

WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student

Name: Sai Sandeep Pasham

Email: pashamsp@mail.uc.edu



Figure 1: Sai Sandeep Pasham

Lab 1 - Foundations of the WEB

Overview: This lab focuses on the fundamental elements of the web, with a primary emphasis on the HTTP protocol. It involves getting familiar with the Wireshark tool, which is employed for in-depth examination of HTTP requests and responses. Furthermore, the understanding of HTTP requests and responses is enhanced through the exploration of Telnet. The practical aspect of the lab involves web programming using CGI in C, along with the development of a basic web application that incorporates user input using PHP. A pivotal aspect of the learning process is the comprehension of HTTP GET and POST requests, contributing to a holistic understanding of web technologies in a hands-on manner.

<https://github.com/pashamsp/waph-pashamsp/blob/main/labs/lab1/README.MD>.

Part 1 : The WEB and the HTTP Protocol

Task 1. Familiar with the Wireshark tool and HTTP protocol

Wireshark is a popular open-source network analyzer that captures and displays network data in real time. It is especially useful for troubleshooting network problems, analysing network protocols, and maintaining network security. First, I installed wireshark in an Ubuntu VM, then I ran it, picked ANY, clicked on the

capture icon on the input interface, and enabled the promiscuous mode on filters checkbox. The packets were then recorded and filtered to include the HTTP protocol among all others. The HTTP requests and answers are then selected for analysis and observation. The HTTP request specifies the type of request, the target URL, the HTTP version, the content type, authorization, and the data provided to the web servers. The HTTP response includes information such as the status code, status text, content type, and data returned to the web browser.

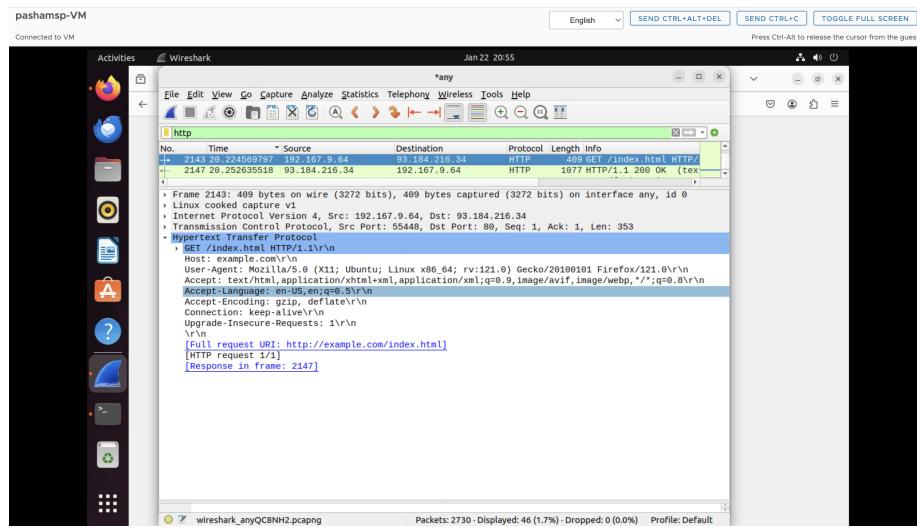


Figure 2: Wireshark HTTP Request

Task 2. Understanding HTTP using telnet and Wireshark

I started Wireshark and began capturing network packets preceding an HTTP request to `example.com/index.html` using TELNET in the terminal. To use TELNET, create an initial connection with the `example.com` web server using the syntax “`telnet example.com portNumber`.” Once connected, the request type, path file, HTTP version, and host name were required to make the HTTP request. After making the request, hit Enter twice to receive the response.

When Wireshark was used to compare HTTP requests made through the browser and TELNET, it revealed a significant difference: the server fields were missing from the TELNET-generated request. It is critical to note that the TELNET HTTP request is manually created, as opposed to the browser-sent request, which automatically populates request headers such as user-agent, accept, accept-language, authorization, encoding, and content.

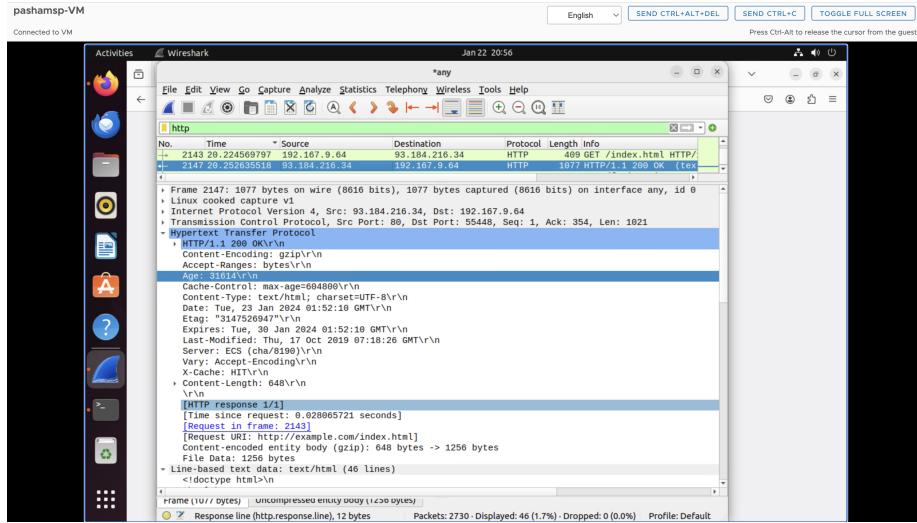


Figure 3: Wireshark HTTP Response

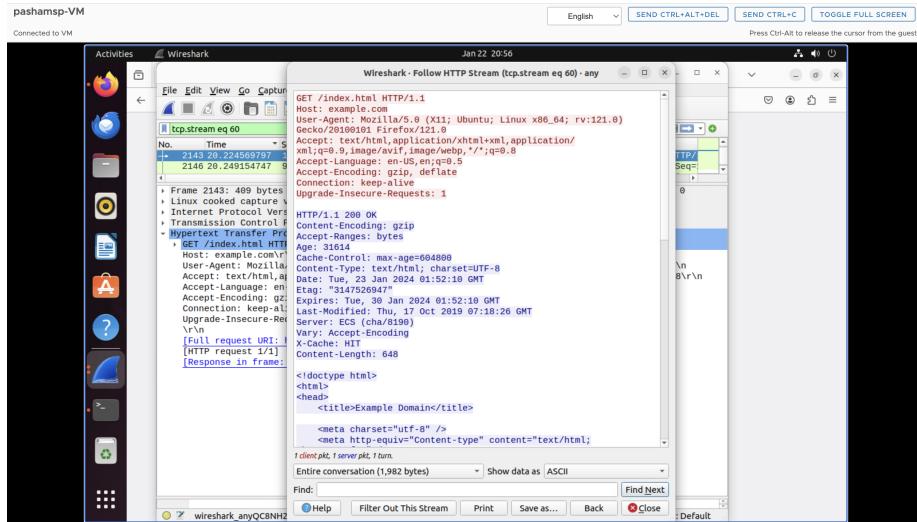


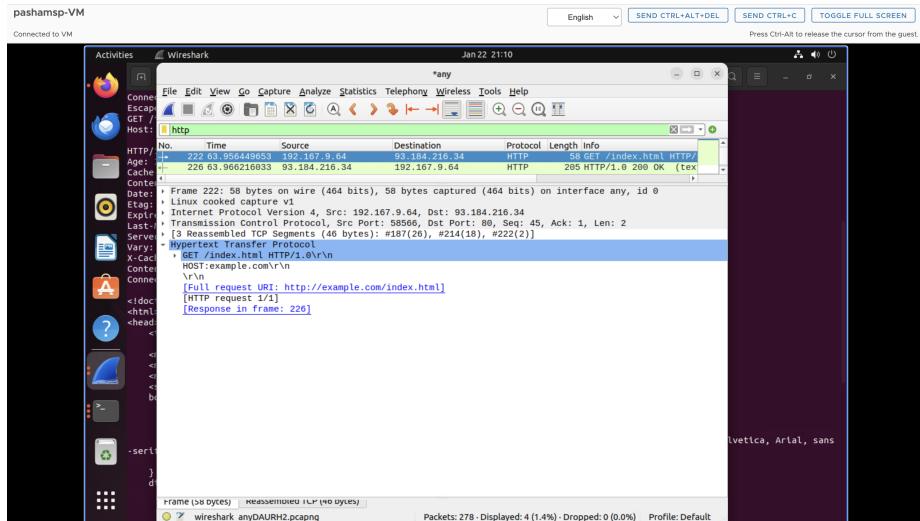
Figure 4: Wireshark HTTP Stream

```

Connected to VM
pashamsp-VM
Activities Terminal Jan 22 21:06
administrator@mwpv-vm: ~
** (wreshark:2336) 20:52:30.744987 [Capture MESSAGE] -- Capture stopped.
~c
administrator@mwpv-vm: $ telnet example.com 80
Trying 93.184.216.34...
Connected to example.com.
Escape character: '^A'.
GET /index.html HTTP/1.0
Host : example.comConnection closed by foreign host.
administrator@mwpv-vm: $ Host: example.com
Hosts file not found.
administrator@mwpv-vm: $ telnet example.com 80
Trying 93.184.216.34...
Connected to example.com.
Escape character: '^A'.
GET /index.html HTTP/1.0
Host: example.com
HTTP/1.0 200 OK
Age: 32391
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Thu, 30 Jan 2024 02:05:07 GMT
Etag: "314752694714997"
Expires: Tue, 30 Jan 2024 02:05:07 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (chab9190)
Content-Encoding: gzip
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">

```

Figure 5: Telnet request



both the HTTP responses in wireshark through browser and TELNET were same.

Part II - Basic Web Application Programming

Task 1: CGI Web applications in C

A.I created a simple CGI programme that simply prints. Hello, world! I compiled this with gcc and deployed the resultant cgi file by copying it to `/usr/lib/cgi-bin`

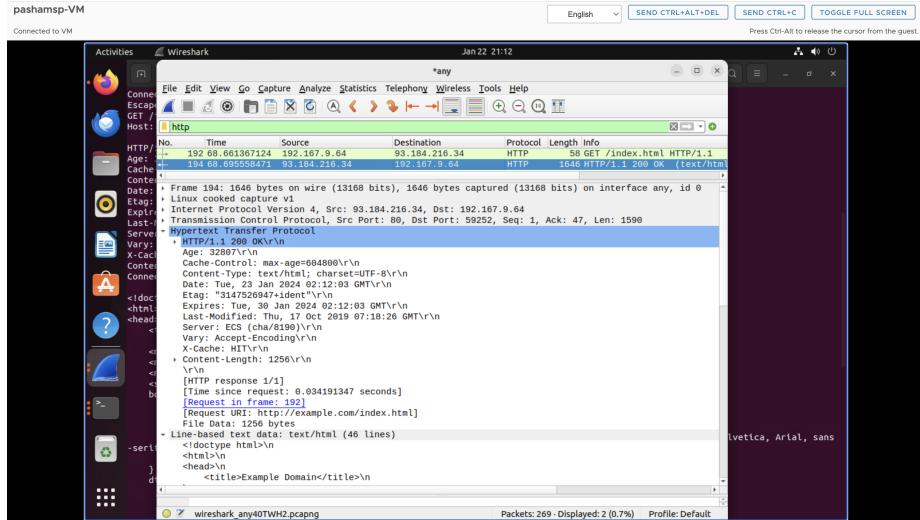


Figure 6: Telent response in wireshark

before accessing it via localhost/cgi-bin/helloWorld.cgi in the browser.

B. I designed another CGI programme that incorporates a simple HTML template directly into the C code. The template has the course name as the title, the student's name as the heading, and additional information within each paragraph. Following development, the file was compiled with the gcc compiler and copied to the usr/lib/cgi-bin directory before being viewed via the browser.

Included file `helloworld.c`:

```
#include<stdio.h>
int main(void) {
    const char *html = "<!DOCTYPE html> <html> <head> <title>Web Application Programming and</title> </head> <body> <h1>Student: Sai Sandeep Pasham</h1></body> </html>";
    printf("Content-Type: text/html\n\n");
    printf("%s", html);
    return 0;
}
```

Task 2: A simple PHP Web Application with user input.

A. As component of this assignment, I built a PHP web application using basic syntax, including information such as my name and PHP version. The application was deployed to Apache2's root directory (var/www/html). The IP address was then followed by "/helloworld.php" in the URL.

The screenshot shows a Linux desktop environment with a terminal window and a web browser. The terminal window is titled "administrator@mwpf-vm: /usr/lib/cgi-bin" and displays the following command-line session:

```

Hello World CGI! From Sai Sandeep Pasham, WAPH
administrator@mwpf-vm: /usr/lib/cgi-bin$ cd ~
administrator@mwpf-vm: $ cd waph-pashamspl/labs/lab1
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ sudo cp helloWorld.cgi /usr/lib/cgi-bin/
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ cd /usr/lib/cgi-bin/
administrator@mwpf-vm: /usr/lib/cgi-bin$ ls
helloWorld.cgi
administrator@mwpf-vm: /usr/lib/cgi-bin$ suddddd
Your MPN seems to be threaded. Selecting cgid instead of cgi.
Enabling module cgid
To activate the new configuration, you need to run:
    systemctl restart apache2
administrator@mwpf-vm: /usr/lib/cgi-bin$ systemctl restart apache2
administrator@mwpf-vm: /usr/lib/cgi-bin$ ./helloWorld.cgi
Content-Type: text/plain; charset=utf-8

Hello World CGI! From Sal Sandeep Pasham, WAPH
administrator@mwpf-vm: /usr/lib/cgi-bin$ 

```

The web browser window shows the URL "localhost/cgi-bin/helloWorld.cgi" and displays the output "Hello World CGI! From Sal Sandeep Pasham, WAPH".

Figure 7: CGI program in C

The screenshot shows a Linux desktop environment with a terminal window and a web browser. The terminal window is titled "administrator@mwpf-vm: /usr/lib/cgi-bin" and displays the following command-line session:

```

Student: Sai Sandeep Pasham
This exercise is done for Lab 1 , CGI Web Applications with C.

administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ cd ~
administrator@mwpf-vm: $ cd waph-pashamspl/labs/lab1
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ ls
Index.c
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ gcc index.c -o index.cgi
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ sudo cp index.cgi /usr/lib/cgi-bin/
[sudo] password for administrator:
administrator@mwpf-vm: /waph-pashamspl/labs/lab1$ cd /usr/lib/cgi-bin/
administrator@mwpf-vm: /usr/lib/cgi-bin$ ls
helloWorld.cgi
administrator@mwpf-vm: /usr/lib/cgi-bin$ sudo azenmod cgi
Your MPN seems to be threaded. Selecting cgid instead of cgi.
Module cgid already enabled
administrator@mwpf-vm: /usr/lib/cgi-bin$ systemctl restart apache2
administrator@mwpf-vm: /usr/lib/cgi-bin$ ./index.cgi
Content-Type: text/html

<!DOCTYPE html> <html> <head> <title>Web Application Programming and Hacking</title></head> <body> <hi>Student: Sal Sandeep Pasham</hi><p>This exercise is done for Lab 1 , CGI Web Applications with C.</p></body></html>
administrator@mwpf-vm: /usr/lib/cgi-bin$ 

```

The web browser window shows the URL "localhost/cgi-bin/index.cgi" and displays the output "Student: Sal Sandeep Pasham" followed by the message "This exercise is done for Lab 1 , CGI Web Applications with C.".

Figure 8: CGI in C and HTML

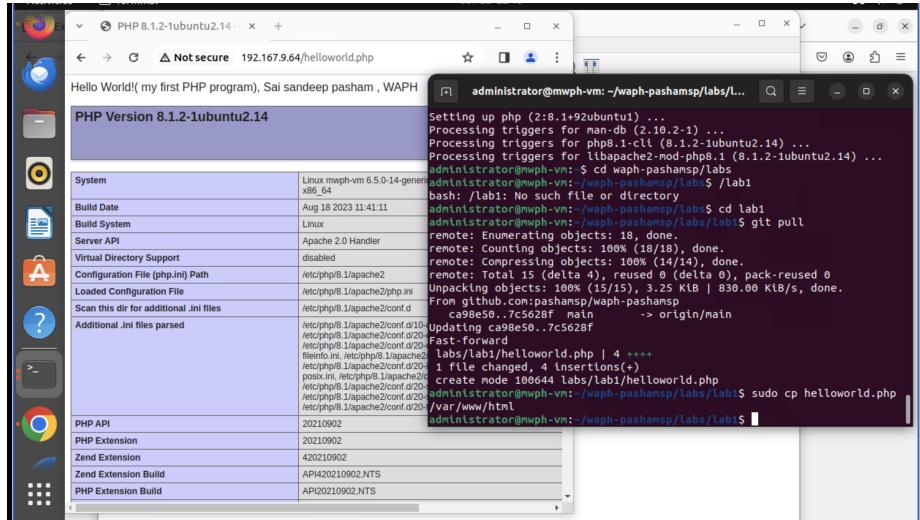


Figure 9: helloWorld.php

Included file `helloworld.php`:

```
<?php
    echo "Hello World!( my first PHP program), Sai sandeep pasham , WAPH";
    phpinfo();
?>
```

B. A simple PHP web application has been constructed to echo the path variable obtained via HTTP request. However, using PHP's `$_REQUEST['data']` to record path variables in both GET and POST requests exposes various security issues, including data manipulation, SQL injections, and the possibility of Remote Code Execution. To mitigate these risks, implement input validation, use prepared statements for SQL inputs, and sanitise user inputs. These techniques are effective mitigation tactics for improving the security of the online application.

Included file `echo.php`:

```
<?php
    $input = $_REQUEST["data"];
    echo "Hi from <strong>" . $input . "</strong>. <br>";
?>
```

Task 3: Understanding HTTP GET and POST requests.

A. The browser made an HTTP GET request by default, with the path variable sent in the URL as `IPaddress/echo.php?data="value"`. The input variable was then supplied as part of the response. The entire process, including the request, response, and accompanying HTTP stream, was analysed with Wireshark.

The screenshot shows a Linux desktop environment. On the left is a vertical dock containing icons for various applications like a file manager, terminal, and browser. In the center, there's a terminal window titled "administrator@mwpf-vm: ~/waph-pashamsp/labs/lab1\$". It displays a command-line session where a file named "echo.php" is being copied from the current directory to "/var/www/html". Below the terminal is a browser window showing the URL "localhost/echo.php?data=sai%20sandee%20pasham". The page content is "Hi from sai sandee pasham".

```

administrator@mwpf-vm: ~/waph-pashamsp/labs/lab1$ sudo cp echo.php /var/www/html
[...]
administrator@mwpf-vm: ~/waph-pashamsp/labs/lab1$ git pull
[...]
administrator@mwpf-vm: ~/waph-pashamsp/labs/lab1$ sudo cp echo.php /var/www/html
[...]

```

Figure 10: echo.php

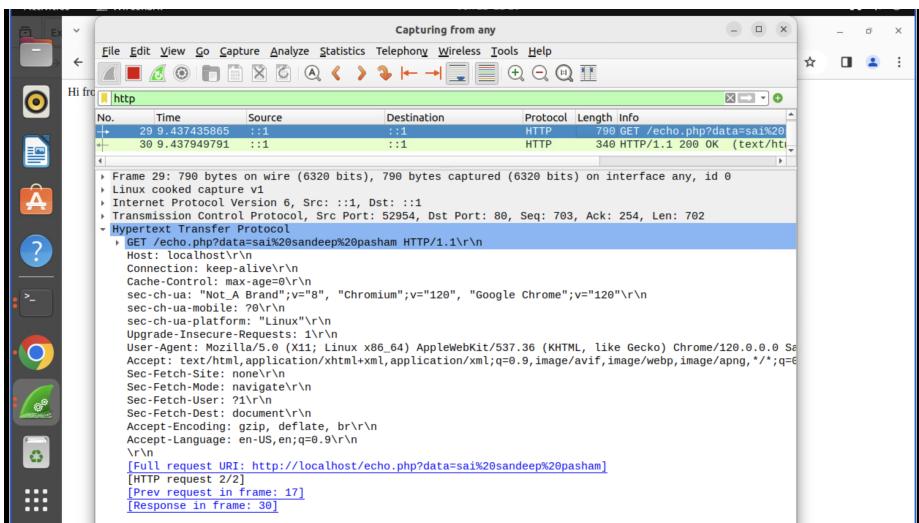


Figure 11: HTTP GET request in Wireshark

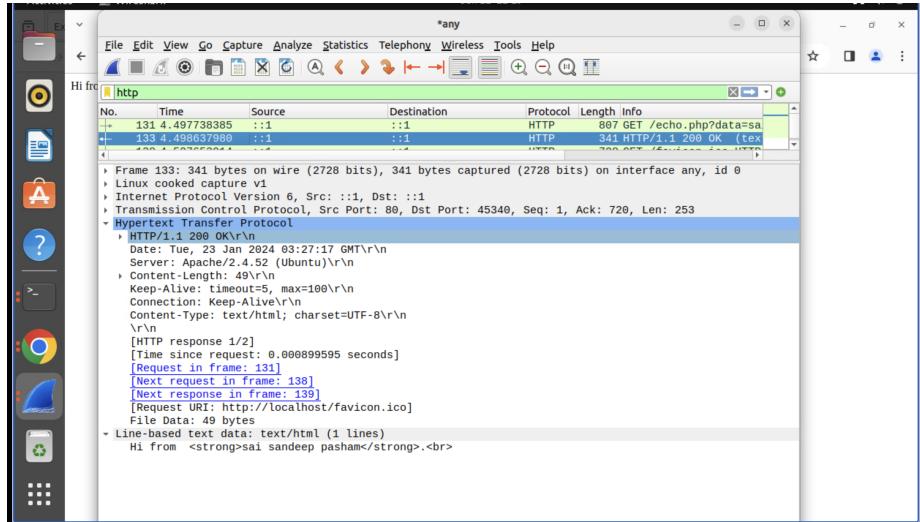


Figure 12: HTTP response in Wireshark

B.Curl, the Client URL command-line tool, is used to process data over a variety of network protocols. I used CURL in the terminal to make a post request to echo.php.

```
curl -X POST localhost/echo.php -d "data=Sai Sandeep Pasham"
```

C.HTTP GET and POST requests are similar in that they both use HTTP to facilitate communication between clients and servers. Analytical tools like as Wireshark can collect and analyse both sorts of requests. Furthermore, both use request headers and receive answers with status codes and headers, albeit the specific headers may differ.

However, there are distinctions between the two. GET requests deliver data via the URL, but POST requests send data via the HTTP body, which improves security by avoiding exposure to regular users. The aim also differs; GET is used for information retrieval, whereas POST is used to update information.

Since the echo.php is a mirror application that simply prints the received input, the responses in both HTTP GET and HTTP POST are identical.

Following this, the Labs/Lab1 folder was created to hold the project report, and the changes were pushed. The project report was generated using the Pandoc tool and the README.md file.

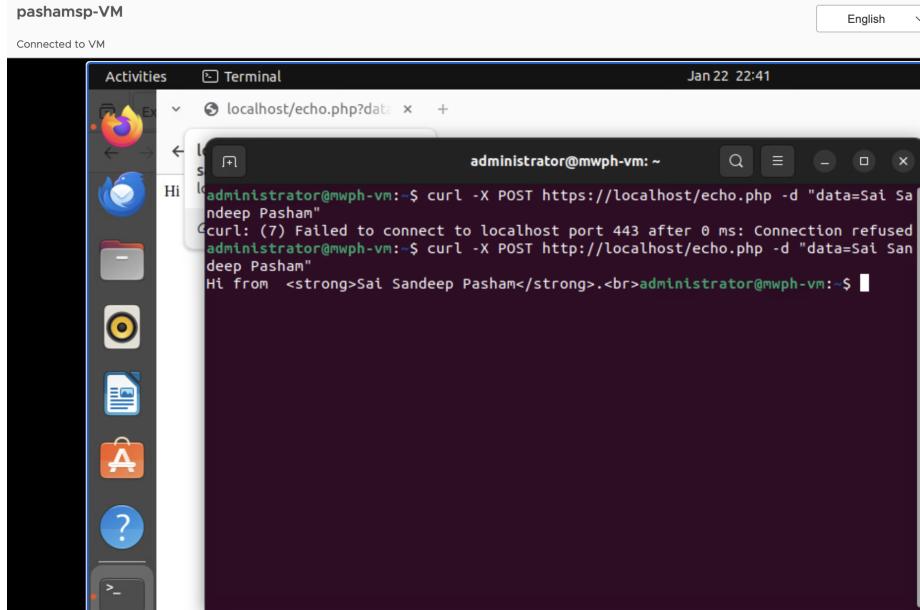


Figure 13: HTTP POST request using CURL

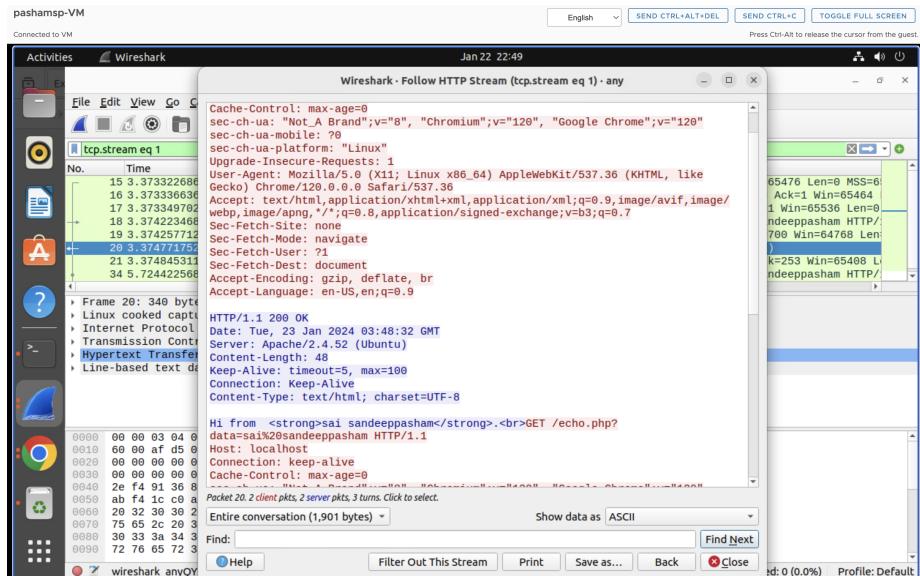


Figure 14: HTTP Stream in Wireshark