

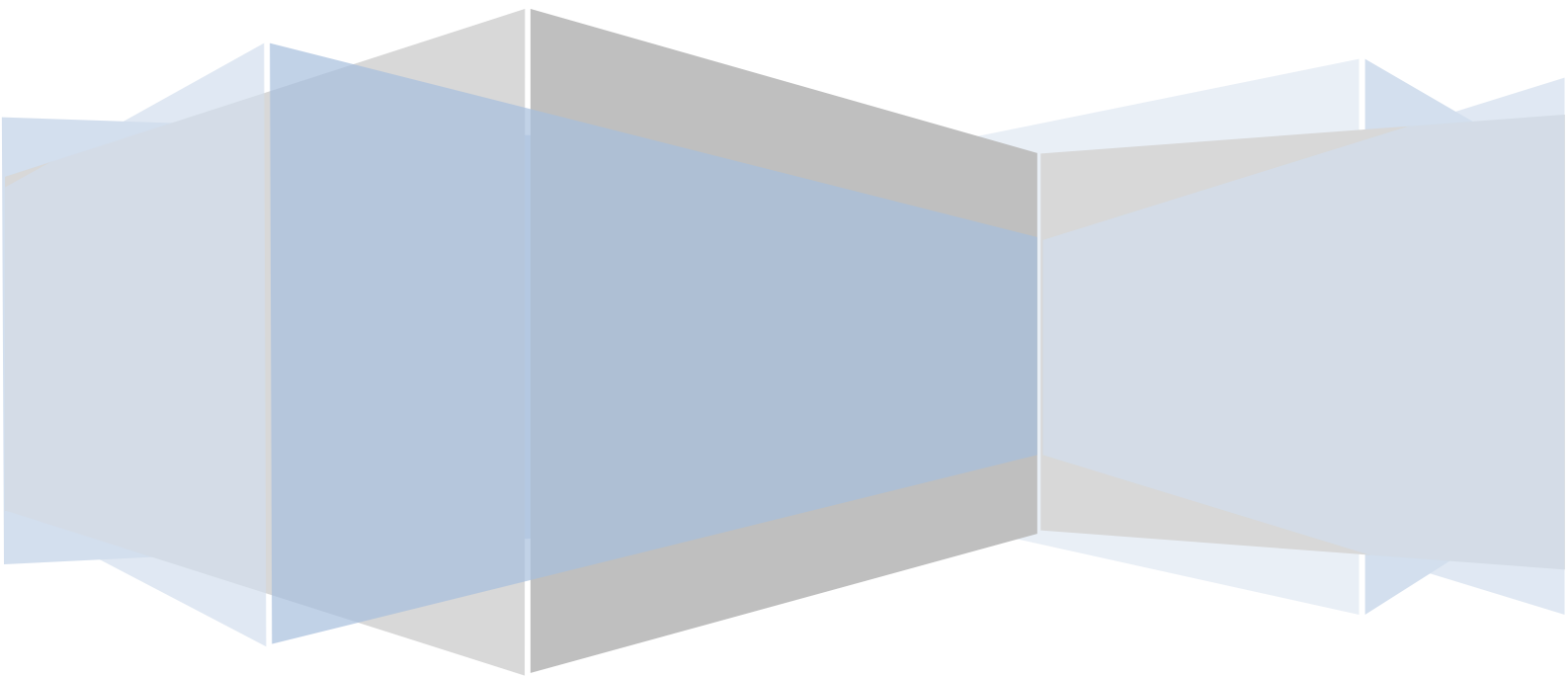
SSO Plugin

Identity Federation Service

J System Solutions

<http://www.javasystemsolutions.com>

Version 3.5



JSS SSO Plugin - Identity federation service

| | |
|--|----|
| Introduction..... | 3 |
| Enabling the identity federation service..... | 4 |
| Federation key..... | 4 |
| Token lifetime..... | 4 |
| Overview of the identity federation process..... | 5 |
| Retrieving the authenticated user..... | 5 |
| Installing the identity federation acceptor filter into a JEE application..... | 7 |
| Updating the web.xml file..... | 7 |
| Installing the jar files..... | 9 |
| Locating the jars and web.xml file in the installation files..... | 9 |
| Example web application..... | 10 |

Introduction

The SSO Plugin provides a range of comprehensive range of SSO capabilities that allow you to provide SSO for third party Java based applications.

The functionality has been specifically designed to be quick and easy to implement and involves installing a JEE filter into the third party web application.

Assuming the third party application uses the standard JEE technique for checking a user has validated, the integration should be quick and easy for any developer with a working knowledge of JEE development.

The third party application can use the service to access the raw SSO username, or the SSO provider username and groups, ie from the BMC AR System User form or HP Service Manager Operator table.

SSO Plugin sets the `java.security.Principal` object on the `javax.servlet.http.HttpServletRequest`, and also implements the `HttpServletRequest.isUserInRole` method.

This document refers to the `examplewebapp` shipped with SSO Plugin. This is purely an example of how the Identity Federation Service can be used in practice.

For integration advice, contact the JSS support team.

Enabling the identity federation service

The identity federation service isn't enabled by default. To enable it, go to the SSO Plugin setup page and click "Enable identity federation service". The following fields must also be set.

Federation key

This is known only to the SSO Plugin and the third party applications that are integrating with the SSO Plugin. It is used as a seed to a hashing function that allows the identity federation service and acceptor to trust each other.

Token lifetime

When a user has been authenticated, the JSS identity federation acceptor sets a cookie (with the name `jss-ssoplugin`). The cookie contains an encoded token that includes the logged in user. The cookie is set to expire after a period of time that is calculated to be the time at which the user was authenticated with SSO Plugin, plus the value defined in this field.

Typically, a value of one day will ensure every client has to re-authenticate with SSO Plugin once a day.

Overview of the identity federation process

The third party application runs the JSS identity federation acceptor filter. It uses the following two conditions to define a user as authenticated:

1. The request hits the identity federation acceptor filter. If the session has already been validated by the filter then it is allowed to continue.
2. If the session has not been validated and the `jss-ssoplugin` cookie exists and can be validated (using the federation key), the username contained within it is set as the authenticated user and the request is allowed to continue.

If there's no valid session or cookie, the following process is followed:

1. The request is redirected (via an HTTP 302 response) to the SSO Plugin identity federation service. This URL is defined in the filter's initialisation parameters in the `web.xml` file.
 - a) The URL requested by the client browser is passed as the `jss-return` parameter to the SSO Plugin.
 - b) The URL is also hashed (using the federation key) and set as the `jss-return-hash` parameter. This allows the identity federation service to confirm the request to authenticate was made from a correctly configured identity federation acceptor.
2. The SSO Plugin will authenticate the user using the configured authentication method, ie built-in AD, OpenID, Siteminder, etc.
3. If the user is successfully authenticated:
 - a) The `jss-return-hash` parameter is validated (using the federation key) to ensure the request was made by a valid identity federation acceptor (see 1(b)). If this test fails, the request is sent to an error page.
 - b) If the `jss-return-hash` parameter is valid, the request is redirected (via an HTTP 302 response) to the URL defined in the `jss-return` parameter, with an encoded token (using the federation key) set as the `jss-token` parameter.
4. The request hits the third party application and the identity federation acceptor filter. The `jss-token` parameter is validated (using the federation key) to ensure it was sent by the identity federation service. If so, the user is set on the session as the authenticated user and the request allowed to continue.

Retrieving the authenticated user

The third party application can retrieve the logged in user through the standard JEE technique of accessing the [java.security.Principal](#) object on the [javax.security.http.HttpServletRequest](#) as follows:

```
java.security.Principal p= request.getUserPrincipal();
```

```
if (p!=null) // Is the user authenticated?  
    System.out.println("Username: "+p.getName());
```

Please note, if the application is protected by the JSS identity federation acceptor filter, the request will not proceed to the application code unless it is authenticated and `request.getUserPrincipal()` will always return the Principal object.

Installing the identity federation acceptor filter into a JEE application

The filter is installed by updating the applications web.xml file and including jar file(s).

Updating the web.xml file

Open the web.xml file and include the following patch:

```
<filter>
  <filter-name>ssoplugin-identity-federation-acceptor</filter-name>
  <filter-class>
    com.javasystemsolutions.sso.identityfederation.IdentityFederationAcceptor
  </filter-class>
  <!--
    Enter the URL of the identity federation service here.
    This is typically:
    http://hostname:8080/arsys/jss-ss0/identityfederationservice
  -->
  <init-param>
    <param-name>identityFederationServiceURL</param-name>
    <param-value>
      http://localhost:8080/arsys/jss-ss0/identityfederationservice
    </param-value>
  </init-param>
  <!--
    Enter the federation key set up in the SS0 Plugin
    configuration interface. This is used to create hashes
    that allow the acceptor and service to trust each other.
  -->
  <init-param>
    <param-name>key</param-name>
    <param-value>mysecretkey</param-value>
  </init-param>
  <!-- Map incoming groups to different values -->
  <init-param>
    <param-name>groupToRoleMap</param-name>
    <param-value>Administrator=ITAdmin;Incident
    Master=INCMaster</param-value>
  </init-param>
</filter>

<!-- Map URLs to protect -->
<filter-mapping>
  <filter-name>ssoplugin-identity-federation-acceptor</filter-name>
```

```
<url-pattern>/secure.jsp</url-pattern>
</filter-mapping>
```

The following parameters are required by the filter:

1. **identityFederationServiceURL:** This points to the identity federation service running on the SSO Plugin installation. The identity federation service URL is /jss-ssso/identityfederationservice, relative to the SSO Plugin enabled web application installation, ie BMC Mid Tier , HP Service Manager or the standalone edition of SSO Plugin. Therefore, if SSO Plugin is installed at:

```
http://bmcmidtier:8080/arsys
```

then the identityFederationServiceURL is:

```
http://bmcmidtier:8080/arsys/jss-ssso/identityfederationservice
```

2. **key:** This must be set to the federated identity key set in the SSO Plugin interface.

The following parameters are optional:

1. **logLevel:** This controls the amount of logging generated by the identity federation acceptor, the valid values are INFO, DEBUG and TRACE. You are advised to set this to INFO in production.
2. **useProviderUsername:** If this is true, the Principal object will be populated with the SSO Provider username if a valid SSO account exists. Please note, it is entirely possible for a user to have corporate SSO access (and an SSO username) but no SSO enabled account in the provider (ie BMC AR System or HP Service Manager).
3. **stripdomain:** If you are validating against an Active Directory then your SSO token will contain the Windows domain (ie JAVASYSTEMSOLUTIONS\dkellett or dkellett@javasystemsolutions.com). Setting this value to true will remove the domain element, returning dkellett in both cases.
4. **convertcase:** The valid values are upper and lower, which will convert the SSO token to upper or lower case. This is useful for matching SSO usernames to third party user accounts.
5. **groupToRoleMap:** The group names provided by the SSO provider may need to be translated for the Java based web application to which you are integrating. Therefore, a semi-colon separated list of group mappings can be provided, ie A=B implies map incoming group A to the value B. The groups are set on the Principal, which is placed on the HttpServletRequest, and the request.isUserInRole method is also implemented.

Please note: Options (3) and (4) are only available if option (2) is false.

The filter is set to protect the web application via the <filter-mapping> directives. You can set as many mappings as required in your application, or even protect it all by setting /*.

Installing the jar files

The minimum requirement is to copy the `jss-ssso-thirdparty.jar` file to the third party application. If the application isn't already using commons logging and log4j, these jars must be copied too.

Locating the jars and web.xml file in the installation files

The `web.xml` patch and jar files can be found in the example web application. This is found in the `thirdpartyauthentication` directory.

Example web application

An example web application is provided with the product to demonstrate how the Identity Federation Service can be used in practice and for debugging your own third party integrations.

The example web application can be found in the `thirdpartyauthentication` directory. It is called `examplewebapp` and you can install it as follows:

1. Update the `web.xml` file and alter the `identityFederationServiceURL` and key parameters as discussed above.
2. Copy the web application into the Tomcat `webapps` directory.
3. Restart Tomcat.

You can now navigate your browser to the example web application (ie `http://bmcmidtier:8080/examplewebapp`) and test the integration by clicking the link to the secure page.