Newcastle University Grouper - WS Security with Rampart

WS-Security with Rampart

Overview

WS-Security which is now part of the Axis2 framework, is implemented by the WSS4J library. Rampart is an Axis2 module that implements WS-Security functionality and can easily be added to a base Axis installation. The Apache Rampart distribution can be downloaded at http://ws.apache.org/axis2/download/1_3/download.cgi

WS-Security supports 4 different actions: Timestamping, Authentication, Encryption and Signature. At the moment, we have only worked with Authentication. Authentication is based on a UsernameToken, and consists of a username and password.

Rampart can be configured via a parameter based approach or a policy based approach. However, the parameter based approach doesn't allow for much flexibility and most of the parameters have to be hardcoded on the client side. This doesn't allow parameters to be set at runtime. The policy based approach on the other hand, does enable configuration at runtime. As such, much of our work is based on the policy based configuration and this page reflects that.

Installation

Axis2 needs to be properly installed and configured and this can be checked at the Axis installation home page (http://localhost:8080/axis2/) under the Validate link. The **AXIS2_HOME** environment variable also has to point to the directory where the Axis2 distribution is installed.

The next step is to install the Rampart module. Copy **rampart-1.1.mar** and **addressing-1.1.mar** files that are part of the rampart distro download to the **LOCAL_TOMCAT_INSTALLATION/webapps/axis2/WEB-INF/modules** directory. Also copy the jar files in /rampart-1.3/lib to **LOCAL_TOMCAT_INSTALLATION/webapps/axis2/WEB-INF/lib** and restart the Axis2 web app. Rampart should now be listed on the Axis2 admin page (http://localhost:8080/axis2/axis2-admin/login, default username is 'admin' with password 'axis2') under Available Modules.

Engaging Rampart at the service

Rampart has to be engaged on both the service and client side. What this really means is that the rampart-1.3.mar and addressing-1.1.mar files have to be added to both the service and client installations. A simple EchoService is used to illiustrate the service side of Rampart. The contents of EchoService.aar is as given below:

```
+ META-INF
   - services.xml
 + org
   + apache
     + rampart
       + echo
         - EchoService.class
         - PWHandlerServer.class
```

PWHandlerServer is the callback class. In a real situation, this class might access an LDAP directory or use other methods for associating a username with a password. However, since this is a proof-of-concept, a simple callback class that returns arbitrary values is used instead.

The service expects the username and password to be sent as input from the client, which the service then validates:

```
package org.apache.rampart.echo;

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class PWHandlerServer implements CallbackHandler {

        // the username and password we expect incoming WS calls to use
        private String user = "sanjay";
        private String pwd = "wspwd";

        public void handle (Callback[] callbacks) throws IOException, UnsupportedCallbackException {
                for (int i = 0; i < callbacks.length; i++) {
                        if (callbacks[i] instanceof WSPasswordCallback) {
                                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                                System.out.println("identifier: "+pc.getIdentifer()+", usage: "+pc.getUsage());

                                if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
                                        // for passwords sent in digest mode we need to provide the password,
                                        // because the original one can't be un-digested from the message

                                        // we can throw either of the two Exception types if authentication
fails
                                        if (! user.equals(pc.getIdentifer()))
                                                throw new IOException("unknown user: "+pc.getIdentifer());

                                        // this will throw an exception if the passwords don't match
                                        pc.setPassword(pwd);

                                } else if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN_UNKNOWN) {
                                        // for passwords sent in cleartext mode we can compare passwords
directly

                                        if (! user.equals(pc.getIdentifer()))
                                                throw new IOException("unknown user: "+pc.getIdentifer());

                                        // we can throw either of the two Exception types if authentication
fails
                                        if (! pwd.equals(pc.getPassword()))
                                                throw new IOException("password incorrect for user: "+pc.
getIdentifer());
                                }
                        } else {
                                throw new UnsupportedCallbackException(callbacks[i], "Unrecognized Callback");
                        }
                }
        }
}
```

The contents of services.xml is as follows:

```
<service name="SecureGrouperService" scope="application" targetNamespace="http://webservicesSecurity.grouper.
middleware.internet2.edu/">
        <description>
        Secure Grouper Service
       </description>
        <messageReceivers>
          <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
                         class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
           <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
                         class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
       </messageReceivers>
      <schema schemaNamespace="http://webservicesSecurity.grouper.middleware.internet2.edu/xsd"/>
      <parameter name="ServiceClass">edu.internet2.middleware.grouper.webservicesSecurity.SecureGrouperWS<
/parameter>

        <module ref="rampart" />
        <module ref="addressing" />

        <wsp:Policy wsu:Id="UTOverTransport" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
                                             xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" >
                <wsp:ExactlyOne>
                  <wsp:All>
                        <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                          <wsp:Policy>
                                <sp:TransportToken>
                                  <wsp:Policy>
                                        <sp:HttpsToken RequireClientCertificate="false"/>
                                  </wsp:Policy>
                                </sp:TransportToken>
                                <sp:AlgorithmSuite>
                                  <wsp:Policy>
                                        <sp:Basic256/>
                                  </wsp:Policy>
                                </sp:AlgorithmSuite>
                                <sp:Layout>
                                  <wsp:Policy>
                                        <sp:Lax/>
                                  </wsp:Policy>
                                </sp:Layout>
                                <sp:IncludeTimestamp/>
                          </wsp:Policy>
                        </sp:TransportBinding>
                        <sp:SignedSupportingTokens xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07
/securitypolicy">
                                <wsp:Policy>
                                        <sp:UsernameToken sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07
/securitypolicy/IncludeToken/AlwaysToRecipient" />
                          </wsp:Policy>
                        </sp:SignedSupportingTokens>

                        <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
                            <ramp:passwordCallbackClass>proxy.PWHandlerServer</ramp:passwordCallbackClass>
                        </ramp:RampartConfig>

                  </wsp:All>
                </wsp:ExactlyOne>
        </wsp:Policy>

</service>
```

The significant part in the above services.xml file is:

```
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
        <ramp:passwordCallbackClass>proxy.PWHandlerServer</ramp:passwordCallbackClass>
</ramp:RampartConfig>
```

The <passwordCallbackClass> element is used to obtain the CallBackHandler implementation to access the password of the user to perform authentication. This allows us to check the internal user store of the service to find the given user and to set the user's password in the org.apache.ws.security.WSPasswordCallback instance, which is then passed into the handle() method of the CallbackHandler implementation.

Engaging Rampart at the client

The first step to securing Rampart on the client side is to add rampart-1.3.mar and addressing-1.3.mar to the client installation. To do this, a new directory, **<CLIENT_HOME>/modules** is created. **rampart-1.3.mar** and **addressing-1.3.mar** should then be copied over to this directory. A second directory, **<CLIENT_HOME>/conf** should then be created and should contain **policy.xml** as shown below:

```
<wsp:Policy wsu:Id="UTOverTransport" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
                                    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsp:ExactlyOne>
          <wsp:All>
                <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                  <wsp:Policy>
                        <sp:TransportToken>
                          <wsp:Policy>
                                <sp:HttpsToken RequireClientCertificate="false"/>
                          </wsp:Policy>
                        </sp:TransportToken>
                        <sp:AlgorithmSuite>
                          <wsp:Policy>
                                <sp:Basic256/>
                          </wsp:Policy>
                        </sp:AlgorithmSuite>
                        <sp:Layout>
                          <wsp:Policy>
                                <sp:Lax/>
                          </wsp:Policy>
                        </sp:Layout>
                        <sp:IncludeTimestamp/>
                  </wsp:Policy>
                </sp:TransportBinding>
                <sp:SignedSupportingTokens xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                        <wsp:Policy>
                                <sp:UsernameToken sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07
/securitypolicy/IncludeToken/AlwaysToRecipient" />
                  </wsp:Policy>
                </sp:SignedSupportingTokens>

                <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
                        <ramp:user>sanjay</ramp:user>
                        <ramp:passwordCallbackClass>policy.PWHandlerClient</ramp:passwordCallbackClass>
                </ramp:RampartConfig>

          </wsp:All>
        </wsp:ExactlyOne>
</wsp:Policy>
```

The significant part of the policy.xml above is:

```
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
                        <ramp:user>sanjay</ramp:user>
                        <ramp:passwordCallbackClass>policy.PWHandlerClient</ramp:passwordCallbackClass>
</ramp:RampartConfig>
```

The above code indicates that PWHandlerClient.class is the callback handler class. <ramp:user> is not used when <ramp:passwordCallbackClass> is employed. So the PWHandlerClient will always overide any value in <ramp:user>. Basically this means that the client passes on the username/password information found in PWHandlerClient.class to the service. It also specifies that a UsernameToken should be added to the SOAP header.In the case where the UsernameToken is to be added to the SOAP header the value should be **UsernameToken**. Furthermore, it tells Axis which username to use, and which class will supply the password that goes along with this username. The callback class on the client side is given below. In this instance, the client makes a request, which the callback class then finds and "fills in" the password as shown below. This client callback class should be in the same classpath as the proxy client.

```
import org.apache.ws.security.WSPasswordCallback;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import java.io.IOException;

public class PWHandlerClient implements CallbackHandler {

    public void handle (Callback[] callbacks) throws IOException, UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {
            WSPasswordCallback pwcb = (WSPasswordCallback) callbacks[i];
            String id = pwcb.getIdentifer();
            if ("sanjay".equals(id)) {
                pwcb.setPassword("wspwd");
            }
        }
    }
}
```

So in brief, the configuration on the client side should be as follows:

```
+rampart
     +conf
       - policy.xml
     +modules
       - rampart-1.3.mar
       - addressing-1.3.mar
```

The callback class

This section briefly explains about the usage of callback classes in Rampart. In the sample EchoService, there are 2 callback classes, PWHandlerServer and PWHandlerClient. The PWHandlerServer is found on the service side while PWHandlerClient is deployed on the client side. Both these classes can be grouped into one class but it would make more sense to have them separate.

Rampart implements the java.auth.CallbackHandler interface to set/retrieve the username/password of a particular user. This class might access an LDAP directory or use other methods for associating a username with a password, but thus far we have a simple callback class that returns arbitrary values as a proof-of-concept. The callback class keeps the actual service and authentication method separate. The service doesn't need to know how the user was authenticated. The callback class implements the authentication method and it is upto the developer to implement this class. This is a light-weight intergration and as such, very little code needs to be integrated into the Grouper Web Service.

From the client point of view, the callback class is used to find and "fill in" the password:

```
if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                logInfo(pc);
                // We need the password to fill in, so the usage code must
                // match the WSPasswordCallback.USERNAME_TOKEN value
                // i.e. "2"
                if (pc.getUsage() != WSPasswordCallback.USERNAME_TOKEN) {
                     throw new UnsupportedCallbackException(callbacks[i],
                        "Usage code was not USERNAME_TOKEN - value was " + pc.getUsage());
                }
                // Get the username that was sent
                String username = pc.getIdentifer();
                // Now find the password from the user store, and set it
                String password = findPassword(username);
                pc.setPassword(password);
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
                        "Unrecognized Callback");
            }
```

The significant part of the above code is:

```
String username = pc.getIdentifer();

// Now find the password from the user store, and set it
String password = findPassword(username);
pc.setPassword(password);
```

In a real situation, this class might access an LDAP directory or use other methods for associating a username with a password.

From the service point of view, the service expects a username and password, which it then validates:

```
if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                logInfo(pc);
                // We are doing authentication only, so the usage code must
                // match the WSPasswordCallback.USERNAME_TOKEN_UNKNOWN value

                // i.e. "5"
                if (pc.getUsage() !=WSPasswordCallback.USERNAME_TOKEN_UNKNOWN) {
                      throw new UnsupportedCallbackException(callbacks[i],
                               "Usage code was not USERNAME_TOKEN_UNKNOWN - value was "+ pc.getUsage());
                }
                // Get the username and password that were sent
                String username = pc.getIdentifer();
                String password = pc.getPassword();
                // Now pass them to your authentication mechanism
                authenticate(username, password);
           // throws  WSSecurityException.FAILED_AUTHENTICATION on failure
           } else {
                throw new UnsupportedCallbackException(callbacks[i],
                        "Unrecognized Callback");
           }
```

In the above code, the username and password is validated against an authentication mechanism. However, since this is a proof-of-concept scenario, simple callback classes that return arbitrary values are used instead.

Invoking the service with a client

A service call can be made using the client below:

```
import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;
import org.apache.axiom.om.impl.builder.StAXOMBuilder;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.context.ConfigurationContext;
import org.apache.axis2.context.ConfigurationContextFactory;
import org.apache.neethi.Policy;
import org.apache.neethi.PolicyEngine;
import org.apache.rampart.RampartMessageData;

import javax.xml.namespace.QName;

public class Client {
        private static EndpointReference targetEPR =
        new EndpointReference(
            "http://localhost:8080/axis2/services/PolicyEchoService");

        private static String confPath = "C:\\rampart\\policy";


    public static void main(String[] args) throws Exception {

            ConfigurationContext ctx =
                ConfigurationContextFactory.
                   createConfigurationContextFromFileSystem(confPath, null);

        ServiceClient client = new ServiceClient(ctx, null);
        Options options = new Options();
        options.setAction("urn:echo");
        options.setTo(targetEPR);
        options.setProperty(RampartMessageData.KEY_RAMPART_POLICY,  loadPolicy( confPath + "/conf/policy.xml"));
        //this username is authenticated against the PWHandlerClient class
        options.setUserName("sanjay");
        client.setOptions(options);

        client.engageModule("addressing");
        client.engageModule("rampart");

        OMElement result = client.sendReceive(getPayload("Hello World"));

        String response = result.getFirstElement().getText();
        System.out.println(response);
    }

    private static Policy loadPolicy(String xmlPath) throws Exception {
        StAXOMBuilder builder = new StAXOMBuilder(xmlPath);
        return PolicyEngine.getPolicy(builder.getDocumentElement());
    }

    private static OMElement getPayload() {
        OMFactory factory = OMAbstractFactory.getOMFactory();
        OMNamespace ns = factory.createOMNamespace("http://policy.rampart.apache.org","ns1");
        OMElement elem = factory.createOMElement("echo", ns);
        OMElement childElem = factory.createOMElement("param0", null);
        childElem.setText(value);
        elem.addChild(childElem);

        return elem;
    }

}
```

The key to the whole process is:

```
ConfigurationContext ctx =
               ConfigurationContextFactory.
                  createConfigurationContextFromFileSystem(confPath, null);
```

When calling the above method, you need to make sure you pass the correct repository path and a proper policy.xml The path to the repository has to have a sub directory called modules (which contains the rampart and addressing modules).

The username is set at client level and is then authenticated against the PWHandlerClient class (whereby the password is set to "wspwd"):

```
options.setUserName("sanjay");
```

Engaging Rampart with Grouper

To engage Rampart with Grouper, the contents of **/rampart-1.3/lib** has to be copied to ***local_grouper_installation**/grouper-ui-1.2.0/webapp/WEB-INF/lib*. **rampart-1.3.mar** and **addressing-1.3.mar** also has to be copied to ***local_grouper_installation**/grouper-ui-1.2.0/webapp/WEB-INF/modules*. The rest of the installation is similiar to example EchoService.

The code snippet below shows how a simple GrouperService that adds users to a group can obtain UsernameToken information at the service level:

```
import java.util.Set;
import java.util.Vector;

import  edu.internet2.middleware.grouper.*; // Import Grouper API
import  edu.internet2.middleware.subject.*; // Import Subject API

import org.apache.axis2.context.MessageContext;
import  org.apache.commons.logging.*;       // For logging
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSSecurityEngineResult;
import org.apache.ws.security.WSUsernameTokenPrincipal;
import org.apache.ws.security.handler.WSHandlerConstants;
import org.apache.ws.security.handler.WSHandlerResult;

public class SecureGrouperWS {

  public static String getAuth() {
       String user=null;
       String password=null;

       MessageContext msgCtx = MessageContext.getCurrentMessageContext();
       Vector results = null;
       if ((results = (Vector) msgCtx
                .getProperty(WSHandlerConstants.RECV_RESULTS)) == null) {
           System.out.println("No security results!!");
           throw new RuntimeException("No security results!!");
       } else {
           System.out.println("Number of results: " + results.size());
           for (int i = 0; i < results.size(); i++) {
               WSHandlerResult rResult = (WSHandlerResult) results.get(i);
               Vector wsSecEngineResults = rResult.getResults();

               for (int j = 0; j < wsSecEngineResults.size(); j++) {
                   WSSecurityEngineResult wser = (WSSecurityEngineResult) wsSecEngineResults.get(j);
                   if (wser.getAction() == WSConstants.UT
                           && wser.getPrincipal() != null) {

                       //Extract the principal
                       WSUsernameTokenPrincipal principal = (WSUsernameTokenPrincipal)wser.getPrincipal();

                       //Get user/pass
                       user = principal.getName();
                       password = principal.getPassword();

                   }
               }
           }
```

```
            return user;
        }

    }// end of getAuth

    public String addUserToGroup(String groupID, String userID){

                //declaring variables required for method
                Stem ncl=null;
                Subject subjectAdd=null;
                Group group=null;
                String subjectID=null;

                try {
                      subjectID = getAuth();
                      Subject subject = SubjectFinder.findById(subjectID);

                      try {
                          GrouperSession s = GrouperSession.start(subject);
                          try {

                            Stem rootStem = StemFinder.findRootStem(s);

                            try {
                                ncl = StemFinder.findByName(s, "ncl");
                            }
                            catch (StemNotFoundException eNSNF) {
                              String StemNotFound=eNSNF.getMessage();
                              return StemNotFound;
                            }

                            if (ncl != null) {
                                group = null;
                                try {
                                  group = GroupFinder.findByName(s, groupID);
                                }
                                catch (GroupNotFoundException eGNF) {
                                        return (groupID + " does not exist");
                                }

                                  try {
                                          subjectAdd = SubjectFinder.findById(userID);
                                          group.addMember(subjectAdd);
                                          //System.out.println("Added user=" + subjectAdd.getName() + " to
group="+group.getName());
                                  }
                                  catch (MemberAddException eMA) {
                                    // unable to add member
                                          return(userID + " is already a member of " + groupID);
                                  }
                                  catch (InsufficientPrivilegeException eIP) {
                                    // Not privileged to add group
                                    return("You do not have sufficient privileges for this operation");
                                  }
                              }//end of if(ncl!=null)


                            s.stop();
                          }
                          catch (SessionException eStop) {
                            // Error stopping session
                                  return ("Error stopping Grouper session");
                          }
                        }
                        catch (SessionException eStart) {
                          // Error starting session
                                return ("Error starting Grouper session");
                        }

                    }
                    catch (SubjectNotFoundException   eSNF) {
```

```
                    // No matching subject id found
                          return (subjectID + " was not found");
                    }
                    catch (SubjectNotUniqueException  eSNU) {
                      // More than one subject with this subject id was found
                          return (subjectID + " is not unique");
                    }

                  return(userID + " has been added to " + groupID);
          }//end of addUser
}
```

The results of the security processing can be obtained at any stage of the execution flow. Rampart stores this result in the WSHandlerConstants. RECV_RESULTS key. The WSHandlerResult vector holds the security processing results of a "Security" header of a specific user.

See Also

Newcastle University Intro Page
