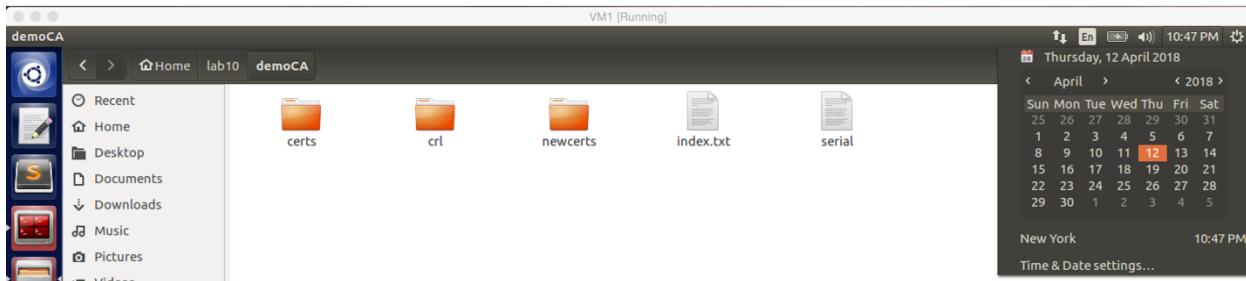


CSE644 Lab10
Yishi Lu
4/17/2018

Task 1: Becoming a Certificate Authority (CA)



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

A screenshot of a terminal window titled "/bin/bash". The terminal window has a dark blue background with white text. It shows the command "openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf" being run. The output of the command is displayed, including prompts for a PEM pass phrase and a Distinguished Name (DN). The terminal window also shows the user's path as "[04/12/18]seed@VM:~/lab10\$". At the bottom of the terminal window, there is a list of files: "ca.crt" and "ca.key". The terminal window has a scroll bar on the right side.

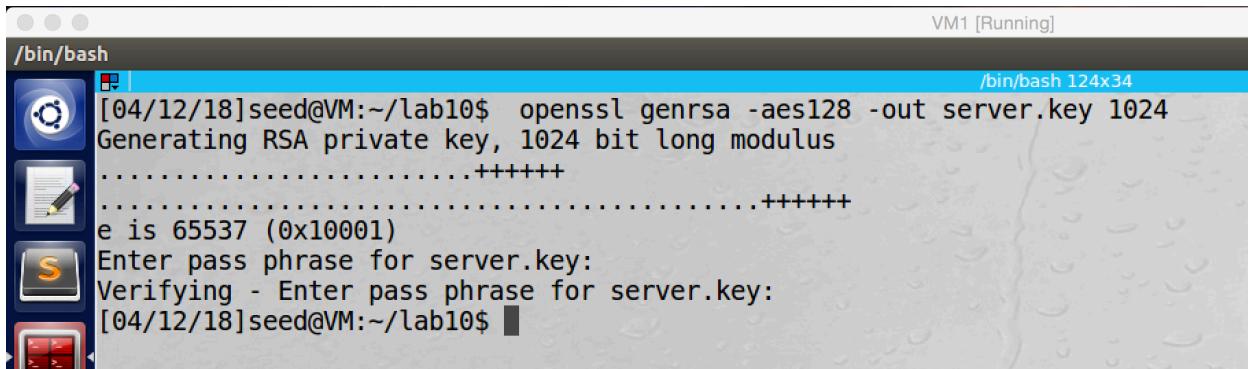
screenshot2. I use openssl command to generate self-signed certificate.

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.

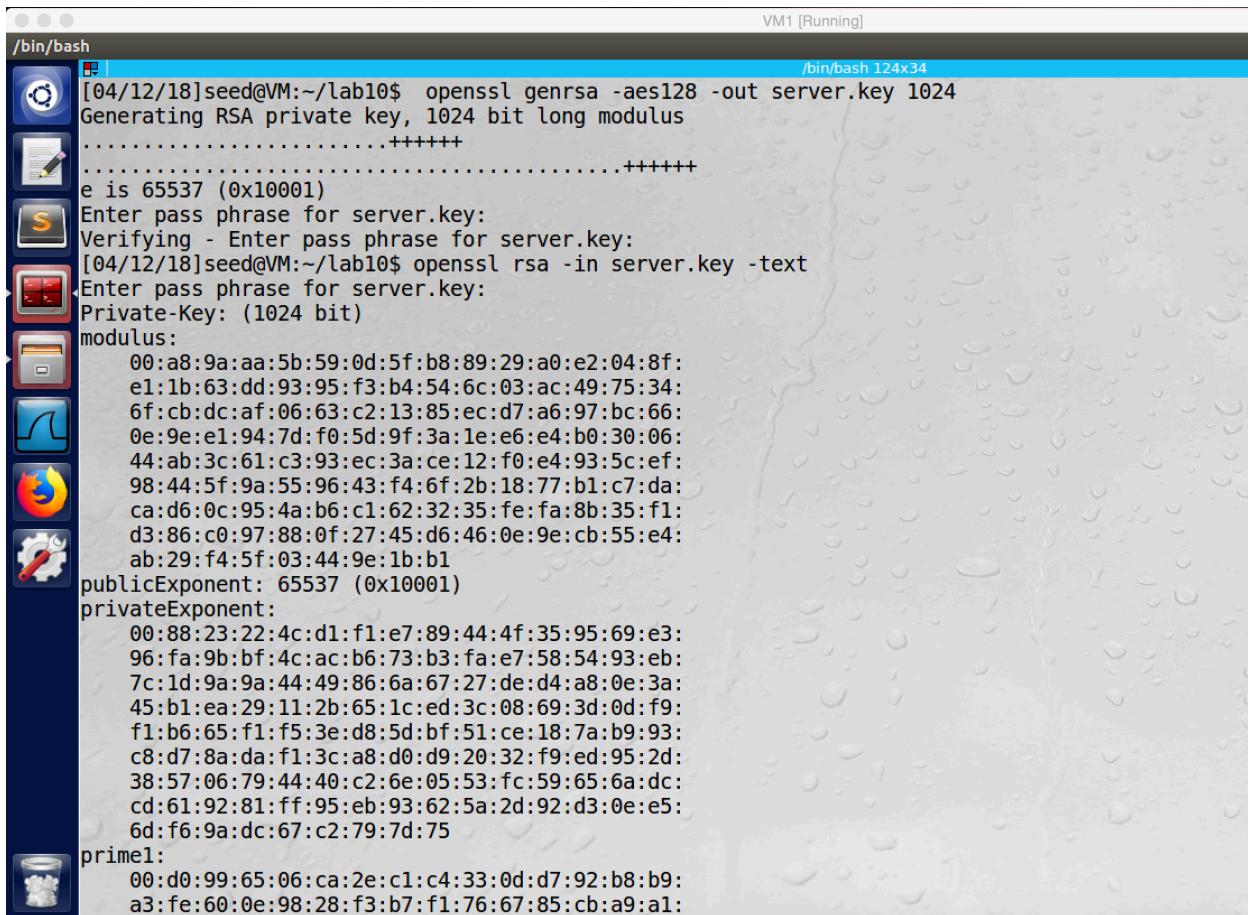
ca.key and ca.crt, they contain the CA's private key and public key certificate respectively. For this CA, we just call it demoCA.

Task 2: Creating a Certificate for SEEDPKILab1028.com



```
[04/12/18]seed@VM:~/lab10$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[04/12/18]seed@VM:~/lab10$
```

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



```
[04/12/18]seed@VM:~/lab10$ openssl genrsa -aes128 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[04/12/18]seed@VM:~/Lab10$ openssl rsa -in server.key -text
Enter pass phrase for server.key:
Private-Key: (1024 bit)
modulus:
 00:a8:9a:aa:5b:59:0d:5f:b8:89:29:a0:e2:04:8f:
  e1:1b:63:dd:93:95:f3:b4:54:6c:03:ac:49:75:34:
  6f:cb:dc:af:06:63:c2:13:85:ec:d7:a6:97:bc:66:
  0e:9e:e1:94:7d:f0:5d:9f:3a:1e:e6:e4:b0:30:06:
  44:ab:3c:61:c3:93:ec:3a:ce:12:f0:e4:93:5c:ef:
  98:44:5f:9a:55:96:43:f4:6f:2b:18:77:b1:c7:da:
  ca:d6:0c:95:4a:b6:c1:62:32:35:fe:fa:8b:35:f1:
  d3:86:c0:97:88:0f:27:45:d6:46:0e:9e:cb:55:e4:
  ab:29:f4:5f:03:44:9e:1b:b1
publicExponent: 65537 (0x10001)
privateExponent:
  00:88:23:22:4c:d1:f1:e7:89:44:4f:35:95:69:e3:
  96:fa:9b:bf:4c:ac:b6:73:b3:fa:e7:58:54:93:eb:
  7c:1d:9a:9a:44:49:86:6a:67:27:de:d4:a8:0e:3a:
  45:b1:ea:29:11:2b:65:1c:ed:3c:08:69:3d:0d:f9:
  f1:b6:65:f1:f5:3e:d8:5d:bf:51:ce:18:7a:b9:93:
  c8:d7:8a:da:f1:3c:a8:d0:d9:20:32:f9:ed:95:2d:
  38:57:06:79:44:40:c2:6e:05:53:fc:59:65:6a:dc:
  cd:61:92:81:ff:95:eb:93:62:5a:2d:92:d3:0e:e5:
  6d:f6:9a:dc:67:c2:79:7d:75
prime1:
  00:d0:99:65:06:ca:2e:c1:c4:33:0d:d7:92:b8:b9:
  a3:fe:60:0e:98:28:f3:b7:f1:76:67:85:cb:a9:a1:
```

screenshot2. We can open the key by openssl command

```
[04/12/18]seed@VM:~/lab10$ openssl req -new -key server.key -out server.csr -config openssl.cnf
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SEEDPKILab2018.com
Organizational Unit Name (eg, section) []:SEEDPKILab2018.com
Common Name (e.g. server FQDN or YOUR name) []:SEEDPKILab2018.com
Email Address []:SEEDPKILab2018@email.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:dees
An optional company name []:SEEDPKILab2018.com
[04/12/18]seed@VM:~/lab10$
```

screenshot3. Generating a certificate signing request (CSR) for SEEDPKILab2018.com

```
[04/12/18]seed@VM:~/lab10$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \
> -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
The organizationName field needed to be the same in the
CA certificate (SU) and the request (SEEDPKILab2018.com)
[04/12/18]seed@VM:~/lab10$
```

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

The screenshot shows a terminal window titled '/bin/bash' running on a virtual machine (VM1). The command entered is:

```
[04/12/18]seed@VM:~/lab10$ openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
```

The output shows the process of generating a certificate:

- Using configuration from openssl.cnf
- Enter pass phrase for ca.key:
- Check that the request matches the signature
- Signature ok
- Certificate Details:
 - Serial Number: 4096 (0x1000)
 - Validity
 - Not Before: Apr 13 03:12:43 2018 GMT
 - Not After : Apr 13 03:12:43 2019 GMT
- Subject:
 - countryName = US
 - stateOrProvinceName = NY
 - localityName = Syracuse
 - organizationName = SEEDPKILab2018.com
 - organizationalUnitName = SEEDPKILab2018.com
 - commonName = SEEDPKILab2018.com
 - emailAddress = SEEDPKILab2018@email.com
- X509v3 extensions:
 - X509v3 Basic Constraints:
 - CA:FALSE
 - Netscape Comment:
 - OpenSSL Generated Certificate
 - X509v3 Subject Key Identifier:
 - D8:D7:76:DE:51:13:7B:C1:D8:B5:BD:39:95:ED:9F:71:D0:B1:9F:B1
 - X509v3 Authority Key Identifier:
 - keyid:50:33:70:41:4E:C1:7B:2E:8B:6C:A3:59:8A:6F:F8:09:B5:45:5B:6C
- Certificate is to be certified until Apr 13 03:12:43 2019 GMT (365 days)
- Sign the certificate? [y/n]:y
- 1 out of 1 certificate requests certified, commit? [y/n]
- Write out database with 1 new entries
- Data Base Updated

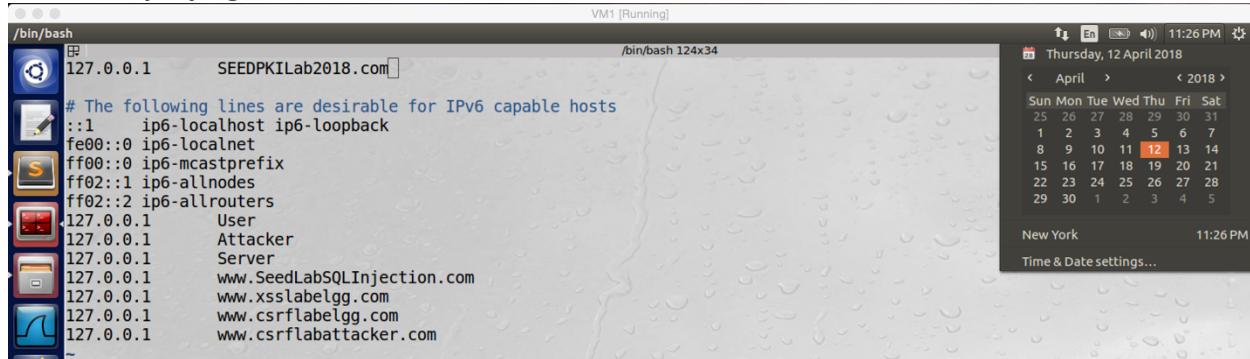
```
[04/12/18]seed@VM:~/lab10$
```

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:

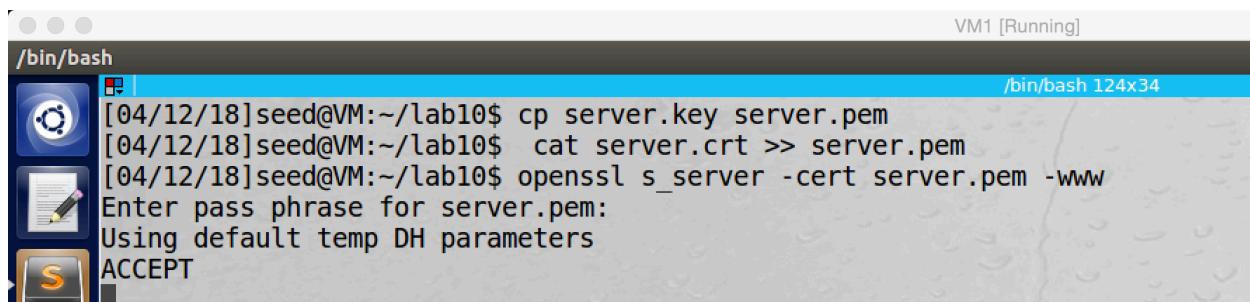
In this task, we use the demoCA to generate a certificate to a company, this company is called SEEDPKILab2018.com. Before the company can make CSR, it must have public/private key. So we run openssl command to generate the public/private key (screenshot1 and 2). And then the company can make CSR to the 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



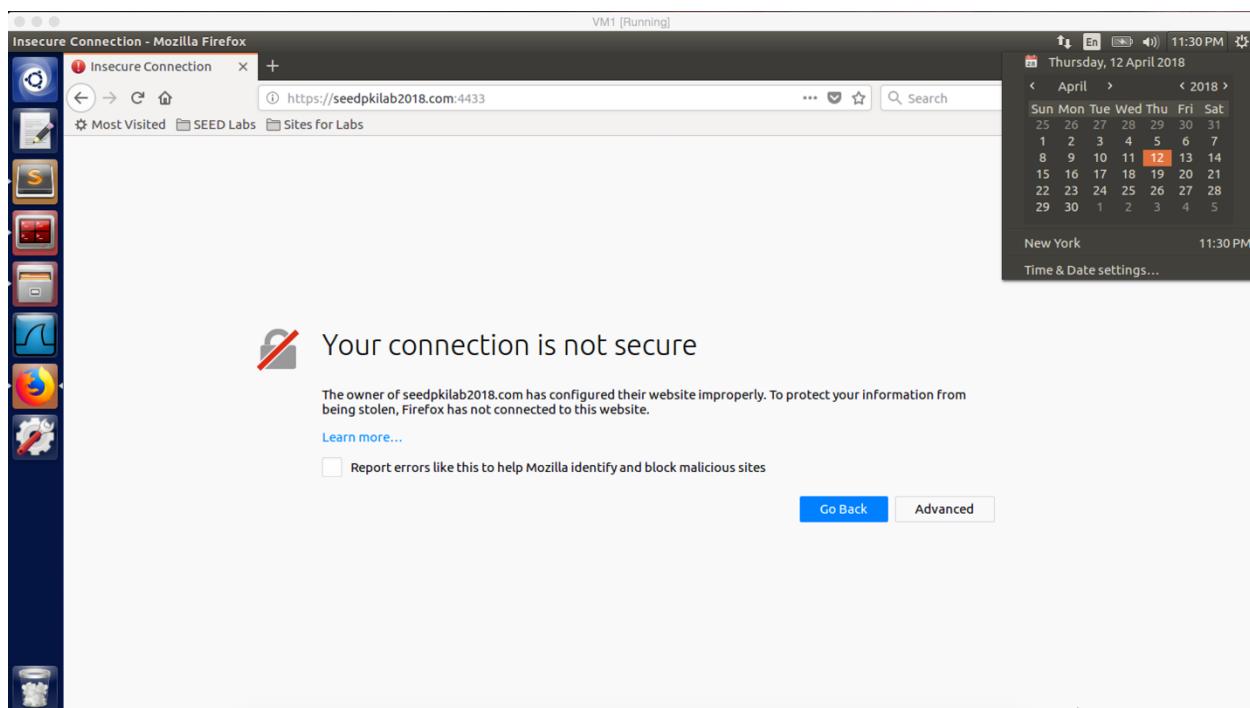
```
VM1 [Running]
/bin/bash
127.0.0.1      SEEDPKILab2018.com
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1      User
127.0.0.1      Attacker
127.0.0.1      Server
127.0.0.1      www.SeedLabSQLInjection.com
127.0.0.1      www.xsslabelgg.com
127.0.0.1      www.csrflabelgg.com
127.0.0.1      www.csrflabattacker.com
```

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

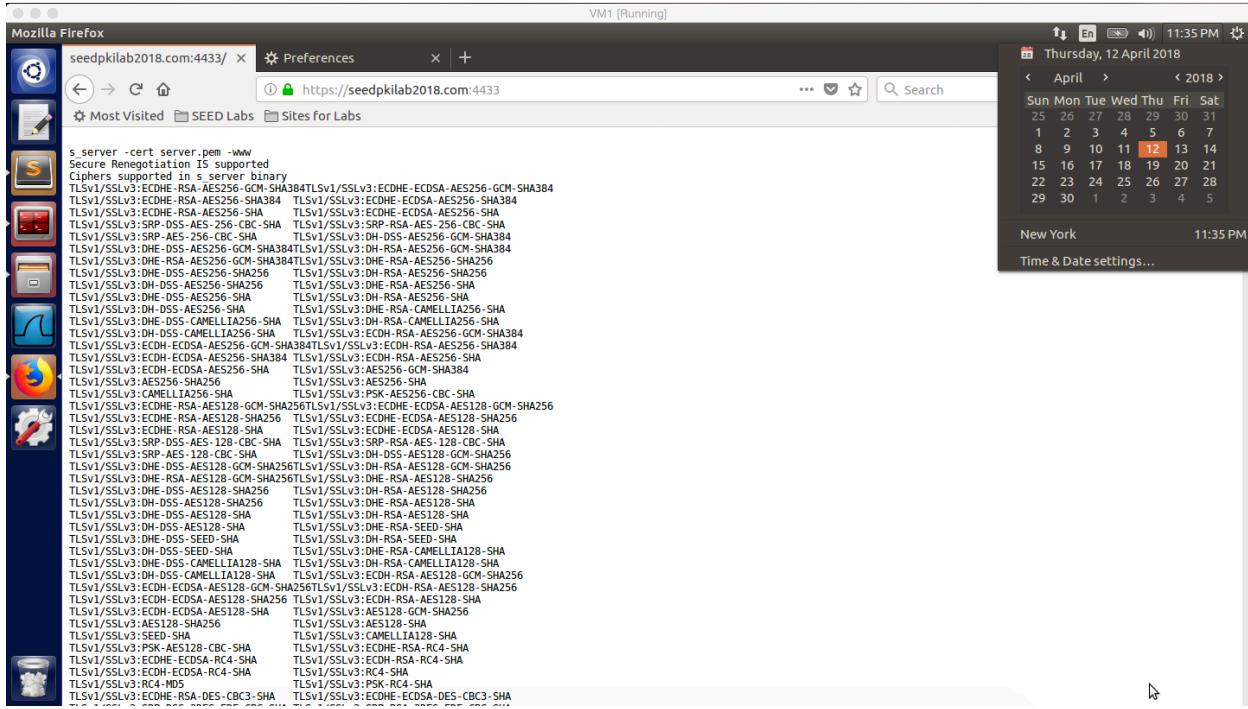


```
VM1 [Running]
/bin/bash
[04/12/18]seed@VM:~/lab10$ cp server.key server.pem
[04/12/18]seed@VM:~/lab10$ cat server.crt >> server.pem
[04/12/18]seed@VM:~/lab10$ openssl s_server -cert server.pem -WWW
Enter pass phrase for server.pem:
Using default temp DH parameters
ACCEPT
```

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.

seedpkilab2018.com:4433/ seedpkilab2018.com:4433/ seedpkilab2018.com:4433/ +

<https://seedpkilab2018.com:4433>

... Search

```

TLSv1/SSLv3:ECDH-ECDSA-AES128-GCM-SHA256 TLSv1/SSLv3:ECDH-RSA-AES128-SHA256
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA256 TLSv1/SSLv3:ECDH-RSA-AES128-SHA
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA TLSv1/SSLv3:AES128-GCM-SHA256
TLSv1/SSLv3:AES128-SHA256 TLSv1/SSLv3:AES128-SHA
TLSv1/SSLv3:SEED-SHA TLSv1/SSLv3:CAMELLIA128-SHA
TLSv1/SSLv3:PSK-AES128-CBC-SHA TLSv1/SSLv3:ECDOHE-RSA-RC4-SHA
TLSv1/SSLv3:ECDOHE-ECDSA-RC4-SHA TLSv1/SSLv3:ECDH-RSA-RC4-SHA
TLSv1/SSLv3:ECDH-ECDSA-RC4-SHA TLSv1/SSLv3:RC4-SHA
TLSv1/SSLv3:RC4-MDS TLSv1/SSLv3:PSK-RC4-SHA
TLSv1/SSLv3:ECDOHE-RSA-DES-CBC3-SHA TLSv1/SSLv3:ECDOHE-ECDSA-DES-CBC3-SHA
TLSv1/SSLv3:SRP-3DES-EDE-CBC-SHA TLSv1/SSLv3:SRP-RSA-3DES-EDE-CBC-SHA
TLSv1/SSLv3:SRP-3DES-EDE-CBC-SHA TLSv1/SSLv3:EDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:EDH-DSS-DES-CBC3-SHA TLSv1/SSLv3:DH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:DH-DSS-DES-CBC3-SHA TLSv1/SSLv3:ECDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:ECDH-ECDSA-DES-CBC3-SHA TLSv1/SSLv3:DES-CBC3-SHA
TLSv1/SSLv3:PSK-3DES-EDE-CBC-SHA
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:0x04+0x08:0x05+0x08:0x06+0x08:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
-- 
Reused, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
SSL-Session:
Protocol : TLSv1.2
Cipher   : ECDOHE-RSA-AES128-GCM-SHA256
Session-ID: A9CF25BF630A90DFF3129434458AD1C1E370CDCC368F6DAEA3C83133CE232AD3
Session-ID-ctx: 01000000
Master-Key: 63E9EB93D198CD2B5D4B4A5B5D1E2DAD80A06B3766EC09A68F5731860CB8133A18A6FFBA5C9FCAC90EC1B2A0A3FAC253
Key-Ag : None
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1524006918
Timeout  : 300 (sec)
Verify return code: 0 (ok)
-- 
0 items in the session cache
0 client connects (SSL_connect())
0 client renegotiates (SSL_connect())
0 client connects that finished
2 server accepts (SSL_accept())
0 server renegotiates (SSL_accept())
2 server accepts that finished
1 session cache hits
0 session cache misses
0 session cache timeouts
0 callback cache hits
0 cache full overflows (128 allowed)
-- 
no client certificate available

```

screenshot for webpage with unchanged server.pem file

seedpkilab2018.com:4433/ seedpkilab2018.com:4433/ seedpkilab2018.com:4433/ +

<https://seedpkilab2018.com:4433>

... Search

```

TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA256 TLSv1/SSLv3:ECDH-RSA-AES128-SHA256
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA256 TLSv1/SSLv3:ECDH-RSA-AES128-SHA
TLSv1/SSLv3:ECDH-ECDSA-AES128-SHA TLSv1/SSLv3:AES128-GCM-SHA256
TLSv1/SSLv3:AES128-SHA256 TLSv1/SSLv3:AES128-SHA
TLSv1/SSLv3:SEED-SHA TLSv1/SSLv3:CAMELLIA128-SHA
TLSv1/SSLv3:PSK-AES128-CBC-SHA TLSv1/SSLv3:ECDOHE-RSA-RC4-SHA
TLSv1/SSLv3:ECDOHE-ECDSA-RC4-SHA TLSv1/SSLv3:ECDH-RSA-RC4-SHA
TLSv1/SSLv3:ECDH-ECDSA-RC4-SHA TLSv1/SSLv3:RC4-SHA
TLSv1/SSLv3:RC4-MDS TLSv1/SSLv3:PSK-RC4-SHA
TLSv1/SSLv3:ECDOHE-RSA-DES-CBC3-SHA TLSv1/SSLv3:ECDOHE-ECDSA-DES-CBC3-SHA
TLSv1/SSLv3:SRP-3DES-EDE-CBC-SHA TLSv1/SSLv3:SRP-RSA-3DES-EDE-CBC-SHA
TLSv1/SSLv3:SRP-3DES-EDE-CBC-SHA TLSv1/SSLv3:EDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:EDH-DSS-DES-CBC3-SHA TLSv1/SSLv3:DH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:DH-DSS-DES-CBC3-SHA TLSv1/SSLv3:ECDH-RSA-DES-CBC3-SHA
TLSv1/SSLv3:ECDH-ECDSA-DES-CBC3-SHA TLSv1/SSLv3:DES-CBC3-SHA
TLSv1/SSLv3:PSK-3DES-EDE-CBC-SHA
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:0x04+0x08:0x05+0x08:0x06+0x08:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
-- 
Reused, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
SSL-Session:
Protocol : TLSv1.2
Cipher   : ECDOHE-RSA-AES128-GCM-SHA256
Session-ID: 2B2F31161C3DCC3236B985D5B21C1DC7EDF201F7B31546DE3BB807B55D28C757
Session-ID-ctx: 01000000
Master-Key: 60172C5E889267058CE0D127B80AE1AB467650CFDC9D56E5B280970C123397376F1DE05CB514FC1DD07B11073AFF1A7
Key-Ag : None
PSK identity: None
PSK identity hint: None
SRP username: None
Start Time: 1524006968
Timeout  : 300 (sec)
Verify return code: 0 (ok)
-- 
0 items in the session cache
0 client connects (SSL_connect())
0 client renegotiates (SSL_connect())
0 client connects that finished
2 server accepts (SSL_accept())
0 server renegotiates (SSL_accept())
2 server accepts that finished
1 session cache hits
0 session cache misses
0 session cache timeouts
0 callback cache hits
0 cache full overflows (128 allowed)
-- 
no client certificate available

```

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=igmp, 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$
```

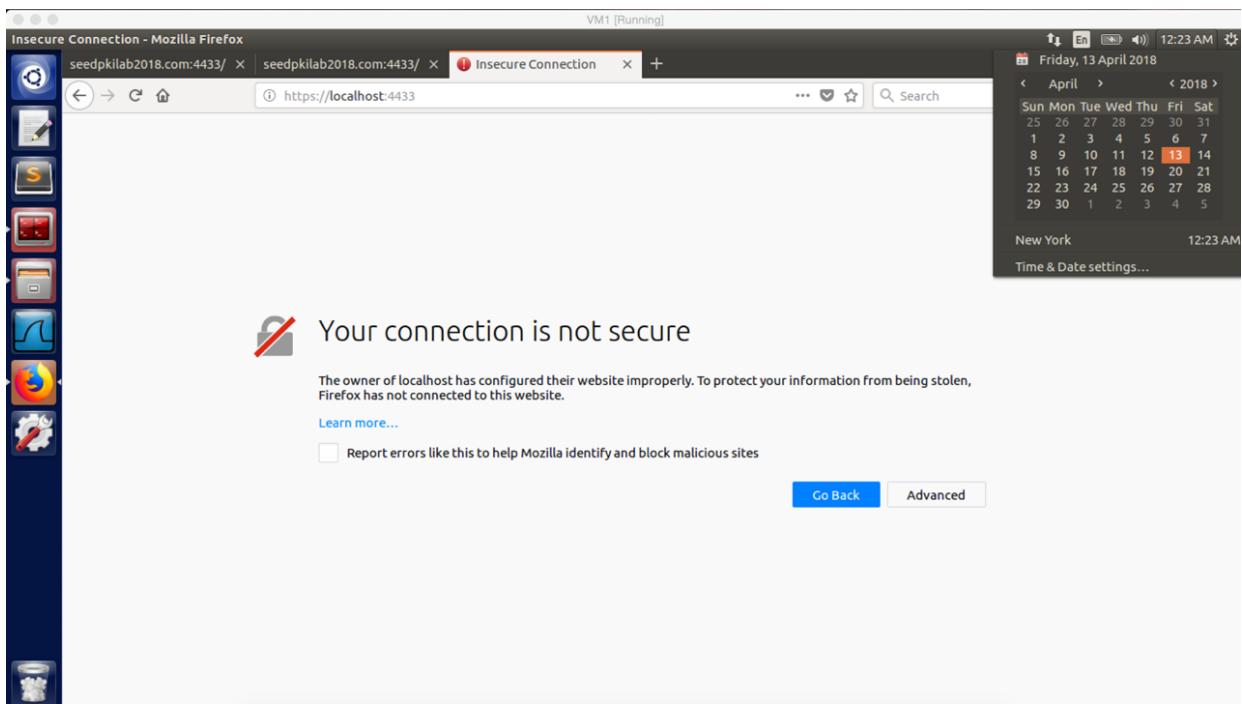
After changed the private key section, the web server cannot be launched

```
/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$
```

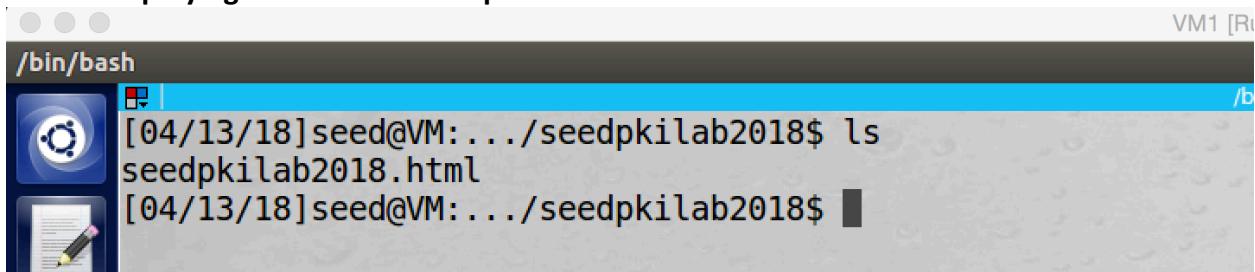
After changed the certificate section, the webserver cannot be launched

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 browser 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.

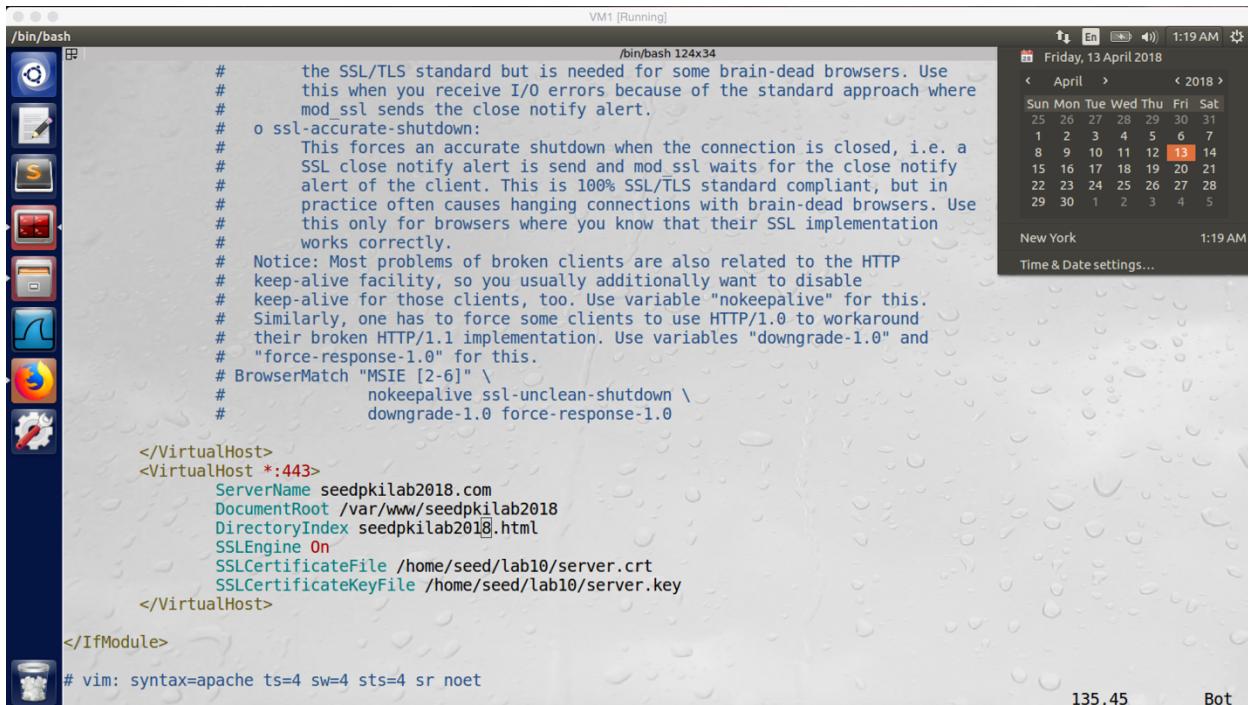


Task 4: Deploying Certificate in an Apache-Based HTTPS Website





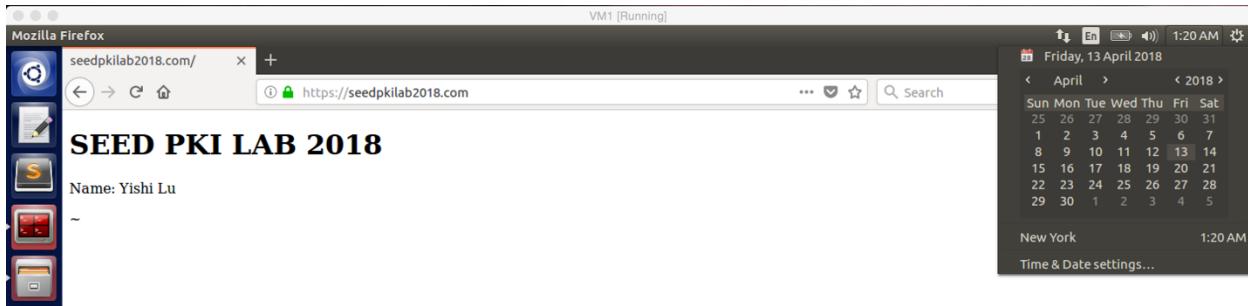
screenshot1. I first create a html file which called seedpkilab2018.html, and this file is under /var/www/seedpkilab2018 directory.



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

```
[04/13/18]seed@VM:~$ sudo apachectl configtest
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 198.105.244.228. Set the 'ServerName' directive globally to suppress this message
Syntax OK
[04/13/18]seed@VM:~$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
[04/13/18]seed@VM:~$ sudo a2ensite default-ssl
Site default-ssl already enabled
[04/13/18]seed@VM:~$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for seedpkilab2018.com:443 (RSA): ****
[04/13/18]seed@VM:~$
```

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.

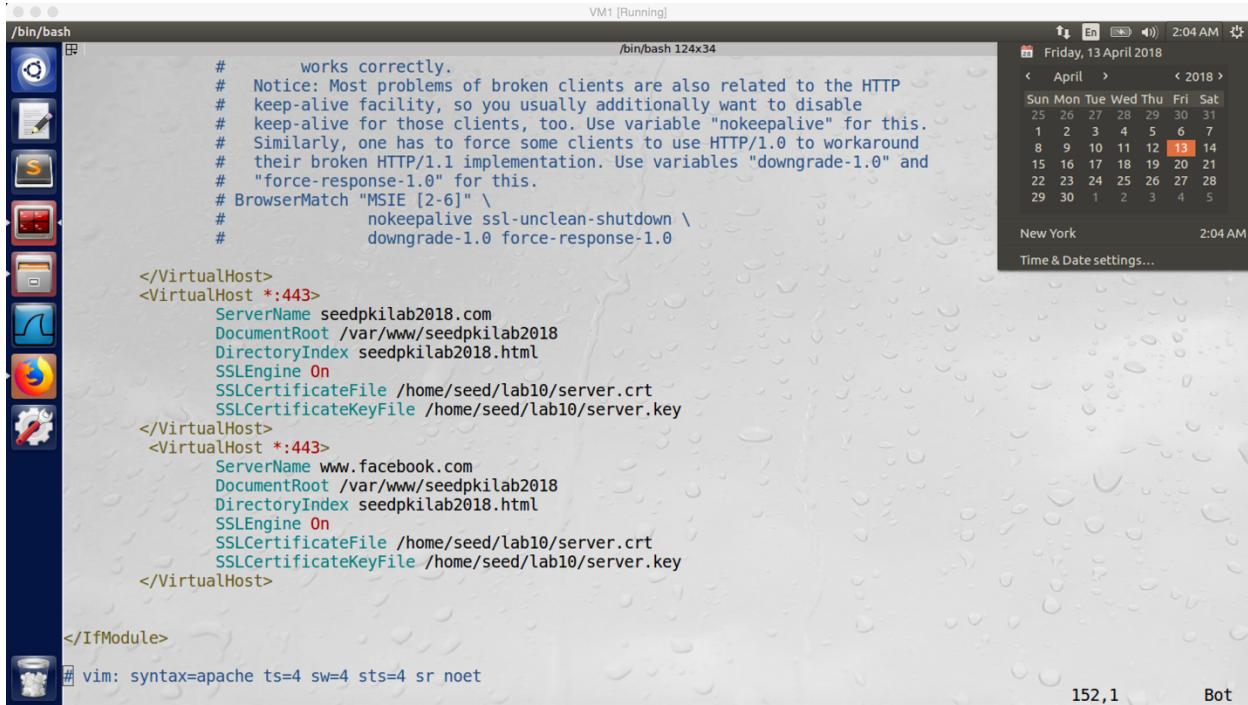
Second step, we need to setup the configuration file for the apache server. We go to the directory /etc/apache2/sites-available, and we open file default-ssl.conf. Afterwards, we add a VirtualHost entry in this file, the contents are shown in screenshot2. It includes, server name, directory of website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

Last step, now we can restart the apache server. As screenshot3 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.



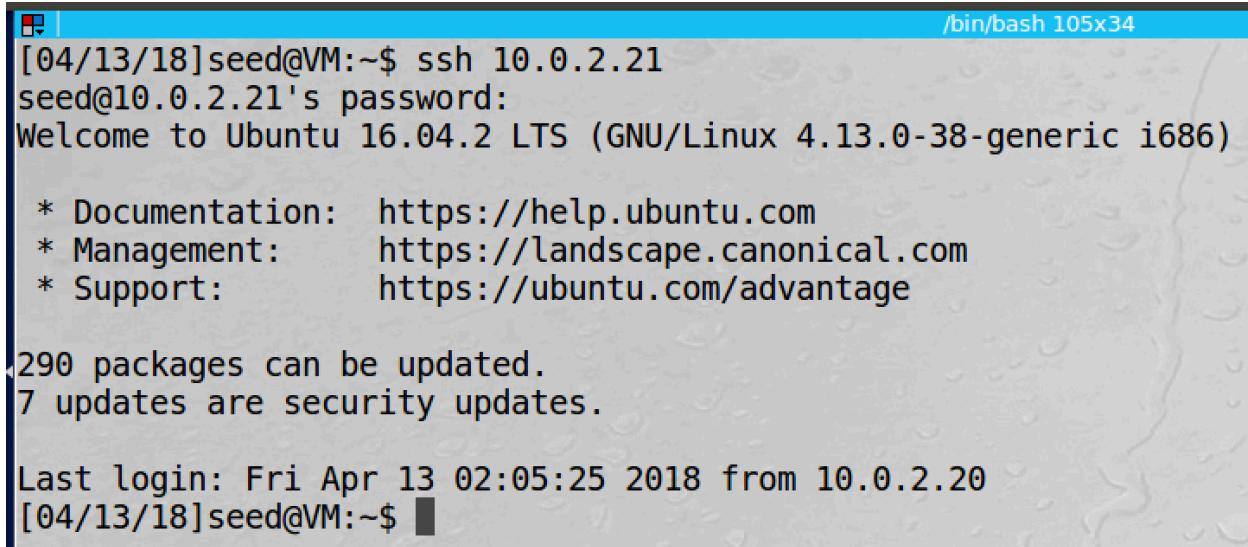
The screenshot shows a desktop environment with a terminal window open. The terminal window title is "/bin/bash" and the command being run is "vim /etc/apache2/sites-available/seedpkilab2018.conf". The code in the terminal is an Apache configuration file. It includes sections for virtual hosts on port 443. The first virtual host is for "seedpkilab2018.com" and the second is for "www.facebook.com". Both use SSL certificates from "/home/seed/lab10/server.crt" and keys from "/home/seed/lab10/server.key". The configuration also includes directives for "nokeepalive ssl-unclean-shutdown \ downgrade-1.0 force-response-1.0". The terminal window has a status bar at the bottom showing "# vim: syntax=apache ts=4 sw=4 sts=4 sr noet". The desktop background shows a calendar for April 2018 with the 13th highlighted in red. The desktop icons include a browser, file manager, terminal, and system settings.

```
#       works correctly.
# Notice: Most problems of broken clients are also related to the HTTP
# keep-alive facility, so you usually additionally want to disable
# keep-alive for those clients, too. Use variable "nokeepalive" for this.
# Similarly, one has to force some clients to use HTTP/1.0 to workaround
# their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and
# "force-response-1.0" for this.
# BrowserMatch "MSIE [2-6]" \
#               nokeepalive ssl-unclean-shutdown \
#               downgrade-1.0 force-response-1.0

</VirtualHost>
<VirtualHost *:443>
    ServerName seedpkilab2018.com
    DocumentRoot /var/www/seedpkilab2018
    DirectoryIndex seedpkilab2018.html
    SSLEngine On
    SSLCertificateFile /home/seed/lab10/server.crt
    SSLCertificateKeyFile /home/seed/lab10/server.key
</VirtualHost>
<VirtualHost *:443>
    ServerName www.facebook.com
    DocumentRoot /var/www/seedpkilab2018
    DirectoryIndex seedpkilab2018.html
    SSLEngine On
    SSLCertificateFile /home/seed/lab10/server.crt
    SSLCertificateKeyFile /home/seed/lab10/server.key
</VirtualHost>

</IfModule>
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

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



The screenshot shows a terminal session on VM1. The user runs "ssh 10.0.2.21" and enters the password for the victim machine. The victim machine's welcome message is displayed, followed by standard Ubuntu 16.04 LTS information. The user then runs "apt-get update" and "apt-get upgrade" to check for updates. The output shows 290 packages can be updated, with 7 security updates. The terminal ends with the prompt "[04/13/18]seed@VM:~\$".

```
[04/13/18]seed@VM:~$ ssh 10.0.2.21
seed@10.0.2.21's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.13.0-38-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

290 packages can be updated.
7 updates are security updates.

Last login: Fri Apr 13 02:05:25 2018 from 10.0.2.20
[04/13/18]seed@VM:~$
```

```

/bin/bash
10.0.2.20      www.facebook.com
127.0.0.1      localhost
127.0.1.1      VM

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
127.0.0.1      User
127.0.0.1      Attacker
127.0.0.1      Server
127.0.0.1      www.SeedLabSQLInjection.com
127.0.0.1      www.xsslabelgg.com
127.0.0.1      www.csrflabelgg.com
127.0.0.1      www.csrflabattacker.com

```

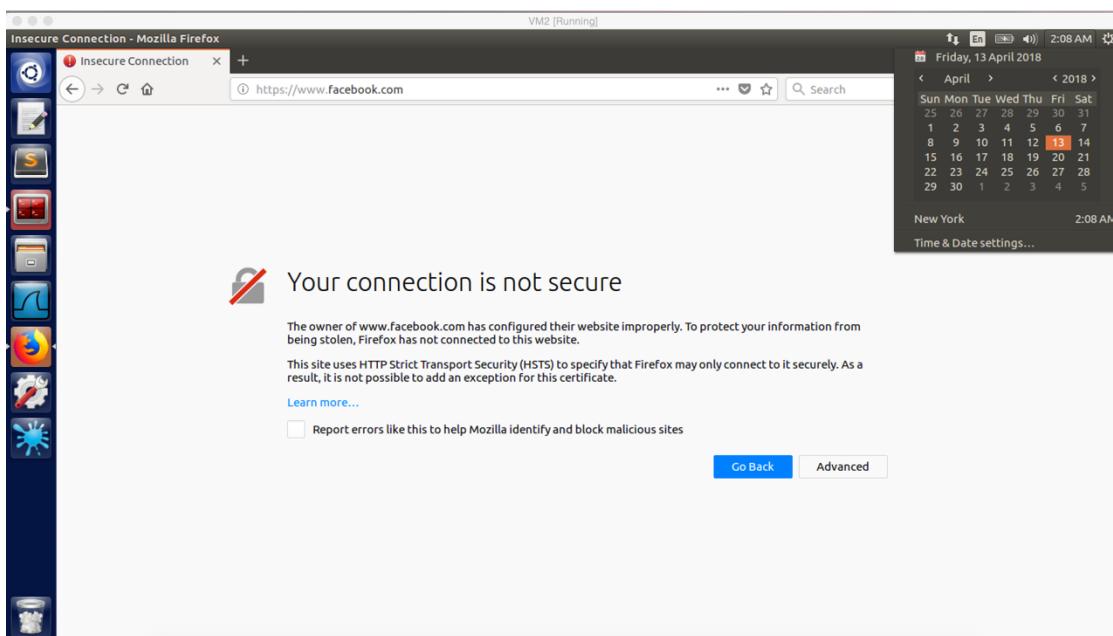
screenshot2. We use SSH to connect to victim machine VM2, and we open the hosts file to map www.facebook.com to the attacker IP 10.0.2.20

```

[04/13/18]seed@VM:.../sites-available$ sudo apachectl configtest
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 198.105.244.228. Set the 'ServerName' directive globally to suppress this message
Syntax OK
[04/13/18]seed@VM:.../sites-available$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
[04/13/18]seed@VM:.../sites-available$ sudo a2ensite default-ssl
Site default-ssl already enabled
[04/13/18]seed@VM:.../sites-available$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for www.facebook.com:443 (RSA): ****
[04/13/18]seed@VM:.../sites-available$ sudo vi /etc/hosts
[04/13/18]seed@VM:.../sites-available$ 

```

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:

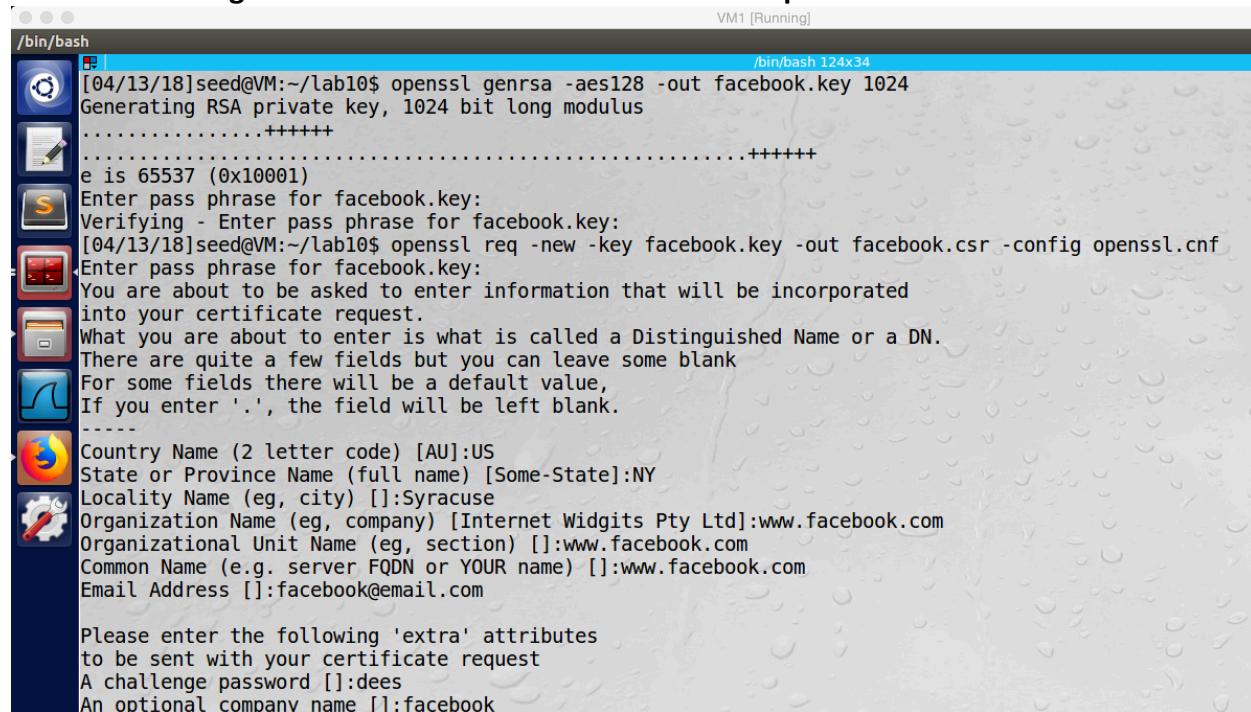
In this task, we are going to perform MITM attack, VM1 is the attacker machine (10.0.2.20), VM2 is the victim machine (10.0.2.21). First, we need to setup a malicious website. In my case, I decide to create a fake fackebook.com. We just follow the steps in the task4 to setup a website (screenshot1). 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.

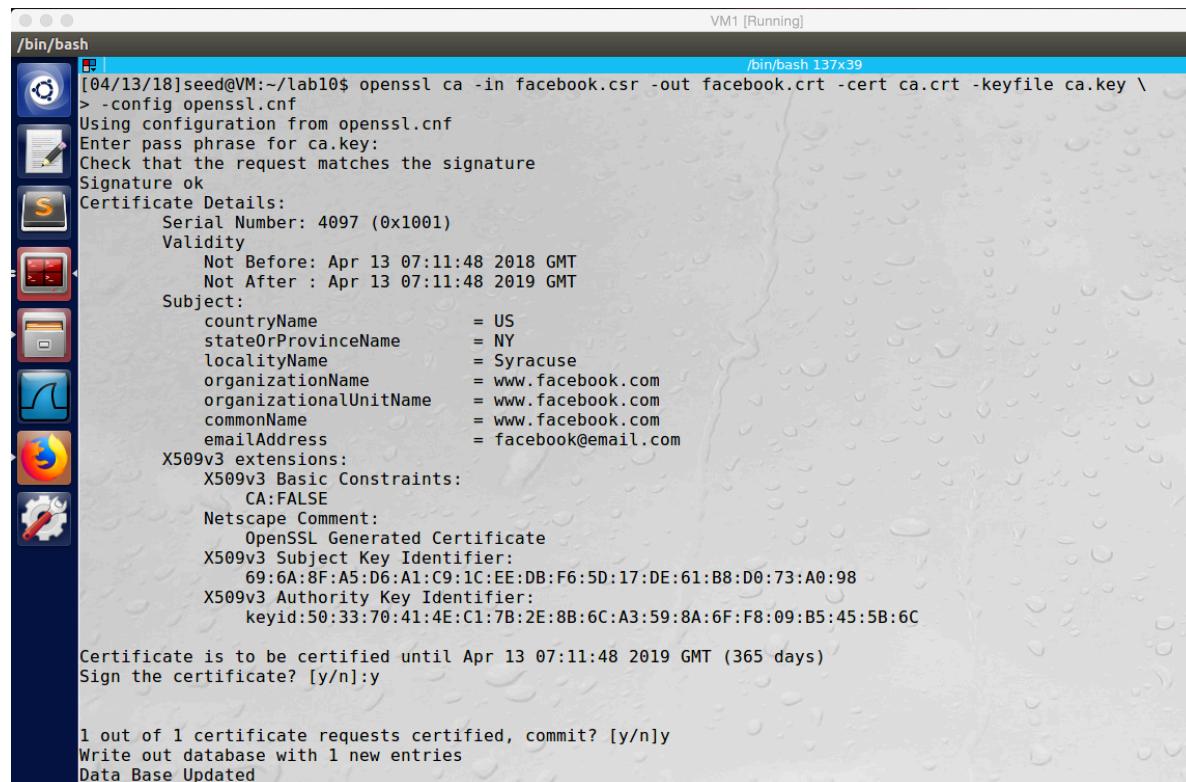
Task 6: Launching a Man-In-The-middle Attack with a Compromised CA



```
[04/13/18]seed@VM:~/lab10$ openssl genrsa -aes128 -out facebook.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
Enter pass phrase for facebook.key:
Verifying - Enter pass phrase for facebook.key:
[04/13/18]seed@VM:~/lab10$ openssl req -new -key facebook.key -out facebook.csr -config openssl.cnf
Enter pass phrase for facebook.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:Syracuse
Organization Name (eg, company) [Internet Widgits Pty Ltd]:www.facebook.com
Organizational Unit Name (eg, section) []:www.facebook.com
Common Name (e.g. server FQDN or YOUR name) []:www.facebook.com
Email Address []:facebook@email.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:dees
An optional company name []:facebook
```

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

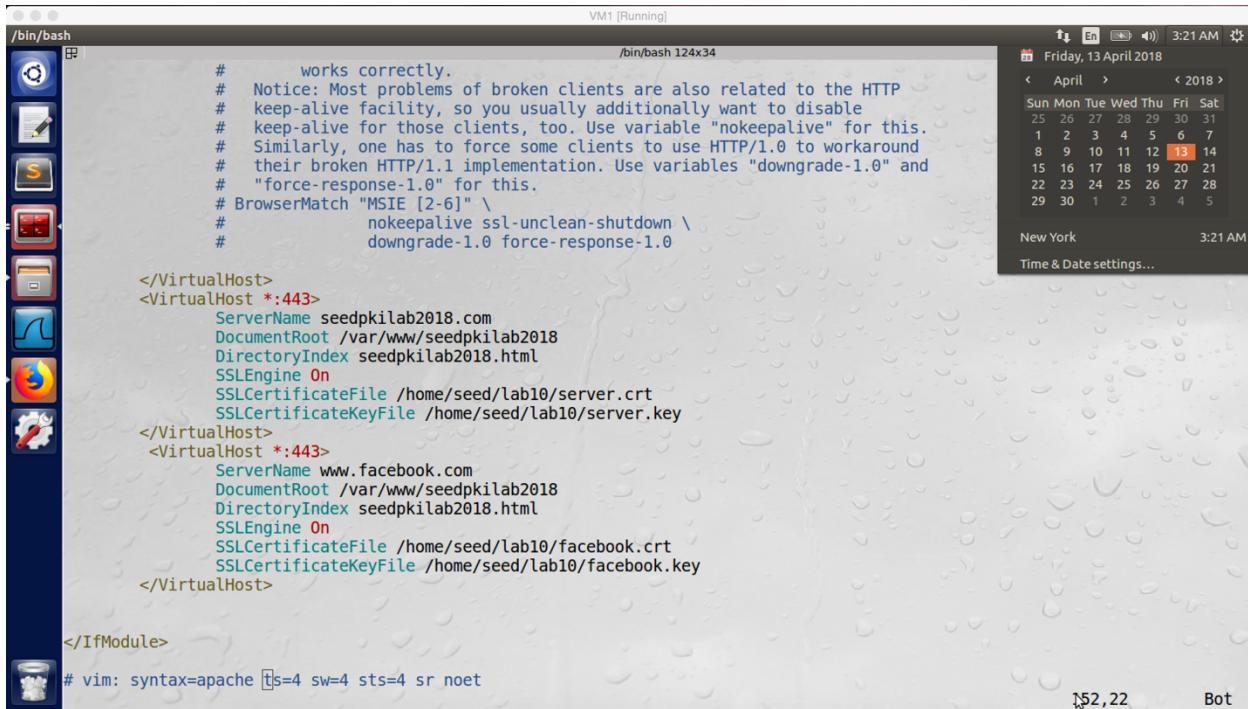


```
[04/13/18]seed@VM:~/lab10$ openssl ca -in facebook.csr -out facebook.crt -cert ca.crt -keyfile ca.key \
> -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 4097 (0x1001)
    Validity
        Not Before: Apr 13 07:11:48 2018 GMT
        Not After : Apr 13 07:11:48 2019 GMT
    Subject:
        countryName          = US
        stateOrProvinceName = NY
        localityName        = Syracuse
        organizationName   = www.facebook.com
        organizationalUnitName = www.facebook.com
        commonName           = www.facebook.com
        emailAddress         = facebook@email.com
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
        69:6A:8F:A5:D6:A1:C9:1C:EE:DB:F6:5D:17:DE:61:B8:D0:73:A0:98
    X509v3 Authority Key Identifier:
        keyid:50:33:70:41:4E:C1:7B:2E:8B:6C:A3:59:8A:6F:F8:09:B5:45:5B:6C

Certificate is to be certified until Apr 13 07:11:48 2019 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

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



```
#      works correctly.
# Notice: Most problems of broken clients are also related to the HTTP
# keep-alive facility, so you usually additionally want to disable
# keep-alive for those clients, too. Use variable "nokeepalive" for this.
# Similarly, one has to force some clients to use HTTP/1.0 to workaround
# their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and
# "force-response-1.0" for this.
# BrowserMatch "MSIE [2-6]" \
#               nokeepalive ssl-unclean-shutdown \
#               downgrade-1.0 force-response-1.0

</VirtualHost>
<VirtualHost *:443>
    ServerName seedpkilab2018.com
    DocumentRoot /var/www/seedpkilab2018
    DirectoryIndex seedpkilab2018.html
    SSLEngine On
    SSLCertificateFile /home/seed/lab10/server.crt
    SSLCertificateKeyFile /home/seed/lab10/server.key
</VirtualHost>
<VirtualHost *:443>
    ServerName www.facebook.com
    DocumentRoot /var/www/seedpkilab2018
    DirectoryIndex seedpkilab2018.html
    SSLEngine On
    SSLCertificateFile /home/seed/lab10/facebook.crt
    SSLCertificateKeyFile /home/seed/lab10/facebook.key
</VirtualHost>

</IfModule>

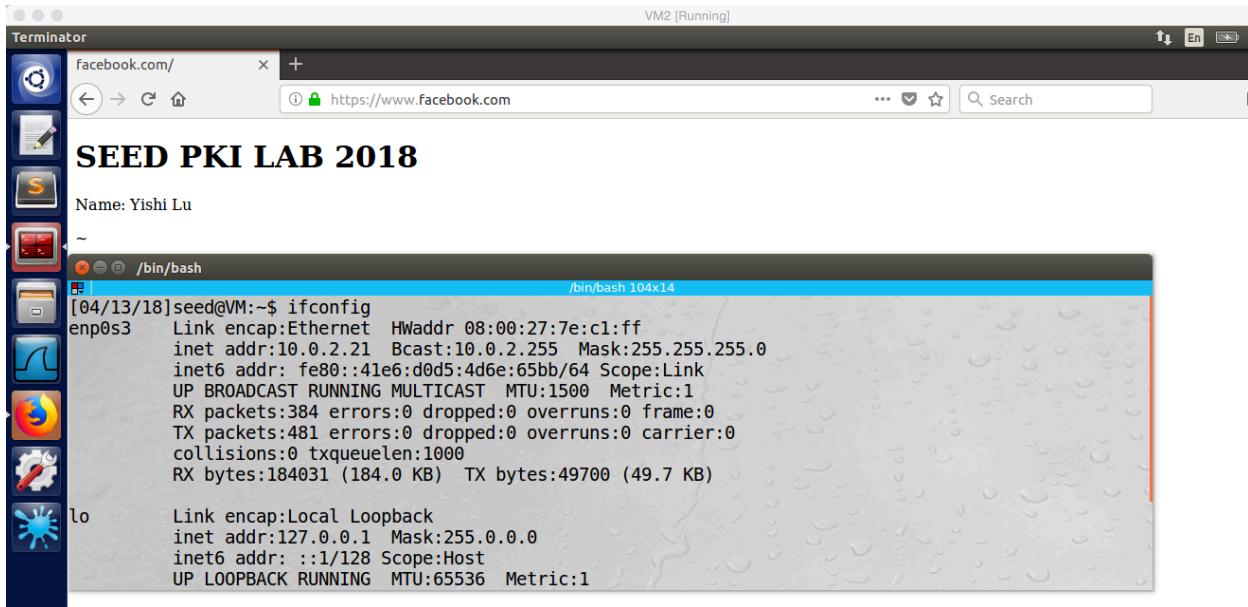
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

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



```
[04/13/18]seed@VM:.../sites-available$ sudo apachectl configtest
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 198.105.244.228. Set the 'ServerName' directive globally to suppress this message
Syntax OK
[04/13/18]seed@VM:.../sites-available$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
[04/13/18]seed@VM:.../sites-available$ sudo a2ensite default-ssl
Site default-ssl already enabled
[04/13/18]seed@VM:.../sites-available$ sudo service apache2 restart
Enter passphrase for SSL/TLS keys for www.facebook.com:443 (RSA): ****
[04/13/18]seed@VM:.../sites-available$
```

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.