Lab 6: Securing Apache Web Server - 2

Objectives:

- To secure a apache web server using different authentication mechanisms.

Submission:

- Three checkpoints.

Instruction:

In lab 5, your goal was to create a secure web server that supports HTTPS. However, there was no authentication mechanism installed and this meant anyone could access the website. In real life scenarios, you often need to add authentication mechanisms to facilitate the basic built-in access control mechanism of Apache. In this lab, you will extend lab 5 to enable this functionality. In real life scenarios, you will often use additional third-party plug-ins or write your own mechanism using any server-side language. However, Apache provides a simple basic authentication mechanism. You will explore two of them in this lab.

There will be three tasks with three checkpoints. Please follow the instructions, complete each tasks and show it to your teacher to tick-off the corresponding checkpoint.

Task-1 (5 Marks):

In lab 5 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. Try it to test it. If you want to fully secure your website, you must ensure that nobody can access it via HTTP. For this, we will use a specific module of Apache called ***mod_rewrite***.

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*. If this happens, show it to your teacher and tick off the first checkpoint.

Task-2 (7 Marks):

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.

Step 5: 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 second checkpoint.

Task-3 (8 Marks):

In this task, you will add a bit more advanced authentication mechanism. Even though you can use the basic authentication mechanism using the steps of Task-2, it is not very convenient. Modern web servers often rely on a database to authenticate a user. In this task, you will need an authentication mechanism that relies on MySQL database. For this at first, you will need to add a user with his/her corresponding password into the database and then configure the Apache server to initiate authentication using MySQL.

Step 1: Install MySQL server on your ubuntu using the following commands:

- *sudo apt-get update*
- *sudo apt-get install mysql-server.* You might be required to submit a password for root. Use *cse* for this.
- *sudo apt-get installlibaprutil1-dbd-mysql*

Step 2: Configuring MySQL using the following command:

- *mysql_secure_installation*

Select no for every option except the last when you are prompted to reload the privilege table. Remember, in production environment you most probably will need to choose more sensible options as per your organisation's policy. So don't choose all nos in a production environment.

Step 3: After this, check the status of the mysql service:

- *systemctl status mysql.service*

If you see the service is running, then proceed to the next step.

Step 4: Login to the MySQL using the following command:

- *mysql -u root -p*

Step 5: Create a database called apache using the following command in the MySQL console:

- *CREATE DATABASE apache;*

Step 6: Use the following command to use the apache database in the MySQL console:

- use apache;

Step 7: Create a table called **users** using the following command with the supplied attributes in the MySQL console:

- *CREATE TABLE users (username VARCHAR(30) PRIMARY KEY,password VARCHAR(512) NOT NULL);*

Step 8: Now, we will add users to our table. But before that we need to create a hashed password which will be stored in the database. Remember, **never store** plain password in your database. Use the following command to create hashed password for a user called sammy in a separate console:

- *htpasswd -bns sammy sammys_password* (change this to whatever you want)

You will see an output like the following in your console:

- sammy:{SHA}tk7HEH6Wo7SKT6+3FHCgiGnJ6dA=

Copy everything after ':' (that starts with {SHA}) and save it somewhere. In MySQL console, use the following command to add a user called Sammy:

- *INSERT INTO users VALUES ('sammy', '{SHA}+9C5w2dyQYmbrXe+Sdy7aUcafvU=');*

Change '*{SHA}+9C5w2dyQYmbrXe+Sdy7aUcafvU=*' with the content that you copied earlier.

Repeat the same steps to add another user called alice.

Step 9: Next, issue the following commands to enable the *mod_authn_dbd* module of Apache. This can be done by enabling the following modules:

- *sudo a2enmod dbd*
- *sudo a2enmod authn_dbd*
- *sudo a2enmod socache_shmcb*
- *sudo a2enmod authn_socache*

Step10: Get rid of what you added in the Step 3 of Task 2 and replace it with the following contents:

```
DBDriver mysql
DBDParams "dbname=apache user=root pass=cse"
DBDMin 4
DBDKeep 8
DBDMax 20
DBDExptime 300
<Directory "/var/www/html">
  AuthType Basic
  AuthName "My Server"
  AuthBasicProvider socache dbd
  AuthnCacheProvideFor dbd
  AuthnCacheContext my-server
  Require valid-user
  AuthDBDUserPWQuery "SELECT password FROM users WHERE
username = %s"
</Directory>
```

Step 11: Restart the Apache server.

Step 12: Try to access the example.com page and when prompted for username/password, provide the one that have in your MySQL database. If everything works fine, you will be able access the page. At this stage, show it to your teacher to tick-off the final checkpoint.