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. Apache is the most widely-used web servers in the world. In fact, it is one of the most widely used open source software 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).

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

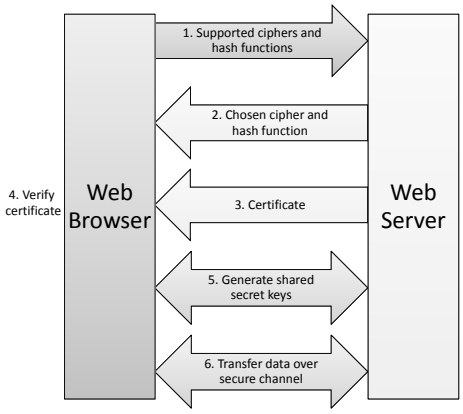


Figure 1: HTTPS Protocol

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: Setting up an Apache web server

The first step to setup a secure web server is to setup an Apache web server. The following is a tutorial collected from: https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-18-04.

Step 1 — Installing Apache

Apache is available within Ubuntu's default software repositories, making it possible to install it using conventional package management tools.

Let's begin by updating the local package index to reflect the latest upstream changes. If *apt* is not recognised as a command, try ***apt-get*** instead of *apt*.

- *sudo apt update*

Then, install the apache2 package:

- *sudo apt install apache2*

After confirming the installation, apt will install Apache and all required dependencies.

Step 2 — Adjusting the Firewall

Before testing Apache, it's necessary to modify the firewall settings to allow outside access to the default web ports. Assuming that you followed the instructions in the prerequisites, you should have a UFW firewall configured to restrict access to your server.

During installation, Apache registers itself with UFW to provide a few application profiles that can be used to enable or disable access to Apache through the firewall.

List the *ufw* application profiles by typing:

- *sudo ufw app list*

You will see a list of the application profiles:

```
Output
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

As you can see, there are three profiles available for Apache:

- **Apache**: This profile opens only port 80 (normal, unencrypted web traffic)
- **Apache Full**: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
- **Apache Secure**: This profile opens only port 443 (TLS/SSL encrypted traffic)

It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80:

- *sudo ufw allow 'Apache'*

You can verify the change by typing:

- sudo ufw status

You should see HTTP traffic allowed in the displayed output:

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

As you can see, the profile has been activated to allow access to the web server.

Step 3 — Checking your Web Server

At the end of the installation process, Ubuntu 18.04 starts Apache. The web server should already be up and running.

Check with the *systemd init system* to make sure the service is running by typing:

- sudo systemctl status apache2

```
Output
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Tue 2018-04-24 20:14:39 UTC; 9min ago
 Main PID: 2583 (apache2)
    Tasks: 55 (limit: 1153)
   CGroup: /system.slice/apache2.service
           ├─2583 /usr/sbin/apache2 -k start
           ├─2585 /usr/sbin/apache2 -k start
           └─2586 /usr/sbin/apache2 -k start
```

As you can see from this output, the service appears to have started successfully. However, the best way to test this is to request a page from Apache.

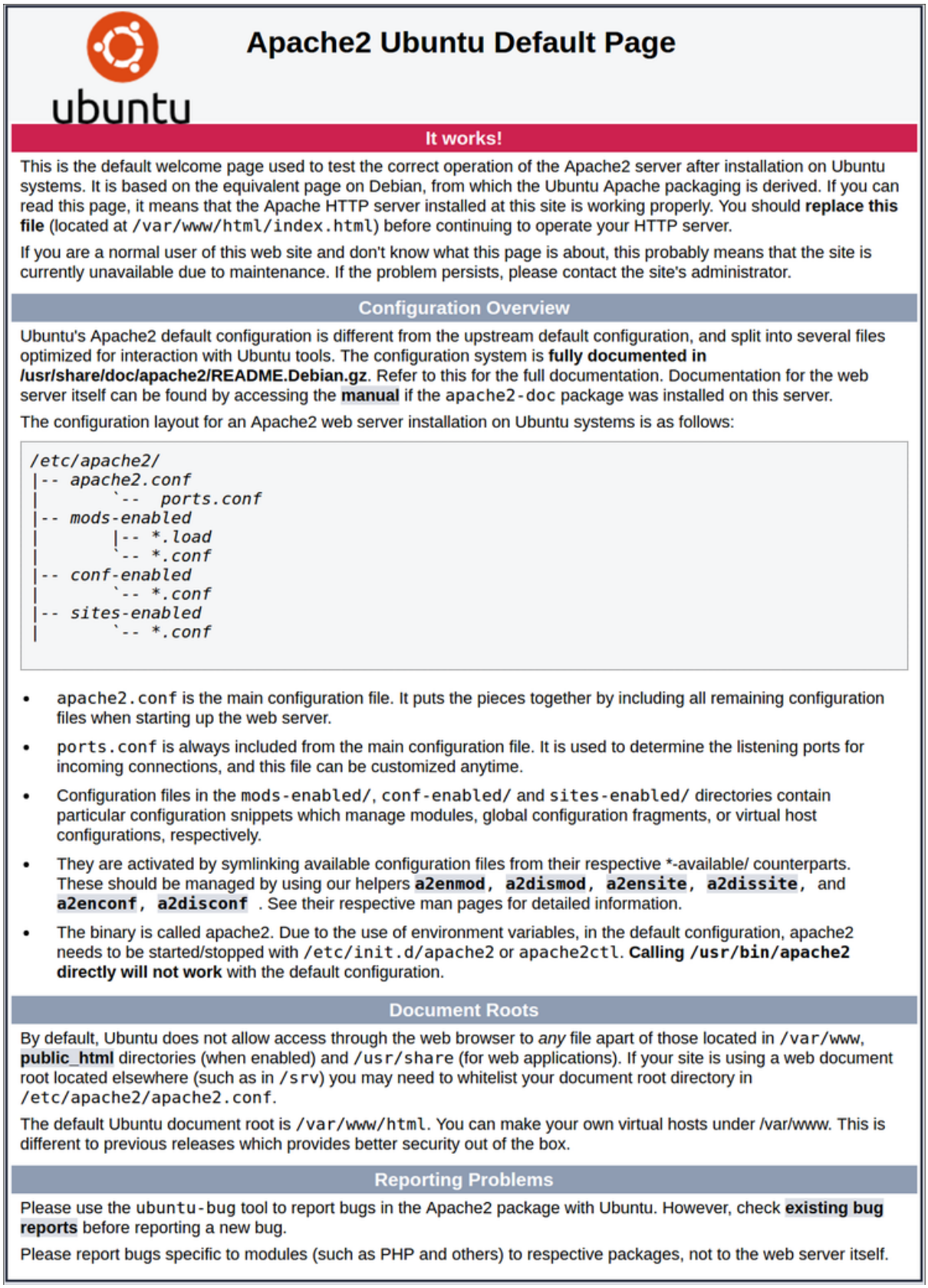
You can access the default Apache landing page to confirm that the software is running properly through your IP address or by just typing *localhost (127.0.0.1)* in the browser. Let us use pkilabserver.com as our domain name. To get our computers recognize this domain name, let us add the following entry to /etc/hosts; this entry basically maps the domain name pkilabserver.com to our localhost (i.e., 127.0.0.1):

- *127.0.0.1 pkilabserver.com*

Now, to check the installation of Apache, enter this domain or its IP address into your browser's address bar:

http://pkilabserver.com or http://localhost or http://127.0.0.1

You should see the default Apache web page:



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/ counterparts. These should be managed by using our helpers **`a2enmod`**, **`a2dismod`**, **`a2ensite`**, **`a2dissite`**, and **`a2enconf`**, **`a2disconf`** . See their respective man pages for detailed information.
- The binary is called apache2. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

By default, Ubuntu does not allow access through the web browser to *any* file apart of those located in `/var/www`, **public_html** directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Ubuntu document root is `/var/www/html`. You can make your own virtual hosts under /var/www. This is different to previous releases which provides better security out of the box.

Reporting Problems

Please use the `ubuntu-bug` tool to report bugs in the Apache2 package with Ubuntu. However, check **existing bug reports** before reporting a new bug.

Please report bugs specific to modules (such as PHP and others) to respective packages, not to the web server itself.

This page indicates that Apache is working correctly. It also includes some basic information about important Apache files and directory locations.

Checkpoint – 1: Show this to your course teacher.

Step 3 — Managing the Apache Process

Now that you have your web server up and running, let's go over some basic management commands.

To stop your web server, type:

- *sudo systemctl stop apache2*

To start the web server when it is stopped, type:

- *sudo systemctl start apache2*

To stop and then start the service again, type:

- *sudo systemctl restart apache2*

If you are simply making configuration changes, Apache can often reload without dropping connections. To do this, use this command:

- *sudo systemctl reload apache2*

By default, Apache is configured to start automatically when the server boots. If this is not what you want, disable this behavior by typing:

- *sudo systemctl disable apache2*

To re-enable the service to start up at boot, type:

- *sudo systemctl enable apache2*

Apache should now start automatically when the server boots again.

Step 5 — Setting Up Virtual Hosts

When using the Apache web server, you can use *virtual hosts* (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up another domain called **example.com**.

Apache generally has one server block enabled by default that is configured to serve documents from the /var/www/html directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying /var/www/html, let's create a directory structure within /var/www for our **example.com** site, leaving /var/www/html in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for **example.com** as follows, using the -p flag to create any necessary parent directories:

- *sudo mkdir -p /var/www/example.com/html*

Next, assign ownership of the directory with the $USER environmental variable:

- *sudo chown -R $USER:$USER /var/www/example.com/html*

The permissions of your web roots should be correct if you haven't modified your unmask value, but you can make sure by typing:

- *sudo chmod -R 755 /var/www/example.com*

Next, create a sample index.html page using *nano* or your favorite editor:

- *nano /var/www/example.com/html/index.html*

Inside, add the following sample HTML:

/var/www/example.com/html/index.html

```
<html>
  <head>
    <title>Welcome to Example.com!</title>
  </head>
  <body>
    <h1>Success!  The example.com server block is working!</h1>
  </body>
</html>
```

Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at /etc/apache2/sites-available/000-default.conf directly, let's make a new one at /etc/apache2/sites-available/example.com.conf:

- *sudo nano /etc/apache2/sites-available/example.com.conf*

Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:

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

```
<VirtualHost *:80>
  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
</VirtualHost>
```

Notice that we've updated the DocumentRoot to our new directory and ServerAdmin to an email that the **example.com** site administrator can access. We've also added two directives: ServerName, which establishes the base domain that should match for this virtual host definition, and ServerAlias, which defines further names that should match as if they were the base name.

Save and close the file when you are finished.

Let's enable the file with the a2ensite tool:

- *sudo a2ensite example.com.conf*

Disable the default site defined in 000-default.conf:

- *sudo a2dissite 000-default.conf*

Next, let's test for configuration errors:

- *sudo apache2ctl configtest*

If you see a "*Syntax OK*" output, then it's properly configured.

Restart Apache to implement your changes:

- *sudo systemctl restart apache2*

Apache should now be serving your domain name. You can test this by navigating to http://example.com, where you should see something like this:

Success! The example.com virtual host is working!

Checkpoint – 2: Show this to your course teacher.

Now issue the following command:

- *sudo a2ensite example.com.conf*

Restart Apache to implement your changes:

- *sudo systemctl restart apache2*

Try navigating to http://pkilabserver.com, observe what happens. Think about what is happening. Try to navigate to http://127.0.0.1. What happened and why?

Checkpoint – 3: Show this to your course teacher and explain what is happening.

Now, disable the default configuration so that pkilabserver.com and example.com all point to the virtual server domains. Test both domains work by using your browser.

Other steps – Getting Familiar with Important Apache Files and Directories

Now that you know how to manage the Apache service itself, you should take a few minutes to familiarize yourself with a few important directories and files.

Content

- /var/www/html: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the /var/www/html directory. This can be changed by altering Apache configuration files.

Server Configuration

- /etc/apache2: The Apache configuration directory. All of the Apache configuration files reside here.
- /etc/apache2/apache2.conf: The main Apache configuration file. This can be modified to make changes to the Apache global configuration. This file is responsible for loading many of the other files in the configuration directory.
- /etc/apache2/ports.conf: This file specifies the ports that Apache will listen on. By default, Apache listens on port 80 and additionally listens on port 443 when a module providing SSL capabilities is enabled.
- /etc/apache2/sites-available/: The directory where per-site virtual hosts can be stored. Apache will not use the configuration files found in this directory unless they are linked to the sites-enabled directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory with the a2ensite command.
- /etc/apache2/sites-enabled/: The directory where enabled per-site virtual hosts are stored. Typically, these are created by linking to configuration files found in the sites-availabledirectory with the a2ensite. Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.

- /etc/apache2/conf-available/, /etc/apache2/conf-enabled/: These directories have the same relationship as the sites-available and sites-enabled directories, but are used to store configuration fragments that do not belong in a virtual host. Files in the conf-available directory can be enabled with the a2enconf command and disabled with the a2disconf command.
- /etc/apache2/mods-available/, /etc/apache2/mods-enabled/: These directories contain the available and enabled modules, respectively. Files in ending in .load contain fragments to load specific modules, while files ending in .conf contain the configuration for those modules. Modules can be enabled and disabled using the a2enmod and a2dismod command.

Server Logs

- /var/log/apache2/access.log: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- /var/log/apache2/error.log: By default, all errors are recorded in this file. The LogLeveldirective in the Apache configuration specifies how much detail the error logs will contain.

Task-2: Becoming a certificate authority

A Certificate Authority (CA) 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, we need to create digital certificates, but we are not going to pay any commercial CA. We will become a root CA ourselves, and then use this CA to issue certificate for others (e.g. servers). In this task, we will make ourselves 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 PKI. Root CA's certificates are unconditionally trusted.

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 we described before, we 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, we 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. We 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, we 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.

Using OpenSSL to demonstrate HTTPS

In this lab, we will explore how public-key certificates are used by web sites to secure web browsing. 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, we 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.

We 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 we will not pursue this direction.

Load ca.crt into Firefox: We 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, we 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 – 4: 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.

Task-3: Deploy HTTPS into Apache

Now, we 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>
```

Now, we 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 Apache.

- *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 – 5: Access the https://example.com in your browser and show it to your instructor.

Task-4: HTTP to HTTPS Redirect

Till now, you have created an HTTPS profile for your web server that you can access via *https://example.com*. However, the HTTP profile still remains there, meaning anyone can access you site using HTTP (*http://example.com*) as well. If you want to fully secure your website, you would like to ensure that nobody can access it via HTTP. In this task, you will ensure this. For this, we will use a specific module of Apache. Apache is quite modular in the sense it supports the development of additional module which can add extended functionalities. To complete our first task, we will use the ***mod_rewrite*** module of Apache .

The mod_rewrite (*https://httpd.apache.org/docs/current/mod/mod_rewrite.html*) module can be utilised to redirect a user from one URL to another URL or one port to another port. In this task, you will need to redirect the user from the default port (80) of HTTP to the default port of HTTPS which is 443. In this way, even when a user tries to access *http://example.com*, the user will be redirected to *https://example.com* by the Apache web server!

Step 1: You can use the *a2enmod* command to enable a module and the *a2dismod* command to disable a module. Enable the mod_rewrite module using a2enmod command.

- sudo a2enmod rewrite Adding Authentication to Apache:

Step 2: Look at the /etc/apache2/sites-enabled directory to find the configuration file for port 80 for example.com. If unsure, look at the lab 4 manual once again to find this. Add the following lines in the virtual host for the 80 port within your respective configuration file.

```
RewriteEngine On
RewriteCond %{HTTPS}  !=on
RewriteRule ^/?(.*)
https://%{SERVER_NAME}/$1 [R,L]
```

Step 3: Test your Apache configuration using the following command:

- *sudo apache2ctl configtest*

Step 4: Restart the Apache server.

- *sudo systemctl restart apache2*

Step 5: Test *example.com* on your browser. It should be redirected automatically to *https://example.com*.

Checkpoint – 6: Show it to your teacher that the automatic redirection works.

Task-5: Rudimentary Authentication

In this task, we will utilise a rudimentary authentication mechanism of Apache. The premise is that not all users can access your site. It can be accessed only by properly authenticated users. There are a couple of ways to do this. However, we will be using a method that uses an authentication file (called .htpasswd) containing the names and hashed passwords of allowed users.

Step 1: Add users to your Apache web server using the following command:

- sudo htpasswd -c /etc/apache2/.htpasswd username (change username with the username that you want for your first user). The -c option is used to create the first user. To add other users, you will need to skip that option.
- Add a second user to your Apache server using the htpasswd command.

Step 2: Use the following command to cat the contents of .htpasswd file:

- cat /etc/apache2/.htpasswd

You will something like the following, containing the usernames and their corresponding hashed passwords:

```
sammy:$apr1$lzxsIfXG$tmCvCfb49vpPFwKGVsuYz.
another_user:$apr1$p1E9MeAf$kiAhneUwr.MhAE2kKGYHK.
```

Step 3: Next, add the following into your https configuration file for example.com.

```
<Directory "/var/www/html">
        AuthType Basic
        AuthName "Restricted Content"
        AuthUserFile /etc/apache2/.htpasswd
        Require valid-user
</Directory>
```

Step 4: Restart the apache server.

Checkpoint – 7: Try to access https://example.com from your browser. When prompted for username and password, provide the ones that you created earlier. If you can access the site, show it to your teacher to tick off the final checkpoint.