Lab 5: Securing Apache Web Server

Name: Amit Kumar Sharma
Registration: 2020831009
Platform: Ubuntu on WSL (Windows Subsystem for Linux)

Objectives

- Setup a secure web server using Apache and digital certificates.
- Understand HTTPS and TLS/SSL.
- Become a Certificate Authority (CA) and issue server certificates.
- Test the server using OpenSSL and curl.

Task 1: Becoming a Certificate Authority (CA)

1. **Create a working folder for Lab 5**:

```
mkdir Lab5_SSL
cd Lab5_SSL
```

2. **Copy and configure `openssl.cnf`**:

```
cp /usr/lib/ssl/openssl.cnf ./openssl.cnf
mkdir certs crl newcerts private
touch index.txt
echo 1000 > serial
echo 1000 > crlnumber
```

- `certs/`, `crl/`, `newcerts/`, `private/` are subdirectories required by OpenSSL.
- `index.txt` is the database file.

- `serial` stores the initial serial number for certificates.

Step 2: Create Root CA

Command:

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

- Entered a secure passphrase for CA’s private key.
- Provided distinguished name details (Country, State, Organization, Common Name).
- Output files:
  - `ca.key` → CA private key
  - `ca.crt` → CA public certificate (self-signed)

✅ **Checkpoint 1 Complete:** Root CA created.

Step 3: Create certificate for example.com

1. **Generate server key:**

```
openssl genrsa -des3 -out server.key 2048
```

- Protected the key with a passphrase.

2. **Create a Certificate Signing Request (CSR):**

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

- Common Name: `example.com`

- Entered challenge password and optional company name.

3. **Sign the CSR with CA to generate certificate:**

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

- The server certificate `server.crt` was generated.

Step 4: Test HTTPS using OpenSSL s_server

1. **Combine key and certificate into one PEM file:**

```
cp server.key server.pem
cat server.crt >> server.pem
```

2. **Launch OpenSSL server:**

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

- Entered passphrase for server.pem.
- Output showed: `ACCEPT` → server is running and listening on default port 4433.

Step 5: Testing with curl

- **Test using localhost**:

```
curl -k https://localhost:4433/
```

- **Issue on WSL:**
  Curl failed with `Couldn't connect to server`.

Reason:

- WSL has network limitations — it does not bind to Windows localhost the same way.
- `OpenSSL s_server` may run, but `curl` in WSL cannot connect to `localhost:4433` reliably.
- Workaround: Use Windows browser or PowerShell curl with WSL IP, or configure `/etc/hosts` with `127.0.0.1 example.com`.

✅ **Checkpoint 1 Observed:** HTTPS session attempted; certificate warning expected in browsers because CA is self-signed.

Step 6: Repeat for webserverlab.com (Checkpoint 2)

1. **Generate key and CSR for webserverlab.com**:

```
openssl genrsa -des3 -out webserver.key 2048
openssl req -new -key webserver.key -out webserver.csr -config
openssl.cnf
```

- Common Name: `webserverlab.com`
- Challenge password & optional company name entered.

2. **Sign the CSR using CA**:

```
openssl ca -in webserver.csr -out webserver.crt -cert ca.crt
-keyfile ca.key -config openssl.cnf -batch
```

3. **Combine key and certificate into PEM:**

```
cp webserver.key webserver.pem
cat webserver.crt >> webserver.pem
```

4. **Launch OpenSSL server on port 4433:**

```
openssl s_server -cert webserver.pem -accept 4433 -www
```

- **Issue:** Key too small error resolved by using 2048-bit RSA.
- Entered passphrase → `ACCEPT` displayed.

5. **Test with curl**:

```
curl -k https://webserverlab.com:4433/
```

- Failed to connect.