Red Hat JBoss Enterprise Application Platform 7.2.Beta

How To Set Up SSO with SAML v2

For Use with Red Hat JBoss Enterprise Application Platform 7.2.Beta

Last Updated: 2018-08-06

Red Hat JBoss Enterprise Application Platform 7.2.Beta How To Set Up SSO with SAML v2

For Use with Red Hat JBoss Enterprise Application Platform 7.2.Beta

Legal Notice

Abstract

The intent of this guide is to provide a deeper dive into what SSO with SAML v2 is, as well as how to set up and configure it within Red Hat JBoss Enterprise Application Platform (JBoss EAP). Before reading this guide, users should read through the JBoss EAP Security Architecture guide and have a solid understanding of the SSO and SAML v2 information presented in that guide. When completing this guide, you should have a solid, working understanding of SSO and SAML v2, how it relates to JBoss EAP, and how to configure it.

Table of Contents

CHAPTER 1. SSO WITH SAML V2 DEEPER DIVE

The basics of SSO and SAML are covered in the JBoss EAP *Security Architecture* guide. This section takes a deeper dive into the components involved in SAML v2 and SSO.

1.1. WHAT IS SAML V2?

Security Assertion Markup Language, or SAML, is a data format and protocol that allows two parties, usually an identity provider and a service provider, to exchange authentication and authorization information. This information is exchanged in the form of SAML tokens that contain assertions, and are issued by Identity Providers to subjects for authenticating with Service Providers. The ability for subjects to use and reuse SAML tokens issued from an identity provider with multiple service providers allow SAML v2 to facilitate browser-based SSO.

1.1.1. Building Blocks

The most important concept to keep in mind with SAML is that its all about passing security assertions between entities. SAML has several components it uses to accomplish this task.

1.1.1.1. Entities

Entities are all parties involved in creating and passing assertions. SAML has the concept of three distinct entities:

subject

The **subject**, also referred to as the **principal**, which is the user in most cases, is requesting access to a resource on a **service provider**, which is secured by SAML.

service provider

The **service provider**, or **SP**, requires proof, as an assertion, of the **subject** 's identity, which it needs from the **identity provider**.

identity provider

The **identity provider**, or **IDP**, provides a set of assertions, in the form of a token about a **subject**, that can be used in authentication and authorization decisions by **service providers**.

In summary, **subjects** get issued assertions, **identity providers** issue those assertions, and **service providers** use those assertions to authenticate and authorize **subjects**.

1.1.1.2. Security Assertions

A security assertion is a set of statements issued by an identity provider about a subject. Service providers use these assertions to make access-control decisions about a subject. Statements can take the following forms:

Authentication

Authentication assertions assert that a subject successfully authenticated using a specified method at a specific point in time. An authentication context containing other information about the authenticated subject can also be specified in an authentication statement.

Attribute

Attribute assertions assert that a subject has certain attributes.

Authorization Decision

`Authorization Decision` assertions assert a response, **`accept`** or **`deny`**, to an authorization request for a subject on a resource.

Example

```
This user logged in as Sarah at 9:30 using a username and password.  Sarah
is a member of the Managers group.  Sarah is accepted to access the
Employee Information resource.
```

- The statement **`This user logged in as Sarah at 9:30 using a username and password`** is an **`Authentication`** assertion.
- The statement **`Sarah is a member of the Managers group`** is an **`Attribute`** assertion.
- The statement **`Sarah is accepted to access the Employee Information resource`** is an **`Authorization Decision`** assertion.

Assertions are packaged as SAML tokens and transported using SAML protocols.

1.1.1.3. Protocols

A SAML protocol describes how assertions are packaged, usually in the form of a request and response, as well as the rules on the correct way to process them. These rules must be followed by both the producers and consumers of the requests and responses. A request can ask for specific, known assertions or query identity providers for authentication, attribute, or authorization decisions. The request and response messages, which include security assertions, are formatted in XML and adhere to a specified schema.

1.1.1.4. Bindings

SAML bindings specify how SAML protocols map to other standard protocols used for transport and messaging. Some examples include:

- A SAML binding that maps to an HTTP redirect.
- A SAML binding that maps to an HTTP **`POST`**.
- A SAML binding that maps SAML requests/responses to SOAP requests and responses.

1.1.1.5. Profiles

SAML profiles use assertions, protocols, and bindings to support specific use cases, such as Web Browser SSO, Single Logout, and Assertion Query.

1.2. HOW DOES SAML V2 WORK WITH SSO

The basics of browser-based SSO with SAML v2 are covered in the JBoss EAP *Security Architecture* guide, specifically in the Browser-Based SSO Using SAML and Multiple Red Hat JBoss Enterprise Application Platform Instances and Multiple Applications Using Browser-Based SSO with SAML sections. This section gives a more in-depth explanation regarding the SAML profiles and bindings related to browser-based SSO with SAML v2.

1.2.1. Web Browser SSO Profile

The **`Web Browser SSO`** profile specifies the way an IDP, SP, and principal, in the form of a browser agent, handle browser-based SSO. Both the SP and IDP have several bindings that each can be used in the Web Browser SSO profile, allowing many possible flows. Additionally, this profile supports message flows initiated from either the IDP or SP. This profile also supports the IDP pushing the SAML assertion to the SP, or the SP pulling the assertion from the IDP. Flows initiated from either the SP or IDP are explained at a high level in the JBoss EAP *Security Architecture* guide. SAML assertions pushed from the IDP utilize HTTP **`POST`** messages or HTTP redirects. SAML assertions that are pulled by SPs involve sending an artifact to the receiver, which is then dereferenced to obtain the assertions.

The basic flow of the Web Browser SSO profile is as follows:

1. Principal HTTP request to SP.
   The principal first attempts to access a secured resource at the SP using an HTTP User Agent, for example a browser. If the principal has already been issued a SAML token with a valid security context, the SP will allow or decline the principal. This is the last step. Otherwise, the SP will attempt to locate the IDP for the authentication request.

2. SP determines IDP.
   The SP locates the IDP and its endpoint that supports the SP's preferred binding. This allows the SP to send an authentication request to the IDP. The specific means of this process can vary between implementations.

3. Authentication Request issued from SP to IDP using the principal.
   Once the SP determines the IDP location and endpoint, the SP issues an Authentication Request in the form of an **`<AuthnRequest>`** message, which will be delivered by the user agent, principal to the IDP. The HTTP Redirect, HTTP **`POST`**, or HTTP Artifact SAML bindings can be used to transfer the message to the IDP using the user agent.

4. IDP identifies principal.
   Once the Authentication Request is delivered to the IDP by the principal, the principal will be identified by the IDP. The identification method is not specifically defined by the Web Browser SSO profile and may be accomplished in a number of ways, for example authentication using **`FORM`**, using existing session information, kerberos authentication, etc.

5. IDP issues Response to SP.
   Once the principal is identified, the IDP issues a Response in the form of a **`<Response>`** message, to be delivered back to the SP for granting or declining access by the principal using the user agent. This message will contain at least one authentication assertion and can also be used to indicate errors. HTTP **`POST`** or HTTP Artifacts can be used to transfer this message, but HTTP Redirect cannot be used due to URL length constraints with most user agents. If the user agent initiated an IDP-based flow, for example by attempting to access the IDP directly instead of an SP, the process would begin at this step. If successful, the HTTP **`POST`** or HTTP Artifact will be sent to a location, which is pre-configured in the IDP.

6. SP allows or declines access to principal.
   Once the SP receives the Response, it may grant access for the requested resource to the principal by creating a security context, or it may deny access, or do its own error handling.

NOTE

JBoss EAP does not support the SAML artifact binding.

HTTP REDIRECT VS. POST BINDINGS

HTTP Redirect bindings make use of HTTP **GET** requests and the URL query parameters to transmit protocol messages. Messages sent in this manner are also URL and Base-64 encoded before being sent and decoded by the receiver. HTTP **POST** bindings send messages using form data, and also do a base-64 encode/decode on the message. Both SPs and IDPs can transmit and receive messages using redirect or **POST** bindings. Due to the limitation of URL lengths in certain scenarios, HTTP Redirect is usually used when passing short messages, and HTTP **POST** is used when passing longer messages.

1.2.2. Global Logout Profile

The Global Logout Profile allows a principal, who has authenticated with a set of IDPs and SPs, to log out and have that assertion be propagated to one or more associated IDPs and SPs.

When a principal authenticates with an IDP, the principal and IDP have established an authentication session. The IDP may then issue assertions to various SPs, or relying parties, based on that authentication. From there if the principal attempts to access a secured resource within those SPs, the SPs may choose to establish additional sessions with the principal based on that assertion issued from the IDP, hence relying on the IDP.

Once a session or set of sessions is created, a principal might be logged out of sessions individually using various means, or they can use the Global Logout Profile to logout of all sessions and from all SPs and IDPs at once. The Global Logout Profile can use the HTTP Redirect, HTTP **POST** or HTTP Artifact bindings in its flow. It can also use SOAP binding in certain cases which are not in the scope of this document.

NOTE

Single Logout Profile can be used as a synonym to Global Logout Profile.

NOTE

JBoss EAP does not support the SAML artifact binding.

As with the Web Browser SSO profile flow, the Global Logout Profile flow may be initiated either at the IDP or the SP.

The basic flow of the Global Logout Profile is as follows:

1. Logout issued to IDP by Session Participant.
   A session participant, such as Service Providers or other relying parties, terminates its own session with the principal and sends a Logout Request, in the form of a **<LogoutRequest>** message, to the IDP that initially issued the security assertion for the principal. This request can be sent directly between the IDP and relying party, or indirectly by using the principal's user agent as a pass through.

2. IDP identifies Session Participant.
   Once the IDP receives the Logout Request, it uses that request to determine what

sessions to terminate with which relying parties, including any sessions the IDP owns as a session authority or session participant. For each session, the IDP issues a Logout Request to the relying party and waits for a Logout Response from each party before issuing a new Logout Response back to the original session participant. In cases where the Global Logout Profile flow was initiated at the IDP, the flow begins at this step, and some other mechanism is used to determine the sessions and SPs.

3. Logout issued by IDP.
   Once the IDP determines all of the sessions and associated relying parties, it sends a Logout Request, in the form of a **<LogoutRequest>** message, to each relying party and awaits a Logout Response. These requests may be sent directly between the IDP and the relying parties, or indirectly through the principal's user agent.

4. Logout response issued by Session Participant or Authority.
   Each relying party, including the IDP itself in some cases, attempts to terminate the session as directed by the IDP in the Logout Request, and returns a Logout Response in the form of a **<LogoutResponse>** message, back to the IDP. As with the Logout Request, the response can be issued directly between the relying party and the IDP, or indirectly through the principal's user agent.

5. IDP issues Logout response to original Session Participant.
   Once all the Logout Responses has been received from the relying parties, the IDP sends a new Logout Response, in the form of a **<LogoutResponse>** message, back to original session participant who requested the logout. As with the other parts of this flow, this response can be passed directly between the IDP and the session participant, or indirectly through the principal's user agent. In cases where the Logout Request was initiated at the IDP, this step is omitted.

NOTE

The direct communication between the IDP and SP portion of the Global Logout Profile is not supported in JBoss EAP.

1.2.3. Multiple IDPs and the Identity Discovery Profile

Browser-based SSO using SAML v2 also supports having multiple IDPs, and can be used in both the Web Browser SSO profile as well as the Global Logout profile. In cases where multiple IDPs are configured, the Identity Discovery SAML profile is used to determine which IDP a principal uses. This is accomplished by reading and writing cookies with domain information and a list of IDPs.

1.3. FURTHER READING

For full details on the SAML v2, see the official SAML 2.0 specification.

CHAPTER 2. HOW TO SET UP SSO WITH SAML V2

IMPORTANT

Setting up SSO with SAML v2 using Picketlink is deprecated. Going forward, you should use Red Hat Single Sign-On, which supports OAuth and OpenID Connect as well as SAML. You can find more information on how to configure SSO with Red Hat Single Sign-On in the *Securing Applications and Services Guide*.

This section details the actual steps for setting up SSO with SAML v2 using JBoss EAP.

2.1. COMPONENTS

As covered in the Entities section as well as in the JBoss EAP *Security Architecture* guide, there are three **entities**, or parties, involved in browser-based SSO using SAML v2:

- A principal using a user agent (browser) to request access to a secured resource.
- A service provider housing the secured resource.
- An identity provider which issues security assertions to principals, allowing them to access secured resources on service providers.

In addition, the following is needed to support browser-based SSO using SAML v2:

- Separate web applications serving as SPs and IDPs
- JBoss EAP instances to host the SPs and IDPs
- Security Domains to support the SPs and IDPs

2.2. IDP AND SP SETUP AND CONFIGURATION

This section covers setting up an application to be either an SP or IDP, as well as setting up a JBoss EAP instance to host those applications.

2.2.1. Setting up an IDP

To set up an application to serve as an IDP, the following steps must be performed:

NOTE

The security domain should be created and configured before creating and deploying the application.

1. Create a security domain for an IDP.
   The IDP handles challenging a principal for their credentials, handling the authentication and authorization of that principal, and issuing the proper SAML v2 security assertions based on the result. This requires that an identity store be configured using a security domain. The only requirement around creating this security domain and identity store is that it has authentication and authorization mechanisms properly defined. This means that many different identity stores — for

example, properties file, database, or LDAP — and their associated login modules, could be used to support an IDP application. For more information on security domains, see the Security Domains section of the JBoss EAP *Security Architecture* guide.

The following example uses a simple **UsersRoles** login module using properties files.

Management CLI Commands for Creating a Security Domain

```
/subsystem=security/security-domain=idp:add(cache-type=default)

/subsystem=security/security-domain=idp/authentication=classic:add

/subsystem=security/security-
domain=idp/authentication=classic/login-
module=UsersRoles:add(code=UsersRoles,flag=required,module-options=
[usersProperties=${jboss.server.config.dir}/idp-
users.properties,rolesProperties=${jboss.server.config.dir}/idp-
roles.properties])

reload
```

The **UsersRoles** login module utilizes properties files to store the user/password and user/role information. For more information on the **UsersRoles** module, see the JBoss EAP *Login Module Reference*. In this example, the properties files contain the following:

idp-users.properties

```
Eric=samplePass
Alan=samplePass
```

idp-roles.properties

```
Eric=All
Alan=
```

2. Configure the **web.xml** file for an IDP.
   The **web.xml** file for an IDP must contain the following:

   - A **<security-constraint>** with a **<web-resource-collection>** containing a **<url-pattern>** that maps to the URL pattern of the secured area. Optionally, **<security-constraint>** can also contain an **<auth-constraint>** stipulating the allowed roles.
   - A **<login-config>** configured for **FORM** authentication.
   - If any roles were specified in **<auth-constraint>**, those roles should be defined in a **<security-role>**.
   - Optionally, resources used by the login form, for example images and styles, can be specified by an additional security constraint to be unsecured so they can be accessed prior to authentication, for example on the login page.

The **<security-constraint>** and **<security-role>** elements enable administrators to setup restricted or unrestricted areas based on URL patterns and roles. This allows resources to be secured or unsecured.

The **<login-config>** tag defines the login and error pages used by the IDP when authenticating users.

Example `web.xml` file:

```
<web-app>
  <display-name>IDP</display-name>
  <description>IDP</description>
  <!-- Define a security constraint that gives unlimited access to
images -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Images</web-resource-name>
      <url-pattern>/images/*</url-pattern>
    </web-resource-collection>
  </security-constraint>
  <!-- Define a security constraint that requires the All role to
access resources -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>IDP</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>All</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Define the Login Configuration for this Application -->
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>IDP Application</realm-name>
    <form-login-config>
      <form-login-page>/jsp/login.jsp</form-login-page>
      <form-error-page>/jsp/error.jsp</form-error-page>
    </form-login-config>
  </login-config>
  <!-- Security roles referenced by this web application -->
  <security-role>
    <description>The role that is required to log in to the IDP
Application</description>
    <role-name>All</role-name>
  </security-role>
</web-app>
```

NOTE

It is recommended that a welcome page is defined in the application. By default, JBoss EAP will look for a file named **index.jsp**, but this can be configured using **<welcome-file-list>** in **web.xml**.

Example `login.jsp` file:

```
<html>
  <head></head>
  <body>
    <form id="login_form" name="login_form" method="post"
action="j_security_check" enctype="application/x-www-form-
urlencoded">
      <center> <p>Welcome to the <b>IDP</b></p> <p>Please login to
proceed.</p> </center>
      <div style="margin-left: 15px;">
        <p> <label for="username">Username</label> <br /> <input
id="username" type="text" name="j_username"/> </p>
        <p> <label for="password">Password</label> <br /> <input
id="password" type="password" name="j_password" value=""/> </p>
        <center> <input id="submit" type="submit" name="submit"
value="Login"/> </center>
      </div>
    </form>
  </body>
</html>
```

Example `error.jsp` file:

```
<html>
  <head></head>
  <body>
    <p>Login failed, please go back and try again.</p>
  </body>
</html>
```

3. Configure the authenticator for an IDP.
   The authenticator is responsible for the authentication of users and for issuing and validating SAML v2 security assertions. Part of the authenticator is configured in the **`jboss-web.xml`** file by defining the security domain to be used in authenticating and authorizing principals (see Step 1). You also must ensure that a **`<login-config>`** is specified in the **`web.xml`** and the necessary dependencies have been declared.

   The **`jboss-web.xml`** file must have the following:

   - A **`<security-domain>`** to specify which security domain to use for authentication and authorization.

   Example `jboss-web.xml` File

   ```
   <jboss-web>
     <security-domain>idp</security-domain>
     <context-root>identity</context-root>
   </jboss-web>
   ```

4. Declare the necessary dependencies for an IDP.
   The web application serving as the IDP requires a dependency to be defined in **`jboss-deployment-structure.xml`**, so that the **`org.picketlink`** classes can be located. JBoss EAP provides all necessary **`org.picketlink`** and related classes, and the application only needs to declare them as dependencies to use them.

Using jboss-deployment-structure.xml to Declare Dependencies

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" services="import"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

NOTE

In previous versions of JBoss EAP, you would have declared this same dependency, but would have declared a valve to install the SAML authenticators. With the introduction of Undertow in JBoss EAP 7, you now use **services="import"** to install the SAML authenticators.

5. Create and configure a **picketlink.xml** file for an IDP.
The **picketlink.xml** file is responsible for the behavior of the Authenticator and is loaded at the application's startup.

   The file must contain at least the following elements:

   - **<PicketLinkIDP>** defining the URL of the IDP, using**<IdentityURL>**, and any hosts trusted by the IDP.
   - **<Handlers>** defining the set of handlers needed for processing the SAML requests and responses.

Example picketlink.xml File

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1">

<IdentityURL>${idp.url::http://localhost:8080/identity/}</IdentityUR
L>
    <Trust>
      <Domains>localhost,example.com</Domains>
    </Trust>
  </PicketLinkIDP>
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Is
suerTrustHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Lo
gOutHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Au
thenticationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGe
```

```
nerationHandler" />
  </Handlers>
</PicketLink>
```

WARNING

Handlers are implemented using a Chain of Responsibility, with each individual handler performing logic on request and responses in the order defined in **picketlink.xml**. It is very important to pay attention to the order in which the handlers are configured.

By default, **picketlink.xml** is located in the **WEB-INF** directory of the IDP web application. However, a custom path to **picketlink.xml** that is external to the application can be configured. This is useful in cases where multiple applications across one or more JBoss EAP instances share the same **picketlink.xml** configuration.

6. *Optional:* Setting a custom location for **picketlink.xml**
   You can specify a custom location for the **picketlink.xml** using the **CONFIG_FILE** parameter. This is done by adding a **<context-param>** element to the **web.xml**.

   Using the `CONFIG_FILE` Parameter

   ```
   <context-param>
     <param-name>CONFIG_FILE</param-name>
     <param-value>/path/to/picketlink.xml</param-value>
   </context-param>
   ```

   You can also use the **org.picketlink.federation.saml.CONFIG_PROVIDER** parameter to specify a custom configuration provider. This allows you to create a custom implementation that extends **org.picketlink.identity.federation.web.util.SAMLConfigurationProvider** to provide your own configuration logic.

   Using the `org.picketlink.federation.saml.CONFIG_PROVIDER` Parameter

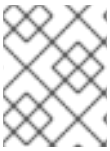
   ```
   <context-param>
     <param-name>org.picketlink.federation.saml.CONFIG_PROVIDER</param-
   name>
     <param-value>MyConfigurationProvider</param-value>
   </context-param>
   ```

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

2.2.2. Setting up an SP

To set up an application to serve as an SP, the following steps must be performed:



NOTE

The security domain must be created and configured before creating and deploying the application.

1. Configure a security domain for an SP.
   Since the IDP handles challenging the user for their credentials and issuing SAML v2 security assertions, the SP is in charge of validating those assertions. A security domain is still needed to perform this validation, but an identity store is not. In this case, the security domain for the SP must use **`SAML2LoginModule`**.

   Management CLI Commands for Adding Security Domain

   ```
   /subsystem=security/security-domain=sp:add(cache-type=default)

   /subsystem=security/security-domain=sp/authentication=classic:add

   /subsystem=security/security-domain=sp/authentication=classic/login-
   module=org.picketlink.identity.federation.bindings.jboss.auth.SAML2L
   oginModule:add(code=org.picketlink.identity.federation.bindings.jbos
   s.auth.SAML2LoginModule,flag=required)

   reload
   ```

WARNING

The **SAML2LoginModule** is intended for use only with applications using PicketLink with SAML, and should not be used without the PicketLink Service Provider Undertow ServletExtension (**org.picketlink.identity.federation.bindings.wildfly.sp.SPServletExtension**). Doing so presents a possible security risk, as **SAML2LoginModule** or **SAML2CommonLoginModule** will always accept the default password of **EMPTY_STR**. For example, this can also occur if the PicketLink Service Provider Undertow ServletExtension is not installed in the SP application. The PicketLink Service Provider Undertow ServletExtension is installed automatically when configuring the SP application for JBoss EAP. This can also occur if **SAML2LoginModule** is stacked with other login modules:

```
<security-domain name="sp" cache-type="default">
  <authentication>
    <login-module
code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule" flag="optional">
      <module-option name="password-stacking"
value="useFirstPass"/>
    </login-module>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties"
value="users.properties"/>
      <module-option name="rolesProperties"
value="roles.properties"/>
      <module-option name="password-stacking"
value="useFirstPass"/>
    </login-module>
  </authentication>
</security-domain>
```

SAML2LoginModule is used to build a security context for a user based on assertions. The PicketLink SAML Authenticator, which is installed by the PicketLink Service Provider Undertow ServletExtension (**org.picketlink.identity.federation.bindings.wildfly.sp.SPServletExtension**), uses **SAML2LoginModule** and allows for authentication decisions to be deferred to an IDP, which is configured in the SP's **picketlink.xml**.

The authenticator is responsible for the authentication of principals based on the security assertions, in this case SAML assertions, issued by the IDP. They intercept each request made to the application, check if a SAML assertion is present in the request, validate the assertions, execute principal's SAML specific validations, and create a security context for the principal in the requested application.

Part of the authenticator is configured in the **jboss-web.xml** file by defining the security domain to be used in authenticating and authorizing principals. You also must ensure that a **<login-config>** is specified in the **web.xml** and the necessary

dependencies have been declared, which are done in later steps.

The **jboss-web.xml** file for an SP must have the following:

- A **<security-domain>** to specify which security domain to use for authentication and authorization.

Example jboss-web.xml File

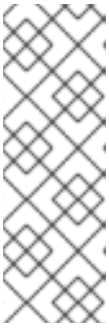
```
<jboss-web>
  <security-domain>sp</security-domain>
  <context-root>sales-post</context-root>
</jboss-web>
```

2. Configure the **web.xml** file for an SP.
The **web.xml** file for an SP must contain the following:

- A **<security-constraint>** with a **<web-resource-collection>** containing a **<url-pattern>** that maps to the URL pattern of the secured area. Optionally, **<security-constraint>** can also contain an **<auth-constraint>** stipulating the allowed roles.
- If any roles were specified in the **<auth-constraint>**, those roles should be defined in a **<security-role>**.
- A **<login-config>** with an **<auth-method>** specifying **FORM** authentication. This is required for the SAML authenticators.

Example web.xml File

```
<web-app>
  <display-name>SP</display-name>
  <description>SP</description>
  <!-- Define a Security Constraint on this Application -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>SP</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>All</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Security roles referenced by this web application -->
  <security-role>
    <description> The role that is required to log in to the SP
Application </description>
    <role-name>All</role-name>
  </security-role>
  <!-- Define the Login Configuration for this Application -->
  <login-config>
    <auth-method>FORM</auth-method>
  </login-config>
</web-app>
```



NOTE

It is recommended that you define a welcome page in the application. By default, JBoss EAP looks for a file named **index.jsp** but this can be configured using the **<welcome-file-list>** in **web.xml**.

The logout process attempts to redirect principals to **logout.jsp** on successful logout. Ensure this file is defined at the root directory of the application.

3. Declare the necessary dependencies for an SP.
   The web application serving as the SP requires a dependency to be defined in **jboss-deployment-structure.xml**, so that the**org.picketlink** classes can be located. JBoss EAP provides all necessary **org.picketlink** and related classes, and the application only needs to declare them as dependencies to use them.

   Using jboss-deployment-structure.xml to Declare Dependencies

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" services="import"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

NOTE

In previous versions of JBoss EAP, you would have declared this same dependency, but would have declared a valve to install the SAML authenticators. With the introduction of Undertow in JBoss EAP 7, you do not need to manually configure your deployments with the PicketLink SAML Authenticators. Now, they are configured automatically when you define use **services="import"** with the **org.picketlink** dependency. You must declare this configuration to enable SAML to your deployment.

4. Create and Configure a picketlink.xml File for an SP.
   The **picketlink.xml** file is responsible for the behavior of the Authenticator, and is loaded at the application's startup.

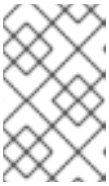
   The file must contain at least the following elements:

   - **<PicketLinkSP>** defining the URL of the IDP (**<IdentityURL>**) and the URL the SP (**<ServiceURL>**).
   - **<Handlers>** defining the set of handlers needed for processing the SAML requests and responses.

   Example picketlink.xml File

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkSP xmlns="urn:picketlink:identity-
federation:config:2.1" BindingType="POST">
```

```
<IdentityURL>${idp.url::http://localhost:8080/identity/}</IdentityURL>
    <ServiceURL>${sales-post.url::http://localhost:8080/sales-post/}</ServiceURL>
  </PicketLinkSP>
  <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
    <Handler class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler" />
    <Handler class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler" />
    <Handler class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler" />
  </Handlers>
</PicketLink>
```



NOTE

While not recommended, the SP may also be configured to use HTTP/REDIRECT by changing **BindingType="POST"** to **BindingType="REDIRECT"**.

WARNING

Handlers are implemented using a Chain of Responsibility, with each individual handler performing logic on request and responses in the order defined in **picketlink.xml**. It is very important to pay attention to the order in which the handlers are configured.

By default, **picketlink.xml** is located in the **WEB-INF** directory of the SP web application; however, a custom path to **picketlink.xml** that is external to the application can be configured. This is useful in cases where multiple applications across one or more JBoss EAP instances share the same **picketlink.xml** configuration.

5. *Optional:* Setting a custom location for **picketlink.xml**.
   You can specify a custom location for **picketlink.xml** using the **CONFIG_FILE** parameter. This is done by adding a **<context-param>** element to **web.xml**.

   Using the `CONFIG_FILE` Parameter

```
<context-param>
  <param-name>CONFIG_FILE</param-name>
  <param-value>/path/to/picketlink.xml</param-value>
</context-param>
```

You can also use the **org.picketlink.federation.saml.CONFIG_PROVIDER** parameter to specify a custom configuration provider. This allows you to create a custom implementation that extends **org.picketlink.identity.federation.web.util.SAMLConfigurationProvider** to provide your own configuration logic.

Using the org.picketlink.federation.saml.CONFIG_PROVIDER Parameter

```
<context-param>
  <param-name>org.picketlink.federation.saml.CONFIG_PROVIDER</param-
name>
  <param-value>MyConfigurationProvider</param-value>
</context-param>
```

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

2.2.3. Using SP-initiated Flow

The SP-inited Flow is a common use case citied when describing browser-based SSO, and is covered in the Browser-Based SSO Using SAML and Multiple Red Hat JBoss Enterprise Application Platform Instances and Multiple Applications Using Browser-Based SSO with SAML sections of the JBoss EAP*Security Architecture* guide. In summary, a principal attempts to access a secured resource in a SP. The SP starts the flow by checking for a principal's security assertions and redirecting any unauthenticated principal to the IDP. Upon successful authentication with the IDP, the principal is then redirected back to the initial SP with their security assertions for the SP to validate and permit/deny access to the original resource requested.

Walkthrough

1. A principal attempts to access secured resource on a SP.
2. SP performs a check on the principal. If the principal has not yet authenticated, they need to be redirected to the IDP. If they have already authenticated, then all other steps are skipped and the final step is performed.
3. The SP locates the IDP and issues an authentication request to the IDP using the principal's browser.
4. The IDP attempts to authenticate the principal, for example challenging them with a login page, using the configured identity store.
5. After the principal is identified by the IDP, for example after a successful login, the IDP issues a response, containing SAML v2 assertions with the principal's security-related information, to the SP using the principal's browser.
6. The SP performs a check on the principal's security assertions, and based on the information contained in those assertions, the SP allows or denies access to the requested resource.

This flow requires no additional steps or configuration for setup. All redirects are handled by the configured SPs and IDPs, and links to both secured and unsecured resources require no additional changes.

2.2.4. Using IDP-Initiated Flow

Most examples of browser-based SSO with SAML v2 use a SP-initiated flow, as covered in the previous section. However, SAML v2 supports an additional flow: the IDP-initiated or Unsolicited Response flow. In this scenario, the SP does not initiate the authentication flow and receive a SAML response from the IDP. Instead, the flow starts on the IDP-side, and once authenticated, the principal can choose a specific SP from a list and then get redirected to its URL.

2.2.4.1. Walkthrough

1. Principal accesses the IDP.
2. The IDP, seeing that there is neither SAML request nor response, assumes an IDP-first scenario using SAML.
3. The IDP challenges the principal to authenticate.
4. Upon authentication, the IDP shows the hosted section, where the principal is shown a page that links to all the SP applications.
5. The principal chooses an SP application.
6. The IDP redirects the principal to the SP with a SAML assertion in the query parameter. In the cases where the **POST** binding is used, the IDP sends the SAML assertion to the service provider using an HTTP **POST**.
7. The SP checks the SAML assertion and provides access.

2.2.4.2. Hosted Section

The **hosted section** is a location to direct users after a successful authentication in the IDP-initiated flow, or if a principal, who has already authenticated, attempts to access the root of the IDP directly. By default, the hosted section is located at **/hosted/**, but may be changed in the **picketlink.xml** file by adding the **HostedURI** attribute to the **<PicketLinkIDP>** element:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkIDP xmlns="urn:picketlink:identity-federation:config:2.1"
HostedURI="/hosted/">
    ...
  </PicketLinkIDP>
</Picketlink>
```

2.2.4.3. Linking to SPs

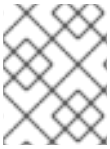
When the user is authenticated, the IDP shows a page with links to all service provider applications. A link will usually look like this:

```
<a href="http://localhost:8080/identity?
SAML_VERSION=2.0&TARGET=http://localhost:8080/sales-post/">Sales</a>
```

Note that the link above redirects the user to the IDP passing the **TARGET** query parameter, whose value is the URL to the target SP application. When the user clicks the link above, the IDP extracts the **TARGET** parameter from the request, builds a SAML v2 response, and redirects the user to the target URL. When the user hits the SP, they are automatically authenticated. The **SAML_VERSION** query parameter is used to specify the SAML version that must be used by the IDP to create the SAML response.

2.2.5. Configuring the Global Logout Profile

A Global Logout Profile initiated at one service provider, logs out the user from the Identity Provider (IDP) and all the service providers.



NOTE

For a Global Logout Profile to function appropriately, ensure that only up to five SPs are configured per IDP.

1. Configure **picketlink.xml**.
   Add the **SAML2LogOutHandler** in the **picketlink.xml**.

2. Create a **logout.jsp** page.
   As part of the logout process, the user is redirected to a **logout.jsp** page located in the root directory of the service provider application. Ensure that this page is created.

   Example logout.jsp

   ```
   <html>
     <head></head>
     <body>
       <p>You have successfully logged out.</p>
     </body>
   </html>
   ```

3. Configure Global Logout Profile links for the SPs.
   Use **GLO=true** as a URL parameter in a link to an SP resource to initiate the Global Logout Profile process.

   Example Logout Link

   ```
   <a href="?GLO=true">Click to log out</a>
   ```

2.2.6. Using Local Logout

In addition to Global Logout Profile, Local Logout can also be used. Local Logout, in contrast to Global Logout Profile, logs a principal out of a single SP while leaving the session at the IDP and other SPs intact. Essentially, Local Logout allows principals to be "locally logged out" at a single SP.

The process for using Local Logout is essentially the same as Global Logout Profile, with the URL parameter in the logout link taking the form of **LLO=true**.

Example Logout Link

```
<a href="?LLO=true">Click to log out</a>
```

When a principal clicks on the Local Logout link at the SP, the SP will invalidate their session and forward the principal to the configured logout page.

WARNING

When a principal is only disconnected from one service provider, they still have an active session with the IDP and other SPs that might allow them to still access secured resources. While this behavior may be desired in certain circumstances, it is strongly recommended to use Global Logout in most scenarios.

2.3. CONFIGURING IDPS AND SPS VIA THE FEDERATION SUBSYSTEM

In addition to configuring IDPs and SPs manually, SSO using SAML v2 may also be configured using a JBoss EAP subsystem. This method of configuration is known as the **`Domain Model`**, and allows all the configuration to reside centrally on the JBoss EAP instance and not with the individual applications. This also enables the SSO configuration to be created and updated using the JBoss EAP management interfaces, such as the management console and management CLI.

Federations

When using the JBoss EAP subsystem to configure and deploy IDPs and SPs, they are grouped together in a **`Federation`**. A Federation can be understood as a**`Circle of Trust`**. A Circle of trust contains applications that share common configurations, including certificates and SAML-specific configurations. It also contains domains that trust each other to accurately document the processes used to identify a user, the type of authentication system used, and any policies associated with the resulting authentication credentials. Each federation has one IDP and many SPs. The federation also defines trust relationship between SPs and IDPs, removing the need for each SP to individual track and maintain that information.

2.3.1. Configuring the Subsystem

Before the subsystem can be used to setup federations, it needs to be enabled and configured in JBoss EAP. The following steps are needed to enable and configure the subsystem:

NOTE

It is recommended that you shut down the JBoss EAP instance before performing these steps.

1. Update the extensions.

In the JBoss EAP configuration file, **standalone.xml** for standalone instances or **domain.xml** for domains, add the **org.wildfly.extension.picketlink** extension:

```
<extensions>
  ...
  <extension module="org.wildfly.extension.picketlink"/>
  ...
</extensions>
```

2. Add the subsystems.
In the JBoss EAP configuration file, **standalone.xml** for standalone instances or **domain.xml** for domains, add the **jboss:domain:picketlink-federation:2.0** subsystem:

```
<profile>
  ...
  <subsystem xmlns="urn:jboss:domain:picketlink-federation:2.0"/>
   ...
</profile>
```

NOTE

Examples of configuration can also be found in ***EAP_HOME*/docs/examples/configs/standalone-picketlink.xml**.

2.3.2. Setting up a Federation

Once the subsystem has been setup and configured, you can use the management interfaces to configure federations. Before you can set up a federation using the subsystem, you must prepare both the IDP and SP applications.

2.3.2.1. Preparing the SP and IDP Applications

As covered in the previous section, when configuring SSO using SAML v2 manually, the following files are required to be created or updated:

- **web.xml**
- **jboss-web.xml**
- **picketlink.xml**
- **jboss-deployment-structure.xml**

When using the subsystem to set up federations for SSO using SAML v2, the vast majority of the configuration happens from the management interfaces without having to update any of those files. The only configuration that must be done to the application is to configure the **<security-constraint>** and associated **<security-role>** in the **web.xml** file of the IDPs and SPs. In addition, **<login-config>** in **web.xml**, as well as the login and error pages must also be present in the IDP.

If an IDP or SP has already been configured as outlined in the previous section, the following files will need to be removed:

- **jboss-web.xml**

- **`picketlink.xml`**
- **`jboss-deployment-structure.xml`**

When the applications have been prepared, they must be deployed to the JBoss EAP instance.

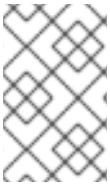
2.3.2.2. Creating a Federation Using the Management Console

When you have configured and deployed the applications, you can create a federation using the JBoss EAP management console. To create a federation using the management console:

1. Navigate to the management console using a web browser.
2. Click on the **Configuration** tab.
3. Click on **Subsystems** → **PicketLink** in the menu on the left side.
4. Click on the **Add** button in the **Federation** column to create a new federation.
    a. Enter a name, identity provider, security domain, and URL for the federation, and click **Save**. Ensure that **Identity Provider** matches the deployment name of your application, and that you provide the full URL for the **Url** field.
    For example, if you want **`IDP.war`** running at **`http://localhost:8080/identity/`** to be your identity provider, you must enter **`IDP.war`** for **Idenity Provider** and **`http://localhost:8080/identity/`** for **Url**.
5. Click on the federation you just created and click on the **Add** button in the **Service Provider** column.
    a. Enter a name, security domain, and URL for the service provider, and click **Save**. Ensure that the **Name** matches the deployment name of the service provider, and that you provide the full URL for the **Url** field.
    For example, if you want **`SP.war`** running at **`http://localhost:8080/sales-post/`** to be your service provider, you must enter **`SP.war`** for **Name** and **`http://localhost:8080/sales-post/`** for **Url**.
6. Click on the **View** button in the federation.
7. Click on **Trusted Domains** on the left side.
8. Click on the **Add** button above the table.
    a. Enter in the name of all the domains running service providers and identity providers. For example, if you have a service provider running at **`http://localhost:8080/sales-post/`** and an identity provider running at **`http://localhost:8080/identity/`**, you would add **`localhost:8080`** as a trusted domain.

NOTE

You can view the details of either the federation, including the IDP, or the SP, by clicking on the **View** button next to the appropriate row in the **Federation** or **Service Provider** column. The details section can be used to configure additional details or make updates to an existing IDP or SP.



NOTE

When you make configuration changes to any IDPs or SPs within a federation, including any security domains used by the IDPs or SPs, you should restart the affected IDPs and/or SPs.

2.3.2.3. Creating a Federation Using the Management CLI

The following commands show how an example federation called **new-federation** can be added with the following information:

- The identity provider **IDP.war** is deployed to the JBoss EAP instance at **http://localhost:8080/identity/**
- The service provider **SP.war** is deployed to the JBoss EAP instance at **http://localhost:8080/sales-post/**
- The security domains **idp** and **sp** have been configured correctly for the identity provider and service provider respectively

To configure a federation using the management CLI, you must:

1. Add a new federation.

```
/subsystem=picketlink-federation/federation=new-federation:add
```

2. Add an identity provider to the federation.

```
/subsystem=picketlink-federation/federation=new-federation/identity-
provider=IDP.war:add(url="http://localhost:8080/identity/",security-
domain=idp)
```

3. Add a service provider to the federation.

```
/subsystem=picketlink-federation/federation=new-federation/service-
provider=SP.war:add(url="http://localhost:8080/sales-
post/",security-domain=sp)
```

4. Add a trust domain to the federation.

```
/subsystem=picketlink-federation/federation=new-federation/identity-
provider=IDP.war/trust-domain="localhost:8080":add
```

2.3.2.4. Reference of Attributes for the Federation Subsystem

The **picketlink-federation** subsystem has the following structure:

federation

- saml
- key-store
  - keys
    - key
- identity-provider
  - trust
    - trust-domain
  - role-generator
  - attribute-manager
  - handlers
    - handler
      - handler-parameter
- service-providers
  - service-provider
    - handlers
      - handler
        - handler-parameter

Example Federation Subsystem

```
<subsystem xmlns="urn:jboss:domain:picketlink-federation:2.0">
  <federation name="federation-redirect-with-signatures">
    <key-store file="/jbid_test_keystore.jks" password="store123" sign-
key-alias="servercert" sign-key-password="test123">
      <keys>
        <key name="servercert"
host="${jboss.bind.address:localhost},127.0.0.1"/>
      </keys>
    </key-store>
    <identity-provider name="idp-redirect-sig.war"
url="http://${jboss.bind.address:127.0.0.1}:8080/idp-redirect-sig/"
security-domain="idp" support-signatures="true" strict-post-
binding="false">
      <trust>
        <trust-domain name="${jboss.bind.address:127.0.0.1}"/>
      </trust>
      <handlers>
        <handler class-name="com.mycompany.CustomHandler">
          <handler-parameter name="param1" value="paramValue1"/>
          <handler-parameter name="param2" value="paramValue2"/>
          <handler-parameter name="param3" value="paramValue3"/>
```

```
        </handler>
      </handlers>
    </identity-provider>
    <service-providers>
      <service-provider name="sp-redirect-sig1.war" security-domain="sp"
url="http://${jboss.bind.address:127.0.0.1}:8080/sp-redirect-sig1/" post-
binding="false" support-signatures="true">
        <handlers>
          <handler class-name="com.mycompany.CustomHandler">
            <handler-parameter name="param1" value="paramValue1"/>
            <handler-parameter name="param2" value="paramValue2"/>
            <handler-parameter name="param3" value="paramValue3"/>
          </handler>
        </handlers>
      </service-provider>
      <service-provider name="sp-redirect-sig2.war" security-domain="sp"
url="http://${jboss.bind.address:127.0.0.1}:8080/sp-redirect-sig2/" post-
binding="false" support-signatures="true"/>
    </service-providers>
  </federation>
</subsystem>
```

Table 2.1. federation

Attribute	Default	Description
name		The federation name.

saml

Defines the SAML type. This type defines all configurations about how SAML assertions are processed and created.

Attribute	Default	Description
clock-skew	0	Defines the clock skew for SAML assertions. The value must be specified in milliseconds.
token-timeout	5000	Defines the timeout for SAML assertions. The value must be specified in milliseconds.

key-store

Defines the **KeyStore** type. This type defines how keystores are configured.

Attribute	Default	Description
password		Defines the password for the keystore.
sign-key-alias		Defines the alias to be used when signing documents.

Attribute	Default	Description
sign-key-password		Defines the password for the sign-key-alias.
file		Defines the file location.
relative-to		One of the system-provided named paths, such as **`jboss.home.dir`**, **`user.home`**, **`user.dir`**, relative to which the absolute path will be calculated for the path specified in the file attribute.

keys

Keys Configuration.

key

Defines a Key.

Attribute	Default	Description
name		Defines the name or alias of a key in a given keystore.
host		A single or a comma separated list of strings representing the host names validated by the given key.

identity-provider

Defines the Identity Provider type.

Attribute	Default	Description
name		A unique name for the Identity Provider. The name must be the deployment unit name. For example, **`idp.war`**.
url		URL for this Identity Provider.
support-signatures	false	Indicates if signature is supported.
encrypt	false	Indicates if encryption is supported.
security-domain		The name of a security-domain that will be used to authenticate and authorize users. This attribute is required if the IDP is not external. See the **`external`** attribute for more details.

Attribute	Default	Description
strict-post-binding	true	Indicates if the IDP should always respond using HTTP **POST** binding.
external	false	Indicates if the configuration is a reference to a external IDP.
support-metadata	false	Enable/Disable SAML Metadata Support.
ssl-authentication	false	Indicates if the identity provider should also support HTTP **CLIENT_CERT** authentication.

trust

Groups Trusted Domain Types.

trust-domain

Defines the Trusted Domain Type.

Attribute	Default	Description
name		Defines the domain name.
cert-alias		Defines the certificate alias for this domain.

role-generator

The **RoleGenerator** implementation that will be used to load roles and push them to SAML assertions.

Attribute	Default	Description
name		Defines the role generator name.
class-name		The fully qualified name of the **RoleGenerator** type.
code		Defines an alias which maps to a built-in type.
module		Defines the module to be used when loading **class-name**.

attribute-manager

The **AttributeManager** implementation that will be used to load roles and push them to SAML assertions.

Attribute	Default	Description
name		Defines the attribute manager name.
class-name		The fully qualified name of the **`AttributeManager`** type.
code		Defines an alias which maps to a built-in type.
module		Defines the module to be used when loading **`class-name`**.

handlers

Groups Handler Types.

handler

Defines the Handler Type.

Attribute	Default	Description
name		Defines the handler name.
class-name		Defines the handler class name.
code		Defines an alias which maps to a built-in type.

handler-parameter

Defines the Handler Parameter Type.

Attribute	Default	Description
name		Defines the parameter name.
value		Defines the parameter value.

service-providers

Groups Service Provider types.

service-provider

Defines the Service Provider type.

Attribute	Default	Description

Attribute	Default	Description
name		Name for this instance. This name must be the deployment unit name.
url		URL for this Service Provider.
post-binding	true	Indicates which SAML Binding to use. If it is true, HTTP **POST** binding will be used. Otherwise HTTP REDIRECT binding will be used.
strict-post-binding	true	Indicates which SAML Binding to use. If it is true, HTTP **POST** binding will be used. Otherwise HTTP REDIRECT binding will be used.
support-signatures	false	Indicates if signature is supported.
support-metadata	false	Enable/Disable SAML Metadata Support.
security-domain		Security Domain name used to authenticate users.
error-page	`/error.jsp`	Defines a custom error page location.
logout-page	`/logout.jsp`	Defines a custom logout page location.

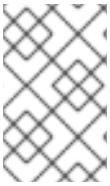
2.4. CONFIGURING IDENTITY STORES FOR IDPS

Since IDPs use security domains, the functionality of an IDP is independent from the actual identity store that backs it. As a result, administrators have many options when configuring security domains for IDPs. The specifics around security domains and login modules can be found in the Security Subsystem section of the JBoss EAP *Security Architecture* guide. Just as with setting up any login module for a security domain, please keep in mind that different identity stores offer different functionality and performance tradeoffs.

The following steps are needed for setting up a security domain that uses an identity store:

NOTE

For the purposes of this document, the Database login module and LDAP login module are shown as examples, but other identity stores and login modules can also be configured for use with IDPs.



NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

1. Set up the identity store.
   Before a security domain and login module can be configured to use an identity provider, the identity provider, and sometimes a connection to that identity provider, must be setup.

   a. Configure the identity store for the Database Login Module.
      The first item needed for a Database-backed identity store is a database for the login module to use.

      The following data points are needed:

      - Usernames
      - Passwords
      - Roles
      - Role Groups

      The Database Login module requires the ability to create a query that maps usernames to passwords, and a query that maps usernames to roles and role groups. This information can be stored within the database in a variety of ways, but creating a database with tables is not in the scope of this guide. For the purposes of this example, it is assumed the following tables have been created:

      Table 2.2. sso-users

username	passwd
Sarah	Testing123!

      Table 2.3. sso-roles

username	role
role-group	Sarah
Sample	SSO-Users

      Creating datasources is not in the scope of this guide. For details on setting up a datasource, see the Datasource Management section of the JBoss EAP *Configuration Guide*.

      For the purposes of this example, it is assumed that a datasource named **`idpDS`** has been created, properly configured, and deployed to the JBoss EAP instance. This datasource has a connection to the database storing the **`sso-users`** and **`sso-roles`** tables.

b. Configure the identity store for the LDAP login module.
A properly configured LDAP server is required prior to setting up the LDAP login module. Unlike the Database login module, a datasource is not needed for setting up the LDAP login module. The basics of LDAP and how it relates to JBoss EAP security are covered in the JBoss EAP *Security Architecture* guide.

i. Set up an LDAP server.
Setting up an LDAP server is not in the scope of this guide. For the purposes of this example, the LDAP server can be reached at **http://ldaphost.example.com:1389/**.

ii. Example directory information.
The directory structure and organization of an LDAP server can vary greatly depending on the use case and organizational needs. For the purposes of this example, the below entries have been created, shown in LDIF format:

```
dn: dc=example,dc=com
objectclass: top
objectclass: dcObject
objectclass: organization
dc: example
o:  Example
#============================
dn: ou=People,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
ou: People
#============================
dn: uid=jsmith,ou=People,dc=example,dc=com
objectclass: top
objectclass: uidObject
objectclass: person
uid: jsmith
cn: John
sn: Smith
userPassword: theduke
#============================
dn: ou=Roles,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
ou: Roles
#============================
dn: cn=Sample,ou=Roles,dc=example,dc=com
objectclass: top
objectclass: groupOfNames
cn: Sample
member: uid=jsmith,ou=People,dc=example,dc=com
description: the Sample group
```

2. Add the security domain.
Once the identity store itself has been setup, and the connection between the JBoss EAP instance and the identity store has been configured, the security domain can be created and configured. The following command shows how to create an empty security domain. Replace **MY-DOMAIN** with the desired name of the security domain.

Management CLI Command for Adding a Security Domain

```
/subsystem=security/security-domain=MY-DOMAIN:add(cache-
type=default)
```

3. Add the authentication section and login module to the security domain. After the empty security domain has been created, the authentication section must be created with a login module added to it. The below command shows how to add an empty authentication section to an existing security domain.

   Management CLI Command for Adding an Authentication Section to a Security Domain

   ```
   /subsystem=security/security-domain=MY-
   DOMAIN/authentication=classic:add
   ```

   Once the empty authentication section has been created, a login module can be added to it and configured to use the desired identity store. After adding a login module to a security domain, a configuration reload is usually required.

   Below is the general command structure for adding a login module and reloading the configuration. Replace **MY-DOMAIN**, **MY-LOGIN-MODULE**, and **MY-CONFIGURATION** with the appropriate values:

   ```
   /subsystem=security/security-domain=MY-
   DOMAIN/authentication=classic/login-module=MY-LOGIN-MODULE:add(MY-
   CONFIGURATION)

   reload
   ```

   - Add a Database login module.

     NOTE

     This example assumes a datasource named **idpDS** has been created in Step 1 and a security domain named **idp-db-domain** was created in Step 2.

     Management CLI Command for Configuring the Authentication Section to use the Database Login Module

     ```
     /subsystem=security/security-domain=idp-db-
     domain/authentication=classic/login-
     module=Database:add(code=Database,flag=required,module-options=
     [("dsJndiName"=>"java:/idpDS"),("principalsQuery"=>"select passwd
     from 'sso-users' where username=?"),("rolesQuery"=>"select role,
     role-group from 'sso-roles' where username=?")])

     reload
     ```

   - Add an LDAP login module.
     The steps necessary for configuring the **LdapExtended** login module can be found in *How to Configure Identity Management*.

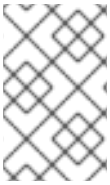
2.5. CONFIGURING SSL/TLS WITH SPS AND IDPS

The basics of SSL/TLS are covered in the JBoss EAP *Security Architecture* guide. Adding SSL/TLS support to a browser-based SSO environment is not much different from adding it to non-SSO environment. Both the IDPs and SPs can have an HTTPS connector added to allow for traffic to be secured.

If one has not already been created, a security realm with an SSL/TLS server identity must be created. It must also be configured with an SSL/TLS certificate.

2.5.1. Enable One-way SSL/TLS for Applications Using Legacy Security Realms

This example assumes that the keystore, **`identity.jks`**, has been copied to the server configuration directory and configured with the given properties. Administrators should substitute their own values for the example ones.



NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, see the JBoss EAP *Management CLI Guide*.

1. Add and configure an HTTPS security realm first. Once the HTTPS security realm has been configured, configure an **`https-listener`** in the **`undertow`** subsystem that references the security realm:

```
batch

/core-service=management/security-realm=HTTPSRealm:add

/core-service=management/security-realm=HTTPSRealm/server-
identity=ssl:add(keystore-path=identity.jks, keystore-relative-
to=jboss.server.config.dir, keystore-password=password1,
alias=appserver)

/subsystem=undertow/server=default-server/https-
listener=https:write-attribute(name=security-realm,
value=HTTPSRealm)

run-batch
```

WARNING

Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 or TLSv1.2 in all affected packages.

2. Restart the JBoss EAP instance for the changes to take effect.

2.6. CONFIGURING AN IDENTITY PROVIDER TO USE CERTIFICATE-BASED AUTHENTICATION

In addition to configuring SPs and IDPs to use SSL/TLS, you can also configure an IDP to use certificate-based authentication. Before setting up an IDP to use certificate-based authentication you will need to configure the IDPs and SPs to use SSL/TLS.

1. Create a client certificate and truststore.
   You must create a certificate and truststore for clients to use to authenticate. You will need to use these in the server configuration as well as with the client's browser.

   Example Client Certificate and Truststore

   ```
   $ keytool -genkeypair -alias client -storetype jks -keyalg RSA -
   keysize 2048 -keypass change_it -keystore client.jks -storepass
   change_it -dname "CN=client,OU=Sales,O=Systems
   Inc,L=Raleigh,ST=NC,C=US" -validity 730 -v

   $ keytool -export -alias client -keystore client.jks -storepass
   change_it -file client.cer

   $ keytool -import -file client.cer -alias client -keystore
   client.truststore
   ```

2. Creating a Security Domain for the IDP.
   You need to create a security domain that uses a certificate-based login module for the IDP to use for authentication. For more details on certificate-based login modules, see the *Login Module Reference*.

   Example Security Domain with a CertificateRoles Login Module

   ```
   /subsystem=security/security-domain=idp-cert:add

   /subsystem=security/security-domain=idp-
   cert/authentication=classic:add

   /subsystem=security/security-domain=idp-
   cert/authentication=classic/login-
   module=CertificateRoles:add(code=CertificateRoles,flag=optional,modu
   le-options=[("password-stacking"=>"useFirstPass"),
   ("securityDomain"=>"idp-cert"),
   ("verifier"=>"org.jboss.security.auth.certs.AnyCertVerifier")])

   /subsystem=security/security-domain=idp-
   cert/jsse=classic:add(truststore=
   {url=>"/path/to/client.jks",password=>change_it})

   reload
   ```

   You also need to configure your IDP to use this security domain. For more details on configuring the IDP, see the Setting up an IDP section.

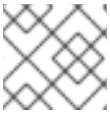
NOTE

You can also use the **RegExUserNameLoginModule** in conjunction with Certificate login modules to extract a username, UID, or other information from the principal name. For more details on the **RegExUserNameLoginModule**, see the JBoss EAP*Login Module Reference*.

3. Import the client certificate into the client's browser. Once the IDP and server configuration is complete, you must configure the client's browser to use the client certificate. This configuration varies between browsers. Once the client's browser is configured to use the client certificate, the client will be able to authenticate with the IDP using the certificate.

2.7. CONFIGURING AN IDENTITY PROVIDER TO USE KERBEROS AUTHENTICATION

In addition to other identity stores, an IDP can also use Kerberos as its authentication mechanism. To setup an IDP to use Kerberos, you will need to do the following:



NOTE

It is assumed you have a working Kerberos environment.

1. Configure the security domains for Kerberos authentication.
   You can use the following commands to configure the security domains required by the IDP. For additional information, see the JBoss EAP Configure the Legacy Security Subsystem section of the *How to Set Up SSO with Kerberos* guide.

```
/subsystem=security/security-domain=host:add(cache-type=default)

/subsystem=security/security-domain=host/authentication=classic:add

/subsystem=security/security-
domain=host/authentication=classic/login-
module=Kerberos:add(code=Kerberos, flag=required, module-options=
[debug=false, storeKey=true, refreshKrb5Config=true, useKeyTab=true,
doNotPrompt=true, keyTab=/home/username/service.keytab,
principal=host/SERVER_NAME@REALM_NAME])

/subsystem=security/security-domain=app-spnego:add(cache-
type=default)

/subsystem=security/security-domain=app-
spnego/authentication=classic:add

/subsystem=security/security-domain=app-
spnego/authentication=classic/login-module=SPNEGO:add(code=SPNEGO,
flag=required, module-options=[serverSecurityDomain=host])
```

IMPORTANT

You should also ensure that all relevant system properties are enabled as well.

For more information on login modules, see Kerberos Login Module and SPNEGO Login Module sections of the JBoss EAP*Login Module Reference* reference.

2. Configure the security domain for the SPs.
   You can use the following commands to configure the security domains required by the SP. For additional information, see the full documentation at Setting up an SP.

```
/subsystem=security/security-domain=sp:add(cache-type=default)

/subsystem=security/security-domain=sp/authentication=classic:add

/subsystem=security/security-domain=sp/authentication=classic/login-
module=org.picketlink.identity.federation.bindings.jboss.auth.SAML2L
oginModule:add(code=org.picketlink.identity.federation.bindings.jbos
s.auth.SAML2LoginModule,flag=required)
```

3. Reload the server for the changes to take effect.

```
reload
```

4. After completing the above steps the following configuration is created.

Example: Security Domains for the IDP and SPs

```
<security-domain name="host" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="debug" value="false"/>
      <module-option name="storeKey" value="true"/>
      <module-option name="refreshKrb5Config" value="true"/>
      <module-option name="useKeyTab" value="true"/>
      <module-option name="doNotPrompt" value="true"/>
      <module-option name="keyTab"
value="/home/username/service.keytab"/>
      <module-option name="principal"
value="HTTP/testserver@MY_REALM"/>
    </login-module>
  </authentication>
</security-domain>
<security-domain name="app-spnego" cache-type="default">
    <authentication>
        <login-module code="SPNEGO" flag="required">
            <module-option name="serverSecurityDomain"
value="host"/>
        </login-module>
    </authentication>
    <mapping>
    ...
    </mapping>
</security-domain>
```

```
<security-domain name="sp" cache-type="default">
    <authentication>
        <login-module
code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule" flag="required"/>
    </authentication>
</security-domain>
```

5. Configure the IDP application.
   The process of configuring the IDP is the same as covered in the Setting up an IDP section, but with the following changes:

   - Declaring an additional dependency for JBoss Negotiation
   - Configure the IDP application to use the security domain with the **SPNEGO** login module

NOTE

When configuring the IDP, you do not need to specify the **PicketLinkSTS** element in the configuration. If it is ommited **PicketLink** will load the default configurations from a file named **core-sts** inside **picketlink-core-*VERSION*.jar**.

Override this configuration only if you need to. For example, change the token timeout or specify a custom Security Token Provider for SAML assertions.

Example: `jboss-deployment-structure.xml` with Kerberos and Picketlink Dependencies

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" services="import"/>
      <module name="org.jboss.security.negotiation"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Example: `jboss-web.xml` in the IDP

```
<jboss-web>
  <security-domain>app-spnego</security-domain>
  <context-root>identity</context-root>
</jboss-web>
```

Example: `picketlink.xml` with the `PicketLinkSTS` Element

```
<PicketLink xmlns="urn:picketlink:identity-
federation:config:2.1">
  <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1">
```

```
<IdentityURL>${idp.url::http://localhost:8080/idp/}</IdentityURL>
    <Trust>
     <Domains>redhat.com,localhost,amazonaws.com</Domains>
    </Trust>
  </PicketLinkIDP>
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML
2IssuerTrustHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML
2LogOutHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML
2AuthenticationHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.Role
sGenerationHandler" />
  </Handlers>
  <!-- The configuration bellow defines a token timeout and a
clock skew. Both  configurations will be used during the SAML
Assertion creation. This configuration is optional. It is defined
only to show you how to set the token timeout and clock skew
configuration.   -->
  <PicketLinkSTS xmlns="urn:picketlink:identity-
federation:config:1.0" TokenTimeout="5000" ClockSkew="0">
    <TokenProviders>
    <TokenProvider

ProviderClass="org.picketlink.identity.federation.core.saml.v1.pr
oviders.SAML11AssertionTokenProvider"
       TokenType="urn:oasis:names:tc:SAML:1.0:assertion"
       TokenElement="Assertion"
TokenElementNS="urn:oasis:names:tc:SAML:1.0:assertion" />
    <TokenProvider

ProviderClass="org.picketlink.identity.federation.core.saml.v2.pr
oviders.SAML20AssertionTokenProvider"
       TokenType="urn:oasis:names:tc:SAML:2.0:assertion"
       TokenElement="Assertion"
TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion" />
    </TokenProviders>
  </PicketLinkSTS>
</PicketLink>
```

IMPORTANT

You must ensure that any roles configured in the IDP's **web.xml** match up with the roles configured in your Kerberos environment. This can be accomplished by configuring a second login module in the IDP's security domain to map the appropriate roles to after the SPNEGO authentication, or by using a mapping provider in the IDP's security domain.

2.8. CHANGES FROM PREVIOUS VERSIONS OF JBOSS EAP

The addition of the Undertow server in JBoss EAP 7 has introduced some changes to how SAML SSO is configured. You will have to account for these changes when setting up IDPs and SPs on JBoss EAP 7, as well as when migrating IDPs and SPs running on previous versions of JBoss EAP.

- Valves are no longer used and their parameters are now configured using **context-param**.
- The dependency declaration has changed.
- SAML authenticators now require **FORM** authentication to be configured in **web.xml**.
- The dynamic account chooser configuration has changed.

2.8.1. Valve and Valve Configuration Changes

In JBoss EAP 7, valves are no longer used, but similar functionality is available in Undertow handlers. The net result is that you no longer need to add the valve declaration in **jboss-web.xml**. For details on this change and how it affects migration in general, see the Migrate Custom Application Valves section of the JBoss EAP *Migration Guide*.

NOTE

Other configuration, such as specifying a security domain, is still configured in **jboss-web.xml**.

Since valves are no longer used, the following items are now configured using **<context-param>** in **web.xml**.

Configuration Provider

```
<context-param>
  <param-name>org.picketlink.federation.saml.CONFIG_PROVIDER</param-name>
  <param-value>MyConfigurationProvider</param-value>
</context-param>
```

Audit Helper

```
<context-param>
  <param-name>org.picketlink.federation.saml.AUDIT_HELPER</param-name>
  <param-value>MyAuditHelper</param-value>
</context-param>
```

Configuration Refresh Internval

```
<context-param>
  <param-
name>org.picketlink.federation.saml.REFRESH_CONFIG_TIMER_INTERVAL</param-
name>
  <param-value>1000</param-value>
</context-param>
```

Character Encoding

```
<context-param>
  <param-name>org.picketlink.federation.saml.CHARACTER_ENCODING</param-
name>
  <param-value>UTF-8</param-value>
</context-param>
```

Pass User Principal to Attribute Manager

```
<context-param>
  <param-
name>org.picketlink.federation.saml.PASS_USER_PRINCIPAL_TO_ATTRIBUTE_MANAG
ER</param-name>
  <param-value>true</param-value>
</context-param>
```

Custom Location for picketlink.xml

```
<context-param>
  <param-name>CONFIG_FILE</param-name>
  <param-value>/path/to/picketlink.xml</param-value>
</context-param>
```

2.8.2. Dependency Declaration Changes

As with previous versions of JBoss EAP, you will still need to declare the proper dependencies using the **jboss-deployment-structure.xml** file, but you will now also need to include **services="import"** in the module. In previous versions of JBoss EAP, you would have declared this same dependency, but would have declared a valve to install the SAML authenticators and excluded **services="import"**. With the introduction of Undertow in JBoss EAP 7, you now use **services="import"** to install the SAML authenticators.

Using jboss-deployment-structure.xml to Declare Dependencies

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" services="import"/>
    </dependencies>
  </deployment>
</jboss>
```

2.8.3. Authenticator Changes

The SAML authenticators extend the **FORM** authenticator in JBoss EAP. In order to enable the SAML authenticators, you must define a **<login-config>** configured for **FORM** authentication in **web.xml**.

Example `web.xml` File

```
<web-app>
  ...
```

```
  <login-config>
    <auth-method>FORM</auth-method>
    ...
  </login-config>
</web-app>
```

2.8.4. Dynamic Account Chooser Changes

In previous versions of JBoss EAP, certain parts of the dynamic account chooser were configured using parameters passed to the valve. Since valves are no longer used in JBoss EAP 7, this configuration has been moved into the **Provider** element contained in **picketlink.xml**. For more details on those configuration options, see the Configuring a Dynamic Account Chooser section.

Previous versions of JBoss EAP also defined the **org.picketlink.identity.federation.bindings.tomcat.sp.AbstractAccountChooserValve.AccountIDPMapProvider** interface, which was implemented when creating a custom **IDPMapProvider**. In JBoss EAP 7, this interface no longer exists, and you must now implement the **org.picketlink.identity.federation.web.config.IdentityURLConfigurationProvider** interface. The contract on this new interface is the same.

2.9. ADDITIONAL FEATURES

2.9.1. SAML Assertion Encryption

In addition to offering SSL/TLS encryption between IDPs and SPs, the SAML assertions themselves may also be encrypted. This is useful in securing SAML v2 assertions that are transmitted in an unsecured manner, for example not using SSL/TLS.

To enable encryption of security assertions directly in IDPs and SPs, the following steps must be performed to both the IDP and SP **picketlink.xml** files:

1. Enable **Encrypt** and **SupportsSignatures**.
   To enable encryption, the **<PicketLinkIDP>** and **<PicketLinkSP>** must be updated.

   For the IDP, add or update the **Encrypt** and **SupportsSignatures** attributes in **<PicketLinkIDP>** to be true:

   ```
   <PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
     ...
     <PicketLinkIDP xmlns="urn:picketlink:identity-
   federation:config:2.1" Encrypt="true" SupportsSignatures="true">
       ...
     </PicketLinkIDP>
   </PicketLink>
   ```

   For the SP, add or update the **SupportsSignatures** attribute in **<PicketLinkSP>** to be true:

   ```
   <PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
     ...
     <PicketLinkSP xmlns="urn:picketlink:identity-
   federation:config:2.1" SupportsSignatures="true">
   ```

```
    ...
  </PicketLinkSP>
</PicketLink>
```

2. Add handlers.
In addition, handlers must be added to **<Handlers>**.

For the IDP add **SAML2EncryptionHandler** and **SAML2SignatureValidationHandler** to the **picketlink.xml** file:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Is
suerTrustHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Lo
gOutHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Au
thenticationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGe
nerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2En
cryptionHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureValidationHandler" />
  </Handlers>
</PicketLink>
```

For the SP add **SAML2SignatureGenerationHandler** and **SAML2SignatureValidationHandler** to the **picketlink.xml** file:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Lo
gOutHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Au
thenticationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGe
nerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureGenerationHandler" />
    <Handler
```

```
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureValidationHandler" />
  </Handlers>
</PicketLink>
```

WARNING

Handlers are implemented using a Chain of Responsibility, with each individual handler performing logic on request and responses in the order defined in **picketlink.xml**. It is very important to pay attention to the order in which the handlers are configured.

The **SAML2SignatureGenerationHandler** must not be configured in the same chain as the **SAML2EncryptoinHandler**. This will cause SAML messages will be signed several times.

3. Configure key provider.
Lastly, a **<KeyProvider>** element must be added to **BOTH picketlink.xml** files. This element provides the location and credentials for accessing the Java keystore used for encrypting and decrypting security assertions. An example of generating a Java keystore can be found in the JBoss EAP *How to Configure Server Security* guide.

For the IDP, the element should be added to **<PicketLinkIDP>**:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1" Encrypt="true" SupportsSignatures="true">
    ...
    <KeyProvider
ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyM
anager">
      <Auth Key="KeyStoreURL" Value="/my_keystore.jks" />
      <Auth Key="KeyStorePass" Value="store123" />
      <Auth Key="SigningKeyPass" Value="test123" />
      <Auth Key="SigningKeyAlias" Value="servercert" />
      <ValidatingAlias Key="idp.example.com" Value="servercert" />
      <ValidatingAlias Key="localhost" Value="servercert" />
      <ValidatingAlias Key="sp1.example.com" Value="servercert" />
      <ValidatingAlias Key="sp2.example.com" Value="servercert" />
    </KeyProvider>
    ...
  </PicketLinkIDP>
  ...
<PicketLink>
```

For the SP the element should be added to **<PicketLinkSP>**:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
```

```
  <PicketLinkSP xmlns="urn:picketlink:identity-
federation:config:2.1" SupportsSignatures="true">
    ...
    <KeyProvider
ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyM
anager">
      <Auth Key="KeyStoreURL" Value="/my_keystore.jks" />
      <Auth Key="KeyStorePass" Value="store123" />
      <Auth Key="SigningKeyPass" Value="test123" />
      <Auth Key="SigningKeyAlias" Value="servercert" />
      <ValidatingAlias Key="idp.example.com" Value="servercert" />
      <ValidatingAlias Key="localhost" Value="servercert" />
    </KeyProvider>
    ...
  </PicketLinkSP>
</PicketLink>
```

NOTE

In order to properly encrypt and decrypt assertions, the IDP needs to generate signatures and the SP needs to verify those signatures as well as identify where they came from. This is accomplished using the **<ValidatingAlias>** element. IDPs need to have a**<ValidatingAlias>** for each trusted server/domain that is trusted, which is every entry in the **<Trust>** element. SPs need to have a**<ValidatingAlias>** for each server/domain containing an IDP.

2.9.2. Digital Signatures in Assertions

Digital Signatures allow IDPs to sign their SAML v2 security assertions and have those signatures and assertions validated by the SPs. This is useful for validating the authenticity of assertions, especially for assertions that are transmitted in an unsecured manner, for example not using SSL/TLS.

To enable digital signatures in security assertions directly in IDPs and SPs, the following steps must be performed to both the IDP and SP **picketlink.xml** files:

1. Enable **SupportsSignatures**.
   To enable digital signatures, the **<PicketLinkIDP>** and **<PicketLinkSP>** elements must be updated.

   For the IDP and SP, add or update the **SupportsSignatures** attribute in **<PicketLinkSP>** to be true:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1" SupportsSignatures="true">
    ...
  </PicketLinkIDP>
</PicketLink>
```

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <PicketLinkSP xmlns="urn:picketlink:identity-
```

```
federation:config:2.1" SupportsSignatures="true">
    ...
  </PicketLinkSP>
</PicketLink>
```

2. Add handlers.
   In addition, handlers must be added to **<Handlers>**.

   For the IDP and SP, add **SAML2SignatureGenerationHandler** and **SAML2SignatureValidationHandler** to the **picketlink.xml** file:

IDP picketlink.xml

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Is
suerTrustHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Lo
gOutHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Au
thenticationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGe
nerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureGenerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureValidationHandler" />
  </Handlers>
</PicketLink>
```

SP picketlink.xml

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Lo
gOutHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Au
thenticationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGe
nerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
```

```
gnatureGenerationHandler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Si
gnatureValidationHandler" />
  </Handlers>
</PicketLink>
```

WARNING

Handlers are implemented using a Chain of Responsibility, with each individual handler performing logic on request and responses in the order defined in **picketlink.xml**. It is very important to pay attention to the order in which the handlers are configured.

The **SAML2SignatureGenerationHandler** must not be configured in the same chain as the **SAML2EncryptionHandler**. This will cause SAML messages will be signed several times.

3. Configure key provider.
   Lastly, a **<KeyProvider>** element must be added to **BOTH picketlink.xml** files. This element provides the location and credentials for accessing the Java keystore used for signing security assertions. An example of generating a Java keystore can be found in the JBoss EAP *How to Configure Server Security* guide.

   For the IDP, the element should be added to **<PicketLinkIDP>**:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1" SupportsSignatures="true">
    ...
    <KeyProvider
ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyM
anager">
      <Auth Key="KeyStoreURL" Value="/my_keystore.jks" />
      <Auth Key="KeyStorePass" Value="store123" />
      <Auth Key="SigningKeyPass" Value="test123" />
      <Auth Key="SigningKeyAlias" Value="servercert" />
      <ValidatingAlias Key="idp.example.com" Value="servercert" />
      <ValidatingAlias Key="localhost" Value="servercert" />
      <ValidatingAlias Key="sp1.example.com" Value="servercert" />
      <ValidatingAlias Key="sp2.example.com" Value="servercert" />
    </KeyProvider>
    ...
  </PicketLinkIDP>
  ...
<PicketLink>
```

   For the SP the element should be added to **<PicketLinkSP>**:

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  ...
  <PicketLinkSP xmlns="urn:picketlink:identity-
federation:config:2.1" SupportsSignatures="true">
    ...
    <KeyProvider
ClassName="org.picketlink.identity.federation.core.impl.KeyStoreKeyM
anager">
      <Auth Key="KeyStoreURL" Value="/my_keystore.jks" />
      <Auth Key="KeyStorePass" Value="store123" />
      <Auth Key="SigningKeyPass" Value="test123" />
      <Auth Key="SigningKeyAlias" Value="servercert" />
      <ValidatingAlias Key="idp.example.com" Value="servercert" />
      <ValidatingAlias Key="localhost" Value="servercert" />
    </KeyProvider>
    ...
  </PicketLinkSP>
</PicketLink>
```

NOTE

In order to properly encrypt and decrypt assertions, the IDP needs to generate signatures and the SP needs to verify those signatures as well as identify where they came from. This is accomplished using the **<ValidatingAlias>** element. IDPs need to have a**<ValidatingAlias>** for each trusted server/domain that is trusted, which is every entry in the **<Trust>** element. SPs need to have a**<ValidatingAlias>** for each server/domain containing an IDP.

2.9.3. Configuring a Dynamic Account Chooser

If a service provider is configured with multiple identity providers, you can configure that service provider to prompt the user to choose which IDP to use to authenticate their credentials. To configure a service provider with a dynamic account chooser, must to do the following:

1. Configure all identity providers.
   For more details on setting up identity providers, see the Setting up an IDP section.

2. Configure a **WEB-INF/idpmap.properties** file.
   You need to create a **WEB-INF/idpmap.properties** file that lists all available identity providers using the format **name=url**.

   Example **WEB-INF/idpmap.properties**

   ```
   Domain=http://localhost:8080/idp/
   Domain-Alt=http://localhost:8080/idp-alt/
   ```

3. Create an account chooser landing page.
   In order for the user to select an identity provider to authenticate against, you must create an account chooser landing page to present to them, and include it in your service provider. This page should contain links to all identity providers that you want to allow them to authenticate against.

Example `accountChooser.html`

```
<html>
  <head>...</head>
  <body>
    <h1>Account Chooser</h1>
    <ul>
      <li><a href="?idp=Domain">Domain</a>
      <li><a href="?idp=Domain-Alt">Domain Alt</a>
    </ul>
  </body>
</html>
```

4. Configure the **`IdentityURL`** element in **`picketlink.xml`**.
   You will also need to configure the **`IdentityURL`** element in **`picketlink.xml`** to reference the account chooser landing page. For more details on configuring the rest of **`picketlink.xml`** for the service provider, please see the Setting up an SP section.

Example `picketlink.xml`

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkSP xmlns="urn:picketlink:identity-
federation:config:2.1" BindingType="REDIRECT">
    <IdentityURL>
      <Provider Page="/accountChooser.html"/>
    </IdentityURL>
    ...
  </PicketLinkSP>
  <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
    ...
  </Handlers>
</PicketLink>
```

You can configure additional options for the dynamic account chooser using the attributes available in the **`Provider`** element.

Table 2.4. Provider Element Attributes

Option	Type	Default	Description
Page	String	/accountChooser.html	The name of the HTML/JSP page for listing the different IDP accounts.
Expiration	Integer	-1	The cookie expiry in seconds. Default is **-1**, which means the cookie expires when the browser is closed.
DefaultURL	String		URL of the default IDP.

Option	Type	Default	Description
Domain	String		The domain name to be used for the cookie that is sent to the user's browser.
Type	String		The fully qualified name of the implementation for IDP Mapping to replace the default implementation. This implementation must implement **`org.picketlink.identity.federation.web.config.IdentityURLConfigurationProvider`**. The default implementation uses the **`WEB-INF/idpmap.properties`** file in your SP web application.

IMPORTANT

Configuring a dynamic account chooser using the **`picketlink-federation`** subsystem is not supported.

2.9.4. Handling AJAX Requests

In certain instances, SPs may need to receive AJAX requests to secured resources. This is handled automatically without the need of any additional configuration, and enables authenticated and authorized users to make AJAX calls.

This is accomplished by checking for the existence of the **`X-Requested-With`** header in the request. AJAX requests are identified by the value **`XMLHttpRequest`** in the **`X-Requested-With`** header. In addition, in cases where a user is not authenticated and sends a request to both the IDP and SP using AJAX, PicketLink will respond with a **`403`** HTTP status code instead of the login page.

Revised on 2018-08-06 14:41:33 EDT