Documentation & Responses

Task 1 (Hosting a Local Web Server)

In setting up the HTTP web server I used Python's HTTP server class. This was done in the following simple manner:

1. '$ mkdir website'
2. '$ cd website'
3. '$ code index.html'

The file 'index.html' was then populated with the following information to add some text to the HTTP server. [Note: I consulted Claude Sonnet 4.5 for formatting guidance on an HTML file.]

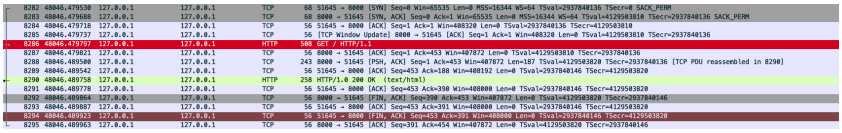
```
<!DOCTYPE html>
<html>
  <head>
    <title>Local Server</title>
  </head>
  <body>
    <h1>Hello! This is my web server.</h1>
    <p> This is running on HTTP.</p>
  </body>
</html>
```

4. Then, in the 'website' folder, I executed the command '$ python3 -m http.server 8000'.
5. I then entered the URL 'http://localhost:8000' into my web browser and verified my HTTP page:

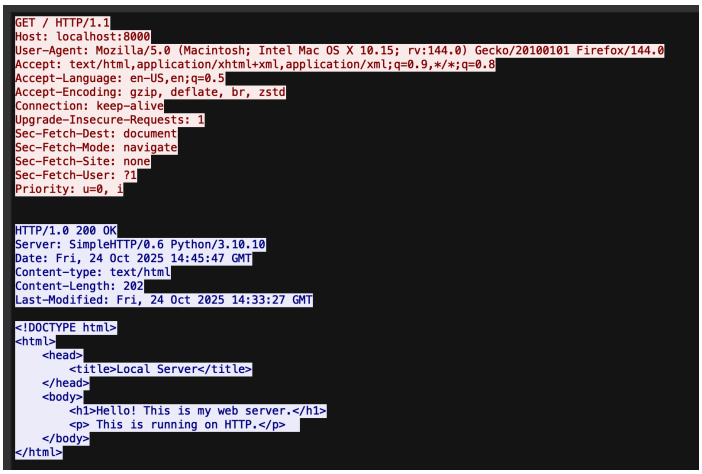
Task 2 (Insecurity of HTTP)

Having set up the HTTP server and using Wireshark, I then began to capture HTTP traffic. Considering that the server created in the prior task is a local web server, I captured traffic on the Loopback interface (designated as ‘Loopback: lo0’ in Wireshark). With packet traffic now being visible, I paired it down to HTTP traffic using the filter bar in Wireshark.

Next, I refreshed the page ‘http://localhost:8000’ a few times to generate packet traffic that could be intercepted.



Above, Wireshark displays a number of transmitted packets that essentially confirm the HTTP server is up and running as intended. Indeed, there are multiple GET requests for the server and others with a code of ‘200’, indicating that the server has been fetched successfully. Following the TCP stream for one of the GET / HTTP/1.1 requests we see the following:



This is troubling because it is basically confirmation that HTTP is unencrypted. All information exchanged between the client and server is visible in plaintext here. Both the HTTP request headers (indicating what the browser sent) and the HTTP response (the content of the page) are completely visible.

Response: HTTP is fundamentally insecure because any eavesdropper can 'sniff' traffic and read any credentials or content on the page. Whether this consists of URLs, body content, or headers, everything is readable with HTTP to someone capturing packets on the same network. Importantly, this could also include any sensitive data (such as passwords or personal information). This is because all information for the server is in plaintext and is completely unencrypted. As a result, it is just as visible to me as it would be to someone using Wireshark on the same network.

Task 3 (SSL Certificate and HTTPS Upgrade)

Response: It is not possible to obtain an SSL certificate for my local web server from a certificate authority because these are reserved for public domain names. Since I am running a local server (localhost) that is not a public domain and cannot conclusively be tied to me, a CA will not grant a certificate. Instead, I would need a domain name that I both verifiably control and is publicly accessible.

Nonetheless, I can generate a self-signed certificate from my machine. This is done by running the following command:

'openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365'

I then filled in the requisite information (fields such as location and organization name), which generated the files cert.pem (with my certificate) and key.pem (with my private key).

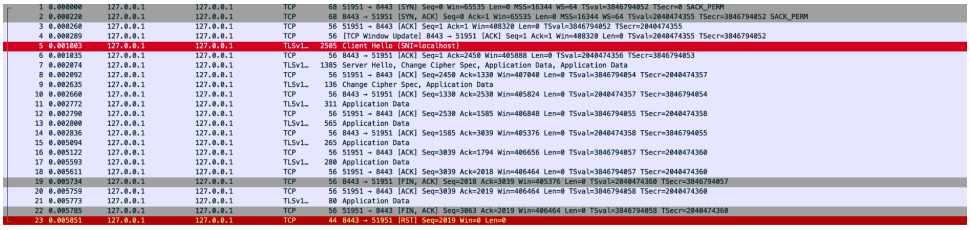
Next, I created a short Python file to serve my HTTPS server, with the following contents:

```
import http.server
import ssl

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

httpd.socket = ssl.wrap_socket(httpd.socket,
```

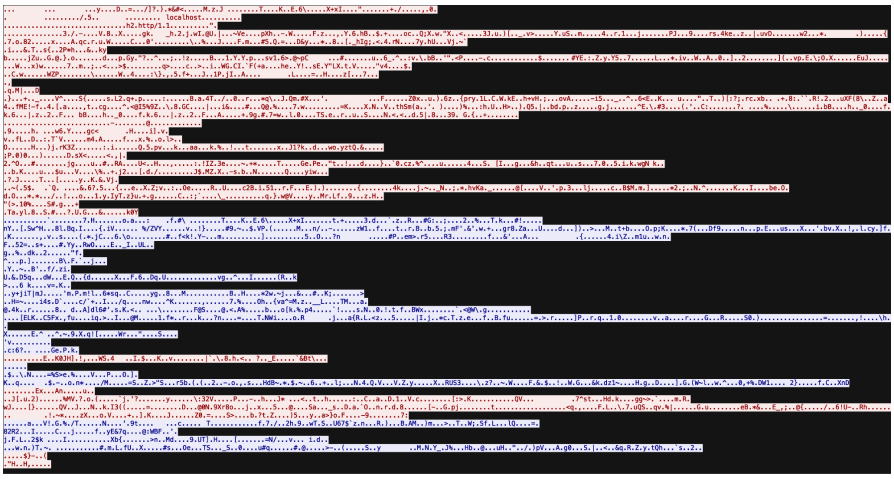
```
                    server_side=True,
                    certfile='cert.pem',
                    keyfile='key.pem',
                    ssl_version=ssl.PROTOCOL_TLS)

    httpd.serve_forever()
```

Upon running '$ python3 -m https_server.py', I entered the URL 'https://localhost:8443' and verified my HTTPS page.

Then, I started a capture and refreshed the page 'https://localhost:8443' to generate packet traffic.



As with the second task, Wireshark displays a number of transmitted packets that essentially confirm the HTTPS server is up and running as intended. Following the TCP stream for one of the Client Hello requests we see the following:



This signifies that the encryption (specifically, TLS encryption) is working as intended because all of the information exchanged between the client and server is gibberish to an eavesdropper. Although someone can tell that a connection has

occurred, the content of it is completely scrambled and any private information is unintelligible.