Shibboleth Identity Provider

Installation and Setup

covers

Shibboleth v 5.2.1

Jetty 12.0

Apache 2.4

document written by

Pascal PANNEELS
(*mailto:pascal.panneels@belnet.be*)

18/07/2024

Table of content

1. Introduction

This document describes the typical processes used to deploy a federated identity provider based on Shibboleth middleware.

It focuses on main version 5.1.2 of Shibboleth, the latest available one at the time of writing this document.

There are many ways to deploy it, but the one choosen here, is following :

- *Shibboleth IdP* as a Java servlet (WAR file) ;
- servlet access provided by *Jetty 12.0* ;
- the access to Jetty is proxied by *Apache v2.4* ;
- Java's VM is provided by *OpenJDK 17.0.11* 64 bits version ;
- users data is provided by a local *OpenLDAP* server ;
- all software are running on an *Ubuntu 22.04.4* 64 bits server.

Furthermore, a few optional features are also detailed :

- the use of *memcached* to conserve sessions informations in memory ;
- multi-factors authentication using built-in *Cisco DUO* client (working in conjunction of an existing DUO server, for which installation is out of scope of this document) ;
- high-availability feature (in a master/slave setup) using VRRP messages exchanged by *keepalived* daemon.

This document, such as other ones previously published, can serve as an introduction to the essential steps to implement a working SAML2 Shibboleth identity provider base system.

It doesn't replace the official documentation about the various used softwares, but can be seen more as a summary of the absolute essential points to keep in mind when implementing it.

It doesn't cover specific topics such as integration with peculiar products such as Microsoft Active Directory or Microsoft O365, other Java virtual machines or another HTTP server.

I suppose that the reader will be smart enough to adapt the explanations to her/his own configuration.

Also, even if some specific versions of used softwares are mentionned, it doesn't mean that it will not work for versions **onwards** the cited ones. (I could imagine that most of the explanations will still be valid for a server running Ubuntu 24.04, Shibboleth 5.8.5 and Jetty 13... even if there will be unavoidable tweaks to adjust! As I said, you're probably smart enough to adapt to a new situation).

2. *Shibboleth* Installation

2.1. Software download

Point your web browser to *https://shibboleth.net/downloads/identity-provider/latest5/* and download the *tgz*/*tar.gz* package file available (here in the context, it will be *shibboleth-identity-provider-5.1.2.tgz*).

You should then copy the file to your server that will become the IdP:

```
scp  shibboleth-identity-provider-5.1.2.tgz user@youridpdomain:~
```

You may also download it directly from your server with a command such as :

```
wget  https://shibboleth.net/downloads/identity-provider/latest5/shibboleth-identity-provider-5.1.2.tgz
```

2.2. Untar the package

Usual command :

```
tar -xzf shibboleth-identity-provider-5.1.2.tgz
```

2.3. Install the package

Go into the new created directory and launch the install script :

```
cd shibboleth-identity-provider-5.1.2/
./bin/install.sh
```

The script will ask you several informations (if accepting default value, simply press [*Enter]*):

- the *installation directory* (default is */opt/shibboleth-idp*) ; you may leave the default ;
- the *SAML EntityID* for your server (default is *https://yourservername/idp/shibboleth*) ; you may leave the default unless you want/need to specify something else ;
- the *Attribute scope* (by default it is the top level domain of your server); adapt to your own needs, but in principle, it should be alright.

While doing the installation, the script has created several configuration files, either as XML files (**.xml)* or with the *.properties* extension ; we will review some of these files in a later chapter.

It has also created a couple of self signed certificates that are stored in a Java keystore file, as well as in files in the created directory */opt/shibboleth-idp/credentials/*.

3. *Jetty* installation

Jetty is used to serve the *Shibboleth* Java application itself ; it is the file */opt/shibboleth/war/idp.war*.

It works in conjunction of other XML/configuration/extension files present in the Shibboleth directory.

Previously, *Shibboleth* supported other servlet servers such as *Apache Tomcat*, but due to bugs and performance issues, it is not supported anymore (since *Shibboleth* v4 already) by the *Shibboleth* developpers ; *Jetty* is the recommended server to use, but if you have another *servlets* application of choice, you may try to set it up for *Shibboleth*, but it is out of scope of this document.

3.1. Download *Jetty*

Point your web browser to *https://jetty.org/download.html* and download the latest tgz/tar.gz package available (here in the example it will be *jetty-home-12.0.10.tar.gz*).

You should then copy the file to your server that will become the IdP:

```
scp jetty-home-12.0.10.tar.gz user@youridpdomain:~
```

You may also download it directly from your server with a command such as :

```
wget  https://repo1.maven.org/maven2/org/eclipse/jetty/jetty-home/12.0.10/jetty-home-12.0.10.tar.gz
```

3.2. Untar the package

Go to a directory that should contain the Jetty source install, such as /usr/local/src/, and use the usual untar command :

```
cd /usr/local/src/

tar -xzf jetty-home-12.0.10.tar.gz
```

To simplify the reference to the installation directory, create a symbolic link to it :

```
ln -s jetty-home-12.0.10 jetty-src
```

According to *Jetty*'s documentation, it is better to split Jetty in 2 separates directory, one containing the *home* source (it is the directory we have just created) and one for the *base* application (that we will create further). Environnement variables reflecting the setup should be created as *Jetty* will need it as well.

I propose to create a file in /etc/default/jetty to contain these variables :

```
JETTY_HOME=/usr/local/src/jetty-src
JETTY_BASE=/opt/jetty-shibboleth
JETTY_USER=jetty
```

```
JETTY_START_LOG=/var/log/jetty/start.log
TMPDIR=$JETTY_BASE/tmp
JETTY_ARGS="jetty.ssl.port=8443"
```

Read the variables to reflect their existence in your current shell :

```
source /etc/default/jetty
```

3.3. Get the *Shibboleth* base configuration for *Jetty*

Create the directory that will contain the base for your application :

```
mkdir $JETTY_BASE
```

Type in the following commands in your directory of choice (**not in** $JETTY_BASE):

```
git clone https://git.shibboleth.net/git/java-idp-jetty-base

cd java-idp-jetty-base

git checkout 12

cp -r src/main/resources/jetty-base/* $JETTY_BASE
```

3.4. Configure *Jetty*

3.4.1. $JETTY_BASE/start.d/idp.ini

You should open $JETTY_BASE/start.d/idp.ini with a text editor (such as *vim*) and adjust the lines as described below.

Add following setting lines at the top of the file :

```
--exec
-Xmx1500m
-Djava.security.egd=file:/dev/urandom
-Djava.io.tmpdir=tmp
-Dlogback.configurationFile=resources/logback.xml
```

You should modify the password used to access the Java keyfile that holds the access to the certificates used by *Jetty* (it is the lines with *.keyXXXPassword* in it with the password *changeit*); see further for the setup of the keyfile.

You should comment the lines with *jetty.ssl.host* and *jetty.ssl.port*.

Finally add following lines :

```
jetty.http.host=127.0.0.1
jetty.http.port=8080
```

These settings will be used further on the *Apache2* setup.

3.4.2. $JETTY_BASE/webapps/idp.xml

You should verify that the content corresponds to your *Shibboleth* installation, especially the path to *Shibboleth* install

The content should looks to something like :

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Jetty//Configure//EN"
"https://www.eclipse.org/jetty/configure_10_0.dtd">
<!-- ========================================================== -->
<!-- Configure the Shibboleth IdP webapp                        -->
<!-- ========================================================== -->
<Configure class="org.eclipse.jetty.ee9.webapp.WebAppContext">
  <Set name="war">/opt/shibboleth-idp/war/idp.war</Set>
  <Set name="contextPath">/idp</Set>
  <Set name="extractWAR">false</Set>
  <Set name="copyWebDir">false</Set>
  <Set name="copyWebInf">true</Set>
</Configure>
```

3.4.3. Install the *logging* module

In the beginning of this chapter, in the file *idp.ini*, there is a reference to the *logging-logback* module.

You need to be sure that the module is installed :

```
java -jar /opt/jetty-src/start.jar --add-module=logging-logback
```

3.4.4. Check the loaded modules in $JETTY_BASE/modules/idp.mod

The section *[depend]* should looks to something like :

```
[depend]
ee9-annotations
ee9-deploy
ext
ee9-webapp
http
ee9-jsp
ee9-jstl
ee9-plus
resources
ee9-servlets
```

As the service is proxied by *Apache*, there is no need to enable *https/ssl* here, as it will be handled by *Apache*.

It is also possible to make *Jetty* run directly (thus without be *guarded* by a proxy), but it is not a choice I've made here. If you want to make it run differently, adapt the setup according to your own choices (it will probably require that you install additional *Jetty* modules using the same kind of command used to install the *logging* module).

3.4.5. Create a *systemd service file* launcher for *Jetty*

Depending on the version of your OS, you should adapt to your own configuration, here is an example to launch it on *Ubuntu 22.04* ; create the file */lib/systemd/system/jetty.service* containing following lines *:*

```
[Unit]
Description=Jetty servlet for Shibboleth
After=network.target auditd.service

[Service]
EnvironmentFile=-/etc/default/jetty
ExecStart=java -jar /usr/local/src/jetty-src/start.jar jetty.home=/usr/local/src/jetty-src
jetty.base=/opt/jetty-shibboleth
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=simple

[Install]
WantedBy=multi-user.target
Alias=jetty.service
```

Enable the service to start at boot :

```
systemctl enable jetty.service
```

3.4.6. Change Jetty's files ownership

If not already done, create a system user for *Jetty* :

```
useradd -r -M jetty
```

Then, change the ownership of all configuration files used by *Jetty* :

```
chown -R jetty: /opt/jetty-shibboleth/
```

You should now be able to start *Jetty* :

```
systemctl start jetty.service
```

4. *Apache* configuration

4.1. Prerequisites

We suppose you've a fully qualified registered domain name for your server and that you've installed a public certificate to let your server be authentified .

4.2. Create a configuration file for the site

Create a configuration for your site that could looks to following (stored in */etc/apache2/sites-available/idp.conf*) :

```
<VirtualHost *:443>
        ServerName idp.yourdomain
        ServerAdmin webmaster@yourdomain
        CustomLog /var/log/apache2/idp.access.log combined
        ErrorLog /var/log/apache2/idp.error.log

        DocumentRoot /var/www
        <Directory />
            Options FollowSymLinks
            AllowOverride None
        </Directory>
        <Directory /var/www/>
            Options FollowSymLinks MultiViews
            AllowOverride None
            Order allow,deny
            allow from all
        </Directory>

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel debug

        SSLEngine On
        SSLProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
        SSLCipherSuite ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA- CHACHA20-
POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-
SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-
AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-
```

```
ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-
GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-
SHA:DES-CBC3-SHA:!DSS
        SSLHonorCipherOrder on
        SSLCompression off
        SSLOptions +StrictRequire

        Header set Access-Control-Allow-Origin "*"

        ProxyPreserveHost On
        RequestHeader set X-Forwarded-Proto "https"
        RequestHeader set X-Forwarded-Port 443
        <Proxy "*">
                Order allow,deny
                Allow from all
        </Proxy>
        ProxyPass /idp http://localhost:8080/idp retry=5
        ProxyPassReverse /idp http://localhost:8080/idp retry=5

        <Location /idp>
          Require all granted
        </Location>

SSLCertificateFile /etc/letsencrypt/live/yourserver/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/yourserver/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
```

You may now activate the *Apache* site :

```
a2ensite idp
```

You should also enable some necessary *Apache* modules :

```
a2enmod ssl, proxy, proxy_http, headers, rewrite,
```

Then, don't forget to restart your *Apache* server :

```
systemctl restart apache2.service
```

5. *Shibboleth* configuration

5.1. Setup the source for users database

As mentionned in the introduction, the setup is based on an *OpenLDAP* database to store the users, their passwords and all other attributes.

In following paragraphs, when referring to a file, it will be based on */opt/shibboleth/* directory, unless explicitely referred otherwise.

5.1.1. conf/ldap.properties

The file contains the essential properties that describe the *OpenLDAP* server to use.

The important lines are the following (I think that property's name is descriptive enough to let you know how to adjust it) :

```
idp.authn.LDAP.ldapURL                    = ldap://localhost:389
idp.authn.LDAP.useStartTLS                = false
idp.authn.LDAP.returnAttributes           = uid,cn,mail
idp.authn.LDAP.baseDN                     = dc=yourdomain,dc=be
idp.authn.LDAP.subtreeSearch              = true
idp.authn.LDAP.userFilter                 = (uid={user})
idp.authn.LDAP.bindDN                     = cn=superadmin,dc=yourdomain,dc=be
idp.authn.LDAP.bindDNCredential           = SuperPassword
idp.authn.LDAP.dnFormat                   = uid=%s,dc=yourdomain,dc=be
```

5.1.2. conf/attribute-resolver.xml

In this file, we will create a description of a data connector object used by other Java objects to interact with the described *OpenLDAP* server; it will use the properties defined earlier.

It contains also, the definition of the attributes used by *Shibboleth*, generaly in term of a mapping with *LDAP*'s attributes, referred by the *LDAP*'s data connector's ID.

You should create a data connector as :

```
  <!-- LDAP  attributes definitions -->
  <DataConnector id="myLDAP" xsi:type="LDAPDirectory"
    ldapURL="%{idp.attribute.resolver.LDAP.ldapURL}"
    baseDN="%{idp.attribute.resolver.LDAP.baseDN}"
    principal="%{idp.attribute.resolver.LDAP.bindDN}"
    principalCredential="%{idp.attribute.resolver.LDAP.bindDNCredential}"
    useStartTLS="%{idp.attribute.resolver.LDAP.useStartTLS:false}">
    <FilterTemplate>
      <![CDATA[
        %{idp.attribute.resolver.LDAP.searchFilter}
      ]]>
    </FilterTemplate>
```

```
</DataConnector>
```

5.2. User attributes

5.2.1. conf/attribute-resolver.xml

As previsouly said, the file contains the definition of user attributes, having their names used by *Shibboleth* on one side, and the corresponding names in the used data source.

Each entry looks like following example :

```
<AttributeDefinition id="commonName" xsi:type="Simple" >
   <InputDataConnector ref="myLDAP" attributeNames="cn" />
      <AttributeEncoder xsi:type="SAML2String"  name="urn:oid:2.5.4.3"
       friendlyName="cn" />
</AttributeDefinition>
```

An entry has a unique identifier, a type, a reference to the data source with its corresponding attribute name in the source, and also an encoder that, more or less, specify its type and with which name it is exported to outside.

These attributes are *dynamic ones* as they come from a data source, and usually change for most users in the database.

5.2.2. Static attributes

Sometimes, it is useful to create attributes that are common for all the users, and that are not even present in the source database itself.

It is called *static attributes* and can be created with following kind of statements :

```
<DataConnector id="staticAttributes" xsi:type="Static">
   <Attribute id="affiliation">
      <Value>member</Value>
   </Attribute>
   <Attribute id="schacHomeOrganization">
      <Value>yourdomain.be</Value>
   </Attribute>
   <Attribute id="schacHomeOrg">
      <Value>yourdomain.be</Value>
   </Attribute>
   <Attribute id="schacHomeOrganizationType">
      <Value>urn:schac:homeOrganizationType:be:university</Value>
   </Attribute>
</DataConnector>
```

Here, we have defined 4 static attributes, that can all be exported when a user logs in, along with his specific personal attributes.

5.3. conf/attribute-filter.xml

In this file, we defined which attributes can be given to specific service.

It consists of rules to match, for example, a *SP's entityID* and a set of rules for releasing attributes to.

```
<AttributeFilterPolicy id="specific-sp">
      <PolicyRequirementRule xsi:type="OR">
        <Rule xsi:type="Requester" value="https://sptest.belnet.be/shibboleth" />
        <Rule xsi:type="Requester" value="https://filesender.belnet.be"/>
      </PolicyRequirementRule>
      <AttributeRule attributeID="eduPersonPrincipalName">
        <PermitValueRule xsi:type="ANY" />
      </AttributeRule>
      <AttributeRule attributeID="mail">
        <PermitValueRule xsi:type="ANY" />
      </AttributeRule>
      <AttributeRule attributeID="commonName">
        <PermitValueRule xsi:type="ANY" />
      </AttributeRule>
  </AttributeFilterPolicy>
```

In the given example, it means that if the requester' SP with an entityID's value of either *https://sptest.belnet.be/shibboleth* or *https://filesender.belnet.be*, it will release the attributes *eduPersonPrincipalName*, *mail* and *commonName*.

You may create as many *AttributeFilterPolicy* as you want, as long as they have each a distinct *id*.

There is also a way to create a default one, if no condition is met, following rule will catch by default :

```
  <AttributeFilterPolicy id="alwaysRelease">
     <PolicyRequirementRule xsi:type="ANY" />
     <AttributeRule attributeID="schacHomeOrganization" permitAny="true" />
     <AttributeRule attributeID="eduPersonPrincipalName" permitAny="true" />
  </AttributeFilterPolicy>
```

It will only release *schacHomeOrganization* and *eduPersonPrincipalName* attributes by default.

5.4. conf/metadata-providers.xml

The file defines the providers of metadata in the federation.

Usually, it should contain at least one source coming from the federation your are belonging to, thus *Belnet federation*.

```
  <MetadataProvider
    id="BelnetFederationMD"
    xsi:type="FileBackedHTTPMetadataProvider"
    metadataURL="https://federation.belnet.be/metadata/belnet-edugain-merged.xml"
    backingFile="%{idp.home}/metadata/metadata.belnet-federation.xml"
    maxRefreshDelay="PT2H" >

  <MetadataFilter xsi:type="SignatureValidation" requireSignedRoot="true" certificateFile="%
{idp.home}/credentials/federation.belnet.be.pem"/>
    <MetadataFilter xsi:type="RequiredValidUntil" maxValidityInterval="P9D" />
  </MetadataProvider>
```

There are 2 important parts :

- *MetadataProvider* that includes the URL where to retrieve the metadata from, and where to store it locally ;
- *MetadataFilter* that defines how to verify the validity of the signed metadata, and for how long it is valid.

You may define several sources of metadata as long as there are referenced by an unique *id*.

From the given example, you should not forget to download and install the certificate used to sign the federation's metadata.

It can be downloaded with a command such as :

```
wget https://federation.belnet.be/metadata/federation.belnet.be.crt
```

And it should be copied in directory *credentials/*.

5.5. conf/logback.xml

In this file, you will be able to adjust the logging information, that is useful for debugging when something goes wrong or not as expected.

If you look in it, you will see several sections, but you should probably limit your modifications to what's below the comment line :

```
<!-- Logging level shortcuts. -->
```

You may change the tag *INFO* by *WARN*, *ERROR* or *DEBUG*. It will increase the verbosity in the logging files */opt/shibboleth-idp/logs/idp-process.log* and */opt/shibboleth-idp/logs/idp-warn.log*.

What I usually do, is to create 2 copies of the logback.xml file, one with *INFO* and another one with *DEBUG* level; I let the *INFO* one as the original and increase the verbosity levels in the *DEBUG* version.

I then create a *Un*x* symbolic link to the one I want to use, with the name of expected XML file such as for example :

```
ln -sf conf/logback.xml.INFO logback.xml
```

5.6. Adjusting the look of the interface

You will certainly want to modify the look of the presented interface.

To proceed, you should bring your modifications, add your logos, etc. in the directories :

- /opt/shibboleth-idp/views/
- /opt/shibboleth-idp/edit-webapp/

The later one should contains the new images, logos, CSS... you want to add or modify.

The former one contains the files written as *Velocity templates* (*.vm* files) that contain the interface's elements itself.

I suggest to refer to an official documentation explaining how to use it, as it is far too complicated to be included in the present document.

When you have finished to add or modify the interfaces files, you should rebuild the servlet WAR file itself.

To do that, simply run following command :

```
/opt/shibboleth-idp/bin/build.sh
```

5.7 Adjust ownership of the files

Don't forget to change the ownership of the generated files/directories to jetty's user :

```
chown -R jetty:jetty /opt/shibboleth-idp/*
```

Otherwise, the system will not start and it will complain about access' permissions problems or other weird errors, sometimes hard to figure their exact meaning.