Exercises: FreeBSD: Apache and SSL: SANOG 9 IP Services Workshop

January 14, 2007

Exercises

Some notes:

- Remember, "#" implies you need to be root, while "$" means you can work as a general user.
- Your instructor will do exercises 1 through 3 with you in class, and then we'll discuss what it all means after exercise 3.

1.) Install Apache with SSL support [Top]

For this exercise you need to be root user.

We are going to install the Apache web server with Secure Socket Layer (ssl) support using the mod_ssl Apache module. We'll do this by using a package that we have available to us. We could use the ports system, but we are not planning on making any configuration changes, so there's no need to use ports in this case.

Let's get started:

```
# cd /usr/ports/www/apache13-modssl
# make install
```

The package creates an initial digital certificate that is not signed and installs this for use with Apache.

Important Apache files can be found here:

- /usr/local/sbin/httpd [Apache server binary]
- /usr/local/etc/apache/ [Apache configuration files]
- /usr/local/etc/rc.d/apache.sh [Apache startup script]

Type the following:

```
# man httpd
```

And read until the end of the man page. At the end is a list are some more files that are important to the Apache web server. These include:

```
/usr/local/apache/conf/httpd.conf
/usr/local/apache/conf/srm.conf
/usr/local/apache/conf/access.conf
/usr/local/apache/conf/mime.types
/usr/local/apache/conf/magic
/usr/local/apache/logs/error_log
/usr/local/apache/logs/access_log
/usr/local/apache/logs/httpd.pid
```

At this point Apache is installed, but not running. Our next exercise will be to configure your machine to automatically start Apache at boot, and to start and stop Apache manually. We'll be using some of the files listed above for these exercises.

2.) Configure Apache to start at boot [Top]

For this exercise you need to be root.

****

Remember on Day 1 that you learned that third party software might install their startup scripts in /usr/local/etc/rc.d? This is the case for the Apache web server. If you look in /usr/local/etc/rc.d/ you will see a file named "apache.sh". This file is the startup script for the Apache web server.

Remember that startup scripts look in your various rc.conf files to see whether they have been enabled. This is the same with the startup script /usr/local/etc/rc.d/apache.sh. First have a look at this script to understand where it is looking to see if it should execute at system startup:

```
# less /usr/local/etc/rc.d/apache.sh
```

The first screen of information includes the follwing:

```
# Define these apache_* variables in one of these files:
#       /etc/rc.conf
#       /etc/rc.conf.local
#       /etc/rc.conf.d/apache
#
# DO NOT CHANGE THESE DEFAULT VALUES HERE
#
apache_enable="NO"
apache_flags="-DSSL"
apache_pidfile="/var/run/httpd.pid"
```

As you can see this script will check in /etc/rc.conf then /etc/rc.conf.local (deprecated), and then /etc/rc.conf.d/apache to see if Apache has been enabled.

Now let's edit the file /etc/rc.conf and add the line that will tell Apache to start each time you boot your machine:

```
# vi /etc/rc.conf
```

And, in this file you should add a line at the bottom of the file that reads:

```
apache_enable="YES"
```

Now that the file /etc/rc.conf includes the line apache_enable="YES" you should be able to use the script apache.sh in /usr/local/etc/rc.d/ to start, stop, verify, etc. the Apache server.

Let's start the Apache web server by doing:

```
# /usr/local/etc/rc.d/apache.sh start
```

If you get an error message about a proper domain name this is because your machine does not have a proper name assigned to it. This should be rectified during the DNS section of this workshop.

If you want to see the various things you can do with this script just type:

```
# /usr/local/etc/rc.d/apache.sh
```

And you should see:

```
Usage: /usr/local/etc/rc.d/apache.sh [fast|force|one](start stop restart rcvarstatus poll)
```

3.) Verify that http and https (Apache) are Working [Top]

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://localhost/
```

You should have gotten the page with the text:

```
Hey, it worked !
The SSL/TLS-aware Apache webserver was
successfully installed on this website.
```

Now go to this address, and be sure you press "View Certificate" when presented with the Server Certificate Expired dialogue:

```
https://localhost/
```

You should see, under the General tab, a bunch of information pertaining to the Snake Oil, Ltd. CA. ("Snake Oil" in English refers to an item being sold that's really bad).

Now click on the "Close" button and then click on "OK". You'll get another prompt asking you what you'd like to do with the certificate. For now, keep the default choice of "Accept this certificate temporarily for this session", press "OK" again if you receive another dialogue, and then you should see the same main page as before except that your browser will indicate in some manner that you now have a "secure session" in place. Probably by a closed lock in the lower, right corner of the browser window, or by highlighting the "https" URL address, or both.

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. After that we'll generate our own, signed certificate.

4.) Generate a Local Certificate [Top]

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.

For these exercises you need to be root.

Lets create our own self signed certificate:

```
# cd /usr/local/etc/apache
```

We already have a local certificate (by Snake Oil, Ltd.), so let's not overwrite this. We'll create our certificate in a different subdirectory:

```
# mkdir mycert
# cd mycert
```

First use openssl to generate a key:

```
# 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:

```
# 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 (pcN.ws.sanog.org.bt, etc.). This is important:

First the command:

```
# openssl req -new -key server.key -out server.csr
```

And the sample session:

```
Country Name (2 letter code) [AU]:lk
State or Province Name (full name) [Some-State]:   (can be blank)
Locality Name (eg, city) []:Colombo
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SANOG
Organizational Unit Name (eg, section) []:Workshop
Common Name (eg, YOUR name) []:pcN.ips.ws.sanog.lk
Email Address []:hervey@nsrc.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:   (Useful if you want to verify the CA!)
An optional company name []:
```

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.

```
# 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=lk/ST=Some-State/L=Colombo/O=SANOG/OU=Workshop/CN=pcN.ips.ws.sanog.lk/emailAddress=hervey@nsrc.
Getting Private key
Enter pass phrase for server.key:
```

Part Two

Now that we have our own locally signed certificate in /usr/share/etc/apache/mycert we still need to configure Apache to actually use this certificate. Right now Apache is looking for the server.key and server.crt files located in /usr/local/etc/apache when starting. We're going to edit the file /usr/local/etc/apache/httpd.conf and change where the Apache server looks. We'll use vi to edit httpd.conf in our example, but you can use whatever editor you choose:

```
# vi /usr/local/etc/apache/httpd.conf
```

Search for the line in the file that reads:

```
## SSL Virtual Host Context
```

Note that the https server is just one instance of a virtual host. Now find the following line (just scroll down a bit):

```
SSLCertificateFile /usr/local/etc/apache/ssl.crt/server.crt
```

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

```
SSLCertificateFile /usr/local/etc/apache/mycert/server.crt
```

Now scroll down and find:

```
SSLCertificateKeyFile /usr/local/etc/apache/ssl.key/server.key
```

Comment this out and below it add the line:

```
SSLCertificateKeyFile /usr/local/etc/apache/mycert/server.pem
```

Note we used the ".pem" file as this is your server.key file, but without a passphrase.

Save the httpd.conf file and exit from it at this point.

You can now stop and start your Apache web server using a facility called `apachectl`. This is the Apache HTTP server control interface. See `man apachectl` for more information.

To start and stop the server do this (You may need to run "`rehash`" first):

```
# apachectl stop
# apachectl startssl
```

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/messages
/var/log/httpd-error.log
/var/log/ssl_engine_log
```

Part Three

To finish this excercise please connect to your server using both standard http and http with ssl (https) by using a web browser and connecting to:

```
http://localhost/
https://localhost/
```

Go ahead and test some of your neighbor's machines if you want to see what their local certificates look like. Just type in their IP address using "http" and "https".

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.

5.) Advanced Verification Methods (Optional exercise) [Top]

For this exercise you can use your standard *username* account.

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 httpd
```

Remember that Apache uses the actual binary file /usr/local/sbin/httpd to start the Apache web server as indicated by the final message during installation. That's why we grep'ed on "httpd" instead of "apache".

The output you should see will look something like this:

```
root     54884  0.0  0.9  5224 3296  ??  Ss   12:43PM   0:00.22 /usr/local/sbin/httpd -DSSL
www      54885  0.0  0.9  5264 3328  ??  I    12:43PM   0:00.01 /usr/local/sbin/httpd -DSSL
```

```
www      54886  0.0  0.9  5264 3328  ??  I    12:43PM   0:00.01 /usr/local/sbin/httpd -DSSL
www      54887  0.0  0.9  5248 3324  ??  I    12:43PM   0:00.01 /usr/local/sbin/httpd -DSSL
www      54888  0.0  0.9  5248 3328  ??  I    12:43PM   0:00.01 /usr/local/sbin/httpd -DSSL
www      54889  0.0  0.9  5248 3324  ??  I    12:43PM   0:00.01 /usr/local/sbin/httpd -DSSL
www      54890  0.0  0.9  5240 3308  ??  I    12:43PM   0:00.00 /usr/local/sbin/httpd -DSSL
root     54951  0.0  0.2  1476  796  p5  S+   12:59PM   0:00.01 grep httpd
```

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" - 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? To check for this you can take advantage of the "Lynx" text-based web browser that you installed on the first day. To do this type:

```
$ lynx 127.0.0.1
```

To exit from Lynx you press "q" and then you answer "y" to the prompt "Are you sure you want to quit?"

Now, what if you did not have the Lynx text-based web browser package installed? When you first installed FreeBSD this was not installed. 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 pcN.ws.sanog.org.bt.
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:

```
^]                      [press CTRL key and ']' character to exit]

$ cd                    [to go your home directory]

$ script apache.txt     [use FreeBSD 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:

```
$ cd                  [to go your home directory]

$ less apache.txt
```

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

```
HTTP/1.1 200 OK
Date: Wed, 06 Jul 2005 06:01:45 GMT
Server: Apache/1.3.33 (Unix) mod_ssl/2.8.22 OpenSSL/0.9.7e
Content-Location: index.html.en
Vary: negotiate,accept-language,accept-charset
TCN: choice
Last-Modified: Mon, 04 Apr 2005 01:26:24 GMT
ETag: "f2027-a71-425097c0;42cae5bd"
Accept-Ranges: bytes
Content-Length: 2673
Connection: close
Content-Type: text/html
Content-Language: en
```

```
Expires: Wed, 06 Jul 2005 06:01:45 GMT
```

Notice that you can now see exactly what version of Apache is running, that it appears to be ssl-enabled and it is using OpenSSL 0.9.7d and mod_ssl to do this.

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 could try this:

```
$ lynx https://localhost/
```

But, you will receive an error message that reads:

```
Alert!: This client does not contain support for HTTPS URLs.
lynx: Can't access startfile https://noc/
```

That is because we did not install the Lynx package that includes ssl support. We could do this now, but instead 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 ENTER if your screen pauses]

$ 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.

[Return to Top]

Last modified: Sat Jan 13 04:52:10 IST 2007