



Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ENCS5322

Crypto Lab - TLS Lab

Prepared by: Diana Naseer.

Std. ID: 1210363.

Section: 1.

Instructor : Dr. Ahmad Alsadeh.

Date: 1/8/2024

Abstract:

This lab explores the fundamentals of Transport Layer Security (TLS), focusing on its role in ensuring secure communication. Participants begin by implementing a TLS client to understand the handshake process, certificate validation, and secure data exchange. The lab extends to creating a TLS server and managing server certificates, including multi-domain setups with Subject Alternative Names (SANs). Finally, a basic HTTPS proxy demonstrates vulnerabilities like Man-In-The-Middle (MITM) attacks. Through hands-on tasks, this lab provides practical insights into TLS, PKI, and HTTPS, equipping participants to secure communication in real-world applications.

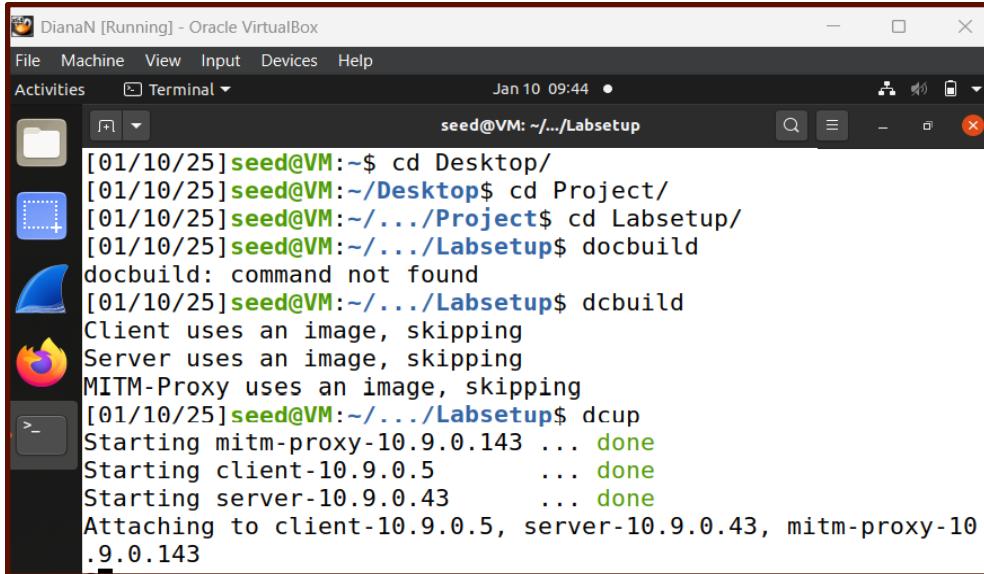
Contents

Abstract:.....	2
Task1: TLS Client	4
Task 1.a: TLS Handshake.....	6
1.a.1: Part 1 & 2 (cipher,server certificate):	7
1.a.2: Part 3 (/etc/ssl/certs.):.....	8
1.a.3 Part 4 (Wireshark):.....	9
Task 1.b: CA's Certifications.....	13
Certificate Details:	14
Results:	14
Certificate Details	16
Results	16
Key Observations	16
Advantages of Custom CA Configuration	18
Task 1.c: Experiment with the hostname check	19
Task 1.d: Sending and getting Data.....	23
Task2:TLS Server.....	28
Task 2.a. Implement a simple TLS server	31
Task 2.b. Import the CA Certificate	34
Task 2.c. Certificate with multiple names:.....	35
TASK3:A Simple HTTPS Proxy	42
Appendcies:	49

Task1: TLS Client

In this task, we will incrementally build a simple TLS client program. Through the process, students will understand the essential elements and security considerations in TLS programming. We will run this client program on the client container..

First we setup the containers:

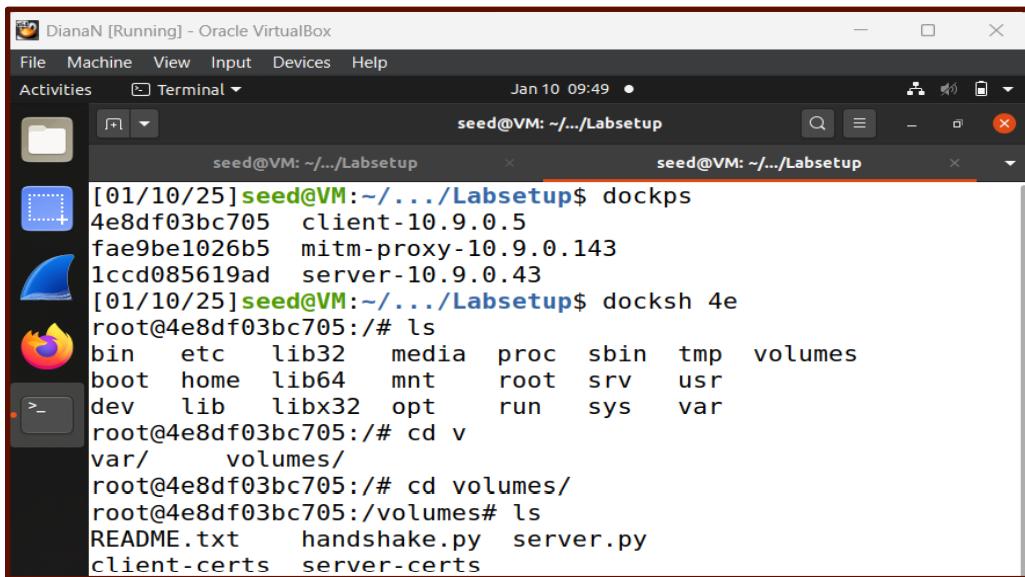


```
[01/10/25] seed@VM:~$ cd Desktop/
[01/10/25] seed@VM:~/Desktop$ cd Project/
[01/10/25] seed@VM:~/.../Project$ cd Labsetup/
[01/10/25] seed@VM:~/.../Labsetup$ docbuild
docbuild: command not found
[01/10/25] seed@VM:~/.../Labsetup$ dcbuild
Client uses an image, skipping
Server uses an image, skipping
MITM-Proxy uses an image, skipping
[01/10/25] seed@VM:~/.../Labsetup$ dcup
Starting mitm-proxy-10.9.0.143 ... done
Starting client-10.9.0.5      ... done
Starting server-10.9.0.43    ... done
Attaching to client-10.9.0.5, server-10.9.0.43, mitm-proxy-10.9.0.143
```

Now, All the containers will be running in the background.

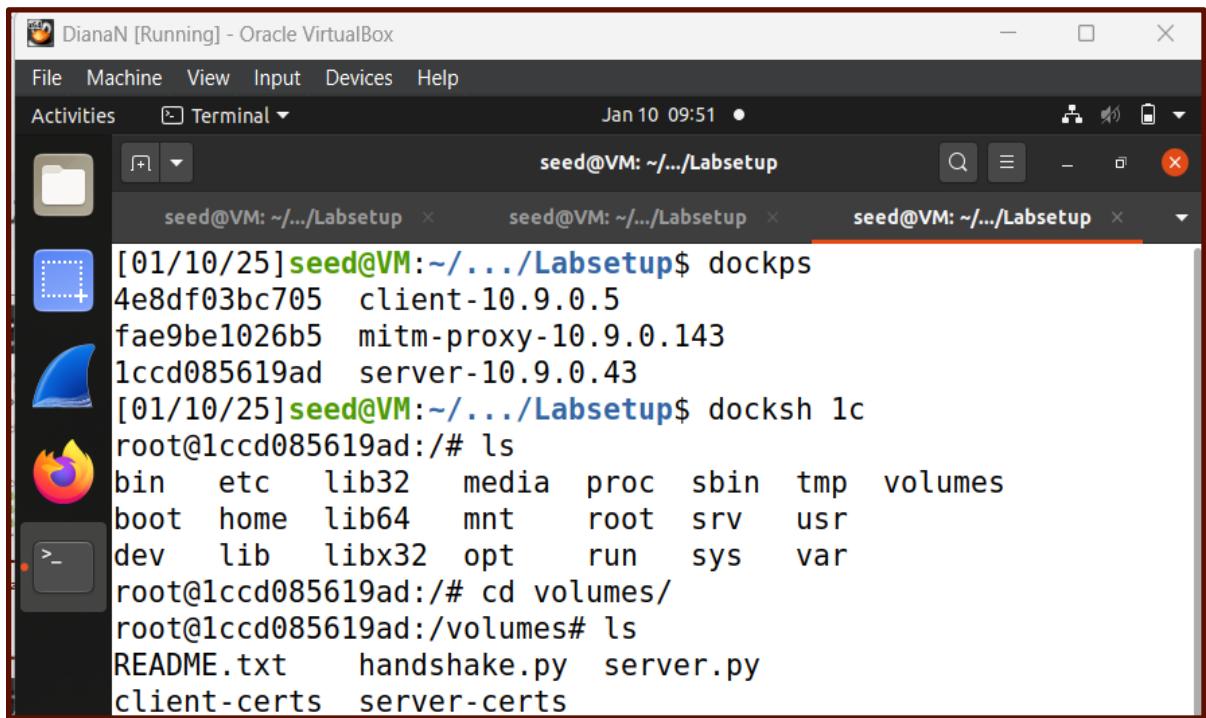
- To start a shell on that container.

For Client:



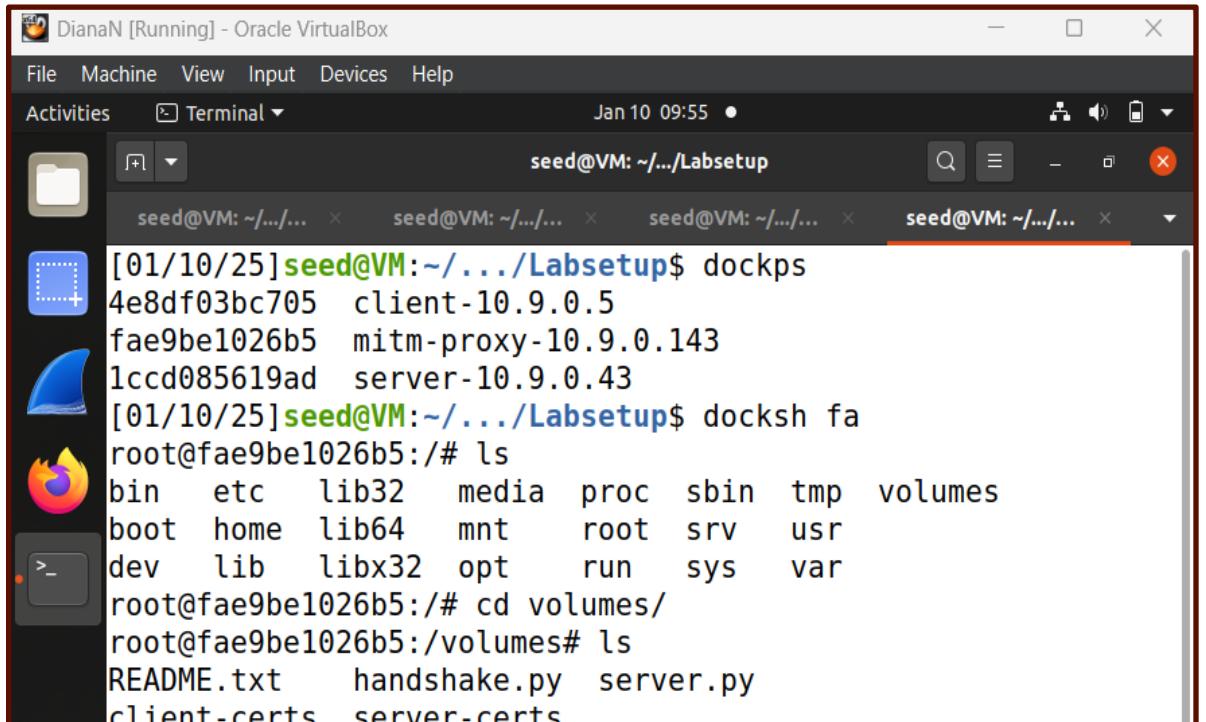
```
[01/10/25] seed@VM:~/.../Labsetup$ dockps
4e8df03bc705  client-10.9.0.5
fae9be1026b5  mitm-proxy-10.9.0.143
1ccd085619ad  server-10.9.0.43
[01/10/25] seed@VM:~/.../Labsetup$ docksh 4e
root@4e8df03bc705:/# ls
bin  etc  lib32  media  proc  sbin  tmp  volumes
boot  home  lib64  mnt  root  srv  usr
dev  lib  libx32  opt  run  sys  var
root@4e8df03bc705:/# cd v
var/  volumes/
root@4e8df03bc705:/# cd volumes/
root@4e8df03bc705:/volumes# ls
README.txt  handshake.py  server.py
client-certs  server-certs
```

For Server:



```
[01/10/25]seed@VM:~/.../Labsetup$ dockps
4e8df03bc705 client-10.9.0.5
fae9be1026b5 mitm-proxy-10.9.0.143
1ccd085619ad server-10.9.0.43
[01/10/25]seed@VM:~/.../Labsetup$ docksh 1c
root@1ccd085619ad:/# ls
bin etc lib32 media proc sbin tmp volumes
boot home lib64 mnt root srv usr
dev lib libx32 opt run sys var
root@1ccd085619ad:/# cd volumes/
root@1ccd085619ad:/volumes# ls
README.txt handshake.py server.py
client-certs server-certs
```

For Man in the middle proxy:



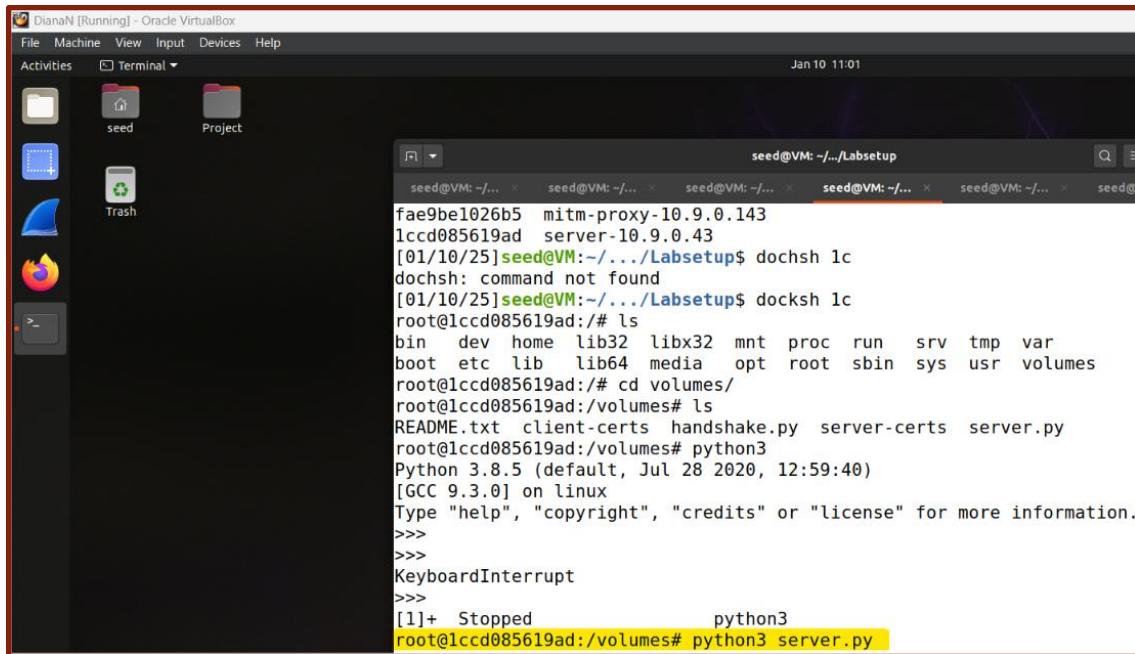
```
[01/10/25]seed@VM:~/.../Labsetup$ dockps
4e8df03bc705 client-10.9.0.5
fae9be1026b5 mitm-proxy-10.9.0.143
1ccd085619ad server-10.9.0.43
[01/10/25]seed@VM:~/.../Labsetup$ docksh fa
root@fae9be1026b5:/# ls
bin etc lib32 media proc sbin tmp volumes
boot home lib64 mnt root srv usr
dev lib libx32 opt run sys var
root@fae9be1026b5:/# cd volumes/
root@fae9be1026b5:/volumes# ls
README.txt handshake.py server.py
client-certs server-certs
```

Task 1.a: TLS Handshake

Before a client and a server can communicate securely, several things need to be set up first, including what encryption algorithm and key will be used, what MAC algorithm will be used, what algorithm should be used for the key exchange, etc. These cryptographic parameters need to be agreed upon by the client and the server. That is the primary purpose of the TLS Handshake Protocol. In this task, we focus on the TLS handshake protocol.

In server container the user attempts to execute the `server.py` script using:

```
python3 server.py
```

A screenshot of a Linux desktop environment titled "DianaN [Running] - Oracle VM VirtualBox". The desktop has a dark theme with icons for seed, Project, and Trash. A terminal window is open with the command "python3 server.py" highlighted in yellow at the bottom. The terminal output shows the user navigating to a directory, listing files, and then running the script. The script starts with "fae9be1026b5 mitm-proxy-10.9.0.143" and ends with "root@lccd085619ad:/volumes# python3 server.py".

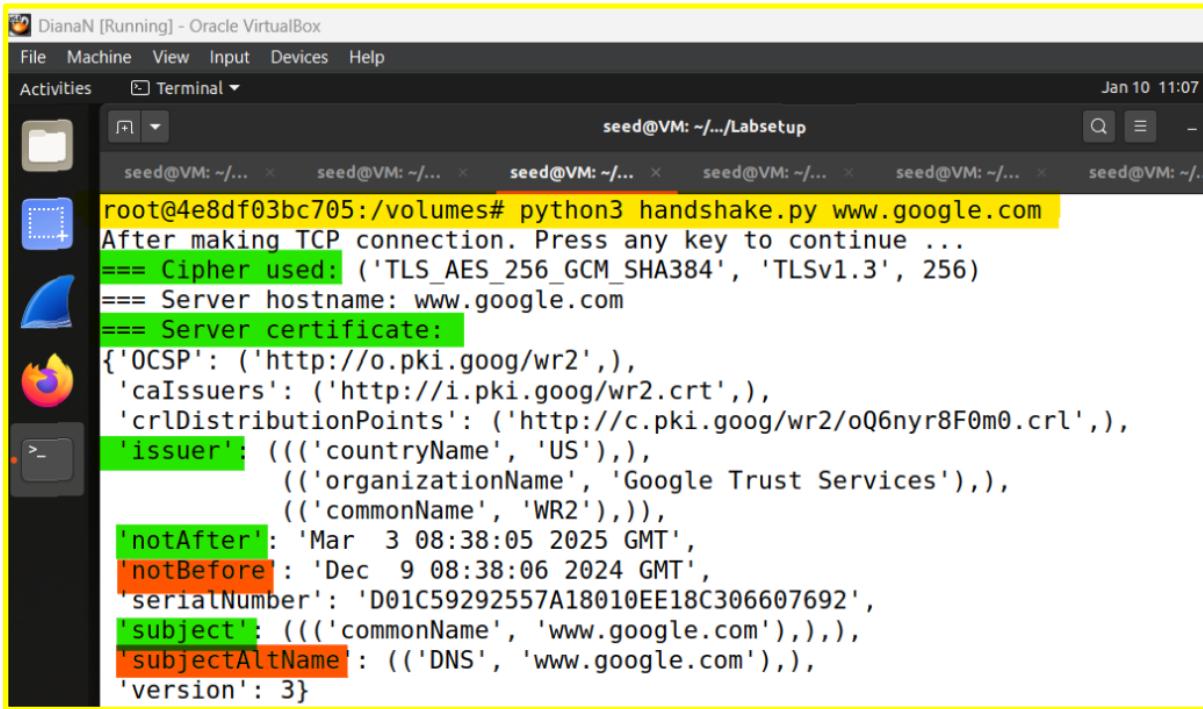
```
fae9be1026b5 mitm-proxy-10.9.0.143
lccd085619ad server-10.9.0.43
[01/10/25]seed@VM:~/.../Labsetup$ dochsh 1c
dochsh: command not found
[01/10/25]seed@VM:~/.../Labsetup$ docksh 1c
root@lccd085619ad:# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@lccd085619ad:# cd volumes/
root@lccd085619ad:/volumes# ls
README.txt client-certs handshake.py server-certs server.py
root@lccd085619ad:/volumes# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
KeyboardInterrupt
>>>
[1]+ Stopped python3
root@lccd085619ad:/volumes# python3 server.py
```

This action likely initializes the TLS server, allowing it to listen for client connections on a predefined port (likely 443, as per the lab instructions). The script facilitates the lab's experimentation with secure server-client communications.

Run the Client Program:

- Navigate to the directory where the `handshake.py` script is located.
- Run the script against a real server, such as www.google.com

1.a.1: Part 1 & 2 (cipher,server certificate):



The screenshot shows a terminal window titled "DianaN [Running] - Oracle VirtualBox". The terminal has multiple tabs open, all labeled "seed@VM: ~...". The current tab displays the command "root@4e8df03bc705:/volumes# python3 handshake.py www.google.com" followed by the output of the TLS handshake. The output includes the cipher suite used ("TLS_AES_256_GCM_SHA384"), the server's hostname ("www.google.com"), and a detailed breakdown of the server's certificate. The certificate information is color-coded: 'OCSP' (green), 'caIssuers' (green), 'crlDistributionPoints' (green), 'issuer' (green), 'notAfter' (green), 'notBefore' (orange), 'serialNumber' (green), 'subject' (green), 'subjectAltName' (orange), and 'version' (green).

```
root@4e8df03bc705:/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://o.pki.goog/wr2',),
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/oQ6nyr8F0m0.crl',),
 'issuer': (((('countryName', 'US'),),
              (('organizationName', 'Google Trust Services'),),
              (('commonName', 'WR2'))),
             'notAfter': 'Mar 3 08:38:05 2025 GMT',
             'notBefore': 'Dec 9 08:38:06 2024 GMT',
             'serialNumber': 'D01C59292557A18010EE18C306607692',
             'subject': (((('commonName', 'www.google.com'),),
                          'subjectAltName': ((('DNS', 'www.google.com'),),
                                         'version': 3}
```

The command `python3 handshake.py www.google.com` was executed from the `/volumes` directory inside a Docker container, initiating a TLS handshake with Google's server.

- **The cipher** suite provides robust encryption (AES-256), fast and secure data integrity (GCM), and strong hashing (SHA384), ensuring a highly secure communication channel between the client and the server. =>"TLS_AES_256_GCM_SHA384 "
- **the server certificate in the program:**

Issuer:

Country Name (C): US Organization Name (O): Google Trust Services

Common Name (CN): WR2

This means the certificate was issued by Google Trust Services, specifically through their intermediate certificate authority "WR2."

Not Before (Start Date): December 9, 2024, at 08:38:06 GMT.

Not After (Expiration Date): March 3, 2025, at 08:38:05 GMT.

The certificate is valid during this period, ensuring it hasn't expired.

Subject (Server Information):

Common Name (CN): www.google.com

Subject Alternative Name (SAN):

Lists additional DNS names the certificate is valid for.

In this case, it's also www.google.com.

1.a.2: Part 3 (/etc/ssl/certs.):

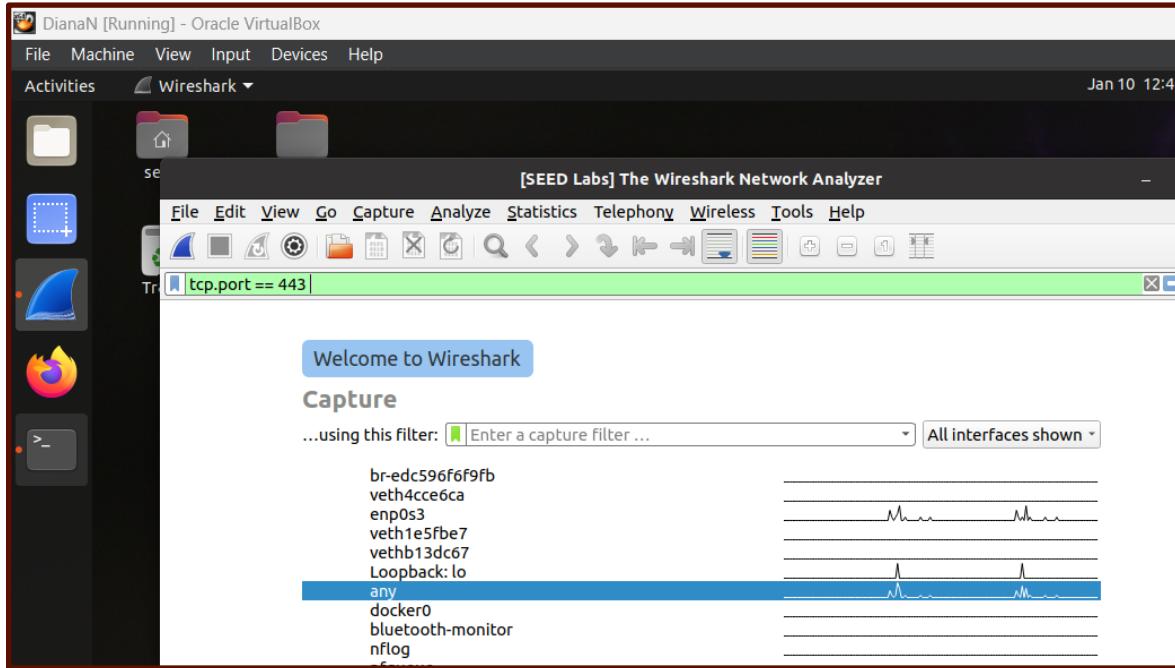
The purpose of /etc/ssl/certs:

During the TLS handshake, it presents its certificate to the client in order to establish a secure connection. The client then verifies this certificate against trusted Certificate Authority (CA) certificates stored in /etc/ssl/certs. For the client to validate the server's certificate, it needs to find a matching CA certificate by looking at its issuer details. If a valid and matching CA certificate is found, the server certificate is accepted as trustworthy. It relies on the chain of trust, meaning that the server certificate must go back to some trusted CA within /etc/ssl/certs. In case the chain is partially missing or at least one of its parts is an invalid intermediate or root CA certificate, the server certificate is refused, and the connection is terminated.

1.a.3 Part 4 (Wireshark):

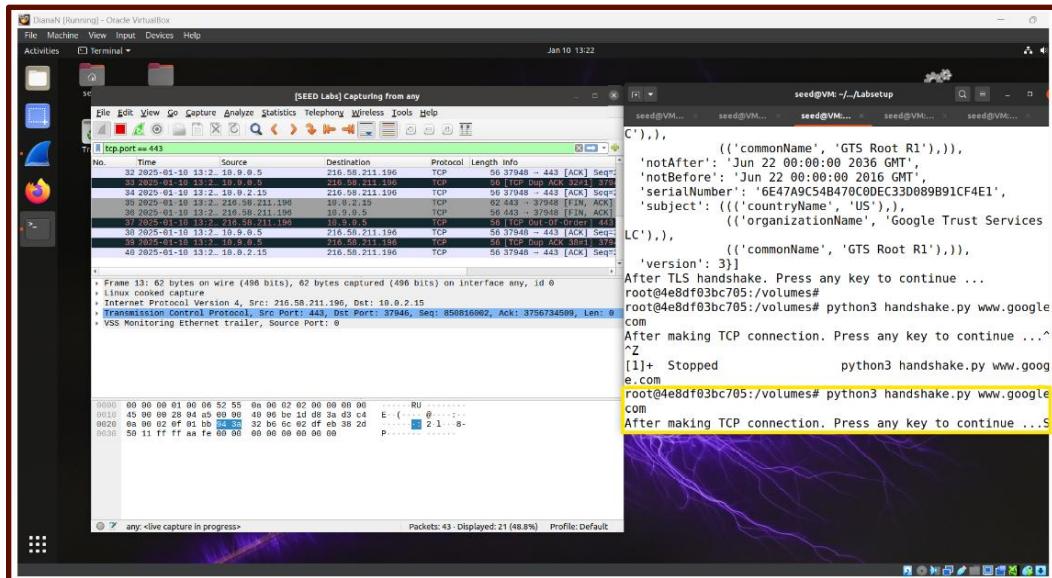
To capture the network traffic during the execution of the program.

I Set a capture filter for HTTPS traffic on port 443:



After this; Execute the Python script inside the client container, then capture the wireshark network traffic in TCP connection.

1. TCP Handshake



• Packets	Observed	in	Wireshark:
-----------	----------	----	------------

SYN (Client → Server): Initiates the connection to the server on port 443.

SYN-ACK (Server → Client): Acknowledges the connection request.

ACK (Client → Server): Completes the handshake and establishes a reliable connection.

• Triggered	By:
-------------	-----

The **TCP handshake** begins when the `socket.connect()` function is invoked in the `handshake.py` program. This function ensures that a reliable connection is set up before any higher-layer protocols are involved.

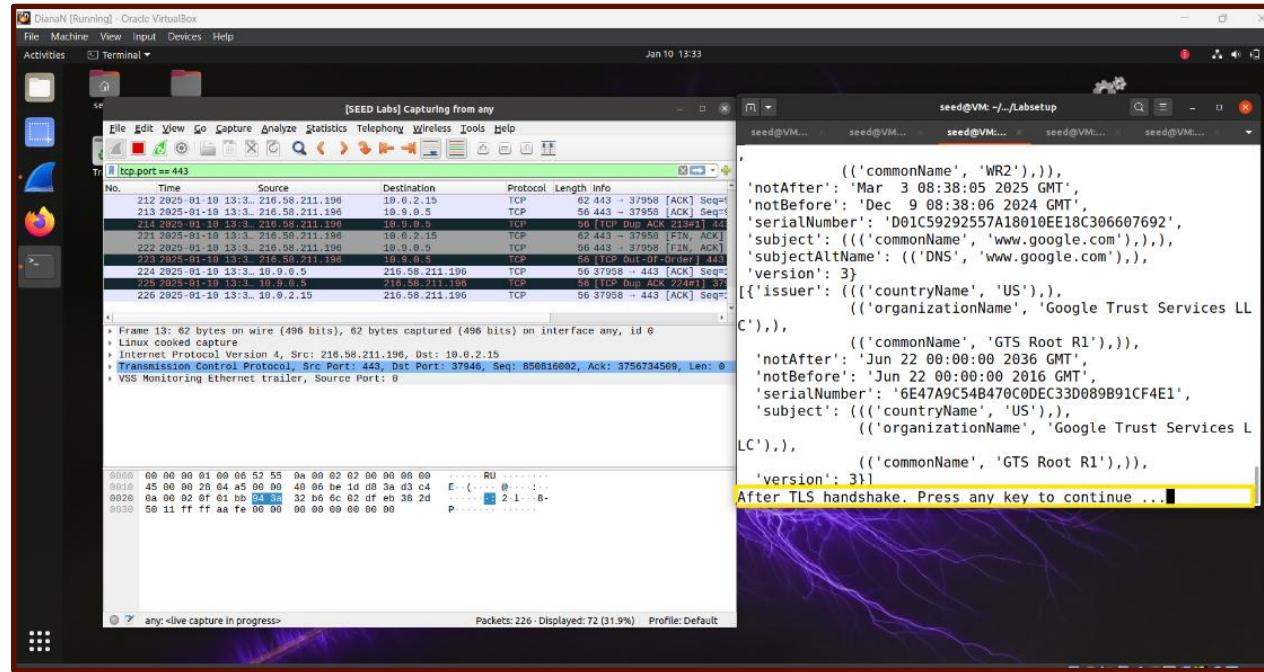
• Purpose	of	the	TCP	Handshake:
-----------	----	-----	-----	------------

Establishes a reliable and bidirectional communication channel between the client and server, ensuring readiness for data exchange.

=====

After press another time; Execute the Python script inside the client container, then capture the wireshark network traffic in TLS connection.

2. TLS Handshake



<ul style="list-style-type: none"> ● Packets 	Observed	in	Wireshark:
--	-----------------	-----------	-------------------

- **ClientHello (Client → Server):**

Starts the TLS handshake, providing supported protocols ,cipher suites, and random data.

- **ServerHello (Server → Client):**

Responds with the selected TLS version, cipher suite, and server's random data.

- **Certificate (Server → Client):**

Server sends its certificate for client validation.

- **Key Exchange (Both Directions):**

Key material is exchanged to establish encryption parameters.

- **Finished (Both Directions):**

Final confirmation from both sides indicating the secure connection is ready.

- **Triggered By:**

- The **TLS handshake** is initiated when the `ssock.do_handshake()` function is executed in the `handshake.py` program. This occurs after the TCP connection has been established.

- **Purpose of the TLS Handshake:**

- Secures the connection by:

Negotiating encryption parameters (cipher suite and keys).

Authenticating the server's identity using its certificate.

3. Relationship Between TCP and TLS Handshakes

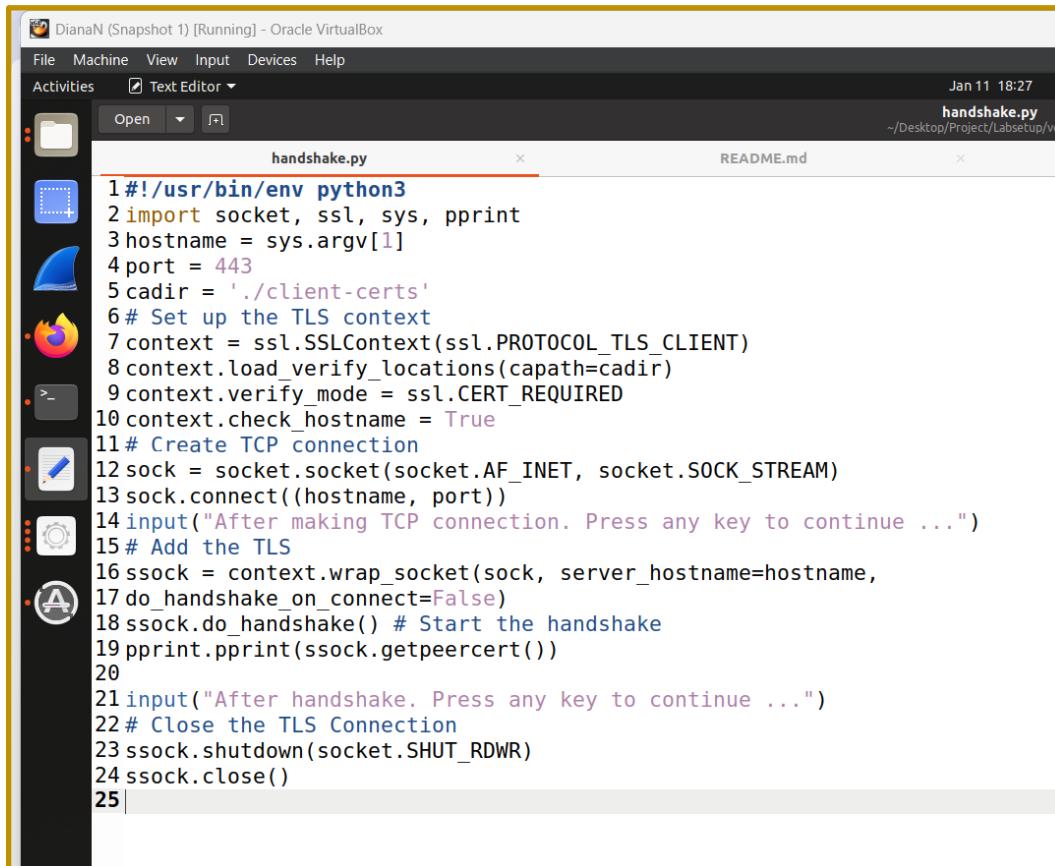
- The TLS handshake relies on the TCP handshake to have a reliable channel of communication; without TCP, TLS cannot proceed. While the TCP handshake works at the transport layer, which is Layer 4 of the OSI model, creating the basic communication channel, the TLS handshake adds encryption and authentication at the application layer, Layer 7. The TCP handshake provides reliability in establishing a connection, while the TLS handshake makes sure the connection is secure by encrypting data and verifying the identity of the server.

Task 1.b: CA's Certifications.

The objective is to set up the client program so that it validates server certificates using a specified custom directory for Certificate Authority (CA) certificates. This configuration ensures that the client only establishes secure connections with trusted servers by referencing the CA certificates in the designated directory.

The Python script ([handshake.py](#)) is designed to:

- Establish a TCP connection with a specified server (www.example.com or www.google.com).
- Perform a TLS handshake to validate the server's certificate against a set of trusted CA certificates located in a custom directory ([./client-certs](#)).



DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Text Editor Jan 11 18:27

Open handshake.py ~/Desktop/Project/Labsetup/vol

handshake.py

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

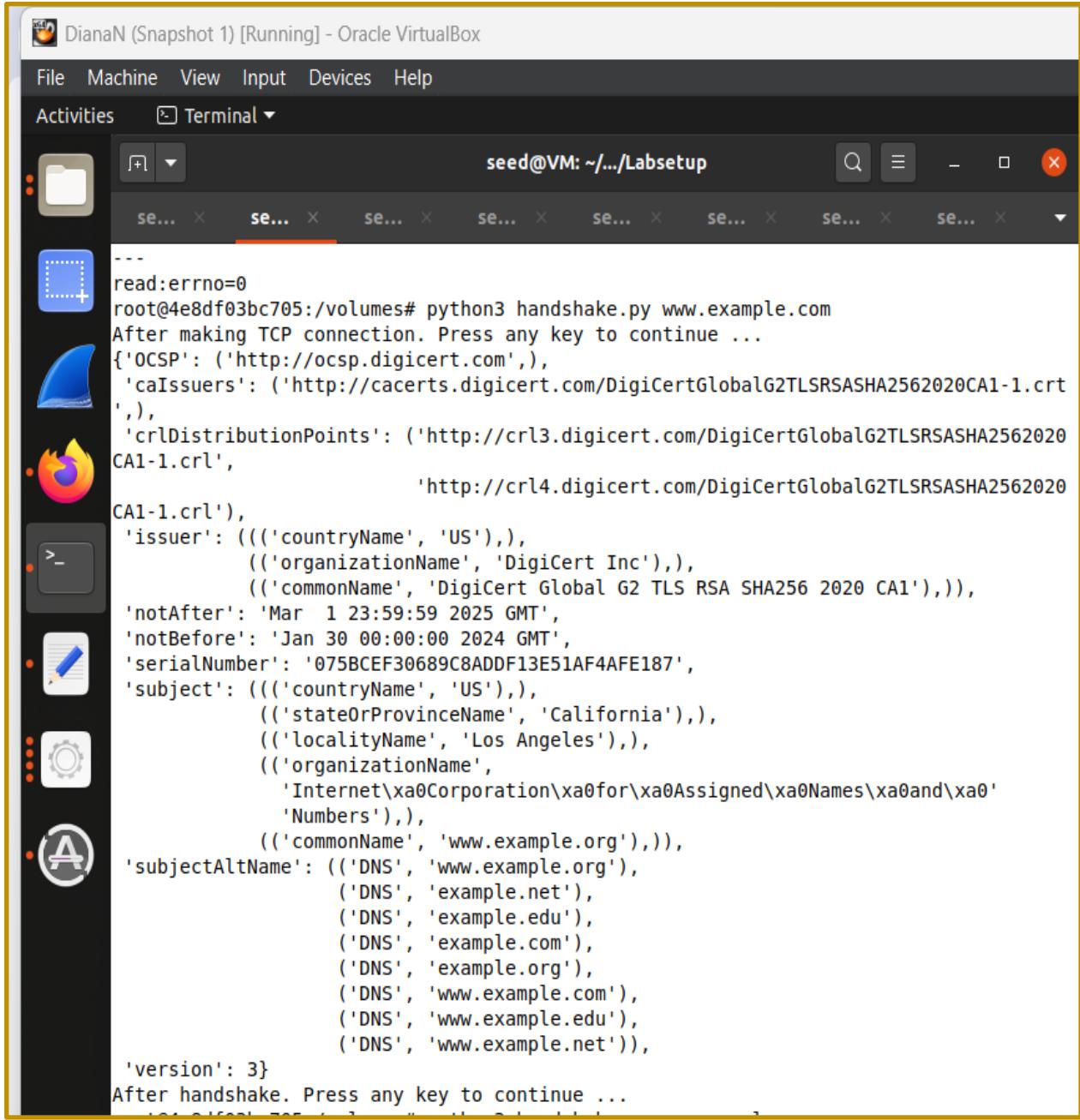
The test case describes the successful process of establishing a secure connection to www.example.com and validating its TLS certificate:

Certificate Details:

- **Issuer:** DigiCert Global G2 TLS RSA SHA256 2020 CA1
- **Validity:**
 - Start Date: January 30, 2024
 - Expiration Date: March 1, 2025
- **Subject:** www.example.org (including SANs such as example.com, example.net, etc.)

Results:

- **Handshake Outcome:** Successful
- **Certificate Validation:** Performed using CA certificates sourced from a specified custom directory.



```
seed@VM: ~/Labsetup
-- 
read:errno=0
root@4e8df03bc705:/volumes# python3 handshake.py www.example.com
After making TCP connection. Press any key to continue ...
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crt',
'), 
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020
CA1-1.crl',
                           'http://crl4.digicert.com/DigiCertGlobalG2TLSRSASHA2562020
CA1-1.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': '075BCEF30689C8ADD13E51AF4AFE187',
 'subject': (((('countryName', 'US'),),
               (('stateOrProvinceName', 'California'),),
               (('localityName', 'Los Angeles'),),
               (('organizationName',
                 'Internet\xa0Corporation\x0for\x0Assigned\x0Names\x0and\x0
'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}
After handshake. Press any key to continue ...
```

The test case describes a successful connection to www.google.com with insights into the certificate validation process and key observations:

Certificate Details

- **Issuer:** Google Trust Services (GTS)
 - **Validity:**
 - Start Date: December 9, 2024
 - Expiration Date: March 3, 2025
 - **Subject:** www.google.com
-

Results

- **Handshake Outcome:** Successful
 - **Certificate Validation:** Completed using CA certificates from a custom directory (`./client-certs`).
-

Key Observations

1. **Certificate Chain:**
 - The server's certificate and the intermediate CA certificate were retrieved.
 - Validation required both the root and intermediate CA certificates to be present in the custom directory (`./client-certs`).
2. **Custom CA Directory:**
 - The script loaded CA certificates from the custom directory, bypassing the system-wide directory (`/etc/ssl/certs`).
 - This approach allowed for more granular control over trusted CAs.
3. **Validation Errors:**
 - Missing or misconfigured CA certificates in the custom directory would result in a **CERTIFICATE_VERIFY_FAILED** error, halting the process.

DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Terminal

```
seed@VM: ~/.../Labsetup
```

Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 20 (unable to get local issuer certificate)

read:errno=0
root@4e8df03bc705:/volumes# python3 handshake.py www.example.com
After making TCP connection. Press any key to continue ...
{'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': '075BCEF30689C8ADDFF13E51AF4AFE187',
 'subject': (((('countryName', 'US'),),
 (('stateOrProvinceName', 'California'),),
 (('localityName', 'Los Angeles'),),
 ('organizationName',
 'Internet\x0Corporation\x0for\x0Assigned\x0Names\x0and\x0Numbers'),),
 (('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}
After handshake. Press any key to continue ...
root@4e8df03bc705:/volumes#

```
root@4e8df03bc705:/volumes# python3 handshake.py www.google.com  
After making TCP connection. Press any key to continue ...  
{'OCSP': ('http://o.pki.goog/wr2',),  
 'caIssuers': ('http://i.pki.goog/wr2.crt',),  
 'crlDistributionPoints': ('http://c.pki.goog/wr2/oQ6nyr8F0m0.crl',),  
 'issuer': (((('countryName', 'US'),),  
 (('organizationName', 'Google Trust Services'),),  
 (('commonName', 'WR2'),)),  
 'notAfter': 'Mar 3 08:38:05 2025 GMT',  
 'notBefore': 'Dec 9 08:38:06 2024 GMT',  
 'serialNumber': 'D01C59292557A18010EE18C306607692',  
 'subject': (((('commonName', 'www.google.com'),),),  
 'subjectAltName': (('DNS', 'www.google.com'),),  
 'version': 3}  
After handshake. Press any key to continue ...  
root@4e8df03bc705:/volumes#
```

Advantages of Custom CA Configuration

1. Enhanced Security:

- Restricts trust to explicitly added CAs in the custom directory, reducing exposure to potentially untrusted CAs.

2. Custom Validation:

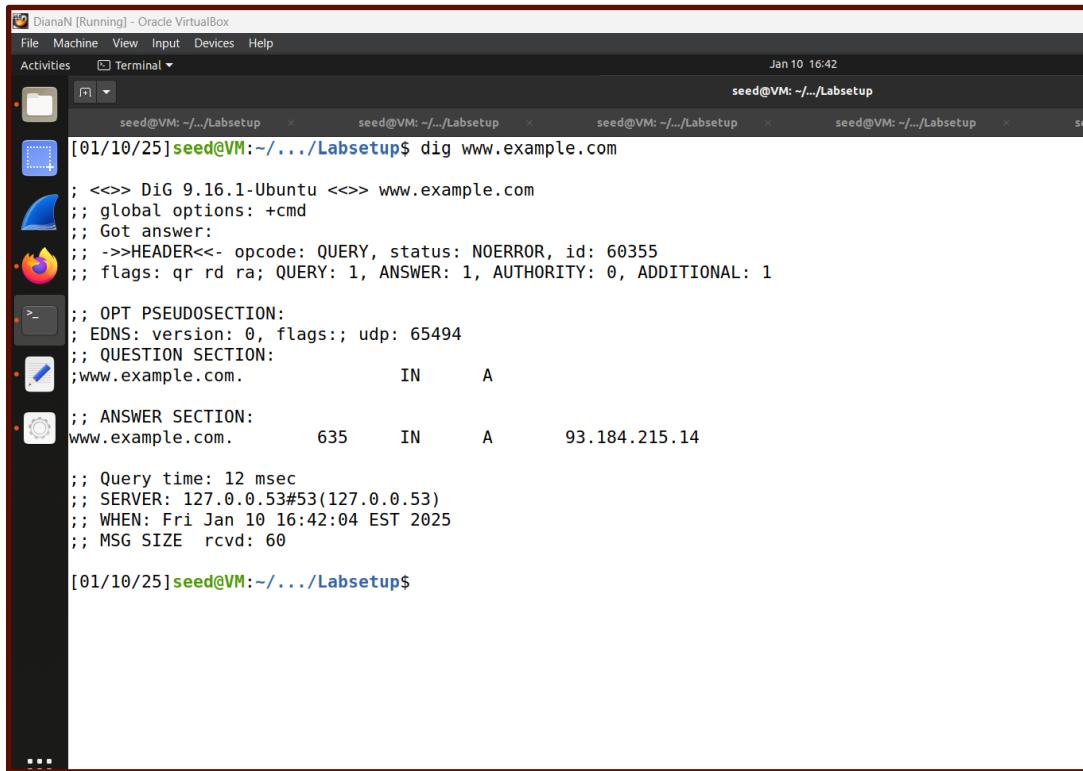
- Suitable for environments where only specific CAs are permitted, such as private networks or specialized applications.

3. Testing and Debugging:

- Enables isolated certificate validation, making it easier to troubleshoot and validate configurations in controlled scenarios.

Task 1.c: Experiment with the hostname check

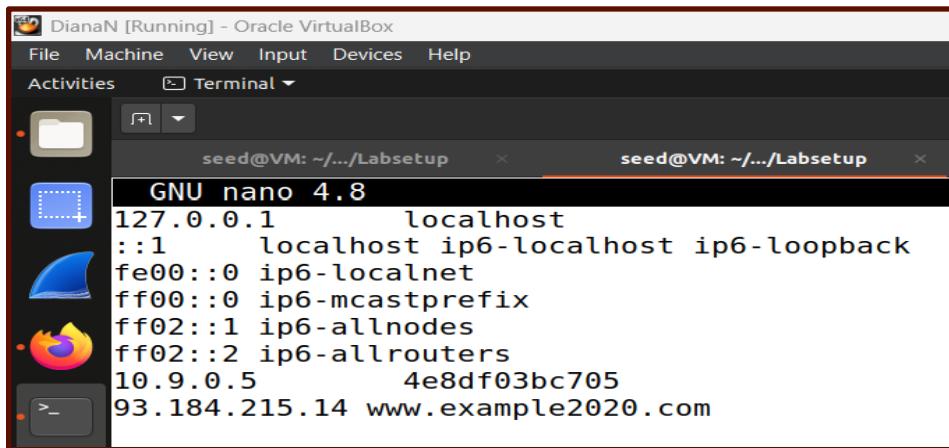
The `dig` command, work in this setup, and does not work in containers typically configured without extensive network utilities.



DianaN [Running] - Oracle VirtualBox

```
[01/10/25]seed@VM:~/.../Labsetup$ dig www.example.com
; <>> DiG 9.16.1-Ubuntu <>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 60355
;; 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.    635     IN      A      93.184.215.14
;
;; Query time: 12 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Jan 10 16:42:04 EST 2025
;; MSG SIZE  rcvd: 60
[01/10/25]seed@VM:~/.../Labsetup$
```

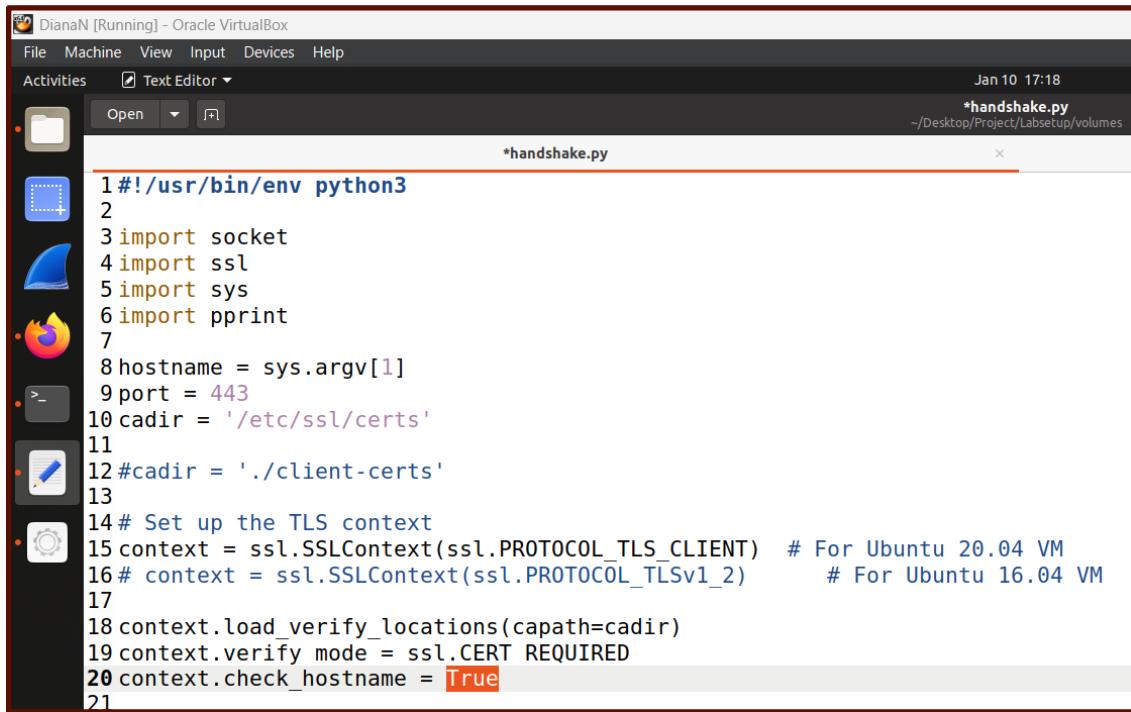
Modify the `/etc/hosts` file (inside the container), add the following entry at the end of the file (the IP address is what you get from the `dig` command). 93.184.216.34 www.example2020.com



DianaN [Running] - Oracle VirtualBox

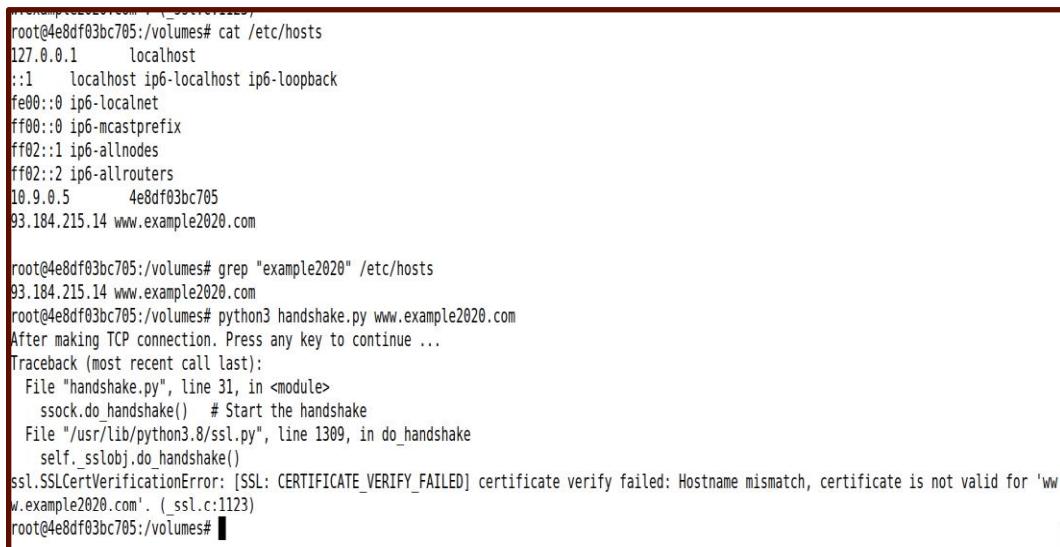
```
seed@VM: ~/.../Labsetup$ nano /etc/hosts
GNU nano 4.8
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5        4e8df03bc705
93.184.215.14  www.example2020.com
```

When we put the “check_hostname = True” i get this when i try to run www.example2020.com:



A screenshot of a Linux desktop environment. On the left is a file manager window titled "DianaN [Running] - Oracle VirtualBox". On the right is a terminal window titled "handshake.py" with the path "/Desktop/Project/Labsetup/volumes". The terminal contains the following Python script:

```
#!/usr/bin/env python3
import socket
import ssl
import sys
import pprint
hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'
#cadir = './client-certs'
# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2) # For Ubuntu 16.04 VM
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
```

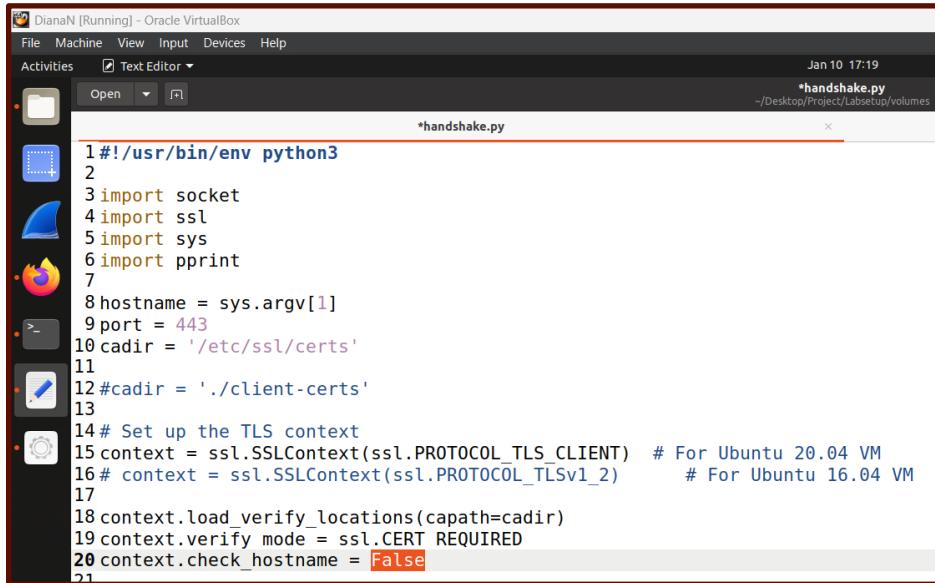


```
root@e8df03bc705:/volumes# cat /etc/hosts
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5 4e8df03bc705
93.184.215.14 www.example2020.com

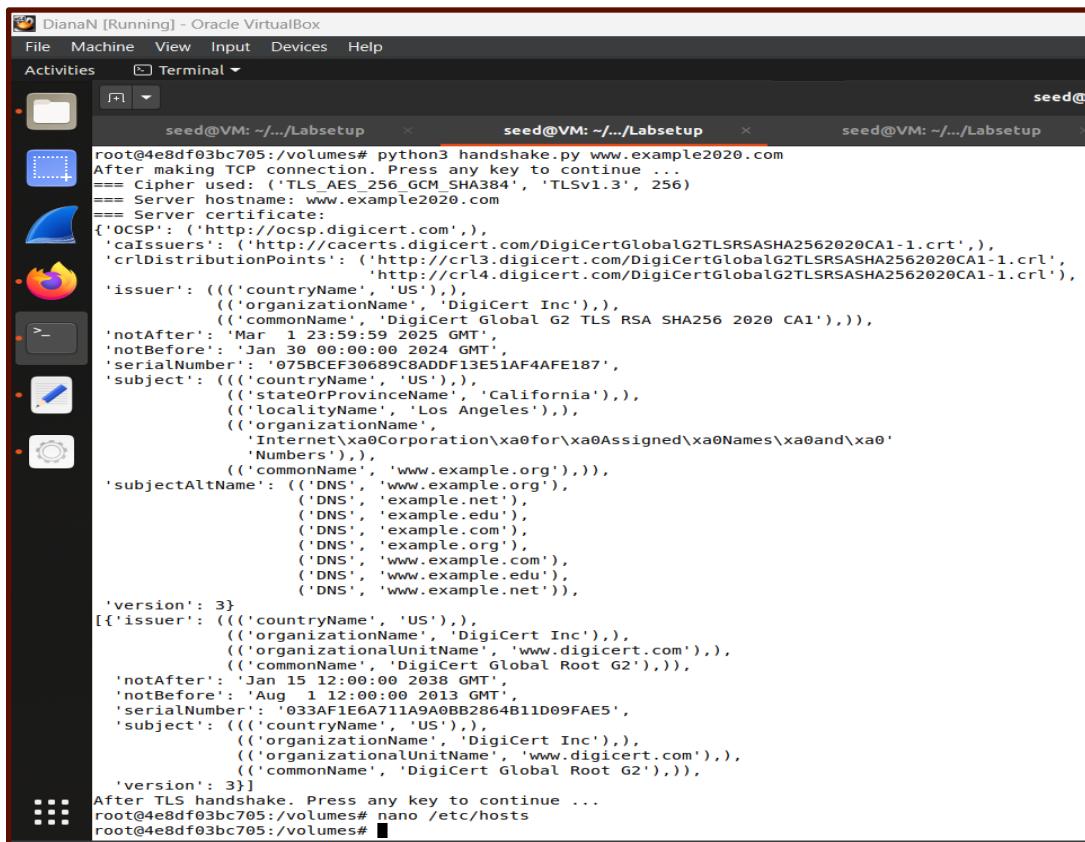
root@e8df03bc705:/volumes# grep "example2020" /etc/hosts
93.184.215.14 www.example2020.com
root@e8df03bc705:/volumes# python3 handshake.py www.example2020.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake.py", line 31, in <module>
    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.example2020.com'. ( ssl.c:1123)
```

The error message indicates that the hostname validation failed. This occurred because the server's certificate was issued for www.example.com, but your client is attempting to connect to www.example2020.com.

When we put the “check_hostname = False” i get this when i try to run www.example2020.com:



```
#!/usr/bin/env python3
import socket
import ssl
import sys
import pprint
hostname = sys.argv[1]
port = 443
cadir = '/etc/ssl/certs'
#cadir = './client-certs'
# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2) # For Ubuntu 16.04 VM
context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = False
```



```
root@4e8df03bc705:/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': '075BCEF30689C8ADDFF13E51AF4AFE187',
              'subject': (((('countryName', 'US'),),
                           (('stateOrProvinceName', 'California'),),
                           (('localityName', 'Los Angeles'),),
                           (('organizationName',
                             'Internet\x00\Corporation\x00for\x00Assigned\x00Names\x00and\x00Numbers'),),
                           ('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 ...
root@4e8df03bc705:/volumes# nano /etc/hosts
root@4e8df03bc705:/volumes#
```

This time, the hostname check will be bypassed, and the connection should succeed.

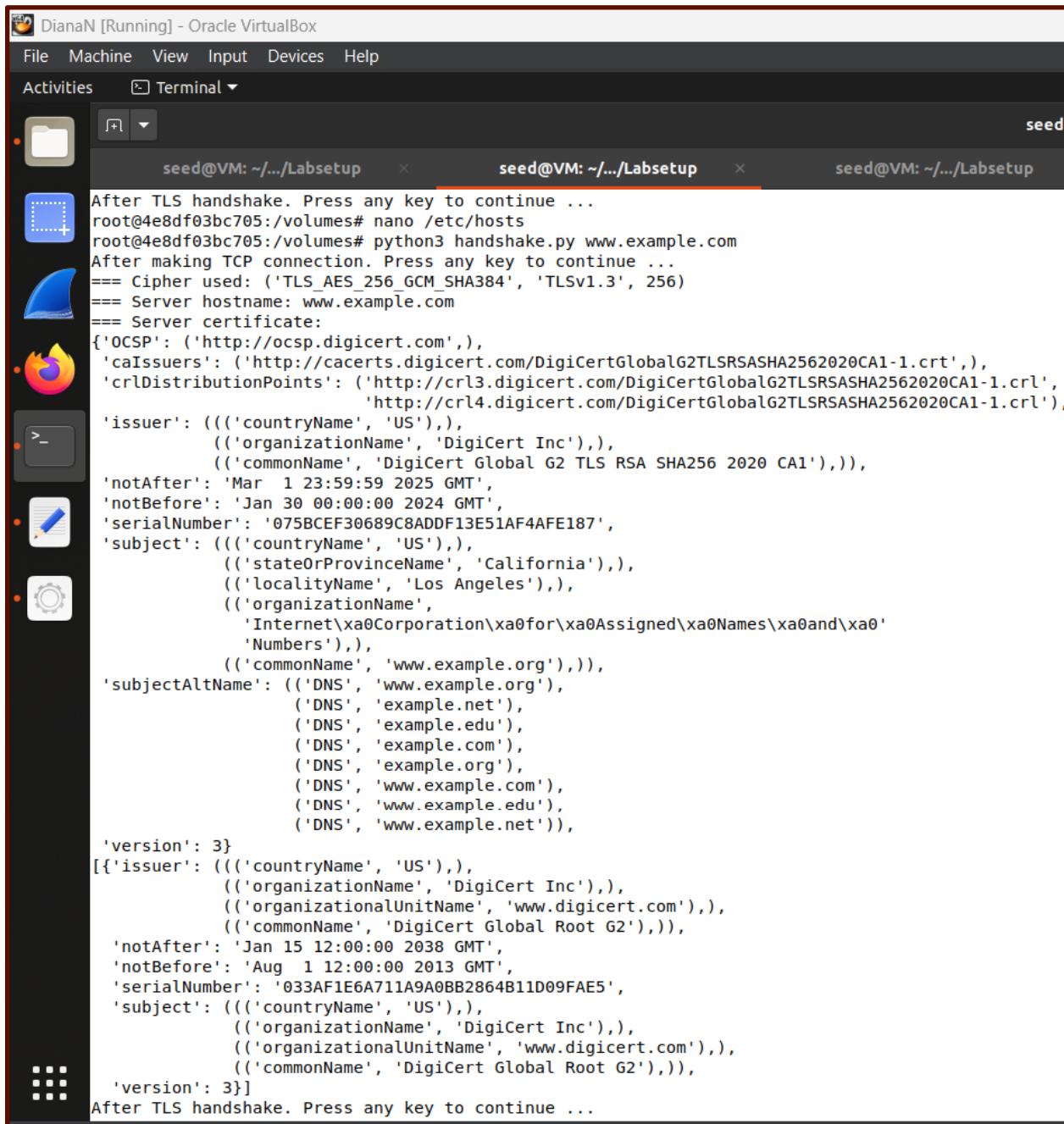
When `check_hostname = True` (default setting), the client verifies that the hostname in the certificate matches the requested hostname (www.example2020.com). This is why the connection fails.

By setting `check_hostname = False`, the client skips the hostname verification step, allowing the connection to succeed. However, this introduces a security risk, as the client cannot confirm the server's identity.

Checks of hostnames should be performed to ensure that the server it connects to is the one that was identified by its SSL/TLS certificate. This makes sure that it communicates with the right server and not some impersonator. If it does not perform host name checking, it introduces a rather critical security weakness whereby an attacker can set up malicious servers with valid but unrelated certificates, which they can use to conduct Man-in-the-Middle or spoofing of servers. It provides means for unauthorized access to sensitive information, theft of personal or financial data, or the compromise of communication security as a whole. Hence, checks on hostnames form one of those steps which become imperative to create trusted, secured connections over the Internet for protection of the users and their systems against specific kinds of cyber threats.

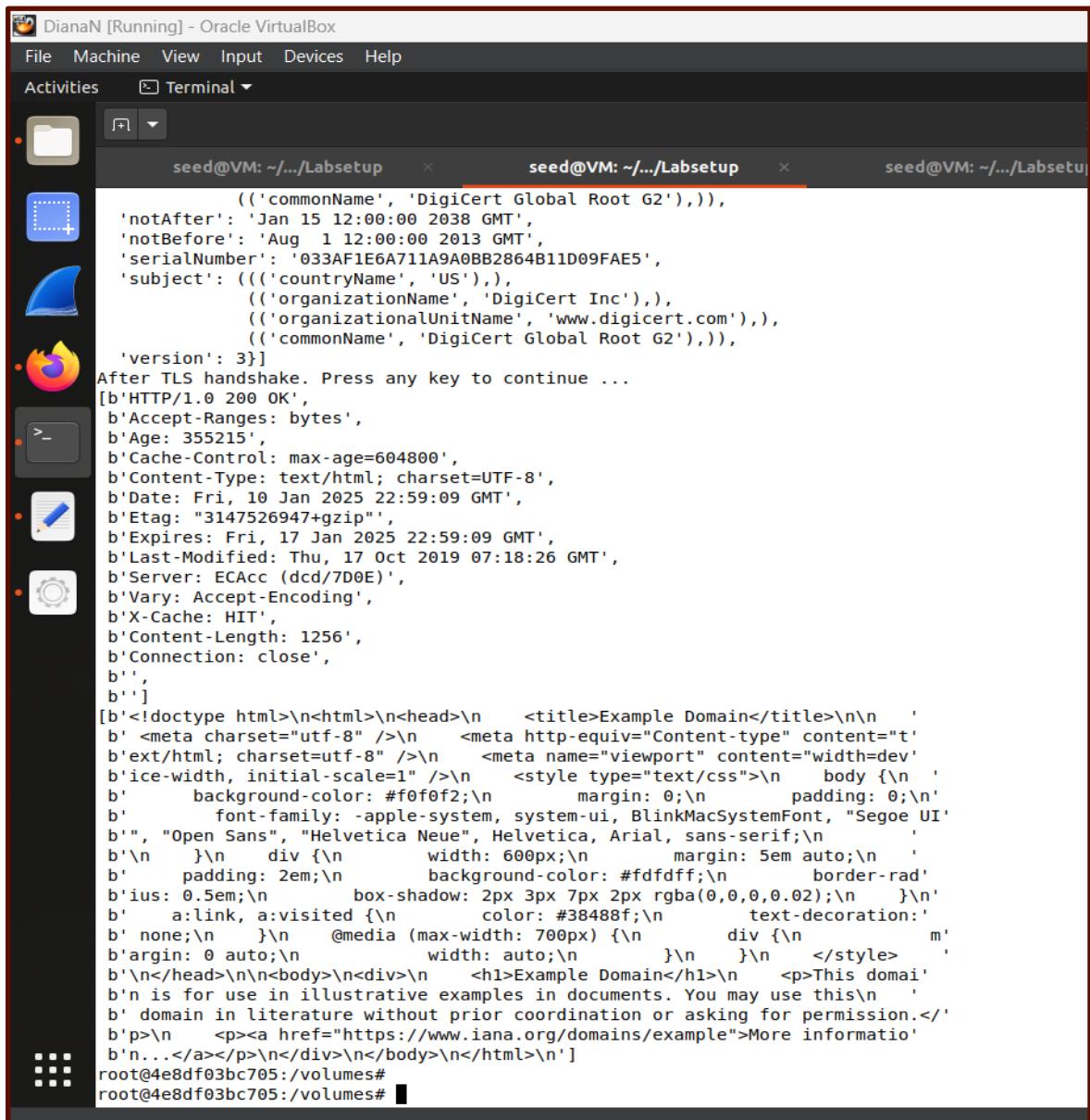
Task 1.d: Sending and getting Data

TLS handshake and server certificate are displayed. It confirms that the handshake was successful, specifying the cipher suite used ([TLS_AES_256_GCM_SHA384](#)) and providing server certificate details, including the issuer ([DigiCert Inc.](#)) and its validity period. This ensures that the server's identity is verified before data exchange occurs.



```
After TLS handshake. Press any key to continue ...
root@4e8df03bc705:/volumes# nano /etc/hosts
root@4e8df03bc705:/volumes# python3 handshake.py www.example.com
After making TCP connection. Press any key to continue ...
==> Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
==> Server hostname: www.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': '075BCEF30689C8ADD13E51AF4AFE187',
            'subject': (((('countryName', 'US'),),
                         (('stateOrProvinceName', 'California'),),
                         (('localityName', 'Los Angeles'),),
                         (('organizationName',
                           'Internet\x00Corporation\x00for\x00Assigned\x00Names\x00and\x00'
                           '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 ...
```

A successful HTTP GET request sent to the server would look something like the following, such as to www.example.com over a secure TLS connection. It would include the HTTP status code from the server, such as 200 OK, headers including Content-Type and Date, and the actual content of the response. This confirms that the client program is capable of establishing a secure TLS connection, sending a request, and receiving a response.

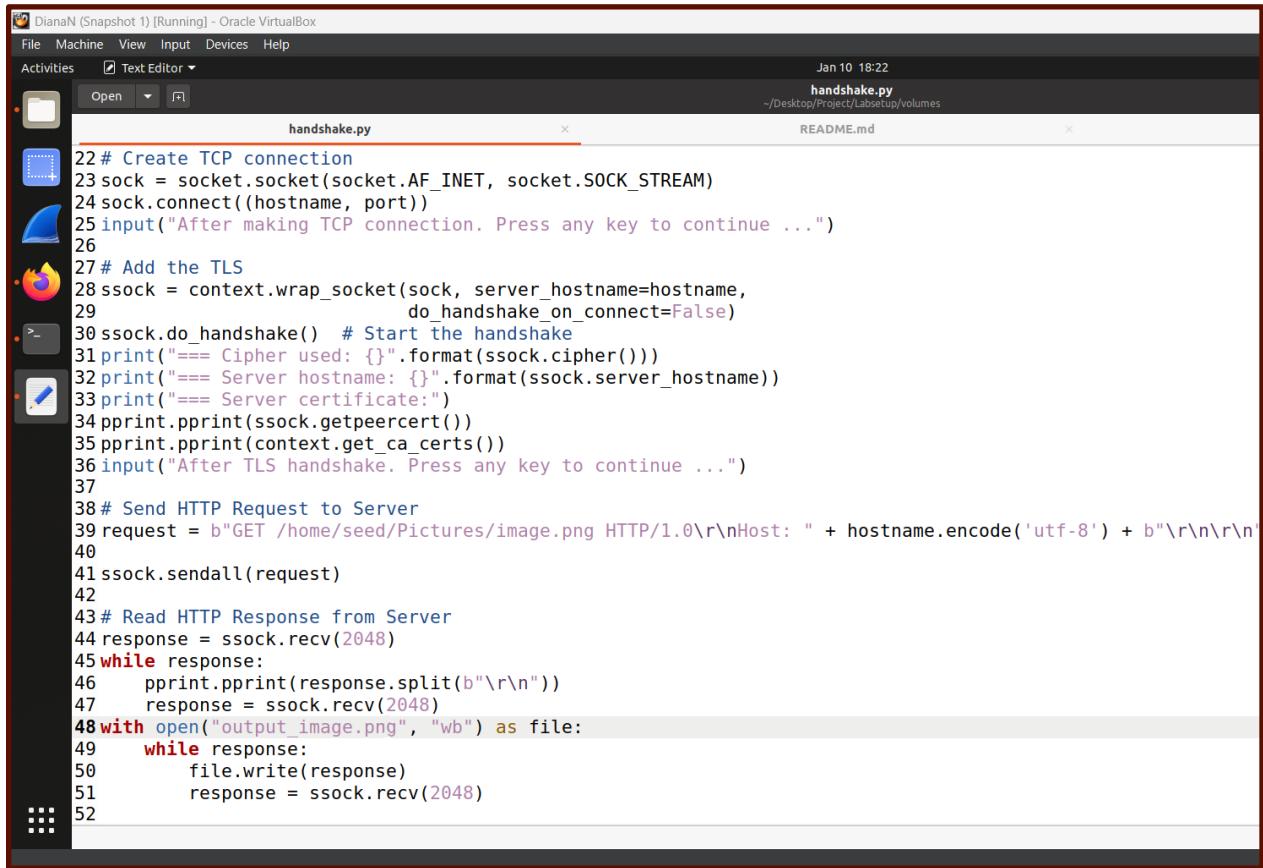


The screenshot shows a Linux desktop environment with a dark theme. A terminal window is open, showing the output of an SSL/TLS handshake. The output includes the certificate details of the server, such as commonName, DigiCert Global Root G2, and various timestamps. It also shows the HTTP response headers, including the status code 200 OK, content type, date, etag, expires, last-modified, server, vary, and cache-control. Finally, it displays the HTML content of the page, which is a placeholder example domain page with styling for a main content area.

```
seed@VM: ~.../Labsetup  x  seed@VM: ~.../Labsetup  x  seed@VM: ~.../Labsetup
('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 ...
[b'HTTP/1.0 200 OK',
b'Accept-Ranges: bytes',
b'Age: 355215',
b'Cache-Control: max-age=604800',
b'Content-Type: text/html; charset=UTF-8',
b'Date: Fri, 10 Jan 2025 22:59:09 GMT',
b'Etag: "3147526947+gzip"',
b'Expires: Fri, 17 Jan 2025 22:59:09 GMT',
b'Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT',
b'Server: ECacc (dcd/7D0E)',
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    <meta charset="utf-8" />\n    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />\n    <meta name="viewport" content="width=device-width, initial-scale=1" />\n    <style type="text/css">\n        body {\n            margin: 0;\n            padding: 0;\n            font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;\n        }\n        div {\n            width: 600px;\n            margin: 5em auto;\n            padding: 2em;\n            background-color: #fdfdff;\n            border-radius: 0.5em;\n            box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n        }\n        a:link, a:visited {\n            color: #38488f;\n            text-decoration: none;\n        }\n        @media (max-width: 700px) {\n            div {\n                width: auto;\n                margin: 0 auto;\n            }\n        }\n    </style>\n</head>\n<body>\n    <div>\n        <h1>Example Domain</h1>\n        <p>This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.</p>\n        <p><a href="https://www.iana.org/domains/example">More information</a></p>\n    </div>\n</body>\n</html>\n']
root@4e8df03bc705:/volumes#
root@4e8df03bc705:/volumes#
```

Now, when i put image request:

Also i add while loop to This saves the image data as `output_image.jpg` in the current directory



```
22 # Create TCP connection
23 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
24 sock.connect((hostname, port))
25 input("After making TCP connection. Press any key to continue ...")
26
27 # Add the TLS
28 ssock = context.wrap_socket(sock, server_hostname=hostname,
29                             do_handshake_on_connect=False)
30 ssock.do_handshake() # Start the handshake
31 print("==> Cipher used: {}".format(ssock.cipher()))
32 print("==> Server hostname: {}".format(ssock.server_hostname))
33 print("==> Server certificate:")
34 pprint.pprint(ssock.getpeercert())
35 pprint.pprint(context.get_ca_certs())
36 input("After TLS handshake. Press any key to continue ...")
37
38 # Send HTTP Request to Server
39 request = b"GET /home/seed/Pictures/image.png HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
40
41 ssock.sendall(request)
42
43 # Read HTTP Response from Server
44 response = ssock.recv(2048)
45 while response:
46     pprint.pprint(response.split(b"\r\n"))
47     response = ssock.recv(2048)
48 with open("output_image.png", "wb") as file:
49     while response:
50         file.write(response)
51         response = ssock.recv(2048)
52
```

A successful HTTP GET request sent to the server would look something like the following, such as to www.example.com over a secure TLS connection. It would include the HTTP status code from the server, such as 200 OK, headers including Content-Type and Date, and the actual content of the response. This confirms that the client program is capable of establishing a secure TLS connection, sending a request, and receiving a response.

DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

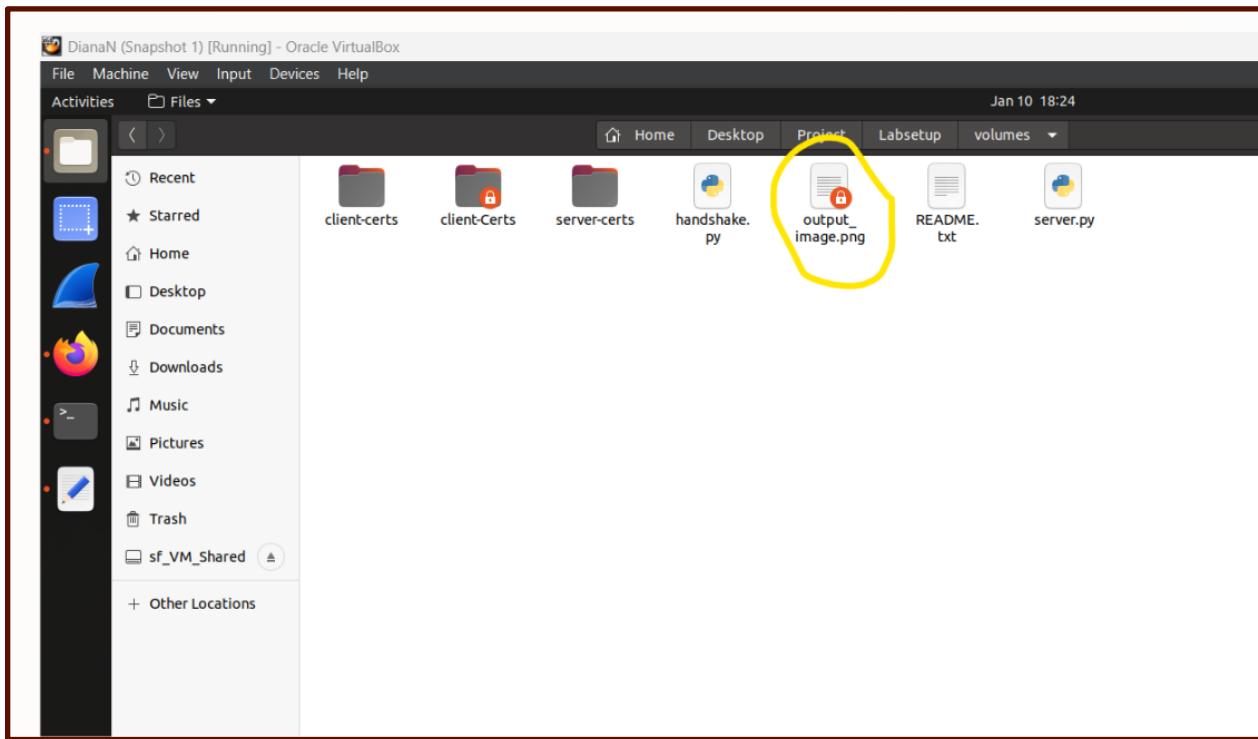
Activities Terminal

seed@VM: ~/.../Labsetup

```
((('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 ...
[b'HTTP/1.0 404 Not Found',
 b'Age: 583557',
 b'Cache-Control: max-age=604800',
 b'Content-Type: text/html',
 b'Date: Fri, 10 Jan 2025 23:21:17 GMT',
 b'Etag: "1088432560+gzip+ident"',
 b'Expires: Fri, 17 Jan 2025 23:21:17 GMT',
 b'Last-Modified: Sat, 12 Oct 2024 03:26:51 GMT',
 b'Server: ECAcc (dcd/7D26)',
 b'Vary: Accept-Encoding',
 b'X-Cache: 404-HIT',
 b'Content-Length: 1256',
 b'Connection: close',
 b '',
 b'']
[b'<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    <meta charset="utf-8" />\n    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />\n    <meta name="viewport" content="width=device-width, initial-scale=1" />\n    <style type="text/css">\n        body {\n            background-color: #f0f0f2;\n            margin: 0;\n            padding: 0;\n            font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;\n        }\n        div {\n            width: 600px;\n            margin: 5em auto;\n            padding: 2em;\n            background-color: #fdfdff;\n            border-radius: 0.5em;\n            box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n        }\n        a:link, a:visited {\n            color: #38488f;\n            text-decoration: none;\n        }\n        @media (max-width: 700px) {\n            div {\n                width: auto;\n            }\n        }\n    </style>\n</head>\n<body>\n<div>\n    <h1>Example Domain</h1>\n    <p>This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.</p>\n    <p><a href="https://www.iana.org/domains/example">More information</a></p>\n</div>\n</body>\n</html>\n']
root@4e8df03bc705:/volumes#
```

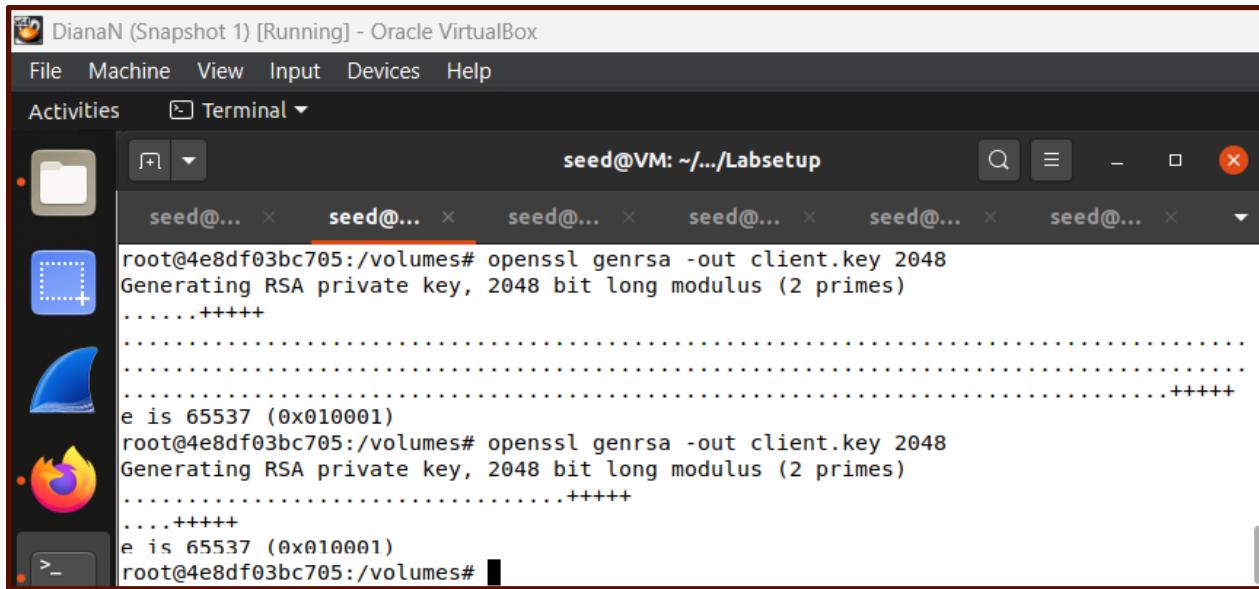
And to be sure that it work:

We see that the image upload here after run the command , and that is mean it work successfully!



Task2:TLS Server.

This command creates the private key `client.key` for the client.



```
root@4e8df03bc705:/volumes# openssl genrsa -out client.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
root@4e8df03bc705:/volumes# openssl genrsa -out client.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
root@4e8df03bc705:/volumes#
```

then , to generate a Certificate Signing Request (CSR).

The first step: **Generate a CA Certificate (ca.crt):**

Generate a CA certificate (self-signed) using the private key and fill the info.

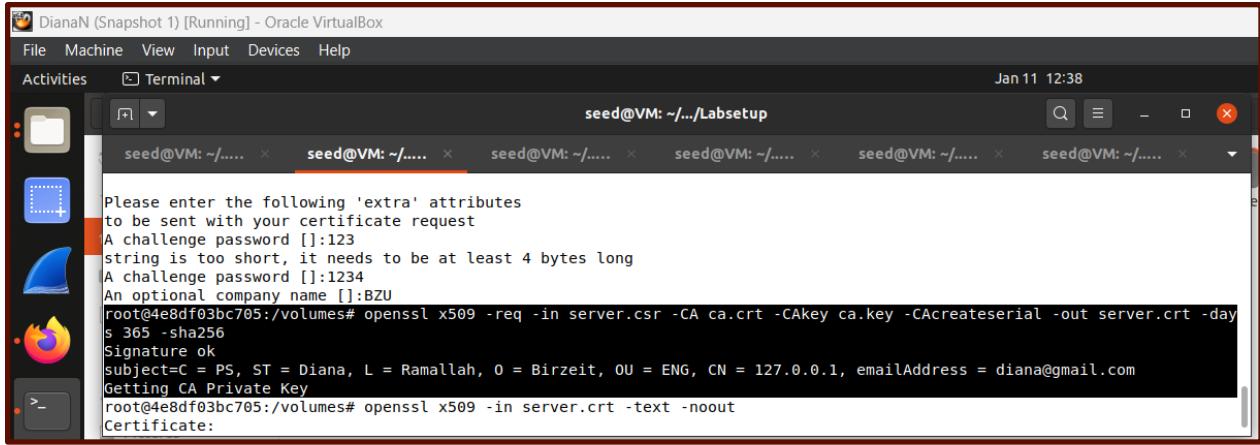
```
cp: cannot stat '/usr/share/doc/openvpn/examples/sample-keys/ca.crt': No such file or directory
root@4e8df03bc705:/volumes# cd ca.key
root@4e8df03bc705:/volumes# openssl genrsa -out ca.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
root@4e8df03bc705:/volumes# openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PS
State or Province Name (full name) [Some-State]:Diana
Locality Name (eg, city) []:Ramallah
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Birzeit
Organizational Unit Name (eg, section) []:ENG
Common Name (e.g. server FQDN or YOUR name) []:D_CA
Email Address []:diana@gmail.com
root@4e8df03bc705:/volumes# openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
```

2nd step: Generate a Server Certificate (server.crt):

```
Email Address []:diana@gmail.com
root@4e8df03bc705:/volumes# openssl genrsa -out server.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
root@4e8df03bc705:/volumes# openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PS
State or Province Name (full name) [Some-State]:Diana
Locality Name (eg, city) []:Ramallah
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Birzeit
Organizational Unit Name (eg, section) []:ENG
Common Name (e.g. server FQDN or YOUR name) []:127.0.0.1
Email Address []:diana@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123
string is too short, it needs to be at least 4 bytes long
A challenge password []:1234
An optional company name []:BZU
```

3rd step: Sign the CSR with the CA to generate the server certificate:



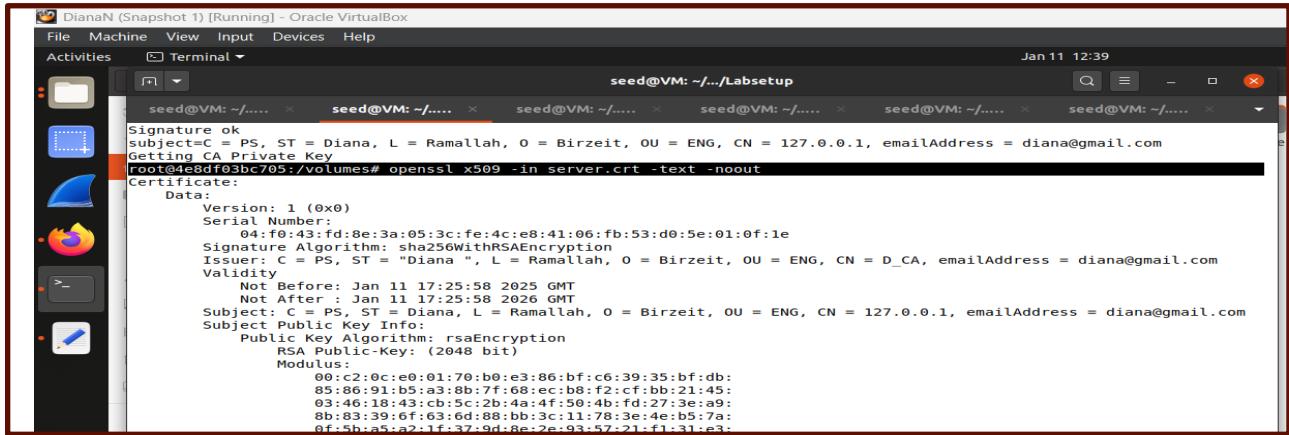
DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Terminal Jan 11 12:38

```
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123
String is too short, it needs to be at least 4 bytes long
A challenge password []:1234
An optional company name []:BZU
root@4e8df03bc705:/volumes# openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -day
s 365 -sha256
Signature ok
subject=C = PS, ST = Diana, L = Ramallah, O = Birzeit, OU = ENG, CN = 127.0.0.1, emailAddress = diana@gmail.com
Getting CA Private Key
root@4e8df03bc705:/volumes# openssl x509 -in server.crt -text -noout
Certificate:
```

4th step: Verify the server certificate.



DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Terminal Jan 11 12:39

```
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
seed@VM: ~/.....
Signature ok
subject=C = PS, ST = Diana, L = Ramallah, O = Birzeit, OU = ENG, CN = 127.0.0.1, emailAddress = diana@gmail.com
Getting CA Private Key
root@4e8df03bc705:/volumes# openssl x509 -in server.crt -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            04:f0:43:fd:8e:3a:05:3c:fe:4c:e8:41:06:fb:53:d0:5e:01:0f:1e
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = "Diana ", L = Ramallah, O = Birzeit, OU = ENG, CN = D_CA, emailAddress = diana@gmail.com
        Validity
            Not Before: Jan 11 17:25:58 2025 GMT
            Not After : Jan 11 17:25:58 2026 GMT
        Subject: C = PS, ST = Diana, L = Ramallah, O = Birzeit, OU = ENG, CN = 127.0.0.1, emailAddress = diana@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                    Modulus:
                        00:c2:0c:e0:01:70:b0:e3:86:bf:c6:39:35:bf:db:
                        85:86:91:b5:a3:8b:7f:68:ec:b8:f2:cfc:b1:45:
                        03:46:18:43:cb:5c:2b:4a:4f:50:4b:fd:27:3e:a9:
                        8b:83:39:6f:63:od:88:bb:3c:11:78:3e:4e:b5:7a:
                        0f:5h:a5:a2:1f:37:9d:8e:2e:93:97:21:f1:31:e3:
```

DianaN (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

Activities Terminal Jan 11 12:39

```
seed@VM: ~/.../Labsetup
seed@VM: ~/.... x seed@VM: ~/.... x
Issuer: C = PS, ST = "Diana ", L = Ramallah, O = Birzeit, OU = ENG, CN = D_CA, emailAddress = diana@gmail.com
Validity
    Not Before: Jan 11 17:25:58 2025 GMT
    Not After : Jan 11 17:25:58 2026 GMT
Subject: C = PS, ST = Diana, L = Ramallah, O = Birzeit, OU = ENG, CN = 127.0.0.1, emailAddress = diana@gmail.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
            Modulus:
                00:c2:0c:e0:01:70:b0:e3:86:bf:c6:39:35:bf:db:
                85:86:91:b5:a3:8b:7f:68:ec:b8:f2:cff:bb:21:45:
                03:46:18:43:cb:5c:2b:4a:4f:50:4b:fd:27:3e:a9:
                8b:83:39:6f:63:6d:88:bb:3c:11:78:3e:4e:b5:7a:
                0f:5b:a5:a2:1f:37:9d:e2:9e:93:57:21:f1:31:e3:
                85:c3:70:05:8b:c0:31:d1:d3:fd:95:34:53:5e:ad:
                fe:ad:8c:a8:89:be:86:2f:87:54:3b:86:5a:f1:04:
                e0:fe:eb:f3:7f:c5:51:c4:03:cc:8b:e3:31:52:91:
                ce:ef:41:4b:38:89:a7:61:8e:75:7e:f5:a3:56:99:
                9f:dc:9a:ab:d4:f3:e3:c8:41:aa:9f:ed:7d:2d:
                49:b1:ab:c5:e7:7f:fd:fc:95:bb:b8:8a:e0:ca:20:
                1d:79:fe:ce:6f:72:16:e9:48:43:c5:1a:8b:4c:95:
                f6:20:89:f3:e3:09:0b:ee:f8:98:ea:4b:da:9b:19:
                8f:16:67:99:ab:cf:95:94:15:df:c3:43:c1:99:aa:
                c2:66:37:26:19:de:48:f2:16:2e:ef:ca:eb:c4:58:
                f8:96:9f:82:9d:97:b1:6f:0d:a7:50:77:50:55:69:
                45:66:e0:d9:9d:9c:57:dcc:f5:8a:3e:c8:79:1e:5f:
                7f:b7
            Exponent: 65537 (0x10001)
Signature Algorithm: sha256WithRSAEncryption
    13:f0:38:16:66:94:f2:00:b1:b5:5c:77:f0:6f:db:f4:c1:bb:
    bf:68:9e:7f:45:a3:02:77:28:af:2b:57:42:e8:c5:1a:bd:d8:
    ba:ce:ba:ce:98:c1:75:c3:f7:cd:55:f7:37:56:11:56:04:ed:
    bf:42:60:ca:16:95:0e:fb:16:31:9d:0f:f3:de:e0:9b:61:91:
    54:ec:2f:31:44:74:99:de:90:ad:5f:cb:2c:7b:a9:31:1c:dd:
    c7:7b:ad:40:03:14:1b:0f:ad:09:ed:8c:41:ce:9c:03:12:7c:
    e4:a6:f9:4b:fe:df:96:45:a0:c8:72:c3:1c:20:8e:c4:35:0f:
    a6:77:2f:9b:4d:8e:54:a0:a8:88:de:1e:18:0a:ce:06:5e:1c:
    70:49:67:42:1d:07:6d:83:7d:e5:e6:d1:ec:f1:18:ce:25:93:
    8c:17:66:4b:52:c5:e3:39:16:b9:fdf:b1:35:5:f9:d1:54:f9:
    24:dd:36:8d:39:b6:e1:a1:f3:52:a9:f6:1f:24:73:18:da:c6:
    3f:0c:d4:37:21:09:0c:62:6e:f2:72:cf:67:54:0e:9e:07:cd:
    33:2e:f1:e0:10:38:2f:f0:5f:10:41:87:15:d3:50:aa:9b:bc:
    7b:a3:45:1e:6d:2c:56:0f:f4:31:e8:eb:f8:a7:fa:b5:db:2d:
    a8:00:fe:8c
root@4e8df09bc705:/volumes#
```

To Check if the files `ca.crt`, `server.crt`, and `server.key` exist:

```
root@4e8df03bc705:/volumes# CD client-certs/
bash: CD: command not found
root@4e8df03bc705:/volumes# cd client-certs/
root@4e8df03bc705:/volumes/client-certs# ls
4a6481c9.0 GTS_Root_R1.pem Globalsign_Root_CA.crt README.md abcd1234.0 ca.crt
root@4e8df03bc705:/volumes/client-certs# cd ..
root@4e8df03bc705:/volumes# ls
README.txt ca.key client-Certs client.csr handshake.py server-certs server.csr server.py
ca.crt ca.srl client-certs client.key output_image.png server.crt server.key
root@4e8df03bc705:/volumes#
```

Task 2.a. Implement a simple TLS server

First, I modified the server.py by the new code that it uploaded in the TLS lab.

```

DianaN (Snapshot 1) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Activities Terminal Jan 11 13:52
myc
>>>
[1]+ Stopped python3
root@lccd085619ad:/volumes# python3 server.py
123
fdf
^Z
[2]+ Stopped python3 server.py
root@lccd085619ad:/volumes# python3 server.py
Traceback (most recent call last):
  File "server.py", line 21, in <module>
    sock.bind(('0.0.0.0', 4433))
OSError: [Errno 98] Address already in use
root@lccd085619ad:/volumes# openssl s_client -connect [website]:443
140316419945792:error:2008F002:BIOS routines:BIOLookup_ex:system lib:../crypto/bio
hostname
connect:errno=0
root@lccd085619ad:/volumes# python3 server.py

^Z
[3]+ Stopped python3 server.py
root@lccd085619ad:/volumes# netstat -tuln | grep 443
tcp      0      0 0.0.0.0:4433          0.0.0.0:*
tcp      0      0 0.0.0.0:443          0.0.0.0:*
LISTEN
LISTEN

```

To test this process; when the `cadir => from etc/ssl/certs`

```

DianaN (Snapshot 1) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Activities Terminal Jan 12 06:41
handshake.py
server.py
handshake.py
client.csr
client.key
handshake.py
output.image.p
1#!/usr/bin/env python3
2import socket, ssl, sys, pprint
3hostname = sys.argv[1]
4port = 443
5cadir = '/etc/ssl/certs'
6# Set up the TLS context
7context =
8    ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
9context.load_verify_locations(capath=cadir)

seed@VM: ~.../Labsetup
root@4e8df03bc705:/volumes# python3 handshake.py 10.9.0.43
^Z
[1]+ Stopped python3 handshake.py 10.9.0.43
root@4e8df03bc705:/volumes# python3 handshake.py www.google.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "handshake.py", line 18, in <module>
    sssock.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: unable to get local issuer certificate (ssl.c:1123)

```

When testing with the **system-wide certificates directory** (`/etc/ssl/certs`), the client failed to verify the server's certificate if the CA that signed the server's certificate was not included in this directory. This led to a "**certificate verify failed**" error unless the custom CA certificate was

manually added to the system's trusted store. However, adding a custom CA certificate to the system-wide store is not recommended for testing purposes, as it impacts all applications relying on the system's certificates, potentially introducing broader security risks or unintended consequences.

To test this process; when the `cadir => from client-certs`

```
#!/usr/bin/env python3
import socket, ssl, sys, pprint
hostname = sys.argv[1]
port = 443
cadir = 'client-certs'
# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)

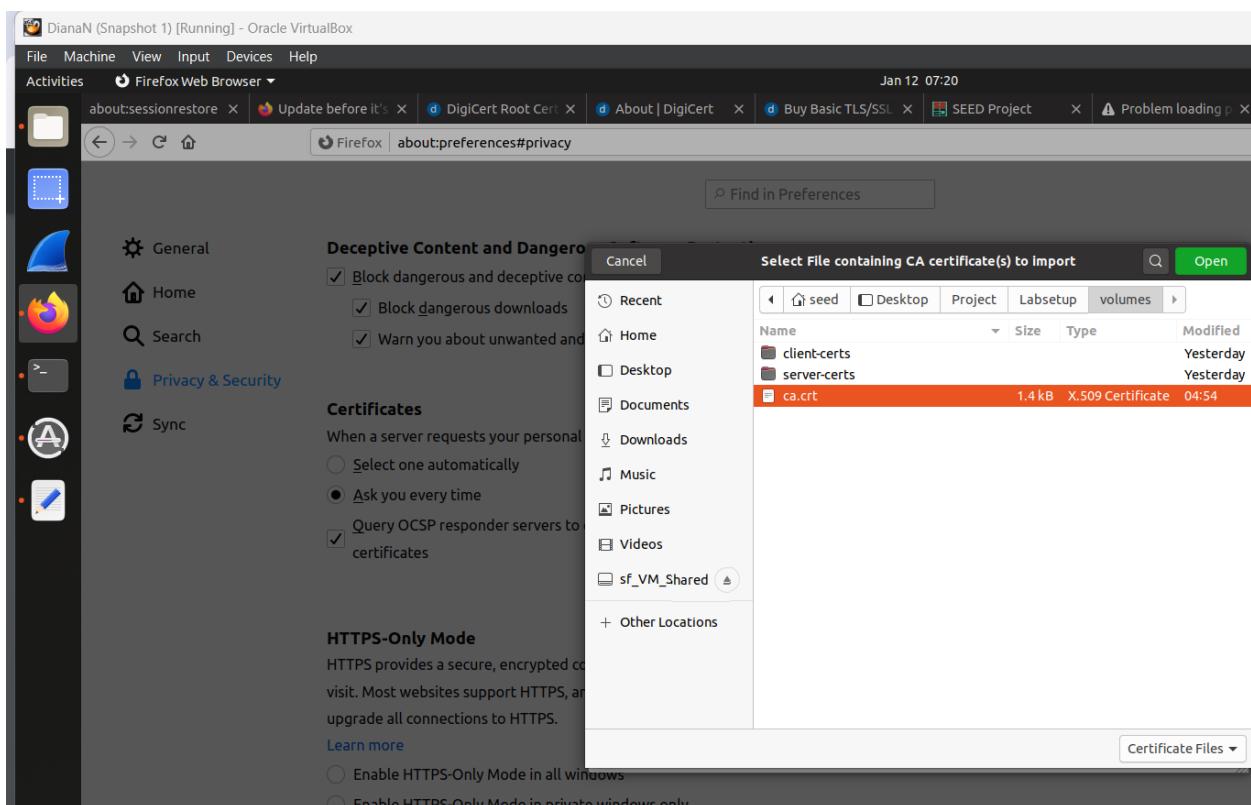
root@4e8df03bc705:/volumes# python3 handshake.py www.google.com
After making TCP connection. Press any key to continue ...
{'OCSP': ('http://o.pki.goog/wr2'),,
 'caIssuers': ('http://i.pki.goog/wr2.crt',),
 'crlDistributionPoints': ('http://c.pki.goog/wr2/o06nyr8F0m0.crl',),
 'issuer': (((('countryName', 'US'),),
             (('organizationName', 'Google Trust Services'),),
             (('commonName', 'WR2'),)),
            'notAfter': 'Mar 3 08:38:05 2025 GMT',
            'notBefore': 'Dec 9 08:38:06 2024 GMT',
            'serialNumber': 'D01C59292557A18010EE18C306607692',
            'subject': (((('commonName', 'www.google.com'),),
                         'subjectAltName': (('DNS', 'www.google.com'),),
            'version': 3}
After handshake. Press any key to continue ...
root@4e8df03bc705:/volumes#
```

The image demonstrates testing a TLS handshake with www.google.com using a Python script that loads CA certificates from a custom directory (`client-certs`). The script successfully establishes the connection, validates the certificate issued by Google Trust Services, and prints details such as the certificate's validity (December 9, 2024, to March 3, 2025) and subject (`www.google.com`). This approach showcases the use of a custom CA directory for enhanced security and control over certificate validation.

Task 2.b. Import the CA Certificate

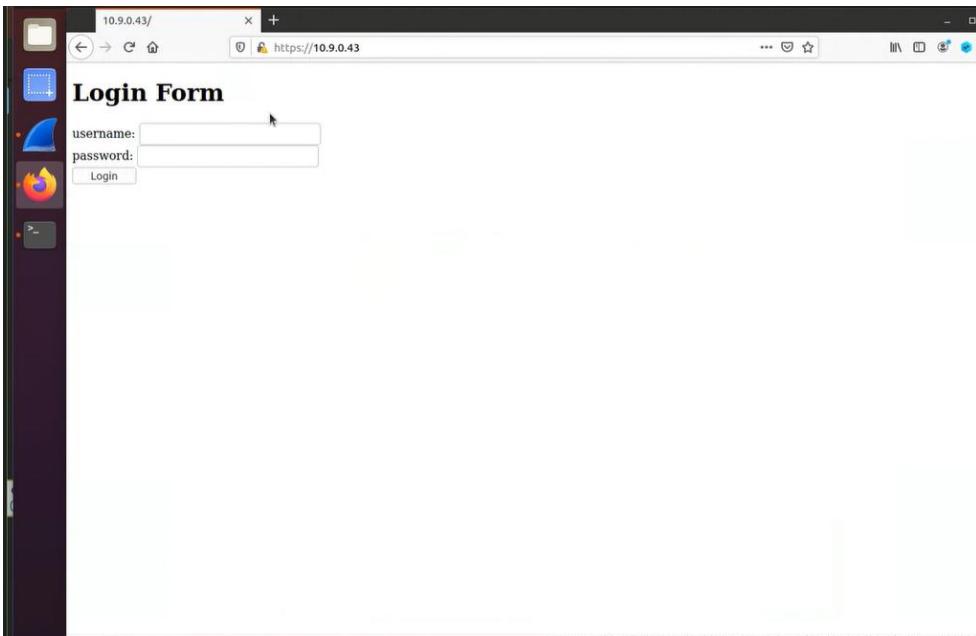
After server connection setup in port 443

1. In the Certificate Manager window,I go to the **Authorities** tab.
2. Navigate to the location where your **ca.crt** file is saved, Select the **ca.crt** file.
3. Imported it in this file.



After importing, CA appears in the list of certificate authorities in Firefox. This ensures that Firefox will trust certificates signed by this CA.

Now test it by Access the TLS Server: POST()



Task 2.c. Certificate with multiple names:

Create san_config.cnf file and add to it:

```
[ req ]
```

```
prompt = no
```

```
distinguished_name = req_distinguished_name
```

```
req_extensions = req_ext
```

SEED Labs 8

```
[ req_distinguished_name ]
```

```
C = US
```

```
ST = New York
```

```
L = Syracuse
```

O = XYZ LTD.

CN = www.bank32.com

[req_ext]

subjectAltName = @alt_names

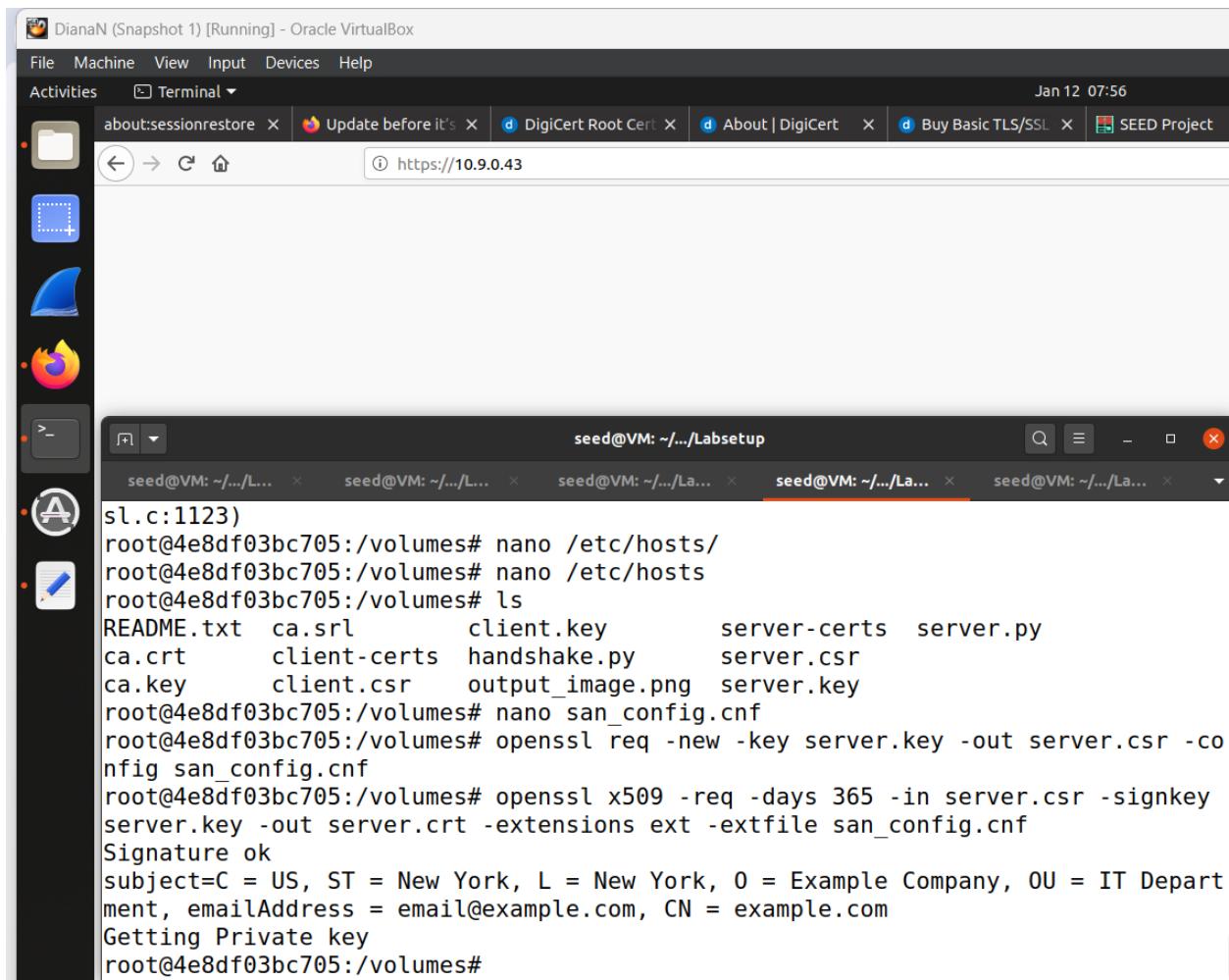
[alt_names]

DNS.1 = www.bank32.com

DNS.2 = www.example.com

DNS.3 = *.bank32.com

Generate a New CSR and Certificate with SAN:



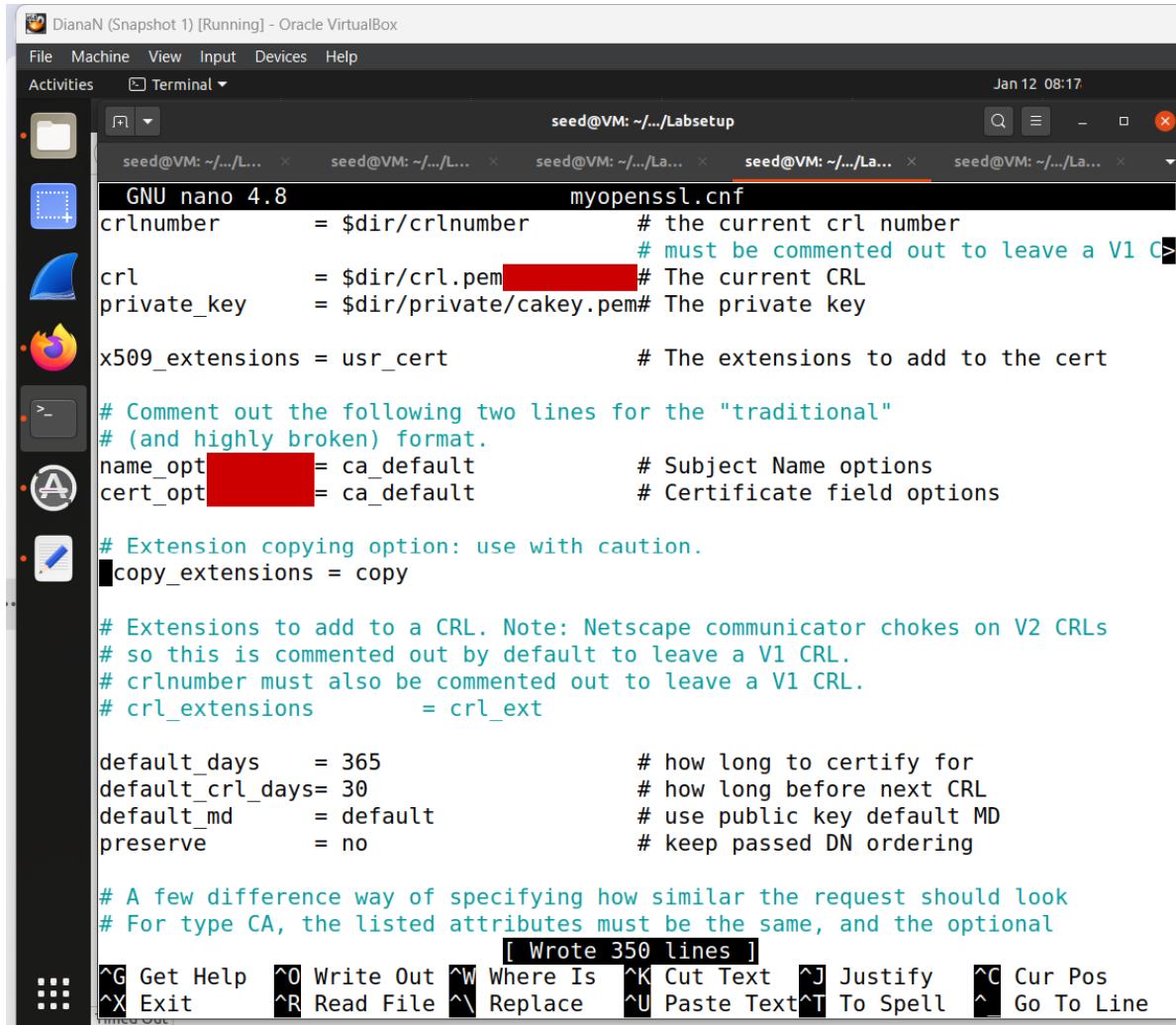
```
seed@VM: ~.../Labsetup
root@4e8df03bc705:/volumes# nano server_openssl.cnf
root@4e8df03bc705:/volumes# openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch -sha256 -keyout server.key -out server.csr
Can't open ./server_openssl.cnf for reading, No such file or directory
139904159098176:error:02001002:system library:fopen:No such file or directory...
/crypt/bio/bss_file.c:69:fopen('./server_openssl.cnf','r')
139904159098176:error:2006D080:BIO routines:BIO_new_file:no such file:.../crypt/bio/bss_file.c:76:
root@4e8df03bc705:/volumes# ls
README.txt  client-certs  output_image.png  server.crt
ca.crt      client.csr    san_config.cnf   server.csr
ca.key      client.key    server           server.key
ca.srl      handshake.py  server-certs    server.py
root@4e8df03bc705:/volumes# nano server_openssl.cnf
root@4e8df03bc705:/volumes# openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch -sha256 -keyout server.key -out server.csr
req: Error on line 5 of config file "./server_openssl.cnf"
140078901372224:error:0E079065:configuration file routines:def_load_bio:missing
equal sign:../crypto/conf/conf_def.c:391:line 5
root@4e8df03bc705:/volumes# nano server_openssl.cnf
root@4e8df03bc705:/volumes# nano server_openssl.cnf
root@4e8df03bc705:/volumes# openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch -sha256 -keyout server.key -out server.csr
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'server.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
root@4e8df03bc705:/volumes#
```

During the setup of SSL/TLS security for our server, we encountered an issue with the OpenSSL configuration file (`server_openssl.cnf`), which could not initially be accessed due to an incorrect file path or its absence. After locating the file and confirming its contents using the `nano` text editor, we successfully resolved the issue. We then executed the OpenSSL command to generate a 2048-bit RSA private key and a certificate signing request (CSR), securing the private key with a passphrase for added security. The CSR was created using the SHA-256 hashing algorithm to ensure strong cryptographic security. Resolving the file issue and completing the key and CSR generation were vital steps in configuring secure server communications.

```
root@4e8df03bc705:/volumes# cp /usr/lib/ssl/openssl.cnf ./myopenssl.cnf
root@4e8df03bc705:/volumes# nano myopenssl.cnf
root@4e8df03bc705:/volumes#
```

In my terminal session, I copied the default OpenSSL configuration file from its system directory ([/usr/lib/ssl/openssl.cnf](#)) to my current working directory, renaming it to [myopenssl.cnf](#). This allowed me to safely modify OpenSSL settings without affecting the global configuration, thus preventing potential disruptions to other system applications. Using Nano, a command-line text editor, I opened the renamed configuration file for customization, such as enabling specific certificate extensions essential for my SSL/TLS setup. This localized approach ensures that the changes are isolated and tailored to meet the specific requirements of my SSL configuration.

Modified ti by removing comment sign before copy_extention:



```
GNU nano 4.8          myopenssl.cnf
crlnumber      = $dir/crlnumber      # the current crl number
                # must be commented out to leave a V1 CRL
crl           = $dir/crl.pem        # The current CRL
private_key     = $dir/private/cakey.pem# The private key

x509_extensions = usr_cert          # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default        # Subject Name options
cert_opt         = ca_default        # Certificate field options

# Extension copying option: use with caution.
#copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
#crl_extensions      = crt_ext

default_days    = 365                  # how long to certify for
default_crl_days= 30                  # how long before next CRL
default_md      = default             # use public key default MD
preserve        = no                  # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
```

[Wrote 350 lines]

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

Generate the Certificate Using the Modified Configuration:

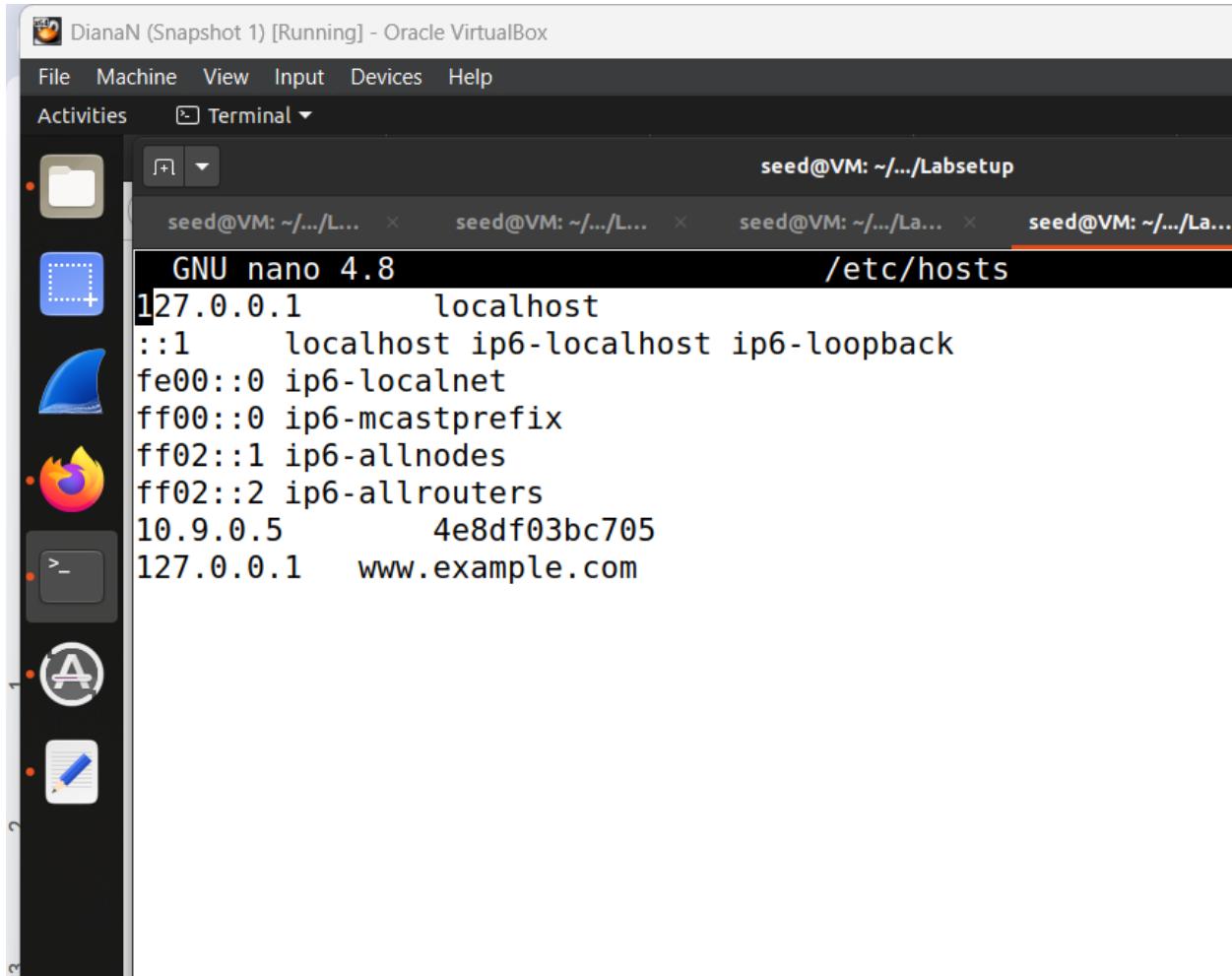
In my terminal session, I used the OpenSSL `ca` command to sign a Certificate Signing Request (CSR) with my Certificate Authority (CA) certificate to generate a server certificate (`server.crt`) valid for 3650 days, utilizing the SHA-256 hashing algorithm. The process referenced a custom configuration file, `./myopenssl.cnf`. During the signing, OpenSSL validated the CSR's signature for integrity but flagged a mismatch in the `countryName` field between the CA certificate (`PS`) and the CSR (`US`). While this mismatch did not block the signing process, it could potentially affect the certificate's trust level in environments requiring strict validation. Repeated execution of the command confirmed a consistent issue with the country code discrepancy, emphasizing the need to align the country details in both the CSR and CA certificate to maintain consistency and avoid potential trust issues in the future.

```
root@4e8df03bc705:/volumes# echo '1000' > ./demoCA/serial
root@4e8df03bc705:/volumes# openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch \
> -in server.csr -out server.crt \
> -cert ca.crt -keyfile ca.key
Using configuration from ./myopenssl.cnf
Check that the request matches the signature
Signature ok
The countryName field is different between
CA certificate (ps) and the request (US)
root@4e8df03bc705:/volumes# openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch \
> -in server.csr -out server.crt \
> -cert ca.crt -keyfile ca.key
Using configuration from ./myopenssl.cnf
Check that the request matches the signature
Signature ok
The countryName field is different between
CA certificate (ps) and the request (US)
root@4e8df03bc705:/volumes# openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch \
> -in server.csr -out server.crt \
> -cert ca.crt -keyfile ca.key
```

TASK3:A Simple HTTPS Proxy

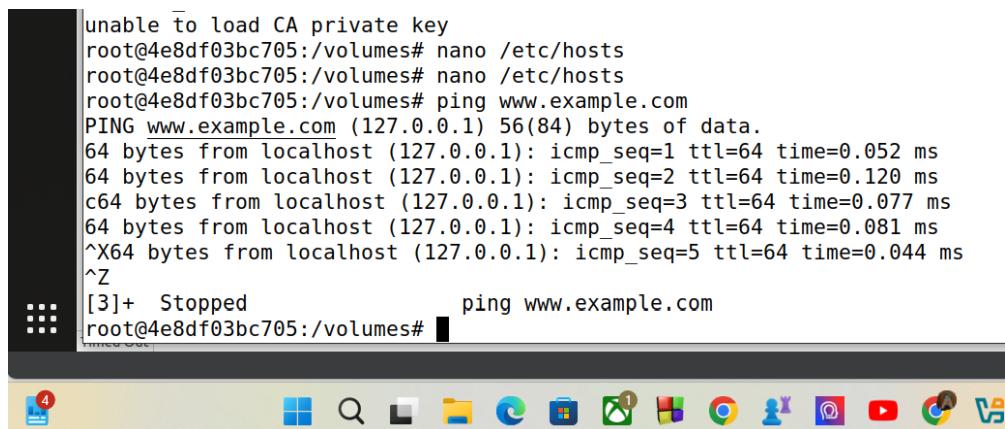
Modified the nano/hosts:

Map the domains that I want to intercept to our desired IP addresses. For testing a proxy or a MITM attack, I usually map these domains to your local machine's IP address (127.0.0.1).

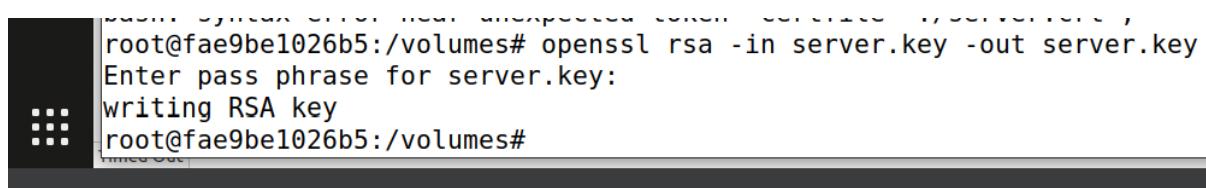


```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../L... x seed@VM: ~/.../L... x seed@VM: ~/.../La... x seed@VM: ~/.../La...
GNU nano 4.8 /etc/hosts
127.0.0.1      localhost
:1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5      4e8df03bc705
127.0.0.1      www.example.com
```

The i test it bu using ping:

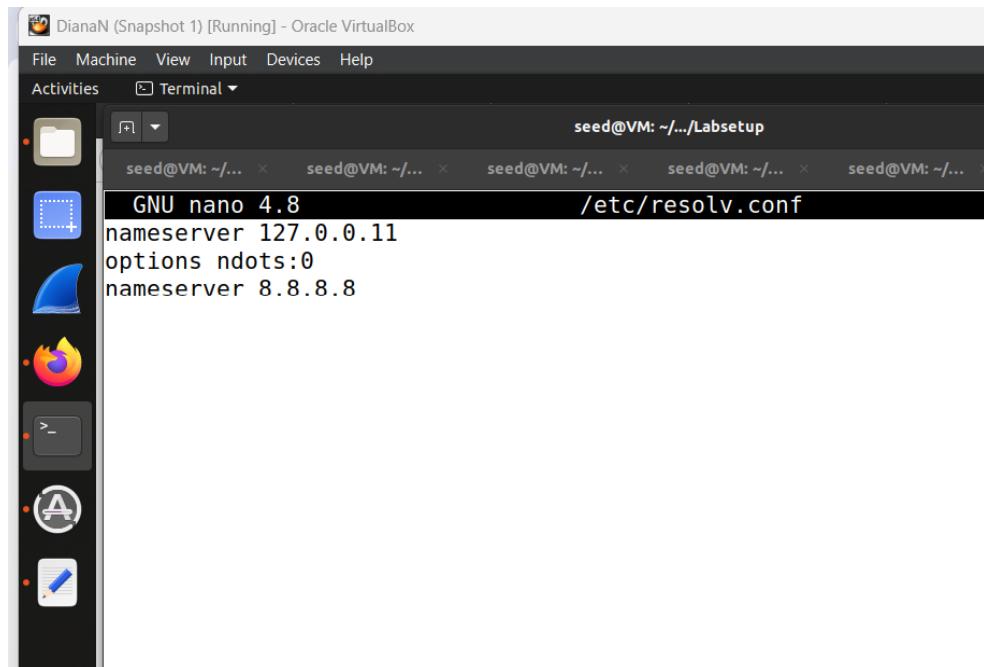


```
unable to load CA private key
root@4e8df03bc705:/volumes# nano /etc/hosts
root@4e8df03bc705:/volumes# nano /etc/hosts
root@4e8df03bc705:/volumes# ping www.example.com
PING www.example.com (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.120 ms
c64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.081 ms
^X64 bytes from localhost (127.0.0.1): icmp_seq=5 ttl=64 time=0.044 ms
^Z
[3]+  Stopped                  ping www.example.com
root@4e8df03bc705:/volumes#
```

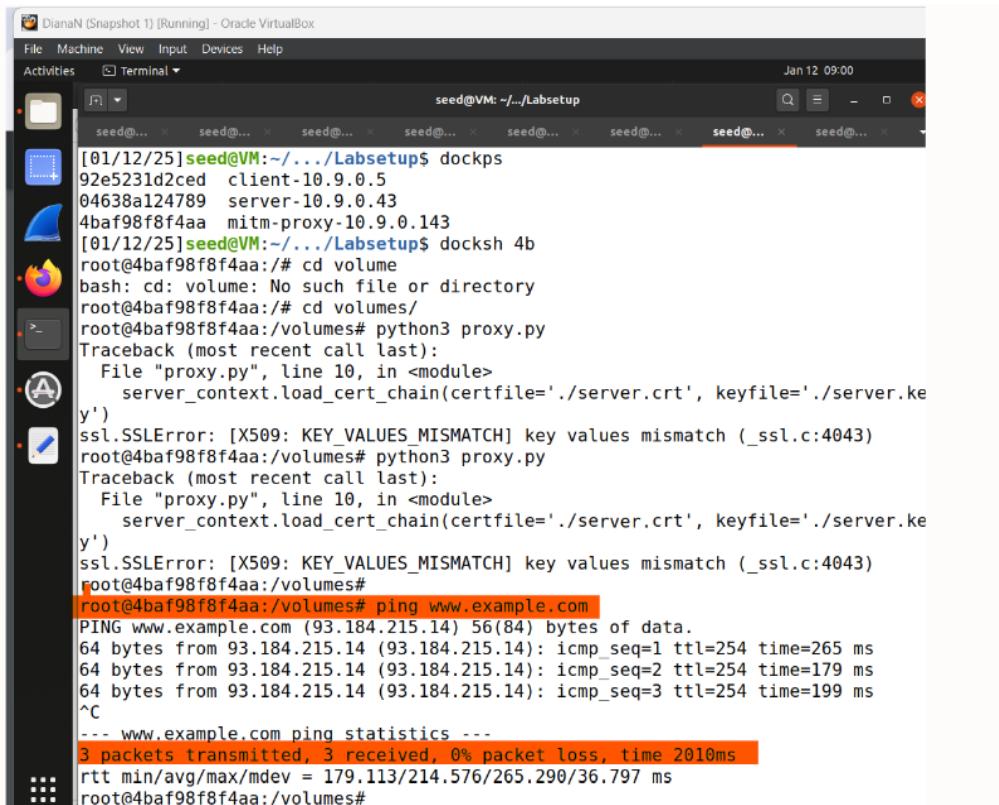


```
root@fae9be1026b5:/volumes# openssl rsa -in server.key -out server.key
Enter pass phrase for server.key:
writing RSA key
root@fae9be1026b5:/volumes#
```

Add nameserver 8.8.8.8 to /etc/resolv.conf:

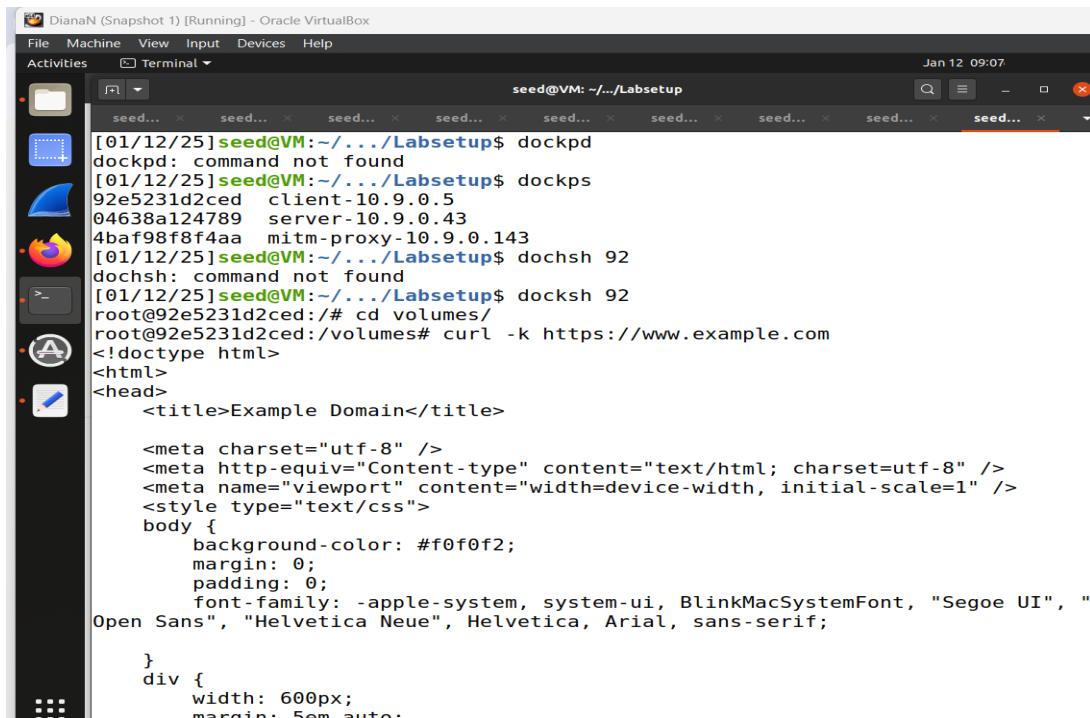


```
DianaN (Snapshot 1) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Activities Terminal
seed@VM: ~.../Labsetup
seed@VM: ~... x seed@VM: ~... x seed@VM: ~... x seed@VM: ~... x seed@VM: ~... x
GNU nano 4.8 /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
nameserver 8.8.8.8
```



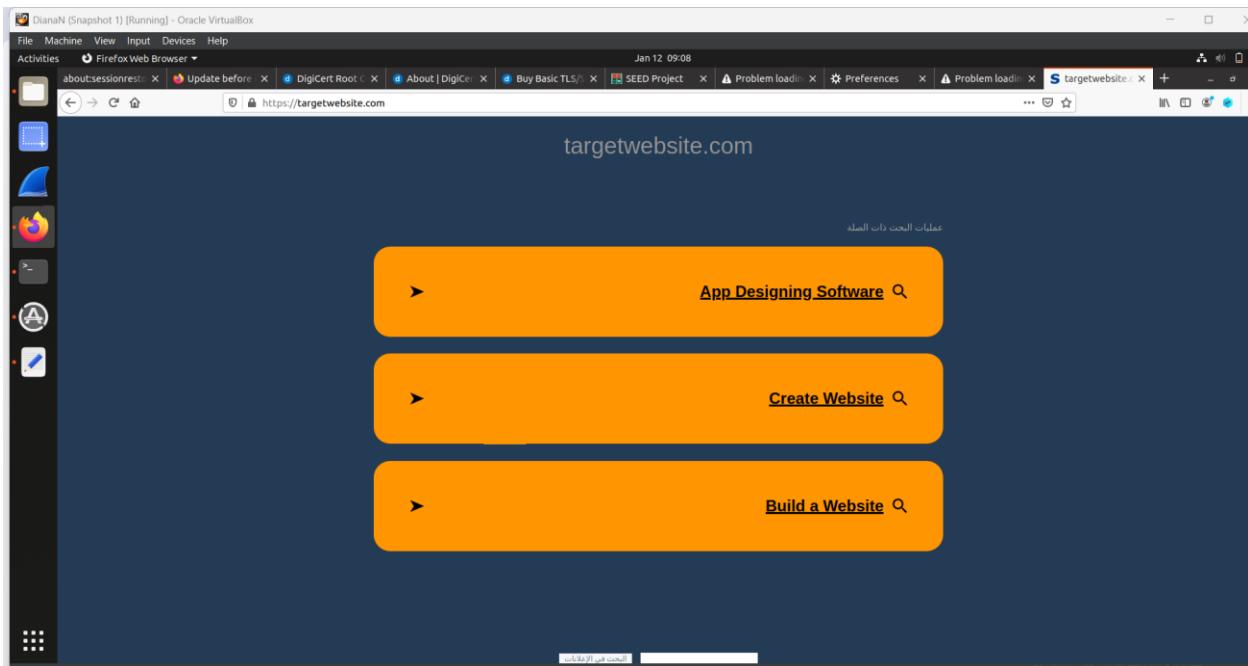
```
[01/12/25]seed@VM:~/.../Labsetup$ dockps
92e5231d2ced client-10.9.0.5
04638a124789 server-10.9.0.43
4baf98f8f4aa mitm-proxy-10.9.0.143
[01/12/25]seed@VM:~/.../Labsetup$ docksh 4b
root@4baf98f8f4aa:/# cd volume
bash: cd: volume: No such file or directory
root@4baf98f8f4aa:/# cd volumes/
root@4baf98f8f4aa:/volumes# python3 proxy.py
Traceback (most recent call last):
  File "proxy.py", line 10, in <module>
    server_context.load_cert_chain(certfile='./server.crt', keyfile='./server.key')
ssl.SSLError: [X509: KEY_VALUES_MISMATCH] key values mismatch (_ssl.c:4043)
root@4baf98f8f4aa:/volumes# python3 proxy.py
Traceback (most recent call last):
  File "proxy.py", line 10, in <module>
    server_context.load_cert_chain(certfile='./server.crt', keyfile='./server.key')
ssl.SSLError: [X509: KEY_VALUES_MISMATCH] key values mismatch (_ssl.c:4043)
root@4baf98f8f4aa:/volumes#
root@4baf98f8f4aa:/volumes# ping www.example.com
PING www.example.com (93.184.215.14) 56(84) bytes of data.
64 bytes from 93.184.215.14 (93.184.215.14): icmp_seq=1 ttl=254 time=265 ms
64 bytes from 93.184.215.14 (93.184.215.14): icmp_seq=2 ttl=254 time=179 ms
64 bytes from 93.184.215.14 (93.184.215.14): icmp_seq=3 ttl=254 time=199 ms
^C
--- www.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 179.113/214.576/265.290/36.797 ms
root@4baf98f8f4aa:/volumes#
```

On the **client container**, send an HTTPS request to your server:



```
[01/12/25]seed@VM:~/.../Labsetup$ dockpd
dockpd: command not found
[01/12/25]seed@VM:~/.../Labsetup$ dockps
92e5231d2ced client-10.9.0.5
04638a124789 server-10.9.0.43
4baf98f8f4aa mitm-proxy-10.9.0.143
[01/12/25]seed@VM:~/.../Labsetup$ dochsh 92
dochsh: command not found
[01/12/25]seed@VM:~/.../Labsetup$ docksh 92
root@92e5231d2ced:/# cd volumes/
root@92e5231d2ced:/volumes# curl -k https://www.example.com
<!doctype html>
<html>
<head>
<title>Example Domain</title>
<meta charset="utf-8" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style type="text/css">
body {
    background-color: #f0f0f2;
    margin: 0;
    padding: 0;
    font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
div {
    width: 600px;
    margin: 5em auto;
```

Add 10.9.0.143 targetwebsite.com to etc/hosts:



httpbin.org/forms/post × Preferences × +

← → C ⌂ https://httpbin.org/forms/post

Customer name:

Telephone:

E-mail address:

Pizza Size

Small

Medium

Large

Pizza Toppings

Bacon

Extra Cheese

Onion

Mushroom

Preferred delivery time: : :

Delivery instructions:

\Test httpbin.org and we noticed that the proxy can take the data when we enter it and this is very high risk.

Customer name:

Telephone:

E-mail address:

Pizza Size

Small

Medium

Large

Pizza Toppings

Bacon

Extra Cheese

Onion

Mushroom

Preferred delivery time:

Delivery instructions:

```
seed@VM: ~/Labsetup
seed@VM: ~... > seed@VM: ~...
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1

[*] Handling connection for: httpbin.org
POST /post HTTP/1.1
Host: httpbin.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
Origin: https://httpbin.org
Connection: keep-alive
Referer: https://httpbin.org/forms/post
Upgrade-Insecure-Requests: 1

custname=Diana&custtel=8932p99&custemail=diana@gmail.com&size=large&topping=bac
```

As we see that the proxy can catch the data.

In this task, I executed a Man-in-the-Middle (MITM) attack using an HTTPS proxy. To start, I modified the /etc/hosts file on my virtual machine to redirect traffic from targetwebsite.com to the MITM proxy container's IP address (10.9.0.143). This enabled interception of traffic originally intended for the target website. After setting up the proxy server within the MITM container, I tested the configuration by accessing the target website in a browser. The redirection worked as expected, and the proxy successfully intercepted HTTP and HTTPS traffic.

To further demonstrate the attack, I tested form submissions on the intercepted webpage. By submitting sample form data, I observed that the proxy captured sensitive information such as the customer's name, telephone number, email address, and preferences in plaintext. This highlighted the severe security risks associated with compromised or redirected traffic in a MITM scenario.

Finally, I documented the results and ensured proper cleanup by restoring the original state of the /etc/hosts file, removing any changes made during the attack. This task underscores the importance of secure traffic handling and the vulnerabilities of unprotected networks.

Appendices:

Code for Task1.d:

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

import socket
import ssl
import sys
import pprint

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

# cadir = './client-certs'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu 20.04 VM
# context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)    # For Ubuntu 16.04 VM

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

# Create TCP connection
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((hostname, port))
input("After making TCP connection. Press any key to continue ...")

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
```

```

ssock.do_handshake() # Start the handshake
print("==== Cipher used: {}".format(ssock.cipher()))
print("==== Server hostname: {}".format(ssock.server_hostname))
print("==== Server certificate:")
pprint.pprint(ssock.getpeercert())
pprint.pprint(context.get_ca_certs())
input("After TLS handshake. Press any key to continue ...")

# Send HTTP Request to Server
request = b"GET / 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)

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

```

=====

server.py modified code:
`#!/usr/bin/env python3`

```

import socket
import ssl

# HTML content to be sent to the client
html = """
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
<DOCTYPE html><html><body><h1>Hello, World!</h1></body></html>
"""

```

```

# Paths to the server certificate and private key
SERVER_CERT = './server-certs/server.crt' # Adjust the path as needed
SERVER_PRIVATE = './server-certs/server.key'

# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

# Create a socket for the server
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock.bind(('0.0.0.0', 443)) # Bind to port 443 (HTTPS)
sock.listen(5)

while True:
    newsock, fromaddr = sock.accept()
    try:
        # Wrap the socket with TLS
        ssock = context.wrap_socket(newsock, server_side=True)
        print("TLS connection established!")

        # Read data from the client
        data = ssock.recv(1024)
        print(f"Request: {data.decode('utf-8')}")

        # Send HTML response to the client
        ssock.sendall(html.encode('utf-8'))

        # Close the TLS connection
        ssock.shutdown(socket.SHUT_RDWR)
        ssock.close()
    except Exception as e:
        print(f"Error: {e}")
        continue

```