

CESNET Technical Report 12/2008

Using LDAP with Shibboleth Identity Provider 2.x

IVAN NOVAKOV

Received 1.12.2008

Abstract

The article covers the most common aspects about using LDAP directory service with Shibboleth Identity Provider. It is mostly technically oriented providing configuration howto with practical examples. It explains briefly the background of each of the used tools and technologies and provides references for further study. Basic knowledge of Shibboleth and LDAP is required.

Keywords: shibboleth, identity provider, ldap, federation, SAML

1 Introduction

Shibboleth Identity Provider handles two important tasks – provides user authentication and releases user attributes. Since the identity provider itself does not contain any user database nor attribute storage, it uses external authentication authority and retrieves user attributes from an external database.

Many organizations have their identity management based on LDAP directory services [1]. The directory structure is usually very suitable for storing user data, which can be accessed through standard protocol. The same protocol may be also used for standard-compliant user authentication, supported by numerous applications.

These qualities of the directory services make them extremely suitable for use with the Shibboleth Identity Provider, both as an authentication authority and attribute storage. Additionally, in case the LDAP has been deployed at the organization before (which is very likely), the identity provider will be then naturally connected to the existing identity management infrastructure.

Although user authentication and attribute retrieval is often performed at the same time, these are two different operations, which are configured separately. That means, if we want to interconnect our identity provider with our directory, we need to configure it twice – once as an authentication handler and once as a data storage.

2 LDAP as an authentication authority

Shibboleth Identity Provider has a modular authentication architecture. Different authentication back-ends may be used as login handlers. Some login handlers are shipped with Shibboleth, but if there is need for some custom functionality, it is possible to write your own custom login handler.

LDAP authentication can be performed by the standard *UsernamePassword* login handler. It uses the *Java Authentication and Authorization Service* (JAAS) [2],

which is also modular and permits plugging different authentication modules while the application itself remains unmodified.

2.1 JAAS configuration file

The location of the JAAS configuration file may be anywhere on the system, but its default location is *conf/login.conf* (configured at the *UsernamePassword* login handler, see below). Its structure is relatively simple:

```
<name used by application to refer to this entry> {
    <LoginModule> <flag> <LoginModule options>;
    <optional additional LoginModules, flags and options>;
};
```

Shibboleth provides an LDAP authentication module for JAAS – *edu.vt.middleware.ldap.jaas.LdapLoginModule* [3], which supports the following options [4]:

- **base** - Search base for the LDAP server
- **host** - Hostname(s) of the LDAP server. This is a space separated list where each host is tried in turn until either one is found that accepts the connection or no connection is made.
- **port** - Port of the LDAP server, AD users should use port 3268 if connecting to their global catalog
- **serviceCredential** - Password for the privileged user
- **serviceUser** - DN of privileged user used to connect to the LDAP server
- **ssl** - Whether to use SSL when connecting to the server, acceptable values are *true* or *false*, default value is *false*
- **tls** - Whether to use startTLS when connecting to the server, acceptable values are *true* or *false*, default value is *false*
- **userField** - LDAP attribute containing the username of the user
- **subtreeSearch** - Whether to perform the search against all subtrees of the search base DN specified, or just the immediate children of the base DN, acceptable values are *true* or *false*, default value is *false*

Simple configuration with three different LDAP servers:

```
ShibUserPassAuth {
    edu.vt.middleware.ldap.jaas.LdapLoginModule required
        host="ldap1.example.org ldap2.example.org
            ldap3.example.org"
        base="ou=people,dc=example,dc=org"
        tls="true"
        userField="uid"
    ;
};
```

It is possible to stack multiple login modules in a single configuration. For example if we need to search for the user in two different search bases:

```

ShibUserPassAuth {
    edu.vt.middleware.ldap.jaas.LdapLoginModule sufficient
        host="ldap.example.org"
        base="ou=Employees,dc=example,dc=org"
        ssl="true"
        userField="uid";
    edu.vt.middleware.ldap.jaas.LdapLoginModule sufficient
        host="ldap.example.org"
        base="ou=Students,dc=example,dc=org"
        ssl="true"
        userField="uid";
};

```

Note that the flag after the login module specification is in that case set to *sufficient* instead of *required*. Other possible values are *requisite* and *optional* (see [5]).

2.2 The UsernamePassword login handler

Generally, Shibboleth Identity Provider 2.x may use a specific login handler required by the service provider upon the authentication request. But mostly, service providers do not require specific handlers and a default login handler is used instead. Such default login handler may be configured for any relying party.

The *UsernamePassword* login handler is configured in the *conf/handler.xml* file [4]. Uncomment it, if it is commented out (as default). Here you can specify some custom attributes:

- **jaasConfigurationLocation** – the path to the JAAS configuration file used by the login handler, default is *conf/login.conf*
- **authenticationDuration** – length of time in minutes that the authentication method associated with this login handler is active, default is 30 minutes
- **authenticationServletURL** – path to the servlet responsible for collecting using credentials and authenticating the user (usually some kind of a login form), default is *CONTEXT_PATH/Authn/UserPassword*, where *CONTEXT_PATH* is the root directory of the servlet context

Example configuration with authentication duration of 12 hours:

```

<LoginHandler xsi:type="UsernamePassword"
    jaasConfigurationLocation="file:///opt/shibboleth-idp/conf/login.config"
    authenticationDuration="720">
    <AuthenticationMethod>
        urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport
    </AuthenticationMethod>
</LoginHandler>

```

After we have our *UsernamePassword* login handler configured, we need to assign it to a relying party in the *conf/relying-party.xml* configuration file [4]. Relying party may represent a single service provider, a group of service providers or a whole federation. The default login handler for the relying party is set through the *defaultAuthenticationMethod* attribute in the appropriate *RelyingParty* element.

Its value corresponds to the value of the `AuthenticationMethod` element specified in the login handler configuration (see above):

```
<RelyingParty id="some:federation"
  provider="https://login.example.org/idp/shibboleth"
  defaultSigningCredentialRef="loginCreds"
  defaultAuthenticationMethod=
    "urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport">
  <ProfileConfiguration xsi:type="saml:ShibbolethSSOProfile"/>
  <ProfileConfiguration xsi:type="saml:SAML1AttributeQueryProfile"/>
  <ProfileConfiguration xsi:type="saml:SAML1ArtifactResolutionProfile"/>
  <ProfileConfiguration xsi:type="saml:SAML2SSOProfile"/>
  <ProfileConfiguration xsi:type="saml:SAML2AttributeQueryProfile"/>
  <ProfileConfiguration xsi:type="saml:SAML2ArtifactResolutionProfile"/>
</RelyingParty>
```

2.3 Custom login page

Shibboleth Identity Provider ships a simple login page for the *UsernamePassword* login handler. It contains a login form with a username and password fields. In order to customize the login page, we need to modify the *login.jsp* file located in the directory *src/main/webapp/* of the distribution package (the directory from which the installer is run). Each time a modification is made, the installer should be re-run to deploy the new WAR package.

The names of the form input fields must remain *j_username* and *j_password* and the form action must remain */Authn/UserPassword*. If the page uses other encoding than iso-8859-1, for example utf-8, we need to put the following lines at the beginning of the file:

```
<%@ page
  contentType="text/html; charset=UTF-8"
%>
```

Some custom settings may be specified in the servlet configuration, which can be found in the *web.xml* file located in the *src/main/webapp/WEB-INF/* directory of the distribution package. For example, another path for the *login.jsp* file may be specified:

```
<servlet>
  <servlet-name>UsernamePasswordAuthHandler</servlet-name>
  <servlet-class>
edu.internet2.middleware.shibboleth.idp.authn.provider.UsernamePasswordLoginServlet
  </servlet-class>
  <init-param>
    <param-name>loginPage</param-name>
    <param-value>myFederation/login.jsp</param-value>
  </init-param>
</servlet>
```

The path is relative to the servlet context. That means, that our custom *login.jsp* file should be placed in the *src/main/webapp/myFederation* directory.

After any modifications made to *web.xml*, *login.jsp* or possibly other files like images and CSS files we need to re-run the installer, which rebuilds and deploys the WAR file.

2.4 Logging

Shibboleth Identity Provider uses the Logback¹. LDAP processing is not logged by default. To enable LDAP logging we should add a logger with the needed verbosity level to the *conf/logging.xml* file:

```
<logger name="edu.vt.middleware.ldap">
  <level value="DEBUG"/>
</logger>
```

3 LDAP as an attribute storage

Shibboleth Identity Provider does not store user attributes itself. It uses external data storages instead. These storages are configured as data connectors in the *conf/attribute-resolver.xml* configuration file. Shibboleth ships some common data connectors in its distribution, but it is also possible to write a custom data connector, if there is a need for some extra functionality.

3.1 Basic configuration

The data connectors are defined in `DataConnector` elements with unique `id` attributes and their type is distinguished through the `type` attribute. To define an LDAP data connector we need to create a `DataConnector` element of the `LDAPDirectory` type. The LDAP connection properties may be set through the following attributes:

- **ldapURL** – space separated list of LDAP base URLs, each listed URL is tried in turn until one accepts the connection
- **baseDN** – the base DN for the search
- **principal** – the DN of the user, used to bind to the directory
- **principalCredential** – the password for the principal

The filter for the search is specified in the `FilterTemplate` element and uses a template engine based on the Velocity Project². The template engine offers a very flexible way for constructing queries, but in most cases we need just a simple uid filter as shown in the example below, so there is no need to study the filter syntax. It is recommended to wrap the filter definition into a CDATA block. Additionally, we may specify a list of attributes to be returned from LDAP (instead of retrieving all of them) in the `ReturnAttributes` element.

This is an example of a common LDAP data connector:

¹ <http://logback.qos.ch/>

² <http://velocity.apache.org/engine/releases/velocity-1.5/>

```

<resolver:DataConnector
  xsi:type="LDAPDirectory"
  xmlns="urn:mace:shibboleth:2.0:resolver:dc"
  id="myLDAP"
  ldapURL="ldap://localhost"
  baseDN="o=myorg"
  principal="attrretriever"
  principalCredential="secret">
  <FilterTemplate>
    <![CDATA[
      (uid=${requestContext.principalName})
    ]]>
  </FilterTemplate>
  <ReturnAttributes>uid cn givenName sn mail</ReturnAttributes>
</resolver:DataConnector>

```

3.2 Advanced options

The LDAP data connector offers far more options to customize and fine-tune its functionality. Some common cases are described in the following sections. For complete reference, check the official Shibboleth 2 documentation [4].

3.2.1 Dependencies and advanced filter expressions

It is possible to use multiple data connectors on a single attribute query. The filter expression [6] in the `FilterTemplate` element may use information retrieved from another component – data connector or attribute. That means that the data connector is dependant on the other component. We need to express that dependency through a `Dependency` element with a `ref` attribute referencing the unique `id` attribute of the target component.

For example, if we need to retrieve information about a specific resource assigned to the user and the resource data are stored in a different database, we need to perform two queries. The first one retrieves user information, which also contains the name of the resource. The second one retrieves the resource information itself from the other database based on the resource name retrieved from the first database.

To accomplish that, we need to specify a second data connector along with the first one (shown in the example above). In that connector we specify a dependency on the attribute retrieved by the first data connector and containing the resource name. Then we can use its value in the filter:

```

<resolver:DataConnector
  xsi:type="LDAPDirectory"
  xmlns="urn:mace:shibboleth:2.0:resolver:dc"
  id="myResourceLDAP"
  ldapURL="ldap://resource.myorg.org"
  baseDN="o=myorg" principal="attrretriever"
  principalCredential="secret">
  <resolver:Dependency ref="userResource"/>

```

```

    <FilterTemplate>
      <![CDATA[
(resourceName=${userResource.get(0)})
]]>
    </FilterTemplate>
    <ReturnAttributes>resourceAttr1 resourceAttr2</ReturnAttributes>
</resolver:DataConnector>

```

3.2.2 Retrieving group information

Often, directory services store user group information in a special sub-tree, for example *ou=groups,o=myorg*. User membership for each group is then expressed by multiple *uniquemember* attributes containing the DNs of the member users. With such structure it is not possible to retrieve user's group membership along with his other attributes in a single query. So the need of a second query emerges.

To perform a secondary query, we need to define another data connector with a different baseDN attribute and of course a different filter. By default, the data connector expects only one result, but if the user is a member of multiple groups, the LDAP server will return more than one result and an error will occur. We need to specify some appropriate number of expected results through the *maxResultSize* attribute. Additionally, we should set the *mergeResults* attribute to *true* in order to make the data connector merge all the results into a single set like a multiple value attribute.

```

<resolver:DataConnector
  xsi:type="LDAPDirectory"
  xmlns="urn:mace:shibboleth:2.0:resolver:dc"
  id="myLDAPgroups"
  ldapURL="ldap://localhost"
  baseDN="ou=groups,o=myorg"
  principal="attrretriever"
  principalCredential="secret"
  mergeResults="true"
  maxResultSize="30">
  <FilterTemplate>
    <![CDATA[
(uniquememberof=uid=${requestContext.principalName},ou=people,o=myorg)
]]>
  </FilterTemplate>
  <ReturnAttributes>cn</ReturnAttributes>
</resolver:DataConnector>

```

Then we specify an attribute definition dependent on the data connector (shown above) and with an appropriate source attribute (the attribute containing the group name, usually *cn*):

```

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

```

```
<!-- attribute encoders ... -->
</resolver:AttributeDefinitio>
```

3.2.3 SSL/TLS Support

The LDAP data connector supports both LDAP over SSL and LDAP with startTLS. To enable LDAP over SSL we need to specify an *ldaps://* URL in the `ldapURL` attribute. Additionally we should add the LDAP server's digital certificate to the JVM's keystore:

```
$ keytool -import -trustcacerts -alias "myldap" -file myldap.crt \
  -keystore $JAVA_HOME/lib/security/cacerts
```

To enable LDAP with startTLS we need to set the attribute `useStartTLS` to *true*. Instead of using a keystore, the certificate may be placed inline as a `StartTLSTrustCredential` element in the data connector's definition. It is also possible to use client authentication when accessing the LDAP server. To achieve that, we need to supply a client certificate placed in a `StartTLSAuthenticationCredential` element in the data connector's definition. All inline certificates should be defined as an IdP credentials (see [7]).

```
<resolver:DataConnector [...] useStartTLS="true">
  <!-- [...] data connector configuration -->
  <StartTLSTrustCredential xsi:type="X509Inline"
    xmlns="urn:mace:shibboleth:2.0:security"
    id="MyLDAPCredential">
    <Certificate>
      <!-- Some DER or PEM encoded cert -->
    </Certificate>
  </StartTLSTrustCredential>
  <StartTLSAuthenticationCredential xsi:type="X509Inline"
    xmlns="urn:mace:shibboleth:2.0:security"
    id="MyClientCredential">
    <Certificate>
      <!-- Some DER or PEM encoded cert -->
    </Certificate>
  </StartTLSAuthenticationCredential>
</resolver:DataConnector>
```

4 Conclusion

Shibboleth Identity Provider is very "LDAP-friendly" and offers flexibility and wide variety of options for optimal interconnection with the organizational identity management based on LDAP. Additionally, the LDAP support in Shibboleth is based on standard-compliant, well-documented open source tools, with all the benefits it brings.

References

- [1] Lightweight Directory Access Protocol. In *Wikipedia, The Free Encyclopedia*. Available online³.
- [2] SUN MICROSYSTEMS. *Java™ Authentication and Authorization Service (JAAS): Reference Guide*, 2001. Available online⁴
- [3] FISHER, D. *LDAP utilities*, 2008. Available online⁵.
- [4] INTERNET 2. *Shibboleth 2 Documentation*, 2009. Available online⁶.
- [5] SUN MICROSYSTEMS. javax.security.auth.login Class Configuration. In *Java™ 2 Platform Standard Edition 5.0: API Specification*. Available online⁷.
- [6] INTERNET 2. *JavaTemplateEngine*, 2008. Available online⁸.
- [7] INTERNET 2. *IdPCredentials*, 2007. Available online⁹.

³ <http://en.wikipedia.org/wiki/LDAP>

⁴ <http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html>

⁵ <http://www.middleware.vt.edu/doku.php?id=middleware:opensource:ldap>

⁶ <https://spaces.internet2.edu/display/SHIB2/Home>

⁷ <http://java.sun.com/j2se/1.5.0/docs/api/javax/security/auth/login/Configuration.html>

⁸ <https://spaces.internet2.edu/display/SHIB2/JavaTemplateEngine>

⁹ <https://spaces.internet2.edu/display/SHIB2/IdPCredentials>