IdPUpgradesViaDuplication

A "brute force" approach to getting an IdP to live at two different sets of locations (whether on the same or a different physical server) is to simply run a second copy.

I found it simpler to do this on the same server, because it made it possible for me to maintain SSO between the two copies without having to rely on some kind of external SSO mechanism.

Since the specifics will depend on how you have things set up, I can only document the scenario I found myself in.

Same Servers in Cluster, Container-Managed Authentication

I had multiple copies of the IdP running behind a load balancer, and have configured container-managed authentication via the Tomcat JAAS Realm. The IdP was configured with the cookie-based SSO approach described here (with authHeaderName="COOKIE"). Initially the IdP was deployed to a servlet context named "shibboleth" with the various paths otherwise matching the 1.3 defaults.

My desired state was to have a second copy of the 1.3 IdP running in a servlet context named "idp" and to move the various legacy profile handlers to locations that match the new IdP (e.g., /idp/profile/Shibboleth/SSO, (idp/profile/SAML1/SOAP/AttributeQuery). I wanted as much of the configuration as possible (metadata, resolver, ARPs, authentication) to be shared and to have only a single compiled warfile. But I also needed to maintain SSO between the copies, and to make the logging separate. The latter also meant that the path to the primary configuration file had to be different in the two copies.

Tomcat makes all this relatively easy, since the same warfile can be deployed multiple times using "context descriptors" that place the application at different base locations. It's also possible using the context descriptor to override servlet parameters from outside the web.xml file, in this case the "IdPConfigFile" parameter.

web.xml Modifications

First, I prepared the single warfile to operate on the two context paths and the additional locations by modifying the `webAppConfig/web.xml` template as follows:

- Added `<filter-mapping>` elements so that various Java filters I had in use would run in front of the new SSO request paths.

- Added a duplicate set of `<servlet-mapping>` elements so the named "IdP" servlet would run at the new paths needed.

Examples of Additional Mappings

```
<servlet-mapping>
    <servlet-name>IdP</servlet-name>
    <url-pattern>/profile/Shibboleth/SSO</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>IdP</servlet-name>
    <url-pattern>/profile/Shibboleth/HS</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>IdP</servlet-name>
    <url-pattern>/profile/SAML1/SOAP/AttributeQuery</url-pattern>
</servlet-mapping>
```

- Added an additional `<url-pattern>` to the set of container-protected paths.

idp.xml Modifications

I needed two copies of the master configuration file, but only to keep the logging separate; the rest is identical. I copied the existing `idp.xml` to a second copy called `shibboleth.xml` and configured the logging to load different log4j property files with different logging paths. Then I modified the configuration to enable the additional profile handler locations as follows:

ProtocolHandler Expressions

```
<ProtocolHandler implementation="edu.internet2.middleware.shibboleth.idp.provider.ShibbolethV1SSOHandler">
    <Location>https://[^:/]+(:443)?/(shibboleth|idp/profile/Shibboleth)/SSO</Location>
</ProtocolHandler>
<ProtocolHandler implementation="edu.internet2.middleware.shibboleth.idp.provider.ShibbolethV1SSOHandler">
    <Location>https://[^:/]+(:443)?/(shibboleth|idp/profile/Shibboleth)/HS</Location>
</ProtocolHandler>
<ProtocolHandler implementation="edu.internet2.middleware.shibboleth.idp.provider.SAMLv1_AttributeQueryHandler">
    <Location>.+:8443/(shibboleth/AA|idp/profile/SAML1/SOAP/AttributeQuery)</Location>
</ProtocolHandler>
<ProtocolHandler implementation="edu.internet2.middleware.shibboleth.idp.provider.Shibboleth_StatusHandler">
    <Location>https://[^:/]+(:443)?/(shibboleth|idp(/profile)?)/Status</Location>
</ProtocolHandler>
```

Technically I could have made each one distinct and only authorized the specific paths expected for that copy, but it was simpler just to keep them consistent using the regular expressions.

Tomcat Context Descriptor

I was already using a context fragment to deploy the original warfile, so I simply modified it and added a second. Each copy points to the same warfile, but one is named "shibboleth.xml" and the other "idp.xml", and they reference the appropriate configuration file as follows:

idp.xml Context Fragment

```
<Context docBase="${catalina.home}/shibboleth/webapps/shibboleth.war">

    <Parameter name="IdPConfigFile" value="file:///usr/local/shibboleth-idp/etc/prod/idp.xml" override="false"/>
...
</Context>
```

shibboleth.xml Context Fragment

```
<Context docBase="${catalina.home}/shibboleth/webapps/shibboleth.war">

    <Parameter name="IdPConfigFile" value="file:///usr/local/shibboleth-idp/etc/prod/shibboleth.xml" override="
false"/>
...
</Context>
```

This allows the single warfile to configure itself differently by pushing the context parameter in from outside it.

Code Fixes

The last set of changes were required to account for bugs or deficiencies I ran into while testing the second copy. There were two issues.

First, there was a bug in the handling of embedded links in various JSP pages served by the IdP. Since these files were already being customized by me, the bug was more a matter of customizing them correctly. The bug is described at https://issues.shibboleth.net/jira/browse/SIDPO-31. Fixing this requires making sure any content like style sheets or images that are inside the root of the warfile are referenced with a `request.getContextPath()` prefix. Examples:

Old (broken) URL references

```
<link rel="stylesheet" type="text/css" href="main.css" />
<img src="images/logo.jpg" alt="Logo" />
```

Fixed versions

```
<% String base = request.getContextPath(); %>
<link rel="stylesheet" type="text/css" href="<%= base %>/main.css" />
<img src="<%= base %>/images/logo.jpg" alt="Logo" />
```

The other issue concerned making SSO work, and was caused by the default path property associated with the cookies that the IdP was creating using the authHeaderName="COOKIE" feature. Since I wanted the SSO cookie to be visible to both copies, I needed to modify `src/edu/internet2/middleware/shibboleth/idp/provider/SSOHandler.java` and add `cookie.setPath("/")` to the `getRemoteUser` method.