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>
```

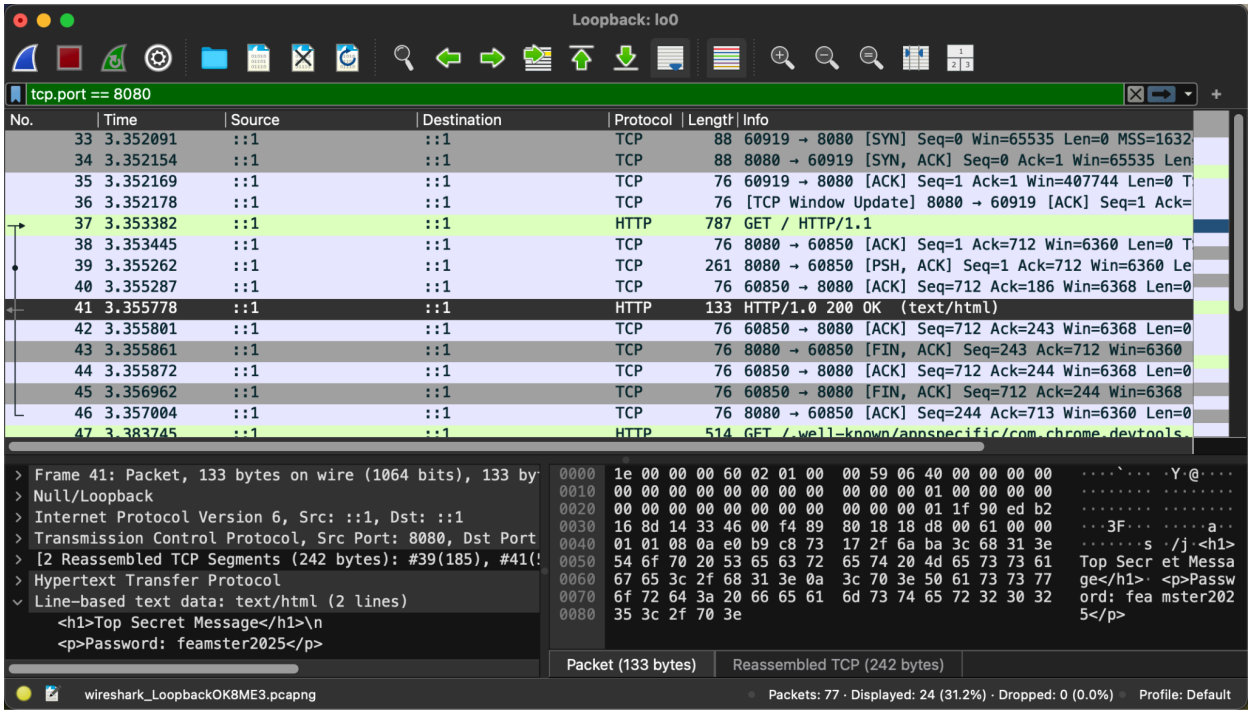
Next, I hosted a local web server using Python's HTTP server class:

```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
```

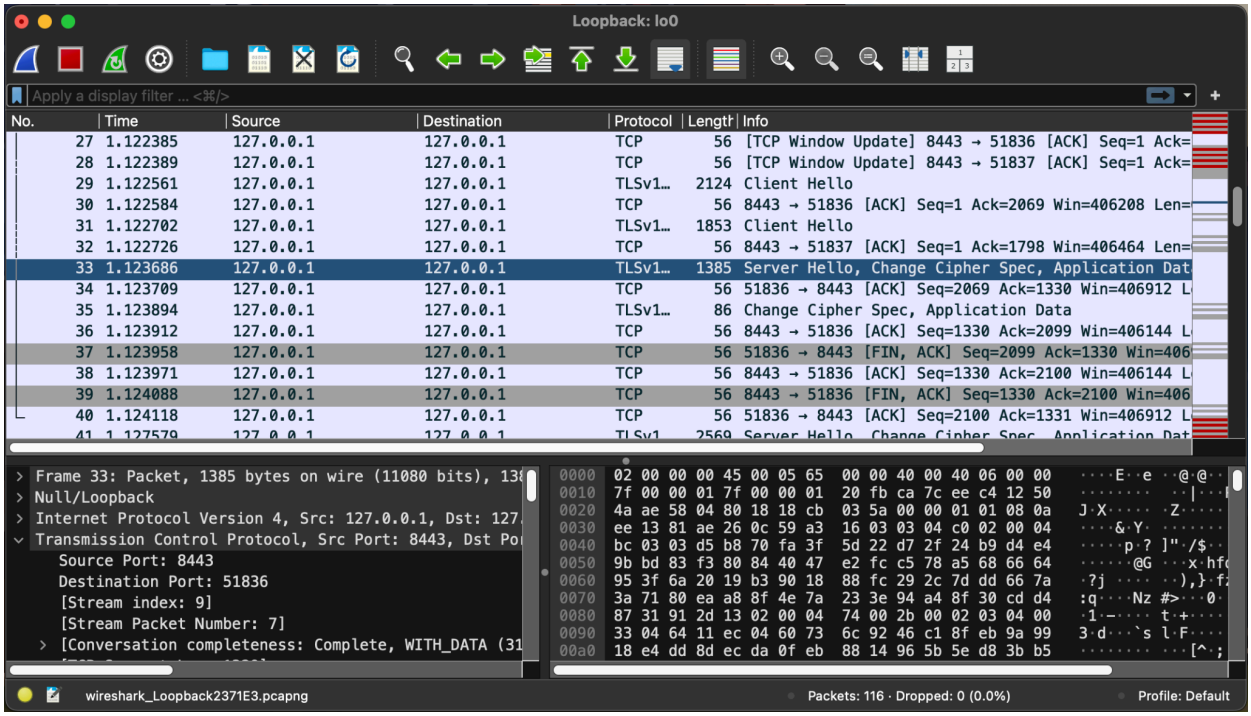
I had ChatGPT write a short Python script, https_server.py, to serve the same page securely using TLS:

```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:



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.