Sakai/OSPortfolio: Configuring Sakai/Tomcat for secure SSL over HTTPS

Previously this article had talked about getting Tomcat itself secured via a Keystore, and having Tomcat deal directly with the SSL traffic. Well, we've got a whole new paradigm now. :)

So you've got Sakai up and running, and now you'd like to enable secure-sockets layer (SSL) so you can run it encrypted over HTTPS instead of plain-text HTTP...

No problem. First make sure you have Sakai installed and configured, and test it to make sure it works as expected via port 8080 in clear-text (http). Once that's confirmed to be working well, then you take the steps you'll need to secure it for https.
Using APACHE to handle HTTPS traffic, with Tomcat backstage

We've found that the most flexible way to do this is to **use Apache to handle the secure https:// traffic**, and have Mod_JK communicate with Tomcat (Using the "AJP13" protocol) behind the scenes. That is, Apache will handle all of the actual encrypted user interaction on port 443 -- Tomcat will only talk to Apache via AJP, and your users can't even get directly to Tomcat, only through Apache.

Here's what you will need, certificate-wise:

- A private key (certificate)
- A certificate-signing request (CSR)
- A certificate authority (CA) to sign the public key
- A public key signed by the CA

Note that you can be your own CA -- but your users' browsers will alert them to the fact that the certificate-authority that signed the certificate isn't a recognized one, and they should "proceed at their own risk, with extreme caution", and "don't complain if something odd happens because we're

not responsible for you trespassing in these dangerous, murky waters"... warnings like that. If you pay a recognized CA to sign your certificate instead, they won't see any warnings at all and your users will have a nice, seamless experience as they browse securely.

You will also need the following components -- and if you use Debian or a Debian derivative such as Ubuntu, we've included the commands used to install them:

- Apache (**apt-get install apache2**)
- Apache's Mod_SSL (**dpkg -S mod_ssl** indicates that 'apache2.2-common' supplies it, meaning that it comes with apache2)
- Apache's Mod_JK (**apt-get install libapache2-mod-jk**, this is where the AJP protocol setup comes from)
- OpenSSL (**apt-get install openssl**)

Creating your private and public keys/certificates:

Ready? First set up your fully-qualified-domain-name and server-name:

```
HOSTNAME=`hostname`
SERVER=`hostname | cut -f1 -d.`
mkdir sakai-cert
cd sakai-cert/
```

So here $HOSTNAME should be something like "sakaisys.university.edu" and $SERVER is "sakaisys". Set yours according to your context! (Note that we will be using $HOSTNAME and $SERVER below instead of fixed strings, even when it's not possible to use the variable in a particular context.)

Next, generate your private key:

```
# openssl genrsa -des3 -out $SERVER.private.key 1024
Generating RSA private key, 1024 bit long modulus
.....................................++++++
.................................++++++
```

```
e is 65537 (0x10001)
Enter pass phrase for $SERVER.private.key: pazwrd-goeth-here
Verifying - Enter pass phrase for $SERVER.private.key: pazwrd-goeth-here
```

If you want to remove the password on your private key, try this:

```
# openssl rsa -in $SERVER.private.key -out $SERVER.private.nopassword.key
```

Now we do the hard part, creating a certificate-signing request (CSR). Here you give your geographical and structural information, and note in particular that *you must have your server's FQDN as the COMMON NAME!*

```
# openssl req -new -key $SERVER.private.key -out $SERVER.csr
Enter pass phrase for sakaibb.private.key: pazwrd-goeth-here
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]: New Hampshire
Locality Name (eg, city) []: Portsmouth
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Serensoft University
Organizational Unit Name (eg, section) []: Information Technology
Common Name (eg, YOUR name) []: sakaisys.university.edu
Email Address []: helpdesk@university.edu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Again, just to be certain you're aware -- the COMMON NAME must be the fully-qualified domain name (FQDN) of your server. That's the value that was originally put into $HOSTNAME when we got started.
At this point you have a private key (**$SERVER.private.nopassword.key**) and a certificate-signing request (**$SERVER.csr**) to get signed. You can try signing it yourself, or ship it off to a known/trusted certificate authority (CA) to get it signed officially. There's nothing wrong with creating

your own CA and self-signing your certificates, but your users will get a warning saying that the certificate was signed by an unrecognized CA.
Once you get a signed public certificate (**$SERVER.public.key**) you might want to have a look and see what's in it. Here's how:

```
# openssl x509 -in $SERVER.public.key -text -noout
```

Post your public and private keys where Apache can use them:

```
# cp $SERVER.public.key /etc/apache2/ssl/certs/
# cp $SERVER.private.nopassword.key /etc/apache2/ssl/private/
```

Configure Apache, Mod_SSL and Mod_JK

Apache will be handling all our web traffic, mod_ssl will encrypt it all for us, and mod_jk will pass off appropriate requests to Tomcat for handling by Sakai code.

1. Create a directory /var/www/$SERVER and put an empty "index.html" there.

You might also have a minimal "splash" page there instead, with a link to the https:// version of your Sakai instance.

2. Edit /etc/apache2/sites-available/$SERVER thus:

Note that this example shows a bare-bones setup. You can enhance your configuration to do what you please, but it's a good idea to start with a simple setup first, test it, and once you are confident that all is well, you can start adding tweaks later.

```
‹VirtualHost *:80›
        ServerName  $HOSTNAME
        ServerAlias 192.168.123.234
        ServerAlias $SERVER
        ServerAlias 127.0.0.1

        #ServerAdmin webmaster@localhost

        RedirectMatch ^/$         https://$HOSTNAME/portal
```

```
        RedirectMatch ^/(.+)      https://$HOSTNAME/$1
‹/VirtualHost›
```

So, for all HTTP requests, we redirect all traffic to the HTTPS secure port. (Instead of $HOSTNAME above you'll have your actual FQDN, and instead of $SERVER you'll have the node-name of your server, of course.)

Now for the SSL and JK (aka AJP) portions of our program:

```
‹IfModule mod_ssl.c›
JkWorkersFile /etc/apache2/workers.properties
‹VirtualHost *:443›
        ServerName  $HOSTNAME:443
        # ServerAlias -- not for HTTPS! wouldn't match the certificate

        # Just in case:
        DocumentRoot /var/www/$SERVER

        # Servlet for context to worker named sakai, see workers.properties
        JkMount /* sakai
        JkUnmount /library/skin/* sakai
        JkUnmount /library/content/* sakai

        Alias /library/skin    /path/to/tomcat/webapps/library/skin
        Alias /library/content /path/to/tomcat/webapps/library/content

        #   SSL Engine Switch:
        #   Enable/Disable SSL for this virtual host.
        SSLEngine on
        SSLCertificateFile    /etc/ssl/certs/$SERVER.public.key
        SSLCertificateKeyFile /etc/ssl/private/$SERVER.private.nopassword.key

‹/VirtualHost›
‹/IfModule›
```

Both of the above ‹VirtualHost› snippets should be in the same /etc/apache2/sites-available/$SERVER config file.

Here if we don't have SSL (IfModule mod_ssl.c) we don't need to worry about mod_jk, and if we do have mod_ssl but we don't have mod_jk we'd rather have Apache break anyway, so we can fix it. You'll need to replace the BOLD items above with the items that make sense for your server, of course ("$SERVER" isn't valid inside Apache config files, for example -- you'll need to replace that with the real value for your server).

Note that we will have Tomcat handle most requests (**JkMount /* sakai**) but not the static items in /library/skin or /library/content (**JkUnmount ...**).

3. Edit /etc/apache2/workers.properties.

The above Apache configuration refers to /etc/apache2/workers.properties. That's what configures the core of the Tomcat AJP connector -- here is the **workers.properties** file in its entirety:

```
# mod_jk config to connect apache to tomcat
workers.tomcat_home=/home/sakai/tomcat
workers.java_home=/usr/local/java
ps=/
worker.list=sakai

worker.sakai.port=8009
worker.sakai.host=localhost
worker.sakai.type=ajp13
worker.sakai.lbfactor=1
```

The first # line is just a comment in the file, not a command-line entry! What this does is tell Apache to connect to Tomcat using port 8009 on localhost.

4. Enable your virtual website

```
# a2ensite $SERVER
```

5. Restart Apache:

```
# /etc/init.d/apache2 restart
```

Note that Apache can be restarted just about any old time -- it's just a conduit for traffic bound for Tomcat. Tomcat is where all the session information is stored. Restarting Apache might get a few of your users a quick "server not found" message, but a quick refresh will resume their session as if nothing happened.

Restarting Tomcat, on the other hand, **is** a big deal. Your users will be booted off and their sessions closed completely. Careful!

Configure Tomcat

Finally, you must edit $CATALINA_HOME/conf/server.xml to make sure that the AJP connection is enabled on port 8009, and to turn off clear-text traffic on port 8080. The AJP connector should NOT be commented out:

```
‹!-- Define an AJP 1.3 Connector on port 8009 --›
‹Connector port="8009"
       address="127.0.0.1"
       enableLookups="false" redirectPort="8443" protocol="AJP/1.3" /›
```

As shown above, you might also want to make sure you're asking Tomcat to look for connection on localhost (127.0.0.1) only, meaning it'll ignore your public-facing traffic, for even tighter security.

And the HTTP connector on port 8080 should be commented out:

```
‹!-- Define a non-SSL HTTP/1.1 Connector on port 8080 --›
‹!-- ‹Connector port="8080" maxHttpHeaderSize="8192"
...
/› --›
```

If you leave the port-8080 traffic open, folks might still keep on using it, and any network packet sniffer would be able to cull passwords and everything else! Better to lock it down.

Finally, restart Tomcat (after checking either A: to see that your users are off, or B: you're willing to suffer the consequences from those who aren't):

```
# $CATALINA_HOME/bin/shutdown.sh
# tail -f $CATALINA_HOME/logs/catalina.out
```

Once your Tomcat logs show "INFO main org.apache.coyote.http11.Http11BaseProtocol - Stopping Coyote HTTP/1.1 on http-8080" you can ^C to get out of "tail -f" and then start Sakai back up again:

```
# $CATALINA_HOME/bin/startup.sh
# tail -f $CATALINA_HOME/logs/catalina.out
```

Watch for "INFO main org.apache.catalina.startup.Catalina - Server startup in #### ms" and then you're ready!

You could easily configure Apache to listen on port 8080 and forward all requests to http://$HOSTNAME/portal as well.

Easy!

Unique solution ID: #1061
Author: will trillich
Last update: 2009-10-01 16:26