Invoking C3PR Web Services — a Developer's Approach

Introduction

C3PR team has developed a number of Web services, namely Subject Management service, Study Utility service, and Subject Registry service, which are integral part of the C3PR application and are available in every C3PR deployment. Detailed description of these services' semantics and capabilities are outside of the scope of this write-up and can be found in corresponding PSM documents, such as https://ncisvn.nci.nih.gov/svn/c3pr/trunk/c3prv2/documentation/design/essn/SubjectManagement/PSM_SS_Subject_Management.doc. This write-up deals only with technical issues related to invocation of the services.

Here we will consider an example of invoking an instance of the Subject Management service. This example can be easily modified and applied to other services as well.

Target Audience

Technical.

Preparations

To go through this example, you would have the following items:

- A configured C3PR instance running against a database
- An SVN client
- Eclipse IDE
- Java 5 or above.
- soapUI

WSDL

Just as any Web service does, C3PR services start with a WSDL:

- https://<host>:<port>/<context>/services/services/SubjectManagement?wsdl
- https://<host>:<port>/<context>/services/services/StudyUtility?wsdl
- https://<host>:<port>/<context>/services/services/SubjectRegistry?wsdl
- https://<host>:<port>/<context>/services/services/SubjectRegistration?wsdl

Duplicate "/services" entries are a known issue and will be fixed in a future release.

Invoking the Services

Invoking C3PR Web services is not much different from invoking any other SOAP-based Web service, with an exception for security. We won't go into much detail here: you could inspect sample SOAP envelopes attached to this write-up or use soapUI's features to generate your own automatically.

Introduction to Services Security

C3PR Web services are secure services. Confidentiality is enforced by SSL, just like in Grid services. However, authentication approach deviates from the one used by Grid services, in which you would use a short-lived X.509 proxy certificate. C3PR Web services use SAML tokens of a predefined format to pass authentication information and then use CSM to make authorization decisions.

SAML Token Format

In short, **every** SOAP envelope must contain a SAML v1.x token, which contains authentication data, in its `Header`. High-level token format requirements are as follows:

- Token must be a valid SAML v1.0 or v1.1 token.
- `NameIdentifier` element must contain a login ID of the authenticated user. This ID will be used to look up the user in the CSM database.
- `ConfirmationMethod` should be `urn:oasis:names:tc:SAML:1.0:cm:bearer`.
- Token must be signed by issuer's certificate.
    - Instructions on adding issuer's certificate to C3PR trust-store are provided further in this tutorial.

The example below shows a valid token. It has been pretty-printed for readability purposes:

```
<saml:Assertion MajorVersion="1" MinorVersion="1"
      AssertionID="_ae7e24a5-d8ae-4274-9515-b62a7f1408be"
      Issuer="http://R0177808MFADIDM.mfad.mfroot.org/adfs/services/trust"
      IssueInstant="2010-10-05T18:33:40.953Z" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
      <saml:Conditions NotBefore="2010-10-05T18:33:40.938Z"
       NotOnOrAfter="2010-10-05T19:33:40.938Z">
       <saml:AudienceRestrictionCondition>
        <saml:Audience>http://www.sample.com</saml:Audience>
       </saml:AudienceRestrictionCondition>
      </saml:Conditions>
      <saml:AttributeStatement>
       <saml:Subject>
        <saml:NameIdentifier>JOHNT1</saml:NameIdentifier>
        <saml:SubjectConfirmation>
         <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
         </saml:ConfirmationMethod>
        </saml:SubjectConfirmation>
       </saml:Subject>
       <saml:Attribute AttributeName="windowsaccountname"
        AttributeNamespace="http://schemas.microsoft.com/ws/2008/06/identity/claims">
        <saml:AttributeValue>JOHNT1</saml:AttributeValue>
       </saml:Attribute>
      </saml:AttributeStatement>
      <saml:AuthenticationStatement
       AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
       AuthenticationInstant="2010-10-05T18:33:40.928Z">
       <saml:Subject>
        <saml:NameIdentifier>JOHNT1</saml:NameIdentifier>
        <saml:SubjectConfirmation>
         <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
         </saml:ConfirmationMethod>
        </saml:SubjectConfirmation>
       </saml:Subject>
      </saml:AuthenticationStatement>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
       <ds:SignedInfo>
        <ds:CanonicalizationMethod
         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod
         Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
        <ds:Reference URI="#_ae7e24a5-d8ae-4274-9515-b62a7f1408be">
         <ds:Transforms>
          <ds:Transform
           Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
         </ds:Transforms>
         <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
         <ds:DigestValue>0vvCAZ4mYaO2DurWClcxsYhjdrgoAuW+MTIo2MC7aio=
         </ds:DigestValue>
        </ds:Reference>
       </ds:SignedInfo>

<ds:SignatureValue>juvA2DBBjQwllMzPv5nfaLHLecB+ZNse47OM9n+XgEng0oGBcRsmKztoNeYDLEuckPEr4MtTwTnt25v2FminZ
</ds:SignatureValue>
       <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <X509Data>

<X509Certificate>MIIC+jCCAeKgAwIBAgIQLnO5KcVRlpVH6adxv5WNjTANBgkqhkiG9w0BAQsFADA5MTcwNQYDVQQDEy5BREZTIFN
</X509Certificate>
        </X509Data>
       </KeyInfo>
      </ds:Signature>
     </saml:Assertion>
```

Also, please refer to the sample SOAP envelopes at the end of this write-up.

Running Java Client

Developing a Java client capable of invoking C3PR Web services requires setting up a *Secure Token Service* (STS), which is normally a part of an Identify Provider software. The **client** code presented in this section has been tested with Sun GlassFish STS and Microsoft ADFS, but is also expected to work with any Identity Provider that supports WS-Trust v1.3 specification.

There is a handful of ways to build a client for C3PR Web services including languages other than Java. This particular example uses JAX-WS 2.1 and Apache CXF 2.2.10. You are not restricted to any particular choice of technology as long as you can satisfy the service's security requirements.

Obtaining the Code

Please do an SVN checkout from https://ncisvn.nci.nih.gov/svn/c3pr/trunk/c3prv2/codebase/projects/ws-client. This will give you an Eclipse project that you can import into your workspace. Build the project before proceeding.

Configure STS

- Download and install latest NetBeans and GlassFish
    - You can download them bundled together or separately.
- Deploy an STS service in GlassFish by following the steps described here or here
    - Use *Message Authentication over SSL* as Security Mechanism
        - Authentication Token: Username
        - WSS Version: 1.1
        - Algorithm Suite: Basic 128bit
        - Security Header Layout: Lax
        - Uncheck *Require Signature Confirmation*
    - Uncheck *Encrypt Issued Key*.
- Create a user in GlassFish by following the steps described here.
    - User ID must match the one of your C3PR user; password, of course, does not

Please check `SecureTokenServiceService.wsdl` attached to this write-up to make sure that your STS configuration matches the one used for this example. You can always find your STS detailed configuration in a similar WSDL file located within your NetBeans project.

Create C3PR User

Create a user in your C3PR instance using the same user name as in the previous step. The user must have permissions necessary to create participants.

Configure Client

Open up `/src/applicationContext.xml`. Perform the following modifications and save:

- In `jaxws:client` element, change `wsdlLocation` and `address` to point to your C3PR instance,
- Within `stsClient` bean configuration:
    - Change `wsdlLocation` to point to your STS' WSDL.
    - Change `serviceName` and `endpointName` to specify your STS' service name and port. These values can be retrieved from the STS' WSDL.
    - Change `ws-security.username` and `ws-security.password` to specify user name and password of GlassFish user that you created previously.

Add Certificates to JDK's Truststore

SSL certificates presented by Tomcat, on which C3PR is running, and GlassFish, on which STS is running, need to be added to the JDK truststore in order to bypass SSL verification. One of possible ways to do so is described below:

- Open up Subject Management service WSDL in FireFox
    - https://<host>:<port>/<context>/services/services/SubjectManagement?wsdl
- In the right bottom corner, double-click *Lock* icon, then View Certificate, Details, Export, save it in a .crt file using X.509 Certificate PEM format.
- Go to JAVA_HOME\jre\lib\security folder, and execute "keytool -import -alias glassfish -file <FILENAME>.crt -keystore cacerts" replacing alias (can be any string) and file name. In case you have multiple JDKs installed, it should be the same one as what's used to run the client.
- Repeat the steps once again for STS WSDL

Also, please make sure that host names used in C3PR and STS URLs match those in SSL certificates. SSL verification will continue to fail otherwise.

Add STS Certificate to C3PR Truststore

Every SAML token issued by STS is signed by STS's private key contained in STS certificate. This certificate may be different from SSL certificate described above. In order to establish trust relationship between STS and C3PR, STS certificate needs to be added to C3PR truststore.

STS certificate is located in <GLASSFISH_HOME>\domains\domain1\config\keystore.jks under `xws-security-server` alias. You can double-check this by looking into your STS WSDL.

Use `keytool` to:

- Extract the certificate and save it in a file,
- Import the certificate into C3PR keystore:
    - Path to the keystore must be `/local/c3pr/publicstore.jks`. On Windows, drive letter resolves to the drive with Tomcat installation.
    - Keystore password must be `c3prcerts`.

Run the Client

Execute `edu.duke.cabig.c3pr.webservice.subjectmanagement.client.Client.main(String[])`. This will list IDs of all subjects that exist in C3PR instance.

Invoking without STS

Having to maintain an STS can be inconvenient during development and testing. Often you would want to invoke services with some pre-existent SAML token using soapUI or generic Java client and without dealing with token acquisition issues. This section will briefly describe the approach.

Running soapUI

- Generate sample Web service request in soapUI, or better yet use a sample request attached to this write-up
- Place SAML token inside `<soapenv:Header>/<wsse:Security>` element
    - You can use a test token from here. If you do, then
        - Also, use this keystore as C3PR keystore
        - Create a C3PR user with login ID `jdoe01` and having necessary permissions
- Execute request as usual.
    - If using soapUI, you normally don't need to bother with SSL certificates for HTTPS.

Running a Generic Java Client

You can invoke C3PR Web services using a generic JAX-WS Java client generated from the service's WSDL. Details of creating JAX-WS client are outside of the scope of this write-up can be found, for example, here. This section will describe how to bypass Web service security when using generated JAX-WS client and a pre-existing SAML token without STS.

- Generate Java client code by using `wsimport` or your favorite IDE
- Use the following code snippet to create `SubjectManagement` at runtime (*adjust URLs, of course*):

```
private static final QName SERVICE_NAME = new QName(
   "http://enterpriseservices.nci.nih.gov/SubjectManagementService",
   "SubjectManagementService");
 private static URL endpointURL;
 private static URL wsdlLocation;

 static {
  try {
   endpointURL = new URL(
     "https://dev.semanticbits.com/c3pr/services/services/SubjectManagement");
   wsdlLocation = new URL(endpointURL.toString() + "?wsdl");
   disableSSLVerification();
  } catch (MalformedURLException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }


 private static SubjectManagement getService() {
  SubjectManagementService service = new SubjectManagementService(
    wsdlLocation, SERVICE_NAME);
  SOAPUtils.installSecurityHandler(service);
  SubjectManagement client = service.getSubjectManagement();
  return client;
 }
 private static void disableSSLVerification() {
  TrustManager[] trustAllCerts = new TrustManager[] { new X509TrustManager() {

   public java.security.cert.X509Certificate[] getAcceptedIssuers() {
    return null;
   }

   public void checkClientTrusted(
     java.security.cert.X509Certificate[] certs, String authType) {
   }

   public void checkServerTrusted(
     java.security.cert.X509Certificate[] certs, String authType) {
   }
  } };

  try {
   SSLContext sc = SSLContext.getInstance("SSL");
   sc.init(null, trustAllCerts, new java.security.SecureRandom());
   HttpsURLConnection
     .setDefaultSSLSocketFactory(sc.getSocketFactory());
  } catch (Exception e) {
   e.printStackTrace();
  }

  com.sun.net.ssl.HostnameVerifier hv = new com.sun.net.ssl.HostnameVerifier() {

   public boolean verify(String urlHostname, String certHostname) {
    return true;
   }
  };
  com.sun.net.ssl.HttpsURLConnection.setDefaultHostnameVerifier(hv);

  HostnameVerifier hv2 = new HostnameVerifier() {

   public boolean verify(String urlHostName, SSLSession session) {
    return true;
   }
  };
  HttpsURLConnection.setDefaultHostnameVerifier(hv2);

 }
```

The code of the entire class can be found here

- In order for the above snippet to compile, you will need SOAPUtils.java
- Your `SAMLToken.xml` containing a pre-existing token needs to be somewhere on your classpath
    - You will need to adjust the value of `edu.duke.cabig.c3pr.webservice.integration.SOAPUtils.PATH_TO_SAML_TOKEN` to point to your token.
    - You can use a test token from here. If you do, then
        - Also, use this keystore as C3PR keystore
        - Create a C3PR user with login ID `jdoe01` and having necessary permissions
- Invoke service operations as usual.

Attachments

Subject Management Platform-Specific Specification
Sample "Query All Subjects" SOAP envelope
STS WSDL