[转]Web Services Over SSL - HOW TO

atojaoomy 于 2012-05-07 15:31:54 发布 67 收藏
分类专栏： WebService SSL Java 文章标签： java 数据库 网络

WebService 同时被 3 个专栏收录
6 篇文章 0 订阅
订阅专栏

SSL
2 篇文章 0 订阅
订阅专栏

Java
20 篇文章 0 订阅
订阅专栏

一篇好文章，原文地址：http://www.pankaj-k.net/WSOverSSL/WSOverSSL-HOWTO.html

顺手贴过来，找起来方便

Summary
This HOWTO guide explains steps involved in (i) deploying SSL accessible Java Web Services in Servlet based SOAP platforms such as Apache Axis or HP Web Services Platform (HP-WSP); and (ii) running Java client programs that access these Web Services with **https** protocol. The guide accompanies scripts and sources for a simple application consisting of a simple Web Service (Hello World variety) and a client to illustrate the setup steps.

By Pankaj Kumar. Revision: 0.81, Date: Apr. 2, 2002.

Sep. 12, 2003: This writeup is being obsoleted by Web Services Security, the freely downloadable sample chapter of my recently published book J2EE Security for Servlets, EJBs and Web Services.

Introduction

This guide is for you if

1. **you are working with Apache Axis or HP-WSP or any other Servlet based SOAP platform and need to make your Web Service accessible over SSL to all or only certain clients using certificate based client authentication; and/or**
2. **you need to run a Java client program that invokes a Web Service with https URL using SSL.**

Apache Axis is an opensource software and can be downloaded from http://xml.apache.org/axis. Evaluation copy of HP-WSP can be downloaded from http://www.hpmiddleware.com. This guide and the supplied example scripts and sources are based on Apache Axis Alpha3 and March 2002 release of HP-WSP.

A Servlet based SOAP platform is usually not aware of SSL. It is the Servlet container who takes care of all the encryption, decryption and verification stuff. In this respect, use of SSL is transparent to the SOAP platform or the Web Service. So, what is needed is the steps to setup the servlet container for SSL. In addition, the WSDL description of the Web Service must specify the appropriate URL to access the service.

This guide includes the steps for Apache Jakarta Tomcat 4.0.1 and HP-AS 8.0, a full-fledged J2EE AppServer free for commercial use. You can download HP-AS 8.0 from http://www.hpmiddleware.com. It is noteworthy that HP-AS 8.0 supports certificate based client authentication whereas my preliminary investigation on Tomcat 4.0.1 leads me to beleive that Tomcat doesn't support this capability. However, I don't track Tomcat development very closely and I could be wrong here. Let me know if I this is so.

Certain steps mentioned in this guide require the Java source files and scripts available in the caompanion resource `WSOverSSL.zip`. Download WSOverSSL.zip and unzip it.

This would create the directory with name `WSOverSSL` and place the contents within this directory. As the scripts to compile, deploy and run the client are SOAP Platform specific and there are minot differences in the client source code as well, I have kept sources for Apache Axis and HP-WSP in separate sub-directories with names `axis` and `hpwsp`.

I have written only Windows scripts to compile, deploy and run the examples only. Translation to Linux/Unix scripts are straightforward. A future revision of this document will use OS neutral Apache Jakarta Ant scripts. File and directory pathnames and environment variable used in this guide follow the Windows convention. Again, translation to Linux/Unix names are trivial.

Environment variable `JAVA_HOME` is used in all the scripts for easy switch among different JDKs, either from one or different vendors.

Rest of the guide is organised in following sections:

- **Install Java Secure Socket Extension (JSSE) package. [not required for JDK1.4 or above]**
- **Generate certificates and keystores.**
- **Configure Servlet Container for SSL:**
  - **Apache Jakarta Tomcat 4.0.1**
  - **HP-AS 8.0**
- **Running the Example Program**
- **Troubleshooting Tips**
- **Known Problems**
- **Further References**

Install JSSE

You can skip this if you using JDK1.4 or above as JSSE is now part of the JDK.

I tried the steps on my WIndows NT machine with Sun's JDK1.3. It sould work on other OS platforms and JDK1.2.x and 1.3.x with no or minor changes.

Note that you must have JSSE, either as part of JDK1.4 or separately installed, to be able to setup Apache Jakarta TomCat or HP-AS 8.0 for SSL. It is also needed for running the client program.

1. **Download JSSE package. You can get it from http://java.sun.com/products/jsse/. Be prepared to setup an**

account at Sun's Download Center (if you don't have one).

2. **Install JSSE. In most basic form, it involves unzipping the downloaded file and copying .jar files `jsse.jar`, `jcert.jar` and `jnet.jar` to your `%JAVA_HOME%\jre\lib\ext` directory and editing the `%JAVA_HOME%\jre\lib\security\java.security` file to add the following line:**

   ```
   security.provider.3=com.sun.net.ssl.internal.ssl.Provider
   ```

   Refer to the documentation of JSSE for more details. But I must warn you that the documentation is quite voluminous and may not always match with your existing platform. For example, it mentions copying the .jar files into `%JAVA_HOME%\lib\ext`, but that didn't work for me. Later on, I figured that Sun's documentation for JSSE uses `JAVA_HOME` to point to `jre` sub-directory of JDK installation directory whereas I have used JAVA_HOME to point to JDK Installation directory itself.

3. **Test JSSE installation. Go to sub-directory `axis` or `hpwsp` and edit the script `testhttps.bat` for your environment. This script invokes the `URLReader` program (part of the download) to access url `https://www.etrade.com` with proper options to the JVM. If you are behind a corporate firewall, you should edit the script file to use appropriate values for System properties `https.proxyHost` and `https.proxyPort`. Following worked fine for me:**

   ```
   %java_home%\bin\java -Dhttps.proxyHost=web-proxy -Dhttps.proxyPort=8088 \
   -Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol \
   URLReader https://www.etrade.com
   ```

   Note that character `\` is used to indicate line continuation and must not appear in the actual command.

   Pay attention to these System properties and understand their significance. A proper understanding now will save you a lot of grief later on.

   If the script `testhttps.bat` doesn't work, recheck the steps in **Installing JSSE**. If you are not using environment variable `%JAVA_HOME%`, you may want to verify that the *right* JVM (the one whose `jre\lib` sub-directory has been updated) is being used. One way to do this is to display System property `java.ext.dirs`. This should have `%JAVA_HOME%\jre\lib\ext` as a component.

Note that there are other setups possible. For example, you could keep JSSE jar files in a separate directory and have them in your `CLASSPATH`.

Also, you could add the SSL Provider to the `java.lang.Security` class within your program and not specify in the commandline.

Refer to JSSE documentation for more details.

Generate Certificates and KeyStores

In this section, we will use the `keytool` utility of JDK to generate keys, *certificates*, *keystores* and *truststores*. Though you don't need to understand the underlying concepts for setting up your system for this example, but if you run into problems or want to modify the setup for your own specific needs, I would advise going through the references at the end of this guide or any other source to understand the functioning of `keytool` utility and its relationship to certificates, public and private keys, keystores, truststores and other such topics.

Take a look at the script file `genks.bat`:

```
if not "%JAVA_HOME%" == "" goto gotJavaHome
echo You must set JAVA_HOME to point at your Java Development Kit installation
goto cleanup
:gotJavaHome

echo Generating the Server KeyStore in file server.keystore
%java_home%\bin\keytool -genkey -alias tomcat-sv \
-dname "CN=localhost, OU=X, O=Y, L=Z, S=XY, C=YZ" \
-keyalg RSA -keypass changeit -storepass changeit -keystore server.keystore

echo Exporting the certificate from keystore to an external file server.cer
%java_home%\bin\keytool -export -alias tomcat-sv -storepass changeit \
-file server.cer -keystore server.keystore

echo Generating the Client KeyStore in file client.keystore
%java_home%\bin\keytool -genkey -alias tomcat-cl \
-dname "CN=Client, OU=X, O=Y, L=Z, S=XY, C=YZ" \
-keyalg RSA -keypass changeit -storepass changeit -keystore client.keystore

echo Exporting the certificate from keystore to external file client.cer
%java_home%\bin\keytool -export -alias tomcat-cl -storepass changeit \
-file client.cer -keystore client.keystore

echo Importing Client's certificate into Server's keystore
%java_home%\bin\keytool -import -v -trustcacerts -alias tomcat -file \
server.cer -keystore client.keystore -keypass changeit -storepass changeit

echo Importing Server's certificate into Client's keystore
%java_home%\bin\keytool -import -v -trustcacerts -alias tomcat -file \
client.cer -keystore server.keystore -keypass changeit -storepass changeit

:cleanup
```

Note that the character `\` is used to indicate continuation of the same line. You won't find this in the actual script file.

As the comments indicate, this script creates a self signed cerficate for the server and the client and stores them in keystores `server.keystore` and `client.keystore`, respectively. Also, the server's certificate is *exported* to file `server.cer` and *imported* to client's keystore `client.keystore`. Similarly, the client's certificate is *exported* to file `client.cer` and *imported* to server's keystore `server.keystore`. This way, keystore `client.keystore` can also be used by the client as the truststore to verify the cetificate supplied by the server and vice-versa.

With respect to the script `genks.bat`, it is important to note that:

1. **The servaer name specified in the server's certificate is `localhost`. If you plan to access the server with its symbolic name or IP address on the network, you must specify that string while generating the key. You can also have multiple certificates for the same physical machine.**
2. **Though this script is okay for running the example, a real production deployment must consider:**
   1. **Use of proper *distinguished name* or `dname` values.**
   2. **Getting certificates signed by an authorised CA (Certificate Authority).**
   3. **Not specifying the password in the script file. If the passowrd is not specified in the commandline then you will be prompted for it.**
   4. **Use of a production grade tool to manage client certificates, especially if the server wants to authenticate the client based on the supplied certificate.**

Note that in a typical use of SSL to access a web application, the client authenticates the server based on the certificate presented but the server rarely authenticates the client based on certificate. That is how we use https protocol from our browsers most of the time. This works fine as there are other mechanisms to identify and authenticate the client such as username and password, credit card no. and the billing address and so on. However, these mechansims will not be availalble to a Web Services client and hence it makes sense to use certificate based client authentication.

Configure Servlet Container for SSL

Configure Tomcat 4.0.x for SSL

This configuration is really simple. Edit the %TOMCAT_HOME%\conf\server.xml file and uncomment the entry for SSL connector. Also add the attributes for keystore filename and password. Find below the edited entry for my setup:

```
<!-- Define an SSL HTTP/1.1 Connector on port 8443 -->
<Connector className="org.apache.catalina.connector.http.HttpConnector"
           port="8443" minProcessors="5" maxProcessors="75"
           enableLookups="true"
    acceptCount="10" debug="0" scheme="https" secure="true"<
  <Factory className="org.apache.catalina.net.SSLServerSocketFactory"
           keystoreFile="c:\progs\java\WSOverSSL\hpwsp\server.keystore"
           keystorePass="changeit"
           clientAuth="false" protocol="TLS"/<
</Connector>
```

The bold lines are specific to my setup. You may need to change these values for your setup.

To test the new configuration, restart the container and access the URL `https://localhost:8443` from your browser. It might warn you that the certificate presented by the server is not recognised. We did not get our certificates signed by a well-known CA, did we? However, keep pressing 'Yes' to the questions and eventually you should see the welcome page for Tomcat.

Tomcat 4.0.1 doesn't seem to support client authentication. I did not find any attribute of element `Factory` to specify the truststore and its password. A truststore is needed to verify the certificate presented by a client. As I have the client's certificate exported to the same keystore file, I tried just setting the attribute `clientAuth` to `true`. But this didn't work.

However, you can supply the truststore and its password as System properties to the JVM running the Tomcat and that would enable client authentication. Though I did not try this myself, a couple of folks have pointed to me that it does work.

We will see in the next section that HP-AS 8.0 supports specification of truststores and SSL based client authentication.

Configure HP-AS 8.0 for SSL

This section assumes that you have HP-AS 8.0 installed in directory `%HPAS_HOME%`. If you have selected default options while installing HP-AS, its value should be `c:\hpmw\hpas` If you don't have HP-AS 8.0, you can get it from http://www.hpmiddleware.com/.

1. **Create HTTPS listener configuration file. Go to directory `%HPAS_HOME%\config\hpas` and copy file `https-service-config-example.xml` to `https-service-config.xml`. Now edit this file to reflect your setup. Find below the edited file for my setup:**

```
<listener-service
   ports="9443"
   backlog="50"
   max-request-handlers="20"
   max-request-contexts="20"
   max-response-contexts="20"
   request-handler=
     "com.hp.mwlabs.as.services.listeners.https.HttpsRequestHandler"
   container="servlet" socket-timeout="PT0S"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation=
     "http://www.hp.bluestone.com/xml/schemas/listener-service-2_0.xsd">
   <server-socket-factory
        class-name="com.hp.mwlabs.as.services.listeners.JsseSslServerSocketFactory">
      <property name="remoteTrustAlgorithm" value="SunX509"/>
      <property name="keyStorePassword" value="changeit"/>
      <property name="protocolVersion" value="TLS"/>
      <property name="keyStoreFile"
        value="file:///c:/progs/java/WSOverSSL/hpwsp/server.keystore"/>
      <property name="enableSessionCreation" value="true"/>
      <!-- rarely will this be false -->
      <property name="keyType" value="SunX509"/>
      <property name="keyStoreType" value="JKS"/>
      <property name="keyPass" value="changeit"/>
      <property name="requireClientAuth" value="true"/>
      <property name="useKeyStorePassAsKeyPass" value="false"/>
      <property name="trustStoreFile"
        value="file:///c:/progs/java/WSOverSSL/hpwsp/server.keystore"/>
      <property name="trustStorePassword" value="changeit"/>
   </server-socket-factory>
</listener-service>
```

Lines that changed are in bold. Note that this configuration file requires that the clients also present certificates. Also, in my installation of HP-AS, the original file `https-service-config-example.xml` had XML comments at the end of the file. Delete these lines. I had to go through tech. support to get this right.

2. **Add HTTPS listener to the main deployment file. Edit file `hpas-deploy.xml` to add an entry for HTTPS listener. I added this entry just after the entry for HTTP listener:**

```
<Service name="https"
  class="com.hp.mwlabs.as.services.listeners.ListenerService">
 <Configuration url="https-service-config.xml" reload="30"/>
</Service>
```

Note that this entry is very similar to the entry for HTTP.

3. **Test the Configuration. Launch the HP-AS console. Point your browser to the URL `http://localhost:9443`. It should take you the HPAS 8.0 start page.**

Running the Example Program

The supplied example demonstrates the use of HTTP over SSL (or https) for communication between two Web Service end points. Use of https protects the content exchanged and can also be used by one end point to authenticate the other.

The example consists of a simple Java class `Hello` with method `String greet(String anme)` and a client class `client.Client`. Class `Hello` is deployed as a Web Service and the client invokes the method `greet()` with the string specified in the command line parameter.

Instructions to run the example program and troubleshooting tips can be found in readme.html file stored with example sources and scripts, in sub-directory `axis` for Apache Axis example and in sub-directory `hpwsp` for HP-WSP.

Troubleshooting Tips

Here are some generic troubleshooting tips.

1. **There is a significant setup delay for https communication. If you do not see any response immediately after running your program, don't panic. Patience pays off while working with encryption and communication over SSL.**
2. **Try to isolate the problem. Test your configuration for correct JSSE installation. Test TomCat ot HP-AS for correct HTTPS listener configuration. Test correct input key generation, certificate import and export. Check the `setHPWSEnv.bat` file for values reflecting your setup. Check and recheck the spelling of values. Once you have identified the problem, look at the appropriate resource for troubleshooting tips.**

Look at the platform specific readme.html for more specific troubleshooting tips.

Known Problems

I have observed following problems during my experimentation. These may be specific to the version of the software I am using.

1. **Axis alpha3 client doesn't work when the service to be accessed through https is outside a corporate firwall. This happens even when I correctly specified the System properties `https.proxyHost` and `https.proxyPort`. The exact error message I got was: `Unable to tunnel through web-proxy:8088. Proxy returns "HTTP/1.0 503 Service Unavailable`. Utility `Wsdl2java` retrieved the WSDL file correctly from https url outside the corporate firewall but the client program failed. This is possible as `Wsdl2java` uses the JDK classes to access the URL whereas Axis has its own implementation of HTTP over TCP/IP.**
2. **When I run a program that invokes the same method on the same service in a tight loop over HTTPS, then sometimes the client throws the following exception: `BindException: Address in use: connect`. I have never encountered this over HTTP. Also, I didn't observe this on my slower 350MHz Pentium machine, but can reproduce very quickly on faster 900MHz Athlon machine.**

References

1. **Inside Java 2 Platform Security by Li Gong. *The Java Series* book published by ADDISON-WESLEY.**
2. **Java Security, 2nd Edition by Scott Oaks. O'REILLY.**
3. **Use HTTPS in your Java client code Java Tip 96 from JavaWorld.**
4. **Setting up Apache Tomcat and a Simple Apache SOAP Client for SSL Communication**
5. **http://docs.oracle.com/javase/6/docs/technotes/guides/security/jgss/tutorials/index.html**
6. **http://docs.oracle.com/javase/6/docs/technotes/guides/security/sasl/sasl-refguide.html**
7. **http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html**