

Ahmad Nadi  
Security, Privacy, and Consumer Protection  
Prof. Nick Feamster  
October 24, 2025

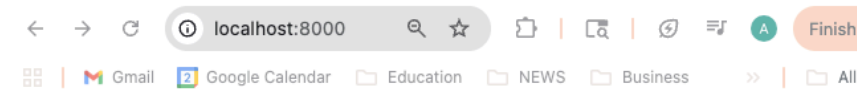
## HW1: Public Key Infrastructure HW

### Task 1:

I set up a Python HTTP server by running the simple command:

```
ahmad@MacBook-Air-6 ~ % python3 -m http.server
```

It then displayed all of my file structure:



### Directory listing for /

- [.android/](#)
- [CFUserTextEncoding](#)
- [codex/](#)
- [config/](#)
- [DS\\_Store](#)
- [emulator\\_console\\_auth\\_token](#)
- [expo/](#)
- [npm/](#)
- [swiftpm/](#)
- [thumbnails/](#)
- [.Trash/](#)
- [vim/](#)
- [viminfo](#)
- [vscode/](#)
- [vscode-react-native/](#)
- [zprofile](#)
- [zsh\\_history](#)
- [zsh\\_sessions/](#)
- [Applications/](#)
- [compsecurityclass/](#)
- [Desktop/](#)
- [Documents/](#)
- [Downloads/](#)
- [focusapp/](#)
- [HCL/](#)
- [lab2/](#)
- [Library/](#)
- [miktex-console.lock](#)
- [Movies/](#)
- [muhammadalifirstprogram.py](#)
- [Music/](#)
- [Pictures/](#)
- [ReaderApp/](#)
- [Sites/](#)
- [tommy/](#)
- [Tradez/](#)
- [vscode-remote-wsl/](#)
- [websiteCompany/](#)

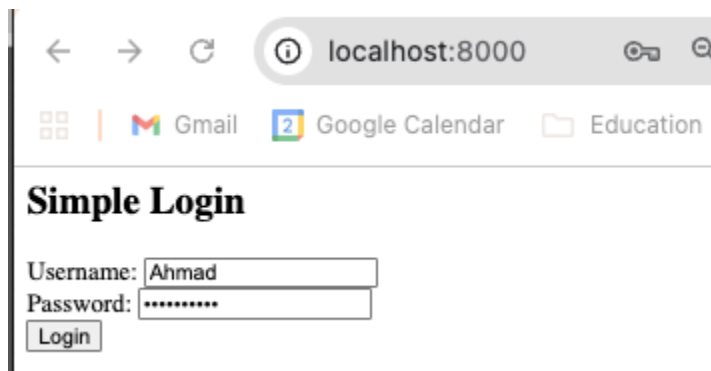
### Task 2:

Given the spoiler, I knew it would be beneficial to send data across the site I hosted so that I could test how a person could sniff unencrypted data between my HTTP server and me. Thus, I

had to create a more complex script that could set up a Python HTTP server and include a basic HTML login, which allowed me to send data to the server. I used some ChatGPT help to fix some bugs, and the resulting code looked like this:

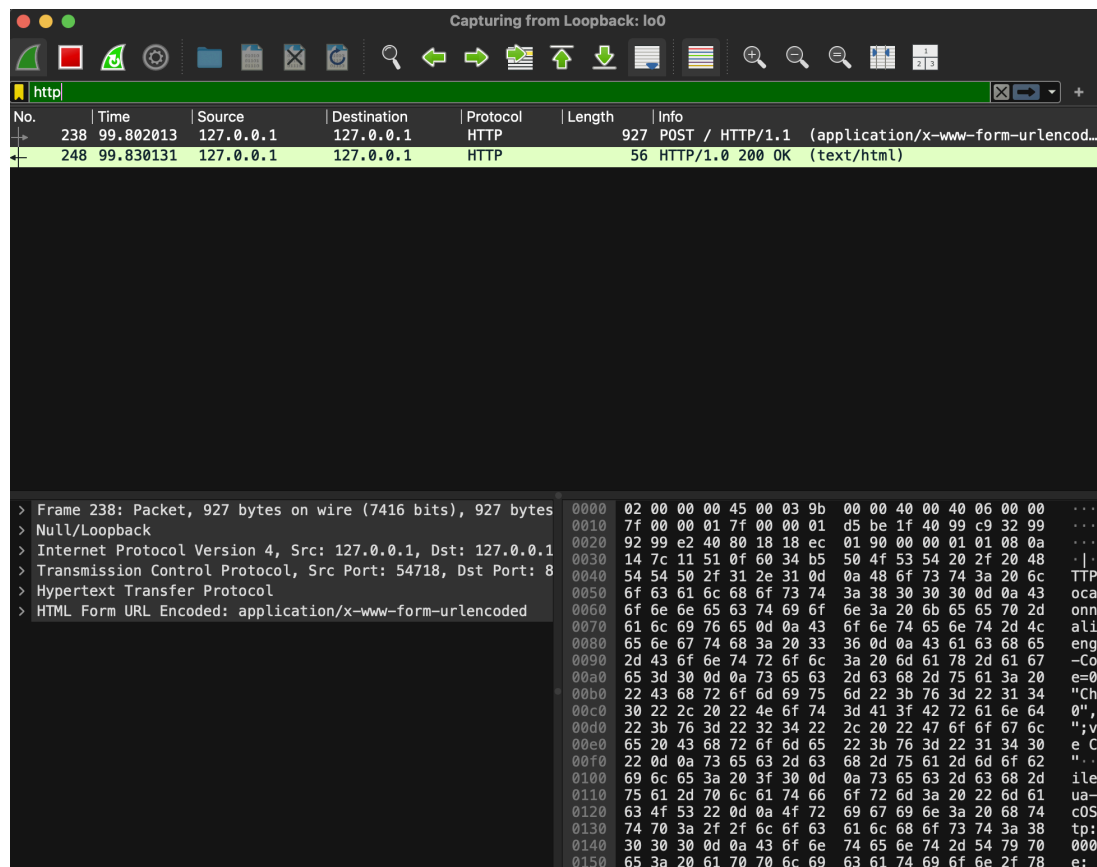
```
server.py x
server.py > LoginHandler > do_GET
1  ~./compsecurityclass/hw1/server.py  seHTTPRequestHandler, HTTPServer
2  import urllib.parse
3
4  class LoginHandler(BaseHTTPRequestHandler):
5      def do_GET(self):
6          self.send_response(200)
7          self.send_header("Content-type", "text/html")
8          self.end_headers()
9          self.wfile.write(b"""
10             <html><body>
11             <h2>Simple Login</h2>
12             <form method='POST'>
13                 Username: <input name='username'><br>
14                 Password: <input name='password' type='password'><br>
15                 <input type='submit' value='Login'>
16             </form>
17             </body></html>
18             """)
19
20     def do_POST(self):
21         length = int(self.headers['Content-Length'])
22         post_data = self.rfile.read(length)
23         data = urllib.parse.parse_qs(post_data.decode())
24
25         username = data.get('username', [''])[0]
26         password = data.get('password', [''])[0]
27
28         if username == 'admin' and password == 'password':
29             message = f"<h3>Welcome, {username}</h3>"
30         else:
31             message = "<h3>Invalid credentials.</h3>"
32
33         self.send_response(200)
34         self.send_header("Content-type", "text/html")
35         self.end_headers()
36         self.wfile.write(message.encode())
37
38     if __name__ == "__main__":
39         PORT = 8000
40         with HTTPServer(("", PORT), LoginHandler) as httpd:
41             print(f"Serving on port {PORT}")
42             httpd.serve_forever()
43
```

It displayed this:

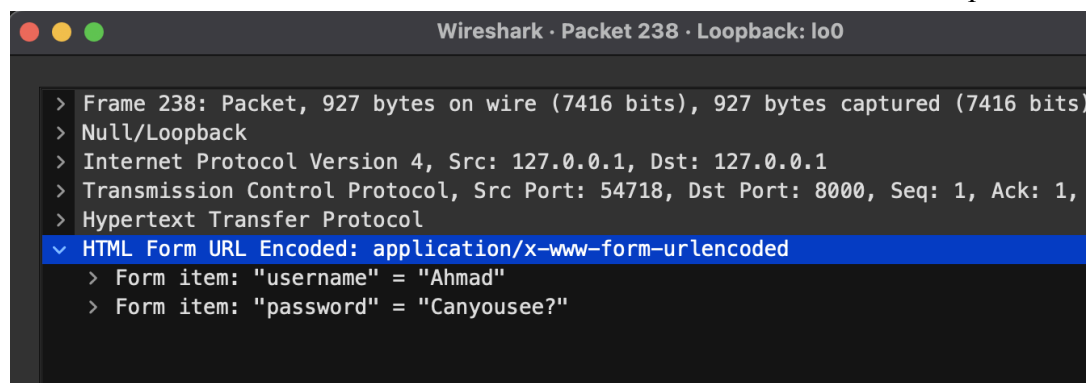


The screenshot shows a web browser window with the address bar set to `localhost:8000`. The browser's tab bar shows several tabs: "Gmail", "Google Calendar", and "Education". The main content area displays a simple login form titled "Simple Login". The form contains two input fields: "Username:" with the text "Ahmad" entered, and "Password:" with masked characters (dots). Below the password field is a "Login" button.

In Wireshark, I filtered for HTTP traffic:



And I was able to see what I had submitted to the form for a username and password:



I was able to sniff the username (“Ahmad”) and password (“Canyousee?”) sent to the server, which is pretty cool but also risky and scary. This really shows how HTTP is not secure because it transmits everything in plaintext, making it easily readable to anyone trying to catch my data. This means that anyone on the same Wi-Fi network with malicious intent could see exactly what I send to websites and which sites I visit. Relying solely on HTTP means not only can someone see the websites you visit, but they can also see all the information you send to those sites. For

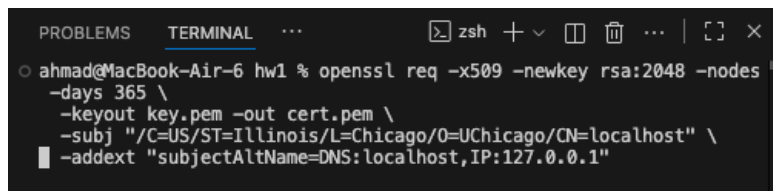
example, if I were on a banking website that uses HTTP, a sniffer could capture my username and password, then log in and do whatever they want with my money.

### Task 3:

*(a) Why can't you obtain an SSL certificate for your local web server from a certificate authority?*

You can't obtain an SSL certificate for a local web server because certificate authorities only issue certificates for publicly verifiable domains. The domain needs to be public in order for it to be verified by the certificate authority, as they need to access it. Local servers that run on my computer can't be accessed from outside, and thus I need to use a self-signed certificate.

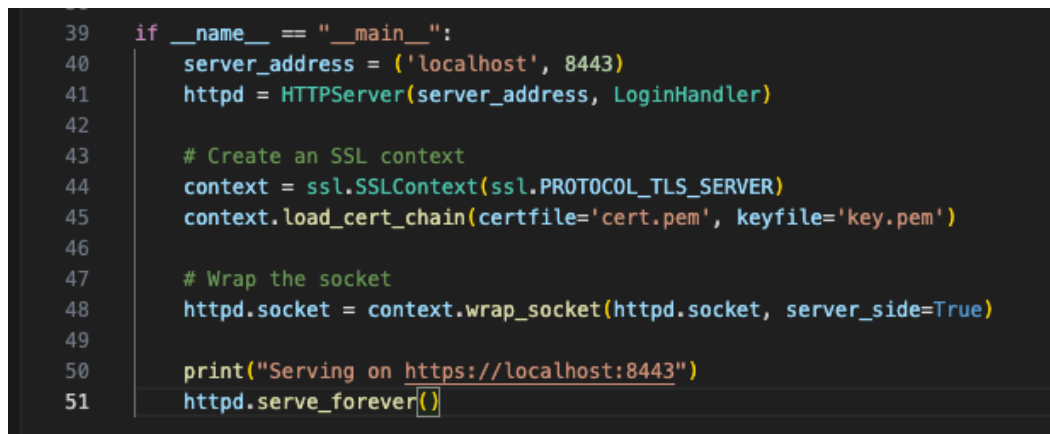
To enable HTTPS, I created an SSL/TLS certificate using OpenSSL. The command generated two files: a private key (key.pem) and a public certificate (cert.pem). My local server uses these keys to encrypt connections.



```
PROBLEMS  TERMINAL  ...  zsh  +  -  [ ]  [ ]  ...  |  [ ]  [ ]  x
o ahmad@MacBook-Air-6 hw1 % openssl req -x509 -newkey rsa:2048 -nodes \
  -days 365 \
  -keyout key.pem -out cert.pem \
  -subj "/C=US/ST=Illinois/L=Chicago/O=UChicago/CN=localhost" \
  -addext "subjectAltName=DNS:localhost,IP:127.0.0.1"
```

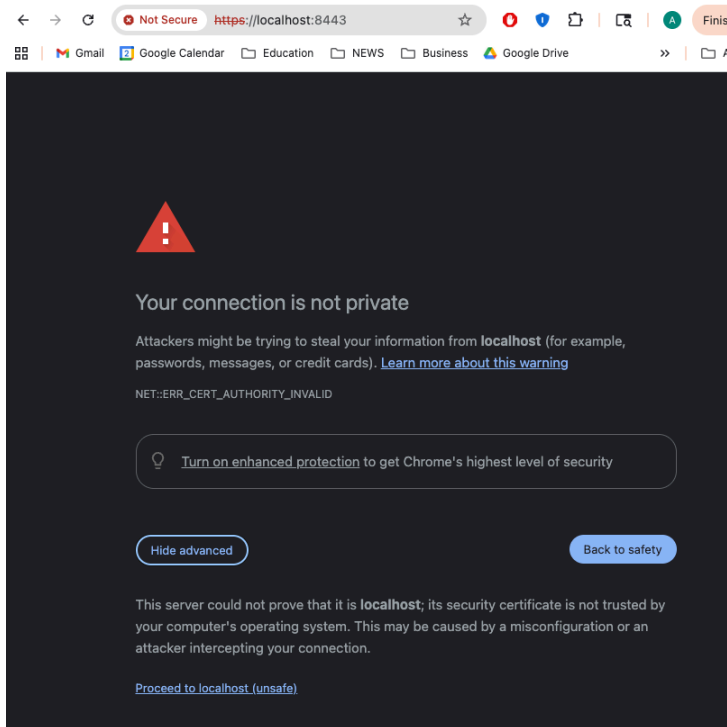
I also edited my script with the help of ChatGPT in order to have it work and serve my Python web server securely. The main change was that I needed to create an SSL context that specified the use of the TLS protocol and loaded the self-signed certificate and private key I made earlier.

In addition to importing ssl, this was the addition to my script:

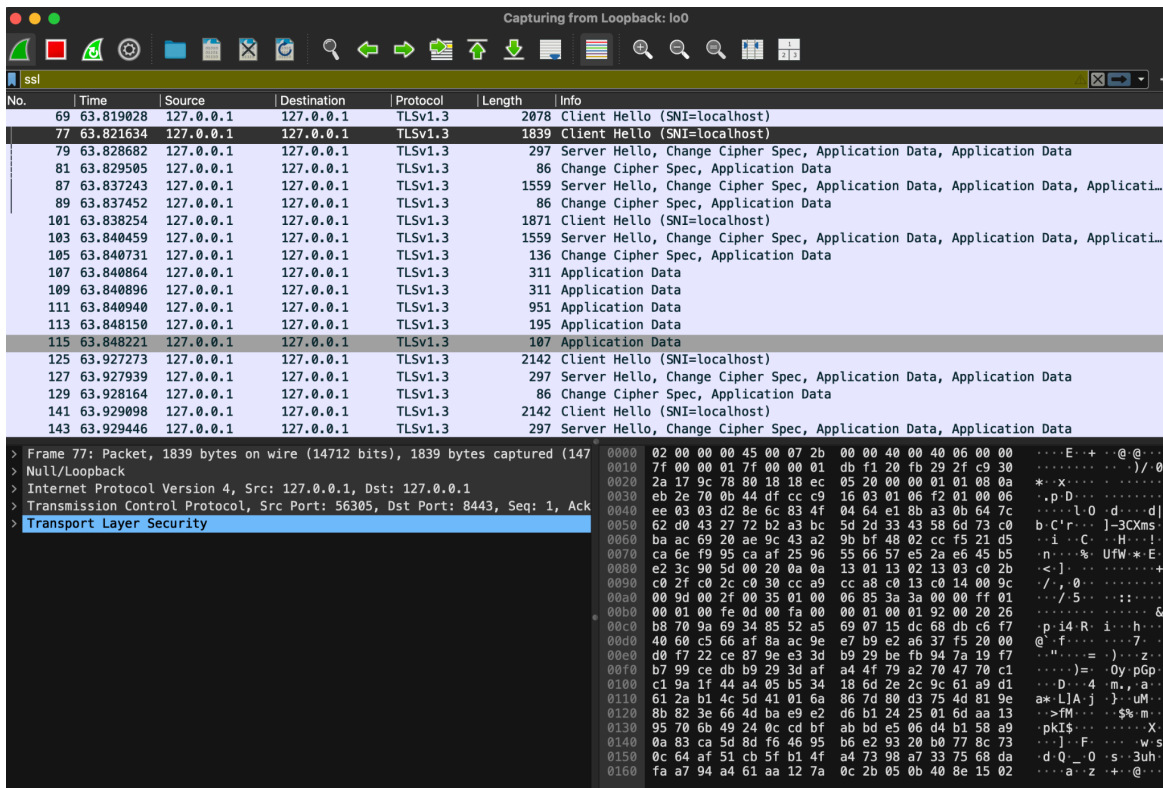


```
39  if __name__ == "__main__":
40      server_address = ('localhost', 8443)
41      httpd = HTTPServer(server_address, LoginHandler)
42
43      # Create an SSL context
44      context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
45      context.load_cert_chain(certfile='cert.pem', keyfile='key.pem')
46
47      # Wrap the socket
48      httpd.socket = context.wrap_socket(httpd.socket, server_side=True)
49
50      print("Serving on https://localhost:8443")
51      httpd.serve_forever()
```

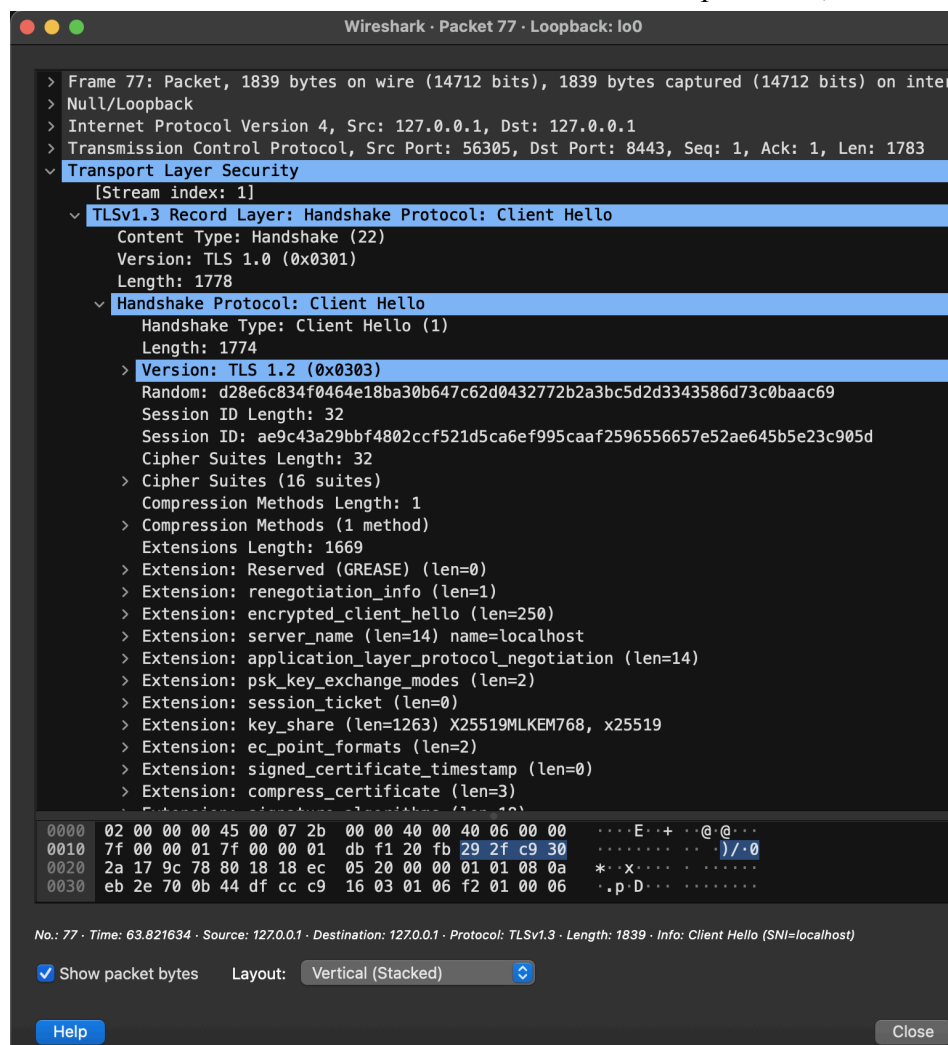
This was how it loaded:



But I proceeded to my localhost and captured my traffic using Wireshark:



And I couldn't see what I had sent for a username and password, as it was encrypted.



This overall showed just how much more secure and necessary HTTPS is, as it protects your information when it's transmitted and ensures privacy even over all networks. It really highlighted how vulnerable plain HTTP can be and how easily data could be intercepted without encryption. I had never used Wireshark, but clearly it is a very powerful tool that I'm sure many malicious people have access to and try to use in order to gain access to information they want but don't have access to. This project highlighted the importance of HTTPS, as with it, any sniffers on your network will not be able to see the plain text of what you're sending, and through Wireshark, they can only see encrypted gibberish that's practically unreadable without the right key.