Portal Design

SP Integration

Exposing attributes from SP to portal

The Shibboleth Service Provider represents the user attributes extracted from a SAML authentication response as HTTP headers.

> **See also**
>
> See also documentation of adding an attribute to the Shibboleth native SP.

For instance, this mapping in attribute-map.xml would instruct the SP to map the user's favorite fruit user attribute to an HTTP header named "favFruit".

Configuring the SP to map the user's FavoriteFruit user attribute to a header named "favFruit"

```
<Attribute name="https://example.org/myAttributes/FavoriteFruit" id="favFruit"/>
```

uPortal has a pluggable API for sources of user attributes (represented as Person Attribute Data Access Objects (DAOs)). Johns Hopkins University has developed a Person Attribute DAO with the behavior of presenting HTTP request attributes as uPortal Person Attribute API attributes.

The Person Attribute subsystem doesn't have access to the HTTP request and its headers. The Johns Hopkins implementation solves this with a bridging Java Servlet Filter.

This filter is declared in *web.xml* like this:

Configuring a filter to capture selected HTTP headers and feed them to the Person Directory DAO

```
<filter>
    <filter-name>HttpHeaderFilter</filter-name>
    <filter-class>edu.jhu.services.persondir.support.http.HttpHeaderFilter</filter-class>
    <init-param>
        <param-name>personDirectoryContextLocation</param-name>
        <param-value>properties/contexts/personDirectoryContext.xml</param-value>
    </init-param>
</filter>
...
<filter-mapping>
    <filter-name>HttpHeaderFilter</filter-name>
    <servlet-name>Login</servlet-name>
</filter-mapping>
```

The data access object itself is declared something like this:

Declaring a HTTP header as a source of a user attribute

```
<bean id="httpHeaderAttributeSource" class="edu.jhu.services.persondir.support.http.
HttpHeaderPersonAttributeDao">
  <constructor-arg>
    <value>edu.jhu.services.persondir.support.http.HttpHeaderPersonAttributeDao</value>
  </constructor-arg>
  <property name="columnsToAttributes">
    <map>
      <entry key="favFruit">
        <value>favorite_fruit</value></entry>
    </map>
  </property>
</bean>
```

What this says is "Map the HTTP header named 'favFruit' to the portal user attribute named 'favorite_fruit'."

This source of user attributes can be declared alongside sources such as RDBMS queries and LDAP queries.

Configuring HTTP headers as source of attributes alongside other sources

```
<bean id="mergedPersonAttributeDao" class="org.jasig.services.persondir.support.MergingPersonAttributeDaoImpl">
  <property name="personAttributeDaos">
    <list>
      <ref bean="uPortalJdbcAttributeSource"/>
      <ref bean="uPortalLdapAttributeSource"/>
      <ref bean="httpHeaderAttributeSource"/>
    </list>
  </property>
  <property name="merger">
    <bean class="org.jasig.services.persondir.support.merger.MultivaluedAttributeMerger"/>
  </property>
</bean>
```

What this says is "Consider the database, LDAP, and HTTP headers as sources of user attributes, in the configured ways."

This binding of HTTP headers to uPortal person attribute APIs is uPortal specific (actually, JASIG Person Attribute API specific, with uPortal being the only major portal platform making use of this API). However, a similar approach of binding the attribute-bearing headers provided by the Shibboleth SP to the APIs whereby the portal expects to represent user attributes is applicable to all portal platforms.

Exposing User Attributes from SP to JSR-168 Portlets

Background on User attributes in JSR-168 Portlets

Once user attributes are available to the portal, these attributes can be used for several purposes. They can be inputs to decisions about user layout and evaluation of user group membership. The attributes can also be presented to particular portlets running in the portal. Not all user attributes known to the portal need be available to a given portlet running within the portal.

JSR-168 portlets declare at deployment time which user attributes they may use by means of declaration in their portlet.xml deployment descriptor.

```
<user-attribute>
  <description>User Favorite Fruit</description>
  <name>favorite_fruit</name>
</user-attribute>
```

What this says is "This portlet relies upon the portal to provide a user attribute 'favorite_fruit'".

JSR-168 portlets can read values of their declared user attributes at runtime via a JSR-168 API:

Reading a portlet user attribute at runtime in a JSR-168 portlet

```
// in the course of handling a PortletRequest named 'request'
Map userInfo = (Map) request.getAttribute(PortletRequest.USER_INFO);
String surname = (String) userInfo.get("favorite_color");
// presumably, do something interesting based on the user's favorite color...
```

JSR-168 portlets do not necessarily have access to the HTTP headers of the portal HTTP request (have no API-defined access to the HttpRequest object). They rely rather on JSR-168 APIs for conveyance of username, user attributes, and other information about the request from portal to portlet.

Exposing Shibboleth SP-provided user attributes to portlets

Under the Johns Hopkins approach, Shibboleth-provided user attributes are treated like other user attributes. Exposing particular attributes to portlets is a matter of collecting those attributes from the HTTP headers in the HttpHeaderPersonAttributeDao, and then mapping the resulting portal user attribute into the particular portlets making use of it.

Limiting access to attributes from SP to portlets

Existing Solution

Access to user attributes is already limited by the JSR-168 API and declaration of JSR-168 portlet attribute names in each portlet's deployment descriptor *portlet.xml*. If a portlet should have access to the portal user attribute that derives from a particular Shibboleth-provided user attribute that was represented by the Service Provider as an HTTP header, then that portlet's portlet.xml should map the corresponding portal user attribute. If another portlet should not have access to this attribute, its portlet.xml should fail to declare that attribute.

Directions for further limiting access to attributes from SP to portlets

This approach of the SP presenting user attributes to the portal and then allowing (and relying upon) the portal to present appropriate user attributes to appropriate portlets works well so long as the IDP is not modeling finer-grained attribute release policy as regards releasing attributes to some portlets and not others. If this richer attribute release guidance is present, then a more sophisticated solution may be needed.

This might involve a "smarter" Service Provider that represents this detail in the namespace of the HTTP headers representing the attributes, a "smarter" implementation of the JSR-168 portlet person attribute provider API which restricts programmatically which attributes which portlets may read, or even a "smarter" Http header reading Person Attribute DAO which uses Aspect-oriented access control to just-in-time gate access to restricted Shibboleth-provided user attributes.

Portal/Portlet Security Token Handoff

Mechanism for portlet to ask for an assertion to proxy with

Background on how assertion-like things are currently handled in uPortal

JSR-168 Portlets can read end user's password in a special user attribute

uPortal's JSR-168 support has special handling for the user attribute "password", such that if it is mapped in the portlet's portlet.xml then if the portal is aware of the user's password it will make it available to the portlet alongside the other user attributes.

This convention for treatment of the JSR-168 user attribute "password" is uPortal-specific, though of course the JSR-168 user attribute API itself is generic across all JSR-168-supporting portals.

See also

See also uPortal 3 Manual Page on this topic.

JSR-168 Portlets can obtain a CAS proxy ticket via a special user attribute

uPortal's JSR-168 support has special handling for the user attribute "casProxyTicket", such that if it is mapped in the portlet's portlet.xml then if the portal is aware of a relevant CAS single sign on session (is in possession of a valid Proxy Granting Ticket for the current user session) the portal will obtain from CAS a Proxy Ticket for the purpose of proxying authentication to the portlet. By convention, the CAS Service URL of a portlet is represented as the URL of the servlet application containing the portlet, up to and including context path.

The portlet then validates this Proxy Ticket with CAS and itself receives a Proxy Granting Ticket from which it can then obtain Proxy Tickets for proxying authentication to backing services in the normal way of CAS proxy authentication.

Example of the Service URL representing a JSR-168 portlet and the CAS protocol

Suppose that in the portal at https://portal.example.edu/ there is a portlet web application OrderStatusPortlet.war such that, to the extent that this portlet application is also a servlet application, it answers at https://portal.example.edu/OrderStatusPortlet/ .

The Service URL of this portlet is therefore https://portal.example.edu/OrderStatusPortlet/ . The portal will use its Proxy Granting Ticket to obtain a Proxy Ticket for the purpose of authenticating to https://portal.example.edu/OrderStatusPortlet/ , which it will then present to the portlet as the special JSR-168 user attribute "casProxyTicket".

The Portlet will then validate this ticket with CAS, specifying at validation that it would itself like to obtain a Proxy Granting Ticket and providing to CAS a callback URL at which it would like to receive that Proxy Granting Ticket. CAS calls back to a URL like

https://portal.example.edu/OrderStatusPortlet/proxyTicketReceptor

providing a Proxy Granting Ticket which authenticates the chain of user-portal-portlet. The portlet then uses this Proxy Granting Ticket to obtain Proxy Tickets for the purpose of authenticating to specific backing services.

Presenting Shibboleth assertion to portlet

As a JSR-168 user attribute

It is tempting to model the Shibboleth assertion as a JSR-168 User Attribute for similarity to the approaches for handling passwords and CAS proxy tickets.

A portlet's request to access the assertion Shibboleth provided to the portal might be modeled as a request for user attribute "samlAssertion". Similar to the current Johns Hopkins solution of capturing HTTP headers and presenting them as user attributes, the full SAML Assertion would be exposed by the Service Provider as an HTTP header which a filter would capture into the user's portal session (likely in a uPortal SecurityContext in a manner similar to the handling of CAS Proxy Granting Tickets). This assertion would then be available to JSR-168 portlets as a virtual user attribute akin to "casProxyTicket" and "password", supplied by an additional UserInfoService.

JSR-168 portlets declare at deployment time which user attributes they may use by means of declaration in their portlet.xml deployment descriptor.

```
<user-attribute>
  <description>SAML Assertion authenticating user</description>
  <name>samlAssertion</name>
</user-attribute>
```

A JSR-168 portlet so provisioned would then read the SAML assertion as a user attribute at runtime via the JSR-168 user attribute API:

Reading the 'samlAssertion' portlet user attribute at runtime in a JSR-168 portlet

```
// in the course of handling a PortletRequest named 'request'
Map userInfo = (Map) request.getAttribute(PortletRequest.USER_INFO);
String surname = (String) userInfo.get("samlAssertion");
// make use of this Assertion to authenticate to backing services via the Portlet Client Library to be designed
below...
```

Encoding the SAML Assertion

The SAML Assertion could be modeled as a Base64 encoded String so that a more complex object need not be passed across the classloader boundary (with all those attendant classloader issues) between portal and portlet. If done carefully this should preserve the validity of any signatures and allow the portlet to reconstruct the assertion.

Portlet Client Library

There's now a child page for exploring this library.

- Need an HTTP client API allowing access to WSPs while handling authentication internally
- Client stack needs support for TLS client and server authentication, cookies, and whatever features WSPs would impose
- Internals must support detection of ECP requests, Liberty-defined interactions with IdP, and handling ECP responses