

# TCSS 481

Midterm Lab

Dustin Ray – [dustiv2@uw.edu](mailto:dustiv2@uw.edu)

## 3.1 Task 1.a: TLS handshake

The objective of this task is to instantiate a TLS connection from self-hosted client software to an arbitrary web server. The first step in this process is to establish a handshake with the web server. We use provided code to connect to an HTTPS web server:

```
[11/22/21]seed@VM:~/../volumes$ python3 '/home/seed/Documents/TLS Lab/Labsetup/volumes/handshake.py'
After making TCP connection. Press any key to continue ...
```

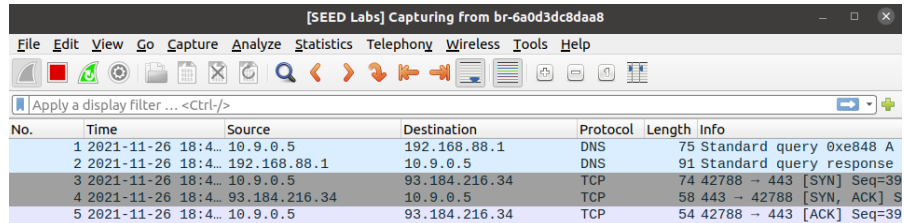
After making the connection, we retrieve the following certificate from the server verifying its authenticity:

```
=== Server certificate:
{'OCSP': ('http://ocsp.digicert.com',),
 'caIssuers': ('http://cacerts.digicert.com/DigiCertTLRSASHA2562020CA1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertTLRSASHA2562020CA1.crl',
                           'http://crl4.digicert.com/DigiCertTLRSASHA2562020CA1.crl'),
 'issuer': (((('countryName', 'US'),),
               (('organizationName', 'DigiCert Inc'),),
               (('commonName', 'DigiCert TLS RSA SHA256 2020 CA1'),)),),
 'notAfter': 'Dec 25 23:59:59 2021 GMT',
 'notBefore': 'Nov 24 00:00:00 2020 GMT',
 'serialNumber': '0FBE08B0854D05738AB0CCE1C9AFEEC9',
 'subject': (((('countryName', 'US'),),
                 (('stateOrProvinceName', 'California'),),
                 (('localityName', 'Los Angeles'),),
                 (('organizationName',
                  'Internet Corporation for Assigned Names and Numbers'),),
                 (('commonName', 'www.example.org'),)),),
 'subjectAltName': (('DNS', 'www.example.org'),
                    ('DNS', 'example.com'),
                    ('DNS', 'example.edu'),
                    ('DNS', 'example.net'),
                    ('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 CA'),)),),
 'notAfter': 'Nov 10 00:00:00 2031 GMT',
 'notBefore': 'Nov 10 00:00:00 2006 GMT',
 'serialNumber': '083BE056904246B1A1756AC95991C74A',
 'subject': (((('countryName', 'US'),),
                 (('organizationName', 'DigiCert Inc'),),
                 (('organizationalUnitName', 'www.digicert.com'),),
                 (('commonName', 'DigiCert Global Root CA'),)),),
 'version': 3}]
```

The purpose of the `/etc/ssl/certs` directory is to hold the certificates for well known and trusted authorities. This moves the authenticity check offline and onto the client side of the handshake which reduces the cost of the handshake procedure and also provides an additional layer of security in that if a connection is ever compromised, the system has a backup of authentic authority certificates that it can rely on.

There are two initial steps in the handshake software. The first is to establish a TCP connection with the server, this happens in the following step:

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



The image shows a Wireshark capture window titled "[SEED Labs] Capturing from br-6a0d3dc8daa8". The capture list table shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-11-26 18:4...	10.9.0.5	192.168.88.1	DNS	75	Standard query 0xe848 A
2	2021-11-26 18:4...	192.168.88.1	10.9.0.5	DNS	91	Standard query response
3	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	74	42788 → 443 [SYN] Seq=39
4	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TCP	58	443 → 42788 [SYN, ACK] S
5	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39

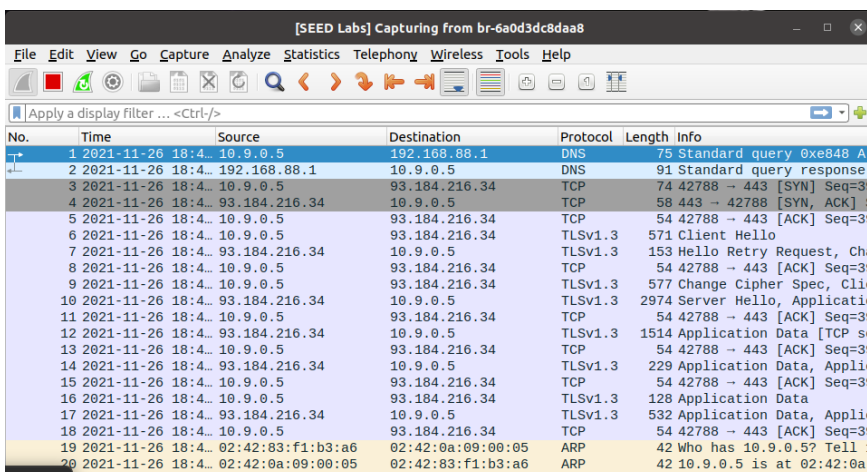
Line 23 shows where the network socket on the local machine is being connected to the host name, in this case [www.example.com](http://www.example.com). The port object is defined previous to be port 443, which is the default port in which HTTPS, SSL, and TLS connections are initiated. Port 80 is reserved for unencrypted HTTP traffic. The TCP connection is part of the transport layer of the Internet Protocol stack and provides reliable, ordered, and error-checked delivery of byte streams. TLS operates in the application layer which is built upon the transport layer, and so a successful TLS handshake depends entirely on a successful TCP connection. The following step initiates the TLS connection:

```
26 # Add the TLS
27 ssock = context.wrap_socket(sock, server_hostname=hostname,
28                             do_handshake_on_connect=False)
29 ssock.do_handshake() # Start the handshake
```

From the python 3 documentation, the `wrap_socket` function performs the following:

Takes an instance `sock` of `socket.socket`, and returns an instance of `ssl.SSLSocket`, a subtype of `socket.socket`, which wraps the underlying socket in an SSL context. `sock` must be a `SOCK_STREAM` socket; other socket types are unsupported.

And wireshark shows the handshake in progress:



The image shows a Wireshark capture window titled "[SEED Labs] Capturing from br-6a0d3dc8daa8". The capture list table shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-11-26 18:4...	10.9.0.5	192.168.88.1	DNS	75	Standard query 0xe848 A
2	2021-11-26 18:4...	192.168.88.1	10.9.0.5	DNS	91	Standard query response
3	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	74	42788 → 443 [SYN] Seq=39
4	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TCP	58	443 → 42788 [SYN, ACK] S
5	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
6	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TLSv1.3	571	Client Hello
7	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TLSv1.3	153	Hello Retry Request, Cha
8	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
9	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TLSv1.3	577	Change Cipher Spec, Clie
10	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TLSv1.3	2974	Server Hello, Applicatio
11	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
12	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TLSv1.3	1514	Application Data [TCP se
13	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
14	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TLSv1.3	229	Application Data, Applic
15	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
16	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TLSv1.3	128	Application Data
17	2021-11-26 18:4...	93.184.216.34	10.9.0.5	TLSv1.3	532	Application Data, Applic
18	2021-11-26 18:4...	10.9.0.5	93.184.216.34	TCP	54	42788 → 443 [ACK] Seq=39
19	2021-11-26 18:4...	02:42:83:f1:b3:a6	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 1
20	2021-11-26 18:4...	02:42:0a:09:00:05	02:42:83:f1:b3:a6	ARP	42	10.9.0.5 is at 02:42:0a:

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

The previous handshake made use the system CA directory that is increasingly shipped client-side with an OS installation for reasons described previously. We can demonstrate the importance of this design by modifying the directory used in the handshake software:

```
10 cadir = '/etc/ssl/certs'
11 #cadir = './client-certs'
```

The handshake software will now use the directory established on line 10 as the source of the certificate authorities that it uses for server verification instead of the system certificate authority directory. We receive the following result after making this change:

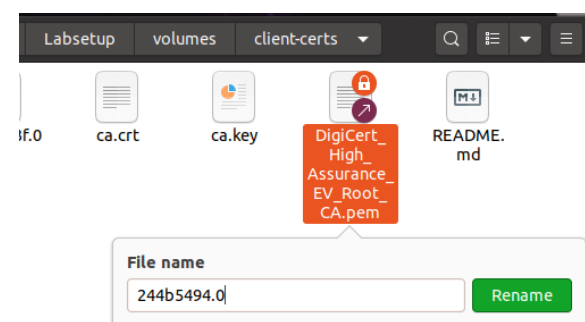
```
self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED]
certificate verify failed: self signed certificate in certifica
te chain (_ssl.c:1123)
```

The handshake software is attempting to find the CA for the provided host name, but it is unable to. We first need to ascertain the proper certificate to find in the system folder, we can search the certificate we obtained previously to find this information:

```
'version': 3}]
[{'issuer': (((('countryName', 'US'),),
               (('organizationName', 'DigiCert Inc'),),
               (('organizationalUnitName', 'www.digicert.com'),),
               (('commonName', 'DigiCert High Assurance EV Root CA'))),),
  'notAfter': 'Nov 10 00:00:00 2031 GMT',
  'notBefore': 'Nov 10 00:00:00 2006 GMT',
  'serialNumber': '02AC5C266A0B409B8F0B79F2AE462577',
  'subject': (((('countryName', 'US'),),
                 (('organizationName', 'DigiCert Inc'),),
                 (('organizationalUnitName', 'www.digicert.com'),),
                 (('commonName', 'DigiCert High Assurance EV Root CA'))),),
  'version': 3}]
After TLS handshake. Press any key to continue ...
[11/22/21] seed@VM: ~/.../volumes$
```

We need the DigiCert High Assurance EV Root CA, but the client software is looking for a file with the hash of the subject of section of the certificate.

```
seed@VM: ~/.../client-certs
[11/22/21] seed@VM: ~/.../client-certs$ openssl x509 -in '/home/seed/Documents/TLS
Lab/Labsetup/volumes/client-certs/DigiCert_Assured_ID_Root_CA.pem' -noout -sub
ject_hash
b1159c4c
```



The connection is successful following the placement of the CA with the name changed to a hash into the client-certs folder:

```
'version': 3}
[{'issuer': (((('countryName', 'US')),
                (('organizationName', 'DigiCert Inc')),
                (('organizationalUnitName', 'www.digicert.com')),
                (('commonName', 'DigiCert High Assurance EV Root CA'))),
  'notAfter': 'Nov 10 00:00:00 2031 GMT',
  'notBefore': 'Nov 10 00:00:00 2006 GMT',
  'serialNumber': '02AC5C266A0B409B8F0B79F2AE462577',
  'subject': (((('countryName', 'US')),
                  (('organizationName', 'DigiCert Inc')),
                  (('organizationalUnitName', 'www.digicert.com')),
                  (('commonName', 'DigiCert High Assurance EV Root CA'))),
  'version': 3}]
After TLS handshake. Press any key to continue ...
[11/22/21]seed@VM:~/.../volumes$
```

And so we repeat the process for two different websites: (The previous example was for facebook.com)

Here we have example.com:

```
((('commonName', 'DigiCert Global Root CA'))),
': 'Nov 10 00:00:00 2031 GMT',
e': 'Nov 10 00:00:00 2006 GMT',
mber': '083BE056904246B1A1756AC95991C74A',
: (((('countryName', 'US')),
      (('organizationName', 'DigiCert Inc')),
      (('organizationalUnitName', 'www.digicert.com')),
      (('commonName', 'DigiCert Global Root CA'))),
```

And nasa.gov:

```
hostname = "www.nasa.gov"
port = 443
#cadir = '/etc/ssl/certs'
l_cadir = './client-certs'
```

```
[{'issuer': (((('countryName', 'US')),
                (('organizationName', 'Amazon')),
                (('commonName', 'Amazon Root CA 1'))),
  'notAfter': 'Jan 17 00:00:00 2038 GMT',
  'notBefore': 'May 26 00:00:00 2015 GMT',
  'serialNumber': '066C9FCF99BF8C0A39E2F0788A43E696365BCA',
  'subject': (((('countryName', 'US')),
                  (('organizationName', 'Amazon')),
                  (('commonName', 'Amazon Root CA 1'))),
  'version': 3}]
```

```
[11/22/21]seed@VM:~/.../volumes$ openssl x509 -in '/home/seed/Documents/TLS Lab/Labsetup/volumes
/client-certs/Amazon_Root_CA_1.pem' -noout -subject_hash
ce5e74ef
```



```
[{'issuer': (((('countryName', 'US')),
                (('organizationName', 'Amazon')),
                (('commonName', 'Amazon Root CA 1'))),
  'notAfter': 'Jan 17 00:00:00 2038 GMT',
  'notBefore': 'May 26 00:00:00 2015 GMT',
  'serialNumber': '066C9FCF99BF8C0A39E2F0788A43E696365BCA',
  'subject': (((('countryName', 'US')),
                  (('organizationName', 'Amazon')),
                  (('commonName', 'Amazon Root CA 1'))),
  'version': 3}]
After TLS handshake. Press any key to continue ...
[11/22/21]seed@VM:~/.../volumes$
```

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

Here we show the results of mapping a hostname to an arbitrary IP address. This can cause unsafe situations if attackers are able to reroute legitimate domain names to addresses of their choosing. We map the domain example2020.com to the IP address of example.com:

```
[11/22/21]seed@VM:~/.../Labsetup$ docksh 2e82b5269a3d
root@2e82b5269a3d:/# sudo nano /etc/hosts
```

```
seed@VM: ~/.../Labse
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    2e82b5269a3d
93.184.216.34 www.example2020.com
```

We then attempt the handshake and receive the following:

```
Traceback (most recent call last):
  File "handshake.py", line 29, in <module>
    ssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify
y failed: Hostname mismatch, certificate is not valid for 'www.example2020.com'.
(ssl.c:1123)
root@2e82b5269a3d:/volumes#
```

If we disable the host name check in our handshake program, the following occurs:

```
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/DigiCertTLRSASHA2562020CA1.crt',),
 'crlDistributionPoints': ('http://crl3.digicert.com/DigiCertTLRSASHA2562020CA1
.crl',
                          'http://crl4.digicert.com/DigiCertTLRSASHA2562020CA1
.crl'),
 'issuer': (((('countryName', 'US'),),
              (('organizationName', 'DigiCert Inc'),),
              (('commonName', 'DigiCert TLS RSA SHA256 2020 CA1'))),),
 'notAfter': 'Dec 25 23:59:59 2021 GMT',
 18 context.verify_mode = ssl.CERT_REQUIRED
 19 context.check_hostname = False
 20
 21 # Create TCP connection
```

What is being demonstrated here is the importance of the host name check during the TLS handshake process. It is crucial to use cryptographic authentication measures in order to verify that the connection being made to a given host is legitimate, otherwise as we have demonstrated in previous experiments, it is possible to direct traffic to malicious servers with ease.

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

It is also possible to send and receive arbitrary data during a handshake. Using the `sendall` and `recv` commands, we can specify how much data we wish to send and receive in the interaction with the server:

```
37 # Send HTTP Request to Server
38 request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode('utf-8') + b"\r\n\r\n"
39 ssock.sendall(request)
40 # Read HTTP Response from Server
41 response = ssock.recv(2048)
42 while response:
43     pprint.pprint(response.split(b"\r\n"))
44     response = ssock.recv(2048)
45
46
47 # Close the TLS Connection
48 ssock.shutdown(socket.SHUT_RDWR)
49 ssock.close()
```

Here we are sending the request string on line 38, and receiving in return a response from the server of size 2048:

```
After TLS handshake. Press any key to continue ...
[b'HTTP/1.0 200 OK',
 b'Age: 595299',
 b'Cache-Control: max-age=604800',
 b'Content-Type: text/html; charset=UTF-8',
 b'Date: Tue, 23 Nov 2021 01:28:31 GMT',
 b'Etag: "3147526947+ident"',
 b'Expires: Tue, 30 Nov 2021 01:28:31 GMT',
 b'Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT',
 b'Server: ECS (sec/9717)',
 b'Vary: Accept-Encoding',
 b'X-Cache: HIT',
 b'Content-Length: 1256',
 b'Connection: close',
 b'',
 b'']
[b'<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    '
 b'    <meta charset="utf-8" />\n    <meta http-equiv="Content-type" content="t'
 b'ext/html; charset=utf-8" />\n    <meta name="viewport" content="width=dev'
 b'ice-width, initial-scale=1" />\n    <style type="text/css">\n        body {\n
 b'            background-color: #f0f0f2;\n                margin: 0;\n                padding: 0;\n'
 b'            font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI'
 b'", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;\n'
 b'\n        }\n        div {\n            width: 600px;\n                margin: 5em auto;\n'
 b'            padding: 2em;\n                background-color: #fdfdff;\n                border-rad'
 b'ius: 0.5em;\n                box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n        }\n'
 b'        a:link, a:visited {\n                color: #38488f;\n                text-decoration:'
 b'        none;\n        }\n        @media (max-width: 700px) {\n            div {\n                m'
 b'argin: 0 auto;\n                width: auto;\n                }\n        }\n    </style>'
 b'\n</head>\n<body>\n<div>\n    <h1>Example Domain</h1>\n    <p>This domai'
 b'n is for use in illustrative examples in documents. You may use this\n'
 b'    domain in literature without prior coordination or asking for permission.</'
 b'p>\n    <p><a href="https://www.iana.org/domains/example">More informatio'
 b'n...</a></p>\n</div>\n</body>\n</html>\n']
```

The response is an HTML document which is the landing page of the server. We are only sent this information after a successful handshake, and this process would happen the same way in a web browser when we make a successful connection and subsequent page load.



## 4 Task 2: TLS Server

We move on to create a simple TLS server using the CA details created in previous documents. The process is outlined below.

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

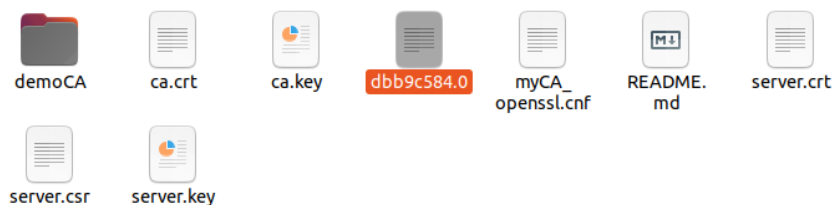
We proceed to implement a simple server that is capable of creating a TLS connection between a itself and a client application. We are provided with server software to accomplish this task. We will create a server that maps the address of the hosting container to <https://www.ray2021.com>. Since authentication is a critical part of this process, we must ensure that the server is able to present the client with the correct certificate for this domain, so we modify the certificate directory in the server program as follows:

```
11
12 SERVER_CERT = './server-certs/server.crt'
13 SERVER_PRIVATE = './server-certs/server.key'
14
```

The server will now present the calling client with the correct certificate for <https://www.ray2021.com>, but it is also necessary to ensure that the client has a copy of the valid certificate to look for:

```
,
8 hostname = 'www.ray2021.com'
9 port = 443
10 #cadir = '/etc/ssl/certs'
11 cadir = './client-certs'|
12
```

We also ensure that the CA is renamed to the subject hash accordingly, so it can be found by the openssl library in use by the server and the handshake:





If we do not store the proper certificates in the correct location and direct the server and client software to this directory, then certificate verification will fail, and this result is displayed in the server software:

```
root@9915fdd4dbbc:/volumes# python3 server.py
Enter PEM pass phrase:
attempting TLS connection...
TLS connection fails
attempting TLS connection...
TLS connection fails
attempting TLS connection...
TLS connection fails
```

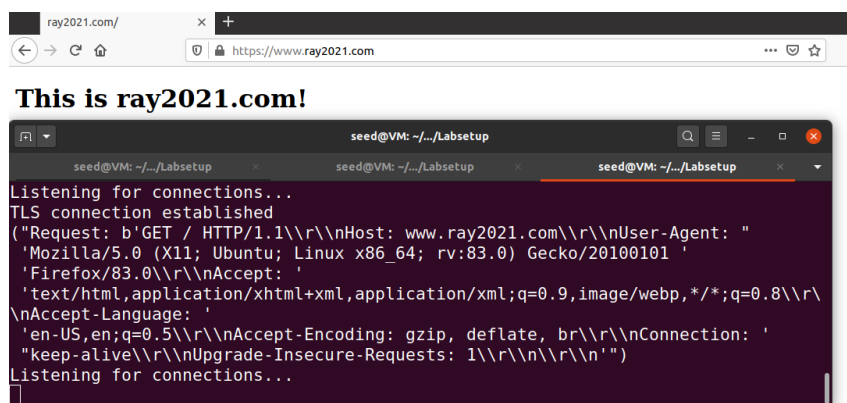
When the certificates are correctly configured within the client and the server, then we receive the following result:

```
TLS connection established
"Request: b'GET / HTTP/1.0\\r\\nHost: www.ray2021.com\\r\\n\\r\\n'"
```

The connection was established successfully. Due to the request for a certain amount of data that we had included within our handshake, we are returned with the header shown in the above image.

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

We can repeat the same process as above, replacing the client container with a web browser. This requires correct configuration of the certificates for the server and the CA. We import the authority certificate in the browser, and make a connection to our domain, receiving the following result:



We have made a secure HTTPS connection to our domain that is being hosted in the server container.

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

It is also significant to demonstrate the utility of hosting multiple domains that point to the same web server. This is made possible by ensuring to include these alternate domains within the certificate generation and signing process. It must be noted that the original domain is included in the list of alternates during this process. We start as we usually do in this process by modifying the system hosts file to map the chosen domains to the address of the web server:

```
# For TLS Lab
10.9.0.43      www.ray2021.com
10.9.0.43      www.ray2021a.com
10.9.0.43      www.ray2021b.com
```

Next we proceed to generate an authority signature request for the domain and its alternate names:

```
[11/23/21]seed@VM:~/.../client-certs$ openssl req -newkey rsa:2048 -sha256 -keyout server.key -out server.csr -subj "/CN=www.ray2021.com/O=ray2021 Inc./C=US" -passout pass:dees -addext "subjectAltName = DNS:www.ray2021.com, DNS:www.ray2021a.com, DNS:www.ray2021b.com"
Generating a RSA private key
+++++
.....+++++
writing new private key to 'server.key'
-----
```

After signing the request, we inspect the new certificate to ensure that the alternate names have been included:

```
X509v3 Subject Alternative Name:
    DNS:www.ray2021.com, DNS:www.ray2021a.com, DNS:www.ray2021b.com
Certificate is to be certified until Nov 22 00:51:31 2031 GMT (3650 days)
Write out database with 1 new entries
```

And then proceed to connect to these alternate domains through our browser:



**This is ray2021.com!**



**This is ray2021.com!**

We have shown how to host multiple domains that point to the same address and how to establish secure TLS connections to those domains.

## 5 Task 3: A Simple HTTPS Proxy

An HTTPS proxy can serve as useful tool that can add privacy and security to internet connections. In principle, an HTTPS proxy sits between a client and a server, relaying the traffic between the two. The server knows nothing of the client because it only communicates with the proxy. A TLS connection is established in the usual way from the client to the proxy, and then from the proxy to the webserver.

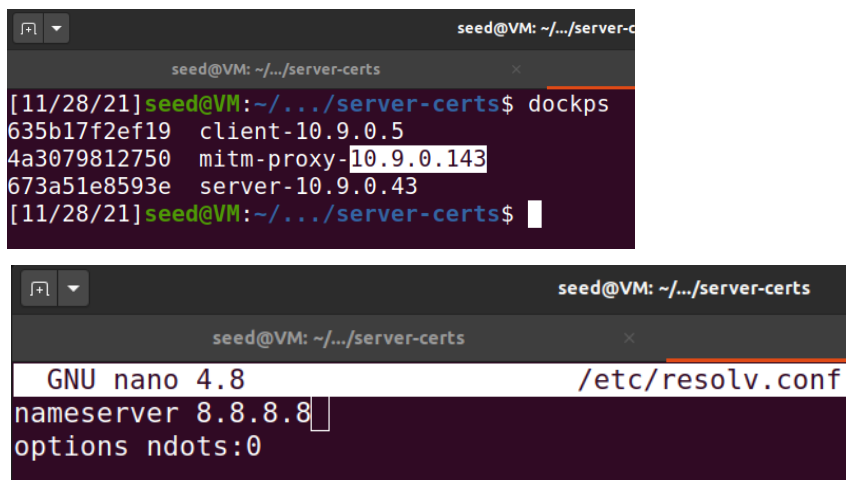
In a scenario in which the private key of a CA is compromised, a proxy could also be used to carry out a man in the middle attack. This is possible in this scenario because the proxy can use the private key of the issuing CA to create arbitrary valid certificates. The TLS connection to the proxy will succeed with a valid certificate on the server side, and thus the proxy can deliver any traffic it chooses back to the client. An attacker could carry out a DNS cache poisoning attack as we have demonstrated previously, and in doing so could route the client's request of a valid domain to their own server. The attacker could then establish a TLS connection to the legitimate web server. If the attacker presents the client with a fake version of the webserver, say a social media site, then it would be possible for them to trick the client into entering their account credentials. This is only one of the reasons why many popular websites will enforce identity checks through 2fa or otherwise when a connection is made to their services on a given account from an unrecognized IP address.

The scenario of a man in the middle proxy attack allows an adversary to place themselves between the client and the webserver, intercepting, analyzing, and modifying the traffic in transit as they please. Assuming the CA of a given webserver is not compromised, then as part of the TLS handshake with the proxy, the client will attempt to verify the identity of the proxy against the CA purportedly used to authenticate the proxy. The process will fail if the client does not have a copy of the malicious CA in its certificate root to verify the proxy with. If the malicious CA is present somehow in the client system, then this verification will be successful and the client will make a valid TLS connection to the malicious proxy.

We proceed to construct an HTTPS proxy in the following fashion:

1. We configure the client to connect by TLS to the proxy.
2. We configure the proxy to connect by TLS to the server.
3. The client will send an HTTP request to the proxy.
4. The proxy will forward the HTTP request to the server.
5. The server will return the request to the proxy.
6. The proxy will return the request to the client.

### Part a.) Connection from client to proxy to a real webserver:



The first screenshot shows a terminal window with the command `dockps` being executed. The output lists three containers: `client-10.9.0.5`, `mitm-proxy-10.9.0.143`, and `server-10.9.0.43`. The second screenshot shows the `/etc/resolv.conf` file being edited with `nameserver 8.8.8.8` and `options ndots:0`.

```
seed@VM: ~/.../server-certs
[11/28/21]seed@VM:~/.../server-certs$ dockps
635b17f2ef19 client-10.9.0.5
4a3079812750 mitm-proxy-10.9.0.143
673a51e8593e server-10.9.0.43
[11/28/21]seed@VM:~/.../server-certs$

GNU nano 4.8 /etc/resolv.conf
nameserver 8.8.8.8
options ndots:0
```

Taking note of the local network address of the mitm-proxy machine, we log into a docker shell and modify the `etc/resolv.conf` file to route the name server to the public DNS provided google.com. The reason this is done is because of the docker container will by default route to addresses in the main VM `etc/hosts` file and will try to connect to what ever address is mapped to our target domain, which in this case is itself.

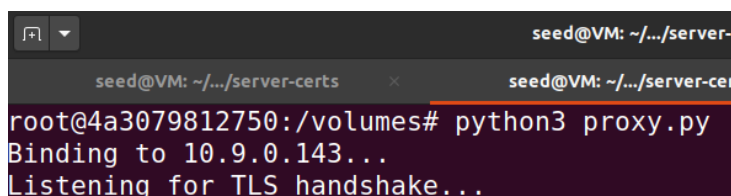
From within the proxy container, we ping the actual server for `www.facebook.com`, and take note of the ip address that we receive in return:



The screenshot shows a terminal window where the command `ping www.facebook.com` is executed. The output shows the IP address `157.240.3.35` and the size of the data received.

```
root@4a3079812750:/volumes# ping www.facebook.com
PING star-mini.c10r.facebook.com (157.240.3.35) 56(84) bytes of data.
```

Next we start the proxy server inside of the proxy container:



The screenshot shows a terminal window where the command `python3 proxy.py` is executed. The output shows the proxy server binding to `10.9.0.143` and listening for TLS handshake.

```
seed@VM: ~/.../server-certs
seed@VM: ~/.../server-certs
root@4a3079812750:/volumes# python3 proxy.py
Binding to 10.9.0.143...
Listening for TLS handshake...
```

And proceed to edit the hosts file in the client container to map [www.facebook.com](https://www.facebook.com) to the proxy server:

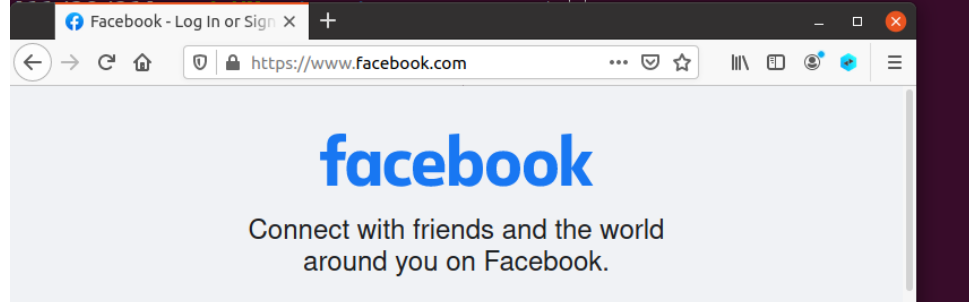
```
seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~
GNU nano 4.8 /etc/hosts
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
10.9.0.5 635b17f2ef19
10.9.0.143 www.facebook.com
```

We also ensure that a valid copy of the authority is present for the proxy to use to verify the connection to facebook.com:

Documents Labsetup volumes client-certs									
Name					Size	Modified	Star		
244b5494.0					1.4 kB	27 Oct 2020	☆		
dbb9c584.0					1.9 kB	15:40	☆		

Pinging facebook.com from the client yields the following result:

```
[11/28/21]seed@VM:~/.../server-certs$ ping www.facebook.com
PING www.facebook.com (10.9.0.143) 56(84) bytes of data.
64 bytes from www.facebook.com (10.9.0.143): icmp_seq=1 ttl=64 time=0.124 ms
64 bytes from www.facebook.com (10.9.0.143): icmp_seq=2 ttl=64 time=0.126 ms
64 bytes from www.facebook.com (10.9.0.143): icmp_seq=3 ttl=64 time=0.132 ms
^C
--- www.facebook.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2148ms
rtt min/avg/max/mdev = 0.124/0.126/0.132/0.003 ms
```



The container is communicating through the IP address for the proxy container, but it is receiving the real facebook.com. This means that the client is successfully routing traffic through the proxy server and facebook.com has no knowledge of the client who is receiving the data.

## Part b.) Connection from client to proxy to a malicious webserver:

We now demonstrate how a MITM can be executed using the above construction.

We simulate a malicious proxy that routes traffic for a certain website to a malicious webserver designed to capture user credentials:

```
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
10.9.0.143 4a3079812750
10.9.0.43 www.facebook.com
```

This way, whenever the client requests [www.facebook.com](http://www.facebook.com) from the proxy, the proxy will forward the received response from the malicious server instead of [www.facebook.com](http://www.facebook.com).

We then modify the HTML payload in the server:

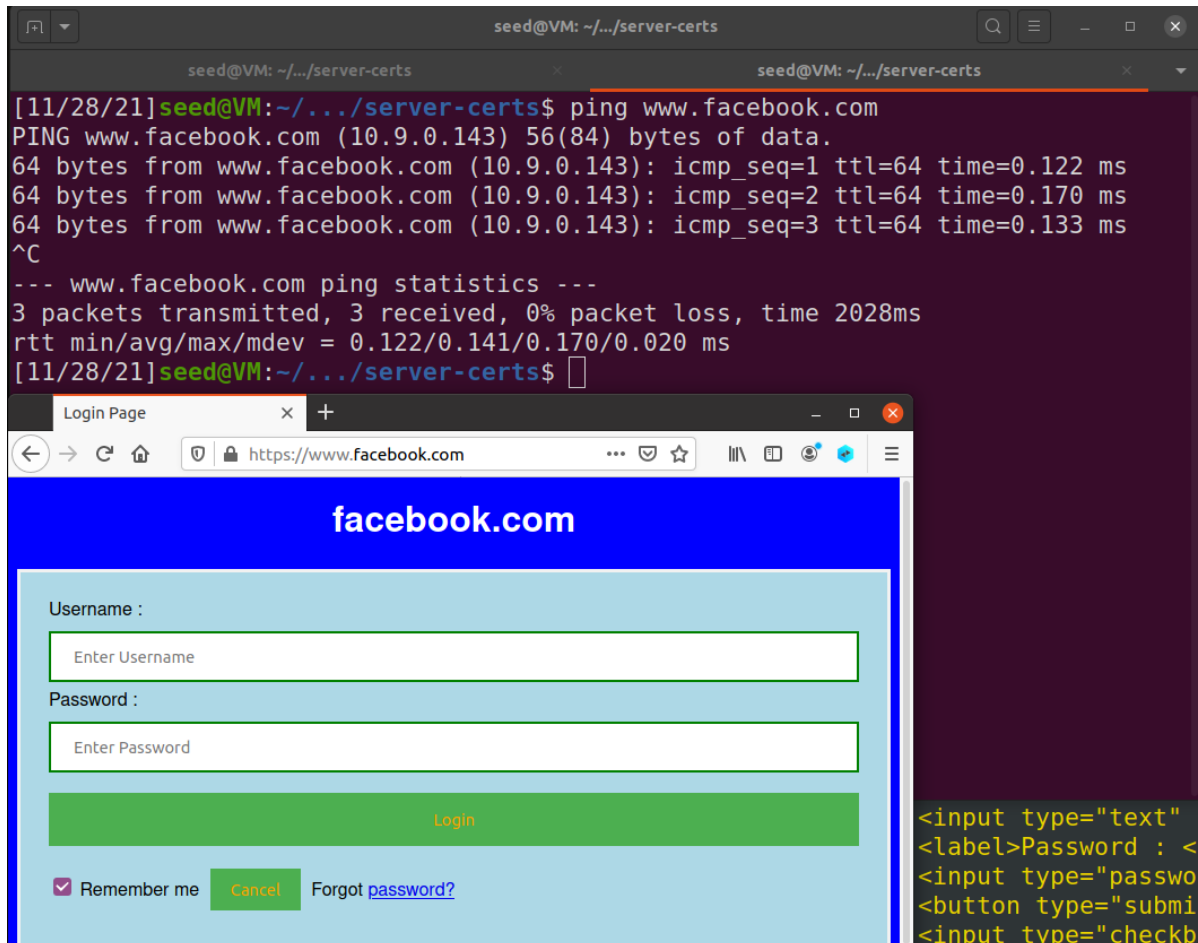
```
handshake.py x proxy.py x server.py x
1#!/usr/bin/env python3
2
3import socket
4import ssl
5import pprint
6
7html = """
8HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
9<!DOCTYPE html>
10<html>
11<head>
12<meta name="viewport" content="width=device-width, initial-scale=1">
13<title> Login Page </title>
14<style>
15Body {
16    font-family: Calibri, Helvetica, sans-serif;
17    background-color: pink;
18}
19button {
20    background-color: #4CAF50;
21    width: 100%;
22    color: orange;
23    padding: 15px;
24    margin: 10px 0px;
25    border: none;
26    cursor: pointer;
27    }
28form {
29    border: 2px solid #f3f3f3
```

And with the proxy mapped to the server, we initialize them both:

```
root@4a3079812750:/volumes# python3 proxy.py
Binding to 10.9.0.143...
Listening for TLS handshake...
Enter PEM pass phrase:
```

```
root@673a51e8593e:/volumes# python3 server.py
Enter PEM pass phrase:
Binding to 10.9.0.43...
Listening for TLS handshake...
```

And we receive the following when we browse to facebook.com from the web browser in the main host VM:



What is striking about this result is the fact that the browser has made a secure TLS connection to what it thinks is [www.facebook.com](https://www.facebook.com). This is enabled by the fact that the browser has the compromised CA added to its list of trusted authorities. In this scenario, if an attacker possesses the secret key for a given CA, they can create forged certificates for real sites, and deliver to the client payloads designed to capture sensitive data.



## Conclusion:

We have demonstrated how TLS connections can be used to create secure channels between parties who wish to exchange communications over the internet. We have also shown the many vulnerabilities in TLS that can be exploited if the proper setup steps are not taken during the initial setup. We proceeded to investigate how a simple HTTPS proxy can be used to securely forward traffic from clients to web servers, and subsequently we also showed how this construction can be used in a malicious manner by routing traffic to websites designed to trick users into revealing sensitive information. We again demonstrated the strengths and weaknesses of applications of public key infrastructure in securing communications. There are many ways to improperly configure TLS connections, but when setup correctly, TLS provides extremely strong security in securing web traffic.