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 promiscous 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.

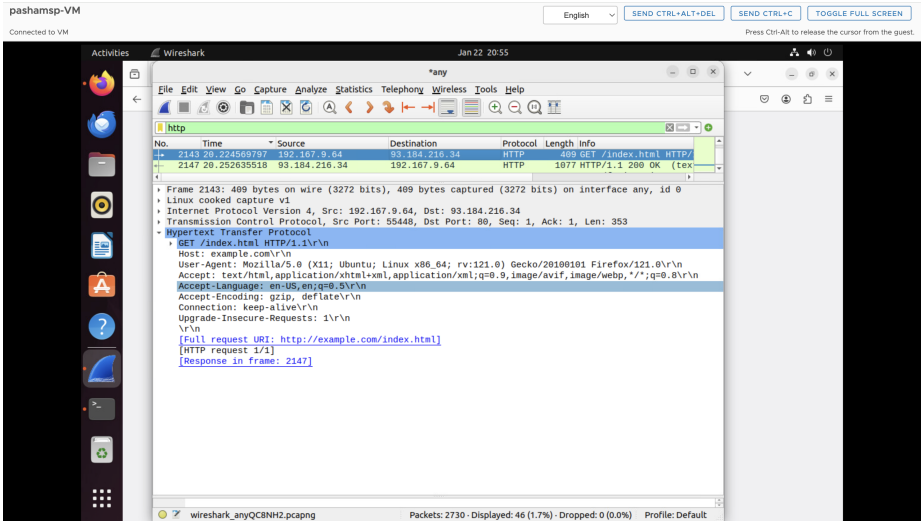


Figure 2: Wireshark HTTP Request

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.

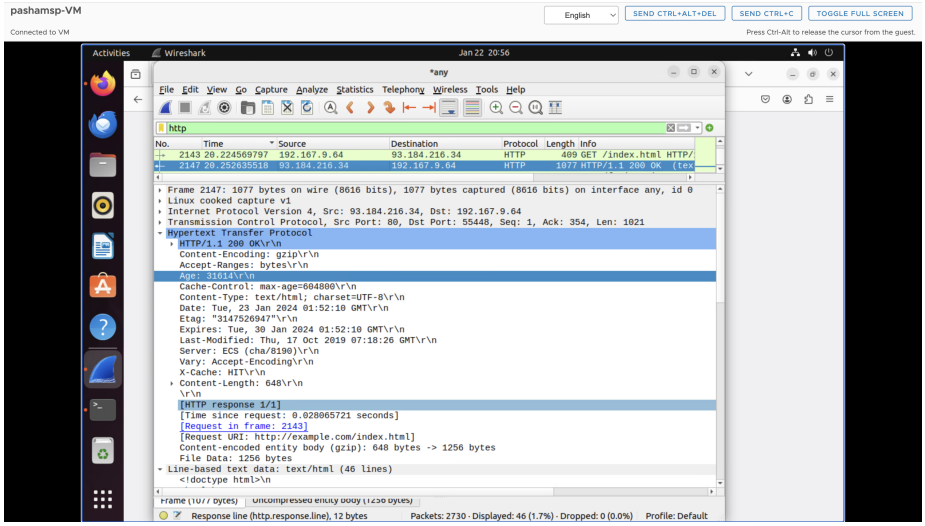


Figure 3: Wireshark HTTP Response

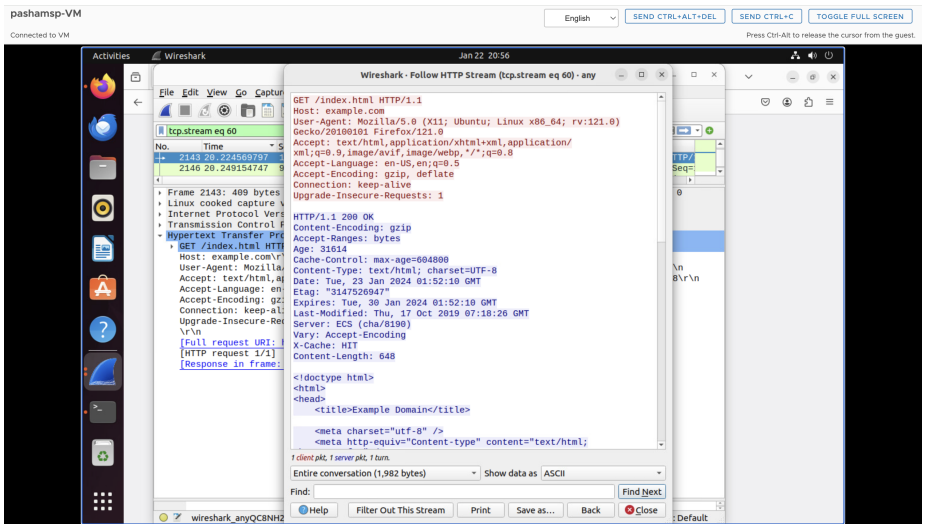


Figure 4: Wireshark HTTP Stream

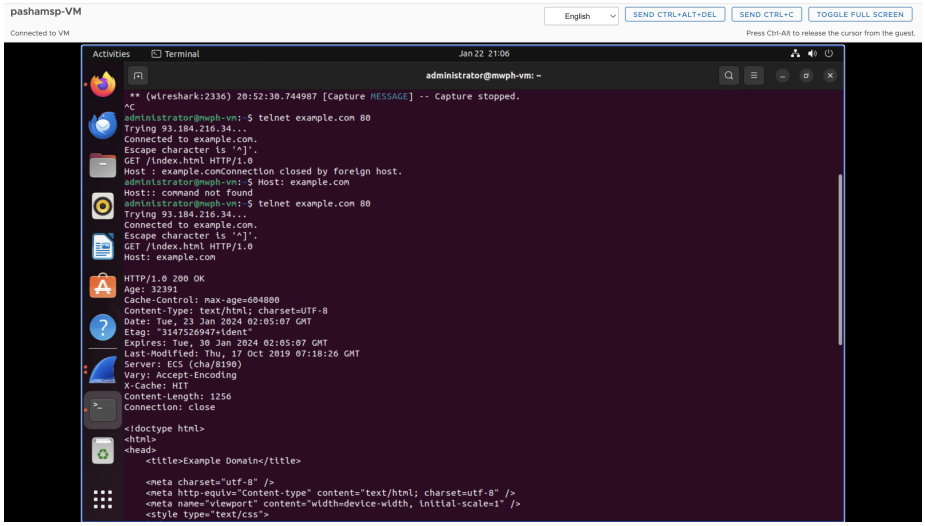
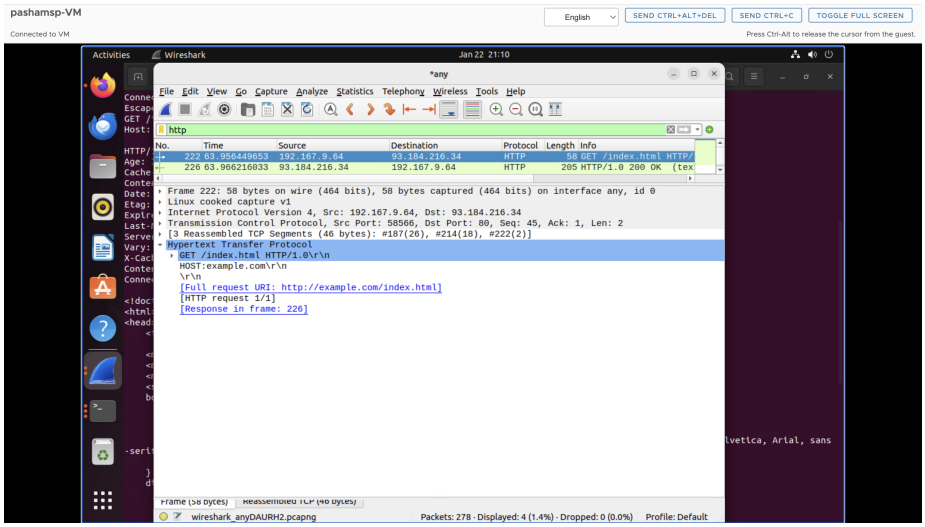


Figure 5: Telnet request



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

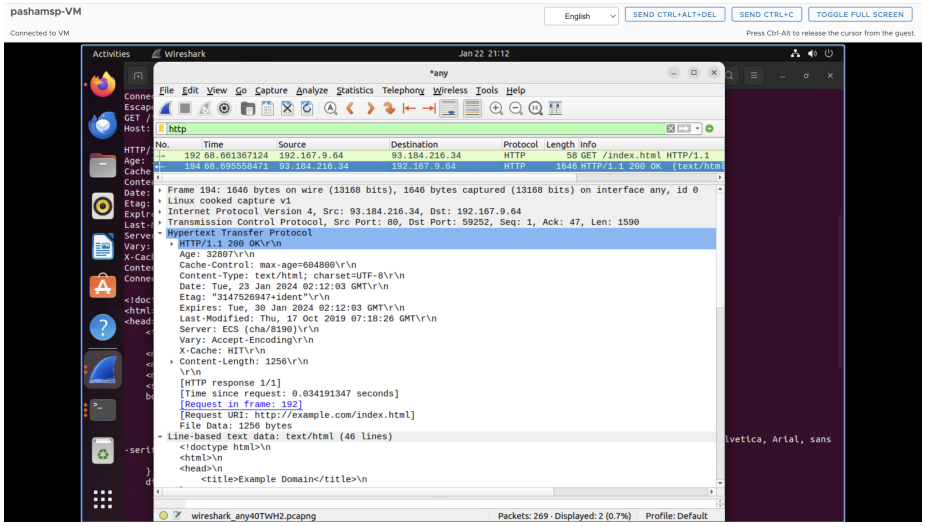


Figure 6: Telent response in wireshark

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
                                "</head> <body> <h1>Student: Sai Sandeep Pasham</h1>"
                                "<p>This exercise is done for Lab 1 , CGI Web Applications wit

    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.

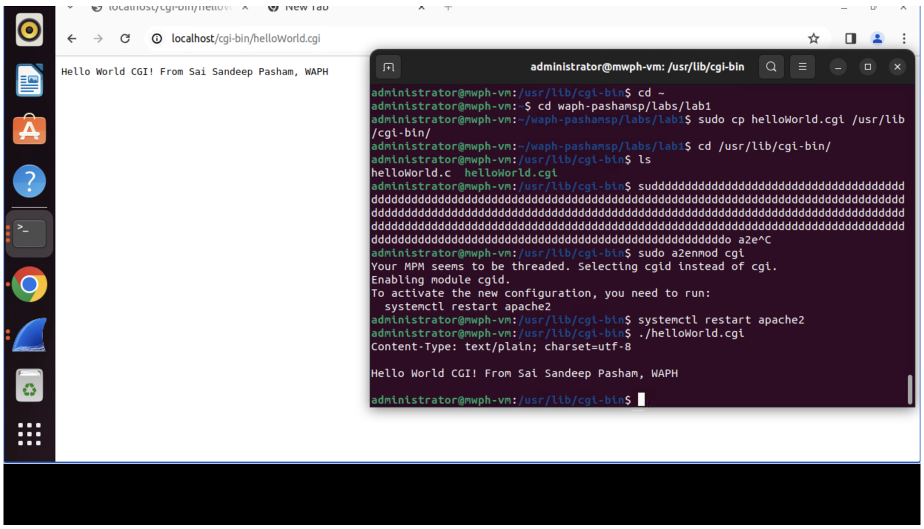


Figure 7: CGI program in C

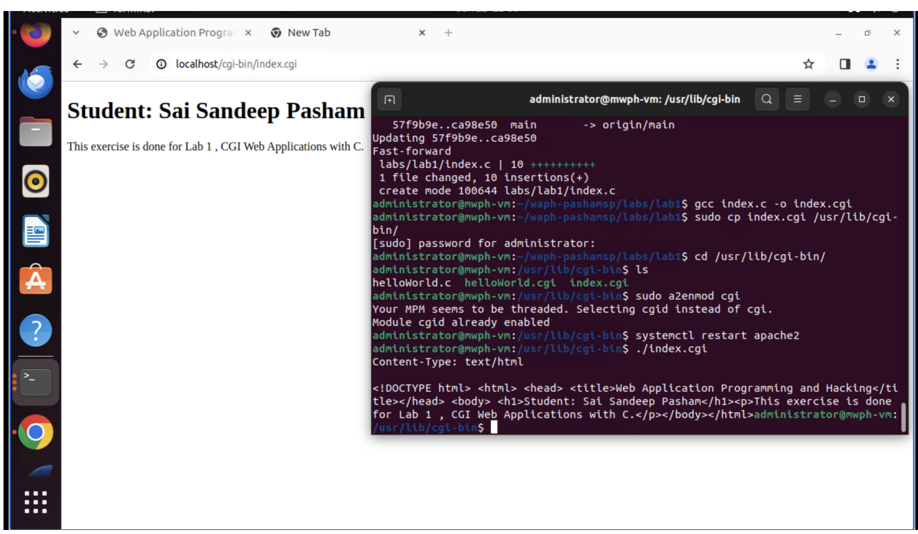


Figure 8: CGI in C and HTML

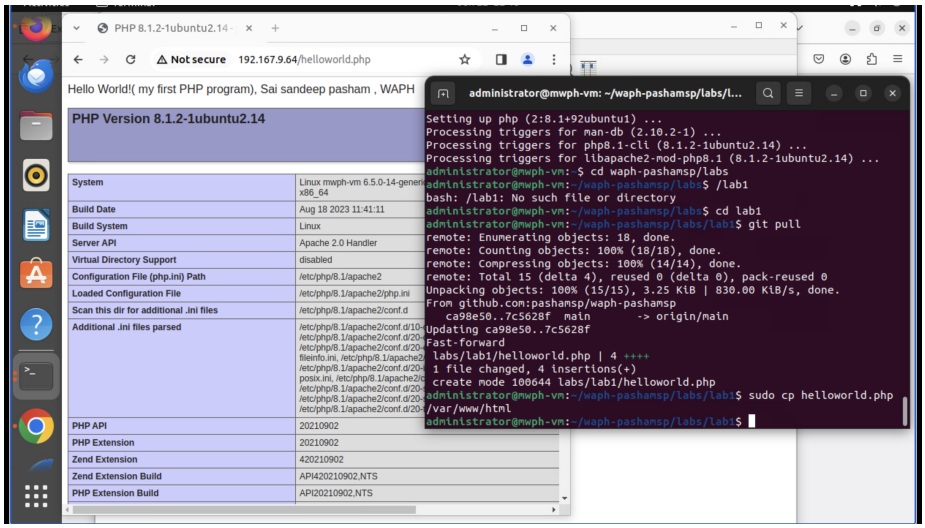


Figure 9: helloWorld.php

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.

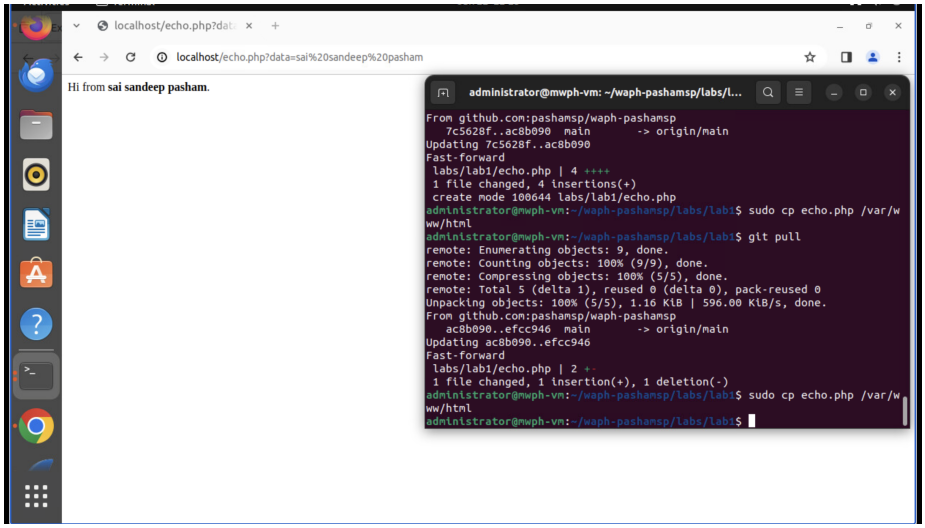


Figure 10: echo.php

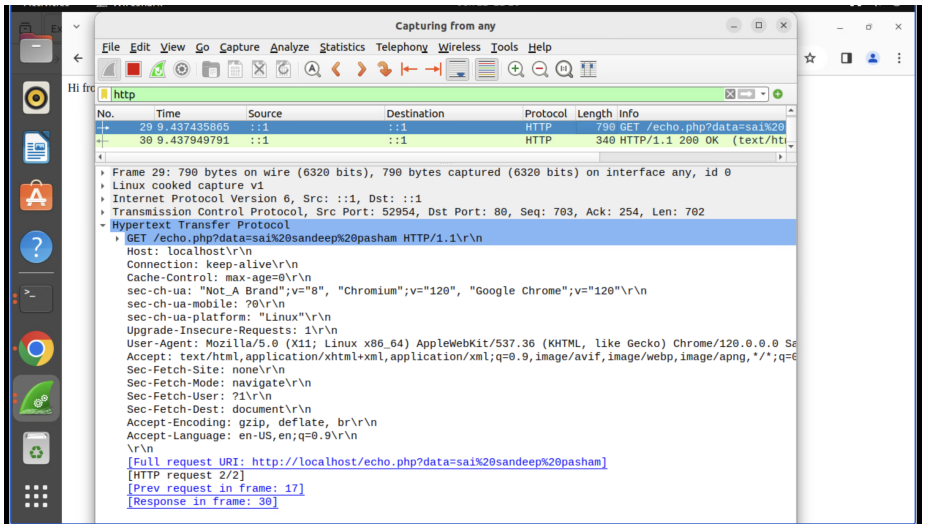


Figure 11: HTTP GET request in WireShark

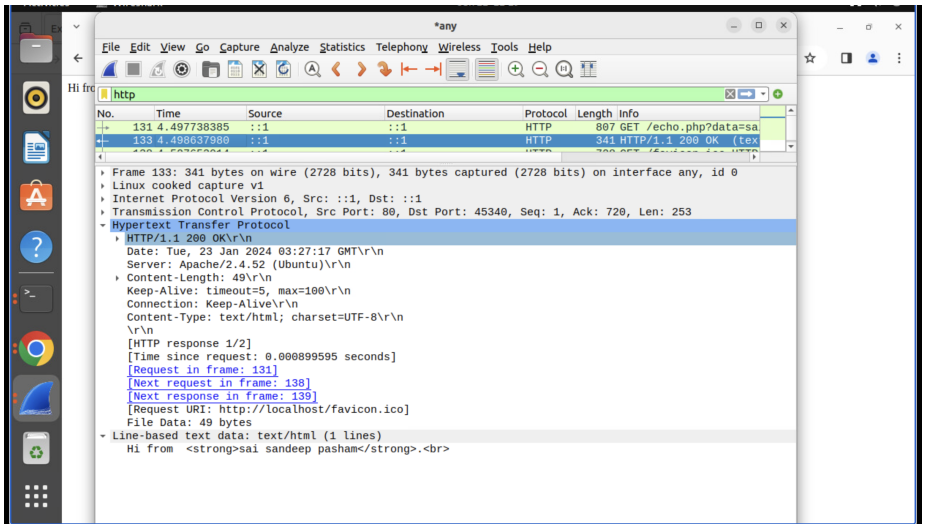


Figure 12: HTTP response in WireShark

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.

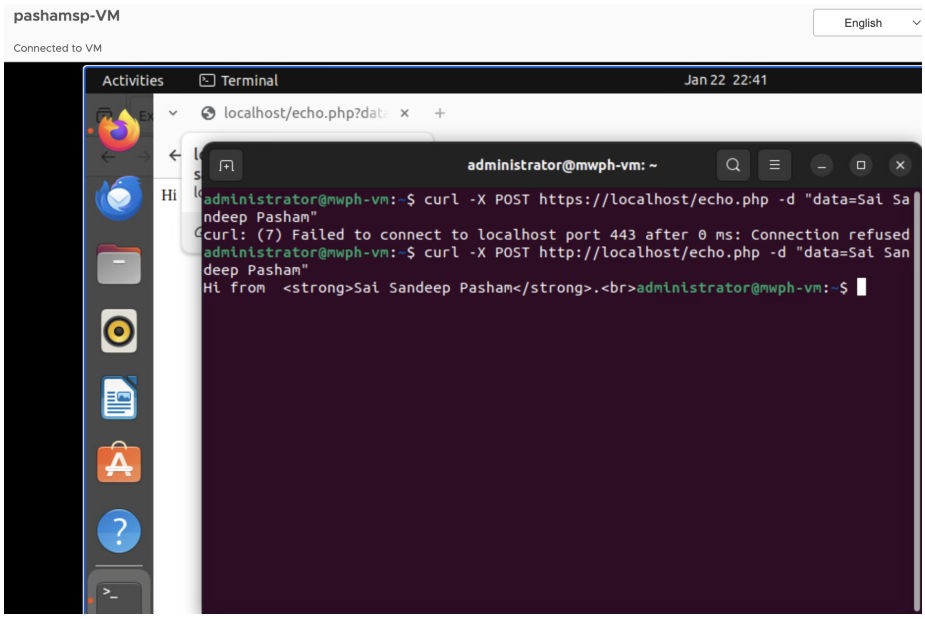


Figure 13: HTTP POST request using CURL

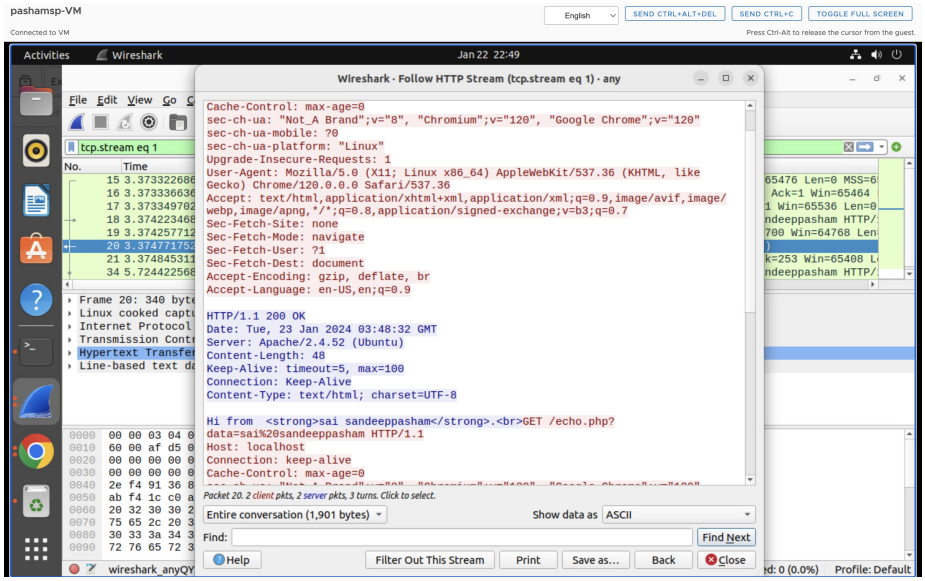


Figure 14: HTTP Stream in Wireshark