IBM Cúram Social Program Management
8.0.0

Cúram Security Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 100

Edition

This edition applies to IBM® Cúram Social Program Management 8.0.0.

Contents

Figures

Tables

Chapter 1. Securing Social Program Management

Ensure that you secure your IBM Cúram Social Program Management applications. Authentication and authorization are two key components of application security. The Social Program Management web client is configured to support form-based authentication. You can configure different authentication modes with the JAAS login module. Functional elements in Social Program Management are secured by security identifiers. This data is linked to a user and can be configured.

Authentication Overview

In Cúram, authentication is the process of determining if a user is who they say they are. Authentication is needed where a user must be verified in order to access a secure resource on a system.

Form-based authentication is where a user is presented with a form allowing them to enter username and password credentials. These credentials are compared against the credentials stored on the system for this username, if they match the user is considered an authenticated user for the system. For security reasons the password for authenticating a user is stored on the system in a digested form.

The Cúram web client is configured to support form-based authentication, which means that before a user can access any of the web client content, they will be redirected to a login form to authenticate.

The authentication process involves the verification of the username and password, and this is performed by default by a JAAS (Java™ Authentication and Authorization Service) login module. HTTPS/SSL is turned on by default in the web client ensuring the form-based login authentication mode is secure.

Authentication

Different authentication modes can be configured (depending on authentication requirements) by the Cúram Java Authentication and Authorization Service (JAAS) login module.

The following are the authentication modes supported:

- Default Authentication
- Identity Only Authentication
- External Access Security Authentication

Each of these modes is described in detail in the sections that follow.

Authentication Architecture

Use the information in this flow chart to understand the architecture for the authentication process of a user.

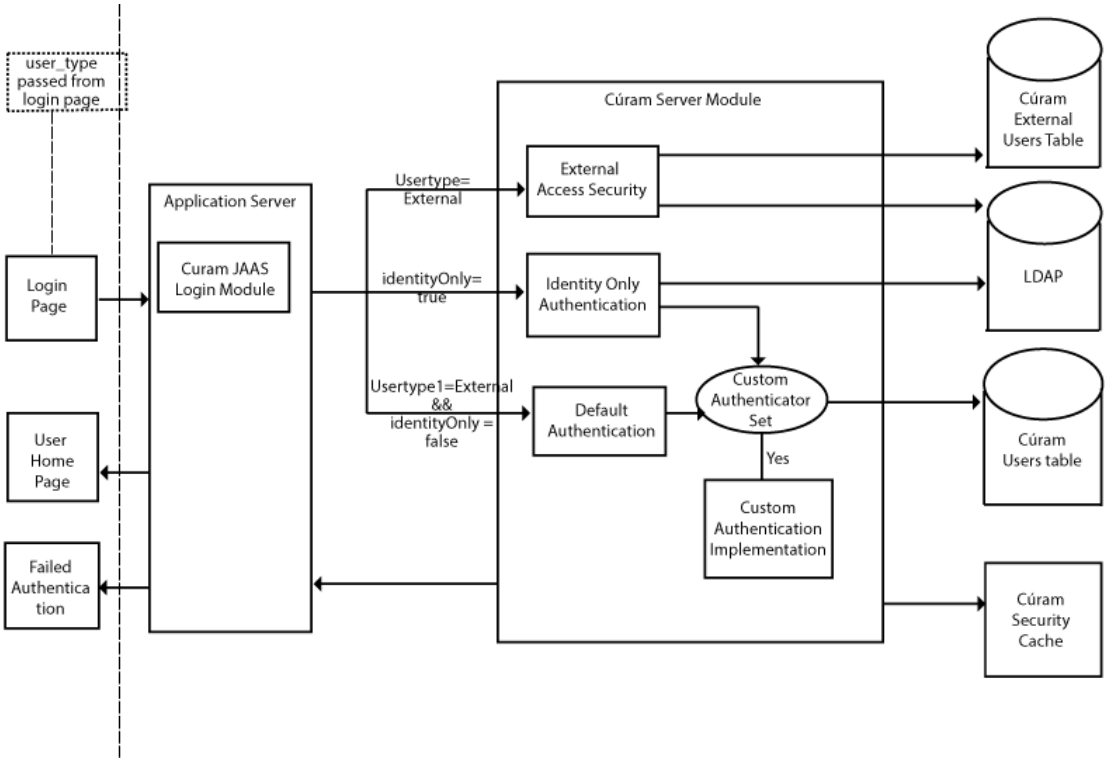

Figure 1. Authentication architecture

The flow chart shown here outlines the architecture for the authentication process of a user. The default authentication is completed for a user. This behavior can be customized for both internal and external users, depending on the authentication requirements. The sections in Authentication Overview chapter that follow describe in detail each of the functional areas that make up the authentication architecture, indicating where customizations are possible.

Default Authentication

Default authentication for Cúram involves the user who logs in through the login screen, where the user is prompted for a `username` and `password` as credentials. These credentials then are passed to the Cúram Java Authentication and Authorization Service (JAAS) login module configured in the application server.

The default authentication is run and the `username` and `password` entered are checked against the `username` and `password` stored on the Cúram Users database table. The Cúram `username` is immutable, but you have the option of configuring your system to use a Cúram `login ID` instead, which is changeable. The `login ID` is a logical extension of the Cúram user and the same verifications that are checked for the `username` also are checked for the `login ID`. For more information about alternate login IDs, see “Alternate Login IDs” on page 3.

Authentication runs a number of verifications against the login credentials. For more information on the login verifications, see “Default Authentication” on page 9.

Provided all verifications are successful, the user is considered to be authenticated by the application.

After the user is authenticated, the user then is added to the Cúram Security Cache. The Cúram Security Cache stores the `username` and all related authorization data for that user to optimize the authorization data retrieval for a user. For more information on the Cúram Security Cache, see “Security Data Caching” on page 18. Figure 2.3 highlights the path taken for default authentication.

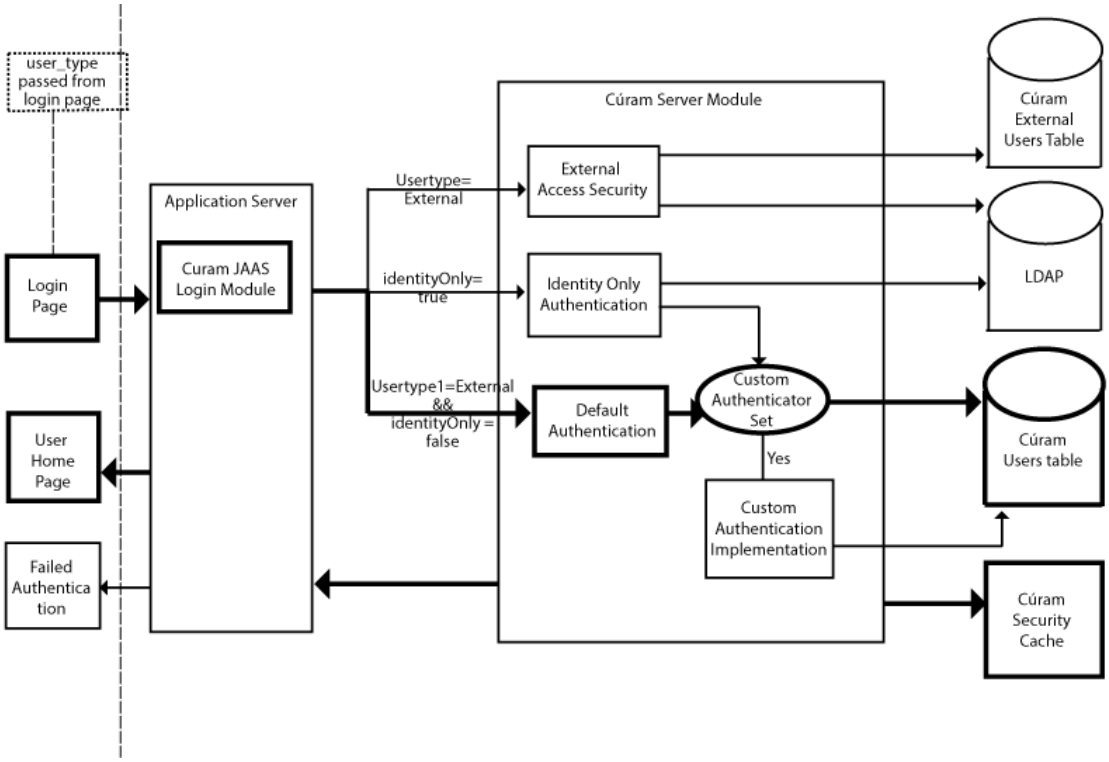

Figure 2. Default authentication

Alternate Login IDs

By default, IBM Cúram Social Program Management uses the `username` and digested `password` that is stored in the `Cúram Users` table for authentication. The `username` cannot be changed after it is created and the lack of flexibility might not meet requirements for some installations. You have the option to configure an alternate `login ID` that can be updated. However, if the default security implementation that is configured during installation meets your requirements, it is not necessary to configure an alternate `login ID`.

The `login ID` functions as a logical extension of the `Cúram Users` table. When the alternate `login ID` is used the `username` still exists and is used internally, but the user logs in with the `login ID`.

Things to note when using the alternate `login ID`:

- Users can log in with their alternate login IDs if available or user names if not. When the property alternate login is disabled, users are only allowed to log in with their user names.
- The Cúram `ExtendedUsersInfo` table, where the `login ID` is stored, must be populated before the application turns on the alternate `login ID` feature, which is explained in more detail in the following explanation.
- When using login IDs, authentication results are stored in the `AuthenticationLog` table and the `AltLogin` column indicates whether the `UserName` column represents a `username` (false) or `login ID` (true).
- Login IDs are only applicable to internal Cúram users; that is, users stored on the `Cúram Users` table. However, if you are using identity-only with alternate Login IDs then wherever those IDs are stored (for example, WebSphere registry, Lightweight Directory Access Protocol (LDAP), and so on) must match the login IDs stored in the Cúram `ExtendedUsersInfo` table.
- When assigning login IDs, you need to take care with IDs that are used internally or have dependencies (for example, with property values) outside of the `Cúram Users` table. These IDs are the user names that would cause issues if their`login ID` differed from the `username` without a corresponding change as indicated:

- SYSTEM - In WebSphere Application Server, WebLogic Server, or Kubernetes WebSphere Liberty this user name is associated with Java Message Service (JMS) processing and is made part of the your application server configuration at deployment time. For more information on changing this ID, see "Mandatory Cúram Users" on page 19 and the appropriate application server in *Cúram Deployment Guide*.
- DBTOJMS - this value is the default DBtoJMS username used by batch processing and is referenced by property curam.security.credentials.dbtojms.username. For more information, see "Mandatory Cúram Users" on page 19, "JMS Messaging" on page 20, and"Deferred Processing" on page 20 the *Cúram Batch Processing Guide*.
- WEBSVCS - this value is the default web services username and is referenced by property curam.security.credentials.ws.username. For more information, see "Mandatory Cúram Users" on page 19, "Web Services" on page 19, and the *Cúram Web Services Guide*.
- Unauthenticated - is the principal WebSphere uses for unauthenticated users and this login ID should not be changed.

To enable the use of the alternate login ID, after you have populated the ExtendedUsersInfo table, set the curam.security.altlogin.enabled property to true. For more information about Social Program Management properties, see the *Cúram Server Developer's Guide*). This value is a static property and Cúram must be restarted for it to take effect.

When the curam.security.altlogin.enabled property is set to true, authentications are not processed directly through the user name column in the Cúram Users table. Instead, authentications are all processed through the ExtendedUsersInfo login ID, which references the Cúram Users table.

The **Login ID** field is displayed in the administrative pages only when the corresponding curam.security.altlogin.enabled property is set to true.

To populate the ExtendedUsersInfo table (see table that follows for ExtendedUsersInfo you have a number of options; for instance:

- With a simple SQL statement, you can populate the table by using the user name in the Users table; so, there is no immediate user impact: INSERT INTO EXTENDEDUSERSINFO (USERNAME, LOGINID, UPPERLOGINID, VERSIONNO) (SELECT USERNAME, USERNAME, UPPER(USERNAME), 1 FROM USERS); You can then roll out your modifications to the login IDs in a controlled manner.
- You can implement an SQL application that implements your user name and login ID mapping (for example, LDAP common names).

  Note: You must maintain the user name foreign key relationship between the Users and ExtendedUsersInfo tables.

Table 1. ExtendedUsersInfo Table Structure

Name	Type	Size	Description
USERNAME	VARCHAR	256	Username is an immutable string. This field has a foreign key relationship with username field in Users table.

Table 1. ExtendedUsersInfo Table Structure (continued)			
Name	**Type**	**Size**	**Description**
LOGINID	VARCHAR	1280	Login ID is associated to the user name and can be updated. The login ID functions as a logical extension of the Cúram Users table. Users can log in to Cúram application by using Login ID.
UPPERLOGINID	VARCHAR	1280	Login ID in uppercase. Uppercase login ID is used for supporting case-insensitivity.
Version No	VARCHAR	4	Version Number.

Configuring internal and external users

If you have both internal and external users, extra calls might occur to the `getRegisteredUserName()` method in the `ExternalAccessSecurity` class. The security cache calls the `getRegisteredUserName()` method if the login ID is not found in the security cache. Therefore, all internal and external login IDs and user names must be unique, unless the `curam.util.security.UserScope interface` is implemented. Otherwise, an external user that matches a login ID might be found in the security cache and therefore not found as an external user. If a login ID can't be found either in the cache, or through the External Access Security implementation if it is provided, then an `INFRASTRUCTURE.INFO_LOGIN_ID_DOES_NOT_MAP_TO_USERNAME` exception occurs.

Configuring a custom alternate login implementation

A customer can set the `curam.citizenworkspace.alternate.login.implementation` property to point to a custom alternate login implementation, as shown in the following example:

```
curam.citizenworkspace.alternate.login.implementation=curam.citizenworkspace.security.impl.Sampl
eCitizenWorkspaceAlternateLogin
```

A customer can use the alternate login implementation to specify custom code that returns the user name when an alternate login ID is submitted. The alternate login implementation must extend the `CitizenWorkspaceAlternateLogin` abstract class and provide an implementation for the `getRegisteredUsername(final String loginId)` method.

The Login Page

The default preconfigured login page is represented by the `logon.jsp` file. This `logon.jsp` represents the login page for the user to complete form-based login authentication. By default, the `logon.jsp` file contains the `username` and `password` fields.

However, the `logon.jsp` file can be customized to pass an additional parameter by adding the user_type field. This field determines the type of user who is logging in, that is, internal or external user. The `username`, `password`, and `user_type` (if present) are all passed to the Cúram Java Authentication and Authorization Service (JAAS) login module as part of the authentication process.

The default preconfigured `logon.jsp` file does not have the `user_type` property set. If this property is omitted, the user is assumed to be internal. When this property is set, it indicates that an external user is logging in. This property can be set to any value other than `INTERNAL`.

Customization of the Login Page

The `logon.jsp` file can be customized; that is, the `logon.jsp` file can be replaced by a custom `logon.jsp` file, for a number of reasons.

The reasons the file can be replaced include the following: reasons.

An external user client application is being developed
If an external user client application is being developed, a new `logon.jsp` file needs to be created, as the user type needs to be set to indicate that an external user is logging in. For more information, see "Creating an External User Client Login Page" on page 90.

Automatic login is needed
Some external user client applications require no user authentication and hence a `username` and `password` need not be requested, that is, if an external public access application. It is not possible to disable authentication, so the best way to achieve this requirement is to write an automatic login script. This procedure is done by customizing the `logon.jsp` file for the external public access application. For more information, see "Creating an External User Client Automatic Login Page" on page 90.

Different styling is required
The section on Login Pages in the *Cúram Web Client Reference Manual* for more information on styling for the `logon.jsp` file.

A requirement exists for user names to contain extended characters (valid only for Oracle WebLogic Server)
Web Logic Server provides a proprietary attribute, `j_character_encoding`, which must be added to the `logon.jsp` file. For more information, see "Enabling Usernames With Extended Characters for WebLogic Server" on page 79.

Cúram JAAS Login Module

Authentication is performed by a Java Authentication and Authorization Service (JAAS) login module. It is configured in the application server and is started automatically by the application server as part of the authentication process for any access to the Cúram application. The advantage to this approach is that the default authentication mechanism can be used with, or replaced by, a custom approach, without affecting the Cúram application.

As mentioned earlier, the Cúram JAAS login module can be configured to operate in three modes. For more information on the configuration of the login modules and any application server-specific behavior, see the section on Application Server Configuration within the *Cúram Server Deployment Guide* for the application server that is being used.

Project specific requirements might mean that more than one login module is needed, for example, a user might be required to enter more than the `username` and `password` for verification purposes. It is possible to configure multiple login modules in the application server. Each login module is run in the order as determined by the settings in the application server.

For more information on these settings, see the WebSphere or WebLogic Server documentation.

When the user is authenticated successfully by all login modules that require successful authentication of the user (this login is configurable in the application server), the user is considered authenticated by the application.

Password Management

The passwords for all Cúram internal and external users are stored in their digest format on the `Cúram Users` and `ExternalUsers` database tables. When the Cúram Java Authentication and Authorization Service (JAAS) login module receives the `password`, it is digested before it is sent to the login bean for comparison.

Digesting is a one-way process to ensure the security of the password. The `password` stored for the user on the database uses the same digest algorithm, subject to your encryption settings, ensuring the encrypted passwords can be compared successfully to each other, but remain secure.

Users who are managed externally, for example, through Lightweight Directory Access Protocol (LDAP) with Cúram identity-only configured, are not subject to the process described previously. When a user is being authenticated against a third-party party system (for example, LDAP or a Single sign-on (SSO) Server), where a need exists for the Cúram application to pass the user-entered credentials to the third-party system, the custom implementation of `curam.util.security.PublicAccessUser` can be used. This process allows access to the credentials with a plain-text `password`.

Default Configuration for WebLogic Server

The Cúram Java Authentication and Authorization Service (JAAS) login module is configured as an authentication provider in WebLogic Server. The Cúram authentication provider is the only provider configured by the configuration scripts provided for WebLogic Server. Since it is the only configured authentication provider, the Cúram authentication provider is responsible for authenticating and verifying the user.

As mentioned previously, it is possible there might be more than one authentication provider configured in WebLogic Server. In this case, the Cúram authentication provider might not be responsible for authenticating and verifying the user. For more information, see "Configuring SSO by using Oracle WebLogic Server WL_Token" on page 75.

Default Configuration for WebSphere

The Cúram Java Authentication and Authorization Service (JAAS) login module is configured as a system login module in WebSphere. The default, scripted security configuration within WebSphere involves the default file-based user registry and the Cúram system login module.

Multiple system login configurations exist for WebSphere. The Cúram system login module is configured for the `DEFAULT`, `WEB_INBOUND`, and `RMI_INBOUND` configurations. The same login module is used for all three configurations. WebSphere automatically starts the login modules configured as system login modules under certain circumstances:

- `DEFAULT`

  The login modules that are specified for the `DEFAULT` configuration are started for authentication of web services and JMS invocations. They also are started during the startup phase of WebSphere
- `WEB_INBOUND`

  The login modules that are specified for the `WEB_INBOUND` configuration are used for authentication of web requests
- `RMI_INBOUND`

  The login modules that are specified for the `RMI_INBOUND` configuration are used for authentication of Java clients.

The Cúram JAAS login module exists as a login module within a chain of login modules that are set up in WebSphere. It is expected that at least one of these login modules be responsible for adding credentials for the user. By default, the Cúram login module adds credentials for an authenticated user. As a result of this process, the configured WebSphere user registry that is handled by a subsequent login module does not add credentials.

Therefore, it is not necessary to define Cúram users within the WebSphere user registry. This behavior is configurable by using the `curam.security.user.registry.enabled` property set in the `AppServer.properties` file. For more information on setting this property, see *Cúram Deployment Guide for WebSphere Application Server* or *Cúram Deployment Guide for WebSphere Application Server on z/OS*.

This figure illustrates the default authentication flow for WebSphere.

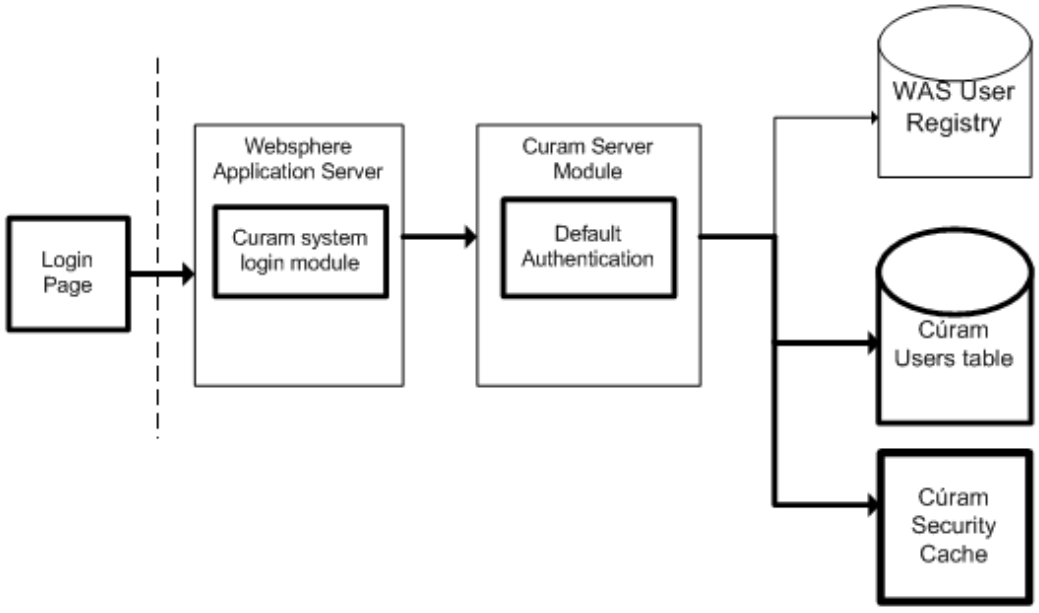

Figure 3. Default authentication flow for WebSphere

This figure illustrates the authentication flow for WebSphere where its user registry is also queried, that is, where the `curam.security.user.registry.enabled property` is set to `true`.

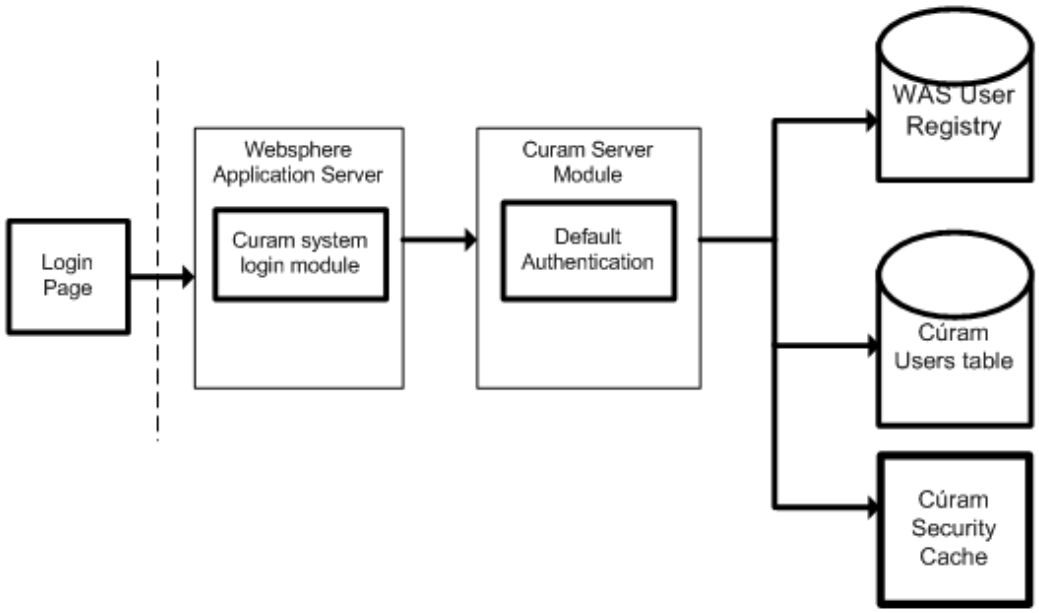

Figure 4. Authentication Flow for WebSphere with User Registry Enabled

As part of the security configuration, certain users exist that are excluded from authentication and for these users the configured user registry is queried. This list of users is configured automatically to be the WebSphere security user, as specified by the `security.username` property in `AppServer.properties` and the database user, as specified by the `curam.db.username` property in `Bootstrap.properties` . These two users are classified administrative users and not application users. It is possible to extend this list of excluded users manually. For more information, see the *Cúram Deployment Guide for WebSphere Application Server* and *Cúram Deployment Guide for WebSphere Application Server on z/OS*.

Warning: The `security.username` and `curam.db.username` users are automatically added to the WebSphere file-based user repository by the provided configuration scripts. If the configured WebSphere user registry is not the default, these users must exist in the alternate WebSphere user registry.

Customizing the login module

It is possible that the Cúram Java Authentication and Authorization Service (JAAS) login module might not support the authentication requirements for a particular custom solution. We strongly recommend that when users develop a custom login module, that the Cúram JAAS login module needs to be left in place and used with identity only authentication enabled. However, if deemed necessary, the Cúram JAAS login module can be removed and replaced by a custom solution. If this is the case, Support must be consulted.

Note: While it is possible to remove the Cúram JAAS login module completely, it needs to be noted that users must still exist in the Cúram Users database table for authorization reasons.

The Cúram JAAS login module adds new users to the Cúram Security Cache automatically, and when this Cúram JAAS login module is replaced by a custom JAAS login module, this function no longer is present. If a custom JAAS login module is replacing the Cúram JAAS login module completely, it is the responsibility of the custom JAAS login module to ensure that an update of the Security Cache is triggered when a new user is added to the database.

Verification Process for Authentication

The type of verifications that are performed depends on the authentication mode that is being used.

Authentication is the process of determining if a user is who they say they are. Authentication is needed where a user must be verified in order to access a secure resource on a system.

Form-based authentication is where a user is presented with a form allowing them to enter username and password credentials. These credentials are compared against the credentials stored on the system for this username, if they match the user is considered an authenticated user for the system. For security reasons the password for authenticating a user is stored on the system in a digested form.

The Cúram web client is configured to support form-based authentication, which means that before a user can access any of the web client content, they will be redirected to a login form to authenticate.

The authentication process involves the verification of the username and password, and this is performed by default by a JAAS (Java™ Authentication and Authorization Service) login module. HTTPS/SSL is turned on by default in the web client ensuring the form-based login authentication mode is secure.

The following list shows authentication modes and configurations with details on the verifications completed for each authentication mode.

Default Authentication

Default authentication is part of the initial configuration and this mode of authentication involves verifying the `username` and `password` specified during login against the `Cúram Users` database table. All login information in this case is maintained by the Cúram application.

Default Verification Process

Several verifications are required by the Cúram login module during default authentication. These verifications include queries that include the user name, password, and account information.

The verifications included during the default authentication are:

- `username` and `password`.
- Account and password expiry
- User name synchronization with security cache
- Break-in detection, for example, upper limit on password entry attempts, incorrect user names, password change failures
- Day and time access restrictions - day of the week and time range within the day

The authentication and authorization of user names is case sensitive by default. However, it is possible to disable case-sensitive authentication. If duplicate case insensitive user names exist (for example, caseworker, CaseWorker), authentication fails due to an ambiguous user name. For more information, see "Changing the Case-Sensitivity of the Username" on page 79.

Authentication Attempts

Authentication failures are not reported directly to a client as this reporting would provide extra information to an intruder who is attempting to break into the system. For example, reporting an incorrect password would indicate that the user name is valid.

All authentication attempts (both success and failure) instead are logged in a database table called the `AuthenticationLog`.For more information, see "Analyzing the AuthorisationLog Database Table" on page 85.

Customization of Default Authentication

The default implementation can be customized to use a mutable `login ID` instead of the Cúram `username` and the ability to add extra verifications is available by implementing the custom authenticator.

For more information, see "Custom Verifications" on page 12.

Identity Only Authentication

Identity only verification means that the authentication mechanism only ensures that the user name for the user who is logging in exists on the `Cúram Users` database table. Full authentication must be completed by an alternative mechanism to be configured in the application server.

Authentication can be configured to perform identity-only verification, in place of the default verifications listed in "Default Verification Process" on page 10.

An example of an alternative mechanism is a Lightweight Directory Access Protocol (LDAP) directory server, which is supported as an authentication mechanism by WebSphere Application Server, WebLogic Server, and WebSphere Liberty. Another alternative is to use a Single Sign-On (SSO) Solution for authentication, or to implement a custom login module. For custom application server solutions, the IBM or Oracle documentation needs to be consulted.

With identity-only authentication (as for default authentication), entries are added to the `AuthenticationLog` database table at the end of the authentication process.

For a successful login the following status is used:

- `AUTHONLY`

For a failure scenario, the following status is used:

- `BADUSER`

  This scenario is the only possible failure scenario where a user does not exist.

The `loginFailures` and `lastLogin` fields of the `AuthenticationLog` are not set. This condition is true even if customized verifications are implemented.

When the password expiry information for a user is set (on the `Cúram Users` database table), the password expiry warning is displayed if it is about to expire. With identity-only authentication, this warning is misleading. It is recommended that any fields that relate to the authentication verifications, such as password expiry or account enabled, are not used if identity-only authentication is enabled.

When identity-only authentication is enabled, security is not used for authentication but is still used for authorization purposes. As a result of this requirement, all users who require access to the application needs to still exist in the `Cúram Users` database table, and in the alternative authentication mechanism, for example, Lightweight Directory Access Protocol (LDAP).

Note: Two users must exist in both locations, that is, the `SYSTEM` user and the `DBTOJMS` user. For more information, see "Security for Alternative Clients" on page 19.

For more information on how to configure identity only for an application server, see "Configuring Identity Only Authentication" on page 80.

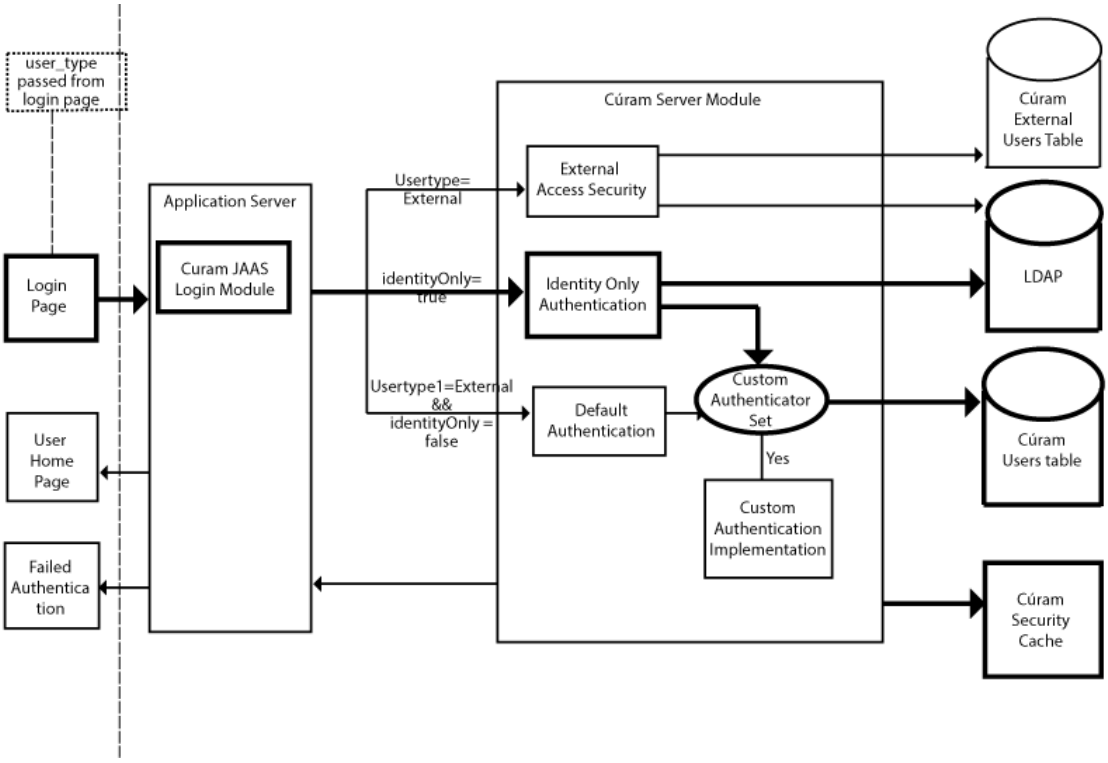

Figure 5. Identity Only Authentication

Customization of Identity Only Authentication

The identity-only implementation cannot be customized, but extra verifications can be added by implementing the custom authenticator.

For more information, see "Custom Verifications" on page 12.

External Access Security Authentication

The architecture allows a developer to implement their own custom authentication solution for external users by providing a hook into the existing authentication and authorization infrastructure.

To hook the custom solution into the application, the `curam.util.security.PublicAccessUser` class must be extended, which requires implementing the `curam.util.security.ExternalAccessSecurity` interface. This class is used during the

authentication and authorization process to determine required information that is related to the External User.

For more information, see "Customizing External User Applications" on page 89.

Custom Verifications

Support is provided for adding custom verifications to the authentication process. For example, a user might be required to answer a security question that must then be verified. The custom code, if implemented, is started after the relevant Cúram verifications or identity assertion, and only if they were successful.

After the custom verifications are started, the authentication process will update the relevant fields on the `Users` database table.

For more information, see "Adding Custom Verifications to the Authentication Process" on page 80.

Authorization Overview

In Cúram, authorization is the process of granting or refusing a user access to functional elements of an application.

The functional element can be anything to which a unique identifier can be attached, such as:

- a server process call,
- an element of the application that requires security checking, for example, a series of registered welfare products.

Access to the functional element is controlled by a Security Identifier (SID) that forms part of the Cúram authorization data. This data is linked to a user and can be configured through the Cúram Administration screens or through the Data Manager. For more information, see the *Cúram Server Developer's Guide*.

The security data that is created for authorization is central to the processing performed during every client-server call, and it is important that access is optimized for performance reasons. The Cúram Security Cache is responsible for caching authorization data for a user. For more information, see "Cúram Security Cache" on page 18.

The following topics describe the relationship for these authorization concepts and how authorization works within Cúram.

Users, Roles and Groups

The security information associated with an application must first be organized into security profiles before it can be utilized in a runtime environment. A security profile consists of a security role, one or more security groups and the associations between security identifiers (SIDs) and securable elements of an application.

Every authorized user is assigned a security role during security configuration and these roles are associated with a number of security groups. Each security group is associated with a number of security identifiers. The security identifier represents the securable elements of Cúram, for example., a method or a field. The role, groups and identifier information is stored on the database in a number of tables and is configured using the application Data Manager or the Cúram Administration screens.

This data structure makes it possible to authorize every user against any secured element of an application. This is a powerful and flexible method of providing authorization to Cúram users.

There is a minimum set of SIDs required for a user to operate the Cúram Platform application. These SIDs are associated to the out-of-the-box BASESECURITYGROUP group. The `EJBServer/components/core/data/initial/handcraftedscripts/Supergroup.sql` file should be consulted to identify the list of these SIDs. This file is responsible for linking the SIDs to the BASESECURITYGROUP out-of-the-box.

A simple way to ensure that all users have the privileges from this set of SIDs is to create a single security group for them and then associate that security group with every security role in the system.

Security Identifiers (SIDs)

Every secured element in Cúram is given a security identifier (SID) that is unique across the entire application.

The authorization process is built into the infrastructure and once the securable elements have been identified, the rest is handled by code generators, scripts and the Cúram Administration screens. The analysis of what elements must be securable is a manual process that must be done by the developer or security administrator. This section outlines the infrastructure available to set up authorization.

The first type of authorization to consider is that of the process method(facade) also known as *function-level security* . In the Cúram model, a developer may choose if security is switched on or off at the process method level. The option applies only to Business Process Objects (BPOs) since they encapsulate the calls exposed to the client. Entity object methods are not included in the authorization process.

There are a number of types of SIDs and these include:

- Function Identifiers (FIDs)
- Field Level Security Identifiers
- User defined SID types.

Function Identifiers (FIDs)

Function identifiers (FIDs) are a specialized type of security identifier (SID) where the type is set to FUNCTION. When a method is made publicly accessible (by setting the stereotype as facade in the model), a FID is generated for that method and security is automatically turned on.

It is possible to turn off security for a process method at design time. For more information, see “Switching Security off for a Process Method” on page 84.

Adding an FID

To add an FID, do the following steps:

1. Log on as the `sysadmin` user and click **System Configurations**.
2. In the Shortcuts panel, click **Security** > **Identifiers**.
3. In the actions menu, click **New Function Identifier** and enter the details for the FID.
4. In the actions menu, click **Publish**.
5. In the Shortcuts panel, click **Security** > **Groups**.
6. Click a group to add the FID to, and then click **Add Identifiers**.
7. From the list of alphabetically ordered identifiers that is displayed, select the identifier that your created and click **Save**.
8. Click **Publish**.

Field Level Security Identifiers

The Field Level SID allows authorization to be applied to specific fields on a publicly accessible method. At runtime, if a user does not have access rights to view the field to be displayed, the contents of the field are displayed as a number of asterisks (***). For more information on Field Level SIDs , the *Cúram Modeling Reference Guide* should be consulted.

User Defined SIDs

In the previous sections, we have described

FIDs;
An automatically generated SID of type function.

Field Level SID;
Security applied to specific fields on a method.

There is also the concept of a user defined SID. The authorization process is sufficiently flexible to accommodate any securable element of an Cúram application. The developer can effectively customize the authorization process by defining new *types* of SIDs. The new types represent a conceptual element requiring security. The following server interface method enables authorization to be invoked directly on these new user defined SID types.

`curam.util.security.Authorisation.isSIDAuthorised()`

Out-of-the-box, the LOCATION and PRODUCT SIDs are SIDs of this type. Using the above method there is effectively no limit to the SID types that can be defined. "Authorizing New SID Types" on page 84 should be consulted for further details.

Runtime Authorization

The Cúram infrastructure performs authorization checks from both the web client and server side.

Client Authorization Checks

Before a user can access a method or field, the web client performs authorization checks before the page is initially loaded. If the user does not have access, the client authorization check fails, and the server is not invoked. This check is configurable in the `curam-config.xml` by setting the SECURITY_CHECK_ON_PAGE_LOAD property. Section 3.12.13 General Configuration in the *Cúram Web Client Reference Manual* should be consulted for further details on this.

By default any such web client authorization failures are not recorded. This behavior is configurable. "Controlling the Logging of Authorization Failures for the Client" on page 84 should be consulted for further details.

Server Authorization Checks

To cater for other access to Cúram, and where the web client authorization check is disabled, there is a second level authorization check made by the server. This server side check will always log authorization failures, and the client property does not affect this logging.

The log of all authorization failures is stored on the database to allow these failures to be audited at a later stage. The AuthorisationLog table contains the User Name and Security Identifier for the failed authorization, as well as a timestamp indicating when the failure occurred. "Analyzing the AuthorisationLog Database Table" on page 85 should be consulted for further details on the AuthorisationLog table.

Cryptography in Cúram

In Cúram, cryptography refers broadly to ciphers and digests, two types of functionality that are related to keeping your Cúram systems safe and secure.

You can use ciphers and digests as follows in Cúram:

- ciphers - for two-way encryption of passwords that are used at various processing points
- digests - for one-way hashing (or digesting) of passwords; for example, used at login

You can select the values for configuring cryptographic behavior with the `CryptoConfig.properties` property file to provide you with the most control and security possible for your Cúram installation. This flexibility provides the capability to adjust to changing security standards. For more information about configuring and customizing cryptography, see "Customizing Cryptography" on page 85.

If you are migrating for the first time to a level of Cúram that has this level of cryptographic support, which was introduced in version 6.0.5.0, it is recommended that you upgrade system (new cipher) and user (new digest) passwords from the default values to improve your security.

Supported cryptographic configurations are:

1. AES: 128, 192, 256 (FIPS 140-2 and SP800-131a compliant);
2. Two-key Triple DES - DESede: 112 (FIPS 140-2 compliant);
3. Three-key Triple DES - DESede: 168 (FIPS 140-2 and SP800-131a compliant);
4. No cryptography configuration, which is configured by removing the `CryptoConfig.properties` file, in which case Cúram reverts to its previous default crypto settings.

In the environment where Cúram runs, the application server, database, and other software, such as web server or LDAP software, has its own cryptographic support and you can refer to the relevant vendor's documentation.

Ciphering

Ciphering refers to the process of encrypting passwords, which are listed in "Cipher-Encrypted Passwords" on page 17. That is, this is a two-way process representing decrypt-able values. There are about a dozen of these encrypted passwords in various property files in Cúram and encrypting them helps keep them secure and they are are decrypted at the necessary points for usage; e.g. connecting to your database system.

Digesting

Digesting refers to the one-way process of handling passwords that do not require decrypting, but is used for storing passwords for later comparison; e.g. Cúram user logins. That is, this is a one-way process representing non-decryptable values.

Cryptography Properties

The Cúram `CryptoConfig.properties` file contains settings for cipher and digest cryptography. Therefore, this file and all the files it refers to (i.e., keystore and salt) should be considered critical items to the security of your system and should be provided with adequate access controls (e.g., file permissions) and specifically modified and segregated when used for production systems. That is, if the details of these files were to become widely known, while not necessarily a security risk themselves, would remove a level of protection that might necessitate a disruptive crypto change (see "Cipher Customization" on page 85 and "Digest Customization" on page 87).

Related topics:

- "Cúram Cipher Settings" on page 15
- "Cúram Digest Settings" on page 16

Cúram Cipher Settings

Various passwords within Cúram property files and configurations are stored in an encrypted format out-of-the-box (OOTB).

The Cúram crypto configuration will work for you out-of-the box, but it is recommended you modify these settings with respect to your local security requirements. For instance, the OOTB settings may be adequate in development, but for production environments it is strongly recommended that you modify them (e.g. by changing the cipher secret key).

The cipher settings are stored in the `CryptoConfig.properties` file. The properties and their values are as follows:

- `curam.security.crypto.cipher.algorithm`

  - **Valid values:** In JCE documentation, for example: http://docs.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html#Cipher. The supported ciphers are AES and the various forms of Triple DES.
  - **Default:** AES (FIPS 140-2 and SP800-131a compliant)

- `curam.security.crypto.superseded.cipher.algorithm`
  - **Valid values:** See `curam.security.crypto.cipher.algorithm`
  - **Default:** None
  - **Purpose:** Provides for flexibility to support an upgrade/migration period for Cúram user passwords with custom code (e.g. a batch program) via the `curam.util.security.EncryptionUtil.decryptSupersededPassword()` API. The use of an upgrade/migration period is explained in more detail in "How to Utilize the Superseded Digest Settings for a Period of Migration" on page 88.

- `curam.security.crypto.cipher.keystore.location`
  - **Valid values:** Path to keystore file containing secret key. This can be an absolute path specification or relative to the classpath (e.g. `CuramSample.keystore`).
  - **Default:** None

- `curam.security.crypto.cipher.keystore.storepass`
  - **Valid values:** As per the JDK **`keytool`** command.
  - **Default:** password
  - **Purpose:** Specify the password used to access the keystore.

- `curam.security.crypto.cipher.provider.class`
  - **Valid values:** Fully-qualified name of a JCE cryptography provider class.
  - **Default:** blank
  - **Purpose:** Optional way to enable the use of an alternate standards-compliant provider.

This ciphering functionality applies to the properties as described in "Cipher-Encrypted Passwords" on page 17.

These Cúram cryptographic settings are enabled by default OOTB and represents changes that existing Cúram installations must address as documented in the *Cúram Upgrade Guide*.

Cúram Digest Settings

Cúram users, internal and external, when not invoked with identity-only, are authenticated using form-based login and the password entered in the form is digested and compared to the digest value stored in the database for the user.

Note: This processing does not apply to users authenticated in third party systems like LDAP.

The Cúram crypto configuration will work for you out-of-the box, but it is recommended you modify these settings with respect to your local security requirements. For instance, the OOTB settings may be adequate in development, but for production environments it is strongly recommended that you modify them (e.g. digest salt encrypted value).

The digest settings are stored in the `CryptoConfig.properties` file. The properties and their values are as follows:

- `curam.security.crypto.digest.algorithm`

  - **Valid values:** In JCE documentation, for instance: http://docs.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html#MessageDigest. The supported digests are the SHA variants (1, 256, etc.) and MD5.
  - **Default:** SHA-256 (FIPS 140-2 and SP800-131a compliant)
  - **Purpose:** Specification of the digest algorithm.

- `curam.security.crypto.digest.salt.location`
  - **Valid values:** A path identifying the file containing the encrypted secret digest salt.
  - **Default:** None
  - **Purpose:** An optional file to specify the salt (encrypted) for digesting.

- `curam.security.crypto.digest.iterations`
  - **Valid values:** 0 or a positive integer.
  - **Default:** 0
  - **Purpose:** Typically, higher values give better security, but at the cost of processing (e.g. at login time).

There are a set of corresponding "superseded" properties to allow for flexibility when migrating from one set of digest settings or standards to another. The following have a similar function to their counterparts above, but are used by the Cúram encryption functionality to support both old and new settings for a time of migration:

- `curam.security.crypto.superseded.digest.algorithm`
- `curam.security.crypto.superseded.digest.salt.location`
- `curam.security.crypto.superseded.digest.iterations`

The usage and behavior of the superseded properties are controlled by the `curam.security.convertsupersededpassworddigests.enabled` property as managed by the Properties Administration user interface. See "How to Utilize the Superseded Digest Settings for a Period of Migration" on page 88 for more information on using the superseded properties.

Cipher-Encrypted Passwords

The following passwords are cipher-encrypted in Cúram:

- `Bootstrap.properties`:
  - `curam.db.password` - database password
  - `curam.searchserver.sync.password` - see *Cúram Generic Search Server* for more information
- `AppServer.properties`: (typically this property file is used for configuring test servers and is not appropriate for production systems)
  - `security.password` - application server administration console password
  - `curam.security.credentials.async.password` - replacing the `runas.password property`
- `Application.prx` - individual property descriptions are as documented with the properties in the Curam Property Administration user interface:
  - `curam.security.credentials.dbtojms.password` - (in conjunction with `curam.security.credentials.dbtojms.username`), which replaces the curam.omega3.DBtoJMSCredentialsIntf interface APIs previously used to provide custom credentials for DB-TO-JMS
  - `curam.security.credentials.ws.password` (in conjunction with `curam.security.credentials.ws.username`), which replaces the build-time default web services default credential settings.
  - `curam.meeting.request.reply.password` - (an SMTP password)

  - `curam.ldap.password`
  - `curam.citizenworkspace.password.protection.key`
- `BIBootstrap.properties` - BIRT users only; see the *Cúram Business Intelligence BIRT Developer Guide*:
  - `curamsource.db.password`
  - `central.db.password`
  - `centraldm.db.password`
- Web Services - See the *Cúram Web Services Guide*:
  - `ws_inbound.xml` - `<ws_service_password>`
  - `services.xml` - `<parameter name="jndiPassword">`
- CTM - Cúram Transport Manager:
  - The Password column of the TargetSystemService table contains an encrypted password

Security Data Caching

An overview of the Cúram Security Cache, which stores all authorization data for a user. Details on the WebSphere cache and how this affects the authentication of a user at login are also included.

Cúram Security Cache

Security information from the database tables supporting the profiles mentioned in "Users, Roles and Groups" on page 12 is cached by the infrastructure. This is done to optimize the search and retrieval of data during the authorization process.

To optimize performance, the cache is loaded on demand as security authorization requests come into the application and is a shared resource. For application code, the cache is a protected resource and cannot be accessed directly. It is accessible, for queries only, through the authorization interface (`curam.util.security.Authorisation`) which allows a developer to implement a customized authorization procedure. "Authorizing New SID Types" on page 84 should be referenced for further details on this.

When the curam.security.casesensitive property is set to false the security cache will store all usernames in upper case and all queries to the cache will automatically change the specified username into the upper case equivalent. It is also worth noting that the existence of duplicate case insensitive usernames will cause a fatal error during the initialization of the security cache. "Changing the Case-Sensitivity of the Username" on page 79 should be consulted for further details on this.

Cache Refresh

As security data is so important to the operation of Cúram , the cache must be refreshed whenever any changes have been made to security related database tables. The refreshing of the Cúram Security Cache is an asynchronous process.

Cache Refresh Failure

The refreshing of the Cúram Security Cache is triggered by either an application reboot, or by the system administrator (sysadmin) via the Cúram Administration screens, therefore, the administrator receives no feedback if the cache reload fails. Having to check the system logs or manually verify the application following a refresh to verify its success can be cumbersome. It is therefore recommended that the optional callback interface for providing feedback in the event of a cache reload failure be implemented. "Adding the Cache Refresh Failure Callback Interface" on page 80 should be consulted for further details.

WebSphere Caching Behavior

WebSphere caches user information and credentials in its own security cache. The Cúram login module will not be invoked while a user entry is valid in this cache. The default invalidation time for this security cache is ten minutes, where the user has been inactive for ten minutes.

For example, the first time a user logs into the application from the web client they will be requested for their username and password. The Cúram login module will be invoked, and will authenticate the information specified. If the same user opens a second new web browser and attempts to access the application, they will again be requested for their username and password. When WebSphere receives this information it will query the security cache to determine if the username and password are already in the cache. If they are, and the password matches, WebSphere will not query the login modules.

The impact of this behavior is that any modifications to a user's account restrictions or password will not take effect until the user has been invalidated from the WebSphere security cache.

For more information see the appropriate *WebSphere Application Server Information Center*.

Security for Alternative Clients

Certain processes cannot be associated with a specific logged-in user. These include alternative clients, for example, non-web processes such as batch processing, web services, and deferred processing. As any process that interacts with a Cúram application must be authenticated, a valid user must exist for each of these processes. These topics provide details on the users that must exist on the Cúram Users table and details on the processes that depend on these users.

Mandatory Cúram Users

A number of users must always exist in the Cúram Users database table. These users are necessary for application processes such as deferred processing and workflow. If these users do not exist, then authentication will fail and subsequently these processes will fail.

The usernames and passwords for each of the processed below are the default out-of-the-box credentials and it is recommended that these credentials be changed for security reasons.

These users include:

- SYSTEM

  The SYSTEM user is the user under which JMS messages are executed. This user must exist and the username is case sensitive. “JMS Messaging” on page 20 should be referenced for further details.

- DBTOJMS

  The DBTOJMS user is the default user under which the Database to JMS (DBToJMS) trigger for batch processing is executed. This user must exist and the username is case sensitive. “Batch Processing” on page 20 should be referenced for further details.

- WEBSVCS

  The WEBSVCS user is the default user under web services are executed. This user must exist and the username is case sensitive. “Web Services” on page 19 should be referenced for further details.

Web Services

For Apache Axis2 (the recommended implementation for web services) there are default credentials for authentication. A user has the ability to change these credentials at a global level or per service if required. To ensure that web services are not vulnerable to a security breach this default user is not authorized to access web services by default. For authorization, a web service must be associated with a security group and in turn a security role that is linked to the user (e.g. WEBSVCS) in order to access it. Ensuring the user is authorized is a manual process. Please see the *Customizing Receiver Runtime Functionality* section in the *Cúram Web Services Guide* for further details on web services and also the chapter on Authorization in this book.

There are a number of other topics related to the security of web services - for example, encrypting data - using Rampart. The *Cúram Web Services Guide* should be consulted for further details on these.

Batch Processing

Since the Batch Launcher does not require the application server to be running, it does not perform any application level authentication or authorization. It must only authenticate against the database. The same credentials as used by the application server (located in `%SERVER_DIR%/project/properties/Bootstrap.properties`) are used by the Batch Launcher to connect to the database and run batch programs.

The Batch Launcher or batch programs can optionally trigger the application server to begin a DB-to-JMS transfer. This involves logging in and invoking a method on the server, which in turn requires a valid username and password. By default the DB-to-JMS transfer operation uses default credentials; therefore, the DBTOJMS account must exist on the Cúram Users table and must be enabled and assigned the role 'SYSTEMROLE' to allow authorization. The locale DB-to-JMS transfer is the default locale for this user as specified in field 'defaultLocale' on the Users table.

The Security Considerations section in the *Cúram Batch Processing Guide* guide should be consulted for further details on changing the user for the DB-to-JMS transfer.

The property batch.username can be used to specify the user name for the operations run by the Batch Launcher. This is set using the -D parameter. For example: build runbatch -Dbatch.username=admin

JMS Messaging

JMS messages are used for communication purposes by deferred processes and Workflow. Since JMS messages are triggered by the application server and need to interact with the Cúram application, valid Cúram credentials must exist. The SYSTEM user account must exist on the Cúram Users table and must be enabled and assigned the role 'SYSTEMROLE' to ensure authorization. The locale for JMS messages is the default locale for this user as specified in field 'defaultLocale' on the Users table.

It is possible to change the SYSTEM username during or after the deployment of the application. For more information the *Cúram Server Deployment Guide* for the relevant application server should be consulted.

Deferred Processing

A deferred process in Cúram is a business method that is invoked asynchronously. As deferred processes interact with the application, valid Cúram credentials must exist. The SYSTEM user account must exist on the Cúram Users table and must be enabled and assigned the role 'SYSTEMROLE' to ensure authorization. The locale for deferred processes is the default locale for this user as specified in field 'defaultLocale' on the Users table. In the case of offline unit-testing of deferred processes, the username is blank and the effective locale is the default locale for the Cúram server.

External User Applications

Typically, there are users outside the organization with limited access who needs to securely access parts of the Cúram application. These users are considered external users and authentication for these users is completely customizable through the use of the External Access Security hook point provided. As external users are processed differently to internal users, a specific web application is required for external users.

The default Cúram application is enabled for internal users. Internal users are users that exist on the Cúram Users database table. A typical internal user would be a case worker who creates and manages claims for participants and has full access to the application. The infrastructure provides functionality for authenticating and authorizing these internal users.

External User Applications

When developing an application for an external user, the following must be implemented:

- An external user client application, i.e., a separate EAR file containing the web client application.

- A custom `logon.jsp` , where the external application must pass in a parameter user_type indicating an external user is logging in.
- A custom class that extends `curam.util.security.PublicAccessUser`, which requires implementing the `curam.util.security.ExternalAccessSecurity` interface, must be provided. This abstract class contains methods responsible for the authentication and authorization of an external user.

As well as there being internal and external user types. There can also be different types of external users. For example, there may be an external user of type 'PUBLIC' who could have limited access to an external application. There could be another external user of type 'PROVIDER' who is a registered external user. The ability to have different types of external users provides more flexibility within an external application, allowing finer grained control over authentication of the external user based on the external user type.

User Scope

There are two different types, or scopes, of users within the Cúram application: internal and external. The type of a user is determined in one of the following ways:

- By the Cúram Security Cache;

  If the user exists in the Cúram Security Cache, the type is assumed to be in internal. If the user does not exist in the cache, the type is assumed to be external. In this case, (which is the default behavior) all usernames, internal and external, must be unique.

- By the `UserScope` custom interface;

  If the `UserScope` custom interface is implemented. This custom interface, takes precedence over the check for a user in the Cúram Security Cache to determine the user type. Consult "Determining if a User is Internal or External using the UserScope Interface" on page 96 for further details.

When the type of a user is external the implementation of the `curam.util.security.ExternalAccessSecurity.getSecurityRole()` method will be used to determine the user role instead of the internal security roles. "Authorizing an External User" on page 93 should be consulted for further details on this method.

To support alternative methods for determining if a user is internal or external the custom interface, `UserScope` , is available. Consult "Determining if a User is Internal or External using the UserScope Interface" on page 96 for more details.

Deployment of an External Application

When deploying an application to an application server, the security configuration for the application server is applicable to all Cúram applications deployed to that application server instance. Therefore, care must be taken when considering the deployment architecture for more than one application. This is important when deciding if an internal and external application will be deployed to the same application server instance.

An example of some considerations to think about are:

- Is identity only being used for internal users?
- Is an alternative authentication mechanism used , e.g., LDAP;
- Will both internal and external users be authenticated by LDAP?

The answers to the considerations above will affect the setting of the application server properties (i.e. properties specified in the `AppServer.properties` file), that affect the behavior of the Cúram JAAS login module. These considerations will also drive the implementation of the `curam.util.security.PublicAccessUser` class and `curam.util.security.ExternalAccessSecurity` interface for external users.

The application server properties in the Cúram JAAS login module allow for finer grained control over the authentication of user types. External users and internal users can be authenticated differently, as can

different types of external users, in a situation where the internal and external applications are deployed to the same application server. These properties include the following:

- curam.security.user.registry.disabled.types ;

  Set this property to a comma separated list of user types for which the application server user registry *will not* be queried, i.e. the implementation within the `PublicAccessUser.authenticate()` method is responsible for authenticating the external user of this type. For example, LDAP could be configured to be the user registry.

- curam.security.user.registry.enabled.types.

  Set this property to a comma separated list of user types for which the user registry *will* be queried, i.e., the implementation within the `PublicAccessUser.authenticate()` method does not have to fully authenticate the user. The user registry will be responsible for authenticating this type of external user. For example, LDAP could be configured as the user registry, and in this case, LDAP could be responsible for the authentication of these external user types.

These properties are dependent on the implementation of the `curam.util.security.PublicAccessUser` class and `ExternalAccessSecurity` interface.

Consider the following example project requirements:

- An internal user must authenticate with LDAP.
- An external user of type 'EXT_PUBLIC' must authenticate with Cúram and not LDAP;
- An external user of type, 'EXTERNAL' must authenticate with LDAP only and not Cúram.
- Both the internal and external applications are deployed to the same application server instance.

The following settings could cater for the example above:

- curam.security.check.identity.only set to `true` ;
- curam.security.user.registry.disabled.types=EXT_PUBLIC.

As well as the properties being set, the `PublicAccessUser` extension (and `curam.util.security.ExternalAccessSecurity` implementation) must have the logic to cater for the different types of external users and how they will be authenticated.

Configuring Single Sign On (SSO)

SSO is an authentication process that allows users to access many applications with one set of credentials. Once users log into one application, they do not have to log in repeatedly to access other applications that are part of one application domain.

SSO systems usually maintain the user accounts on a lightweight directory application protocol (LDAP) server. If user accounts are stored in one location, it is easier for system administrators to safeguard the accounts. Also, it is easier for users to reset one account password for multiple applications.

Note: The implementation of an SSO solution is the responsibility of the customer. It is recommended that an IBM or third-party tool is used. For example, IBM Tivoli tools or CA SiteMinder.

IBM Cúram Social Program Management provides two mechanisms to implement SSO:

Configuring SAML SSO

Configure SAML-based SSO to support IBM Cúram Social Program Management, IBM Cúram Universal Access, and IBM Cloud Kubernetes Service.

Configuring SAML SSO on Kubernetes

Implement federated SSO that uses SAML 2.0 browser profile, using either an IdP-initiated HTTP POST binding or an SP-initiated HTTP POST binding, through the IBM Cúram Social Program Management application.

The following information describes the scenario where IBM Cúram Social Program Management is deployed on WebSphere Application Server Liberty.

Note: The sample configuration in this section uses IBM Security Access Manager (ISAM) to show you how to configure SSO. You can use any SAML-compliant configuration tool in your SSO configuration.

Social Program Management SSO on Kubernetes initiation and flow

For single sign-on, the SAML response, by HTTP POSTs, is interpreted and controlled by logic in IBM Cúram Social Program Management.

In all SAML web SSO profile flows, the binding defines the mechanism that is used to send information through assertions between the identity provider (IdP) and the service provider (SP). WebSphere Liberty supports HTTP POST binding for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML response document is an XML document that contains data about the user and the assertion, some of which is optional.

Browser-based single sign-on (SSO) through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IdP) and the service provider (SP). The user is guided seamlessly from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single-page application, all the screens are contained within the application and dynamic content is expected to be passed only in JSON messages through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it would to leave the application and hand back control to either the user agent or the browser, in which case the application state would be lost.

IdP-initiated use case

The IdP can send an assertion request to the service provider ACS through one of the following methods:

- The IdP sends a URL link in a response to a successful authentication request. The user must click the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.
- The user authenticates into IdP and accesses the application that is configured as a partner to the IdP.

The ACS then validates the assertion, creates a JAAS subject, and redirects the user to the SP resource.

SP-initiated use case

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML `AuthnRequest` from the SP to the IdP. The IdP receives the `AuthnRequest`, validates that the request comes from a registered SP, and then authenticates the user. When the user is authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the `AuthnRequest` that was received from the SP.

Assertions and the SAML Response document

To prove the authenticity of the information, the assertion in the SAML response is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted. A typical SAML response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server's cookie tied to the user's HTTP session.

IdP-initiated flow

When Social Program Management is configured in WebSphere Liberty with an-IdP initiated web SSO flow, any attempt to connect to a protected resource without first authenticating through IdP results in the application server falling back to an SP-initiated SSO flow. In an SP-initiated SSO flow, any authentication requests that are initiated through SP result in a 403 HTTP response, and the application redirects the user to the IdP login page for the user to authenticate. After the user is authenticated successfully, the control is redirected to the Social Program Management application page.

The following figure illustrates the IdP initiated flow that is supported by Social Program Management in a default installation.

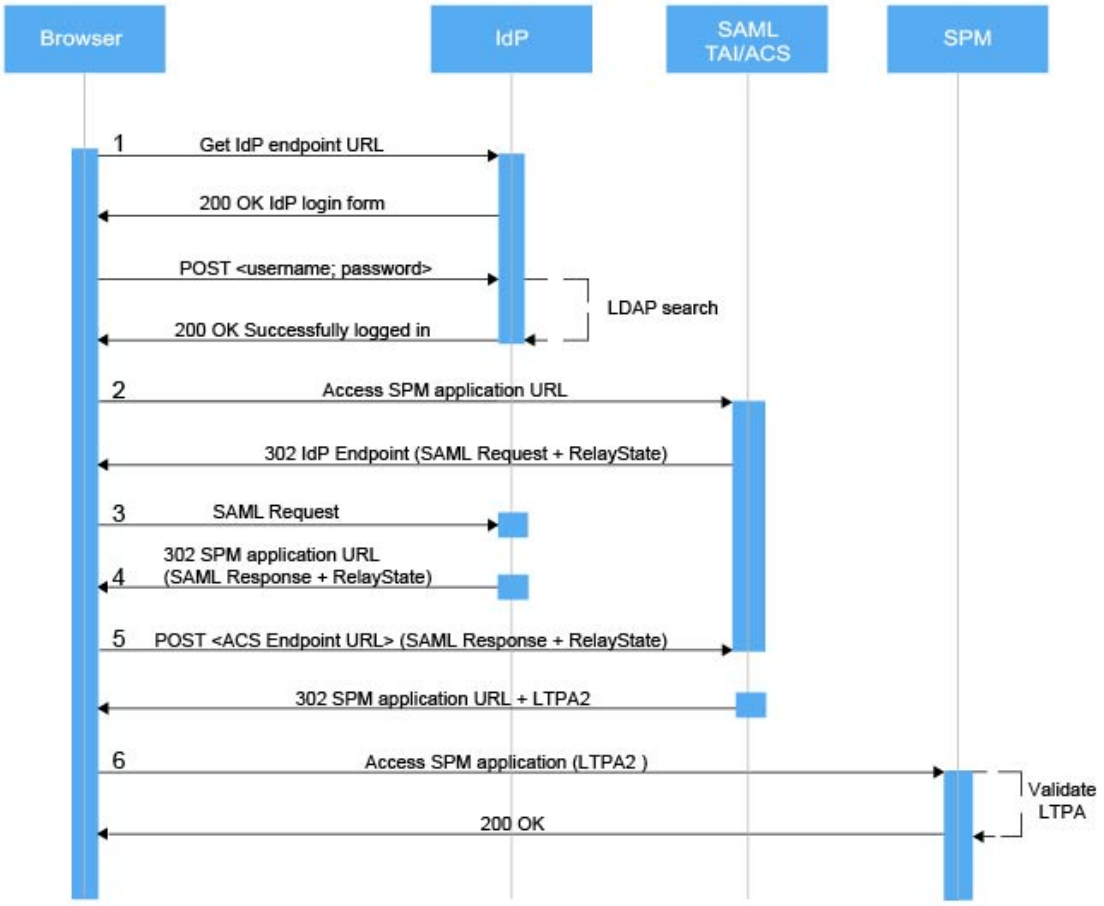

Figure 6. IdP-initiated flow

1. In an IdP-initiated flow, the user completes the IDP login form and authenticates.

2. After successful authentication in IdP, the user tries to access the Social Program Management application that is deployed in the application server.
3. The Trust Association Interceptor (TAI) and Assertion Consumer Service (ACS) (SAML TAI/ACS) that is deployed on the application server intercepts the request and redirects it to the IdP endpoint with a generated SAML request.
4. Because the user is already logged into the IdP, the IdP responds with a SAML response and redirects the user to the Social Program Management application.
5. The application server ACS validates the signature that is contained in the SAML Response. If the validation is successful, the ACS sends an HTTP redirect request that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

SP-initiated flow

When Social Program Management is configured with an SP-initiated web SSO flow, any attempt to connect to a protected resource without first authenticating results in a 401 HTTP response from the application server Assertion Consumer Service's Trust Association Interceptor, and the generation of the SAML AuthnRequest message to be sent to the IdP.

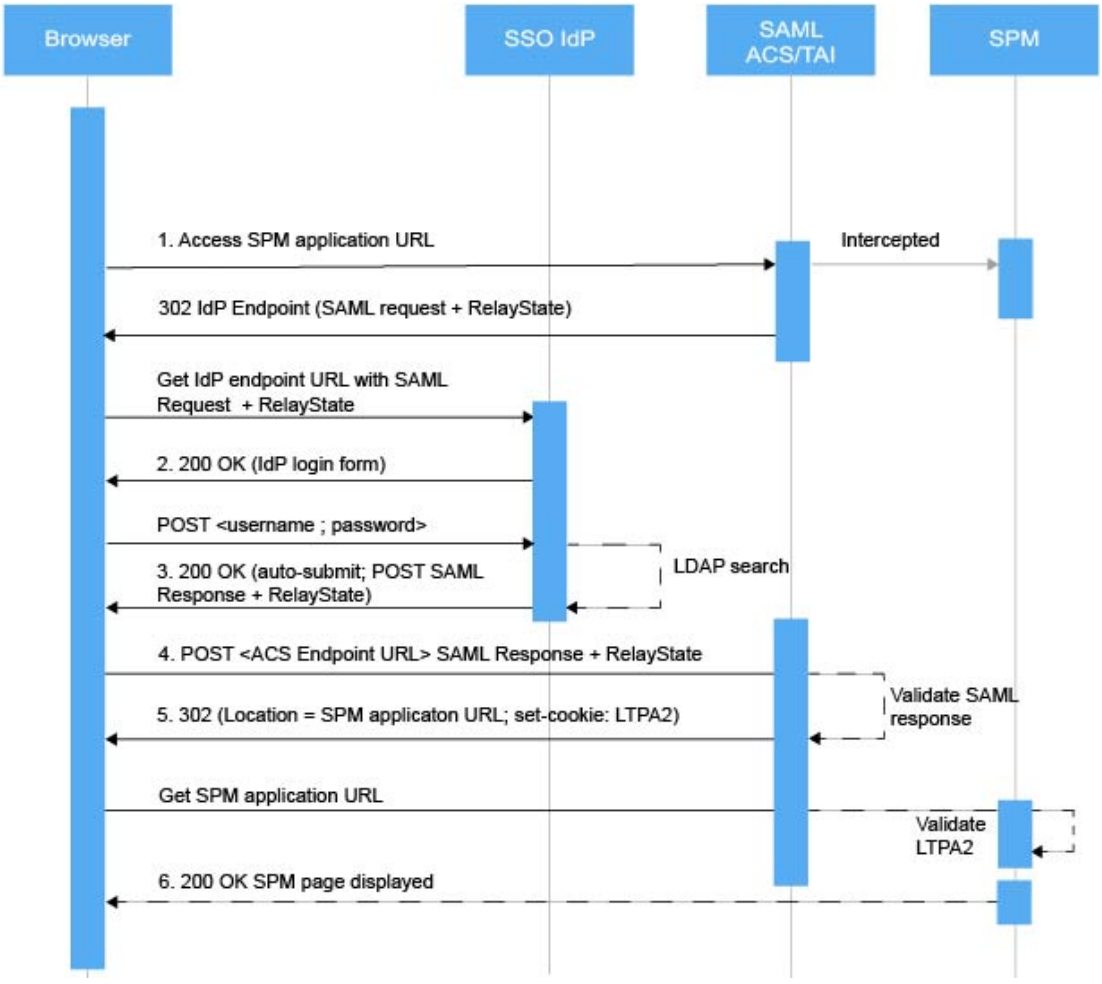

Figure 7. SP-initiated flow

1. When a user tries to access a Social Program Management application resource without authenticating, the TAI intercepts the request and redirects the user to the IdP endpoint with the generated SAML request.

2. The IdP endpoint displays the login form that the user completes to authenticate, then directs the SAML request to the IdP SAML endpoint.
3. After successful validation of the user credentials at the IdP, the IdP populates the SAML response and returns it in an HTML form that contains hidden input fields.
4. The HTML form is autosubmitted to the Social Program Management application with the SAML response and RelayState parameter.
5. The application server ACS validates the signature that is contained in the SAML response. If the validation is successful, the ACS sends an HTTP redirect that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

IdP-initiated flow for Universal Access

The following figure illustrates the IdP initiated flow that is supported by Universal Access in a default installation.

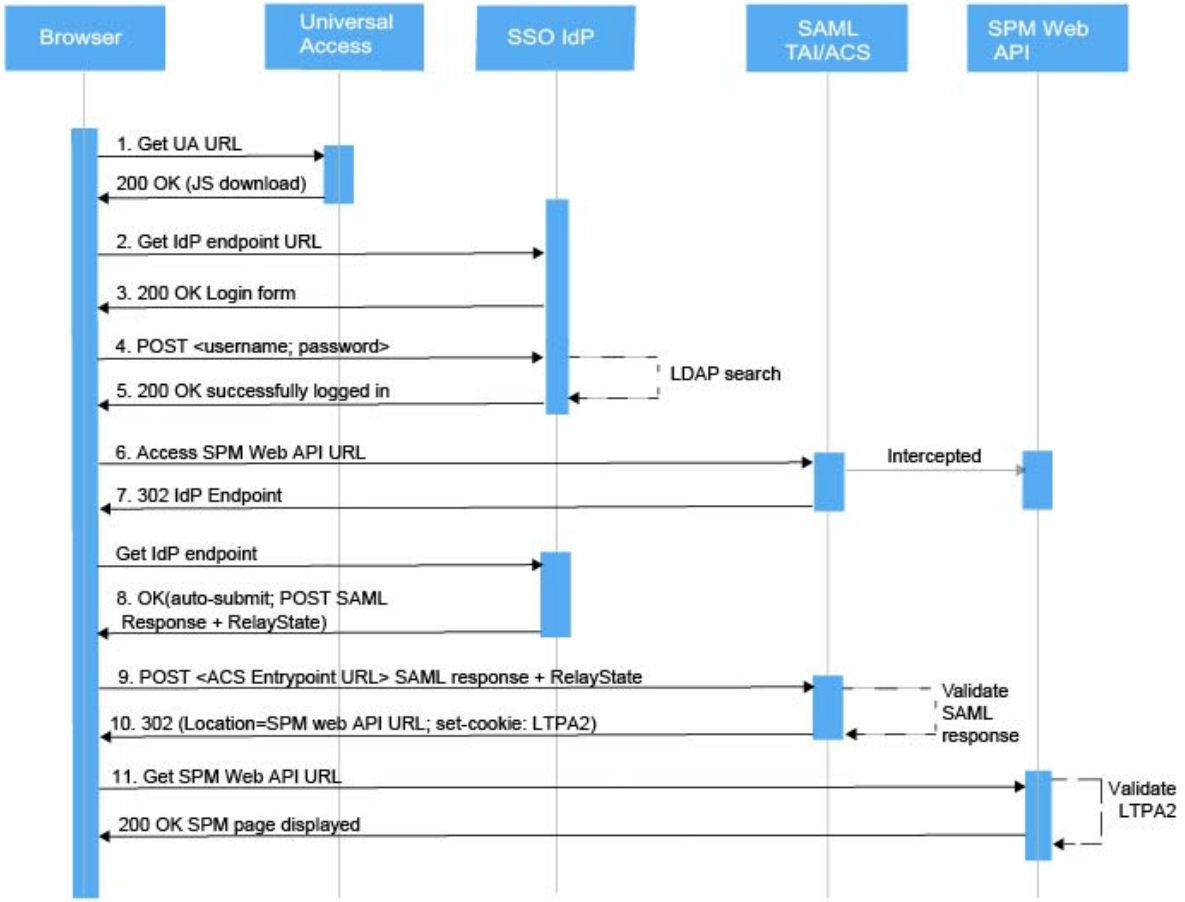

Figure 8. IdP-initiated flow for Universal Access

1. In an IdP-initiated flow, the user completes the IDP login form and authenticates.
2. After successful authentication in IdP, the user tries to access the Social Program Management application that is deployed in the application server.

3. The Trust Association Interceptor (TAI) and Assertion Consumer Service (ACS) (SAML TAI/ACS) that is deployed on the application server intercepts the request and redirects it to the IdP endpoint with a generated SAML request.
4. Because the user is already logged into the IdP, the IdP responds with a SAML response and redirects the user to the Social Program Management application.
5. The application server ACS validates the signature that is contained in the SAML Response. If the validation is successful, the ACS sends an HTTP redirect request that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

Configure SAML SSO for IBM Cúram Social Program Management on WebSphere Liberty
Code samples and steps are provided as a guide for enabling SAML SSO in WebSphere Liberty.

About this task

The following code samples and steps are intended for general guidance only. They are not intended to be a substitute for detailed analysis or the exercise of professional judgment.

For more information about configuring SAML SSO for IBM Cúram Social Program Management on WebSphere Liberty, see the related link.

Procedure

1. Enable the SAML feature in the WebSphere Liberty `server.xml` file, as shown in the following example:

```
<featureManager>
    <feature>samlWeb-2.0</feature>
    <feature>appSecurity-2.0</feature>
</featureManager>
```

2. Download the SAML metadata XML and ask your SSO administrator to use it to configure the SSO provider as IBM ISAM, as shown in the following example:

```
https://application-domain.com/ibm/saml20/defaultSP/samlmetadata
```

3. Configure and enable SAML SSO in the WebSphere Liberty `server.xml` file, as shown in the following example:

```
<server description="Curam Server">
(...)
<samlWebSso20 id="defaultSP"
    idpMetadata="/path/to/file/federation_metadata.xml"
    wantAssertionsSigned="false"
    authnRequestsSigned="false"
    authFilterRef="curamAuthFilter"
    spHostAndPort="https://application-domain.com"
    disableLtpaCookie="false"
    allowCustomCacheKey="false"
    enabled="true">
</samlWebSso20>
<authFilter id="curamAuthFilter">
    <requestUrl id="curamRequestUrl1" urlPattern="/Curam/j_security_check"
matchType="notContain"/>
    <requestUrl id="curamRequestUrl2" urlPattern="/Curam/logon.jsp" matchType="notContain"/>
    <requestUrl id="curamRequestUrl3" urlPattern="/Curam/logonerror.jsp"
matchType="notContain"/>
</authFilter>
(...)
</server>
```

Note: The `federation_metadata.xml` file is generated by the identity provider, which is IBM ISAM.

4. In the `server.xml` file, change the `spHostAndPort="https://spm-application-url.com"` property to the appropriate domain URL.

5. Verify the authentication attributes that are extracted from the subject in the `Curam JAAS Login Module`, as shown in the following example:

```
Set<object> privateCredentials = loginSubject.getPrivateCredentials();
if (privateCredentials != null && privateCredentials.size() > 0) {
  for (Object credObject : privateCredentials) {
    if (credObject instanceof java.util.Hashtable) {
      java.util.Hashtable credPrivate = (java.util.Hashtable) credObject;
      username = (String)credPrivate.get("com.ibm.wsspi.security.cred.securityName");
     if (username != null && username.trim().length() > 0)
      {
        authenticationResult = true;
      }
    }
  }
}
```

Universal Access SSO on Kubernetes initiation and flow

For single sign-on, the SAML response, by HTTP POSTs, is interpreted and controlled by logic in IBM Cúram Universal Access.

In all SAML web SSO profile flows, the binding defines the mechanism that is used to send information through assertions between the identity provider (IdP) and the service provider (SP). WebSphere Liberty supports HTTP POST binding for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML response document is an XML document that contains data about the user and the assertion, some of which is optional.

Browser-based single sign-on (SSO) through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IdP) and the service provider (SP). The user is guided seamlessly from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single-page application, all the screens are contained within the application and dynamic content is expected to be passed only in JSON messages through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it would to leave the application and hand back control to either the user agent or the browser, in which case the application state would be lost.

IdP-initiated use case

The IdP can send an assertion request to the service provider ACS through one of the following methods:

- The IdP sends a URL link in a response to a successful authentication request. The user must click the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.
- The user authenticates into IdP and accesses the application that is configured as a partner to the IdP.

The ACS then validates the assertion, creates a JAAS subject, and redirects the user to the SP resource.

SP-initiated use case

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML `AuthnRequest` from the SP to the IdP. The IdP receives the `AuthnRequest`, validates that the request comes from a registered SP, and then authenticates the user. When the user is authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the `AuthnRequest` that was received from the SP.

Assertions and the SAML Response document

To prove the authenticity of the information, the assertion in the SAML response is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted. A

typical SAML response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server's cookie tied to the user's HTTP session.

IdP-initiated flow

When Social Program Management is configured in WebSphere Liberty with an-IdP initiated web SSO flow, any attempt to connect to a protected resource without first authenticating through IdP results in the application server falling back to an SP-initiated SSO flow. In an SP-initiated SSO flow, any authentication requests that are initiated through SP result in a 403 HTTP response, and the application redirects the user to the IdP login page for the user to authenticate. After the user is authenticated successfully, the control is redirected to the Social Program Management application page.

The following figure illustrates the IdP initiated flow that is supported by Social Program Management in a default installation.

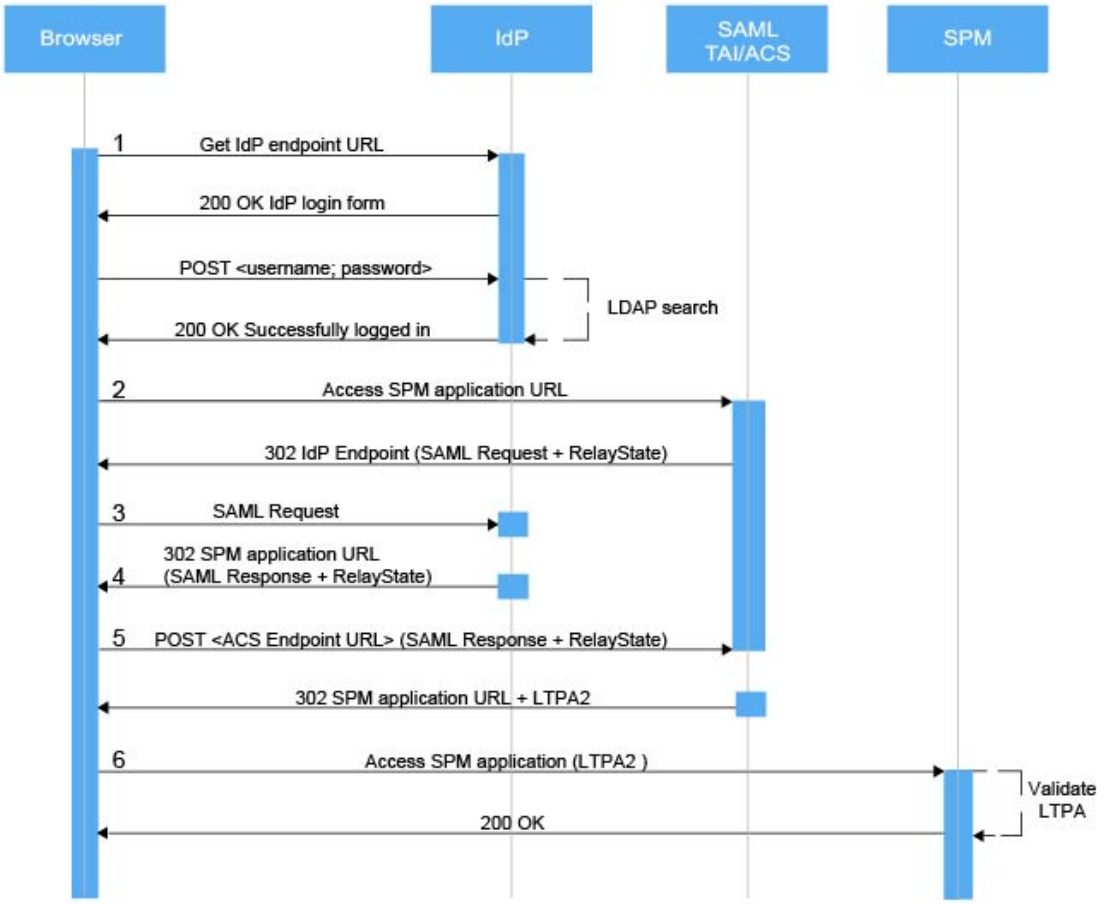

Figure 9. IdP-initiated flow

1. In an IdP-initiated flow, the user completes the IDP login form and authenticates.
2. After successful authentication in IdP, the user tries to access the Social Program Management application that is deployed in the application server.
3. The Trust Association Interceptor (TAI) and Assertion Consumer Service (ACS) (SAML TAI/ACS) that is deployed on the application server intercepts the request and redirects it to the IdP endpoint with a generated SAML request.

4. Because the user already logged into the IdP before the user accessed the Social Program Management application, the IdP responds with a SAML response and redirects the user to the Social Program Management application.
5. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Liberty also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect request that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

SP-initiated flow

When Social Program Management is configured with an SP-initiated web SSO flow, any attempt to connect to a protected resource without first authenticating results in a 401 HTTP response from the application server Assertion Consumer Service's Trust Association Interceptor, and the generation of the SAML AuthnRequest message to be sent to the IdP.

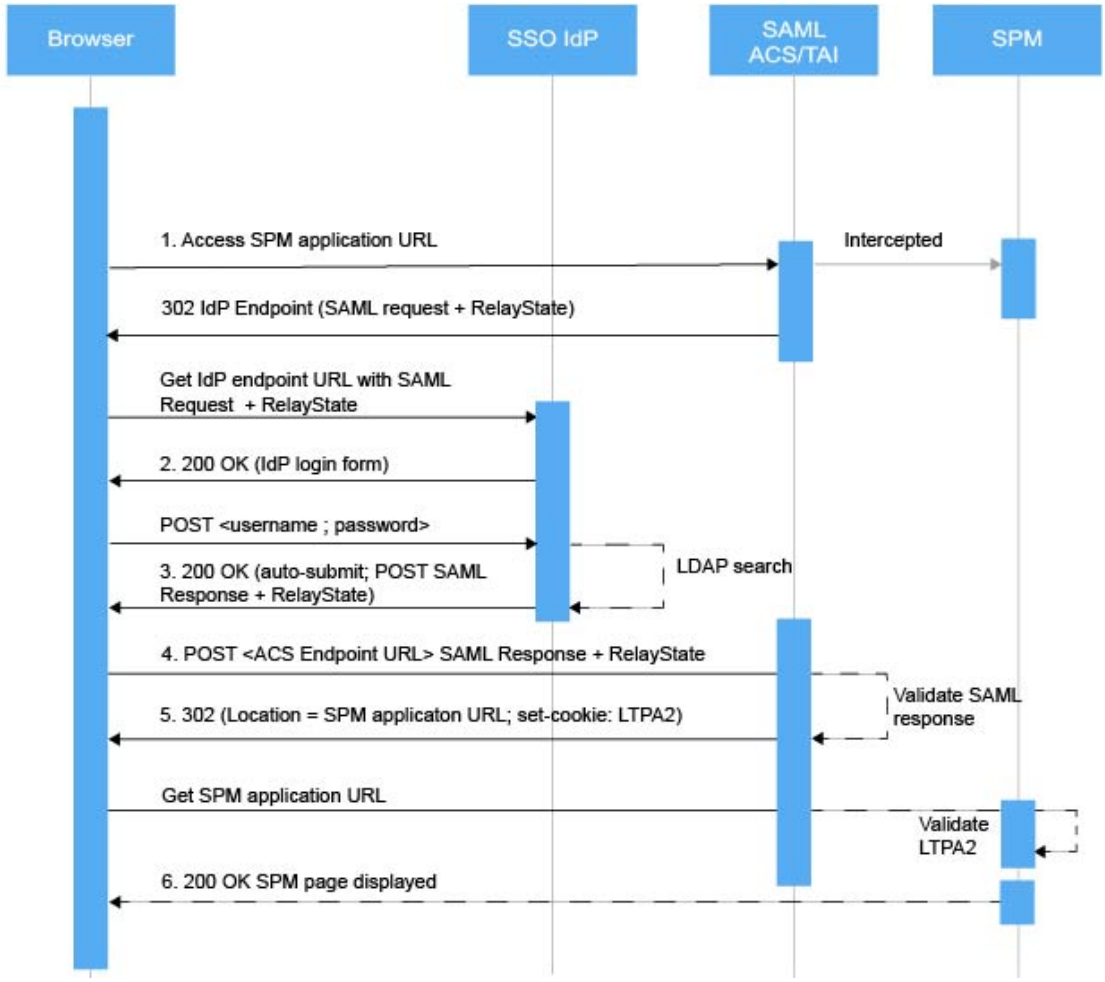

Figure 10. SP-initiated flow

1. When a user tries to access a Social Program Management application resource without authenticating, the TAI intercepts the request and redirects the user to the IdP endpoint with the generated SAML request.
2. The IdP endpoint displays the login form that the user completes to authenticate, then directs the SAML request to the IdP SAML endpoint.
3. After successful validation of the user credentials at the IdP, the IdP populates the SAML response and returns it in an HTML form that contains hidden input fields.

4. The HTML form is autosubmitted to the Social Program Management application with the SAML response and RelayState parameter.
5. The Social Program Management application extracts the RelayState parameter and SAML response values, and inserts them in a new POST request to the application server ACS.
6. The application server ACS validates the signature that is contained in the SAML response. WebSphere Liberty also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.
7. The Social Program Management application landing page is displayed in the browser.

Configuring the Universal Access Responsive Web Application for SSO

To enable the Universal Access Responsive Web Application to work with SAML single sign-on (SSO), configure the appropriate properties in the `.env` environment variable file in the root of the React application and rebuild the application.

About this task

- The <IdP_URL> consists of three parts: the HTTPS protocol, the IdP hostname or IP address, and the listener port number. For example, `https://192.168.0.1:12443`.
- The <ACS_URL> consists of three parts: the HTTPS protocol, the Assertion Consumer Service (ACS) hostname or IP address, and the listener port number. For example, `https://192.168.0.2:443`.

Procedure

1. Set the authentication method to SSO, see Customizing the authentication method.
2. Set the related environment variables for your SSO environment, see React environment variable reference. These properties are applicable to both identity provider (IdP)-initiated and service-provider (SP)-initiated SAML 2.0 web SSO unless otherwise stated.

SAML SSO on Kubernetes configuration example using ISAM

The example uses ISAM as an RPL-based SSO and outlines an SSO configuration for both IBM Cúram Social Program Management and IBM Cúram Universal Access that implements federated single sign-on by using the SAML 2.0 Browser POST profile. The example applies to both IdP-initiated and SP-initiated flows. Some additional steps are required to configure SP-initiated flows.

Note: This example configuration uses ISAM, you are free to use any SAML-based authorization and network security policy management solution.

SSO configuration components

Figure 1 shows the components that are included in a Social Program Management SSO configuration.

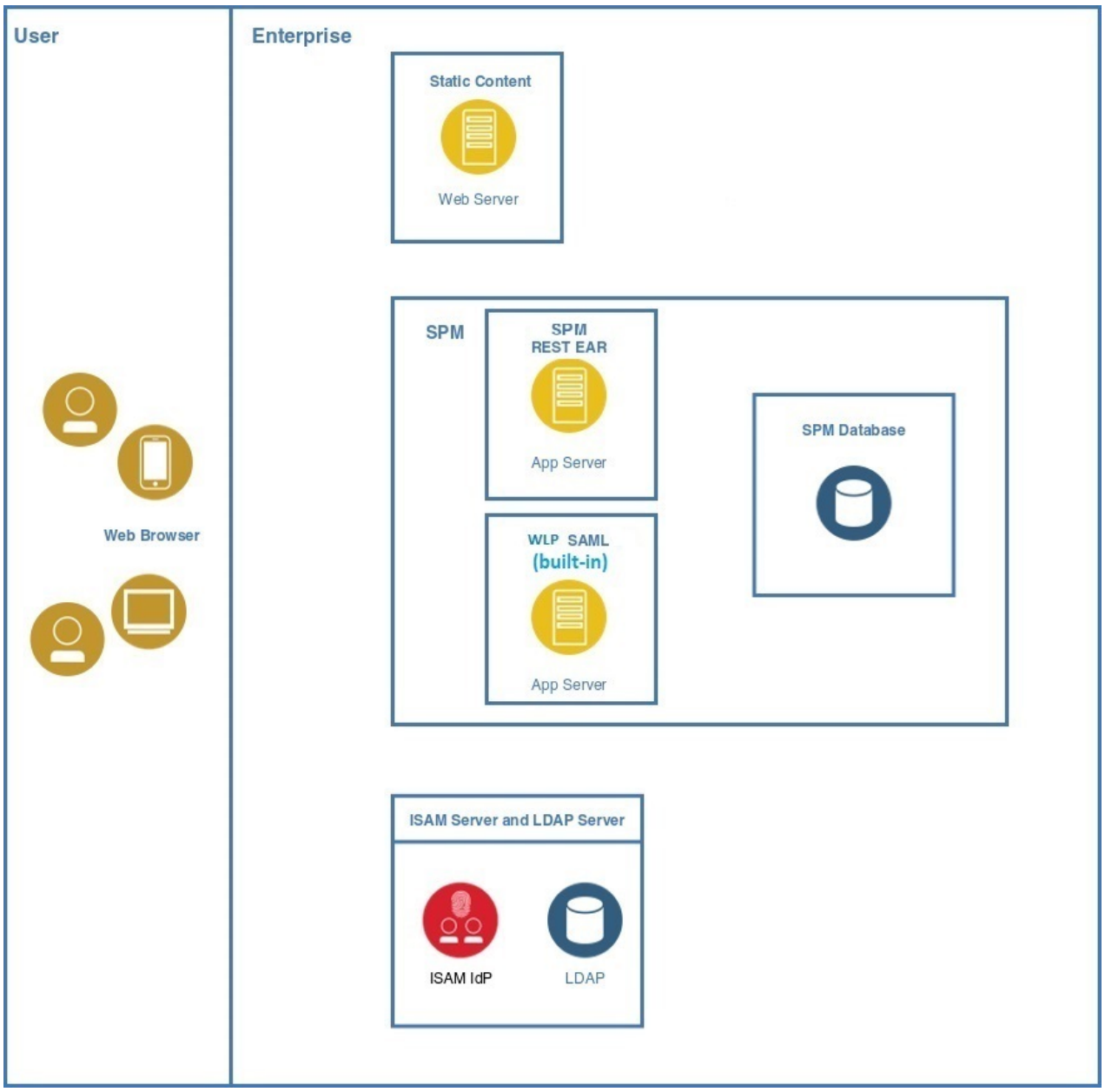

Figure 11. SSO configuration components

Web browser
A user sends requests from their web browser for applications in the SSO environment.

Web server
Social Program Management static content is deployed on a web server.

SAML-based SSO (ISAM) server
The IBM Security Access Manager server includes the identity provider (IdP).

LDAP server (user directory)
Among other items, the LDAP server contains the user name and password of all the valid users in the SSO environment.

WebSphere Application Server Liberty
Among other applications, WebSphere Liberty contains the deployed Social Program Management, Citizen WorkSpace, and REST enterprise applications.

Build-in WebSphere Liberty SAML configuration
Contains the features to run the SAML Trust Assertion Interceptor (TAI) and Consumer Service (ACS).

Social Program Management Database
Data storage for the Social Program Management, Citizen WorkSpace, and REST enterprise applications.

Configuring SSO with IBM Security Access Manager
Use the ISAM management console to configure single sign-on (SSO) in IBM Cúram Social Program Management.

Before you begin

1. Start IBM Security Access Manager.
2. In the management console, log on as an administrator.
3. Accept the services agreement.
4. If required, change the administrative password.

About this task

In the IBM Security Access Manager management console, complete the following steps, with reference to IBM Security Access Manager 9 Federation Cookbook.

Procedure

1. Configure the IBM Security Access Manager database:
   a) In the top menu, click **Home Appliance Dashboard** > **Database Configuration**.
   b) Enter the database configuration details, such as **Database Type**, **Address**, **Port**, and so on, and click **Save**.
   c) When the **Deploy Pending Changes** window opens, click **Deploy**.
2. To install all the required product licenses, complete the steps in section *4.3 Product Activation* from IBM Security Access Manager 9 Federation Cookbook
3. Configure the LDAP SSL database by completing section *25.1.1 Load Federation Runtime SSL certificate into pdsrv trust store* from the IBM Security Access Manager 9 Federation Cookbook .
4. Configure the runtime component by completing *4.6 Configure ISAM Runtime Component on the Appliance* from the IBM Security Access Manager 9 Federation Cookbook.

Configuring IBM Security Access Manager as an IdP
To configure IBM Security Access Manager as an identity provider (IdP), complete the outlined steps from the IBM Security Access Manager 9.0 Federation Cookbook that is available from IBM Developer Works.

Before you begin

Download the IBM Security Access Manager 9.0 Federation Cookbook from IBM Developer Works, as shown in the related link. Also, download the mapping files that are provided with the cookbook.

About this task

To set up the example environment, see Access Manager Federation Cookbook and complete the specified sections in the *IBM Security Access Manager Federation Cookbook 9.0.6.0* PDF that is attached to the page.

Procedure

1. Complete *Section 5, Create Reverse Proxy instance*.
2. Complete *Section 6, Create SAML 2.0 Identity Provider federation*.

In Section 6.1, if you are using the ISAM docker deployment, it is possible to reuse the existing keystore that is included in the container instead of creating a new keystore. It is important to reflect this change in subsequent sections where the myidpkeys certificate database is referenced.

3. Complete *Section 8.1, ISAM Configuration for the IdP*.

   In Section 8.1, use the hostname of the IdP federation.

4. Optional: After you complete Section 8.1.1, if you require ACLs to be defined to allow and restrict access to the IdP junction, then follow the instructions in *Section 25.1.3, Configure ACL policy for IdP*.

5. Complete *Section 9.1, Configuring Partner for the IdP*.

   The export from WebSphere Application Server Liberty does not contain all the relevant data. Therefore, in Section 9.1, after you complete configuring partner for the IdP, you must click **Edit configuration** and complete the remaining advanced configuration.

Adding and enabling users in LDAP
Add the users from LDAP and enable them in SAML-based SSO.

Procedure

1. To create LDAP and IBM Security Access Manager runtime users, create an `ldif` file that can be used to populate OpenLDAP, as shown in the following sample:

```
# cat usersCreate_ISAM.ldif
dn: dc=watson-health,secAuthority=Default
objectclass: top
objectclass: domain
dc: watson-health

dn: c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: country
c: ie

dn: o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organization
o: curam

dn: ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamint

dn: cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: caseworker
sn: caseworkersurname
uid: caseworker
mail: caseworker@curam.com
userpassword: Passw0rd

dn: ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamext

dn: cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: jamessmith
sn: Smith
uid: jamessmith
mail: jamessmith@curamexternal.com
userpassword: Passw0rd
```

2. Add users to the OpenLDAP database:

a) On the host server that is running the docker containers, enter the following command:

```
docker cp usersCreate_ISAM.ldif idpisam9040_isam-ldap_1:/tmp
```

b) To log on to the OpenLDAP container, enter the following command:

```
docker exec –ti idpisam9040_isam-ldap_1 bash
```

c) To add the users to OpenLDAP, enter the following command:

```
ldapadd -H ldaps://127.0.0.1:636 -D cn=root,secAuthority=default -f /tmp/
Curam_usersCreate_ISAM.ldif
```

3. Import the users into IBM Security Access Manager:

a) To log on to the IBM Security Access Manager command-line interface, enter the following commands:

```
docker exec -ti idpisam9040_isam-webseal_1 isam_cli
isam_cli> isam admin
pdadmin> login -a sec_master -p <password>
```

b) To import the users into IBM Security Access Manager, enter the following commands:

```
pdadmin sec_master> user import caseworker
cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify caseworker account-valid yes
pdadmin sec_master> user import jamessmith
cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify jamessmith account-valid yes
```

4. To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://IdP_URL/isam/sps/saml20idp/saml20/
 logininitial?RequestBinding=HTTPPost&PartnerId=ACS_URL/samlsps/acs
 &NameIdFormat=Email&Target=WLP_hostname:WLP_port/Rest/v1
```

Replace the following values in the URL with the appropriate values for your configuration:

- *IdP_URL* is the IBM Security Access Manager login initial URL
- *ACS_URL* is the SAML Assertion Consumer Service URL
- *WLP_hostname* is the WebSphere Liberty application server host name
- *WLP_port* is the WebSphere Liberty application server port, where in IBM Cúram Social Program Management the default value is 9044

When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by Social Program Management, an XHR timeout occurs before the initialization finishes. Therefore, this test step is required to ensure that the initialization of the IdP endpoints is completed.

5. In a browser, go to the home page and log in.

Testing IdP-initiated SAML SSO infrastructure
When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by IBM Cúram Social Program Management, an XHR timeout occurs before the initialization finishes. This test step is required to ensure that the initialization of the IdP endpoints is completed.

Procedure

To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://<isam_url>/isam/sps/saml20idp/saml20/logininitial?
RequestBinding=HTTPPost&PartnerId=https://<was_url>/samlsps/acs&NameIdFormat=Email&Target=<
was_url>/Rest/api/definitions
```

Where:

- `<isam_url>` - The URL for IBM Security Access Manager. It consists of the IBM Security Access Manager hostname, and port number, for example, `https:// 192.168.0.1:12443`.
- `<junction_name>` - The junction name that is used during the federation configuration in reverse proxy. The default value is `isam`.
- `<idp_endpoint>` - The endpoint that is configured for the IDP federation. The default value is `sps`.
- `<federation_name>` - The name that was used when you created the federation.
- WebSphere Application Server Liberty hostname
- WebSphere Liberty hostname
- WebSphere Liberty port. The default value is 9044 for IBM Cúram Social Program Management.

SP-initiated only: Testing SP-initiated SAML SSO infrastructure
Test the SP-initiated SAML SSO infrastructure.

About this task

Open your browser, with network devtools, and load a protected IBM Cúram Social Program Management application URL like this example: `<SPM_Kubernetes_URL>/curam`. `<SPM_Kubernetes_URL>` is the URL of the Social Program Management that is deployed in the Kubernetes environment, for example `https://spm.dev.watson-health.ibm.com/curam`. You are redirected to the ISAM SSO log-in page. Log in with the credentials of a user who is authorized to access Social Program Management. You are redirected to the Social Program Management application after a successful authentication.

Procedure

1. Open your browser with network devtools, and load a protected IBM Cúram Social Program Management application URL, as shown in the following example:

   ```
   https://application-domain.com
   ```

   You are redirected to the ISAM SSO log-on page.
2. Log on with the credentials of a user who is authorized to access the Social Program Management application.

   After a successful authentication, you are redirected to the Social Program Management application.

Configuring SAML SSO on WebSphere Application Server

Configure SAML SSO for Social Program Management on WebSphere Application Server. If you are using IBM Cúram Universal Access, you must do some additional Universal Access configuration.

SAML SSO initiation and flow on WebSphere Application Server

In all SAML web SSO profile flows, the binding defines the mechanism that is used to send information through assertions between the identity provider (IdP) and the service provider (SP). For Universal Access, the SAML response by HTTP POSTs is interpreted and controlled by logic in the IBM Universal Access Responsive Web Application.

WebSphere Application Server supports HTTP POST binding for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML response document is an XML document that contains certain data about the user and the assertion, some of which is optional.

Browser-based single sign-on (SSO) through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IdP) and the service provider (SP). The user is guided seamlessly from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single-page application such as the IBM Universal Access Responsive Web Application, all screens are contained in the application and dynamic content is expected to be passed only in JSON messages

through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it would be necessary to leave the application and hand back control to either the user agent or the browser, in which case the application state would be lost.

IdP-initiated use case

The IdP can send an assertion request to the service provider ACS in one of two ways:

- The IdP sends a URL link in a response to a successful authentication request. The user must click the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.

The ACS then validates the assertion, creates a JAAS subject, and redirects the user to the SP resource.

SP-initiated use case

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML AuthnRequest from the SP to the IdP. The IdP receives the AuthnRequest, validates that the request comes from a registered SP, and then authenticates the user. After the user is authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the AuthnRequest that was received from the SP.

Assertions and the SAML Response document

To prove the authenticity of the information, the assertion in the SAML response is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted. A typical SAML response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server’s cookie tied to the user’s HTTP session.

SAML SSO initiation and flow diagrams

Review the flow diagram that matches your environment.

- “IdP-initiated flow for Social Program Management in WebSphere Application Server” on page 37
- “IdP-initiated flow for Universal Access in WebSphere Application Server” on page 38
- “SP-initiated flow for Social Program Management in WebSphere Application Server” on page 40
- “SP-initiated flow for Universal Access in WebSphere Application Server” on page 40

IdP-initiated flow for Social Program Management in WebSphere Application Server

The following figure illustrates the IdP-initiated flow that is supported by Social Program Management in a default installation.

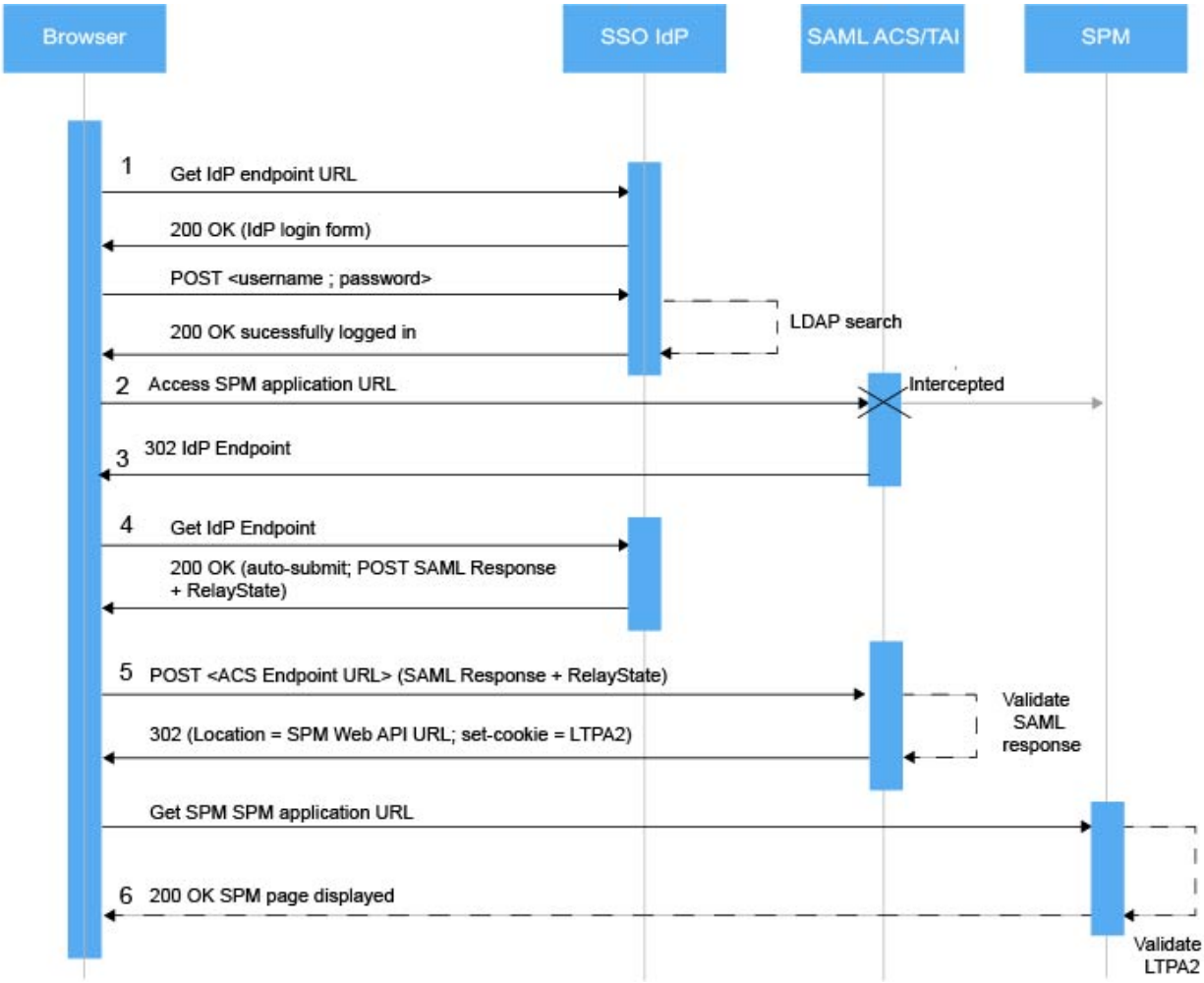

Figure 12. IdP-initiated flow for Social Program Management in WebSphere Application Server

1. In an IdP-initiated flow, the user completes the IDP login form and authenticates.
2. After successful authentication in IdP, the user tries to access the Social Program Management application that is deployed in the application server.
3. The Trust Association Interceptor (TAI) and Assertion Consumer Service (ACS) that is deployed on the application server intercepts the request and redirects it to the IdP endpoint.
4. Because the user already logged into the IdP, the IdP responds with a SAML response and redirects the user to the Social Program Management application.
5. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Application Server also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured Social Program Management target landing page, along with an LTPA2 cookie to be used in any subsequent communication.
6. The Social Program Management application landing page is displayed in the browser.

IdP-initiated flow for Universal Access in WebSphere Application Server

When Universal Access is configured with an IdP initiated web SSO flow, any attempt to connect to a protected resource without first authenticating through IdP results in a 403 HTTP response from IBM Cúram Social Program Management web API. Any authentication requests that are initiated through SP result in a 403 HTTP response, and the application redirects the user to the login page in Universal Access.

The following figure illustrates the IdP-initiated flow that is supported by Universal Access in a default installation.

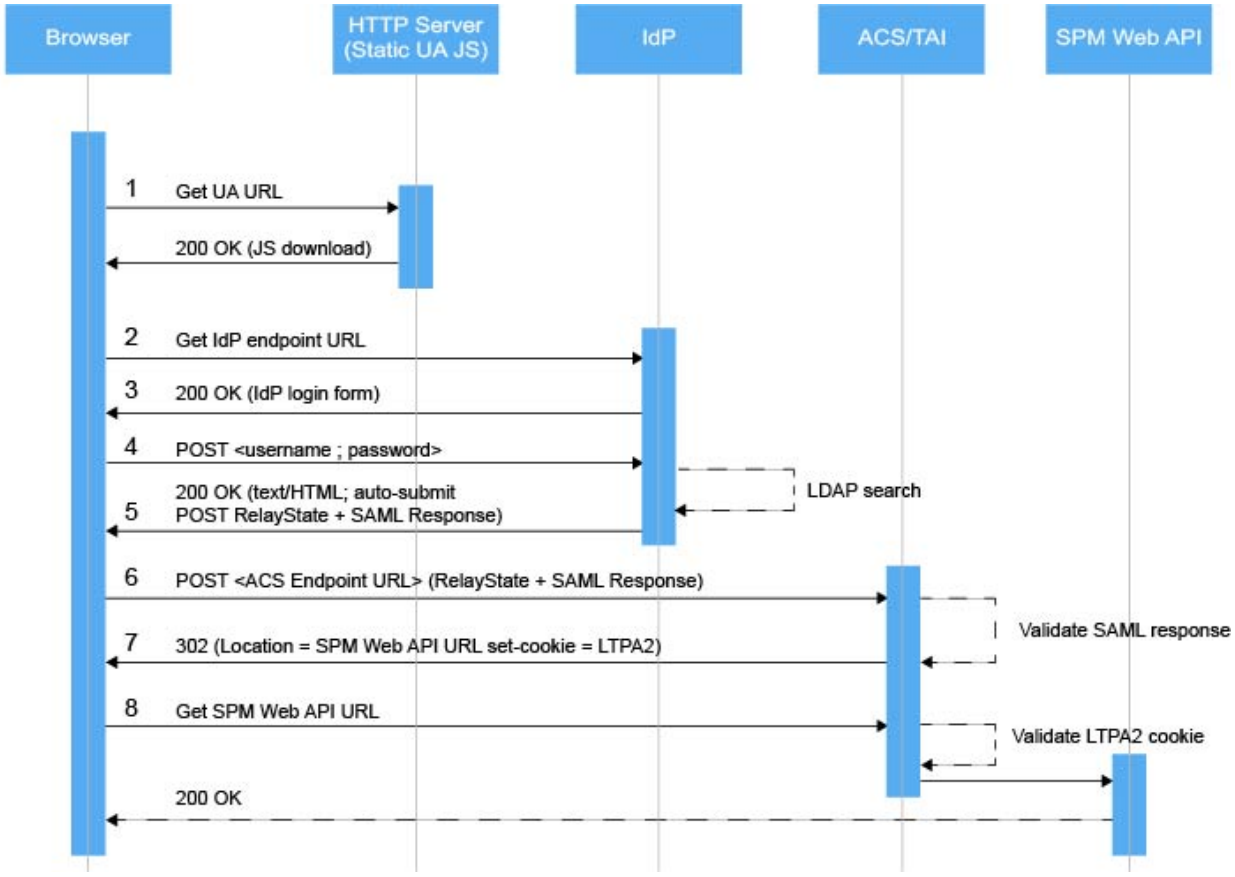

Figure 13. IdP-initiated flow for Universal Access in WebSphere Application Server

1. A user browses to the HTTP server that contains Universal Access.
2. The user can browse as normal by interacting with IBM Cúram Social Program Management as either a public or a generated user (which is not shown in the diagram). The user then opens the login page to access protected content, which triggers an initial request to the IdP endpoint.
3. In most IdP configurations, an HTML login form responds to the request. Universal Access ignores the response.
4. To authenticate, the user completes the login form and clicks **Submit**. The form submission triggers an HTTP POST request that contains login credentials to the IdP.
5. After successful validation of the user credentials at the IdP, the IdP populates the SAML Response and returns it in an HTML form that contains hidden input fields. Several redirects might occur before the 200 OK HTTP response that contains the SAML information is received. Universal Access does not respond to the redirects.
6. Universal Access extracts the `RelayState` and `SAMLResponse` values, and inserts them in a new POST request to the application server Assertion Consumer Service (ACS).
7. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Application Server also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured IBM Cúram Social Program Management target landing page, along with an LTPA2 Cookie that will be used in any subsequent communication.
8. Universal Access begins its standard user setup by requesting account and profile information from the relevant web API endpoints.

SP-initiated flow for Social Program Management in WebSphere Application Server

The following figure illustrates the SP-initiated flow that is supported by Social Program Management in a default installation.

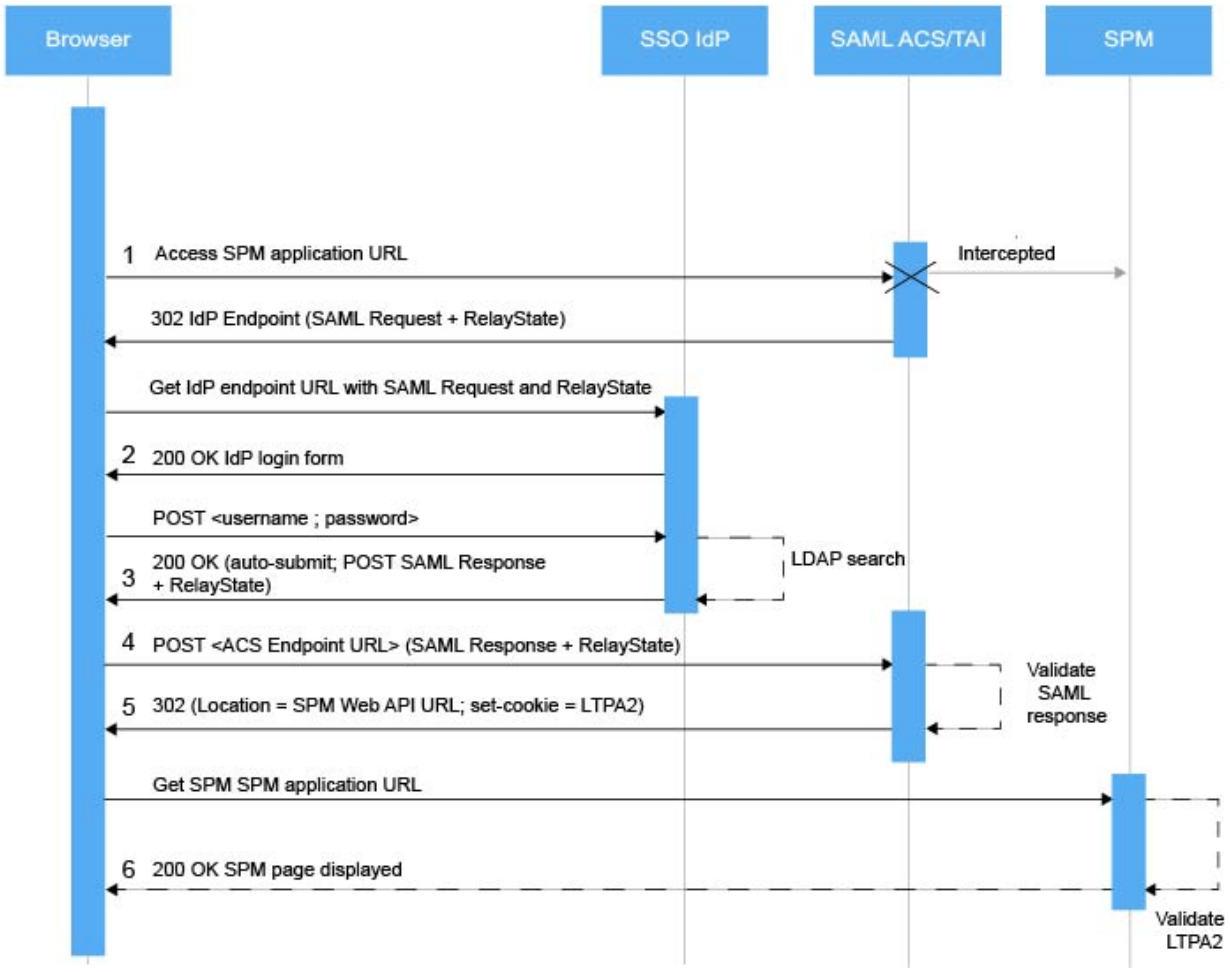

Figure 14. SP-initiated flow for Social Program Management in WebSphere Application Server

1. When a user tries to access an Social Program Management application resource without authenticating, the TAI intercepts the request and redirects the user to the IdP endpoint with the generated SAML request.
2. The IdP endpoint displays the login form that the user completes to authenticate, then directs the SAML request to the IdP SAML endpoint.
3. After successful validation of the user credentials at the IdP, the IdP populates the SAML response and returns it in an HTML form that contains hidden input fields.
4. The HTML form is autosubmitted to the Social Program Management application with the SAML response and RelayState parameter.
5. The application server ACS validates the signature that is contained in the SAML response. WebSphere Application Server also ensures that the originator is a Trusted Authentication Realm. If the validation is successful, the ACS sends an HTTP redirect that points to the configured Social Program Management target landing page, along with an LTPA2 cookie that is used in any subsequent communication.

SP-initiated flow for Universal Access in WebSphere Application Server

When Universal Access is configured with an SP-initiated web SSO flow, any attempt to connect to a protected resource without first authenticating results in a 401 HTTP response from the application

server Assertion Consumer Service's Trust Association Interceptor, and the generation of the SAML AuthnRequest message to be sent to the IdP.

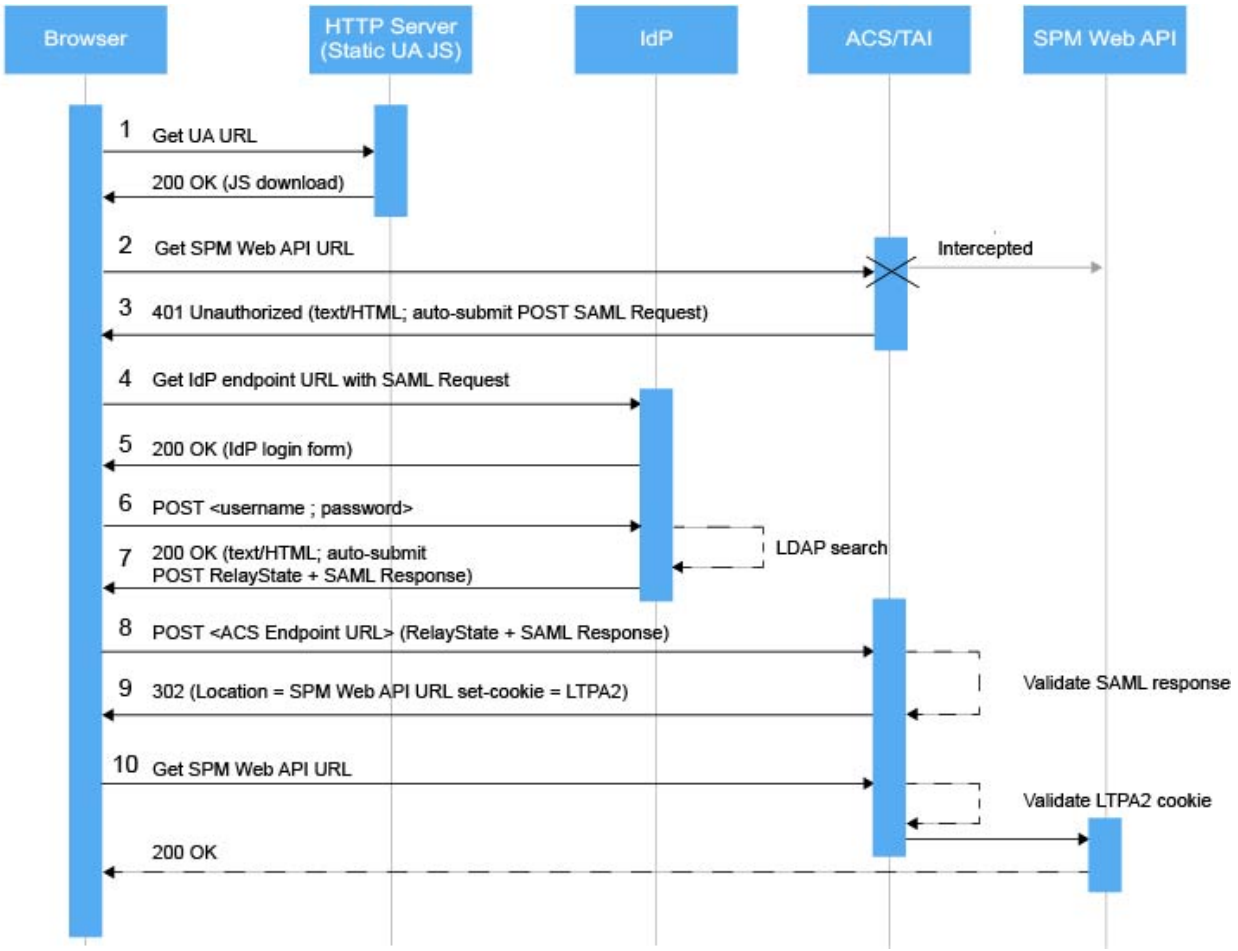

Figure 15. SP-initiated flow for Universal Access in WebSphere Application Server

1. A user browses to the HTTP server that contains Universal Access.
2. The user can browse as normal by interacting with IBM Cúram Social Program Management as either a public or a generated user (which is not shown in the diagram). The user then accesses protected content in the application, which is intercepted by the Assertion Consumer Service Trust Association Interceptor (TAI).
3. The TAI triggers an 401 HTTP response with the SAML request message to be sent to the IdP.
4. Universal Access then directs the SAML Request to the IdP SAML endpoint.
5. In most IdP configurations, an HTML login form responds to the request. Universal Access extracts a hidden authentication token in the login form if present, ignoring the rest of the response.
6. To authenticate, the user completes the login form and clicks **Submit**. The form submission triggers an HTTP POST request that contains login credentials to the IdP, along with the token extracted in the previous step if present.
7. After successful validation of the user credentials at the IdP, the IdP populates the SAML Response and returns it in an HTML form that contains hidden input fields. Several redirects might occur before the 200 OK HTTP response that contains the SAML information is received. Universal Access does not respond to the redirects.
8. Universal Access extracts the `RelayState` and `SAMLResponse` values, and inserts them in a new POST request to the application server Assertion Consumer Service (ACS).
9. The application server ACS validates the signature that is contained in the SAML Response. WebSphere Application Server also ensures that the originator is a Trusted Authentication Realm.

If the validation is successful, the ACS sends an HTTP redirect that points to the configured IBM Cúram Social Program Management target landing page, along with an LTPA2 Cookie that will be used in any subsequent communication.

10. The browser automatically sends a new request to the target URL, but Universal Access does not respond to the request. Universal Access begins its standard user setup by requesting account and profile information from the relevant web API endpoints.

Configuring WebSphere Application Server as a SAML service provider

To configure SSO for Social Program Management, you must configure IBM WebSphere Application Server as a SAML service provider.

About this task

For more information, see the related link to the WebSphere Application Server documentation.

Procedure

1. Deploy the `WebSphereSamlSP.ear` file.

   Note: So that SAML Assertion Consumer Service (ACS) works with cross-origin resource sharing (CORS) security requirements during redirections, you must map its modules to the same virtual host used for the REST target application (that is, *client_host*).

   The `WebSphereSamlSP.ear` file is available as an installable package. Choose one of the following methods:

   - Log on to the WebSphere Application Server administrative console, and install the `app_server_root/installableApps/WebSphereSamlSP.ear` file to your application server or cluster.
   - Install the SAML ACS application by using a Python script. In the `app_server_root/bin` directory, enter the following command to run the `installSamlACS.py` script:

     ```
     wsadmin -f installSamlACS.py install nodeName serverName
     ```

     Where *nodeName* is the node name of the target application server, and *serverName* is the server name of the target application server. When you complete this step, you must map the modules to the REST application, for more information see: Mapping virtual hosts for web modules.

2. Configure the ACS trust association interceptor:
   a) In the WebSphere Application Server administrative console, click **Global security** > **Trust association** > **Interceptors** > **New**.
   b) For **Interceptor class name**, enter `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.
   c) Under custom properties, enter the values that are shown in the following table:

      In a standard WebSphere Application Server configuration, you would also define a value for the `login.error.page` custom property. However, the preferred method is to log on to the IdP first. Therefore, if you do not define a value for `login.error.page`, WebSphere Application Server returns a 403 error if a user logs on without first logging on to the identity provider (IdP).

Table 2. ACS trust association interceptor custom properties	
Custom property name	**Value**
sso_1.sp.acsUrl	https://*WAS_host_name:ssl port*//samlsps/acs
sso_1.idp_1.EntityID	https://*isam_hostname:isam_port*//*URL of ISAM*/*ISAM Junction*/*IdP endpoint*/*federation name*/saml20

Table 2. ACS trust association interceptor custom properties (continued)	
Custom property name	**Value**
sso_1.idp_1.SingleSignOnUrl	https:// *isam_hostname:isam_port//URL of ISAM/ISAM Junction/IdP endpoint/federation name*/saml20/login
sso_1.sp.targetUrl	https://*WAS_host_name*:*WAS_port*/Rest
sso_1.idp_1.certAlias	isam-conf
sso_1.sp.filter	request-url^=/Rest;request-url!=/Rest/ j_security_check
sso_1.sp.enforceTaiCookie	false

3. Add the IdP federation partner data. The following substeps describe how to add the IdP data by using the WebSphere Application Server administrative console.
   a) To add the IdP host name or IP address as a trusted realm, click **Global security** > **Trusted authentication realms - inbound** > **Add External Realm**.
   b) Enter either the IBM Security Access Manager host name or IP address.
   c) To load the IdP certificate from IBM Security Access Manager, click **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates** > **Retrieve from port**
   d) Enter the IBM Security Access Manager IP address and listener port, for example, `12443, alias = isam-conf`.

      Note: When the browser first attempts to connect to the IBM Cúram Social Program Management web API, an LTPA2 cookie is sent as part of the request. If the WebSphere Application Server **`com.ibm.ws.security.web.logoutOnHTTPSessionExpire`** property is set to true, which is the default configuration in IBM Cúram Social Program Management, then authentication fails because an HTTP session does not exist on the application server. By setting the property to `false`, the check for a valid HTTP session is not completed and when the LTPA2 token is valid, authentication succeeds.

      To configure the property in the WebSphere Application Serveradministrative console, click **Security** > **Global security** > **Custom properties**, and set the value of **`com.ibm.ws.security.web.logoutOnHTTPSessionExpire`** to `false`.
4. Implement cross-origin resource sharing (CORS) from the HTTP server to the WebSphere Application Server SAML ACS.
   a) To add a CORS header, configure a servlet filter for the `WebSphereSamlSP.ear` file that is deployed by a Trust Association Interceptor (TAI). The servlet filter adds a CORS HTTP header to HTTP responses. You can archive the implemented servlet filter as a `jar` file, and then store it in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war\WEB-INF\lib` directory that is in the `installedApps` directory of your project in WebSphere Application Server.
      See the following example of how to implement a servlet filter:

```
public class SampleFilter implements Filter {

       @Override
       public void doFilter(ServletRequest arg0, ServletResponse servletResponse,
            FilterChain arg2) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) servletResponse;
HttpServletRequest request = (HttpServletRequest) arg0;


response.setHeader("Access-Control-Allow-Origin",
     "http://dubxpcvm156.mul.ie.ibm.com:9880");     <hostname or IP address of IBM UA
server>
response.setHeader("Access-Control-Allow-Credentials", "true");
response.setHeader("Access-Control-Allow-Headers", "x-requested-with, Content-Type,
origin, authorization, accept, client-security-token");
```

```
response.setHeader("Access-Control-Expose-Headers", "content-length");
        arg2.doFilter(request, response);
    }
}
```

b) Configure the `web.xml` file for the deployed TAI EAR file to use the servlet filter for all the requests. Add the filter element that is shown in the following sample to the `web.xml` file, with the actual fully qualified name of the filter.

You can add the filter element as a sibling to any existing element n the `web.xml` file, such as `<servlet>`. The `web.xml` file is in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war\WEB-INF\lib` directory, which is in the `installedApps` directory of your project in WebSphere Application Server.

```
<filter>
    <filter-name> SampleFilter </filter-name>
    <filter-class> SampleFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name> SampleFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Configuring IBM Cúram Universal Access for SSO

To configure SSO for Universal Access, you must configure the Universal Access Responsive Web Application to use SSO authentication, and configure cross-origin resource sharing (CORS) for Universal Access.

Before you begin

Ensure that IBM Cúram Social Program Management is configured for SSO. For IBM WebSphere Application Server, see "Configuring WebSphere Application Server as a SAML service provider" on page 42.

Configuring the Universal Access Responsive Web Application for SSO

To enable the Universal Access Responsive Web Application to work with SAML single sign-on (SSO), configure the appropriate properties in the `.env` environment variable file in the root of the React application and rebuild the application.

About this task

- The `<IdP_URL>` consists of three parts: the HTTPS protocol, the IdP hostname or IP address, and the listener port number. For example, `https://192.168.0.1:12443`.
- The `<ACS_URL>` consists of three parts: the HTTPS protocol, the Assertion Consumer Service (ACS) hostname or IP address, and the listener port number. For example, `https://192.168.0.2:443`.

Procedure

1. Set the authentication method to SSO, see Customizing the authentication method.
2. Set the related environment variables for your SSO environment, see React environment variable reference. These properties are applicable to both identity provider (IdP)-initiated and service-provider (SP)-initiated SAML 2.0 web SSO unless otherwise stated.

Configuring CORS for IBM Cúram Universal Access

You must configure cross-origin resource sharing (CORS) for IBM Cúram Universal Access. For security reasons, browsers restrict cross-origin HTTP requests, including XMLHttpRequest HTTP requests, that are initiated in Universal Access. When the Universal Access application and the Universal Access web API are deployed on different hosts, extra configuration is needed.

About this task

Universal Access can request HTTP resources only from the same domain that the application was loaded from, which is the domain that contains the static JavaScript. To enable Universal Access to support cross-origin resource sharing (CORS), enable the use of CORS headers.

Procedure

1. Log on to the IBM Cúram Social Program Management application as a system administrator, and click **System Configurations**.
2. In the Shortcuts menu, click **Application Data** > **Property Administration**.
3. Configure the **`curam.rest.allowedOrigins`** property with the values of either the hostnames or the IP addresses of the IdP server and the web server on which Universal Access is deployed.

SAML SSO configuration example with IBM Security Access Manager

The example outlines a single sign-on (SSO) configuration for IBM Cúram Social Program Management and IBM Cúram Universal Access that uses IBM Security Access Manager to implement federated single sign-on by using the SAML 2.0 Browser POST profile. The example applies to both IdP-initiated and SP-initiated flows. Some additional steps are needed to configure SP-initiated flows.

Universal Access SSO configuration components

The following figure shows the components that are included in a Universal Access SSO configuration.

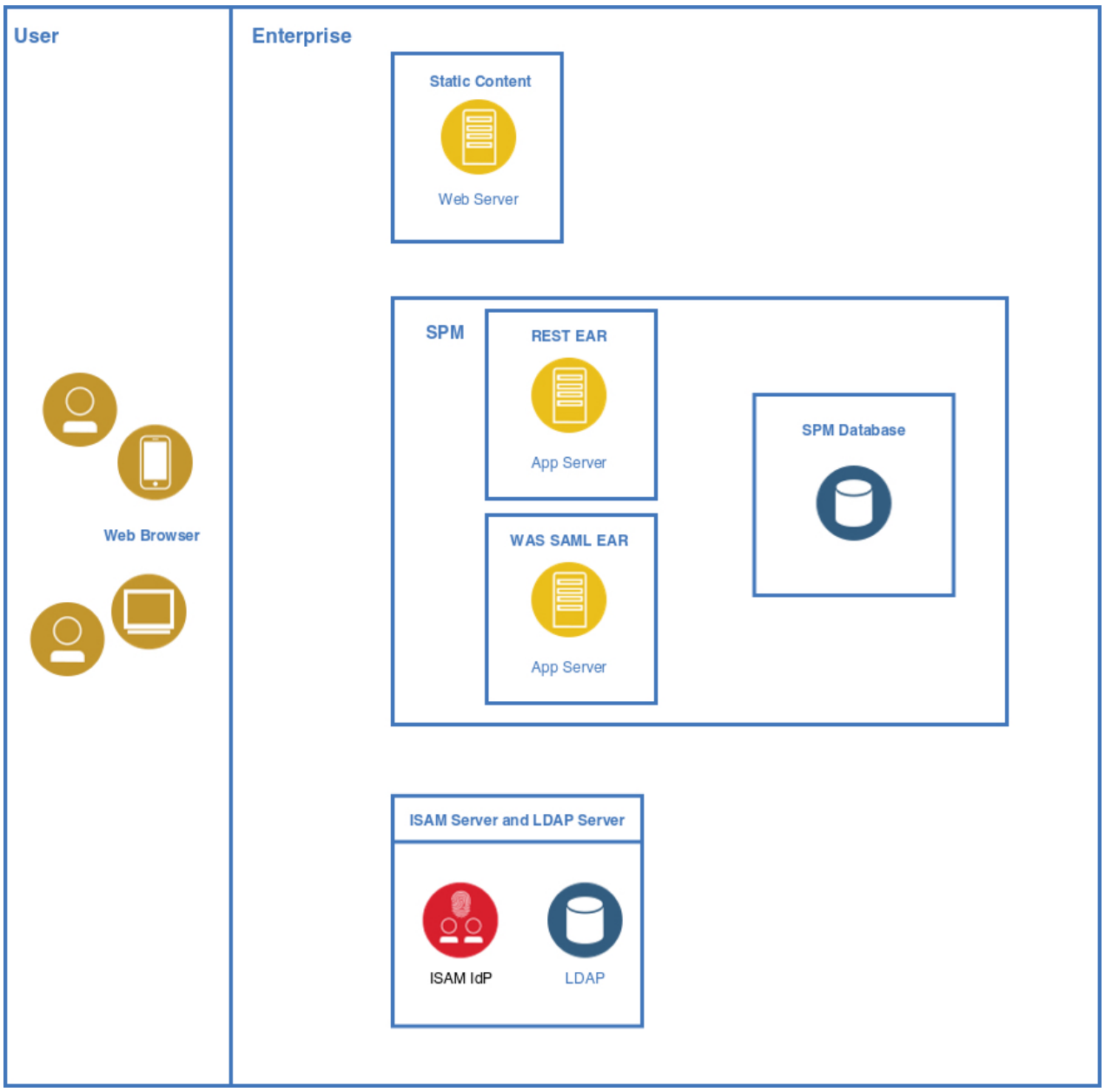

Figure 16. Universal Access SSO configuration components

Web browser
A user sends requests from their web browser for applications in the SSO environment.

Web server
The Universal Access ReactJS static content is deployed on a web server, such as IBM HTTP Server, or Apache HTTP Server.

IBM Security Access Manager (ISAM) server
The IBM Security Access Manager server includes the identity provider (IdP).

LDAP server (user directory)
Among other items, the LDAP server contains the username and password of all the valid users in the SSO environment.

IBM WebSphere Application Server
Among other applications, WebSphere Application Server contains the deployed IBM Cúram Social Program Management, Citizen Workspace, and REST enterprise applications.

WebSphere Application Server SAML EAR
A WebSphere Application Server package that contains the packages to run the SAML Assertion Consumer Service (ACS).

Social Program Management database
Data storage for the IBM Cúram Social Program Management, Citizen Workspace, and REST enterprise applications.

Configuring single sign-on through IBM Security Access Manager
Use the IBM Security Access Manager management console to configure single sign-on (SSO) in IBM Cúram Universal Access.

Before you begin

1. Start IBM Security Access Manager.
2. In the management console, log on as an administrator.
3. Accept the services agreement.
4. If required, change the administrative password.

About this task

In the IBM Security Access Manager management console, complete the following steps, with reference to the *IBM Security Access Manager 9 Federation Cookbook*.

Procedure

1. Configure the IBM Security Access Manager database:
   a) In the top menu, click **Home Appliance Dashboard** > **Database Configuration**.
   b) Enter the database configuration details, such as **Database Type**, **Address**, **Port**, and so on, and click **Save**.
   c) When the **Deploy Pending Changes** window opens, click **Deploy**.
2. To install all the required product licenses, complete the steps in section *4.3 Product Activation* from the *IBM Security Access Manager 9 Federation Cookbook*.
3. Configure the LDAP SSL database by completing section *25.1.1 Load Federation Runtime SSL certificate into pdsrv trust store* from IBM Security Access Manager Federation Cookbook .
4. Configure the runtime component by completing *4.6 Configure ISAM Runtime Component on the Appliance* from IBM Security Access Manager Federation Cookbook .

Configuring IBM Security Access Manager as an IdP
To configure IBM Security Access Manager as an identity provider (IdP), see the IBM Security Access Manager 9.0 Federation Cookbook that is available from IBM Developer Works.

Before you begin

Download the IBM Security Access Manager 9.0 Federation Cookbook from IBM Developer Works, as shown in the related link. Also download the mapping files that are provided with the cookbook.

About this task

To set up the example environment, complete the specified sections in the IBM Security Access Manager 9.0 Federation Cookbook.

Procedure

1. Complete *Section 5, Create Reverse Proxy instance*.
2. Complete *Section 6, Create SAML 2.0 Identity Provider federation*.

In Section 6.1, if you are using the ISAM docker deployment, it is possible to re-use the existing keystore that is included in the container instead of creating a new keystore. It is important to reflect this change in subsequent sections where the myidpkeys certificate database is referenced.

3. Complete *Section 8.1, ISAM Configuration for the IdP*.

   In Section 8.1, use the host name of the IdP federation.

4. Optional: After completing Section 8.1.1, if you require ACLs to be defined to allow and restrict access to the IdP junction, then follow the instructions in *Section 25.1.3, Configure ACL policy for IdP*.

5. Complete *Section 9.1, Configuring Partner for the IdP*.

   The export from Websphere does not contain all the relevant data. Therefore, in Section 9.1, after you complete configuring partner for the IdP, you must click **Edit configuration** and complete the remaining advanced configuration.

Configuring WebSphere Application Server as a SAML service provider
The procedure outlines the high-level steps that are required to configure IBM WebSphere Application Server as a SAML service provider.

About this task

For more information, see the related link to the WebSphere Application Server documentation.

Procedure

1. Deploy the `WebSphereSamlSP.ear` file.

   Note: So that SAML Assertion Consumer Service (ACS) works with cross-origin resource sharing (CORS) security requirements during redirections, you must map its modules to the same virtual host used for the REST target application (that is, *client_host*).

   The `WebSphereSamlSP.ear` file is available as an installable package. Choose one of the following methods:

   - Log on to the WebSphere Application Server administrative console, and install the `app_server_root/installableApps/WebSphereSamlSP.ear` file to your application server or cluster.
   - Install the SAML ACS application by using a Python script. In the `app_server_root/bin` directory, enter the following command to run the `installSamlACS.py` script:

     ```
     wsadmin -f installSamlACS.py install nodeName serverName
     ```

     Where *nodeName* is the node name of the target application server, and *serverName* is the server name of the target application server. When you complete this step, you must map the modules to the REST application, for more information see: Mapping virtual hosts for web modules.

2. Configure the ACS trust association interceptor:

   a) In the WebSphere Application Server administrative console, click **Global security** > **Trust association** > **Interceptors** > **New**.

   b) For **Interceptor class name**, enter `com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor`.

   c) Under custom properties, enter the values that are shown in the following table:

   In a standard WebSphere Application Server configuration, you would also define a value for the `login.error.page` custom property. However, the preferred method is to log on to the IdP first. Therefore, if you do not define a value for `login.error.page`, WebSphere Application Server returns a 403 error if a user logs on without first logging on to the identity provider (IdP).

Table 3. ACS trust association interceptor custom properties

Custom property name	**Value**
sso_1.sp.acsUrl	https://*WAS_host_name:ssl port*//samlsps/acs

Table 3. ACS trust association interceptor custom properties (continued)	
Custom property name	**Value**
sso_1.idp_1.EntityID	https://*isam_hostname:isam_port*//*URL of ISAM*/*ISAM Junction*/*IdP endpoint*/*federation name*/saml20
sso_1.idp_1.SingleSignOnUrl	https:// *isam_hostname:isam_port*//*URL of ISAM*/*ISAM Junction*/*IdP endpoint*/*federation name*/saml20/login
sso_1.sp.targetUrl	https://*WAS_host_name*:*WAS_port*/Rest
sso_1.idp_1.certAlias	isam-conf
sso_1.sp.filter	request-url^=/Rest;request-url!=/Rest/j_security_check
sso_1.sp.enforceTaiCookie	false

3. Add the IdP federation partner data. The following substeps describe how to add the IdP data by using the WebSphere Application Server administrative console.
   a) To add the IdP host name or IP address as a trusted realm, click **Global security** > **Trusted authentication realms - inbound** > **Add External Realm**.
   b) Enter either the IBM Security Access Manager host name or IP address.
   c) To load the IdP certificate from IBM Security Access Manager, click **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates** > **Retrieve from port**
   d) Enter the IBM Security Access Manager IP address and listener port, for example, `12443, alias = isam-conf`.

   Note: When the browser first attempts to connect to the IBM Cúram Social Program Management web API, an LTPA2 cookie is sent as part of the request. If the WebSphere Application Server **`com.ibm.ws.security.web.logoutOnHTTPSessionExpire`** property is set to true, which is the default configuration in IBM Cúram Social Program Management, then authentication fails because an HTTP session does not exist on the application server. By setting the property to `false`, the check for a valid HTTP session is not completed and when the LTPA2 token is valid, authentication succeeds.

   To configure the property in the WebSphere Application Serveradministrative console, click **Security** > **Global security** > **Custom properties**, and set the value of **`com.ibm.ws.security.web.logoutOnHTTPSessionExpire`** to `false`.
4. Implement cross-origin resource sharing (CORS) from the HTTP server to the WebSphere Application Server SAML ACS.
   a) To add a CORS header, configure a servlet filter for the `WebSphereSamlSP.ear` file that is deployed by a Trust Association Interceptor (TAI). The servlet filter adds a CORS HTTP header to HTTP responses. You can archive the implemented servlet filter as a `jar` file, and then store it in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war\WEB-INF\lib` directory that is in the `installedApps` directory of your project in WebSphere Application Server.
   See the following example of how to implement a servlet filter:

```
public class SampleFilter implements Filter {

       @Override
       public void doFilter(ServletRequest arg0, ServletResponse servletResponse,
            FilterChain arg2) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) servletResponse;
HttpServletRequest request = (HttpServletRequest) arg0;


response.setHeader("Access-Control-Allow-Origin",
```

```
     "http://dubxpcvm156.mul.ie.ibm.com:9880");     <hostname or IP address of IBM UA
server>
response.setHeader("Access-Control-Allow-Credentials", "true");
response.setHeader("Access-Control-Allow-Headers", "x-requested-with, Content-Type,
origin, authorization, accept, client-security-token");
response.setHeader("Access-Control-Expose-Headers", "content-length");
        arg2.doFilter(request, response);
     }
}
```

b) Configure the `web.xml` file for the deployed TAI EAR file to use the servlet filter for all the requests. Add the filter element that is shown in the following sample to the `web.xml` file, with the actual fully qualified name of the filter.

You can add the filter element as a sibling to any existing element n the `web.xml` file, such as `<servlet>`. The `web.xml` file is in the `WebSphereSamlSP.ear\WebSphereSamlSPWeb.war\WEB-INF\lib` directory, which is in the `installedApps` directory of your project in WebSphere Application Server.

```
<filter>
    <filter-name> SampleFilter </filter-name>
    <filter-class> SampleFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name> SampleFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

Add and enable the users in LDAP
Complete the following steps to add the users from LDAP and enable them in ISAM.

Procedure

1. To create LDAP and IBM Security Access Manager runtime users, create an `ldif` file that can be used to populate OpenLdap, as shown in the following sample:

```
# cat UA_usersCreate_ISAM.ldif
dn: dc=watson-health,secAuthority=Default
objectclass: top
objectclass: domain
dc: watson-health

dn: c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: country
c: ie

dn: o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organization
o: curam

dn: ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamint

dn: cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: caseworker
sn: caseworkersurname
uid: caseworker
mail: caseworker@curam.com
userpassword: Passw0rd

dn: ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamext

dn: cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
```

```
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: jamessmith
sn: Smith
uid: jamessmith
mail: jamessmith@curamexternal.com
userpassword: Passw0rd
```

2. Add users to the OpenLDAP database:
   a) On the host server that is running the docker containers, enter the following command:

   ```
   docker cp UA_usersCreate_ISAM.ldif idpisam9040_isam-ldap_1:/tmp
   ```

   b) To log on to the OpenLDAP container, enter the following command:

   ```
   docker exec –ti idpisam9040_isam-ldap_1 bash
   ```

   c) To add the users to OpenLDAP, enter the following command:

   ```
   ldapadd -H ldaps://127.0.0.1:636 -D cn=root,secAuthority=default -f /tmp/
   Curam_usersCreate_ISAM.ldif
   ```

3. Import the users into IBM Security Access Manager:
   a) To log on to the IBM Security Access Manager command line interface, enter the following commands:

   ```
   docker exec -ti idpisam9040_isam-webseal_1 isam_cli
   isam_cli> isam admin
   pdadmin> login -a sec_master -p <password>
   ```

   b) To import the users into IBM Security Access Manager, enter the following commands:

   ```
   pdadmin sec_master> user import caseworker
   cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
   pdadmin sec_master> user modify caseworker account-valid yes
   pdadmin sec_master> user import jamessmith
   cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
   pdadmin sec_master> user modify jamessmith account-valid yes
   ```

4. To test the identity provider (IdP) flow, enter the following URL in a browser:

   ```
   https://ISAM login initial URL?RequestBinding=HTTPPost
   &PartnerId=webspherehostname:9443/samlsps/acs&NameIdFormat=Email
   &Target=WAS hostname:WAS port/Rest/v1
   ```

   Replace the following values in the URL with the appropriate values for your configuration:

   - *IBM Security Access Manager login initial URL*
   - *WebSphere host name*
   - *WebSphere Application Server host name*
   - *WebSphere Application Server port*; inIBM Cúram Social Program Management the default value is 9044

   When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by IBM Cúram Universal Access, an XHR timeout occurs before the initialization finishes. Therefore, this test step is required to ensure that the initialization of the IdP endpoints is completed.

5. In a browser, go to the home page and log in.

Test IdP-initiated SAML SSO infrastructure
When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by

Universal Access, an XHR timeout occurs before the initialization finishes. This test step is required to ensure that the initialization of the IdP endpoints is completed.

Procedure

To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://<isam_url>/isam/sps/saml20idp/saml20/logininitial?
RequestBinding=HTTPPost&PartnerId=https://<was_url>/samlsps/acs&NameIdFormat=Email&Target=<
was_url>/Rest/api/definitions
```

where:

- `<isam_url>` - The URL for IBM Security Access Manager. It consists of the IBM Security Access Manager host name, and port number, for example, `https:// 192.168.0.1:12443`.
- `<junction_name>` - The junction name that is used during the federation configuration in reverse proxy. The default value is `isam`.
- `<idp_endpoint>` - The endpoint that is configured for the IDP federation. The default value is `sps`.
- `<federation_name>` - The name that was used when creating the federation.
- WebSphere host name
- WebSphere Application Server host name
- WebSphere Application Server port. The default value is 9044 for IBM Cúram Social Program Management.

SP-Initiated only: Implementing the SAML AuthnRequest functionality in WebSphere Application Server
WebSphere Application Server does not support SP-initated SAML web SSO by default. In addition to the previous steps, you must also implement the provided `com.ibm.wsspi.security.web.saml.AuthnRequestProvider` interface to handle the AuthnRequest functionality that is needed in the service provider.

About this task

For more information, see Enabling SAML SP-Initiated web single sign-on (SSO) in the WebSphere Application Server documentation.

Procedure

1. Implement the `AuthnRequestProvider` interface as in the following example. Note that in the `getAuthnRequest` method, the `ssoUrl` variable is set to the value of the `ACSTrusAssociationInterceptor` interceptor property `sso_1.idp_1.SingleSignOnUrl`, while `acsUrl` is set to the value of the `sso_1.sp.acsUrl` property.

```
package curam.sso;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Base64;
import java.util.Date;
import java.util.HashMap;
import java.util.TimeZone;
import javax.servlet.http.HttpServletRequest;
import com.ibm.websphere.security.NotImplementedException;
import com.ibm.wsspi.security.web.saml.AuthnRequestProvider;
public class SPInitTAI implements AuthnRequestProvider {
    @Override
    public String getIdentityProviderOrErrorURL(HttpServletRequest arg0, String arg1, String
arg2,
            ArrayList<String> arg3) throws NotImplementedException {

        return null;
    }
    @Override
    public HashMap<String, String> getAuthnRequest(HttpServletRequest arg0, String arg1,
String arg2,
            ArrayList<String> paramArrayList) throws NotImplementedException {
```

```
        //create map with following keys
        HashMap <String, String> map = new HashMap <String, String>();

        String ssoUrl = "https://<isam_hostname>:<isam_port>/<URL of ISAM>/<ISAM Junction>/
<IdP endpoint>/<federation name>/saml20/login";
        String acsUrl = "https://<WAS_host_name>:<ssl port>/samlsps/acs";
        String issuer = acsUrl;
        String destination = ssoUrl;

        map.put(AuthnRequestProvider.SSO_URL, ssoUrl);
        map.put(AuthnRequestProvider.RELAY_STATE, acsUrl);
        String requestID = "Test" + Double.toString(Math.random());
        map.put(AuthnRequestProvider.REQUEST_ID, requestID);


        String authnMessageNew = "<samlp:AuthnRequest
xmlns:samlp=\"urn:oasis:names:tc:SAML:2.0:protocol\" "
                + "ID=\""+requestID+"\" "
                + "Version=\"2.0\" "
                + "IssueInstant=\""+getDateTime()+"\" ForceAuthn=\"false\"
IsPassive=\"false\" "
                + "ProtocolBinding=\"urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST\" "
                + "AssertionConsumerServiceURL=\""+acsUrl+"\" "
                + "Destination=\""+destination+"\"> "
                + "<saml:Issuer xmlns:saml=\"urn:oasis:names:tc:SAML:2.0:assertion\">"+issuer
                + "</saml:Issuer> <samlp:NameIDPolicy
Format=\"urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress\" AllowCreate=\"true\" />"
                +"<samlp:RequestedAuthnContext Comparison=\"exact\">
<saml:AuthnContextClassRef xmlns:saml=\"urn:oasis:names:tc:SAML:2.0:assertion\">"
                + "urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</
saml:AuthnContextClassRef></samlp:RequestedAuthnContext> </samlp:AuthnRequest>";


        String encodedAuth = Base64.getEncoder().encodeToString(authnMessageNew.getBytes());

        map.put(AuthnRequestProvider.AUTHN_REQUEST, encodedAuth);

            return map;
    }


    private String getDateTime() {
        // e.g 2018-11-11T23:52:45Z
        String pattern =  "yyyy-MM-dd'T'HH:mm:ss'Z'";
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);
        simpleDateFormat.setTimeZone(TimeZone.getTimeZone("Zulu"));
        String date = simpleDateFormat.format(new Date());
        return date;
    }
}
```

2. Pack your `AuthnRequestProvider` implementation in a JAR, and place it in `WAS_HOME/lib/ext`.
3. Ensure that your `AuthnRequestProvider` implementation class is added to the `ACSTrusAssociationInterceptor` custom property `sso_1.sp.login.error.page` so that it can handle errors.
   a) In the WebSphere Application Server admin console, go to **Security** > **Global Security** > **Web and Sip Security** > **Trust association** > **Interceptors** > **com.ibm.ws.security.web.saml.ACSTrustAssociationInterceptor**.
   b) Set the `sso_1.sp.login.error.page` custom property to the value `curam.sso.SPInitTAI`.
   c) Click **OK** and save the configuration.
4. You might need to restart the application server for the changes to take effect.

SP-Initiated only: Test SP-initiated SAML SSO infrastructure
Complete the following steps to test the SP-initiated SAML SSO infrastructure.

Procedure

1. Open your browser, with network devtools, and load a protected REST URL like this example:

```
<was_url>/Rest/api/definitions
```

where <was_url> is the WebSphere URL, for example https:// 192.168.0.1.

2. You are redirected to the ISAM log-in page. Log in with the credentials that were used to set the reverse proxy instance as outlined in “Configuring IBM Security Access Manager as an IdP” on page 47.
3. You should be redirected to the definitions page that you opened in step 1.

Configuring SAML SSO on Oracle WebLogic Server

Configure SAML SSO for IBM Cúram Social Program Management on WebLogic Server. If you are using IBM Cúram Universal Access, you must do some additional Universal Access configuration.

SAML SSO initiation and flow on Oracle WebLogic Server

In all SAML web SSO profile flows, the binding defines the mechanism that is used to send information through assertions between the identity provider (IdP) and the service provider (SP). For Universal Access, the SAML response by HTTP POSTs is interpreted and controlled by logic in the IBM Universal Access Responsive Web Application.

The SAML 2.0 web single sign-on (SSO) profile that is supported by WebLogic Server implements the Authentication Request Protocol along with the HTTP Redirect and HTTP POST bindings for sending web SSO profiles. The browser sends an HTTP POST request, whose POST body contains a SAML response document. The SAML response document is an XML document that contains data about the user and the assertion, some of which is optional.

Browser-based SSO through SAML v2.0 works well with many web applications where the SAML flow is controlled by HTTP redirects between the identity provider (IdP) and the service provider (SP). The user is guided from login screens to SP landing pages by HTTP redirects and hidden forms that use the browser to POST received information to either the IdP or the SP.

In a single-page application such as the Universal Access Responsive Web Application, all screens are contained in the application and dynamic content is expected to be passed only in JSON messages through XMLHttpRequests. Therefore, the rendering of HTML content for login pages and the automatic posting of hidden forms in HTML content is more difficult. If the SP processes the content in the same way, it must leave the application and hand back control to either the user agent or the browser, in which case the application state is lost.

The SSO profile has two execution flows, Service Provider initiated (SP-initiated) or Identity Provider initiated (IdP-initiated) SSO.

IdP-initiated use case

The IdP can send an assertion request to the service provider ACS in one of the following ways:

- The IdP sends a URL link in a response to a successful authentication request. The user must click the URL link to post the SAML response to the service provider ACS.
- The IdP sends an auto-submit form to the browser that automatically posts the SAML response to the service provider ACS.

The ACS then validates the assertion, creates a JAAS subject, and redirects the user to the SP resource.

SP-initiated use case

When an unauthenticated user first accesses an application through an SP, the SP directs the user's browser to the IdP to authenticate. To be SAML specification compliant, the flow requires the generation of a SAML AuthnRequest from the SP to the IdP. The IdP receives the AuthnRequest, validates that the request comes from a registered SP, and then authenticates the user. After the user is authenticated, the IdP directs the browser to the Assertion Consumer Service (ACS) application that is specified in the AuthnRequest that was received from the SP.

Assertions and the SAML Response document

To prove the authenticity of the information, the assertion in the SAML response is almost always digitally signed. To protect the confidentiality of parts of the assertion, the payload can be digitally encrypted. A typical SAML response contains information that can be sent only through a login by a POST parameter. After login, an alternative mechanism is typically used to maintain the logged-in security context. Most systems use some cookie-based, server-specific mechanism, such as a specific security cookie, or the server's cookie tied to the user's HTTP session.

SAML SSO initiation and flow diagrams

Review the flow diagram that matches your environment.

- "IdP-initiated flow for SPM in WebLogic Server" on page 55
- "IdP-initiated flow for Universal Access in WebLogic Server" on page 56
- "SP-initiated flow for SPM in WebLogic Server" on page 57
- "SP-initiated flow for Universal Access in WebLogic Server" on page 58

IdP-initiated flow for SPM in WebLogic Server

The following figure illustrates the IdP-initiated flow that is supported by SPM in a default installation.

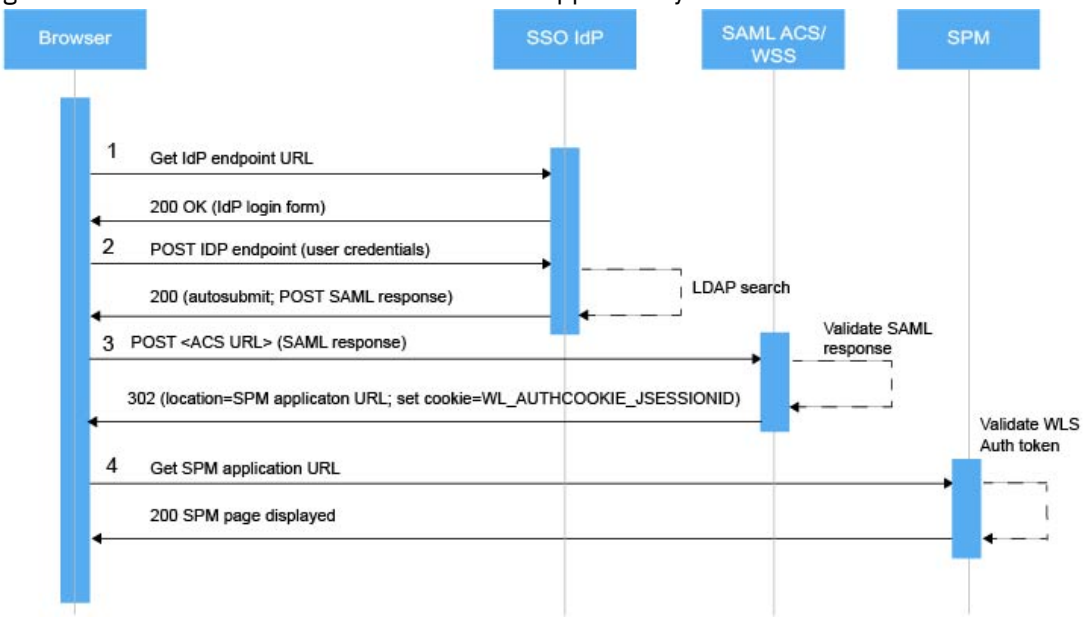

Figure 17. IdP-initiated flow for SPM in WebLogic Server

1. The user makes a request to IdP and is presented with a login web application that is hosted by an Identity Provider that authenticates the user. The Identity Provider challenges the user to enter credentials.
2. The user provides a username and password to the Identity Provider, which completes the authentication process. Typically, a user issues a request on a resource that is hosted by a Service Provider.
3. The SSO service that is hosted by the Identity Provider sends an unsolicited authentication response to the Service Provider's Assertion Consumer Service (ACS). The ACS validates the assertion, extracts the identity information, and maps that identity to a subject in the local security realm. The ACS sends an HTTP redirect message to the browser, passing a cookie that contains a session ID and enabling the browser to access the requested resource.
4. The WebLogic Security Service performs an authorization check to determine whether the browser can access the requested resource. If the authorization check succeeds, access to the resource is granted.

IdP-initiated flow for Universal Access in WebLogic Server

When Universal Access is configured with an IdP initiated web SSO flow, any attempt to connect to a protected resource without first authenticating through IdP results in a 302 HTTP response from IBM Cúram Social Program Management web API. Any authentication requests that are initiated through SP result in a 302 HTTP response to the **IdP login** page.

The following figure illustrates the IdP-initiated flow that is supported by Universal Access in a default installation.

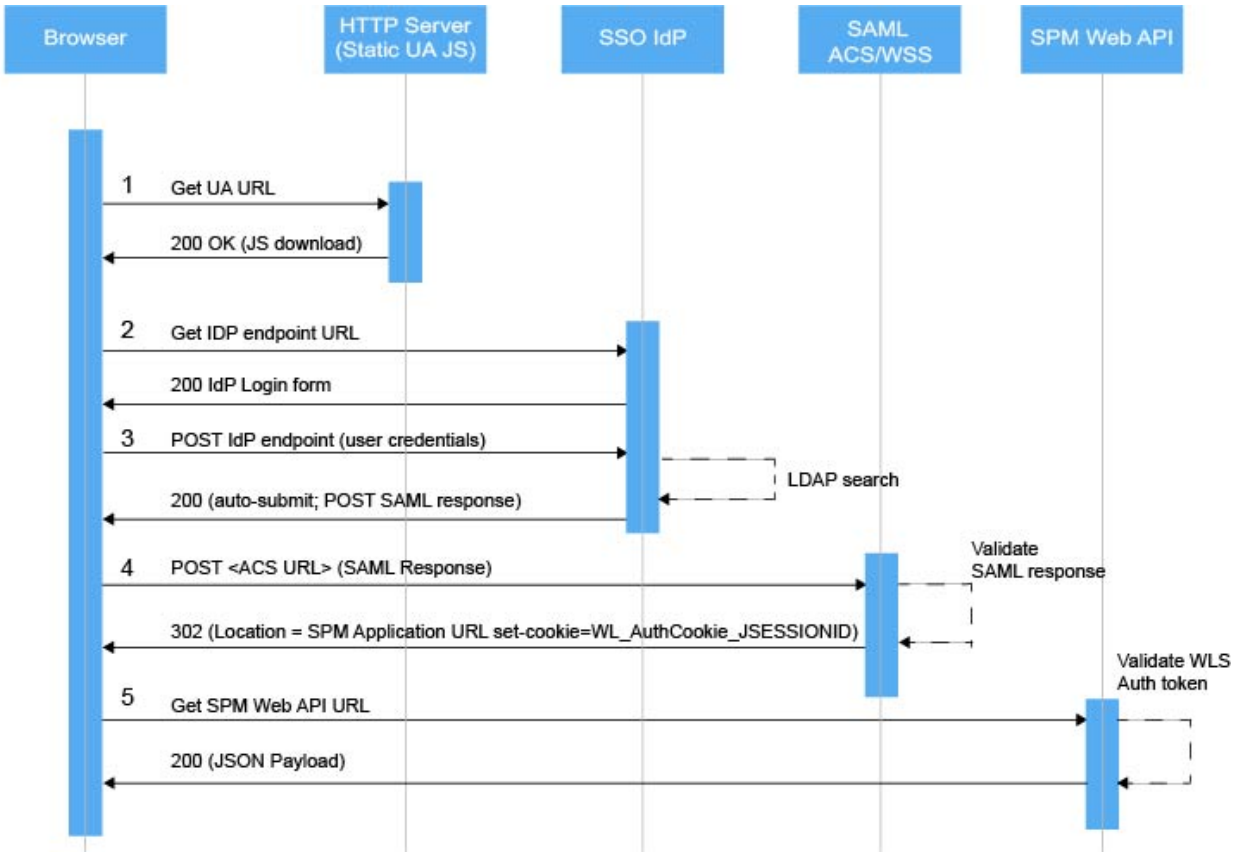

Figure 18. IdP-initiated flow for Universal Access in WebLogic Server

1. A user browses to the HTTP Server that contains Universal Access. The user can browse as normal by interacting with IBM Cúram Social Program Management as either a public or a generated user (which is not shown in the diagram).
2. The user then opens the login page to access protected content, which triggers an initial request to the IdP endpoint. In most IdP configurations, an HTML login form responds to the request. Universal Access ignores the response, and generates its own login form.
3. To authenticate, the user completes the login form and selects **Submit**. The form submission triggers an HTTP POST request that contains login credentials to the IdP. After successful validation of the user credentials at the IdP, the IdP populates the SAML Response and returns it in an HTML form that contains hidden input fields.
4. Universal Access extracts the SAMLResponse values, and inserts them in a new POST request to the application server Assertion Consumer Service (ACS). The application server ACS validates the signature that is contained in the SAML Response. If the validation is successful, the ACS sends an HTTP redirect that points to the configured IBM Cúram Social Program Management target landing page. The validation also passes a cookie that contains a session ID that is used in any subsequent communication.
5. The browser automatically sends a new request to the target URL, but Universal Access does not respond to the request. Instead, Universal Access begins its standard user setup by requesting account and profile information from the relevant web API endpoints.

SP-initiated flow for SPM in WebLogic Server

The following figure illustrates the SP-initiated flow that is supported by SPM in a default installation.

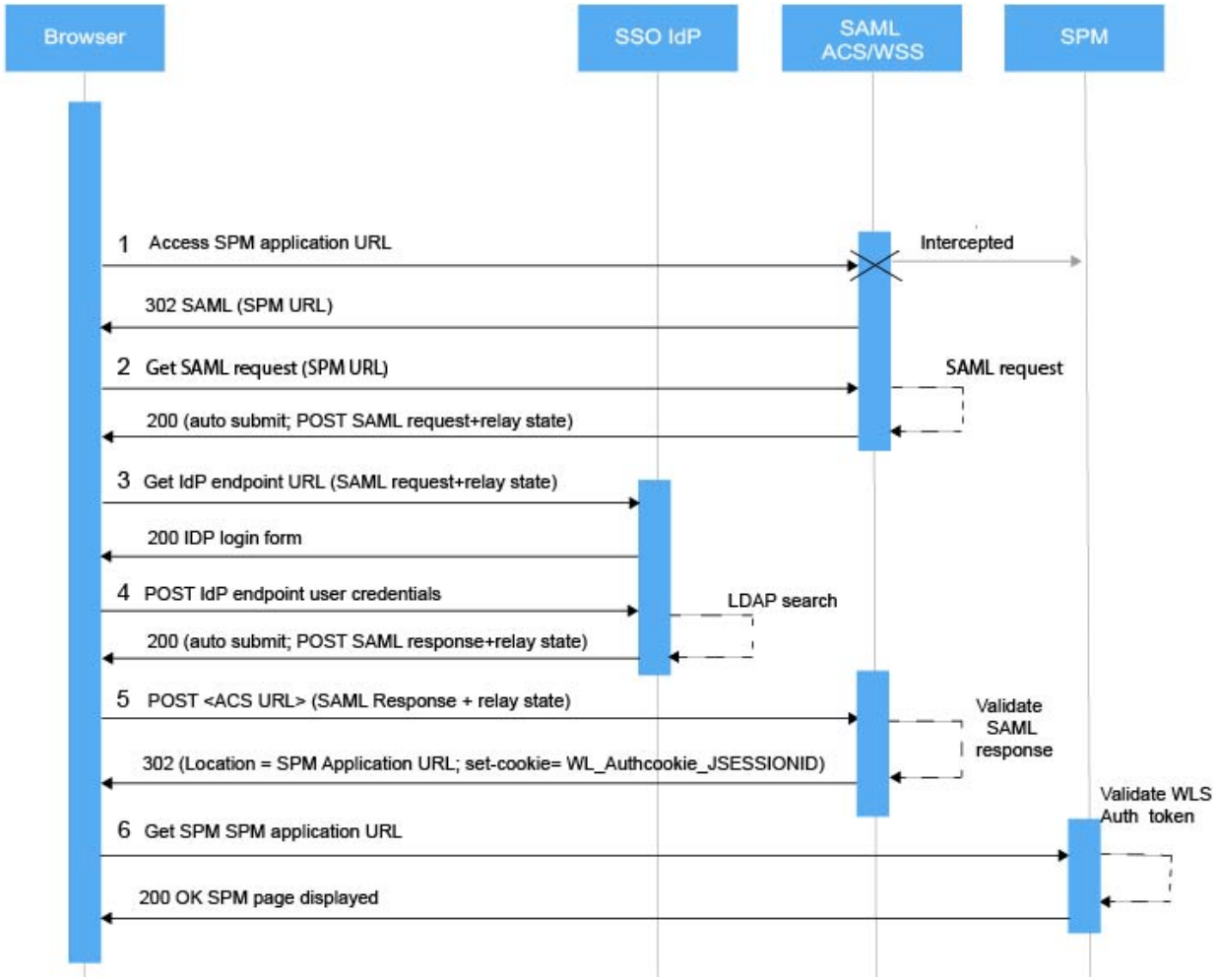

Figure 19. SP-initiated flow for Social Program Management in WebLogic Server

1. From a web browser, a user attempts to access SPM, a protected resource that runs in a WebLogic Server container that is hosted by a service provider. The container starts the WebLogic Server Security Service (WSS) to determine whether the user is authenticated.
2. Because the user is not authenticated, the service provider generates an SAML authentication request that contains information about the unauthenticated user.
3. The service provider sends the SAML request to the Identity Provider, by using the endpoint of the Identity Provider's SSO Service. The user is presented with a login web application that is hosted by an Identity Provider that can authenticate that user. The Identity Provider challenges the user for their credentials.
4. The user provides a username and password to the Identity Provider, which completes the authentication operation. The SSO Service that is hosted by the Identity Provider generates a SAML response for the user and sends this authentication response, which contains the assertion, to the user's browser.
5. The SAML response is sent to the service provider's Assertion Consumer Service (ACS) by using an auto-submit HTTP POST message. If the validation is successful, the ACS sends an HTTP redirect that points to the configured SPM target landing page, along with an authorization cookie that contains a session ID that enables the browser to access the requested resource.
6. The user accesses the requested resource.

SP-initiated flow for Universal Access in WebLogic Server

When Universal Access is configured with an SP-initiated web SSO flow, any attempt to connect to a protected resource without first authenticating results in a 302 HTTP response from the application server Assertion Consumer Service's Web Services Security. The generation of the SAML AuthnRequest message is also sent to the IdP.

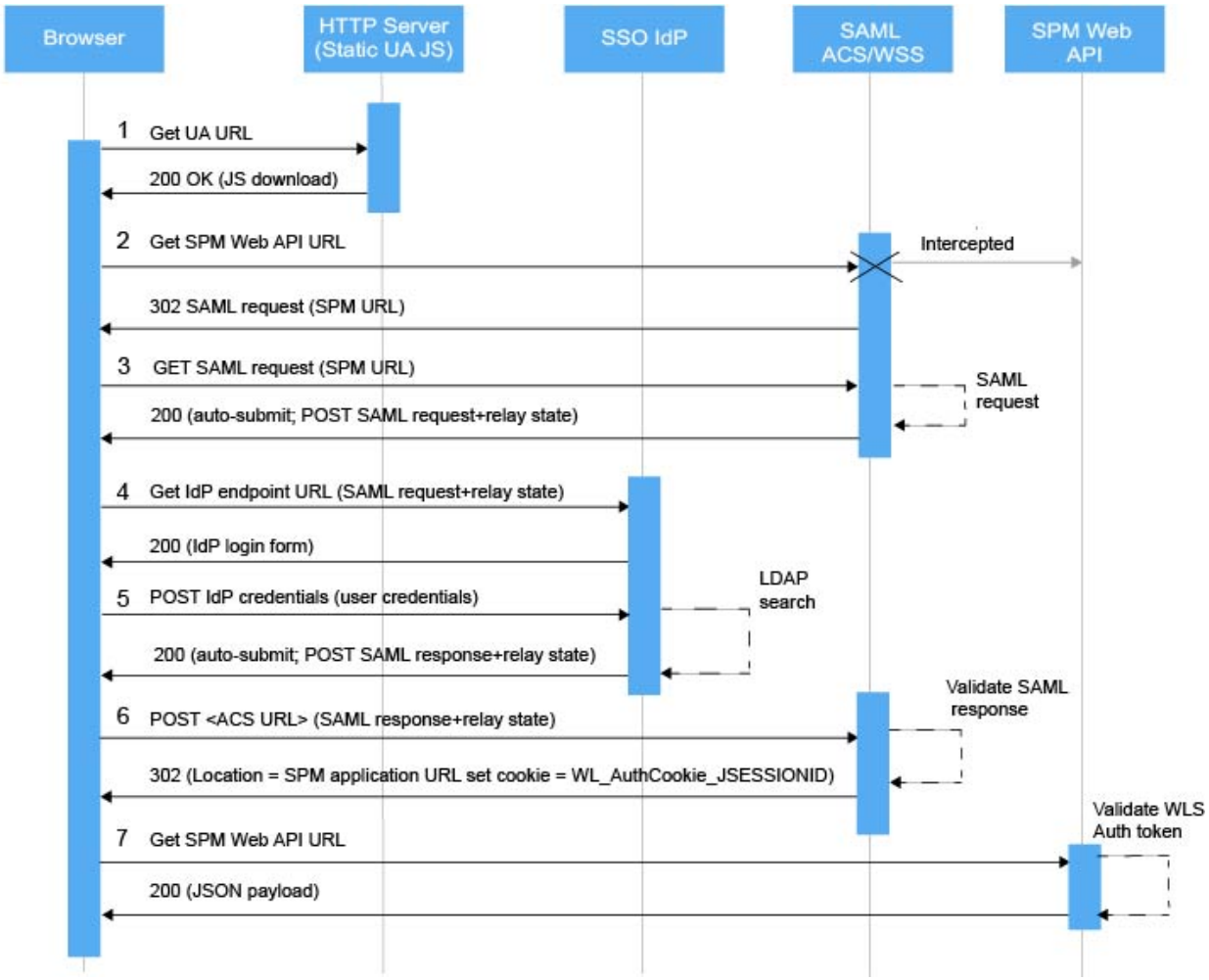

Figure 20. SP-initiated flow for Universal Access in WebLogic Server

1. A user browses to the HTTP Server that contains Universal Access. The user can browse as normal by interacting with IBM Cúram Social Program Management as either a public or a generated user (which is not shown in the diagram).
2. The user attempts to access a protected resource that starts the WebLogic Security Service (WSS) to determine whether the user is authenticated.
3. Because the user is not authenticated, the service provider generates an SAML authentication request that contains information about the unauthenticated user.
4. Universal Access directs the SAML request to the IdP SAML endpoint. The IdP responds to this request with an HTML login form, which Universal Access intercepts and extracts a hidden authentication token from login form if present, ignoring the rest of the response.
5. Universal Access constructs its own login form. The user completes this login form and selects **Submit**. The form submission triggers an HTTP POST request that contains login credentials to the IdP, along with the token extracted in the previous step if present. If authentication is successful, the Identity Provider generates a SAML response that contains the SAML assertion and returns it in an HTML form that contains hidden input fields.

6. Universal Access extracts the Relay State and SAML Response values and inserts them in a new HTTP POST message to the application server Assertion Consumer Service (ACS). If the validation is successful, the ACS sends an HTTP redirect that points to the configured SPM target landing page, along with a WebLogic Server authorization cookie. The cookie contains a session ID that enables the browser to access the requested resource.
7. The browser automatically sends a new request to the target URL, but Universal Access does not respond to the request. Instead, Universal Access begins its standard user setup by requesting account and profile information from the relevant web API endpoints.

Configuring Oracle WebLogic Server as a SAML service provider

To configure SSO for IBM Cúram Social Program Management, you must configure WebLogic Server as a SAML service provider.

WebLogic Security Providers are modules that provide security services to protect WebLogic Server resources. WebLogic Server. There are different types of WebLogic Security Providers, but here the focus is on the **Identity Assertion provider** because it supports SSO. WebLogic Server supplies several different types of Identity Assertion providers to support different token formats but the focus is on SAML SSO and the **SAML 2.0 Identity Asserter**.

In WebLogic Server, the SAML Identity Assertion Provider acts as a consumer of SAML security assertions, which enables WebLogic Server to act as a SAML destination site (Security Provider) and supports by using SAML for single sign-on.

For more information, see Oracle's WebLogic Application Server 14.1 documentation Oracle WebLogic Server.

Configuring a SAML 2.0 Identity Assertion provider

Create and configure an instance of the SAML 2.0 Identity Assertion provider in the security realm.

Procedure

1. Log in to the WebLogic Server administrator console and browse to **myrealm** > **Providers** > **Authentication**.
2. Create an Authentication Provider. Set the type to **SAML2IdentityAsserter** and provide an appropriate name, for example `SAML2_IdentityAsserter`

   Note: If you are using clustering there are more factors to consider, see Oracle's official documentation for administering security for WebLogic Server: 24: Configuring SAML 2.0 Services.

Configuring SAML 2.0 general services

Configure the SAML 2.0 general services in the WebLogic Server instance in the domain that runs SAML 2.0 services.

Procedure

1. From the administrator's console **home page**, browse to **>Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
2. For a basic configuration, make the following changes:
   a) **Replicated Cache Enabled: [false]** . For clustering see Oracle documentation.
   b) **Published site URL**. For example, `[https://<Weblogic_hostname>/<PORT>/saml2]`. This URL specifies the base URL that is used to construct endpoint URLs for the various SAML 2.0 services.
   c) **Entity ID**. For example, `[SPM_SAML_SP_Destination]` The entity ID is a human-readable string that uniquely distinguishes your site from the other partner sites in your federation.
   d) **Whether recipient check is enabled: [true]**. If enabled, the recipient of the authentication request or response must match the URL in the HTTP request.
   e) **Configuration settings for the SAML artifact cache [default values]**

3. Optional settings:
   a) Information about the local site.
   b) **TLS/SSL client authentication is required** - If enabled, SAML artifacts are encrypted when transmitted to partners.
   c) **TLS keystore alias and passphrase** - used to store and retrieve the server's private key.
   d) **Basic client authentication enabled** - Specifies whether Basic Authentication is required.
   e) **Only Accept Signed Artifact Requests** - Specifies whether requests for SAML artifacts that are received from your partners must be signed.
   f) **Keystore alias and passphrase for the key to be used when signing documents sent to your federated partners.**

Configuring SAML 2.0 service provider
Configure the SAML 2.0 Service Provider services in the Oracle WebLogic Server instance in the domain that runs SAML 2.0 services.

Procedure

1. From the administrators console **Home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 Service Provider**.
2. For a basic configuration, make the following changes:
   a) **Enabled: [true]** Allows the WebLogic Server instance to serve as a Service Provider site.
   b) **Enable binding types:** Oracle recommends enabling all the available binding types for the endpoints of the Service Provider services.
3. Optional settings:
   a) **Specify how documents must be signed**
   b) **Specify how authentication requests are managed**
   c) **Default URL** For example, `https://<Weblogic_hostname>:<PORT>/Curam or /Rest`. Unsolicited SSO responsesare redirected re to this default URL.
4. Publish the metadata file that describes your site, and manually distribute it to your Identity Provider partners.

   Note: The local site information that your federated partners need, for example the local site contact information, entity ID, published site URL, or whether TLS/SSL client authentication is required is published to a metadata file by selecting **Publish Meta Data** in the SAML 2.0 **General console** page.

   a) From the administrators console **home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
   b) Export the SP metadata into an XML file. Select **Publish Meta Data** and save to a local directory. For example, `SP_metadata.xml`.
   c) Distribute the metadata file to your federated partner (IdP) in a secure manner.
5. Cate and configure your Identity Provider partners.
   a. The configuration of Identity Provider partners is available from the WebLogic Server administration console, by using the **Security Realms** > **RealmName** > **Providers** > **Authentication** > **SAML2IdentityAsserterName** > **Management page**.
   b. Obtain Your Identity Provider Partner's metadata File by using a trusted and secure mechanism. Your partner's metadata file describes that partner site and binding support, includes the partner's certificates and keys. Copy the partner's metadata file into a location that can be accessed by each node in your domain that is configured for SAML 2.0.
   c. Create partner and enable interactions for web single sign-on, take the following steps:
   a) Specify the partner's name and metadata file:

i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"**.

ii) Select **New** > **New Web Single Sign-On Identity Provider Partner**

iii) Provide a name, for example: **SAML_SSO_IDP01**

iv) Select **idp_metadata.xml** > **Save**

b) Configure interactions between the partner and the WebLogic Server instance:

i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"** > **“Name of new partner”** > **general**.

ii) **Enable flag: [true]**

iii) **Description: SAML_SSO_IdP_01**

iv) **Whether to consume attribute information contained in assertions received from this partner [true]**

v) **Whether authentication requests sent to this Identity Provider partner must be signed**. This attribute is read-only and is derived from the partner's metadata file.

vi) Optional settings:

vii) **Identity Provider Name Mapper Class name**

viii) **Whether the identities contained in assertions received from this partner are mapped to virtual users in the security realm**

ix) **Whether SAML artifact requests received from this Identity Provider partner must be signed**

a) Configure redirect URIs as follows:

i) Browse to the **General** tab of the partner configuration page.

ii) Provide a set of URIs from which unauthenticated users are redirected to the Identity Provider partner. A URI might include a wildcard pattern, but the wildcard pattern must include a file type to match specific files in a directory `[/Curam/*] [/Rest/*]`. Refer to the Oracle documentation for more details.

b) Use the **General** tab of the **Service Provider partner configuration** page to configure Binding and Transport optional settings.

Publishing the metadata
Publish the metadata file that describes your site, and manually distribute it to your Identity Provider partners.

About this task

Note: The local site information that your federated partners need, for example the local site information, entity ID, published site URL, whether TLS/SSL client authentication is required is published to a metadata file by selecting **Publish Meta Data** in the SAML 2.0 **General console** page.

Procedure

1. From the administrators console **home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
2. Export the SP metadata into an XML file. Select **Publish Meta Data** and save to a local directory. For example, `SP_metadata.xml`.
3. Distribute the metadata file to your federated partner (IdP) in a secure manner.

Creating your Identity Provider partners
Create and configure your Identity Provider partners.

Procedure

1. The configuration of Identity Provider partners is available from the WebLogic Server administration console, by using the **Security Realms** > **RealmName** > **Providers** > **Authentication** > **SAML2IdentityAsserterName** > **Management page**.
2. Get your Identity Provider Partner's metadata file by using a trusted and secure mechanism. Your partner's metadata file describes that partner site and binding support, includes the partner's certificates and keys. Copy the partner's metadata file into a location that can be accessed by each node in your domain that is configured for SAML 2.0.
3. Create your partner and enable interactions for web single sign-on, take the following steps:
   a) Specify the partner's name and metadata file:
      i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"**.
      ii) Select **New** > **New Web Single Sign-On Identity Provider Partner**
      iii) Provide a name, for example: **SAML_SSO_IDP01**
      iv) Select **idp_metadata.xml** > **Save**
   b) Configure interactions between the partner and the WebLogic Server instance:
      i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"** > **“Name of new partner”** > **general**.
      ii) **Enable flag: [true]**
      iii) **Description: SAML_SSO_IdP_01**
      iv) **Whether to consume attribute information contained in assertions received from this partner [true]**
      v) **Whether authentication requests sent to this Identity Provider partner must be signed**. This attribute is read-only and is derived from the partner's metadata file.
   c) Optional settings:
      i) **Identity Provider Name Mapper Class name**
      ii) **Whether the identities contained in assertions received from this partner are mapped to virtual users in the security realm**
      iii) **Whether SAML artifact requests received from this Identity Provider partner must be signed**
   d) Configure redirect URIs as follows:
      i) Browse to the **General** tab of the partner configuration page.
      ii) Provide a set of URIs from which unauthenticated users are redirected to the Identity Provider partner. A URI might include a wildcard pattern, but the wildcard pattern must include a file type to match specific files in a directory `[/Curam/*] [/Rest/*]`. Refer to the Oracle documentation for more details.
   e) Use the **General** tab of the **Service Provider partner configuration** page to configure Binding and Transport optional settings

Configuring IBM Cúram Universal Access for SSO

To configure SSO for Universal Access, you must configure the Universal Access Responsive Web Application to use SSO authentication, and configure cross-origin resource sharing (CORS) for Universal Access.

Before you begin

Ensure that SPM is configured for SSO. For IBM WebSphere Application Server, see “Configuring Oracle WebLogic Server as a SAML service provider” on page 59.

Configuring the Universal Access Responsive Web Application for SSO
To enable the Universal Access Responsive Web Application to work with SAML single sign-on (SSO), configure the appropriate properties in the `.env` environment variable file in the root of the React application and rebuild the application.

About this task

- The <IdP_URL> consists of three parts: the HTTPS protocol, the IdP hostname or IP address, and the listener port number. For example, `https://192.168.0.1:12443`.
- The <ACS_URL> consists of three parts: the HTTPS protocol, the Assertion Consumer Service (ACS) hostname or IP address, and the listener port number. For example, `https://192.168.0.2:443`.

Procedure

1. Set the authentication method to SSO, see Customizing the authentication method.
2. Set the related environment variables for your SSO environment, see React environment variable reference. These properties are applicable to both identity provider (IdP)-initiated and service-provider (SP)-initiated SAML 2.0 web SSO unless otherwise stated.

Configuring CORS for IBM Cúram Universal Access
You must configure cross-origin resource sharing (CORS) for IBM Cúram Universal Access. For security reasons, browsers restrict cross-origin HTTP requests, including XMLHttpRequest HTTP requests, that are initiated in Universal Access. When the Universal Access application and the Universal Access web API are deployed on different hosts, extra configuration is needed.

About this task

Universal Access can request HTTP resources only from the same domain that the application was loaded from, which is the domain that contains the static JavaScript. To enable Universal Access to support cross-origin resource sharing (CORS), enable the use of CORS headers.

Procedure

1. Log on to the IBM Cúram Social Program Management application as a system administrator, and click **System Configurations**.
2. In the Shortcuts menu, click **Application Data** > **Property Administration**.
3. Configure the **`curam.rest.allowedOrigins`** property with the values of either the hostnames or the IP addresses of the IdP server and the web server on which Universal Access is deployed.

SAML SSO configuration example with IBM Security Access Manager
The example outlines a single sign-on (SSO) configuration for IBM Cúram Social Program Management and IBM Cúram Universal Access that uses IBM Security Access Manager to implement federated single sign-on by using the SAML 2.0 Browser POST profile. The example applies to both IdP-initiated and SP-initiated flows. Some additional steps are needed to configure SP-initiated flows.

Universal Access SSO configuration components

The following figure shows the components that are included in a Universal Access SSO configuration.

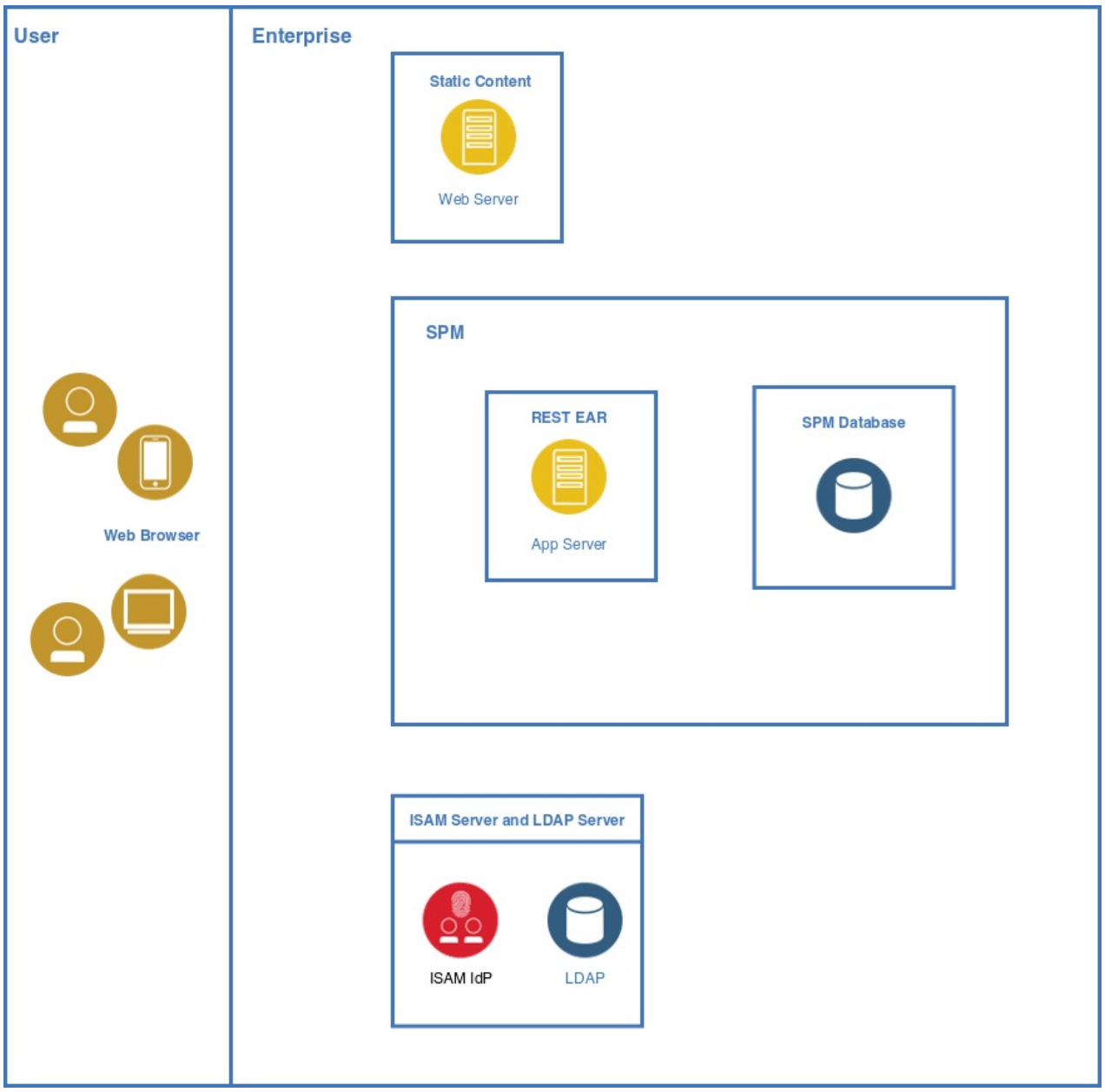

Figure 21. Universal Access SSO configuration components

Web browser
A user sends requests from their web browser for applications in the SSO environment.

Web server
The Universal Access ReactJS static content is deployed on a web server, such as IBM HTTP Server, or Apache HTTP Server.

IBM Security Access Manager (ISAM) server
The IBM Security Access Manager server includes the identity provider (IdP).

LDAP server (user directory)
Among other items, the LDAP server contains the username and password of all the valid users in the SSO environment.

Oracle WebLogic Server
Among other applications, WebLogic Server contains the deployed IBM Cúram Social Program Management, Citizen Workspace, and REST enterprise applications.

SPM database
Data storage for the IBM Cúram Social Program Management, Citizen Workspace, and REST enterprise applications.

Configuring single sign-on through IBM Security Access Manager
Use the IBM Security Access Manager management console to configure single sign-on (SSO) in IBM Cúram Universal Access.

Before you begin

1. Start IBM Security Access Manager.
2. In the management console, log on as an administrator.
3. Accept the services agreement.
4. If required, change the administrative password.

About this task

In the IBM Security Access Manager management console, complete the following steps, about the *IBM Security Access Manager 9 Federation Cookbook*.

Procedure

1. Configure the IBM Security Access Manager database:
   a) In the menu, click **Home Appliance Dashboard** > **Database Configuration**.
   b) Enter the database configuration details, such as **Database Type**, **Address**, **Port**, and click **Save**.
   c) When the **Deploy Pending Changes** window opens, click **Deploy**.
2. To install all the required product licenses, complete the steps in section *4.3 Product Activation* from the *IBM Security Access Manager 9 Federation Cookbook*.
3. Configure the LDAP SSL database by completing section *25.1.1 Load Federation Runtime SSL certificate into pdsrv trust store* from IBM Security Access Manager Federation Cookbook.
4. Configure the runtime component by completing *4.6 Configure ISAM Runtime Component on the Appliance* from IBM Security Access Manager Federation Cookbook.

Configuring IBM Security Access Manager as an IdP
To configure IBM Security Access Manager as an identity provider (IdP), see the IBM Security Access Manager 9.0 Federation Cookbook that is available from IBM Developer Works.

Before you begin

Download the IBM Security Access Manager 9.0 Federation Cookbook from IBM Developer Works, as shown in the related link. Also, download the mapping files that are provided with the cookbook.

About this task

To set up the example environment, complete the specified sections in the IBM Security Access Manager 9.0 Federation Cookbook.

Procedure

1. Complete *Section 5, Create Reverse Proxy instance*.
2. Complete *Section 6, Create SAML 2.0 Identity Provider federation*.
   In Section 6.1, if you are using the ISAM docker deployment, it is possible to reuse the existing keystore that is included in the container instead of creating a new keystore. It is important to reflect this change in subsequent sections where the myidpkeys certificate database is referenced.
3. Complete *Section 8.1, ISAM Configuration for the IdP*.

In Section 8.1, use the hostname of the IdP federation.

4. Optional: When you complete Section 8.1.1, if you require ACLs to be defined to allow and restrict access to the IdP junction, then follow the instructions in *Section 25.1.3, Configure ACL policy for IdP*.
5. Complete *Section 9.1, Configuring Partner for the IdP*.

   The export from Oracle WebLogic Server does not contain all the relevant data. Therefore, in Section 9.1, after you complete configuring partner for the IdP, you must click **Edit configuration** and complete the remaining advanced configuration.

Configuring Oracle WebLogic Server as a SAML service provider
To configure SSO for IBM Cúram Social Program Management, you must configure WebLogic Server as a SAML service provider.

WebLogic Security Providers are modules that provide security services to protect WebLogic Server resources. WebLogic Server. There are different types of WebLogic Security Providers, but here the focus is on the **Identity Assertion provider** because it supports SSO. WebLogic Server supplies several different types of Identity Assertion providers to support different token formats but the focus is on SAML SSO and the **SAML 2.0 Identity Asserter**.

In WebLogic Server, the SAML Identity Assertion Provider acts as a consumer of SAML security assertions, which enables WebLogic Server to act as a SAML destination site (Security Provider) and supports by using SAML for single sign-on.

For more information, see Oracle's WebLogic Application Server 14.1 documentation Oracle WebLogic Server.

Configuring a SAML 2.0 Identity Assertion provider
Create and configure an instance of the SAML 2.0 Identity Assertion provider in the security realm.

Procedure

1. Log in to the WebLogic Server administrator console and browse to **myrealm** > **Providers** > **Authentication**.
2. Create an Authentication Provider. Set the type to **SAML2IdentityAsserter** and provide an appropriate name, for example `SAML2_IdentityAsserter`

   Note: If you are using clustering there are more factors to consider, see Oracle's official documentation for administering security for WebLogic Server: 24: Configuring SAML 2.0 Services.

Configuring SAML 2.0 general services
Configure the SAML 2.0 general services in the WebLogic Server instance in the domain that runs SAML 2.0 services.

Procedure

1. From the administrator's console **home page**, browse to **>Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
2. For a basic configuration, make the following changes:
   a) **Replicated Cache Enabled: [false]** . For clustering see Oracle documentation.
   b) **Published site URL**. For example, `[https://<Weblogic_hostname>/<PORT>/saml2]`. This URL specifies the base URL that is used to construct endpoint URLs for the various SAML 2.0 services.
   c) **Entity ID**. For example, `[SPM_SAML_SP_Destination]` The entity ID is a human-readable string that uniquely distinguishes your site from the other partner sites in your federation.
   d) **Whether recipient check is enabled: [true]**. If enabled, the recipient of the authentication request or response must match the URL in the HTTP request.
   e) **Configuration settings for the SAML artifact cache [default values]**
3. Optional settings:

a) Information about the local site.

b) **TLS/SSL client authentication is required** - If enabled, SAML artifacts are encrypted when transmitted to partners.

c) **TLS keystore alias and passphrase** - used to store and retrieve the server's private key.

d) **Basic client authentication enabled** - Specifies whether Basic Authentication is required.

e) **Only Accept Signed Artifact Requests** - Specifies whether requests for SAML artifacts that are received from your partners must be signed.

f) **Keystore alias and passphrase for the key to be used when signing documents sent to your federated partners.**

Configuring SAML 2.0 service provider
Configure the SAML 2.0 Service Provider services in the Oracle WebLogic Server instance in the domain that runs SAML 2.0 services.

Procedure

1. From the administrators console **Home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 Service Provider**.
2. For a basic configuration, make the following changes:
   a) **Enabled: [true]** Allows the WebLogic Server instance to serve as a Service Provider site.
   b) **Enable binding types:** Oracle recommends enabling all the available binding types for the endpoints of the Service Provider services.
3. Optional settings:
   a) **Specify how documents must be signed**
   b) **Specify how authentication requests are managed**
   c) **Default URL** For example, `https://<Weblogic_hostname>:<PORT>/Curam or /Rest`. Unsolicited SSO responsesare redirected re to this default URL.
4. Publish the metadata file that describes your site, and manually distribute it to your Identity Provider partners.

   Note: The local site information that your federated partners need, for example the local site contact information, entity ID, published site URL, or whether TLS/SSL client authentication is required is published to a metadata file by selecting **Publish Meta Data** in the SAML 2.0 **General console** page.

   a) From the administrators console **home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
   b) Export the SP metadata into an XML file. Select **Publish Meta Data** and save to a local directory. For example, `SP_metadata.xml`.
   c) Distribute the metadata file to your federated partner (IdP) in a secure manner.
5. Cate and configure your Identity Provider partners.
   a. The configuration of Identity Provider partners is available from the WebLogic Server administration console, by using the **Security Realms** > **RealmName** > **Providers** > **Authentication** > **SAML2IdentityAsserterName** > **Management page**.
   b. Obtain Your Identity Provider Partner's metadata File by using a trusted and secure mechanism. Your partner's metadata file describes that partner site and binding support, includes the partner's certificates and keys. Copy the partner's metadata file into a location that can be accessed by each node in your domain that is configured for SAML 2.0.
   c. Create partner and enable interactions for web single sign-on, take the following steps:

   a) Specify the partner's name and metadata file:

      i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"**.

ii) Select **New** > **New Web Single Sign-On Identity Provider Partner**

iii) Provide a name, for example: **SAML_SSO_IDP01**

iv) Select **idp_metadata.xml** > **Save**

b) Configure interactions between the partner and the WebLogic Server instance:

i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"** > **“Name of new partner”** > **general**.

ii) **Enable flag: [true]**

iii) **Description: SAML_SSO_IdP_01**

iv) **Whether to consume attribute information contained in assertions received from this partner [true]**

v) **Whether authentication requests sent to this Identity Provider partner must be signed**. This attribute is read-only and is derived from the partner's metadata file.

vi) Optional settings:

vii) **Identity Provider Name Mapper Class name**

viii) **Whether the identities contained in assertions received from this partner are mapped to virtual users in the security realm**

ix) **Whether SAML artifact requests received from this Identity Provider partner must be signed**

a) Configure redirect URIs as follows:

i) Browse to the **General** tab of the partner configuration page.

ii) Provide a set of URIs from which unauthenticated users are redirected to the Identity Provider partner. A URI might include a wildcard pattern, but the wildcard pattern must include a file type to match specific files in a directory `[/Curam/*] [/Rest/*]`. Refer to the Oracle documentation for more details.

b) Use the **General** tab of the **Service Provider partner configuration** page to configure Binding and Transport optional settings.

Publishing the metadata
Publish the metadata file that describes your site, and manually distribute it to your Identity Provider partners.

About this task

Note: The local site information that your federated partners need, for example the local site information, entity ID, published site URL, whether TLS/SSL client authentication is required is published to a metadata file by selecting **Publish Meta Data** in the SAML 2.0 **General console** page.

Procedure

1. From the administrators console **home page**, browse to **Servers (Environment panel)** > **Admin Server** > **Federation Services** > **SAML 2.0 General**.
2. Export the SP metadata into an XML file. Select **Publish Meta Data** and save to a local directory. For example, `SP_metadata.xml`.
3. Distribute the metadata file to your federated partner (IdP) in a secure manner.

Creating your Identity Provider partners
Create and configure your Identity Provider partners.

Procedure

1. The configuration of Identity Provider partners is available from the WebLogic Server administration console, by using the **Security Realms** > **RealmName** > **Providers** > **Authentication** > **SAML2IdentityAsserterName** > **Management page**.

2. Get your Identity Provider Partner's metadata file by using a trusted and secure mechanism. Your partner's metadata file describes that partner site and binding support, includes the partner's certificates and keys. Copy the partner's metadata file into a location that can be accessed by each node in your domain that is configured for SAML 2.0.
3. Create your partner and enable interactions for web single sign-on, take the following steps:
   a) Specify the partner's name and metadata file:
      i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"**.
      ii) Select **New** > **New Web Single Sign-On Identity Provider Partner**
      iii) Provide a name, for example: **SAML_SSO_IDP01**
      iv) Select **idp_metadata.xml** > **Save**
   b) Configure interactions between the partner and the WebLogic Server instance:
      i) Browse to **Security Realms** > **myrealm** > **Providers** > **Authentication** > **"Name of the SAML identity asserter"** > **“Name of new partner”** > **general**.
      ii) **Enable flag: [true]**
      iii) **Description: SAML_SSO_IdP_01**
      iv) **Whether to consume attribute information contained in assertions received from this partner [true]**
      v) **Whether authentication requests sent to this Identity Provider partner must be signed**. This attribute is read-only and is derived from the partner's metadata file.
   c) Optional settings:
      i) **Identity Provider Name Mapper Class name**
      ii) **Whether the identities contained in assertions received from this partner are mapped to virtual users in the security realm**
      iii) **Whether SAML artifact requests received from this Identity Provider partner must be signed**
   d) Configure redirect URIs as follows:
      i) Browse to the **General** tab of the partner configuration page.
      ii) Provide a set of URIs from which unauthenticated users are redirected to the Identity Provider partner. A URI might include a wildcard pattern, but the wildcard pattern must include a file type to match specific files in a directory `[/Curam/*] [/Rest/*]`. Refer to the Oracle documentation for more details.
   e) Use the **General** tab of the **Service Provider partner configuration** page to configure Binding and Transport optional settings

Adding and enabling the users in LDAP
Add the users from LDAP and enable them in ISAM.

Procedure

1. To create LDAP and IBM Security Access Manager runtime users, create an `ldif` file that can be used to populate OpenLdap, as shown in the following sample:

```
# cat UA_usersCreate_ISAM.ldif
dn: dc=watson-health,secAuthority=Default
objectclass: top
objectclass: domain
dc: watson-health

dn: c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: country
c: ie

dn: o=curam,c=ie,dc=watson-health,secAuthority=Default
```

```
objectclass: top
objectclass: organization
o: curam

dn: ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamint

dn: cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: caseworker
sn: caseworkersurname
uid: caseworker
mail: caseworker@curam.com
userpassword: Passw0rd

dn: ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: top
objectclass: organizationalUnit
ou: curamext

dn: cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: jamessmith
sn: Smith
uid: jamessmith
mail: jamessmith@curamexternal.com
userpassword: Passw0rd
```

2. Add users to the OpenLDAP database:
   a) On the host server that is running the docker containers, enter the following command:

```
docker cp UA_usersCreate_ISAM.ldif idpisam9040_isam-ldap_1:/tmp
```

   b) To log on to the OpenLDAP container, enter the following command:

```
docker exec –ti idpisam9040_isam-ldap_1 bash
```

   c) To add the users to OpenLDAP, enter the following command:

```
ldapadd -H ldaps://127.0.0.1:636 -D cn=root,secAuthority=default -f /tmp/
Curam_usersCreate_ISAM.ldif
```

3. Import the users into IBM Security Access Manager:
   a) To log on to the IBM Security Access Manager command line interface, enter the following commands:

```
docker exec -ti idpisam9040_isam-webseal_1 isam_cli
isam_cli> isam admin
pdadmin> login -a sec_master -p <password>
```

   b) To import the users into IBM Security Access Manager, enter the following commands:

```
pdadmin sec_master> user import caseworker
cn=caseworker,ou=curamint,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify caseworker account-valid yes
pdadmin sec_master> user import jamessmith
cn=jamessmith,ou=curamext,o=curam,c=ie,dc=watson-health,secAuthority=Default
pdadmin sec_master> user modify jamessmith account-valid yes
```

4. To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://ISAM login initial URL?RequestBinding=HTTPPost
&PartnerId=webspherehostname:9443/samlsps/acs&NameIdFormat=Email
&Target=WAS hostname:WAS port/Rest/v1
```

Replace the following values in the URL with the appropriate values for your configuration:

- *IBM Security Access Manager login initial URL*
- *WebSphere host name*
- *WebSphere Application Server host name*
- *WebSphere Application Server port*; inIBM Cúram Social Program Management the default value is 9044.

When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by IBM Cúram Universal Access, an XHR timeout occurs before the initialization finishes. Therefore, this test step is required to ensure that the initialization of the IdP endpoints is completed.

5. In a browser, go to the home page and log in.

Testing IdP-initiated SAML SSO infrastructure
When the IBM Security Access Manager docker container starts, the IdP endpoints are initialized only when the first connection request is received. However, if the first connection request is triggered by Universal Access, an XHR timeout occurs before the initialization finishes. This test step is required to ensure that the initialization of the IdP endpoints is completed.

Procedure

To test the identity provider (IdP) flow, enter the following URL in a browser:

```
https://<isam_url>/isam/sps/saml20idp/saml20/logininitial?RequestBinding=HTTPPost
&PartnerId=<SP Partner Name>&NameIdFormat=Email&Target=< wls_url>/Rest/api/definitions
```

where:

- `<isam_url>` - The URL for IBM Security Access Manager. It consists of the IBM Security Access Manager host name, and port number, for example, `https:// 192.168.0.1:12443`.
- `<junction_name>` - The junction name that is used during the federation configuration in reverse proxy. The default value is `isam`.
- `<idp_endpoint>` - The endpoint that is configured for the IDP federation. The default value is `sps`.
- `<federation_name>` - The name that was used when creating the federation.
- `<SP Partner Name>` - The name configured to reference the Service Provider ACS.
- `<WLS_URL>` - The Oracle WebLogic Server host name and Port. Default port is 7002.

SP-Initiated only: Testing SP-initiated SAML SSO infrastructure
Test the SP-initiated SAML SSO infrastructure.

Procedure

1. Open your browser by using network devtools, and load a protected REST URL like this example:

   ```
   <wls_url>/Rest/api/definitions
   ```

   where `<was_url>` is the Oracle WebLogic Server URL, for example `https:// 192.168.0.1`.
2. You are redirected to the ISAM log-in page. Log in with the credentials that were used to set the reverse proxy instance as outlined in “Configuring IBM Security Access Manager as an IdP” on page 65.
3. You are redirected to the definitions page that you opened in step 1.

Customizing the login module

Create a custom security provider that includes creating a custom Cúram Java Authentication and Authorization Service (JAAS) login module.

The following advice is limited to creating a custom security provider which includes creating a custom (JAAS) login module. It is not intended to advise about integration with any specific SSO or other third party authentication mechanisms. WebLogic Server security includes many unique terms and concepts that you need to understand. You will encounter this in this documentation.

Familiarize yourself with the JAAS documentation and WebLogic Server prerequisites in the following resources:

- WebLogic Security Service Architecture
- Introduction to Developing Security Providers for WebLogic Server
- Java Authentication and Authorization Service (JAAS) Developer's Guide

JAAS login module support for authentication in a customized solution

The Cúram Java Authentication and Authorization Service (JAAS) login module might not support the authentication requirements for a particular custom solution. When you develop a custom login module, the Cúram JAAS login module must be left in place and used with identity only authentication enabled. However, if deemed necessary, the Cúram JAAS login module can be removed and replaced by a custom solution. If so, consult IBM support.

Note: The *CuramLoginModule* version is only shipped as a sample for basic authentication that supports username and password. Because IBM Cúram Social Program Management supports JAAS (Java Authentication and Authorization Service) and the *CuramLoginModule* is based on JAAS specification, you can implement your own custom login module and plug it in.

Warning: While it is possible to remove the Cúram JAAS login module completely, it needs to be noted that users must still exist in the Cúram Users database table for authorization reasons.

The Cúram JAAS login module adds new users to the Cúram Security Cache automatically, and when this Cúram JAAS login module is replaced by a custom JAAS login module, this function no longer is present. If a custom JAAS login module is replacing the Cúram JAAS login module completely, it is the responsibility of the custom JAAS login module to ensure that an update of the Security Cache is triggered when a new user is added to the database.

Replacing the Cúram JAAS login module with a custom login module

Replace the Cúram JAAS login module with a custom login module for Oracle WebLogic Server

De-registering the existing Cúram security provider

Delete the existing Cúram security provider by using the Oracle WebLogic Server administration console.

Procedure

1. Log in to the WebLogic Server administrator console and navigate to **<domain name>** > **Security Realms**.
2. Select **myrealm** in the **Realms** list.
3. Select the **Providers** tab.
4. Select the **Authentication** tab.
5. Select **myrealmCuramAuthenticator**
6. Select **Delete** > **OK**

What to do next

Make a note of the Security Provider's Admin username and password credentials, you might want to reuse these credentials when you register a new security provider.

Creating and registering a custom security provider
Create and register a custom security provider to replace the Cúram security provider that you de-registered.

The runtime classes which implement the authentication provider SSPIs and the MBean type, which you define, from what is called the security provider

Creating the authentication provider and login module runtime classes
The *authenticationProvider* exposes the services of a security provider to the Oracle WebLogic Server Security Framework. Exposing that *authenticationProvider* allows the security provider to be initialized, started, and stopped by using theWebLogic Server Administration Console to supply the custom security provider with configuration information.

Creating the authentication provider
Use authentication methods to implement the authentication provider SSPI.

Authentication methods

Initialize(ProviderMBean providerMBean, SecurityServices securityServices)
Takes as an argument a ProviderMBean. The MBean instance is created from the MBean type you generate, and contains configuration data that allows the custom security provider to be managed in the WebLogic Server environment. If this configuration data is available, use the initialize method to extract it.

getDescription()
Returns a brief textual description of the custom security provider.

shutdown()
Shuts down the custom security provider.

getLoginModuleConfiguration()
Gets information about the authentication provider's associated *LoginModule*, which is returned as an *AppConfigurationEntry*.

getAssertionModuleConfiguration()
Gets information about an identity assertion provider's associated *LoginModule*, which is returned as an *AppConfigurationEntry*.

getPrincipalValidator()
Gets a reference to the principal validation provider's runtime class. That is, the PrincipalValidator SSPI implementation.

getIdentityAsserter()
Gets a reference to the new identity assertion provider's runtime class.

Creating the JAAS LoginModule
Use the Cúram login module as a template to implement the JAAS provider SSPI you can.

The *javax.security.auth.spi.LoginModule* interface

The *javax.security.auth.spi.LoginModule* interface is as follows. In preparation, review JAAS LoginModule interface and the following methods:

public void initialize (Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options)
Initializes the *LoginModule*. It takes as arguments a subject in which to store the resulting principals, a *CallbackHandler* that the authentication provider uses to call back to the container for authentication information, a map of any shared state information, and a map of configuration options, that is, any additional information you want to pass to the Login Module.

public boolean login() throws LoginException
Attempts to authenticate and create principals for the user by calling back to the container for authentication information.

public boolean commit() throws LoginException
Attempts to add the principals that are created in the login method to the subject.

public boolean abort() throws LoginException
The abort method is called for each configured *LoginModule*, as part of the configured authentication providers. If any commits for the LoginModules fail, that is, if the relevant REQUIRED, REQUISITE, SUFFICIENT, and OPTIONAL *LoginModules* do not succeed. The abort method removes that *LoginModule's* principals from the subject, effectively rolling back the actions performed.

public boolean logout() throws LoginException
Attempts to log the user out of the system. This method also resets the subject so that its associated principals are no longer stored.

Generating an MBean type using Oracle WebLogic Server MBeanMaker
Generate an MBean type by using Oracle WebLogic Server MBean Maker.

Links to WebLogic Server content

Use the following links and the associated sample file to generate an MBean type:

- Create an MBean Definition File (MDF). See also the sample MDF file in the sampled MDF file.
- Use the WebLogic MBeanMaker to Generate the MBean Type

  Note: Custom providers and classpaths.

  Classes that loaded from `WL_HOME\server\lib\mbeantypes` are not visible to other JAR and EAR files deployed on WebLogic Server. There are a number of library dependencies. You must set the class path before using the WebLogic Server MBeanMaker to create the new MBean and JAR File (MJF) and when running your new security provider.
- Use the WebLogic MBeanMaker to Create the MBean JAR File (MJF)
- Install the MBean Type Into the WebLogic Server Environment

Configuring the custom authentication provider using the administration console
Once you have installed the MBean type on the server, restart the AdminServer and start the Administration Console

Procedure

1. Log in to the WebLogic Server administrator console and navigate to **<domain name>** > **Security Realms**.
2. Select **myrealm** in the **Realms** list.
3. Select the **Providers** tab.
4. Select the **Authentication** tab.
5. Select **New** and enter the following fields:
   - **Name**: "myrealmCuramAuthenticator"
   - **Type**: "CustomAuthenticator"
6. Select **OK**
7. Select **myrealmCuramAuthenticator** in the **Authentication Providers** list.
8. Ensure that **Control Flag** is set to **REQUIRED** and select **Save**.

What to do next

Select the **Provider Specific** tab. This tab contains settings to configure Cúram security in WebLogic Server. Use this tab to modify the security configuration.

Configuring SSO

Token-based SSO is implemented on IBM WebSphere Application Server or Oracle WebLogic Server server. SSO on WebSphere Application Server can be implemented by using the WebSphere lightweight third-party authentication mechanism (LTPA) and additional custom login modules. SSO on WebLogic Server can be implemented by using the WebLogic Server authentication provider or a custom authentication provider.

Note: Token-based SSO is tested on IBM Cúram Social Program Management only.

Configuring SSO by using IBM WebSphere Application Server LTPA

When SSO is required with WebSphere Application Server, it can be achieved by using the WebSphere Application Server lightweight third-party authentication mechanism (LTPA) and additional custom login modules. The LTPA protocol results in a token being created for an authenticated user. In WebSphere Application Server, a token is generated once credentials are added for an authenticated user. This token is then used to retrieve identity information for an authenticated user in an SSO environment.

Security is implemented as a Cúram login module within a chain of login modules set up in the application server. It is expected that at least one of these login modules be responsible for adding credentials for the user. By default, the Cúram login module adds credentials for an authenticated user. As a result of this, the configured application server user registry handled by a subsequent login module does not add credentials. The recommended approach to implementing an SSO solution is to add a custom login module somewhere along the chain of login modules.

The ability to disable the addition of credentials for an unauthenticated user is provided, thus enabling an SSO solution to be implemented.

The Cúram JAAS login module for WebSphere Application Server checks if an LTPA token exists within the application server by using the WSCredTokenCallbackImpl callback for. If this token exists and is valid, then no authentication is performed by the Cúram login module.

Credentials may be added to the user registry. Credentials include authentication information on the user logging in, including the unique identifier for the user.WebSphere Application Server checks that credentials exist for a user after all configured system login modules have executed, if the credentials exist, then the user registry is not queried. Credentials are not added by the Cúram JAAS login module if the following settings are in place:

- curam.security.check.identity.only property is set to true.
- curam.security.user.registry.enabled property is set to true.

As mentioned in “Deployment of an External Application” on page 21, there are properties relating to the type of external user that control if credentials are added to WebSphere for a specific external user type. These include:

- curam.security.user.registry.enabled.types property.
- curam.security.user.registry.disabled.types property.

These properties provide fine grained control over authentication for external user types.

In the case where the Cúram JAAS login module does not add credentials, the user registry will be queried to attempt to add credentials for the user.

Configuring SSO by using Oracle WebLogic Server WL_Token

Configure SSO by using the WebLogic Server WL_Token.

When SSO is required with WebLogic Server, it can be achieved by using the WebLogic Server authentication provider or a custom authentication provider. Consult the WebLogic Server documentation for further information on authentication providers.

WebLogic Server expects credentials/principals and the group the user belongs to, to be added by the configured authentication provider. For an SSO solution the Cúram JAAS login module does not add

credentials to the JAAS subject to allow for an alternative authentication provider to be responsible for adding credentials.

Credentials are not added if the following settings are in place:

- curam.security.check.identity.only is set to true.
- curam.security.user.registry.enabled is set to true.

As mentioned in "Deployment of an External Application" on page 21 , there are properties relating to the type of external user that control if credentials are added to WebLogic Server for a specific external user type. These include:

- curam.security.user.registry.enabled.types property.
- curam.security.user.registry.disabled.types property.

These properties provide fine grained control over authentication for external user types.

The responsibility for adding credentials is left to another authentication provider, that is, the main authentication provider for authenticating the user. In an SSO scenario, only one of the authentication providers needs to add credentials to the JAAS subject during the commit() method of the login module for a user

Other Security Considerations

Another important security concern is protecting content as it is entered, displayed, and transferred across the network for the Cúram application. The default configuration uses SSL provided by the application server to secure content as it is transferred.

In addition to this protection, industry-leading products are used during the development lifecycle to regularly monitor for security vulnerabilities in the application. Examples of such potential vulnerabilities include cross-site scripting, and SQL injection. Such threats are resolved within the infrastructure when discovered.

For the best security, customers must do similar security monitoring of their application.

SSL settings for the application

SSL is on by default for access to the web application. This ensures a secure SSL connection between the client and server and also ensures data is encrypted. SSL is turned on for the client through settings in the `web.xml` file for the web client application.

SSL is turned on at the application server level by settings in IBM WebSphere Application Server, Kubernetes WebSphere Application Server Liberty, or Oracle WebLogic Server. These settings for the application servers are done through the Cúram configuration scripts.

Important: The configuration scripts ensure SSL is turned on by default, however, this is a default configuration that must be updated and new certificates must be established for the SSL protocol.

Leave SSL on for access to the Cúram application, however depending on specific project configurations, there may be a need to turn SSL off for the application.

It is possible, but not recommended to turn off SSL. "Turning off SSL settings for the application" on page 80 should be consulted for further details.

Using Social Program Management in a secure environment

IBM Cúram Social Program Management can be used in a secure server environment (for example, FIPS 140-2), and it depends on the requirements and capabilities of that environment (for example WebSphere Application Server or Kubernetes WebSphere Liberty, or WebLogic Server FIPS configurations). However, there are a few specific areas where Social Program Management operation or configuration is required:

- When you use the DB-to-JMS feature, which is enabled by using the **curam.batchlauncher.dbtojms.notification.ssl** property, described in the *Cúram Batch Processing Guide*
- When you use the Word Integration Control, used for the FILE_EDIT widget  , documented in the *Cúram Web Client Reference Manual*, which has two aspects to consider:
  - When needing to use it with a browser in a TLS v1.2 environment, which is discussed in the "User Machine Configuration" topic of the *Cúram Web Client Reference Manual*.
  - The SP800-131a-compliant version of the supporting jar file can be used if your browser JVM supports SHA2, regardless of whether the server environment supports SP800-131a. To digitally sign the Word Integration jar for SP800-131a compliance you must build your environment by using the `enable-sha-2-signed-jars` property (e.g. `-Denable-sha-2-signed-jars=true`) when starting the Social Program Management build targets.

Client HTML error pages

Errors that occur on the client cause HTML error pages to be displayed. For debugging purposes, in the development environment, you can output a Java exception stack trace of the errors that have occurred in the HTML error pages. However, the HTML error pages that contain the Java exception stack trace are not included in the IBM Cúram Social Program Management application malicious code and filtering checks. Therefore, because the HTML error pages could potentially make the application more susceptible to injection attacks such as cross-site scripting and link injection, the Java exception stack trace should not be output in a production environment. You can use the `errorpage.stacktrace.output` client property to determine whether the Java exception stack trace is written to the HTML error pages.

The `errorpage.stacktrace.output` property is set to `false` by default. In a development environment,for debugging purposes, you can set the property value to `true`. For more information about the `errorpage.stacktrace.output` property, see *Application Configuration Properties*.

Related reference

Application configuration properties

Enabling HTTP verb permissions

Verb tampering is an attack that exploits vulnerabilities in HTTP verbs authentication and access control mechanisms. To mitigate verb tampering in your web server, configure the web server's HTTP verb permissions to limit access to only selected HTTP verbs.

About this task

Hypertext transfer protocol provides a list of methods that you can use to perform actions on a web server. Verb tampering vulnerabilities can occur when security constraints that specify HTTP verbs allow more access than intended.

In Java Platform, Enterprise Edition version 7 or later, you can limit access to only permitted HTTP verbs by configuring the web application deployment descriptor. However, the required web application deployment descriptor configuration is not supported by the Java Platform, Enterprise Edition version that the IBM Cúram Social Program Management application currently supports. As an alternative solution, you can configure the web server in your Social Program Management application deployment environment to permit only required HTTP verbs. Use the following procedure to configure both IBM HTTP Server and Oracle HTTP Server.

Procedure

- Use the following steps to enable HTTP verb permissions by using IBM HTTP Server as a gateway or filter:

  a) Check that the application is working correctly and all pages load and work as expected:

  - In a web browser, navigate to your applications URLs and inspect the network panel.

b) Log on to the web server and locate the IBM HTTP Server home directory, for example, `/opt/IBM/HTTPServer`.

c) In the `$IHS_HOME/conf.d/` directory, edit the `custom_ihs_perf.conf` file and insert configuration example 1 from the example that follows this procedure.

d) In the `$IHS_HOME/conf.d/` directory, edit the `custom_ssl.conf` file and insert configuration example 2 from the example that follows this procedure.

e) To restart the IBM HTTP Server, enter the following commands:

```
/opt/IBM/HTTPServer/bin/apachectl stop
/opt/IBM/HTTPServer/bin/apachectl start
```

f) Recheck that the application is working correctly, and that all pages load and work as previously.

g) Check that nonpermitted verbs are blocked from accessing the application.

- Use the following steps to enable HTTP verb permissions by using Oracle HTTP Server as a gateway or filter:

a) Check that the application is working correctly and all pages load and work as expected:

– In a web browser, navigate to your applications URLs and inspect the network panel.

b) Log on to the web server and locate the Oracle HTTP Server home directory, for example, `/home/oracle/Oracle/Middleware/HTTP_Oracle_Home`.

c) In the `$OHS_HOME/user_projects/domains/ohs_{domain}/config/fmwconfig/components/OHS/ohs1/` directory, edit the `moduleconf/custom_ohs_perf.conf` file and insert configuration example 1 from the example that follows this procedure.

d) In the `$OHS_HOME/user_projects/domains/ohs_{domain}/config/fmwconfig/components/OHS/ohs1/` directory, edit the `ssl.conf` file and insert configuration example 2 from the example that follows this procedure.

e) To log on as the Oracle user, enter the following command:

```
su - oracle
```

f) To restart the Oracle HTTP Server, enter the following commands:

```
$OHS_HOME/user_projects/domains/ohs_{machine_domain}/bin/stopComponent.sh ohs1
$OHS_HOME/user_projects/domains/ohs_{machine_domain}/bin/startComponent.sh ohs1
```

g) Recheck that the application is working correctly, and that all pages load and work as previously.

h) Check that nonpermitted verbs are blocked from accessing the application.

Example

Configuration example 1

Configuration example 1 works as shown in the following description:

1. Loads the Apache `mod_rewrite` module that is available in IBM HTTP Server and Oracle HTTP Server, if it is not loaded.
2. Enables the Rewrite Engine, signifying a code block to enable rewrite.
3. Applies an `If` condition on the Request method if it does not match (`!`) the Regex expression that is denoted by the string between the start (^) and end ($) delimiters, in this case the GET, POST, PUT, DELETE, or OPTIONS verbs.
4. If the condition is `true`, in that it does not match the condition HTTP verbs on the matching Regex url (`.* = all URLs`), send a 403 Forbidden response (`[F]`) while also using the pass-through flag (`[PT]`) to overwrite any IBM WebSphere Application Server plug-in.

```
```
<IfModule !mod_rewrite.c>
    LoadModule rewrite_module {path_to_modules}/mod_rewrite.so
</IfModule>
<IfModule mod_rewrite.c>
```
```

```
    RewriteEngine On
    RewriteCond %{REQUEST_METHOD} !^(GET|POST|PUT|DELETE|OPTIONS)$
    RewriteRule .* - [PT,F]
</IfModule>
```

Configuration example 2

Configuration example 2 ensures that the mod_rewrite rules also act in https protocol and not just in http protocol.

```
```
    RewriteEngine On
    RewriteOptions Inherit
```
```

Insert the previous example in the block that contains the following code:

```
```
<VirtualHost *:443>
  ...
</VirtualHost>
```
```

Customizing Authentication

You can use the following customization points and development artifacts to customize Cúram authentication.

Customizing the Login Page

The default out-of-box login screen is represented by the `logon.jsp` file located in the `lib/curam/web/jsp` directory of the Client Development Environment for Java (CDEJ). The `logon.jsp` file can be customized by creating a copy of the out-of-the-box file and placing this in a `webclient/components/<custom>/WebContent` folder, where *<custom>* represents the name of the custom web client component.

The section on Login Pages in the *Cúram Web Client Reference Manual* has guidelines on what needs to remain in place in the `logon.jsp` file and should be referenced for further details.

Applying Styling to the Login Page

Styling changes can be applied to the `logon.jsp` in the usual way, i.e., by adding the relevant CSS to any .css file in the custom component. The *Cúram Web Client Reference Manual* should be consulted for details on styling.

Enabling Usernames With Extended Characters for WebLogic Server

If the WebLogic Server application server is not being used, this section can be ignored.

If you have Cúram user names or passwords with extended characters (e.g. "üßer") WebLogic Server provides a proprietary attribute, **j_character_encoding** , which must be added to the **logon.jsp** form-based login page. The WebLogic Server documentation should be consulted for more information. The attribute must be added to the table element in the `logon.jsp` file, as shown.

```
<input type="hidden" name="j_character_encoding" value="UTF-8"/>
```

Changing the Case-Sensitivity of the Username

The curam.security.casesensitive property controls the case sensitivity of usernames. By default, this is set to `true` in the `Application.prx` file. When set to `false` in the `Application.prx` file, this will result in the authentication and authorization mechanisms ignoring the case of the username.

The *Cúram Configuration Settings* chapter in the *Cúram Server Developer's Guide* should be consulted for further details on the `Application.prx` file.

Adding Custom Verifications to the Authentication Process

To add custom verifications, the `curam.util.security.CustomAuthenticator` interface must be implemented. This interface contains one method - `authenticateUser()` . The `authenticateUser()` method is invoked for both default authentication and identity only authentication. The results of this method are expected to be an entry from the `curam.util.codetable.SECURITYSTATUS` codetable. In the case of successful authentication, the result must be `curam.util.codetable.SECURITYSTATUS.LOGIN`

For authentication failures anything, including null, can be returned. It is recommended though that another code from the `curam.util.codetable.SECURITYSTATUS` codetable be used. This codetable can be extended to include custom codes as detailed in the chapter on Code Tables in the *Cúram Server Developer's Guide*.

After the custom verifications are invoked, the authentication process will update the relevant fields on the Users database table. For example, if the result of the customized verifications is not `SECURITYSTATUS.LOGIN` the number of login failures is increased by 1, and if the break-in threshold is reached, the account will be disabled. Alternatively, if the result is `SECURITYSTATUS.LOGIN` , the login failures are reset to 0 and the last successful login field is updated.

Note: When identity-only authentication is enabled the fields of the Users database table are not updated, irrespective of the result of the custom verification.

Configuring the Custom Authenticator

To configure the application to use this custom extension, the property curam.custom.authentication.implementation in the `Application.prx` must be set to the fully qualified name of the class implementing the `CustomAuthenticator` interface.

The *Cúram Configuration Settings* chapter in the *Cúram Server Developer's Guide* should be consulted for further details on the `Application.prx` file.

Configuring Identity Only Authentication

To configure identity-only authentication, set the `curam.security.check.identity.only` property to `true` in the `AppServer.properties` file before you run the **`configure`** target. You can also set this property after the application is deployed through the application server console. For more information about configuring the application server, see the deployment guides for your application server in Deploying the application.

Adding the Cache Refresh Failure Callback Interface

The new callback class must implement the interface: `curam.util.security.SecurityCacheFailureCallback` in a class that has a public default constructor. The implementation of the callback is registered by setting the application property curam.security.cache.failure.callback to the name of the implementation class. If the property is not set, no attempt is made to invoke a callback handler.

Turning off SSL settings for the application

SSL is on by default for access to IBM Cúram Social Program Management. Enabling SSL ensures a secure SSL connection between the client and server and also ensures data is encrypted. SSL can be turned on and off for the client through settings in the `web.xml` file for the web client application, and at the application server level by settings in WebSphere Application Server, WebLogic Server, or Kubernetes WebSphere Application Server Liberty. These settings for the application servers are configured via the configuration scripts. Leave SSL on for access to the application, however depending on specific project configurations, you might need to turn SSL off for the application.

Modifying the web.xml File for the Client Application

This can be modified by changing the <transport-guarantee> from `CONFIDENTIAL` to `NONE` in the `web.xml` file. Note, this does not disable access to the web client over HTTPS, but enables additional access via HTTP. For further details on modifying the `web.xml` file, the section on *Customizing the Web Application Descriptor* in the *Cúram Web Client Reference Manual* should be referenced. An example of setting this property is shown.

```
<user-data-constraint>
         <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
```

Modifying the Application Server Configuration

Modifying the configuration for WebSphere can be done in one of two ways. The first approach below being the recommended approach.

- Use the existing non-secure port, setup by default for Web Services (recommended approach). This caters for both SSL and non-SSL connections.
  1. Navigate to Environment -> Virtual Hosts -> client_host->Host aliases
  2. Click New and enter * for host name and 9082 for port number, then click OK
  3. On the next page click Save to store your new value to the server configuration. Please note that the port 9082 corresponds to the *CuramWebServicesChain* configured in the default client application and this port is now the port that can be used to access the application using HTTP
- Reuse the current SSL port of 9044 :

  The current port can be set up as a non-secure port. The steps to do this are described in the *Cúram Deployment Guide for WebSphere Application Server* - Section A.2.11 Server Configuration - Set up port access. Follow Steps 7 to 11 inclusive. The only difference for Step 11, is that the Transport Chain Template should be set to 'WebContainer' (and not WebContainer Secure).
- Complete the below steps after following any of the above step, to turn of SSL in Global Security Settings :
  1. Navigate to Security -> GlobalSecurity ->
  2. Select Web and SIP Security -> Single Sign-On (SSO)
  3. UnTick requires SSL , then click OK, save the server configuration.

Analyzing the AuthenticationLog Database Table

All authentication attempts (both successes and failures) are logged in the AuthenticationLog database table. The following are the rows of interest on this table:

Table 4. Contents of the Authentication Log

Field	**Meaning**
timeEntered	The timestamp of the entry in the log.
userName	The username associated with the login attempt.
altLogin	Boolean indication of whether the username represents an alternate Login ID. When this column equals '1' (true) the value in the userName column is an alternate login ID as per “Alternate Login IDs” on page 3; otherwise, the userName column represents the userName from the Users or ExternalUser table.

Table 4. Contents of the Authentication Log (continued)	
Field	**Meaning**
loginFailures	The number of login failures for this user since their last successful login.
lastLogin	The date and time of the last successful login.
loginStatus	The status of the login attempt. This may be one of: • LOGIN: Successful login. • ACCDISABLE: The account has been explicitly disabled. • ACCEXPIRED: The password expiry date has been reached. • PWDEXPIRED: The number of days which the user was given to change their password has been exceeded. • BADUSER: The user does not exist. • AUTHONLY: This is used in the case of identity only authentication and indicates that only authorization verifications will be performed. • BADPWD: The specified password was incorrect. • BREAKIN: A specified number of incorrect passwords has been reached. The account is disabled. • RESTRICTED: The user is not allowed access the system at this time. • LOGEXPR: The number of login attempts which the user was given to change their password has been exceeded. • AMBIGUOUS: The specified username is ambiguous as it is a case insensitive duplicate of another username.

The `LogAdmin` API can be used to query the AuthenticationLog database table. The Java documentation for this class should be referenced for further details.

Customizing Authorization

Use this information to set up authorization for Cúram users.

Creating Authorization Data Mapping

The authorization data for a user can be set up through the use of the Data Manager (DMX files) or through the Cúram Administration screens. The *Cúram System Configuration Guide* should be consulted for details on identifying how to group security from a business perspective.

To create a new security role for a user, the security identifiers (SIDs) that the user must have access to, need to be identified. These SIDs should then be organized into groups of SIDs. The role, groups and SIDs, once identified, need to be set up on the security tables that these represent.

Security data is considered essential for the set up of a Cúram application. As such, the examples below describe adding security data to the `data/initial` directory within the component.

Creating a New Security Role

To create a new security role, a new entry must be added to the SecurityRole database table, setting the `rolename` attribute.

To do this, create/add to the SecurityRole.dmx file in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core.

Creating a New Security Group

To create a new security group, a new entry must be added to the SecurityGroup database table setting the groupname attribute.

To do this, create/add to the SecurityGroup.dmx file in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core.

Linking the Security Group to the Security Role

The security role must be linked to the security group. To do this, create a new entry in the SecurityRoleGroup table, setting the rolename and groupname attributes.

To do this, create/add to the SecurityRoleGroup.dmx file in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core.

Creating the Security Identifier (SID)

The create a new SID, an entry must be added to the SecurityIdentifier table, setting the sidname and sidtype attributes.

To do this, create/add to the SecurityIdentifier.dmx file in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core.

Linking the Security Group to the SID

To link the security group with the SID, an entry must be added to the SecurityGroupSID table, setting the groupname and sidname attributes.

To do this, create/add to the SecurityGroupSID.dmx file in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core.

Linking the Security Role to the User

To associate authorization data to a user, the security role must be linked to the user.

To do this, update the entry for the specified user in the Users.dmx file located in the %SERVER_DIR%/components/<custom>/data/initial , where <custom> is any new directory created under components that conforms to the same directory structure as components/core , setting the rolename attribute to be the rolename as specified on the SecurityRole table.

Loading Security Information onto the Database

Once all of the information has been entered in the various DMX files, the Data Manager should be used to load the DMX data onto the database. The *Data Manager* chapter in the *Cúram Server Developer's Guide* should be consulted for further details.

Creating Function Identifiers (FIDs)

When a method is made publicly accessible; by setting the stereotype to be <<facade>>, security is automatically switched on. This means a SID is automatically generated for that method and the security enabled flag for the method is set to true . The SID and its fidenabled flag are stored in the database-

independent <ProjectName>_Fids.xml file located in the /build/svr/gen/ddl subdirectory. This file is used to insert the FID information onto the database via the Data Manager.

A FID follows the naming convention of <classname>.<methodname> , and the maximum length of a FID is 100 characters. For example, for a BPO called ProductEligibility , with two methods called insertProduct and testProduct , two FIDs are created: ProductEligibility.insertProduct and ProductEligibility.testProduct.

If security for a process method is switched off at design time in the model, a SID/FID is still generated but the security enabled flag is set to false . Setting the security enabled flag to false means that no authorization check is performed for this method.

Switching Security off for a Process Method

Setting the Generate_Security option on the process method to false in the model switches off security for a process method.

If security for a process method is switched off at design time in the model, a FID is still generated but the security enabled flag is set to false . Setting the security enabled flag to false means that no authorization check is performed for this method.

Security Considerations During Development

It is important to consider the effect of these design options when implementing security during the development of a Cúram application. They are the first and last line of defense against unauthorized access to application process functionality. Generally speaking, security will be switched on for almost all process methods. Security may be switched off for a process method that does not need security, e.g., a login method that gets invoked when a user tries to login to an application. As a user has not yet been authenticated or authorized, they need access to this method in order to login, therefore switching off security for this method may be necessary.

During the initial design phase of an application the overhead of keeping the security environment "in sync" with an evolving application can be tedious. It is possible to disable the authorization check by setting the curam.security.disable.authorisation property in the Application.prx file.

warning: Warning

The curam.security.disable.authorisation property should only be turned on at design phase. This should never be set to true in a production environment.

Finally, it should be noted that once the code and scripts have been generated from a working model, the information associated with a FID cannot be changed. To change this information requires modifying the model, re-generating and re-building the database.

Controlling the Logging of Authorization Failures for the Client

By default, web client authorization failures are not recorded.

The curam.enable.logging.client.authcheck property controls whether the authorization failures encountered by the web client are logged or not. This property is false by default, meaning these failures will not be logged. When set to true a log of these authorization failures is stored on the database table AuthorisationLog . The *Cúram Server Developers Guide* , Application.prx - Dynamic properties section should be consulted for more information on this property.

Authorizing New SID Types

A server interface method is provided to enable authorization to be performed directly. This method may be added to a class that manipulates data on the conceptual element being secured by the new SID type.

curam.util.security.Authorisation.isSIDAuthorised()

A usage example of the isSIDAuthorised() method is below:

```
// The SID associated with the conceptual element
       // to be secured.
       String someSID = "someSID";

       // Get the logged in username
       String loggedUser =
         curam.util.transaction.TransactionInfo.getProgramUser();

       // Check if the user has access rights
       if (curam.util.security.Authorisation.isSIDAuthorised(
             someSID, loggedUser)) {
         // Do something sensitive that this user has rights to do
         ...
       } else {
       // Throw an exception indicating the user doesn't have
       // access to perform this action
         AppException exception
           = new AppException(MESSAGE.ERR_USER_NO_ACCESS);
         throw exception;
       }
```

Analyzing the AuthorisationLog Database Table

All authorization failures are logged in a database table called the AuthorisationLog. The following are the rows of interest on this table:

Table 5. Contents of the Authorization Log

Field	**Meaning**
timeEntered	The timestamp of the entry in the log.
userName	The username associated with the authorization attempt.
identifierName	The security identifier (SID) or functional identifier (FID) associated with the failure.

The `LogAdmin` API can be used to query the AuthorisationLog database table. The Java documentation for this class should be referenced for further details.

Customizing Cryptography

Use this information to configure and customize cryptography for Cúram.

Cipher Customization

Modification of the default cipher settings is a relatively straightforward process, but needs to be adequately planned and tested. You will require an application restart for the changes to be implemented and depending on the size and topology of your organization and deployments you need to choose a time when in-progress changes won't be an impact. Also, consider any data (e.g., properties containing encrypted passwords) managed by the Cúram Transport Manager (CTM) that will either need to be updated or managed to prevent systems from being out of sync with one another (see the *Cúram Transport Manager Guide* for more information).

Modification of the default cipher settings involves the following steps:

1. Choosing new settings for the `CryptoConfig.properties` and underlying artifacts - see “Cúram Cipher Settings” on page 15
2. Depending on the settings, you may need to perform additional steps (e.g. when modifying the keystore as per “How to Create a New Keystore” on page 86).

3. Modify the `CryptoConfig.properties` file; note the default location is `<SERVER_DIR>/project/properties`.
4. Remove any existing `CryptoConfig.jar` files (these contain `CryptoConfig.properties`) that are found in the `<JAVA_HOME>/jre/lib/ext` directory (`$JAVA_HOME/lib/ext` on IBM z/OS®). If any Cúram clients or servers are running these will need to be terminated in order to be able to deploy an updated `CryptoConfig.jar` file with the updated settings.
5. Re-encrypt the passwords in all existing property files as identified in "Cipher-Encrypted Passwords" on page 17. The Apache Ant configtest, configure, and installapp targets will place an updated `CryptoConfig.jar` file in the Java `lib/ext` directory.
6. Test and verify your changes.

Testing of your changes should include verifying any functionality that would be impacted; for example:

- Ensure the Ant configtest target still works.
- Ensure batch programs still work.
- If you utilize the Ant configure target ensure it still works.

Related topics:

- "Cúram Digest Settings" on page 16
- "Cipher-Encrypted Passwords" on page 17

Key Management

The management of the secret key for Cúram encrypted passwords is done via the JDK-provided **keytool** command, or equivalent. You will need to make local decisions about placement and isolation of the secret key for Cúram that are compatible with your local organization and standards.

Keep in mind that some settings passed to the **keytool** command need to be reflected in the `CryptoConfig.properties` settings, which needs to be coordinated for successful deployment as discussed in "Cipher Customization" on page 85. The following table shows the relationship between **keytool** command arguments and the Cúram crypto properties.

Table 6. Relationship of keytool Command Arguments to Cúram Crypto Properties

Keytool argument	**CryptoConfig.properties property**
-keyalg	curam.security.crypto.cipher.algorithm
-alias	curam.security.crypto.cipher.keystore.seckey.alias
-keystore	curam.security.crypto.cipher.keystore.location
-storepass	curam.security.crypto.cipher.keystore.storepass

Note: The secret key password defaults to the storepass password and should not be changed.

See the JDK documentation for more information on using the **keytool** command.

Related topics:

- "Cúram Cipher Settings" on page 15
- "Cryptography Properties" on page 15
- "How to Create a New Keystore" on page 86

How to Create a New Keystore

Creating a new keystore to replace the Cúram default requires running the **keytool** command provided with the JDK (or equivalent), modifying the CryptoConfig.properties settings to correspond (necessary,

only if the keystore name and/or location is changed from the default, but changing the name can make your customizations more obvious), and ensure the Curam Ant targets can find the new keystore (necessary, only if the default location is changed).

For example:

```
keytool -genseckey -v  -alias MySecretKey -keyalg AES -keysize 128
-keystore MyOrganization.keystore -storepass secretpw -storetype jceks
```

The section "Key Management" on page 86 identifies the **keytool** command arguments that relate to the `CryptoConfig.properties` settings.

The default location of the keystore file is the <SERVER_DIR>/project/properties directory with a sub-directory structure that reflects the JDK in use: "ibm" for the IBM JDK and "sun" for the Oracle JDK. So, when creating a keystore file the Curam build scripts expect to find it in the case of the IBM JDK in: <SERVER_DIR>/project/properties/ibm. If you desire to use a location different from the default you can do one of two things:

1. Use an absolute location for the keystore file as described in "Cryptography Properties" on page 15. In this case the Curam default keystore files in `CryptoConfig.jar` will be ignored in favor of the absolute setting `CryptoConfig.properties`.
2. Use the Ant `crypto.prop.file.location` property when you run any of the targets, described in "Cipher Customization" on page 85, that create and copy the `CryptoConfig.jar` to point to your alternate location. The location specified will have to reflect the structure of your JDK - "ibm" or "sun". For instance:
   - Place the new keystore file in a location like this on Windows for the IBM JDK: `C:\Curam\keystore\ibm\MyOrganization.keystore`
   - Point to that location when running the build targets: `ant configure -Dcrypto.prop.file.location=C:\Curam\keystore`

Note: In the example above the change of keystore file name to `MyOrganization.keystore` will require a corresponding change to `CryptoConfig.properties` as per "Cryptography Properties" on page 15.

Note: The only supported keystore type for Cúram cryptography is jceks.

Following the keystore creation you need to follow the steps in "Cipher Customization" on page 85.

Related topics:

- "Key Management" on page 86
- "Cipher Customization" on page 85

Digest Customization

Modification of the default digest settings is a relatively straightforward process, but needs to be adequately planned and tested. You will require an application restart for the changes to be implemented and depending on the size and topology of your organization and deployments you need to choose a time when in-progress changes won't be an impact. Also, consider any data (e.g., User passwords) managed by the Cúram Transport Manager (CTM) that will either need to be updated or managed to prevent systems from being out of sync with one another (see the *Cúram Transport Manager Guide* for more information).

The process is covered in detail in "How to Utilize the Superseded Digest Settings for a Period of Migration" on page 88.

Related topics:

- "Cúram Digest Settings" on page 16
- "How to Specify a Digest Salt" on page 88

How to Specify a Digest Salt

While Cúram doesn't specify one out-of-the-box, specifying a salt for digested passwords provides an additional level of protection against brute-force attacks.

To specify a salt for your digested passwords:

1. Choose a sufficiently long and random string.
2. Encrypt this string using the Ant **encrypt** target (as documented in the *Cúram Server Developer's Guide*).
3. Place the encrypted string in a file.
4. Specify the location of the file containing the encrypted salt string using the `curam.security.crypto.digest.salt.location` property in `CryptoConfig.properties` and ensure that any deployed `CryptoConfig.jar` files reflect the updated settings.

For manageability you should make these changes in conjunction with the steps in "How to Utilize the Superseded Digest Settings for a Period of Migration" on page 88.

How to Utilize the Superseded Digest Settings for a Period of Migration

Utilizing the superseded digest settings means you are migrating your existing digested passwords to a new crypto configuration (e.g. new salt) and would like Cúram user passwords automatically migrated for a period of time. This applies to Cúram internal and external users, but does not apply to users managed by third-party security systems such as LDAP.

The process to do this is:

1. Choose a time when your Cúram system can be down and with the Cúram system not running.
2. Copy the existing digest property names and values in `CryptoConfig.properties` and rename the properties to the new superseded property names.
3. Modify the existing digest property names in `CryptoConfig.properties`.
4. Set the `curam.security.convertsupersededpassworddigests.enabled` property to '`true`'.
5. Set the `curam.security.crypto.upgrade.start` property to help you track when you introduced the updated configuration. This value can be used below to help manage unmigrated user passwords.
6. Restart the application server, but note the following.

Note: The Cúram default web services user (WEBSVCS), or any user not processed via the `CuramLoginModule`, is not available for automatic password migration. You must reset these users before restarting the application server. To do this:

1. Obtain the new digest password value via the Ant digest target (e.g. `ant digest -Dpassword=password`).
2. Update the password value in the database, which is easily done via SQL (e.g. `UPDATE USERS SET PASSWORD='<new digest value>' WHERE USERNAME='WEBSVCS';`).
3. You can now start the application server

After a period of time (e.g. weeks or months) when you consider the migration period to be over set the `curam.security.convertsupersededpassworddigests.enabled` property to '`false`' and unset the `curam.security.crypto.upgrade.start property`.

Users who did not login during the migration period will now see their logins fail due to password mismatches. You have two approaches for addressing the passwords not updated during the migration period:

1. Require these users to contact your internal support to have their password reset via the admin user interface.
2. Manually identify the users in the Cúram USERS table who were not updated during the migration period and either manually set new default password either via SQL (see the **digest** target described in the *Cúram Server Developer's Guide* to obtain new digest password values)

or via the admin user screens. For example, using the following query: `SELECT username FROM users WHERE lastwritten between timestamp('2013-06-01 15:00:00') AND timestamp('2013-09-01 00:00:00')`

You should not leave `curam.security.convertsupersededpassworddigests.enabled` set to true indefinitely because:

1. It's meaningless to have gone to the trouble of upgrading from configuration 'A' to configuration 'B' and leave the original 'A' configuration active;
2. It leaves potentially weaker crypto settings active in the system; and
3. In order to use this functionality for a future upgrade, say from configuration 'B' to 'C', you would have to have upgraded all the 'A' passwords to at least 'B'.

Note: Any files, e.g. DMX, with stored digests need to be considered with respect to your migration strategy so they reflect the correct values.

Note: Any use of the Cúram Transport Manager (CTM) during a migration needs to be considered in terms of ensuring compatible settings and expectations between the source and target systems.

Related topics:

- "Cúram Cipher Settings" on page 15
- "Cúram Digest Settings" on page 16

Modifying Your Crypto Configuration for a Production System

While the out-of-the-box (OOTB) crypto settings are adequate for typical development or test environments, they should be modified for production environments to protect and provide isolation between these relatively low-risk environments and high-risk production environments.

Some typical changes to the OOTB crypto configuration, in preparation for production, might include:

- Providing a new secret key.
  - Such a key can be generated using the JDK keytool utility; see "How to Create a New Keystore" on page 86
    - This secret key should be stored in a separate keystore.
    - The properties for these secret key changes would be as described in "Key Management" on page 86.
- Providing new digest settings
  - New digest settings can include a new salt, iteration count, and/or algorithm.
    - The properties for these digest changes would be as described in "Cúram Digest Settings" on page 16 and "How to Specify a Digest Salt" on page 88 and the process described in "How to Utilize the Superseded Digest Settings for a Period of Migration" on page 88.

Remember to keep your configuration files isolated from personnel who do not absolutely have to access; specifically, keeping development, test, and production configuration information isolated.

Customizing External User Applications

Use this information to customize external user applications. As external users are processed differently to internal users, a separate Cúram web application is required specifically for external users.

Creating an External User Application

A new web client application must be developed for external users. The *Cúram Web Client Reference Manual* should be consulted for details on creating a new web client application.

Creating an External User Client Login Page

A new logon.jsp must be created for an external user application. The Cúram Platform ships with a default login page, `logon.jsp` , located in the `lib/curam/web/jsp` directory of the CDEJ (Client Development Environment for Java). This file should be copied to a `webclient/components/<custom component>/WebContent` folder in the web client application and modified as follows:

The `table` element should be extended to include a hidden input field user_type:

```
<input type="hidden" name="user_type"
         value="EXTERNAL"/>
```

Where EXTERNAL indicates the type of external user. This can be set to any value, excluding `INTERNAL`.

Creating an External User Client Automatic Login Page

Some external user client applications require no user authentication and hence a username and password should not be requested. It is not possible to disable authentication in Cúram , so the best way to achieve this requirement is to write an automatic login script.

The automatic login script takes a hard coded username and password and provides that as the authentication information when requested. This means that all users for such an application will always execute under the same username. Use of such a script should be limited to true open access applications.

When implementing applications that have a need for an automatic login, the implications for session management must be considered. Session management in Cúram maintains a user's session information to ensure when the user logs back in, the relevant session information, i.e., their tabs and navigation opens to where they left off for them. In the case of a user that has been automatically logged in, this information must not be maintained, therefore session management may need to be turned off in this scenario. The *Cúram Web Client Reference Manual* should be referenced for further details on how to turn this off.

The following are examples of automatic login and logout JSP scripts.

Note: Security implementations and configurations differ across application server vendors so these examples may not work in all cases or for all application server versions.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:prefix="URI"
  version="2.0">
  <jsp:directive.page buffer="32kb"
                      contentType="text/html; charset=UTF-8"
                      pageEncoding="UTF-8" />
  <jsp:text>
    <![CDATA[
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
  </jsp:text>

  <!-- Automatic redirect to login security check of user
          details specified below -->

  <html>
    <head>
      <script type="text/javascript">
        function autoSubmit() {
          document.getElementById("loginform").submit();
        }
      </script>
      <meta content="text/html; charset=UTF-8"
            http-equiv="Content-Type" />
    </head>
    <body class="logonBody"
          style="visibility: hidden;"
          onload="autoSubmit()">
      <form id="loginform"
            name="loginform"
            action="j_security_check"
```

```
            method="post">
        <input type="hidden"
               name="j_username"
               value="generalpublic" />
        <input type="hidden"
               name="j_password"
               value="password" />
        <input type="hidden"
               name="user_type"
               value="EXTERNAL" />
      </form>
    </body>
  </html>
</jsp:root>
```

Automatic Logout JSP

```
<?xml version="1.0" encoding="UTF-8"?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:prefix="URI"
  version="2.0">
  <jsp:directive.page buffer="32kb"
                      contentType="text/html; charset=UTF-8"
                      pageEncoding="UTF-8" />
  <jsp:text>
    <![CDATA[
      <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">]]>
  </jsp:text>
  <html>
    <head>
      <script type="text/javascript">
        function autoSubmit() {
          document.getElementById("logout").submit();
        }
      </script>
      <meta content="text/html; charset=UTF-8"
            http-equiv="Content-Type" />
    </head>
    <body class="logoutBody"
          style="visibility: hidden;"
          onload="autoSubmit()">
      <form id="logout"
            name="logout"
            action="servlet/ApplicationController"
            method="post">
        <input type="submit"
               name="j_logout"
               value="Log Out" />
        <input type="hidden"
               name="logoutExitPage"
               value="redirect.jsp" />
      </form>
    </body>
  </html>
</jsp:root>
```

Extending the Public Access User Class

To "hook" the custom solution into the application the `curam.util.security.PublicAccessUser` abstract class must be extended, which requires implementing the `curam.util.security.ExternalAccessSecurity` interface. That concrete class will be used during the authentication and authorization process to determine required information relating to the external user. This class and its methods are described in detail below.

Authenticating an External User

The `authenticate()` method is responsible for authenticating an external user. It is invoked during the authentication process if the user is identifier as an external user. In the case of external users this method is invoked in place of the configured authentication.

Note: If an alternative authentication mechanism, e.g. LDAP, is configured, the external users must be able to authenticate against this mechanism.

```
/**
   * The implementation of this method should validate the identifier and
   * password and return the result of the validation. If the information is
   * valid, the codetable code SecurityStatus.LOGIN should be returned.
   *
   * @param identifier The identifier of the external user.
   * @param password The password as array of characters.
   * @param userType The type of external user.
   *
   * @return The status of the authentication in the form of a codetable code.
   *
   * @throws AppException Generic Exception Signature.
   * @throws InformationalException Generic Exception Signature.
   */

  public abstract String authenticate(String identifier,
    char[] password, String userType)
    throws AppException, InformationalException;
```

The input parameters to the method include an identifier, the digested password as an array of characters, and the type of the external user to be authenticated.

The `userType` parameter is intended to allow for support of multiple types of external users that require different authentication mechanisms. The use of this parameter depends on the custom implementation.

The expected result of this method will be an entry from the `curam.util.codetable.SECURITYSTATUS` codetable. In the case of successful authentication the result must be:

```
curam.util.codetable.SECURITYSTATUS.LOGIN
```

For authentication failures this codetable contains a number of entries, including BADUSER , BADPWD and PWDEXPIRED . This codetable can be extended to include custom codes as detailed in the *Cúram Server Developer's Guide*.

The authentication result returned by this method is automatically logged in the `AuthenticationLog` database table. For more information on this table see the *Cúram Server Developers Guide*.

The abstract class `PublicAccessUser` also defines the following abstract methods that any concrete subclass must implement:

- Method `upgradeSafePasswordValidation()` is required to allow for password comparison and is defined as follows:

  ```
  public final boolean upgradeSafePasswordValidation(
  final String userName,
  final String storedPasswordHash,
  final String plaintextPassword)
  ```

- Method `setPassword()` is to allow the implementor to persist the password (e.g. a new password) in the case of crypto upgrades. So this method gets called when the `upgradeSafePasswordValidation()` method is called. Here is the method definition:

  ```
  public abstract void setPassword(String username, String hashedPassword)
  throws AppException, InformationalException;
  ```

See the associated Javadoc of the `PublicAccessUser` class for more details regarding the above methods.

Determine External User Details

Details for an external user are retrieved by calling the `getLoginDetails()` method of the `curam.util.security.ExternalAccessSecurity` interface. These details are returned directly after authentication to direct the external user to the correct application homepage.

```
/**
 * The implementation of this method should retrieve the
 * details of the user required to redirect them to the correct
 * application page. This information includes the name of the
 * application home page for the user, the default locale for
 * the user and a list of warnings/messages for the user.
 *
 * @param identifier The identifier of the external user.
 *
 * @return The user details, including the application
 *         home page.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
UserLoginDetails getLoginDetails(String identifier)
  throws AppException, InformationalException;
```

An instance of the `curam.util.security.UserLoginDetails` class must be created and returned from this method. The following information should be returned using this class:

- `UserLoginDetails.setApplicationCode(String code)`

  The code corresponding to the application homepage for the external user.

  This must be a valid entry in the `APPLICATION_CODE` codetable.
- `UserLoginDetails.setDefaultLocale(String defaultLocale)`

  The default locale for the external user.

  This is the locale the application will be displayed in by default for the external user.
- `UserLoginDetails.setFirstName(String firstName)`

  The first name of the external user.

  This will make the user's first name available for display in the user-message for an application banner.
- `UserLoginDetails.setSurname(String surname)`

  The surname of the external user.

  This will make the user's surname available for display in the user-message for an application banner.
- `UserLoginDetails.addInformationals(InformationalManager informationalManager)`

  Any informationals that must be displayed to the external user.

  The `curam.util.exception.InformationalManager` class can be used to create a number of informational or warning messages that will be displayed when the external user logs in. For example, a warning to let the external user know that their password is due to expire.

Authorizing an External User

The `getSecurityRole()` method is used during authorization to determine the security role associated with the external user. The security roles used for external users are configured in the same way as the security roles for internal users.

```
/**
 * The implementation of this method should return the security
 * role associated with the external user for authorization
 * purposes. If the user does not exist null should be
 * returned.
 *
 * @param identifier The identifier of the external user.
 *
```

```
  * @return The security role for authorization.
  *
  * @throws AppException Generic Exception Signature.
  * @throws InformationalException Generic Exception Signature.
  */
 String getSecurityRole(String identifier)
   throws AppException, InformationalException;
```

The SDEJ will invoke an implementation of this method during the authorization process if the user does not exist in the security cache. Only internal users can exist in the security cache. This means that the identifiers used to identify external users must be unique and not conflict with usernames setup for internal users, unless the custom `UserScope` interface as described in “User Scope” on page 21 , is implemented. Otherwise, if any usernames conflict the access rights assigned to the internal user will also be used for the external user.

If a role cannot be determined for the external user, null must be returned so that the SDEJ can report the authorization error correctly.

Determining the User Type

The `getUserType()` method is used to determine if a user is an external user.

```
/**
 * Return the type of the user. This is to allow support for
 * different types of external user. If there is only one
 * type of external user, simply return "EXTERNAL".
 *
 * @param identifier The identifier of the external user.
 *
 * @return The type of the external user.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
 String getUserType(final String identifier)
   throws AppException, InformationalException;
```

The `getProgramUserType()` in `curam.util.transaction.TransactionInfo` will invoke this method to return the type of user if the user is not recognized as an internal user. For internal users "INTERNAL" is always returned.

For external users, there may be multiple types of external users, so this method should return the specific type of external user.

Preventing the Deletion of a Security Role: Role Usage Count

The `getRoleUsageCount()` method is used to prevent the deletion of a security role that is currently referenced by an external user.

```
/**
 * Return the number of users using a particular role. This
 * method is used to ensure that a role cannot be deleted when
 * it is in use by an external user.
 *
 * @param role The security role name.
 *
 * @return The number of users currently using the
 *         specified role.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
 int getRoleUsageCount(String role)
   throws AppException, InformationalException;
```

Security roles that are referenced by any user, internal or external, cannot be removed. This method should return a number of 1 or more if any external users reference the specified role.

Retrieving a Registered Username

The `getRegisteredUserName()` method is used retrieve the correct case username, which may be independent of the username typed during login.

```
/**
 * Gets the correct casing for this user independent of mixed
 * case which may have been typed in by the logged in user.
 *
 * @param identifier The identifier of the external user,
 * whose casing may not match that of the persisted identifier
 * for the user.
 *
 * @return The actual case for this user, before its case has
 * been modified by external factors.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
 public String getRegisteredUserName(final String identifier)
   throws AppException, InformationalException;
```

The default implementation for this method should return the username that has been provided. It is only if the curam.security.casesensitive has been set to false that this method may need to change the case of the username returned.

Note: Where the curam.security.casesensitive property has been set to false and is required for external users, it is the responsibility of all methods in this interface to handle any case specific requirements.

Reading User Preferences

The `getUserPreferenceSetID()` method is used to retrieve the user preference set ID associated with an external user. If no user preferences exist for an external user, then the default preferences will be used for the external user. The *User Preferences* chapter in the *Cúram Server Developer's Guide* should be referenced for further details on user preferences.

```
/**
 * This method is used to retrieve a set of user preferences
 * associated with an external user. The userPrefSetID is a
 * foreign key to the UserPreferenceInfo table.
 * The UserPreferenceInfo table contains information on
 * the user preferences.
 *
 * @param identifier The identifier of the external user.
 *
 * @return The userPrefSetID for the external user.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
 String getUserPreferenceSetID(final String identifier)
   throws AppException, InformationalException;
```

The default implementation for this method should return the user preference set ID for the user preferences associated with an external user.

Modifying User Preferences

The `modifyUserPreferenceSetID()` method is used to update the external user details with a new set of user preferences. Please see User Preferences for further details on user preferences.

```
/**
 * This method updates the external user details with new user
 * preferences.
 *
 * @param userPreferenceSetID The ID for the user preferences.
 * @param username The identifier of the external user.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
```

```
void modifyUserPreferenceSetID(
 final String userPreferenceSetID, final String username)
    throws AppException, InformationalException;
```

The default implementation for this method should update the user preference set id associated with an external user.

Configuring External Access Security

The `curam.custom.externalaccess.implementation` property must be set in the `Application.prx` to indicate the fully qualified name of the class which implements the above interface.

Note: The curam.custom.externalaccess.implementation property is not dynamic, and if changed the application must be restarted before the change will take effect.

Determining if a User is Internal or External using the UserScope Interface

To support alternative methods for determining if a user is internal or external the custom interface `UserScope` is available. For example, even though usernames must be unique across the set of internal and external users, this custom interface can be implemented to allow duplicate usernames across internal and external applications in a limited way.

To provide a custom implementation for determining the type of user, the `curam.util.security.UserScope` interface must be implemented. This interface has one method `isUserExternal()` that determines the type of user. This method should return true if the user is considered external or false indicating the user is internal.

For example, an installation might have application1 deployed with userA, a Cúram internal user, and application2 deployed with userA being external (e.g. defined to LDAP). The ability for application1 to use internal userA and application2 to use external userA would be controlled by different properties. That is, `Bootstrap.properties` in `properties.jar` in the application1 EAR would have a different custom property setting from application2 EAR and the implementation of `curam.util.security.UserScope.isUserExternal()` would interrogate this setting to decide if the user is internal or external.

To specify a custom implementation of the `UserScope` interface the `curam.custom.userscope.implementation` property must be set in `Application.prx`. This should be set to the fully qualified name of the class that implements the `UserScope` interface.

Note: The `curam.custom.userscope.implementation` property is not dynamic, and if changed the application must be restarted before the change will take effect.

The `isUserExternal()` method of the `UserScope` interface is detailed in “User Type Determination” on page 96.

User Type Determination

The `isUserExternal()` method is invoked anywhere in the application where the type of user is to be determined. This includes when the user logs into the application and when they attempt authorization to access secured elements of Cúram .

```
/**
 * The implementation of this method should determine the type of
 * User that is logged into the application. There are 2 types of
 * users: INTERNAL and EXTERNAL. If the user is an EXTERNAL user,
 * then this method should return true. If false is returned,
 * then the user is considered INTERNAL.
 *
 * @param username - The username.
 * @return A boolean value of true indicating an EXTERNAL user,
 * false indicates an INTERNAL user.
 *
 * @throws AppException Generic Exception Signature.
 * @throws InformationalException Generic Exception Signature.
 */
```

```
boolean isUserExternal(String username)
  throws AppException, InformationalException;
```

Customizing Sanitization Settings

IBM Cúram Social Program Management contains a sanitization library. The library sanitizes data and property values throughout the application to remove HTML markup that is potentially malicious.

About this task

The allowlist, which is installed by default, supports a set of HTML elements and attributes that are deemed safe and, therefore, do not require filtering out. To customize the allowlist, add HTML elements and attributes that are deemed safe, and remove HTML elements and attributes that are deemed potentially malicious.

Note: The Rich Text Editor uses its own unique allowlist. For more information about how to configure the sanitizing of text that is entered through the Rich Text Editor, see the *Enabling configuration of a security allowlist for the Rich Text Editor* related link.

The following example outlines the format that entries in the allowlist file must match:

```
tag=attribute1,attribute1
```

For example, an allowlist that contains the following entries is declaring that the a, div, and h1 HTML elements are safe:

```
a=href
div=
h1=
```

The allowlist also declares the href attribute is safe when it is used on an a HTML element. All other HTML elements and attributes are filtered out.

The allowlist of HTML elements and attributes is defined in the `default-secure-sanitize-allowlist.properties` application resource file. To customize the allowlist, choose one of the options in the following procedure.

Procedure

Choose one of the following options:

- Customize the allowlist and persist the changes permanently to the database:
  1. Copy the `default-secure-sanitize-allowlist.properties` file in `EJBServer/components/CEFWidgets/data/initial/blob` to an equivalent location in a custom EJBServer component.
  2. Modify the copied file, as required.
  3. Update the custom DMX file for the AppResource table and add a row that points to the newly modified `default-secure-sanitize-allowlist.properties` file.
  4. Build the server and the database.
- Customize the allowlist through the administration user interface:
  1. Log on as an administrative user.
  2. In the **Shortcuts** panel, click **Intelligent Evidence Gathering** > **Application Resources**.
  3. Search for and download the `default-secure-sanitize-allowlist.properties application` resource file.
  4. Modify the downloaded file, as required.
  5. Edit the `default-secure-sanitize-allowlist.properties application` resource file.
  6. Select the modified file as its Content.

7. To apply the changes, click **Publish**.

Related tasks

Enabling configuration of a security allowlist for the Rich Text Editor

Cross-Site Request Forgery (CSRF) and IBM Cúram Social Program Management

IBM Cúram Social Program Management web pages and RESTful web services use a combination of mechanisms to protect against Cross-Site Request Forgery (CSRF) attacks.

For more information about CSRF, see the Open Web Application Security Project's *Cross-Site Request Forgery Prevention Cheat Sheet* related link. For more information about CSRF in Social Program Management, see the *Cross-Site Request Forgery (CSRF) protection for IBM Cúram Social Program Management web pages* and *Cross-Site Request Forgery (CSRF) protection for RESTful web services* related links.

Related concepts

Cross-Site Request Forgery (CSRF) protection for RESTful web services

Related information

Cross-Site Request Forgery Prevention Cheat Sheet

Cross-Site Request Forgery (CSRF) protection for Cúram web pages

IBM Cúram Social Program Managementuser interface (UI) infrastructure uses a combination of mechanisms, including an HTTP referrer header check, to protect Social Program Management against Cross-Site Request Forgery (CSRF) attacks. The referrer header check validates all incoming requests. Only requests from trusted domains are permitted. If no referrer header is supplied, which can happen because the user types directly into the browser URL, for example, then the request is also rejected.

About this task

You configure the `curam.referer.domains` property in the `Application.prx` file for your custom component or by using the Social Program Management system administration application.

The mandatory `curam.referer.domains` property configures a list of allowed domains that you can set in the referrer header of a request. The property protects against CSRF attacks. By default, the property is set `localhost`. However, in a deployed environment the property must be set and normally this includes the host domain. Set the property as a comma-separated list of domains that are accepted in the referrer header. For example, the value `abc.com, def.com` permits all requests with subdomains of `abc.com` and `def.com` that are set in the referrer header to successfully connect to Social Program Management. The property is not required at development time.

The following steps outline how you can configure CSRF protection in the Social Program Management system administration application.

Procedure

1. Log in to Social Program Management as a system administrator.
2. Select **System Configurations** > **Shortcuts** > **Application Data**.
3. Type `curam.referer.domains` in the **Name** field and click **Search**.
4. Select **...** > **Edit Value**.
5. Set the string value to a comma-separated list of allowed domains and click **Save** to save your changes.
6. Click **Publish** for your changes to take effect.

What to do next

Complete the required postinstallation configuration tasks to ensure that the Social Program Management software is configured and is working correctly with the prerequisite software. For more information, see the *Social Program Management postinstallation configuration* related link.

Related concepts

Cross-Site Request Forgery (CSRF) protection for RESTful web services

Related tasks

Social Program Management postinstallation configuration

Notices

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Privacy Policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies or other similar technologies that collect each user’s name, user name, password, and/or other personally identifiable information for purposes of session management, authentication, enhanced user usability, single sign-on configuration and/or other usage tracking and/or functional purposes. These cookies or other similar technologies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Programming Interface Information

This publication documents intended programming interfaces that allow the customer to write programs to obtain the services of IBM Cúram Social Program Management.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be

trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at " Copyright and trademark information " at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.

Part Number:

(1P) P/N: