

Lab No: 03 Securing websites using SSL or TLS

Aim: To demonstrate web server security using two tier client-server JSP or three tier client-browser, Apache tomcat 8 application server and back end third tier of database MySQL. Aim is to demonstrate secured communication between browser and web server using Secured Socket Layer (SSL)/ Transport Layer Security (TLS)

Introduction to SSL/TLS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are technologies which allow web browsers and web servers to communicate over a secured connection. This means that the data being sent is encrypted by one side, transmitted, then decrypted by the other side before processing. This is a two-way process, meaning that both the server AND the browser encrypt all traffic before sending out data.

Another important aspect of the SSL/TLS protocol is Authentication. This means that during your initial attempt to communicate with a web server over a secure connection, that server will present your web browser with a set of credentials, in the form of a "Certificate", as proof the site is who and what it claims to be. In certain cases, the server may also request a Certificate from your web browser, asking for proof that *you* are who you claim to be. This is known as "Client Authentication," although in practice this is used more for business-to-business (B2B) transactions than with individual users. Most SSL-enabled web servers do not request Client Authentication.

Following diagram depicts dynamic web server system three tiers. In case of two tier, only browser and web server will be available client server model.

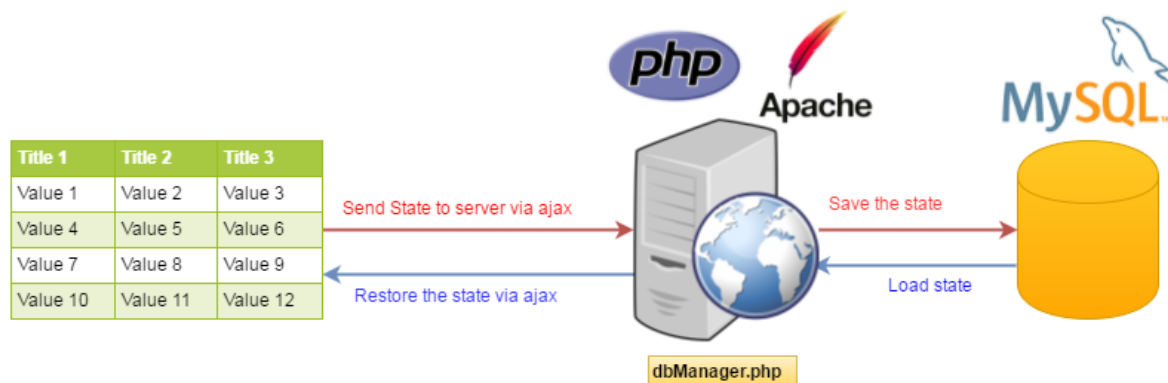


Figure 1: Ref: <http://refreshmymind.com/wp-content>

Above diagram shows three components of web server system. Presentation layer on left side i.e. browser, middle layer that is tomcat server and data layer is implemented with mysql.

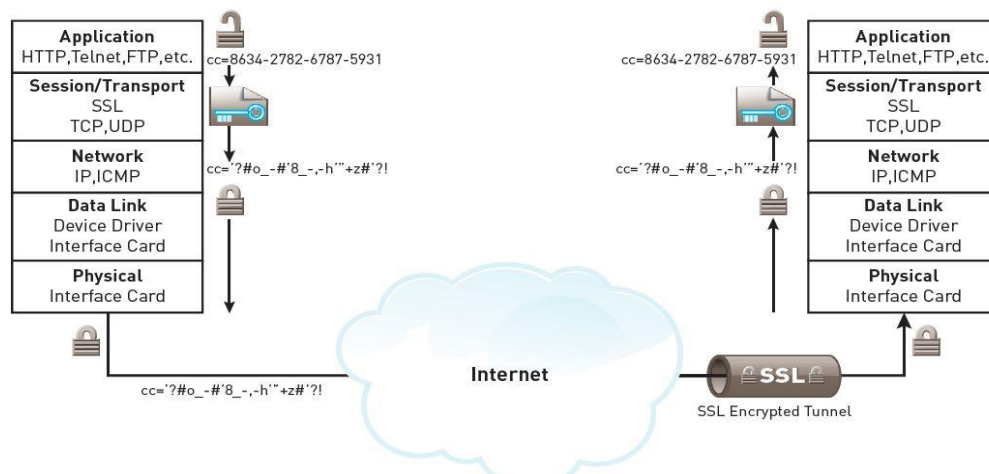


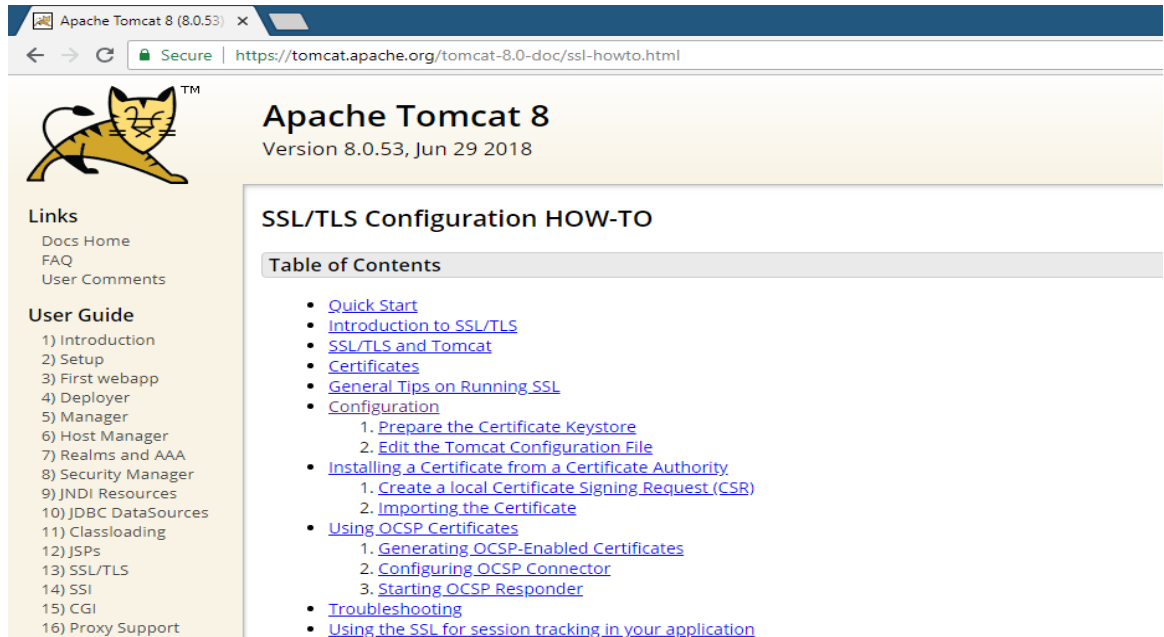
Figure 2: Client server protocol stack with SSL functionality

Now let us have practical steps of demonstration on Ubuntu system.

1. First prerequisite is install jdk on Ubuntu. (Preferably latest version) and add JAVA_HOME path in .bashrc which is available as in home directory. Also add jdk bin path in .bashrc so that you can access java command from any working directory.
2. Download the apache tomcat 8.5.33 application tar file from apache site. And extract it in your home directory. Also add CATALINA_HOME in .bashrc as follows.

export CATALINA_HOME=/home/bnjagdale/web/apache-tomcat-8.5.33

3. **Documentation:** Additionally you may refer the SSL/TLS manual of securing web sites



4. **Start the Tomcat.** Go to the apache bin directory and run following command

```
.../bin$sh startup.sh
```

Stop tomcat: And you can shutdown tomcat server with following command.

```
.../bin$sh shutdown.sh
```

Note: Every time you change configuration files of tomcat, make sure to restart the tomcat server to adopt the changes

5. **Access the Web Interface of tomcat server (Testing Web server)**

Now that we have create a user, we can access the web management interface again in a web browser. Once again, you can get to the correct interface by entering your server's domain name or IP address followed on port 8080 in your browser:


Open in web browser

http://server_domain_or_IP:8080 OR


http://127.0.0.1:8080/

The page you see should be the same one you were given when you tested earlier:

[Home](#) [Documentation](#) [Configuration](#) [Examples](#) [Wiki](#) [Mailing Lists](#) [Find Help](#)

Apache Tomcat/8.0.33  **The Apache Software Foundation**
<http://www.apache.org/>

If you're seeing this, you've successfully installed Tomcat. Congratulations!



Recommended Reading:
[Security Considerations HOW-TO](#)
[Manager Application HOW-TO](#)
[Clustering/Session Replication HOW-TO](#)

[Server Status](#)
[Manager App](#)
[Host Manager](#)

Developer Quick Start

[Tomcat Setup](#)
[First Web Application](#)

[Realms & AAA](#)
[JDBC DataSources](#)

[Examples](#)

[Servlet Specifications](#)
[Tomcat Versions](#)

6. Web server Manager interface

In order to use the manager web app that comes with Tomcat, we must add a login to our Tomcat server. We will do this by editing the tomcat-users.xml file:

```
sudo nano .../conf/tomcat-users.xml
```

You will want to add a user who can access the manager-gui and admin-gui (web apps that come with Tomcat). You can do so by defining a user, similar to the example below, between the tomcat-users tags. Be sure to change the username and password to something secure:

```
tomcat-users.xml - Admin User
<tomcat-users . . .>
  <user username="admin" password="password" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

Save and close the file when you are finished. You need to restart the tomcat to adopt the changes.

Let's take a look at the Manager App, accessible via the link or http://server_domain_or_IP:8080/manager/html. You will need to enter the account credentials that you added to the tomcat-users.xml file. Afterwards, you should see a page that looks like this:

Tomcat Web Application Manager

Message:	OK
----------	----

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy	
Deploy directory or WAR file located on server	
Context Path (required): <input type="text"/> XML Configuration file URL: <input type="text"/> WAR or Directory URL: <input type="text"/> <input type="button" value="Deploy"/>	
WAR file to deploy	
Select WAR file to upload <input type="button" value="Choose File"/> No file chosen <input type="button" value="Deploy"/>	

The Web Application Manager is used to manage your Java applications. You can Start, Stop, Reload, Deploy, and Undeploy here. You can also run some diagnostics on your apps (i.e. find memory leaks). Lastly, information about your server is available at the very bottom of this page.

7. Let us test the web pages / JSP pages

Now let us deploy jsp and html and test them. Switch to webapps directory of tomcat and create you won folder like

```
..webspps$mkdir myapps
```

```
..webspps$cd myapps
```

```
..webspps$gedit login1.html
```

```
<html>
  <head>
    <title>SSL demonstration</title>
  </head>
  <body>
    <h3>Web communicaton in plain text- NO SSL</h3>
    <br>
    <form action="http://127.0.0.1:8080/myapps/action.jsp"
      method=post>
```

```
User Name:<br>
<input type="text" name="username" value=""><br>
Password:<br>
<input type="password" name="password"
value=""><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

One more file to respond to above html as action.jsp as follows

```
<html>
  <head>
    <title>Using POST Method to Read Form Data</title>
  </head>

  <body>
    <h3>Returning the user Form Data back</h3>
    <br>

    <p><b>Login name:</b>
      <%= request.getParameter("username") %>
    </p>

    <p><b>Password:</b>
      <%= request.getParameter("password") %>
    </p>
  </body>
</html>
```

...webapps/myapps\$ls

login1.html

action.jsp

Create two files. First file of html is to demonstrate the web form login page. action file is jsp program for response for login html file

8. Now test the web pages as follows.

You will observe that all traffic is plain in format. So far we have not talked about SSL or TLS between client and server.

SSL demonstration x +

127.0.0.1:8080/myapps/login1.html

Web communicaton in plain text- NO SSL

User Name:

Password:

Submit

Using POST Method to x +

127.0.0.1:8080/myapps/action.jsp

Returning the user Form Data back

Login name: bnjagdale

Password: mysecret

9. SSL/TLS installation (Java keytool)

SSL/TLS and Tomcat

It is important to note that configuring Tomcat to take advantage of secure sockets is usually only necessary when running it as a stand-alone web server. Details can be found in the Security Considerations Document. When running Tomcat primarily as a Servlet/JSP container behind another web server, such as Apache or Microsoft IIS, it is usually necessary to configure the primary web server to handle the SSL connections from users. Typically, this server will negotiate all SSL-related functionality, then pass on any requests destined for the Tomcat container only after decrypting those requests. Likewise, Tomcat will return clear text responses, which will be encrypted before being returned to the user's browser. In this environment, Tomcat knows that communications between the primary web server and the client are taking place over a secure connection (because your application needs to be able to ask about this), but it does not participate in the encryption or decryption itself.

Certificates

In order to implement SSL, a web server must have an associated Certificate for each external interface (IP address) that accepts secure connections. The theory behind this design is that a server should provide some kind of reasonable assurance that its owner is who you think it is, particularly

before receiving any sensitive information. While a broader explanation of Certificates is beyond the scope of this document, think of a Certificate as a "digital passport" for an Internet address. It states which organization the site is associated with, along with some basic contact information about the site owner or administrator.

This certificate is cryptographically signed by its owner, and is therefore extremely difficult for anyone else to forge. For the certificate to work in the visitors browsers without warnings, it needs to be signed by a trusted third party. These are called *Certificate Authorities* (CAs). To obtain a signed certificate, you need to choose a CA and follow the instructions your chosen CA provides to obtain your certificate. A range of CAs is available including some that offer certificates at no cost.

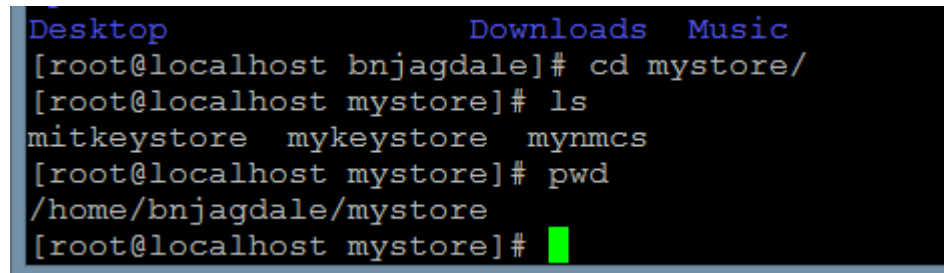
Java provides a relatively simple command-line tool, called `keytool`, which can easily create a "self-signed" Certificate. Self-signed Certificates are simply user generated Certificates which have not been signed by a well-known CA and are, therefore, not really guaranteed to be authentic at all. While self-signed certificates can be useful for some testing scenarios, they are not suitable for any form of production use.

10. Above traffic between server and client is without SSL implementation

1. Now create the digital certificate using java keytool in jdk/bin
2. Issue command as follows.

```
[root@localhost ~]# keytool -genkey -alias mykeys -keyalg RSA -keystore mynmcs
```

In this example keynamed mynmcs is created in working directory and public key pair is also generated in keystore supported by passwords.

A terminal window with a dark background and light blue text. At the top, there are three window titles: 'Desktop', 'Downloads', and 'Music'. The terminal shows a sequence of commands and their outputs. The user is at the root prompt on localhost. They navigate to a directory named 'mystore' using 'cd mystore/'. Then they run 'ls', which lists three files: 'mitkeystore', 'mykeystore', and 'mynmcs'. Next, they run 'pwd', which outputs '/home/bnjagdale/mystore'. Finally, they run another prompt '[root@localhost mystore]#' which is followed by a green cursor.

```
Desktop Downloads Music
[root@localhost bnjagdale]# cd mystore/
[root@localhost mystore]# ls
mitkeystore mykeystore mynmcs
[root@localhost mystore]# pwd
/home/bnjagdale/mystore
[root@localhost mystore]#
```

3. Now configure the digital certificate in server.xml file available in conf directory under apache tomcat directory. As follows


```
root@localhost:/home/bnjagdale/apache-tomcat-7.0.52/conf

connector should be using the OpenSSL style configuration -->

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
  acceptCount="100"
  maxThreads="150" scheme="https" secure="true"
  keystoreFile="/home/bnjagdale/mystore/mymcs"
  keystorePass="secret1" keyPass="secret2"
  clientAuth="false" sslProtocol="TLS" />

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

<!-- An Engine represents the entry point (within Catalina) that
processes
```

- Restart the tomcat server by using command as under tomcat 8 bin directory

```
$sh shutdown.sh
```

```
$sh startup.sh
```

- Now issue the url in client browser as follows

```
https://172.20.1.121:8443/myapps/login2.html
```

SSL demonstration

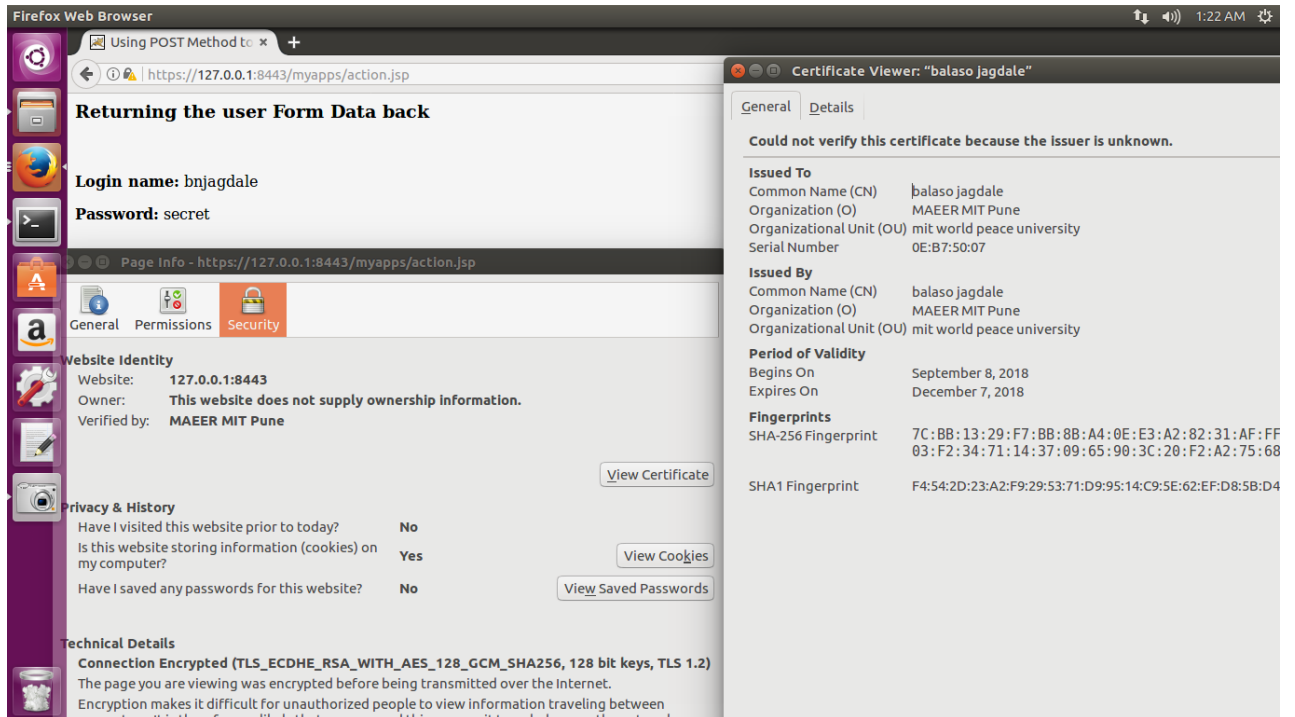
127.0.0.1:8080/myapps/login2.html

Web communication in plain text- With SSL

User Name:

Password:

- Now this form will (submit) initiate ssl dialog and hence communication will be secured now.



7. Certificate warning error is shown as certificate is self-signed. Accept certificate and check the output. So now traffic is secured with SSL between transport and application.
Click on certificate error and see the details. Certificates installed in tomcat server for SSL security are self-signed. If signed by CA like VeriSign, even legality of this communication is applicable. Thus by deploying SSL in tomcat server, websites can be secured to achieve Authentication, integrity and secrecy goals in the communication between web client and web server.
11. Now check the plain traffic by deploying sniffer such as tcpdump or wireshark etc. and verify that traffic is plain without SSL and with https://localhost:8443/... Traffic is secured in term of authentication, secrecy and integrity.

Wireshark network traffic capture showing a web form submission. The packet list shows an HTTP POST request to /myapps/action.jsp. The packet details pane shows the raw data of the request body, which includes a login form with a password field containing 'secret123'.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.1	DNS	90	Standard query 0xc50a A incoming.telemetry.moz
2	0.000202254	127.0.0.1	127.0.1.1	DNS	90	Standard query 0xc50a A incoming.telemetry.m
3	0.034506451	127.0.1.1	127.0.0.1	DNS	347	Standard query response 0xc50a A incoming.telen
4	1.786642227	127.0.0.1	127.0.1.1	DNS	84	Standard query 0xc37f A detectportal.firefox.cc
5	1.786978599	127.0.0.1	127.0.1.1	DNS	84	Standard query 0xc37f A detectportal.firefox
6	1.821804298	127.0.1.1	127.0.0.1	DNS	197	Standard query response 0xc37f A detectportal.f
7	3.850292518	127.0.0.1	127.0.0.1	TCP	74	57382 → 8080 [SYN] Seq=0 Win=43690 Len=0 MSS=65
8	3.850308821	127.0.0.1	127.0.0.1	TCP	74	8080 → 57382 [SYN, ACK] Seq=9 Ack=1 Win=43690 L
9	3.850321100	127.0.0.1	127.0.0.1	TCP	66	57382 → 8080 [ACK] Seq=1 Ack=1 Win=43776 Len=0
10	3.851021216	127.0.0.1	127.0.0.1	HTTP	565	POST /myapps/action.jsp HTTP/1.1 (application/
11	3.851033477	127.0.0.1	127.0.0.1	TCP	66	8080 → 57382 [ACK] Seq=1 Ack=500 Win=44800 Len=
12	3.855557577	127.0.0.1	127.0.0.1	HTTP	587	HTTP/1.1 200 (text/html)

Packet 10 details:

```

    </head>\n
    \n
    <body>\n
    <h3>Returning the user Form Data back</h3>\n
    <br>\n
    \n
    <p><b>Login name:</b>\n
    0140 64 20 46 6f 72 6d 20 44 61 74 61 3c 2f 74 69 74 d Form D ata</tit
    0150 6c 65 3e 0a 20 20 20 3c 2f 68 65 61 64 3e 0a 20 le>. < /head>.
    0160 20 20 0a 20 20 20 3c 62 6f 64 79 3e 0a 20 20 20 . <b ody>.
    0170 20 20 20 3c 68 33 3e 52 65 74 75 72 6e 69 6e 67 <h3>R eturning
    0180 20 74 68 65 20 75 73 65 72 20 46 6f 72 6d 20 44 the use r Form D
    0190 61 74 61 20 62 61 63 6b 3c 2f 68 33 3e 0a 20 20 ata back </h3>.
    01a0 20 20 20 20 3c 62 72 3e 0a 20 20 20 20 20 20 <br> .
    01b0 0a 20 20 20 20 20 20 20 20 20 20 20 3c 70 3e 3c 62 3e . <p><b>
    01c0 4c 6f 67 69 6e 20 6e 61 6d 65 3a 3c 2f 62 3e 0a Login na me:</b>.
    01d0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 bnja
    01e0 67 64 61 6c 65 0a 20 20 20 20 20 20 20 20 20 20 <
    01f0 2f 70 3e 0a 0a 20 20 20 20 20 20 20 20 20 20 3c gdale.
    0200 3e 3c 62 3e 50 61 73 73 77 6f 72 64 3a 3c 2f 62 ><b>Pass word:</b
    0210 3e 0a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 79 >.< my
    0220 73 65 63 72 65 74 31 32 33 0a 20 20 20 20 20 20 secret12 3.
    0230 20 20 20 3c 2f 70 3e 0a 20 20 20 3c 2f 62 6f 64 </p>. </bod
    0240 79 3e 0a 3c 2f 68 74 6d 6c 3e 0a y>.</htm l>.
  
```

As you can see that wire shark has sniffed the web traffic and found what server and client are talking. Here, toward the bottom of screen capture, you can see password in plain format, which you will not be able to sniff after https SSL is deployed.



Wish you joy while experimenting...