

Configuring a Shibboleth IdP

Getting Started

The first thing that you need is VMWare so that you can run the guest VM we have provided. A decent amount of work has already been done to get this image set up: CentOS 5.2 installation, ntpd configuration – synchronized time is very important, Apache Tomcat has been downloaded/installed/partially configured, and the Java environment has been set up. We've already gone through and disabled most of the unneeded services so you should only need 256MB of RAM, although you could likely get away with using 128MB, to run this test VM smoothly.

Your IdP will have to have a valid DNS A record and be reachable on port 443 and/or 80. We have already configured Apache to proxy requests to “/idp” to Tomcat. Make sure you make any necessary changes to /etc/resolv.conf because you will need to be able to do hostname lookups.

The root password for the machine is: “password”. It is highly recommended that you change this password immediately.

We recommend checking out the official Shibboleth IdP wiki, <https://spaces.internet2.edu/display/SHIB2/IdPInstall>.

Java Environment

The Java JRE has already been installed on your virtual machine and JAVA_HOME has already been set. If running “echo \$JAVA_HOME” prints out a blank line then there is a problem. You should manually set the variable and set up the system-wide profile to set it by default:

```
export JAVA_HOME=/usr/java/latest
echo “export JAVA_HOME=/usr/java/latest” > /etc/profile.d/java.sh
```

Install Shibboleth Identity Provider

Install the Shibboleth Identity Provider package that we have already downloaded. The installer will prompt you for where to install the IdP home directory as well as your FQDN. Make sure you have a FQDN before this step. The defaults should be fine for both questions. Finally it will prompt you for a password, remember this password as it will be needed again later when configuring Tomcat.

```
cd /opt/shibboleth-identityprovider-2.1.2/
sh install.sh
ln -s /opt/shibboleth-idp /usr/local/idp
```

For future reference, IDP_HOME=/usr/local/idp/ & IDP_SRC=/opt/shibboleth-identityprovider-2.1.2/.

Prepare Apache Tomcat 6

The Shibboleth documentation recommends Apache Tomcat as the "popular" choice in a Java servlet so that's what we used. For future reference, CATALINA_HOME=/usr/local/tomcat/. We have already downloaded and installed Apache Tomcat for you.

Help Tomcat run the IdP by "endorsing" additional libraries (IDP_SRC/endorsed/*.jar) for XML processing and deploying the IdP's WAR file (IDP_HOME/war/idp.war) to CATALINA_HOME/webapps:

```
mkdir CATALINA_HOME/endorsed
cp IDP_SRC/endorsed/*.jar CATALINA_HOME/endorsed
cp IDP_HOME/war/idp.war CATALINA_HOME/webapps
```

We have found and slightly modified a Tomcat init script, it can be found in /etc/init.d. You should modify the JAVA_OPTS line before starting Tomcat for the first time. "-Xmx128m" sets the maximum amount of memory that Tomcat can use. The Shibboleth Tomcat page recommends at least 512MB, however we are running this on a virtual machine with very little memory so providing 128MB isn't unreasonable. "-XX:MaxPermSize=64m" sets the maximum amount of memory allowed for the permanent generation object space. The Shibboleth Tomcat page recommends half the maximum memory or 512MB, whichever is smaller.

Don't try to start Tomcat just yet! We will need to modify some permissions so Tomcat can write log and work files, we have already created the "tomcat" user on your virtual machine:

```
chown -R tomcat CATALINA_HOME/ IDP_HOME/
```

These are broad strokes as far as setting permissions go. It's possible that these permissions can be refined to only a few sub-directories however there's no obvious security risk, at least while testing, for setting the permissions as described.

Install Shibboleth Security Provider

To install the Shibboleth Security Provider do the following:

```
cp IDP_SRC/lib/shibboleth-jce-1.0.0.jar JAVA_HOME/lib/ext/
```

If the ext/ directory doesn't exist in the path indicated above, create it. Now we just need to edit the java.security file in JAVA_HOME/lib/security. Add the following line after the last security.provider entry in that file,

```
security.provider.#=edu.internet2.middleware.shibboleth.DelegateToApplicationProvider.
```

Where the # is replaced with the a number one more than the last provider in the list. For example:

```
security.provider.1=sun.security.provider.Sun
...
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=edu.internet2.middleware.shibboleth.DelegateToApplicationProvider
```

Configuring Tomcat

We have already defined the following Connector container in Tomcat's CATALINA_HOME/conf/server.xml. You will need to modify the keystoreFile, keystorePass, truststoreFile, and truststorePass entries:

```
<Connector port="8443"
  maxHttpHeaderSize="8192"
  maxSpareThreads="75"
  scheme="https"
  secure="true"
  clientAuth="want"
  SSLEnabled="true"
  sslProtocol="TLS"
  keystoreFile="IDP_HOME/credentials/idp.jks"
  keystorePass="PASSWORD"
  truststoreFile="IDP_HOME/credentials/idp.jks"
  truststorePass="PASSWORD"
  truststoreAlgorithm="DelegateToApplication"/>
```

The password above should be the same plaintext password that was used when installing the IdP. If you forgot this password, you can run the IdP installer again using the same path and FQDN you specified previously to update the credentials, creating Java keystores by hand is outside the scope of this document.

Normally you deploy web applications to Tomcat by copying the WAR file into CATALINA_HOME/webapps/. However, when you do this Tomcat will explode the war, giving you idp.war and idp/ within webapps/, and then cache another version of the application in CATALINA_HOME/work/Catalina/localhost/. This can lead to cases where you copy a new WAR into place but Tomcat continues to use the older, cached version.

To address this, it is recommended that you use a context deployment fragment, which is just a small bit of XML that tells Tomcat where to get the WAR and provides some properties used when Tomcat load the application. This approach will also prevent those times when you create a new WAR and forget to copy it into Tomcat as this makes Tomcat load the war from IDP_HOME/war/ where the installer places it. Create the file, and directory path if needed, CATALINA_HOME/conf/Catalina/localhost/idp.xml and place the following content in it:

```
<Context docBase="IDP_HOME/war/idp.war"
  privileged="true"
  antiResourceLocking="false"
  antiJARLocking="false"
  unpackWAR="false" />
```

There won't be a lot of time spent redeploying the WAR file, especially during this test. However this may be useful down the road when you need to upgrade and it removes the need to copy the WAR file into CATALINA_HOME/webapps/ every time, although it is still a good practice to do so.

You can now start Tomcat if you haven't already:

/etc/init.d/tomcat start

Testing with TestShib.org

First you need to register your IdP with TestShib. You can do so by going here, <http://www.testshib.org/testshib-two/login.do>, and creating a ProtectNetwork or OpenID login (if it matters, we used ProtectNetwork) if you haven't already. Once you're logged in you should navigate to <https://www.testshib.org/testshib-two/menu.do>. You are registering a new IdP so click that link and you will be presented with a form. Enter in the domain name, the FQDN (should match what you entered during the IdP installation), the SSL cert (IDP_HOME/credentials/idp.crt) and your first and last name. we unchecked the checkbox that says "Check domain name validity". Click continue and your IdP should now be registered with TestShib.

Configuring IdP for TestShib.org

Now onto configuring your IdP to communicate with the TestShib Service Provider (TestShib Two). The following files need to be modified, and conveniently enough they are all located in IDP_HOME/conf.

relying-party.xml:

Configure metadata sources in this file. Place the following container inside <MetadataProvider id="ShibbolethMetadata" /> after the block where the IdP's own metadata is loaded:

```
<MetadataProvider id="URLMD" xsi:type="FileBackedHTTPMetadataProvider"
  xmlns="urn:mace:shibboleth:2.0:metadata"
  metadataURL="http://www.testshib.org/metadata/testshib-providers.xml"
  backingFile="/usr/local/idp/metadata/testshib.xml" />
```

The "metadataURL" is pretty self explanatory, it points to the SP's metadata providers file. The "backingFile" is just a location to save a local copy incase the "metadataURL" is unreachable.

In the future you would want to use a <MetadataFilter /> inside your MetadataProvider container for increased security. There is an example how you would do this underneath the comment "Example metadata provider.". Since this is just a test and TestShib is known to be untrustworthy, it's OK not to use a filter in this context.

attribute-resolver.xml:

Attributes are defined in the attribute-resolver, also the DataConnector is configured in this file so that the IdP knows where to pull attributes from.

Trying this out with static attributes first is probably a good idea. It requires very little to get it up and running so we'll go through the steps for doing this first.

First, uncomment the "Example Static Connector".

```

<resolver:DataConnector id="staticAttributes" xsi:type="Static"
xmlns="urn:mace:shibboleth:2.0:resolver:dc">
  <Attribute id="eduPersonAffiliation">
    <Value>member</Value>
    <Value>staff</Value>
    <Value>faculty</Value>
  </Attribute>
  <Attribute id="eduPersonEntitlement">
    <Value>urn:example.org:entitlement:Setting up a Shibboleth IdP.</Value>
    <Value>urn:mace:dir:entitlement:common-lib-terms</Value>
  </Attribute>
</resolver:DataConnector>

```

As you can see we modified the entitlement slightly just so we can be certain our attributes were passed. Using a Static connector would be a good idea if you want to pass an attribute for every single person that authenticates through your IdP.

Next, uncomment the first two eduPerson attribute definitions. You can uncomment more if you like but we chose to just do the first two: "eduPersonAffiliation" and "eduPersonEntitlement":

```

<resolver:AttributeDefinition id="eduPersonAffiliation" xsi:type="Simple"
xmlns="urn:mace:shibboleth:2.0:resolver:ad"
sourceAttributeID="eduPersonAffiliation">
  <!--<resolver:Dependency ref="myLDAP" />-->
  <resolver:Dependency ref="staticAttributes" />

  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
name="urn:mace:dir:attribute-def:eduPersonAffiliation" />

  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" friendlyName="eduPersonAffiliation" />
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="eduPersonEntitlement" xsi:type="Simple"
xmlns="urn:mace:shibboleth:2.0:resolver:ad"
sourceAttributeID="eduPersonEntitlement">
  <!--<resolver:Dependency ref="myLDAP" />-->
  <resolver:Dependency ref="staticAttributes" />

  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
name="urn:mace:dir:attribute-def:eduPersonEntitlement" />

  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" friendlyName="eduPersonEntitlement" />

```

```
</resolver:AttributeDefinition>
```

Add a `<resolver:Dependency ref="staticAttributes" />` to the `eduPersonEntitlement` attribute definition, just like the `eduPersonAffiliation` example above it and comment out the `resolver:Dependency` for `"myLDAP"` if one exists.

attribute-filter.xml:

Specify what attributes will be passed to the Service Provider in this file.

Here we can just modify the "portal service provider" example. Uncomment the last `<AttributeFilterPolicy>`. Change the `<PolicyRequirementRule>` value of `"urn:example.org:sp:myPortal"` to `"https://sp.testshib.org/shibboleth-sp"`. Change the `<AttributeRule>` `attributeID` to `"eduPersonAffiliation"` and create a second, identical `<AttributeRule>` with an `attributeID` of `"eduPersonEntitlement"`:

```
<AttributeFilterPolicy>
  <PolicyRequirementRule xsi:type="basic:AttributeRequesterString"
value="https://sp.testshib.org/shibboleth-sp" />
  <!--PolicyRequirementRule xsi:type="basic:AttributeRequesterString"
value="urn:example.org:sp:myPortal" /-->

  <AttributeRule attributeID="eduPersonAffiliation">
    <PermitValueRule xsi:type="basic:ANY" />
  </AttributeRule>
  <AttributeRule attributeID="eduPersonEntitlement">
    <PermitValueRule xsi:type="basic:ANY" />
  </AttributeRule>
</AttributeFilterPolicy>
```

Configure Httpd Authentication

Using Apache's `htpasswd` utility we can set up some basic authentication to test the installation.

```
htpasswd -c IDP_HOME/credentials/user.db testuser
```

Next we need to add the following lines to the end of `/etc/httpd/conf.d/ssl.conf`:

```
<Location /idp/Authn/RemoteUser>
  AuthType Basic
  AuthName "My Identity Provider"
  AuthUserFile IDP_HOME/credentials/user.db
  require valid-user
</Location>
```

Test It!

Make sure you have restarted Tomcat and Apache before you test out your installation. Point your browser at, <https://sp.testshib.org/>, enter in your FQDN where it says "your.host.here", and click "Go!". You will be prompted by an htaccess style login box, authenticate, and you should be redirected to a page that contains some of your Shibboleth session information and any attributes that you configured to pass along.

LDAP Configuration

If you already have an LDAP server and would like to see how it could potentially tie into Shibboleth then this section is for you!

For Shibboleth, you will need to obtain the latest eduPerson schema (LDIF format) from internet2.edu. It can be found here, <https://spaces.internet2.edu/display/macedir/LDIFs>. The LDIFs are platform specific so make sure you download the right LDIF for your LDAP platform.

For example, when using OpenLDAP:

```
wget http://middleware.internet2.edu/dir/schema/ldifs/OpenLDAP\_eduPerson-200412.tar.gz
tar -zxf OpenLDAP_eduPerson-200412.tar.gz
cp eduperson-200412.ldif /etc/openldap/schema/eduperson.ldif
```

Afterwards add the following line to the slapd.conf file near where the other schemas are included:

```
include /etc/openldap/schema/eduperson.schema
```

Now, restart the LDAP server:

```
/etc/init.d/ldap restart
```

Make sure that you now add the objectClass eduPerson to whatever user you plan on authenticating with so that you can add eduPerson related attributes to their LDAP entry and pass them to TestShib. Not knowing how you manage your LDAP we'll leave this as an exercise for the reader. You should now add the "eduPersonAffiliation" attribute to the user account you plan on testing with and add a couple of values: "member", "student" and "faculty" should suffice.

Now we need to go back and edit our attribute-resolver.xml so that it uses the LDAP to pull the the attributes as opposed to using static ones. You might as well comment out the `<resolver:DataConnector />` for staticAttributes that we set up earlier. Now we can set up a `<resolver:DataConnector />` for the LDAP using the example that's already in the file:

```
<resolver:DataConnector id="myLDAP" xsi:type="LDAPDirectory"
xmlns="urn:mace:shibboleth:2.0:resolver:dc"
  ldapURL="ldap://ldapservers.example.com" baseDN="ou=People,dc=example,dc=com"
  principal="cn=root,dc=example,dc=com"
  principalCredential="cleartext_rootdn_password">
  <FilterTemplate>
    <![CDATA[
      (uid=$requestContext.principalName)
    ]]>
```

```
</FilterTemplate>
</resolver:DataConnector>
```

"id" field is arbitrary, use it to describe your directory.

"ldapURL" is the FQDN of your LDAP server.

"baseDN" is the base dn used for searching.

"principal" is the rootdn defined in /etc/openldap/slapd.conf.

"principalCredential" is the cleartext version of rootpw that was defined in /etc/openldap/slapd.conf.

Note: We were able to use a read-only account in place of the rootdn when configuring the principal. This has an obvious potential security bonus since the password is in cleartext.

Set up AttributeDefinitions for attributes you want to pass, including how to get them from the LDAP. Note these are almost identical to the AttributeDefinitions from the static attributes, except that we have removed the resolver:Dependency for "staticAttribute" and uncommented the one for "myLDAP", which we already defined above:

```
<resolver:AttributeDefinition id="eduPersonAffiliation" xsi:type="Simple"
xmlns="urn:mace:shibboleth:2.0:resolver:ad"
  sourceAttributeID="eduPersonAffiliation">
  <resolver:Dependency ref="myLDAP" />

  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
  name="urn:mace:dir:attribute-def:eduPersonAffiliation" />

  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
  name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" friendlyName="eduPersonAffiliation" />
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="eduPersonEntitlement" xsi:type="Simple"
xmlns="urn:mace:shibboleth:2.0:resolver:ad"
  sourceAttributeID="eduPersonEntitlement">
  <resolver:Dependency ref="myLDAP" />

  <resolver:AttributeEncoder xsi:type="SAML1String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
  name="urn:mace:dir:attribute-def:eduPersonEntitlement" />

  <resolver:AttributeEncoder xsi:type="SAML2String"
xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
  name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" friendlyName="eduPersonEntitlement" />
</resolver:AttributeDefinition>
```

Feel free to pass any additional attributes you would like, there are examples for many of the attributes that may already be in use in your LDAP.

Now we will need to reconfigure the htaccess we used before to use the LDAP for authentication. If you are using a self-signed certificate then modify or add the following lines in

/etc/httpd/conf.d/ssl.conf:

```
LDAPVerifyServerCert off
<Location /idp/Authn/RemoteUser>
  AuthType Basic
  AuthName "Secure Area"
  AuthBasicProvider ldap
  AuthzLDAPAuthoritative Off
  AuthLDAPURL "ldap://your.ldap.server.here:389/ou=People,dc=example,dc=com?uid?sub?
(objectClass=*)" TLS
  Require valid-user
</Location>
```

We'll also need to copy security certificate for the LDAP over and configure /etc/openldap/ldap.conf:

```
BASE dc=example, dc=com
URI ldap://ldap.server.here
TLS_CERT /etc/pki/tls/certs/slapd.pem
TLS_CACERT /etc/pki/tls/certs/ca-bundle.crt
TLS_REQCERT allow
```

Make sure that the permissions on slapd.pem are 640 and the ownership is root:apache. Now, Apache will be able to encrypt/decrypt messages to and from the LDAP server.

If you are using a CA signed certificate then the configuration will be slightly different. In /etc/httpd/conf.d/ssl.conf:

```
LDAPVerifyServerCert on
LDAPTrustedGlobalCert CA_BASE64 /etc/pki/tls/certs/CA.crt
<Location /idp/Authn/RemoteUser>
  AuthType Basic
  AuthName "Test Identity Provider"
  AuthBasicProvider ldap
  AuthLDAPBindDN "uid=read-only-user,ou=Users,ou=System,dc=example,dc=com"
  AuthLDAPBindPassword cleartext-password
  AuthzLDAPAuthoritative Off
  AuthLDAPURL "ldap://ldap-server.example.com:389/dc=example,dc=com?uid?sub?
(objectClass=*)" TLS
  Require valid-user
</Location>
```

The configuration above will largely depend on your LDAP installation/configuration. If you need to, refer to Apache's mod_authnz_ldap documentation page found here:

http://httpd.apache.org/docs/2.2/mod/mod_authnz_ldap.html

Lastly, we'll need to configure /etc/openldap/ldap.conf to use our CA signed certificate and to force verification:

```
BASE dc=example, dc=com
```

```
URI ldap://ldap-server.example.com
TLS_CERT /etc/pki/tls/certs/ldap-server.example.com.crt
TLS_KEY /etc/pki/tls/private/ldap-server.example.com.key
TLS_CACERT /etc/pki/tls/certs/CA.crt
TLS_CACERTDIR /etc/pki/tls/certs/
TLS_REQCERT demand
```

Now restart Apache and Tomcat and try testing with TestShib again. Hopefully you will see your new attributes when you authenticate through to TestShib. If not, try checking the logs for Apache/Tomcat/IdP to see what might be the problem.