Two Way SSL With Web Services

Using JAAS and Jboss to provide confidentiality, authentication and access control.

Scott Streit
scott@thestreits.com
RABA Technologies, LLC

Introduction

- Rationale
- The "How To"
- Code Examples
- Summary

Discussion

- The Reasons Why
- What changes in Jboss xml files?
- Small amount of code/definitional constructs
- Conclusion

Architecture

Jboss 4.0.3 as the container for Web and Enterprise Beans.

Discussion

- The project will run on any, web browser, web server, EJB server
- The smallest configuration changes required to move from EJB server to EJB Server.
- Jboss is the host and target EJB Server.

Rationale

Design Objectives

According to W3C there are six important security considerations for a comprehensive security framework:

- Authentication,
- Authorization,
- Confidentiality,
- Integrity,
- Non-repudiation, and
- Accessibility.

We consider unique solutions for authentication, authorization and confidentiality

Rationale

Design Objectives

The objectives for the data persistence design include the following:

1. maximize the use of definitional constructs while minimizing the amount of written software. This specifically refers to using the EJB 2.1 specification.
2. maximize the portability across all platforms while minimizing software changes. This specifically refers to the ability to change any product (browser, web server, EJB server, RDBMS) without changing written software.
3. maximize the use of generalized constructs that are typical of EJB development.
4. The same JAAS/Login module is used for 2 way SSL irrespective of whether the access is from a browser or web services.

Rationale

Page Counter Example

A simple page counter application illustrates:

- **one and two-way Secure Socket Layers (SSL), Web Services**
- **Enterprise Java Beans (EJB).**

The page counter may appear on any web page as a jpg image through a servlet, as an enterprise bean, as a Java Server Page (jsp), and as a web service.

How To

Traditional HTTP Security

Traditionally we secured http access by using the following:

- HTTP basic authorization
- HTTPS (HTTP Secure) or HTTP with secure socket layer (SSL)
- HTTP authorization + HTTPS

The question to consider is if we use Enterprise Beans for our business logic, how do we secure the accesses at the HTTP and EJB layers.

How To 7

Traditional HTTP Authorization

- **HTTP provides authorization (BASIC-AUTH, CLIENT_CERT) as simple mechanism defined in the web.xml file for a web application.**
- **Using this mechanism we can protect Web resources from unauthorized access. To access resources protected using HTTP BASIC-AUTH, a user has to provide a username and password.**
- **To access resources protected using HTTP CLIENT_CERT the browser provides a certificate.**

How To 8

EXAMPLE 1 : A sample web.xml with no security

```
<!DOCTYPE web-app PUBLIC
   "-//Sun Microsystems, Inc.//DTD Web
  Application 2.3//EN"
   "http://java.sun.com/dtd/web-
  app_2_3.dtd">
<web-app>
  <display-name>Breit</display-name>
  <description>Breit</description>
  <ejb-ref>
    <ejb-ref-
  name>ejb/CountMgrEJB</ejb-ref-
  name>
    <ejb-ref-type>Session</ejb-ref-type>

    <home>breit.ejb.client.CountMgrHom
  e</home>
```

How To 9

EXAMPLE 2 : Sample web.xml with Basic Authentication

```
…
  <security-constraint>
    <web-resource-collection>
        <web-resource-name>breit</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>breit:administrator</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Tomcat Supported Realm</realm-name>
  </login-config>
  <security-role>
    <role-name>breit:administrator</role-name>
  </security-role>
…
```

How To 10

EXAMPLE 3. : Sample web.xml - Client Certificate

```
<servlet>
 <servlet-name>CountServlet</servlet-name>
 <servlet-class>breit.web.count.CountServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>CountServlet</servlet-name>
 <url-pattern>/count/Count.jpg</url-pattern>
</servlet-mapping>
 <security-constraint>
    <web-resource-collection>
        <web-resource-name>breit</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>breit:administrator</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>Tomcat Supported Realm</realm-name>
  </login-config>
  <security-role>
    <role-name>breit:administrator</role-name>
  </security-role>
```

How To 11

Traditional HTTP Authentication: HTTPS

Secure socket layer (SSL) is a technology which allows Web browsers and Web servers to communicate over a secured connection.

This means that the data being sent are encrypted by one side, transmitted, and decrypted by the other side before processing. This is a two-way process, meaning that both the server and the browser encrypt all traffic before sending out data.

The trusted store provides the handshaking between the browser and the http server allowing encrypted communication.

How To 12

EXAMPLE 4 : A partial server.xml file

```
<Connector port="8443" address="${jboss.bind.address}"
  maxThreads="100" strategy="ms"
  maxHttpHeaderSize="8192" emptySessionPath="true"
  scheme="https" secure="true" clientAuth="false"
  sslProtocol="TLS"
  keystoreFile="${user.home}/pkg/certs/server.keystore"
  keystorePass="changeit"
  truststoreFile="${user.home}/pkg/certs/trusted.keystore"
  truststorePass="changeit" />

<Realm
  className="org.jboss.web.tomcat.security.JBossSecurityMgr
  Realm"
  certificatePrincipal="breit.util.SubjectPKICNMapping"
  verbosityLevel="WARNING"
  category="org.jboss.web.localhost.Engine" />

<Valve
  className="org.apache.catalina.authenticator.SingleSignOn"
  />
```


Securing Enterprise Java Beans (EJBs)

- **To secure an Enterprise Bean, we would like to leverage the roles used at the web layer and provide the same type of role based security just at the EJB method.**
- **Let us take our CountMgr bean and define security restrictions though the ejb-jar.xml file. We add additional detail in our product specific jboss.xml file.**

EXAMPLE 5 : The ejb-jar.xml File

```
- <security-role>
- <description>
- <![CDATA[An Administrator]]>
  </description>
  <role-name>breit:administrator</role-name>
  </security-role>
- <method-permission>
- <description>
- <![CDATA[Methods accessed by a breit:administrator ]]>
  </description>
  <role-name>breit:administrator</role-name>
- <method>
- <description>
- <![CDATA[Count Manager access allowing all methods]]>
  </description>
  <ejb-name>CountMgr</ejb-name>
  <method-name>*</method-name>
  </method>
  </method-permission>
```


EXAMPLE 6 : The jboss.xml File

```
<security-domain>java:/jaas/cert</security-domain>
- <session>
  <ejb-name>CountMgr</ejb-name>
  <jndi-name>ejb/CountMgrEJB</jndi-name>
  <local-jndi-name>ejb/CountMgrEJBLocal</local-jndi-name>
- <port-component>
  <port-component-name>CountMgrBeanEndpointPort</port-
    component-name>
  <port-component-
    uri>breitEJB/CountMgrBeanEndpointPort</port-component-
    uri>
  <auth-method>CLIENT-CERT</auth-method>
  <transport-guarantee>CONFIDENTIAL</transport-
    guarantee>
  </port-component>
  <method-attributes />
  </session>
```


EXAMPLE 7 : Partial jboss-service.xml file

```
<mbean code="org.jboss.security.plugins.JaasSecurityDomain"
    name="jboss.security:service=JaasSecurityDomain">
- <constructor>
  <arg type="java.lang.String" value="cert" />
  </constructor>
  <attribute
    name="KeyStoreURL">${user.home}/pkg/certs/server.keysto
    re</attribute>
  <attribute name="KeyStorePass">changeit</attribute>
  <attribute
    name="TrustStoreURL">${user.home}/pkg/certs/trusted.key
    store</attribute>
  <attribute name="TrustStorePass">changeit</attribute>
  </mbean>
```


EXAMPLE 8 : Partial login-config.xml File

```
- <application-policy name="cert">
- <authentication>
- <login-module code="breit.util.DBCertLoginModule"
    flag="required">
  <module-option name="securityDomain">java:/jaas/cert</module-
    option>
  <module-option
    name="verifier">org.jboss.security.auth.certs.AnyCertVerifier</mod
    ule-option>
  <module-option name="dsJndiName">java:/security</module-option>
  <module-option name="rolesQuery">SELECT RoleTable.application,
    RoleTable.name FROM RoleTable, UserRoleTable, UserTable
    WHERE UserTable.userId = UserRoleTable.userId AND
    RoleTable.roleId = UserRoleTable.roleId AND UserTable.login =
    ?</module-option>
  <module-option
    name="unauthenticatedRole">anonymousRole</module-option>
</login-module>
</authentication>
</application-policy>
```

EXAMPLE 9 : The Login Module

```
public class DBCertLoginModule extends DatabaseCertLoginModule
{
    /** Logger for the class */
    private static Log log = LogFactory.getLog(DBCertLoginModule.class);
    private String dsJndiName;

    private String unauthenticatedRole = null;

    private String rolesQuery = null;
    //------------------------ RealmBase abstract methods
    public void initialize(Subject subject,
                           CallbackHandler callbackHandler,
                           Map sharedState, Map options)
    {
        super.initialize(subject, callbackHandler, sharedState, options);
        dsJndiName = (String) options.get("dsJndiName");
        if (dsJndiName == null) {
            dsJndiName = "java:/DefaultDS";
        }
        Object tmp = options.get("rolesQuery");
        if (tmp != null) {
            rolesQuery = tmp.toString();
        }
```


EXAMPLE 9 : The Login Module (continued)

```
            unauthenticatedRole = (String) options.get("unauthenticatedRole");
    }
    protected Group[] getRoleSets() throws LoginException
    {
            Group[] groups = { new SimpleGroup("Roles") };
            Connection conn = null;
            HashMap setsMap = new HashMap();
            PreparedStatement ps = null;
            ResultSet rs = null;
            boolean hasMembers = false;
            try {
                InitialContext ctx = new InitialContext();
                DataSource ds = (DataSource) ctx.lookup(dsJndiName);
                conn = ds.getConnection();
                // Get the user role names
                ps = conn.prepareStatement(rolesQuery);
                ps.setString(1, getIdentity().getName());
                rs = ps.executeQuery();
                while (rs.next()) {
                    String application = rs.getString(1);
                    String name = rs.getString(2);
                    String role = application + ":" + name;
                    Principal principal = new SimplePrincipal(role);
                    groups[0].addMember(principal);
                    hasMembers = true;
                }
```


EXAMPLE 9 : The Login Module (concluded)

```
    } catch (NamingException ex) {
            throw new LoginException(ex.toString(true));
        } catch (SQLException ex) {
            throw new LoginException(ex.toString());
        } finally {
            if (rs != null) {
                try {
                  rs.close();
                } }
            if (ps != null) {
                try {
                  ps.close();
                } }
            if (conn != null) {
                try {
                  conn.close();
                } }
        }

        if (!hasMembers) {
            Principal principal = new SimplePrincipal(unauthenticatedRole);
            tempGroups[0].addMember(principal);

        }
        return groups;
    } }
```


EXAMPLE 10 : The Custom Principal

```
/**
 * A CertificatePrincipal implementation that parses the client cert
 * SubjectDN CN='...' element for PKI format and returns the SID.
 */
public class SubjectPKICNMapping implements CertificatePrincipal
{
    /** Logger for the class */
    private static Log log = LogFactory.getLog(SubjectPKICNMapping.class);

    /** Returns the SID as the principal. The CN is assumed to be in PKI format.
     *
     * @param certs Array of client certificates, with the first one in
     * the array being the certificate of the client itself.
     */
    public Principal toPrinicipal(X509Certificate[] certs)
    {
        Principal cn = null;
        Principal subject = certs[0].getSubjectDN();
        String sid = parseDN(subject.getName());
        if (sid != null) {
            cn = new SimplePrincipal(sid);
        }
        else {
            // Fallback to the DN
            cn = subject;
        }
        log.info("CN mapped to " + cn);

        return cn;
    }
```


Setting Up the Web Service

The key component in setting up the web service is calling wscompile from the java web services developer kit.

Prior to calling wscompile we must have created a service endpoint for each bean requiring web service access.

EXAMPLE 11 : The CountMgr endpoint

```
/*
 * Generated by XDoclet - Do not edit!
 */
package breit.ejb.client.count;
/**
 * Service endpoint interface for CountMgr.
 * @xdoclet-generated at January 20 2006
 */
public interface CountMgrBeanEndpoint
  extends java.rmi.Remote
{
  public breit.ejb.client.count.CountData getCount( java.lang.String page )
    throws java.rmi.RemoteException;
  public java.lang.String getCountAsString( java.lang.String page )
    throws java.rmi.RemoteException;
  public breit.ejb.client.count.CountData increase( java.lang.String page )
    throws java.rmi.RemoteException;
  public java.lang.String[] getArray( )
    throws java.rmi.RemoteException;
}
```

EXAMPLE 12 : Partial wscompile ANT Task

```
-

 <wscompile fork="true" nonclassdir="${gen-etc.dir}/wsdl"
    base="${ejb.classes}" features="rpcliteral" server="true"
    config="mywscompile-config.xml" mapping="${gen-
    etc.dir}/jaxrpc-mapping.xml" verbose="true">

 -    <classpath>
              <path refid="jboss.classpath" />
              <path refid="ws.client.class.path" />
      </classpath>
    </wscompile>
```


EXAMPLE 13 : The wscompile Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
- <configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
- <service name="CountMgrService"
              targetNamespace="http://count.client.ejb.breit"
              typeNamespace="http://count.client.ejb.breit"
              packageName="breit.ejb.client.count">
  <interface name="breit.ejb.client.count.CountMgrBeanEndpoint"
              servantName="breit.ejb.server.count.CountMgrBean" />
  </service>
  </configuration>
```

- The wscompile step creates a wsdl file as well as the jaxrpc-mapping.xml file. It has a dependency on a config file.
- Note in the configuration file the reference to the endpoint as well as the enterprise bean (CountMgrBean).
- Once the web service is jarred it is deployed as a functioning web service.

EXAMPLE 14 : One-way SSL Execution

```
java -classpath .:$CP -
    Djavax.net.ssl.trustStore=/home/scott/pkg/cert
    s/trusted.keystore \
     Djavax.net.ssl.trustStorePassword=changeit \
breit.test.webservice.CountTest2
    'https://localhost:8443/breitEJB/CountMgrBea
    n2EndpointPort?wsdl'
    "http://count.client.ejb.breit" testArray
```


EXAMPLE 15 : Two way ssl Execution

```
java -classpath .:$CP -
Djavax.net.ssl.keyStore=/home/scott/pkg/certs/scott.p12 \
      -Djavax.net.ssl.keyStorePassword=changeit \
      -Djavax.net.ssl.keyStoreType=pkcs12 \
      -Djavax.net.ssl.trustStore=/home/scott/pkg/certs/trusted.keystor
\
      -Djavax.net.ssl.trustStorePassword=changeit
breit.test.webservice.CountTest \
         'https://localhost:8443/breitEJB/CountMgrBeanEndpointPort
wsdl' "http://count.client.ejb.breit
```


EXAMPLE 16 : Partial Test Program

```
protected void setUp() throws Exception {
                super.setUp();

                QName qname = new QName(accessPoint,
                                        "CountMgrService");
        ServiceFactoryImpl factory = (ServiceFactoryImpl) ServiceFactory.newInstance();
      URL url = new URL(wsdlURL);
      URL jaxrpcURL = new URL(JAXRPC);
                Service service = factory.createService(url,jaxrpcURL,null, qname,null);
                endpoint = (CountMgrBeanEndpoint) service
                                        .getPort(CountMgrBeanEndpoint.class);

        }
```

Upon creating the endpoint, all accesses occur as if the caller was using a remote.

Summary

- Using JAAS and the Jboss login modules allow the creation of one class for authentication irrespective of the client.
- Browsers and Web Services communicate through https.
- J2EE 1.4 solution.
- Code available at http://scott.thestreits.com