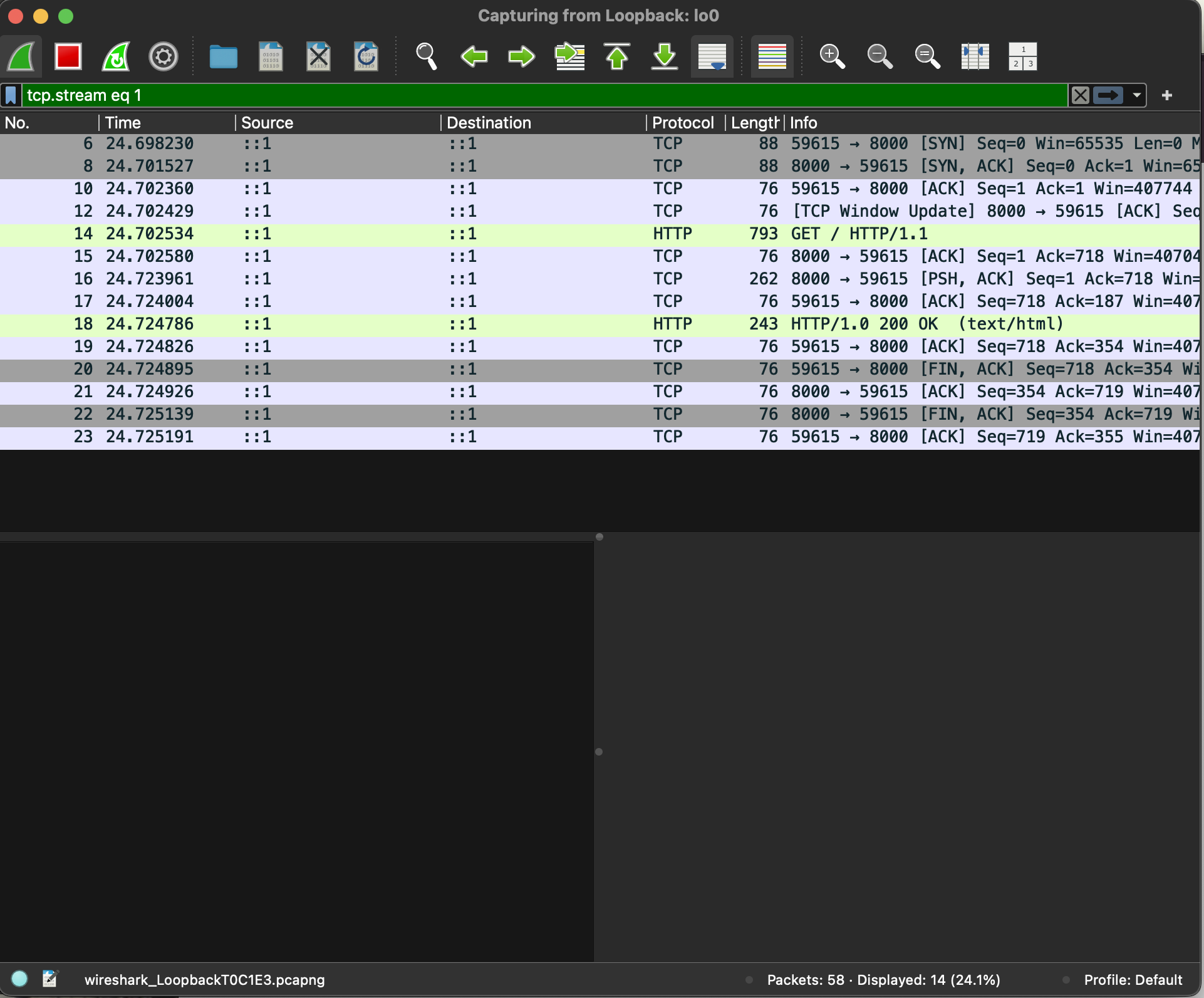
PKI lab:

* http\_trace.pcapng — HTTP capture (plaintext)
* https\_trace.pcapng — HTTPS/TLS capture (encrypted)
* This write-up with screenshots:
  + Fig. 1: HTTP page load (<http://localhost:8000/>)
  + Fig. 2: Wireshark “Follow TCP Stream” for HTTP showing 200 OK and plaintext HTML
  + Fig. 3: TLS ClientHello (SNI = localhost)
  + Fig. 4: TLS Application Data (opaque bytes)

To host an HTTP server, I served a simple page from ~/pki-lab/site/index.html using Python.

I verified in the browser at <http://localhost:8000/> (Fig. 1).

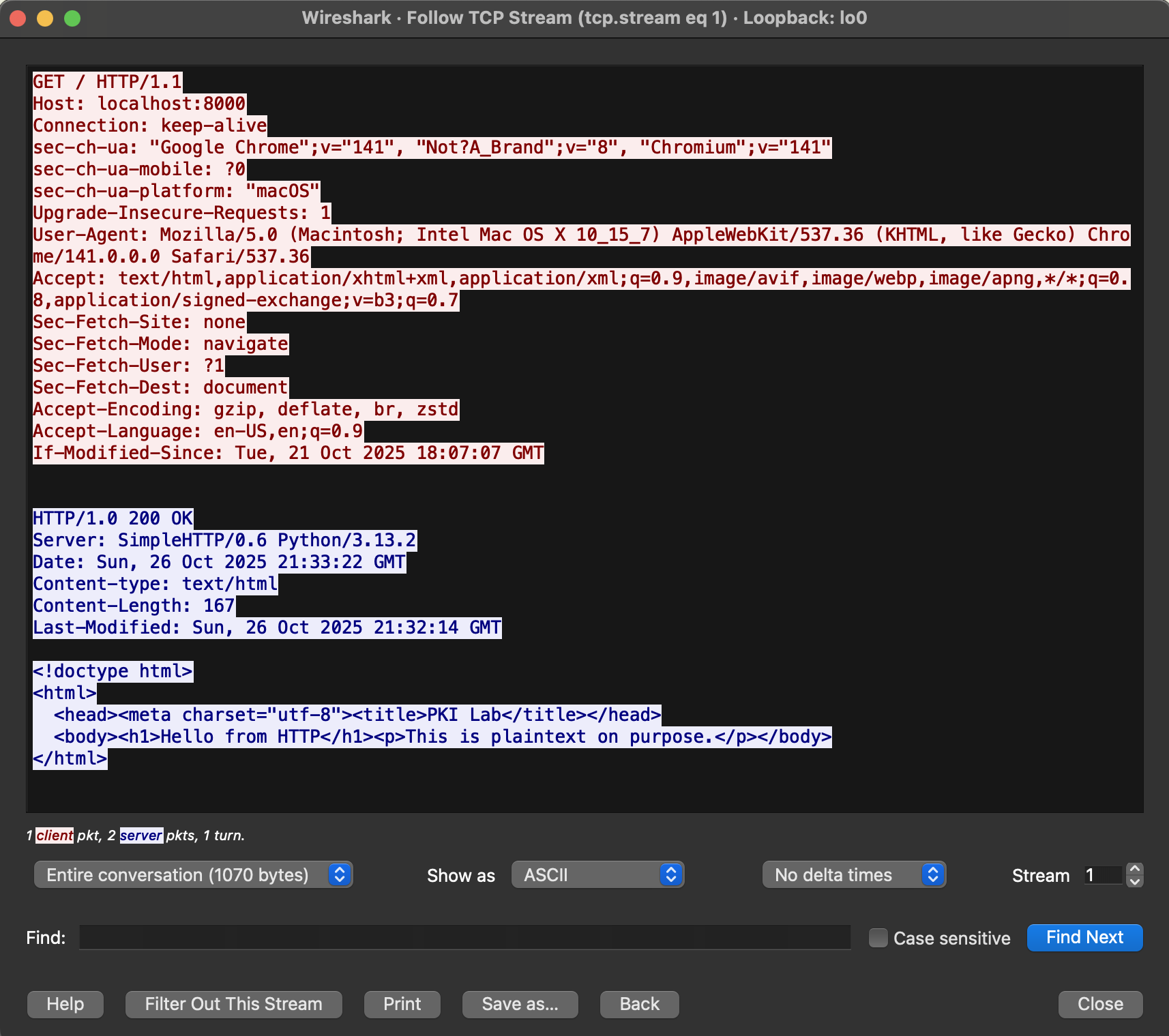


Why HTTP is not secure:

HTTP is unencrypted. Any on-path observer (same Wi-Fi/LAN, ISP, compromised router, MITM AP) can read and modify requests and responses—URLs, headers, cookies, form contents, and full bodies.

Capture method: loopback (lo0 on macOS). Display filter: tcp.port == 8000.

Evidence. The reassembled stream shows the browser’s plaintext request and the server’s plaintext response (Fig. 2):



* Request (client → server): GET / HTTP/1.1, Host: localhost:8000, user agent, accept headers, etc.
* Response (server → client): HTTP/1.0/1.1 200 OK, Content-Type: text/html; charset=utf-8, followed by the HTML body in cleartext.

This confirms an eavesdropper can directly read the content of the page and any sensitive data sent over HTTP.

3) Upgrading to HTTPS with a locally trusted certificate

(a) Why I can’t get a public CA cert for localhost

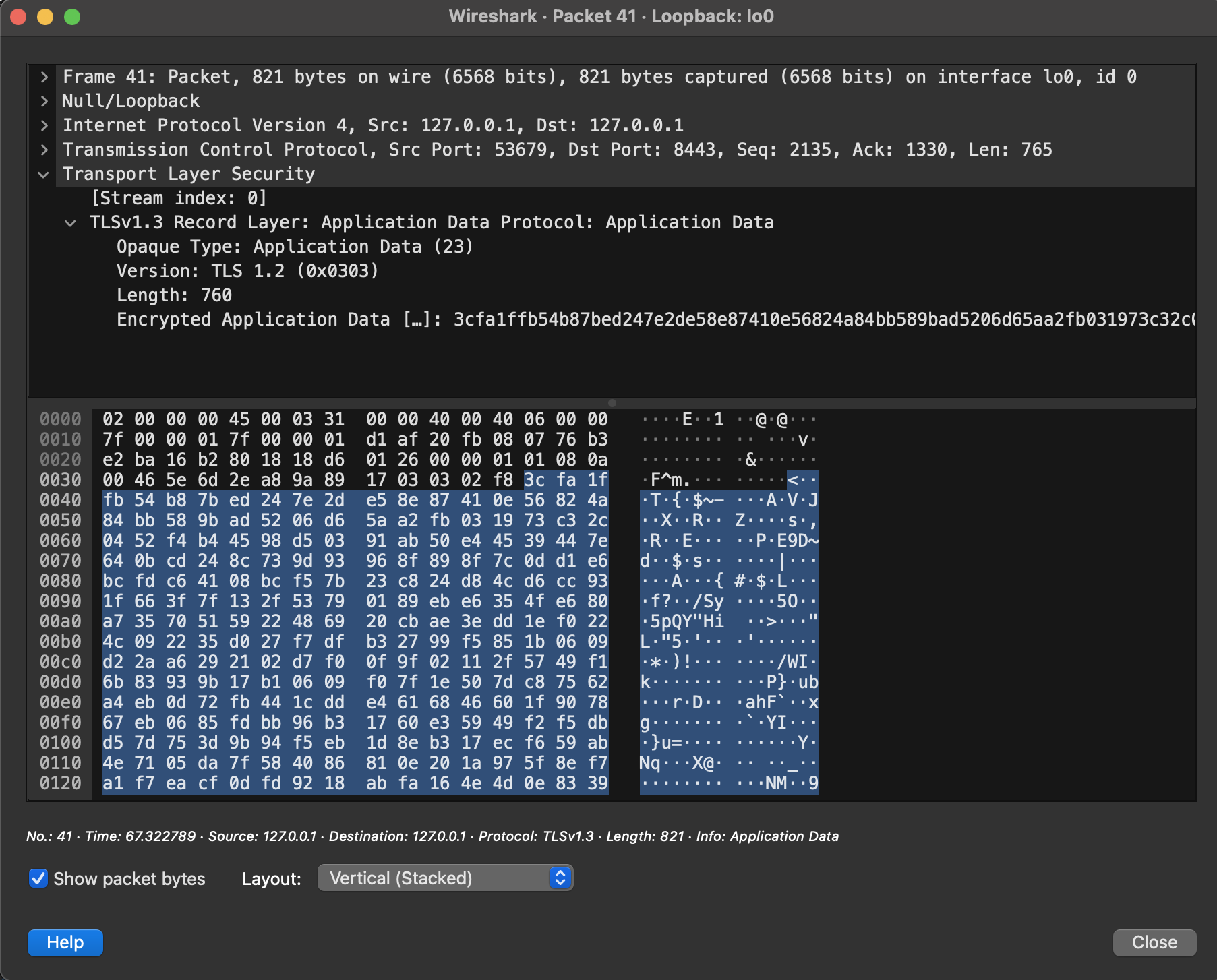
Public certificate authorities only issue certificates for public DNS names after domain-control validation. Special/internal names like localhost or private IPs (e.g., 127.0.0.1) are not eligible. Therefore, I created a local root CA, trusted it on my machine, and used it to issue a server cert for localhost.

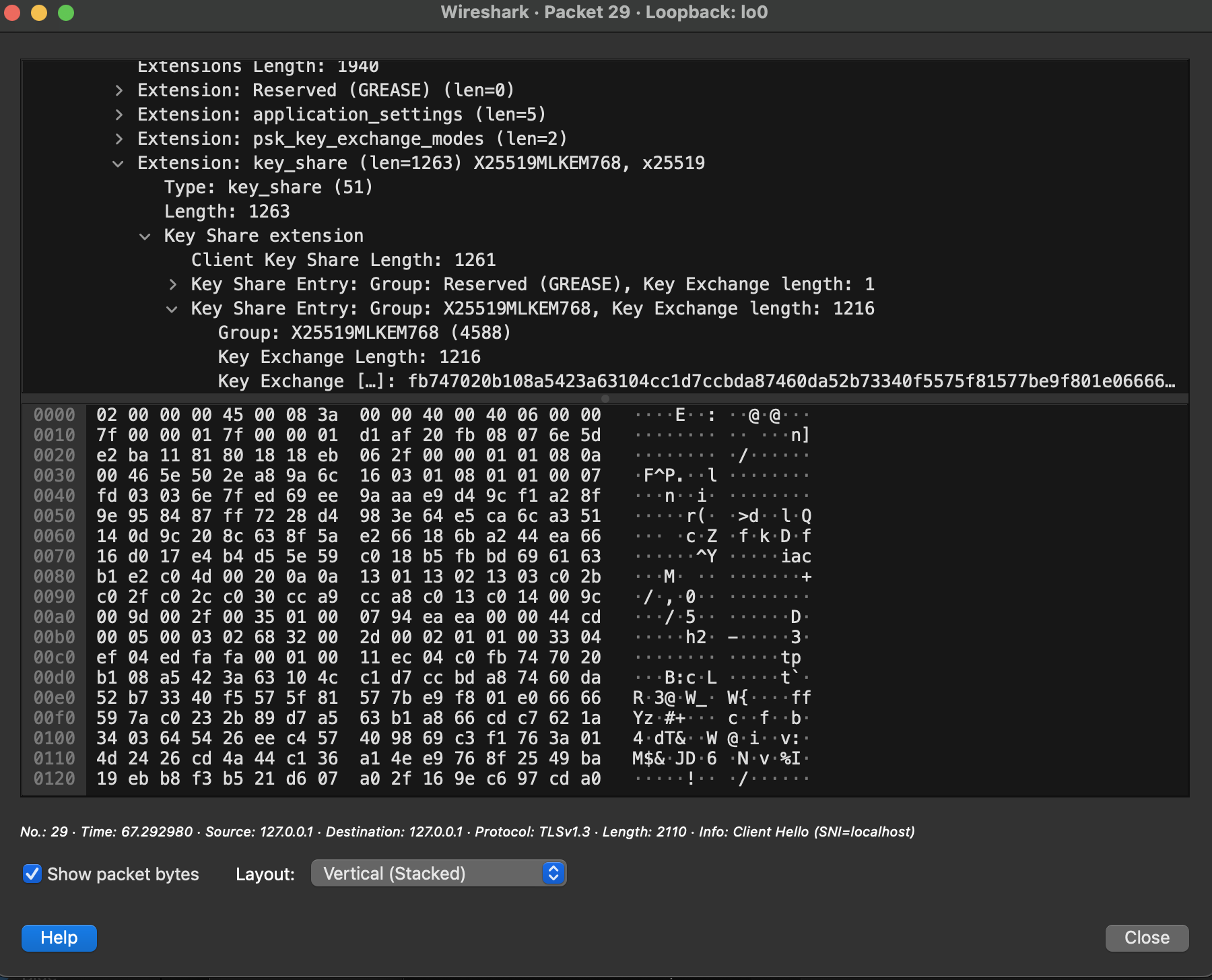
Steps performed:

1. Create a local root CA and a localhost server cert with SANs (DNS:localhost, IP:127.0.0.1, IP:::1).
2. Trust the root CA in macOS Keychain (Always Trust).
3. Run a small HTTPS server on <https://localhost:8443/> using that cert/key.
4. Verify a lock icon in the browser (no warnings).

HTTPS capture (Wireshark):

* Filter: tcp.port == 8443 (or tls).
* Fig 3: ClientHello shows SNI = localhost and ALPN (e.g., h2/http/1.1).
* Fig 4: All subsequent packets carrying the HTTP request/response appear as TLSv1.3 Application Data—opaque, unreadable bytes.





4) HTTP vs. HTTPS (TLS) — observed differences

* Confidentiality:
  + HTTP: None: full headers and body visible (Fig. 2).
  + HTTPS: Encrypted after handshake—only opaque Application Data is visible (Fig. 4).
* Integrity:
  + HTTP: None: tampering undetectable.
  + HTTPS: AEAD ciphers detect modification.
* Authentication:
  + HTTP: None: no server identity.
  + HTTPS: Server proves identity by presenting a cert chaining to a trusted root (here, my locally trusted CA).
* Metadata leakage (still visible under TLS): IPs/ports, packet sizes/timing, and SNI (the hostname) in the ClientHello (Fig. 3).

The HTTP trace demonstrates that traffic is fully readable to any eavesdropper. After upgrading to HTTPS, Wireshark shows only the TLS handshake and encrypted application data, confirming that TLS provides confidentiality, integrity, and server authentication—substantially mitigating the risks of sniffing and tampering present with HTTP.