Applied Information Security

Assignment-3

Submitted by:

Muhammad Osama Khalid

20I-1955 (MS-CNS)

Section: 1

Table of Contents

List of Figures

Part-1 Lab Setup

In this part, I am going to set up the Attack Lab in which I will perform the SQL injection and Cross-Site Scripting Attacks in the next parts.

1. First, I download the Seed Virtual Machine from the SEED Lab Website and then open the virtual box and click on the add button. Then I add details of my VM and click Next. After that, I set the memory size (RAM) to 2 GB and click next. After that from the options, I select “Use an existing virtual hard disk file” and go to the path where I download the Seed VM and select that. After that, I click create to create my VM.

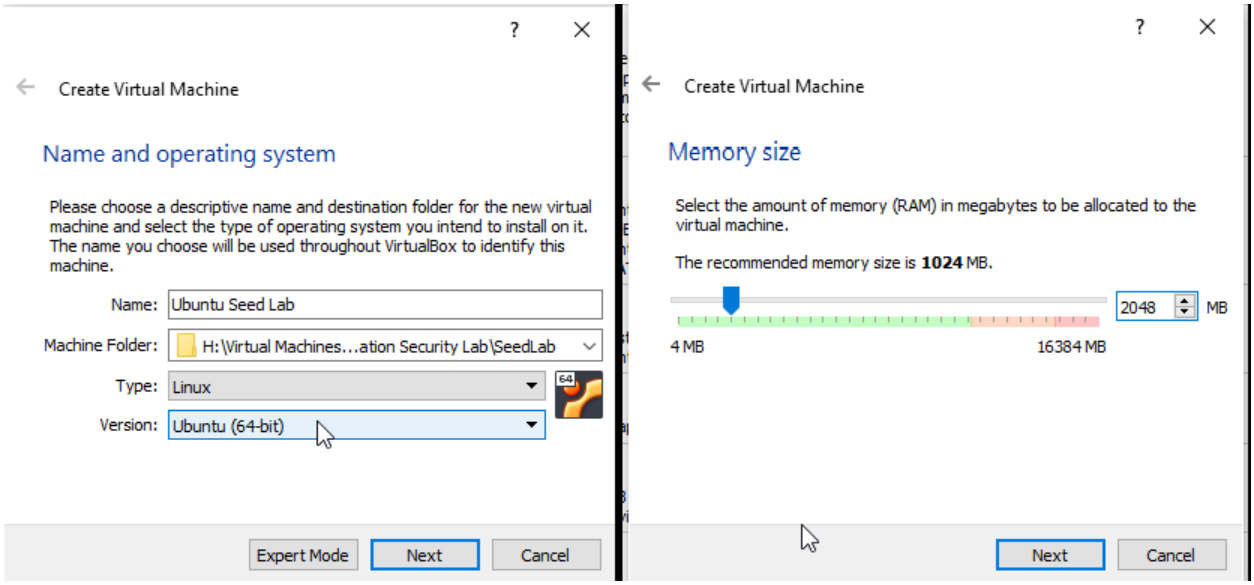

Figure 1: Creating the Virtual Machine

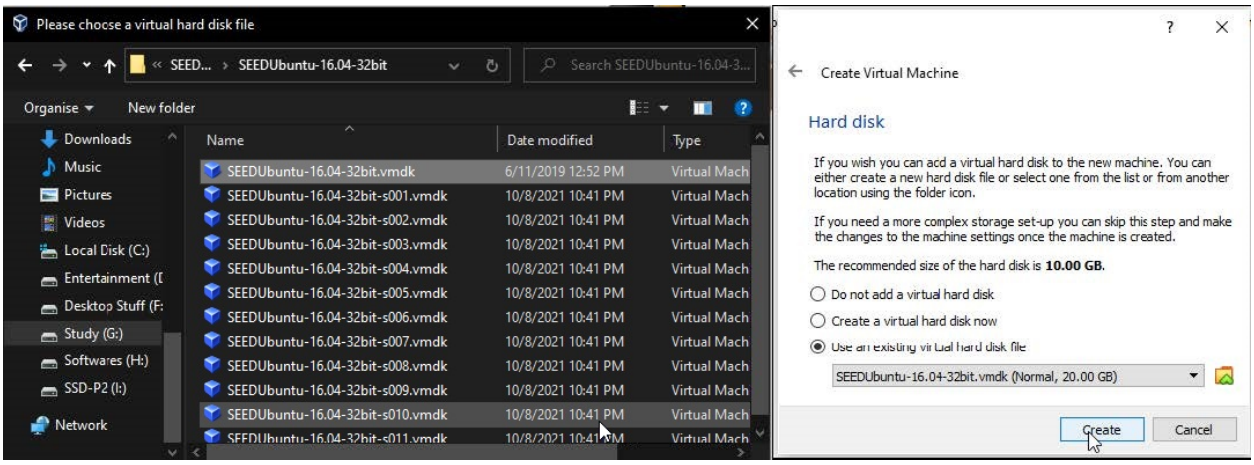

Figure 2: Creating the Virtual Machine

2. After that go to the virtual box setting and select Network from the tab on the left panel. Then I click on the “+” button to create a new NAT Networks adapter. After that, I open the virtual machine setting, select Network from the tab on the left panel, and staying on Adapter 1 and under enable network adapter I click on the “Attah to” drop-down menu and select NAT Network adapter that I just created and click ok.

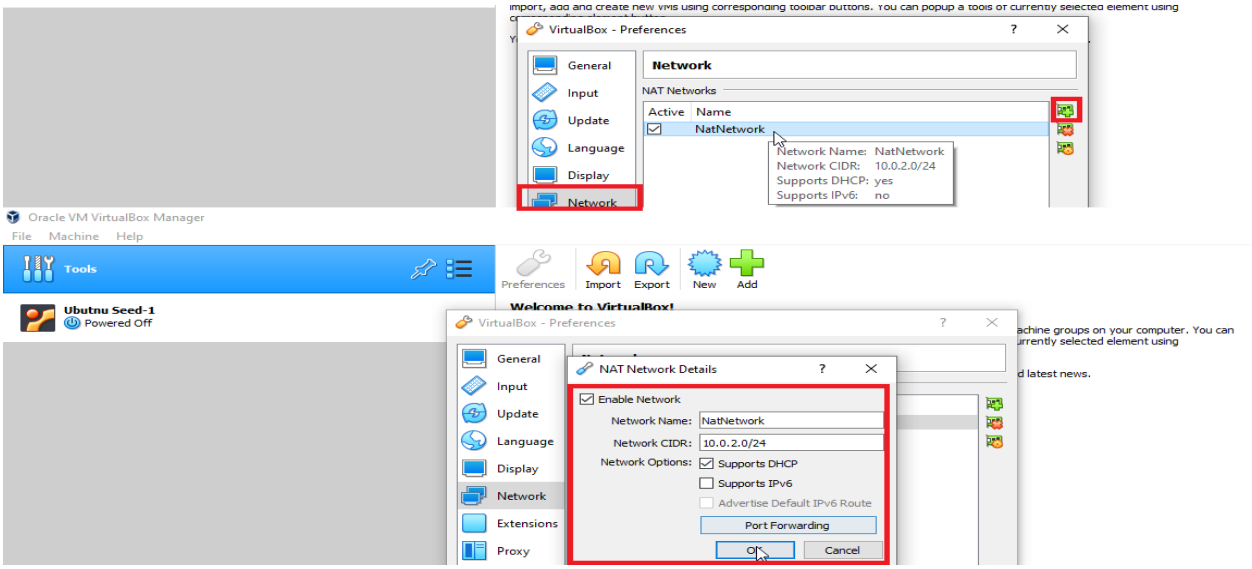

Figure 3: Creating new Nat Adapter from Virtual Box Setting

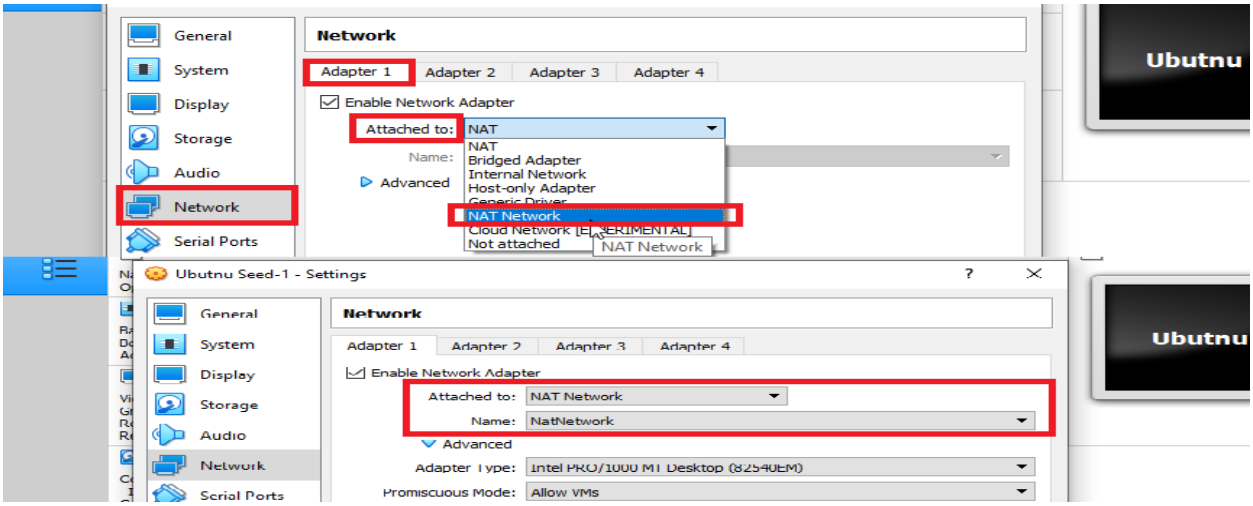

Figure 4: Adding Nat adapter in Virtual Machine setting

3. After that, I go to the VM settings and click on the shared folder to create the shared folder between the host and the virtual machine so that every time I put something in that folder, it can be accessed by both VM and host machine.

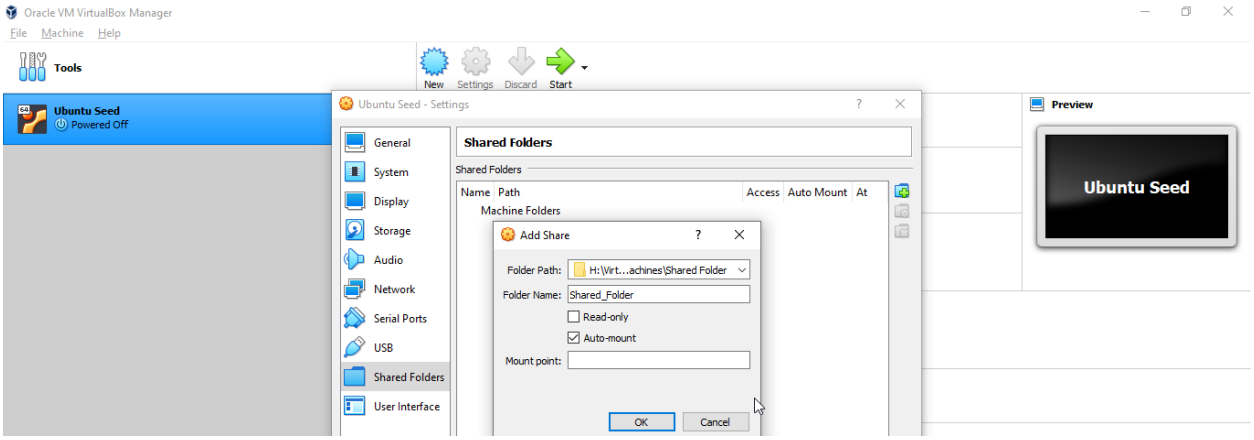

Figure 5: Go to Virtual Machine Shared Folder Setting

4. After that, I start the virtual machine, and then to mount the shared folder to the home directory of the Virtual machine I create a folder called “Share” in the home directory and then mount the shared folder to this “Share” folder.

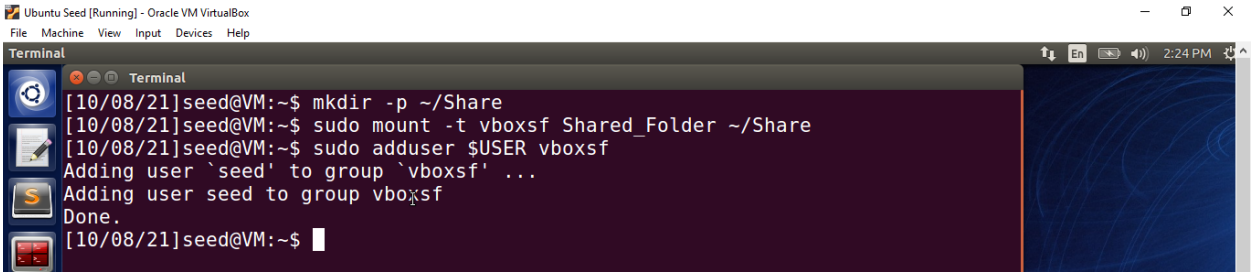

Figure 6: Mount the shared folder to the home directory folder

5. After that, I change the hostname of the virtual machine to my registration number to maintain the authenticity of the work.

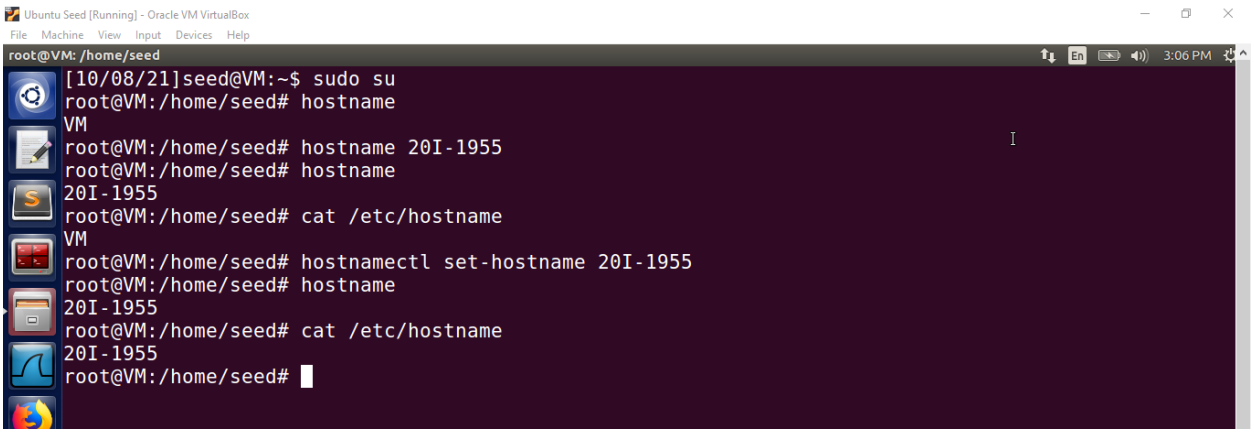

Figure 7: Changing the hostname of the virtual machine

Part-2 Public-Key Infrastructure Lab

In this lab, I have to create a digital certificate, deploy it on HTTPS Webserver and Apache-based HTTPS Website and then launch Man-In-The-Middle Attack on website and Certificate Authority.

Task-1 Becoming the Certificate Authority (CA)

In this task, I have to become the root certificate authority and then use this to generate the certificates for certificate authority. Now to start the task first I have to configure the OpenSSL.

Step-1 Configuring the OpenSSL

To configure the OpenSSL to create a certificate, I have to copy the OpenSSL configuration file from the "/usr/lib/ssl/openssl.cnf" location into the current working directory and then create the sub-directories specify in the configuration file.

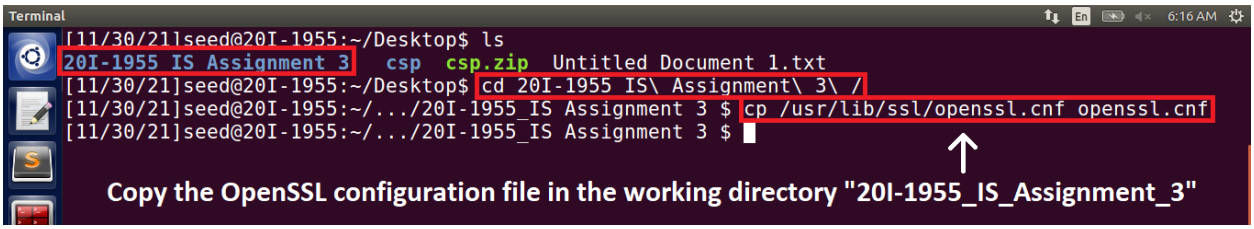

Figure 8: Copy the OpenSSL configuration file in the current directory

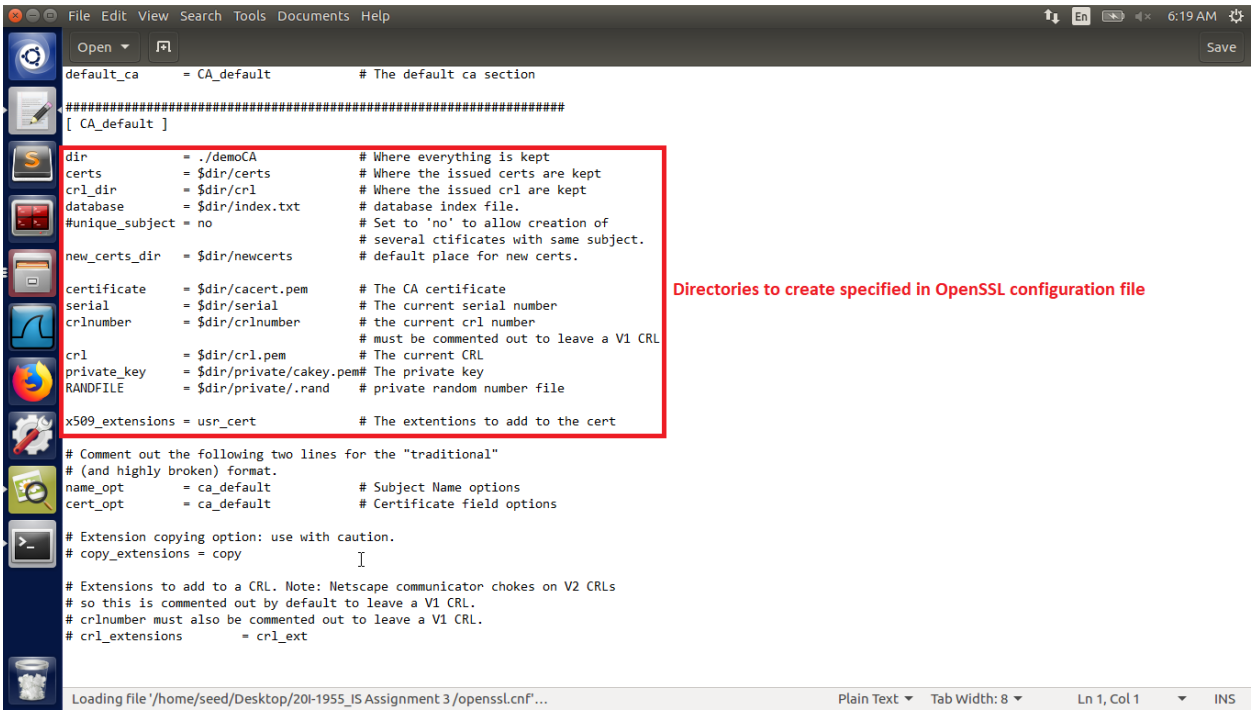

Figure 9: Directories to create specified in OpenSSL config file

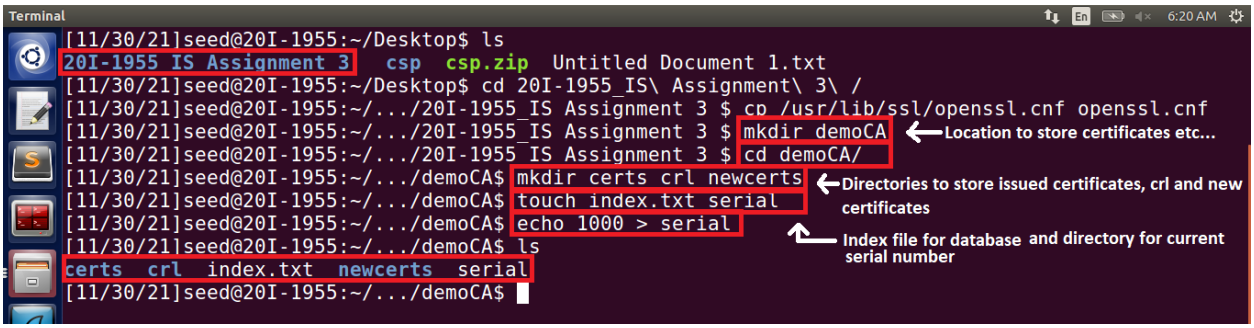

Figure 10: Creating Directories specified in OpenSSL config file

Step-2 Certificate Authority (CA)

In this task I have to generate a self-signed certificate for our Certificate Authority and this certificate will serve as a root certificate. For the creation of the certificate, I used the following command "**openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf**". In this command "**ca.crt**" contains the CA's public key and "**ca.key**" contains CA's private key. After running the command certificate for CA created successfully. In Figure 13 we can see the same information in subject and issuer, indicating the self-signed certificate. Also, we can see in Figure 14 under "Basic Constraints" Certificate Authority is marked "Yes". This indicates that this certificate is for CA and can be used to sign other certificates.

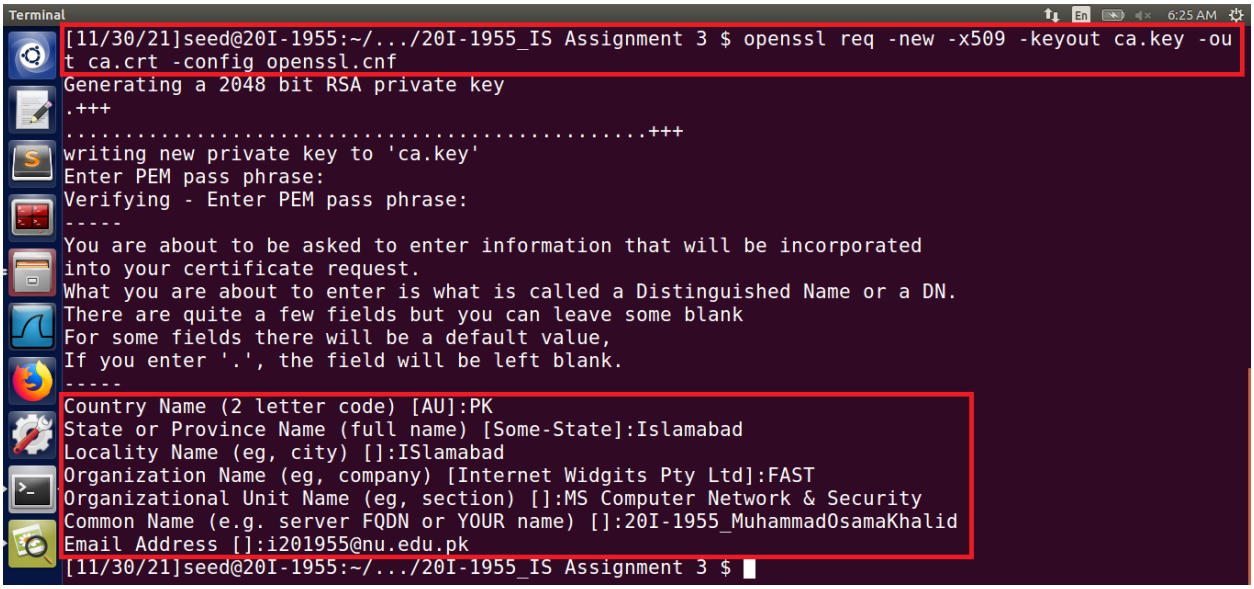

Figure 11: Generating certificate for CA

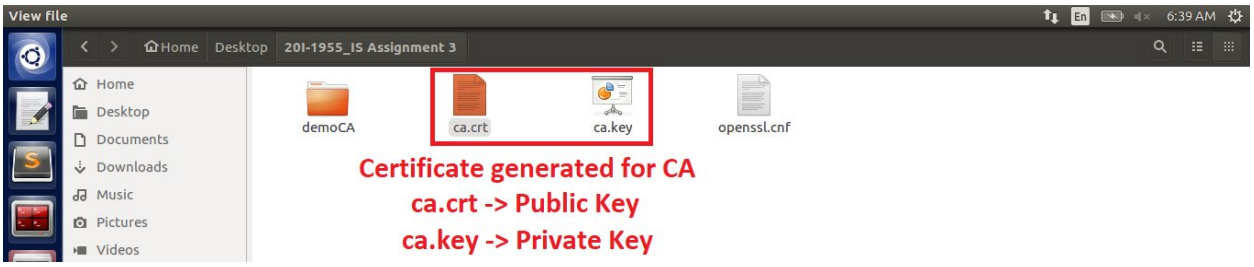

Figure 12: Certificate generated successfully

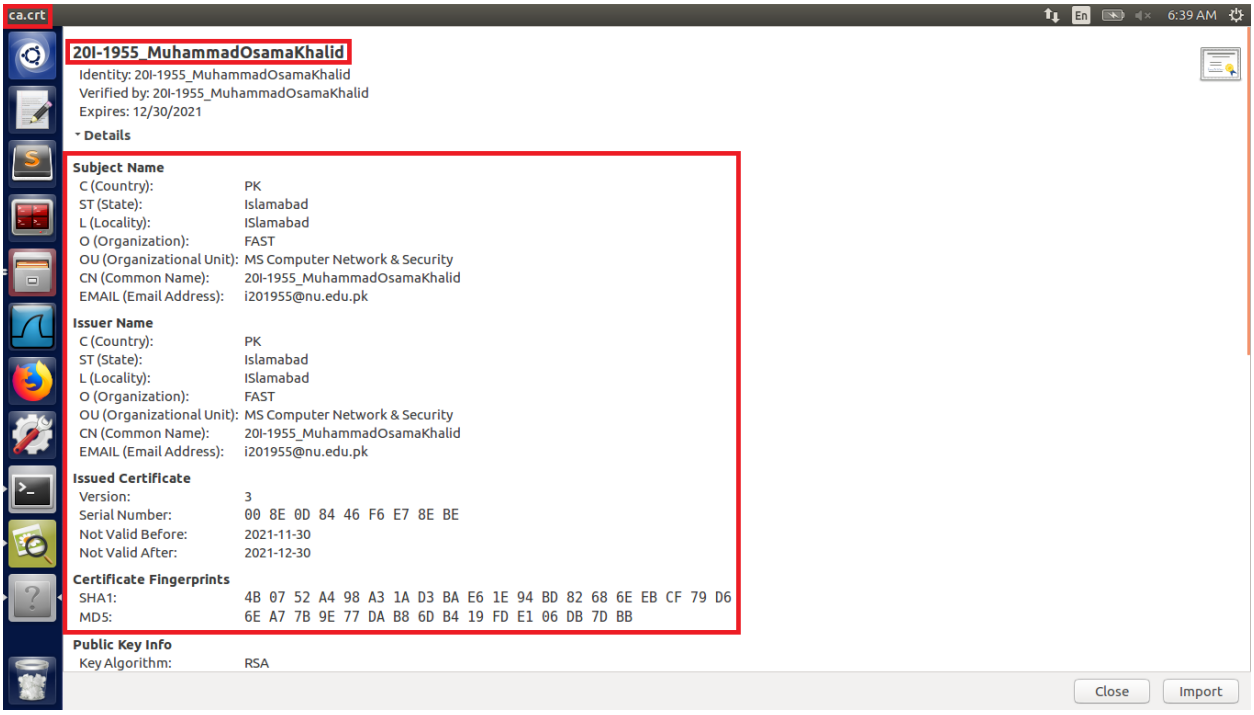

Figure 13: Contents of ca.crt

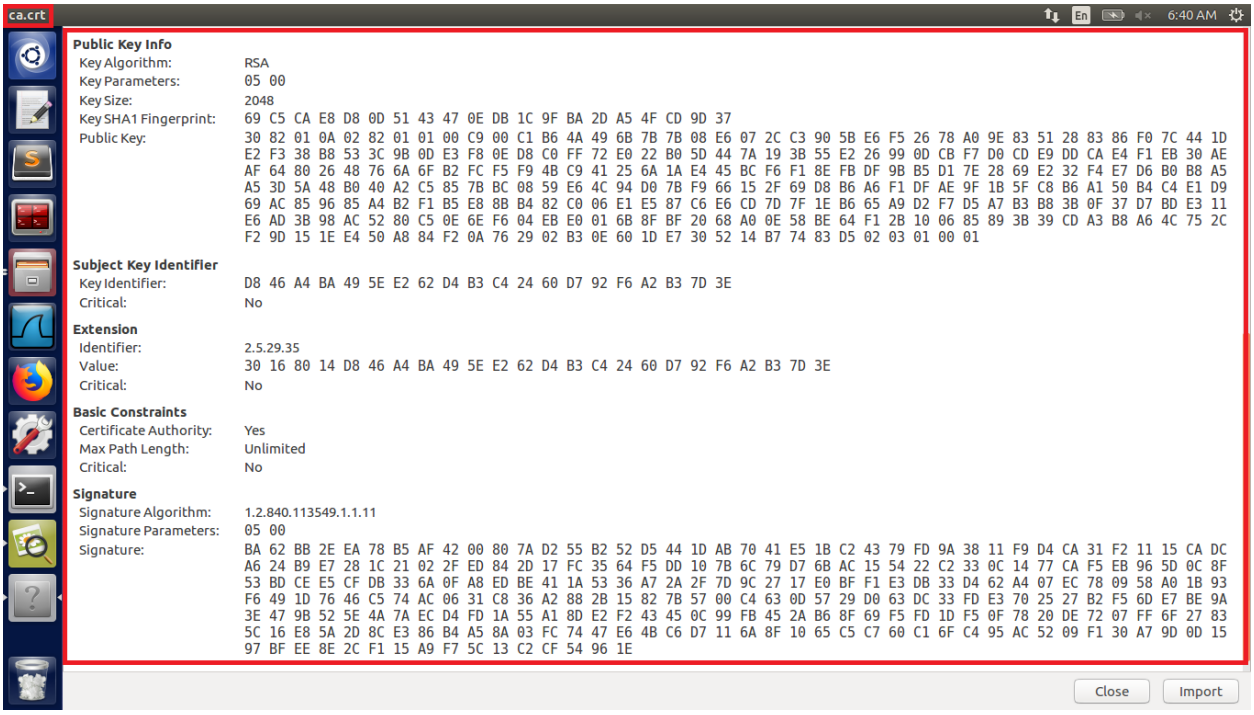

Figure 14: Contents of ca.crt

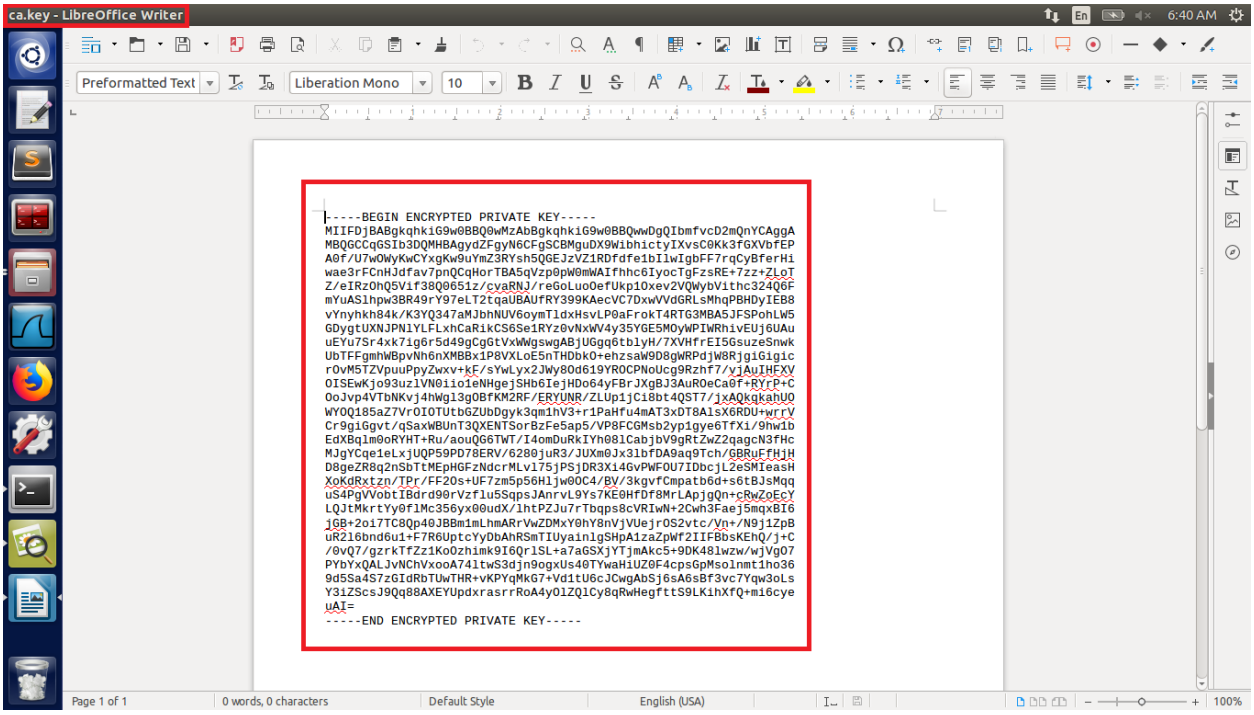

Figure 15: Contents of ca.key

Task-2 Creating a Certificate for osamakhalid.com

Now as we are CA, we can sign digital certificates for the companies' websites and the first website I am going to issue the certificate is "osamakhalid.com". For issuing the certificate to the website I have to go through the following steps.

Step-1 Generate public/private key pair

For getting the certificate from CA, the company first needs to create its public/private key pair. To create the RSA key pair I use the command "**openssl genrsa -aes128 -out server.key 1024**" and also provide the password to encrypt the private key. The private key was encrypted by an AES-128 encryption algorithm. In the command "**server.key**" store the keys. As the "**server.key**" is an encoded text file so to see the contents of the file I use the command "**openssl rsa -in server.key -text**".

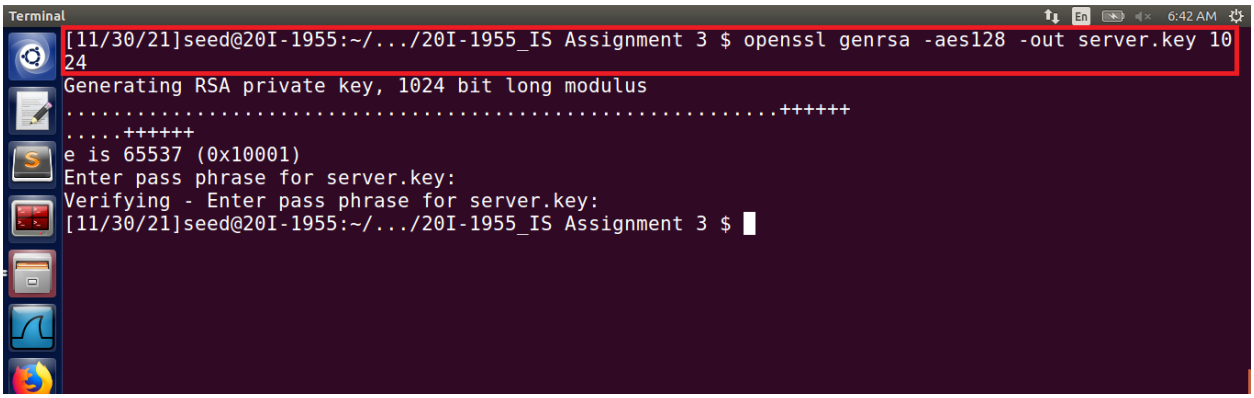

Figure 16: Generating the Public/Private key

Figure 17: Public/Private key generated and stored in the server.key

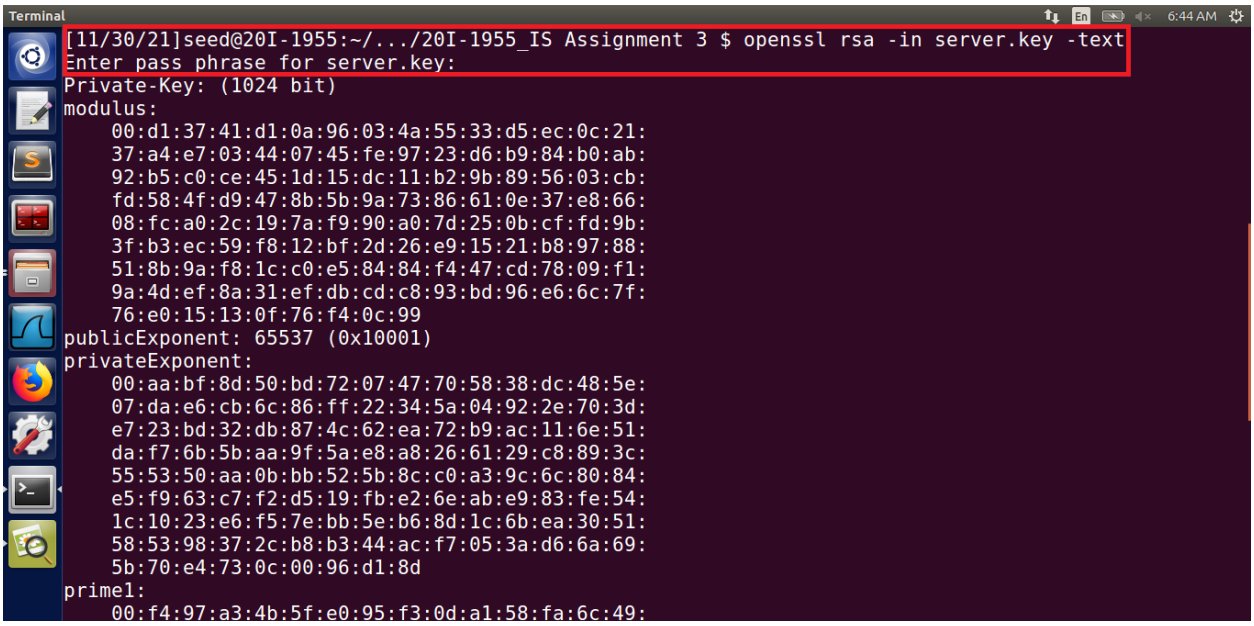

Figure 18: Contents of the server.key

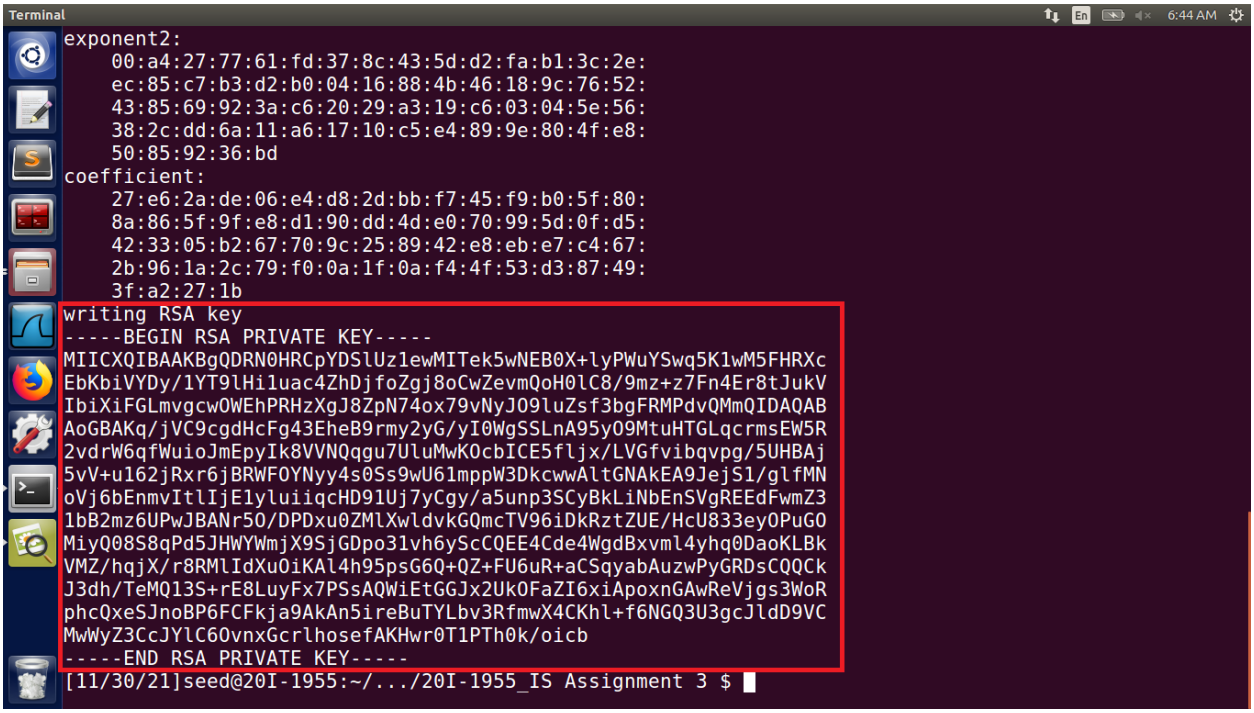

Figure 19: Contents of the server.key

Step-2 Generate a Certificate Signing Request (CSR)

Once the company generates its public and private key file their next task is to generate a Certificate Signing Request (CSR) that contains companies generated public kay and then send this request to CA. On receiving the request then CA will generate the certificate for the key. Now to generate the CSR I use the command "**openssl req -new -key server.key -out server.csr -config openssl.cnf**".

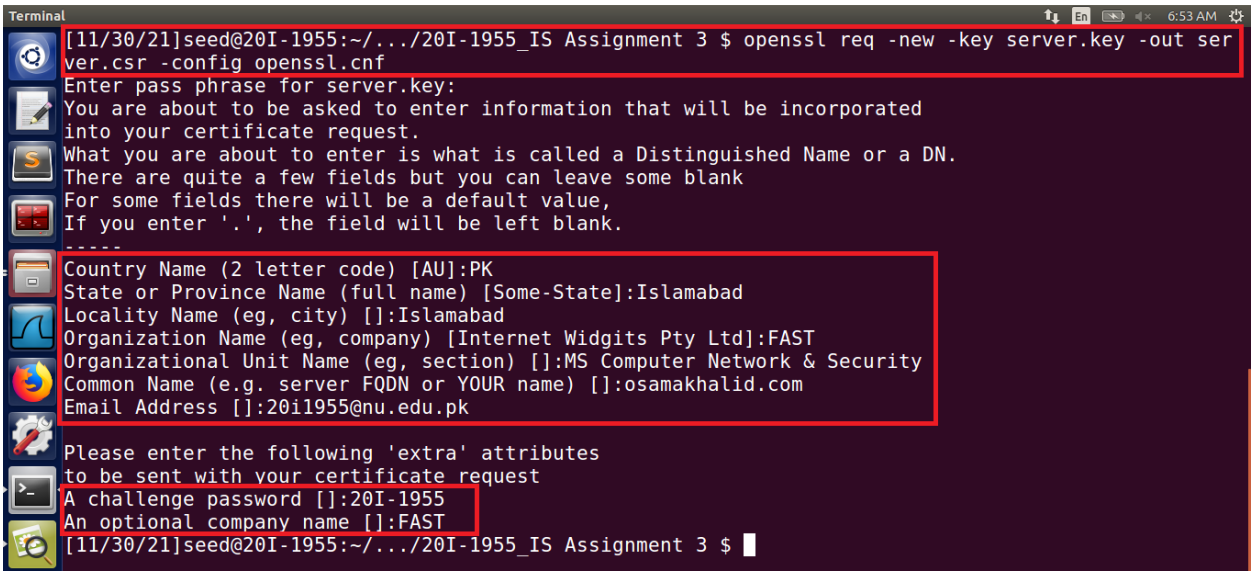

Figure 20: Generating the CSR

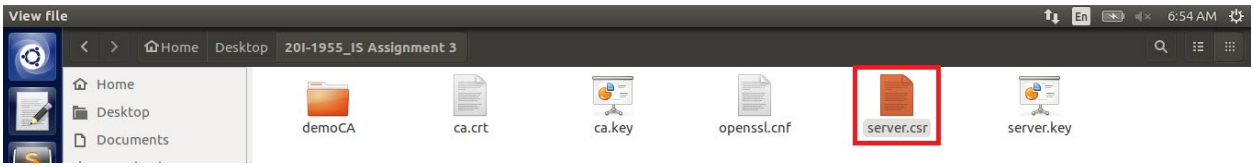

Figure 21: CSR generated successfully

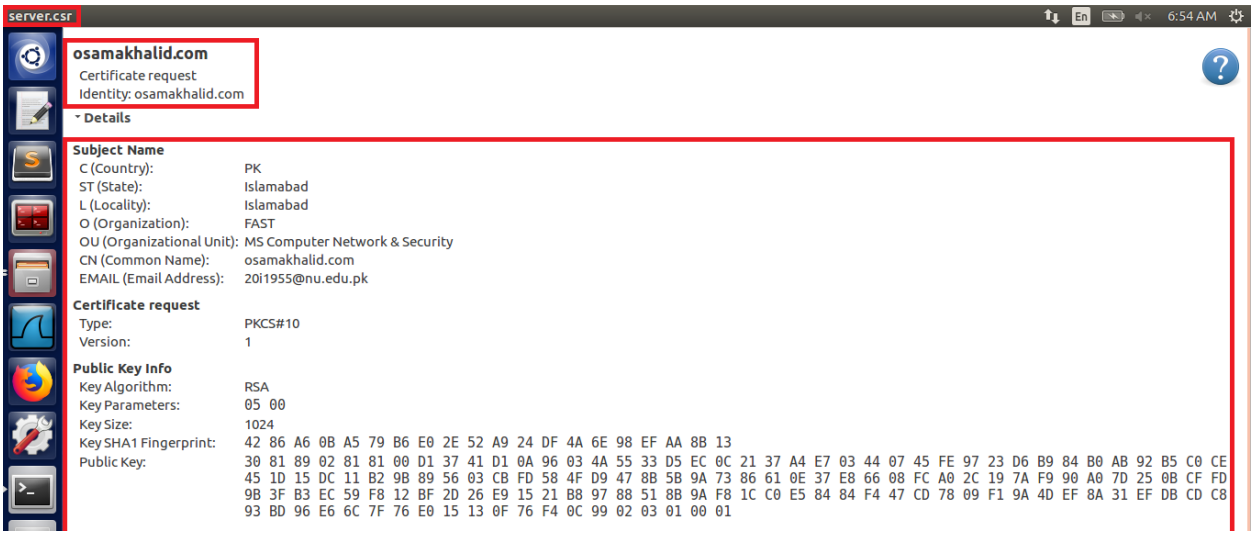

Figure 22: Contents of the server.csr

Step-3 Generating Certificates

On receiving the CSR request from the company, the next task is for the CA to generate the certificate for the company, and for that CSR file needs to have the CA's signature. Now to generate the certificate for the company I use the following command "**openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf**". This command will turn the CSR (server.csr) into an X509 certificate (server.crt) using the CA's "**ca.crt**" and "**ca. the key**". Now before running the command, I change the policy in the "**openssl.cnf**" file to policy_anything from policy_match to avoid any errors.

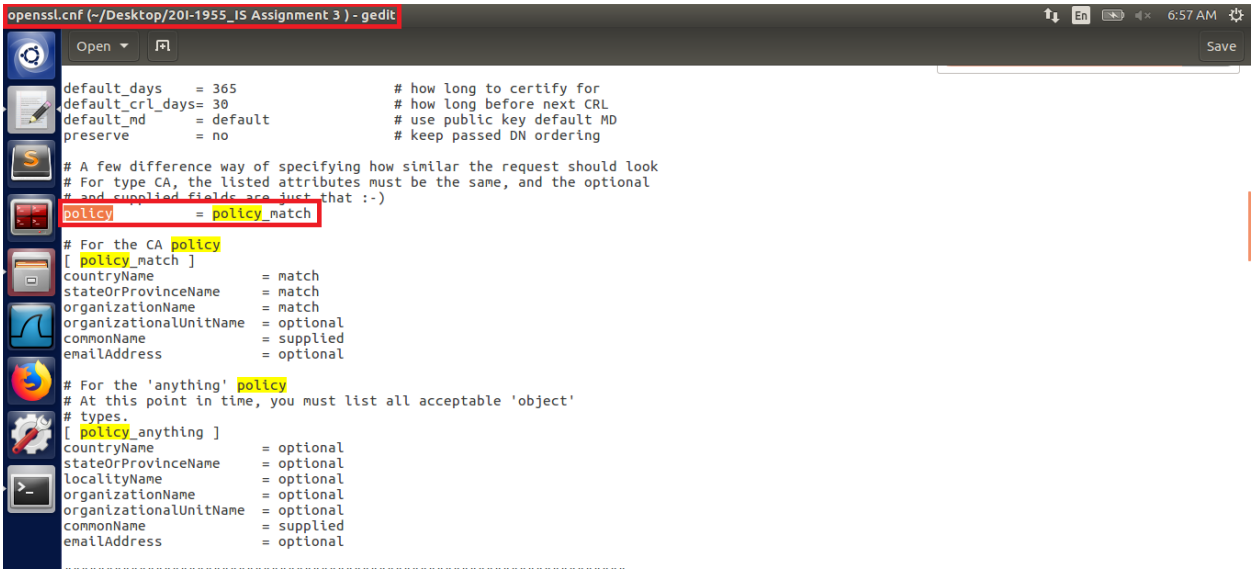

Figure 23: Changing the policy in openssl.cnf

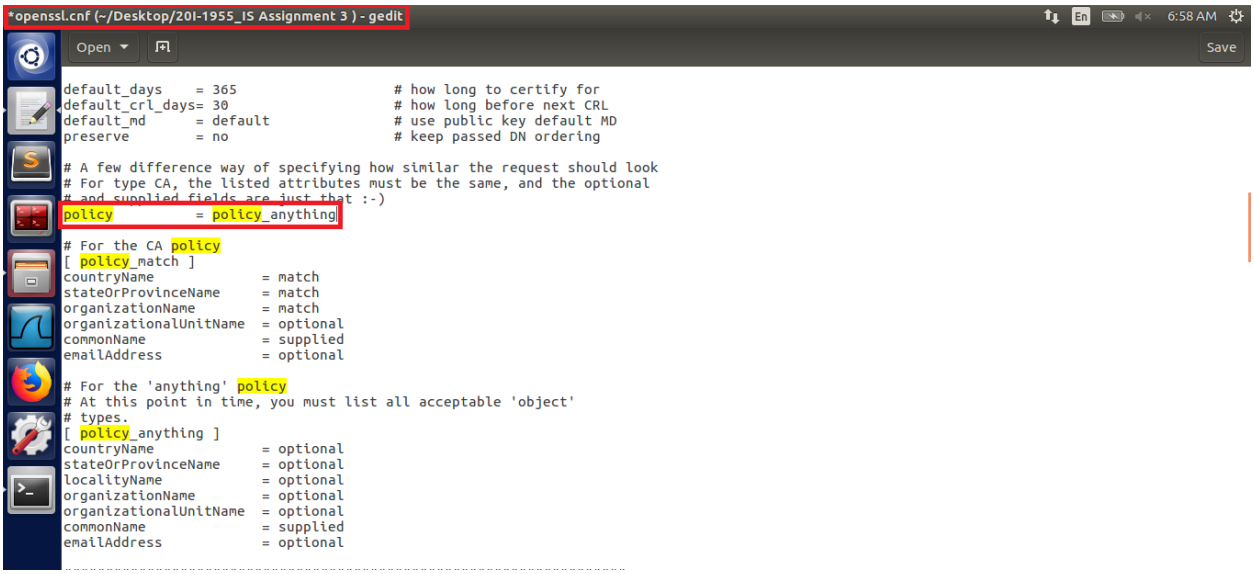

Figure 24: Changing the policy in openssl.cnf

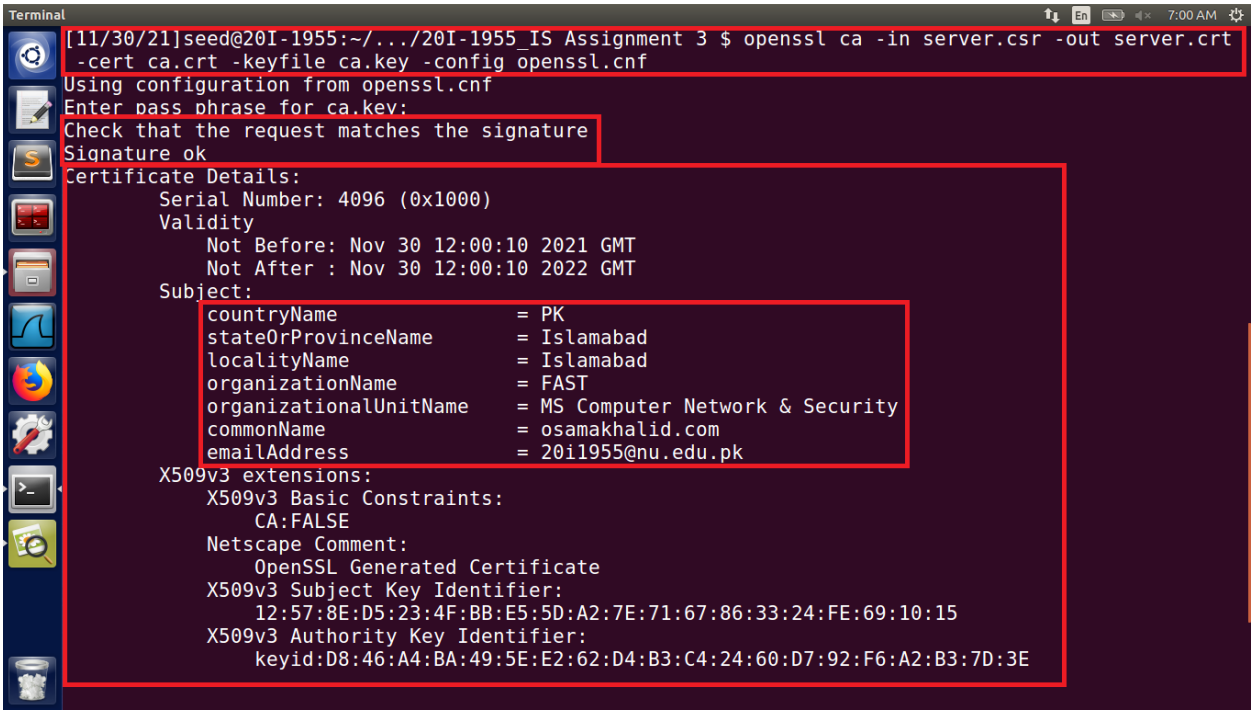

Figure 25: Generating the certificate for the company

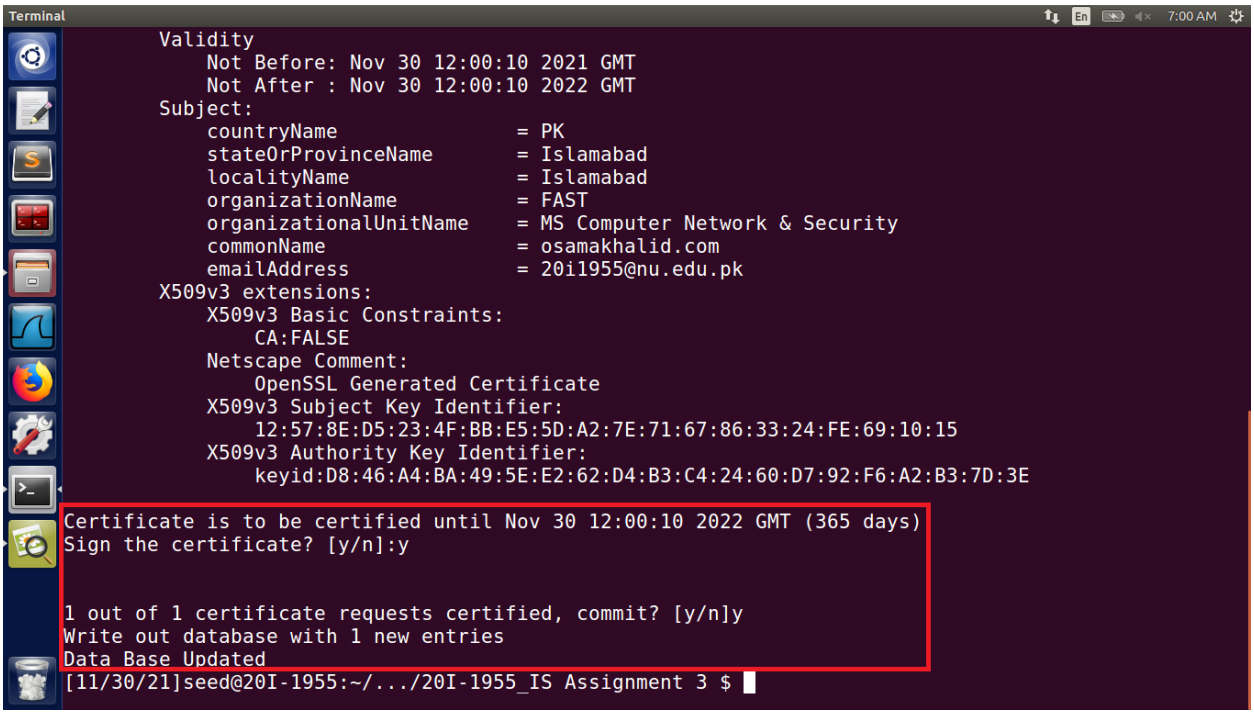

Figure 26: Generating the certificate for the company

Task-3 Deploying Certificate in an HTTPS Web Server

On receiving the certificate from the CA, the company's next task is to use this certificate with the website to secure web browsing. As I am using the lab setup for this task, I first have to set up the HTTPS website using OpenSSL's built-in web server.

Step-1 Configuring DNS

As the website, I am using is “osamakhalid.com”, in this step I need to add this into the “/etc/hosts” file for the machines to recognize the website on the web browser. To add the website in “/etc/hosts” I use the command “**sudo gedit /etc/hosts**” and then to confirm the changes I use the command “**cat /etc/hosts**”.

Figure 27: Adding the website details in the/etc/hosts file

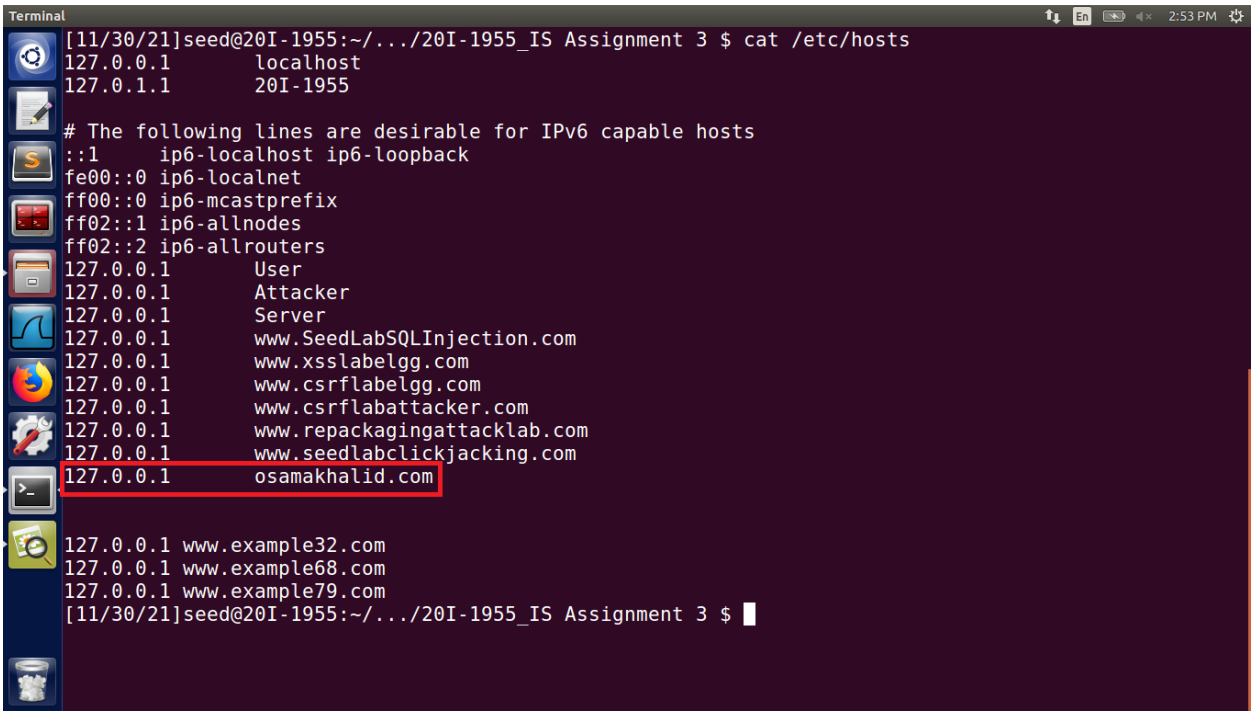

Figure 28: Confirming the changes

Step-2 Configuring the webserver

In this step, I need to start the simple webserver using the s_server command but first I need to combine the key (server.key) and certificate (server.crt) into one file that is generated in the previous task. To combine the key and certificate into one file I use the following commands "**cp server.key server.pem**" and "**cat server.crt >> server.pem**". Then to launch the webserver using "**server.pem**", I use the command "**openssl s_server -cert server.pem -www**".

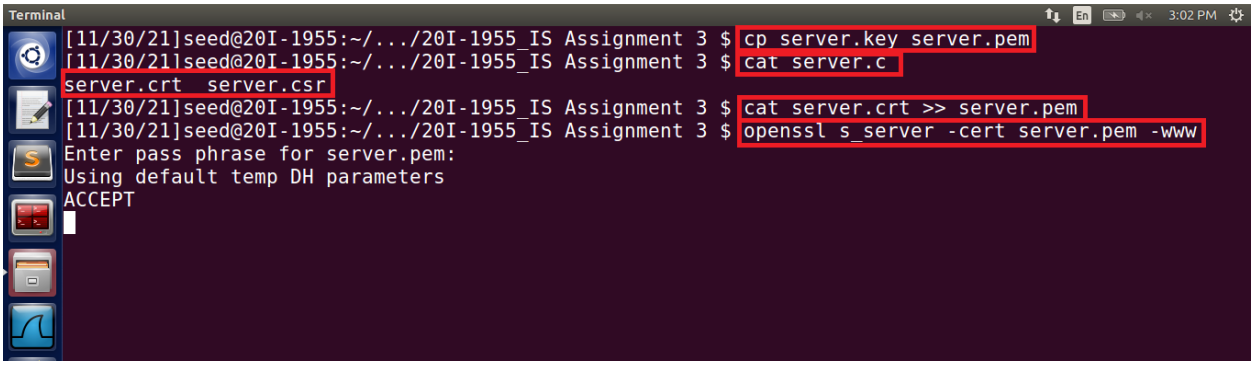

Figure 29: Starting the webserver

By default, the server listens on port 4433 and I can access the server using the URL https://osamakhalid.com:4433. On visiting the URL, I see the web browser is displaying the error that the connection is not secure. On clicking on the advance button, I see the error that the issuer certificate is unknown. This is because, even I create the certificate for CA, but this certificate is not known to the browser and due to this web browser cannot validate the website certificate.

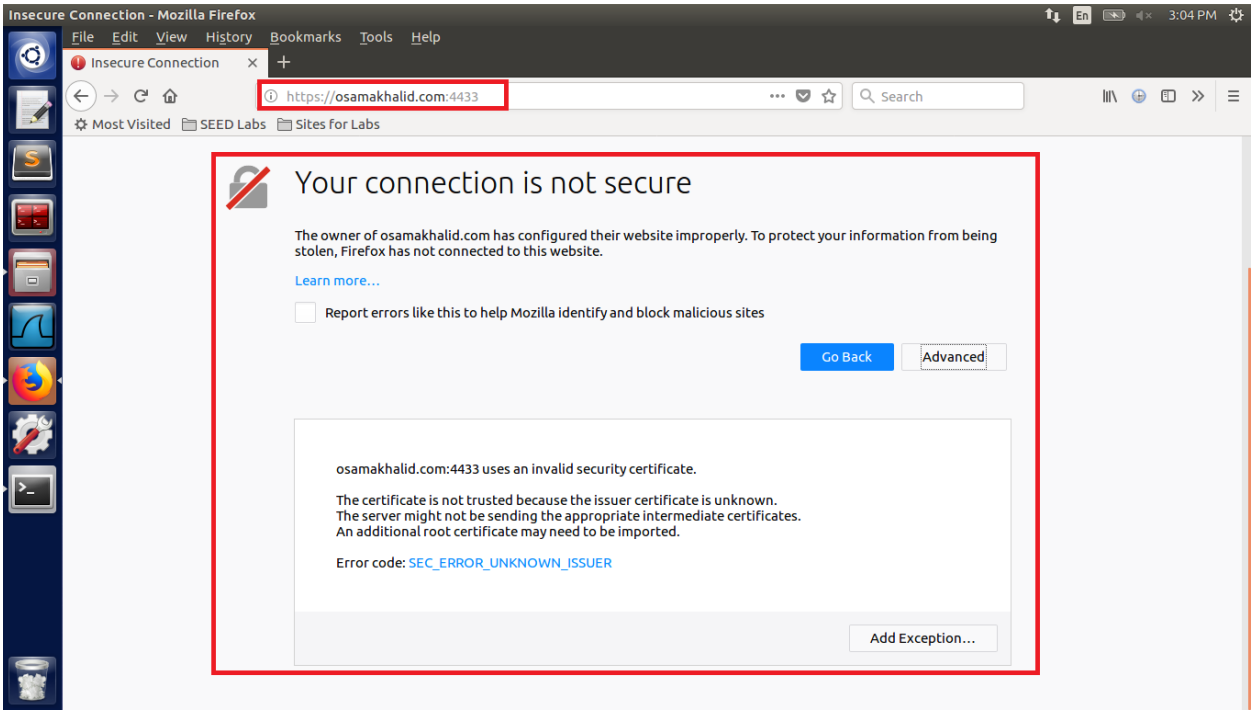

Figure 30: Visiting the website

Step-3 Getting the browser to accept our CA certificate

As we see in the previous step the browser didn't know about our CA certificate so, in this step, I have to manually add the CA's certificate in the browser. To add the CA's certificate in the browser that in this case is Firefox, I go to Menu > preferences > Privacy & Security > Certificates > View Certificates. After that from the popup, I click on "Authorities" and then click on the Import button to import the CA's certificate to the browser. Then I select the certificate and click open. After that in the "Downloading Certificate" popup, I mark the checkboxes and click ok. After that, we can see our CA's certificate in the Certificate Manager, Authorities section.

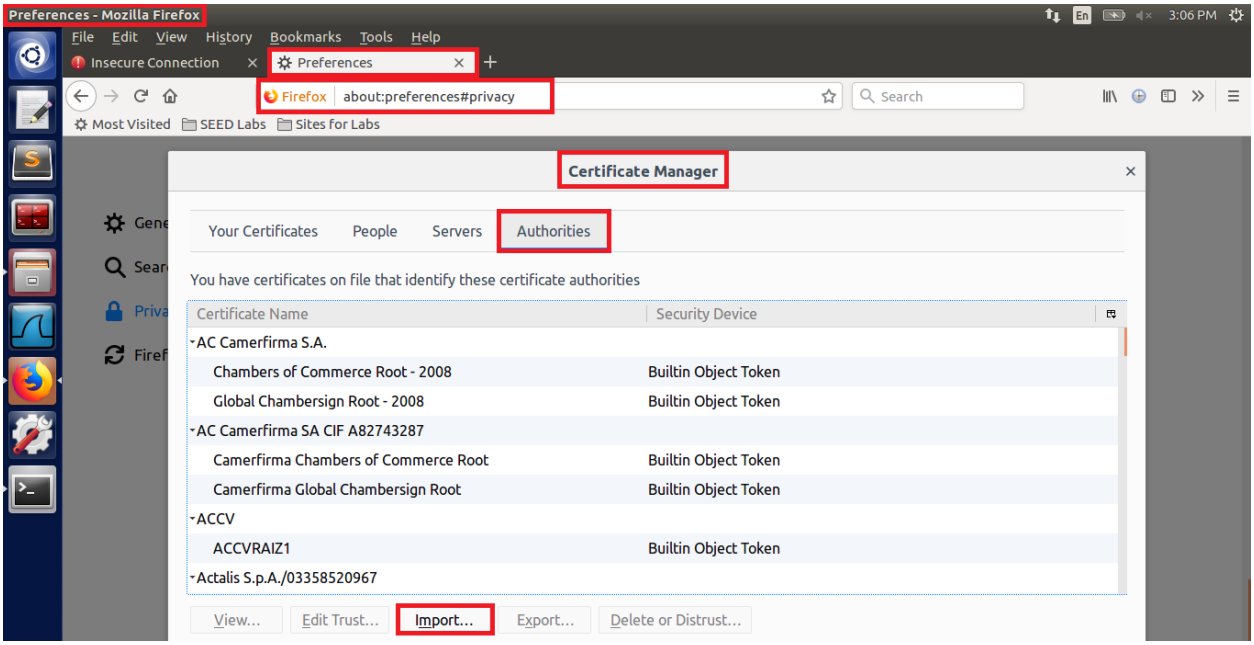

Figure 31: Adding CA's certificate into the browser

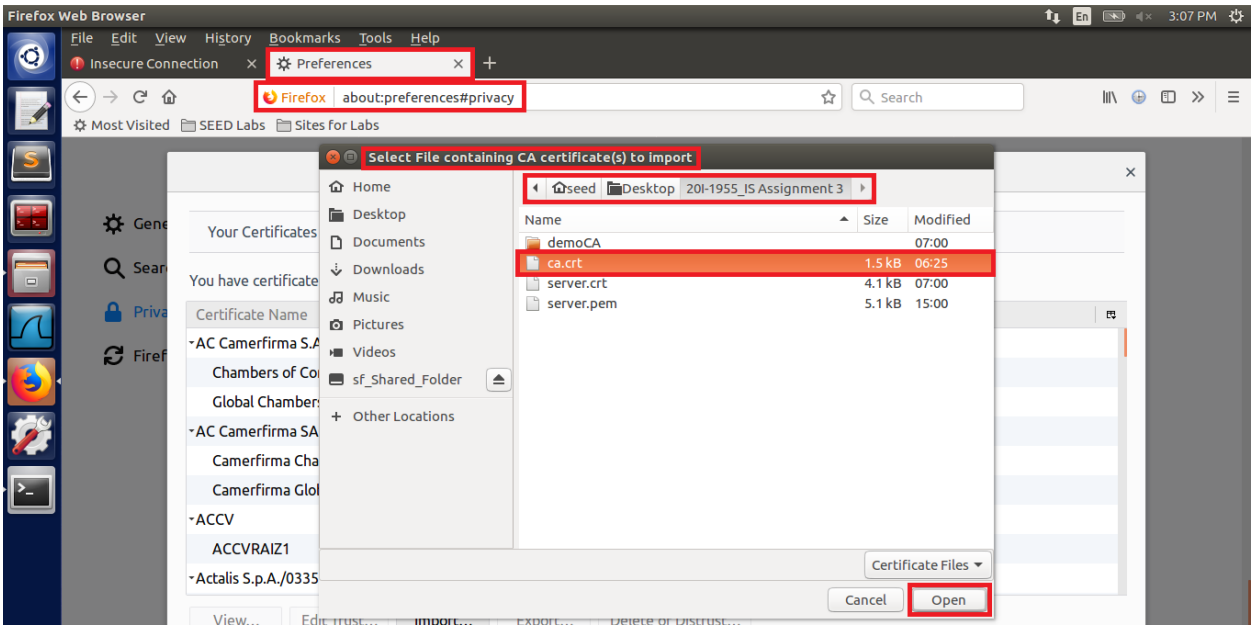

Figure 32: Adding CA's certificate into the browser

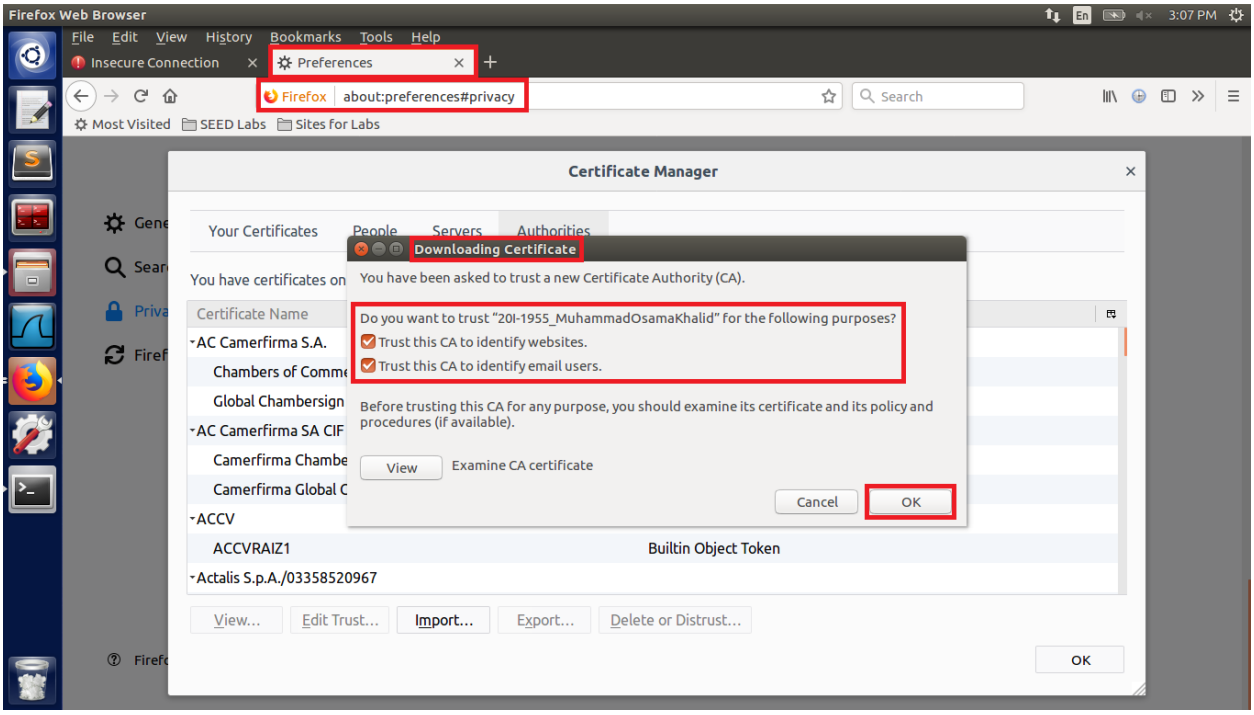

Figure 33: Adding CA's certificate into the browser

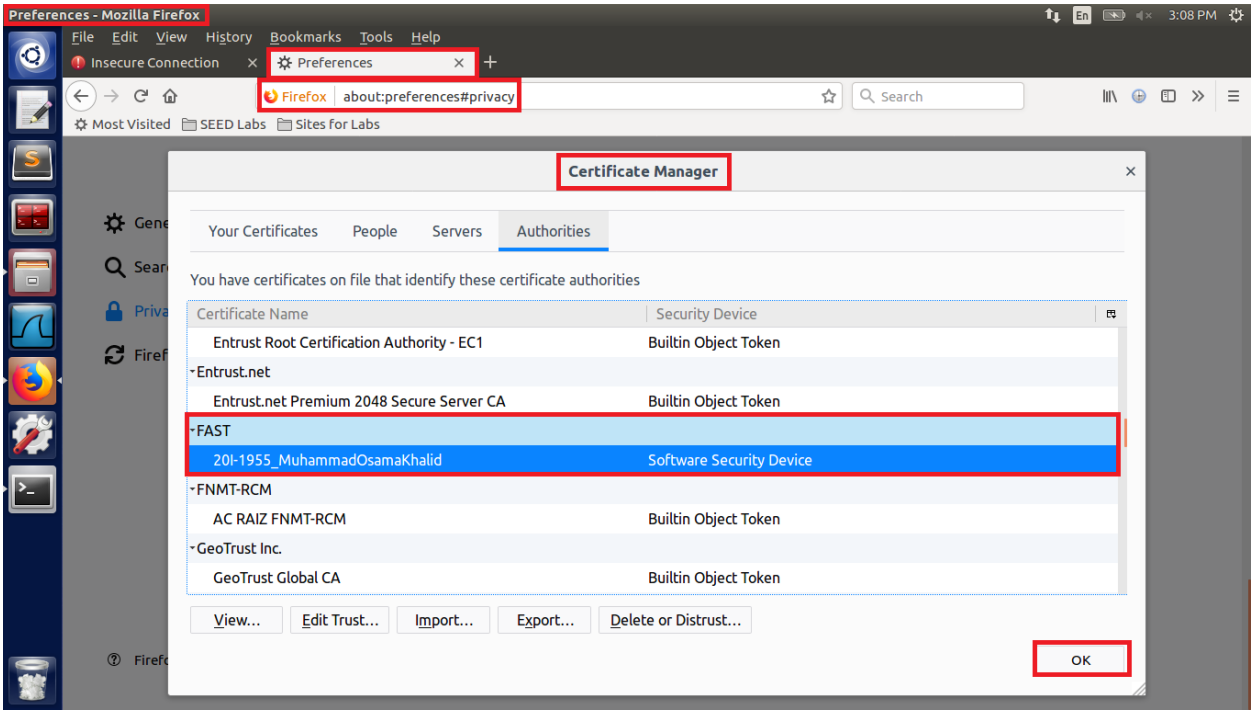

Figure 34: CA's certificate added to the browser successfully

Step-4 Testing our HTTPS website

Now as I add our CA's certificate into the browser now, I have to again visit the URL https://osamakhalid.com:4433. On visiting the URL, I see that the webpage loads successfully and we can see the reply from the server. Moreover, we can see that the webpage is also secure.

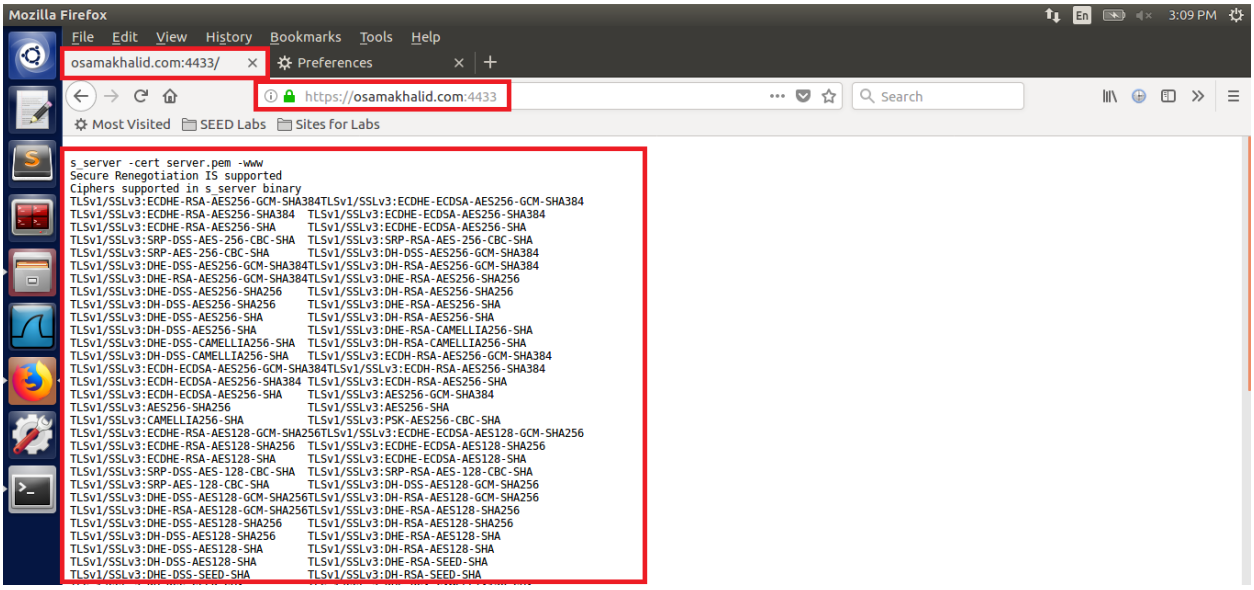

Figure 35: Webpage loads successfully

Q#1 Modify a single byte of the server.pem, and restart the server, and reload the URL. What do you observe? Make sure you restore the original server.pem afterward.

In this question, I have been asked to modify a single byte in the "server.pem" file and then again visit the website. To change modify the "server.pem" file I open the file in the "**Bless Hex Editor**" and then change the single byte of 7A to 7B and then save the file. After that I again run the s_server using the command "**openssl s_server -cert server.pem -www**" and then again visit the URL https://osamakhalid.com:4433. On visiting the URL, I see the "Secure Connection Failed" error which moreover indicates that the certificate has an invalid signature. This indicates that a single change in the certificate's signature will make it invalid and causes the browser to throw a "Secure Connection Failed" error.

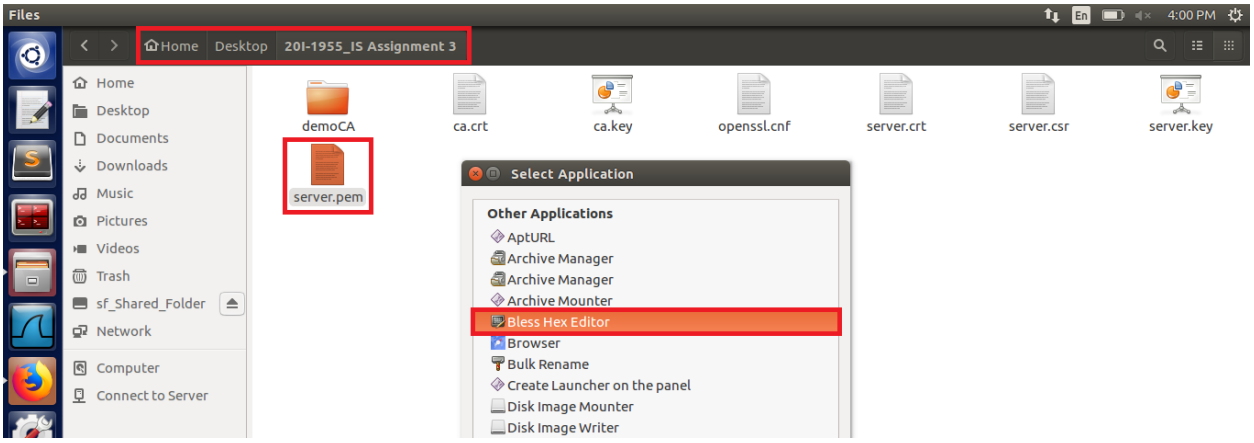

Figure 36: Opening the server.pem file

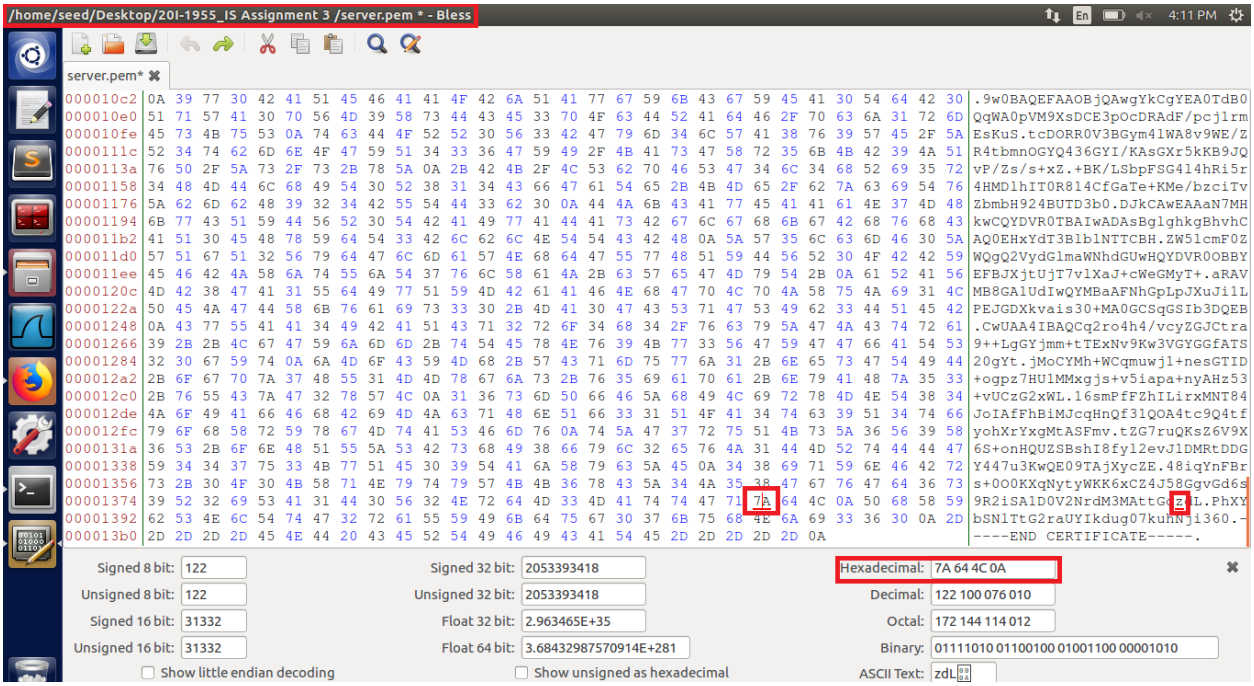

Figure 37: Changing a single bit from 7A to 7B

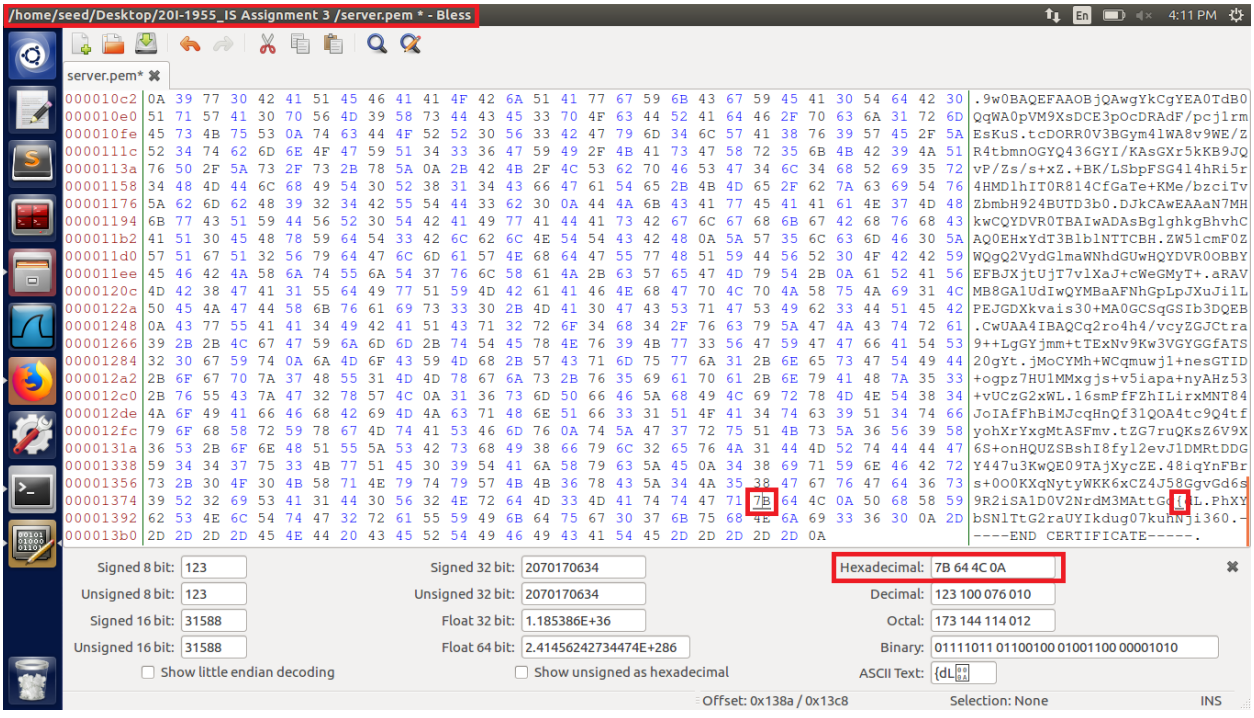

Figure 38: Changing a single bit from 7A to 7B

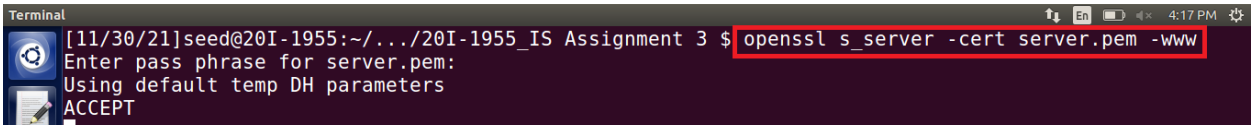

Figure 39: Run the server again

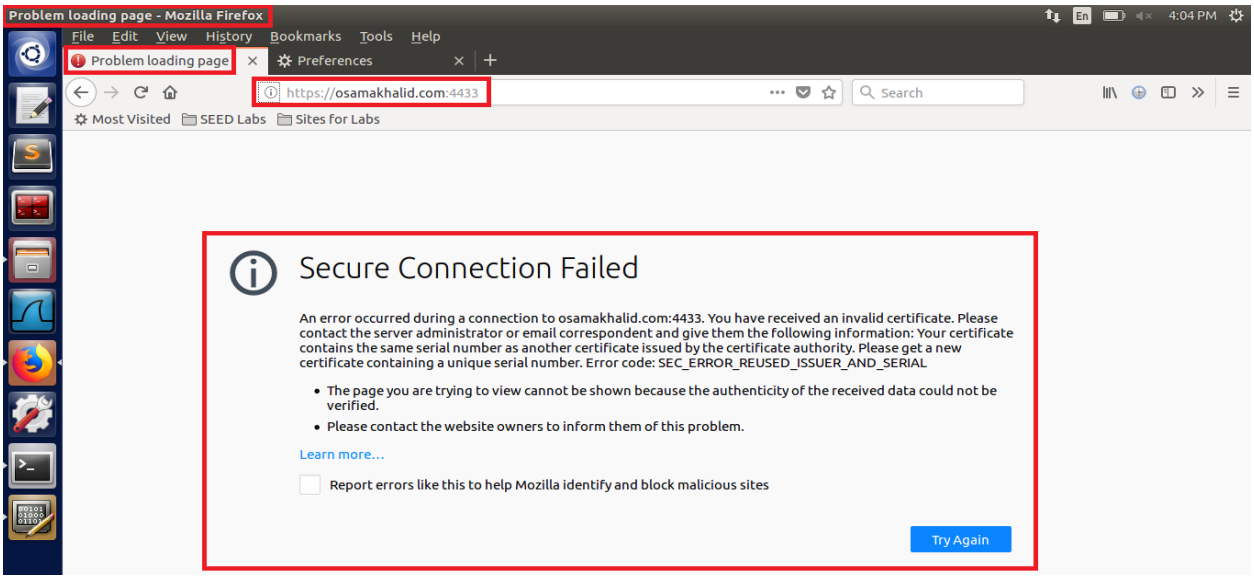

Figure 40: Webpage shows error

Q#2 Since osamakhalid.com points to the localhost, if we use https://localhost:4433 instead, we will be connecting to the same web server. Please do so, describe and explain your observations.

In this question I have been asked to visit localhost instead of osamakhalid.com, for that, I open the browser and visit the URL https://localhost:4433. On visiting the URL, I found that the browser is throwing the error "Your Connection is Not Secure". Moreover, on clicking on the advance button I found the error that indicates that localhost is using the invalid certificate and the certificate used is not valid for localhost. After that, on clicking on "Add Exception" and then clicking on the view button I found that the certificate used is only issued to "osamakhalid.com". This proves that the browser also checks the correctness of the URL requested and the certificate.

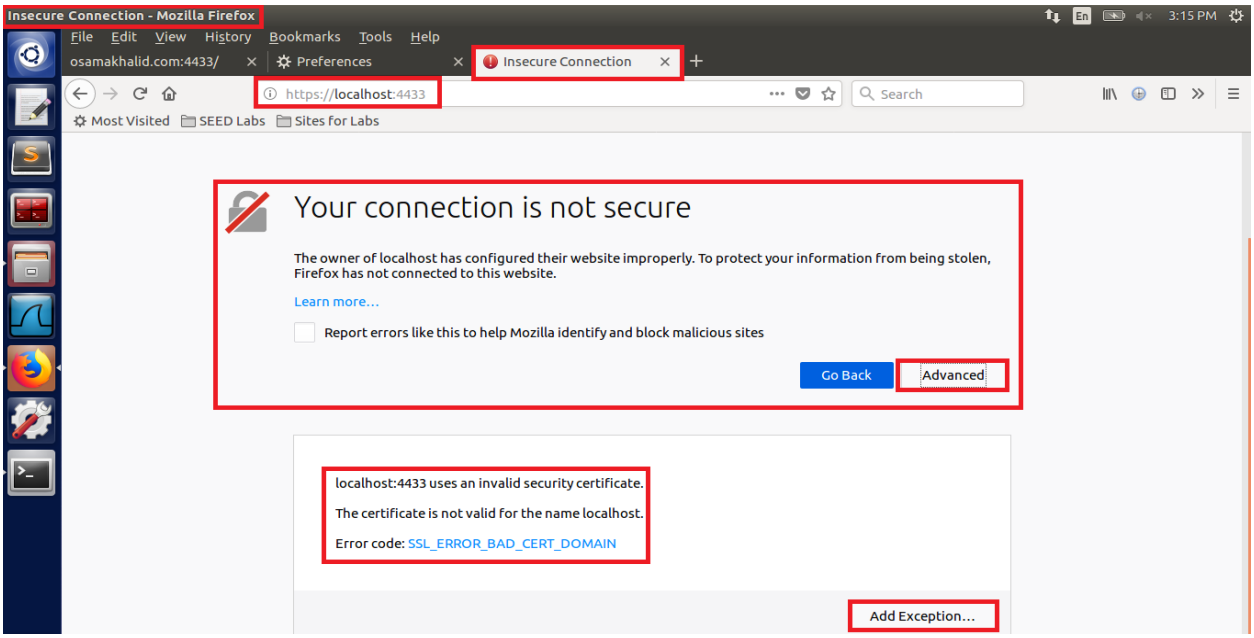

Figure 41: Browser throwing the "Connection not secure" error

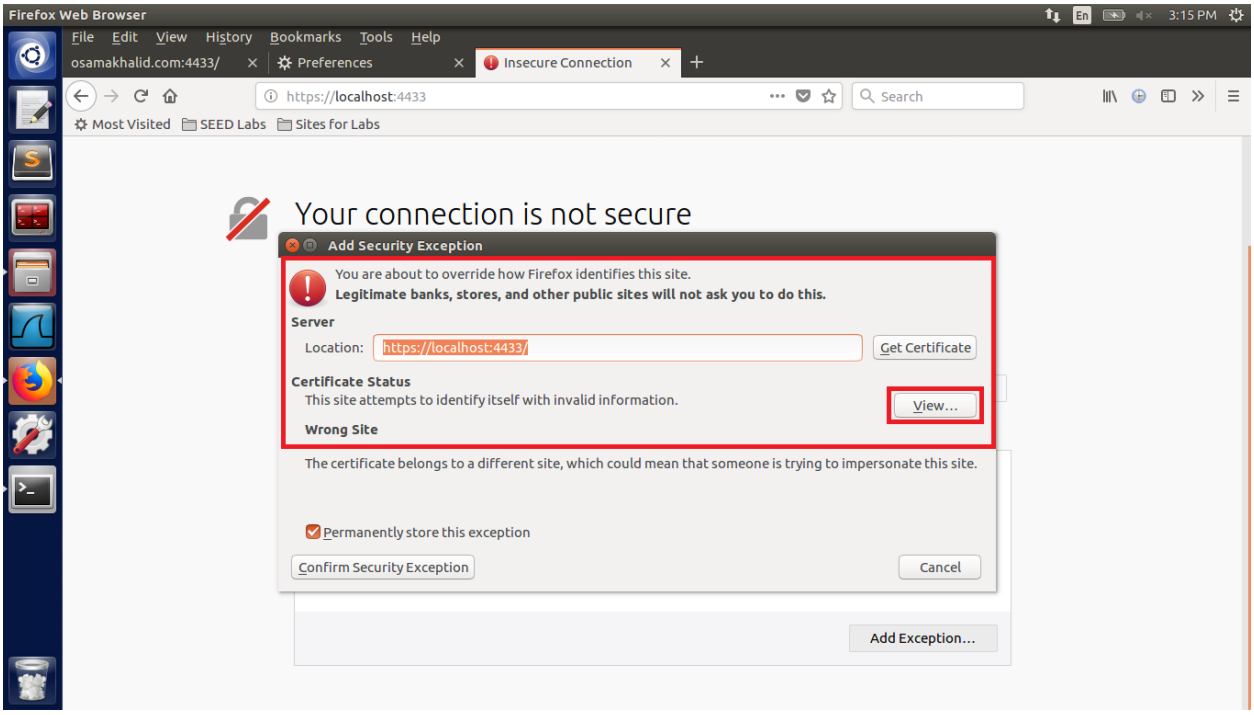

Figure 42: Viewing the Details of the certificate used

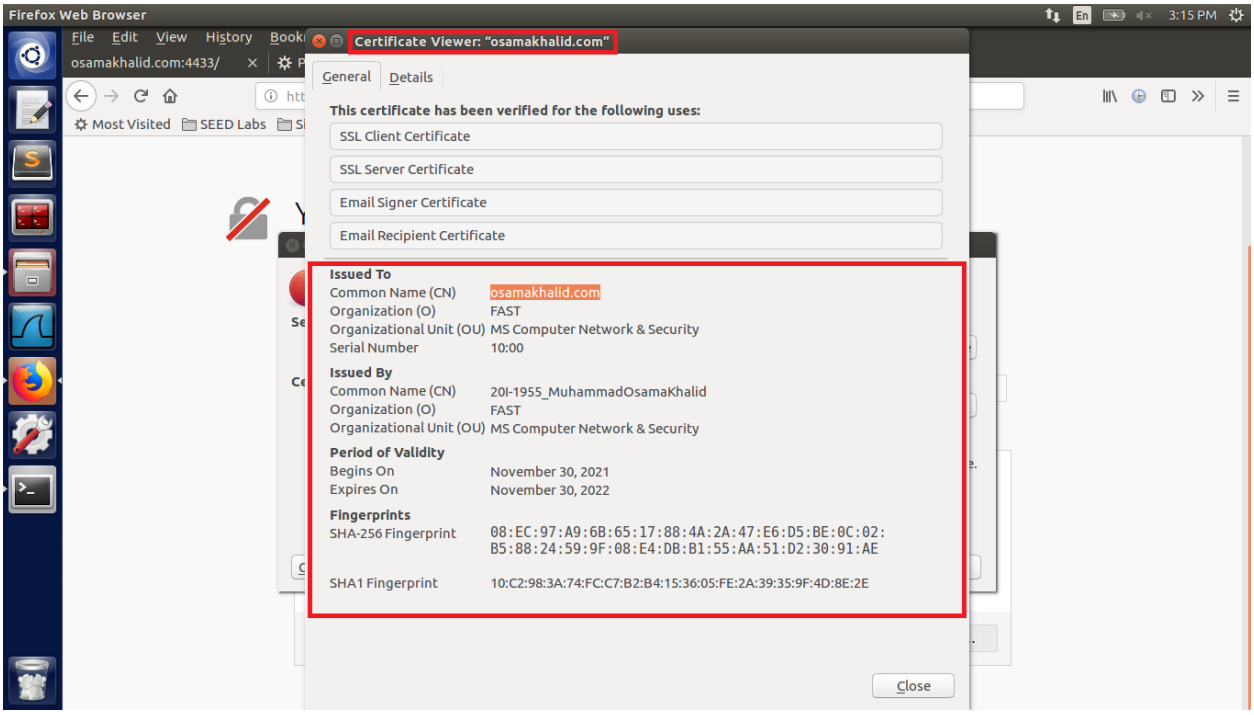

Figure 43: Certificate issued to osamakhalid.com, not localhost

Task-4 Deploying Certificate in an Apache-based HTTPS Website

As in the previous task, we are using OpenSSL's s_server, now in this task, I have to set up the real HTTPS web server based on Apache. First, I need to combine the key (server.key) and certificate (server.crt) into one file and then use this file in the website configuration. To combine the key and certificate into one file I use the following commands "**cp server.crt osamakhalid_cert.pem**" and "**cp server.key osamakhalid_key.pem**".

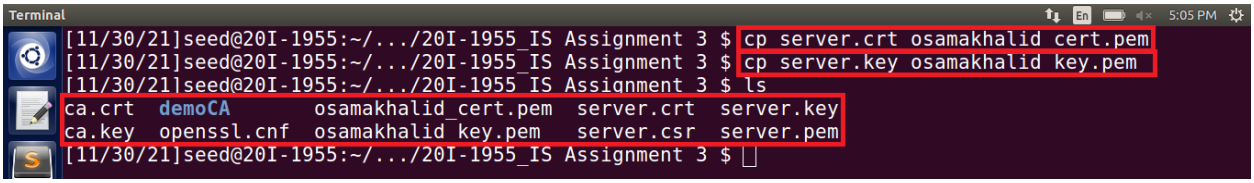

Figure 44: Combine the certificate and key into one file

As Apache server can simultaneously host multiple websites therefore it needs to know that where the website files are being stored and this is done by the VirtualHost file stored in the "/etc/apache2/sites-available" directory. To add the website, I need to add the VirtualHost entry to the two files "000-default.conf" and "default-ssl.conf". The entry "ServerName" specifies the name of the website, "DocumentRoot" specifies where the website files are stored, and "SSLCertificateFile", "SSLCertificateKeyFile" specifies where the certificate and private key are stored.

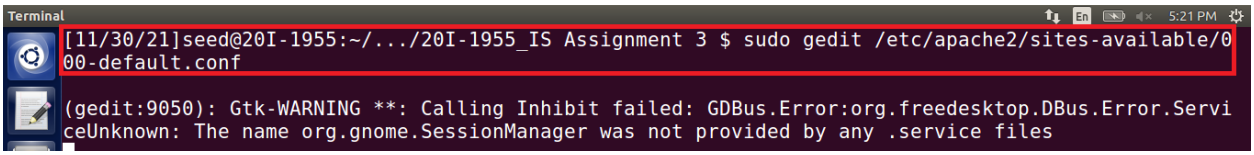

Figure 45:Opening the 000-default.conf file

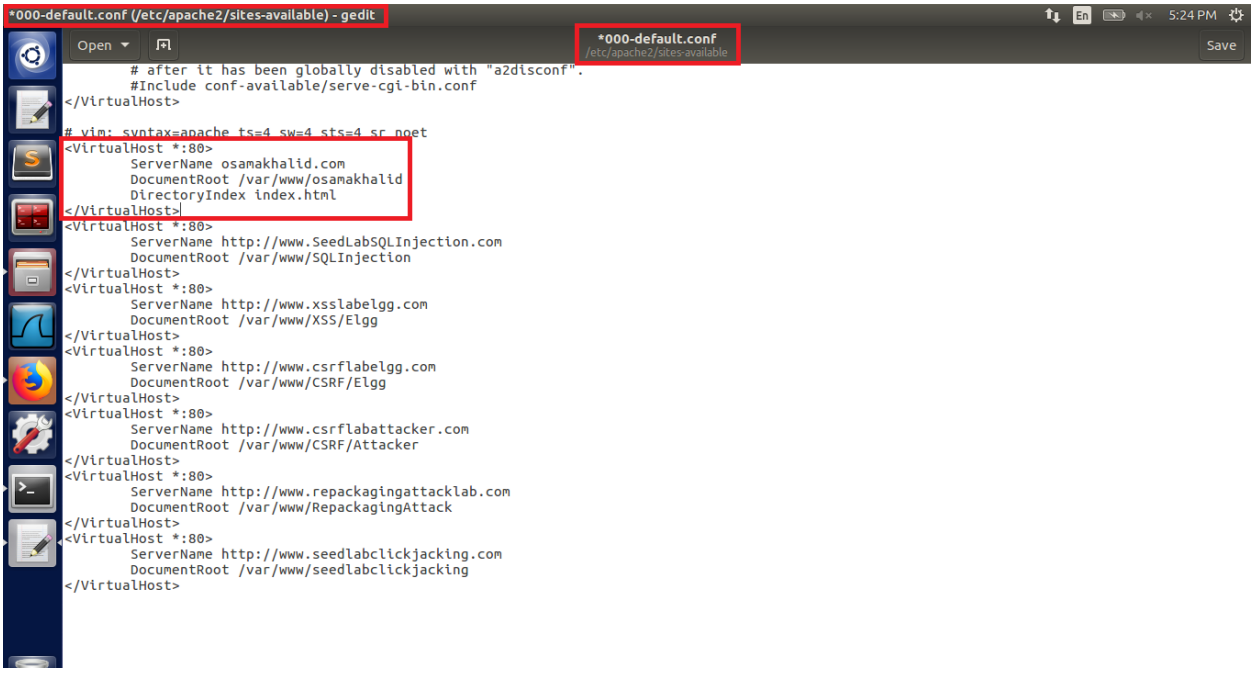

Figure 46: Adding the entry in 000-default.conf file

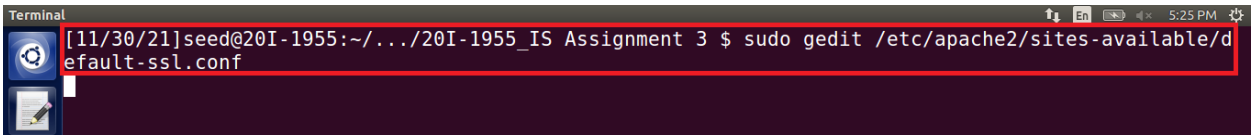

Figure 47: Opening the default-ssl.conf file

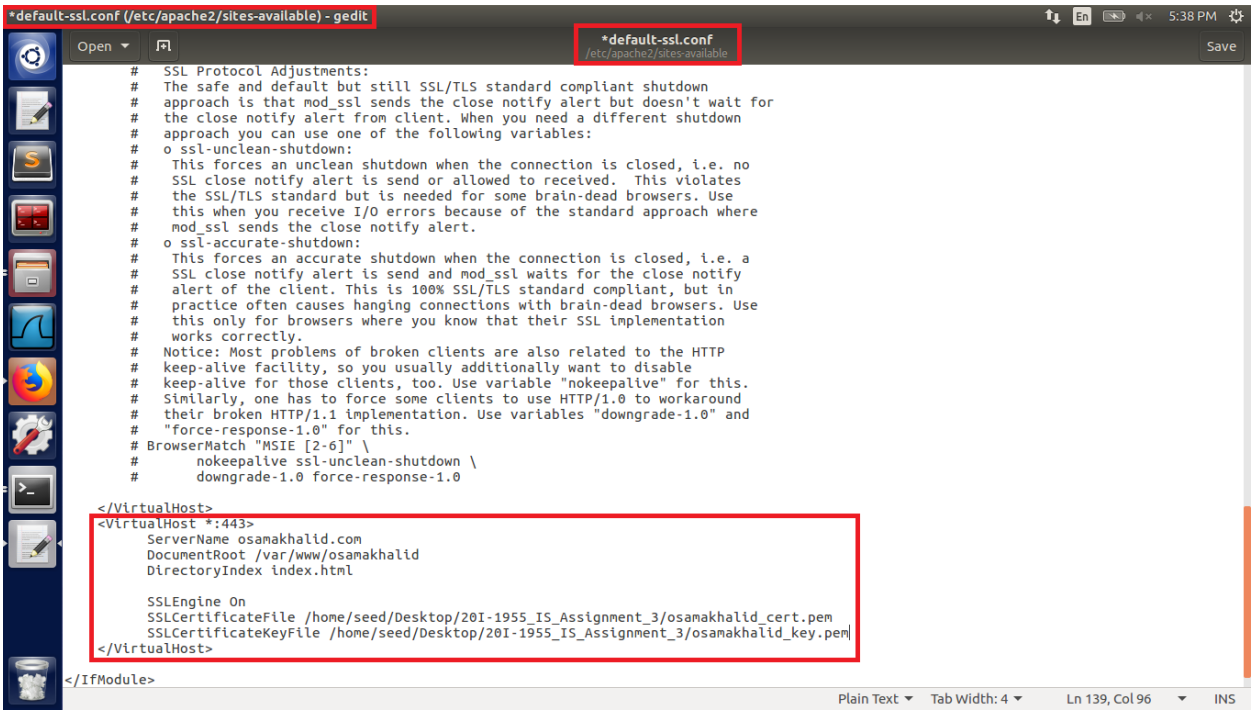

Figure 48: Adding the entry in default-ssl.conf file

After that, I go to the "/var/www/" directory using the command "**cd /var/www**". Then I create the directory in which I will store the website file and then create the simple webpage "index.html" into the website directory "osamakhalid".

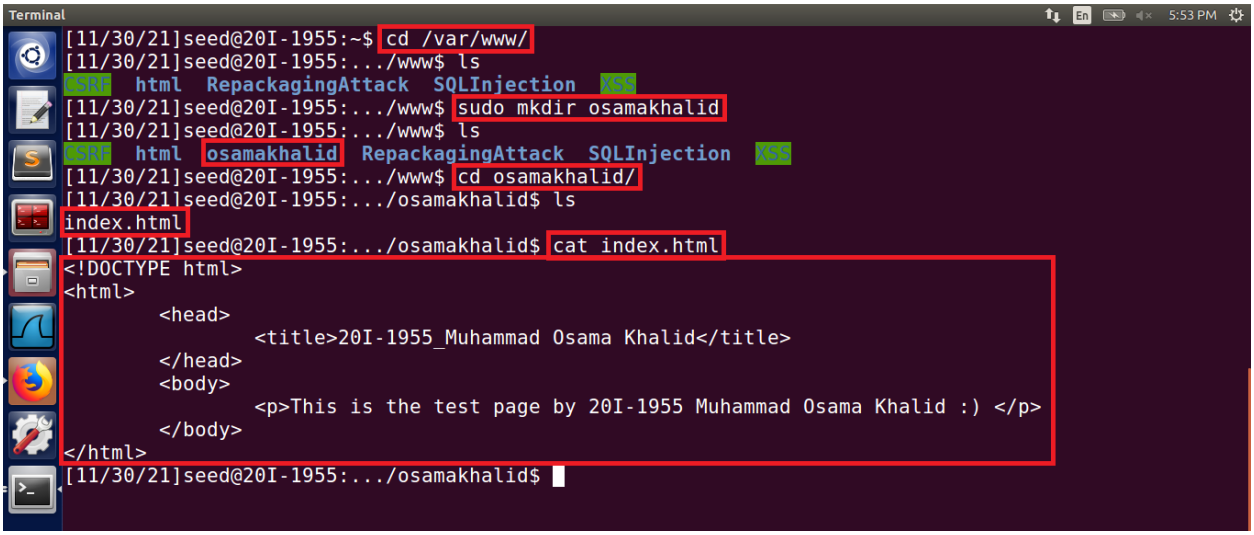

Figure 49: Creating the website files

After that, I run several commands to enable the SSL and then start the Apache server. For that, I first check the Apache configuration file for errors using the command "**sudo apachectl configtest**". Then to enable the SSL module I run the command "**sudo a2enmod ssl**" and then to enable the website I run the command "**sudo a2ensite default-ssl**". In last to restart the Apache server I run the command "**sudo service apache2 restart**".

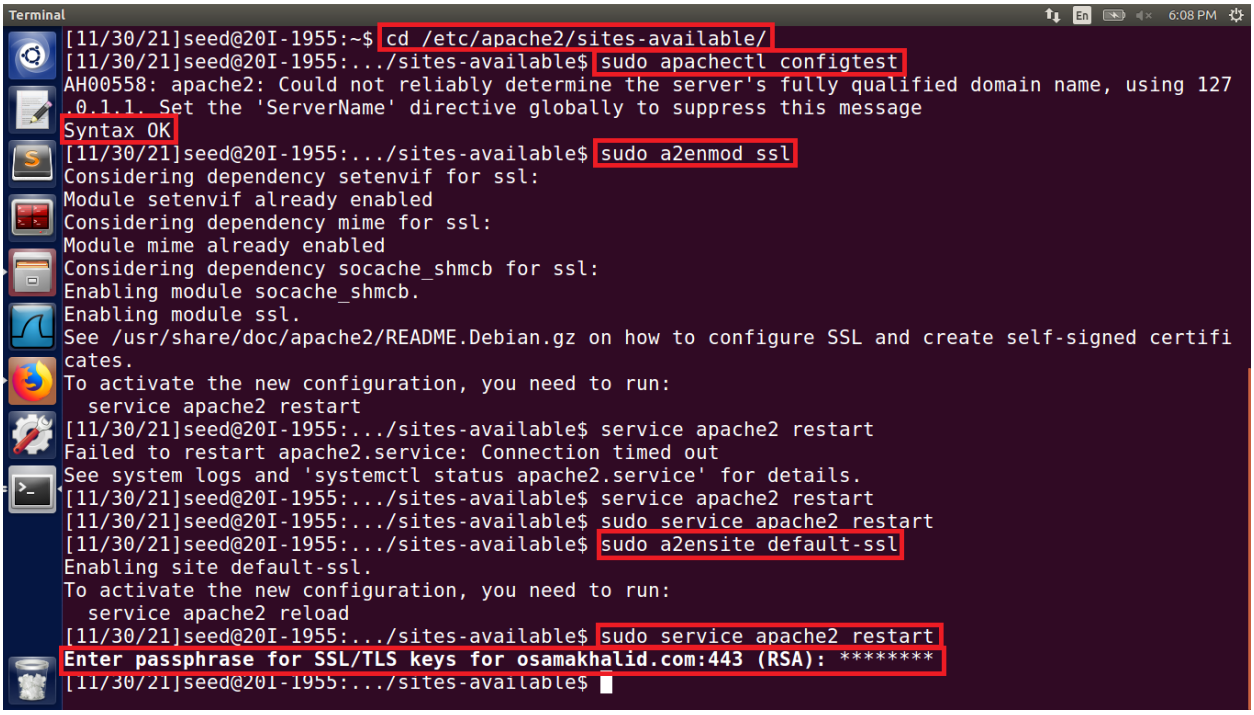

Figure 50: Enabling the SSL for the website

After that when I visit the URL https://osamakhalid.com I found that website loads successfully and displays the webpage that I have created for the website "index.html".

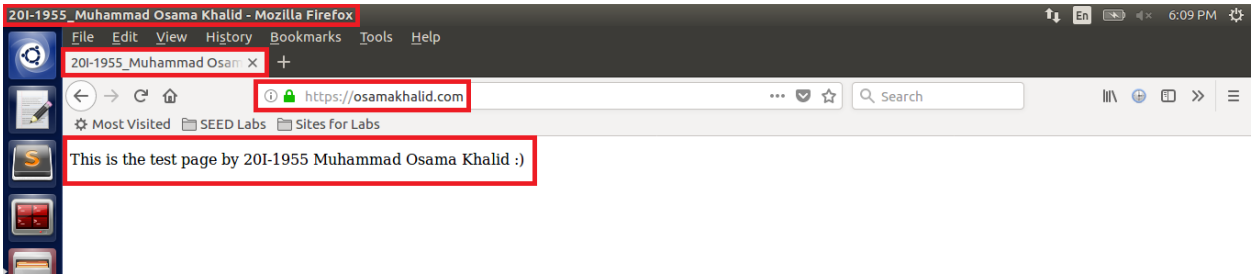

Figure 51: Website loads successfully

Task-5 Launching a Man-In-The-Middle (MITM) Attack

In this task, I have to launch the MITM attack by setting up the malicious website redhat.com that tries to intercept the traffic between the user and osamakhalid.com.

Step-1 Setting up the malicious website

To set up the malicious website I need to add the VirtualHost entry to the two files "000-default.conf" and "default-ssl.conf" located at "/etc/apache2/sites-available".

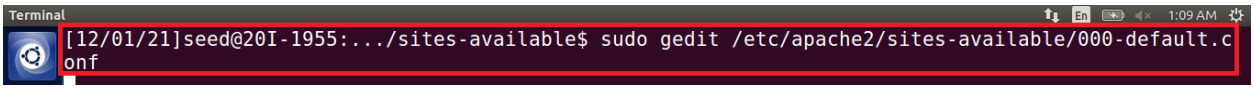

Figure 52: Opening the 000-default.conf file

Figure 53: Adding the entry in 000-default.conf file

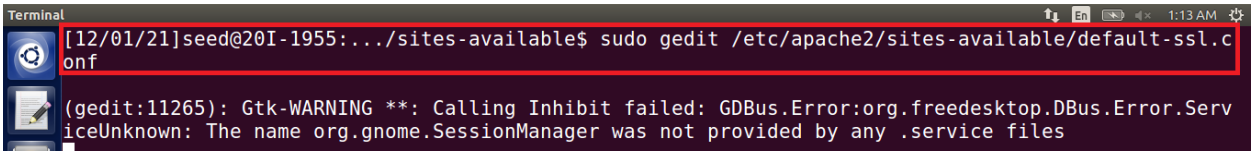

Figure 54: Opening the default-ssl.conf file

Figure 55: Adding the entry in default-ssl.conf file

Step-2 Becoming the man in the middle

Now to become MITM, I have to update the DNS of the victim machine by adding the entry into the "/etc/hosts" file.

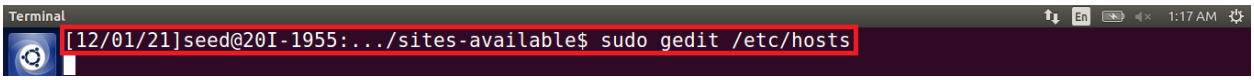

Figure 56: opening the /etc/hosts file

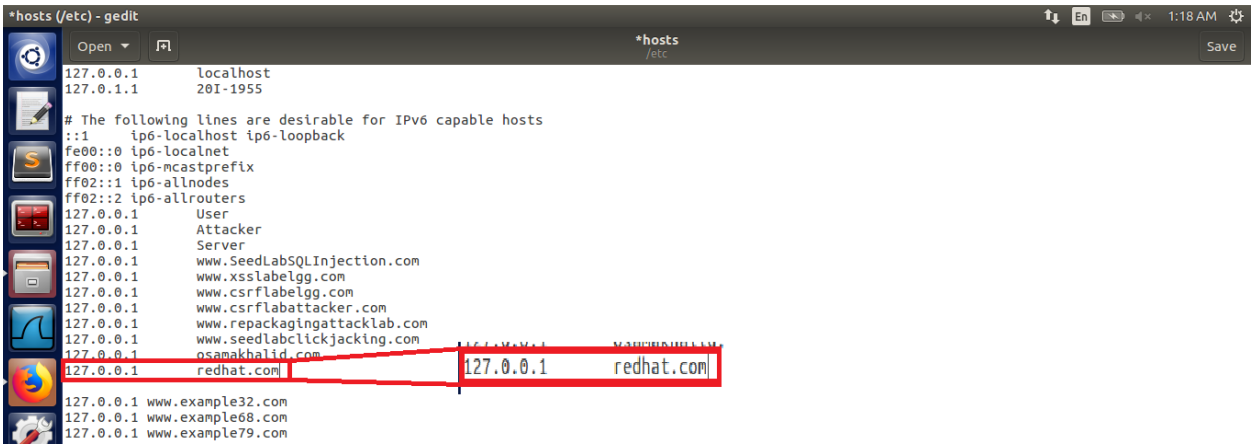

Figure 57: Attacker Adding the malicious entry into /etc/hosts file

Step-3 Browse the target website

Now as we all set up, in this task I have to visit the malicious website https://redhat.com. Before visiting the website, first I have to run several commands to enable the SSL and then restart the Apache server. For that, I first check the Apache configuration file for errors using the command "**sudo apachectl configtest**". Then to enable the SSL module I run the command "**sudo a2enmod ssl**" and then to enable the website I run the command "**sudo a2ensite default-ssl**". In last to restart the Apache server I run the command "**sudo service apache2 restart**". On visiting the website, I found that the browser is throwing the error "Your Connection is Not Secure". Moreover, on clicking on the advance button I found the error that indicates that redhat.com is using the invalid certificate and the certificate used is not valid for redhat.com. After that, on clicking on "Add Exception" and then clicking on the view button I found that the certificate used is only issued to "osamakhalid.com". This proves that with the use of public key infrastructure MITM attack can be mitigated because the browser checks the correctness of the URL requested and the certificate. The website will only load if it has a valid certificate from CA.

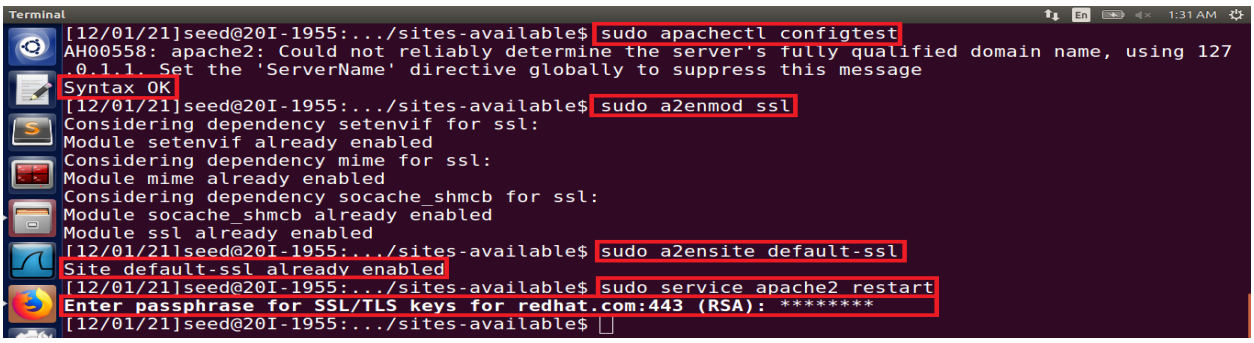

Figure 58: Enabling the SSL for the website

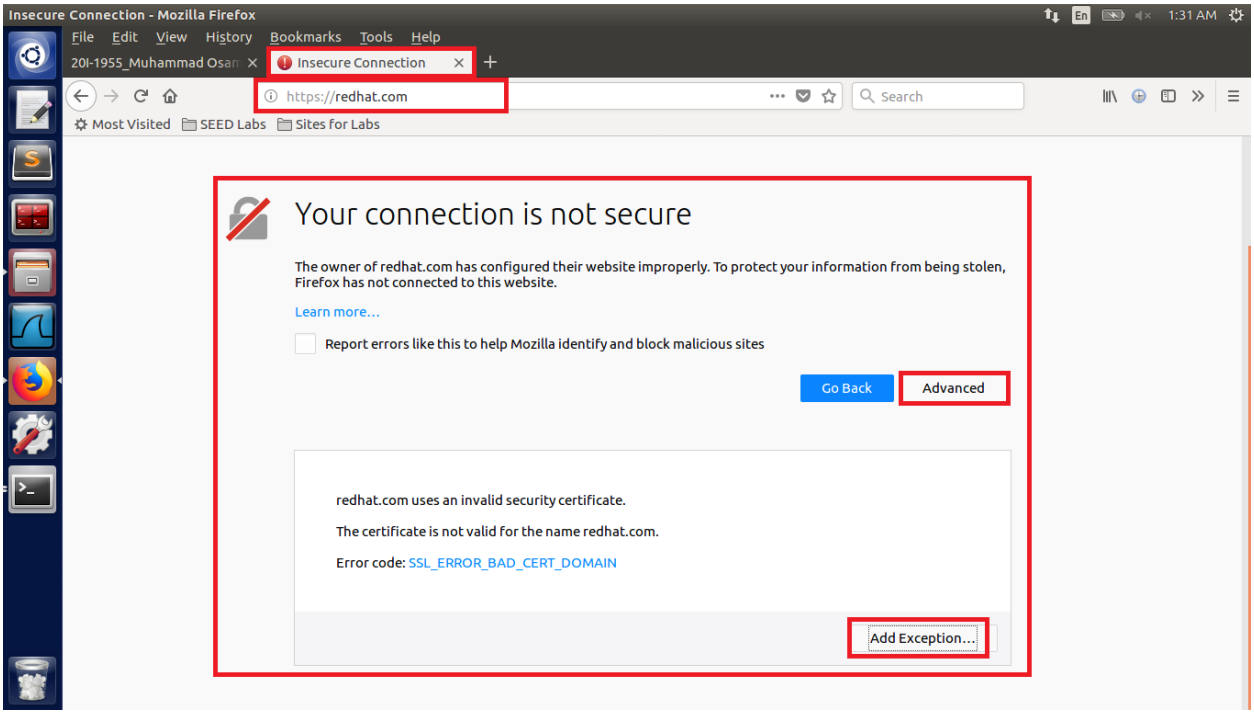

Figure 59:Visiting the website

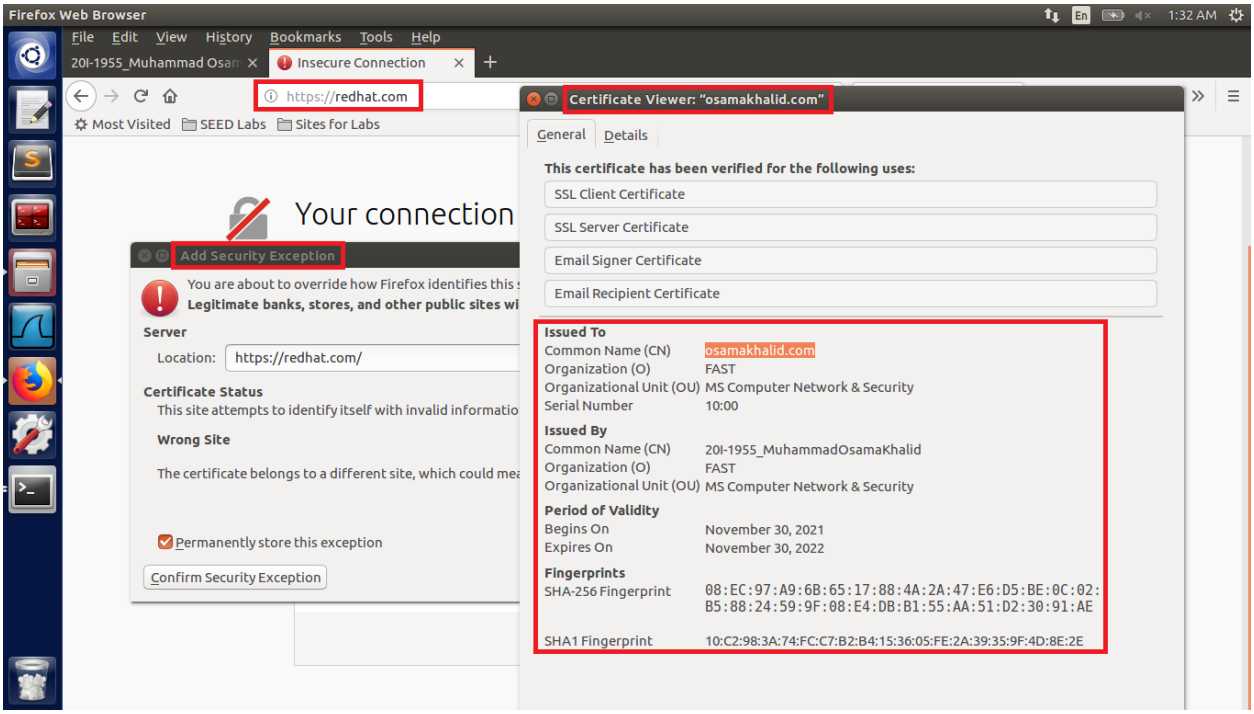

Figure 60: Certificate issued to osamakhalid.com, not redhat.com

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

In this task, I have to launch a MITM attack with a compromised CA. As the attacker stoles the CA's private kay he/she can generate any certificate using this CA's private key. As the attacker gets hands-on CA's private key, he first creates the public and private keys for redhat.com. To create the keys, I use the command "**openssl genrsa -aes128 -out server_redhat.key 1024**".

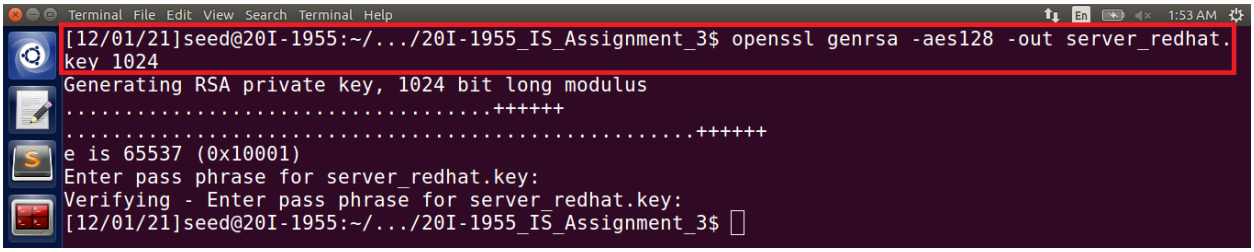

Figure 61: Attacker Generating public-private key pairs from CA's private key

After generating the public and private key pairs, the attacker's next task is to generate Certificate Signing Request (CSR) that contains a public key generated by the attacker for redhat.com and forward this request to CA. On receiving the request CA will generate the certificate for the key. Now to generate the CSR I use the command "**openssl req -new -key server_redhat.key -out server_redhat.csr -config openssl.cnf**".

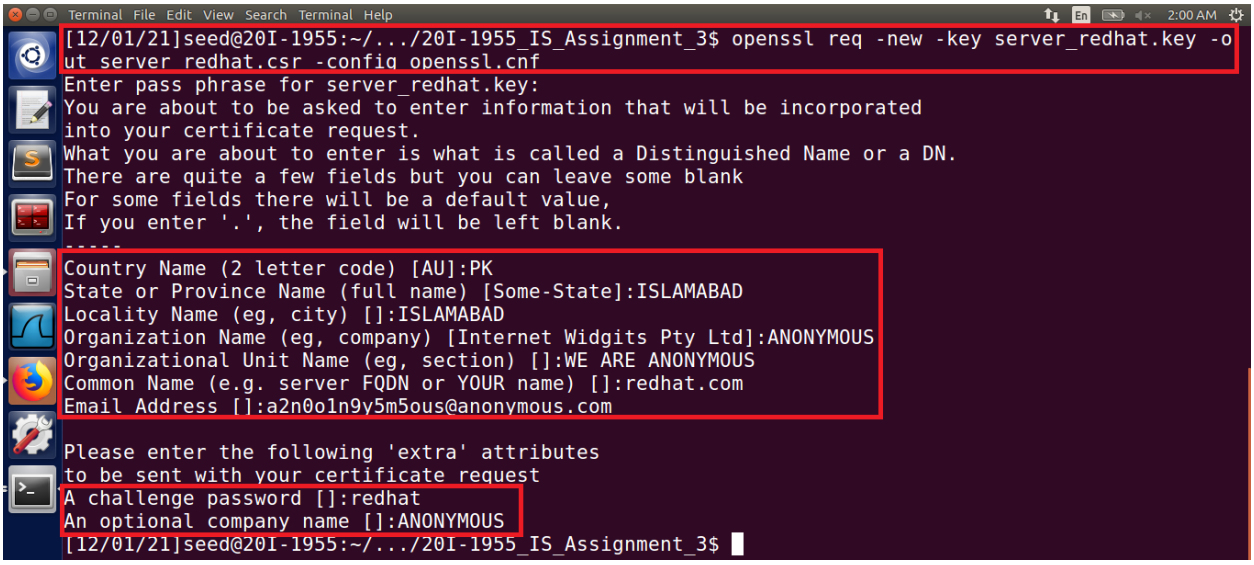

Figure 62: Attacker generating CSR request

Now as the attacker has CA's private key and the certificate so the attacker can generate the certificate for himself from the CSR without needing CA and the attacker can do this by running the command "**openssl ca -in server_redhat.csr -out server_redhat.crt -cert ca.crt -keyfile ca.key -config openssl.cnf**". Now as the certificate is signed by CA that is trusted by the browser, therefore browser will also confirm the validity of this certificate.

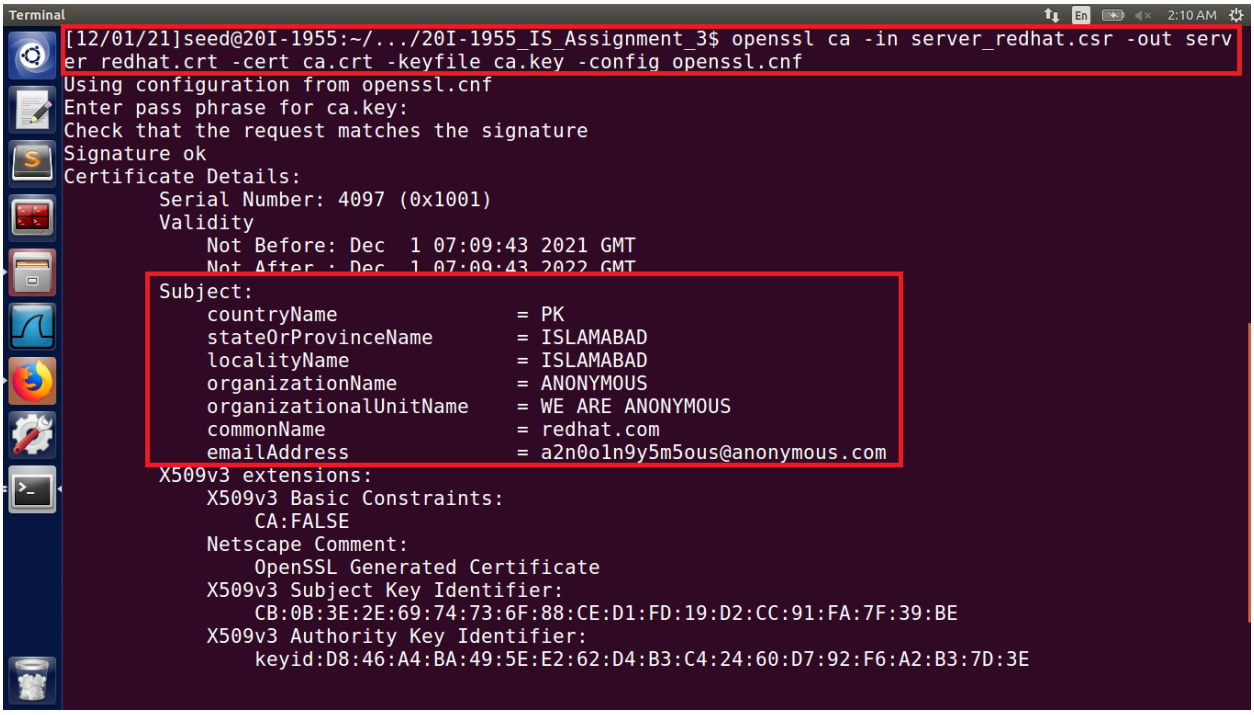

Figure 63: Attacker generating certificate from CSR using CA's key and certificate

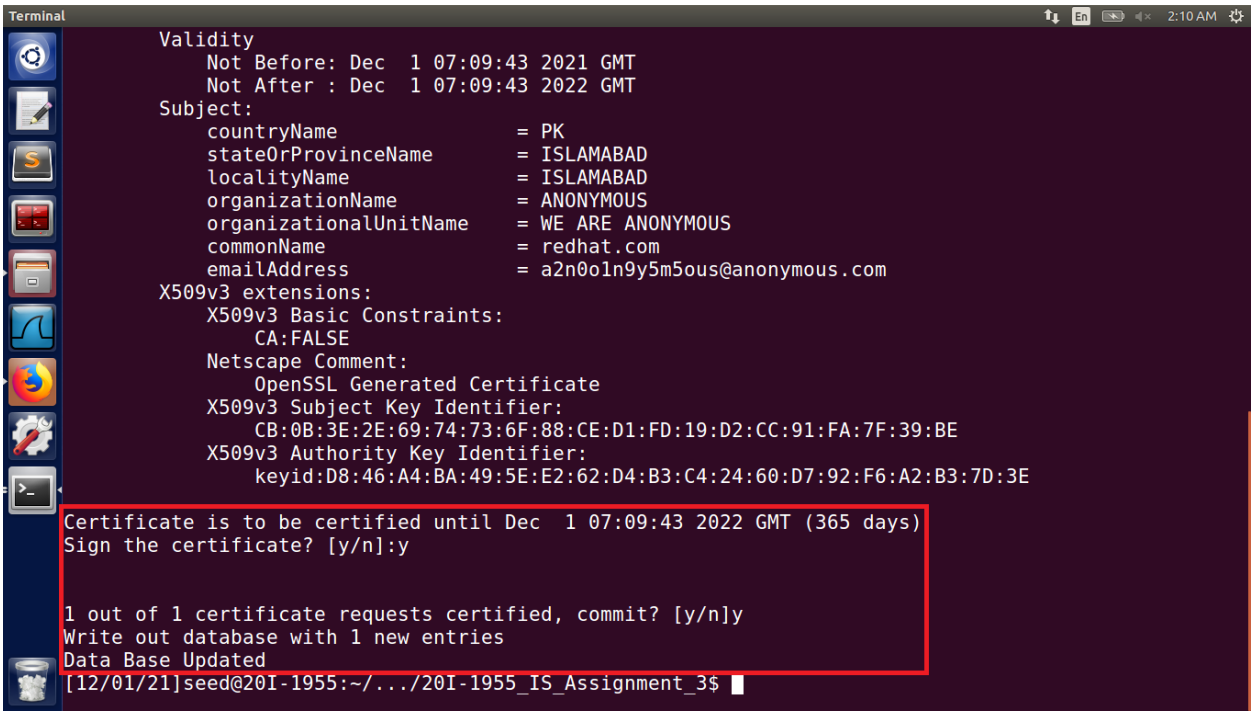

Figure 64: Attacker generating certificate from CSR using CA's key and certificate

After that attacker needs to combine the key (server_redhat.key) and certificate (server_redhat.crt) into one file used by the Apache server and then use this file in the website configuration. To combine the key and certificate into one file I use the following commands "**cp server_redhat.crt osamakhalid_cert.pem**" and "**cp server_redhat.key osamakhalid_key.pem**".

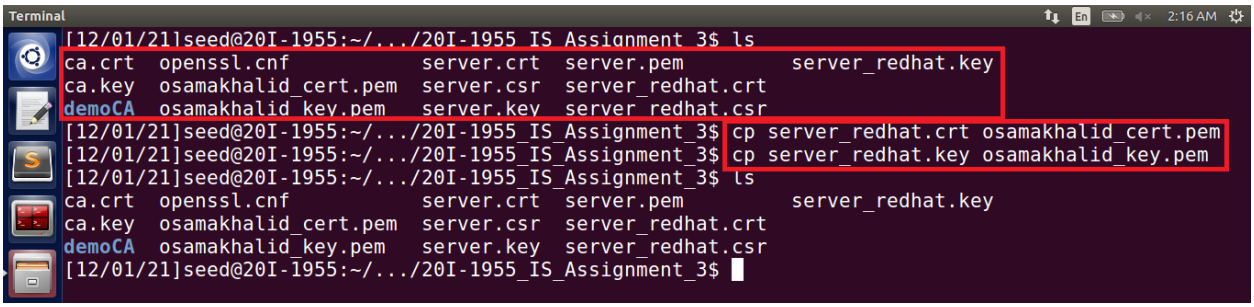

Figure 65: Attacker Combine the certificate and key into one file

After that to save the changes, the attacker restarts the Apache server using the command "**sudo service apache2 restart**".

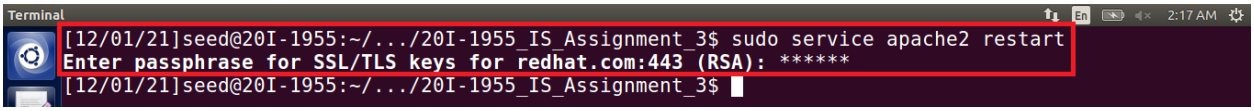

Figure 66: Attacker Restart the Apache Server

Now on visiting the target website of the attacker browser loads the website successfully without throwing any error because the certificate used is valid and signed by the CA. From here we can conclude that if CA gets compromised then anyone can generate the certificate for himself/herself and mimic any other website they want.

Figure 67:Target website redhat.com loads successfully and display the contents of osamakhalid.com