Keycloak SAML Client Adapter Reference Guide

SAML 2.0 Client Adapters

1.8.0.CR3

Preface

In some of the example listings, what is meant to be displayed on one line does not fit inside the available page width. These lines have been broken up. A '\' at the end of a line means that a break has been introduced to fit in the page, with the following lines indented. So:

```
Let's pretend to have an extremely \
long line that \
does not fit
This one is short
```

Is really:

```
Let's pretend to have an extremely long line that does not fit
This one is short
```

Chapter 1. Overview

This document describes the Keycloak SAML client adapter and how it can be configured for a variety of platforms. The Keycloak SAML client adapter is a standalone component that provides generic SAML 2.0 support for your web applications. There are no Keycloak server extensions built into it. As long as the IDP you are talking to supports standard SAML, the Keycloak SAML client adapter should be able to integrate with it.

Chapter 2. General Adapter Config

Each SAML adapter supported by Keycloak can be configured by a simple XML text file. This is what one might look like:

```
<keycloak-saml-adapter>
    <SP entityID="http://localhost:8081/sales-post-sig/"
        sslPolicy="EXTERNAL"
                        nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified"
        logoutPage="/logout.jsp"
        forceAuthentication="false"
        isPassive="false">
        <Keys>
            <Key signing="true" >
                <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                      <PrivateKey alias="http://localhost:8080/sales-post-sig/"
 password="test123"/>
                   <Certificate alias="http://localhost:8080/sales-post-sig/"/>
                </KeyStore>
            </Key>
        </Keys>
        <PrincipalNameMapping policy="FROM_NAME_ID"/>
        <RoleMapping>
            <Attribute name="Role"/>
        </RoleMapping>
        <IDP entityID="idp"
             signaturesRequired="true">
        <SingleSignOnService requestBinding="POST"
                                bindingUrl="http://localhost:8081/auth/realms/
demo/protocol/saml"
                    />

            <SingleLogoutService
                    requestBinding="POST"
                    responseBinding="POST"
                        postBindingUrl="http://localhost:8081/auth/realms/demo/
protocol/saml"
                         redirectBindingUrl="http://localhost:8081/auth/realms/
demo/protocol/saml"
                    />
            <Keys>
                <Key signing="true">
                <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                        <Certificate alias="demo"/>
                    </KeyStore>
```

```
                </Key>
            </Keys>
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

Some of these configuration switches may be adapter specific and some are common across all adapters. For Java adapters you can use `${...}` enclosure as System property replacement. For example `${jboss.server.config.dir}`.

2.1. SP Element

Here is the explanation of the SP element attributes

```
<SP entityID="sp"
    sslPolicy="ssl"
    nameIDPolicyFormat="format"
    forceAuthentication="true"
    isPassive="false">
...
</SP>
```

entityID
: This is the identifier for this client. The IDP needs this value to determine who the client is that is communicating with it. *REQUIRED.*

sslPolicy
: This is the SSL policy the adapter will enforce. Valid values are: ALL, EXTERNAL, and NONE. For ALL, all requests must come in via HTTPS. For EXTERNAL, only non-private IP addresses must come over the wire via HTTPS. For NONE, no requests are required to come over via HTTPS. This is *OPTIONAL.* and defaults to EXTERNAL.

nameIDPolicyFormat
: SAML clients can request a specific NameID Subject format. Fill in this value if you want a specific format. It must be a standard SAML format identifier, i.e. `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` *OPTIONAL.*. By default, no special format is requested.

forceAuthentication
: SAML clients can request that a user is re-authenticated even if they are already logged in at the IDP. Set this to `true` if you want this. *OPTIONAL.*. Set to `false` by default.

isPassive
: SAML clients can request that a user is never asked to authenticate even if they are not logged in at the IDP. Set this to `true` if you want this. Do not use together with `forceAuthentication` as they are opposite. *OPTIONAL.*. Set to `false` by default.

2.2. SP Keys and Key elements

If the IDP requires that the SP sign all of its requests and/or if the IDP will encrypt assertions, you must define the keys used to do this. For client signed documents you must define both the private and public key or certificate that will be used to sign documents. For encryption, you only have to define the private key that will be used to decrypt.

There are two ways to describe your keys. Either they are stored within a Java KeyStore or you can cut and paste the keys directly within `keycloak-saml.xml` in the PEM format.

```
        <Keys>
            <Key signing="true" >
                <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                      <PrivateKey alias="http://localhost:8080/sales-post-sig/"
 password="test123"/>
                   <Certificate alias="http://localhost:8080/sales-post-sig/"/>
                </KeyStore>
            </Key>
        </Keys>
```

The `Key` element has two optional attributes `signing` and `encryption`. When set to true these tell the adapter what the key will be used for. If both attributes are set to true, then the key will be used for both signing documents and decrypting encrypted assertions. You must set at least one of these attributes to true.

2.2.1. KeyStore element

file
: File path to the key store. *OPTIONAL.* The file or resource attribute must be set.

resource
: WAR resource path to the KeyStore. This is a path used in method call to ServletContext.getResourceAsStream(). *OPTIONAL.* The file or resource attribute must be set.

password
: The password of the KeyStore *REQUIRED.*

You can and must also specify references to your private keys and certificates within the Java KeyStore. The `PrivateKey` and `Certificate` elements do this. The `alias` attribute defines the alias within the KeyStore for the key. For `PrivateKey`, a password is required to access this key specify that value in the `password` attribute.

2.2.2. Key PEMS

Within the `Key` element you alternatively declare your keys and certificates directly using the sub elements `PrivateKeyPem`, `PublicKeyPem`, and `CertificatePem`. The values contained in these elements must conform to the PEM key format. You usually use this option if you are generating keys using `openssl`

2.3. SP PrincipalNameMapping element

This element is optional. When creating a Java Principal object that you obtain from methods like HttpServletRequest.getUserPrincipal(), you can define what name that is returned by the Principal.getName() method. The `policy` attribute defines the policy used to populate this value. The values are `FROM_NAME_ID`. This policy just grabs whatever the SAML subject value is. The other is `FROM_ATTRIBUTE`. This will pull the value of Principal.getName() from one of the attributes in the SAML assertion received from the server. The default value is `FROM_NAME_ID`.

2.4. RoleIdentifiers element

```
<RoleIdentifiers>
     <Attribute name="Role"/>
     <Attribute name="member"/>
     <Attribute name="memberOf"/>
</RoleIdentifiers>
```

This element is optional. It defines which SAML attribute values in the assertion should be mapped to a Java EE role. By default `Role` attribute values are converted to Java EE roles. Some IDPs send roles via a `member` or `memberOf` attribute assertion. You define one or more `Attribute` elements to specify which SAML attributes must be converted into roles.

2.5. IDP Element

Everything in the IDP element describes the settings for the IDP the SP is communicating with.

```
<IDP entityID="idp"
     signaturesRequired="true"
     signatureAlgorithm="RSA_SHA1"
```

```
     signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-c14n#">
...
</IDP>
```

entityID
: This is the issuer ID of the IDP. *REQUIRED.*.

signaturesRequired
: If set to true, the client adapter will sign every document it sends to the IDP. Also, the client will expect that the IDP will be signing an documents sent to it. This switch sets the default for all request and response types, but you will see later that you have some fine grain control over this. *OPTIONAL.*

signatureAlgorithm
: This is the signature algorithm that the IDP expects signed documents to use *OPTIONAL.*. The default value is RSA_SHA256, but you can also use RSA_SHA1, RSA_256, RSA_512, and DSA_SHA1.

signatureCanonicalizationMethod
: This is the signature canonicalization method that the IDP expects signed documents to use *OPTIONAL.*. The default value is `http://www.w3.org/2001/10/xml-exc-c14n#` and should be good for most IDPs.

2.6. IDP SingleSignOnService sub element

The `SignleSignOnService` sub element defines the login SAML endpoint of the IDP.

```
<SingleSignOnService signRequest="true"
                     validateResponseSignature="true"
                     requestBinding="post"
                     bindingUrl="url"/>
```

signRequest
: Should the client sign authn requests? *OPTIONAL.*. Defaults to whatever the IDP `signaturesRequired` element value is.

validateResponseSignature
: Should the client expect the IDP to sign the assertion response document sent back from an auhtn request? *OPTIONAL.* Defaults to whatever the IDP `signaturesRequired` element value is.

requestBinding
: This is the SAML binding type used for communicating with the IDP *OPTIONAL.*. The default value is POST, but you can set it to REDIRECT as well.

responseBinding
: SAML allows the client to request what binding type it wants authn responses to use. The values of this can be POST or REDIRECT *OPTIONAL.*. The default is that the client will not request a specific binding type for responses.

bindingUrl
: This is the URL for the ID login service that the client will send requests to. *REQUIRED.*.

2.7. IDP SingleSignOnService sub element

The `SignleSignOnService` sub element defines the login SAML endpoint of the IDP.

```
<SingleLogoutService validateRequestSignature="true"
                     validateResponseSignature="true"
                     signRequest="true"
                     signResponse="true"
                     requestBinding="redirect"
                     responseBinding="post"
                     postBindingUrl="posturl"
                     redirectBindingUrl="redirecturl">
```

signRequest
: Should the client sign logout requests it makes to the IDP? *OPTIONAL.*. Defaults to whatever the IDP `signaturesRequired` element value is.

signResponse
: Should the client sign logout responses it sends to the IDP requests? *OPTIONAL.*. Defaults to whatever the IDP `signaturesRequired` element value is.

validateRequestSignature
: Should the client expect signed logout request documents from the IDP? *OPTIONAL.* Defaults to whatever the IDP `signaturesRequired` element value is.

validateResponseSignature
: Should the client expect signed logout response documents from the IDP? *OPTIONAL.* Defaults to whatever the IDP `signaturesRequired` element value is.

requestBinding
: This is the SAML binding type used for communicating SAML requests to the IDP *OPTIONAL.*. The default value is POST, but you can set it to REDIRECT as well.

responseBinding
: This is the SAML binding type used for communicating SAML responses to the IDP The values of this can be POST or REDIRECT *OPTIONAL.*. The default value is POST, but you can set it to REDIRECT as well.

postBindingUrl
: This is the URL for the IDP's logout service when using the POST binding. *REQUIRED* if using the POST binding at all.

redirectBindingUrl
: This is the URL for the IDP's logout service when using the REDIRECT binding. *REQUIRED* if using the REDIRECT binding at all.

2.8. IDP Keys subelement

The Keys sub element of IDP is only used to define the certificate or public key to use to verify documents signed by the IDP. It is defined in the same way as the *SP's Key's element*. But again, you only have to define one certificate or public key reference.

Chapter 3. JBoss/Wildfly Adapter

To be able to secure WAR apps deployed on JBoss EAP 6.x or Wildfly, you must install and configure the Keycloak SAML Adapter Subsystem. You then provide a keycloak config, `/WEB-INF/keycloak-saml.xml` file in your WAR and change the auth-method to KEYCLOAK-SAML within web.xml. Both methods are described in this section.

3.1. Adapter Installation

SAML Adapters are no longer included with the appliance or war distribution.Each adapter is a separate download on the Keycloak download site. They are also available as a maven artifact.

Install on Wildfly 9 or 10:

```
$ cd $WILDFLY_HOME
$ unzip keycloak-saml-wildfly-adapter-dist.zip
```

Install on JBoss EAP 6.x:

```
$ cd $JBOSS_HOME
$ unzip keycloak-saml-eap6-adapter-dist.zip
```

This zip file creates new JBoss Modules specific to the Wildfly Keycloak SAML Adapter within your Wildfly distro.

After adding the Keycloak modules, you must then enable the Keycloak SAML Subsystem within your app server's server configuration: `domain.xml` or `standalone.xml`.

There is a CLI script that will help you modify your server configuration. Start the server and run the script from the server's bin directory:

```
$ cd $JBOSS_HOME/bin
$ jboss-cli.sh -c --file=adapter-install-saml.cli
```

The script will add the extension, subsystem, and optional security-domain as described below.

```
<server xmlns="urn:jboss:domain:1.4">
```

```
    <extensions>
        <extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>
          ...
    </extensions>

    <profile>
        <subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1"/>
         ...
    </profile>
```

The keycloak security domain should be used with EJBs and other components when you need the security context created in the secured web tier to be propagated to the EJBs (other EE component) you are invoking. Otherwise this configuration is optional.

```
<server xmlns="urn:jboss:domain:1.4">
 <subsystem xmlns="urn:jboss:domain:security:1.2">
    <security-domains>
...
      <security-domain name="keycloak">
         <authentication>
           <login-module code="org.keycloak.adapters.jboss.KeycloakLoginModule"
                         flag="required"/>
          </authentication>
      </security-domain>
    </security-domains>
```

For example, if you have a JAX-RS service that is an EJB within your WEB-INF/classes directory, you'll want to annotate it with the @SecurityDomain annotation as follows:

```
import org.jboss.ejb3.annotation.SecurityDomain;
import org.jboss.resteasy.annotations.cache.NoCache;

import javax.annotation.security.RolesAllowed;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import java.util.ArrayList;
import java.util.List;
```

```
@Path("customers")
@Stateless
@SecurityDomain("keycloak")
public class CustomerService {

    @EJB
    CustomerDB db;

    @GET
    @Produces("application/json")
    @NoCache
    @RolesAllowed("db_user")
    public List<String> getCustomers() {
        return db.getCustomers();
    }
}
```

We hope to improve our integration in the future so that you don't have to specify the @SecurityDomain annotation when you want to propagate a keycloak security context to the EJB tier.

3.2. Required Per WAR Configuration

This section describes how to secure a WAR directly by adding config and editing files within your WAR package.

The first thing you must do is create a `keycloak-saml.xml` adapter config file within the `WEB-INF` directory of your WAR. The format of this config file is describe in the *general adapter configuration* section.

Next you must set the `auth-method` to `KEYCLOAK-SAML` in `web.xml`. You also have to use standard servlet security to specify role-base constraints on your URLs. Here's an example pulled from one of the examples that comes distributed with Keycloak.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/
xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

 <module-name>customer-portal</module-name>

    <security-constraint>
        <web-resource-collection>
```

```
            <web-resource-name>Admins</web-resource-name>
            <url-pattern>/admin/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/customers/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>KEYCLOAK-SAML</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

3.3. Securing WARs via Keycloak SAML Subsystem

You do not have to crack open a WAR to secure it with Keycloak. Alternatively, you can externally secure it via the Keycloak SAML Adapter Subsystem. While you don't have to specify KEYCLOAK-SAML as an `auth-method`, you still have to define the `security-constraints` in `web.xml`. You do not, however, have to create a `WEB-INF/keycloak-saml.xml` file. This metadata is instead defined within XML in your server's `domain.xml` or `standalone.xml` subsystem configuration section.

```
```

```
<extensions>
  <extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>
</extensions>

<profile>
  <subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1">
    <secure-deployment name="WAR MODULE NAME.war">
      <SP entityID="APPLICATION URL">
        ...
      </SP>
    </secure-deployment>
  </subsystem>
</profile>
```

The `secure-deployment name` attribute identifies the WAR you want to secure. Its value is the `module-name` defined in `web.xml` with `.war` appended. The rest of the configuration uses the same XML syntax as `keycloak-saml.xml` configuration defined in *general adapter configuration*.

An example configuration:

```
<subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1">
  <secure-deployment name="saml-post-encryption.war">
    <SP entityID="http://localhost:8080/sales-post-enc/"
        sslPolicy="EXTERNAL"
                        nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified"
        logoutPage="/logout.jsp"
        forceAuthentication="false">
      <Keys>
        <Key signing="true" encryption="true">
          <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                      <PrivateKey  alias="http://localhost:8080/sales-post-enc/"
 password="test123"/>
            <Certificate alias="http://localhost:8080/sales-post-enc/"/>
          </KeyStore>
        </Key>
      </Keys>
      <PrincipalNameMapping policy="FROM_NAME_ID"/>
      <RoleIdentifiers>
        <Attribute name="Role"/>
      </RoleIdentifiers>
      <IDP entityID="idp">
        <SingleSignOnService signRequest="true"
            validateResponseSignature="true"
            requestBinding="POST"
```

```
              bindingUrl="http://localhost:8080/auth/realms/saml-demo/protocol/
saml"/>

        <SingleLogoutService
            validateRequestSignature="true"
            validateResponseSignature="true"
            signRequest="true"
            signResponse="true"
            requestBinding="POST"
            responseBinding="POST"
          postBindingUrl="http://localhost:8080/auth/realms/saml-demo/protocol/
saml"
              redirectBindingUrl="http://localhost:8080/auth/realms/saml-demo/
protocol/saml"/>
        <Keys>
          <Key signing="true" >
            <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
              <Certificate alias="saml-demo"/>
            </KeyStore>
          </Key>
        </Keys>
      </IDP>
    </SP>
   </secure-deployment>
</subsystem>
```

Chapter 4. Tomcat 6, 7 and 8 SAML adapters

To be able to secure WAR apps deployed on Tomcat 6, 7 and 8 you must install the Keycloak Tomcat 6, 7 or 8 SAML adapter into your Tomcat installation. You then have to provide some extra configuration in each WAR you deploy to Tomcat. Let's go over these steps.

4.1. Adapter Installation

Adapters are no longer included with the appliance or war distribution. Each adapter is a separate download on the Keycloak download site. They are also available as a maven artifact.

You must unzip the adapter distro into Tomcat's `lib/` directory. Including adapter's jars within your WEB-INF/lib directory will not work! The Keycloak SAML adapter is implemented as a Valve and valve code must reside in Tomcat's main lib/ directory.

```
$ cd $TOMCAT_HOME/lib
$ unzip keycloak-saml-tomcat6-adapter-dist.zip
    or
$ unzip keycloak-saml-tomcat7-adapter-dist.zip
    or
$ unzip keycloak-saml-tomcat8-adapter-dist.zip
```

4.2. Required Per WAR Configuration

This section describes how to secure a WAR directly by adding config and editing files within your WAR package.

The first thing you must do is create a `META-INF/context.xml` file in your WAR package. This is a Tomcat specific config file and you must define a Keycloak specific Valve.

```
<Context path="/your-context-path">
   <Valve className="org.keycloak.adapters.saml.tomcat.SamlAuthenticatorValve"/
>
</Context>
```

Next you must create a `keycloak-saml.xml` adapter config file within the `WEB-INF` directory of your WAR. The format of this config file is describe in the *general adapter configuration* section.

Finally you must specify both a `login-config` and use standard servlet security to specify role-base constraints on your URLs. Here's an example:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/
xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

 <module-name>customer-portal</module-name>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

Chapter 5. Jetty 9.x SAML Adapters

Keycloak has a separate SAML adapter for Jetty 9.x. You then have to provide some extra configuration in each WAR you deploy to Jetty. Let's go over these steps.

5.1. Adapter Installation

Adapters are no longer included with the appliance or war distribution.Each adapter is a separate download on the Keycloak download site. They are also available as a maven artifact.

You must unzip the Jetty 9.x distro into Jetty 9.x's root directory. Including adapter's jars within your WEB-INF/lib directory will not work!

```
$ cd $JETTY_HOME
$ unzip keycloak-saml-jetty92-adapter-dist.zip
```

Next, you will have to enable the keycloak module for your jetty.base.

```
$ cd your-base
$ java -jar $JETTY_HOME/start.jar --add-to-startd=keycloak
```

5.2. Required Per WAR Configuration

This section describes how to secure a WAR directly by adding config and editing files within your WAR package.

The first thing you must do is create a `WEB-INF/jetty-web.xml` file in your WAR package. This is a Jetty specific config file and you must define a Keycloak specific authenticator within it.

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure//EN" "http://
www.eclipse.org/jetty/configure_9_0.dtd">
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
    <Get name="securityHandler">
        <Set name="authenticator">
                                                                        <New
 class="org.keycloak.adapters.saml.jetty.KeycloakSamlAuthenticator">
            </New>
        </Set>
```

```
    </Get>
</Configure>
```

Next you must create a `keycloak-saml.xml` adapter config file within the `WEB-INF` directory of your WAR. The format of this config file is describe in the *general adapter configuration* section.

Finally you must specify both a `login-config` and use standard servlet security to specify role-base constraints on your URLs. Here's an example:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/
xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

 <module-name>customer-portal</module-name>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

Chapter 6. Jetty 8.1.x SAML Adapter

Keycloak has a separate SAML adapter for Jetty 8.1.x that you will have to install into your Jetty installation. You then have to provide some extra configuration in each WAR you deploy to Jetty. Let's go over these steps.

6.1. Adapter Installation

Adapters are no longer included with the appliance or war distribution.Each adapter is a separate download on the Keycloak download site. They are also available as a maven artifact.

You must unzip the Jetty 8.1.x distro into Jetty 8.1.x's root directory. Including adapter's jars within your WEB-INF/lib directory will not work!

```
$ cd $JETTY_HOME
$ unzip keycloak-saml-jetty81-adapter-dist.zip
```

Next, you will have to enable the keycloak option. Edit start.ini and add keycloak to the options

```
#=====================================================================
# Start classpath OPTIONS.
# These control what classes are on the classpath
# for a full listing do
#   java -jar start.jar --list-options
#---------------------------------------------------------------------
OPTIONS=Server,jsp,jmx,resources,websocket,ext,plus,annotations,keycloak
```

6.2. Required Per WAR Configuration

Enabling Keycloak for your WARs is the same as the Jetty 9.x adapter. See *Required Per WAR Configuration*

Chapter 7. Java Servlet Filter Adapter

If you want to use SAML with a Java servlet application that doesn't have an adapter for that servlet platform, you can opt to use the servlet filter adapter that Keycloak has. This adapter works a little differently than the other adapters. You do not define security constraints in web.xml. Instead you define a filter mapping using the Keycloak servlet filter adapter to secure the url patterns you want to secure.

Warning

Backchannel logout works a bit differently than the standard adapters. Instead of invalidating the http session it instead marks the session id as logged out. There's just no way of arbitrarily invalidating an http session based on a session id.

Warning

Backchannel logout does not currently work when you have a clustered application that uses the SAML filter.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/
xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

 <module-name>customer-portal</module-name>

    <filter>
        <filter-name>Keycloak Filter</filter-name>
           <filter-class>org.keycloak.adapters.saml.servlet.SamlFilter</filter-
class>
    </filter>
    <filter-mapping>
        <filter-name>Keycloak Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

The Keycloak filter has the same configuration parameters available as the other adapters except you must define them as filter init params instead of context params.

To use this filter, include this maven artifact in your WAR poms

```
<dependency>
     <groupId>org.keycloak</groupId>
     <artifactId>keycloak-saml-servlet-filter-adapter</artifactId>
     <version>&project.version;</version>
 </dependency>
```

Chapter 8. Logout

There are multiple ways you can logout from a web application. For Java EE servlet containers, you can call HttpServletRequest.logout(). For any other browser application, you can point the browser at any url of your web application that has a security constraing and pass in a query parameter GLO, i.e. `http://myapp?GLO=true`. This will log you out if you have an SSO session with your browser.

Chapter 9. Obtaining Assertion Attributes

After a successful SAML login, your application code may want to obtain attribute values passed with the SAML assertion. `HttpServletRequest.getUserPrincipal` returns a Principal object that you can typecast into a Keycloak specific class called `org.keycloak.adapters.saml.SamlPrincipal`. This object allows you to look at the raw assertion and also has convenience functions to look up attribute values.

```
package org.keycloak.adapters.saml;

public class SamlPrincipal implements Serializable, Principal {
    /**
     * Get full saml assertion
     *
     * @return
     */
    public AssertionType getAssertion() {
       ...
    }

    /**
     * Get SAML subject sent in assertion
     *
     * @return
     */
    public String getSamlSubject() {
        ...
    }

    /**
     * Subject nameID format
     *
     * @return
     */
    public String getNameIDFormat() {
        ...
    }

    @Override
    public String getName() {
        ...
    }
```

```
    /**
     * Convenience function that gets Attribute value by attribute name
     *
     * @param name
     * @return
     */
    public List<String> getAttributes(String name) {
        ...

    }

    /**
     * Convenience function that gets Attribute value by attribute friendly name
     *
     * @param friendlyName
     * @return
     */
    public List<String> getFriendlyAttributes(String friendlyName) {
        ...
    }

    /**
    * Convenience function that gets first  value of an attribute by attribute name
     *
     * @param name
     * @return
     */
    public String getAttribute(String name) {
        ...
    }

    /**
    * Convenience function that gets first  value of an attribute by attribute name
     *
     *
     * @param friendlyName
     * @return
     */
    public String getFriendlyAttribute(String friendlyName) {
        ...
    }

    /**
     * Get set of all assertion attribute names
     *
     * @return
     */
    public Set<String> getAttributeNames() {
        ...
```

```
    }

    /**
     * Get set of all assertion friendly attribute names
     *
     * @return
     */
    public Set<String> getFriendlyNames() {
        ...
    }
}
```

Chapter 10. Error Handling

Keycloak has some error handling facilities for servlet based client adapters. When an error is encountered in authentication, keycloak will call `HttpServletResponse.sendError()`. You can set up an error-page within your `web.xml` file to handle the error however you want. Keycloak may throw 400, 401, 403, and 500 errors.

```
<error-page>
    <error-code>404</error-code>
    <location>/ErrorHandler</location>
</error-page>
```

Keycloak also sets an `HttpServletRequest` attribute that you can retrieve. The attribute name is `org.keycloak.adapters.spi.AuthenticationError`. Typecast this object to: `org.keycloak.adapters.saml.SamlAuthenticationError`. This class can tell you exactly what happened. If this attribute is not set, then the adapter was not responsible for the error code.

```
public class SamlAuthenticationError implements AuthenticationError {
    public static enum Reason {
        EXTRACTION_FAILURE,
        INVALID_SIGNATURE,
        ERROR_STATUS
    }

    public Reason getReason() {
        return reason;
    }
    public StatusResponseType getStatus() {
        return status;
    }
}
```