

waph-mohamaz

WAPH-Web Application Programming and Hack- ing

Instructor: Dr. Phu Phung

Student

Name: Afroz Mohammad

Email: mohamaz@ucmail.uc.edu



Figure 1: Afroz's headshot

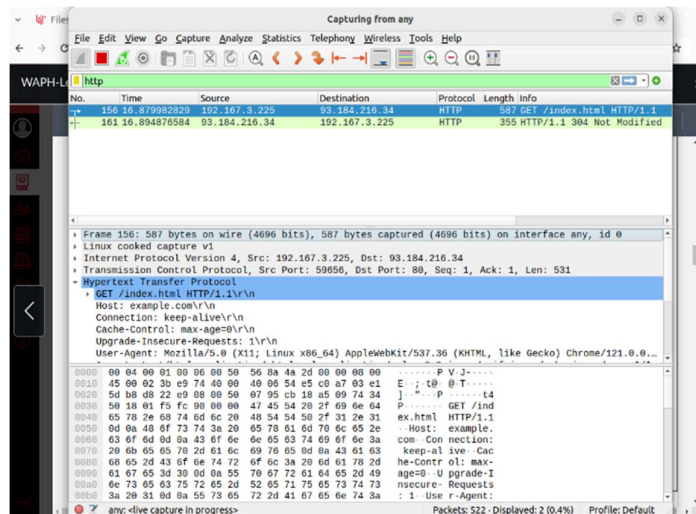
Lab 1 - Foundations of the Web

Overview: This lab provided hands-on experience with web technologies, the HTTP protocol, and basic web application programming. We used Wireshark and TELNET to examine HTTP requests and responses and compare them to what a browser sends. For web application development, we wrote CGI programs in C and incorporated HTML templates. We also covered PHP web app development. The final task was working with HTTP GET and POST requests using Wireshark and curl. We documented the lab in a Markdown report and used Pandoc to generate a PDF for submission. The key focus was gaining practical skills in HTTP, web app programming, and tools like Wireshark for network analysis. Through various programming exercises and network examinations, we gained well-rounded experience with core technologies underlying web applications and services.

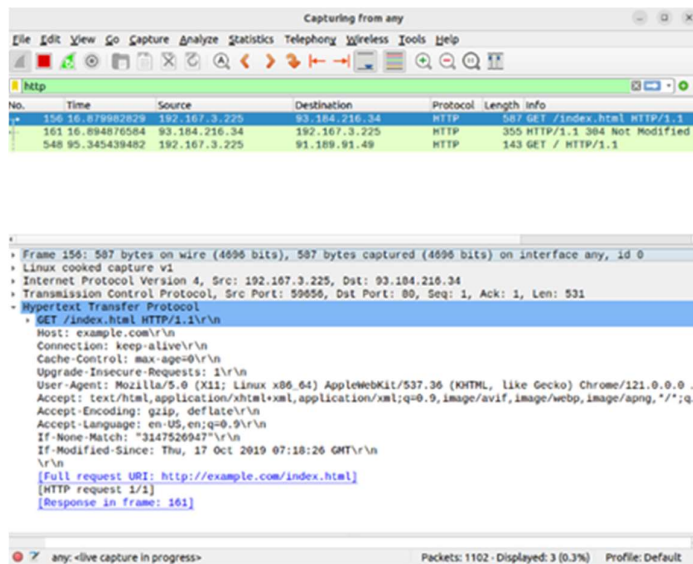
Part I: The Web and HTTP Protocol

1. Task 1 :

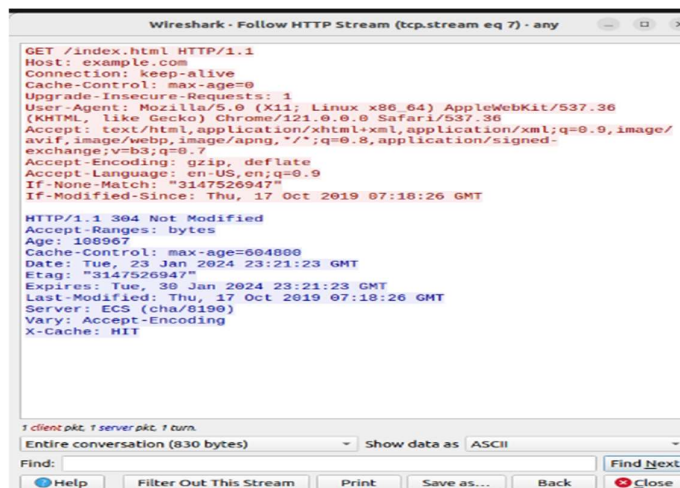
Wireshark is a network analysis and troubleshooting tool used to capture and analyze network traffic. After installing Wireshark in an Ubuntu VM, I configured it to capture packets on the network interface in promiscuous mode and filter for HTTP protocol traffic. This allowed me to isolate and inspect HTTP requests and responses within the overall network stream. By selecting individual HTTP messages, I could analyze the contents including request type, target URL, HTTP version, content type, authorization, and request data sent to the server. The HTTP responses contained status codes, status text, content type, response data, and other headers sent back to the web browser. Using Wireshark's packet capturing and protocol analysis capabilities, I was able to understand the detailed structure and contents of HTTP communications for web transactions. This hands-on experience helped demystify application layer protocols and how they are encoded within network packets.



[1.1] here it is the http request



[1.2] here it is the http response



[1.3] here it is the http stream

2. Task 2:

I used Wireshark to capture network traffic while making an HTTP request to `example.com/index.html` through Telnet in the terminal. To use Telnet, I first established a connection to the `example.com` web server using the syntax `'telnet example.com portNumber'`. After connecting, I manually constructed the HTTP request by typing the request type, file path, HTTP version, and host name, before hitting enter twice to send the request and receive the response.

Comparing the Wireshark capture of the Telnet HTTP request to a browser HTTP request to the same server revealed some differences. The Telnet request lacked certain request headers like user-agent, accept, accept-language etc. that a browser automatically populates. With Telnet, I had to manually construct the entire HTTP request, whereas the browser automatically generates certain headers. However, the HTTP responses captured in Wireshark were identical between the browser and Telnet.

```

Jan 23 18:39
Administrator@omph-vmc -
[+]
Administrator@omph-vmc ~$ telnet example.com 80
Trying 93.184.216.34...
Connected to example.com.
Escape character is '^['.

GET /index.html HTTP/1.0
Host: example.com

HTTP/1.0 200 OK
Age: 135543
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 23 Jan 2024 23:38:02 GMT
Etag: "318726047adent"
Expires: Tue, 30 Jan 2024 23:38:02 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: GCS (cha/18190)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
Connection: close

<!doctype html>
<html>
<head>
<title>example Domain</title>

<meta charset="utf-8" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style type="text/css">
body {
background-color: #f0f0f2;
margin: 0;
padding: 0;
font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
}
div {

```

[2.1] The above snapshot is the result after executing the commands that are provided in the above description

Part II: Basic Web Application Programming

1. Task 1 :

Inorder to develop a CGI C program and deploy in web server we have edit it in sublime by using "\$ subl helloworld.c" command later type the code in that. and run the code by using gcc helloworlds.c -ohelloworld.cgi" and" ./helloworld.cgi" commands. Initially, CGI daemon not enabled in Apache so to enable it

a2enmod cgid and \$ sudo systemctl restart apache2" commands later, they are stored in the one separate folder. Now, To deploy the C code into web server we need to type "localhost/cgi-bin/helloworld.cgi" then we can see output there.

The index.c file is mentioned here- `#include <stdio.h> int main() { printf("`

“);

```
printf("
```

```
"); printf(" Course: WAPH< "); printf("
```

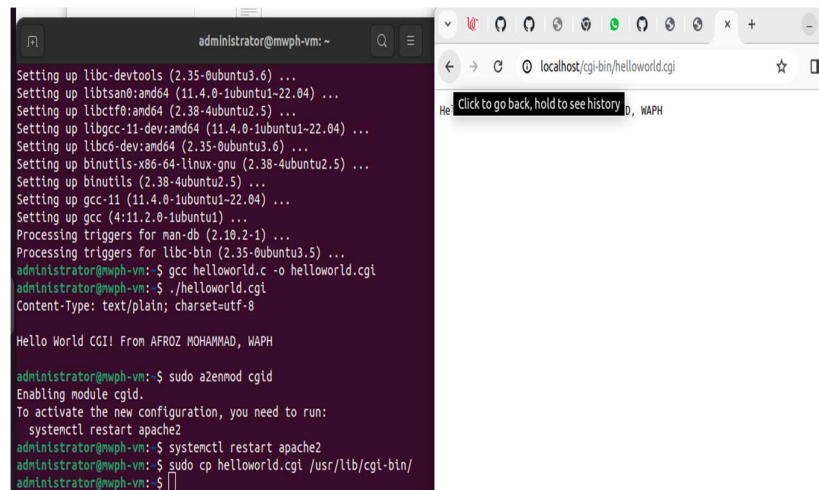
```
"); printf("
```

```
"); printf("<h1Name: Sai Abhishek Avusula</h1"); printf("<pDepartment:Information Technology
```

```
"); printf("
```

```
"); printf("
```

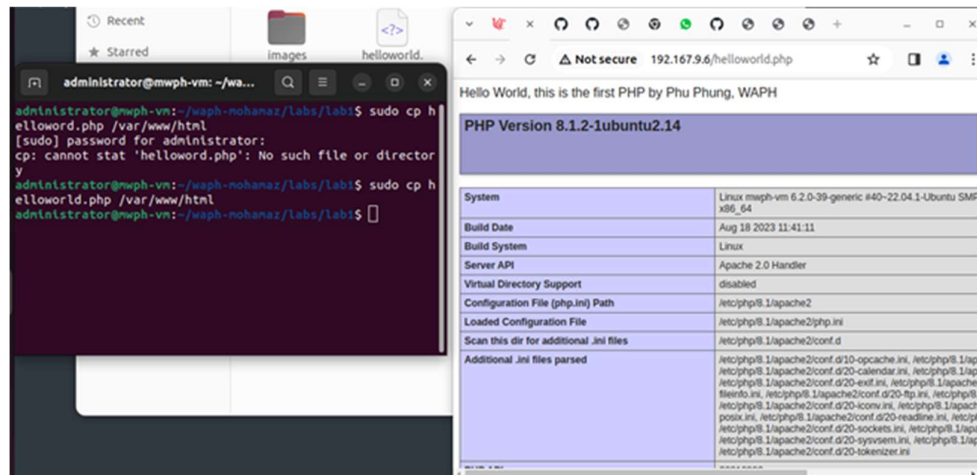
```
"); return 0; }"
```



```
administrator@mwph-vm: ~  
Setting up libc-devtools (2.35-0ubuntu3.6) ...  
Setting up libtsan0:amd64 (11.4.0-1ubuntu1-22.04) ...  
Setting up libctf0:amd64 (2.38-4ubuntu2.5) ...  
Setting up libgcc-11-dev:amd64 (11.4.0-1ubuntu1-22.04) ...  
Setting up libc6-dev:amd64 (2.35-0ubuntu3.6) ...  
Setting up binutils-x86-64-linux-gnu (2.38-4ubuntu2.5) ...  
Setting up binutils (2.38-4ubuntu2.5) ...  
Setting up gcc-11 (11.4.0-1ubuntu1-22.04) ...  
Setting up gcc (4:11.2.0-1ubuntu1) ...  
Processing triggers for man-db (2.10.2-1) ...  
Processing triggers for libc-bin (2.35-0ubuntu3.5) ...  
administrator@mwph-vm: $ gcc helloworld.c -o helloworld.cgi  
administrator@mwph-vm: $ ./helloworld.cgi  
Content-Type: text/plain; charset=utf-8  
  
Hello World CGI! From AFROZ MOHAMMAD, WAPH  
  
administrator@mwph-vm: $ sudo a2enmod cgi  
Enabling module cgi.  
To activate the new configuration, you need to run:  
systemctl restart apache2  
administrator@mwph-vm: $ systemctl restart apache2  
administrator@mwph-vm: $ sudo cp helloworld.cgi /usr/lib/cgi-bin/  
administrator@mwph-vm: $
```

2. Task 2:

- a. Using Sublime Text, I created the basic helloworld.php file and uploaded it to the web server. To accomplish that, I need to type the following command into the terminal: "\$ sudo cp helloworld.php /var/www/html." Afterwards, open the "helloworld.php" file located on the localhost in any browser. I took a screenshot below



Using Sublime Text, I created the basic helloworld.php file and uploaded it to the web server. To accomplish that, I need to type the following command into the terminal: "\$ sudo cp helloworld.php /var/www/html." Afterwards, open the "helloworld.php" file located on the localhost in any browser. I've included a screenshot and the source code below

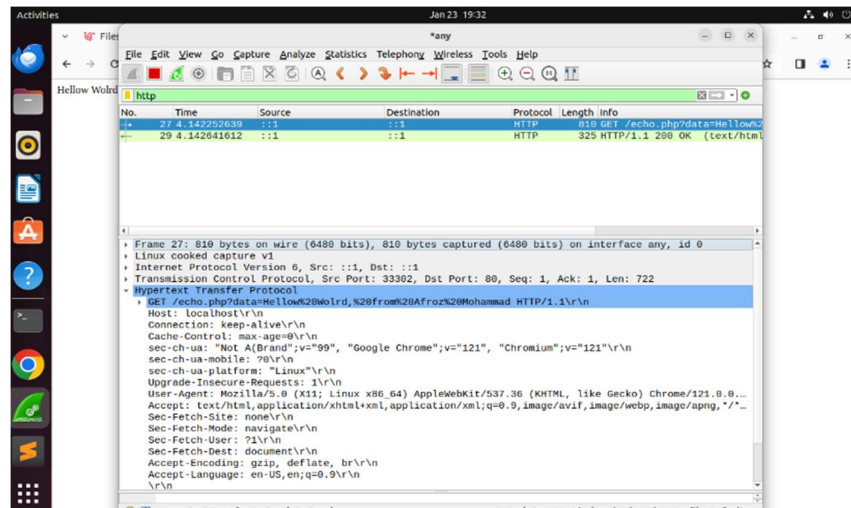


Source code:

```
echo $_REQUEST["data"]; <?> Problem with REQUEST METHOD: There is a risk with this code as it is easily attackable in the address bar.
```

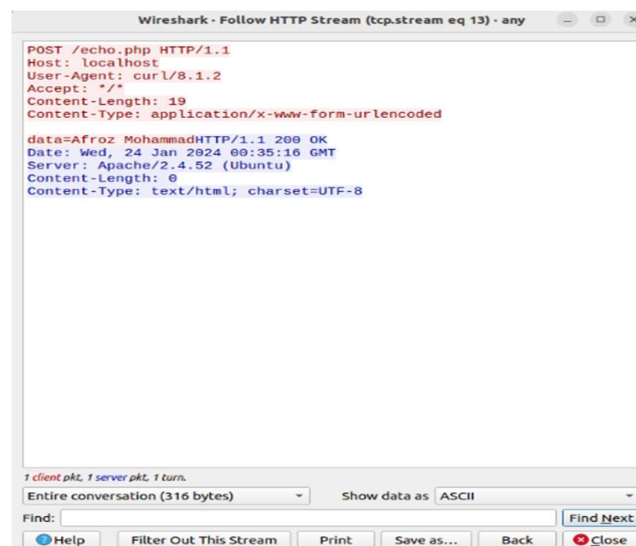
3. Task 3:

I've examined the "echo.php" program's get request and get response using Wireshark. Before running the server, I selected any network option in Wireshark for that purpose. Later, I was able to obtain the request and response from Wireshark by using the http filter. I've retained the screen grabs down below.



Initially I have installed the Curl application in Ubuntu. by using "sudo apt install curl" the i have used a command which I already mentioned in the command prompt screenshot below here before starting the Wireshark.

Later, by right click on the http request which is followed by the Http stream. where you can see the http request and response of the application by using curl clearly. I have kept screenshot below here.



HTTP POST Request and Response are similar in that they both adhere to the cycle of requests and responses and permit the addition of headers to requests in order to offer more information. But Let's Talk About the Difference. There are plenty of them, like this :

» GET is for smaller capacity compared with POST." "We can see that it is visible to users in the browser address bar, whereas POST method is not visible in the browser address bar."

» The POST method is used for data submission that is sent in the request body, whereas the GET method contains data that was used for retrieval where parameters like name are provided in the URL.

The PDF file named as mohamaz-waph-lab1.pdf