WAPH-Web Application Programming and Hacking

Instructor: Dr. Phu Phung

Student: Seth Okai

Lab 1 - Foundations of the Web

The lab's overview

In this lab, I learned a lot about using CGI and PHP in web application development. I also Learned how to use Wireshark to view different types of requests and responses. Lastly, I was able to use curl to create an HTTP post request with my name in the data.

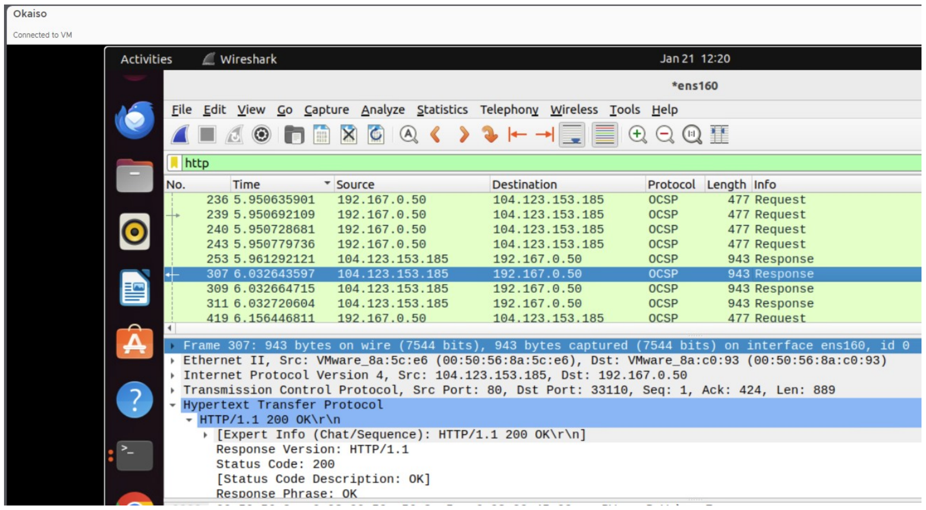
Lab Folder [https://github.com/Sethoka1/waph-okaiso/tree/881c048f60a3d1084ef65ff9d2fb517daa184da5/labs/

Part I: The Web and HTTP Protocol

Task 1 Familiar with the Wireshark tool and HTTP protocol

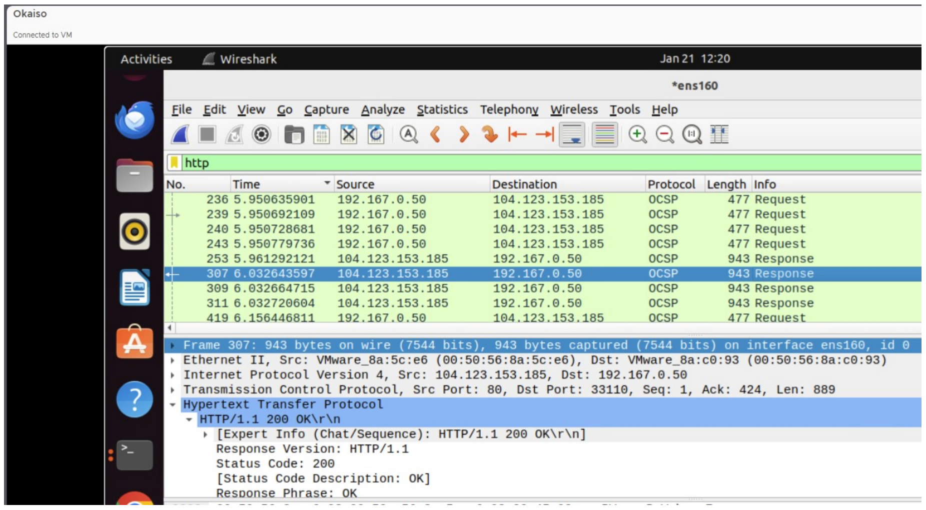
I started a traffic capture on WireShark, sent an HTTP request to 'example.com' via my browser, and used Wireshark filters to isolate only HTTP traffic.

HTTP Request


Okaiso
Connected to VM
Activities
Wireshark
Jan 21 12:20
*ens160
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

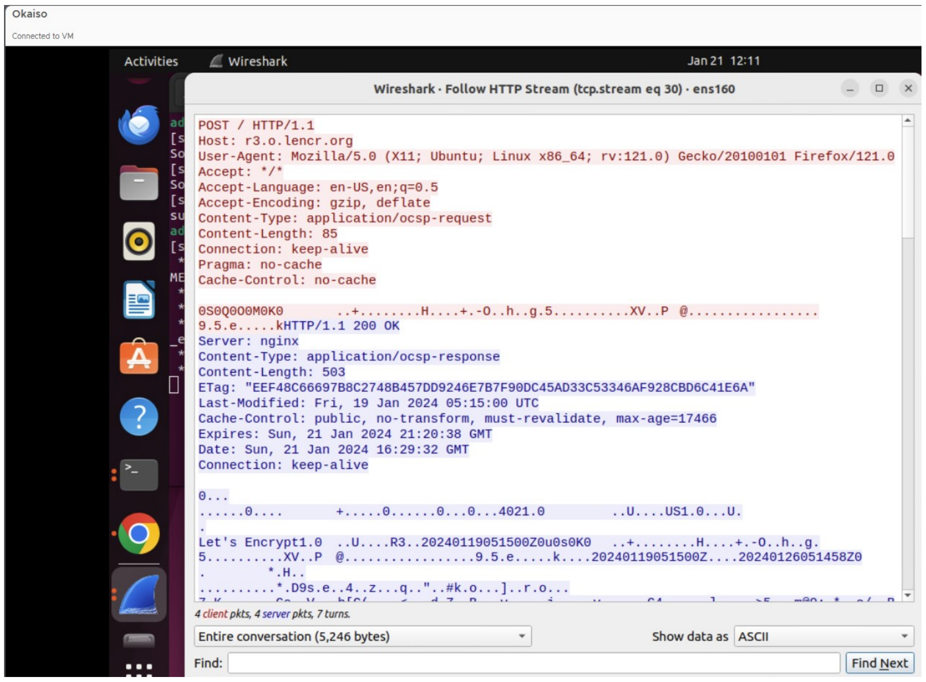
```
No.    Time         Source           Destination      Protocol Length Info
   236 5.950635901  192.167.0.50     104.123.153.185  OCSP        477 Request
   239 5.950692109  192.167.0.50     104.123.153.185  OCSP        477 Request
   240 5.950728681  192.167.0.50     104.123.153.185  OCSP        477 Request
   243 5.950779736  192.167.0.50     104.123.153.185  OCSP        477 Request
   253 5.961292121  104.123.153.185  192.167.0.50     OCSP        943 Response
   307 6.032643597  104.123.153.185  192.167.0.50     OCSP        943 Response
   309 6.032664715  104.123.153.185  192.167.0.50     OCSP        943 Response
   311 6.032720604  104.123.153.185  192.167.0.50     OCSP        943 Response
   419 6.156446811  192.167.0.50     104.123.153.185  OCSP        477 Request

Frame 307: 943 bytes on wire (7544 bits), 943 bytes captured (7544 bits) on interface ens160, id 0
Ethernet II, Src: VMware_8a:5c:e6 (00:50:56:8a:5c:e6), Dst: VMware_8a:c0:93 (00:50:56:8a:c0:93)
Internet Protocol Version 4, Src: 104.123.153.185, Dst: 192.167.0.50
Transmission Control Protocol, Src Port: 80, Dst Port: 33110, Seq: 1, Ack: 424, Len: 889
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
```

Http Response


Okaiso
Connected to VM
Activities
Wireshark
Jan 21 12:20
*ens160
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.    Time         Source           Destination      Protocol Length Info
   236 5.950635901  192.167.0.50     104.123.153.185  OCSP        477 Request
   239 5.950692109  192.167.0.50     104.123.153.185  OCSP        477 Request
   240 5.950728681  192.167.0.50     104.123.153.185  OCSP        477 Request
   243 5.950779736  192.167.0.50     104.123.153.185  OCSP        477 Request
   253 5.961292121  104.123.153.185  192.167.0.50     OCSP        943 Response
   307 6.032643597  104.123.153.185  192.167.0.50     OCSP        943 Response
   309 6.032664715  104.123.153.185  192.167.0.50     OCSP        943 Response
   311 6.032720604  104.123.153.185  192.167.0.50     OCSP        943 Response
   419 6.156446811  192.167.0.50     104.123.153.185  OCSP        477 Request

Frame 307: 943 bytes on wire (7544 bits), 943 bytes captured (7544 bits) on interface ens160, id 0
Ethernet II, Src: VMware_8a:5c:e6 (00:50:56:8a:5c:e6), Dst: VMware_8a:c0:93 (00:50:56:8a:c0:93)
Internet Protocol Version 4, Src: 104.123.153.185, Dst: 192.167.0.50
Transmission Control Protocol, Src Port: 80, Dst Port: 33110, Seq: 1, Ack: 424, Len: 889
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
```

Http Stream



```
POST / HTTP/1.1
Host: r3.o.lencr.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 85
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

0S0Q0O0M0K0       ..+........H....+.-O..h..g.5..........XV..P @................
9.5.e.....kHTTP/1.1 200 OK
Server: nginx
Content-Type: application/ocsp-response
Content-Length: 503
ETag: "EEF48C66697B8C2748B457DD9246E7B7F90DC45AD33C53346AF928CBD6C41E6A"
Last-Modified: Fri, 19 Jan 2024 05:15:00 UTC
Cache-Control: public, no-transform, must-revalidate, max-age=17466
Expires: Sun, 21 Jan 2024 21:20:38 GMT
Date: Sun, 21 Jan 2024 16:29:32 GMT
Connection: keep-alive

0...
......0....       +.....0......0...0...4021.0         ..U....US1.0...U.
.
Let's Encrypt1.0  ..U....R3..20240119051500Z0u0s0K0   ..+........H....+.-O..h..g.
5..........XV..P  @.................9.5.e.....k....20240119051500Z....20240126051458Z0
.        *.H..
..........*.D9s.e..4..z...q.."..#k.o...]..r.o...
```

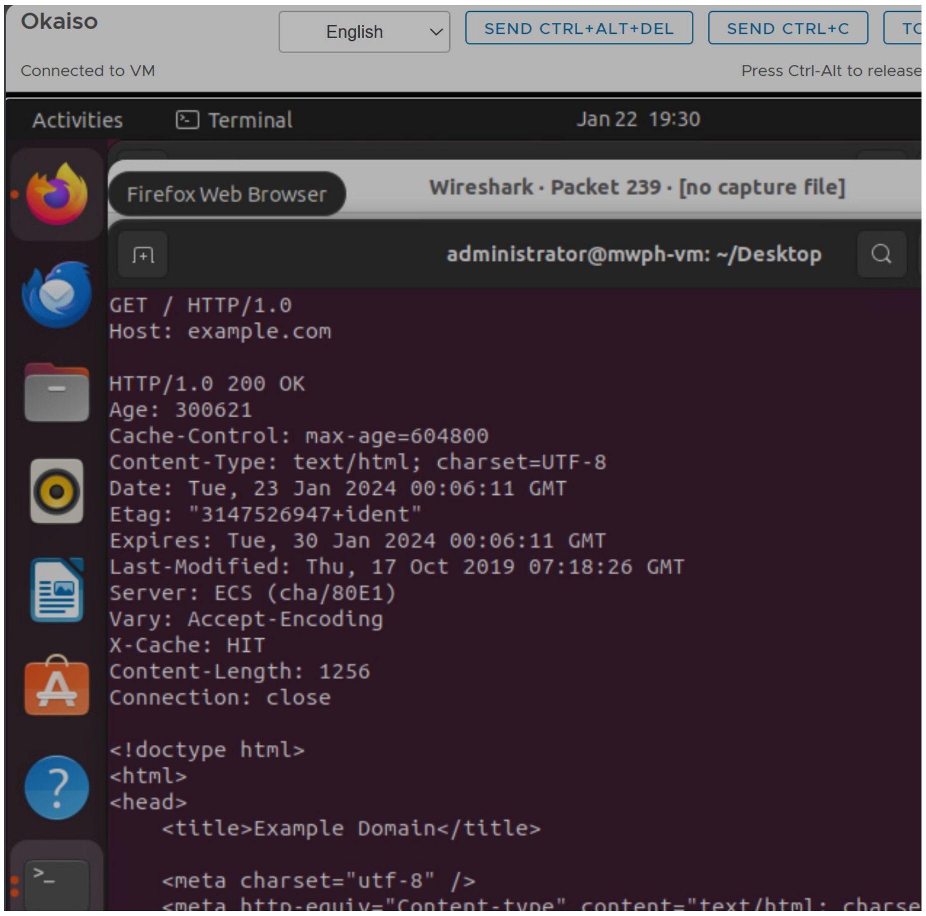
4 client pkts, 4 server pkts, 7 turns.

Entire conversation (5,246 bytes)   Show data as ASCII

Find: Find Next

Task 2 (10 pts). Understanding HTTP using telnet and Wireshark.

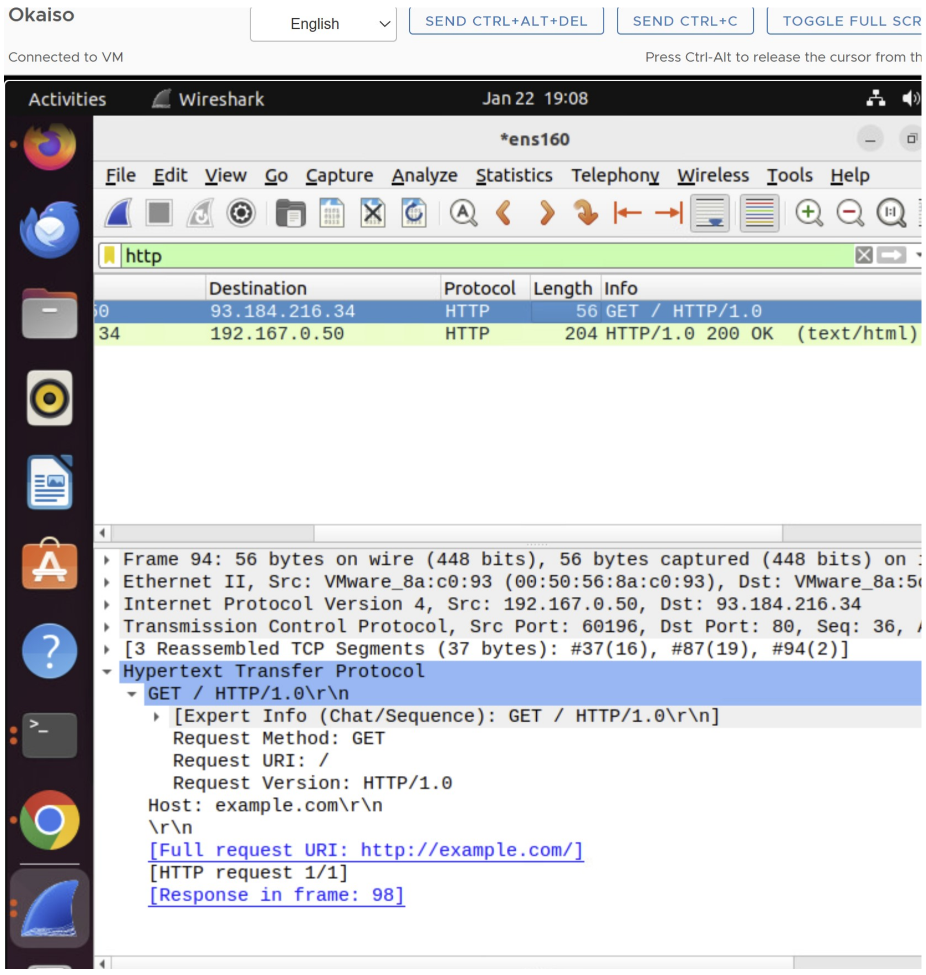
HTTP request from terminal



```
GET / HTTP/1.0
Host: example.com

HTTP/1.0 200 OK
Age: 300621
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 23 Jan 2024 00:06:11 GMT
Etag: "3147526947+ident"
Expires: Tue, 30 Jan 2024 00:06:11 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (cha/80E1)
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; charse
```

HTTP request from Wireshark

- From wire shark the carriage return character and line -feed character is included in the request unlike the terminal request


Okaiso
English
SEND CTRL+ALT+DEL
SEND CTRL+C
TOGGLE FULL SCR
Connected to VM
Press Ctrl-Alt to release the cursor from th
Activities
Wireshark
Jan 22 19:08
*ens160
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
     Destination        Protocol Length Info
50   93.184.216.34      HTTP         56 GET / HTTP/1.0
34   192.167.0.50       HTTP        204 HTTP/1.0 200 OK  (text/html)
```

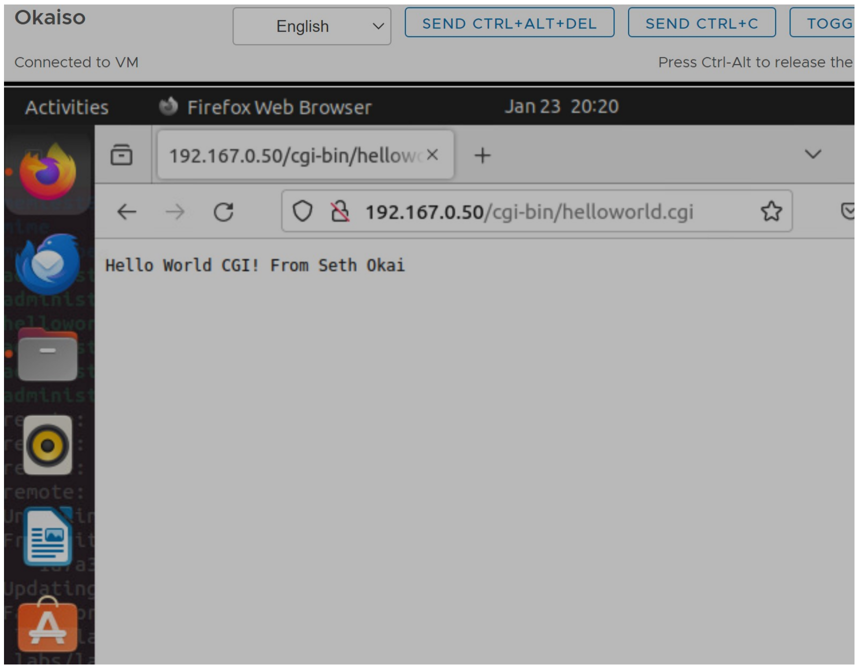
```
▸ Frame 94: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on
▸ Ethernet II, Src: VMware_8a:c0:93 (00:50:56:8a:c0:93), Dst: VMware_8a:5
▸ Internet Protocol Version 4, Src: 192.167.0.50, Dst: 93.184.216.34
▸ Transmission Control Protocol, Src Port: 60196, Dst Port: 80, Seq: 36,
▸ [3 Reassembled TCP Segments (37 bytes): #37(16), #87(19), #94(2)]
▾ Hypertext Transfer Protocol
  ▾ GET / HTTP/1.0\r\n
    ▸ [Expert Info (Chat/Sequence): GET / HTTP/1.0\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.0
    Host: example.com\r\n
    \r\n
    [Full request URI: http://example.com/]
    [HTTP request 1/1]
    [Response in frame: 98]
```

- There isn't any difference between the response message in wire shark and the one from the terminal

Part II: Basic Web Application Programming

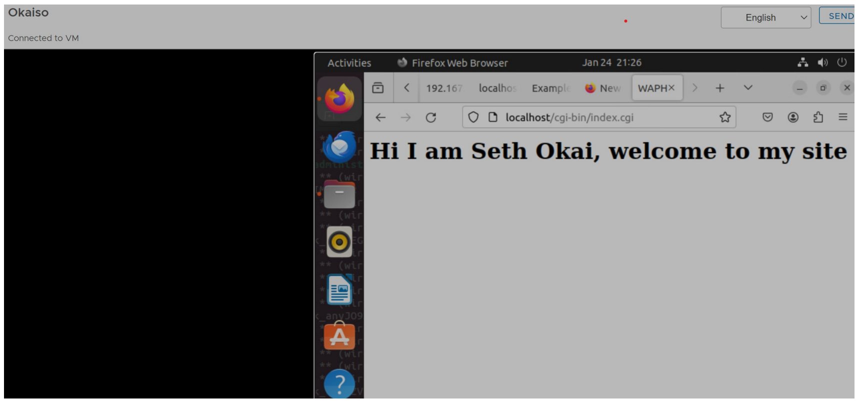
1. CGI Web applications in C
   - Make a c file named hello.world.c
   - Write the c code
   - Install GCC compiler
   - Use GCC to compile the code
   - Test the code using ./helloworld.cgi"
   - Move the code to /usr/lib/cgi-bin/

- Vist the sit using: "ip-address/cgi-bin/helloworld.cgi



```
Hello World CGI! From Seth Okai
```

Below is another c cgi program deployed with html



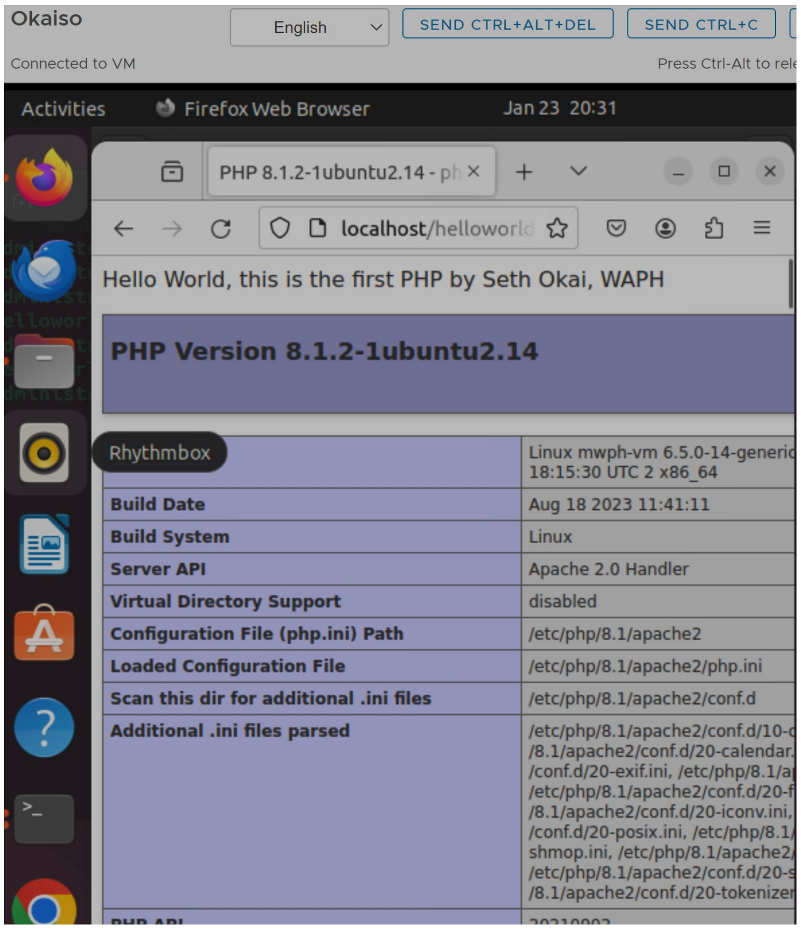
Hi I am Seth Okai, welcome to my site

Included file `index.c`:

```
#include <stdio.h>

int main() {

     printf("Content-Type: text/html; charset=utf-8\n\n");
```

```
    printf("<html>\n");
    printf("<head>\n");
    printf("<title>WAPH</title>\n");
    printf("</head>\n");
    printf("<body>\n");
    printf("<h1>Hi I am Seth Okai, welcome to my site</h1>\n");
    printf("</body>\n");
    printf("</html>\n");

    return 0;
}
```

3. A simple PHP Web Application with user input
   - Install php and configure it to work with Apache webserver ($Sudo agt-get install php libapache2-mod-php -y)
   - Make a PHP file named hello.world.PHP
   - Write the php code
   - Install GCC compiler
   - Deploy the code to webserver root directory (/var/www/html/)
   - Vist the sit using: "ip-address/helloworld.php

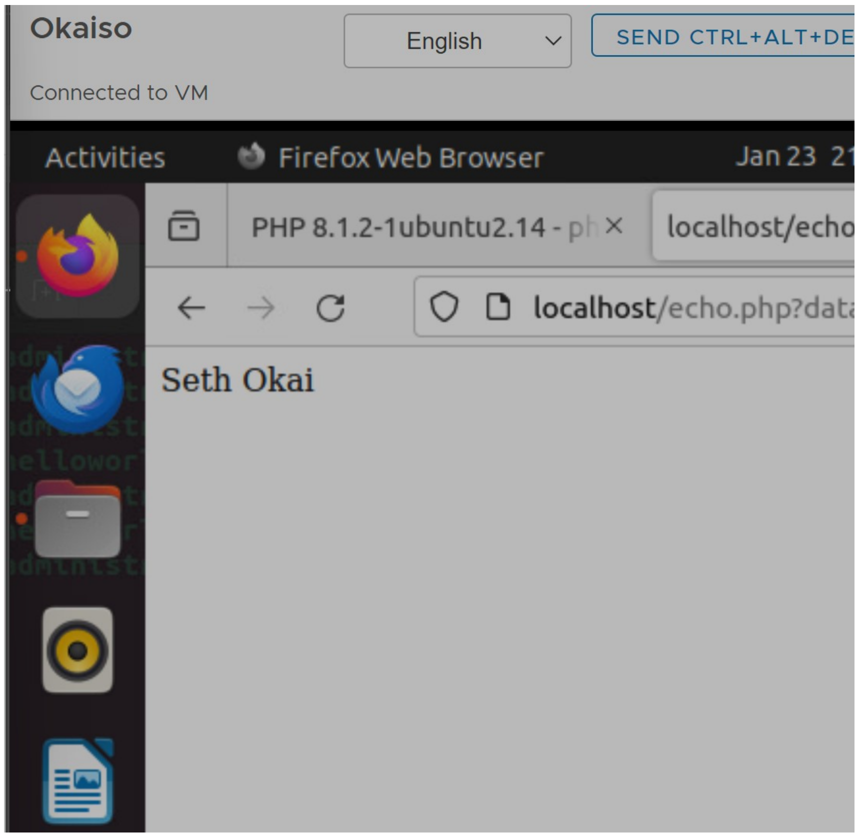


Hello World, this is the first PHP by Seth Okai, WAPH

PHP Version 8.1.2-1ubuntu2.14

	Linux mwph-vm 6.5.0-14-generic 18:15:30 UTC 2 x86_64
Build Date	Aug 18 2023 11:41:11
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.1/apache2
Loaded Configuration File	/etc/php/8.1/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.1/apache2/conf.d
Additional .ini files parsed	/etc/php/8.1/apache2/conf.d/10-o /8.1/apache2/conf.d/20-calendar. /conf.d/20-exif.ini, /etc/php/8.1/a /etc/php/8.1/apache2/conf.d/20-f /8.1/apache2/conf.d/20-iconv.ini, /conf.d/20-posix.ini, /etc/php/8.1/ shmop.ini, /etc/php/8.1/apache2/ /etc/php/8.1/apache2/conf.d/20-s /8.1/apache2/conf.d/20-tokenizer
PHP API	20210902

Below is another php program with my name in the data:



Seth Okai

```
Included file `echo.php`:
```

```
<?php

    echo $_REQUEST['data'];

?>
```

This web application is vulnerable to cross-site scripting (XSS) because there isn't any sanitization of inputs.

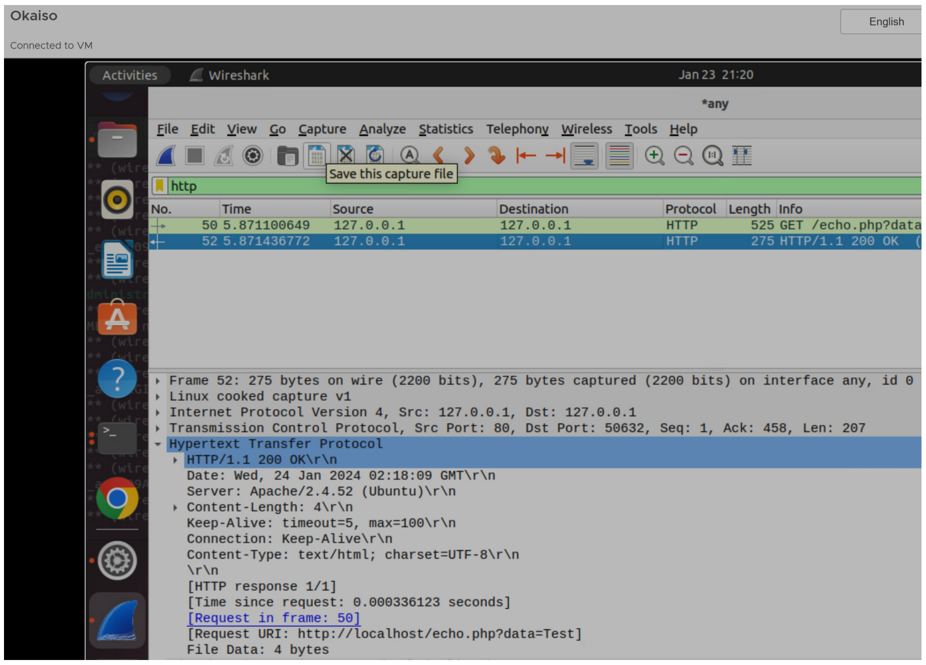
Task 3

Understanding HTTP GET and POST requests

I started a capture on wireshark to examine the echo.php page with my name in it.

Below are the response and request pages

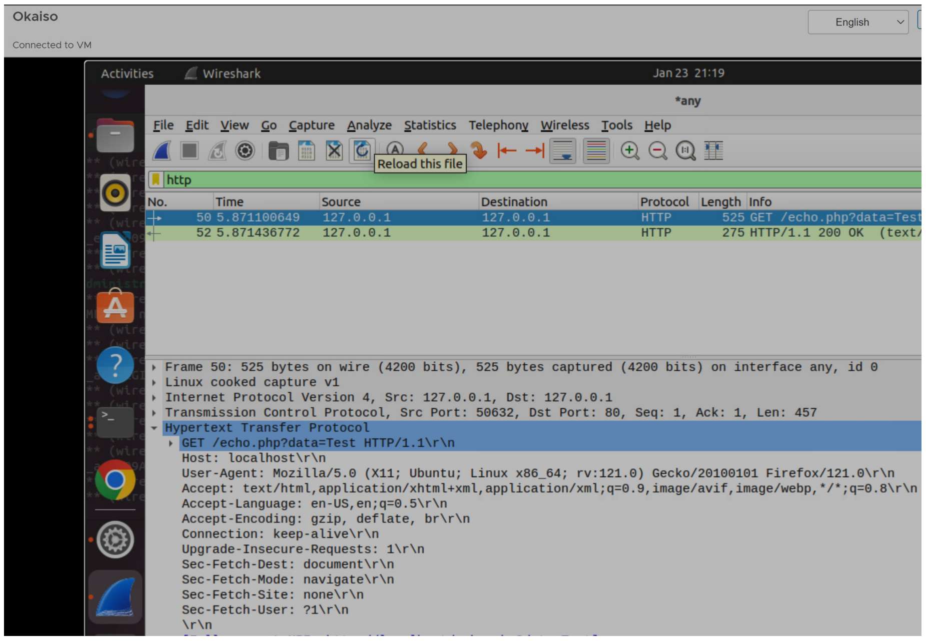
Echo Response



```
No.   Time          Source      Destination  Protocol Length Info
   50 5.871100649   127.0.0.1   127.0.0.1    HTTP        525 GET /echo.php?data
   52 5.871436772   127.0.0.1   127.0.0.1    HTTP        275 HTTP/1.1 200 OK  (
```

```
› Frame 52: 275 bytes on wire (2200 bits), 275 bytes captured (2200 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› Transmission Control Protocol, Src Port: 80, Dst Port: 50632, Seq: 1, Ack: 458, Len: 207
⌄ Hypertext Transfer Protocol
  › HTTP/1.1 200 OK\r\n
    Date: Wed, 24 Jan 2024 02:18:09 GMT\r\n
    Server: Apache/2.4.52 (Ubuntu)\r\n
  › Content-Length: 4\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.000336123 seconds]
    [Request in frame: 50]
    [Request URI: http://localhost/echo.php?data=Test]
    File Data: 4 bytes
```

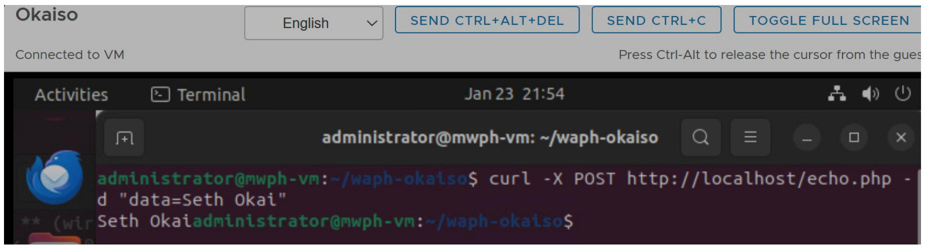
Echo Request



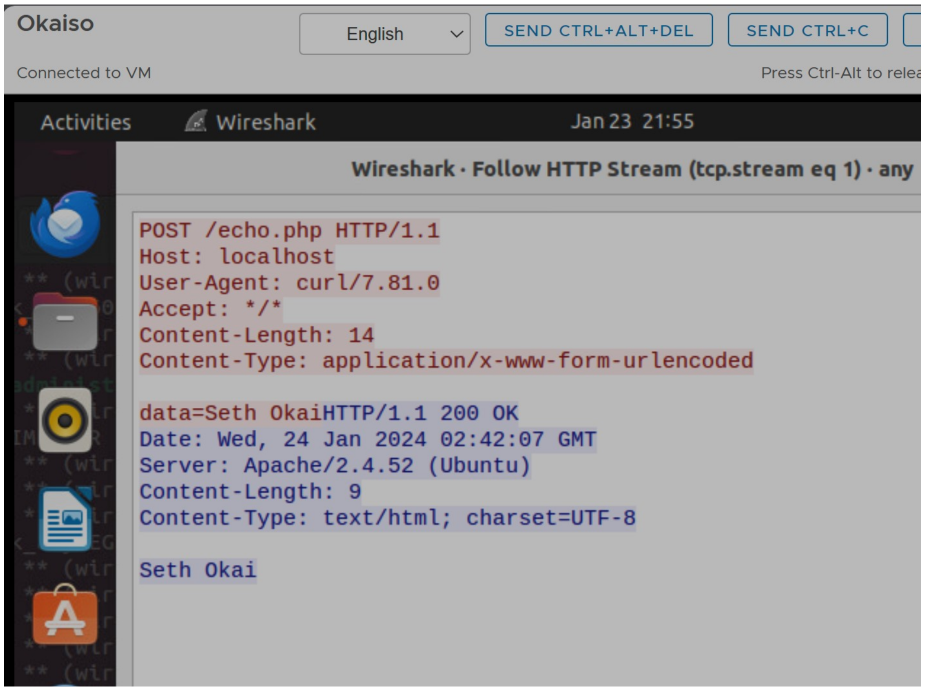
```
No.   Time          Source      Destination  Protocol Length Info
   50 5.871100649   127.0.0.1   127.0.0.1    HTTP        525 GET /echo.php?data=Test
   52 5.871436772   127.0.0.1   127.0.0.1    HTTP        275 HTTP/1.1 200 OK  (text/
```

```
› Frame 50: 525 bytes on wire (4200 bits), 525 bytes captured (4200 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› Transmission Control Protocol, Src Port: 50632, Dst Port: 80, Seq: 1, Ack: 1, Len: 457
⌄ Hypertext Transfer Protocol
  › GET /echo.php?data=Test HTTP/1.1\r\n
    Host: localhost\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    Sec-Fetch-Dest: document\r\n
    Sec-Fetch-Mode: navigate\r\n
    Sec-Fetch-Site: none\r\n
    Sec-Fetch-User: ?1\r\n
    \r\n
```

Using curl I was able to post a request ($curl -X POST [URL] -d [field-name=input]



```
administrator@mwph-vm:~/waph-okaiso$ curl -X POST http://localhost/echo.php -
d "data=Seth Okai"
Seth Okaiadministrator@mwph-vm:~/waph-okaiso$
```



```
POST /echo.php HTTP/1.1
Host: localhost
User-Agent: curl/7.81.0
Accept: */*
Content-Length: 14
Content-Type: application/x-www-form-urlencoded

data=Seth OkaiHTTP/1.1 200 OK
Date: Wed, 24 Jan 2024 02:42:07 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Length: 9
Content-Type: text/html; charset=UTF-8

Seth Okai
```

Both the GET and POST requests are HTTP protocols, but the parameters are visible in the URL for the GET request and data is transmitted in the body of the HTTP packet for the post request.