IdPAttributeConfig

Shibboleth Attributes

The Shibboleth IdP acquires all attributes it sends using a specialized attribute resolver. Every attribute must be converted to a SAML-based XML form to be sent to service providers. SAML attributes include a URN-based name(which will often be `urn:oid:`*`oid`*), a value, and an optional scope value for Scoped attributes. Attribute names and definitions will frequently be defined by both standards bodies and federations in order to ensure that IdPs and SPs can communicate seamlessly.

Attributes are generally drawn from data sources using JNDI connectors and a set of definitions that map this information to SAML attributes. Customized Java classes may be written that can generate attributes on the fly or draw information from anywhere. The `ResolverConfig` attribute of IdPConfig in `idp.xml` points to the location of the active attribute resolver configuration file for this identity provider.

It is important to note no attributes will be sent unless there are corresponding attribute release policies in place. Similarly, attribute release policies cannot release attributes that haven't been defined in the resolver.

The Attribute Resolver

The resolver is essentially a directed graph from attribute definitions to data connectors. The data connectors pull data, in the form of attributes, from external data sources. The attribute definitions then process this data into a from suitable for use by Shibboleth. This procedure can be as simple as taking an unmodified string value from a data connector and tagging it with a name or can include arbitrarily complex business rules.

The `resolver.xml` file that is pointed to by `idp.xml` consists of zero or more attribute definitions followed by zero or more data connectors. Each attribute definition consists of an identifier corresponding to the URN of the attribute, and optional references to data connectors on which it depends. Each data connector consists of a string identifier which is used by attribute definitions that refer to it, and one or more elements specific to the configuration of that data connector.

Shibboleth comes with two attribute definitions provided in version 1.3: the `SimpleAttributeDefinition` , which acts as a basic proxy for attributes supplied by data connectors with some name conversion and attribute scoping added, and a `CustomAttributeDefinition` , which can be used to configure user-created attribute definition plugins.
Shibboleth 1.3.1 added the ScriptletAttributeDefinition executig Java code when the attribute is requested.

Shibboleth 1.3 comes with three data connectors, supporting JNDI, JDBC, and customized interfaces. The `JNDIDirectoryDataConnector` pulls data from any source for which there is a JNDI Directory Context implementation, including LDAP, NDS, etc. `JDBCDataConnector` allows for standard JDBC connections to databases such as MySQL and Oracle. The `CustomDataConnector` allows for definition of user-created data connector plugins.
Shibboleth 1.3.1 added the StaticDataConnector which can add static attribute values declared in the connector's configuration.

Data Connector Definitions

<JNDIDirectoryDataConnector id=string>
Defines a JNDI connector with a unique, textual name used to associate an attribute definition with this connector. `resolver.ldap.xml` comes with the Shibboleth distribution and provides excellent examples. Make sure to use the secure example for binds over SSL.

```
<JNDIDirectoryDataConnector id="directory">
        <Search filter="uid=%PRINCIPAL%">
                <Controls searchScope="SUBTREE_SCOPE" returningObjects="false" />
        </Search>
        <Property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory" />
        <Property name="java.naming.provider.url" value="ldap://id.youngvillains.org/dc=youngvillains,dc=org" />
        <Property name="java.naming.security.principal" value="cn=Admin,dc=youngvillains,dc=org" />
        <Property name="java.naming.security.credentials" value="secret" />
</JNDIDirectoryDataConnector>
```

- `<Property name="`*`<name>`*`" value="`*`<value>`*`"/>` : An element of the element `JNDIDirectoryDataConnector` . Specifies a set of name /value pairs that are used to configure the JNDI Directory Context. This list of name/value pairs is defined by the context itself, but is specified within `resolver.xml` .

  > ⚠ Active Directory users should note that the value of the java.naming.security.principal property should format "user@domain.edu"

- `<Search>` : An element of the element `JNDIDirectoryDataConnector` . This element defines the DN filter used to perform the LDAP search. The search string must return no more than one result.
  - `<Controls>` : An element of the element `Search` . This element grants some fine-grained control over the LDAP API calls.
  - `<cacheTime "`*`<seconds>`*`"/>` : An element of the element `JNDIDirectoryDataConnector` . Specifies an optional duration in `seconds` for which the attribute resolver may cache information retrieved from this connector. The default is zero seconds (no caching).
- The JNDIDirectoryDataConnector implements fail-over to another directory using the optional <FailoverDependency requires="*directory2*" />

<JDBCDataConnector id="*uniqueId*" dbURL="*URL*" dbDriver="*classname*" maxActive="*integer*" maxIdle="*integer*">
The JDBC data connector defines an attribute source and the means used to connect to it. It may contain a <Query> element, or more elaborate queries can be generated using dependencies on other attributes.

<SimpleAttributeDefinition id="*URN*" [smartScope="*domain*"] [sourceName="*string*"]>
Specifies a unique, textual name for the attribute which is used as the attribute's name both on the wire and as the basis for the repository lookup. This will work for most deployments.

- `sourceName="<string>"` : Specifies a different source attribute name to be used in calls to the data connector, while the name on the wire will be the specified `id` . An example use would be to send a local UniversityID attribute as eduPersonPrincipalName.
- `smartScope="<domain>"` : Specifes an optional domain scope to be attached to the attribute. This will make the SAML attribute a `scoped` attribute on the wire. If the value of the attribute as retrieved from the data connector includes a pre-existing scope (`bob@foo.edu]`), that scope is used instead. Contained within the `SimpleAttributeDefinition` element.
- `<lifeTime "<seconds>"/>` : Specifies in the attribute assertion how long the attribute should be cached and retained by the target upon receipt. Federations and trust agreements may have some bearing on the population and use of this field. Contained within the `SimpleAttributeDefinition` element.
- `<DataConnectorDependency requires="<id>"/>` : This element is placed within attribute definitions to designate a repository to draw the attribute from. The `id` must correspond to a `JNDIDirectoryDataConnector` , a `JDBCDataConnector` , or a `CustomDataConnector` .
- `<AttributeDependency requires="<id>"/>` : Shibboleth is also able to build attributes from other attributes. This is mostly useful to provide simple transformations in attribute names or to supply both scoped and simple versions of attributes.
- `<cacheTime "<seconds>"/>` : Specifies an optional duration in `seconds` for which the attribute resolver may cache this attribute for use in additional assertions.

resolvertest

Shibboleth comes bundled with a command line utility `resolvertest`, for testing `Attribute Resolver` configurations. This program takes as input the configuration file `resolver.xml`, the name of a user, and optionally the name of a requesting providerID for 1.2 or laters SP's. It outputs the resulting SAML attribute elements. This allows administrators to view the results of tweaking the resolver configuration without having to continually reload the origin web application. `resolvertest` is also useful for testing when the Attribute Authority is first configured to use an attribute repository (e.g. LDAP or SQL). Initially, the following two steps must be performed:

- Set the shell variable `IDP_HOME` to the directory path where the Shibboleth tarball was exploded (i.e. `/opt/shibboleth-idp`)
- cd $IDP_HOME/bin/

`resolvertest` may then be used by executing the shell script, passing the name of a user and a URL to the Attribute Resolver configuration file as parameters.

This program does not filter the resulting attributes through the applicable ARP's. Although it does show the attributes generated by the resolver for a particular user, it does not necessarily reflect what will be released by the AA to a requesting SP. Before you can start this program, make sure that you have already defined the environment variable `IDP_HOME`:

```
root# export IDP_HOME=/opt/shibboleth-idp
```

Now, you can run `resolvertest` like this:

```
root# ./resolvertest --user=wassa --resolverxml=file:///${IDP_HOME}/etc/resolver.xml
```