Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Mahitha Kalaga

Email: kalagam1@udayton.edu

Repository Information

Repository's URL: https://github.com/MahithaKalaga-cyber/waph-mahitha.git

This is a private repository for Mahitha Kalaga to store all the code from the course. The organization of this repository is as follows.

Labs

Hands-on exercises in Lectures

- Lab 0: Development Environment Setup
- Lab 1: Foundations of the Web

Report

The lab's overview

This lab focused on understanding how the web works, specifically the HTTP protocol, and creating basic web applications using CGI in C and PHP. The main outcomes included:

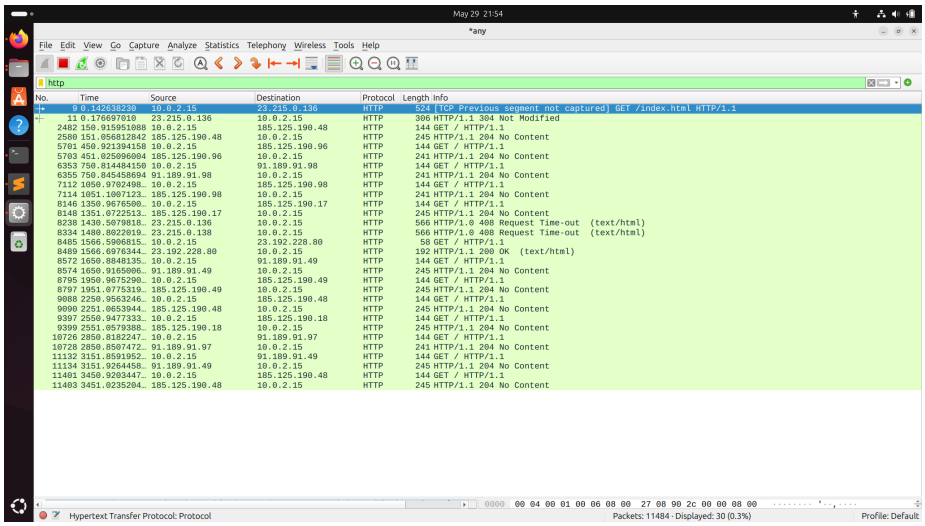
- Capturing and analyzing HTTP traffic with Wireshark.
- Sending HTTP requests using telnet.
- Writing and deploying CGI programs in C.
- Developing simple PHP applications that handle GET and POST requests.

Lab's URL: Lab1

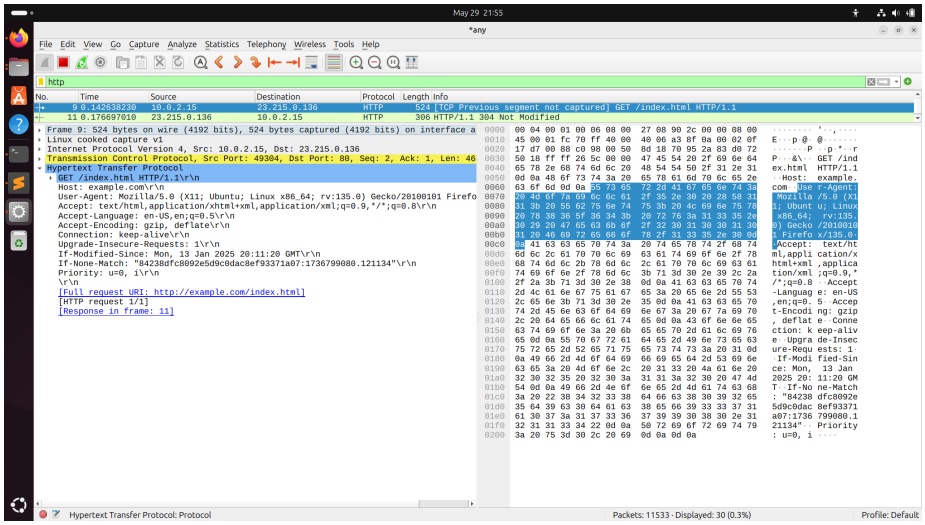
Part 1 - The Web and HTTP Protocol

Task 1: Familiar with the Wireshark Tool and HTTP Protocol

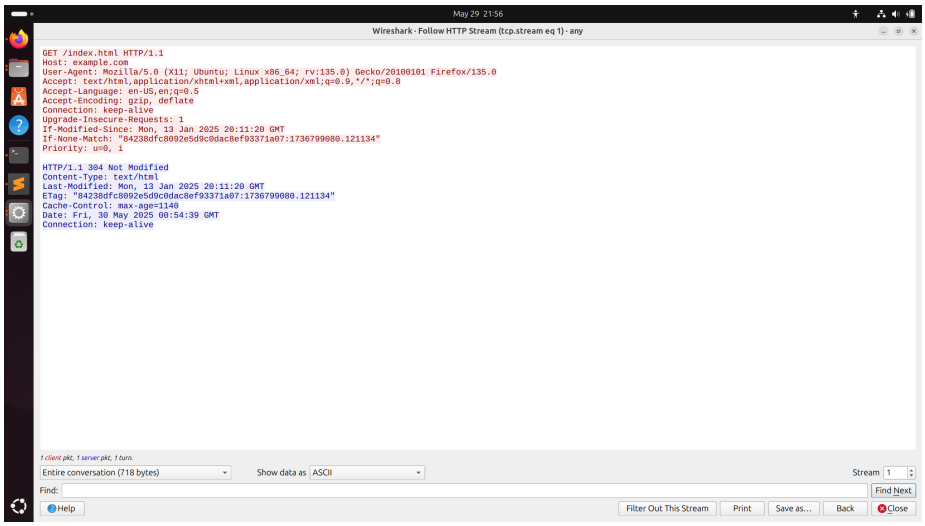
To understand how the HTTP protocol operates, I used Wireshark to capture and inspect traffic between my browser and a web server. I filtered packets using the keyword http and observed both HTTP Request and Response messages. This allowed me to identify the structure of HTTP messages, including headers, status codes, and content.



- This shows the request sent by the browser to example.com, including headers like Host, User-Agent, and Accept.



- This contains the server's response, with headers such as Content-Type, Content-Length, and the actual HTML content of the page.



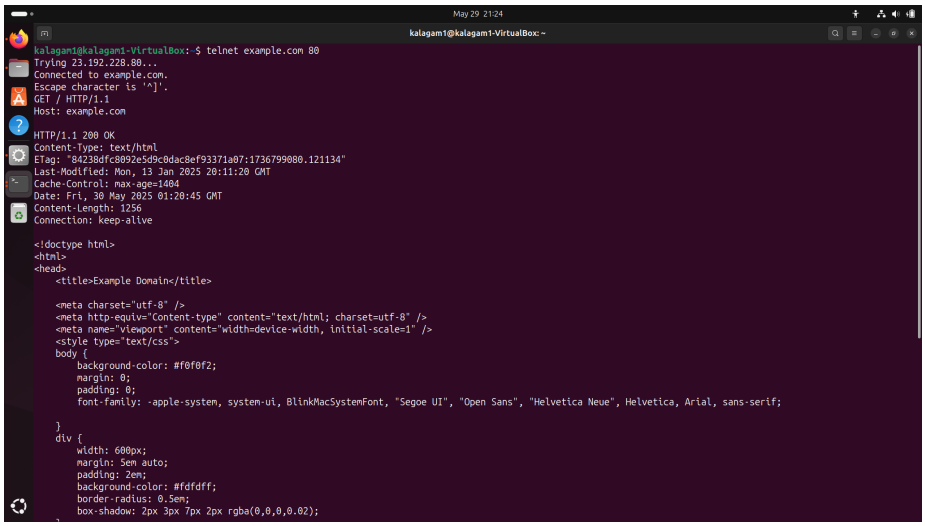
- This shows the full conversation between client and server.

Task 2: Understanding HTTP using Telnet and Wireshark

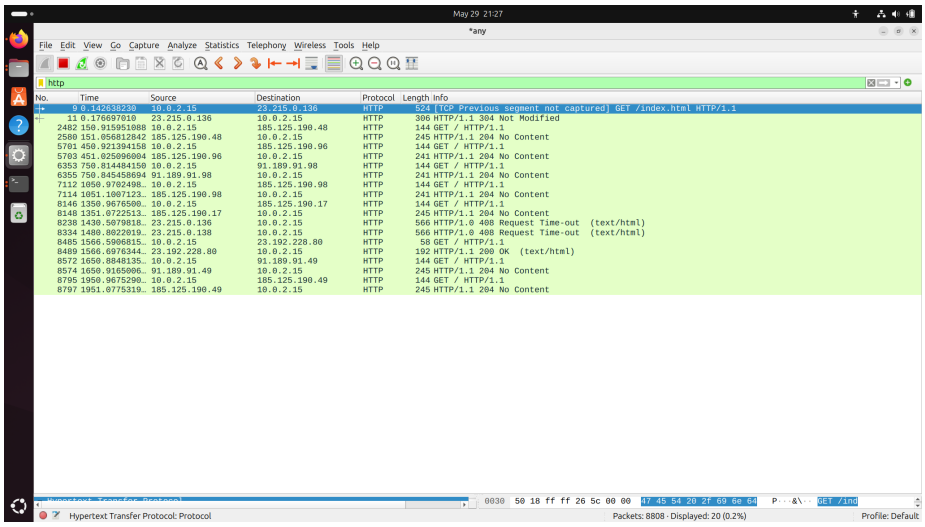
I used the telnet command to send an HTTP GET request to a server manually. Wireshark was used to capture and inspect these messages.

Summary:

- The telnet session displayed a manual HTTP request and the server's response.
- Compared to browser-generated requests, telnet requests were minimal and lacked headers like User-Agent.
- The response message lacked formatting compared to browser responses.



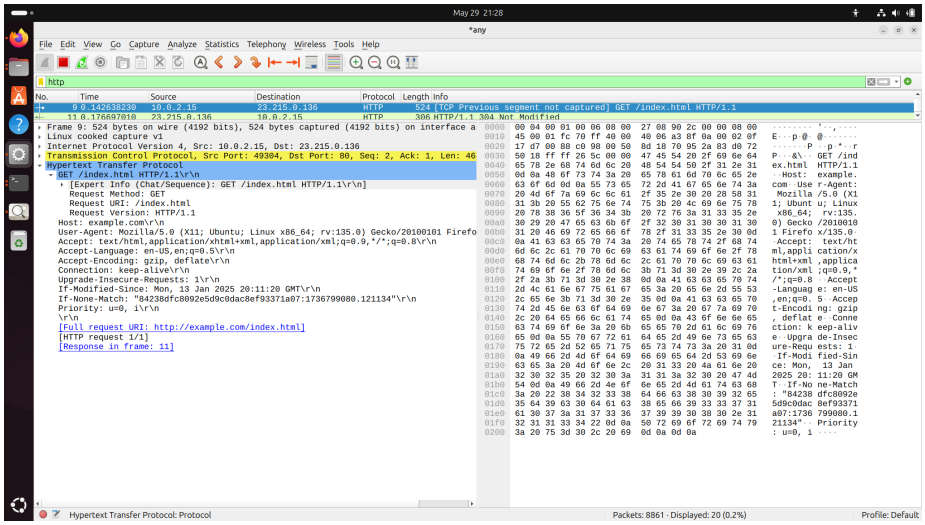
- This shows that the GET request and the HTTP request 200 OK response with the HTML content from the server.



Comparison with browser-based request:

- The telnet request is simpler and lacks headers like:
- User-Agent
- Accept
- Accept-Encoding
- Connection

It only includes GET / HTTP/1.1 and Host.



Comparison with Task 1 response:

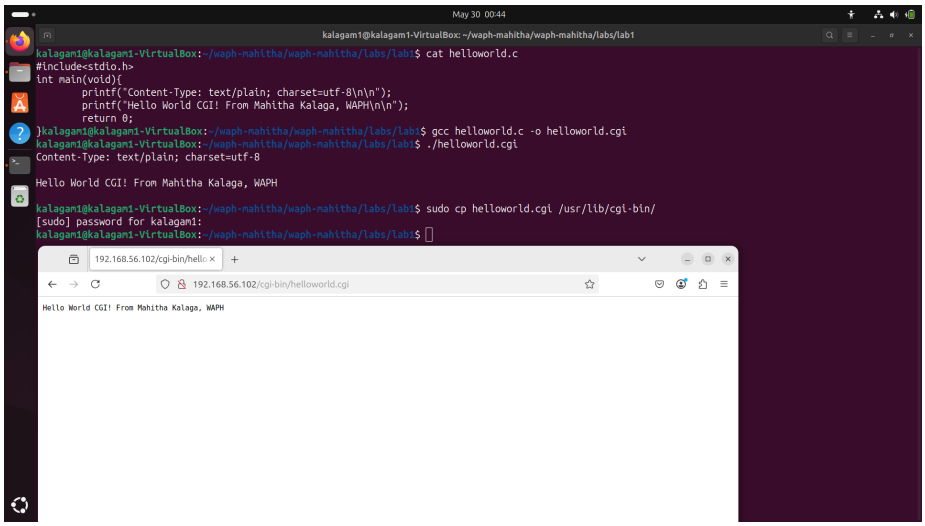
- The content is the same, but some headers like Content-Encoding, ETag, may vary depending on how the request was formed telnet vs browser.

Part 2 - Basic Web Application Programming

Task 1: CGI Web Applications in C

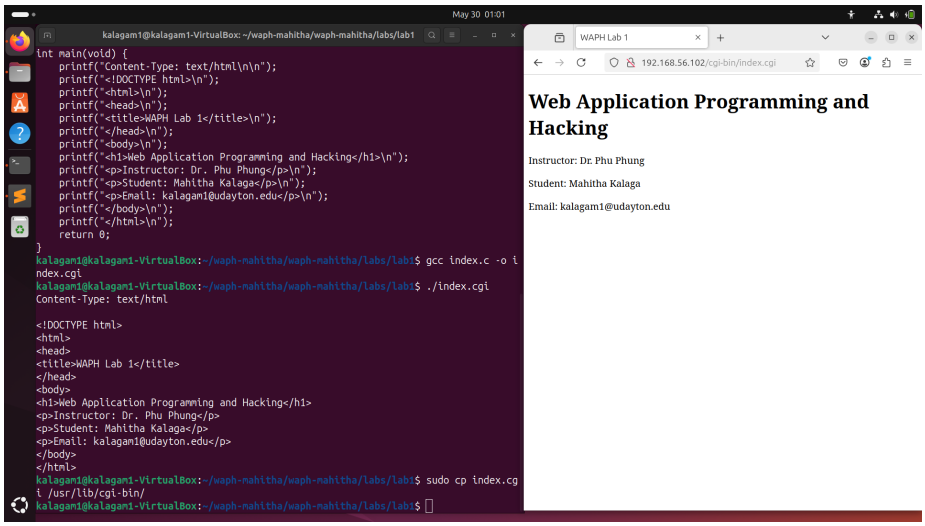
a. Hello World Program

Summary: I wrote a simple C program to demonstrate a CGI web application. The code prints a basic HTTP header and a "Hello World" message. I compiled it using gcc and placed the executable in /usr/lib/cgi-bin/, which is the default CGI directory in Apache.



b. HTML Template with CGI

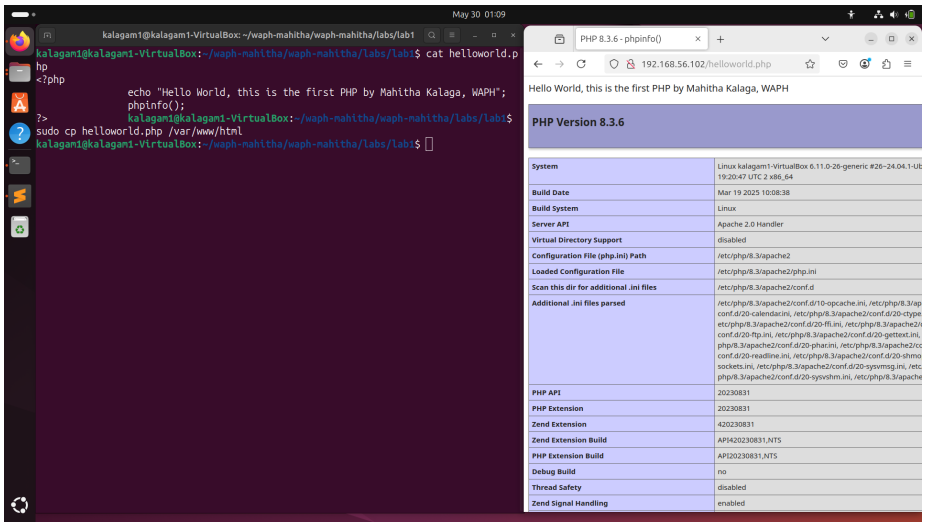
Summary: Next, I wrote another C CGI program that outputs valid HTML. I used a simple template from W3Schools, modified with my course and personal information.



Task 2: A Simple PHP Web Application with User Input

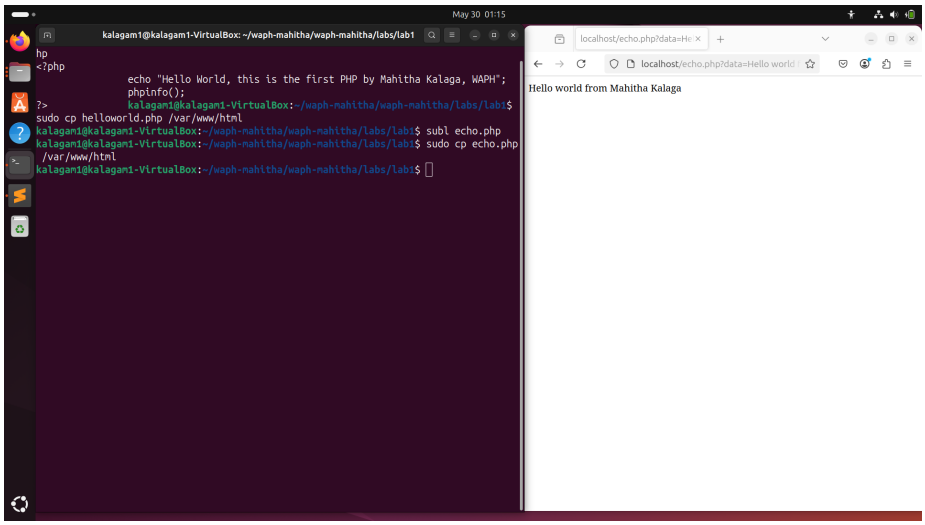
a. Hello World in PHP

Created a helloworld.php file that shows my name and PHP info using phpinfo().



b. Echo Web Application

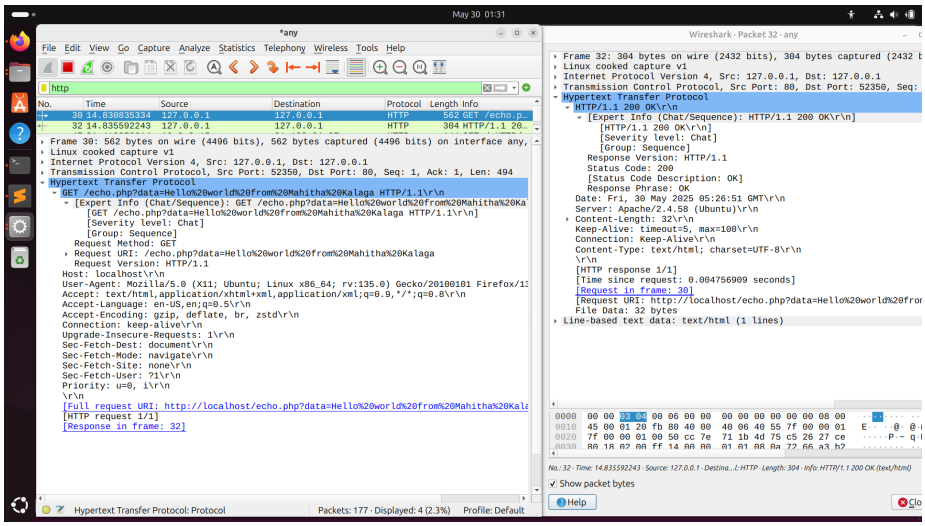
Developed echo.php that captures user input using GET and POST methods. This application echoes the submitted data.



Task 3: Understanding HTTP GET and POST Requests

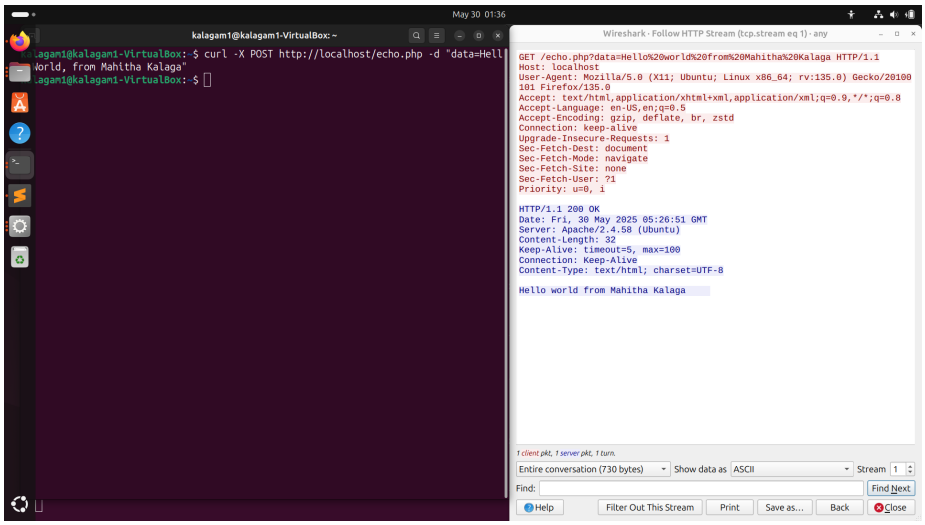
a. Wireshark GET Analysis

Used Wireshark to capture and analyze the GET request/response for echo.php.



b. curl POST Request

Used curl -d “data= Hello World, from Mahitha Kalaga” http://localhost/echo.php to send a POST request.



c. Comparison

With HTTP GET, the data (Hello World, from Mahitha Kalaga) sent are placed as query parameters into the URL, thereby making the data not only visible to the browser but also show up in the server log. Compared to this, POST requests send data in the request body, which cannot be viewed from the URL and is therefore much more secure. Additionally, GET requests are generally

used for retrieving resources and are cached, whereas POST requests are sent for data and are not cached. Whereas the GET response was shorter and faster.