Exercises: SSL and Troubleshooting: AfNOG 2007

April 25, 2007

Exercises

1. Generate a Local Certificate
2. Advanced Verification Methods (Optional Exercise)

Some notes:

- Remember, "#" implies you need to be root, while "$" means you can work as a general user.
- You will install Apache after creating your local certificate, then you will do exercise #2.

1.) 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.

We are going to do this a bit backwards and create a certificate before installing Apache. We will, then, use this certificate after you have installed Apache. So, we will cheat a little bit and create a directory in which to do our work and place the certificate in anticipation of Apache being installed shortly.

First create the direct we want to use:

```
# mkdir -p /usr/local/etc/apache22/mycert
```

If you don't know what the "-p" is doing read `man mkdir`.

Now, we create our own self-signed certificate using openssl:

```
# cd /usr/local/etc/apache22/mycert
# openssl genrsa -des3 -out server.key 2048
```

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

We use triple DES encryption for the key and it's 2048 bits long.

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.e0.ws.afnog.org, 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]:in
State or Province Name (full name) [Some-State]: Federal Capital Territory
(can be blank)
Locality Name (eg, city) []:Abuja
Organization Name (eg, company) [Internet Widgits Pty Ltd]:AfNOG
Organizational Unit Name (eg, section) []:e0
Common Name (eg, YOUR name) []:pcN.e0.ws.afnog.org
Email Address []:email@afnog.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=in/ST=Federal-Capital-
Territory/L=Abuja/O=AfNOG/OU=e0/CN=pcN.e0.ws.afnog.org/emailAddress=email@a
fnog.org
Getting Private key
Enter pass phrase for server.key:
```

Part Two

Now that we have our own locally signed certificate in /usr/share/etc/apache22/mycert we still need to configure Apache to actually use this certificate. You will do this next.

2.) 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. 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.e0.ws.afnog.org
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] [Be sure to use UPPERCASE]

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: Mon, 23 Apr 2007 22:58:30 GMT
Server: Apache/2.2.3 (FreeBSD) mod_ssl/2.2.3 OpenSSL/0.9.7e-p1 DAV/2
Last-Modified: Mon, 23 Apr 2007 08:35:46 GMT
ETag: "a6c4c-235-8e323c80"
Accept-Ranges: bytes
Content-Length: 565
Connection: close
Content-Type: text/html
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 0.9.7e 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 might an error message that reads:

```
Alert!: This client does not contain support for HTTPS URLs.

lynx: Can't access startfile https://noc/
```

This happens if you did not install the Lynx package that includes ssl support. We could do this now if necessary, 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]
