships that are associated with this user are displayed under the **Group Membership** section.
- b. You can add the user to a group or delete the user from a group in the registry.

Add the user to a group

- 1) In the **Group Membership** section, click **Add**.
- 2) In the Add to Group window, select the group to add this user to.

Note: Only a single group can be selected.

- 3) Click **OK**.

Remove the user from a group

- 1) In the **Group Membership** section, select the group to remove the user from.
- 2) Click **Delete**.
- 3) In the Remove from Group window, click **Yes** to confirm the removal.

Tuning runtime application parameters and tracing specifications

To manually tune selected runtime application parameters and tracing specifications, use the Runtime Parameters management page.

Procedure

1. From the top menu, select **Secure Access Control > Global Settings > Runtime Parameters** or **Secure Federation > Global Settings > Runtime Parameters**. This page contains three panels: **Runtime Status**, **Runtime Tuning Parameters**, and **Runtime Tracing**.
2. Perform one or more of the following actions to tune your runtime.

Note: Certain changes might require a restart of the runtime before they can take effect.

Disable automatic restart of the runtime

By default, the runtime is automatically restarted after certain changes are made. You can disable this automatic restart function if you prefer manual restarts.

- a. On the **Runtime Tuning Parameters** panel, select **Auto Restart**.
- b. Click **Edit**.
- c. In the Auto Restart window, define the value as **False**.
- d. Click **OK**.

View the status of the runtime and restart the runtime

- a. Select the **Runtime Status** panel. The status of local and clustered runtimes are displayed.
 - Under **Local Runtime Status**, you can view the runtime operational status, when it was last started, and whether a restart is outstanding. If the value of the **Restart Required** field is **True**, it means that the runtime must be restarted for some changes to take effect.
 - Under **Clustered Runtime Status**, all nodes in the cluster are listed.
 - The **Master** column indicates whether a node is the cluster master.
 - The **Runtime Status** column indicates whether a node is running or stopped.
 - The **Changes Active** column indicates whether changes made to the cluster configuration are active on this node. Having a green indicator in this column means that all changes made are already active. Having a yellow indicator in this column means that this node must be restarted before some changes can take effect.
- b. Depending on which runtime you want to restart, click **Restart Local Runtime** or **Restart All Clustered Runtimes**.

Modify the maximum or initial heap size

These parameters indicate the maximum and initial heap size in megabytes for the runtime Java virtual machine.

- a. On the **Runtime Tuning Parameters** panel, select **Max Heap Size** or **Initial Heap Size**.
- b. Click **Edit**.
- c. In the Max Heap Size or Initial Heap Size window, enter the heap size value as needed.
- d. Click **OK**.

Modify the minimum or maximum threads

These parameters indicate the minimum number of core threads that the runtime server starts with and the maximum number of threads that can be associated with the runtime server.

If the minimum value is not set or is set as -1, a default value is calculated based on the number of hardware threads on the system.

If the maximum value is not set or is set as 0 or less, a default value of unbounded is used.

The minimum **cannot** be set to a value larger than the maximum.

- a. On the **Runtime Tuning Parameters** panel, select **Min Threads** or **Max Threads**.
- b. Click **Edit**.
- c. In the Min Threads or Max Threads window, enter the required value.
- d. Click **OK**.

Modify whether to suppress sensitive trace

Enabling this parameter prevents sensitive information from being exposed in log and trace files. Examples of such sensitive information include bytes received over a network connection.

- a. On the **Runtime Tuning Parameters** panel, select **Suppress Sensitive Trace**.
- b. Click **Edit**.
- c. In the Suppress Sensitive Trace window, select or clear the check box as needed.
- d. Click **OK**.

Modify console log level

Console log level controls the granularity of messages that go to the console.log file.

- a. On the **Runtime Tuning Parameters** panel, select **Console Log Level**.
- b. Click **Edit**.
- c. In the Console Log Level window, select the new value from the list.
- d. Click **OK**.

Set whether to accept client certificates

This parameter controls whether the server accepts client certificates as a form of authentication.

- a. On the **Runtime Tuning Parameters** panel, select **Accept Client Certificates**.
- b. Click **Edit**.
- c. In the Accept Client Certificates window, select or clear the check box as needed.
- d. Click **OK**.

Set session invalidation timeout

This parameter defines the amount of time a session can remain unused before it is no longer valid.

Note: The default setting is **Unset**. When this setting is used, the session invalidation timeout is 1800 seconds.

- a. On the **Runtime Tuning Parameters** panel, select **Session Invalidation Timeout**.
- b. Click **Edit**.
- c. In the Session Invalidation Timeout window, define the value in seconds.
- d. Click **OK**.

Set session reaper poll interval

This parameter defines the wake-up interval in seconds for the process that removes invalid sessions. The minimum value is 30 seconds.

The default setting is **Unset**. When this setting is used, or if a value less than the minimum is entered, an appropriate value is automatically determined and used. This value overrides the default installation value, which is 30 - 360 seconds, based on the session invalidation

timeout value. Because the default session invalidation timeout is 1800 seconds, the reaper interval is usually between 120 and 180 seconds.

- a. On the **Runtime Tuning Parameters** panel, select **Session Reaper Poll Interval**.
- b. Click **Edit**.
- c. In the Session Reaper Poll Interval window, define the value in seconds.
- d. Click **OK**.

Set the keystore that is used by the runtime server

This parameter defines the key database that contains the runtime server's private key.

- a. On the **Runtime Tuning Parameters** panel, select **Keystore**.
- b. Click **Edit**.
- c. In the Keystore window, select the key database from the list.
- d. Click **OK**.

Set the truststore that is used by the runtime server

This parameter defines the key database that contains keys that are trusted by the runtime server

- a. On the **Runtime Tuning Parameters** panel, select **Truststore**.
- b. Click **Edit**.
- c. In the Truststore window, select the key database from the list.
- d. Click **OK**.

Configure an outbound HTTP proxy

You must specify values for the properties for the HTTP proxy. You might also need to import the root CA certificate from the proxy. See the instructions that follow.

Table 23. HTTP proxy properties

Name	Sample Value	Description
http.proxyHost	http.proxy.ibm.com	The hostname or IP address of the HTTP proxy
http.proxyPort	3128	The port of the HTTP proxy
https.proxyHost	https.proxy.ibm.com	The hostname or IP address of the HTTPS proxy
https.proxyPort	3128	The port of the HTTPS proxy

- a. For each property in the table above:
 - 1) On the **Runtime Tuning Parameters** panel, select the property.
 - 2) Click **Edit**.
 - 3) In the property window, enter the value. See the sample values in the table.
 - 4) Click **OK**.
- b. When all properties are set, follow the prompt to deploy the pending changes.

Certain functions, such as the OpenID connect single sign-on flow, require the root CA certificate of the outbound HTTP proxy to be imported to the Security Access Manager runtime keystore.

Complete the following steps:

- a. Go to your HTTP Proxy application and obtain the necessary certificate for exchange. The exact steps to take are specific to the proxy application. Place the certificate on the local file system where it can be accessed by the appliance.
- b. On the Security Access Manager system, log in to the local management interface and select **Manage System Settings > Secure Settings > SSL Certificates**
- c. Select the `rt_profile_keys` keystore.
- d. Select **Manage > Edit SSL Certificate Database**.
- e. Select **Manage > Import**.
- f. On the Signer Certificate panel, browse to locate the **Certificate File**. Enter a **Certificate Label**. Click **Import**.
- g. Deploy the changes.

Delete the value of a parameter

Use this button to delete the existing value of a parameter.

- a. Select the parameter to reset the value for.
- b. Click **Delete**. The value of the parameter is then changed to Unset.

Manage the application interface on which the runtime listens

- a. On the **Runtime Tuning Parameters** panel, under **Runtime Listening Interfaces**, you can add, edit, or delete a listening interface.

To add a listening interface

- 1) Click **Add**.
- 2) In the Runtime Listening Interfaces window, select the listening interface from the list.
- 3) Specify the listening port.
- 4) Select the **SSL** check box if security is required.
- 5) Click **OK**.

To modify a listening interface

- 1) Select the listening interface to edit.
- 2) Click **Edit**.
- 3) In the Runtime Listening Interfaces window, edit the values as needed.
- 4) Click **OK** to save the changes.

To delete a listening interface

- 1) Select the listening interface to delete.
- 2) Select **Delete**.
- 3) Confirm the deletion.

Manage tracing specification

- a. Select the **Runtime Tracing** link from the top of this page. You can also access this panel from the top menu by selecting **Monitor Analysis and Diagnostics > Logs > Runtime Tracing**.
- b. Use one of the following ways to edit the trace level of a component.
 - Select the component name from the **Component** list. Select the ideal trace level for this component from the **Trace Level** list.

Then, click **Add**. Repeat this process to modify trace levels for other components if needed. To clear all of the tracing levels, click **Clear**.

To log all events, select ALL as the trace level.

Note: This setting increases the amount of data in logs, so use this level when necessary.

`com.tivoli.am.fim.authsvc.*`
`com.tivoli.am.fim.trustserver.sts.modules.*`

Table 24. Valid trace levels. The following table contains the valid trace levels.

Level	Significance
ALL	All events are logged. If you create custom levels, ALL includes those levels and can provide a more detailed trace than FINEST.
FINEST	Detailed trace information that includes all of the details that are necessary to debug problems.
FINER	Detailed trace information.
FINE	General trace information that includes methods entry, exit, and return values.
DETAIL	General information that details sub task progress.
CONFIG	Configuration change or status.
INFO	General information that outlines the overall task progress.
AUDIT	Significant event that affects the server state or resources.
WARNING	Potential error or impending error. This level can also indicate a progressive failure. For example: the potential leaking of resources
SEVERE	The task cannot continue, but component, application, and server can still function. This level can also indicate an impending unrecoverable error.
FATAL	The task cannot continue, and component, application, and server cannot function.
OFF	Logging is turned off.

- Enter the name and value of the trace component in the **Trace Specification** field. To modify multiple components, separate two strings with a colon (:). Here is an example.

`com.x.y.*=WARNING:com.a.b.*=WARNING:com.ibm.isam.*=INFO`

c. Click **Save**.

3. When you make changes, the appliance displays a message that there are undeployed changes. If you have finished making changes, deploy them.

Template files

Template files are HTML pages that are presented to your users during the authentication process. The pages prompt users for authentication information, such as user names and passwords, or present information to users, such as one-time passwords, status, or errors.

You can customize any of the HTML pages by exporting, modifying, and importing its corresponding template file. Each template file uses one or more specific macros.

Managing template files

Use the local management interface to manage files and directories in the template files.

About this task

You can run the following tasks on the template files:

- **New**- Use this option if you want to create a new file or directory.
- **Edit**- Use this option if you want to view or modify the template file.
- **Import**- Use this option if you to import a file to the template files root.
- **Export**- Use this option if you want to export a file from the template files root.
- **Rename**- Use this option if you want to rename a file or directory from the template files root.
- **Delete**- Use this option if you want to delete a file or directory from the template files root.
- **Import Zip**- Use this option if you want to import the template files from a compressed file.
- **Export Zip**- Use this option if you want to export the template files as a compressed file.

Note: When you use this option to export template files as a compressed file, you cannot export an individual folder. The root directory, including all the subdirectories, is exported. To access the contents of a specific directory, export the entire template directory, and then view the directory from the extracted compressed file on your local workstation.

Procedure

1. Select **Secure Access Control > Global Settings > Template Files**
2. Work with all the management files and directories.

Create a file or directory in the template files root

- a. Select the directory of interest.
- b. Select **New**.
- c. Select **File** or **Directory**.
- d. Enter the name of the file or directory.
- e. Click **Save**.

View or update the contents of a file in the template files root

- a. Select the file of interest.
- b. Select **Edit**. You can then view the contents of the file.
- c. Edit the contents of the file.

- d. Click **Save**.

Export a file from the template files root

- a. Select the file of interest.
- b. Select **Manage > Export**.
- c. Confirm the save operation when your browser displays a confirmation window.

Rename file from the template files root

- a. Select the file or directory of interest.
- b. Select **Manage > Rename**.
- c. Enter the new resource name.
- d. Click **Save**.

Delete file from the template files root

- a. Select the file or directory of interest.
- b. Select **Manage > Delete**.
- c. Click **Yes**.

Import a file to the template files root

- Select a file.
 - a. Select **Manage > Import**.
 - b. Click **Browse**.
 - c. Browse to the file that you want to import the contents from.
 - d. Click **Open**.
 - e. Click **Import**.
- Select a folder.
 - a. Select **Manage > Import**.
 - b. Click **Browse**.
 - c. Browse to the file that you want to import to the selected folder.
 - d. Click **Open**.
 - e. Click **Import**.

Export the template file as a compressed file

- a. Select **Manage > Export Zip**.
- b. Confirm the save operation when your browser displays a confirmation window.

Import the template files as a compressed file

Make sure that the files in the compressed file are in the same directory structure as the files in the root directory or appliance.

For example, if a file in the compressed file is in the /C directory of the appliance, the compressed file must contain the C folder and the file that you want to import. When you import a compressed file that contains:

- A file that exists in the appliance
The file is replaced in the appliance.
- A file or directory that does not exist in the appliance
The file or directory is created in the appliance. You can put these new files and directories in an existing non-root directory or add a new directory in the root.

Note: You cannot delete a top level directory after you create it.

- a. Select **Manage > Import Zip**.
 - b. Click **Browse**.
 - c. Browse to the file you want to import.
 - d. Click **Open**.
 - e. Click **Import**.
3. When you edit or import template files, the appliance displays a message that there are undeployed changes. If you finish the changes, deploy them.
For more information, see Deploying pending changes.

Customizing the consent page

The consent page of an OpenID Connect Provider Federation can be changed with the Template Files page in the local management interface.

About this task

All OpenID Connect Provider (OP) federations can have their own unique consent pages. Follow these steps to set a consent page to be used by a specific federation.

Procedure

1. Log in to the local management console.
2. Select **Secure Federation > Global Settings > Template Files**.
3. Expand the **C** locale.
4. Highlight the **oidc** folder.
5. Click **New** and select **Directory**.
6. Enter the **Federation Name** of the OpenID Connect Provider Federation to use the custom consent page.
7. Click **Save**.
8. Highlight the new directory.
9. Click **New** and select **File**.
10. Enter `consent.html` as the file name.
11. Populate the file contents.
12. Click **Save**.
13. Deploy the pending changes.

Note: The deploy operation triggers a runtime restart.

Template page scripting

You can use JavaScript to add server-side scripting for Advanced Access Control and Federation template pages. You can use JavaScript functions, closures, objects, and delegations.

Usage

You can customize template files or pages on the server. For example, you can customize an error message that is displayed by the runtime server.

The template files menu is located under both the Secure Federation and Secure Access Control menus.

To edit a Federation template file, go **Secure Federation > Template Files**, select the specific template file, and click **Edit**.

To edit a Secure Access Control template file, go to **Secure Access Control > Template Files**, select the specific template file, and click **Edit**.

The JavaScript engine supports the following syntax:

- Insert JavaScript code between `<%` and `%>`.
- Embed JavaScript expressions between `<%=` and `%>`.

Example tasks

- Access whitelisted Java classes. For example,
`var javaStr = new java.lang.String("Hello")`
- Access all the macro variables through `templateContext`. The standard object to access a Java object is `templateContext`. For example,
`templateContext.macros["@TIMESTAMP@"]`
- Use the `document.write` function to write content to the output stream. For example,
`templateContext.response.body.write("Hello")`

Examples

Table 25. Example JavaScript

Template HTML	Output
<code><% var contents = {product:"ISAM",department:"Lab",country:"SG",region:"Asia"}; templateContext.response.body.write(contents.product); %></code>	ISAM
<code><% var date = templateContext.macros["@TIMESTAMP@"].substring(0, 10); templateContext.response.body.write(date); %></code>	2017-01-25

The following code example shows how to use repeatable macros. The following example shows an OAuth consent page.

```
<%  
var test = templateContext.macros["oauthTokenScopeNewApprovalRepeatable"];  
n = test.length;  
for (i=0; i<n; i++){  
  var scope = test[i]["@OAUTH_TOKEN_SCOPE_REPEAT@"];  
  if (scope == "contacts"){  
    label = "Do you grant permission to the client to access your phone book";  
  }  
  else if (scope == "photos"){  
    label = "Do you grant permission to the client to access your photos";  
  }  
  else if (scope == "messages"){  
    label = "Do you grant permission to the client to access your WhatsApp messages";  
  }  
  else{  
    label = "Do you grant permission to the client to access your "+scope;  
  }  
}  
%>
```


Setting an HTTP response header

You can use `templateContext.response.setHeader(HeaderName, HeaderValue)` to set an HTTP response header.

For example, you can set the Content-Type to support both a mobile-based browser and a traditional browser. A mobile-based browser might expect JSON format while a traditional browser expects forms-based HTML.

```
<%
templateContext.response.setHeader("Content-Type","application/json");
var myObj = { "name":"John", "age":31, "city":"New York" };
templateContext.response.body.write(JSON.stringify(myObj));
%>
```

To set an HTTP header that uses forms-based HTML:

```
templateContext.response.setHeader("Content-Type","text/html");
```

Setting an HTTP status code

You can use `templateContext.response.setStatus(Code)` to set an HTTP response status code.

For example, if you want to set the status to 400 (standard code for a bad request):

```
templateContext.response.setStatus(400);
```

Setting a Redirect URL

You can use `templateContext.response.sendRedirect(URL)` to redirect the HTTP response to a different URL.

For example, when you configure single logout, you can redirect the response to a specific target page, based on the federation name. An example scenario is a deployment that has one SAML 2.0 federation with two partner federations. The partner federations are named `saml20app2` and `saml20sp`. The `saml20app2` federation uses an application that is named `jkebank`. The `saml20sp` federation uses an application that is named `jkeschool`. The page to display on logout is determined by the federation name.

```
var fedName = templateContext.macros[@FEDERATION_NAME@];
if (fedName == "saml20app2")
{
    templateContext.response.sendRedirect("http://jkebank:1337");
}
else if
{
    (fedName == "saml20sp")
    {
        templateContext.response.sendRedirect("http://jkeschool:1400");
    }
}
```

Obtaining a list of macros that are available for a template page

In some scenarios, you might want to write JavaScript based on configuration values in your deployment. For example, you might implement one action based on the authentication type, such as if the OTP type is TOTP. Another example is you might implement an action if the Federation name of the single sign-on partner matches a certain value.

Information such as the OTP type and partner name can be retrieved only through the template page macros. To use such information, you need to know which macros are used by the page. The JavaScript engine support provides a utility that can print the available macros for a page.

Use the following syntax to obtain a list of the available macros.

```
<% templateContext.response.body.write(JSON.stringify(templateContext.macros)); %>
```

For example, the following sample code prints the macros from a template page that ran a single sign-on flow with a partner that does not exist.

```
{
  "@PAGE_IDENTIFIER": "/saml20/invalid_init_msg.html",
  "@TARGET@": "https://www.mysp.ibm.com/isam/mobile-demo/diag",
  "@PARTNER_ENTITY_ID@": "",
  "@ERROR_MESSAGE@": "FBTSMLO02E The value https://saml.partner.com for attribute PartnerId is not valid.",
  "@FEDERATION_NAME@": "saml20idp",
  "@FEDERATION_ENTITY_ID@": "https://www.myidp.ibm.com/isam/sps/saml20idp/saml20",
  "@REQ_ADDR@": "/sps/saml20idp/saml20/logininitial",
  "@ERROR_CODE@": "FBTSMLO02E",
  "@EXCEPTION_STACK@": "",
  "@PARTNER_NAME@": "",
  "@TIMESTAMP@": "2017-06-22T03:34:39Z",
  "@SAMLSTATUS@": "<fim:FIMStatusCollection xmlns:fim=\"urn:ibm:names:ITFIM:saml\"
    xmlns:samlp=\"urn:oasis:names:tc:SAML:2.0:protocol\"><fim:FIMStatusCollectionEntry>
      <samlp:Status><samlp:StatusCode Value=\"urn:oasis:names:tc:SAML:2.0:status:Responder\"></samlp:StatusCode>
      <samlp:StatusDetail><fim:FIMStatusDetail MessageID=\"invalid_attribute_value\">
        <fim:SubstitutionString>https://saml.salesforce.com</fim:SubstitutionString>
        <fim:SubstitutionString>PartnerId</fim:SubstitutionString></fim:FIMStatusDetail>
      </samlp:StatusDetail></samlp:Status></fim:FIMStatusCollectionEntry></fim:FIMStatusCollection>",
  "@EXCEPTION_MSG@": ""
}
```

The format is JSON { "name1":"value1","name2":"value2"}

Limitations

- JavaScript validation is done only when a template file is edited (imported) or created. A template file that is imported as a part of an Import compressed file is not validated.
- You must restart the runtime manually to activate changes to OpenID Connect template files. In the administrative interface, click **Secure Federation -> Runtime Tuning -> Restart Runtime**.
- When you access a variable, do not end the variable name with a semicolon. For example, in the following JavaScript, do not end `<%=example%>` with a semicolon `<%=example%;%>`.

```
<%var example = "Hello World"; %>
<%=example%>
```

The correct syntax is `<%=example%>`. Do not use the incorrect syntax `<%=example%;%>`.

Template files reference

Template files are HTML pages that are presented to your users during the authentication process. The pages prompt users for authentication information, such as user names and passwords, or present information to users, such as one-time passwords, status, or errors.

Consent to register device template files

These files support consent to registering a device.

Consent to register device template files

These files support consent to registering a device.

Table 26. Default template files in the *ac/* directory

Page name	File name and macros	Description
Attribute Collection JavaScript	ac/info.js	Detects the location of the device from which the requests are made. Collects the web browser attributes and sends them to the server for storing in the database. When the user logs out or when the current session times out, the script deletes the attributes from the database.
Dynamics Attributes JavaScript	ac/javascript_rules/dynamic_attributes.js	Runs after each request is processed by risk engine. Use it to capture attributes that do not get captured automatically. Captured attributes are stored either in the session storage or the behavior storage area of the risk-based component, or both. The risk profile configuration dictates where the attributes are stored.

User self-care template files

These files support user self-care tasks.

User self-care template files

These files support user self-care tasks.

Table 27. Default template files in the *mga/* directory

Page name	File name and macros	Description
Common User Profile Management JavaScript	mga/user/mgmt/common.js	Used by one-time password pages and by device management pages. Contains functions and properties that are used for making calls to the user self-care REST services.

Table 27. Default template files in the mga/ directory (continued)

Page name	File name and macros	Description
Device Attributes	mga/user/mgmt/device/device_attributes.html	Enables or disable devices. Renames or removes device. Displays all of the attributes for a device. For more information, see Managing your registered devices.
Device Attributes JavaScript	mga/user/mgmt/device/device_attributes.js	Processes values that are entered in the device_attributes.html template
Device Selection	mga/user/mgmt/device/device_selection.html	Displays device name, status (enabled or disabled), and time of last activity. For more information, see Managing your registered devices.
Device Selection JavaScript	mga/user/mgmt/device/device_selection.js	Processes data to display in the device_selections.html template
Authorization Grant	mga/user/mgmt/device/grant_attributes.html	Enables or disables an OAuth 2.0 authorization grant. Removes an OAuth 2.0 authorization grant. Displays the OAuth 2.0 tokens and attributes of an authorization grant. For more information, see “Managing OAuth 2.0 authorization grants” on page 170.
Authorization Grants JavaScript	mga/user/mgmt/device/grant-attributes.js	Processes data to display in the grant_attributes.html template.
HMAC OTP Shared Key	mga/user/mgmt/otp/otp.html	Resets TOTP and HOTP Secret Key. For more information, see Managing OTP secret keys.
HMAC OTP Shared Key JavaScript	mga/user/mgmt/otp/otp.js	Resets TOTP and HOTP Secret Key.
Knowledge Questions management	mga/user/mgmt/questions/user_questions.html Macros: • @USERNAME @ • @MAX_STORED_QUESTIONS@	Displayed for the user to manage their knowledge questions. The user can select pre-configured questions or write their own questions.

Table 27. Default template files in the *mga/* directory (continued)

Page name	File name and macros	Description
Knowledge Questions JavaScript functions	<i>mga /user/mgmt/questions/ user_questions.js</i>	Consists of the JavaScript functions that: • Display the knowledge questions. • Allow the user to manage their knowledge questions.

Authentication process

These files support the authentication process

Authentication process template files

These files support the authentication process. For more information, see Chapter 9, “Authentication,” on page 31.

Table 28. Default template files in the *authsvc/* directory

Page name	File name and macros	Description
Server Error	<i>authsvc/server_error.html</i> Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays general server errors.
User Error	<i>authsvc/user_error.html</i> Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during authentication policy execution that are caused by user input.

Authentication mechanisms

These files support the authentication mechanisms.

Authentication mechanisms

These files support the authentication mechanisms. For more information, see Chapter 9, “Authentication,” on page 31.

Table 29. Default template files in the *otp/* directory

Page name	File name and macros	Description and link to file contents
Change PIN required	<i>otp/change_pin.html</i> Macros: • @ERROR_MESSAGE@ • @MAPPING_RULE_DATA@ • @DISPLAY_RESELECT_BUTTON@	Enables the user to enter a new PIN.

Table 29. Default template files in the otp/ directory (continued)

Page name	File name and macros	Description and link to file contents
OTP Email Delivery Message	otp/delivery/ email_message.xml	Used by EmailOTPDelivery as the content of the email that it sends to the user. The template file must be a compliant XML file. The content can be plain text or HTML. Following is an example using HTML in the email template: <pre><?xml version="1.0" encoding="UTF-8"?> <root> <Subject> <Value>One-time Password</Value> </Subject> <Message> <Value><![CDATA[<html> <body>
This is your HTML email one-time password. <p>Thank you,
 The Example Co.</p> </body> </html>]]> </Value> </Message> </root></pre> For information on HTML formatting of email messages, see “HTML format for OTP email messages” on page 62.
OTP SMS Delivery Message	otp/delivery/ sms_message.xml	Used by SMSOTPDelivery as the content of the SMS that it sends to the user. The template file must be a compliant XML file.
One-Time Password Delivery Selection	otp/ delivery_selection.html Macros: • @OTP_METHOD_CHECKED@ • @OTP_METHOD_LABEL@	Displays the list of methods for generating, delivering, and verifying the one-time password.
OTP General Error	otp/errors/allerror.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays general errors that happen during the one-time password flow.

Table 29. Default template files in the otp/ directory (continued)

Page name	File name and macros	Description and link to file contents
OTP Validation Error	otp/errors/ error_could_not_validate_otp.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays errors during the validation of the one-time password that the user submits.
OTP Generation Error	otp/errors/ error_generating_otp.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays errors during the generation of a one-time password.
OTP Methods Retrieval Error	otp/errors/ error_get_delivery_options.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays errors during the retrieval of the list of methods for delivering one-time password to the user.
OTP Delivery Error	otp/errors/ error_otp_delivery.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays errors during the delivery of a one-time password to the user.
OTP STS Invocation Error	otp/errors/ error_sts_invoke_failed.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays errors during the invocation of the Security Token Service.
One-Time Password Login	otp/login.html Macros: • @ERROR_MESSAGE@ • @MAPPING_RULE_DATA@ • @DISPLAY_RESELECT_BUTTON@	Displays the form where the user can enter the one-time password.

Table 29. Default template files in the otp/ directory (continued)

Page name	File name and macros	Description and link to file contents
Enter Next OTP Form	otp/next_otp.html Macros: • @ERROR_MESSAGE@ • @MAPPING_RULE_DATA@ • @DISPLAY_RESELECT_BUTTON@	Enables the user to enter the next one time password.

Table 30. Default template files in the authsvc/authenticator/password/ directory

Page name	File name and macros	Description
Change Password	authsvc/authenticator/password/change_password.html Macros: • @USERNAME@ • @ERROR_MESSAGE@	Enables the users to change their registry password.
Username and Password Error	authsvc/authenticator/password/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the user name and password authentication or when the users modify their password.
Username and Password Login	authsvc/authenticator/password/login.html	Displays the form where the users can enter their user name and password to log in.

Table 31. Default template files in the authsvc/authenticator/http_redirect/ directory

Page name	File name and macros	Description
HTTP Redirect Authentication Error	authsvc/authenticator/http_redirect/allerror.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays general errors during for HTTP redirect authentication mechanism.

Table 31. Default template files in the authsvc/authenticator/http_redirect/ directory (continued)

Page name	File name and macros	Description
HTTP Redirect Authentication Failed	authsvc/authenticator/ http_redirect/ error_authenticate.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the HTTP redirect authentication flow.

Table 32. Default template files in the authsvc/authenticator/macotp/ directory

Page name	File name and macros	Description
MAC One-Time Password Delivery Selection	authsvc/authenticator/ macotp/ delivery_selection.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays the list of methods for generating, delivering, and verifying the one-time password.
MAC OTP One-Time Password Error	authsvc/authenticator/ macotp/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the MAC one-time password authentication.
MAC One-Time Password Login	authsvc/authenticator/ macotp/login.html Macros: • @OTP_HINT@ • @OTP_LOGIN_DISABLED@	Displays the form where the user can enter the MAC one-time password

Table 33. Default template files in the authsvc/authenticator/rsa/ directory

Page name	File name and macros	Description
RSA One-Time Password Error	authsvc/authenticator/rsa/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the RSA one-time password authentication.
RSA One-Time Password Login	authsvc/authenticator/rsa/code.html Macro: @ERROR_MESSAGE@	Displays the form where the users can enter the RSA one-time password to log in.
RSA One-Time Password Login (New PIN)	authsvc/authenticator/rsa/new_pin.html Macro: @ERROR_MESSAGE@	Enables users to enter a new RSA pin.
RSA One-Time Password Login (Next OTP)	authsvc/authenticator/rsa/next_code.html Macro: @ERROR_MESSAGE@	Enables users to enter the next RSA one-time password.

Table 34. Default template files in the authsvc/authenticator/totp/ directory

Page name	File name and macros	Description
TOTP One-Time Password Error	authsvc/authenticator/totp/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the TOTP one-time password authentication.
TOTP One-Time Password Login	authsvc/authenticator/totp/login.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays the form where the users can enter the TOTP password to log in.

Table 35. Default template files in the authsvc/authenticator/hotp/ directory

Page name	File name and macros	Description
HOTP One-Time Password Error	authsvc/authenticator/hotp/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the HOTP one-time password authentication.
HOTP One-Time Password Login	authsvc/authenticator/hotp/login.html Macros: @ERROR_MESSAGE@	Displays the form where the users can enter the HOTP password to log in.

Table 36. Default template files in the authsvc/authenticator/consent_register_device/ directory

Page name	File name and macros	Description
Consent to Device Registration Error	authsvc/authenticator/consent_register_device/error.html Macros: • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during the consent to device registration flow.
Consent page	authsvc/authenticator/consent_register_device/consent-form.html Macro: @ERROR_MESSAGE@	Prompts the user to provide consent for registering a device.

Table 37. Default template files in the authsvc/authenticator/eula/ directory

Page name	File name and macros	Description
End-User License Agreement license file display	authsvc/authenticator/eula/license.txt	Contains the license agreement to display to the user. The template does not use replacement macros. Note: You can add more license files to the template tree. Specify the metadata in the End-User License Agreement for the following purposes: • Auditing • Forensic The End-User License Agreement authentication mechanism removes the metadata before it displays the license agreement to the user. The metadata must be on the first line of the license agreement. For example: Metadata: Version: 1.0 When the user accepts the license agreement or declines the license agreement, the mechanism audits: • The user action. • The license file name. • The corresponding metadata.
End-User License Agreement license agreement display	authsvc/authenticator/eula/eula.html Macros: • @USERNAME@ • @LICENSE@	Displays the page where the user views the license and accepts the license agreement.

Identifier

Table 37. Default template files in the authsvc/authenticator/eula/ directory (continued)

Page name	File name and macros	Description
End-User License Agreement license agreement decline	authsvc/authenticator/ eula/ error_license_declined.html Macros: • @USERNAME@ • @ERROR_MESSAGE@ • @REQ_ADDR@ • @TIMESTAMP@ • @ERROR_MESSAGE@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@ • @LICENSE_FILE@ • @LICENSE_METADATA@	Displays the page where the user declines the license agreement.

Table 38. Default template files in the authsvc/authenticator/knowledge_questions/ directory

Page name	File name and macros	Description
Knowledge Questions authentication mechanism knowledge question form	authsvc/authenticator/ knowledge_questions/ login.html Macros: • @ QUESTION_TEXT @ • @ QUESTION_INDEX @ • @QUESTION_UNIQUE_ID@ • @QUESTION_COUNT@ • @ERROR_MESSAGE@ • @NUM_REQUIRED_ANSWERS@	Displays the form where the user enters the answers to the required knowledge questions.
Knowledge Questions authentication mechanism knowledge question authentication errors	authsvc/authenticator/ knowledge_questions/ error.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @ERROR_MESSAGE@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors during knowledge-question authentication.

Table 38. Default template files in the authsvc/authenticator/knowledge_questions/ directory (continued)

Page name	File name and macros	Description
Knowledge Questions authentication mechanism missing knowledge questions with grace period	authsvc/authenticator/knowledge_questions/not_enough_questions_found_complete.html Macros: • @USERNAME@ • @NUM_REQUIRED_ANSWERS@ • @NUM_REGISTERED_QUESTIONS@ • @GRACE_PERIOD_AUTH_COUNT@ • @MAX_GRACE_PERIOD_AUTH_COUNT@	Displayed when the user did not register the required number of knowledge questions and answers that are required for successful authentication. The following conditions must also be true: • The administrator configured the environment to allow grace-period authentication. • The user did not reach the limit of grace-period logins.
Knowledge Questions authentication mechanism missing knowledge questions without grace period	authsvc/authenticator/knowledge_questions/not_enough_questions_found_enough.html Macros: • @USERNAME@ • @NUM_REQUIRED_ANSWERS@ • @NUM_REGISTERED_QUESTIONS@ • @REQ_ADDR@ • @TIMESTAMP@	Displayed when the user did not register the required number of knowledge questions and answers that are required for successful authentication. One of the following conditions must also be true: • The administrator did not configure the environment to allow grace-period authentication. • The user reached the limit of grace-period logins.

Authentication error template files

These files display errors that occur during authentication.

Authentication error template files

These files display errors that occur during authentication.

Table 39. Default files in the proper/ directory

Page name	File name and macros	Description
Access Denied	proper/errors/access_denied.html Macros: • @REQ_ADDR@ • @TIMESTAMP@	Displays a message that the user cannot access the requested resource.

Table 39. Default files in the proper/ directory (continued)

Page name	File name and macros	Description
General Error	proper/errors/ allerror.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_STACK@	Displays general errors that are not displayed in other template files.
Missing Component Error	proper/errors/ missingcomponent.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @DETAIL@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays an error that the component required to process the request was not correctly configured or was not initialized.
Authentication Required	proper/errors/ need_authentication.html Macros: • @REQ_ADDR@ • @TIMESTAMP@	Displays an error that authentication is required to access the requested resource.
Protocol Determination Error	proper/errors/ noprotdet.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays an error that the access request is to an unknown address. The error might occur because no configured endpoint or protocol exists that is mapped to this endpoint.
Protocol Runtime Error	proper/errors/ protocol_error.html Macros: • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@	Displays errors that an error occurred fulfilling a request to a specified address, and the error was caused by an unexpected error on the protocol module.

OAuth template files

These files support OAuth.

OAuth template files

These files support OAuth. For more information, see Chapter 10, “OAuth 2.0 and OIDC support,” on page 105.

Table 40. Default files in the *oauth20/* directory

Page name	File name and macros	Description
OAuth 2.0 Trusted Clients Manager	oauth20/clients_manager.html Macros: • @USERNAME@ • @OAUTH_CLIENT_COMPANY_NAME@ • @PERMITTED_SCOPES@ • @OAUTH_CUSTOM_MACRO@	Used by resource owners to show and manage trusted clients information.
OAuth 2.0 - Consent to Authorize	oauth20/user_consent.html Macros: • @USERNAME@ • @OAUTH_CLIENT_COMPANY_NAME@ • @PERMITTED_SCOPES@ • @OAUTH_CUSTOM_MACRO@	Used by the authorization server to determine and store user consent information about which OAuth clients are authorized to access the protected resource. The page also lists of scopes that the OAuth client requests. These lists are shown in the consent page and can be of indeterminate length. The template supports multiple copies of stanzas that are repeated once for each scope in the lists.
OAuth 2.0 - Error	oauth20/user_error.html Macros: • @OAUTH_CLIENT_COMPANY_NAME@ • @CLIENT_TYPE@ • @CLIENT_ID@ • @REDIRECT_URI@ • @STATE@ • @RESPONSE_TYPE@ • @USERNAME@ • @OAUTH_TOKEN_SCOPE_REPEAT@ • @OAUTH_OTHER_PARAM_REPEAT@ • @OAUTH_OTHER_PARAM_VALUE_REPEAT@	Shows detailed text information when an error occurs in an OAuth 2.0 flow.

Table 40. Default files in the `oauth20/` directory (continued)

Page name	File name and macros	Description
OAuth - Response	oauth20/user_response.html Macros: • <code>@OAUTH_CODE@</code>	Displays the authorization code of an OAuth client that did not specify a client redirection URI upon registration. When the OAuth client does not specify a client redirection URI or cannot receive redirects, the authorization server does not know where to send the resource owner after authorization. As a result, the OAuth client does not receive the authorization code that is required to exchange for an access token or refresh token. The page includes several codes: • The authorization code that the resource owner can provide to the trusted OAuth client. • The authorization code as machine-readable Quick Response (QR) code. Note: The encoder that creates the QR code follows the ISO/IEC 18004:2006 specification. Scanners that support this specification can read the generated QR code.

Customizing SAML 2.0 pages

Access Manager generates files that are displayed in response to events that occur during single sign-on requests. The response that is displayed might be a form, such as when login information is required, or an error or information statement about a condition that occurred while the request was processed.

You can customize the event pages by modifying their appearance or content.

Before you continue with the customization, you need to have a thorough understanding of how event pages are generated and displayed.

Generation of event pages:

Event pages are displayed in response to events that occur during single sign-on requests. They usually contain a form (such as a prompt for user name and password information) or text (such as an informational or error message).

Event pages are dynamic pages that are generated by Security Access Manager using the following information:

Template files

XML or HTML files that are provided with the appliance and contain elements, such as fields, text, or graphics, and sometimes macros that are replaced with information that is specific to the request or to provide a response to the request.

Page identifiers

Event information that corresponds to one or more template files. Each page identifier corresponds to a specific event condition, such as a specific error or a condition in which a message or a form must be displayed.

Message catalogs

Text that is used to replace macros in the template files.

When a request is received, the appropriate response page is generated as follows:

1. Processing of the request occurs and a response to an event is required.
2. Template files and page identifiers are read from the file system.
3. Macros in the template files are replaced with values that are appropriate for the response that is needed.
4. An appropriate event page is generated.
5. The generated event page is displayed.

SAML 2.0 page identifiers:

The SAML 2.0 runtime can display HTML pages in response to events that occur during single sign-on requests. You can select which pages to display and also modify the pages.

Use HTML pages for the following purposes:

- Displaying success and error messages to users
- Asking users for confirmation
- Sending SAML messages

You can customize these HTML pages so that they display what you want. These pages contain *macros* and are similar to other HTML pages in Security Access Manager. A macro is text in an HTML page that is replaced with context-specific information. For example, the macro @ERROR_MESSAGE@ is replaced by text that describes the error that occurred.

You can find the SAML 2.0 pages in the local management interface using these steps:

1. Click **Secure Federation > Global Settings > Template Files**.
2. Expand the locale folder to locate a template file.

For example, the English version of the SAML consent_to_federate.html template is in C/saml20.

All of the available SAML 2.0 HTML pages are listed in the following table.

Table 41. SAML 2.0 HTML page identifiers and macros

Page identifier	Description	Macros and descriptions
saml20/ consent_to_federate.html	Displays during the SAML single sign-on flow whenever the service provider wants to federate the account at the identity provider with the account at the service provider.	@TOKEN:form_action@ The URL to which the SAML message is sent. @TOKEN:SPPProviderID@ The ID of the Service Provider. @TOKEN:SPDisplayName@ The name of the Service Provider. @TOKEN:IPProviderID@ The name of the Identity Provider.
saml20/ logout_partial_success.html	Displays whenever the SAML single log out flow completes with partial success.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @TOKEN:UserName@ The user name that performs the operation.
saml20/logout_success.html	Displays whenever the SAML single log out flow completes successfully.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @TOKEN:UserName@ The user name that performs the operation.
saml20/ nimgmt_terminate_success.html	Displays whenever the SAML name identifier management terminate flow completes successfully.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @TOKEN:UserName@ The user name that performs the operation. @TOKEN:PartnerID@ The ID of the partner.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/ nimgmt_update_success.html	Displays whenever the SAML name identifier management update flow completes successfully.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @TOKEN:UserName@ The user name that performs the operation. @TOKEN:PartnerID@ The ID of the partner.
saml20/ saml_post_artifact.html	Sends the SAML artifact to the partner for HTTP POST binding.	@TOKEN:form_action@ The URL to which the SAML message is sent. @TOKEN:RelayState@ The RelayState. @TOKEN:SamlMessage@ The SAML message.
saml20/ saml_post_request.html	Sends the SAML request message to partner for HTTP POST binding.	@TOKEN:form_action@ The URL to which the SAML message is sent. @TOKEN:RelayState@ The RelayState. @TOKEN:SamlMessage@ The SAML message.
saml20/ saml_post_response.html	Sends the SAML response message to the partner for HTTP POST binding.	@TOKEN:form_action@ The URL to which the SAML message is sent. @TOKEN:RelayState@ The RelayState. @TOKEN:SamlMessage@ The SAML message.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/art_exchange_failed.html	Displays whenever there is a failure during the SAML artifact resolution flow.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/authn_failed.html	Displays whenever there is a failure during the SAML single sign-on flow.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/error_building_msg.html	Displays whenever an outgoing SAML message is not constructed.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/ error_decrypting_msg.html	Displays whenever an incoming SAML message is decrypted.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_missing_config_param.html	Displays whenever a SAML flow is run on a SAML federation with invalid configuration.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_parsing_art.html	Displays whenever an incoming SAML artifact is parsed.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/ error_parsing_msg.html	Displays whenever an incoming SAML message is parsed.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_sending_msg.html	Displays whenever an outgoing SAML message is sent.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_validating_art.html	Displays whenever an incoming SAML artifact is validated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/ error_validating_init_msg.html	Displays whenever a SAML flow is initiated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_validating_msg.html	Displays whenever an incoming SAML message is validated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/ error_validating_msg_signature.html	Displays whenever an incoming SAML message is signature validated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/invalid_art.html	Displays whenever an incoming SAML artifact is validated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/invalid_init_msg.html	Displays whenever a SAML flow is initiated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/invalid_msg.html	Displays whenever an incoming SAML message is validated.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Table 41. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
saml20/logout_failed.html	Displays whenever there is a failure during SAML single logout flow.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/nimgmt_terminate_failed.html	Displays whenever there is a failure during the SAML name identifier terminate management flow.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.
saml20/nimgmt_update_failed.html	Displays whenever there is a failure during the SAML name identifier update management flow.	@REQ_ADDR@ The URL of the request. @TIMESTAMP@ The time stamp of the request. @ERROR_MESSAGE@ The error message. @EXCEPTION_STACK@ The stack trace of the error. Do not use this macro in a production environment.

Template page for the WAYF page:

The Where Are You From (WAYF) page is used at the service provider. The WAYF page enables users to select their identity provider if there is more than one configured in the federation.

When a user arrives at a service provider, a WAYF identifier can be delivered through a cookie or query-string parameter with the request. The entity ID of the identity provider is stored as the value of the cookie or query-string parameter. If the WAYF identifier cookie or query-string parameter is not present, the WAYF page opens.

An example URL that includes the query string parameter for WAYF:

```
https://sp.host.com/isam/sps/samlfed/saml20/
logininitial?RequestBinding=HTTPRedirect&ResponseBinding
=HTTPPost&ITFIM_WAYF_IDP=https://idp.host.com/isam/sps/samlfed/saml20
```

This example is for a SAML 2.0 single sign-on URL. The query string parameter name is ITFIM_WAYF_IDP. The value of the identity provider ID is `https://idp.host.com/isam/sps/samlfed/saml20`.

The WAYF page requires the user to indicate where they came from. If the user is not logged on to their identity provider, they are asked to log on. Depending on the attributes passed, the service provider can grant or deny access to the service.

You can find the template pages for WAYF in the local management interface using these steps:

1. Click **Secure Federation > Global Settings > Template Files**.
2. Expand the locale folder and navigate to `/pages/itfim/wayf`.

Administrators can use the WAYF page without modifications, but in some cases might want to modify the HTML style to match the specific deployment environment.

This template file provides several replacement macros:

@WAYF_FORM_ACTION@

This macro is replaced with the endpoint of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_METHOD@

This macro is replaced with the HTTP method of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_PARAM_ID@

This macro is replaced with ID used by the action for the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_ID@

This macro is replaced with the unique ID of the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_DISPLAY_NAME@

This macro is replaced with the configured display name of the identity provider. This macro is repeated once for each identity provider.

@WAYF_HIDDEN_NAME@

This macro is replaced with the name of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

@WAYF_HIDDEN_VALUE@

This macro is replaced with the value of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

Customizing the Consent to Federate Page:

A *consent to federate page* is an HTML form which prompts a user to give consent to joining a federation. You can customize the *consent to federate page* to specify what information it requests from a user.

Before you begin

Determine what values you want to use for the consent to federate page.

About this task

When a user accesses a federation, they agree to join the federation. The HTML form `saml20/consent_to_federate.html` prompts for this consent. You can customize what the form requests by adding consent values. These values indicate how a user agrees to join a federation and if service providers are notified of the consent. Identity providers receive the consent values in the SAML 2.0 response.

The following values determine how a user joins a federation:

- 1** A user agrees to join a federation without notifying the service provider.
- 0** A user refuses to join a federation.

A URI value

A URI can indicate whether the user agrees to join a federation and if you want to notify the service provider about the user consent. The following table lists and describes the supported URI values.

Table 42. Supported consent values for SAML 2.0 response

Consent value	URI	Description
Unspecified	<code>urn:oasis:names:tc:SAML:2.0:consent:unspecified</code>	The consent of the user is not specified.
Obtained	<code>urn:oasis:names:tc:SAML:2.0:consent:obtained</code>	Specifies that user consent is acquired by the issuer of the message.
Prior	<code>urn:oasis:names:tc:SAML:2.0:consent:prior</code>	Specifies that user consent is acquired by the issuer of the message before the action which initiated the message.
Implicit	<code>urn:oasis:names:tc:SAML:2.0:consent:current-implicit</code>	Specifies that user consent is implicitly acquired by the issuer of the message when the message was initiated.
Explicit	<code>urn:oasis:names:tc:SAML:2.0:consent:current-explicit</code>	Specifies that the user consent is explicitly acquired by the issuer of the message at the instance that the message was sent.
Unavailable	<code>urn:oasis:names:tc:SAML:2.0:consent:unavailable</code>	Specifies that the issuer of the message was not able to get consent from the user.
Inapplicable	<code>urn:oasis:names:tc:SAML:2.0:consent:inapplicable</code>	Specifies that the issuer of the message does not need to get or report the user consent.

Follow the steps in this procedure to customize the consent to federate page.

Procedure

1. Log in to the local management interface.
2. Click **Secure Federation > Global Settings > Template Files**.
3. Expand a locale and select `saml20/consent_to_federate.html`.
4. Click **Edit** and add the appropriate consent values for your federation.
5. Click **Save**.
6. Deploy the changes.

Example

The following example shows an added URI with a consent value Obtained:

```
<input type="radio" checked name="Consent"
value="urn:urn:oasis:names:tc:SAML:2.0:consent:obtained"/>
Consent Obtained.<br/>
```

In this example, the user consent is acquired by the issuer of the message.

Template file macros

Most template pages contain one or more macros. The macros are replaced by values that are specific to the action that is requested on the page.

Macro	Value that replaces the macro
@CLIENT_ID@	The <code>client_id</code> parameter that is specified in the authorization request.
@CONSENT_FORM_VERIFIER@	A unique identifier for the <code>consent_form_verifier</code> parameter value. The value is automatically generated by the authorization server. Do not modify the parameter name or value.
@DETAIL@	The error message.
@ERROR_CODE@	Characters that uniquely identify the error.
@ERROR_DESCRIPTION@	The native language support (NLS) text of the error message that is associated with the error.
@ERROR_MESSAGE@	An error message that is specific to the action in the page. For example, on the One-time password template page for login, the error message indicates that the password submitted contains errors, such as the password is not valid or has expired.
@EXCEPTION_MSG@	The exception message.
@EXCEPTION_STACK@	The stack trace of the error.
@GRACE_PERIOD_AUTH_COUNT@	The amount of grace-period authentication.
@LICENSE@	The contents of the license file.
@LICENSE_FILE@	The name of the license file.
@LICENSE_METADATA@	The metadata that is either: • Defined in the license file. • Not Available if it is not defined.

Macro	Value that replaces the macro
@MAPPING_RULE_DATA@	If the submitted one-time password contains an error, this value is the STS Universal User context attribute with the name @MAPPING_RULE_DATA@ and is type otp.sts.macro.type. This context attribute can be set in the OTPVerify mapping rule.
@MAX_GRACE_PERIOD_AUTH_COUNT@	The maximum count of grace-period authentication that is allotted to a policy.
@MAX_STORED_QUESTIONS@	The maximum number of answers that can be stored per user.
@NUM_REQUIRED_ANSWERS@	The number of valid answers that is required for successful authentication.
@NUM_REGISTERED_QUESTIONS@	The number of questions that the user registered.
@OAUTH_AUTHORIZE_URI@	The URI for the authorization endpoint.
@OAUTH_CLIENT_COMPANY_NAME@	A multi-valued macro that belongs inside an [RPT trustedClients] repeatable replacement list. The values are replaced with the name of the company that requests access to the protected resource.
@OAUTH_CLIENTMANAGERURL@	A multi-valued macro that belongs inside an [RPT trustedClients] repeatable replacement list. The values are replaced with the endpoint of the trusted clients manager.
@OAUTH_CODE@	The oauth_code parameter that is specified in the authorization response.
@OAUTH_CUSTOM_MACRO@	A multi-valued macro that belongs inside an [RPT trustedClients] repeatable replacement list. The values are replaced with trusted client information that contains additional information about an authorized OAuth client.
@OAUTH_OTHER_PARAM_REPEAT@	A multi-valued macro that belongs inside an [RPT oauthOtherParamsRepeatable] repeatable replacement list. The values show the list of extra parameter names.
@OAUTH_OTHER_PARAM_VALUE_REPEAT@	A multi-valued macro that belongs inside an [RPT oauthOtherParamsRepeatable] repeatable replacement list. The values show the list of extra parameter values.

Macro	Value that replaces the macro
@OAUTH_TOKEN_SCOPE_REPEAT@	A multi-valued macro that belongs inside an [RPT oauthTokenScopePreapprovedRepeatable] or [RPT oauthTokenScopeNewApprovalRepeatable] repeatable replacement lists. The values inside the [RPT oauthTokenScopePreapprovedRepeatable] show the list of token scopes that have been previously approved by the resource owner. Alternatively, the values inside the [RPT oauthTokenScopeNewApprovalRepeatable] show the list of token scopes that have not yet been approved by the resource owner.
@OTP_HINT@	The one-time password hint. The hint is a sequence of characters that is associated with the one-time password.
@OTP_METHOD_CHECKED@	For the first method, this macro is replaced with an HTML radio button attribute that causes that radio button to be selected. For the remaining methods that generate, deliver, and verify one-time passwords, this macro is replaced with an empty string.
@OTP_METHOD_ID@	The ID of the method for generating, delivering, and verifying the one-time password. This ID is generated by the OTPGetMethods mapping rule.
@OTP_METHOD_LABEL@	The label of the method for generating, delivering, and verifying the one-time password. This label is generated by the OTPGetMethods mapping rule.
@OTP_METHOD_TYPE@	The type of the currently selected method for generating, delivering, and verifying the one-time password. This type is generated by the OTPGetMethods mapping rule and was selected by the user.
@OTP_STRING@	The one-time password that is generated by the one-time password provider.
@PERMITTED_SCOPES@	A multi-valued macro that belongs inside an [RPT trustedClients] repeatable list. The values are replaced with the token scopes to which the OAuth client has access.
@QUESTION_COUNT@	The number of questions that are presented on the login page.
@QUESTION_TEXT@	The question text. This macro is only populated when the question is a user-provided question.
@QUESTION_INDEX@	The question index. This index corresponds to the array of questions that are presented on the page when questions are presented as a group.
@QUESTION_UNIQUE_ID@	The question unique identifier.

Macro	Value that replaces the macro
@REDIRECT_URI@	The redirect URI that the authorization server uses to send the authorization code to. The value depends on the following items: • Redirect URI that is entered during partner registration. • oauth_redirect parameter that is specified in the authorization request
@REGENERATE_ACTION@	The URI where the Generate button posts the form to regenerate and deliver the new one-time password value.
@RESPONSE_TYPE@	The response_type parameter specified in the authorization request.
@REQ_ADDR@	The URL into which the request from the user is sent.
@RESELECT_ACTION@	The URI where the Reselect button posts the form to reselect the method for generating, delivering, and verifying the one-time password value.
@STATE@	The state parameter that is specified in the authorization request.
@TIMESTAMP@	The time stamp when the error occurred.
@UNIQUE_ID@	A multi-valued macro that belongs inside an [RPT trustedClients] repeatable replacement list. The values are replaced with a unique identifier that identifies the trusted client information for each entry in the list.
@USERNAME@	The Security Access Manager user name.

Mapping rules

Mapping rules are JavaScript code that runs during the authentication flow for Advanced Access Control and Federation.

Mapping rules can be used for multiple purposes. For Advanced Access Control, you can modify rules for the Authentication Service, OTP, and OAuth 2.0. For Federation, you can modify mapping rules to manage identities for OIDC and SAML 2.0. Use the task topic below that applies to the type of mapping rule you want to manage.

Note: Support for the importing of a mapping rule into another mapping rule applies to all mapping rules.

Managing JavaScript mapping rules

Create or edit JavaScript mapping rules.

About this task

When you activate the Advanced Access Control offering, the following mapping rule types are available:

AuthSvc

Authorization service mapping rule.

OAuth

OAuth mapping rule.

OTP One-time password mapping rule.

OIDC OpenID Connect mapping rule.

SAML2_0

SAML 2.0 mapping rule.

Procedure

1. Click **Secure Access Control**.
2. Under **Global Settings**, click **Mapping Rules**. All existing mapping rules are displayed.
3. You can create or modify a mapping rule.
 - To create a mapping rule
 - a. Click **Add**.
 - b. In the **Content** field, enter the JavaScript mapping rule content.
 - c. In the **Name** field, enter a name for the rule.
 - d. In the **Category** field, select the type of the mapping rule from the list.

Note: Only the mapping rule types that apply to your current activated offering are displayed in the list.

- e. Click **Save**.
- To modify a mapping rule
 - a. Select the mapping rule to modify.
 - b. Click **Edit**.
 - c. Modify the mapping rule in the **Content** field as needed.

Note: The **Name** and **Category** fields are not editable.

- d. Click **Save**.

Managing mapping rules

The mapping rules are JavaScript code that run during the authentication flow. Use the rules to customize the authentication service and the one-time password generation, delivery, and verification.

Before you begin

Attention: Use extreme care when you replace mapping rules. Any change that you make to a mapping rule can affect the entire runtime environment. Always export a copy of the original rule you plan to replace so that you have a backup copy.

About this task

You can customize several components through JavaScript code. For example, you can customize the Authentication Service to modify the content of user credential by modifying the AuthSvcCredential mapping rule.

The JavaScript code is run by the Rhino JavaScript engine. Your JavaScript code must conform to JavaScript 1.7. Your JavaScript code is not run under a browser environment. Therefore, you cannot use objects and functions that are available only in a browser environment. You can, however, use standard JavaScript objects (such as Math) and functions (such as parseInt). In addition, your JavaScript code can use white-listed Java classes, which you might need so that you can use operations that are not supported by standard JavaScript functions. You can find the list of these Java classes at “JavaScript whitelist” on page 297. To find out more about using Java classes in JavaScript, see the Rhino documentation <https://developer.mozilla.org/en/docs/Rhino>.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Advanced**.
5. Take one of the following actions:

View a mapping rule:

- a. Select a mapping rule.
- b. Click . The View Mapping Rule panel opens. The content of the mapping rule is displayed.
- c. Click **OK** to close the panel.

Export a mapping rule:

- a. Select a mapping rule.
- b. Click .
- c. Choose a location and save the file.

Replace a mapping rule:

Use an existing mapping rule as the basis for the updated mapping rule.

Attention: When you replace this file, an error in the JavaScript source might be found immediately after it is replaced or it might not be found until the file is run.

- a. Select a mapping rule that you want to replace.
- b. Click .
- c. Click the field or the **Browse** button and select a file.

Attention: The name of the mapping rule cannot be replaced. The name of the uploaded file is ignored.

- d. Click **OK** to upload the mapping rule.

What to do next

When you replace a mapping rule, the appliance displays a message that there are undeployed changes. Deploy the changes when you are done. For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Authentication Service Credential mapping rule

The Authentication Service Credential mapping rule is JavaScript code that you can use to customize the information that is contained in the user credential.

During authentication, the Authentication Service gathers information about the authenticated user, including attributes associated with the user ID. After successful authentication, the Authentication Service provides this information to the Authentication Service Credential mapping rule. The main task of the mapping rule is to modify or add attributes to the user information before it is used to generate a credential.

Customizing the mapping rule is an advanced way to customize the credential. To specify basic credential attributes, use an authentication policy and the Credentials panel in the local management interface instead of creating a custom mapping rule. See [Creating an authentication policy](#).

If you write your own mapping rule and use it to replace the existing rule, be aware of the following considerations:

- Credential attributes are string values. For example, user names and lists of groups are string arrays.
- Do not use spaces, commas, or colons in credential attribute names. Use alphanumeric characters.

The sample mapping rule provides more descriptions about considerations for writing your own mapping rule.

A default `AuthSvcCredential` mapping rule is provided. To review the rule:

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Policy**, click **Authentication**.
4. Click **Advanced**.
5. Select `AuthSvcCredential`.
6. Click  .
7. Choose a location and save the file.

To review an example of a customized credential mapping rule:

1. Log in to the local management interface.
2. Click **Manage System Settings**.
3. Click **File Downloads**.
4. Click `access_control > examples > mapping_rules`.
5. Select `authsvc_credential.js`.
6. Click **Export** to download the file.

If you create your own rule, use it to replace the existing rule. See the replacement instructions in “Managing mapping rules” on page 75.

OTPGetMethods mapping rule

`OTPGetMethods` specifies the methods for delivering the one-time password to the user.

This sample mapping rule sets password delivery conditions for the following delivery methods:

- By email
- By SMS
- No delivery

Each delivery method includes the following attributes and their corresponding value:

id Specifies a unique delivery method ID. This value replaces the @OTP_METHOD_ID@ macro in the OTP Method Selection page. Use a unique value across different methods. For example, sms.

deliveryType

Specifies the delivery plug-in that delivers the one-time password. The value must match one of the types in the DeliveryTypesToOTPDeliveryModuleIds parameter of the OTP response file. For example, sms_delivery.

deliveryAttribute

Specifies an attribute that is associated with the delivery type. The value depends on the one-time password provider plug-in for the delivery type. For example:

- For SMS delivery, the value is the mobile number of the user. For example, mobileNumber.
- For email delivery, the value is the email address of the user. For example, emailAddress.
- For no delivery, the value is an empty string.

label Specifies the unique delivery method to the user. For time-based and counter-based one-time password, use this attribute to specify the secret key of the user. If label is not specified, the time-based and counter-based one-time password code retrieves the key by invoking the user information provider plug-in. This parameter replaces the @OTP_METHOD_LABEL@ macro in the OTP Method Selection page.

otpType

Specifies the one-time password provider plug-in that generates and verifies the password. The value must match one of the types in the OTPTypesToOTPPProviderModuleIds parameter of the OTP response file. For example, mac_otp.

userInfoType

Specifies which user information provider plug-in to use to retrieve user information that is required to calculate the one-time password. This parameter is only required if user information is used for calculation of the one-time password.

To customize one-time password delivery, you can do one of the following actions:

- Create your own mapping rules that are based on the sample OTPGetMethods mapping rule.
- Modify the sample OTPGetMethods mapping rule.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPGenerate mapping rule

OTPGenerate mapping rule specifies the generation of the one-time password for the user.

You can use the OTPGenerate mapping rule in the following configuration:

Modify the one-time password type of the selected method to generate the one-time password

Indicates the one-time password type to determine the one-time password Provider plug-in that generates the one-time password for the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPDeliver mapping rule

The OTPDeliver mapping rule specifies the delivery method of the one-time password to the user.

Use the following OTPDeliver mapping rules:

Generate the one-time password hint

The one-time password hint is a sequence of characters that is associated with the one-time password. The one-time password hint is displayed in the One-Time Password Login page. It is also sent to the user together with the one-time password.

You can customize the way the one-time password hint is generated by modifying the following section in the default OTPDeliver mapping rule:

```
var otpHint = Math.floor(1000 + (Math.random() * 9000));
```

Note: See the comments in the mapping rule for more details.

Generate the formatted one-time password

The formatted one-time password is the formatted version of the one-time password. The formatted one-time password, instead of the actual one-time password, is sent to the user. For example, for one-time password hint abcd, and one-time password 12345678, you can set the formatted one-time password as abcd-12345678. For one-time password hint efgh, and one-time password 87654321, you can set the one-time password as efgh#8765#4321.

You can customize the way that the one-time password is generated by modifying the following section in the sample OTPDeliver mapping rule:

```
var otpFormatted = otpHint + "-" + otp;
```

Note: See the comments in the mapping rule for more details.

Modify the delivery type of the selected method for delivering the one-time password

The delivery type specifies the one-time password Delivery plug-in that delivers the one-time password to the user.

Modify the delivery attribute of the selected method to deliver

The delivery attribute is an attribute that is associated with delivery type. The meaning of the delivery attribute depends on the one-time password provider plug-in for the delivery type. For example, for SMS delivery type, the delivery attribute is the mobile number of the user. For email delivery type, the delivery attribute is the email address of the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

OTPVerify mapping rule

OTPVerify specifies the verification of the one-time password that is submitted by the user.

You can customize the sample OTPVerify mapping rule to modify the following verification rules:

Modify the one-time password type of the user

Indicates the one-time password type to determine the one-time Provider plug-in that verifies the one-time password submitted by the user.

Set the authentication level of the user

After one-time password authentication completes, a credential is issued that contains the authentication level of the user. You can customize the authentication level by modifying the following section in the mapping rule:

```
var authenticationLevel = contextAttributesAttributeContainer.getAttributeValueByNameAndType(
    ("otp.otp-callback.authentication-level", "otp.otp-callback.type"));
var attributeAuthenticationLevel = new Attribute("AUTHENTICATION_LEVEL",
    "urn:ibm:names:ITFIM:5.1:accessmanager", authenticationLevel);
attributeContainer.setAttribute(attributeAuthenticationLevel);
```

Enforce the number of times the user can submit the one-time password in the one-time password login page

If a user exceeds the permitted number of times to submit a one-time password, an error message displays. You can customize the number of times that the user can submit the one-time password in the one-time password login page by modifying the following section in the mapping rule:

```
var retryLimit = 5;
```

By default, this option is set to false.

Note: This setting applies only to MAC OTP.

Identify the secret key of a user

When a user registers with a time-based one-time password application, they are assigned a secret key. Store the secret key in this mapping rule for verification of the user by modifying the following code:

```
var secretStr = new java.lang.String(SECRET_KEY_GOES_HERE);
```

By default, this option is set to false.

Override the one-time password target URL

By default, a user is redirected to a target URL upon completion of an one-time password flow. That target URL was either the initial cached request at the WebSEAL or reverse proxy instance or was specified as part of the one-time password invocation using the **Target** query string parameter.

You can use the OTPVerify mapping rule to override this target URL by adding an attribute called **itfim_override_targeturl_attr**. This attribute ensures that at the completion of a successful one-time password flow, the user is redirected to the override target instead of the initial target.

Example code:

```
var targetUrl = new java.lang.String("http://www.example.com/url");
var targetUrlAttr = new Attribute("itfim_override_targeturl_attr",
"urn:ibm:names:ITFIM:5.1:accessmanager", targetUrl);
attributeContainer.setAttribute(targetUrlAttr);
```

To customize one-time password verification, you can do one of the following actions:

- Create your own verification rules that are based on the sample OTPVerify mapping rule.
- Modify the sample OTPVerify mapping rule.

You can also customize the mapping rule to use access control context data. For details see, “Customizing one-time password mapping rules to use access control context data” on page 81.

Customizing one-time password mapping rules to use access control context data

Some authentication scenarios require that context data used in making an access control decision be available during authentication. You can configure Security Access Manager to capture the content data and make it available to the one-time password mapping rules.

About this task

You can configure Security Access Manager to perform access control policy evaluation when a resource is accessed. The access control policy evaluation can result on a permit with authentication. The required authentication is determined by the access control policy. Some scenarios require that the context data used to perform the access control decision be available during the authentication. In order to provide access to the access control context data, you can persist the context information for the predefined authentication obligations that perform one-time password authentication.

Note: The context data available is limited to the attributes referenced by the access control policy and the request attributes provided by the policy enforcement point. If the policy relies on the risk score to perform access control, the context data available also includes the risk-profile attributes.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control > Global Settings > Advanced Configuration**.
3. Select **attributeCollection.authenticationContextAttributes**.
4. Click  for the property.
5. In the text field, enter a list of comma separated attribute names to be collected during the authorization policy evaluation. For example, if your scenario requires the authentication level and host of the request the configuration property, enter `authenticationLevel, http:host`. The access control context data is provided to the one-time password mapping rules as context attributes values. The following format is used:

```
<stsuser:Attribute name="AttributeName-AttributeURI"
  type="authn.service.context.attribute.type.AttributeDatatype">
<stsuser:Value>AttributeValue</stsuser:Value>
</stsuser:Attribute>
```

Where:

- name is the attribute name and attribute identifier separated by a dash (-).
- type is the attribute data type prefixed by `authn.service.context.attribute.type`.

For example the `authenticationLevel` attribute value is added as:

```
<stsuser:Attribute name="authenticationLevel-urn-ibm:
    security:subject:authenticationLevel"
    type="authn.service.context.attribute.type.Integer">
<stsuser:Value>1</stsuser:Value>
</stsuser:Attribute>
```

6. Click **OK**.
7. When you edit a property, a message indicates that there are undeployed changes. If you have finished making changes, deploy them.
For more information, see Chapter 15, “Deploying pending changes,” on page 213.
8. Configure the mapping rule to use the information collected by this property as the context attribute.

- a. Click **Secure Access Control**.
- b. Under **Policy**, click **Authentication**.
- c. Click **Advanced**.
- d. Select and export the mapping rule.
- e. Use a text editor and modify the rule to access the attributes collected during the access control policy evaluation in the following format:

```
var accessControlAttribute =
contextAttributesAttributeContainer.getAttributeValueByNameAndType
("AttributeName-AttributeURI",
"authn.service.context.attribute.type.AttributeDatatype");
```

Where:

- name is the attribute name and attribute identifier separated by a dash (-).
- type is the attribute data type prefixed by `authn.service.context.attribute.type`.

For example, the `authenticationLevel` attribute can be obtained using the following information:

```
var accessControlAuthenticationLevel =
contextAttributesAttributeContainer.getAttributeValueByNameAndType
("authenticationLevel-urn-ibm:security:subject:authenticationLevel",
"authn.service.context.attribute.type.Integer");
```

- f. Save the mapping rule and take note of its location.
- g. In the local management interface, click **Secure Access Control**.
- h. Under **Policy**, click **Authentication**.
- i. Click **Advanced**.
- j. Select the mapping rule you want to replace.
- k. Click **Replace**. The Replace Mapping Rule panel opens.
- l. Click the field or the **Browse** button and select the file for your saved mapping rule.
Attention: The name of the mapping rule cannot be replaced. The name of the uploaded file is ignored.
- m. Click **OK** to upload the mapping rule.

Managing OAuth 2.0 mapping rules

Use the mapping rules to customize the methods for the OAuth 2.0 or OIDC flow.

About this task

The OAuth 2.0 and OIDC mapping rules are JavaScript code that run during the OAuth 2.0 or OIDC flow. You can view, export, and replace OAuth or OIDC mapping rules.

View the mapping rule if you want to see the content and structure of the mapping rule. Export the mapping rule if you want to save a copy of the mapping rule. You can also edit this copy. Replace a mapping rule if you want to use a new mapping rule.

Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control > Policy > OpenID Connect and API Protection** or **Secure Federation > Manage > OpenID Connect and API Protection**.
3. Click **Mapping Rules**.
4. Perform one or more of the following actions:

View a mapping rule

- a. Select a mapping rule.
- b. Click . The View Mapping Rule panel opens. The content of the mapping rule is displayed.
- c. Click **OK** to close the panel.

Export a mapping rule

- a. Select a mapping rule.
- b. Click .
- c. Choose a location and save the file.

Replace a mapping rule:

Note: Use an existing mapping rule as the basis for the updated mapping rule.

- a. Select a mapping rule that you want to replace.
 - b. Click . The Replace Mapping Rule panel opens.
 - c. Click the field or **Browse** and select a file.
 - d. Click **OK** to upload the mapping rule.
5. When you replace a mapping rule, the appliance displays a message that there are undeployed changes. If you are finished with the changes, deploy them.
For more information, see Chapter 15, “Deploying pending changes,” on page 213.

Related reference:

“OAuth 2.0 and OIDC mapping rule methods” on page 157

You can use Java methods to customize the PreTokenGeneration and PostTokenGeneration mapping rules.

OAuth 2.0 mapping rule methods

You can use Java methods to customize the PreTokenGeneration and PostTokenGeneration mapping rules.

The sample mapping rules are `oauth_20_pre_mapping.js` and `oauth_20_post_mapping.js`.

You can access the sample mapping rules from the LMI. Navigate to **Manage System Settings > Secure Settings > File Downloads**. Continue to either of the following locations:

- **access_control > examples > mapping rules**
- **federation > examples > mapping rules**

The following limitations affect the attribute keys and values that are associated with the `state_id` by using the `OAuthMappingExtUtils` class:

- Keys cannot be null or empty.
- Values cannot be null but can be empty.
- Associated key-value pairs are read and write-allowed and not-sensitive.
- Some keys are reserved for system use and cannot be modified by this utility. For example, the keys and values for the API PIN protection.

For more information, see the Javadoc. In the LMI, navigate to **Manage System Settings > Secure Settings > File Downloads**. Continue to either **access_control > doc** or **federation > doc**.

See also “JavaScript whitelist” on page 297.

Actions to be performed in mapping rules

For certain grant types, you must perform these actions in the pre-token mapping rule.

Resource owner password credentials (ROPC) grant type flow

For the ROPC flow, the pre-token mapping rule is responsible for performing validation of the user name and password. This validation can be performed in various ways. The pre-defined rule that is included with the appliance provides the following examples:

- The java class **PluginUtils** can be used to validate a user name and password against a configured LDAP.
To configure the LDAP to be used, see “Configuring username and password authentication” on page 54.
- Validate the user name and password through an HTTP callout. The mapping rule sends the user name and password to a web service. As the format of the messages is not fixed, various services (for example, REST, SOAP, SCIM) can be used for this purpose. Javadoc on the HTTP client and all other exposed Java classes available in mapping rules can be downloaded from the appliance **File Downloads** page under the path **access_control > doc > ISAM-javadoc.zip**.

JWT and SAML bearer grant type flow

For the JWT or SAML assertion bearer grant type flows, the pre-token mapping rule must perform the following actions:

- Validate the assertion, including but not limited to:
 - Validate the signature (if signed).
 - Decrypt the assertion (if encrypted).
 - Check the expiry and “not before” value of the assertion.
 - Ensure that the issuer is a trusted party.

- Extract the subject from the assertion and set the **USERNAME** field of the STSUU.

The **USERNAME** field of the STSUU can be set via a call, for example:

// username is a variable containing the subject of the assertion

```
stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute(
    "username", "urn:ibm:names:ITFIM:oauth:rule:decision", username));
```

The validation of the assertion can be performed in various ways:

- HTTP callout to a web service. Use the HTTP client to perform this.
- WS-Trust request to the Secure Token Service (STS).
 - A chain must be configured to consume the assertion and return the required information.
 - The **STSClientHelper** will be called to invoke the STS via HTTP. For more information about this class, see the Javadoc that is embedded in the appliance.

Any attributes of the assertion can be extracted and associated to the OAuth grant to be used later. For more information about associating attributes, see “OAuth 2.0 and OIDC mapping rule methods” on page 157.

- The type of the username attribute added must be "urn:ibm:names:ITFIM:oauth:rule:decision" to ensure that only a value populated from the rule is used.

MMFA mapping rule methods

Customize the OAuth **PreTokenGeneration** and **PostTokenGeneration** mapping rules by using these methods.

Sample mapping rules are available from **Manage System Settings > Secure Settings > File Downloads** under the **access_control > examples > mapping rules** directory.

The following limitations affect the attribute keys and values that are associated with the **state_id** by using the **MMFAMappingExtUtils** class:

- Keys cannot be null or empty.
- Values can only be null or empty when specified.
- Associated key-value pairs are read-only and not case sensitive.
- The push token is read-only and case sensitive.

registerAuthenticator

```
public static String registerAuthenticator(
    String stateId
)
```

This method performs the final steps of registering an authenticator. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

These responses come from the runtime after registration.

- The new authenticator's ID if successful.
- Null if not successful.

savePushToken

```
public static boolean savePushToken(
    String stateId,
    String pushToken,
    String applicationID
)
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

pushToken

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

applicationID

The application ID of the authenticator application. This parameter can be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

savePushToken

```
public static boolean savePushToken(
    String stateId,
    String pushToken
)
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

pushToken

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

saveDeviceAttributes

```
public static boolean saveDeviceAttributes(
    String stateId,
    String deviceName,
    String deviceType,
    String osVersion,
    String fingerprintSupport,
    String frontCameraSupport,
    String tenantId
)
```

This method saves various device attributes with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

deviceName

The name of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

deviceType

The type of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

osVersion

The OS version of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

fingerprintSupport

The type of fingerprint sensor that is supported by the device. This parameter can be null or empty. If empty, the value is cleared.

frontCameraSupport

flag that indicates if the device has a front facing camera. This parameter can be null or empty. If empty, the value is cleared.

tenantId

The tenant ID for this registration, if the authenticator application is multi-tenant. This parameter can be null or empty. If empty, the value is cleared.

These responses come from the runtime.

- True if successful.
- False if not successful.

JavaScript whitelist

Advanced Access Control JavaScript mapping rules and Federation mapping rules call Java code from JavaScript. The set of classes that can be called is restricted.

Exercise reasonable caution when you call Java code from JavaScript rules to ensure that accidental damage to appliance resources is avoided.

Common classes allowed in one-time password, OAuth or API protection, dynamic attributes, and JavaScript PIP, federation mapping rules, and access policies.

```
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Class
java.lang.Double
java.lang.Float
java.lang.Integer
java.lang.Long
java.lang.reflect.Array
java.lang.Short
java.lang.String
java.lang.System

java.io.ByteArrayInputStream
java.io.ObjectInputStream
java.io.PrintStream

java.math.BigDecimal

java.util.ArrayList **
java.util.Base64
java.util.Base64$Decoder
java.util.Base64$Encoder
java.util.Date
java.util.HashSet **
java.util.HashMap **
java.util.Iterator
java.util.List
java.util.Map
java.util.Set
java.util.UUID

com.ibm.security.access.httpclient.HttpClient
com.ibm.security.access.httpclient.HttpResponse
com.ibm.security.access.httpclient.Headers
com.ibm.security.access.httpclient.Parameters
com.ibm.security.access.scimclient.ScimClient
com.ibm.security.access.scimclient.ScimConfig
com.ibm.security.access.ciclient.CiClient
com.tivoli.am.rba.attributes.AttributeIdentifier
com.tivoli.am.rba.extensions.RBAExtensions
com.tivoli.am.rba.fingerprinting.ValueContainerIdentifierAdapter
com.tivoli.am.rba.extensions.Attribute$Category
com.tivoli.am.rba.extensions.Attribute$DataType
com.tivoli.am.rba.extensions.Attribute
com.tivoli.am.rba.extensions.PluginUtils
```

** Inner classes for these classes are not supported. Methods that involve an inner class implementation of an interface are not available. For example, do not use the following methods in `java.util.HashMap`:

- `Collection<V> values()`
- `Set<K> keySet()`
- `Set<Map.Entry<K,V>> entrySet()`

For more information about dynamic attributes, see [Dynamic attributes](#).

For information about federation mapping rules, see [Mapping rules](#).

Additional classes allowed in one-time password, OAuth or API protection mapping rules, federation mapping rules, and access policies
--

<pre>com.tivoli.am.fim.base64.BASE64Utility com.tivoli.am.fim.trustserver.sts.modules.http.stsclient.STSClientHelper com.tivoli.am.fim.trustserver.sts.oauth20.Client com.tivoli.am.fim.trustserver.sts.oauth20.Grant com.tivoli.am.fim.trustserver.sts.oauth20.Token com.tivoli.am.fim.trustserver.sts.STSModuleException com.tivoli.am.fim.trustserver.sts.STSUniversalUser * com.tivoli.am.fim.trustserver.sts.utilities.HttpResponse com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtCachedDMAPImp1 com.tivoli.am.fim.trustserver.sts.utilities.InfoCardClaim com.tivoli.am.fim.trustserver.sts.utilities.MMFAMappingExtUtils com.tivoli.am.fim.trustserver.sts.utilities.OAuthMappingExtUtils com.tivoli.am.fim.trustserver.sts.utilities.QueryServiceAttribute com.tivoli.am.fim.trustserver.sts.utilities.USCContextAttributesHelper com.tivoli.am.fim.trustserver.sts.uuser.Attribute * com.tivoli.am.fim.trustserver.sts.uuser.AttributeList * com.tivoli.am.fim.trustserver.sts.uuser.AttributeStatement * com.tivoli.am.fim.trustserver.sts.uuser.ContextAttributes * com.tivoli.am.fim.trustserver.sts.uuser.Group * com.tivoli.am.fim.trustserver.sts.uuser.Principal * com.tivoli.am.fim.trustserver.sts.uuser.RequestSecurityToken * com.tivoli.am.fim.trustserver.sts.uuser.Subject * com.tivoli.am.fim.utils.IteratorWrapper com.tivoli.am.rba.pip.JavaScriptPIP com.tivoli.am.rba.pip.JavaScriptPIP\$Context java.mail.internet.InternetAddress</pre>

* The white list does not contain any implementation of the interfaces that are defined in the org.w3c.dom package. For example, you cannot use the method org.w3c.dom.Document.toXML() in com.tivoli.am.fim.trustserver.sts.STSUniversalUser.

Additional classes allowed in JavaScript PIP

<pre>com.tivoli.am.fim.base64.BASE64Utility com.tivoli.am.rba.pip.JavaScriptPIP com.tivoli.am.rba.pip.JavaScriptPIP\$Context com.tivoli.am.rba.rtss.AttributeLocatorImpl</pre>
--

For more information about policy information points, see Managing policy information points.
--

Additional classes allowed in mapping rules
--

<pre>packages.com.ibm.security.access.user.UserLookupHelper packages.com.ibm.security.access.user.User</pre>
--

For information on mapping rules, see:

- | |
|---|
| <ul style="list-style-type: none">• “Managing OAuth 2.0 and OIDC mapping rules” on page 156• “Managing mapping rules” on page 75 |
|---|

Additional classes to manage server connections
com.ibm.security.access.server_connections.LdapServerConnection com.ibm.security.access.server_connections.LdapServerConnection\$LdapHost com.ibm.security.access.server_connections.ServerConnection com.ibm.security.access.server_connections.ServerConnectionFactory com.ibm.security.access.server_connections.SmtpServerConnection com.ibm.security.access.server_connections.WebServerConnection com.ibm.security.access.server_connections.CiServerConnection
For more information, see “Managing server connections” on page 315.

Classes to use with InfoMap
com.tivoli.am.fim.authsvc.action.authenticator.infomap.InfoMapResult com.tivoli.am.fim.authsvc.action.authenticator.infomap.InfoMapString
For more information, see “Configuring an Info Map authentication mechanism” on page 64.

Classes to use in Access Policies
com.ibm.security.access.policy.Context com.ibm.security.access.policy.Cookie com.ibm.security.access.policy.decision.ChallengeDecisionHandler com.ibm.security.access.policy.decision.DecisionHandler com.ibm.security.access.policy.decision.DenyDecisionHandler com.ibm.security.access.policy.decision.Decision com.ibm.security.access.policy.decision.DecisionType com.ibm.security.access.policy.decision.HtmlPageChallengeDecisionHandler com.ibm.security.access.policy.decision.HtmlPageDecisionHandler com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler com.ibm.security.access.policy.decision.RedirectChallengeDecisionHandler com.ibm.security.access.policy.decision.RedirectDecisionHandler com.ibm.security.access.policy.decision.RedirectDenyDecisionHandler com.ibm.security.access.policy.oauth20.AuthenticationContext com.ibm.security.access.policy.oauth20.AuthenticationRequest com.ibm.security.access.policy.oauth20.ProtocolContext com.ibm.security.access.policy.ProtocolContext com.ibm.security.access.policy.Request com.ibm.security.access.policy.saml20.AuthnRequest com.ibm.security.access.policy.saml20.ProtocolContext com.ibm.security.access.policy.saml20.RequestedAuthnContext com.ibm.security.access.policy.Session com.ibm.security.access.policy.user.Attribute com.ibm.security.access.policy.user.Group com.ibm.security.access.policy.user.User
For more information, see Access policies.

Related tasks:

“Managing OAuth 2.0 and OIDC mapping rules” on page 156

Use the mapping rules to customize the methods for the OAuth 2.0 or OIDC flow.

“Managing mapping rules” on page 75

The mapping rules are JavaScript code that run during the authentication flow. Use the rules to customize the authentication service and the one-time password generation, delivery, and verification.

Managing JavaScript mapping rules

Create, edit, or delete JavaScript mapping rules.

About this task

When you activate the Federation offering, the following mapping rule types are available:

OIDC OpenID Connect mapping rule.

SAML2_0

SAML 2.0 mapping rule.

Procedure

1. Click **Secure Federation**.
2. Under **Global Settings**, click **Mapping Rules**. All existing mapping rules are displayed.
3. You can create, edit, or delete a mapping rule.
 - To create a mapping rule
 - a. Click **Add**.
 - b. In the **Content** field, enter the JavaScript mapping rule content.
 - c. In the **Name** field, enter a name for the rule.
 - d. In the **Category** field, select the type of the mapping rule from the list, or type a name to create your own mapping rule type.

Note: Only the mapping rule types that apply to your current activated offering are displayed in the list.

- e. Click **Save**.
- To modify a mapping rule
 - a. Select the mapping rule to modify.
 - b. Click **Edit**.
 - c. Modify the mapping rule in the **Content** field as needed.

Note: The **Name** and **Category** fields are not editable.

- d. Click **Save**.
- To delete a mapping rule

Note: Do not delete a mapping rule that is currently used by a SAML 2.0 or OpenID Connect federation.

- a. Select the mapping rule to delete.
- b. Click **Delete**.
- c. Confirm the delete operation.

Customizing SAML 2.0 identity mapping

Use mapping rules to map local identities to SAML tokens and to map SAML tokens to local identities.

You can use an attribute source, such as LDAP, for the identity mapping. See [Managing attribute sources](#).

You can use an HTTP external user mapping to map a local identity to a SAML token and to map SAML token to a local identity.

See [Managing JavaScript mapping rules](#) for information about how to create or modify mapping rules.

Mapping a local identity to a SAML 2.0 token

You can map a local identity to a SAML 2.0 token for an identity provider.

The Security Access Manager server places the local user identity information into an XML document that conforms to the security token service universal user (STSUUSER) schema. The identity provider issues a SAML 2.0 token to the service provider. It generates the SAML 2.0 token based on the local identity of the user. You can customize how the local identity is converted into a SAML 2.0 token by using a mapping rule.

Security Access Manager first converts the local identity to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a SAML 2.0 token.

Your mapping rule does not operate directly on local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modification that you make to an STS Universal User has an impact on the output SAML 2.0 token.

The mapping rule is responsible for the following tasks:

1. Mapping Principal Attr Name to a Principal Name entry. When the token module generates the token, this Principal name is not directly used. Instead, the value in the **Name** field is sent as input to the alias service. The alias service obtains the alias name, name identifier, for the principal, and places the returned alias in the generated token module.

The type must be valid for SAML. For example:

`urn:oasis:names:tc:SAML:2.0:assertion`

2. Setting the authentication method to the password mechanism. This action is required by the SAML standard.
3. Setting the audience of the audience restriction condition to the value of the STSUU element AudienceRestriction. If this STSUU element is not present, the audience is set to the Provider ID of the federation partner.
4. Populating the attribute statement of the assertion with the attributes in the AttributeList in the In-STSUU. This information becomes custom information in the token.

Custom attributes might exist that are required by applications that use information that is to be transmitted between federation partners.

5. Specifying whether the assertion conditions should contain the `<saml:OneTimeUse></saml:OneTimeUse>` element. If so, insert a special context attribute into the STSUU as shown:

```
var oneTimeUseAttr = new Attribute("AssertionIncludeOneTimeUse","urn:oasis:names:tc:SAML:2.0:assertion", "true");
stsuu.addContextAttribute(oneTimeUseAttr);
```

6. Setting the NameID attribute in the assertion with Transient NameID format. This action is useful when you want to specify a name value to use instead of the default UUID that is generated by the runtime for Transient NameID format.

To replace the UUID, create a principal name attribute of type `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`, with its value provided by user.

The examples below show the user-provided value *UserGeneratedTransientId* but it could be any other value. The value of the specified STSUU principal name will be set as the NameID in the SAML assertion.

Example mapping rule

```
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.uuser);
var transientNameId = "UserGeneratedTransientId";
stsuu.addPrincipalAttribute(new Attribute("name",
    "urn:oasis:names:tc:SAML:2.0:nameid-format:transient", transientNameId));
```

Example STSUU values after mapping rule applied

```
<stsuuser:Attribute name="name" type="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
  <stsuuser:Value>UserGeneratedTransientId</stsuuser:Value>
</stsuuser:Attribute>
```

Example SAML assertion NameID with Transient NameId formats

```
<saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
  NameQualifier="https://ip-wga/isam/sps/saml20ip/saml20"
  SPNameQualifier="https://sp-wga/isam/sps/saml20sp/saml20"
>UserGeneratedTransientId</saml:NameID>
```

Mapping a SAML 2.0 token to a local identity

You can map a SAML 2.0 token to a local identity for a service provider.

A service provider consumes a SAML 2.0 token that is issued by an identity provider. It generates the local identity of the user based on a SAML 2.0 token. You can customize how a SAML 2.0 token is converted into the local identity of the user by using a mapping rule.

Security Access Manager first converts a SAML 2.0 token to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a local identity of the user.

Your mapping rule does not operate directly on the local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modifications that you make on the STS Universal User impacts the output local identity of the user.

STSRequest and STSResponse access using a JavaScript mapping rule

By using the Default Mapping STS Module and a JavaScript mapping rule, you can perform identity mapping. The mapping rule can access STSRequest and STSResponse objects.

The following two implicit objects and the classes required by these two objects can be exposed (for example, Java DOM, XML classes, and so on):

- STSRequest which represents the WS-Trust request
- STSResponse, which represents the WS-Trust response

Use JavaScript code `stsrequest.getRequestSecurityToken().getBase()` to get the input security token from the WS-Trust request. This returns the input security token as an instance of the Java class `org.w3c.dom.Element`.

Use JavaScript code

`stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken(outputSecurityToken)` to set the output security token in the WS-Trust response. The `outputSecurityToken` is the output security token represented as an instance of Java class `org.w3c.dom.Element`. By default, WS-Trust response contains only one output security token. To return additional output security tokens, you can use the following JavaScript code:

```
stsresponse.addRequestSecurityTokenResponse().setRequestedSecurityToken(outputSecurityToken)
```

The examples in the following topics show the mapping to and from a base64 encoded JSON string. They use the Default Mapping module with a JavaScript mapping rule. The JavaScript mapping rule accesses the STSRequest and STSResponse objects and performs the identity mapping.

Mapping a JSON Web Token to a SAML2 token example

You can map a base64 encoded JSON string to a SAML 2 token by using a JavaScript mapping rule.

About this task

The steps show an end-to-end JSON to SAML2 mapping. STSRequest and STSResponse access using a JavaScript mapping rule provides a description of this support.

Procedure

1. Create a JavaScript mapping rule by using the local management interface.
 - a. Select **Secure Federation > Global Settings > Mapping Rules**.
 - b. Click **Add**.
 - c. In the **Content** field, copy and paste the following code:

```
importClass(com.tivoli.am.fim.base64.BASE64Utility);
importClass(com.tivoli.am.fim.trustserver.sts.user.Attribute);

var jwtElement = stsrequest.getRequestSecurityToken().getBase();
var jwtText     = jwtElement.getTextContent();
var jwtString   = new java.lang.String(BASE64Utility.decode(jwtText), "UTF-8");
var jwt         = JSON.parse(jwtString);

for (var name in jwt) {
    if (jwt.hasOwnProperty(name)) {
        if ("sub".equals(name)) {
            stsuu.addPrincipalAttribute(new Attribute("name",
"urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress", jwt[name]));
        } else {
            stsuu.addAttribute(new Attribute(name,
"urn:oasis:names:tc:SAML:2.0:attrname-format:basic", jwt[name]));
        }
    }
}
```
 - d. In the **Name** field, enter `jwt_saml`.
 - e. In the **Category** field, select **SAML2_0**.
 - f. Click **Save** and deploy the changes.
2. Assemble the Security Token Service (STS) template.
 - a. Select **Secure Federation > Manage > Security Token Service**.
 - b. Click **Templates**.
 - c. Click **Add** and name the template JSON to SAML2. Click **OK**.
 - d. Select the JSON to SAML2 template and add the Default Map Module in Map mode and a Default SAML 2.0 token in Issue mode.
 - e. Save and deploy the changes.
3. Create an STS chain that references the mapping rule and template you created in the previous steps.
 - a. Within the **Security Token Service** panel, select **Module Chains**.
 - b. Click **Add** to create the module chain, with the following values:

Table 43. JSON to SAML2 module chain values

Tab: Field	Value
Overview: Name	JSON to SAML2
Overview: Description	base64 encoded JSON string to SAML2 conversion STS chain
Overview: Template	JSON to SAML2
Lookup: Request Type	Validate
Lookup: Applies to Address	jwtapplies to
Lookup: Issuer Address	jwtissuer
Properties: Default Map Module (JavaScript file containing the identity mapping rule)	jwt_saml
Properties: Default SAML 2.0 Token (Name of the organization issuing the assertions)	isam
Properties: Default SAML 2.0 Token (Amount of time before the issue date that an assertion is considered valid)	60
Properties: Default SAML 2.0 Token (Amount of time that the assertion is valid after being issued)	60
Properties: Default SAML 2.0 Token (List of attribute types to include)	*

Use the defaults for all of the fields that are not specified in the table.

- c. Save and deploy the changes.
4. Use **curl** to test the chain.
 - a. Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">jwtissuer</wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          <wsa:Address>jwtapplies to</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
<JWT>ewogICJ1bWVpbCI6ICJqb2huLmRvZU81eGFtcGx1LmNvbSI6IAogICJmYW1pbH1fbmFtZSI6ICJkb2UiLCAK
ICAiZ212ZW5fbmFtZSI6ICJqb2huIiwgCiAgIm1lzcyciOiI6ICJpc2FtIiwgCiAgInN1YiI6ICiWMTIzNDU2Nzg5Igp9</JWT>
      </wst:Base>
    </ns1:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bold embedded element, **<JWT>** **</JWT>**, is the input to the chain. This is a Base64 encoded JSON string that contains the following data::

```
{
  "email": "john.doe@example.com",
  "family_name": "doe",
  "given_name": "john",
  "iss": "isam",
  "sub": "0123456789"
}
```

- b. Save this file as jwt.xml.

- c. Run the following **curl** command, where **jwt.xml** is the WS-Trust 1.2 message:

```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary @jwt.xml https://ip-rte/TrustServer/SecurityTokenService
```

The following results are returned:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-ENV:Header>
  <soap:Body>
    <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestSecurityTokenResponse xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
wsu:Id="uuidc1288a62-0153-1f8b-bf2a-b4c46f51cd03">
        <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:wsp="http://schemas.xmlsoap.org/ws/
        <wsa:EndpointReference>
          <wsa:Address>jwtapplies-to</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
        <wst:Lifetime xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
          <wsu:Created>2016-03-29T06:56:13Z</wsu:Created>
          <wsu:Expires>2016-03-29T06:57:13Z</wsu:Expires>
        </wst:Lifetime>
        <wst:RequestedSecurityToken>
          <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/X
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ID="Assertion-uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03"
IssueInstant="2016-03-29T06:56:13Z" Version="2.0">
            <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">isam</saml:Issuer>
            <saml:Subject>
              <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
0123456789</saml:NameID>
              <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                <saml:SubjectConfirmationData NotOnOrAfter="2016-03-29T06:57:13Z"></saml:SubjectConfirmationData>
              </saml:SubjectConfirmation>
            </saml:Subject>
            <saml:Conditions NotBefore="2016-03-29T06:55:13Z" NotOnOrAfter="2016-03-29T06:57:13Z">
              <saml:AudienceRestriction>
                <saml:Audience>jwtapplies-to</saml:Audience>
              </saml:AudienceRestriction>
            </saml:Conditions>
            <saml:AuthnStatement AuthnInstant="2016-03-29T06:56:13Z">
              <saml:AuthnContext>
                <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef>
              </saml:AuthnContext>
            </saml:AuthnStatement>
            <saml:AttributeStatement>
              <saml:Attribute Name="given_name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                <saml:AttributeValue xsi:type="xs:string">john</saml:AttributeValue>
              </saml:Attribute>
              <saml:Attribute Name="email" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                <saml:AttributeValue xsi:type="xs:string">john.doe@example.com</saml:AttributeValue>
              </saml:Attribute>
              <saml:Attribute Name="iss" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                <saml:AttributeValue xsi:type="xs:string">isam</saml:AttributeValue>
              </saml:Attribute>
              <saml:Attribute Name="family_name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                <saml:AttributeValue xsi:type="xs:string">doe</saml:AttributeValue>
              </saml:Attribute>
            </saml:AttributeStatement>
          </saml:Assertion>
        </wst:RequestedSecurityToken>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </soap:Body>
</soap:Envelope>
```

The JSON string is mapped into the SAML assertion, as shown by the previous bold text. The attributes in the SAML2 assertion are mapped from JSON attributes.

```
<wst:RequestedAttachedReference xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.
  <wss:SecurityTokenReference xmlns:wss11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.0.
wss11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0">
    <wss:KeyIdentifier xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
ValueURI="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
      Assertion-uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03</wss:KeyIdentifier>
    </wss:SecurityTokenReference>
  </wst:RequestedAttachedReference>
  <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
  <wst:Status>
    <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid</wst:Code>
  </wst:Status>
</wst:RequestSecurityTokenResponse>
</wst:RequestSecurityTokenResponseCollection>
</soap:Body>
</soap:Envelope>
```

Related tasks:

Mapping a SAML2 token to a base64 encoded JSON string example
You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

Mapping a SAML2 token to a JSON Web Token example

You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

About this task

The steps show an end-to-end SAML to JSON mapping. STSRequest and STSResponse access using a JavaScript mapping rule provides a description of this support.

Procedure

1. Create a JavaScript mapping rule using the local management interface.

- a. Select **Secure Federation > Global Settings > Mapping Rules**.

- b. Click **Add**.

- c. In the **Content** field, copy and paste the following code:

```
importClass(com.tivoli.am.fim.base64.BASE64Utility);
importClass(com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils)

var jwt = {};

var it = stsuu.getPrincipalAttributes();
var jt = stsuu.getAttributes();

while (it.hasNext()) {
    var attribute = it.next();
    var name      = new String(attribute.getName());
    var value     = new String(attribute.getValues()[0]);

    if ("name".equals(name)) {
        jwt["sub"] = value;
    } else {
        jwt[name] = value;
    }
}

while (jt.hasNext()) {
    var attribute = jt.next();
    var name      = new String(attribute.getName());
    var value     = new String(attribute.getValues()[0]);

    jwt[name] = value;
}

var document = IDMappingExtUtils.newXMLDocument();
var jwtString = JSON.stringify(jwt);
var jwtText = document.createTextNode(BASE64Utility.encode((new java.lang.String(jwtString)));
var jwtElement = document.createElement("JWT");

jwtElement.appendChild(jwtText);

stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken(jwtElement);
```

- d. In the **Name** field, enter `saml_jwt`.

- e. In the **Category** field, select `SAML2_0`.

- f. Click **Save** and deploy the changes.

2. Assemble the Security Token Service (STS) template.

- a. Select **Secure Federation > Manage > Security Token Service**.
 - b. Click **Templates**.
 - c. Click **Add** and name the template SAML2 to JSON. Click **OK**.
 - d. Select the SAML2 to JSON template and add the Default SAML 2.0 Token in Validate mode and a Default Map Module in Map mode.
 - e. Save and deploy the changes.
3. Create an STS chain that references the mapping rule and template you created in the previous steps.
 - a. Within the **Security Token Service** panel, select **Module Chains**.
 - b. Click **Add** to create a module chain, with the following values:

Table 44. SAML2 to JSON module chain values

Tab: Field	Value
Overview: Name	SAML2 to JSON
Overview: Description	SAML2 to base64 encoded JSON string conversion STS chain
Overview: Template	SAML2 to JSON
Lookup: Request Type	Validate
Lookup: Applies to Address	SAML2_AppliesTo
Lookup: Issuer Address	SAML2_Issuer
Properties: Default Map Module (JavaScript file containing the identity mapping rule	saml_jwt

Use the defaults for all of the fields not in the table.

- c. Save and deploy the changes.
4. Use **curl** to test the chain.
 - a. Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">SAML2_Issuer</wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
          <wsa:Address>SAML2_AppliesTo</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
    </ns1:RequestSecurityToken>
    <wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        ID="Assertion-uuidbcb46a39-0153-1337-8efa-fec506fb7461" IssueInstant="2016-03-28T10:10:53Z" Version="2.0">
        <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">isam</saml:Issuer>
        <saml:Subject>
          <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">0123456789</saml:NameID>
          <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml:SubjectConfirmationData NotOnOrAfter="2016-03-28T10:11:53Z"/>
          </saml:SubjectConfirmation>
        </saml:Subject>
        <saml:Conditions NotBefore="2016-03-28T10:09:53Z" NotOnOrAfter="2016-03-29T10:11:53Z">
          <saml:AudienceRestriction>
            <saml:Audience>jwt_saml</saml:Audience>
          </saml:AudienceRestriction>
        </saml:Conditions>
        <saml:AuthnStatement AuthnInstant="2016-03-28T10:10:53Z">
          <saml:AuthnContext>
            <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef>
          </saml:AuthnContext>
        </saml:AuthnStatement>
        <saml:AttributeStatement>
```


The bold element in the SAML2 assertion is mapped to the JSON attributes in the result.

- ```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary @sam12.xml https://ip-rte/TrustServer/SecurityTokenService
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-ENV:Header>
 <soap:Body>
 <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
 <wst:RequestSecurityTokenResponse
 xmlns:wst="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
 wsu:Id="uuiidc1676e30-0153-16a8-86b5-c34fd1aca7a8">
 <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
 <wsa:EndpointReference>
 <wsa:Address>SAML2_AppliesTo</wsa:Address>
 </wsa:EndpointReference>
 </wsp:AppliesTo>
 <wst:RequestedSecurityToken>
 <JWT>eyJzdWIiOiIwMTIzNDU2Nzg5IiwiaWF0IjEwMTIzNDU2Nzg5ImppZ2ZlZW5fbmFtZS16ImprvaG4iLCJ0b3Rpbk9yQWZ0ZXIiOiIyMDE2LTAz
 LT15VDEwOjEwXzUzWiIsIkFklGh1bnRpZ2F0aW9uTWV0aG9kIjoiaXJ0eU0mhc21zOm5hbWVzOnRjOjNBNUw6MS4w
 OmFtOnBhc3N3b3JkIiwiaWZ1haWw0IjQqb2huLmRvZUB1eGFTcGx1LmNmV3SiIsIkF1ZG11bmN1UmVzdHJpY3Rpb25
 bD52kaXRpb2U4QXVkaWVuY2UiOiJqdG3Rfc2FtbCIsIm1zcyI6Im1zYW0iLCJjc3N1ZU1uc3RhbHQiOiIyMDE2LT
 AzLT15VDEwOjEwXzUzWiIsImZhbW1seV9uYW11IjoiaXZG91IiwiaW90QmVmb3J1IjoiaWJxNjE0My8yOFQxMDow
 To1M1oiLCJBDXRoZW50aWwHdG1vbkluc3RhbHQiOiIyMDE2LTAzLT15VDEwOjEwXzUzWiIsIm1zc3VlciI6Im1
 zYW0ifQ==</JWT>
 </wst:RequestedSecurityToken>
 <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
 <wst:Status>
 <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid</wst:Code>
 </wst:Status>
 </wst:RequestSecurityTokenResponse>
 </wst:RequestSecurityTokenResponseCollection>
 </soap:Body>
</soap:Envelope>
```

```
{
 "sub": "0123456789",
 "given_name": "john",
 "NotOnOrAfter": "2016-03-29T10:11:53Z",
 "AuthenticationMethod": "urn:oasis:names:tc:SAML:1.0:am:password",
 "email": "john.doe@example.com",
 "AudienceRestrictionCondition.Audience": "jwt_saml",
 "iss": "isam",
 "IssueInstant": "2016-03-28T10:10:53Z",
 "family_name": "doe",
 "NotBefore": "2016-03-28T10:09:53Z",
 "AuthenticationInstant": "2016-03-28T10:10:53Z",
 "issuer": "isam"
}
```

## Mapping a base64 encoded JSON string to a SAML2 token example

You can map a base64 encoded JSON string to a SAML 2 token by using a

JavaScript mapping rule.

## OpenID Connect mapping rules

Mapping rules allow users to customize the information that is propagated from an OpenID Connect Provider or what is consumed by a Relying Party.

These mapping rules can either be JavaScript, which is invoked internally via the STS, or the mapping can be performed externally via a HTTP request.

### OpenID Connect Provider mapping rules

When you write mapping rules for a provider, the primary goal is to augment the claims that are included in the ID token.

After mapping rule execution, all attributes in the STSUU will be added to the `id_token` as a claim, where the attribute key is the key in the `id_token`, and the value is the value of the attribute. If there are several attributes with the same key, then an array containing each attribute will be added to the claim. Some context information is made available to the user when writing mapping rules; the context attributes of the passed in STSUU will contain attributes with the type `"urn:ibm:ITFIM:oidc:provider:context"`, which can be used to make decisions on what claims are added, or if any other actions are performed.

These context attributes include:

- The client ID of the client making the request.
- The federation name of the provider servicing the request.
- The redirect URI sent in the request.
- The response type of the request.
- The state parameter of the request.
- The user-consented scopes for the request.

### OpenID Connect Relying Party mapping rules

When you write mapping rules for a Relying Party, the resulting STSUU is turned into a PAC that is used to authenticate the user to a Reverse Proxy via EAI.

The attributes that are included in that PAC will be the attributes of the STSUU, and the principal will be the first principal which was in the STSUU. When writing mapping rules for a Relying Party, the values of the `id_token` will be made available as Attributes in the STSUU. Some additional context is made available to the user via the STSUU's context attributes. These attributes will have the types `"urn:ibm:ITFIM:oidc:client:idtoken:param"` and `"urn:ibm:ITFIM:oidc:client:token:param"`.

These context attributes include:

- All of the claims inside the `id_token`.
- The raw JWT.
- Any issued access or refresh tokens.
- All of the properties of the issued bearer token if an authorization code flow is used.
- All of the parameters issued in the response if an implicit flow is used.

### Attribute sources

Both OpenID Connect Providers and Relying Parties can be configured to use an attribute source.

For an OpenID Connect Provider, this can be used instead of a mapping rule. However for an OpenID Connect Relying Party a mapping rule must still be present, this mapping rule is required to construct the principal used in the `iv-cred`.

For more information about attribute sources, see [Managing attribute sources](#).

## Import a mapping rule from another mapping rule

You can reuse mapping rules by importing a mapping rule from another mapping rule.

When you want to create a new mapping rule, or customize an existing mapping rule, you can reuse JavaScript code from a previously defined mapping rule. With this feature, you can define a mapping rule once and then reuse it in other mapping rules.

Use the function `importMappingRule()` to specify a mapping rule to import. For example, you can define a mapping rule that is called `Utility.js` that contains functions for obtaining an HTTP header and an HTTP cookie.

```
function getHeader(name) {
 // function for getting HTTP header
}

function getCookie(name) {
 // function for getting HTTP cookie
}
```

If you have another mapping rule that is called `Credential.js`, which also needs to obtain HTTP headers, use the following code to include the functions from the `Utility.js` mapping rule:

```
importMappingRule("Utility");
var host = getHeader("Host");
// do something with the host header
var sessionID = getHeader("PD-SESSION-ID");
// do something with the session ID
```

The function `importMappingRule()` accepts a list of mapping rule names and imports each of the mapping rules. For example:

```
importMappingRule("Utility","Credential","UserIdentity");
```

Alternatively, you can also make multiple calls to `importMappingRule()` within one script. For example:

```
importMappingRule("Utility");
importMappingRule("Credential");
importMappingRule("UserIdentity");
```

The JavaScript engine throws an error if you do not specify a mapping rule name, or if you specify the name of a mapping rule that does not exist.

Use the Local Management Interface (LMI) to view existing mapping rules that are defined on your system. Select **Secure Federation > Global Settings > Mapping Rules**, or **Secure Advanced Access > Global Settings > Mapping Rules**.

**Note:**

On the LMI menu, the icon **Import** is for importing mapping rules into IBM Security Access Manager, not for importing a mapping rule into an existing mapping rule. Use the **Edit** icon to add the `importMappingRule()` function to an existing mapping rule.

## Auditing from Mapping Rules

Data that is used in JavaScript Mapping rules can be audited with **IDMappingExtUtils**.

Data that is used in making an access control decision is audited in JavaScript Mapping Rules. You can use the **IDMappingExtUtils.logAuditEvent()** function to capture data and make it available in audit logs.

### Scenario 1: User completed password reset (USC\_PasswordReset\_Success)

```
IDMappingExtUtils.logAuditEvent(username, "Successfully completed password reset", true);
```

produces the following audit log entry:

```
<CommonBaseEvent creationTime="2018-09-04T00:23:05.239Z" extensionName="IBM_SECURITY_AUTHN" globalInstanceId="FIMa1f61b17016516b19127fb3858d13aff+667021443">
 <contextDataElements name="Security Event Factory" type="eventTrailId">
 <contextId>FIM_a1f61b17016516b19127fb3858d13aff+667021443</contextId>
 </contextDataElements>
 <extendedDataElements name="EventName" type="string">
 <values>JavaScriptEvent</values>
 </extendedDataElements>
 <extendedDataElements name="Username" type="string">
 <values>testuser</values>
 </extendedDataElements>
 <extendedDataElements name="Outcome" type="string">
 <values>SUCCESSFUL</values>
 </extendedDataElements>
 <extendedDataElements name="Message" type="string">
 <values>Successfully completed password reset</values>
 </extendedDataElements>
 <extendedDataElements name="progName" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnProvider" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="partner" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="trustRelationship" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="userInfoList" type="noValue">
 <children name="userInfo" type="noValue">
 <children name="registryUserName" type="string">
 <values>Not Available</values>
 </children>
 <children name="appUserName" type="string">
 <values>testuser</values>
 </children>
 </children>
 </extendedDataElements>
 <extendedDataElements name="authnType" type="string">
 <values>authenticationService</values>
 </extendedDataElements>
 <extendedDataElements name="action" type="string">
 <values>validate</values>
 </extendedDataElements>
 <extendedDataElements name="tokenType" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnScope" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="outcome" type="noValue">
 <children name="result" type="string">
 <values>SUCCESSFUL</values>
 </children>
 </extendedDataElements>
</CommonBaseEvent>
```

```

</children>
<children name="majorStatus" type="int">
 <values>0</values>
</children>
</extendedDataElements>
<sourceComponentId application="IBM Security Access Manager" component="Authentication and Federated Identity" comp
<situation categoryName="ReportSituation">
 <situationType
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ReportSituation" reasoningScope="INTERNAL" report
 </situation>
</CommonBaseEvent>

```

## Scenario 2: OAuth flow is used to enroll Mobile Multi-Factor Authentication (\*PostTokenGeneration)

```

var auditData = {"status":"registering MMFA", "deviceName":device_name, "deviceType":device_type,
IDMappingExtUtils.logAuditEvent(displayName, JSON.stringify(auditData), true);

```

produces the following audit log entry:

```

<CommonBaseEvent creationTime="2018-09-04T00:23:05.241Z" extensionName="IBM_SECURITY_AUTHN" globalInstanceId="FIMa1f61b17016516b19127fb3858d13aff+667021443" globalInstanceName="FIMa1f61b17016516b19127fb3858d13aff+667021443">
 <contextDataElements name="Security Event Factory" type="eventTrailId">
 <contextId>FIM_a1f61b17016516b19127fb3858d13aff+667021443</contextId>
 </contextDataElements>
 <extendedDataElements name="EventName" type="string">
 <values>JavaScriptEvent</values>
 </extendedDataElements>
 <extendedDataElements name="Username" type="string">
 <values>displayName</values>
 </extendedDataElements>
 <extendedDataElements name="Outcome" type="string">
 <values>SUCCESSFUL</values>
 </extendedDataElements>
 <extendedDataElements name="Message" type="string">
 <values>{"status":"registering MMFA","deviceName":"Jasmine's iPhone","deviceType":"iPhone","osVersion":"11"}</values>
 </extendedDataElements>
 <extendedDataElements name="progName" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnProvider" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="partner" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="trustRelationship" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="userInfoList" type="noValue">
 <children name="userInfo" type="noValue">
 <children name="registryUserName" type="string">
 <values>Not Available</values>
 </children>
 <children name="appUserName" type="string">
 <values>displayName</values>
 </children>
 </children>
 </extendedDataElements>
 <extendedDataElements name="authnType" type="string">
 <values>authenticationService</values>
 </extendedDataElements>
 <extendedDataElements name="action" type="string">
 <values>validate</values>
 </extendedDataElements>
 <extendedDataElements name="tokenType" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnScope" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="outcome" type="noValue">
 <children name="result" type="string">
 <values>SUCCESSFUL</values>
 </children>
 <children name="majorStatus" type="int">
 <values>0</values>
 </children>
 </extendedDataElements>

```

```

<sourceComponentId application="IBM Security Access Manager" component="Authentication and Federated Identity" componen
<situation categoryName="ReportSituation">
 <situationType
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ReportSituation" reasoningScope="INTERNAL" reportCate
 </situation>
</CommonBaseEvent>

```

### Scenario 3: OTP retry limit is reached (OTPVerify)

```
IDMappingExtUtils.logAuditEvent("", "Retry limit exceeded.", false);
```

produces the following audit log entry:

```

<CommonBaseEvent creationTime="2018-09-04T00:50:24.603Z" extensionName="IBM_SECURITY_AUTHN" globalInstanceId="FIMa20f1ed2016510a09325fb3858d13aff+1389200945">
 <contextDataElements name="Security Event Factory" type="eventTrailId">
 <contextId>FIM_a20f1ed2016510a09325fb3858d13aff+1389200945</contextId>
 </contextDataElements>
 <extendedDataElements name="EventName" type="string">
 <values>JavaScriptEvent</values>
 </extendedDataElements>
 <extendedDataElements name="Username" type="string">
 <values></values>
 </extendedDataElements>
 <extendedDataElements name="Outcome" type="string">
 <values>UNSUCCESSFUL</values>
 </extendedDataElements>
 <extendedDataElements name="Message" type="string">
 <values>Retry limit exceeded.</values>
 </extendedDataElements>
 <extendedDataElements name="progName" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnProvider" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="partner" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="trustRelationship" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="userInfoList" type="noValue">
 <children name="userInfo" type="noValue">
 <children name="registryUserName" type="string">
 <values>Not Available</values>
 </children>
 <children name="appUserName" type="string">
 <values></values>
 </children>
 </children>
 </extendedDataElements>
 <extendedDataElements name="authnType" type="string">
 <values>authenticationService</values>
 </extendedDataElements>
 <extendedDataElements name="action" type="string">
 <values>validate</values>
 </extendedDataElements>
 <extendedDataElements name="tokenType" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="authnScope" type="string">
 <values>Not Available</values>
 </extendedDataElements>
 <extendedDataElements name="outcome" type="noValue">
 <children name="result" type="string">
 <values>UNSUCCESSFUL</values>
 </children>
 <children name="failureReason" type="string">
 <values>Retry limit exceeded.</values>
 </children>
 <children name="majorStatus" type="int">
 <values>1</values>
 </children>
 </extendedDataElements>
</sourceComponentId application="IBM Security Access Manager" component="Authentication and Federated Identity" compon
<situation categoryName="ReportSituation">

```

```
<situationType
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ReportSituation" reasoningScope="INTERNAL"
/>
</situation>
</CommonBaseEvent>
```

---

## Managing Distributed Session Cache

In a clustered appliance environment, session information is stored in the Distributed Session Cache. To work with these sessions, use the Distributed Session Cache management page.

### About this task

The Distributed Session Cache feature replaces the Session Management Server. The Session Management Server (SMS) is not supported on IBM Security Access Manager for Web Version 8 and later.

### Procedure

1. From the top menu, select the menu for your activation level.
  - **Secure Web Settings > Manage > Distributed Session Cache**
  - **Secure Access Control > Global Settings > Distributed Session Cache**
  - **Secure Federation > Global Settings > Distributed Session Cache**

All replica set names and the number of sessions in each replica set are displayed.

2. You can then view the replica set server list and manage sessions in a particular replica set.
  - a. To view a list of the servers that are registered with a replica set, select the replica set and then click **Servers**.
  - b. To manage the sessions in a replica set, select the replica set and then click **Sessions**.

**Tip:** Typically, the list of sessions contains many entries. You can locate a session or a user faster by using the filter in the upper left corner.

#### Delete a specific session

- 1) Select the session to delete.
- 2) Click **Delete**.
- 3) In the confirmation window, click **Delete Session**.

#### Delete all sessions for a user

- 1) Select any session for that user.
- 2) Click **Delete**.
- 3) In the confirmation window, click **Delete User**.

---

## Managing server connections

To use data from outside your appliance in your policies, you must define the server connection to access the data.

### Before you begin

Obtain the connection information for the existing database server you want to define for your policy information point.

## About this task

You can create server connections to data sources, such as Oracle, DB2, solidDB, PostgreSQL, LDAP, SMTP, Web Service, Cloud Identity, and ISAM Runtime. You can have multiple servers for an LDAP connection.

### Procedure

1. Log in to the local management interface.
2. Click **Secure Access Control**.
3. Under **Global Settings**, click **Server Connections**.
4. Take one of the following actions:

#### Filter server connections:

- a. In the Quick Filter field, type one or more characters. For example, enter g to search for all server connection names that contain g or G.
- b. Press Enter.

#### Add a server connection:

- a. Click the  drop-down button.
- b. Select **Oracle, DB2, SolidDB, PostgreSQL, LDAP, SMTP, Web Service, Cloud Identity or ISAM Runtime**.
- c. Complete the properties for the new server connection.

#### Modify an existing server connection:

- a. Select a server connection.
- b. Click .
- c. Complete the properties for the server connection.

#### Delete a server connection:

**Note:** Do not delete a server connection if it returns attributes that are used in a policy or risk score.

- a. Select a server connection.
- b. Click .

5. Optional: For LDAP connections, take one of the following actions in the **Servers** tab.

#### Add a server connection:

- a. Click the  drop-down button.
- b. Complete the properties for the new server connection.

#### Modify an existing server connection:

- a. Click .
- b. Complete the properties for the server connection.

#### Delete a server connection:

- a. Select a server connection.
- b. Click .

#### Move a server connection:



a. Select a server connection.

b. Click  or  .

## What to do next

- For information on server properties, see “Server connection properties.”
- After you define a server connection to a data source, you can create a policy information point to access this data and use it in policies. See [../admin/task/managingpolicyinformationpoints.dita](#).

## Server connection properties

To access a data source outside of the appliance, define the properties of the server.

The Server Connection properties table describes the properties on the **Server Connections** panel for the Advanced Access Control and Federation module activation levels.

- **Advanced Access Control:** Configure LDAP, database, web service, or Cloud Identity server connections so that you can set up policy information points. You can configure any of the server connection types.
- **Federation:** Configure an LDAP server as an attribute source for attribute mapping. Federation does not configure any of the other database server connection types.

Table 45. Server Connection properties

Property	Description
Name	Specifies the name for the server connection. Ensure that the name is unique. Select this name when you define the policy information point. <b>Note:</b> The server connection name must begin with an alphabetic character. Do not use control characters, leading and trailing blanks, and the following special characters ~ ! @ # \$ % ^ & * ( ) +   ` = \ ; " ' < > ? , [ ] { } / anywhere in the name.
Description	Describes the server connection. This property is optional.
Type	Shows the server connection type. (Read only)
JNDI ID (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the JNDI ID that the server uses. Ensure that the ID is unique. Use only alphanumeric characters: a-b, A-B, 0-9
Server name (Oracle, DB2, solidDB, PostgreSQL, SMTP only)	Specifies the name or IP address for the server.
Port (Oracle, DB2, solidDB, PostgreSQL, LDAP, SMTP only)	Specifies the port number where the connection to the server can be made.
URL (Web Service only)	Specifies the URL where the connection to the server can be made.
User name (Oracle, DB2, solidDB, PostgreSQL, SMTP, and Web Service only)	Specifies the user name that has the correct permissions to access the resources.
Password (Oracle, DB2, solidDB, PostgreSQL, SMTP, and Web Service only)	Specifies the password to access the server.

Table 45. Server Connection properties (continued)

Property	Description
<b>SSL</b>	Specifies whether SSL is used for connecting to the server. Select <b>True</b> or <b>False</b> . The default value is <b>True</b> .
<b>Driver type</b> (Oracle only)	Specifies the driver type. Select <b>Thin</b> or <b>OCI</b> . The default value is <b>Thin</b> .
<b>Service name</b> (Oracle only)	Specifies the name of the service.
<b>Database name</b> (DB2, PostgreSQL only)	Specifies the name of the database.
<b>Host name</b> (LDAP only)	Specifies the host name or IP address of the LDAP server.
<b>Bind DN</b> (LDAP only)	Specifies the LDAP distinguished name (DN) that is used when binding, or signing on, to the LDAP server. <b>Note:</b> If this value is set to "anonymous", the appliance uses an anonymous bind to the LDAP directory server. Typically the <b>bind-dn</b> has significant privileges so that it can be used to modify LDAP registry entries, such as creating users and resetting passwords via pdadmin or the Registry Direct Java API. Using an anonymous connection to LDAP typically comes with very limited access, perhaps at most search and view of entries, at the least no access at all. If anonymous access has sufficient privileges, then it might be usable for the WebSEAL level of access on users and groups. This access includes the permission for a user to change password if "bind-auth-and-pwdchg = yes" is set ("ldap.bind-auth-and-pwdchg = true" for Registry Direct Java API).
<b>Bind Password</b> (LDAP only)	Specifies the password for the LDAP bind DN. <b>Note:</b> If bind DN (bind-dn) is set to anonymous, you can use any non-empty string as the value of bind password (bind-pwd).
<b>Administration hostname</b> (Cloud Identity only)	Specifies the administration hostname of the Cloud Identity subscription.
<b>Client ID</b> (Cloud Identity only)	Specifies the client ID of an API Client on Cloud Identity.
<b>Client Secret</b> (Cloud Identity only)	Specifies the client secret of an API Client on Cloud Identity.
<b>SSL Truststore</b> (LDAP, Web Service, and Cloud Identity only)	Specifies the truststore that verifies the credentials.
<b>SSL Mutual Authentication Key</b> (LDAP, Web Service, and Cloud Identity only)	Label of the client certificate to be presented when connecting to the LDAP. This property is sourced from SSL Truststore. <b>Note:</b> This field is required only if mutual SSL authentication is required by the server.

**Note:** For information on SSL configuration, see Configuring SSL connections.

The properties in the following table are connection manager properties. The defaults that are listed are the current known defaults. All tuning properties are optional.

Table 46. Tuning properties

Property	Description
<b>Aged timeout (seconds)</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the amount of time, in seconds, before a physical connection is discarded by pool maintenance. Specify -1 to disable this timeout. The default is -1.
<b>Connection timeout (seconds)</b>	Specifies the amount of time, in seconds, after which a connection times out.  For Oracle, DB2, solidDB, PostgreSQL, and SMTP, specify -1 to disable this timeout. The default is 30 seconds.  For LDAP, specify only integers, 1 or greater. The default is 120 seconds.
<b>Max Idle Time (seconds)</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the maximum amount of time, in seconds, after which an unused or idle connection is discarded during pool maintenance. Specify -1 to disable this timeout. The default is 1800 seconds.
<b>Max Idle Time (seconds)</b> (LDAP only)	Specifies the amount of time, in seconds, after which an established connection is discarded as idle. Set this to a value lower than the connection idle timeout on the LDAP server. <b>Note:</b> This is only applicable for performing Attribute Mapping from an LDAP server.
<b>Reap time (seconds)</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the amount of time, in seconds, between runs of the pool maintenance thread. Specify -1 to disable pool maintenance. The default is 180 seconds.
<b>Max pool size</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the maximum number of physical connections for a pool. Specify 0 for unlimited. The default is 50.
<b>Max pool size</b> (LDAP only)	Specifies the maximum number of connections that are pooled. <b>Note:</b> This is only applicable for performing Attribute Mapping from an LDAP server.
<b>Min pool size</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the minimum number of physical connections to maintain in a pool. The aged timeout can override the minimum.
<b>Purge policy</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies which connections to delete when a stale connection is detected in the pool. Select from the following options:  <b>Entire pool</b>  When a stale connection is detected, all connections in the pool are marked stale, and when no longer in use, are closed. This is the default option.  <b>Failing connection only</b>  When a stale connection is detected, only the connection that was found to be bad is closed.  <b>Validate all connections</b>  When a stale connection is detected, connections are tested and the ones that are found to be bad are closed.
<b>Max connections per thread</b> (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the limit of open connections on each thread.

Table 46. Tuning properties (continued)

Property	Description
Cache connections per thread (Oracle, DB2, solidDB, PostgreSQL only)	Specifies the number of cache connections for each thread.

---

## Point of contact profiles

Use the local management interface to work with your point of contact profiles.

You can perform the following point of contact profile tasks:

- “Creating a point of contact profile”
- “Updating or viewing a point of contact profile” on page 321
- “Deleting a point of contact profile” on page 321
- “Setting a current point of contact profile” on page 322

### Creating a point of contact profile

Create a point of contact server profile to capture the information needed for the runtime to communicate with the point of contact server.

#### About this task

You can create point of contact profiles with the Federation module or the Advanced Access Control module.

Three point of contact profiles provided by Security Access Manager are ready for use.

When you want to create your own profile that is similar to an existing one, use **Create Like** to save time. If you do not want to reuse any of the existing specifications, create a brand new one with **Create**. The details are in the following procedure.

#### Procedure

1. From the local management interface, select **Secure Federation** or **Secure Access Control**. Then, **Global Settings > Point of Contact**. A list of point of contact server profiles displays. The list includes three preconfigured profiles and any other custom profiles that you created.
2. Take one of the following actions:
  - Click **Create** to create a custom point of contact profile.
  - Select a profile from the list and click **Create Like** to start with values similar to an existing profile.
3. On the Profile Name page, enter the name of the profile. The first character of the profile name must be alphanumeric. The maximum number of characters is 200.
4. Optional: Enter a description.
5. Specify the parameter information:
  - Enter the information on each tabbed page, and click **Next**.
  - In the Callback Parameters section on each page, click **Create** to open a window to add a set of parameter name and value pairs. Click **Save** when complete.

- Add as many parameters as you need. The **Value** field might be empty for some parameters.
  - To delete a parameter name from the list, select the parameter and click **Delete**.
6. At the Summary page, if everything is correct, click **Finish**.
  7. Deploy the pending changes.

### What to do next

- See “Callback parameters and values” on page 322 for descriptions.
- You might want to change the current point of contact profile. See “Setting a current point of contact profile” on page 322.

## Updating or viewing a point of contact profile

Update or view a point of contact server profile.

### About this task

You cannot update the preconfigured point of contact profiles.

### Procedure

1. From the local management interface, select **Secure Federation** or **Secure Access Control**. Then, **Global Settings** > **Point of Contact**. A list of point of contact server profiles displays.
2. Perform one of the following actions:
  - Update
    - a. Select a profile from the list that is not a preconfigured profile and click **Update** to change the configuration details.
    - b. Click **Next** to see each page and make updates if necessary.
    - c. On the Summary page, click **Finish** to save your changes.
    - d. Deploy the changes
  - View
    - a. Select a profile from the list and click **Properties** to look at the configuration details without making updates.
    - b. Click on each tab to see the information.
    - c. Click **OK** when finished.

### What to do next

See “Callback parameters and values” on page 322 for more information about the properties.

## Deleting a point of contact profile

Use the local management interface to remove a point of contact profile.

### About this task

You cannot delete the following profiles:

- A preconfigured point of contact profile.
- A profile that is set as the current profile. Select another profile as the current one, if necessary.

See “Setting a current point of contact profile.”

### Procedure

1. From the local management interface, select **Secure Federation > Global Settings > Point of Contact** or **Secure Access Control > Global Settings > Point of Contact**. A list of point of contact server profiles displays.
2. Select a profile from the list, that is not a preconfigured profile, and click **Delete**. The details of the selected profile display.
3. Review the profile to ensure that it is the one you want to delete.
4. Click **Finish**.
5. Click **OK** to confirm.
6. Deploy the change.

## Setting a current point of contact profile

Set a point of contact profile as the current one so that the federation runtime communicates with the point of contact server using the correct set of specifications.

### Procedure

1. From the local management interface, select **Secure Federation > Global Settings > Point of Contact** or select **Secure Access Control > Global Settings > Point of Contact**. A list of point of contact server profiles displays. The list includes three preconfigured profiles and any other custom profiles that you created. The green dot indicates the current profile.
2. To change the current profile, select the profile you want to use as the current one and click **Set As Current**. The current profile indicator displays next to the profile you selected.
3. Deploy the changes.

## Callback parameters and values

Specify the callback parameters and values when you define a point of contact profile.

### Sign In callbacks

#### **fim.user.request.header.name**

The name of the header that contains the user name of the user.

Data type: String

Example: iv-user

#### **fim.attributes.response.header.name**

The name of the header that contains the attributes of the user.

Data type: String

Example: am-fim-eai-xattrs

#### **fim.groups.response.header.name**

The name of the header that contains the groups of the user.

Data type: String

Example: fim.groups

#### **fim.server.response.header.name**

The name of the header that contains the hostname that authenticates the user.

Data type: String

Example: fim.server

**fim.target.response.header.name**

The name of the header that contains the redirect URL.

Data type: String

Example: am-fim-eai-redir-url

**fim.user.response.header.name**

The name of the header that contains the user name of the user.

Data type: String

Example: am-fim-eai-user-id

**fim.user.session.id.response.header.name**

The name of the header that contains the reverse proxy session ID of the user.

Data type: String

Example: user\_session\_id

**fim.cred.response.header.name**

The name of the header that contains the IVCred of the user.

Data type: String

Example: am-fim-eai-pac

**url.encoding.enabled**

Indicates whether the EAI header names and values are URL encoded. The default setting for this property is false. The EAI header names and values are not URL encoded.

Data type: Boolean

Example: false

## Sign Out callbacks

**fim.user.session.id.request.header.name**

The name of the header that contains the reverse proxy session ID of the user.

Data type: String

Example: user\_session\_id

**fim.user.request.header.name**

The name of the header that contains the user name.

Data type: String

Example: iv-user

## Local ID

**fim.attributes.request.header.name**

The name of the header that contains the attributes of the user.

Data type: String

Example: fim.attributes

**fim.cred.request.header.name**

The header that contains the IVCred of the user.

Data type: String

Example: iv-creds

**fim.groups.request.header.name**

The name of the header that contains the groups of the user.

Data type: String

Example: iv-groups

**fim.user.request.header.name**

The name of the header that contains the user name.

Data type: String

Example: iv-user

## **Authenticate**

**fim.user.request.header.name**

The name of the header that contains the user name.

Data type: String

Example: iv-user

**authentication.macros**

A list of macros that defines contextual information to pass to the web reverse proxy login page. The macros you specify can customize an authentication login page for a specific service provider. For more information, see Customizing the SAML 2.0 login form.

Data type: String

Example: If an identity provider wants to display the provider ID and target URL of a partner, specify the following macros:

%PARTNERID%,%TARGET%



---

## Chapter 18. Choose a synchronization mode

You can choose synchronization mode types for the IBM Security Access Manager Advanced Access Control component.

Feature	Recommended Replication Mode	Comment
Context/Risk-based access	NEARSYNC	Context/risk-based access usage data, device fingerprint, and obligation data are used in authentication decision making and requires to be replicated in a failover scenario.
Mobile Multi Factor Authentication (MMFA)		MMFA authenticators required to be presents at standby node during failover.
User Self Care (USC)		

Overall recommendation: NEARSYNC within datacenter. In case of cross data center replication where network latency is significant , it is advised use Supersync as HADR mode to get optimal performance.

For more information on synchronization mode types for the IBM Security Access Manager Federation component, see Choose a synchronization mode for the Federation component.



---

# Index

## A

- access token OAuth 105
- Advanced Access Control
  - point of contact profile 320
- advanced configuration
  - category filter 219
  - property descriptions 219
- API definition
  - API definition
    - attaching to resource 140
    - publishing 140
- API protection
  - client 137, 139
  - definition 132
- API protection client
  - managing 139
  - registering 137
- API protection definition
  - creating 131
  - managing 132
  - PreTokenGeneration mapping rule
    - update 7
- appliances
  - clusters 315
- application interface
  - manage 17
- attribute IDs
  - version 7.0
  - usage 24
- attributes
  - oauthScope 143
- authentication
  - client 120
  - configuring 39
- authentication mechanism
  - configuring
    - end user license agreement 59
    - configuring consent to device registration 58
    - configuring HTTP redirect 57
    - configuring username and password 54
- authentication service
  - configuration 35
- authorization code OAuth 105
- authorization grant OAuth 105

## B

- backward compatibility mode
  - one-time password 6

## C

- callback parameters
  - point of contact profile 322
- certificate authentication 21
- certificates
  - client
    - See client certificates

- client
  - managing API protection 139
  - registering API protection 137
- client authentication
  - OAuth 2.0 token endpoint 120
  - types 120
- client certificate authentication 21
- clients
  - OAuth 105
- cluster
  - cluster configuration management
    - page
      - LMI 94
    - configuration 94
    - master nodes
      - configure 94
      - registration 94
      - unregistration 94
- cluster signature file
  - export 94
  - import 94
- clusters
  - Distributed Session Cache 315
- compliance
  - NIST SP800-131a 27
- configuration
  - advanced 218
  - Knowledge Questions authentication mechanism 68
- Consent to Federate Page
  - customization 280
  - description 280
- custom domain
  - defining 211
  - policy attachments 211

## D

- database configuration
  - upgrade 1
- DB2
  - server connection 315
  - server connection properties 317
- DB2 database
  - upgrade 4
- defining a custom domain 211
- definition
  - creating API protection 131
  - managing API protection 132
- deploying changes 213
- device fingerprints
  - managing 170
- distributed session cache 215
- Distributed Session Cache (DSC)
  - managing 315
- DSC 215

## E

- endpoints
  - OAuth
    - definitions 108
    - URLs 108
- error messages
  - OAuth HTTP 170
- event pages
  - customization overview 269
  - overview 270

## F

- federation
  - OAuth 2.0
    - endpoint definitions 108
    - naming 108
    - URLs 108
  - OAuth configuration 131
  - point of contact profile 320

## G

- getting started
  - configure 13

## H

- HTML pages
  - SAML 2.0 270

## I

- identity mapping
  - SAML 2.0 token, local user 303
- identity provider mapping
  - SAML 2.0 token, local user 302
- isamcfg
  - command line reference 95
  - overview 90
  - reference 95
  - WebSEAL point of contact 92
- isamcfg tool 97
  - appliance 91
  - external machine 91
  - reverse proxy instance 91
  - WebSEAL configuration 99
- isamcfg worksheet 97
  - WebSEAL 99

## L

- LDAP
  - server connection 315
  - server connection properties 317
- listening interfaces 25

- LMI
  - cluster configuration management
    - page 94
- local user identity mapping
  - from 302
  - to 303
- login form
  - customizing (overview) 269

## M

- macros
  - HTML pages for SAML 2.0 270
- mapping rules
  - custom
    - one-time password 297
  - customizing for context data 81, 291
  - managing 76, 285
  - OTPDeliver 79, 289
  - OTPGenerate 79, 288
  - OTPGetMethods 78, 287
  - OTPVerify 80, 290
  - PostTokenGeneration 157, 294
  - PreTokenGeneration 7
  - SAML 2.0 token to local identity 302

## N

- NIST SP800-131a compliance 27

## O

- OAuth
  - API protection client 137
  - endpoints 108
  - federation configuration 131
  - HTTP error responses 170
  - reverse proxy configuration 125
- OAuth 2.0
  - about 111
  - authorization code 105
  - concept 111
  - endpoint
    - definitions 108
    - URLs 108
  - overview 111
  - state management 122
  - template page types 167
  - token endpoint client
    - authentication 120
  - trusted clients management 123
  - workflow 111
- OAuth STS interface 173
- OAuth support 105
- oauth\_20\_pre\_mapping.js mapping rule
  - file 7
- oauthScope attributes 143
- OIDC
  - reverse proxy configuration 125
- one-time password
  - backward compatibility mode 6
  - configuring an RSA mechanism 46
  - configuring delivery 50
  - configuring HOTP 40
  - configuring MAC 45
  - configuring TOTP 42

- one-time password (*continued*)
  - delivery method 78, 157, 287, 294
  - managing mapping rules 76, 285
- Oracle
  - server connection 315
  - server connection properties 317
- Oracle database
  - upgrade 5
- OTP
  - backward compatibility mode 6
- OTPDeliver
  - usage 79, 289
- OTPGenerate
  - usage 79, 288
- OTPGetMethods
  - usage 78, 287
- OTPVerify
  - usage 80, 290

## P

- page identifiers
  - HTML for SAML 2.0 270
- pages, event
  - SAML 2.0 270
- pending changes 213
- point of contact
  - token endpoint 120
- point of contact profile
  - callback parameters 322
  - creating 320
  - current 322
  - deleting 321
  - updating 321
- policy
  - policy
    - attaching to resource 140
    - publishing 140
  - policy attachments 211
- PostgreSQL
  - server connection 315
  - server connection properties 317
- PostTokenGeneration
  - usage 157, 294
- PreTokenGeneration mapping rule 7
- protected resource OAuth 105

## R

- replica sets
  - management 315
- resource owner OAuth 105
- resource server OAuth 105
- reverse proxy
  - OAuth configuration 125
  - OIDC configuration 125
- reverse proxy instance
  - isamcfg 92
- runtime component
  - configure 17
  - manage 17
- runtime listening interfaces 25
- runtime security services
  - attribute ID changes 24
  - configure for client certificate authentication 21

## S

- SAML 2.0
  - Consent to Federate Page
    - customization 280
  - event pages 270
  - local user mapping 302, 303
  - page identifiers 270
  - responses 280
- scenarios
  - authentication configuration 37
- scope attributes
  - OAuth 143
- server connection
  - properties 317
  - tuning properties 317
- server connections
  - DB2 315
  - LDAP 315
  - Oracle 315
  - PostgreSQL 315
  - SMTP 315
  - solidDB 315
- servers
  - OAuth 105
- service provider mapping
  - SAML 2.0 token, local user 303
- sessions
  - information 315
- single sign-on
  - event pages 269
  - HTML pages 270
- SMTP
  - server connection 315
  - server connection properties 317
- solidDB
  - server connection 315
  - server connection properties 317
- solidDB database
  - upgrade 3
- specifications OAuth 105
- starting configuration
  - steps 13
- state management OAuth 2.0 122
- step-up authentication
  - configuring 37

## T

- template files 249
  - macros 281
- template files root
  - manage 249
- template pages
  - consent to authorize 167
  - WAYF page 279
- tool
  - isamcfg 90
- trusted clients management
  - overview 123

## U

- upgrade
  - database configuration 1
  - DB2 database 4
  - Oracle database 5

- upgrade (*continued*)
  - solidDB database 3
- user self-administration 170
- username and password authentication
  - configuring 54

## V

- version 7.0
  - attribute ID updates 24

## W

- WAYF page
  - template 279
- WebSEAL
  - configuring 121
  - point of contact 121
  - token endpoint 121
  - use 7.0 attribute IDs 24
- WebSEAL policy enforcement point
  - isamcfg 92
- Where Are You From (WAYF) page
  - See* WAYF page







Printed in USA