Lab 5: Securing Apache Web Server

Objectives:

- To setup a secure web server using Apache and digital certificates.

Submission:

- Checkpoints that need to be shown to the course teacher.

Instruction:

In this lab, you will setup a secure web server using Apache and digital certificates. As you are probably aware, Apache is the most widely-used web server in the world. However, the HTTP protocol has no security built into it. To handle this issue, a security overlay has been introduced in the transport layer in 1994. It is known as Secure Sockets Layer (SSL). Later it was transformed into Transport Layer Security. SSL/TLS utilises cryptographic mechanisms to create a secure connection within an unprotected network such as the Internet. Combined with the HTTP (Hypertext Transfer Protocol, the de-facto protocol for web), the SSL/TLS introduces the notion of HTTPS (HTTP Secure).

You will study HTTPS in details in our the security lecture. However, for this lab, we present a brief overview of HTTPS next. The HTTPS protocol is illustrated in the following figure.

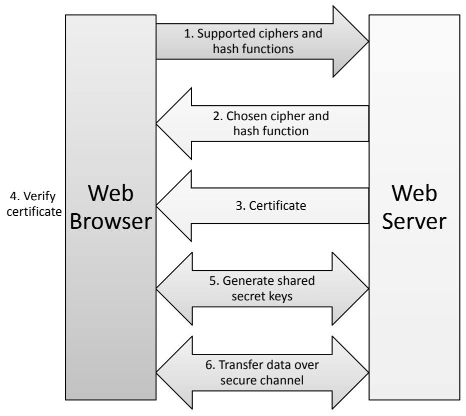
1. Supported ciphers and hash functions
2. Chosen cipher and hash function
4. Verify certificate
Web Browser
3. Certificate
Web Server
5. Generate shared secret keys
6. Transfer data over secure channel

Figure 1: HTTPS Protocol

The process is very briefly described below. To establish an HTTPS session, a web browser sends a request to a web server with the list of supported ciphers and hash functions by the browser. The web server chooses a cipher from the list and sends a response back to the web server with its choice along with a digital certificate indicating its identity. The web browser confirms the identity by verifying the certificate. The certificate also contains a public key of the web server. The public key is then used to encrypt a secret value, which is then sent to the web server. The web server decrypts the secret value with its private key. Thus, a shared secret is established between the web browser and the server. This key is then used to utilise a symmetric encryption between the browser and the server for any subsequent communication.

Thus, one of the crucial steps in setting up a secure web server is to create a digital certificate. In this lab, you will perform different tasks to setup a secure web server using Apache and a digital certificate. ***Complete the checkpoints and show it to your teacher.***

Task-1: Becoming a certificate authority

The first step for a secure web server is to setup an Apache web server. In your first web lab, you have already configured the apache server for different virtual hosts: example.com and webserverlab.com. Before proceeding the following, please ensure that you can access these virtual hosts using your preferred browser. You will secure these virtual hosts in today's lab by using TLS.

TLS is dependent on a Certificate Authority (CA) which is a trusted entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. A number of commercial CAs are treated as root CAs; VeriSign is the largest CA at the time of writing. Users who want to get digital certificates issued by the commercial CAs need to pay those CAs.

In this lab, you will need to create digital certificates, but you will not be going to pay to any commercial CA. You will become a root CA, and then use this CA to issue certificate for others (e.g. servers). In this task, you will make yourself a root CA, and generate a certificate for this CA. Unlike other certificates, which are usually signed by another CA, the root CA's certificates are self-signed. Root CA's certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on certificate-based security. Root CA's certificates are unconditionally trusted.

For this, you will use OpenSSL which are already familiar with from Lab-3. For your convenience, the manual from Lab 3 is provided in the folder. Please ensure that openssl is installed in your machine by following the instructions from Lab 3 manual.

To start this task, create a folder for this task and cd into it. In this folder, you will need to create a particular configuration file as discussed below.

The Configuration File openssl.conf: In order to use OpenSSL to create certificates, you have to have a configuration file. The configuration file usually has an extension *.cnf*. It is used by three OpenSSL commands: *ca*, *req* and *x509*. The manual page of *openssl.conf* can be found using Google search. You can also get a copy of the configuration file from */usr/lib/ssl/openssl.cnf*. After copying this file into your current directory, you need to create several sub-directories as specified in the configuration file (look at the [CA default] section):

```
dir             = ./demoCA          # Where everything is kept
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
new_certs_dir   = $dir/newcerts     # default place for new certs.

database        = $dir/index.txt    # database index file.
serial          = $dir/serial       # The current serial number
```

For the *index.txt* file, simply create an empty file. For the *serial* file, put a single number in string format (e.g. 1000) in the file. Once you have set up the configuration file *openssl.cnf*, you can create and issue certificates.

Certificate Authority (CA): As described before, you need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

- `$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf`

You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two files: *ca.key* and *ca.crt*. The file *ca.key* contains the CA's private key, while *ca.crt* contains the public-key certificate.

Creating a certificate for example.com

After becoming a root CA, you are ready to sign digital certificates for our customers. Our first customer is a company called *example.com*. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 1: Generate public/private key pair. The company needs to first create its own public/private key pair. You can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file server.key:

- `$ openssl genrsa -des3 -out server.key 1024`

Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generates a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use example.com as the common name of the certificate request.

- `$ openssl req -new -key server.key -out server.csr -config openssl.cnf`

Step 3: Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, you will use our own trusted CA to generate certificates:

- `$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf`

If OpenSSL refuses to generate certificates, it is very likely that the names in your requests do not match *with* those of CA. Fix this and re-issue the above command.

Next, let us launch a simple web server with the certificate generated in the previous task. OpenSSL allows us to start a simple web server using the s_server command. Use the following steps:

Step 1: Combine the secret key and certificate into one file

- `$ cp server.key server.pem`
- *cat server.crt >> server.pem*

Step 2: Launch the web server using server.pem

- `$ openssl s_server -cert server.pem -www`

By default, the server will listen on port 4433. You can alter that using the -accept option. Now, you can access the server using the following URL: https://example.com:4433/. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: *"example.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown".*

Had this certificate been assigned by VeriSign, you will not have such an error message, because VeriSign's certificate is very likely preloaded into Firefox's certificate repository already. Unfortunately, the certificate of example.com is signed by our own CA (i.e., using ca.crt), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA's self-signed certificate.

You can request Mozilla to include our CA's certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so you will not pursue this direction.

Load ca.crt into Firefox: You can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

- Preference -> Advanced -> View Certificates

You will see a list of certificates that are already accepted by Firefox. From here, you can "import" our own certificate. Please import ca.crt, and select the following option: "Trust this CA to identify web sites". You will see that our CA's certificate is now in Firefox's list of the accepted certificates. Now, point the browser to https://example.com:4433.

Checkpoint – 1 (5 marks): Show this to your course teacher and explain what is happening. Since example.com points to 127.0.0.1, you can also use https://localhost:4433 to load a web page shown by the OpenSSL server. Please do so, describe and explain your observations.

Checkpoint – 2 (5 marks): Follow the same instructions for webserverlab.com and show this to your course teacher.

Task-3: Deploy HTTPS into Apache

Now, you will deploy the HTTPS capability into Apache web server. At first, stop the Openssl webserver launched in the previous task. Now add the following lines into the example configuration file:

/etc/apache2/sites-available/example.com.conf

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
   ServerAdmin admin@example.com
   ServerName example.com
   ServerAlias www.example.com
   DocumentRoot /var/www/example.com/html
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined

   SSLEngine on
   SSLCertificateFile /path/to/your .crt certificate file
   SSLCertificateKeyFile /path/to/your_private.key
</VirtualHost>
</IfModule>
```

Apache is quite modular in the sense it supports the development of additional module which can add extended functionalities. For this lab, you will need to enable the ssl module in Apache which might not be enabled by default. Use the following command to enable the ssl module.

- `$ sudo a2enmod ssl`

Next, use the following command to test the configuration.

- `$ sudo apache2ctl configtest`

If a syntax is displayed onto the terminal, it indicates everything is okay.

Next restart the apache server using the restart command shown above.

Now, try to access the https://example.com. If everything is properly configured, you should be able to view the webpage in HTTPS.

If your browser is Firefox and it shows a warning, you can fix it by importing the CA certificate as described previously. If you use Chrome and it shows a similar warning, you can also import the CA certificate from the Manage certificate option under the Advanced setting in Chrome.

Checkpoint – 3 (5 marks): Access the https://example.com in your browser and show it to your teacher.

Checkpoint – 4 (5 marks): Set it up for webserverlab.com. Access it via HTTPS and show it to your teacher.