

Contents

waph-najarzs	1
WAPH-Web Application Programming and Hacking	1
Lab 1 - Foundations of the Web	1
Overview	1
Part 1 - The Web and HTTP Protocol	1
Task 1. Familiar with the Wireshark tool and HTTP protocol	1
Task 2. Understanding HTTP using telnet and Wireshark	3
Part II - Basic Web Application Programming	4
Task 1. CGI Web applications in C	4
Task 2. A simple PHP Web Application with user input.	5
Task 3. Understanding HTTP GET and POST requests.	6

waph-najarzs

WAPH-Web Application Programming and Hacking

Lab 1 - Foundations of the Web

Name: Zaid Najar

Instructor: Dr. Phu Phung

Email: najarzs@mail.uc.edu

Overview

this lab shows core concepts of the web and HTTP protocol through Wireshark and web application programming. There are two parts: HTTP protocol and basic web application programming. I should be able to use Wireshark to analyze HTTP traffic, use telnet, and develop applications using CGI and PHP.

Part 1 - The Web and HTTP Protocol

Task 1. Familiar with the Wireshark tool and HTTP protocol

I used Wireshark by running it as the lecture instructed me to do and opened an example html page in chrome. I then used the filter to search through all HTTP protocols and used that to take my screenshots.

No.	Time	Source	Destination	Protocol	Length	Info
240	12.409915675	10.0.3.15	93.184.215.14	HTTP	482	GET /index.html HTTP/1.1
248	12.419670536	93.184.215.14	10.0.3.15	HTTP	1885	HTTP/1.1 200 OK (text/html)

- Frame 248: 482 bytes on wire (3856 bits), 482 bytes captured (3856 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 10.0.3.15, Dst: 93.184.215.14
- Transmission Control Protocol, Src Port: 49060, Dst Port: 80, Seq: 1, Ack: 1, Len: 426
- Hypertext Transfer Protocol
 - GET /index.html HTTP/1.1\r\n
 - Host: example.com\r\n
 - Connection: keep-alive\r\n
 - Upgrade-Insecure-Requests: 1\r\n
 - User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36\r\n
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
 - Accept-Encoding: gzip, deflate\r\n
 - Accept-Language: en-US,en;q=0.8\r\n
 - \r\n

[Full request URI: http://example.com/index.html]

Screenshot 1: Request

No.	Time	Source	Destination	Protocol	Length	Info
246	12.460915675	10.0.3.15	93.184.215.14	HTTP	482	GET /index.html HTTP/1.1
248	12.419670536	93.184.215.14	10.0.3.15	HTTP	1885	HTTP/1.1 200 OK (text/html)

- Frame 248: 1085 bytes on wire (8680 bits), 1085 bytes captured (8680 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 93.184.215.14, Dst: 10.0.3.15
- Transmission Control Protocol, Src Port: 80, Dst Port: 49060, Seq: 1, Ack: 427, Len: 1029
- Hypertext Transfer Protocol
 - HTTP/1.1 200 OK\r\n
 - Content-Encoding: gzip\r\n
 - Accept-Ranges: bytes\r\n
 - Age: 439513\r\n
 - Cache-Control: max-age=604800\r\n
 - Content-Type: text/html; charset=UTF-8\r\n
 - Date: Tue, 14 May 2024 20:29:01 GMT\r\n
 - Etag: "3147526947+gzip"\r\n
 - Expires: Tue, 21 May 2024 20:29:01 GMT\r\n
 - Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT\r\n

Screenshot 2: Response

```

GET /index.html HTTP/1.1
Host: example.com
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

HTTP/1.1 200 OK
Content-Encoding: gzip
Accept-Ranges: bytes
Age: 439513
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 14 May 2024 20:29:01 GMT
Etag: "3147526947+gzip"
Expires: Tue, 21 May 2024 20:29:01 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECAcc (chd/0776)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 648

<!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 {
      width: 600px;

```

Screenshot 3: HTTP Stream

Task 2. Understanding HTTP using telnet and Wireshark

Telnet was used in this task to send a minimal HTTP request to Wireshark. After opening the example.com link and filtering by HTTP, I opened the terminal and used the command `telnet example.com 80`. This connected and then I used a GET method and the Host header to analyze the capture.

1. A screenshot of your terminal showing the HTTP Request (you typed) and

```
zaid@zaid-VirtualBox:~$ telnet -4 example.com 80
Trying 93.184.215.14...
Connected to example.com.
Escape character is '^]'.
GET /index.html HTTP/1.0
Host: example.com

HTTP/1.0 200 OK
Age: 441628
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 14 May 2024 21:04:16 GMT
Etag: "3147526947+gzip+ident"
Expires: Tue, 21 May 2024 21:04:16 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECAcc (chd/0776)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
Connection: close
```

HTTP response from the server

2. A screenshot of the HTTP Request message (you typed in telnet above) in

898.34.828206087	93.184.215.14	10.0.3.15	HTTP	1071 HTTP/1.1 404 Not Found (text/html)
1641 210.072751792	10.0.3.15	93.184.215.14	HTTP	58 GET /index.html HTTP/1.0
1643 210.095952547	93.184.215.14	10.0.3.15	HTTP	1673 HTTP/1.0 200 OK (text/html)
1890 306.436073129	10.0.3.15	91.189.91.49	HTTP	143 GET / HTTP/1.1
1892 306.404781383	91.189.91.49	10.0.3.15	HTTP	245 HTTP/1.1 204 No Content
2229 606.49985953	10.0.3.15	185.125.190.49	HTTP	143 GET / HTTP/1.1
2231 606.592434655	185.125.190.49	10.0.3.15	HTTP	245 HTTP/1.1 204 No Content

• Frame 1641: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface any, id 0

• Linux cooked capture v1

• Internet Protocol Version 4, Src: 10.0.3.15, Dst: 93.184.215.14

• Transmission Control Protocol, Src Port: 50608, Dst Port: 80, Seq: 46, Ack: 1, Len: 2

• [3 Reassembled TCP Segments (47 bytes): #1635(26), #1639(19), #1641(2)]

• Hypertext Transfer Protocol

• GET /index.html HTTP/1.0\r\n

Host: example.com\r\n

\r\n

[Full request URI: http://example.com/index.html]

[HTTP request 1/1]

[Response in frame: 1643]

Wireshark as in Task 1

0000	00 04 00 01 00 06 08 00	27 9b e8 eb 00 00 08 00
0010	45 10 00 2a 3a cb 40 00	40 06 be 1d 0a 00 03 0f	E...: 0 0
0020	5d b8 d7 0e c5 b8 00 56	7a 49 24 cc 15 ae c8 02]....P ZIS
...

The differences are there are two different HTTP versions used with Telnet using 1.0 and the browser using 1.1. The telnet is also missing the User-Agent field, the Accept, the Accept-Language, Accept-Encoding, Connection, and Upgrade-Insecure-Requests fields.

3. A screenshot of the HTTP Response message in Wireshark that the server

1642	219.085952547	93.184.215.14	10.0.3.15	HTTP	1673 HTTP/1.0 200 OK (text/html)
1890	306.436673129	10.0.3.15	93.189.91.49	HTTP	143 GET / HTTP/1.1
1892	306.464781383	93.189.91.49	10.0.3.15	HTTP	245 HTTP/1.1 204 No Content
2229	606.499859953	10.0.3.15	185.125.190.49	HTTP	143 GET / HTTP/1.1
2231	606.592434655	185.125.190.49	10.0.3.15	HTTP	245 HTTP/1.1 204 No Content

Frame 1643: 1673 bytes on wire (13384 bits), 1673 bytes captured (13384 bits) on interface any, id 0					
Linux cooked capture v1					
Ethernet Protocol Version 4, Src: 93.184.215.14, Dst: 10.0.3.15					
Transmission Control Protocol, Src Port: 80, Dst Port: 50608, Seq: 1, Ack: 48, Len: 1617					
Hypertext Transfer Protocol					
HTTP/1.0 200 OK\r\n					
Age: 441628\r\n					
Cache-Control: max-age=604800\r\n					
Content-Type: text/html; charset=UTF-8\r\n					
Date: Tue, 14 May 2024 21:04:16 GMT\r\n					
Etag: "3147526947vgzipidnt"\r\n					
Expires: Tue, 21 May 2024 21:04:16 GMT\r\n					
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT\r\n					
Server: ECAcc (chd/0770)\r\n					
Vary: Accept-Encoding\r\n					
0000	00 00 00 01 00 06 52 54	00 12 35 02 64 00 08 00RT	..5 d...	
0010	45 00 06 79 3b 7a 00 00	40 06 f7 2f 5d b8 d7 0e	E..y;Z..@..:/]...		
0020	0a 00 03 0f 00 50 c5 b0	15 ae c8 02 7a 49 24 ceP....21\$		
0030	50 18 ff ff 4b 41 00 00	48 54 54 50 2f 31 2e 30	P...HA...HTTP/1.0		
0040	20 32 30 30 20 4f 4b 00	0a 41 67 65 3a 20 34 34	...OK...Age: 44		

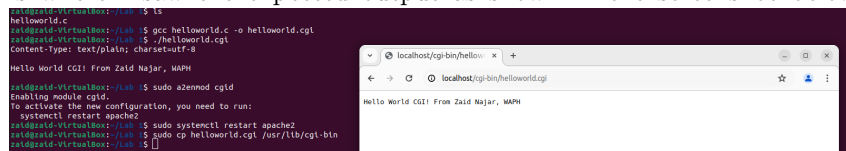
responded to your request.

The differences are that the Telnet uses another HTTP version. It is also missing the fields: Connection Handling, Transger Encoding, Content-Encoding, and other headers such as Cache-Control, ETag, Set-Cookie, etc.

Part II - Basic Web Application Programming

Task 1. CGI Web applications in C

- Summarize how you developed a Hello World CGI program in C and compiled and deployed the program on the web server. I created a new file called helloworld.c that was a simple print statement. I compiled it using gcc to turn it into a .cgi file. I then configured the Apache web server. I then copied the CGI program to the cgi-bin directory and then navigated to the given link. This is where I saw the expected output as shown in the screenshot below.



- Summarize and demonstrate with a screenshot that you can write another C CGI program and deploy it with a simple HTML template provided on <https://www.w3schools.com/html/> with proper title, heading, and paragraph, i.e., the course and your information should be there. I copied the earlier helloworld file and repeated the exact steps as above. This time, I just changed the code up a bit following a template. Here is the source code from index.c as well as a screenshot of the output.

Included file `index.c`:

```
#include <stdio.h>
```

```
int main(void) {
```

```

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

printf("<!DOCTYPE html>\n");

printf("<html>\n");

printf("<head>\n");

printf("<title>WAPH- Zaid Najar</title>\n");

printf("</head>\n");

printf("<body>\n");

printf("<h1>Welcome to WAPH! I am Zaid Najar.</h1>\n");

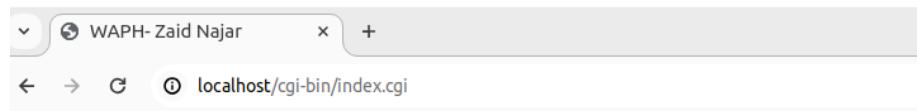
printf("<p> I am a Computer Science Major, class of 2025.</p>\n");

printf("</body>\n");

printf("</html>\n");

return 0;
}

```



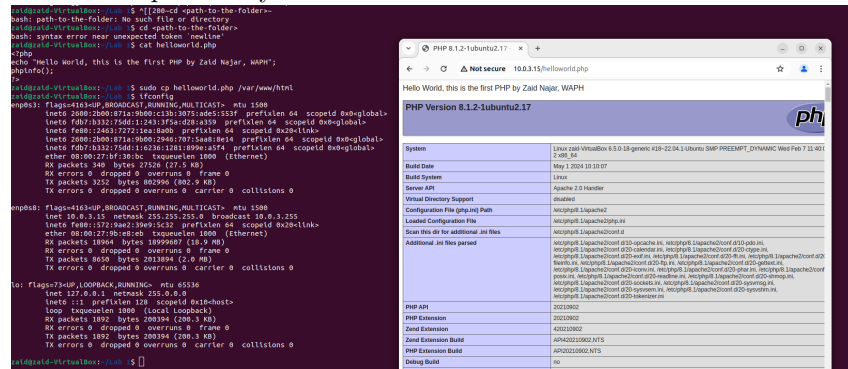
Welcome to WAPH! I am Zaid Najar.

I am a Computer Science Major, class of 2025.

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

- a. Summarize and demonstrate with a screenshot that you have successfully developed a simple helloworld.php PHP page with your name and PHP configuration as guided in Lecture 3. I installed php and followed the instructions given in the lecture. I

created a new file called helloworld.php and then configured it to show up on my browser as shown in the screenshot below.



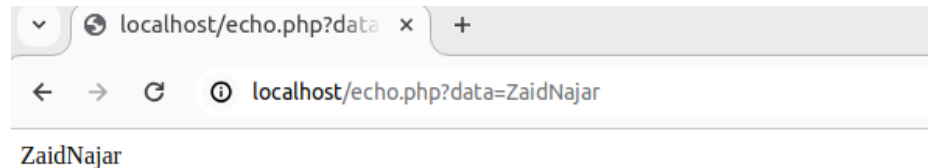
- b. Demonstrate that you developed and deployed an echo Web application in PHP, e.g., echo.php with a screenshot with your name in the data. I created a new file called echo.php and deployed it the same as the previous file. I then made data equal my name.

Included file echo.php:

```
<?php
```

```
echo $_REQUEST["data"];
```

```
?>
```



Screenshot of output:

Security Risks: There are a few risks such as URL manipulation which allows users to manipulate the URL to add unintended data. There is also a lack of input validation which could lead to errors or unwanted access from other users.

Task 3. Understanding HTTP GET and POST requests.

- a. Briefly describe how you used Wireshark to examine the HTTP GET Request and Response for the echo.php page with your name in the data. I used Wireshark by running it as I did in the earlier tasks,

navigating to my echo.php in the browser, and filtering by HTTP.

```

Hypertext Transfer Protocol
  GET /echo.php?data=ZaidNajar HTTP/1.1\r\n
  Host: localhost\r\n
  Connection: keep-alive\r\n
  sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"\r\n
  sec-ch-ua-mobile: ?0\r\n
  sec-ch-ua-platform: "Linux"\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
  Sec-Fetch-Site: none\r\n
  Sec-Fetch-Mode: navigate\r\n
  Sec-Fetch-User: ?1\r\n
  Sec-Fetch-Dest: document\r\n
  Accept-Encoding: gzip, deflate, br, zstd\r\n
  Accept-Language: en-US,en;q=0.9\r\n
  \r\n
  [Full request URI: http://localhost/echo.php?data=ZaidNajar]
  [HTTP request 1/1]
  [Response in frame: 112]

Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Tue, 14 May 2024 22:03:29 GMT\r\n
  Server: Apache/2.4.52 (Ubuntu)\r\n
  Content-Length: 10\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: 0.002268055 seconds]
  [Request in frame: 110]
  [Request URI: http://localhost/echo.php?data=ZaidNajar]
  File Data: 10 bytes
  Line-based text data: text/html (1 lines)

```

- b. Summarize using curl to create an HTTP POST request with your name in the data. I install curl then used the command as instructed with my data input to receive the POST request. The two screenshots below show the request using curl and the request from the HTTP Stream.

```

zaid@zaid-VirtualBox:~$ sudo snap install curl
[sudo] password for zaid:
curl 8.1.2 from Wouter van Bommel (woutervb) installed
zaid@zaid-VirtualBox:~$ curl -X POST http://localhost/echo.php -d "data=ZaidNajar"
ZaidNajar
zaid@zaid-VirtualBox:~$

```

```
GET /echo.php?data=ZaidNajar HTTP/1.1
Host: localhost
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9

HTTP/1.1 200 OK
Date: Tue, 14 May 2024 22:03:29 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Length: 10
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

ZaidNajar
```

- c. Compare the similarity/difference between HTTP POST Request and HTTP GET Request and between the two HTTP Responses above. In the request method, GET adds data to the URL while the POST sends it in the request body. For data visibility, you can see the data in the URL in GET, but in POST the data is in the request body. GET is less secure while post is more secure. GET is for getting data while POST is for sending data to be processed.