RMS Enterprise Security

Notes on this Document

In this document we will be dealing with two forms of security in RMS. Part of the document will be focusing on creating and utilizing certificates, and the other part will focus on some changes that can be made to Tomcat to increase the security it uses. There are many ways to enable various levels of security in RMS. The things in this document can be taken piece meal or in whole depending upon the requirements of your installation. The first and most important thing to do before you start is to speak with your site's IT group to discover their site requirements. Enabling some of these options could potentially cause your server to fail if they are not supported by the local IT infrastructure. Thoroughly research your site's requirements before enacting any of these security measures.

Certificates

About Certificates

Certificates are files that clients and servers use to authenticate information about themselves. They can certify who they are, where they come from, and they can also be used by the parties involved to agree on a method of communication. There are ***MANY*** ways to implement certificates, but this document will only concentrate on a couple of different ways, Self-Signed and Certificate Authority (CA) Issued certificates.

Notes on Certificates

For this document we will be doing all the commands from PowerShell since we will be working on servers. The first line will explain the step, the next line will be the actual command, and the next piece will be an explanation of the command if necessary. Almost all of this will require your user have administrator privileges on the server.

We will be creating our own keystore and configuring Tomcat to use it as well. All these configuration steps will allow Tomcat to accept client connections and server connections to things like LDAPS. These connections will only work once the RMS installation has been fully completed.

With that being said, for the RMS Configuration to correctly test secure connections all certificates must be added to the cacerts keystore located in the Java base folder at C:\Program Files\Java\jdk-11.0.x\lib\security. This is the keystore that RMS Configuration uses to connect and test LDAPS and secure SQL. Using this keystore also allows for direct manipulation of the JVM rather than manipulating Tomcat. Tomcat runs as an instance of JVM, so you are essentially working with the puppet master rather than the puppet. This means you can use this same keystore when configuring Tomcat, but **using cacerts is not a best practice for server security**. There are other ways to get this same functionality, but for this document we will focus on these methods.

Self-Signed Certificate

Self-signed certificates are created by the server and then given to a client to tell them "I certify myself as being secured". From a security point of view, this is probably the worst way you can possibly secure something. Trusting a self-signed certificate is like allowing a thief in your house and him telling you, "Don't worry, I won't take anything. I'm not a thief." Self-signed certificates should only be used as a means to test your implementation or if it is impossible to get an actual certificate from a Certificate Authority (CA). The following steps can be used to create a self-signed certificate, connect Tomcat to the keystore that contains the certificate, and configure Tomcat to redirect to HTTPS.

1. Stop the Tomcat service.

2. Open PowerShell and change the directory to C:\Program Files\Java\jdk-11.0.x\bin. **Note**: The xxx indicates what version of Java 11 is installed.

   Command: `cd ".\Program Files\Java\jdk-11.0.x\bin"`

3. Create a self-signed certificate.

   Command:
   ```
   .\keytool -genkey -alias mydomain -keyalg RSA -keypass changeit -
   keystore keystore.jks -storepass changeit -validity 365
   ```

   Explanation: Use the keytool to generate a key (certificate) with the alias name of mydomain. The key's algorithm will use RSA, and the password to the key will be "changeit". Create a keystore named keystore.jks in the \bin folder, and add the certificate to it. The password for the keystore is "changeit". This key will last for 365 days.

   keytool -genkey -alias <alias> -keyalg RSA -keystore <host name> -keysize 2048
   keytool -certreq -alias <alias> -keystore <host> -file <host>.csr

4. You will be asked a series of questions at this point.

   ```
   What is your first and last name?
     [Unknown]:  *YOUR SERVER'S HOSTNAME*
   What is the name of your organizational unit?
     [Unknown]:  Company
   What is the name of your organization?
     [Unknown]:  Org
   What is the name of your City or Locality?
     [Unknown]:  City
   What is the name of your State or Province?
     [Unknown]:  State
   What is the two-letter country code for this unit?
     [Unknown]:  CC
   Summary: CN=Firstname Lastname, OU=IT&S, O=COMPANY NAME p.l.c, L=locality,
   ST=state, C=country
   ```

5. Once you have answered the questions, you will be prompted to confirm your entries. It says [no], but press Y and then Enter to confirm your entries.

   ```
   Is CN=Firstname Lastname, OU=Company, O=Org, L=City, ST=State, C=CC correct
     [no]: Y
   ```

6. Copy the keystore.jks file to the C:\Program Files\Apache Software Foundation\

Tomcat 9.0\conf folder.

7. Open the server.xml file located in C:\Program Files\Apache Software Foundation\
   Tomcat 9.0\conf.

8. Locate the section that says:

```
<!--
<Connector port="**8number**" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
-->
```

   Explanation: The `**8number**` will either be 80 or 8080 depending upon how you originally setup Tomcat. This is the port that people would normally reach when they type in http://URL/rms or http://URL:8080/rms.

9. Remove the "`<!--`" and the "`-->`" if it surrounds the connector section. These characters comments out the section in between them. We want this section to run, so delete the characters.

10. Change the redirectPort to "`443`". It is suggested you use 443 so the user doesn't have to type in https://URL:8443/rms, instead it will be https://URL/rms.

11. Locate the section that says:

```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
         maxThreads="150" SSLEnabled="true">
  <SSLHostConfig>
      <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                   type="RSA" />
  </SSLHostConfig>
</Connector>
-->
```

12. Remove the "`<!--`" and the "`-->`".

13. Change the "`8443`" to "`443`" to match the previous setting.

14. Change the location to your keystore file: `conf/keystore.jksecure`

15. Add in the line for your keystore's password: `certificateKeystorePass="changeit"`

We have seen some certificates not use this connector port configuration, here is another example that can be used:

```
<Connector
           protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="8443" maxThreads="200"
```

```
scheme="https" secure="true" SSLEnabled="true"
keystoreFile="${user.home}/.keystore" keystorePass="changeit"
clientAuth="false" sslProtocol="TLS"/>
```

16. Save the server.xml file and close it.

17. Open the web.xml file located in C:\Program Files\Apache Software Foundation\Tomcat 9.0\conf.

18. Add the following lines of text to the file between the last line that says `</mime-mapping>` and `<!-- ============== Default Welcome File List =============== -->`:

```
    <security-constraint>
          <web-resource-collection>
                <web-resource-name>RMS Web Application User Interface</web-
                resource-name>
                <url-pattern>/index.html</url-pattern>
                <url-pattern>/rms.swf</url-pattern>
                <url-pattern>/flex-http.crossdomain</url-pattern>
                <url-pattern>/flex-https.crossdomain</url-pattern>
                <url-pattern>/libs/*</url-pattern>
                <url-pattern>/messagebroker/*</url-pattern>
                <url-pattern>/logfile/*</url-pattern>
                <url-pattern>/resources</url-pattern>
                <url-pattern>/report/*</url-pattern>
          </web-resource-collection>
          <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
          </user-data-constraint>
    </security-constraint>
```

19. Save the web.xml file and close it.

20. Right-click on the Monitor Tomcat icon in the system tray

21. Start the Tomcat service.

At this point typing in https://URL/rms should allow you to connect via a secure connection. At this point you may receive an error saying the certificate name doesn't match the server name or that the certificate is unsecure due to it being self-signed, even with this warning it should allow you to connect via HTTPS by telling the browser to continue anyway.

Certificate Authority Issued Certificates

Certificates that have been signed by a Certificate Authority (CA) are the best way of securing your server's security and connections. CAs provide certificates that have been digitally signed by either the company's network security team or a worldwide authorized CA (Verisign, DigiCert, Symantec, GoDaddy, etc.). The certificates will either be given based upon organizational rules, or the security team will request a Certificate Signing Request (.csr) file from the server. For this document we will be creating and providing a .csr file. Once the CA processes the .csr file they will return a number of certificates back to you. These could include root certificates, intermediate certificates, keychain files, and company/site certificates. We will include all of these in this document.

1. Stop the Tomcat service.

2. Open PowerShell and change the directory to C:\Program Files\Java\jdk-11.0.x\bin.

   Command: `cd ".\Program Files\Java\jdk-11.0.x\bin"`

3. Generate a key on your server.

   Command: `.\keytool -genkey -alias mydomain -keyalg RSA -keystore keystore.jks -storepass changeit`

   Explanation: Use the keytool program to generate a key with the alias name of mydomain. The key will use the RSA algorithm and it will create a keystore file in the \bin directory named keystore.jks with the password of "changeit". **Note**: **The alias name for the generated key, the request, and the import of that key/keychain must all be the same.**

4. Generate a .csr file using the key you have just created.

   Command: `.\keytool -certreq -alias mydomain -file certreq.csr -keystore keystore.jks -storepass changeit`

   Explanation: Use the keytool to create a certificate request. The request is based on the mydomain key and will be named certreq.csr. You can find the key in the keystore.jks file with the password of "changeit". **Note**: Use the same alias name used in the key creation step. The .csr file will be generated in the \bin folder.

5. Send the .csr file to the CA. They will return back several files including the root certificate, the intermediate certificate, the keychain file, and the site certificate.

6. Copy the certificates into the C:\Program Files\Java\jdk-11.0.x\bin folder.

7. **EITHER/OR** At this point the instructions will split. Preferably you will be given a keychain file. From here skip to step 10. If you are only given individual certificates continue to step 8 and skip the step for importing the keychain file.

Special Note on Root, Intermediate, and Keychain Certificates

When dealing with larger installations, you may run into different types of certificates known as root, intermediate, and keychain certificates. In large networks you may have a certificate issued by a certificate authority that exists on a different part of the domain from a machine that is trying to connect to your server. In this case the machine trying to connect to the server would not trust the certificate authority that issued the certificate.
There are a couple of ways around this roadblock. The first way is by returning multiple certificates, the root certificate is a certificate authority that ***EVERYONE*** involved trusts. Each intermediate certificate would be a step of certification form the server you are installing back to the root (a chain of certificates).

Another way to solve this is with a keychain certificate. The keychain certificate is a single file that contains all the certificates involved in the chain of certificates bundled together. This is usually the preferred method in this scenario.

8. Import the root certificate to the keystore.

   Command: `.\keytool -import -trustcacerts -alias root -file root.crt -keystore keystore.jks -storepass changeit`

   Explanation: Use the keytool to import a trusted certificate named root.crt with a new alias of 'root' into the keystore.jks file with the password of "changeit".

9. Import the site certificate to the keystore. **Skip step 10 and move on to step 11.**

   Command: `.\keytool -import -trustcacerts -alias mydomain -file site.crt -keystore keystore.jks -storepass changeit`

   Explanation: Use the keytool to import a trusted certificate named site.crt with the alias of mydomain into the keystore.jks with the password of "changeit". **Note**: You must use the same alias as the alias used in the request. If you do this correctly you will receive a message that says, "Certificate reply was installed in keystore."

10. Import the keychain file to the keystore.

    Command: `.\keytool -import -trustcacerts -alias mydomain -file keychain.p7b -keystore keystore.jks -storepass changeit`

    Explanation: Use the keytool to import a trusted certificate keychain named keychain.p7b with the alias of mydomain into the keystore.jks with the password of "changeit". **Note**: You must use the same alias as the alias used in the request. If you do this correctly you will receive a message that says, "Certificate reply was installed in keystore."

11. Copy the keystore.jks file to C:\Program Files\Apache Software Foundation\
    Tomcat 9.0\conf

12. Open the server.xml file located in C:\Program Files\Apache Software Foundation\
    Tomcat 9.0\conf.

13. Locate the section that says:

```
<!--
<Connector port="**8number**" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
-->
```

    Explanation: The `**8number**` will either be 80 or 8080 depending upon how you originally setup Tomcat. This is the port that people would normally reach when they type in http://URL/rms or http://URL:8080/rms.

14. Remove the "`<!--`" and the "`-->`" if they are surrounding the section. These characters comments out the section in between them. We want this section to run, so delete the characters.

15. Change the redirectPort to "`443`". It is suggested you use 443 so the user doesn't have to type in https://URL:8443/rms, instead it will be https://URL/rms.

16. Locate the section that says:
```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
         maxThreads="150" SSLEnabled="true">
  <SSLHostConfig>
      <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                   type="RSA" />
  </SSLHostConfig>
</Connector>
-->
```

17. Remove the "`<!--`" and the "`-->`".

18. Change the "`8443`" to "`443`" to match the previous setting.

19. Change the location to your keystore file: `conf/keystore.jksecure`

20. On the line after the keystore file name, add in the line for your keystore's password: `certificateKeystorePass="changeit"`

21. Save the server.xml file and close it.

22. Open the web.xml file located in C:\Program Files\Apache Software Foundation\Tomcat 9.0\conf.

23. Add the following lines of text to the file between the last line that says `</mime-mapping>` and `<!-- ============== Default Welcome File List =============== -->`:
```
      <security-constraint>
            <web-resource-collection>
                  <web-resource-name>RMS Web Application User Interface</web-
                  resource-name>
                  <url-pattern>/index.html</url-pattern>
                  <url-pattern>/rms.swf</url-pattern>
                  <url-pattern>/flex-http.crossdomain</url-pattern>
                  <url-pattern>/flex-https.crossdomain</url-pattern>
                  <url-pattern>/libs/*</url-pattern>
                  <url-pattern>/messagebroker/*</url-pattern>
                  <url-pattern>/logfile/*</url-pattern>
                  <url-pattern>/resources</url-pattern>
                  <url-pattern>/report/*</url-pattern>
            </web-resource-collection>
            <user-data-constraint>
                  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
            </user-data-constraint>
      </security-constraint>
```

24. Save the web.xml file and close it.

25. Right-click on the Monitor Tomcat icon in the system tray and choose Configure…

26. Click the Java tab.

27. In the Java Options section add this line: `–Djavax.net.ssl.trustStore=C:\Program Files\Apache Software Foundation\Tomcat 9.0\conf\keystore.jks`. This tells the JVM to use that keystore when working with Tomcat.

28. Apply the changes.

29. Start the Tomcat service.

At this point typing in https://URL/rms should allow you to connect via a secure connection. Also, if you are using these certificates to also connect to an LDAP or SQL server over SSL, this should allow you to connect to the server now.

RMS and SQL using SSL

Once the certificates have been setup in the keystore, there are a couple of other steps that need to be completed in order for RMS to communicate with SQL via SSL. One thing to note is that this method does drop the level of security to TLSv1. This is a bug in the JDBC driver that RMS uses, there is a feature request in to fix this issue, but this process is a temporary work around until that fix is implemented.

1. Stop the Tomcat service.

2. Browse to the rms.properties file located in C:\ProgramData\AMX\Resource Management Suite\Server\configuration.

3. Edit the rms.properties file.

4. Locate the line that begins with "`jbdc.url`".

5. At the end of this line add the following string: "`;ssl=require`". Don't add the quotation marks.

6. Save and close the rms.properties file.

7. Open up the Configure Tomcat program in the Start menu.

8. Click on the Java tab.

9. In the Java Options section, add the following line: "`–Djsse.enableCBCProtection=false`". Do not add the quotation marks.

10. Start the Tomcat service. At this point RMS should be able to communicate with SQL via SSL

Additional Keytool and Tomcat Commands

The following is an additional list of very useful keytool and Tomcat commands. Depending upon which commands you use, some of them you may need to use before you do any of the other commands from above (i.e. the change password command).

Additional Certificate Options

Command:
```
-ext SAN=dns:xxxxxx
-ext SAN=ip:aaa.bbb.ccc.ddd
-sigalg SHA256withRSA
-storetype JKS
```

Example:
```
.\keytool -genkey -keyalg RSA -sigalg SHA256withRSA -keysize 2048       -alias
mydomain -ext SAN=dns:tsw2012r2std9 -keystore keystore.jks -storetype JKS
```

Explanation: These are additional parameters you can add to your certificate creation string and to your certificate request. The -ext SAN command adds a subject alternative name to your certificate, some browsers require the certificate have a SAN. The -sigalg command specifies the signature algorithm to be used when the key is signed. The -storetype command tells the keytool program what type of keystore you will be creating.

Change a Java Keystore Password

Command:
```
.\keytool -storepasswd -new new_storepass -keystore "C:\Program Files\Java\jdk-
11.0.x\lib\security/cacerts" -storepass changeit
```

Explanation: Use the keytool to create a new password as "new storepass" for the keystore cacerts located in C:\Program Files\Java\jdk-11.0.x\lib\security that has the current password of "changeit".

Change a Private Key Password

Command:
```
.\keytool -keypasswd -alias Tomcat -keypass changeit -new new_password -
keystore "C:\Program Files\Java\jdk-11.0.x\lib\security/cacerts"
-storepass password
```

Explanation: Use the keytool to create a new password for the certificate with the alias of Tomcat with the current password of "changeit" to the new password of "new_password" located in the cacerts

Delete Certificate from a Keystore

Command:
```
.\keytool -delete -alias mydomain -keystore keystore.jks -storepass password
```

Explanation: Use the keytool to delete the certificate with the alias mydomain from the keystore.jks keystore with the password "password".

Check Certificates in a Keystore

Command:
```
.\keytool -list -v -keystore "C:\Program Files\Java\jdk-
11.0.x\lib\security/cacerts" -storepass changeit
```

Explanation: Use the keytool to produce the long form list of all the keys in the cacerts keystore with the password "changeit"

Export a Certificate

Command: `.\keytool -export -alias Tomcat -file Tomcat.crt -keystore "C:\Program Files\Java\jdk-11.0.x\lib\security/cacerts" -storepass changeit`

Explanation: Use the keytool to export your certificate with the alias name Tomcat to a file named Tomcat.crt. The certificate is in the cacerts keystore located in C:\Program Files\Java\jdk-11.0.x/lib\security with the password "changeit".

Change the Keystore Tomcat Utilizes

In the Tomcat configuration, under the Java tab, in the Java Options section you can change the keystore that Tomcat uses when it presents keys to a client. A keystore holds private keys like the self-signed certificate used for setting up HTTPS.

Command:
```
-Djavax.net.ssl.keyStore=/path/to/keystore
-Djavax.net.ssl.keyStorePassword=password
```

Explanation: These two commands change the path to the keystore and the password to the keystore that Tomcat is using.

Change the Truststore Tomcat Utilizes

In the Tomcat configuration, under the Java tab, in the Java Options section you can change the truststore that Tomcat uses when it connects to another server. A truststore holds the public key for a remote server that you need to trust when you, for example, need to connect to an LDAP server over SSL.

Command:
```
-Djavax.net.ssl.trustStore=/path/to/truststore
-Djavax.net.ssl.trustStorePassword=password
```

Explanation: These two commands change the path to the truststore and the password to the truststore that Tomcat uses to login to other servers.

Tomcat Security Settings

About Tomcat Security Settings

There are far too many options and settings built within Tomcat to list in this document. Tomcat is an open source program, and thus the main source of information concerning Tomcat will be found on the internet. The things covered in this document will be settings specific to the RMS web application or issues that have come across the desks at tech support.

Notes on Tomcat Security Settings

This will be broken up into three different sections Web.xml and Server.xml. Most of the setting changes for Tomcat happen in web.xml and server.xml. There are a few additional changes that can be made in the Tomcat configuration dialog box, but the majority occur in the xml files. All changes to Tomcat must be made while the Tomcat service has been stopped. If any changes are made and saved while the service is running it can cause it to stop incorrectly or corrupt the Tomcat installation itself. Whenever you're working with Tomcat make sure it is stopped. Once you have made changes to the files, save the file, and restart Tomcat for the settings to take effect.

Web.xml

Expanded Enabling of HTTPS

As part of the installation guide for RMS, you will find the following lines included:

```
<security-constraint>
<web-resource-collection>
<web-resource-name>RMS Web Application User Interface</web-resource-name>
      <url-pattern>/index.html</url-pattern>
      <url-pattern>/rms.swf</url-pattern>
      <url-pattern>/flex-http.crossdomain</url-pattern>
      <url-pattern>/flex-https.crossdomain</url-pattern>
      <url-pattern>/libs/*</url-pattern>
      <url-pattern>/messagebroker/*</url-pattern>
      <url-pattern>/logfile/*</url-pattern>
      <url-pattern>/resources</url-pattern>
      <url-pattern>/report/*</url-pattern>
</web-resource-collection>
<user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

The line that says `<transport-guarantee>` indicates that HTTPS will be utilized whenever one of the URL patterns indicated above are used. The ones in the installation manual are all of the basic URLs that can be accessed. If a site requires that all communication be HTTPS, there are additional lines that must be added to the other URL patterns. Please see below:

```
<url-pattern>/image/*</url-pattern>
<url-pattern>/favicon/*</url-pattern>
<url-pattern>/config/flex-settings/</url-pattern>
<url-pattern>/m/*</url-pattern>
<url-pattern>/ui/*</url-pattern>
<url-pattern>/ui-api/*</url-pattern>
```

If your RMS system is only using NX masters, you can also lock down the master's communication to HTTPS. Your NX masters must use the HTTPS address in the RMS Server URL, and if you are using a local CA your certificates must be uploaded to the master. In order to do this, add this additional URL pattern:

```
<url-pattern>/ui-api/*</url-pattern>
```

Enabling HSTS and XSS Security

HSTS stands for HTTP Strict Transport Security, it protects against protocol downgrade attacks and cookie hijacking. XSS stands for Cross-Site Scripting, it enables attackers to inject client-side scripting into web pages viewed by other users. Both of these issues can be fixed by the same configuration settings in web.xml.

One thing to note about the web.xml file. There are several commented out sections of the file that mention different things you can enable/disable within Tomcat. Pay attention to these sections because many of them may outline fixes not mentioned in this document. They are very helpful.

To enable HSTS and XSS security, find the filter for `httpHeaderSecurity`. The section of code should look like this:

```
<!--
  <filter>
        <filter-name>httpHeaderSecurity</filter-name>
        <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
        <async-supported>true</async-supported>
    </filter>
-->
```

Remove the comment tags (`<!-- & -->`) to enable this section of code. Once the filter has been enabled, we must also enable the filter mapping. Find the section of code that looks like this:

```
<!--
    <filter-mapping>
        <filter-name>httpHeaderSecurity</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
    </filter-mapping>
-->
```

Remove the comment tags (`<!-- & -->`) to enable this section of code. Now that both sections of code have been uncommented, header security will be enabled on all communication with the server.

Server.xml

Ports and Connector Ports

All network communication to and from a device is done on network ports. By default, unsecure HTTP traffic occurs on port 80 or 8080. You can make this configuration choice on the initial Tomcat installation, or you can change it in the server.xml file itself. The connector port can be any port number you choose, but it's best to go with industry standards. Here is the default unsecured connector port in server.xml:

```
<Connector port="80" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

You can also choose to make a connector port secure. Whenever you choose to secure a port, there are a few more settings you have to pass along as well. This is the default secured connector port in server.xml:

```
<Connector port="443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" />
```

The server.xml portion of this document will discuss the various additional parameters you can add or remove from your connector port declaration. These are by no means exhaustive lists, and many other options exists. These are just a few options that have been used in tech support.

protocol=

This option allows you to choose the protocol that Tomcat will use on the connector. The default choice of `HTTP/1.1` will make Tomcat auto-detect and choose which protocol to use from the additional settings you have enabled in the connector port. For example, the following two connector ports will both use the same protocol:

```
<Connector port="443" protocol="HTTP/1.1"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" />

<Connector port="443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" />
```

sslProtocol= & sslEnabledProtocols=

This option specifies what SSL protocols you will be using. You can go even more granular using the `sslEnabledProtocols=` option. This second command allows you to specify which versions of TLS you're going to use, including TLSv1, TLSv1.1, and TLSv1.2. Here is an example of using this connector option:

```
<Connector port="443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS”
           sslEnabledProtocols=”TLSv1.1,TLSv1.2”/>
```

ciphers= & Diffie-Hellman Moduli

Whenever you are given a certificate from a major certificate authority or your own certificate authority it will contain many different kinds of ciphers. Some of these will use what is called a Diffie-Hellman Moduli as a basis for the cipher key. Some will use a weaker number of bits than others, i.e. 1024 vs 2048. In order to guarantee the use of higher security ciphers, sometimes we will need to specify the ciphers being used on our server. This is where the `ciphers=` option comes into play. Here is an example of the option being used:

```
<Connector port="443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS”
           sslEnabledProtocols=”TLSv1.1,TLSv1.2”
           ciphers="TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
                    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
                    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
                    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
                    TLS_RSA_WITH_AES_128_CBC_SHA256,
                    TLS_RSA_WITH_AES_128_CBC_SHA,
                    TLS_RSA_WITH_AES_256_CBC_SHA256,
                    TLS_RSA_WITH_AES_256_CBC_SHA"/>
```

Final Notes

This is not an exhaustive list, and some of these options will be based off of the certificate you have received. For more specific information get the used ciphers, protocols, and any other additional information from your certificate authority before setting up this option.