AttributeResolver

The IdP's *Attribute Resolver* performs three main tasks: pulling in data from external systems (e.g., LDAP directories and relational databases), creating attributes from the pulled in data, and associating protocol-specific encoders with the created attributes.

Attributes and Encoders

An *Attribute*, within the IdP, is a protocol-agnostic data structure that contains:

- a unique (within the IdP) identifier for the attribute
- a set of localized names that can be displayed to the user
- a set of localized descriptions that can be displayed to the user
- a set of values for the attribute
- a set of encoders that can transform the attribute in to a protocol-specific representation

For clarity, by protocol-agnostic we mean that within the IdP there is no special implementation of an Attribute for each given protocol supported by the IdP. For example, there is no such thing as a "SAML 2 Attribute" within the IdP. In a bit we will discuss the encoders used to transform attributes in to their protocol-specific formats.

Resolver Components

The attribute resolver is made of three different components: data connectors, attribute definitions, and attribute encoders. We'll cover each of these in turn.

Data Connectors

Data Connectors are used to pull in the raw information that will be made in to attributes. This information is often, but not always, pulled in from external systems like LDAP directories or relational databases. The result of a data connector's work is a collection of attributes that have an identifier and some values.

Data connectors also have a concept of a *failover data connector*. This is another data connector that is executed in place of a data connector that fails. For example, an IdP may have a local copy of an LDAP directory to ensure especially fast read times. However, the data connector hooked up to the local LDAP may specify a failover data connector that connects to the centralized LDAP if the local one is down.

Attribute Definitions

Attribute Definitions, at a minimum, associate encoders and display names and descriptions with an attribute. Most attribute definitions will also perform some form of transformational logic on the attribute's values. For example, an attribute definition may evaluate a regular expression against an attribute's value and replace the current value with only the portion that matches the regular expression.

It is important to note, only those attributes which are processed by an attribute definition are ever released outside the attribute resolver. Anything else is simply considered internal, to the resolver, information.

Attribute Encoders

We've mentioned *Attribute Encoders* a couple of times now. These components, as their name probably suggests, are used to encode attributes in to a protocol specific representation. As such they also contain any of the protocol-specific identification needed for the attribute. So, for example, a SAML 2 encoder would provide mechanisms for setting the attribute name and format and contain the logic necessary to produce the XML structure required by the SAML 2 specification.

This division between attributes and their encoded forms means that the IdP can use its resolution, filtering, user consent, and other services for any protocol, even ones added later by extension. So if OpenID support were ever added to the IdP the only attribute-related components that would need to change would be the addition of an OpenID attribute encoder.

Attribute Mappers (or decoders)

The *Attribute Mapper* is the inverse of attribute encoding. These components inspect metadata (usually of the SP) to extract any SAML2 RequestedAttributes. These are decoded use the same configuration as the attribute encoders to produce IdPRequestedAttribute. This decoding includes not only the name changes (from the on-the-wire names to those used to configure the Attribute Resolver and Filter), but also types, as defined by the Attribute Encoders.

Additional Mechanics

Attribute Dependencies

When reading the above descriptions of the data connectors and attribute definition you may have asked yourself questions like "What if I want to use the value of an attribute within the SQL statement of my relational database data connector?" or "What if I want an attribute definition that joins together the value of two other attributes, for example, to produce a display name from a given name and a surname?".

Both data connectors and attribute definitions allow you to define one or more dependencies on other data connectors and attribute definitions. This does two things. First, it ensure that the resolver has executed the dependencies before you need them. Second, it makes the values of those dependencies available to the data connector or attribute definition that defines the dependency.

This ability to string together data connectors and attribute definitions provides a powerful mechanisms for transforming and preparing data.

Conditional Evaluation

Another thought you may have had, especially when reading about data connectors, is whether it was possible to define some condition under which a particular data connector or attribute definition would never run. For example, assume you're at a university and all your students are in an LDAP directory and all your employees are in a database. It would be wasteful if, for each request, you queries both the student LDAP directory and the employee database. Any given individual is likely only in one of these data sources.

Both data connectors and attribute definitions provide a mechanism for specifying a condition under which they will execute. If the condition is absent or evaluates to 'true' then the component will execute for the given request. Otherwise it will not.

The Resolution Process

So, what actually happens when the resolver runs? How do all of these components fit together? The attribute resolution process goes like this:

For each defined attribute definition:

- if any dependency is listed, and has not yet been executed, it is executed
- if no evaluation condition is given or a condition is given and it evaluates to true then proceed
- the attribute definition is executed
- the results of the execution are cached for the current request

For each data connector that is executed the process goes like this:

- if any dependency is listed, and has not yet been executed, it is executed
- if no evaluation condition is given or a condition is given and it evaluates to true then proceed
- the data connector is executed
- the results of the execution are cached for the current request

It is important to note that as a result of this resolution process a data connector that is never directly, or transitively, a dependency of an attribute definition it will never be executed.

Contexts

Parameters to the Attribute Resolution process are supplied by the AttributeResolutionContext class. In addition the AttributeResolverWorkContext is used as work space for the attribute resolution process; this context is entirely private to the attribute resolution process and is only of interest to those components such as components derived from one of the Resolver Plugins.

Programming Guide to Attribute Resolution

A web flow that wishes to invoke the Attribute Resolution subsystem must do the following.

1. Create an AttributeResolutionContext and populate it with appropriate parameters:
    - The canonical Prinicipal name
    - The entityID of the IdP and the SP
    - Optional ID of an authentication flow used
    - Additionally the precise names of the attributes resolved can also be set.
2. Locate an Attribute Resolver. This would usually be done via a ReloadableSpringService. See ReloadableConfiguration.
3. Call the resolver. After resolution, the result can be extracted via the getResolvedIdPAttributes() method and will usually be inserted into an AttributeContext.
4. If using a ReloadableSpringService. do not forget to call unpinComponent.