PKI Lab Write-Up:

Ryan Ziaee

2025-10-20

1 Environment & Files Produced

All work was done on macOS using Python 3, Wireshark, and `mkcert`.
Project directory contents:

- `index.html` — A simple webpage I made in HTML.
- `https_server.py` — tiny Python HTTPS server (loads cert+key, serves on port 8443).
- `localhost+2.pem` — **certificate** (public, ID badge for `localhost`).
- `localhost+2-key.pem` — **private key** (secret; proves I own the certificate).
- `http_traffic.pcapng` — Wireshark capture of **HTTP** session (plaintext).
- `https_traffic.pcapng` — Wireshark capture of **HTTPS/TLS** session (encrypted).

2 Task 1: Host a Local Web Server (HTTP)

I hosted a simple local web server using Python's built-in module and served `index.html` on port **8000**.

Command (from project folder):

```
python3 -m http.server 8000
# Visit: http://localhost:8000
```

Why `localhost` and ports? `localhost` just means it's my machine and then I chose a port my test server could access since it wasn't already being used by something else on my machine.

3 Task 2: Why HTTP Is Not Secure

HTTP is unencrypted. The way I think of it is like sending a letter with no envelope – anyone can read the data packets being sent as there's not even an attempt at encrypting these data packets (envelope = encryption).

Capture steps (HTTP):

1. Start Wireshark; capture on the **lo0** (loopback) interface.
2. Apply display filter `http`.
3. Browse `http://localhost:8000`.
4. Stop capture and save as `http_traffic.pcapng`.

What is visible:

- **Request line:** e.g., `GET / HTTP/1.1` – browser telling server where it wants to go and what it wants to use to do it.
- **Headers:** `Host`, `User-Agent`, etc. – similar to how request line works with more content about the request and client.
- **Response body:** Most importantly, the actual HTML from `index.html` in plaintext – if there was encryption this plaintext would appear as gibberish, but since http doesn't encrypt, the plaintext is intelligible.

4 Task 3: Create a Self-Signed Certificate and Upgrade to HTTPS

(a) Why a public CA will not issue a cert for `localhost`

Public Certificate Authorities (CAs) issue certificates only for **public, verifiable domains** and localhost is only accessible on my machine. `localhost` is a special private name that always points to my machine; it's not publicly verifiable, so a trusted CA will not sign it.

Generate a locally trusted cert (with `mkcert`)

```
# Install mkcert (one time):
brew install mkcert

# Install mkcert's local root CA into macOS trust store (one time):
mkcert -install

# Issue a cert for localhost and its loopback addresses:
mkcert localhost 127.0.0.1 ::1
# -> produces: localhost+2.pem (certificate), localhost+2-key.pem (private key)
```

How I internalize these concepts:

- **Certificate** (`.pem`) = an ID card with the site name and public key.
- **Private key** (`-key.pem`) = the secret stamp that proves I own that ID.
- **mkcert** creates a small local CA and tells macOS to trust it, so my browser accepts the cert for `https://localhost`.

Run the HTTPS server and capture traffic

```
# Start the HTTPS server (serves on port 8443 by default)
python3 https_server.py

# Visit: https://localhost:8443
# In Wireshark (lo0), use filter: tls      # (or: tcp.port == 8443)
# Save as: https_traffic.pcapng
```

What changes over HTTPS/TLS:

- The handshake shows **Client Hello**, **Server Hello**, and **Certificate**.
- After the handshake, packets are **Application Data** (encrypted gibberish).
- No readable HTML or HTTP headers are visible on the wire.

5 What TLS Adds

TLS provides three protections:

1. **Encryption** (privacy): eavesdroppers see only ciphertext.
2. **Integrity** (tamper detection): modifications are detected.
3. **Authentication** (identity): the certificate proves the server's identity (via a chain of trust).

6 Comparison: What Wireshark Shows

Aspect	**HTTP (port 8000)**	**HTTPS/TLS (port 8443)**
Visibility	Full request/response in cleartext	Only handshake metadata + encrypted application data
Encryption	None	Yes (session key after handshake)
Integrity	None (tampering undetectable)	Protected (tampering detected)
Authentication	None	Server identity checked via certificate
Wireshark filter	`http`	`tls` or `tcp.port == 8443`

7 Answers to the assignment questions

1. **Why is HTTP not secure?**
   To put it simply, it's that there's no encryption of the data that's being transferred between the website and my machine. This introduces many avenues for malicious activity to take place – like in my letter analogy, anyone could read my mail with very little effort to do so involved. They could steal it, misdirect it, alter it, etc...

2. **Why can't I obtain a public CA certificate for my local server?**
   There's a chain of trust when it comes to certificates. These certificates have to be issued by trusted Certificate Authenticators (CAs), and no established, trustworthy CA is going to issue a certificate for a private and non-verifiable domain like localhost.

3. **What is different between HTTP and HTTPS in the packet traces?**
   HTTP shows readable `GET`/`Host`/HTML content. HTTPS shows a TLS handshake (`Client Hello`, `Server Hello`, `Certificate`) and then only encrypted Application Data; the content is not readable.

8 Deliverables

- `This Document` (Writeup) — Documents steps, thought process, and answers.
- `http_traffic.pcapng` — HTTP packet trace.
- `https_traffic.pcapng` — HTTPS packet trace.

Appendix: Terminal Commands Used for Setup

HTTP server

```
python3 -m http.server 8000
# Visit: http://localhost:8000
```

Wireshark (HTTP)

```
Interface: lo0
Display filter: http
Save: http_traffic.pcapng
```

mkcert + HTTPS

```
brew install mkcert
mkcert -install
mkcert localhost 127.0.0.1 ::1
# -> localhost+2.pem, localhost+2-key.pem
python3 https_server.py
# Visit: https://localhost:8443
```

Wireshark (HTTPS)

```
Interface: lo0
Display filter: tls   # or: tcp.port == 8443
Save: https_traffic.pcapng
```
