Public Key Infrastructure

Veronica Sokoloff Cortes

Fall 2025

Goal: Show why HTTP is insecure (plaintext) and how HTTPS (TLS) fixes it (encrypted).

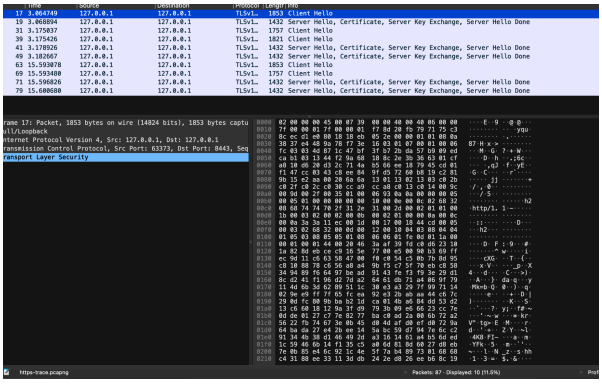
Setup:

- OS: macOS (loopback `lo0`)
- Tools: Python 3 (`http.server`), OpenSSL, Wireshark

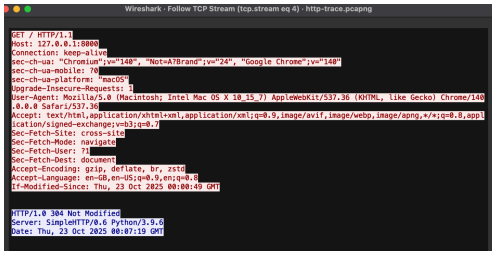
1. Host a local web server: To host the local web server I used Python's built-in http.server. I started by making a working folder with an `index.html` so there was something to serve (`mkdir -p ~/http-demo && cd ~/http-demo`, then I wrote a basic "Hello, HTTP" page). Next, I started the server in the terminal and binded it to my own PC for safety (`python3 -m http.server 8000 --bind 127.0.0.1`). Terminal confirmed it was running, and I double checked by visiting `http://127.0.0.1:8000/` .
    - ⋆ *ChatGBT told me that binding to 127.0.0.1 keeps the server local, nothing exposed to other devices, which made this the fastest, lowest-friction way to demonstrate an HTTP server without configuring nginx or Apache.*

2. Identify why HTTP is not secure: `HTTP` isn't secure because it sends everything unencrypted (requests, headers, and bodies) so literally anyone on the path could read it if they wanted to. An eavesdropper on the same network can "sniff" packets and see exactly which resources are fetched (`GET /index.html`), plus sensitive data in headers or the page/form content. To demonstrate this, I ran my local `HTTP` server and captured traffic on the loopback interface (`lo0`) with Wireshark using the http (or `tcp.port == 8000`) display filter. The capture clearly shows the plaintext request line, Host, other headers, and the `HTML` body in the bytes pane—proof that `HTTP` provides no confidentiality. (Screenshots included below.)

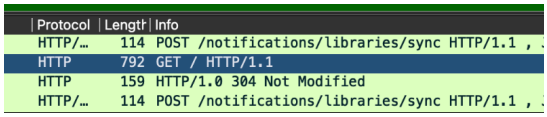
TLS 1.3 handshake begins; only metadata visible



End-to-end HTTP dialogue is readable to an eavesdropper.



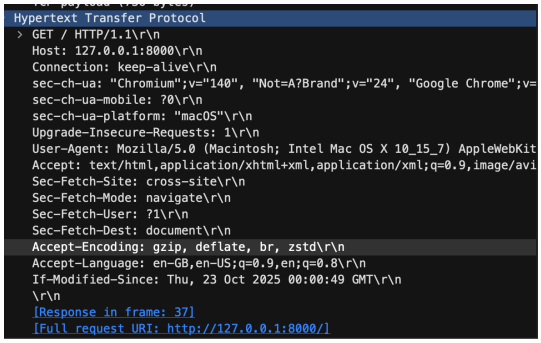
HTTP request is fully readable.



3. Create a self-signed certificate and upgrade your web server to HTTPS:
    a. I found out that public certificate authorities only issue certificates for public DNS names after you prove domain control (via `DNS/HTTP` challenges). `localhost` and `127.0.0.1` are local-only identifiers (not publicly resolvable) and are explicitly disallowed by CA/B rules, so there's no way for me to get validated or issued a cert for my local dev server.

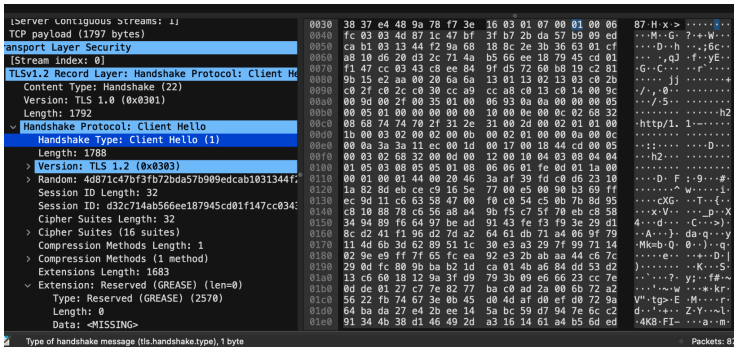
So, I instead generated a self-signed X.509 certificate with Subject Alternative Names for localhost, 127.0.0.1, and ::1, then added it to my macOS System Keychain as Always Trust so the browser would accept it. I restarted my server wrapped in TLS (Python http.server behind ssl.SSLContext) on port 8443 and verified I could load https://127.0.0.1:8443/ without warnings. I recorded my traffic while loading the page and I was shocked to see you can read everything: the request (`GET /`), the Host header, and the page's HTML. In the HTTPS capture, I first saw a "hello" exchange between the browser and server to set up encryption, but after that all the packets just look like scrambled data, hard to read.

- HTTP→ anyone watching the network can see the whole conversation.



Wireshark shows the full request line, headers, and (in the response) the page body in clear text—confirming that HTTP provides no confidentiality.

- HTTPS→the same conversation is encrypted after the initial setup, so it's hidden.



Browser begins the secure session & only the setup metadata is visible rest on left is unreadable

My screenshots show HTTP is readable and HTTPS is unreadable.