ASSIGNMENT # 6

Student Name & ID	**Abdulrazzaq Alsiddiq 202004464**
Student Name & ID	**Anas Madkoor, 202104114**
Student Name & ID	**Omar Amin, 202003122**
Student Name & ID	**Lance Eric Ruben, 202005801**
Student Name & ID	**Ali Zair, 202109964**

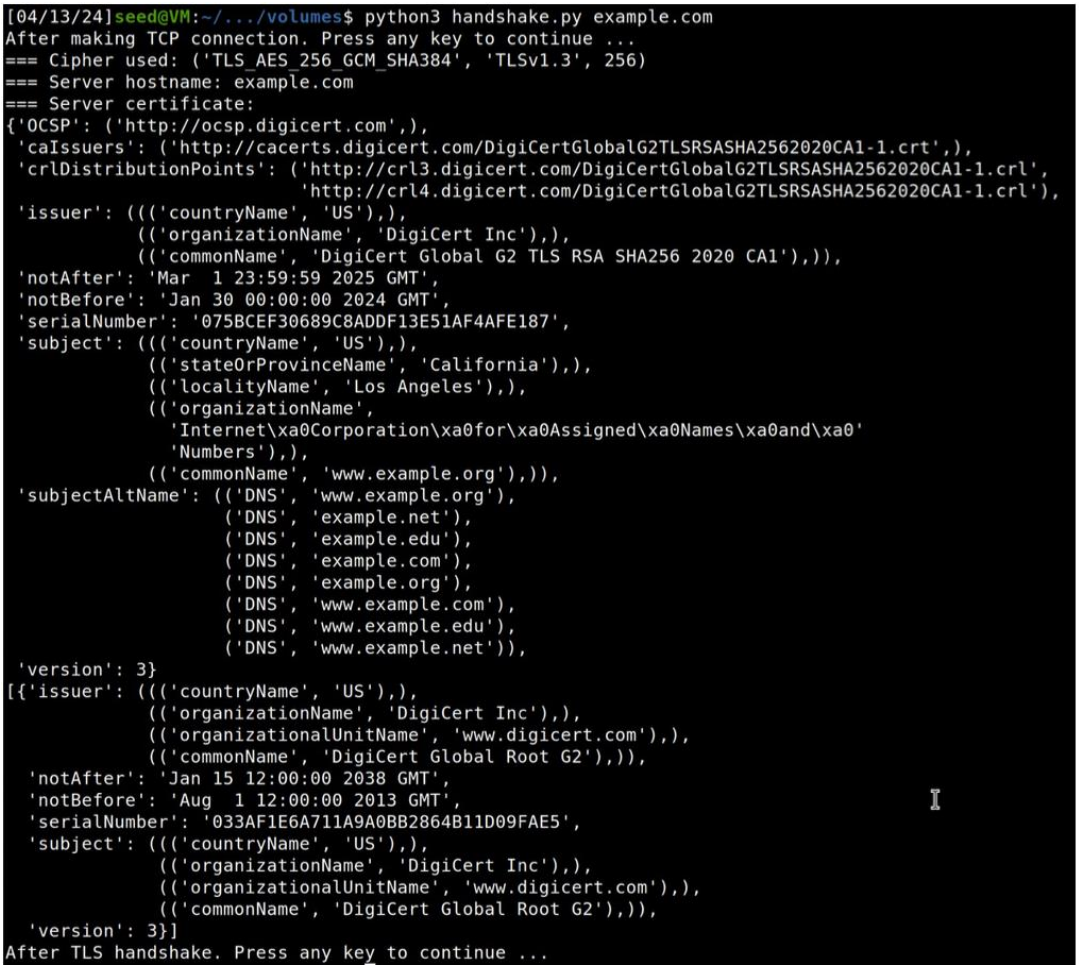
Task 1: TLS Client

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

Task 1.1a: TLS Handshake

In this task, we will incrementally build a simple TLS client program.

Using the handshake.py program to initiate the TLS handshake with example.com:

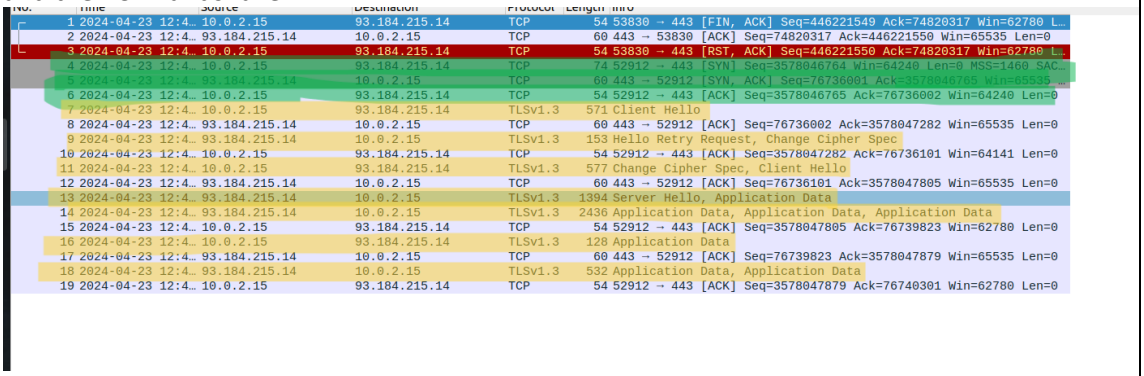


- What is the cipher used between the client and the server?
  The cipher being used is AES 256-bit in Galois/Counter Mode and SHA384 as the hashing algorithm.

- Please print out the server certificate in the program.

```
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('commonName', 'DigiCert Global G2 TLS RSA SHA256 2020 CA1'),)),
 'notAfter': 'Mar  1 23:59:59 2025 GMT',
 'notBefore': 'Jan 30 00:00:00 2024 GMT',
 'serialNumber': '075BCEF30689C8ADDF13E51AF4AFE187',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'Los Angeles'),),
             (('organizationName',
               'Internet\xa0Corporation\xa0for\xa0Assigned\xa0Names\xa0and\xa0'
               'Numbers'),),
             (('commonName', 'www.example.org'),)),
 'subjectAltName': (('DNS', 'www.example.org'),
                    ('DNS', 'example.net'),
                    ('DNS', 'example.edu'),
                    ('DNS', 'example.com'),
                    ('DNS', 'example.org'),
                    ('DNS', 'www.example.com'),
                    ('DNS', 'www.example.edu'),
                    ('DNS', 'www.example.net')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('organizationalUnitName', 'www.digicert.com'),),
             (('commonName', 'DigiCert Global Root G2'),)),
  'notAfter': 'Jan 15 12:00:00 2038 GMT',
  'notBefore': 'Aug  1 12:00:00 2013 GMT',
  'serialNumber': '033AF1E6A711A9A0BB2864B11D09FAE5',
  'subject': ((('countryName', 'US'),),
              (('organizationName', 'DigiCert Inc'),),
              (('organizationalUnitName', 'www.digicert.com'),),
              (('commonName', 'DigiCert Global Root G2'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
```

- Explain the purpose of /etc/ssl/certs
  This directory stores the Certificate Authority (CA) Certificates. These are important because they are used to verify the identity of the servers presenting their SSL/TLS certificates to the clients while establishing a secure encrypted connection.
- Use Wireshark to capture the network traffics during the execution of the program, and explain your observation. In particular, explain which step triggers the TCP handshake, and which step triggers the TLS handshake. Explain the relationship between the TLS handshake and the TCP handshake



The green 3 lines are the tcp handshake, whatever is rest that is a TLS packet is the TLS handshake. The tcp handshake sets up the connection between 2 entities, while the tls handshake adds an extra layer of security. And TLS handshake typically occurs after the tcp handshake was finished and the tcp connection was established, mainly to secure the data transmitted.

Task 1.b: CA's Certificate

In the previous task, we use the certificates in the /etc/ssl/certs folder to verify server's certificates. In this task, we will create our own certificate folder, and place the corresponding certificates in the folder to do the verification.

Domain 1: example.com

Seeing the output from the previous command, we know that the DigiCert Global G2 is the Certificate Authority that has signed the digital certificate of the example.com domain. Therefore we try to find that certificate file in the /etc/ssl/certs directory using the grep command:

```
[04/13/24]seed@VM:~/.../volumes$ cd /etc/ssl/certs
[04/13/24]seed@VM:.../certs$ ls -l | grep DigiCert
lrwxrwxrwx 1 root root     38 Nov 24  2020 244b5494.0 -> DigiCert_High_Assurance_EV_Root_CA.pem
lrwxrwxrwx 1 root root     27 Nov 24  2020 3513523f.0 -> DigiCert_Global_Root_CA.pem
lrwxrwxrwx 1 root root     27 Nov 24  2020 607986c7.0 -> DigiCert_Global_Root_G2.pem
lrwxrwxrwx 1 root root     28 Nov 24  2020 75d1b2ed.0 -> DigiCert_Trusted_Root_G4.pem
lrwxrwxrwx 1 root root     31 Nov 24  2020 7f3d5d1d.0 -> DigiCert_Assured_ID_Root_G3.pem
lrwxrwxrwx 1 root root     33 Feb  3 01:02 9846683b.0 -> DigiCert_TLS_ECC_P384_Root_G5.pem
lrwxrwxrwx 1 root root     31 Nov 24  2020 9d04f354.0 -> DigiCert_Assured_ID_Root_G2.pem
lrwxrwxrwx 1 root root     31 Nov 24  2020 b1159c4c.0 -> DigiCert_Assured_ID_Root_CA.pem
lrwxrwxrwx 1 root root     32 Feb  3 01:02 d52c538d.0 -> DigiCert_TLS_RSA4096_Root_G5.pem
lrwxrwxrwx 1 root root     27 Nov 24  2020 dd8e9d41.0 -> DigiCert_Global_Root_G3.pem
lrwxrwxrwx 1 root root     66 Nov 24  2020 DigiCert_Assured_ID_Root_CA.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Assured_ID_Root_CA.crt
lrwxrwxrwx 1 root root     66 Nov 24  2020 DigiCert_Assured_ID_Root_G2.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Assured_ID_Root_G2.crt
lrwxrwxrwx 1 root root     66 Nov 24  2020 DigiCert_Assured_ID_Root_G3.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Assured_ID_Root_G3.crt
lrwxrwxrwx 1 root root     62 Nov 24  2020 DigiCert_Global_Root_CA.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Global_Root_CA.crt
lrwxrwxrwx 1 root root     62 Nov 24  2020 DigiCert_Global_Root_G2.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Global_Root_G2.crt
lrwxrwxrwx 1 root root     62 Nov 24  2020 DigiCert_Global_Root_G3.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Global_Root_G3.crt
lrwxrwxrwx 1 root root     73 Nov 24  2020 DigiCert_High_Assurance_EV_Root_CA.pem -> /usr/share/ca-certificates/mozilla/DigiCert_High_Assurance_EV_Root
_CA.crt
lrwxrwxrwx 1 root root     68 Feb  3 01:02 DigiCert_TLS_ECC_P384_Root_G5.pem -> /usr/share/ca-certificates/mozilla/DigiCert_TLS_ECC_P384_Root_G5.crt
lrwxrwxrwx 1 root root     67 Feb  3 01:02 DigiCert_TLS_RSA4096_Root_G5.pem -> /usr/share/ca-certificates/mozilla/DigiCert_TLS_RSA4096_Root_G5.crt
lrwxrwxrwx 1 root root     63 Nov 24  2020 DigiCert_Trusted_Root_G4.pem -> /usr/share/ca-certificates/mozilla/DigiCert_Trusted_Root_G4.crt
```

The DigiCert_Global_Root_G2.pem file is the one that points to the DigiCert_Global_Root_G2.crt file. So, we copy the crt file into the .../volumes/client-certs/ directory and then make a symbolic out of the hash value:

```
[04/13/24]seed@VM:~/.../volumes$ cp /usr/share/ca-certificates/mozilla/DigiCert_Global_Root_G2.crt ./client-certs/
[04/13/24]seed@VM:~/.../volumes$ cd client-certs/
[04/13/24]seed@VM:~/.../client-certs$ openssl x509 -in DigiCert_Global_Root_G2.crt -noout -subject_hash
607986c7
[04/13/24]seed@VM:~/.../client-certs$ ln -s DigiCert_Global_Root_G2.crt 607986c7.0
[04/13/24]seed@VM:~/.../client-certs$ ls -l
total 8
lrwxrwxrwx 1 seed seed   27 Apr 13 10:06 607986c7.0 -> DigiCert_Global_Root_G2.crt
-rw-r--r-- 1 seed seed 1294 Apr 13 10:05 DigiCert_Global_Root_G2.crt
-rw-rw-r-- 1 seed seed  103 Jan  2  2021 README.md
[04/13/24]seed@VM:~/.../client-certs$
```

Then we edit the code in the handshake.py file and make the cadir variable point to the client-certs directory:

```
hostname = sys.argv[1]
port = 443
#cadir = '/etc/ssl/certs'
cadir = './client-certs'
```

After that, we execute the program again and supply the hostname example.com:

```
[04/13/24]seed@VM:~/.../volumes$ python3 handshake.py example.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: example.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('commonName', 'DigiCert Global G2 TLS RSA SHA256 2020 CA1'),)),
 'notAfter': 'Mar  1 23:59:59 2025 GMT',
 'notBefore': 'Jan 30 00:00:00 2024 GMT',
 'serialNumber': '075BCEF30689C8ADDF13E51AF4AFE187',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'Los Angeles'),),
             (('organizationName',
               'Internet\xa0Corporation\xa0for\xa0Assigned\xa0Names\xa0and\xa0'
               'Numbers'),),
             (('commonName', 'www.example.org'),)),
 'subjectAltName': (('DNS', 'www.example.org'),
                    ('DNS', 'example.net'),
                    ('DNS', 'example.edu'),
                    ('DNS', 'example.com'),
                    ('DNS', 'example.org'),
                    ('DNS', 'www.example.com'),
                    ('DNS', 'www.example.edu'),
                    ('DNS', 'www.example.net')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('organizationalUnitName', 'www.digicert.com'),),
             (('commonName', 'DigiCert Global Root G2'),)),
  'notAfter': 'Jan 15 12:00:00 2038 GMT',
  'notBefore': 'Aug  1 12:00:00 2013 GMT',
  'serialNumber': '033AF1E6A711A9A0BB2864B11D09FAE5',
  'subject': ((('countryName', 'US'),),
              (('organizationName', 'DigiCert Inc'),),
              (('organizationalUnitName', 'www.digicert.com'),),
              (('commonName', 'DigiCert Global Root G2'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
```

Domain 2: qnb.com

For the second domain we have to first find out which certificate authority has signed the digital certificate for it. So we repeat task 1.1a, but before that we have change the cadir back to /etc/ssl/certs:

```
hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'
#cadir = './client-certs'
```

Then we find the certificate and copy it to the client-certs directory and create a symbolic link using the outputted hash:

```
[04/13/24]seed@VM:~/.../volumes$ cp /usr/share/ca-certificates/mozilla/ISRG_Root_X2.crt ./client-certs/
[04/13/24]seed@VM:~/.../volumes$ cd client-certs/
[04/13/24]seed@VM:~/.../client-certs$ openssl x509 -in ISRG_Root_X2.crt -noout -subject_hash
0b9bc432
[04/13/24]seed@VM:~/.../client-certs$ ln -s ISRG_Root_X2.crt 0b9bc432.0
[04/13/24]seed@VM:~/.../client-certs$ ls -l
total 12
lrwxrwxrwx 1 seed seed   16 Apr 13 10:49 0b9bc432.0 -> ISRG_Root_X2.crt
lrwxrwxrwx 1 seed seed   27 Apr 13 10:06 607986c7.0 -> DigiCert_Global_Root_G2.crt
-rw-r--r-- 1 seed seed 1294 Apr 13 10:05 DigiCert_Global_Root_G2.crt
-rw-r--r-- 1 seed seed  790 Apr 13 10:47 ISRG_Root_X2.crt
-rw-rw-r-- 1 seed seed  103 Jan  2  2021 README.md
[04/13/24]seed@VM:~/.../client-certs$
```

After that, change the cadir variable in the program back to ./client-certs and then execute the program again with the qnb.com domain:

```
[04/13/24]seed@VM:~/.../volumes$ python3 handshake.py qnb.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: qnb.com
=== Server certificate:
{'OCSP': ('http://e1.o.lencr.org',),
 'caIssuers': ('http://e1.i.lencr.org/',),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', "Let's Encrypt"),),
            (('commonName', 'E1'),)),
 'notAfter': 'Jul  3 07:09:59 2024 GMT',
 'notBefore': 'Apr  4 07:10:00 2024 GMT',
 'serialNumber': '040351C18BF486481ECAF69E20620453460B',
 'subject': ((('commonName', 'qnb.com'),),),
 'subjectAltName': (('DNS', '*.ob.qnb.com'), ('DNS', 'qnb.com')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'Internet Security Research Group'),),
             (('commonName', 'ISRG Root X2'),)),
  'notAfter': 'Sep 17 16:00:00 2040 GMT',
  'notBefore': 'Sep  4 00:00:00 2020 GMT',
  'serialNumber': '41D29DD172EAEEA780C12C6CE92F8752',
  'subject': ((('countryName', 'US'),),
              (('organizationName', 'Internet Security Research Group'),),
              (('commonName', 'ISRG Root X2'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
```

Task 1.c: Experiment with the hostname check

Getting the IP address of the www.example.com domain using the dig command:

```
[04/13/24]seed@VM:~/.../volumes$ dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59962
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        2425    IN      A       93.184.216.34

;; Query time: 312 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Sat Apr 13 12:21:28 EDT 2024
;; MSG SIZE  rcvd: 60
```

Editing the /etc/hosts file and adding an IP - hostname entry making the www.example2020.com domain point to the same IP address of the www.example.com domain:

```
93.184.216.34    www.example2020.com
```

Then, we change the context.check_hostname to False:

```
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False
```

After that, we use the execute the handshake.py program with the www.example2020.com domain:

```
[04/13/24]seed@VM:~/.../volumes$ python3 handshake.py www.example2020.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: www.example2020.com
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'DigiCert Inc'),),
            (('commonName', 'DigiCert Global G2 TLS RSA SHA256 2020 CA1'),)),
 'notAfter': 'Mar  1 23:59:59 2025 GMT',
 'notBefore': 'Jan 30 00:00:00 2024 GMT',
 'serialNumber': '075BCEF30689C8ADDF13E51AF4AFE187',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'Los Angeles'),),
             (('organizationName',
               'Internet\xa0Corporation\xa0for\xa0Assigned\xa0Names\xa0and\xa0'
               'Numbers'),),
             (('commonName', 'www.example.org'),)),
 'subjectAltName': (('DNS', 'www.example.org'),
                    ('DNS', 'example.net'),
                    ('DNS', 'example.edu'),
                    ('DNS', 'example.com'),
                    ('DNS', 'example.org'),
                    ('DNS', 'www.example.com'),
                    ('DNS', 'www.example.edu'),
                    ('DNS', 'www.example.net')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('organizationalUnitName', 'www.digicert.com'),),
             (('commonName', 'DigiCert Global Root G2'),)),
  'notAfter': 'Jan 15 12:00:00 2038 GMT',
  'notBefore': 'Aug  1 12:00:00 2013 GMT',
  'serialNumber': '033AF1E6A711A9A0BB2864B11D09FAE5',
  'subject': ((('countryName', 'US'),),
              (('organizationName', 'DigiCert Inc'),),
              (('organizationalUnitName', 'www.digicert.com'),),
              (('commonName', 'DigiCert Global Root G2'),)),
  'version': 3}]
After TLS handshake. Press any key to continue ...
```

The importance of hostname check: The hostname check is important because if someone was able to do a DNS cache poisoning attack on our system, they would be able to map a malicious IP address to a legitimate-looking domain and be able to still not raise the Warning pop-up window for the user. However, if the context.check_hostname is kept as True then, even if someone was able to do a DNS cache poisoning attack on our system, we would still not establish the handshake since the hostname would not match the hostname in the certificate:

```
[04/13/24]seed@VM:~/.../volumes$ python3 handshake.py www.example2020.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake.py", line 29, in <module>
    ssock.do_handshake()   # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1338, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: Hostna
me mismatch, certificate is not valid for 'www.example2020.com'. (_ssl.c:1131)
```

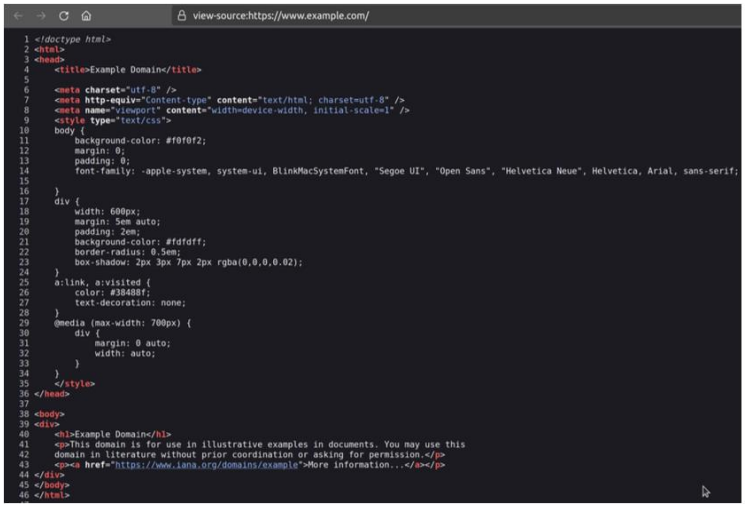
Task 1.d: Sending and getting data

In this task, we will send data to the server and get its response. Since we choose to use HTTPS servers, we need to send HTTP requests to the server; otherwise, the server will not understand our request.

After adding the code given in the lab sheet to the handshake.py program, we execute it with the domain www.example.com to see the following output:

```
After TLS handshake. Press any key to continue ...
[b'HTTP/1.0 200 OK',
 b'Age: 548235',
 b'Cache-Control: max-age=604800',
 b'Content-Type: text/html; charset=UTF-8',
 b'Date: Sat, 13 Apr 2024 18:18:34 GMT',
 b'Etag: "3147526947+ident"',
 b'Expires: Sat, 20 Apr 2024 18:18:34 GMT',
 b'Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT',
 b'Server: ECS (bsb/27E0)',
 b'Vary: Accept-Encoding',
 b'X-Cache: HIT',
 b'Content-Length: 1256',
 b'Connection: close',
 b'',
 b'']
[b'<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    '
 b' <meta charset="utf-8" />\n    <meta http-equiv="Content-type" content="t'
 b'ext/html; charset=utf-8" />\n    <meta name="viewport" content="width=dev'
 b'ice-width, initial-scale=1" />\n    <style type="text/css">\n    body {\n  '
 b'      background-color: #f0f0f2;\n        margin: 0;\n        padding: 0;\n'
 b'        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI'
 b'", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;\n    '
 b'}\n    div {\n        width: 600px;\n        margin: 5em auto;\n    '
 b'    padding: 2em;\n        background-color: #fdfdff;\n        border-rad'
 b'ius: 0.5em;\n        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n    }\n'
 b'    a:link, a:visited {\n        color: #38488f;\n        text-decoration:'
 b' none;\n    }\n    @media (max-width: 700px) {\n        div {\n          m'
 b'argin: 0 auto;\n            width: auto;\n        }\n    }\n    </style>    '
 b'\n</head>\n\n<body>\n<div>\n    <h1>Example Domain</h1>\n    <p>This domai'
 b'n is for use in illustrative examples in documents. You may use this\n    '
 b' domain in literature without prior coordination or asking for permission.</'
 b'p>\n    <p><a href="https://www.iana.org/domains/example">More informatio'
 b'n...</a></p>\n</div>\n</body>\n</html>\n']
```

(1) Please add the data sending/receiving code to your client program, and report your observation: The first array we see has the HTTP header fields and values. So for example, we see that the date we sent the HTTP request is 13th April… The second array contains the HTML of the webpage that we fetched using the GET request. We can verify this by taking a look at the HTML of the webpage:



Then, we change the code and instead of performing a GET request for the root directory of the webserver (/), we perform a GET request for the path of the image we want to see:

```
# Send HTTP Request to Server
request = b"GET /system76/96dd06bf-60e2-4ada-bf2c-af6e8ad09830_product-lemp-600x600-scaled.png HTTP/-
1.0\r\nHost: " + \
hostname.encode('utf-8') + b"\r\n\r\n"
ssock.sendall(request)
# Read HTTP Response from Server
response = ssock.recv(2048)
while response:
    pprint.pprint(response.split(b"\r\n"))
    response = ssock.recv(2048)
```

After executing this updated program, we can see that we fetch the image from the corresponding webserver (images.prismic.io in our case) which is displayed in hexadecimal here:

```
After TLS handshake. Press any key to continue ...
[b'HTTP/1.1 200 OK',
 b'Connection: close',
 b'Content-Length: 145053',
 b'last-modified: Wed, 17 Jan 2024 06:42:59 GMT',
 b'x-imgix-id: 802f18ec22d36092aee6dabbe5d6de64e7e48c64',
 b'cache-control: public, max-age=315360000',
 b'Server: Google Frontend',
 b'Date: Sat, 13 Apr 2024 18:19:48 GMT',
 b'Age: 389',
 b'Accept-Ranges: bytes',
 b'Content-Type: image/png',
 b'Access-Control-Allow-Origin: *',
 b'Timing-Allow-Origin: *',
 b'Cross-Origin-Resource-Policy: cross-origin',
 b'X-Content-Type-Options: nosniff',
 b'X-Served-By: cache-sjc1000124-SJC, cache-ams12731-AMS',
 b'X-Cache: HIT, MISS',
 b'',
 b'']
[b'\x89PNG',
 b'\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02X\x00\x00\x02X\x08\x06\x00\x00\x00\xbe'
 b'f\x98\xdc\x00\x00\x00\tpHYs\x00\x00\x0b\x13\x00\x00\x0b\x13\x01'
 b'\x00\x9a\x9c\x18\x00\x00\x00\x01sRGB\x00\xae\xce\x1c\xe9\x00\x00\x00\x04gAM'
 b'A\x00\x00\xb1\x8f\x0b\xfca\x05\x00\x0262IDATx\x01\xec\xbd\x07\x80nYU&\xba'
 b'\xd6_U7\x87\xbe}\xbb\x9b\xce\xd0@\x93\x9a\x8cJ\x1aA\x0111*\xb4\x01'
 b'\xf1=\x1f\x8acB\x1d}"\xe0<\x15\x04\x14uF\xc4\x80c\x18\xb3\xce\xf8D'
 b'\xc5\x80:\x03\x06rl$uCw\x03\x9d\x13\x9dn\xe8\x1b*\xfdg\xbd\xb3\xc3'
 b'J\xfb\x9c\xbf\xaa:`\xf5\x9bZ_w\xdd\xff\xff\xcf\xd9y\xef\xb3\xd7\xb7\xd7Z'
 b'{\x1f\x80@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81'
 b'@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10'
 b'\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ '
 b'\x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@'
 b' \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08'
 b'\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10'
 b'\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ '
 b'\x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@'
 b' \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08'
 b'\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10'
 b'\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ \x10\x08\x04\x02\x81@ '
```

Task 2: TLS Server

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

Task 2.a: Implement a Simple TLS Server

In this task, we will implement a simple TLS Server.

First we need to create the certificates needed for the CA and server by using these commands used in Lab 4, which is the PKI Lab

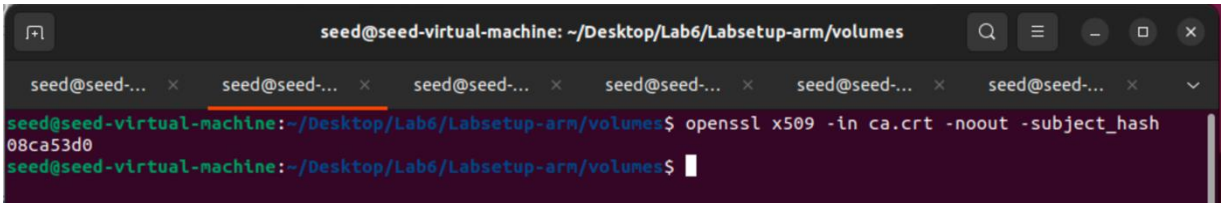
```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \
        -keyout ca.key -out ca.crt



openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \

        -keyout ca.key -out ca.crt  \

        -subj "/CN=www.lance2024.com/O=Model CA LTD./C=US" \

-passout pass:dees



openssl ca -config myCA_openssl.cnf -policy policy_anything \

        -md sha256 -days 3650 \

        -in mycert.csr -out mycert.crt -batch \

        -cert ca.crt -keyfile ca.key
```

Afterwhich, add the IP address, 10.9.0.43, of the Domain and its name to /etc/hosts

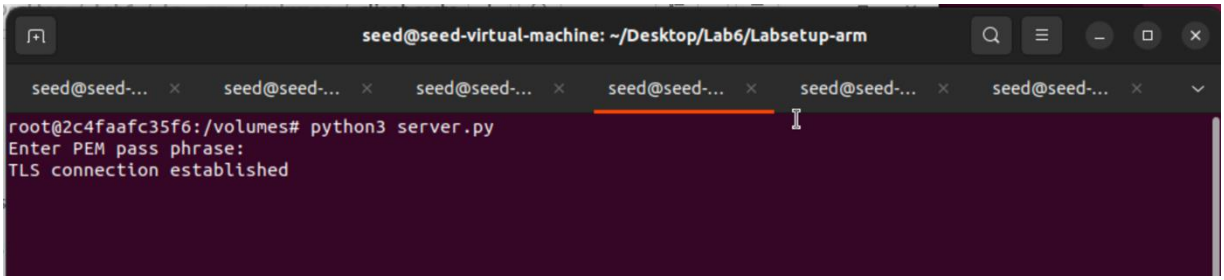
Open hosts /etc

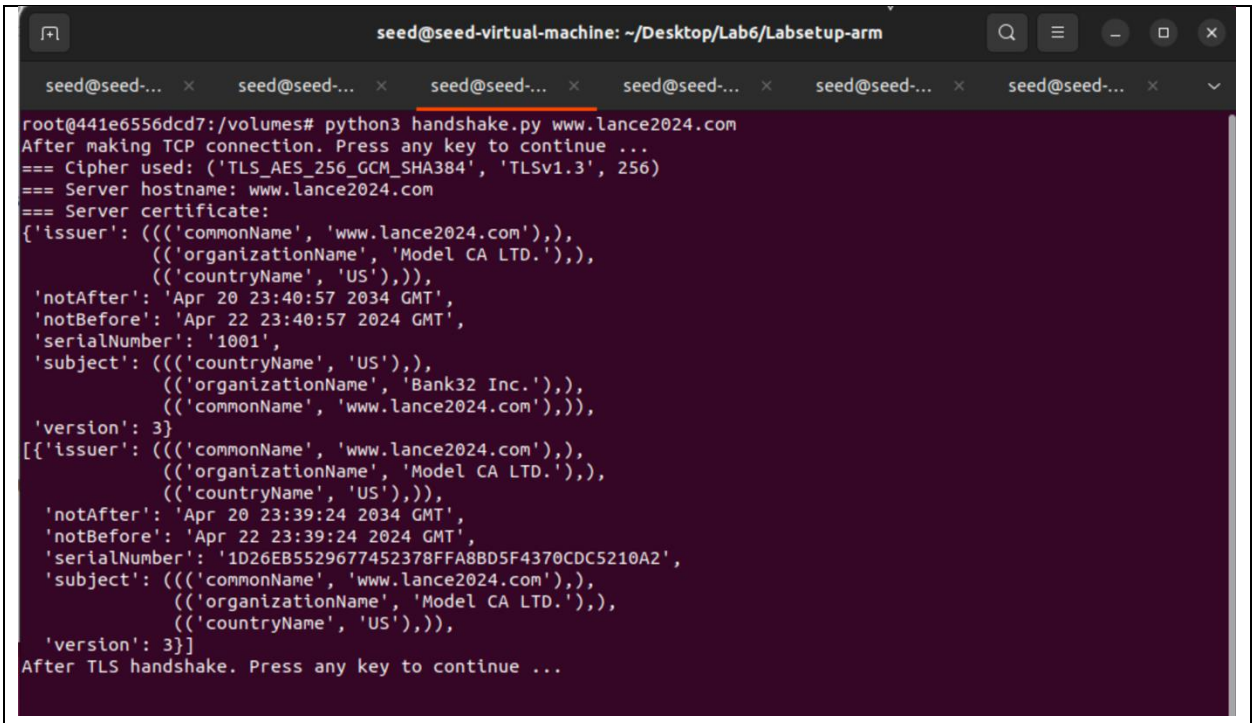
```
1 127.0.0.1       localhost
2 127.0.1.1       seed-virtual-machine
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1     ip6-localhost ip6-loopback
6 fe00::0 ip6-localnet
7 ff00::0 ip6-mcastprefix
8 ff02::1 ip6-allnodes
9 ff02::2 ip6-allrouters
10
11 # For DNS Rebinding Lab
12 192.168.60.80   www.seedIoT32.com
13
14 # For SQL Injection Lab
15 10.9.0.5        www.SeedLabSQLInjection.com
16
17 # For XSS Lab
18 10.9.0.5        www.xsslabelgg.com
19 10.9.0.5        www.example32a.com
20 10.9.0.5        www.example32b.com
21 10.9.0.5        www.example32c.com
22 10.9.0.5        www.example60.com
23 10.9.0.5        www.example70.com
24
25 # For CSRF Lab
26 10.9.0.5        www.csrflabelgg.com
27 10.9.0.5        www.csrflab-defense.com
28 10.9.0.105      www.csrflab-attacker.com
29
30 # For Shellshock Lab
31 #10.9.0.80      www.seedlab-shellshock.com
32 10.9.0.43       www.lance2024.com
```

Use the command in task 1 to generate the hash value of the certificate

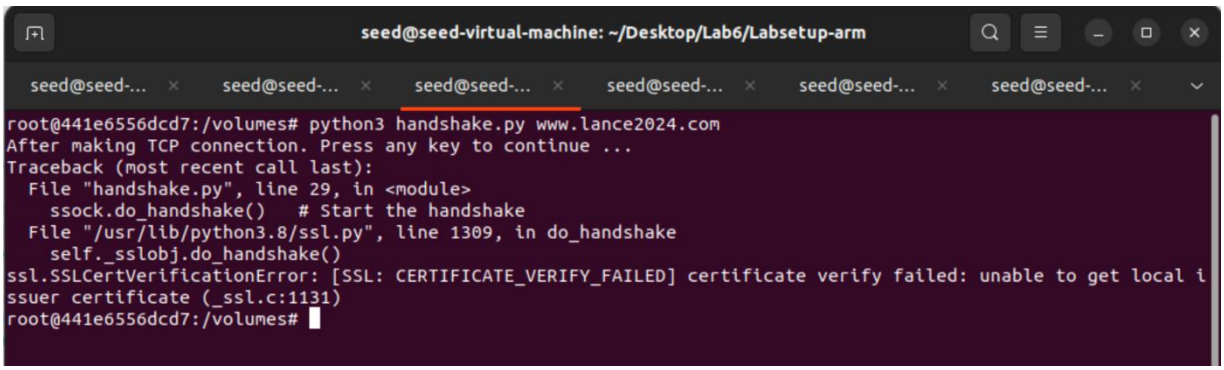


Then, we will run the Server.py first then the handshake



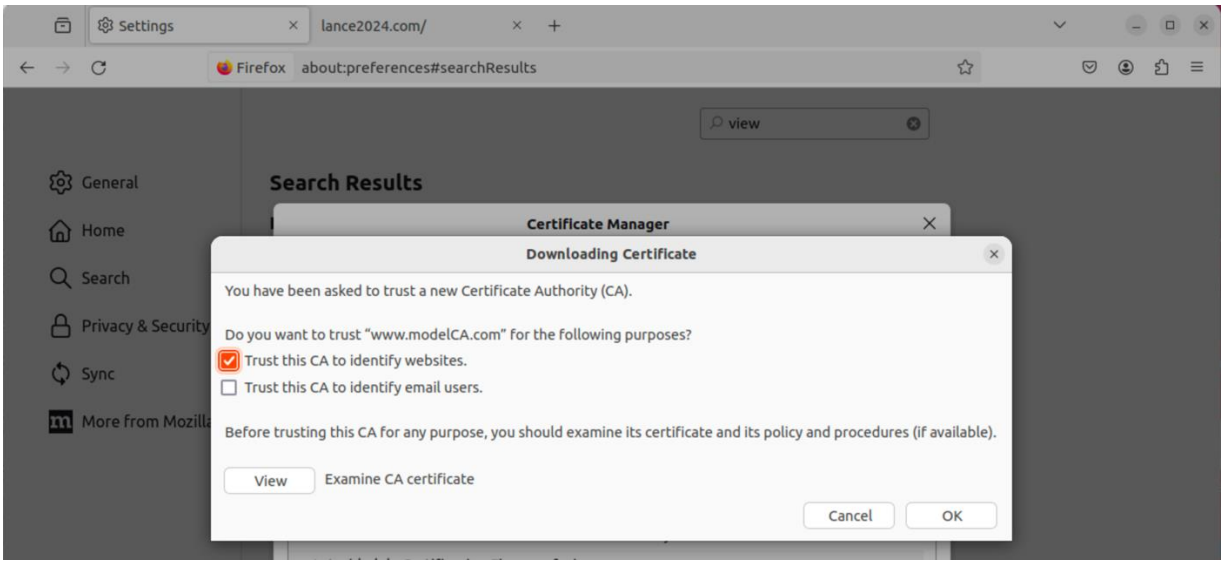


As soon as I changed cadir to /etc/ssl/certs. It will not locate the certificate

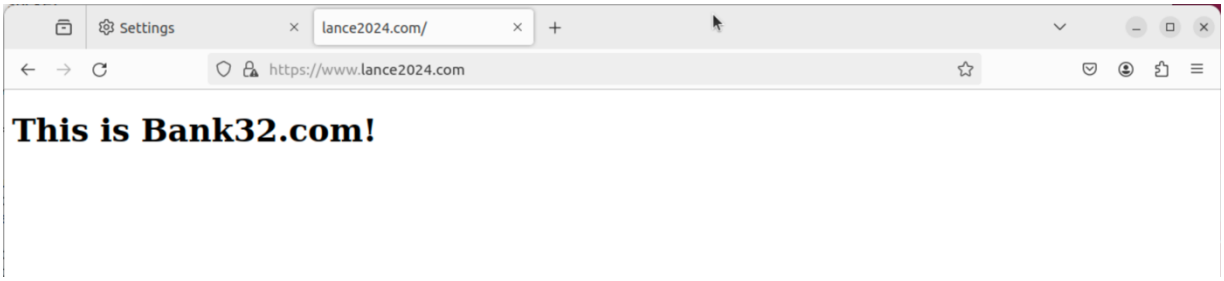


Task 2.b: Testing the Server Program Using Browsers

To access the website, server.py must be running in the background, afterwhich we will upload the certificate: enter about:preferences#privacy > View Certificates > Authorities tab > Import CA certificate > Check "Trust this CA to identify websites".

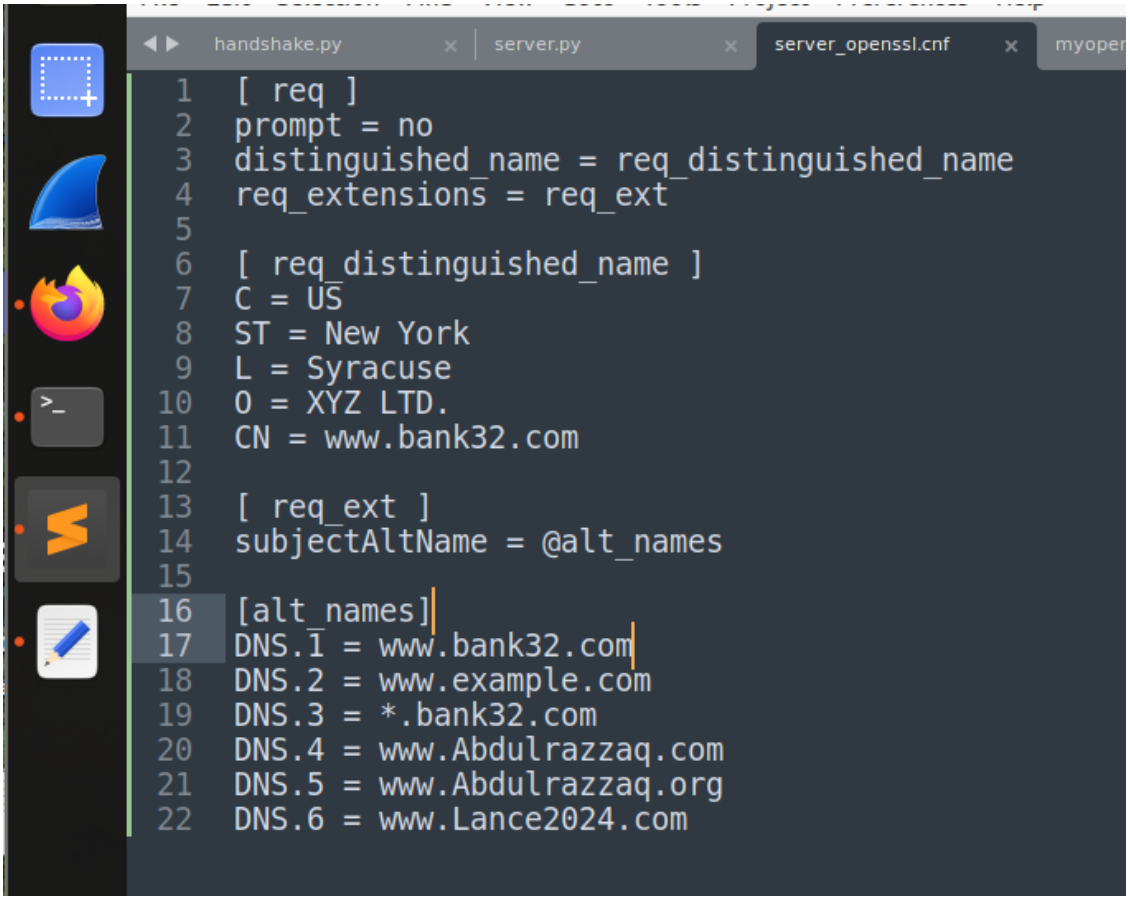


Enter the name of the website.

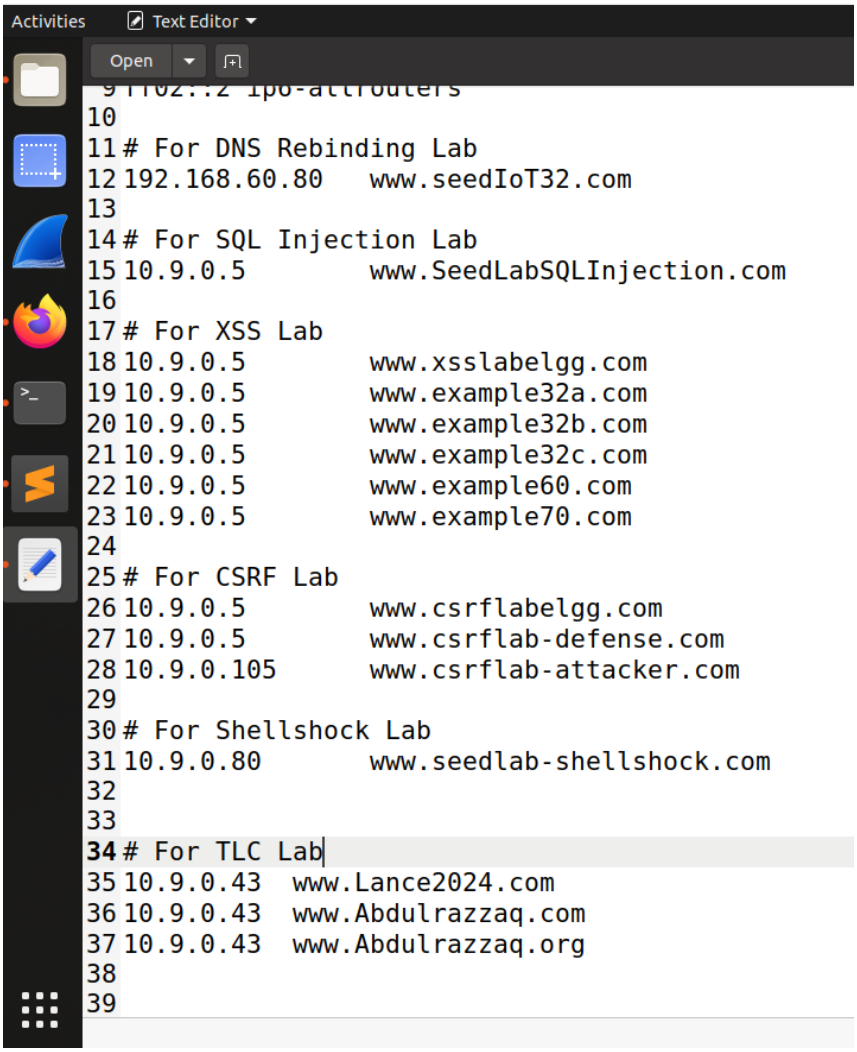


Task 2.c:

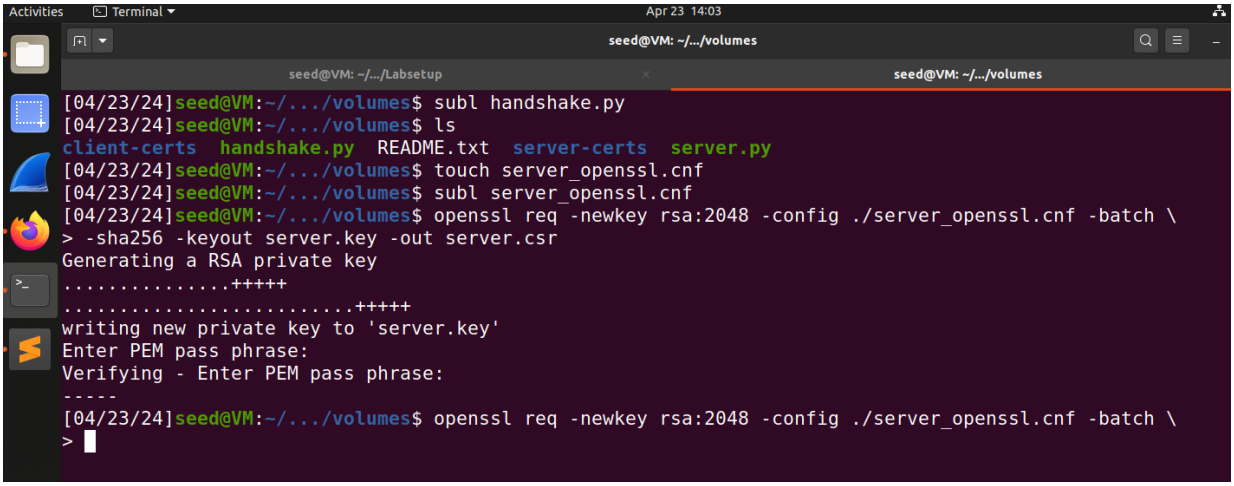
configure server.openssl.cnf as follows:
Adding the alternative names such as (www.Abdulrazzaq.org):



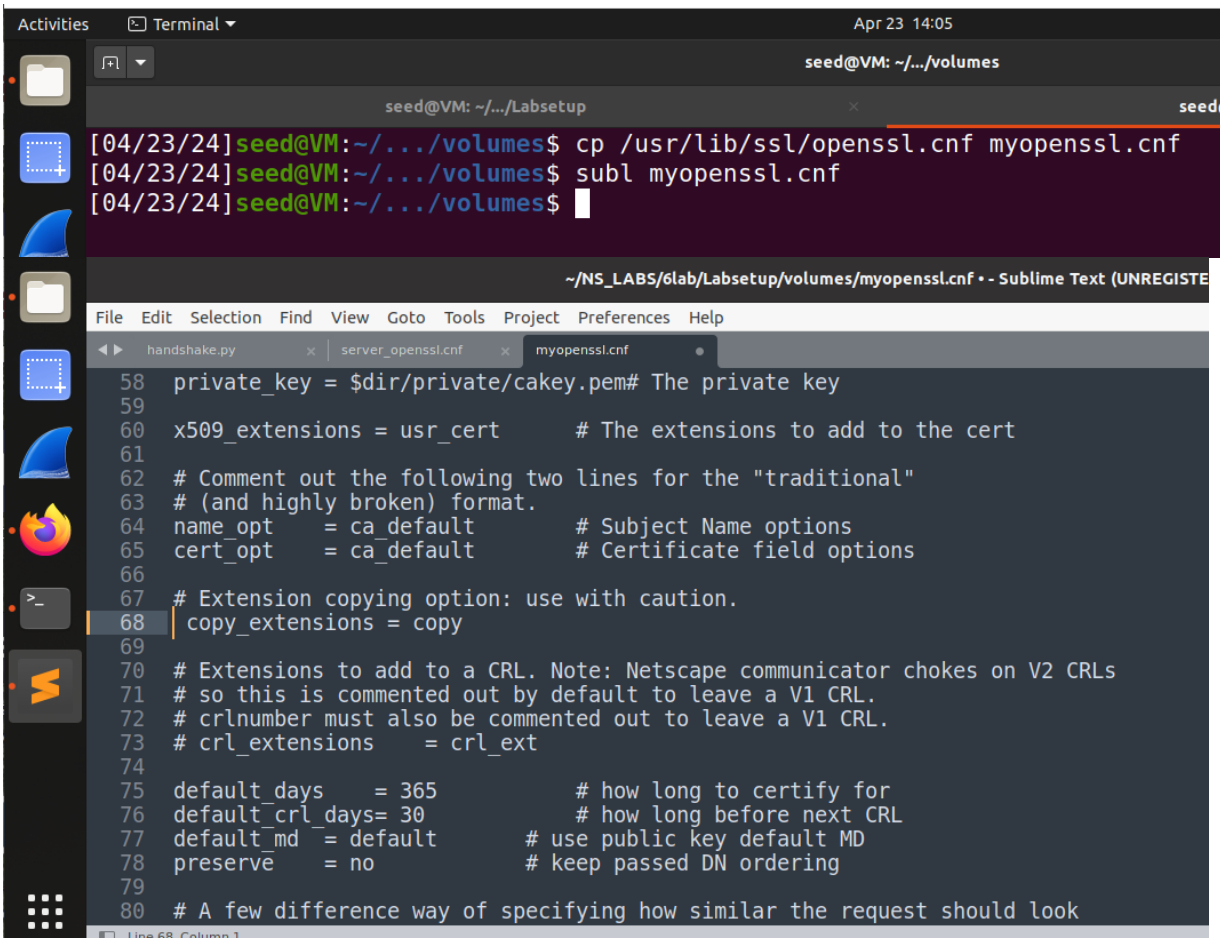
Here, we are editing the hosts file to mimic a DNS cache poisoning attack:



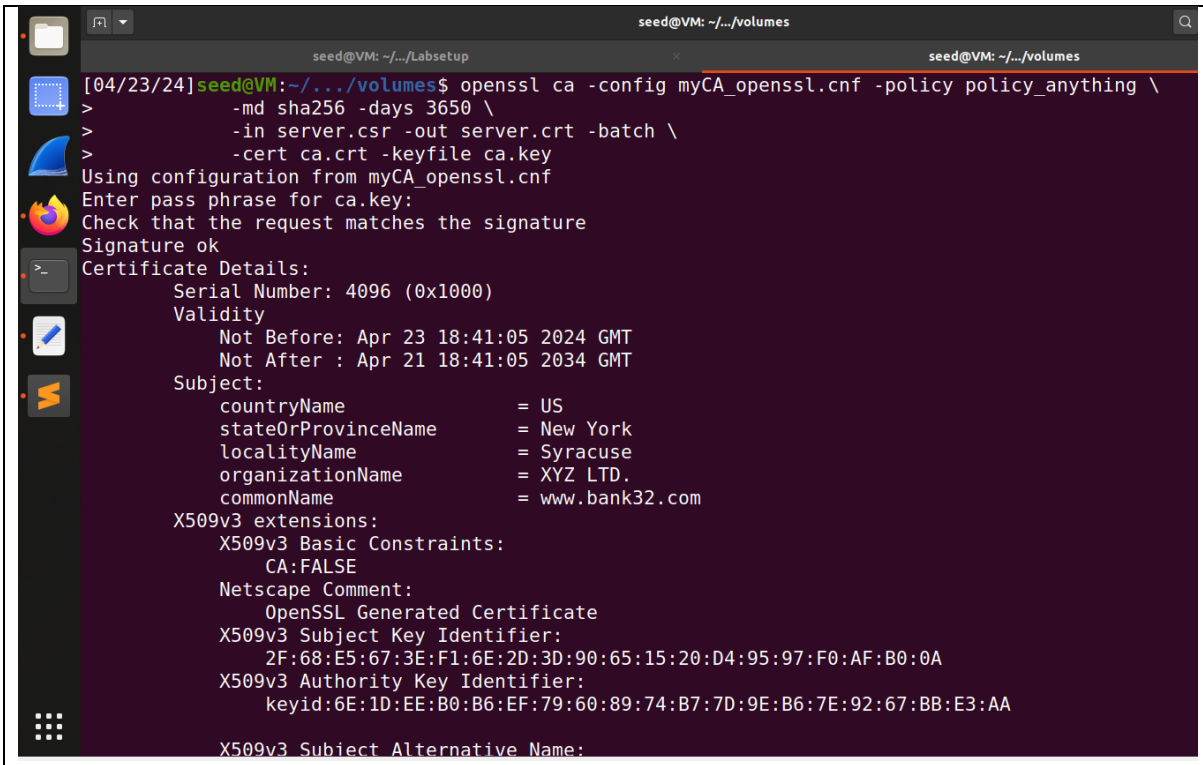
Here we are generating the server's certificate signing request, and then we generate the servers certificate file as well:



Here we are copying the openssl.cnf file in the current directory and uncommenting out the part of the copy_extensions:



Here we use generate the certificate for the server using the previously generated certificate signing request:

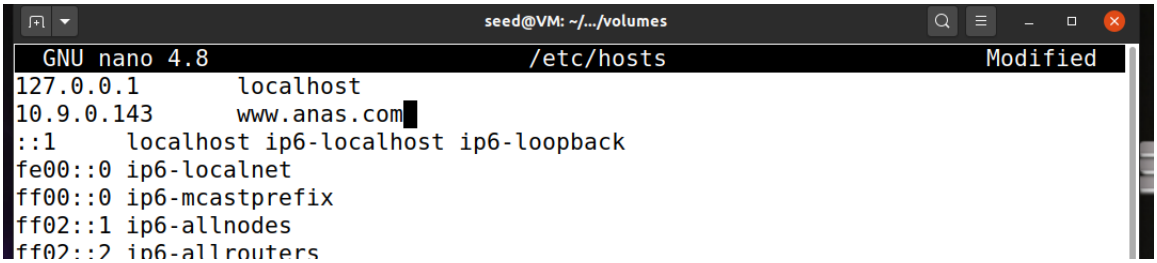


We can see that alternative names have been added to the generated certificate

Task 3: A Simple HTTPS Proxy

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

I used the nano /etc/hosts command to add my web address on the client machine.



Server.py

Open | server.py
~/Downloads/Lab/TLS/Labsetup/volumes

```
#!/usr/bin/env python3

import socket
import ssl
import pprint

html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>This is Bank32.com!</h1></body></html>
"""

SERVER_CERT = './server-certs/mycert.crt'
SERVER_PRIVATE = './server-certs/mycert.key'


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 4433))
sock.listen(5)

while True:
    newsock, fromaddr = sock.accept()
    try:
        ssock = context.wrap_socket(newsock, server_side=True)
        print("TLS connection established")
        data = ssock.recv(1024)              # Read data over TLS
        pprint.pprint("Request: {}".format(data))
        ssock.sendall(html.encode('utf-8'))  # Send data over TLS

        ssock.shutdown(socket.SHUT_RDWR)     # Close the TLS connection
        ssock.close()

    except Exception:
        print("TLS connection fails")
        continue
```

code:

```
#!/usr/bin/env python3

import socket
import ssl
import pprint

html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<!DOCTYPE html><html><body><h1>This is anas.com!</h1></body></html>"""

# SERVER_CERT = './server-certs/mycert.crt'
# SERVER_PRIVATE = './server-certs/mycert.key'
SERVER_CERT = './server-certs/mycert_multiple.crt'
SERVER_PRIVATE = './server-certs/mycert_multiple.key'


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)     # For Ubuntu 16.04 VM
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 443))
sock.listen(5)

while True:
  newsock, fromaddr = sock.accept()
  try:
    ssock = context.wrap_socket(newsock, server_side=True)
    print("TLS connection established")
    data = ssock.recv(1024)           # Read data over TLS
    pprint.pprint("Request: {}".format(data))
    ssock.sendall(html.encode('utf-8'))  # Send data over TLS

    ssock.shutdown(socket.SHUT_RDWR)    # Close the TLS connection
    ssock.close()

  except Exception:
    print("TLS connection fails")
    continue
```

Proxy.py

Open proxy.py ~/Downloads/Lab/TLS/Labsetup/volumes

```
#!/usr/bin/env python3

import socket
import ssl
import pprint
import threading

def process_request(ssock_for_browser):
    hostname = "www.anas.com"

    # Make a connection to the real server
    context_client = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
    # context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM

    cadir = './client-certs'
    context_client.load_verify_locations(capath=cadir)
    context_client.verify_mode = ssl.CERT_REQUIRED
    context_client.check_hostname = True
    sock_for_server = socket.create_connection((hostname, 443))
    ssock_for_server = context_client.wrap_socket(
        sock_for_server,
        server_hostname=hostname,
        do_handshake_on_connect=False
    )

    ssock_for_server.do_handshake()
    request = ssock_for_browser.recv(2048)
    pprint.pprint("Request: {}".format(request))
    if request:
        # Forward request to server
        ssock_for_server.sendall(request)

        # Get response from server, and forward it to browser
        response = ssock_for_server.recv(2048)
        response = response.replace(b"Bank32", b"FEUP22")
        while response:
            ssock_for_browser.sendall(response) # Forward to browser
            response = ssock_for_server.recv(2048)
            response = response.replace(b"Bank32", b"FEUP22")

    ssock_for_browser.shutdown(socket.SHUT_RDWR)
    ssock_for_browser.close()

# SERVER_CERT = './server-certs/mycert.crt'
# SERVER_PRIVATE = './server-certs/mycert.key'
SERVER_CERT = './server-certs/mycert_multiple.crt'
SERVER_PRIVATE = './server-certs/mycert_multiple.key'
```

code:

#!/usr/bin/env python3

import socket
import ssl

```
import pprint
import threading

def process_request(ssock_for_browser):
   hostname = "www.anas.com"

   # Make a connection to the real server
   context_client = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
   # context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)     # For Ubuntu 16.04 VM

   cadir = './client-certs'
   context_client.load_verify_locations(capath=cadir)
   context_client.verify_mode = ssl.CERT_REQUIRED
   context_client.check_hostname = True
   sock_for_server = socket.create_connection((hostname, 443))
   ssock_for_server = context_client.wrap_socket(
      sock_for_server,
      server_hostname=hostname,
      do_handshake_on_connect=False
   )

   ssock_for_server.do_handshake()
   request = ssock_for_browser.recv(2048)
   pprint.pprint("Request: {}".format(request))
   if request:
      # Forward request to server
      ssock_for_server.sendall(request)

      # Get response from server, and forward it to browser
      response = ssock_for_server.recv(2048)
      response = response.replace(b"Bank32", b"FEUP22")
      while response:
         ssock_for_browser.sendall(response) # Forward to browser
         response = ssock_for_server.recv(2048)
         response = response.replace(b"Bank32", b"FEUP22")

   ssock_for_browser.shutdown(socket.SHUT_RDWR)
   ssock_for_browser.close()

# SERVER_CERT = './server-certs/mycert.crt'
# SERVER_PRIVATE = './server-certs/mycert.key'
SERVER_CERT = './server-certs/mycert_multiple.crt'
SERVER_PRIVATE = './server-certs/mycert_multiple.key'
```

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)     # For Ubuntu 16.04 VM
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 443))
sock.listen(5)

while True:
   sock_for_browser, fromaddr = sock.accept()
   try:
      ssock_for_browser = context.wrap_socket(sock_for_browser, server_side=True)
      x = threading.Thread(target=process_request, args=(ssock_for_browser,))
      x.start()
      # print("TLS connection established")
      # data = ssock_for_browser.recv(1024)            # Read data over TLS
      # pprint.pprint("Request: {}".format(data))
      # ssock_for_browser.sendall(html.encode('utf-8'))  # Send data over TLS

      # ssock_for_browser.shutdown(socket.SHUT_RDWR)     # Close the TLS connection
      # ssock_for_browser.close()

   except Exception:
      print("TLS connection fails")
      continue
```

Proxy_realweb.py

Open | proxy_realweb.py
~/Downloads/Lab/TLS/Labsetup/volumes

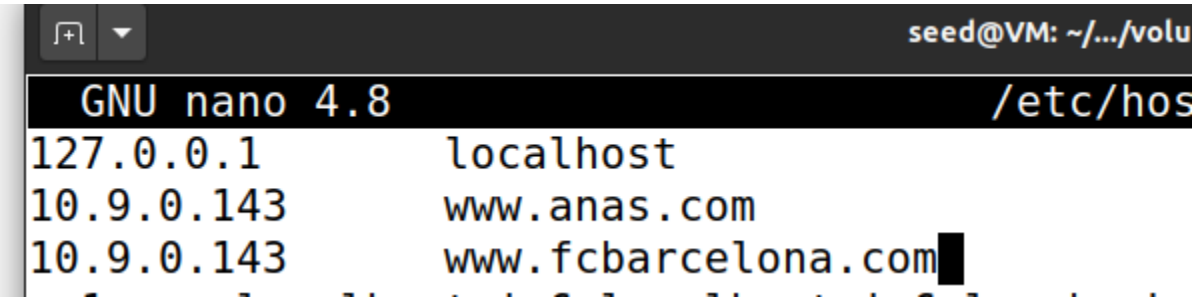
```
#!/usr/bin/env python3

import socket
import ssl
import pprint
import threading

def process_request(ssock_for_browser):
    hostname = "www.fcbarcelona.com"

    # Make a connection to the real server
    context_client = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
    # context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM

    cadir = '/etc/ssl/certs'
    context_client.load_verify_locations(capath=cadir)
    context_client.verify_mode = ssl.CERT_REQUIRED
    context_client.check_hostname = True
    sock_for_server = socket.create_connection((hostname, 443))
    ssock_for_server = context_client.wrap_socket(
        sock_for_server,
        server_hostname=hostname,
        do_handshake_on_connect=False
    )

    ssock_for_server.do_handshake()
    request = ssock_for_browser.recv(2048)
    pprint.pprint("Request: {}".format(request))
    if request:
        # Forward request to server
        ssock_for_server.sendall(request)

        # Get response from server, and forward it to browser
        response = ssock_for_server.recv(2048)
        while response:
            ssock_for_browser.sendall(response) # Forward to browser
            response = ssock_for_server.recv(2048)

    ssock_for_browser.shutdown(socket.SHUT_RDWR)
    ssock_for_browser.close()

# SERVER_CERT = './server-certs/mycert.crt'
# SERVER_PRIVATE = './server-certs/mycert.key'
SERVER_CERT = './server-certs/wayf.crt'
SERVER_PRIVATE = './server-certs/wayf.key'


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)      # For Ubuntu 16.04 VM
```

Code:

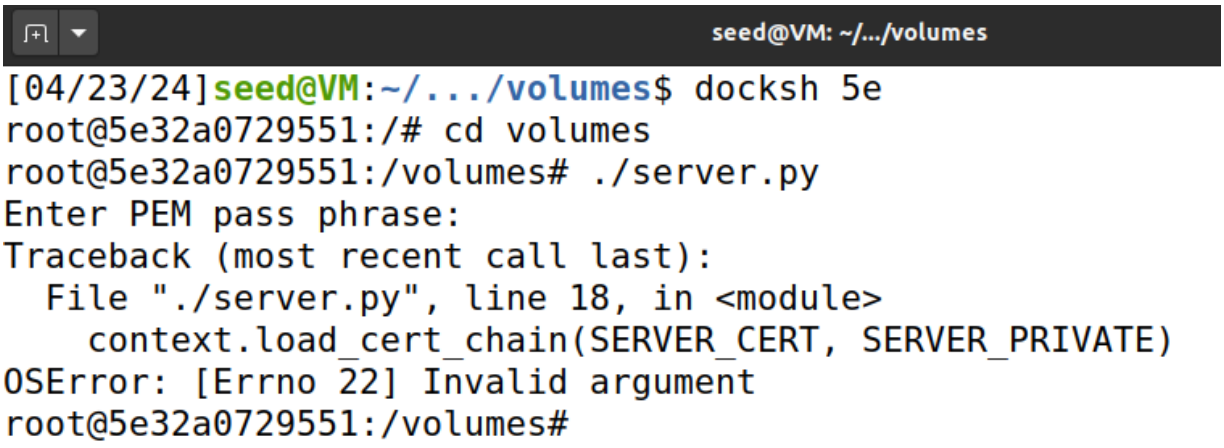
```
#!/usr/bin/env python3

import socket
import ssl
import pprint
import threading

def process_request(ssock_for_browser):
    hostname = "wayf.up.pt"

    # Make a connection to the real server
    context_client = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM
    # context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)     # For Ubuntu 16.04 VM

    cadir = '/etc/ssl/certs'
    context_client.load_verify_locations(capath=cadir)
    context_client.verify_mode = ssl.CERT_REQUIRED
    context_client.check_hostname = True
    sock_for_server = socket.create_connection((hostname, 443))
    ssock_for_server = context_client.wrap_socket(
        sock_for_server,
        server_hostname=hostname,
        do_handshake_on_connect=False
    )

    ssock_for_server.do_handshake()
    request = ssock_for_browser.recv(2048)
    pprint.pprint("Request: {}".format(request))
    if request:
        # Forward request to server
        ssock_for_server.sendall(request)

        # Get response from server, and forward it to browser
        response = ssock_for_server.recv(2048)
        while response:
            ssock_for_browser.sendall(response) # Forward to browser
            response = ssock_for_server.recv(2048)

    ssock_for_browser.shutdown(socket.SHUT_RDWR)
    ssock_for_browser.close()
```

```
# SERVER_CERT = './server-certs/mycert.crt'
# SERVER_PRIVATE = './server-certs/mycert.key'
SERVER_CERT = './server-certs/wayf.crt'
SERVER_PRIVATE = './server-certs/wayf.key'


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)  # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)     # For Ubuntu 16.04 VM
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 443))
sock.listen(5)

while True:
   sock_for_browser, fromaddr = sock.accept()
   try:
      ssock_for_browser = context.wrap_socket(sock_for_browser, server_side=True)
      x = threading.Thread(target=process_request, args=(ssock_for_browser,))
      x.start()
      # print("TLS connection established")
      # data = ssock_for_browser.recv(1024)            # Read data over TLS
      # pprint.pprint("Request: {}".format(data))
      # ssock_for_browser.sendall(html.encode('utf-8'))  # Send data over TLS

      # ssock_for_browser.shutdown(socket.SHUT_RDWR)     # Close the TLS connection
      # ssock_for_browser.close()

   except Exception:
      print("TLS connection fails")
      continue
```

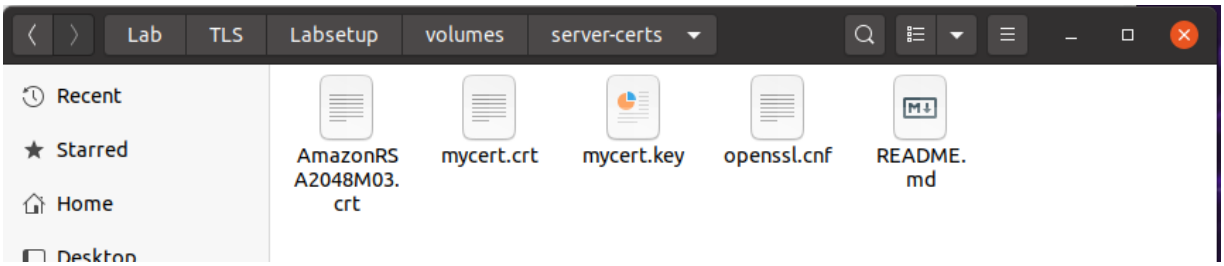
Using nano /etc/hosts on the client VM, I added www.fcbarcelona.com as our proxy server ip address 10.9.0.143, so network traffic to Barcelona website will be forward to our docker proxy server



I faced this error when running server.py code on the server VM (OS Error)



This my server-certs folder



Steps for man in the middle on real website

In the Host VM's /etc/hosts file, add the following entry:

10.9.0.143 fcbarcelona.com

In the proxy container, append the line nameserver 8.8.8.8 to the /etc/resolv.conf file.

Create a duplicate of the server_openssl.cnf file and rename it to cf_openssl.cnf.

Within the cf_openssl.cnf file:

Update the Common Name (CN) and DNS to match "codeforces.com".

Retain only one entry and remove the others.

Generate the cf.crt and cf.key files.

Update the proxy.py script as follows:

Set cadir = "/etc/ssl/certs".

Assign the Hostname variable to fcbarcelona.com.

Execute the proxy.py script to run the proxy server.

During Task 3, I was tasked with implementing a simple HTTPS proxy (mHTTPSproxy) and demonstrating a Man-In-The-Middle (MITM) attack against HTTPS servers. Our objective was to show how an attacker, assuming access to a compromised Certificate Authority (CA) private key, could intercept and manipulate HTTPS traffic.

As we embarked on the task, we encountered some challenges along the way. Despite our best efforts, we faced errors while setting up the proxy server and configuring it to intercept HTTPS traffic.

Despite these difficulties, we remained determined to complete the task to the best of our abilities. We carefully reviewed the instructions, searched for ways to fix the errors we encountered, and iterated on our implementation to overcome obstacles.

While we may not have achieved the desired outcome in its entirety, we believe that the experience gained from tackling these challenges has been invaluable. It has deepened our understanding of TLS/SSL (HTTPS) protocols, certificate management, and the intricacies of conducting MITM attacks in real-world scenarios.