Public-Key Infrastructure (PKI) Lab

Jatin Kesnani
21K-3204 Student
FAST University

Abstract

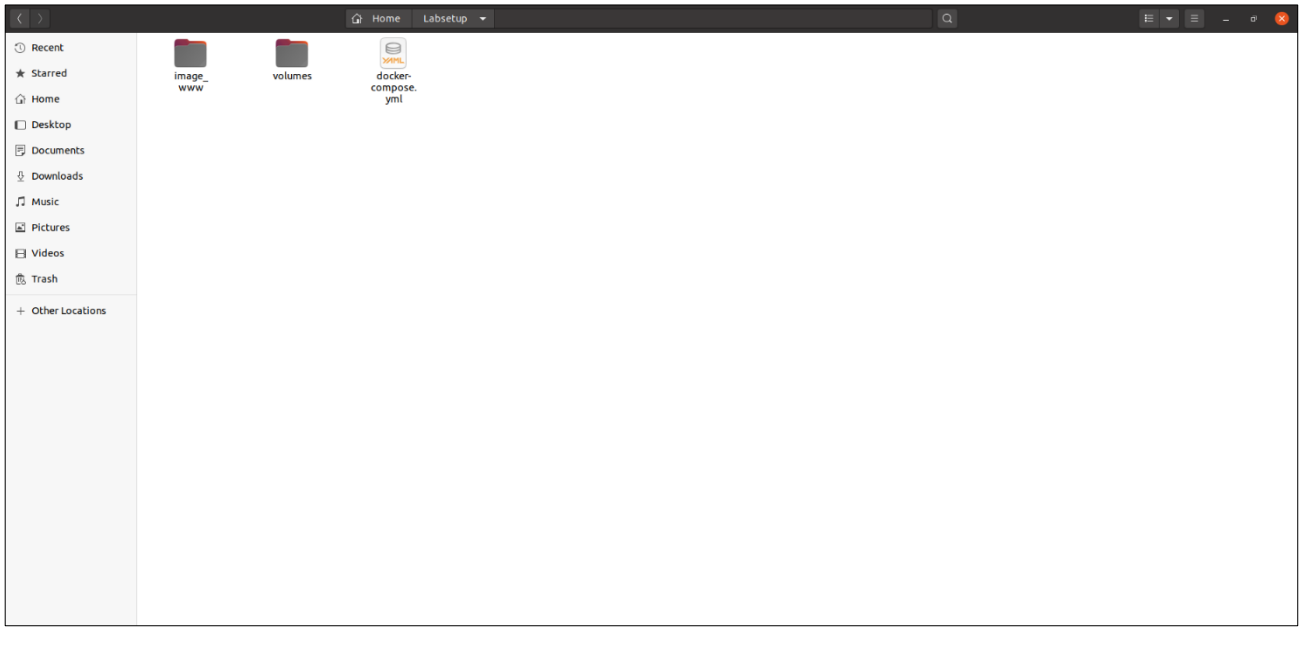
The Public Key Infrastructure (PKI) Lab aims to provide hands-on experience in understanding and implementing secure communications using public key cryptography. PKI serves as a critical solution for verifying the ownership of public keys and preventing man-in-the-middle attacks. This lab covers essential concepts like public-key encryption, X.509 certificates, and the role of Certificate Authorities (CAs) in establishing trust. By exploring these topics, students gain insights into the security mechanisms protecting web communications and the implications of a compromised root of trust.

1. Overview

The PKI Lab focuses on exploring the significance of public key cryptography as a foundation for secure communications. Public key cryptography, while secure, can still be vulnerable to man-in-the-middle attacks when verifying public key ownership. PKI addresses this issue by introducing a hierarchical structure involving CAs and X.509 certificates, providing a method for verifying the legitimacy of public keys. This lab explores these mechanisms in detail, including the use of root CAs, the role of Apache in configuring HTTPS, and the process of mitigating man-in-the-middle attacks. Through this lab, students develop an understanding of the importance of PKI and the risks associated with broken trust in this infrastructure.

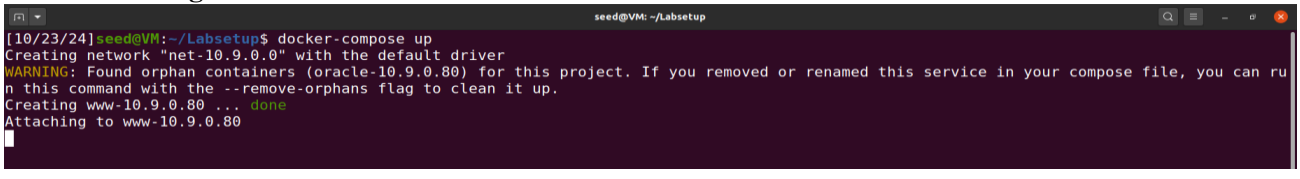
2. Lab Environment

2.1 Navigating to the Labsetup Directory

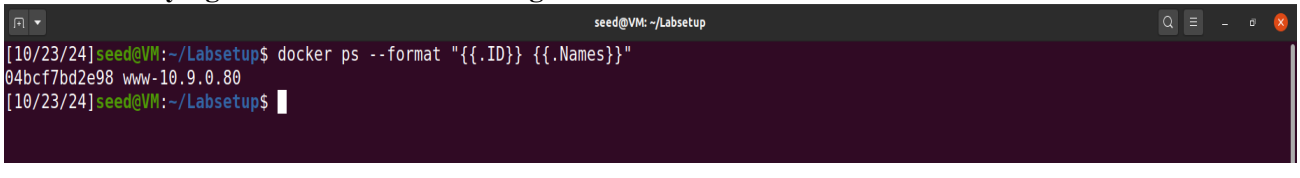


2.2 Building the Docker Container

2.3 Starting the Docker Container

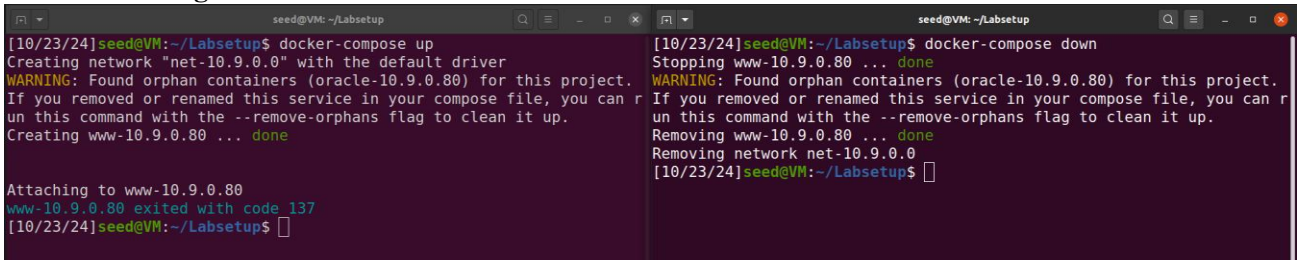


2.4 Verifying the Container is Running

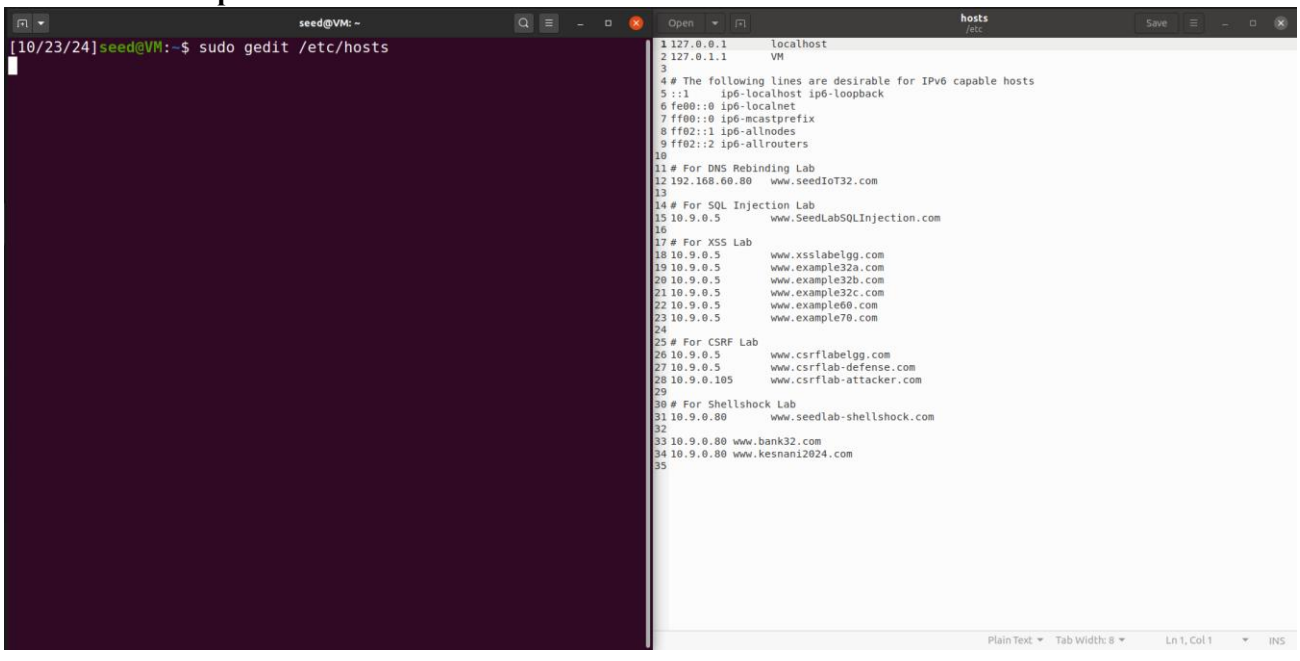


2.5 Accessing the Container Shell

2.6 Shutting Down the Container

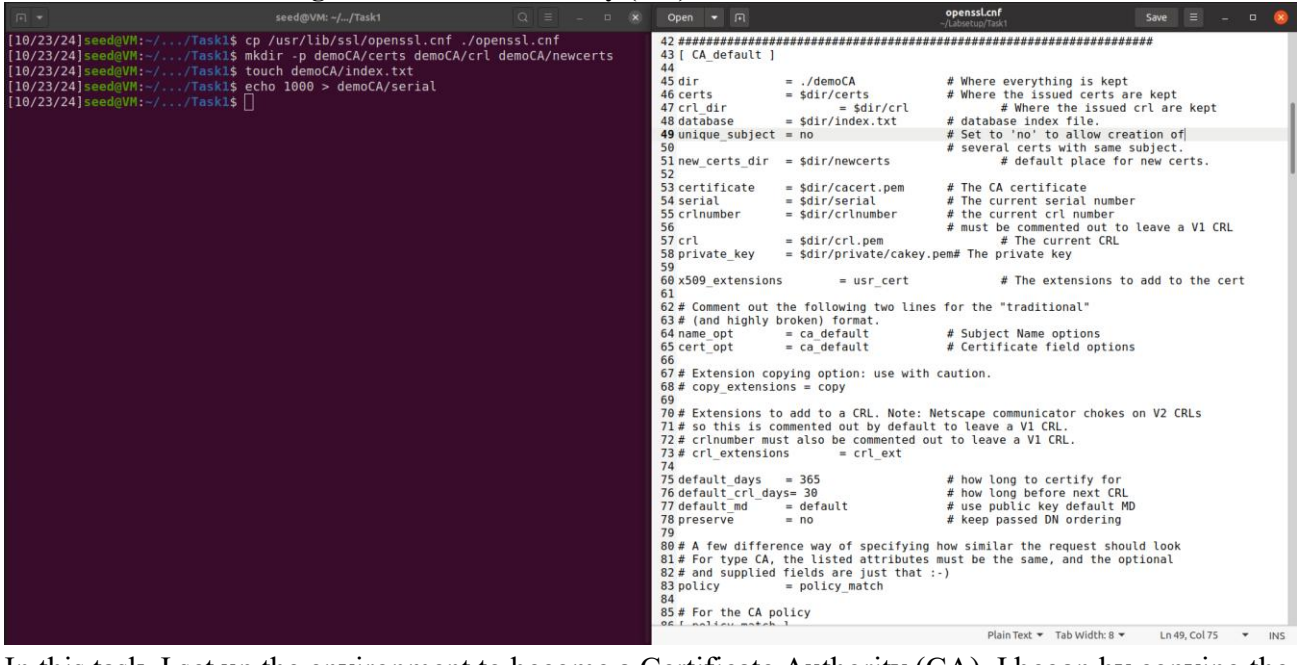


2.7 DNS setup

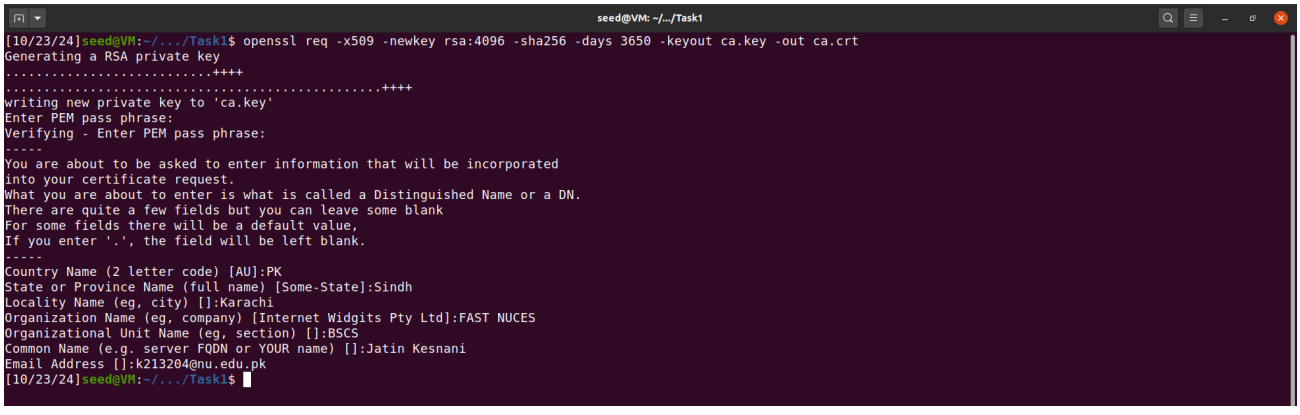


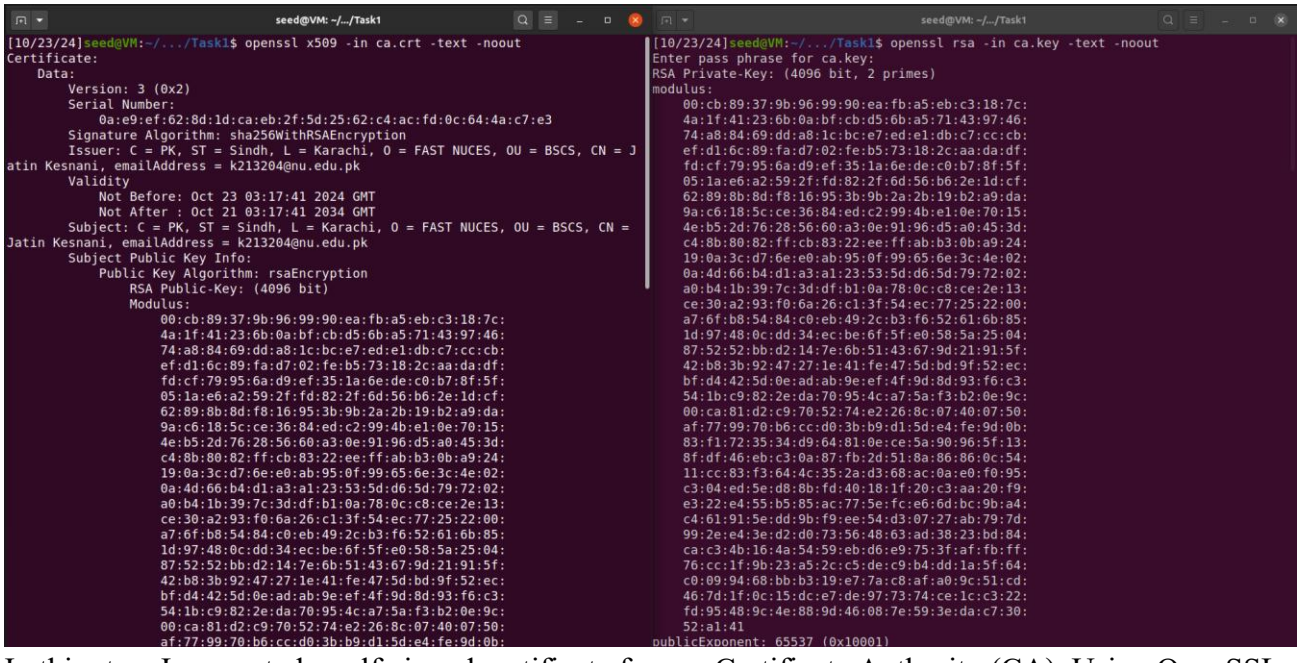
3. Lab Tasks

3.1 Task 1: Becoming a Certificate Authority (CA)



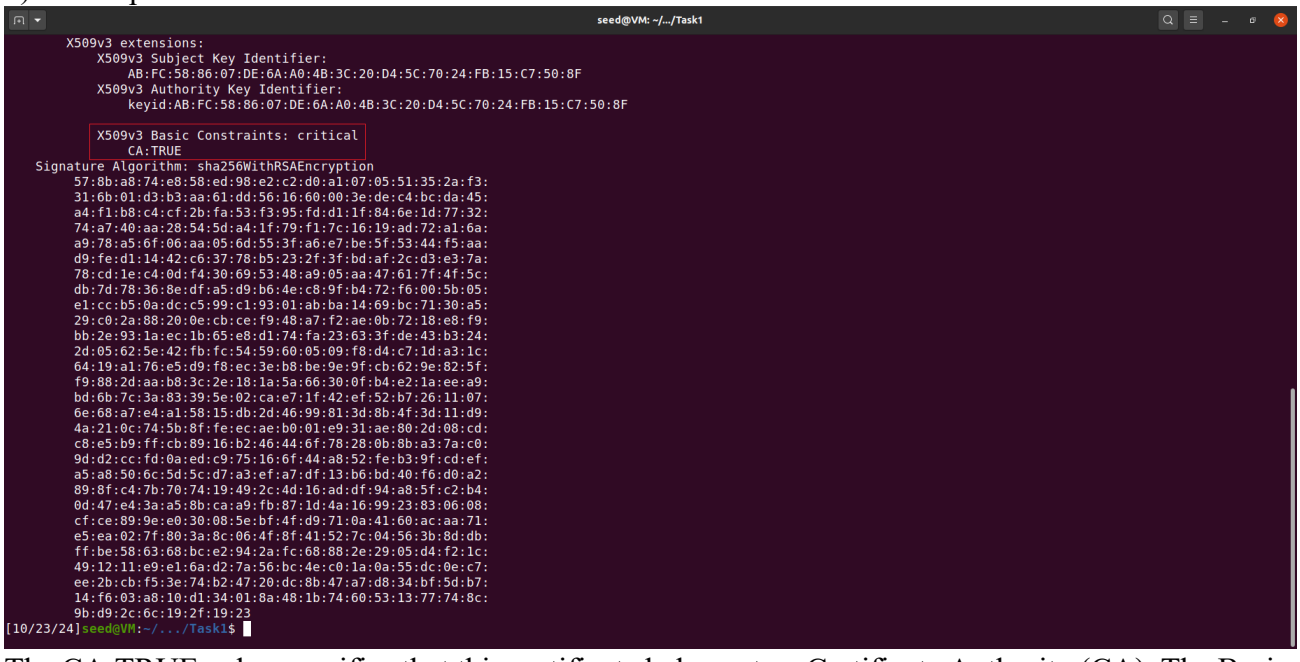
In this task, I set up the environment to become a Certificate Authority (CA). I began by copying the OpenSSL configuration file (openssl.cnf) to my working directory and created the necessary directories for managing certificates and CRLs. Additionally, I prepared the index and serial files required by OpenSSL to track issued certificates, setting the initial serial number to 1000. This setup enables me to issue self-signed root CA certificates for subsequent tasks.





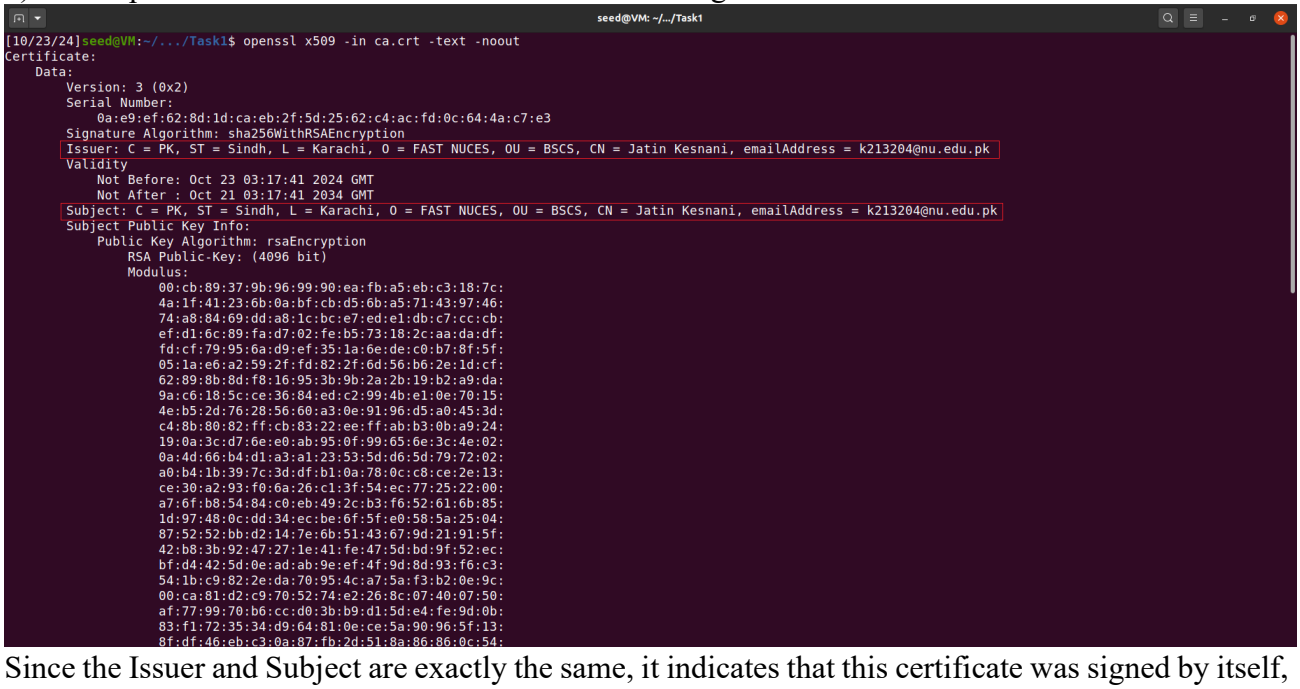
In this step, I generated a self-signed certificate for our Certificate Authority (CA). Using OpenSSL, I created a 4096-bit RSA key pair and a certificate valid for 10 years. The private key was stored in ca.key, and the public certificate was saved in ca.crt. To verify the content, I inspected the details of the CA certificate and private key using the openssl x509 and openssl rsa commands respectively, ensuring successful generation of the CA's credentials.

1) What part of the certificate indicates this is a CA's certificate?



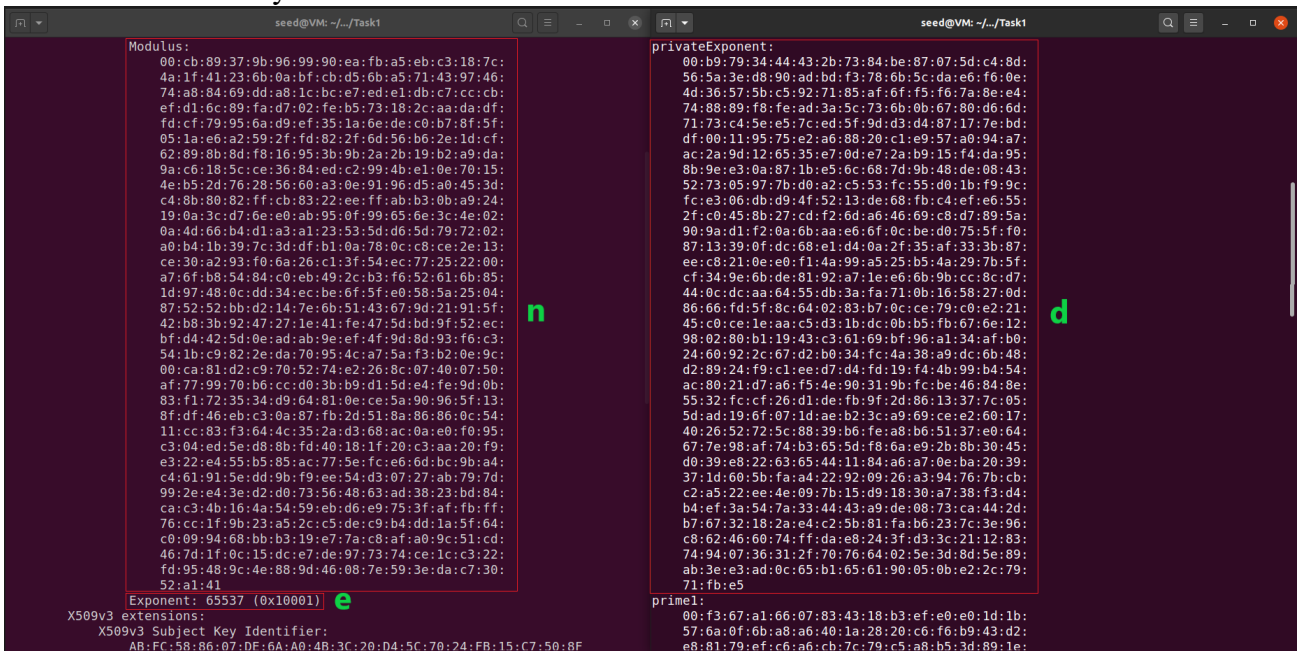
The CA:TRUE value specifies that this certificate belongs to a Certificate Authority (CA). The Basic Constraints extension defines whether a certificate can be used as a CA certificate or not. If this extension has CA:TRUE, the certificate can issue other certificates.

2) What part of the certificate indicates this is a self-signed certificate?



Since the Issuer and Subject are exactly the same, it indicates that this certificate was signed by itself, i.e., it is self-signed.

3) In the RSA algorithm, we have a public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq. Please identify the values for these elements in your certificate and key files.



3.2 Task 2: Generating a Certificate Request for Your Web Server

```
seed@VM: ~/.../Task2
[10/24/24]seed@VM:~/.../Task2$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=www.kesnani2024.com/O=Kesna
ni2024 Inc./C=PK" -passout pass:dees
Generating a RSA private key
.....................................................................+++++
.+++++
writing new private key to 'server.key'
-----
[10/24/24]seed@VM:~/.../Task2$
```

```
seed@VM: ~/.../Task2
[10/24/24]seed@VM:~/.../Task2$ openssl req -in server.csr -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: CN = www.kesnani2024.com, O = Kesnani2024 Inc., C = PK
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d3:e4:31:e2:29:ce:b0:3d:3c:32:df:c4:7c:e3:
                    66:15:e7:bd:44:77:96:cb:1d:a4:d9:44:0e:df:81:
                    ce:d7:b4:6b:8f:55:64:45:a1:8f:f0:7e:dd:6f:cc:
                    fa:b8:aa:3b:da:43:1e:05:6c:00:54:28:a6:12:d2:
                    cc:6e:cb:fc:46:07:ee:87:48:95:48:48:07:14:46:
                    19:8c:64:32:78:37:6e:d3:69:c2:0d:fa:8b:25:d6:
                    7a:06:09:28:7d:0f:1b:14:8d:d9:2e:a0:61:5b:30:
                    7e:c3:a4:1c:f4:3a:e9:fd:02:6c:84:60:09:74:78:
                    f7:be:a2:1b:a0:db:8a:62:f8:47:b4:71:79:46:6d:
                    16:0f:25:b5:c9:28:f1:0e:45:79:f4:29:19:e8:aa:
                    73:3d:28:ed:45:42:34:40:46:c8:cd:e7:44:6b:99:
                    41:4b:4a:f0:7f:fe:2d:e3:32:c1:9e:e6:47:ac:f8:
                    cd:fd:48:24:75:15:37:cd:b3:47:21:fc:21:5a:ff:
                    88:f9:32:a3:94:44:7e:ea:f3:f3:be:ce:8e:a3:e2:
                    5c:a2:2b:56:bc:be:75:ce:df:47:b3:67:eb:a1:e0:
                    20:30:cd:90:51:bb:f9:4d:22:e7:32:8d:39:0b:e5:
                    1b:33:24:b3:25:21:c7:c6:6c:cf:9f:89:58:3d:d6:
                    bd:e1
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         ba:99:2c:34:50:e1:c3:4c:f2:14:a8:f2:f1:2c:32:ac:5e:0b:
         3c:2d:02:9c:4e:9b:55:0a:4f:53:04:69:9c:91:cc:d6:6f:58:
         7f:c0:8b:1c:50:4a:d7:33:d4:bb:ff:64:93:c6:fa:34:66:51:
         7d:b5:fa:c1:75:ba:57:99:9b:31:5f:e7:d8:71:62:90:0b:2a:
         dc:01:2f:e3:78:de:44:8f:d9:11:38:29:cd:ce:93:9d:07:15:
         68:bd:4c:86:bc:b8:67:fd:e5:d8:83:8c:19:4c:af:1d:45:8a:
         58:a5:4a:6a:d8:18:81:ce:c3:82:2c:0c:8f:64:ba:48:b3:2e:
         6d:c9:3e:21:78:48:6a:c0:8c:e3:74:dc:9d:6d:e9:08:bd:8d:
         1e:bc:0f:32:82:3e:b3:ca:c5:1d:22:22:7e:5d:25:48:94:b3:
```

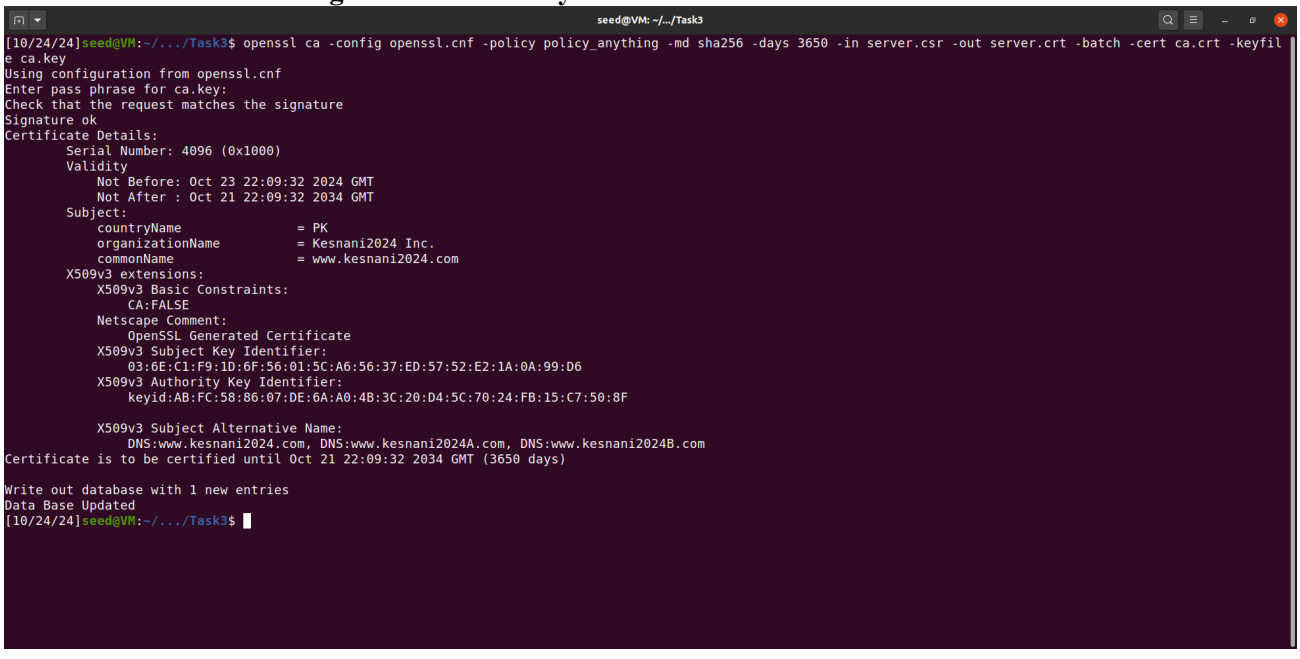
```
seed@VM: ~/.../Task2
[10/24/24]seed@VM:~/.../Task2$ openssl rsa -in server.key -text -noout
Enter pass phrase for server.key:
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:d3:e4:31:e2:29:ce:b0:3d:3c:32:df:c4:7c:e3:
    66:15:e7:bd:44:77:96:cb:1d:a4:d9:44:0e:df:81:
    ce:d7:b4:6b:8f:55:64:45:a1:8f:f0:7e:dd:6f:cc:
    fa:b8:aa:3b:da:43:1e:05:6c:00:54:28:a6:12:d2:
    cc:6e:cb:fc:46:07:ee:87:48:95:48:48:07:14:46:
    19:8c:64:32:78:37:6e:d3:69:c2:0d:fa:8b:25:d6:
    7a:06:09:28:7d:0f:1b:14:8d:d9:2e:a0:61:5b:30:
    7e:c3:a4:1c:f4:3a:e9:fd:02:6c:84:60:09:74:78:
    f7:be:a2:1b:a0:db:8a:62:f8:47:b4:71:79:46:6d:
    16:0f:25:b5:c9:28:f1:0e:45:79:f4:29:19:e8:aa:
    73:3d:28:ed:45:42:34:40:46:c8:cd:e7:44:6b:99:
    41:4b:4a:f0:7f:fe:2d:e3:32:c1:9e:e6:47:ac:f8:
    cd:fd:48:24:75:15:37:cd:b3:47:21:fc:21:5a:ff:
    88:f9:32:a3:94:44:7e:ea:f3:f3:be:ce:8e:a3:e2:
    5c:a2:2b:56:bc:be:75:ce:df:47:b3:67:eb:a1:e0:
    20:30:cd:90:51:bb:f9:4d:22:e7:32:8d:39:0b:e5:
    1b:33:24:b3:25:21:c7:c6:6c:cf:9f:89:58:3d:d6:
    bd:e1
publicExponent: 65537 (0x10001)
privateExponent:
    34:34:ae:01:ae:11:49:a4:dc:b7:20:20:d6:30:64:
    cf:92:d8:34:0d:4e:ee:de:6f:e6:43:f9:72:22:16:
    67:59:01:83:40:23:c3:70:29:f9:b4:18:34:da:89:
    0f:84:89:d8:9d:de:c0:b1:3a:67:81:d8:61:1c:e4:
    d7:8b:94:ad:60:78:e9:85:fd:99:e0:7d:36:06:8b:
    76:e6:9c:f4:b7:9b:ab:a5:0b:f5:cb:bc:0a:e7:8c:
    0a:fe:a8:a6:1f:59:a9:b7:c5:41:d2:ac:09:fe:a9:
    cc:e4:1b:6a:25:35:cf:6b:90:77:10:fe:63:5f:06:
    93:46:77:8e:2b:69:2e:36:d1:14:c1:34:0d:2a:45:
    82:10:9a:2a:bf:70:46:7e:c5:4a:6b:be:a3:7e:a2:
    5c:c0:e7:46:8d:e2:37:76:35:d3:d9:90:36:84:82:
    68:aa:74:90:51:c5:b6:8c:f7:ee:d8:c8:af:74:84:
    e1:22:e0:52:9f:95:90:4a:14:a1:63:27:b4:02:d1:
    7d:7b:da:c4:4d:e6:c2:b4:3e:dd:15:2f:82:be:3e:
    c4:81:b9:00:cf:1a:77:cf:59:0d:c1:21:e1:f8:03:
    f5:0b:81:5d:c9:12:cd:b5:bf:69:6d:f6:f2:a1:57:
```

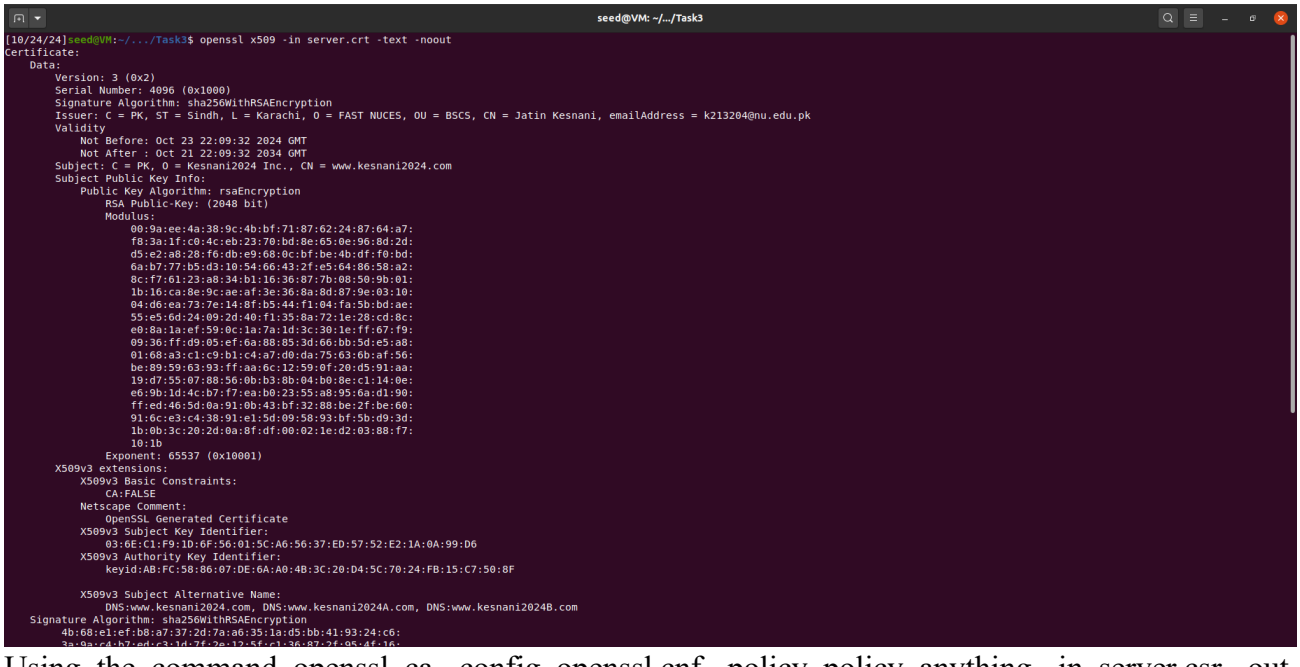
To generate a Certificate Signing Request (CSR) for my server, I used the openssl req command to create a private key and a CSR file. The command included subject information with CN=www.kesnani2024.com, O=Kesnani2024 Inc., and C=PK. After generating the CSR, I examined its content using openssl req -text, and I also inspected the private key using openssl rsa -text.

```
seed@VM: ~/.../Task2
[10/24/24]seed@VM:~/.../Task2$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=www.kesnani2024.com/O=Kesnani2024 Inc./C=PK" -passou
t pass:dees -addext "subjectAltName = DNS:www.kesnani2024.com, DNS:www.kesnani2024A.com, DNS:www.kesnani2024B.com"
Generating a RSA private key
.....................................................+++++
.................+++++
writing new private key to 'server.key'
-----
[10/24/24]seed@VM:~/.../Task2$
```

To generate a Certificate Signing Request (CSR) with alternative names, I used the openssl req command. The CSR included subject information like CN=www.kesnani2024.com, O=Kesnani2024 Inc., and C=PK, and I specified additional domain names using the -addext option for the Subject Alternative Name (SAN) field. This allowed the certificate to include DNS:www.kesnani2024.com, DNS:www.kesnani2024A.com, and DNS:www.kesnani2024B.com. These alternative names are essential for matching various hostnames with the server certificate.

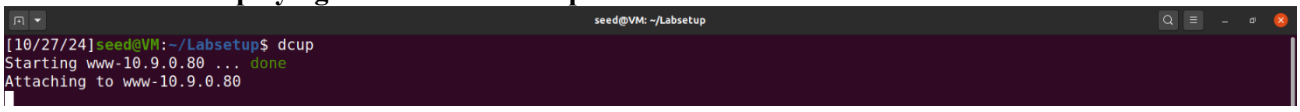
3.3 Task 3: Generating a Certificate for your server

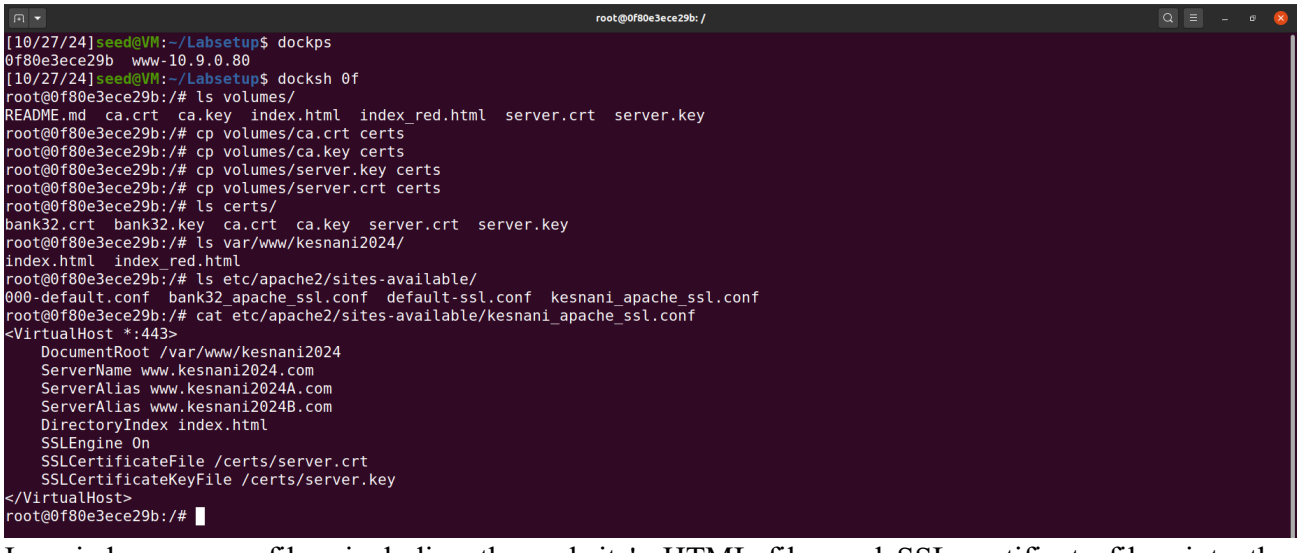




Using the command openssl ca -config openssl.cnf -policy policy_anything -in server.csr -out server.crt -batch -cert ca.crt -keyfile ca.key, we utilized our self-created Certificate Authority (CA) to issue the certificate. This command applied the policy_anything to allow flexible subject information and included our CA's credentials from ca.crt and ca.key. After generating the certificate, we confirmed its details, including the Subject Alternative Names, with the command openssl x509 -in server.crt -text -noout.

3.4 Task 4: Deploying Certificate in an Apache-Based HTTPS Website





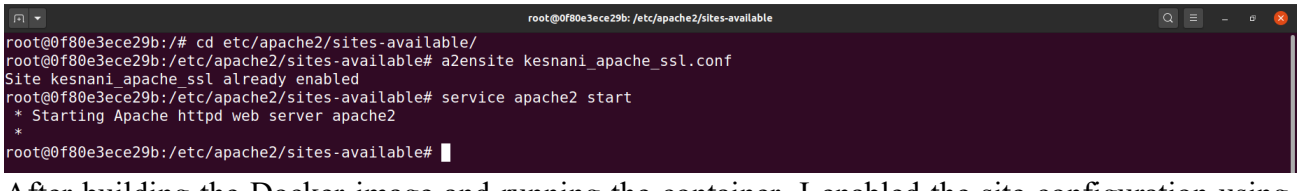
I copied necessary files, including the website's HTML files and SSL certificate files, into the **/volumes** directory within the Apache container. Specifically, I ensured that the SSL certificate and key were located in the **/certs** directory, while the website files were placed in **/var/www/kesnani2024/**.

I configured the Apache VirtualHost settings in the **kesnani_apache_ssl.conf** file to define the document root and SSL configurations. The configuration included specifying the server name, server aliases, enabling the SSL engine, and pointing to the SSL certificate and key files.

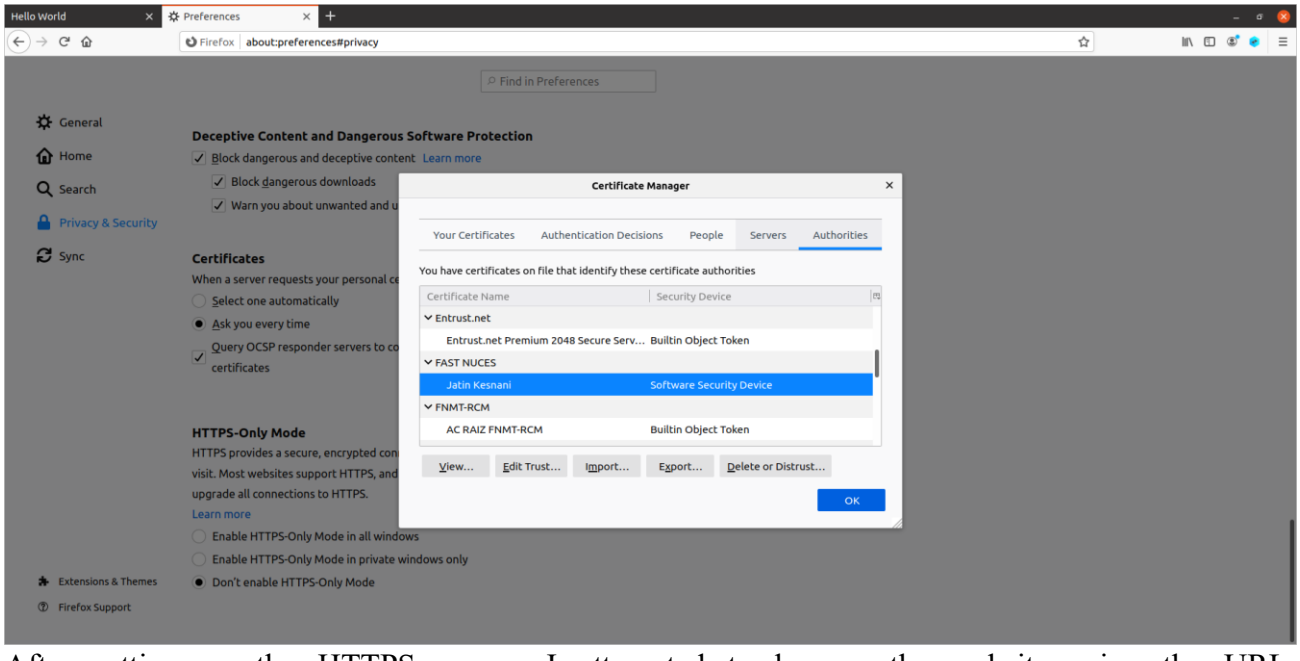
<VirtualHost *:443>

DocumentRoot /var/www/kesnani2024
ServerName www.kesnani2024.com
ServerAlias www.kesnani2024A.com
ServerAlias www.kesnani2024B.com
DirectoryIndex index.html
SSLEngine On
SSLCertificateFile /certs/server.crt
SSLCertificateKeyFile /certs/server.key

</VirtualHost>

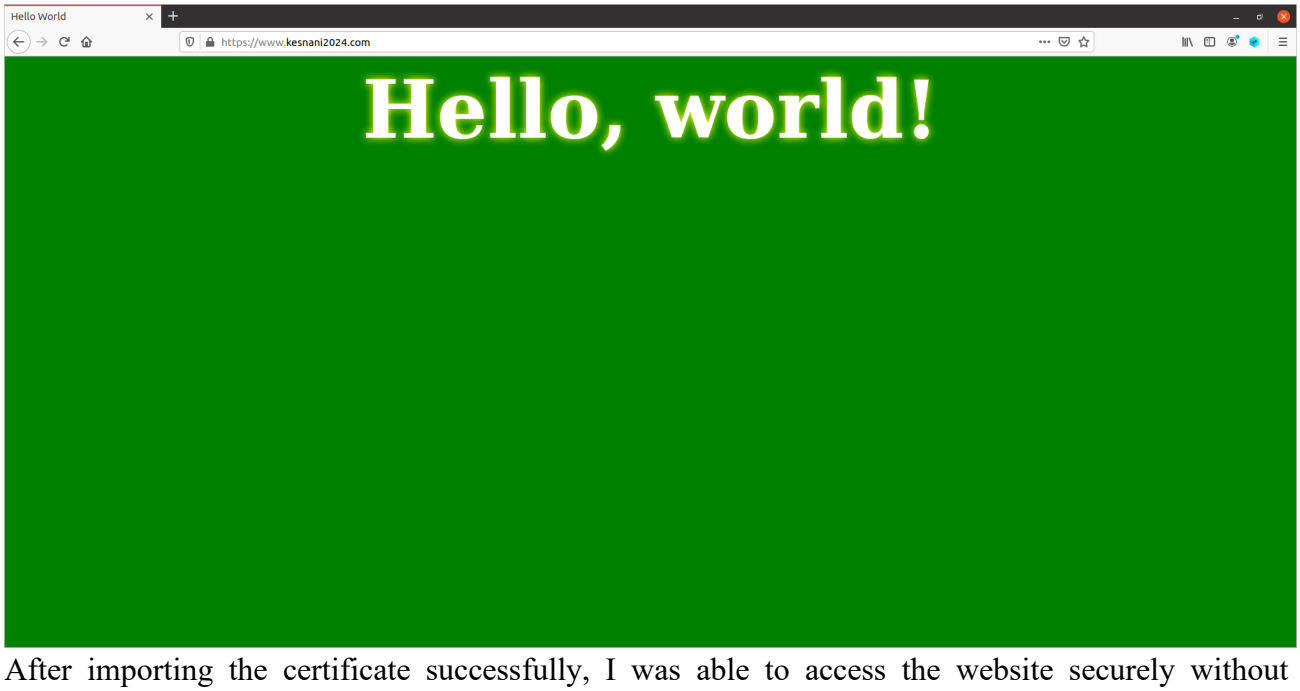


After building the Docker image and running the container, I enabled the site configuration using **a2ensite** and started the Apache server. This allowed Apache to serve the HTTPS site on port 443.



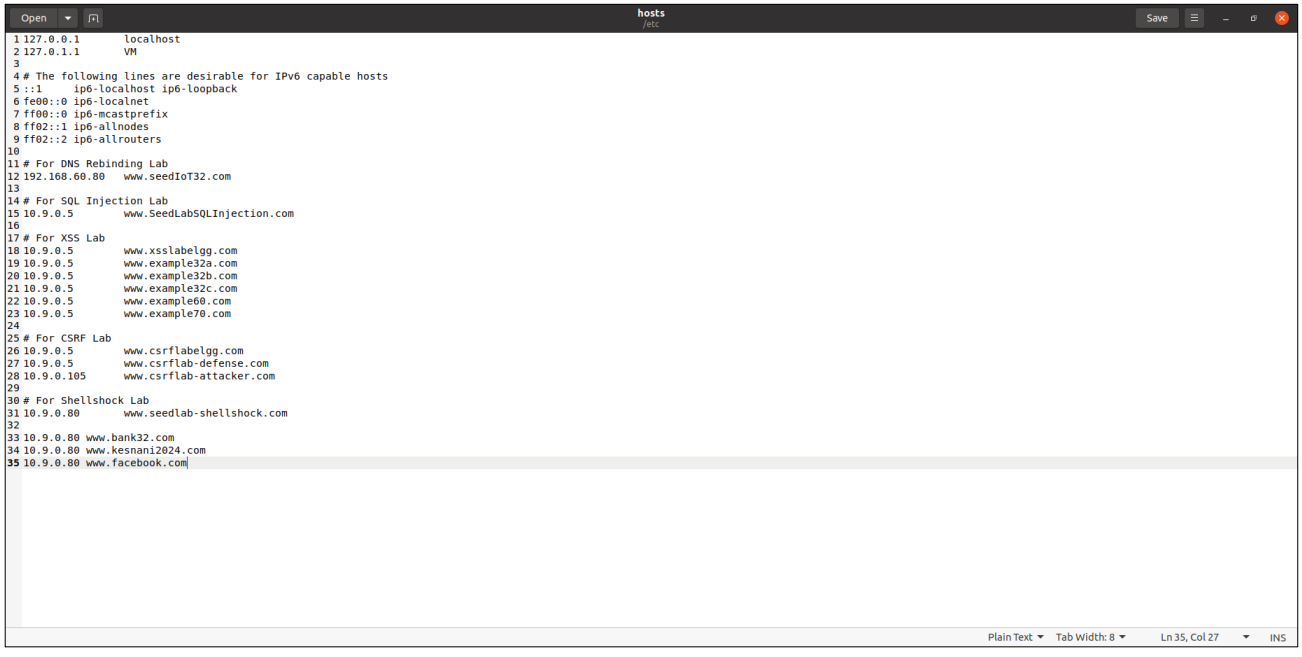
After setting up the HTTPS server, I attempted to browse the website using the URL **"https://www.kesnani2024.com"**. Initially, the browser displayed a security warning and prevented access due to the server's certificate not being recognized by the browser. This occurred because the certificate was not issued by a trusted certificate authority (CA).

To fix this, I imported the **"ca.crt"** certificate into Firefox. I navigated to **"about:preferences#privacy"** in Firefox's address bar and clicked the **"View Certificates"** button. Under the **"Authorities"** tab, I chose to import the **"ca.crt"** certificate and selected the option to **"Trust this CA to identify websites"**.

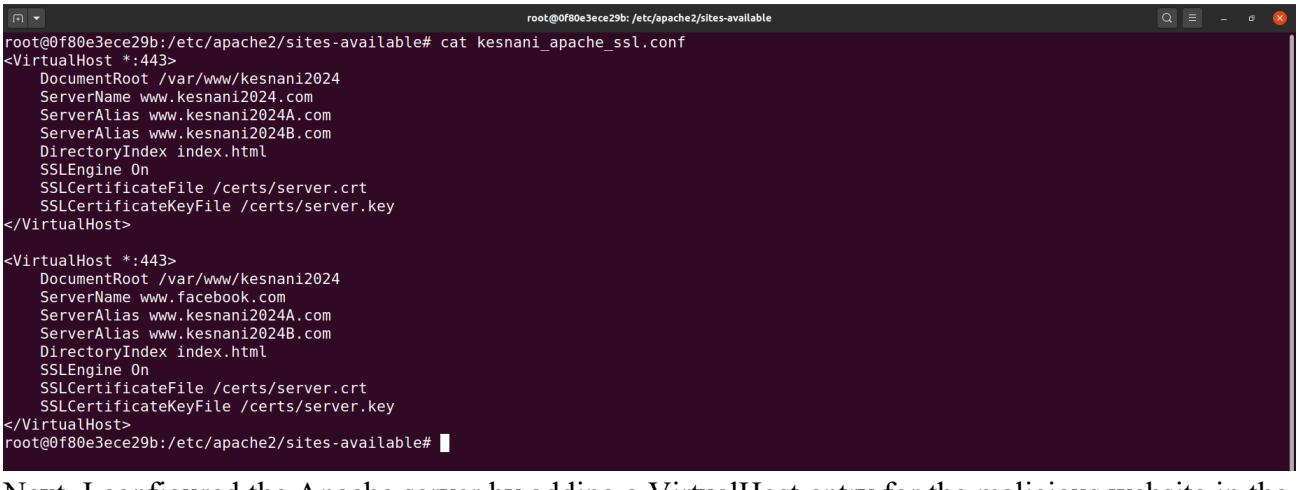


After importing the certificate successfully, I was able to access the website securely without receiving any warnings. This demonstrated the successful configuration of my HTTPS server using the self-signed certificate.

3.5 Task 5: Launching a Man-In-The-Middle Attack

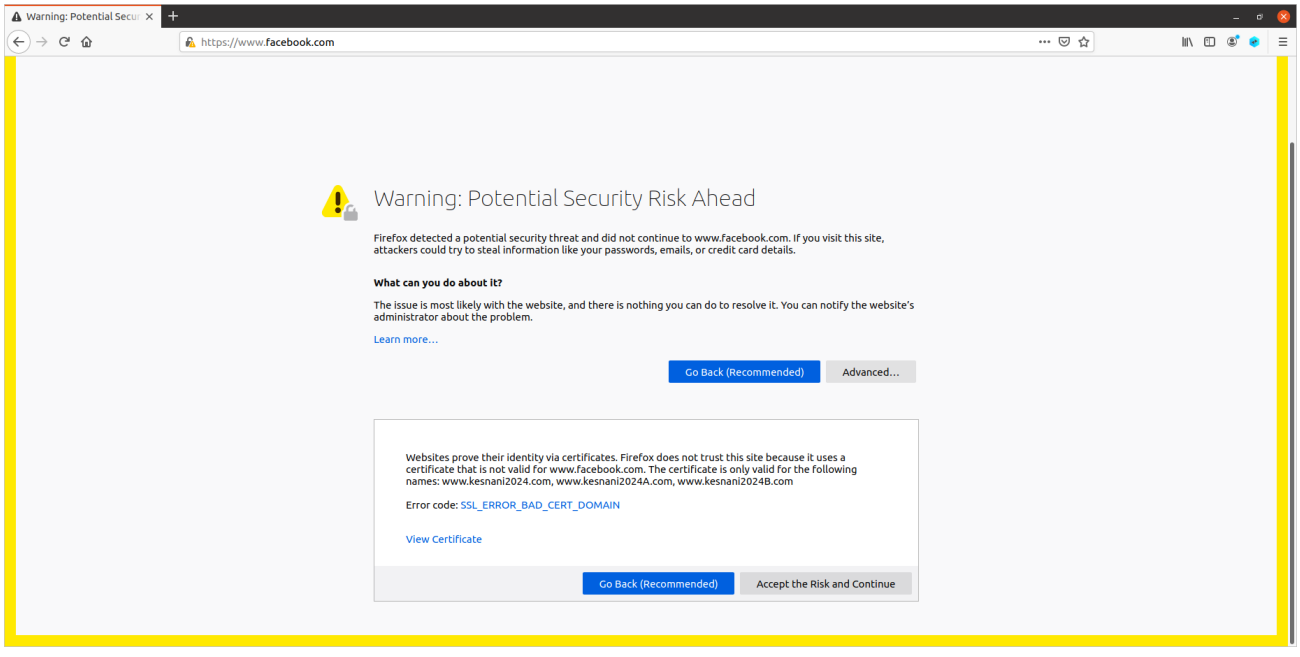


I have targeted the website **www.facebook.com.** I added an entry for the target website in the **/etc/hosts** file. This entry redirected the domain name to the attacker's server.



Next, I configured the Apache server by adding a VirtualHost entry for the malicious website in the **/etc/apache2/sites-available/kesnani_apache_ssl.conf** file. This entry allowed Apache to serve a website for the spoofed domain. Since I can't obtain a legitimate certificate for the target site, I will use the same certificate previously set up for my malicious server.

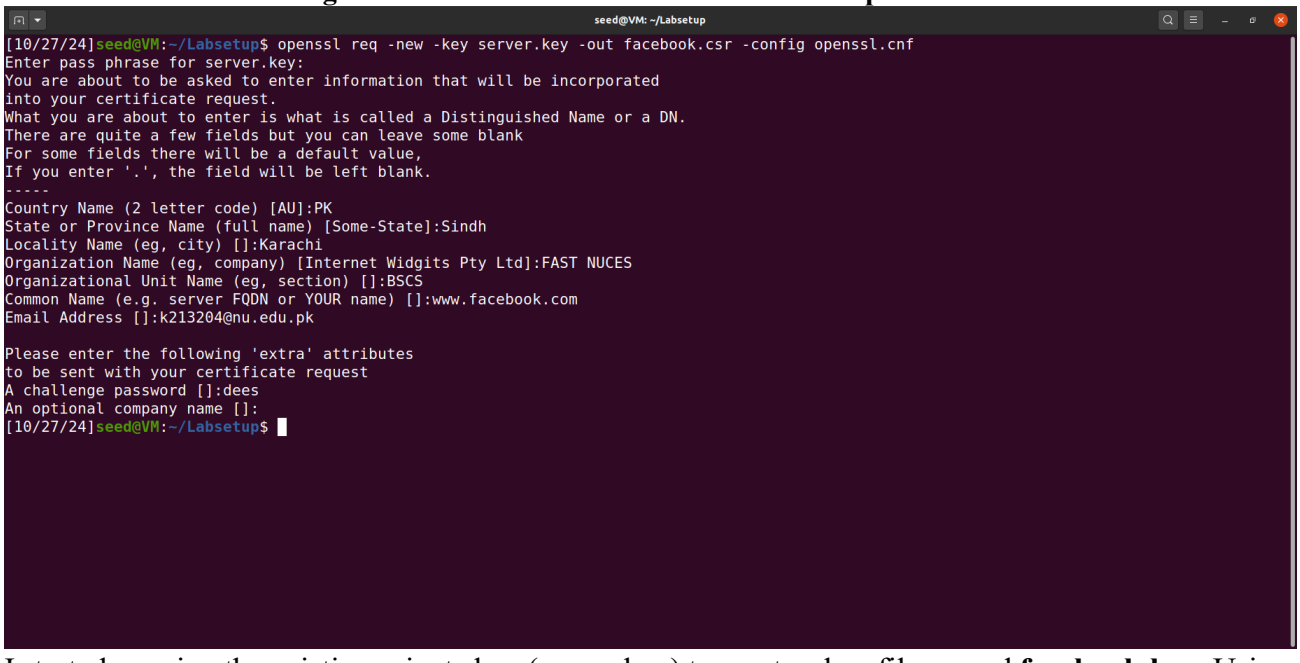
I restarted the Apache server with **service apache2 restart** so as to commit the changes made.



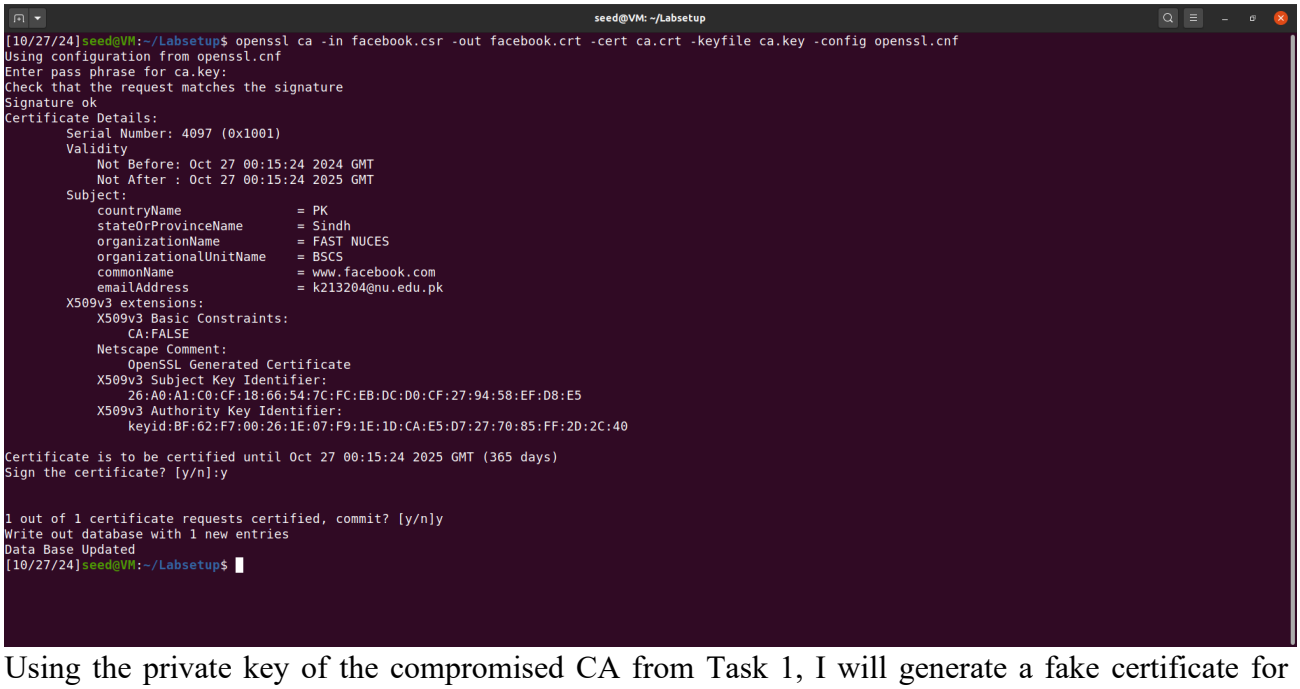
When I accessed the website **www.facebook.com**, the request will be directed to my malicious server. The browser received a **"Warning: Potential Security Risk Ahead"** error, the error is caused by an invalid security certificate being used, with the specific error code of **SSL_ERROR_BAD_CERT_DOMAIN**. It indicates that the certificate presented by my server (**www.kesnani2024.com**) does not match the expected certificate of the legitimate website (**www.facebook.com**). This alert is due to the browser verifying the certificate's authenticity, which is a core security feature of PKI.

Observations: When accessing the target website, I observed a "connection not secure" error in the browser. This happened because the browser detected a mismatch between the presented certificate and the expected certificate of **www.facebook.com**. In this emulation, I used a certificate for the site **www.kesnani2024.com** to impersonate **www.facebook.com**. This is why PKI is crucial—it validates the authenticity of certificates and prevents attackers from successfully impersonating legitimate servers.

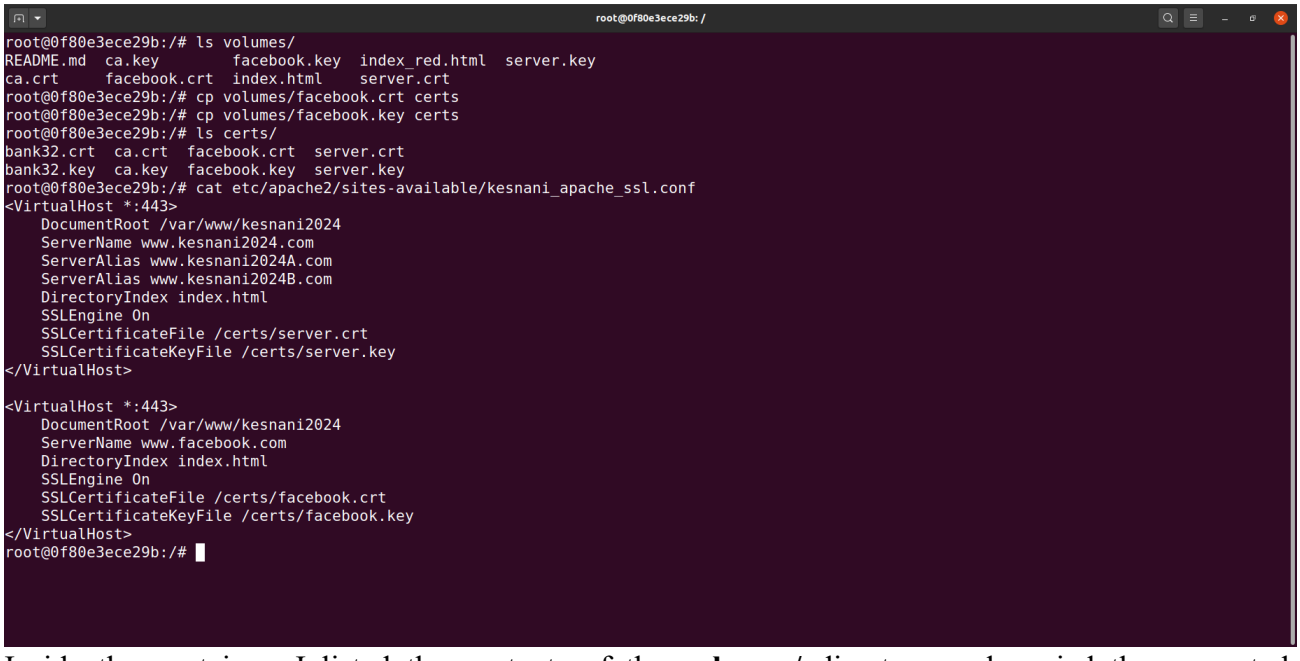
3.6 Task 6: Launching a Man-In-The-Middle Attack with a Compromised CA



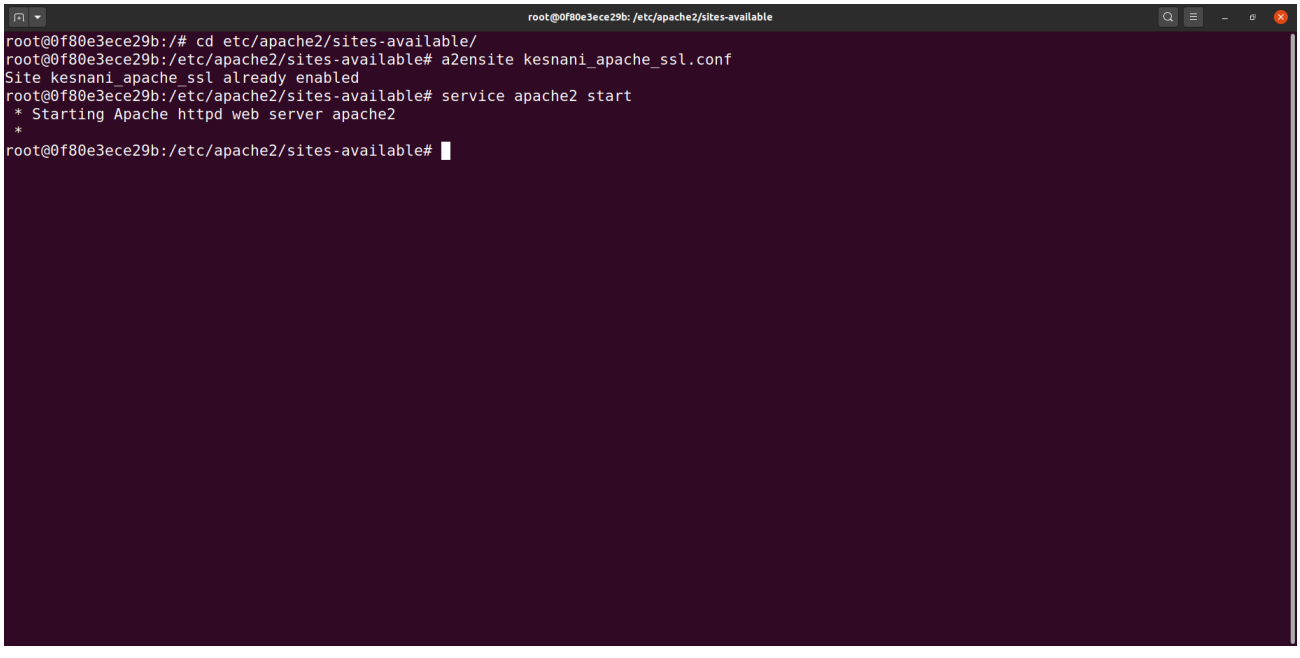
I started copying the existing private key (server.key) to create a key file named **facebook.key**. Using this private key, I generated a Certificate Signing Request (**facebook.csr**)



Using the private key of the compromised CA from Task 1, I will generate a fake certificate for **www.facebook.com**. Since the attacker now has access to the CA's private key, the attacker can issue a certificate that appears to be signed by a trusted CA.



Inside the container, I listed the contents of the **volumes/** directory and copied the generated **facebook.key** and **facebook.crt** files to the **certs/** directory. I also configured my malicious Apache server to use this fake certificate. Next, I manually added the compromised CA certificate (ca.crt) to the browser's trusted certificates.



With the setup complete, I attempt to access **www.facebook.com** in the browser. Since the fake certificate is signed by the compromised CA, and the browser has been configured to trust this CA, the browser will not show any security warnings. The victim is unaware that they are on a fake website.

Observations: When visiting www.facebook.com, I observed that the browser did not raise any warnings. The HTTPS connection appeared secure because the certificate was issued by a CA that the browser trusted. This demonstrates the danger of a compromised CA, as it enables an attacker to impersonate any website and perform a successful MITM attack without alerting the victim.