SSL

Secure Socket Layer (SSL) is a protocol that provides security for communications between client and server by implementing encrypted data and certificate-based authentication. Technically, the term "SSL" now refers to the Transport Layer Security (TLS) protocol, which is based on the original SSL specification.

1. How can we securely transmit data between two parties in such a way that only the two parties can read it?

*Answer*: Encryption. Before transmitting any data, the sender encrypts its message, and the receiver must in turn decrypt the message before processing it. The encryption and decryption is accomplished through a method called "public key encryption".

1. How can one (or more) of the parties involved prove that they are actually the entity we want to grant the ability to decrypt our encrypted transmission?

*Answer.* In order for public key encryption to provide secure communication, one or more of the communicating parties must have some way of proving to the other that they are, in fact, who they claim to be. SSL provides this proof by requiring that one or more of the parties present a digital certificate into the initial negotiation of the connection, prior to the transmission of any encrypted data. This process is called "handshaking".

Configuring Tomcat to Use SSL

### **PART I - The Keystore**

### **Step 1 - Creating the Keystore**

The keys Tomcat will use for SSL transactions are stored in a password-protected file called, creatively, the "Keystore". The first step to enabling SSL on your server is to create and edit this file. You can create this file in one of two ways - by importing an existing key into the Keystore, or by creating an entirely new key.

A program called keytool, which is included with your JDK, will do the actual work of creating your new keystore. To create a new keystore using this program, enter the following command at the command-line, substituting syntax appropriate for your OS.

$JAVA\_HOME/bin> keytool -genkey -alias [your alias] -keyalg RSA -keystore [preferred keystore path]

\*Use an [alias] and [path] of your choice.

Next, keytool will ask you to enter the password you want to use for the keystore. Again, choose whatever you like (but don't forget it).

After you choose the keystore password, you will enter the information required for the Certificate, such as your company and your name. Make sure this information is accurate, as you will have to submit this file to the Certificate Authority of your choice to obtain a certificate.

The last thing keytool will ask you to specify is the key password, which is the password specific to this specific certificates.

### **Step 2 - Creating the Certificate Signing Request**

Now that you've created your keystore, it's time to create a file called the Certificate Signing Request, or CSR, which will be used by the Certificate Authority of your choice to generate the Certificate SSL will present to other parties during the handshake.

You can use the keytool to create this file, as well. To do so, enter the following at the command line:

$JAVA\_HOME/bin> keytool -certreq -keyalg RSA -alias [your alias] -file [your certificate name].csr -keystore [path to your keystore]

\*Substitute the values you chose earlier for the [placeholders].

If you follow the instructions correctly, keytool will create a file called [your certificate name.csr], which you can submit to the CA you've chosen via the process they provide on their website. Using this file, they will generate a custom certificate for your server, which you can download according to the instructions they provide on their website.

### **Step 3 - Installing Your New Certificate**

SSL verifies the authenticity of a site's certificate by using something called a "chain of trust," which basically means that during the handshake, SSL initiates an additional handshake with the Certificate Authority specified in your site's certificate, to verify that you haven't simply made up your own CA.

In order to "anchor" your certificate's chain of trust, you have to download an additional certificate, called a "Root Certificate," from your CA, and then import both this certificate and your site's new certificate into your keystore. Your CA should provide information about obtaining a Root Certificate on their website.

Once you've downloaded both your own Certificate and the Root certificate provided by your CA, import them into your keystore with the following commands, replacing the [placeholders]:

To import the Root Certificate -

$JAVA\_HOME/bin> keytool -import -alias root -keystore [path to your keystore] -trustcacerts -file [path to the root\_certificate]

To import your new Certificate –

$JAVA\_HOME/bin> keytool -import -alias [your alias] -keystore [path to your keystore] -file [path to your\_keystore]

Now you are the proud owner of a functional, certified keystore.

---------------------------------------------------------------------------------------------------------------------

### **PART II - Configuring Tomcat to use SSL**

### **Step 1 - Configuring Tomcat's SSL Connectors**

Open server.xml file located in the Tomcat Installation directory. Search for 8443 until you come across an entry that looks like this:

<!-- Define a SSL HTTP/1.1 Connector on port 8443 This connector uses the JSSE configuration, when using APR, the connector should be using the OpenSSL style configuration described in the APR documentation -->

<!-- <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" maxThreads="150" scheme="https" secure="true" clientAuth="false" sslProtocol="TLS"/> -->

You'll notice that the comment enclosing this connector talks about a choice between APR and JSSE configurations. This refers to the implementation of SSL you are intending to use. JSSE, which is Tomcat's default configuration, is supported by default, and included in all JDKs after version 1.4. So if you don't even know what APR is, you only need to uncomment this entry, and add some additional information to allow Tomcat to find your keystore:

<!-- Define a SSL HTTP/1.1 Connector on port 8443 This connector uses the JSSE configuration, when using APR, the connector should be using the OpenSSL style configuration described in the APR documentation -->

<Connector port="8443" maxThreads="150" scheme="https" secure="true" SSLEnabled="true" keystoreFile="path/to/your/keystore" keystorePass="Your Keystore Password" clientAuth="false" keyAlias="your Alias" sslProtocol="TLS"/>

If, on the other hand, you know that using the Apache Portable Runtime (APR), also known as Tomcat's "native library," is by far the best practice to follow, especially when using Tomcat as a standalone web server (which you probably are), and have already installed it on your server, then you'll need to alter this entry as follows to allow Tomcat to use APR's OpenSSL implementation in place of JSSE, or trying to use SSL will generate an error:

<!-- Define a SSL HTTP/1.1 Connector on port 8443 This connector uses the JSSE configuration, when using APR, the connector should be using the OpenSSL style configuration described in the APR documentation -->

<Connector port="8443" scheme="https" secure="true" SSLEnabled="true" SSLCertificateFile="path to your certificate.crt" SSLCertificateKeyFile=" path to your key file" SSLPassword="Your Keystore Password" SSLCertificateChainFile="path/to/your/root/certificate" keyAlias="your Alias" SSLProtocol="TLSv1"/>

Restart Tomcat. Once you're up and running again, test your configuration by connecting to a secure page, using a URL such as https://[your host]:8443. If you followed the directions correctly, you should be able to view the page over a secure HTTPS connection!

-------------------THANKS FOR READING-------------------

©SinghOnkaar