

Nick Rinaldi
CMSC 23206
October 2025
Prof. Nick Feamster

Public Key Infrastructure

Task 1: Host a local web server

First, I created a directory containing the following file, `index.html`:

```
HTML
<h1>Top Secret Message</h1>
<p>Password: feamster2025</p>
```

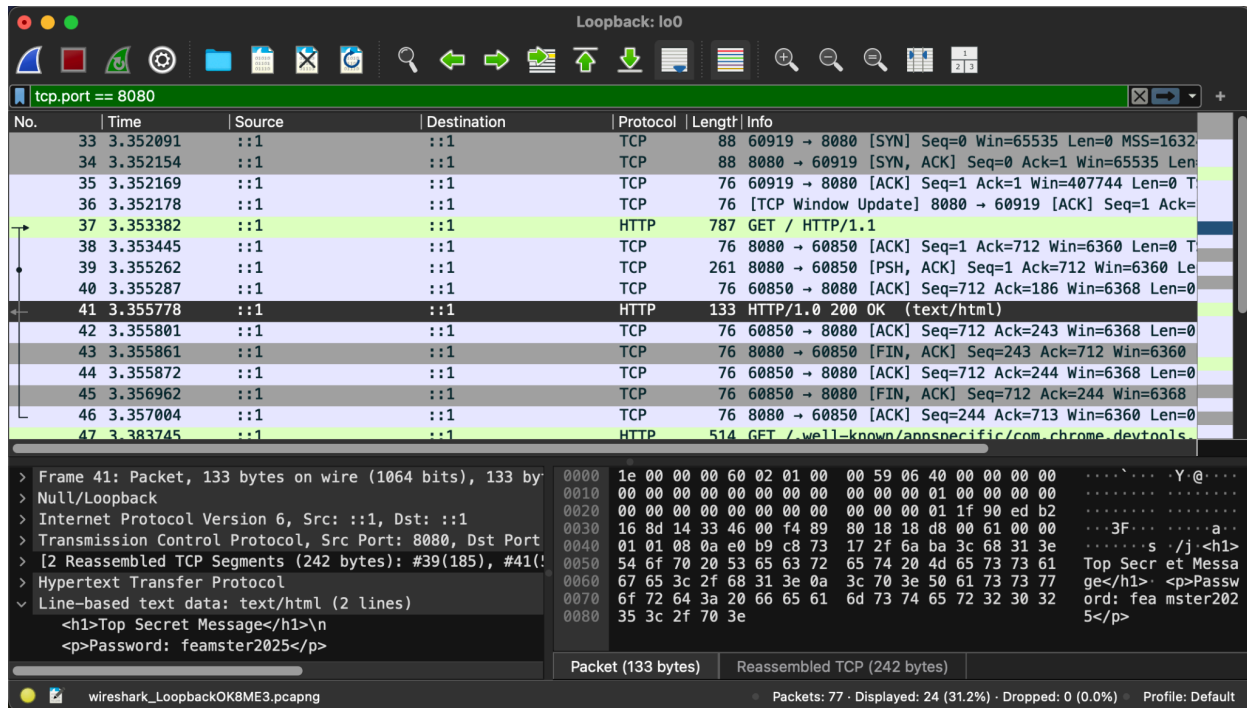
Next, I hosted a local web server using Python's HTTP server class:

```
Shell
$ python3 -m http.server 8080
```

Task 2: Identify why HTTP is not secure

HTTP is insecure because it transmits requests and responses in plaintext. This means that any eavesdropper on the same network can intercept and read the data, including URLs, headers, and body contents. Attackers can also modify responses in transit ("man-in-the-middle" attacks).

To demonstrate this, I downloaded Wireshark, selected `lo0` as my interface, and began capturing. In my browser, I visited <http://localhost:8080>. After the page loaded, I stopped capturing. In Wireshark, I applied the filter `tcp.port == 8080`, resulting in the following output:



As you can see, the Wireshark capture shows the HTTP contents in plaintext, including the line "Password: feamster2025" (bottom left, bottom right).

Task 3: Create a self-signed certificate and upgrade your web server to HTTPS

Since my local server is not accessible from the public internet, I cannot obtain a certificate from a Certificate Authority (CA). CAs only issue certificates for publicly verifiable domain names. Instead, I generated a self-signed certificate that I used to enable HTTPS locally:

Shell

```
$ openssl genrsa -out server.key 2048
$ req -new -key server.key -out server.csr
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
```

```
Certificate request self-signature ok
subject=C=AU, ST=Some-State, O=Internet Widgits Pty Ltd, CN=localhost
```

I had ChatGPT write a short Python script, `https_server.py`, to serve the same page securely using TLS:

```
Python
import http.server
import ssl

server_address = ('127.0.0.1', 8443)
httpd = http.server.HTTPServer(server_address,
http.server.SimpleHTTPRequestHandler)

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("Serving on https://127.0.0.1:8443")
httpd.serve_forever()
```

I hosted the HTTPS server as follows:

```
Shell
$ python3 https_server.py
```

Like before, I began capturing traffic on `lo0` in Wireshark and visited <https://127.0.0.1:8443>. Unlike before, the packets were labeled TLSv1.3 and contained no readable text. The TCP payload appeared as encrypted binary data instead of plaintext HTML:

The image shows a Wireshark capture of network traffic on the 'Loopback: lo0' interface. The packet list shows a series of TCP and TLSv1 packets between 127.0.0.1 and 127.0.0.1. The details pane for frame 33 (1.123686) shows a TLSv1 'Server Hello' message. The packet bytes pane displays the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
27	1.122385	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8443 → 51836 [ACK] Seq=1 Ack=
28	1.122389	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8443 → 51837 [ACK] Seq=1 Ack=
29	1.122561	127.0.0.1	127.0.0.1	TLSv1	2124	Client Hello
30	1.122584	127.0.0.1	127.0.0.1	TCP	56	8443 → 51836 [ACK] Seq=1 Ack=2069 Win=406208 Len=
31	1.122702	127.0.0.1	127.0.0.1	TLSv1	1853	Client Hello
32	1.122726	127.0.0.1	127.0.0.1	TCP	56	8443 → 51837 [ACK] Seq=1 Ack=1798 Win=406464 Len=
33	1.123686	127.0.0.1	127.0.0.1	TLSv1	1385	Server Hello, Change Cipher Spec, Application Dat
34	1.123709	127.0.0.1	127.0.0.1	TCP	56	51836 → 8443 [ACK] Seq=2069 Ack=1330 Win=406912 L
35	1.123894	127.0.0.1	127.0.0.1	TLSv1	86	Change Cipher Spec, Application Data
36	1.123912	127.0.0.1	127.0.0.1	TCP	56	8443 → 51836 [ACK] Seq=1330 Ack=2099 Win=406144 L
37	1.123958	127.0.0.1	127.0.0.1	TCP	56	51836 → 8443 [FIN, ACK] Seq=2099 Ack=1330 Win=406
38	1.123971	127.0.0.1	127.0.0.1	TCP	56	8443 → 51836 [ACK] Seq=1330 Ack=2100 Win=406144 L
39	1.124088	127.0.0.1	127.0.0.1	TCP	56	8443 → 51836 [FIN, ACK] Seq=1330 Ack=2100 Win=406
40	1.124118	127.0.0.1	127.0.0.1	TCP	56	51836 → 8443 [ACK] Seq=2100 Ack=1331 Win=406912 L
41	1.127579	127.0.0.1	127.0.0.1	TLSv1	2569	Server Hello, Change Cipher Spec, Application Dat

Frame 33: Packet, 1385 bytes on wire (11080 bits), 1385 bytes captured (11080 bits) on interface Null/Loopback

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 8443, Dst Port: 51836

Source Port: 8443

Destination Port: 51836

[Stream index: 9]

[Stream Packet Number: 7]

[Conversation completeness: Complete, WITH_DATA (31)]

Packets: 116 · Dropped: 0 (0.0%) Profile: Default

Under HTTP, the entire request and response was human-readable, including sensitive information like "Password: feamster2025". However, under HTTPS, the traffic is encrypted using the TLS protocol, so only the handshake metadata (e.g. Client Hello, Server Hello above) is visible. The actual page contents and credentials are fully encrypted. Even if an attacker captures packets, they cannot read or modify the transmitted data.