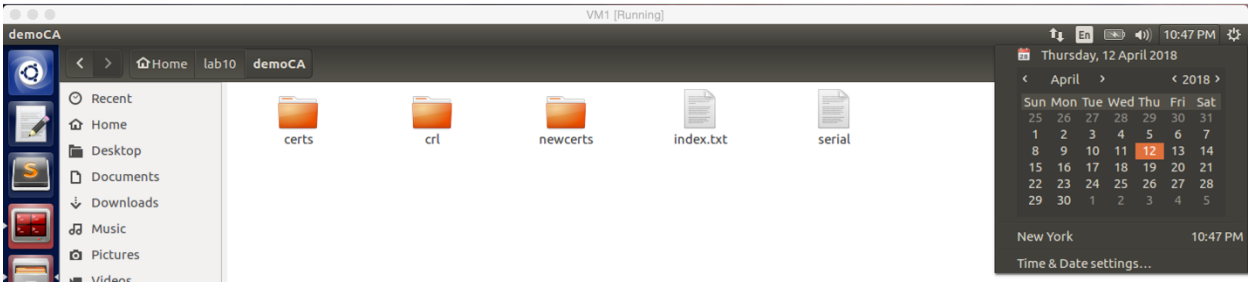
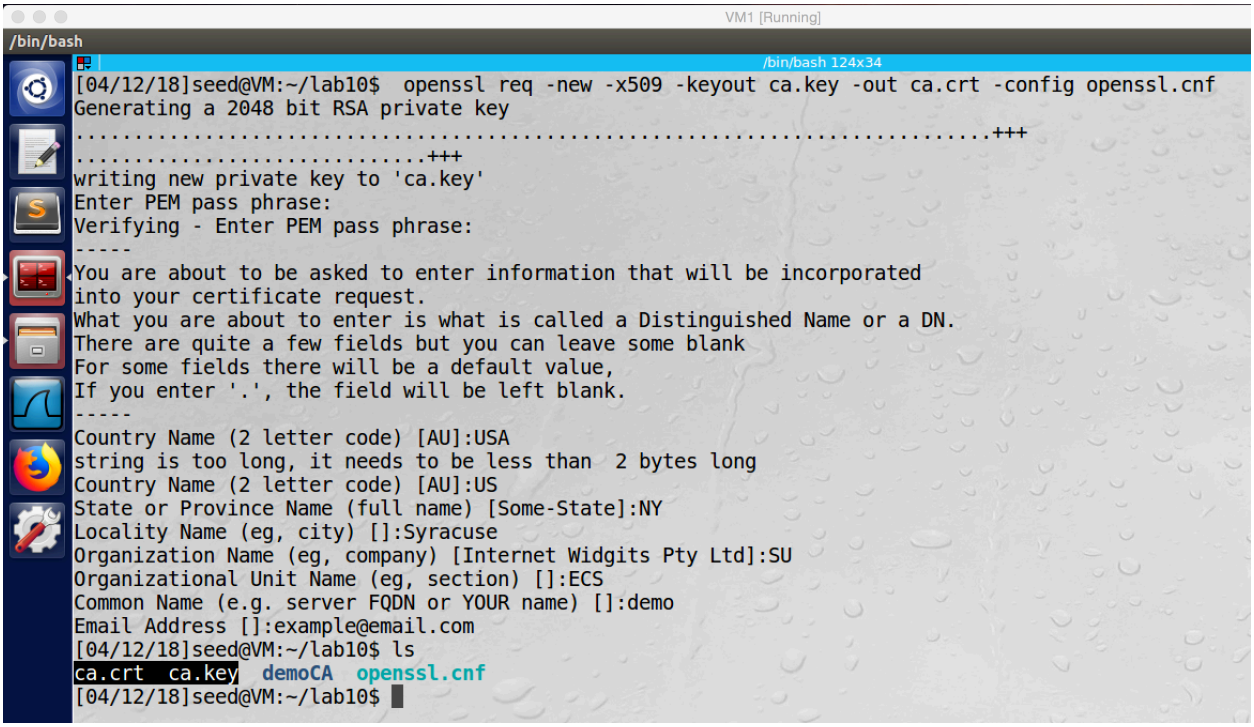
CSE644 Lab10
Yishi Lu
4/17/2018

Task 1: Becoming a Certificate Authority (CA)



screenshot1. I copied the openssl.conf into my directory lab10, and I created several sub-directories.



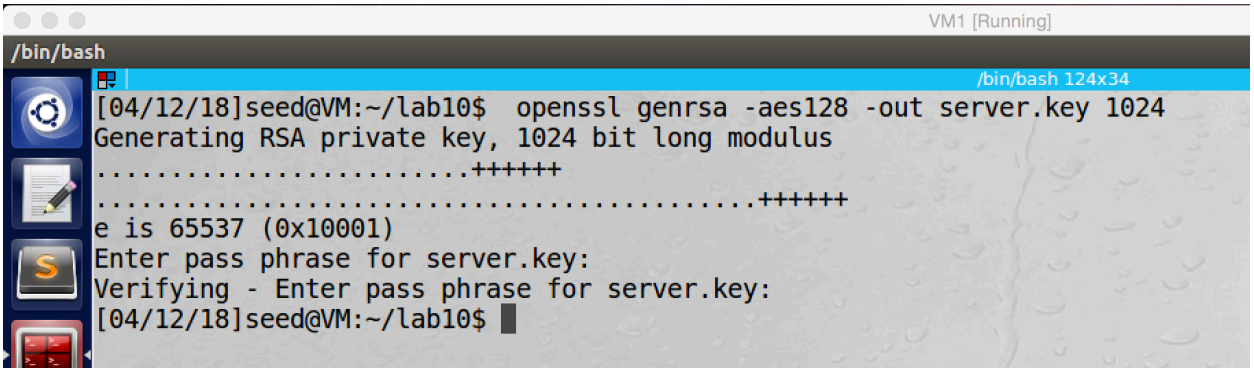
screenshot2. I use openssl command to generate self-signed certificate.

Observation and Explanation:

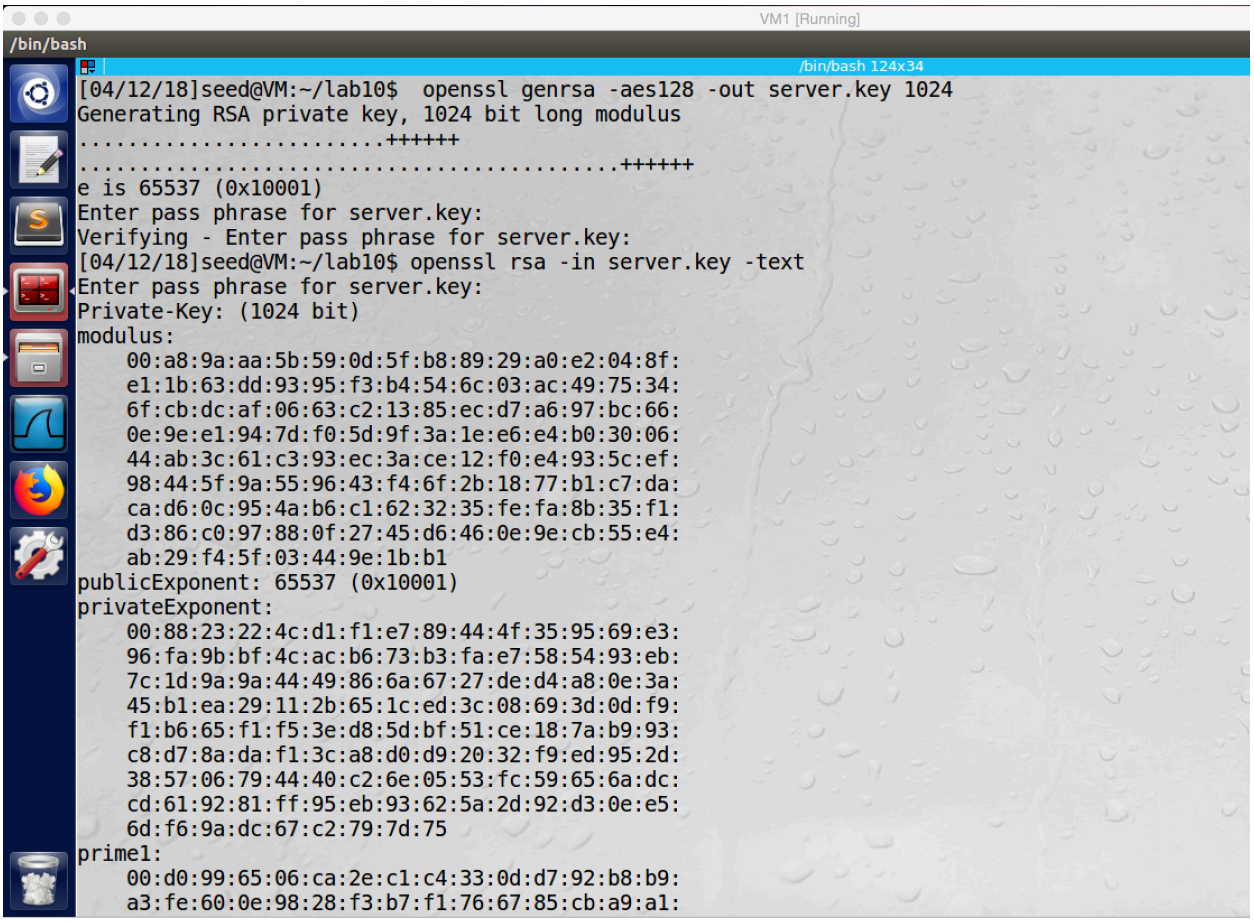
In this task, we are going to generate self-signed certificate. First, we copy file openssl.conf and put it into directory lab10. And then we create several sub-directories under lab10 (screenshot1). After we generated all needed configuration directories and files, we can use openssl command to generate self-signed key. As the screenshot2 shows, we successfully generated a self-signed certificate. There two new files are generated by the openssl command.

ca.key and ca.crt, they contain the CA's private key and public key certificate respectively. For this CA, we just call it demoCA.

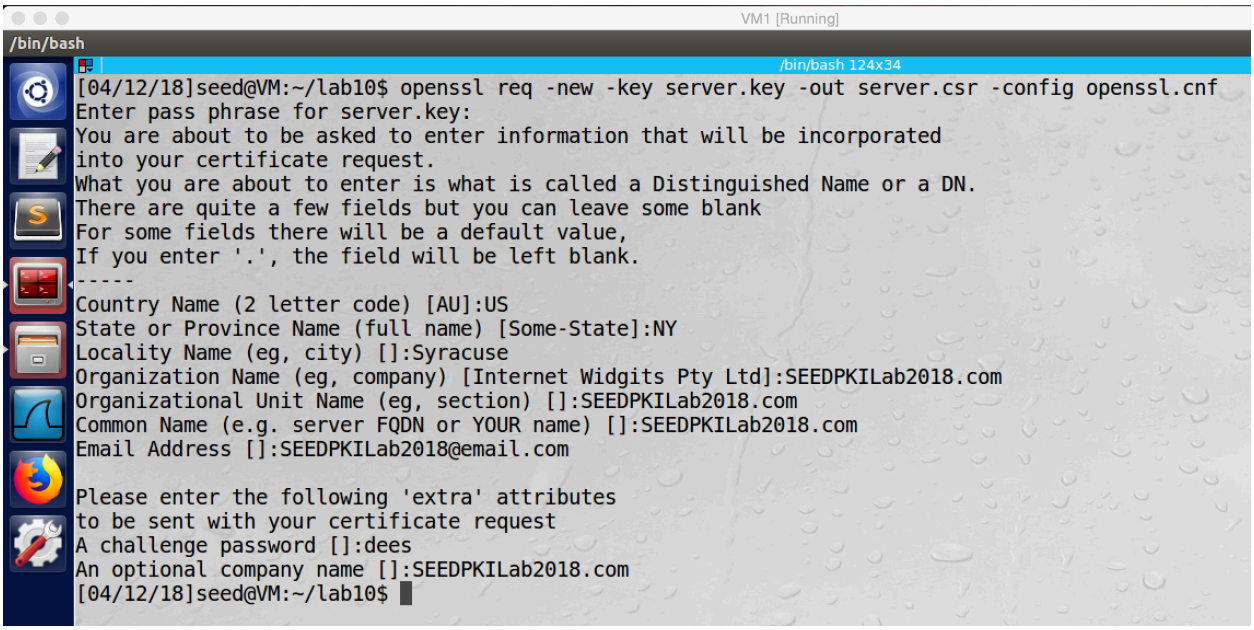
Task 2: Creating a Certificate for SEEDPKILab1028.com



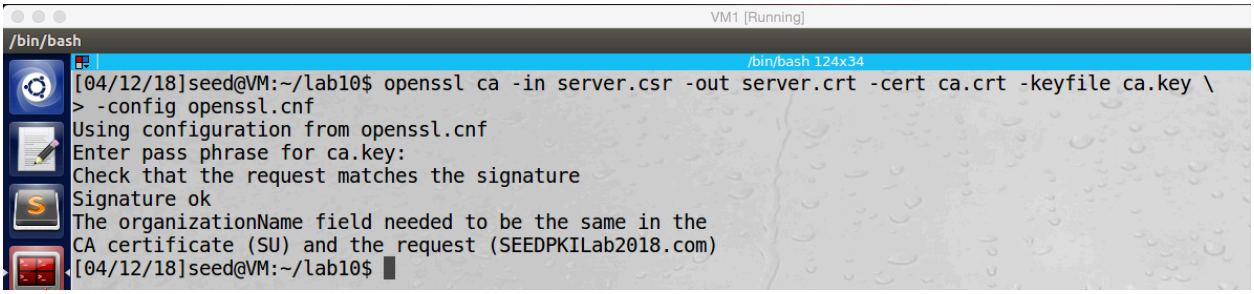
screenshot1. Generating public/private key for SEEDPKILab2018.com, and the key is stored in the file server.key



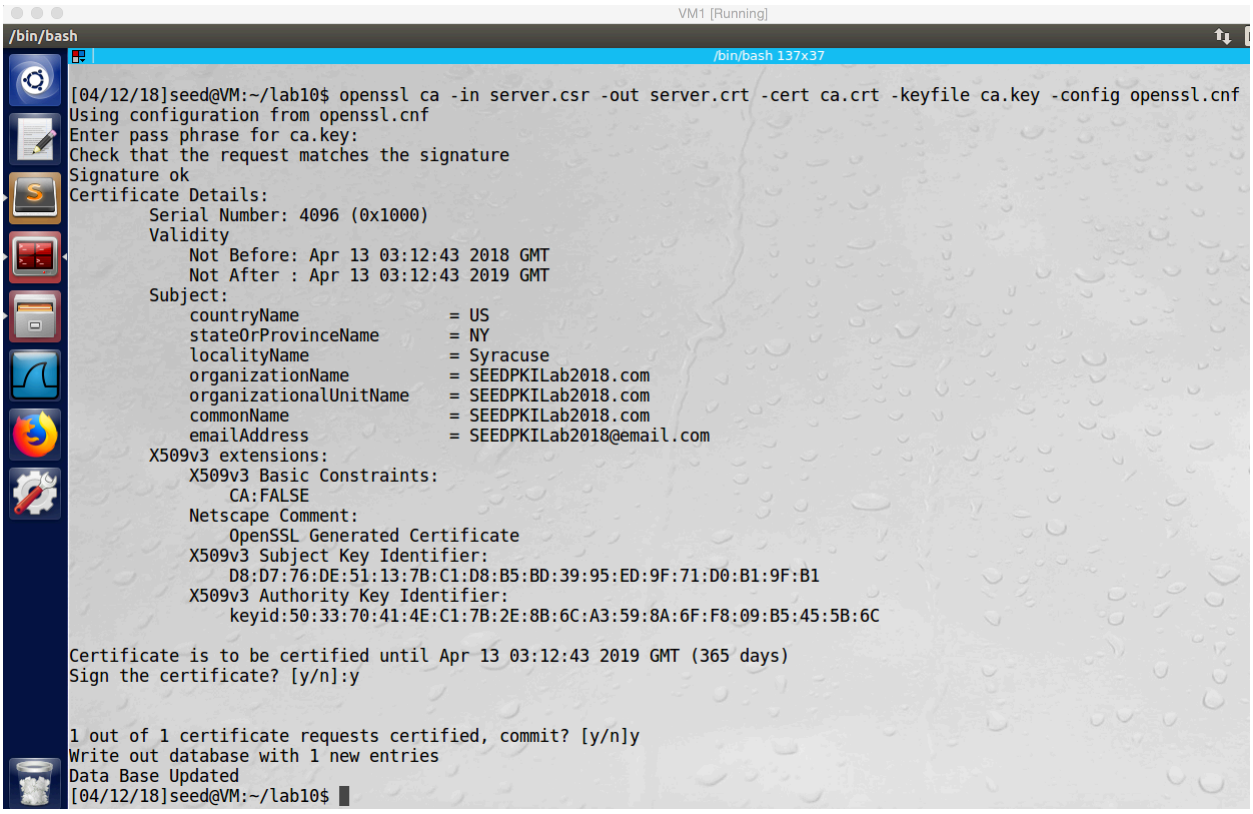
screenshot2. We can open the key by openssl command



screesshot3. Generating a certificate signing request (CSR) for SEEDPKILab2018.com



screenshot4. Fail to generating certificate because the request name and the CA name do not match.

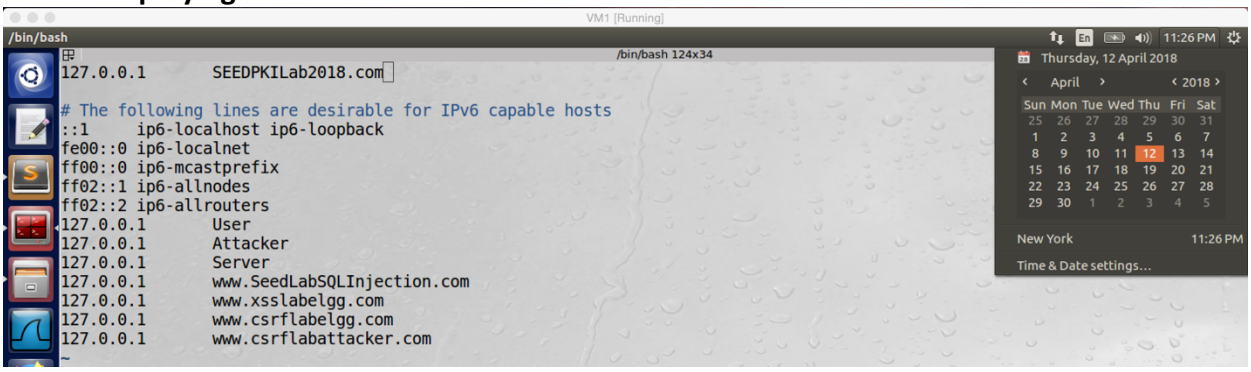


screenshot5. After we change the policy_match to policy_anything in the openssl.conf file, we run the command again. This time we successfully generate a certificate for SEEDPKILab2018.com

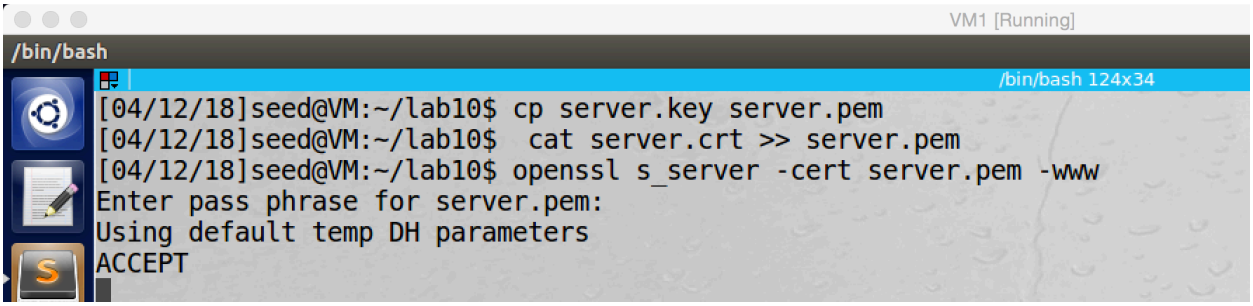
Observation and Explanation:

In this task, we use the demoCA to generate a certificate to a company, this company is called SEEDPKILab2018.com. Before the company can make CSR, it must have public/private key. So we run openssl command to generate the public/private key (screenshot1 and 2). And then the company can make CSR to the demoCA (screenshot3). After the demoCA received CSR from SEEDPKILab2018.com, it should verify the information which provided by SEEDPKILab2018.com. In our case, we just generate a certificate for SEEDPKILab2018.com. As the screenshot4 shows, in the first generating, because the CA and the company name do not match, we fail to generate a certificate. So we modify something in the openssl.conf file, we change policy_match to policy_anything. As the screenshot5 shows, after we make the modification, we successfully generate certificate for SEEDPKILab2018.com.

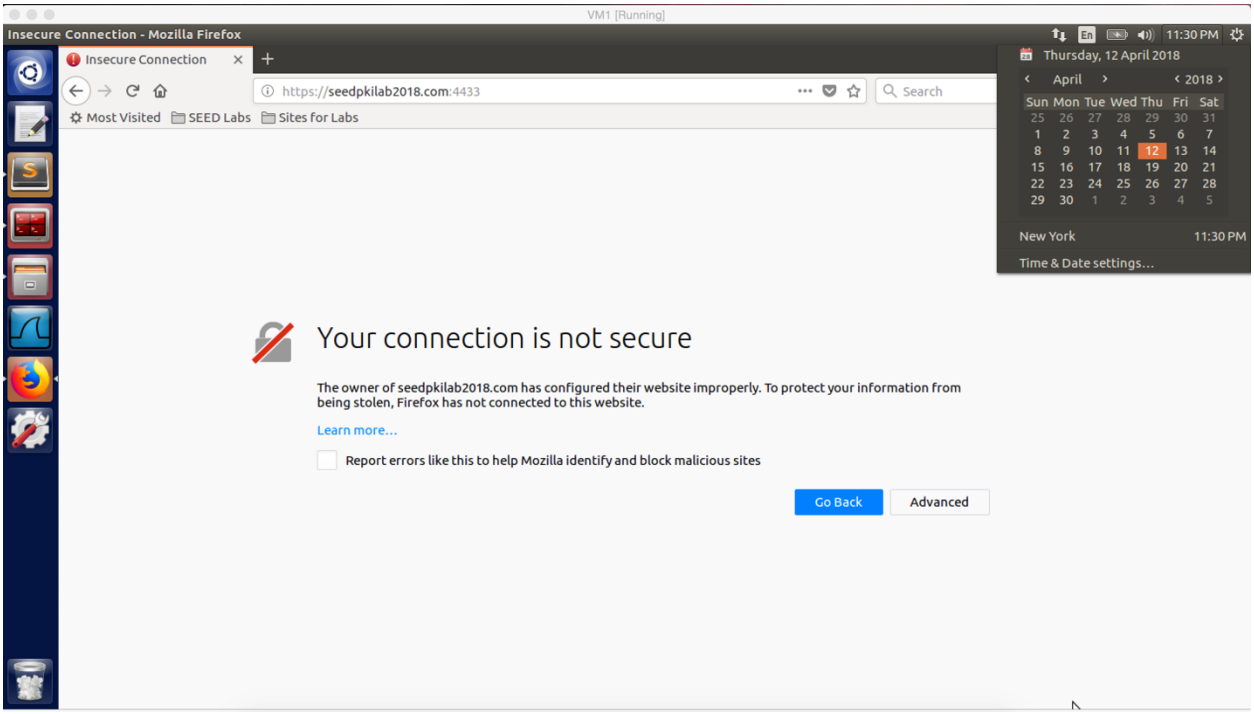
Task 3: Deploying Certificate in an HTTPS Web Server



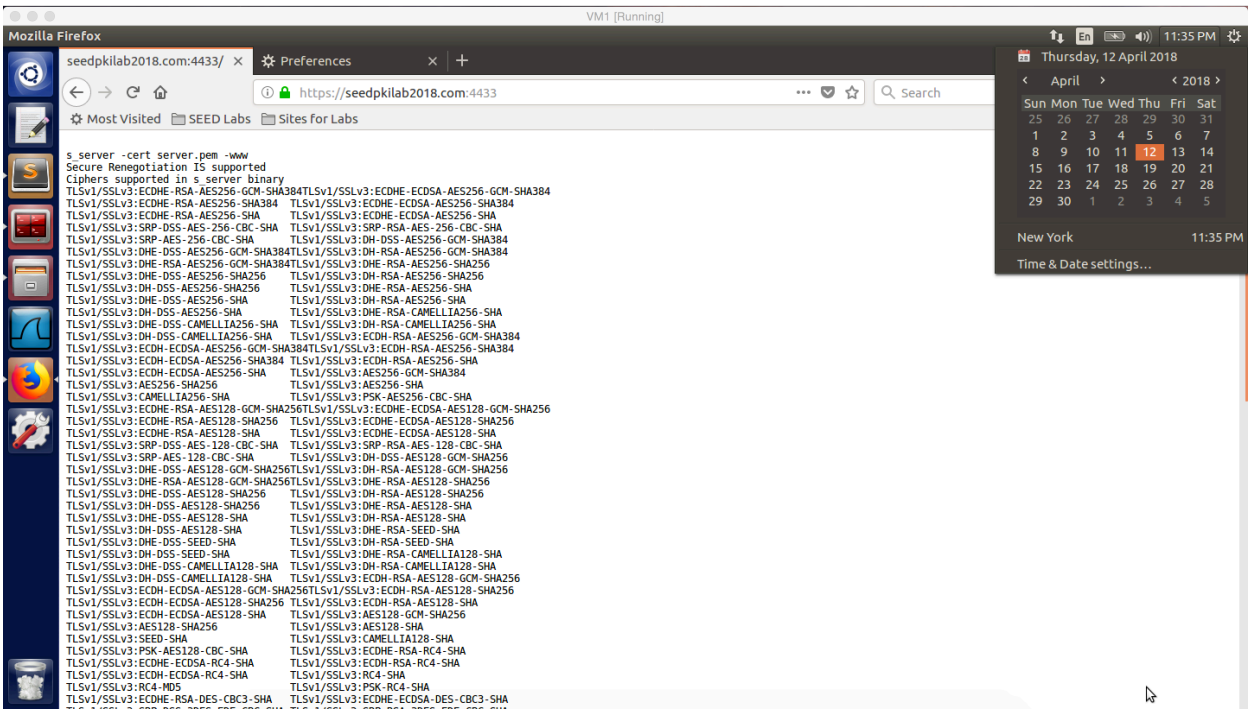
screenshot1. We map hostname SEEDPKILab2018.com to 127.0.0.1 (localhost)



screenshot2. Launching a webserver



screenshot3. After the web server is launched, we try to input https://seedpkilab2018.com:4433 in the browser, but the browser shows message “Your connection is not secure”



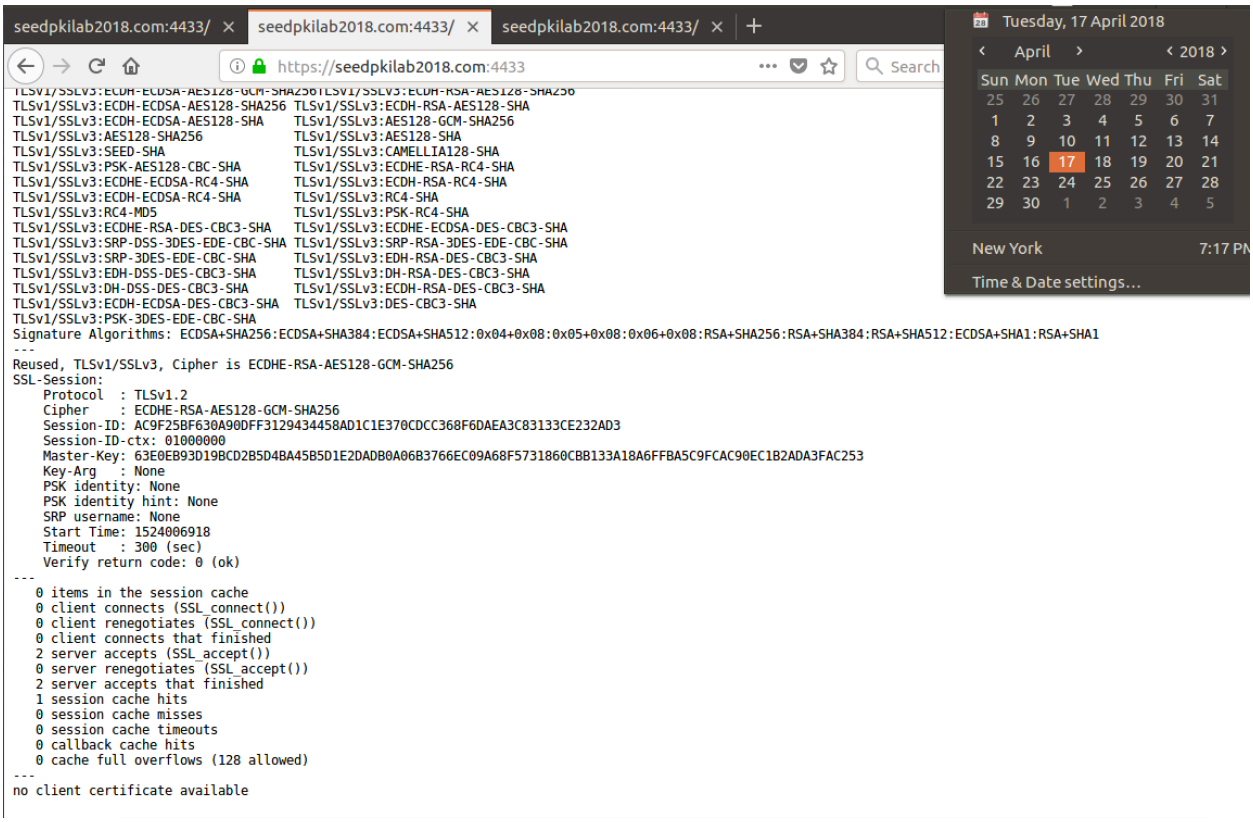
screenshot4. After we imports the certificate of demoCA into the trust list of the browser, we can visit https://seedpkilab2018.com:4433.

Observation and Explanation:

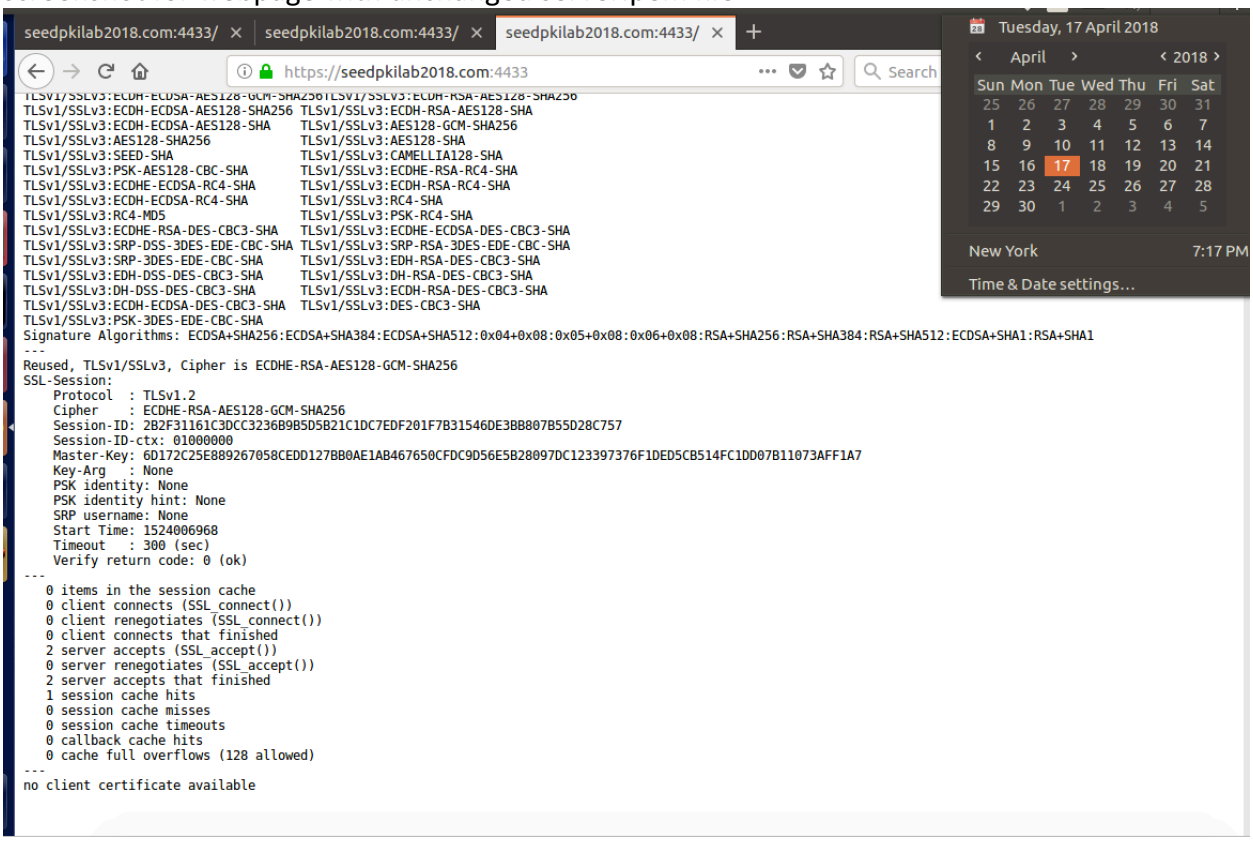
In this task, we are going to deploy the certificate on a web server. First, we need to configure the DNS, so we add the seedpkilab2018.com into the /etc/hosts file, and this address maps to 127.0.0.1 which is the localhost (screenshot1). Afterwards, we need to launch a web server, openssl has command which allow us to launch a web server. Of course, the certificate which we created in task2 will be attached to the command. So the certificate is deployed on the web server (screenshot2). And then we can open a browser and input the https://seedpkilab2018.com:4433. However, the browser warns us that this website is not secure (screenshot3). In fact, the browser maintains a list for CA which it trusts; but demoCA is not in its trust list, so the certificate signed by demoCA is not trusted by the browser as well. Therefore, we need to add demoCA's certificate into the browser trust list. After we did that, we refresh the web page; at this time, we can connect to https://seedpkilab2018.com:4433. The result is shown in screenshot4.

Question1 on Step 4:

After I modify one byte in the server.pem file, I restart the server, but I do not find any difference. The only difference is the session-id, the master-key and start time fields. But this two fields will be change when I open a new page of seedpkilab2018.com. Moreover, I also try to modify private key and certificate part; after changing, the web server cannot be launched.



screenshot for webpage with unchanged server.pem file



screenshot for webpage with changed server.pem file

```
[04/17/18]seed@VM:~/lab10$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load server certificate private key file
3082303168:error:0D07209B:asn1 encoding routines:ASN1_get_object:too long:asn1_lib.c:147:
3082303168:error:0D068066:asn1 encoding routines:ASN1_CHECK_TLEN:bad object header:tasn_dec.c:1205:
3082303168:error:0D06C03A:asn1 encoding routines:ASN1_D2I_EX_PRIMITIVE:nested asn1 error:tasn_dec.c:785:
3082303168:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:717
:Field=iqmp, Type=RSA
3082303168:error:04093004:rsa routines:OLD_RSA_PRIV_DECODE:RSA lib:rsa_ameth.c:119:
3082303168:error:0D0680A8:asn1 encoding routines:ASN1_CHECK_TLEN:wrong tag:tasn_dec.c:1217:
3082303168:error:0D07803A:asn1 encoding routines:ASN1_ITEM_EX_D2I:nested asn1 error:tasn_dec.c:386:Type=X
509_ALGOR
3082303168:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:717
:Field=pkeyalg, Type=PKCS8_PRIV_KEY_INFO
3082303168:error:0907B00D:PEM routines:PEM_READ_BIO_PRIVATEKEY:ASN1 lib:pem_pkey.c:141:
[04/17/18]seed@VM:~/lab10$
```

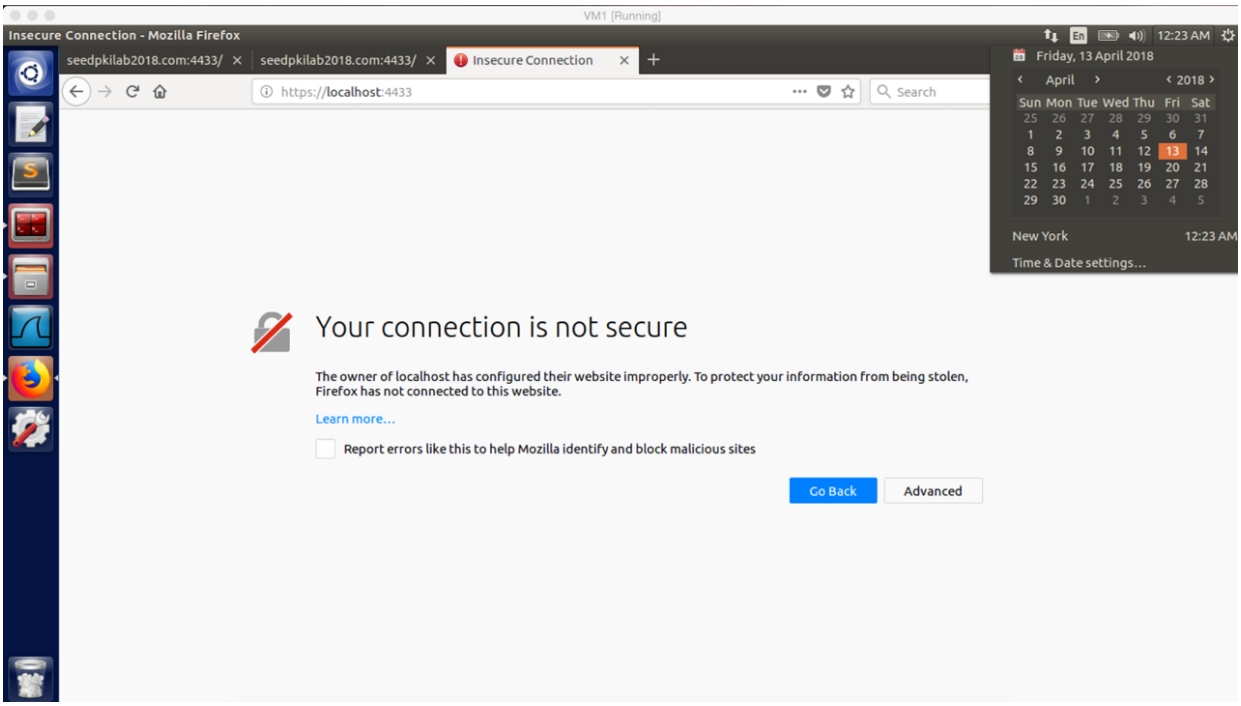
After changed the private key section, the web server cannot be launched

```
/bin/bash 105x34
[04/17/18]seed@VM:~/lab10$ openssl s_server -cert server.pem -www
Enter pass phrase for server.pem:
unable to load certificate
3082323648:error:0D0C40D8:asn1 encoding routines:c2i_ASN1_OBJECT:invalid object encoding:a_object.c:283:
3082323648:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:717
:Field=object, Type=X509_NAME_ENTRY
3082323648:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:689
:
3082323648:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:689
:
3082323648:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:717
:Field=issuer, Type=X509_CINF
3082323648:error:0D08303A:asn1 encoding routines:ASN1_TEMPLATE_NOEXP_D2I:nested asn1 error:tasn_dec.c:717
:Field=cert_info, Type=X509
3082323648:error:0906700D:PEM routines:PEM_ASN1_read_bio:ASN1 lib:pem_oth.c:83:
[04/17/18]seed@VM:~/lab10$
```

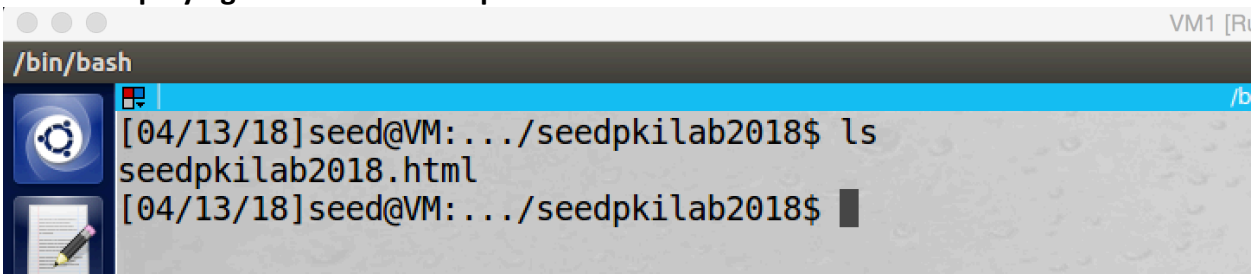
After changed the certificate section, the webserver cannot be launched

Question2 on Step 4

After I input localhost:4433 in the browser, the browser warns me again that the connection is not secure. This is because the URL which I typed in the browser does not match to the domain name on the certificate. When the browse receive certificate, it needs to verify two things. First, the certificate. In our case, the certificate is valid. Second, the URL typed by the user need to match the domain name on the certificate; in our case, they are different. So the browser does not trust this connection and refuse to connect.



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



screenshot1. I first create a html file which called seedpkilab2018.html, and this file is under /var/www/seedpkilab2018 directory.

screenshot2. Afterward, I open the default-ssl.conf file and added the above virtualHost entry into this file.

screenshot3. And then we restart the apache server

screenshot4. We input the seedpkilab2018.com in the browser, there is the page which I created before. We can see the page is stamped by my name.

Observation and Explanation:

In this task, we are going to deploy the certificate which we created in the task2 in an apache based HTTPS website. There are several steps.

First step, we need to create a web page for the seedpkilab2018. We create a folder which called seedpkilab2018 under /var/www directory. In this folder, we create a file which called seedpkilab2018.html. The content of the file is shown in screenshot1.

Second step, we need to setup the configuration file for the apache server. We go to the directory /etc/apache2/sites-available, and we open file default-ssl.conf. Afterwards, we add a VirtualHost entry in this file, the contents are shown in screenshot2. It includes, server name, directory of website file, and directory of key and certificate.

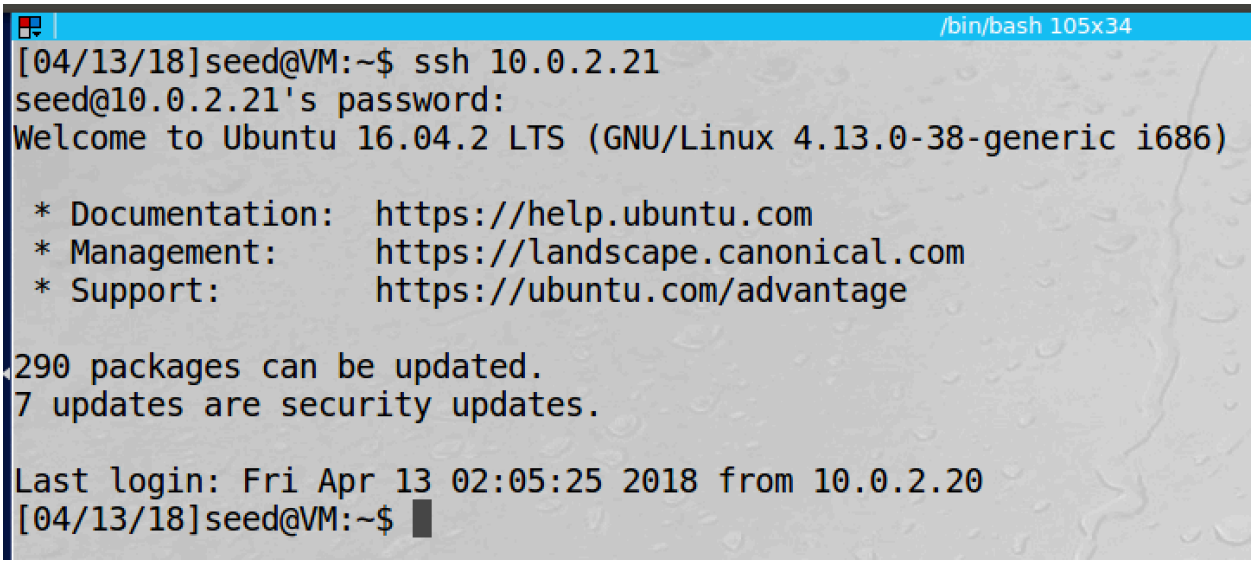
Last step, now we can restart the apache server. As screesshot3 shows, we follow the command on the lab description to restart the apache server.

Finally, we can open a browser, and we input seedpkilab2018.com, the page is exactly we created before which is stamped by my name (screenshot4).

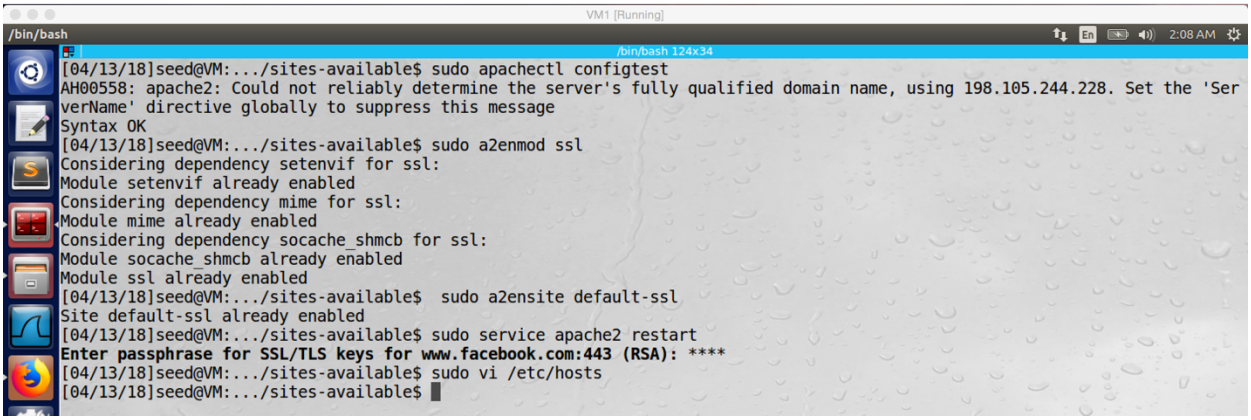
Task 5: Launching a Man-In-The-Middle Attack

In this task, I use two VMs. VM1, IP 10.0.2.20, is the attacker machine which will run fake facebook.com. VM2, IP 10.0.2.21, is the victim machine.

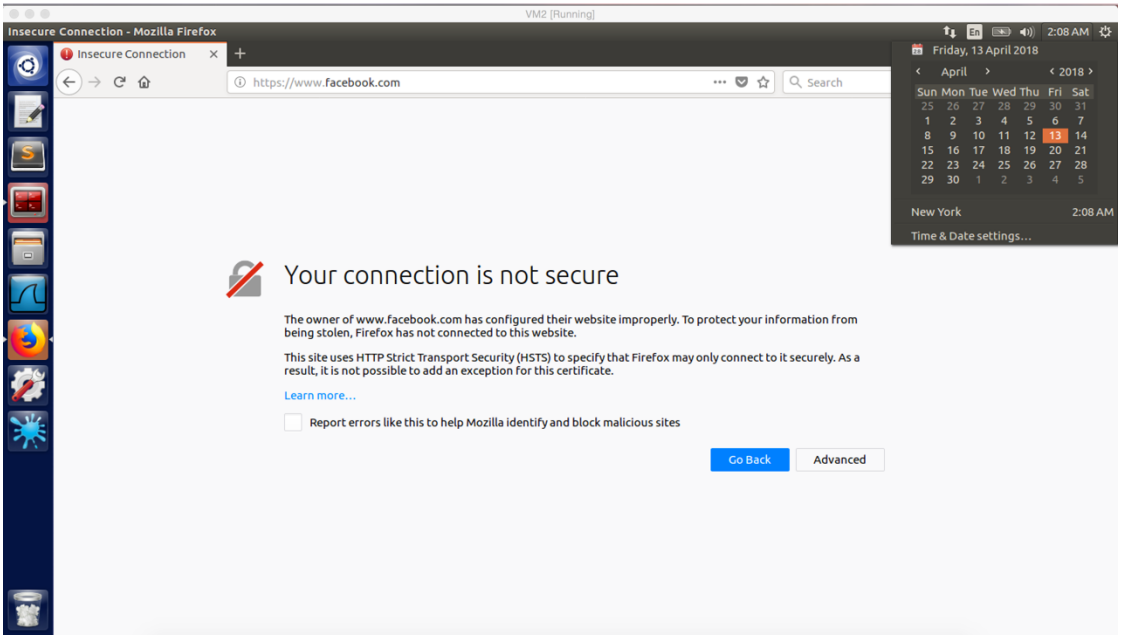
screenshot1. Adding new VirtualHost on attacker machine, but this time the server name we use facebook.com



screenshot2. We use SSH to connect to victim machine VM2, and we open the hosts file to map www.facebook.com to the attacker IP 10.0.2.20



screenshot3. Restarting the apache server on attacker machine



screenshot4. When we type www.facebook.com in the browser on victim machine VM2, the browser warns that the connection is not secure (the certificate of demoCA is imported in browser of VM2).

Observation and Explanation:

In this task, we are going to perform MITM attack, VM1 is the attacker machine (10.0.2.20), VM2 is the victim machine (10.0.2.21). First, we need to setup a malicious website. In my case, I decide to create a fake fackebook.com. We just follow the steps in the task4 to setup a website (screenshot1). After we created the fake facebook, we need to redirect victim machine to malicious website on VM1. For this purpose, we connect to VM2 by SSH, and we can change the local hosts file of victim machine. In my case, I map www.facebook.com to 10.0.2.20 (malicious website IP address) (screenshot2). And then I restart the server (screenshot3). Finally, we open the browser on VM2 (the certificate of demoCA is already imported), and we type www.facebook.com, but the browser warns that the connection is not secure (screenshot4). So our attack fails.

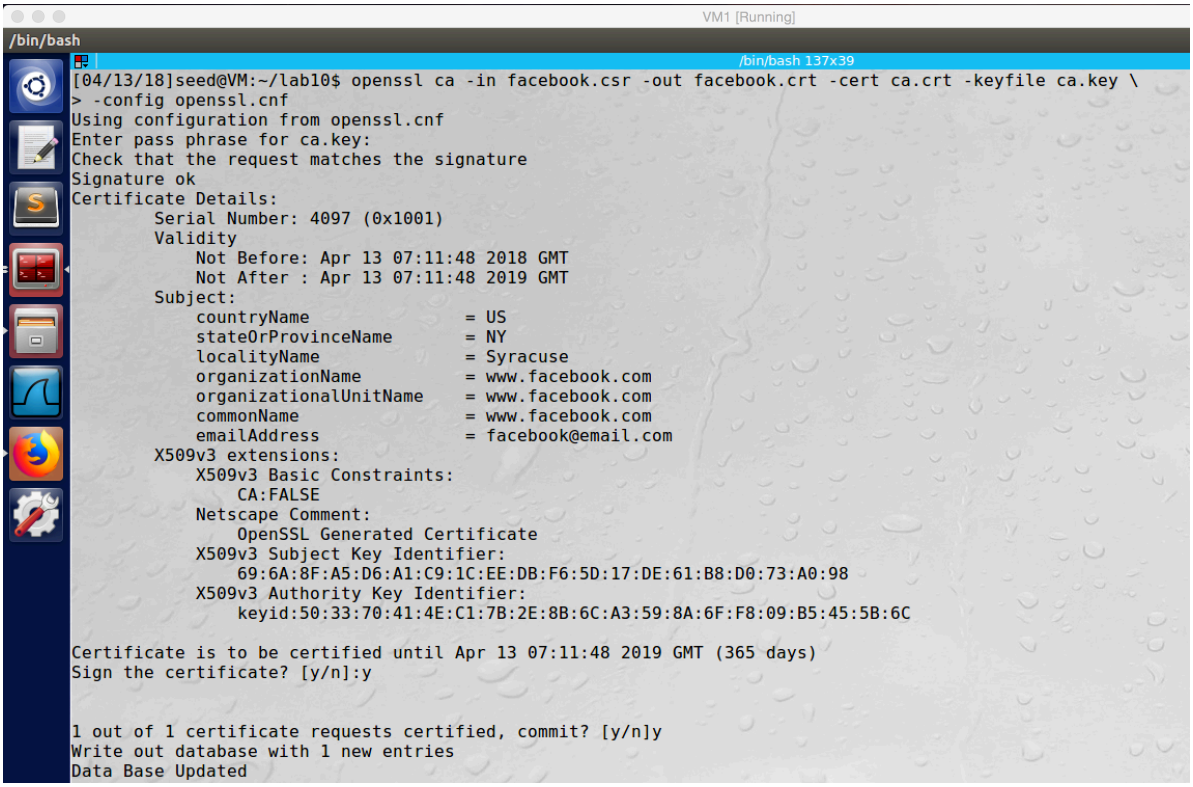
When there is a MITM attack, the attacker can have three choices. First, the attacker can direct authentic certificate of facebook.com to user. Because everything is valid, the connection will be established. However, because the attacker does not know the private key, he cannot get secret. As a result, MITM attack fails (the attacker cannot do any malicious).

Second, the attacker can send fake certificate to the user. Because the attacker does not own facebook.com, no CA will sign certificate to him; so the attacker cannot have valid certificate. However, he still can have a fake certificate which is a self-signed certificate. But when the attacker sends such certificate to the user, the user's browser will not accept it because the browser cannot find any trusted certificate that can be used to verify the certificate. So the MITM fails.

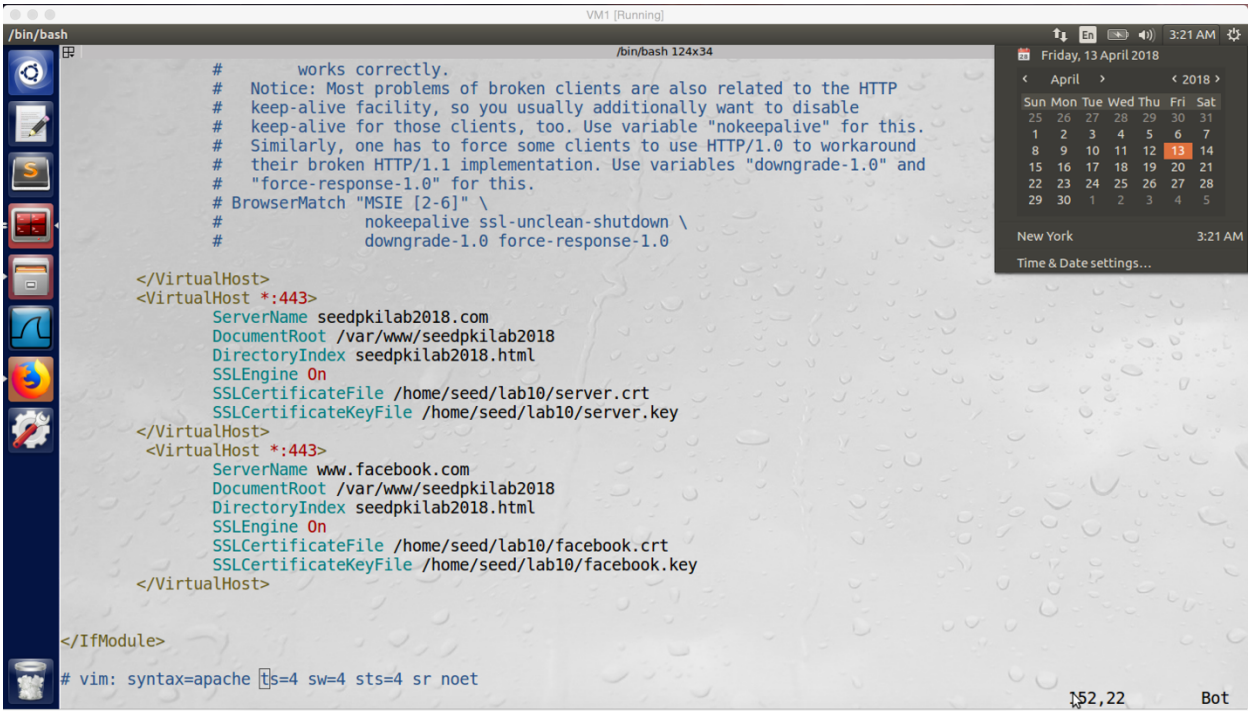
Third, attacker can send his valid certificate to the user, this is what we did in this task. In our case, we have a valid certificate signed by demoCA (this certificate is for SEEDPKILAB2018.com). So if we send this certificate to the user, the browser will trust it. However, there is another problem. The browser needs to make sure that the domain name which is typed by the user must match the domain name on the certificate. In our case, the URL typed in the browser is www.facebook.com, the domain name on the certificate is SEEDPKILAB2018.com. So these two names do not match. Therefore, the browser terminates the connection and warns user this connection is not secure. The MITM attack fails.

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

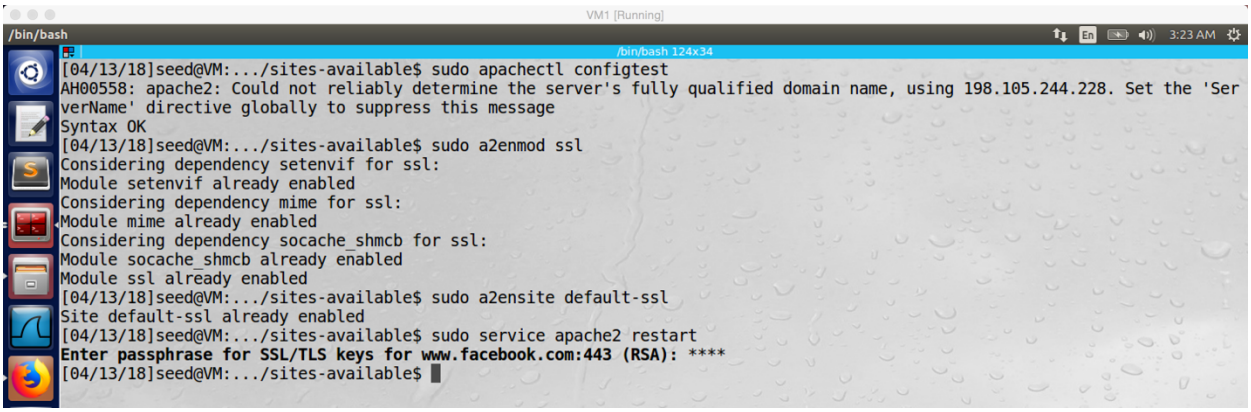
screenshot1. Because we already have the private key of the CA, we can create certificate for ourselves. In my case, I am going to create a certificate for www.facebook.com. We first generate public/private key for facebook. And then we generate CSR



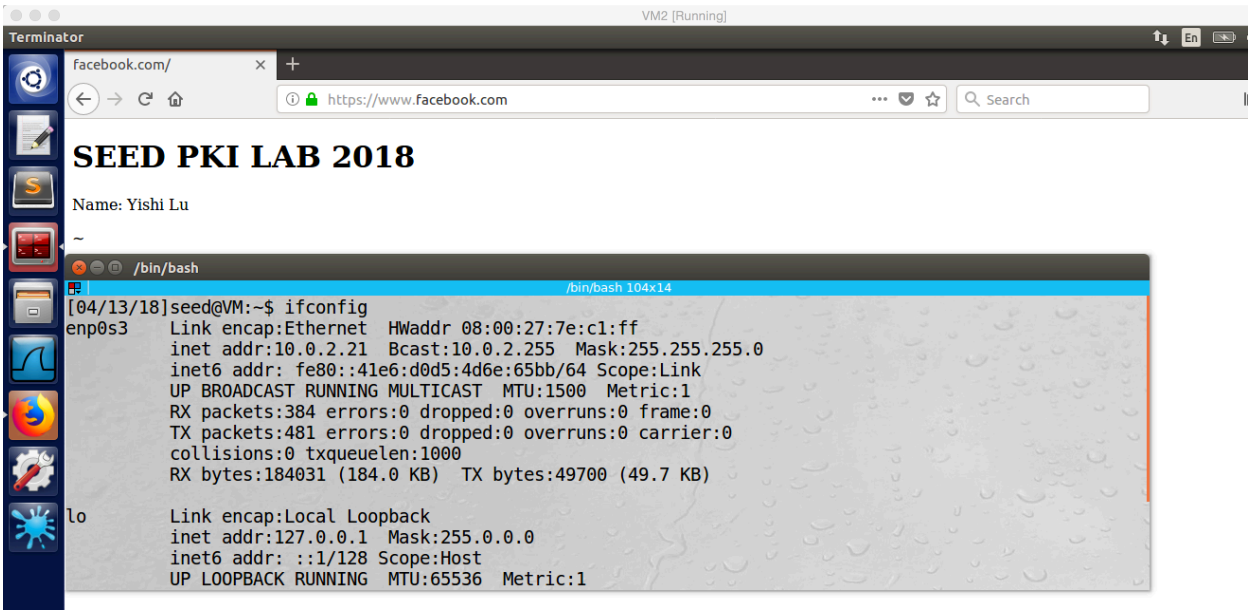
screenshot2. After we have CSR and CA's private key, we can generate certificate for facebook.com.



screenshot3. We add the facebook.com key to the VirtualHost entry



screenshot4. Restarting the server on VM1



screenshot5. This time, when we input www.facebook.com in the browser on VM2, we are redirected to attacker machine (10.0.2.20) successfully. The browser does not warn anything. So our MITM attack is successful

Observation and Explanation:

In this task, we are going to perform MITM attack again; but this time, we have compromised demoCA. Because demoCA is compromised (we have its private key), we can create certificate for www.facebook.com by ourselves. As screenshot 1 and 2 show, we use the private key of demoCA to create a certificate for facebook.com. And then we add the certificate into VirtualHost entry (screenshot3). Afterwards, we restart the server (screenshot4). Finally, we open browser and type www.facebook.com on VM2. We are redirected to the fake facebook on attacker machine (10.0.2.20), and the URL is still www.facebook.com with SSL, and the browser does not warn anything (screenshot5). So our attack is successful.

In fact, in this task, we did same thing as last task. The only difference is that we compromised demoCA, so we can create a valid certificate for domain www.facebook.com. After anything is setup, we visit www.facebook.com by typing in the browser on victim machine VM2. And we actually are redirected to fake www.facebook.com on attacker machine VM1, and the browser does not raise any suspicion. As we mentioned in the last task, the browser need to check two things when it receives certificate. The first is the validation of certificate. In our case, the certificate is valid, it is signed by a trust CA. The second is the matching between URL typed by user and the domain name on the certificate. In our case, the domain name on the certificate is www.facebook.com, and the URL is also www.facebook.com. So they are match. Therefore, the browser will not raise any suspicion, so our attack is successful.