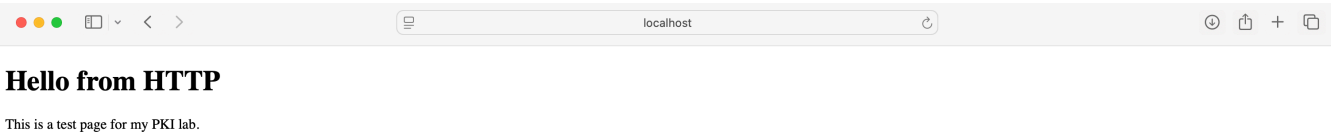
Public Key Infrastructure Lab Assignment

Task 1: Host a Local Web Server

The first part of the lab involved creating a local web server to understand how unencrypted HTTP communication works. I began by creating a dedicated directory for this project by typing "mkdir ~/pki_lab" on my terminal and then navigating into it with "cd ~/pki_lab."

Inside this folder, I created a simple webpage called index.html that displayed the text "Hello from HTTP" and a short message underneath. I then launched a local HTTP server using Python's built-in module by entering "python3 -m http.server 8080."

After running the server, I opened my browser and visited "http://localhost:8080." The page displayed my message, confirming that the local server was working correctly and showing the file. I took screenshots of the browser displaying the page and the terminal showing the server log with the GET requests. These confirmed that the setup was functional and that the server was handling plain HTTP traffic.

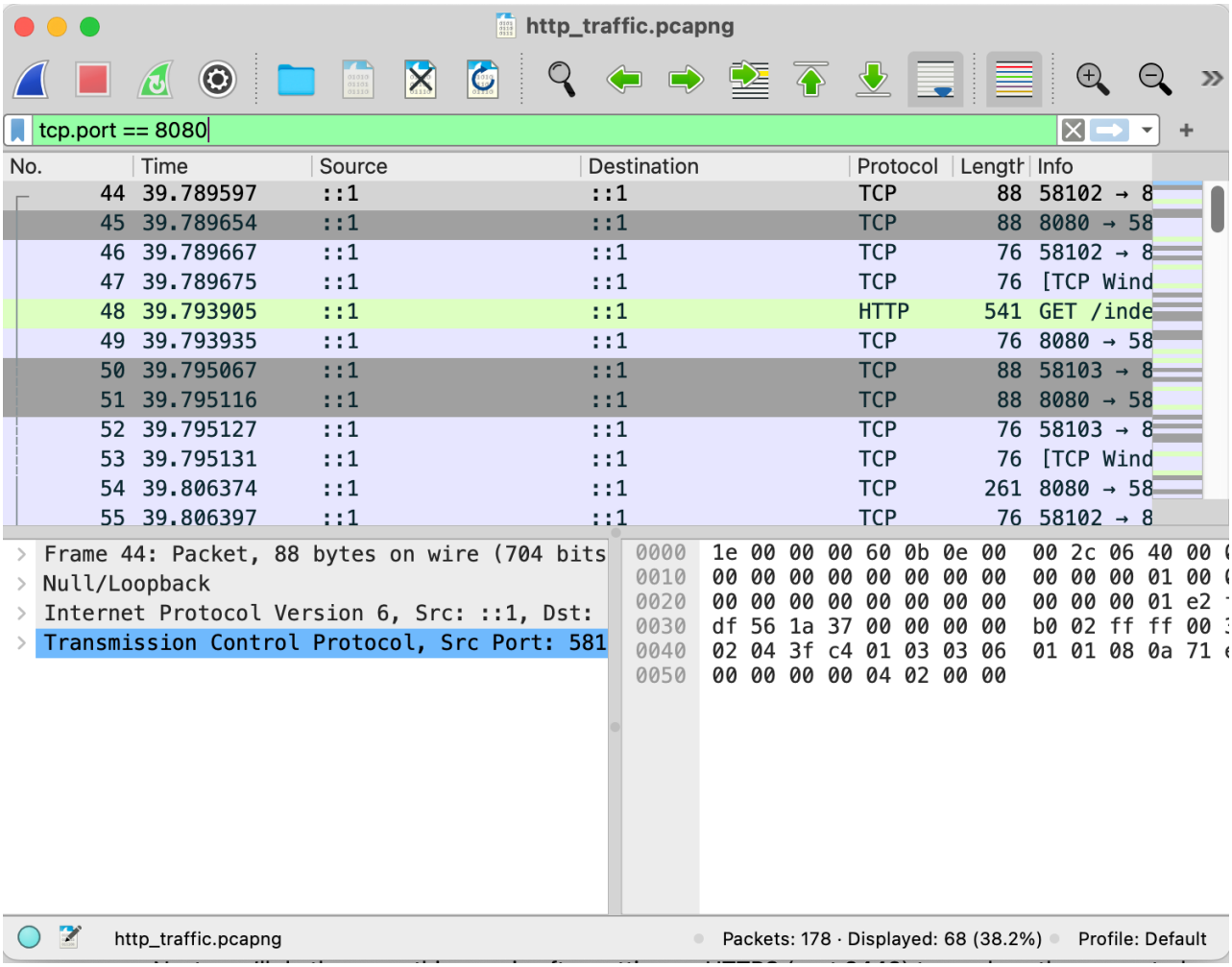


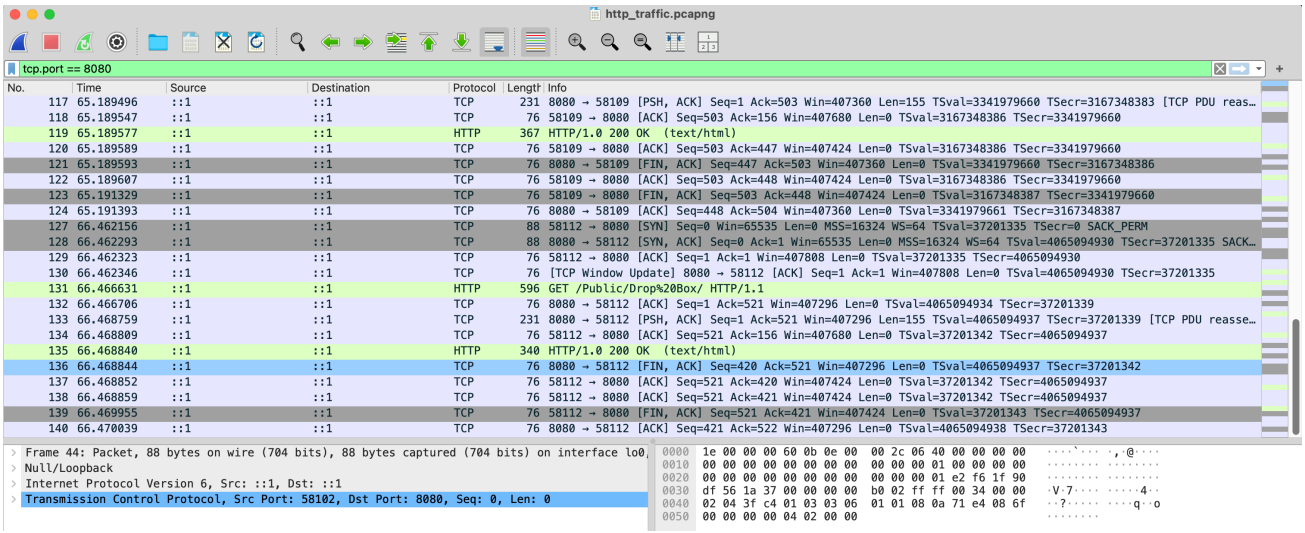

```
Mac:~ mati$ python3 -m http.server 8080
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::1 - - [22/Oct/2025 14:59:34] "GET / HTTP/1.1" 200 -
::1 - - [22/Oct/2025 14:59:34] "GET / HTTP/1.1" 200 -
```

Task 2: Identify Why HTTP Is Not Secure

The goal of this task was to demonstrate that HTTP is insecure because it transmits all data in plaintext. To do this, I used Wireshark to capture and analyze HTTP traffic between my browser and the local web server.

I opened Wireshark, selected the loopback interface (lo0), and started capturing packets. Then, I refreshed the page at “http://localhost:8080” and applied a display filter to isolate the relevant traffic. When examining the packets, I noticed that Wireshark clearly showed HTTP GET requests, headers, and even the content of the HTML file. The request “GET /index.html HTTP/1.1” and the response containing the phrase “Hello from HTTP” were visible in plain text.





This proved that HTTP provides no encryption or integrity protection, and anyone intercepting the network traffic could read or even alter the data being transmitted. In real-world scenarios, this means that passwords, cookies, or other sensitive information could easily be stolen by an attacker monitoring the network.

This task emphasized how HTTP communication is vulnerable to eavesdropping, spoofing, and man-in-the-middle attacks. When a client uses HTTP, all requests and responses travel in plain text across the network. This means an eavesdropper running a packet-sniffing tool such as Wireshark can easily intercept and read everything being communicated between the browser (the client) and the server. The attacker can see which pages and files are being requested (for example, “GET /index.html”), view all the response headers, and even open the HTML content or images inside the packets. In other words, a passive observer can fully reconstruct the website being visited, including any forms, text, or data being transmitted. This demonstrates how insecure HTTP is when it comes to confidentiality and privacy.

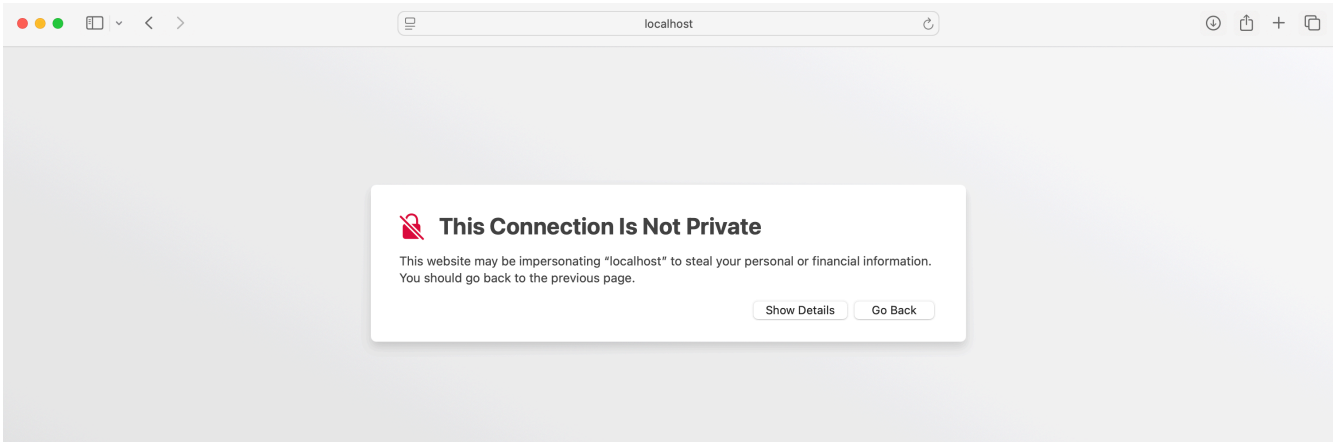
Task 3: Create a Self-Signed Certificate and Upgrade Your Web Server to HTTPS

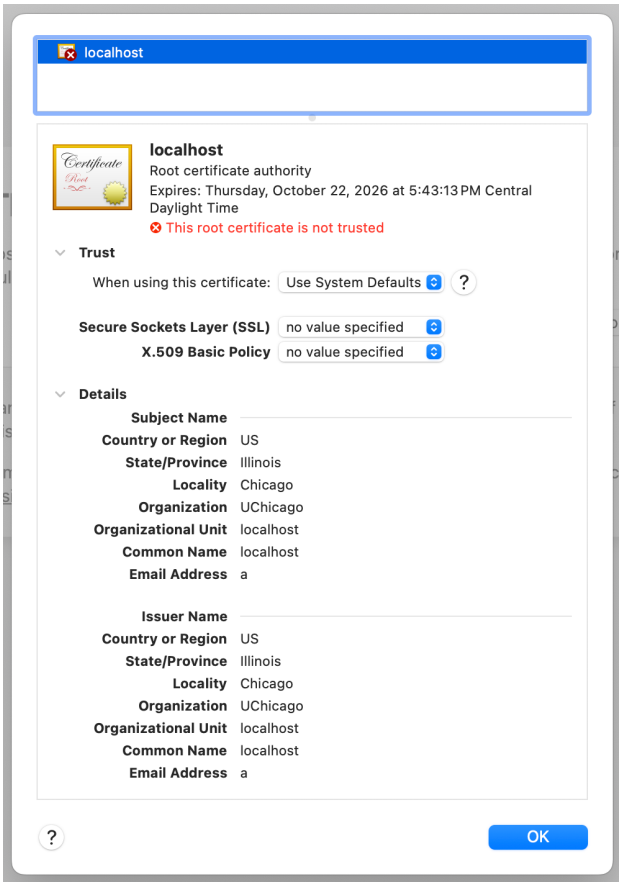
In this final part, the goal was to upgrade the web server to use HTTPS, securing communication through encryption. The first question asked why a certificate authority can’t issue an SSL certificate for a local web server. The reason is that certificate authorities can only verify and issue certificates for public, domain-based servers that can be validated through DNS. Since “localhost” is not a publicly resolvable domain, ownership can’t be proven. Therefore, I had to create a self-signed certificate for local testing.

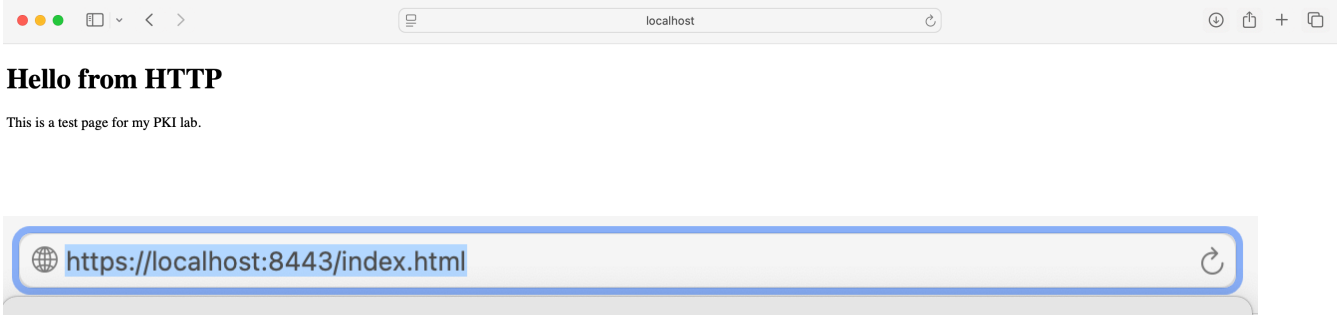
I generated the self-signed certificate and private key using OpenSSL, filled in the requested fields such as country, state, organization, and common name (using “localhost” for the common name), and confirmed that two new files were created: cert.pem and key.pem.

Next, I created a Python HTTPS server using the http.server module and the ssl library. After writing the short script, I ran it, and the terminal confirmed that the HTTPS server was running on port 8443.

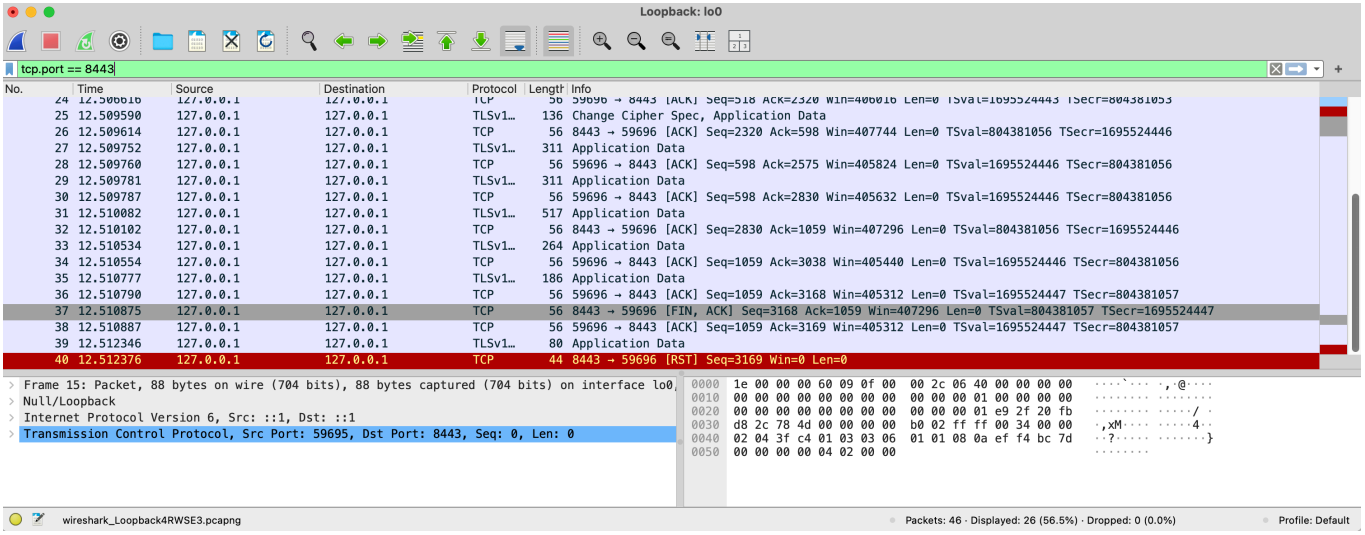
When I navigated to “https://localhost:8443/index.html” in my browser, I encountered a “This Connection Is Not Private” warning. This occurred because the certificate was self-signed and therefore not trusted by the browser. I proceeded by clicking “Show Details” and “Visit Website.” Once I continued, my “Hello from HTTP” page loaded successfully, but this time through an encrypted HTTPS connection.







To confirm encryption, I again opened Wireshark, selected the loopback interface, started a capture, and reloaded the HTTPS page. I applied the filter for port 8443. Unlike before, the packets did not display readable HTML or HTTP headers. Instead, Wireshark showed “TLSv1.3” handshake packets followed by “Application Data,” which was encrypted. This confirmed that HTTPS was properly protecting the confidentiality and integrity of the transmitted data.



To comment more on the difference between the HTTP and the HTTPS (TLS) traffic, when using HTTP, the data sent between the client and the server is transmitted in plain text. This means that anyone monitoring the network can read, capture, and even modify the information being exchanged. In contrast, HTTPS encrypts all communication using the Transport Layer Security (TLS) protocol. This prevents attackers from viewing or tampering with the data, as only the client and the server possess the keys needed to decrypt the information.

While HTTP makes it possible to see everything, including requests, responses, and the contents of a webpage, HTTPS hides all of that behind encryption. HTTPS also ensures server authentication, meaning that clients can verify they are truly communicating with the intended website. Additionally, data integrity is protected, so that messages cannot be altered during transmission without detection. Finally, HTTP uses port 8080 in this lab setup, while HTTPS uses port 8443.