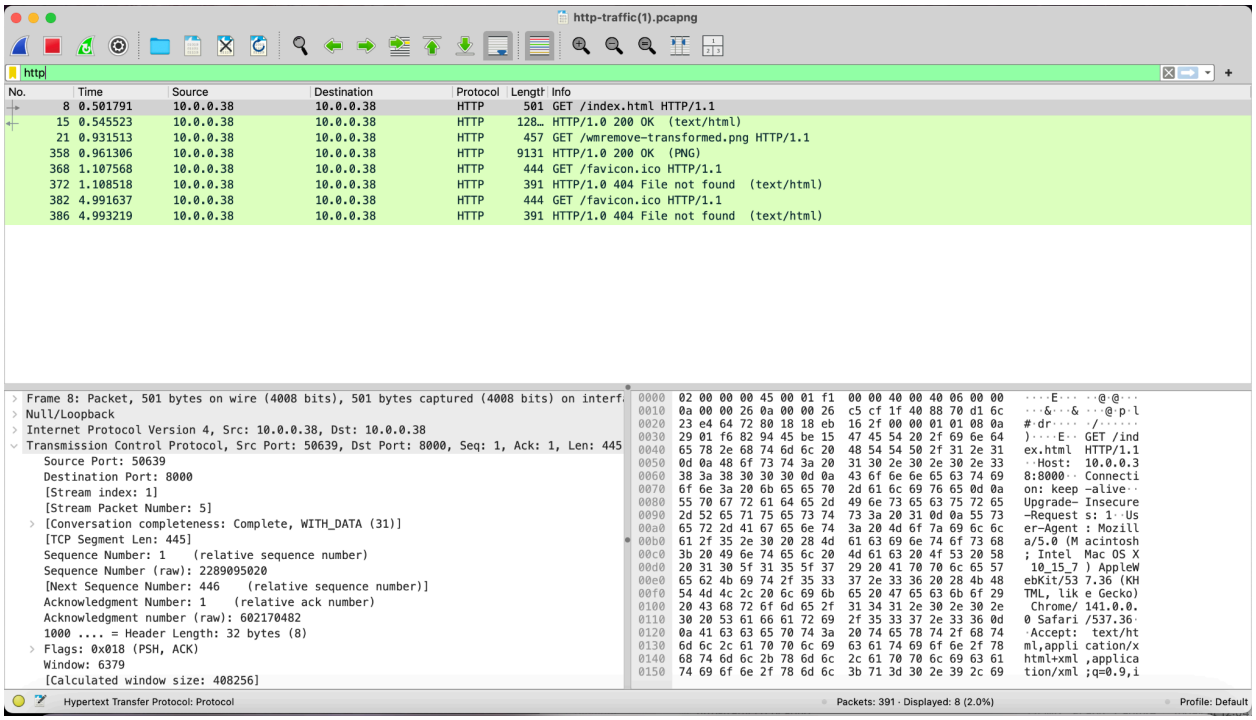
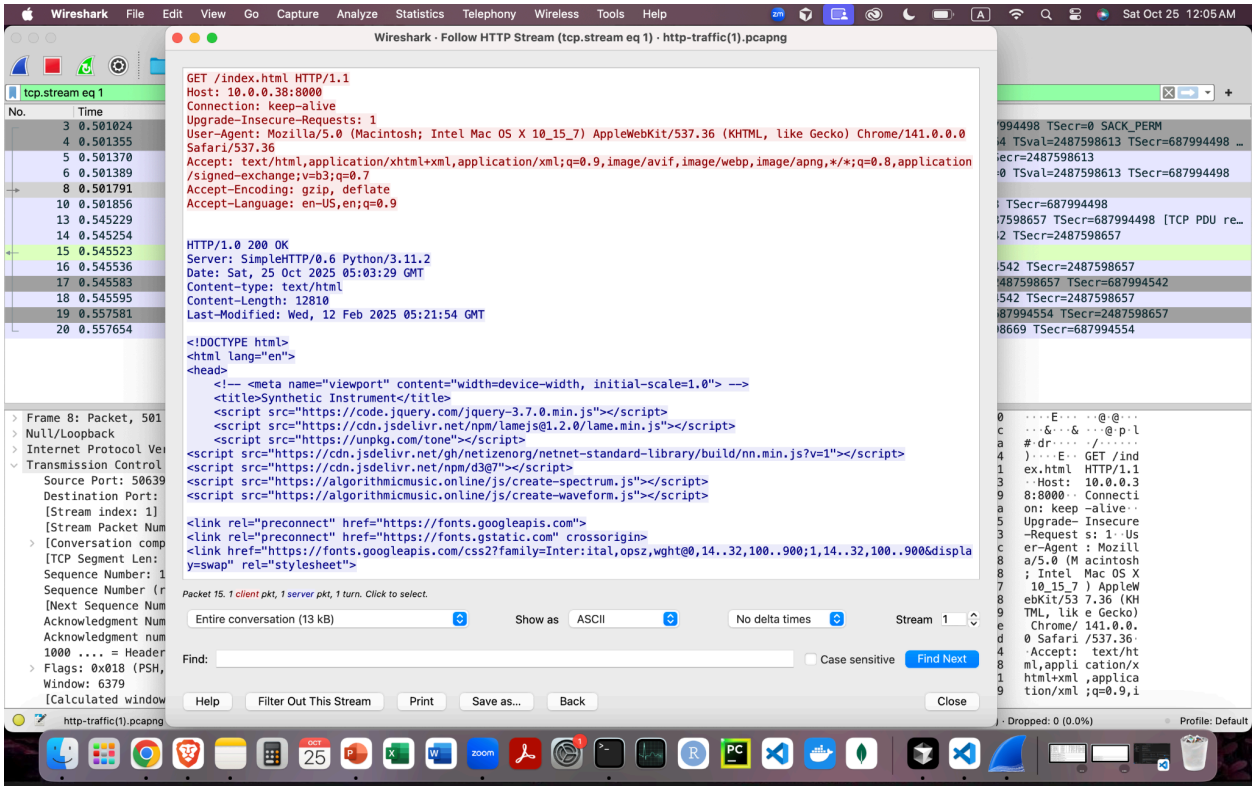
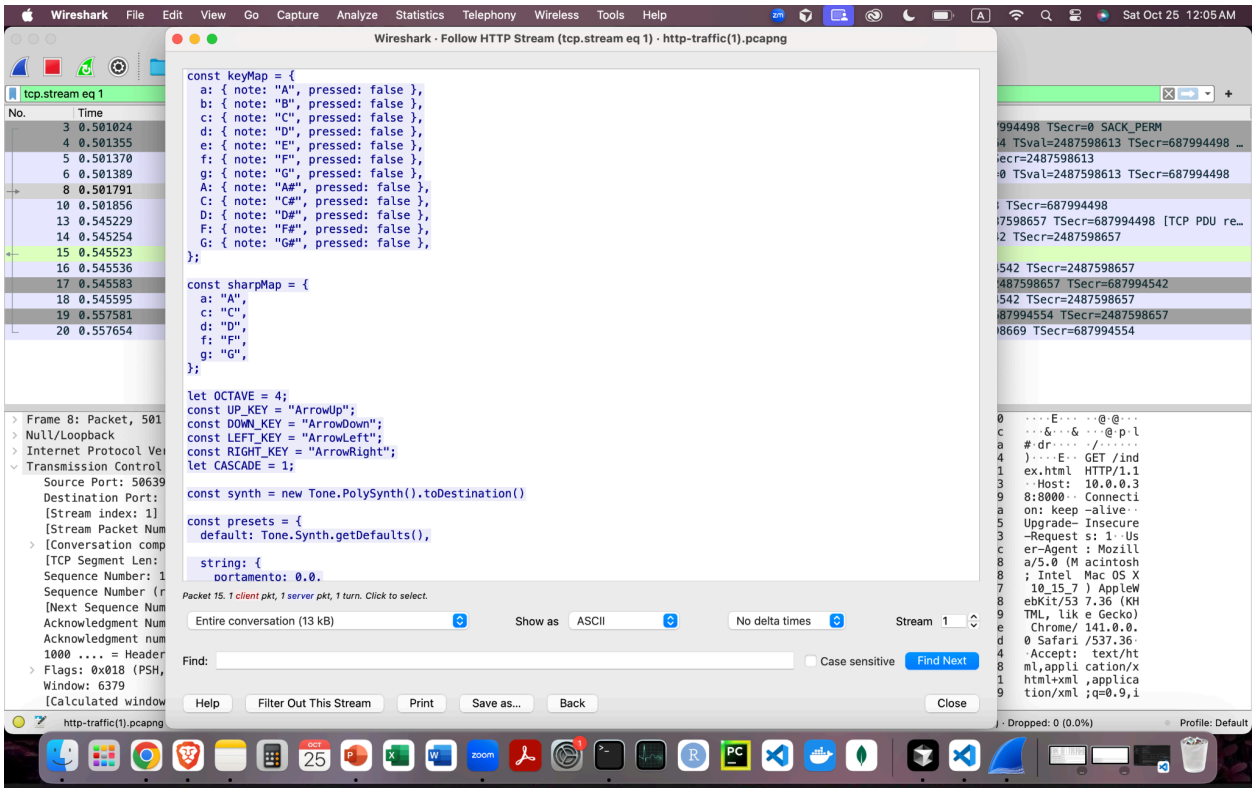
1. For step 1, I just went with using the HTTP class. Specifically, I ran `python3 -m http.server 8000` at first; however, once i downloaded wireshark, I realized that wireshark wasn't actually able to capture the traffic nor in the right format, which a quick search informed me was not an uncommon problem with macs, so I just ahead and bound it to the IP address instead (`python3 -m http.server 8000 --bind 10.0.0.38`), so that even though it's still hosted locally, it'd have to pass through the network all the same, and that fixed the issue.

2. Why HTTP is not secure:
All the data is transmitted unencrypted, in plain text, so anyone could 'listen in' on the connection and intercept and then read the data. Anyone who can see the network traffic can read the information being sent - for example, anyone can see what is being transmitted, as seen in the screenshot below:



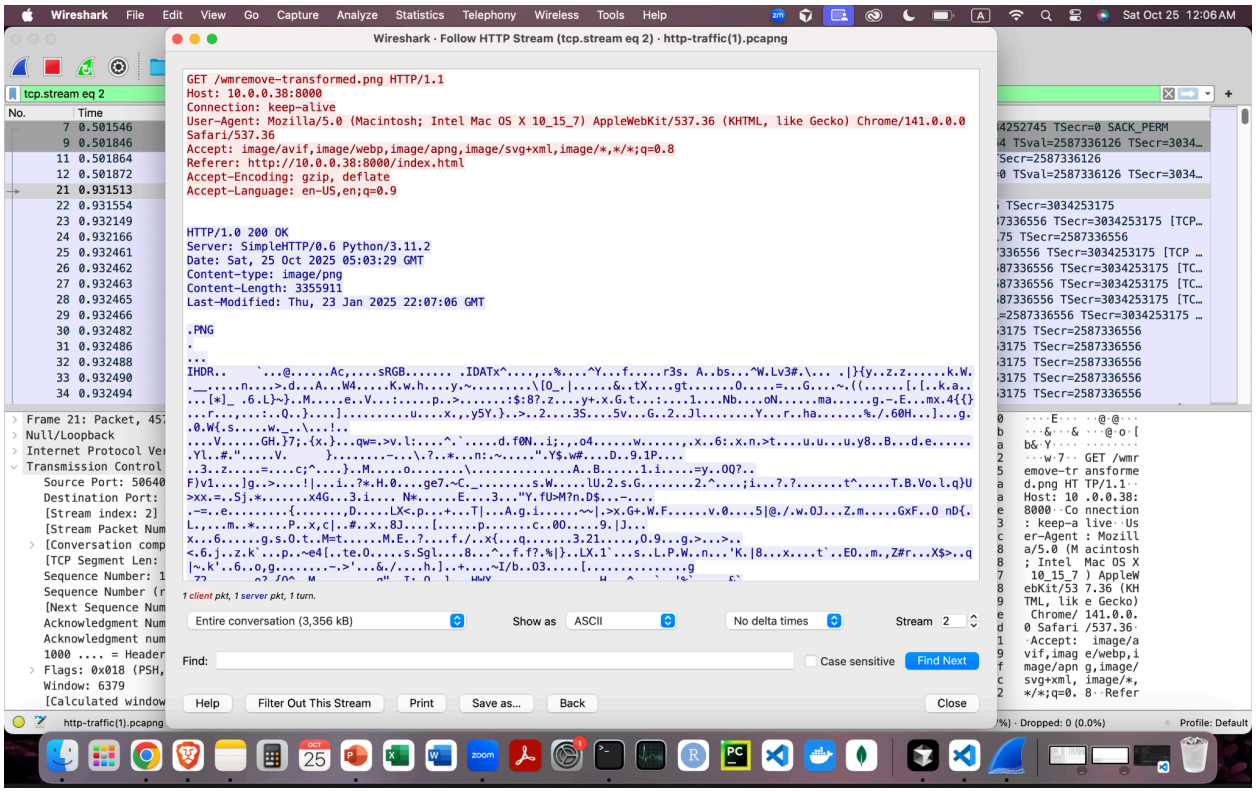
The HTTP requests and responses are all laid out - at every step, anyone monitoring can see exactly what is taking place, what resources are being fetched (the text at no.15, the png at no 358, etc.). Moreover, beyond that, the contents are also plainly available to read as well; take the following two screenshots below of the HTTP stream captured in wireshark:





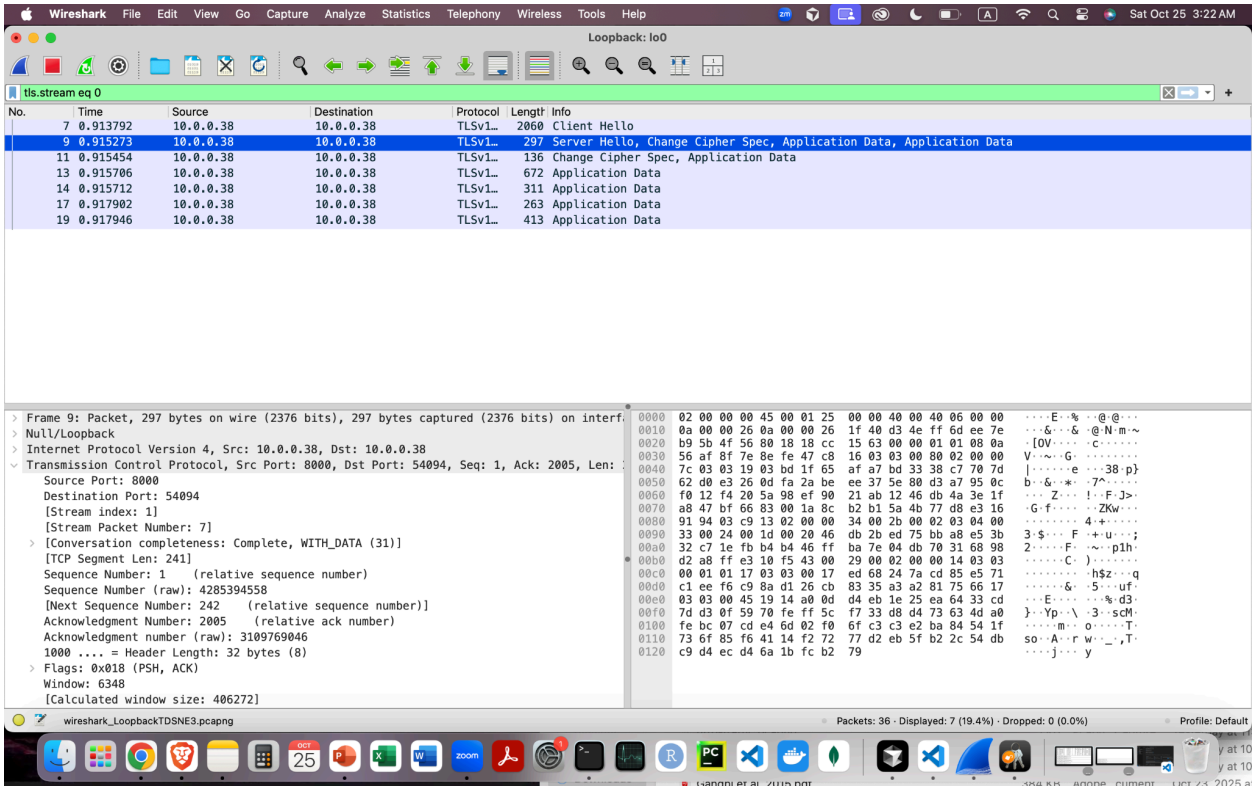
This is exactly the contents of the html file that is being fetched by the browser, out there for anyone to read. So - people can easily see the contents of what is being transmitted. This

extends towards any images and other objects loaded into the page as well, beyond even just the page itself:

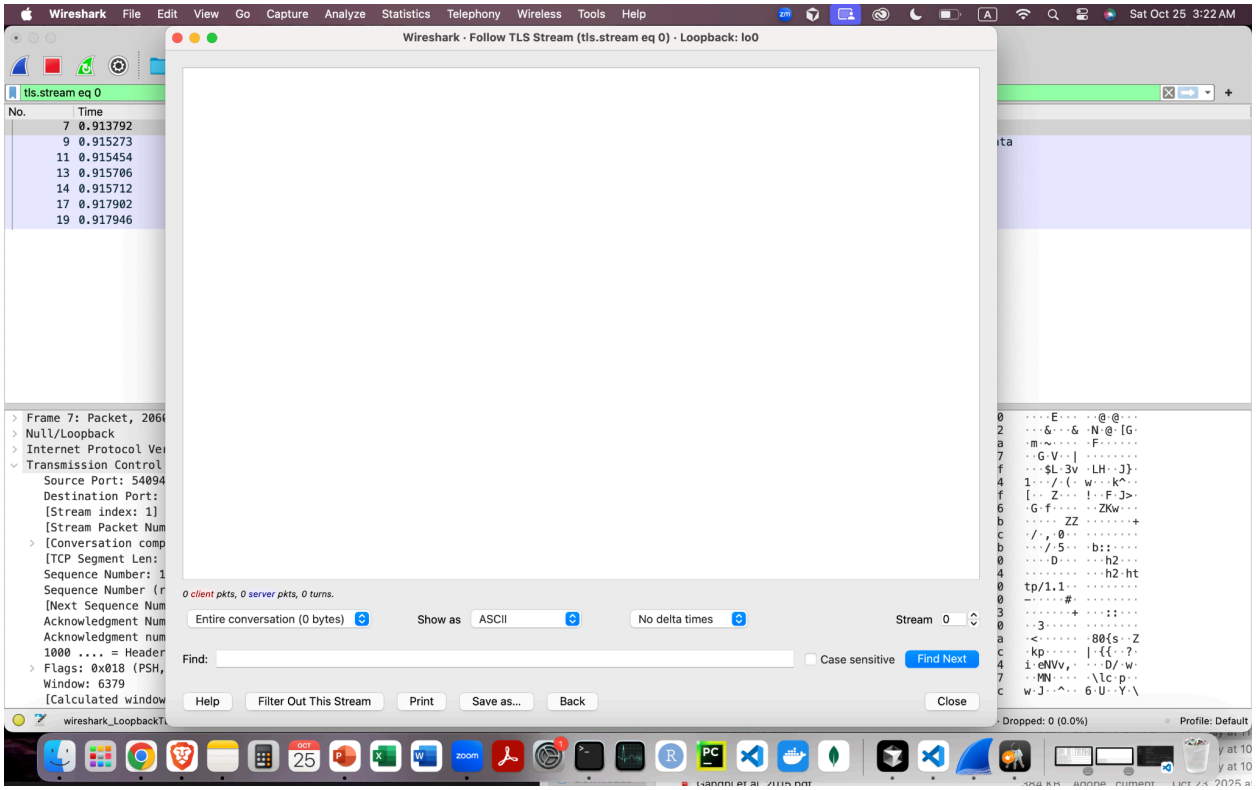


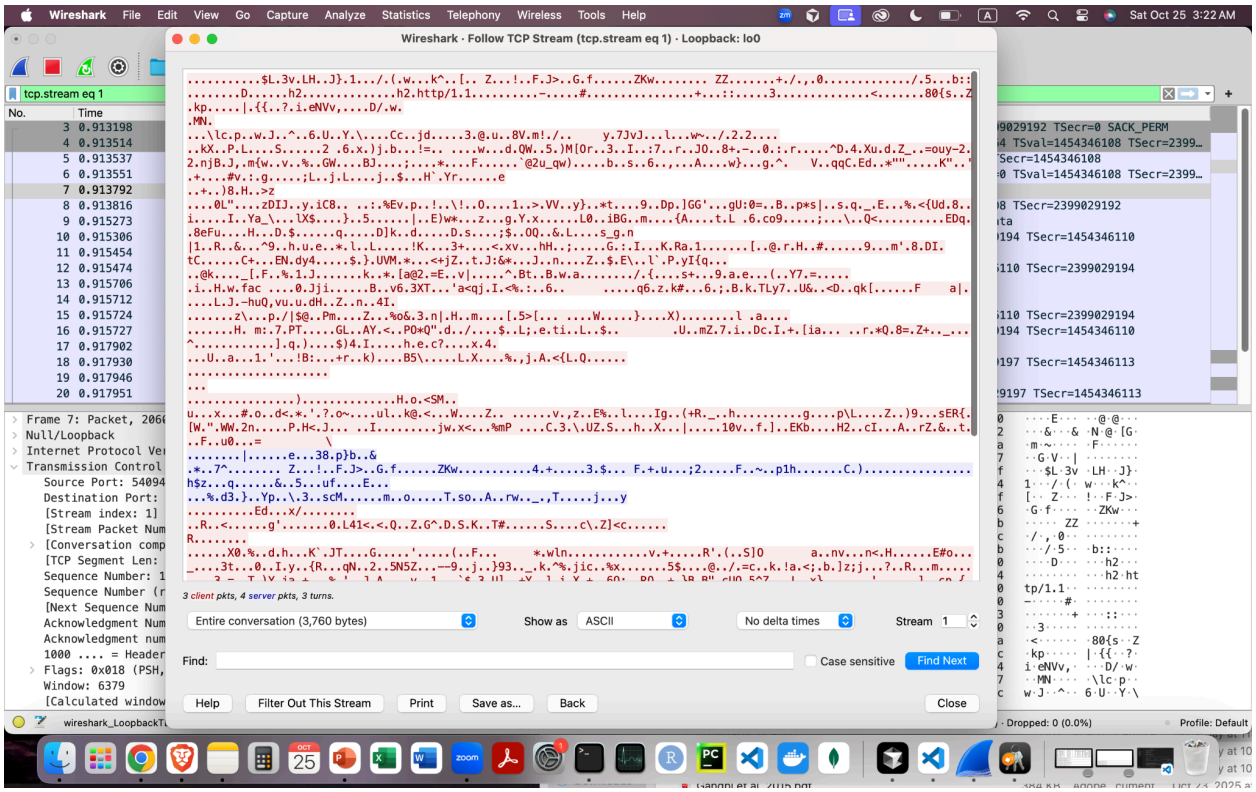
So, by clicking on any stream and going to follow -> HTTP stream you can read any and all activity easily.

3. Now, in comparison, see the network trace logs with HTTPS:



First, the protocol is no longer HTTP - it is using TLS to encrypt the data before transmission. In sharp contract to the HTTP, now exactly what is being exchanged is obfuscated - all we see is the initial exchange between the server and the browser, the client and server hellos, and then the exchange of data - but that is all incredibly vague, such that to anyone looking in it wouldn't be clear what is being transmitted when. Whereas in contrast, in the HTTP file it was clearly indicated when the html was being transmitted, or the png file. Now, if we follow the network stream, we see:





In the TLS stream, instead of nice pretty plain text anyone can read, we get a blank load of nothing - wireshark is unable to access the data being transmitted, encrypted as it is and without a key to the encryption. Communications are now secured.

a) Why you can’t obtain an SSL certificate for your local web server from a certificate authority: this is because a local web server is not publicly accessible and so cannot be verified. In order to obtain a certificate, you need to prove that you own the domain, which you cannot do with a localhost server (that is not a unique domain and is not owned by anyone).

As for how I did it, I just ran it in the terminal (after some consultation with gpt):
`openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout server.key -out server.crt -config openssl-san.cnf`

And then manually went to Keychain access to add the certificate to the system and switched it to ‘trust always’. To run the HTTPS server, I just wrote a quick https_server.py with http.server, ssl.