D08 – Item 6

SECURE COMMUNICATIONS SETUP REPORT

Luis Candelario Luna
Antonio Carrasco Márquez
Gonzalo Delgado Chaves
Francisco Márquez Orellana
Sergio Morales Moreno

Table of contents

1. Introduction
2. Project setup
3. Server setup
4. Conclusions
5. References

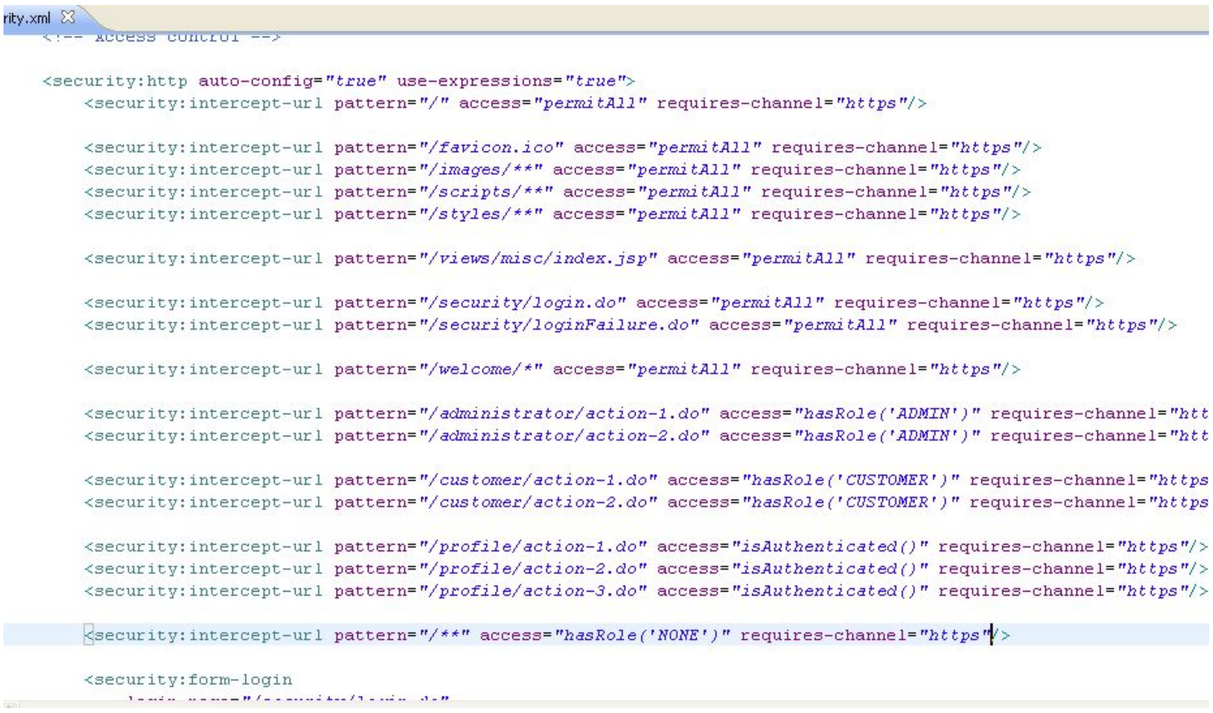
1. Introduction

In this document, we describe the necessary steps required to set up a Spring Maven project in our development configuration to use the https protocol for secure communications, as well as the steps to configure a Tomcat server in a pre-production configuration to accept the project's setup and enable the https protocol on deployment. The document will include images to provide visual aid.

2. Project setup

Every time a new project is expected to use secure communications with https, the following steps are required within the project's context, some of them specific to each project's domain. Since the development team is in charge of performing the changes, it is not required to do anything to the project itself, but it's possible to check on these changes.

Inside the project, we must configure the access of different services to the https protocol. This is done in our security.xml file, adding the requires-channel="https" parameter to those access pages we want to use https with.



```
rity.xml

    <security:http auto-config="true" use-expressions="true">
        <security:intercept-url pattern="/" access="permitAll" requires-channel="https"/>

        <security:intercept-url pattern="/favicon.ico" access="permitAll" requires-channel="https"/>
        <security:intercept-url pattern="/images/**" access="permitAll" requires-channel="https"/>
        <security:intercept-url pattern="/scripts/**" access="permitAll" requires-channel="https"/>
        <security:intercept-url pattern="/styles/**" access="permitAll" requires-channel="https"/>

        <security:intercept-url pattern="/views/misc/index.jsp" access="permitAll" requires-channel="https"/>

        <security:intercept-url pattern="/security/login.do" access="permitAll" requires-channel="https"/>
        <security:intercept-url pattern="/security/loginFailure.do" access="permitAll" requires-channel="https"/>

        <security:intercept-url pattern="/welcome/*" access="permitAll" requires-channel="https"/>

        <security:intercept-url pattern="/administrator/action-1.do" access="hasRole('ADMIN')" requires-channel="htt
        <security:intercept-url pattern="/administrator/action-2.do" access="hasRole('ADMIN')" requires-channel="htt

        <security:intercept-url pattern="/customer/action-1.do" access="hasRole('CUSTOMER')" requires-channel="https
        <security:intercept-url pattern="/customer/action-2.do" access="hasRole('CUSTOMER')" requires-channel="https

        <security:intercept-url pattern="/profile/action-1.do" access="isAuthenticated()" requires-channel="https"/>
        <security:intercept-url pattern="/profile/action-2.do" access="isAuthenticated()" requires-channel="https"/>
        <security:intercept-url pattern="/profile/action-3.do" access="isAuthenticated()" requires-channel="https"/>

        <security:intercept-url pattern="/**" access="hasRole('NONE')" requires-channel="https"/>

        <security:form-login
```

Fig. 1: Setup of security.xml file for a system using https.

By default, it's recommended to use https protocol with every single page of the system. This is done for various reasons: first, it is the best choice from a security standpoint, since it means there are no unprotected pages. Also, mixing the http and https protocols makes the setup harder, and would require switching certain security options off to prevent conflict. As such, we have decided to use https in every page of our systems, as we believe the security improvement outweighs the potential slowdown on page load.

By doing this, the project is configured. Once exported, it can be deployed in a server that accepts https, and will use its functionalities automatically.

3. Server setup

It is necessary to configure the pre-production environment's Tomcat server to enable secure communications, and once done, it won't be necessary to revert the changes unless the user doesn't want to prioritize https over http or similar protocols.

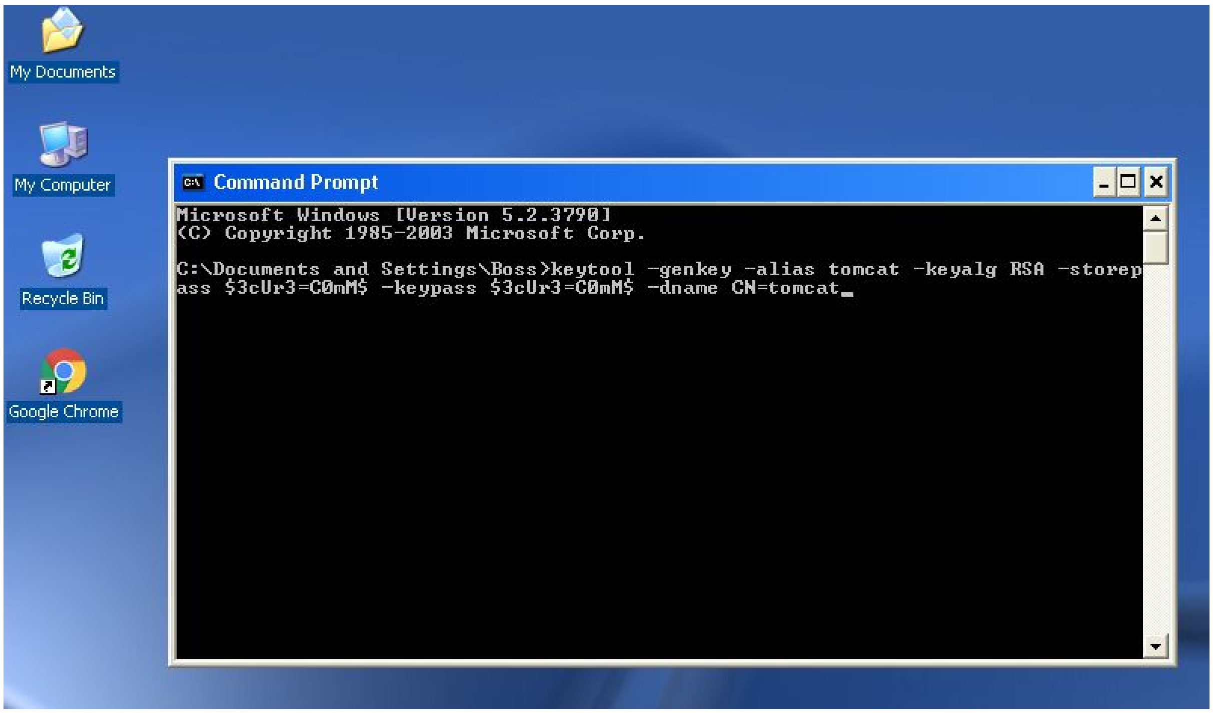
The first thing required to enable secure communications is a secure keystore, which can be obtained in two ways: either contacting a legitimate certification authority to obtain their certificate, or crafting a self-signed certificate.

Since the first option is usually expensive, varies from one authority to another, and we intend to use this keystore for developing and pre-production testing purposes, we will demonstrate how to craft a certificate using keytool, an integrated Java utility.

To create a certificate, open a terminal and execute the following command:

keytool -genkey -alias tomcat -keyalg RSA -storepass storepass -keypass keypass -dname CN=cnname

Where the “storepass” and “keypass” parameters are to be changed to secure passwords. The “alias” and “dname” parameters are information parameters to fill the certificate, and the “keyalg” parameter specifies the encryption algorithm to create the set of keys.



```
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\Documents and Settings\Boss>keytool -genkey -alias tomcat -keyalg RSA -storep
ass $3cUr3=C0mM$ -keypass $3cUr3=C0mM$ -dname CN=tomcat_
```

Fig. 2: Usage of command “keytool” to generate a self-signed keystore. The password used for both ends is “$3cUr3=C0mM$”.

Using this command will generate a .keystore file in the user's home directory. This keystore must be placed in the home directory of every work configuration used for the project.

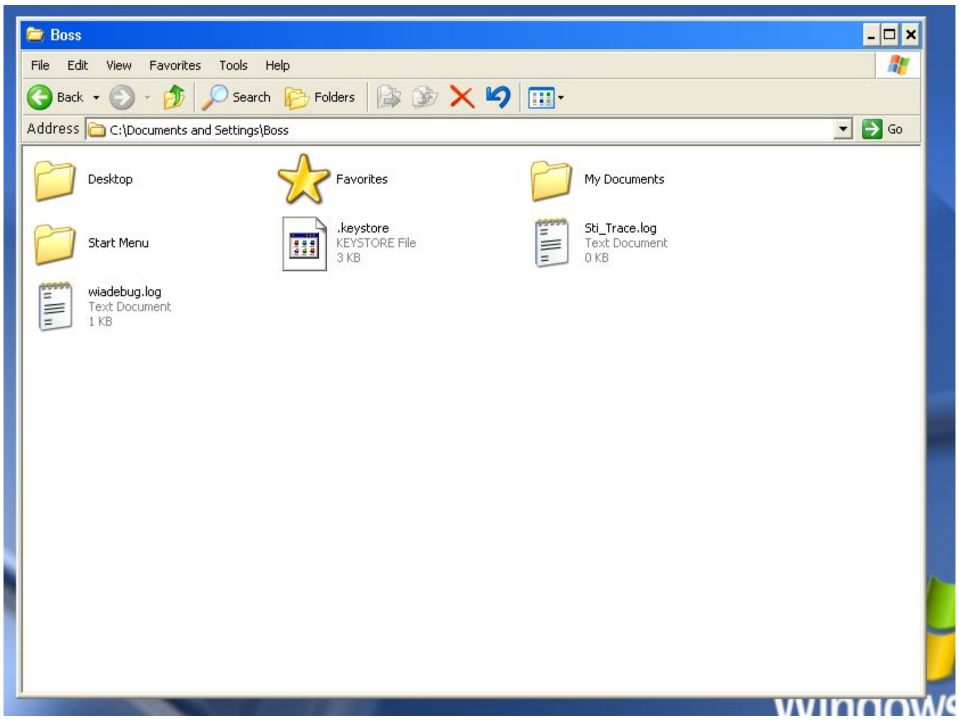
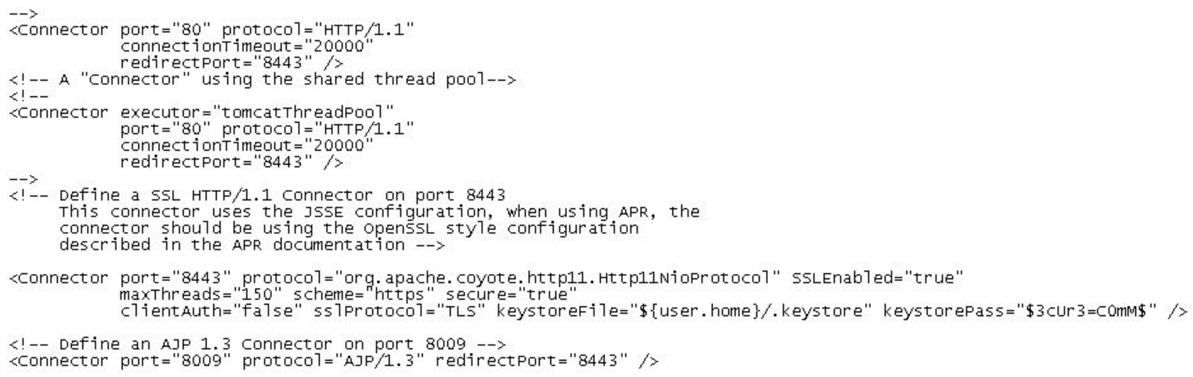


Fig. 3: Location of .keystore file inside the pre-production configuration.

To enable https in the pre-production server, we must head to the conf/server.xml file in Tomcat's install directory. Inside, there are a series of connectors, which are artifacts that redirect different ports to one or another protocol. Under the standard http connector, there should be a commented text, similar or identical to this:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
  maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS"
  keystoreFile="${user.home}/.keystore" keystorePass="keystorePass" />
```

Simply uncomment the text, or write the above into the file, to enable secure connections by default. Note that the http connector has a redirectPort="8443" attribute, to automatically redirect the calls to the https port. If this attribute isn't in the http connector, please add it as well. The “keystoreFile” and “keystorePass” attributes must be added, and the password for the keystore introduced.



```
-->
<Connector port="80" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
<!--
<Connector executor="tomcatThreadPool"
           port="80" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
-->
<!-- 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="org.apache.coyote.http11.Http11NioProtocol" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" keystoreFile="${user.home}/.keystore" keystorePass="$3cUr3=C0mM$" />

<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Fig. 4: Enabling the SSL HTTPS connector in the server.xml file inside Tomcat’s configuration folder.

Once the steps are done, deploying a project with the proper security configuration will make use of the https protocol for secure communications where applied.

4. Conclusions

With a simple number of steps, the development team can configure a project to use https in any way they choose (although it is very recommended to keep the entire system under secure communications), as well as a suitable server to accommodate these secure communications. The steps only need to be repeated for every new project: the secure keystore and Tomcat configuration don’t have to be changed.

5. References

Comparative between the http and secure http protocols:
https://www.instantssl.com/ssl-certificate-products/https.html

Information about how to set https up in Spring with a Tomcat server:
http://www.baeldung.com/spring-channel-security-https

Information about Apache Tomcat with SSL/TLS and secure communications:
http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html