**ELEC3506/9506 Communication Networks**

**-Lab Report 3**



Zhixuan Lin(530034414) -Contributinos(50%) -ELEC3506

Xinran He(540662177) -Contributions(50%) -ELEC9506

Table of Contents

Introduction……………………………………………………………………………2

Phase 1: TCP………………………………………………………………………

Phase 2: HTTP…………………………………………………………………………

Appendix………………………………………………………………………………

**Introduction**

In this lab, we aim to deepen our understanding of how core Internet protocols operate in real network environments. Using **Wireshark**, we captured and analyzed packet exchanges to observe the behavior of both **TCP** and **HTTP**. Through the TCP phase, we examined connection establishment, sequence and acknowledgment numbers, flow and congestion control, and transmission performance. In the HTTP phase, we explored how web communication occurs through GET/response interactions, caching, file transfers, embedded objects, and authentication. By analyzing real packet traces, we connected theory with practice and gained a clearer view of how reliability, efficiency, and security are achieved across the Internet’s transport and application layers.

**Phase 1: TCP**

**Phase 2: HTTP**

**Objective**

**The goal of Phase 2 is to understand how HTTP functions in real network exchanges.**

**Using Wireshark, we analyzed the following HTTP behaviors:**

**1.Basic GET/response interaction**

**2.Conditional GET and caching**

**3.Retrieval of large HTML files**

**4.HTML pages with embedded objects**

**5.Authentication via HTTP Basic scheme**

**All packets were captured between the local client (10.170.56.60) and the web server (128.119.245.12).**

**2.1 Basic HTTP GET/Response**

URL: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html

Main packets: #254 (GET) and #263 (200 OK)

**Findings**

Browser → Server uses HTTP/1.1, and the server replies in the same version.

The client advertises support for language en-US.

Response status code 200 OK confirms successful transfer.

File last-modified time is dynamically updated by the server (changes every minute).

Content length ≈ 486 bytes.

Hidden headers (e.g. Date, Server) are visible in raw data only.

An additional automatic request for favicon.ico returns 404 Not Found, which is browser-generated and unrelated to the experiment.

**Interpretation**

This step shows the fundamental client–server transaction where a single GET request retrieves a simple HTML file through persistent HTTP/1.1 connection.

**2.2 Conditional GET and Browser Cache**

URL: HTTP-wireshark-file2.html

Main packets: #189, #198, #3745, #3747

**Observations**

The first GET has no If-Modified-Since field.

The server replies 200 OK and sends the complete file.

The second GET includes If-Modified-Since: header with a timestamp.

The server replies 304 Not Modified, returning only headers (no entity body).

**Interpretation**

This confirms the Conditional GET mechanism: when the cached version is still valid, the browser avoids re-downloading data, conserving bandwidth and reducing latency.

**2.3 Retrieving Long Documents**

URL: HTTP-wireshark-file3.html

Main packets: #180 (GET) and #195 (200 OK with 4 segments)

**Observations**

One GET request initiates the transaction.

The response status code 200 OK.

The full HTML (≈ 4.8 KB) is transmitted in four TCP segments, each ≈ 1460 bytes.

Wireshark reassembles the pieces and shows “Reassembled TCP Segments (4861 bytes)”.

**Interpretation**

A large document exceeding one MSS (Maximum Segment Size) is automatically divided by TCP and later reconstructed. This demonstrates TCP segmentation + reassembly at work.

**2.4 HTML Documents with Embedded Objects**

URL: HTTP-wireshark-file4.html

Main packets: #217 (main HTML), #233 (pearson.png GET), #255 (200 OK PNG)

**Observations**

The browser sends two GET requests – one for the base HTML and one for an embedded image.

Both requests go to 128.119.245.12.

The HTML response (≈ 1.3 KB) and image response (≈ 0.7 KB) both return 200 OK.

The two requests are sent sequentially, with ~0.03 s interval, not overlapping.

**Interpretation**

The browser first fetches and parses the HTML, then issues new GET requests for referenced objects. In this capture, objects were downloaded serially, not in parallel, revealing potential latency in non-pipelined HTTP/1.1 transfers.

**2.5 HTTP Authentication**

URL: protected\_pages/HTTP-wireshark-file5.html

Main packets: #160 (GET), #190 (401 Unauthorized), #1186 (GET with Authorization), #1206 (200 OK)

**Observations**

The first GET triggers a 401 Unauthorized reply.

The response header specifies authentication scheme Basic realm.

The browser re-issues the GET request with header

Authorization: Basic d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=

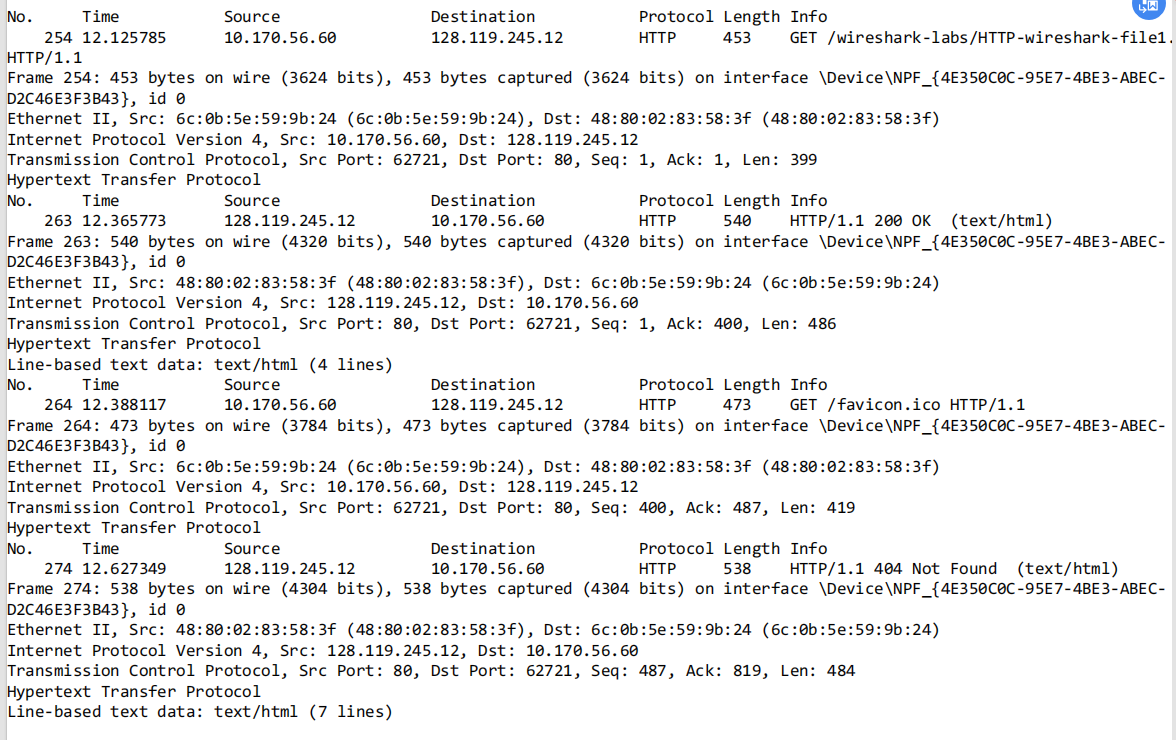
Decoding this Base64 string yields username = wireshark-students, password = network.

The second response returns 200 OK, granting access.

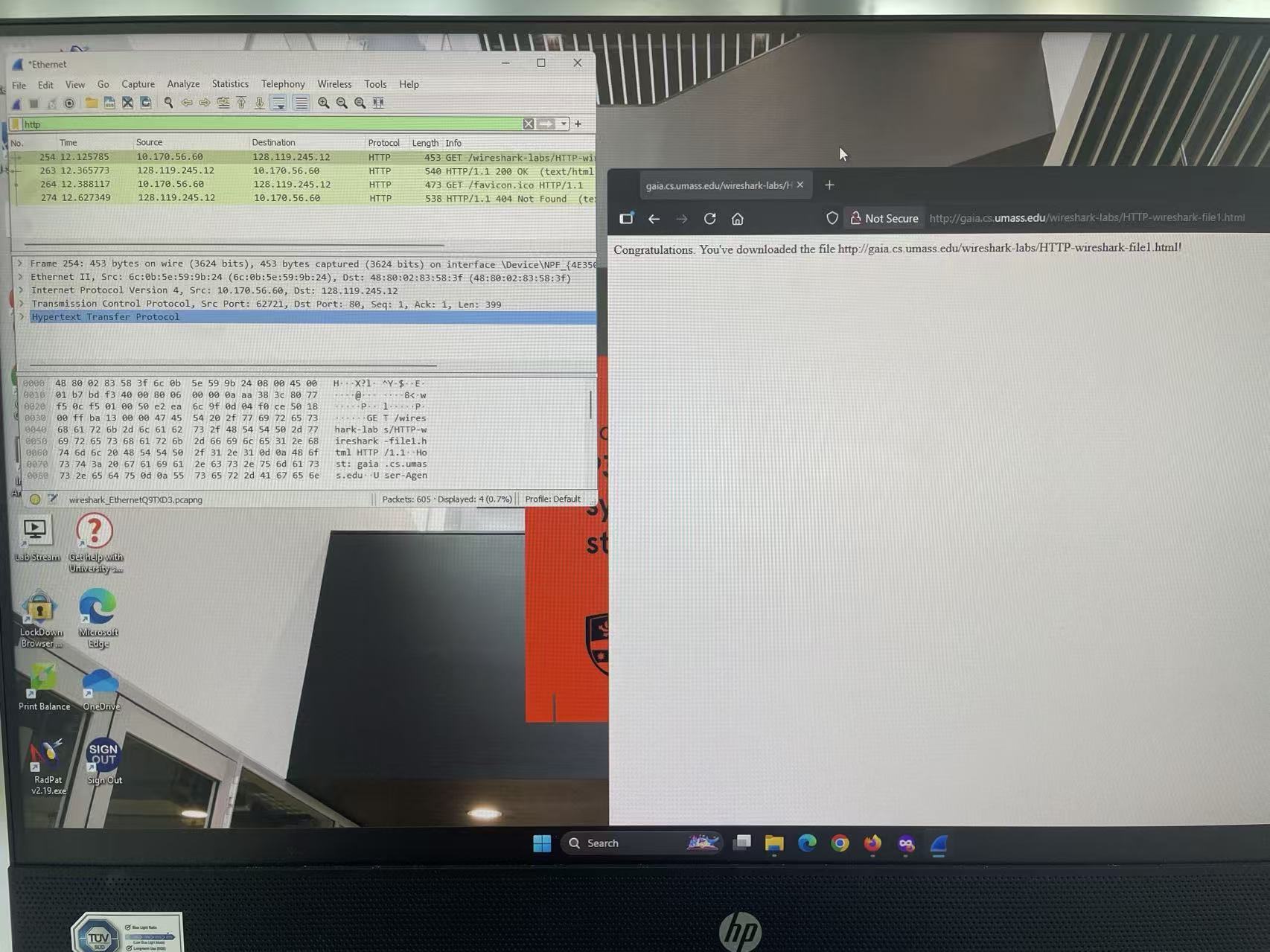
**Interpretation**

Basic Authentication merely encodes credentials; it does not encrypt them. Any packet sniffer can decode the Base64 string. This highlights why secure HTTP (HTTPS) with TLS is essential for protecting sensitive data.

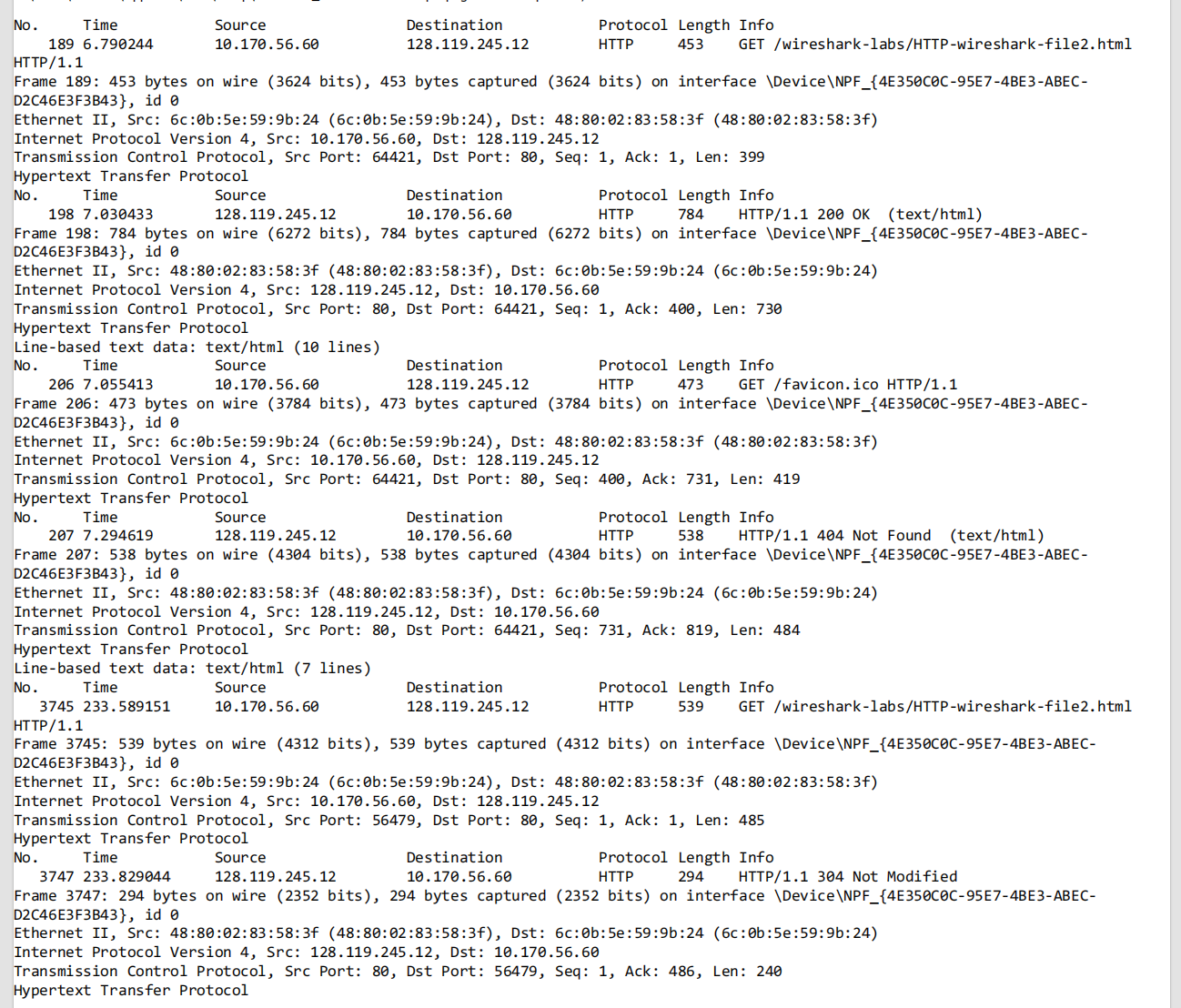
**Appendix**



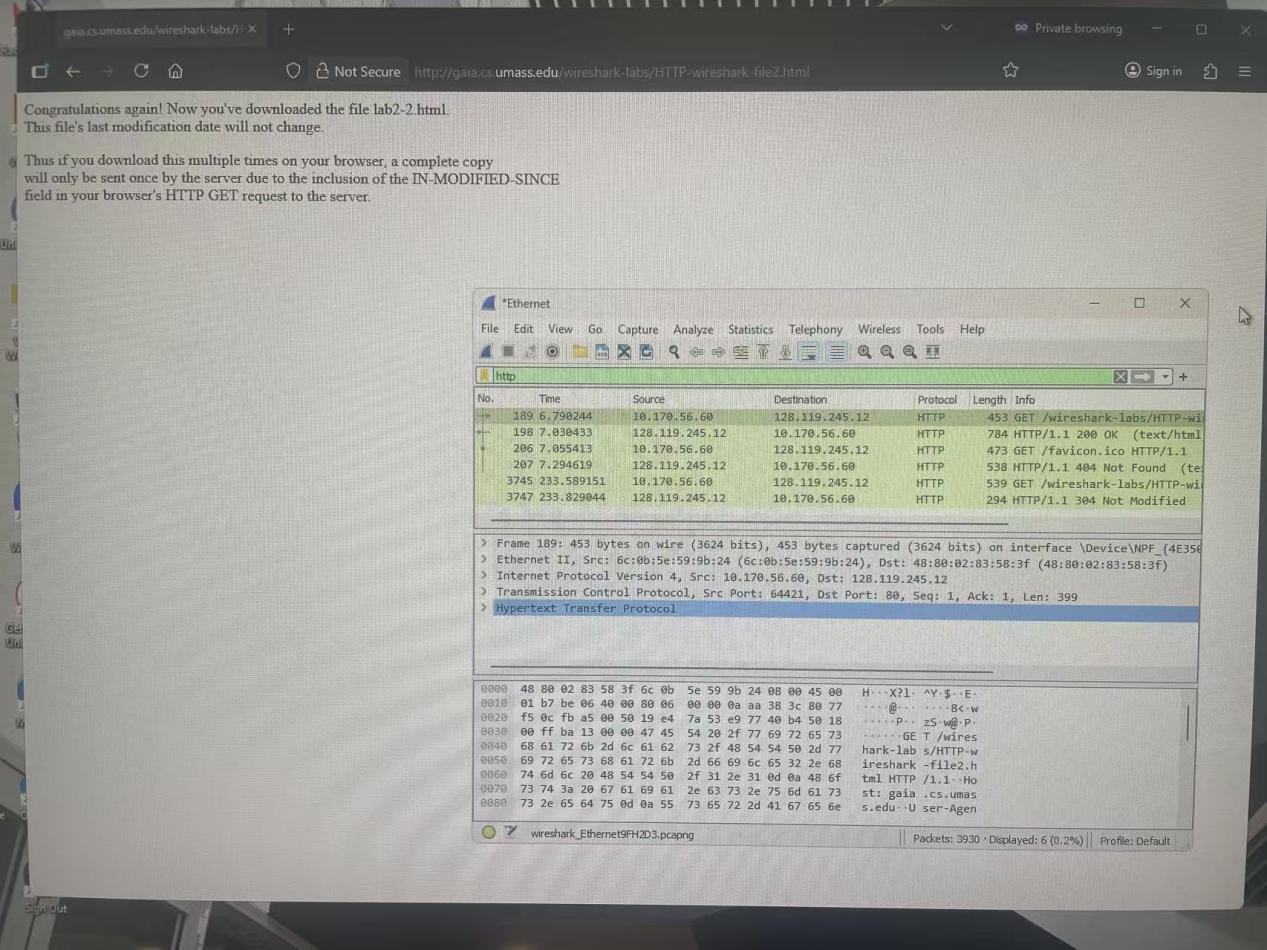
2.1 printed file



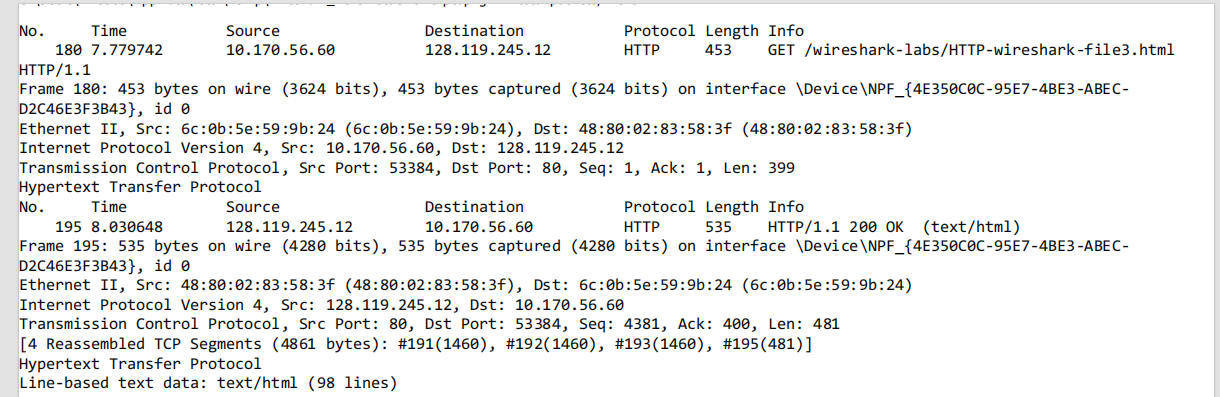
2.1 set up



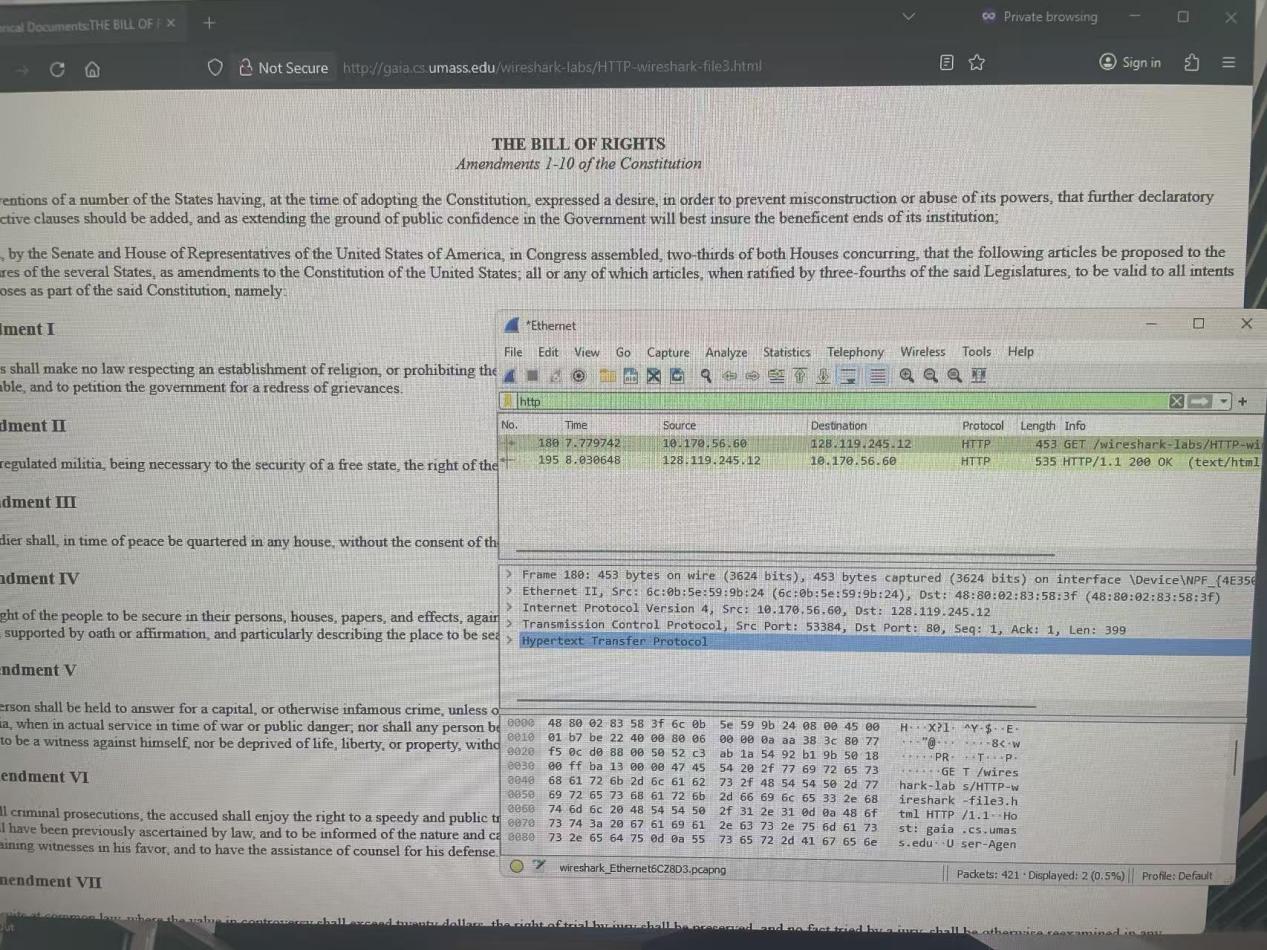
2.2 printed file



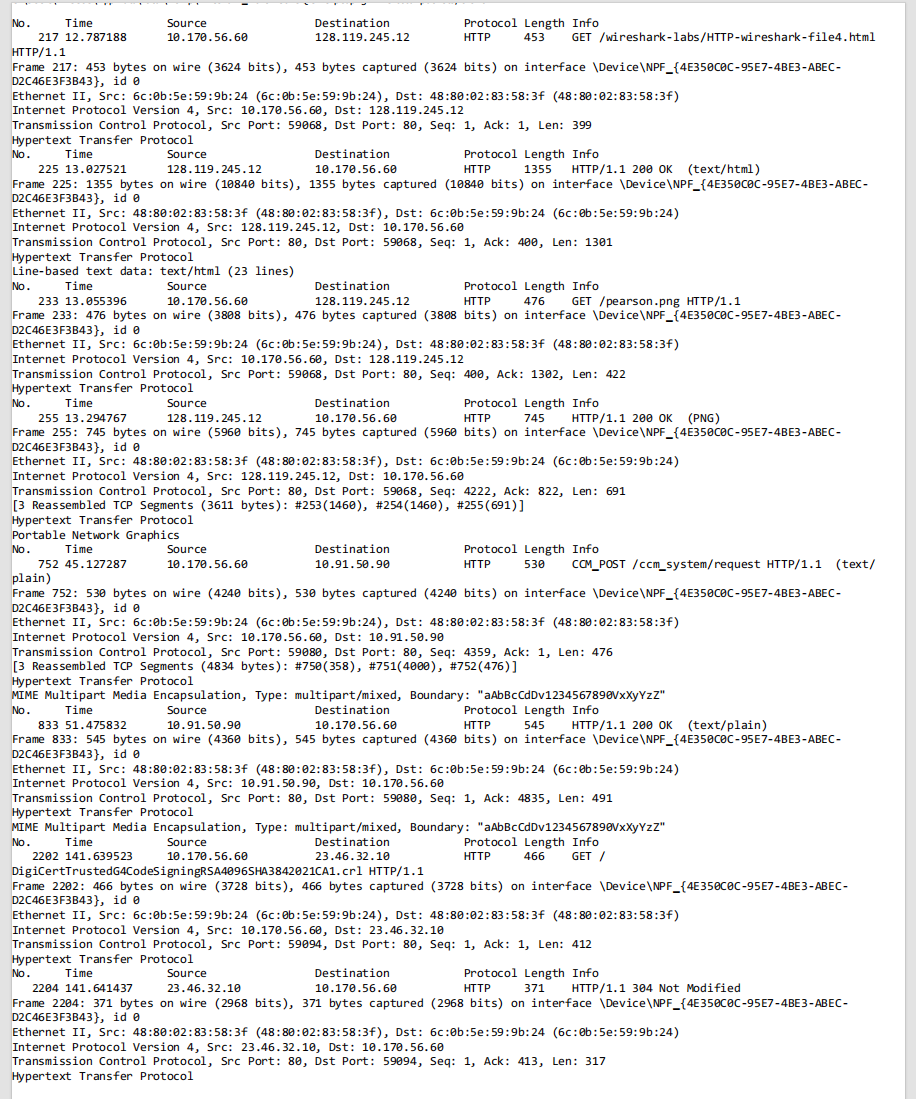
2.2 set up



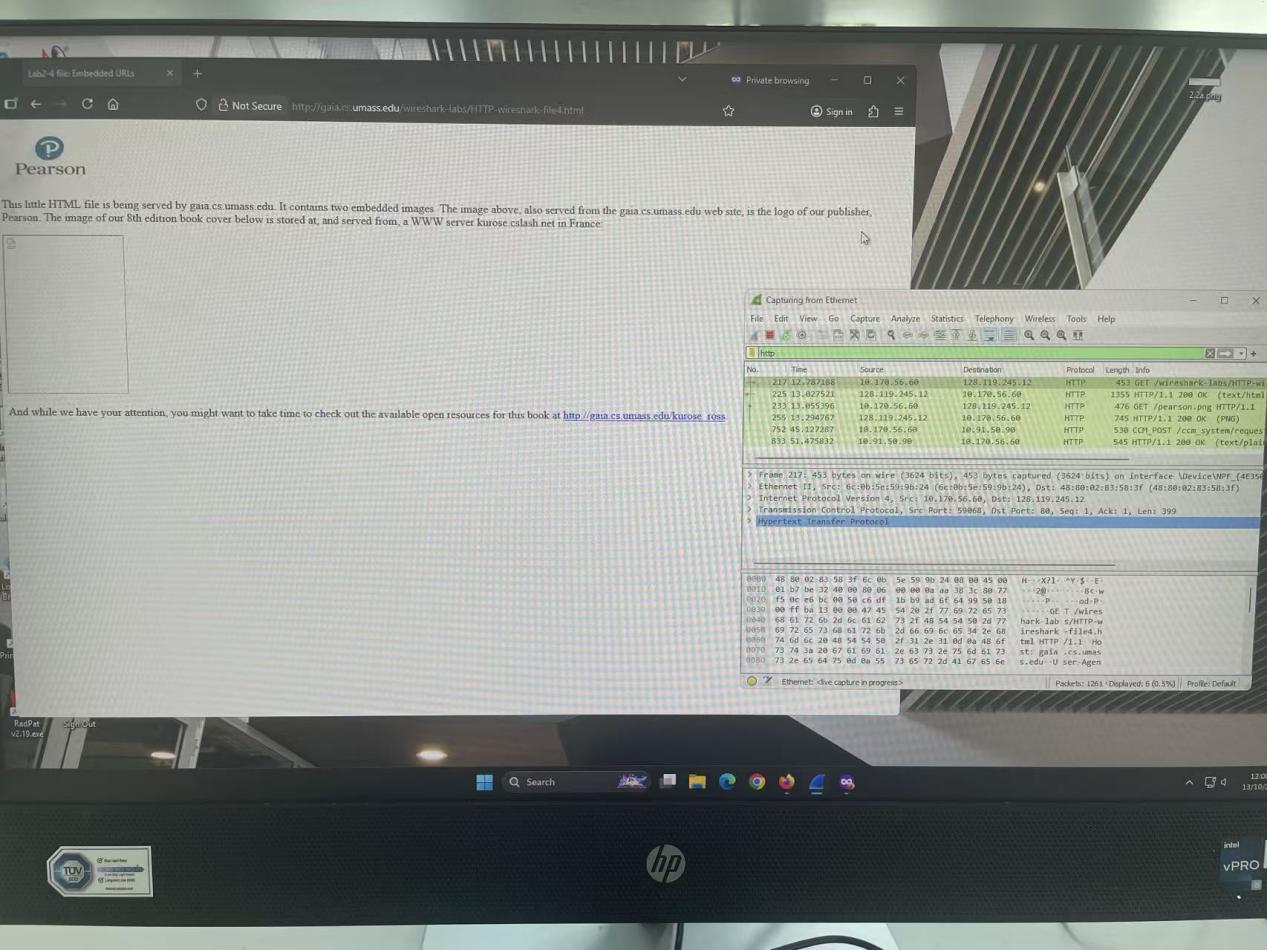
2.3 printed file



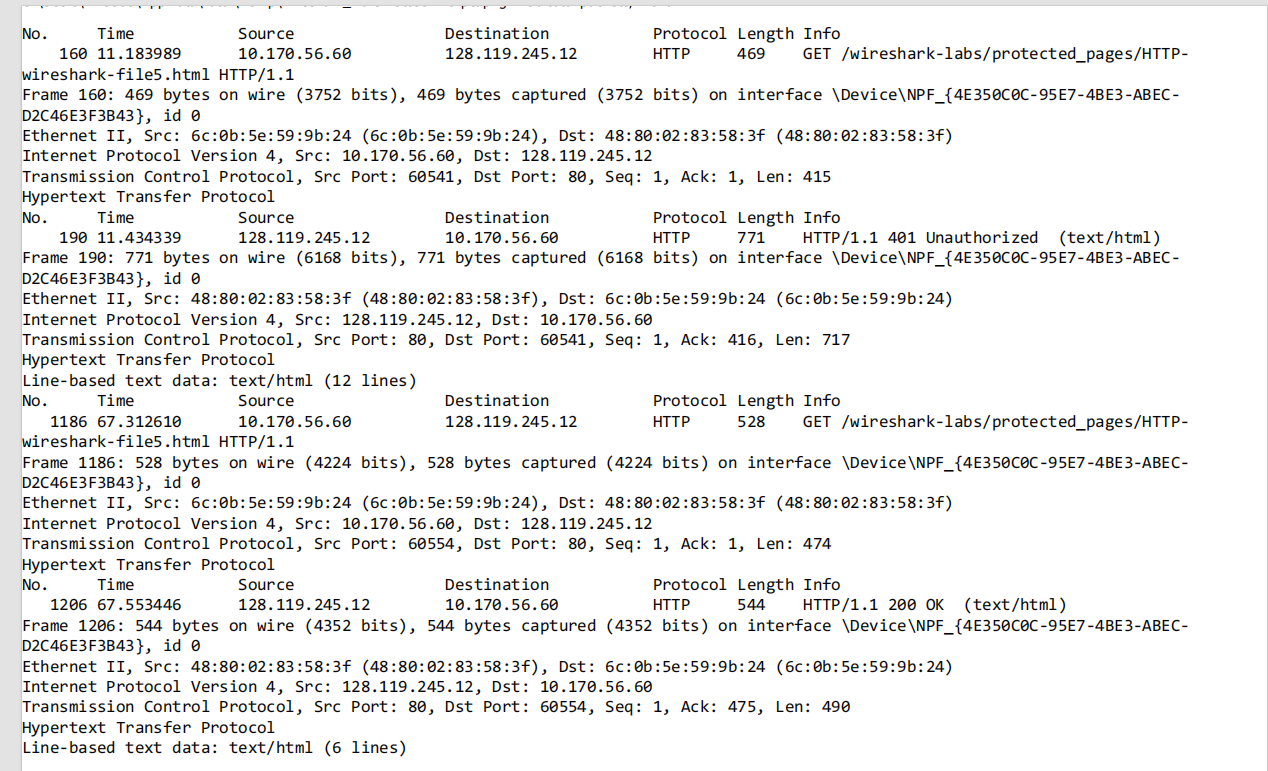
2.3 set up



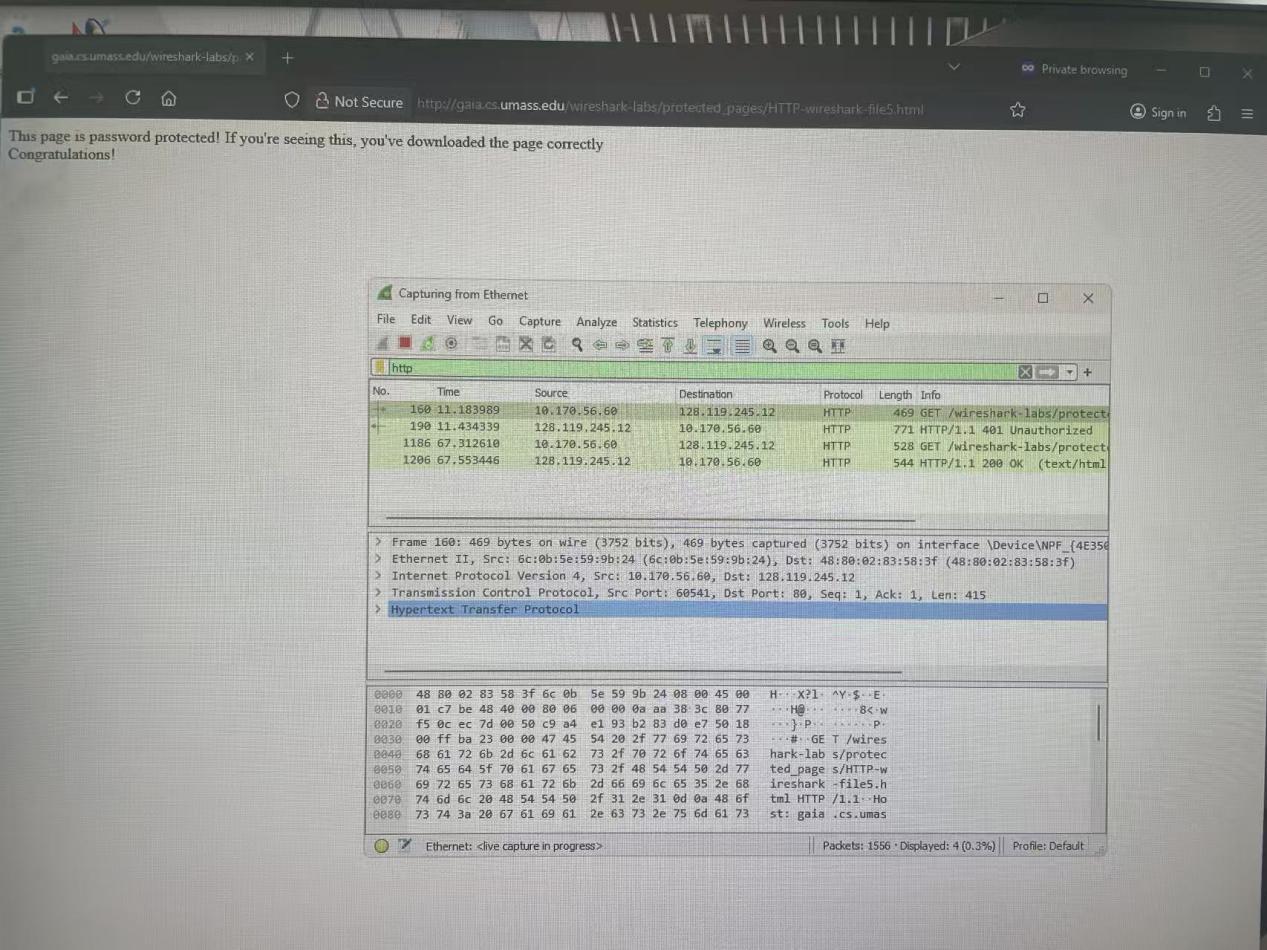
2.4 printed file



2.4 set up



2.5 printed file



2.5 set up