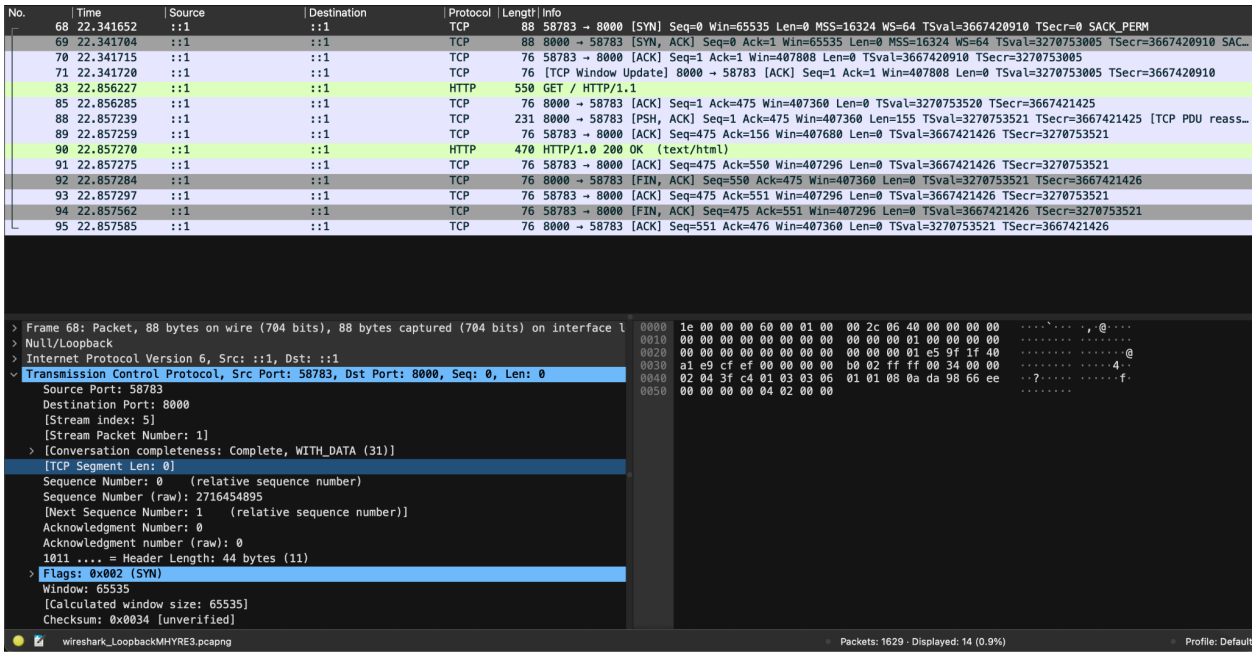
Part 1: Setting up a Local Web Server

To set up my web server I ran “python3 -m http.server 8000” in my desktop directory. This command starts a simple web server on port 8000, serving files from the current directory. When I open this link: [http://[::]:8000/](http://[::]:8000/) .

Part 2: Identifying Why HTTPS is not secure

I have included screenshots of the WireShark view. The top shows several packets that have the HTTP protocol decoded. Since I am running the web server locally, the source and destination are the same.



I now have another screenshot below showing a specific http stream from one of the above http packets. The screenshot demonstrates that HTTP traffic is transmitted in plaintext: an eavesdropper capturing these packets can read the request line, headers, and the full response body. The capture exposes the request line, all request headers, the server response headers, and the full HTML response body. Because the entire exchange is plaintext, an eavesdropper with packet-capture capability can read which resources are requested and the contents of those resources.

Part 3. Create a Self-Signed Certificate and Qpgrade your Web Server to HTTPS

For this part of the assignment we must generate and trust a self signed certificate. The following command runs on my local machine to generate the certificate for localhost.

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr \
  -subj "/C=US/ST=Local/L=Local/O=Dev/OU=Local/CN=localhost"
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain server.crt
```

The last command adds the certificate to macOS's **System keychain**, marking it as a trusted root certificate so that browsers will not show a security warning when accessing the local server.

From here I had to run the server which I did using a script I called lab_one_security.py to start a simple web server.

```python
# lab_one_security.py
import http.server
import ssl
```

```
PORT = 8443
ADDR = ("127.0.0.1", PORT)

handler = http.server.SimpleHTTPRequestHandler
httpd = http.server.HTTPServer(ADDR, handler)

# Modern TLS setup (replaces deprecated ssl.wrap_socket)
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(certfile="server.crt", keyfile="server.key")

httpd.socket = context.wrap_socket(httpd.socket, server_side=True)

print(f"Serving HTTPS on https://{ADDR[0]}:{PORT}")
httpd.serve_forever()
```

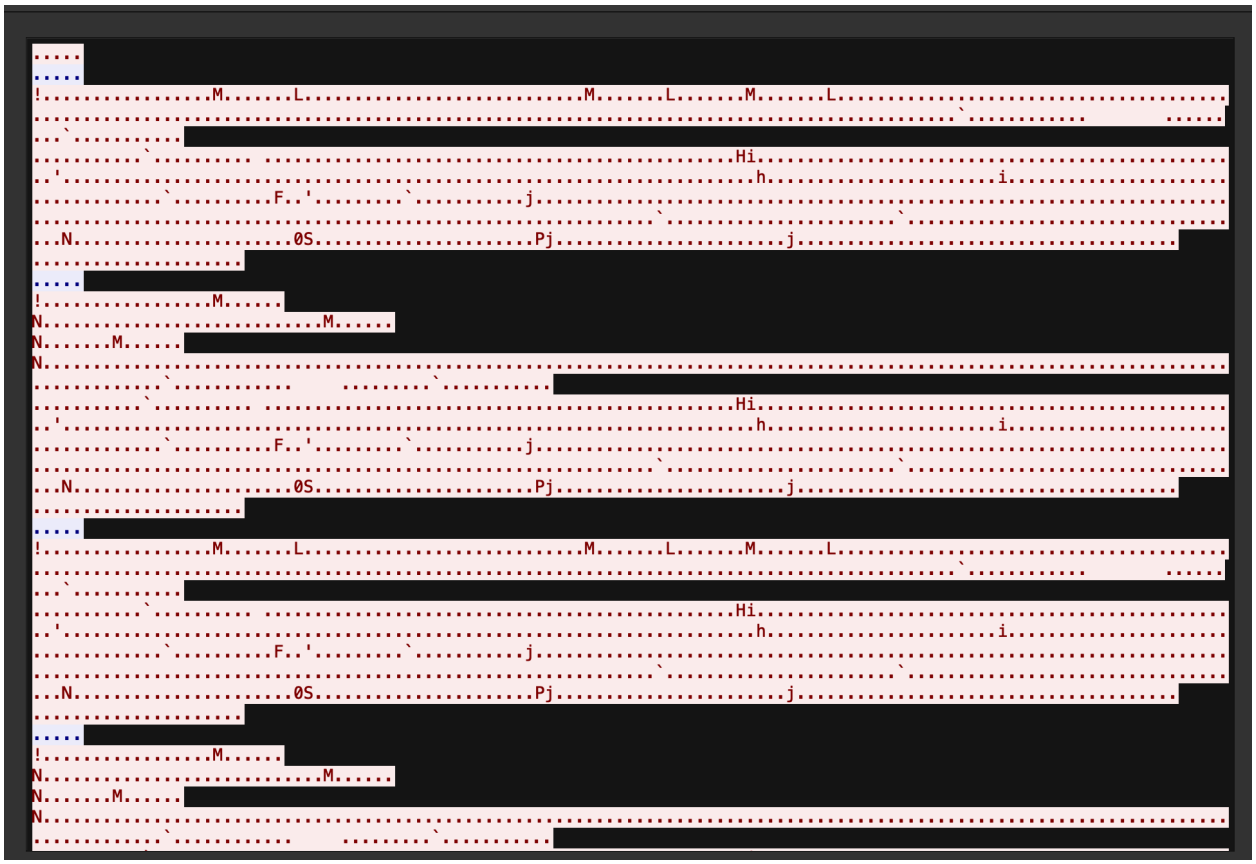
This code applies the generated SSL certificate on my web server using Python's built in HTTP server. From here we run the code and in the terminal we see some general information about the web server. Once executed, the server begins listening on port 8443 and serves files over HTTPS. This is all the code generated after running our script starting the web server.

```
Serving HTTPS on https://127.0.0.1:8443
127.0.0.1 - - [25/Oct/2025 00:17:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:17:39] code 404, message File not found
127.0.0.1 - - [25/Oct/2025 00:17:39] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [25/Oct/2025 00:18:42] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:21:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:21:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:21:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:21:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 00:21:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 01:03:07] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Oct/2025 01:03:08] "GET / HTTP/1.1" 200 -
```

We will now use WireShark again to get more information about how this generated certificate encrypts our web servers packets.

3735	2941.044245	127.0.0.1	127.0.0.1	TCP	57	57343 → 56482 [PSH, ACK] Seq=48252 Ack=406 Win=6380 Len=1 TSval=3790562087 TSecr=981742536
3736	2941.044256	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=406 Ack=48252 Win=5387 Len=0 TSval=981743297 TSecr=3790562087
3737	2941.044265	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=406 Ack=48253 Win=5387 Len=0 TSval=981743297 TSecr=3790562087
3738	2942.291970	127.0.0.1	127.0.0.1	TCP	60	56482 → 57343 [PSH, ACK] Seq=406 Ack=48253 Win=5387 Len=4 TSval=981744545 TSecr=3790562087
3739	2942.291998	127.0.0.1	127.0.0.1	TCP	57	56482 → 57343 [PSH, ACK] Seq=410 Ack=48253 Win=5387 Len=1 TSval=981744545 TSecr=3790562087
3740	2942.292030	127.0.0.1	127.0.0.1	TCP	56	57343 → 56482 [ACK] Seq=48253 Ack=410 Win=6380 Len=0 TSval=3790563335 TSecr=981744545
3741	2942.292056	127.0.0.1	127.0.0.1	TCP	56	57343 → 56482 [ACK] Seq=48253 Ack=411 Win=6380 Len=0 TSval=3790563335 TSecr=981744545
3742	2942.292094	127.0.0.1	127.0.0.1	TCP	60	57343 → 56482 [PSH, ACK] Seq=48253 Ack=411 Win=6380 Len=4 TSval=3790563335 TSecr=981744545
3743	2942.292102	127.0.0.1	127.0.0.1	TCP	89	57343 → 56482 [PSH, ACK] Seq=48257 Ack=411 Win=6380 Len=33 TSval=3790563335 TSecr=981744545
3744	2942.292110	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=48257 Win=5387 Len=0 TSval=981744545 TSecr=3790563335
3745	2942.292121	127.0.0.1	127.0.0.1	TCP	60	57343 → 56482 [PSH, ACK] Seq=48290 Ack=411 Win=6380 Len=4 TSval=3790563335 TSecr=981744545
3746	2942.292125	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=48290 Win=5387 Len=0 TSval=981744545 TSecr=3790563335
3747	2942.292133	127.0.0.1	127.0.0.1	TCP	853	57343 → 56482 [PSH, ACK] Seq=48294 Ack=411 Win=6380 Len=797 TSval=3790563335 TSecr=981744545
3748	2942.292138	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=48294 Win=5387 Len=0 TSval=981744545 TSecr=3790563335
3749	2942.292146	127.0.0.1	127.0.0.1	TCP	60	57343 → 56482 [PSH, ACK] Seq=49091 Ack=411 Win=6380 Len=4 TSval=3790563335 TSecr=981744545
3750	2942.292146	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=49091 Win=5375 Len=0 TSval=981744545 TSecr=3790563335
3751	2942.292150	127.0.0.1	127.0.0.1	TCP	69	57343 → 56482 [PSH, ACK] Seq=49095 Ack=411 Win=6380 Len=13 TSval=3790563335 TSecr=981744545
3752	2942.292156	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=49095 Win=5375 Len=0 TSval=981744545 TSecr=3790563335
3753	2942.292164	127.0.0.1	127.0.0.1	TCP	56	56482 → 57343 [ACK] Seq=411 Ack=49108 Win=5375 Len=0 TSval=981744545 TSecr=3790563335

This is from the same page that we took the screenshot in in part two. This is the list of all our encrypted packets from our web server, after using the SSL certificate. As you can see, all the packages are encrypted and the protocol changed from HTTPS to TCP. Even though the it still uses TCP, the application-layer data (HTTP requests and responses) is now encrypted inside TLS records.



Above we have a specific TCP packet and look at its entry. As we can see, the packet is no longer visible in plaintext like when we ran the server without our certificate. This packet is now encrypted and unreadable, highlihthting how the certificate applying it to our web server provides encryption and safety.