Security Workshop

Apache + SSL exercises in Ubuntu

Contents

1 Install apache2 and enable SSL

i. First we install Apache so we can get the SSL certs if you don't have it installed. In our case the server is already installed, but you can type this just to be sure. *which server do you think this should be installed on?*

$ sudo apt-get install apache2

ii. For Ubuntu the next step is to enable SSL and restart apache

$ sudo a2enmod ssl
$ sudo service apache2 restart

2 Generate a Local Certificate

Remember the presentation? We'll use openssl to generate a local server key, local server certificate, a certificate signing request, and a server key that is unencrypted (no passphrase) to allow Apache to start without prompting for a passphrase. This implies that you believe your server to be secure so that others don't steal your unencrypted server key and certificate and use them in nefarious (bad) ways.

Realistically, however, it's not practical to have Apache ask you for a passphrase each time you boot your server. As a matter of fact this could be disastrous if, say, the power were to go out, your server reboots, and then hangs until you physically arrive to the console to type in a passphrase (***no pass phrase***).

Lets create the directory where we'll create our certificate.

$ sudo mkdir /etc/apache2/ssl
$ cd /etc/apache2/ssl

Then use openssl to generate a key:

$ **sudo openssl genrsa -des3 -out server.key 1024**

Pick a passphrase that you'll remember when prompted. Longer is better...

We use triple DES encryption for the key and it's 1024 bits long. Don't make it longer, not all clients will understand how to use a longer key. Now to remove the password from our key so the web server can start SSL without asking us for a key each time.

$ **sudo openssl rsa -in server.key -out server.pem**

And, you'll need to use the passphrase you just created above to do this.
Before you can generate a certificate you need to create a CSR file, a Certificate Signing Request. Below is the command and a sample session you can use as an example to create your own local certificate. Note, that **common name** is the name of the server (dmzX.sns.tznog.or.tz, etc.). This is important:
First the command:

$ sudo openssl req -new -key server.key -out server.csr

And the sample session:

Country Name (2 letter code) [AU]:tz
State or Province Name (full name) [Some-State]: (leave blank)
Locality Name (eg, city) []:Dodoma
Organization Name (eg, company) [Internet Widgits Pty Ltd]:TzNOG
Organizational Unit Name (eg, section) []:Workshop
Common Name (eg, YOUR name) []:pcX.sns.tznog.or.tz
Email Address []:tznog@pcX.sns.tznog.or.tz

Please enter the following extra attributes
to be sent with your certificate request
A challenge password []: (leave blank)
An optional company name []: (leave blank)

Now we can sign our own certificate with our own private key as we are not sending this CSR to a CA for a commercially signed certificate. You'll need the private key passphrase again (***no pass phrase***).

$ **sudo openssl x509 -req -days 60 -in server.csr -signkey server.key -out server.crt**

Note that the certficiate is only valid for 60 days. You can choose whatever number you like. This might be a typical act if you were waiting for a signed certificate from a CA, but needed to have something right away.

If it all works you should see something like this:

Signature ok
subject=/C=tz/ST=Some-
State/L=Dodoma/O=TzNOG/OU=Workshop/CN=pcX.sns.tznog.or.tz/emailAddr
Getting Private key
Enter pass phrase for server.key:

3 Configure Apache to use the new certificate

Now that we have our own locally signed certificate in /etc/apache22/ssl we still need to configure Apache to actually use this certificate. Right now Apache is configured only to listen to port 80.

We shall leave that running but add an SSL vhost with its own settings.

Debian based distributions do this by having sample configurations in /etc/apache2/sites-available which you enable by linking in /etc/apache2/sites-enabled.

This is to make administration of VirtualHosts easier.

$ **sudo a2ensite default-ssl**

$ sudo a2enmod ssl

$ **sudo vi /etc/apache2/sites-enabled/default-ssl**

(Check here that **SSL v3 is disabled** …./etc/apache2/mods-available/ssl.conf)

$ sudo service apache2 reload

Look for the lines that start with the following and change them to match what we have. Now scroll down a bit further until you find the line that reads:

SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem

Comment that line out ("#" character at front of line), and below it add the following line:

SSLCertificateFile /etc/apache2/ssl/server.crt

Now scroll down and find:

SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

Comment this out and below it add the line:

SSLCertificateKeyFile /etc/apache2/ssl/server.pem

Note we used the ".pem" file as this is your server.key file, but without a passphrase.
Save the file and exit from it at this point (in **vi :wq**).

$ sudo apachectl configtest

AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using ::1. Set the 'ServerName' directive globally to suppress this message

Syntax OK

Restart apache using the Ubuntu system tools but in general apache ships with the script apachectl.

$ **sudo service apache2 reload**

If you run in to any errors try using the log files in /var/log/ to troubleshoot the problem.
The important files are:

/var/log/syslog
/var/log/apache-error.log
/var/log/ssl_engine_log (*depends setting in Vhost*)

If you've run in to problems or don't understand something let the instructor or an assistant know.

Congratulations, you now have a secure web server up and running.

4 Verify that http and https (Apache) are Working

For this exercise you can be any user.

This is very simple. In the web browser of your choice go the to the following address:

http://10.10.0.X/

You should have gotten the page with text:

Index of /

Now go to this address:

https://10.10.0.X/

Depending on what version of Firefox you are using you will see different dialogues warning you that the certificate is not trusted. Go through the process of accepting the certfificate. When you get the chance to view the certificate take a look. You see the information you filled in when generating the CSR (what's that stand for?).

We'll discuss what just happened for a bit, and we'll take a look at a web browser, built-in trusted CA's, etc.

5 Advanced Verification Methods (Optional exercise)

At the most simple level let's verify that the Apache web server daemon appears to be running. We can use the *ps* command to do this:

```
$ ps auxw | grep apache2
```

The output you should see will look something like this:

```
root 20109 0.0 1.5 34988 7736 ? Ss 07:36 0:00 /usr/sbin/apache2
-k start www-data 20114 0.0 0.7 35012 3984 ? S 07:36 0:00
/usr/sbin/apache2 -k start www-data 20115 0.0 0.7 35012 3984 ? S
07:36 0:00 /usr/sbin/apache2 -k start www-data 20116 0.0 0.7 35012
3984 ? S 07:36 0:00 /usr/sbin/apache2 -k start www-data 20117 0.0
0.7 35012 3984 ? S 07:36 0:00 /usr/sbin/apache2 -k start www-data
20118 0.0 0.7 35012 3984 ? S 07:36 0:00 /usr/sbin/apache2 -k start
root 20175 0.0 0.1 3904 812 pts/1 S+ 07:40 0:00 grep –color=auto
apache2
```

Note that Apache runs with multiple instances of the httpd daemon. This is so that the web server can rspond to multiple requests more efficiently. Also notice that the first httpd daemon that starts runs as root, but subsequent daemons use the user "www-data" - This is to make the web server less vulnerable to attacks that might gain root access.

So, this shows you that Apache is running, but is it accessible to users with web browsers? It's possible you might be on a machine in the future and not have a web browser available, even though the machine is running a web server.

You can use telnet to verify if the web server is available. To do this type:

```
$ telnet 127.0.0.1 80
```

If you get back something like:

```
Trying 127.0.0.1... Connected to 127.0.0.1. Escape character is '^]'.
```

This is a good indication that you have a web server working. Still, to be sure that this is not some other server running on port 80 you could go a step further. You can view the initial web server page on port 80 by doing this:

1. quit the telnet session if it's still running

```
^]
telnet> quit                    [press CTRL key and ] character to exit]
```

1. Go to your home directory and try this again but saving the output to a file

```
$ cd                            # [to go your home directory]
$ script apache.txt             # [use UNIX script utility to save session to a file]
$ telnet 127.0.0.1 80

GET / HTTP/1.0                  [press ENTER]
Host: localhost                 [press ENTER twice]

$ exit                          [to leave your script shell]
```

And you will see the initial Apache welcome page scroll by on your screen. Now that you saved the output of this session to the file ~/apache.txt) we can get some additional information.

Type the file apache.txt to your screen by doing:

```
$ less apache.txt                    [remember, "q" to exit the less screen]
```

In the first page of information presented you should see something lie:

HTTP/1.1 200 OK Date: Fri, 02 Aug 2013 07:45:45 GMT Server: Apache/2.2.22 (Ubuntu) Vary: Accept-Encoding Content-Length: 525 Connection: close Content-Type: text/html;charset=UTF-8

Notice that you can now see exactly what version of Apache is running.

So, it appears that Apache is ssl-enabled on this machine, but how can we prove this? A web server with ssl support means that you can go to URL addresses that start with “https” (http secure).

We’ll use a tool that comes with OpenSSL to allow us to make ssl connections, verify encryption in use, view certificates, etc. You can simply type “openssl” and then you will get a prompt where you can use the multiple openssl tools, or you can combine the command “openssl” with the various tools on your command line. This is what we will do

using the openssl s_client tool. Try typing these commands:

```
$ cd
$ script ssltest.txt
$ openssl s_client -connect localhost:443
[Press ctrl-c to exit the information screen]
$ exit
$ less ssltest.txt
```

And you will get several screens of information about your Apache web server, the ssl certificate that is currently installed and it’s detailed information, what protocols are in use, and more.

In most cases this is overkill and you can simply use a web browser to verify functionality, but having alternatives is always nice.