Name: Suhan.B.Revankar

SRN: PES2UG19CS412

Week number: 3

Date: 14/2/2021

Name of the experiment: Understand the working of HTTP Headers

---

Objective: The objective here is to understand the working of HTTP Headers.

Understand working of HTTP headers:

Conditional Get: If-Modified-Since

HTTP Cookies: Cookie and Set-Cookie

Authentication: Auth-Basic

Design a web page that has one embedded page (e.g. image) and sets a cookie and enables authentication. You are required to configure the web server (e.g. apache) with authentication mechanism.

Show the behavior of conditional get when embedded objects is modified and when it is not (you can just change the create date of the embedded object). Decode the Basic-Auth header using Base64 mechanism as per the password setup.

Observation: Show the behavior of browser when is cookie is set and when cookie is removed.

Understanding the Working of HTTP Headers

Question: Understand the working of HTTP headers Conditional Get: If-Modified-Since HTTP Cookies: Cookie and Set-Cookie Authentication: Auth-Basic

Design a web page that has one embedded page (e.g. an image) and sets a cookie and enables authentication. You are required to configure the webserver (e.g. apache) with an authentication mechanism. Show the behaviour of conditional get when embedded objects are modified and when it is not (you can just change the create date of the embedded object). Decode the Basic- Auth header using Base64 mechanism as per the password setup.

Observation: Show the behaviour of browser when is a cookie is set and when a cookie is removed.

Solution: Analyzing Basic Authentication and Cookies

The three parts of the experiment are:

1. Password Authentication

2. Cookie Setting

3. Conditional get

Password Authentication

1.

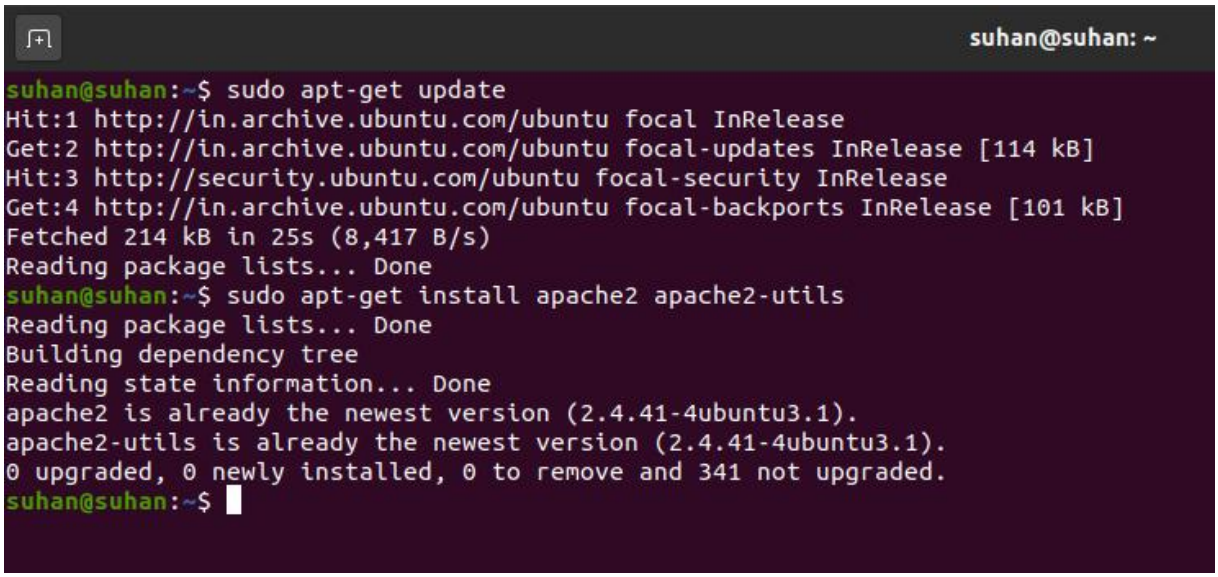
i) The commands below are executed on the terminal.

```
sudo apt-get update
```

This command just installs any missing packages.

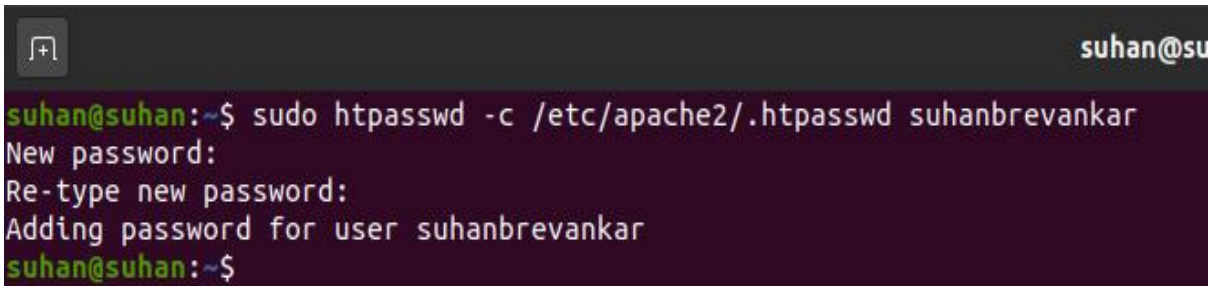
```
sudo apt-get install apache2 apache2-utils
```

The usage of this command suffices the need for a password authentication.



ii) To secure the network with a password, the following command was used:

```
sudo htpasswd -c /etc/apache2/.htpasswd
suhanbrevenkar
```



iii) The network name is now fixed to, `suhanbrevenkar` and is secured with a verified password.

For taking a glance at the network the following command is fruitful:

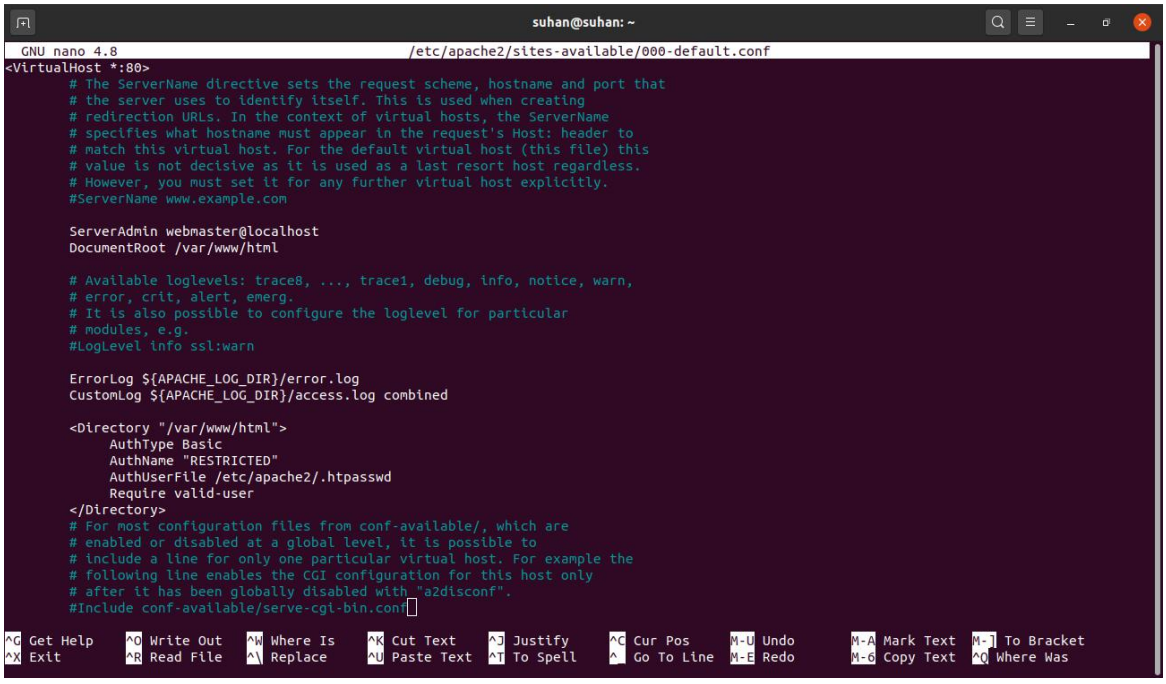
```
sudo cat /etc/apache2/.htpasswd
```

```
suhan@suhan:~$ sudo cat /etc/apache2/.htpasswd
suhanbrevankar:$apr1$DOpXfIRm$Bsuq4DEEXjy630J02njcL0
suhan@suhan:~$
```

2. To set up the authentication phase, the following commands are executed. Configuring access control within the Virtual Host Definition.

Since the file was changed in the text editor, the command that was used to perform this action is:

```
gedit /etc/apache2/sites-available/000-default.conf
```



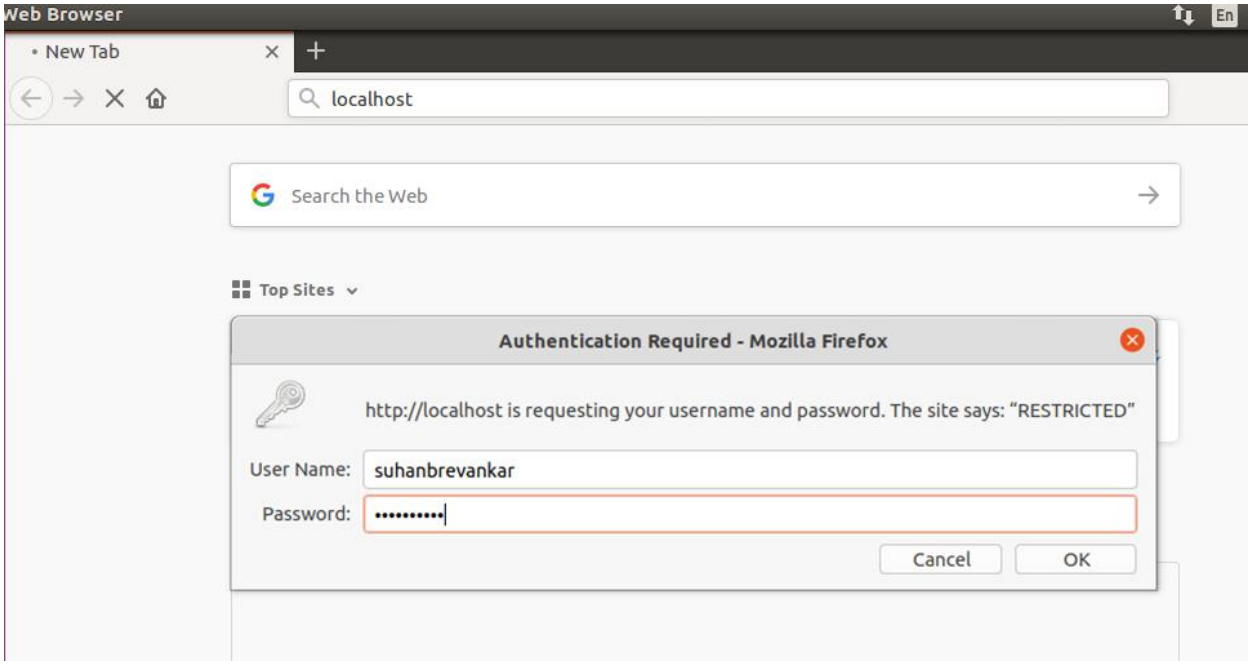
3. The password must now be implemented, henceforth, the apache service is ought to be restarted.

   This command performs the action required: `sudo service apache2 restart`

```
suhan@suhan:~$ sudo service apache2 restart
suhan@suhan:~$
```

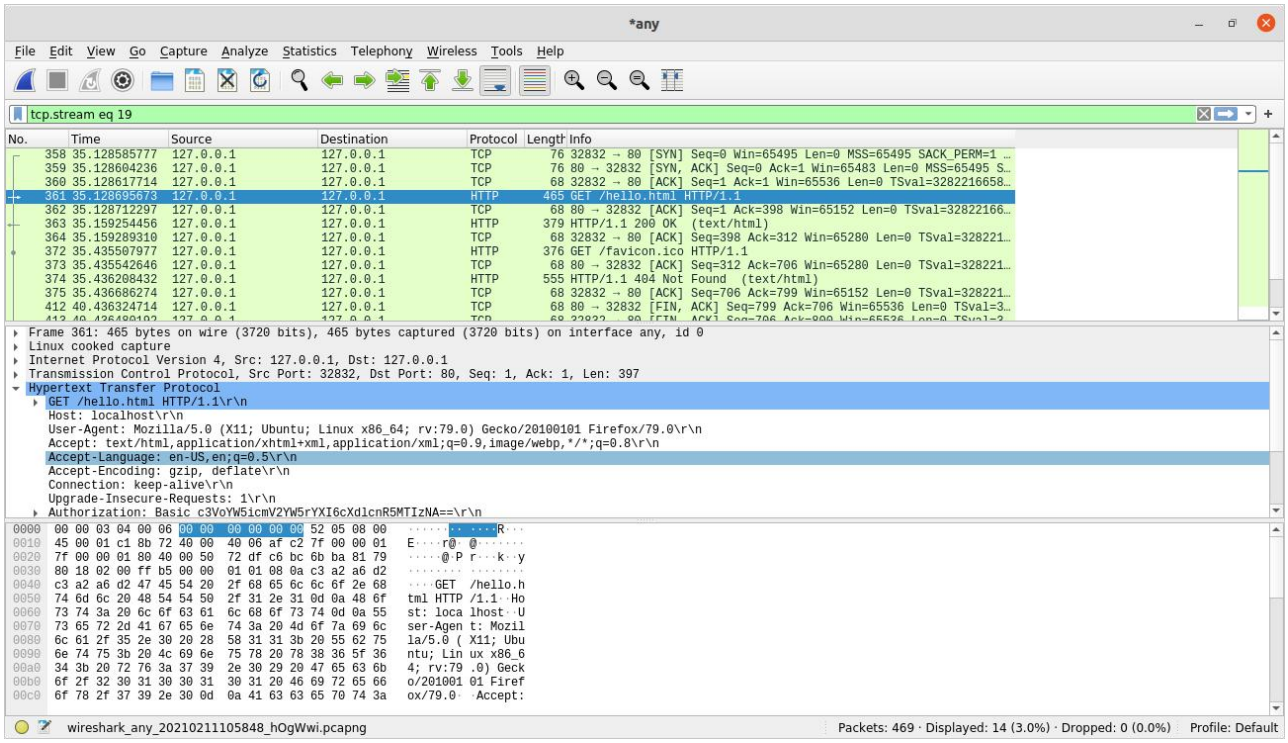
The apache2 restart service program successfully completed its job, since there are no errors.

Now, the network can be tested for security.

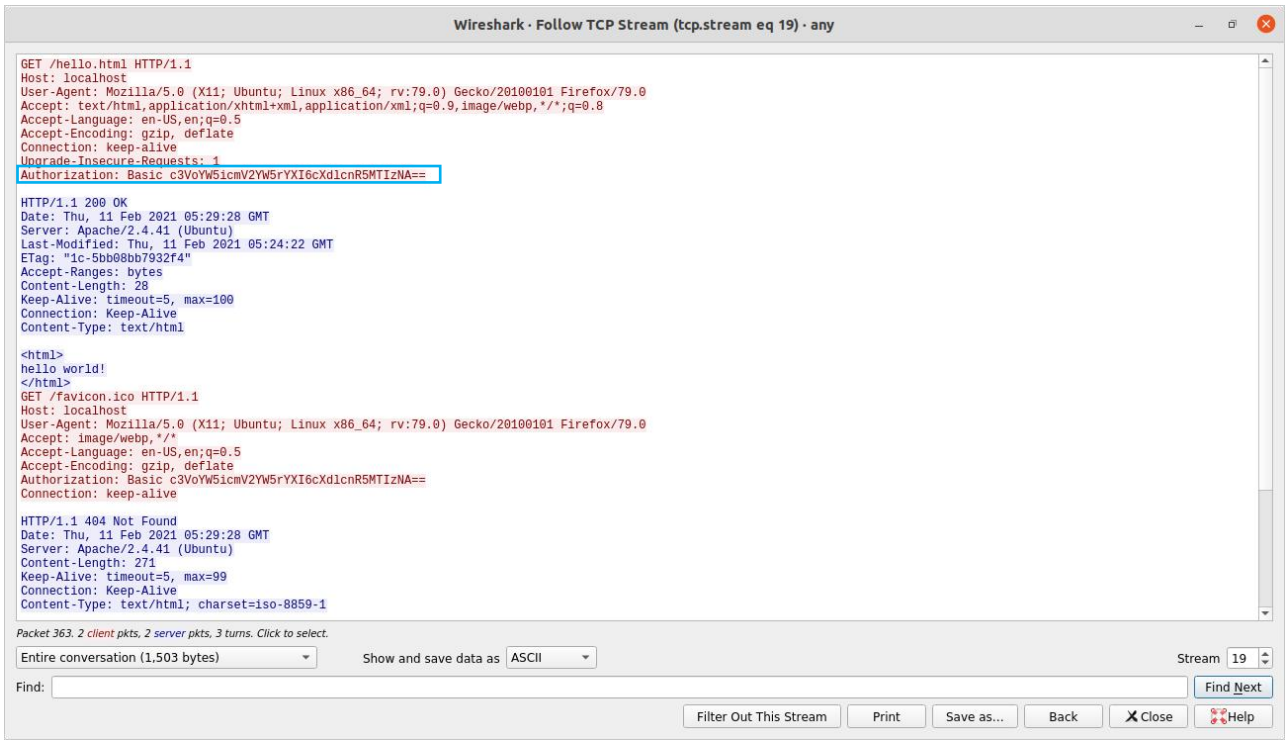


The help of Wireshark is taken to capture the packets.

Since we are accessing the file, index.html over localhost, the IP address of the lo interface is reflected with no doubt at all.



Here, it’s the ipv6 address that was reflected.



Now, on a thorough glance at these results of following a TCP stream on `GET /HTTP/1.1` what can be observed?

The localhost is successfully secured with a password.

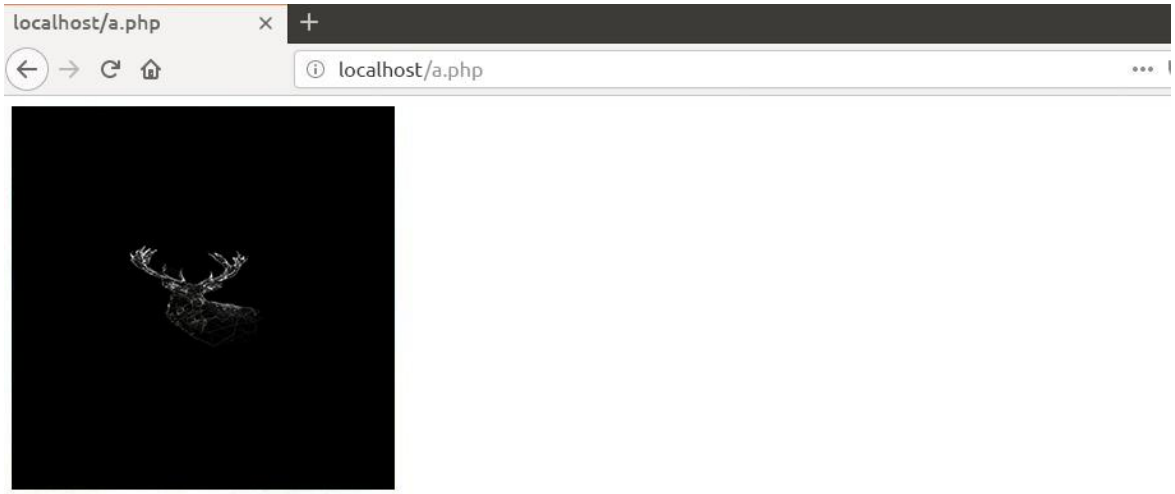
Deeply looking at it, what's that bizarre-looking word beside the "`Authorization`" field?

It's the password that was set to keep the network secured. It's encoded in a format coined as ***Base64*.** This shall be decoded in a later phase. The Base64 code word is, c3VoYW5icmV2YW5rYXI6cXdlcnR5MTIzNA==

Cookie Setting

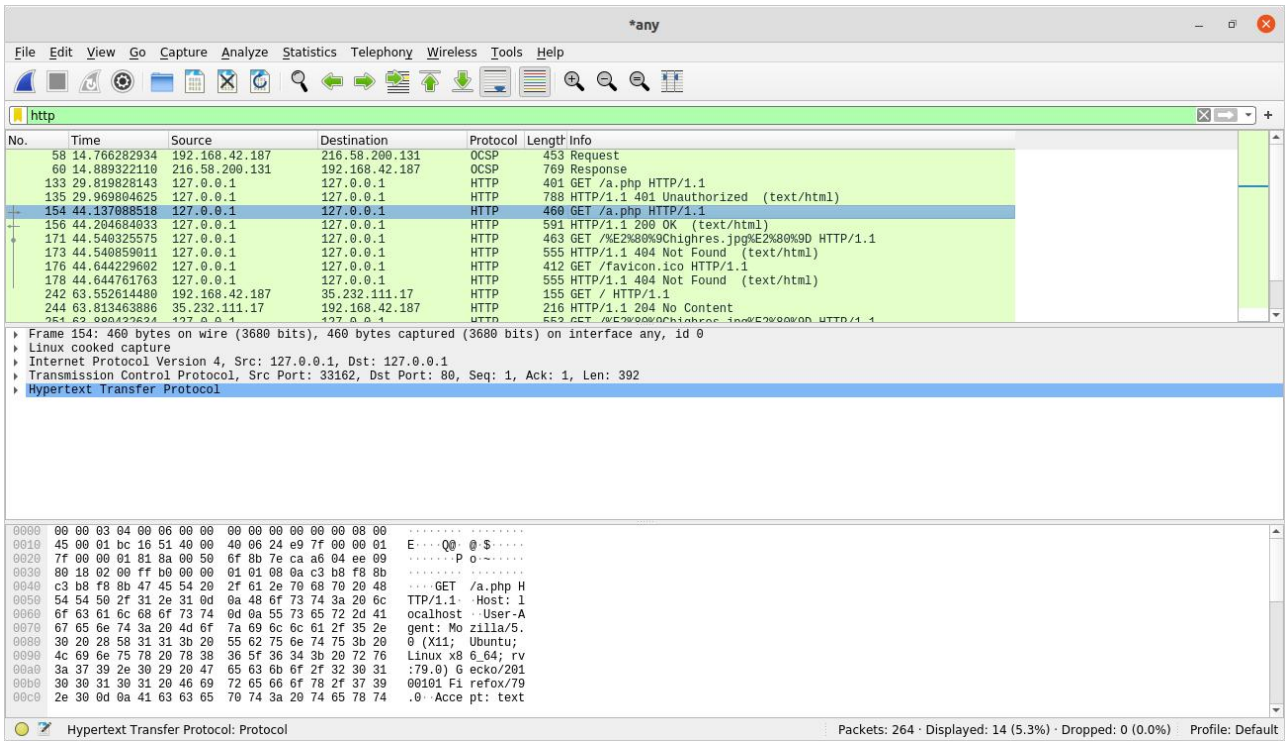
1. A PHP file is created to set the cookie.

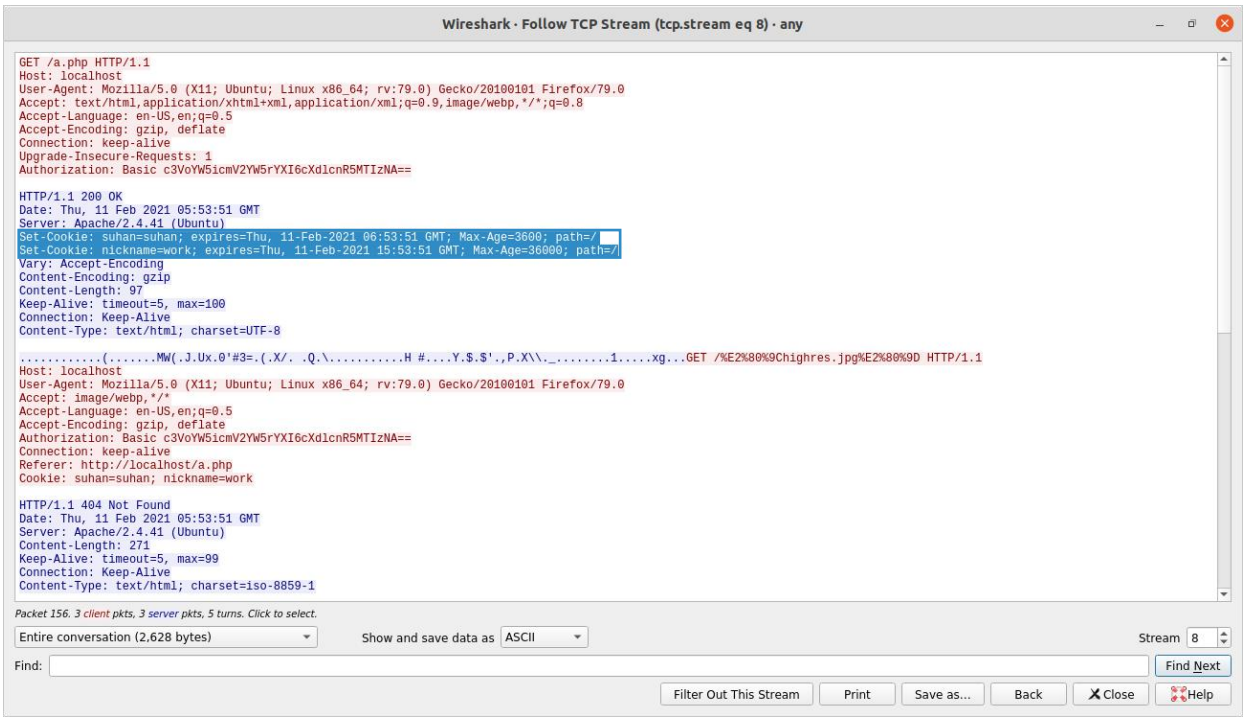
2. The file was saved under the HTML directory in the path `/var/www/html`



That’s the webpage.

3. The packets are captured using Wireshark and using the “follow TCP stream” which checks for the set-cookie field whether the cookie is set or not set.





Henceforth, with no further qualm, it’s been solemnly concluded that the cookie has been set.

Now, the Base64 encrypted codeword is to be decoded.

This is the algorithm:

1. Take down the codeword as it is.
2. Write down the appropriate corresponding number of each character. Use this table for reference:

Upper case alphabets	
Index Number	Character
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z

Lower case alphabets	
Index Number	Character
26	a
27	b
28	c
29	d
30	e
31	f
32	g
33	h
34	i
35	j
36	k
37	l
38	m
39	n
40	o
41	p
42	q
43	r
44	s
45	t
46	u
47	v
48	w
49	x
50	y
51	z

Digits	
Index Number	Digit
52	0
53	1
54	2
55	3
56	4
57	5
58	6
59	7
60	8
61	9

Symbols	
Index Number	Symbol
62	+
63	/

= (equals) symbol shall not be assigned an index number

3. Convert each value to its corresponding binary value, and extend up till six bits.
4. Combine all the bits and split them into eight bits (one byte).
5. Convert each byte to it's corresponding ASCII value. That's the decoded result.

Here's how c3VoYW5icmV2YW5rYXI6cXdlcnR5MTIzNA== was decoded.

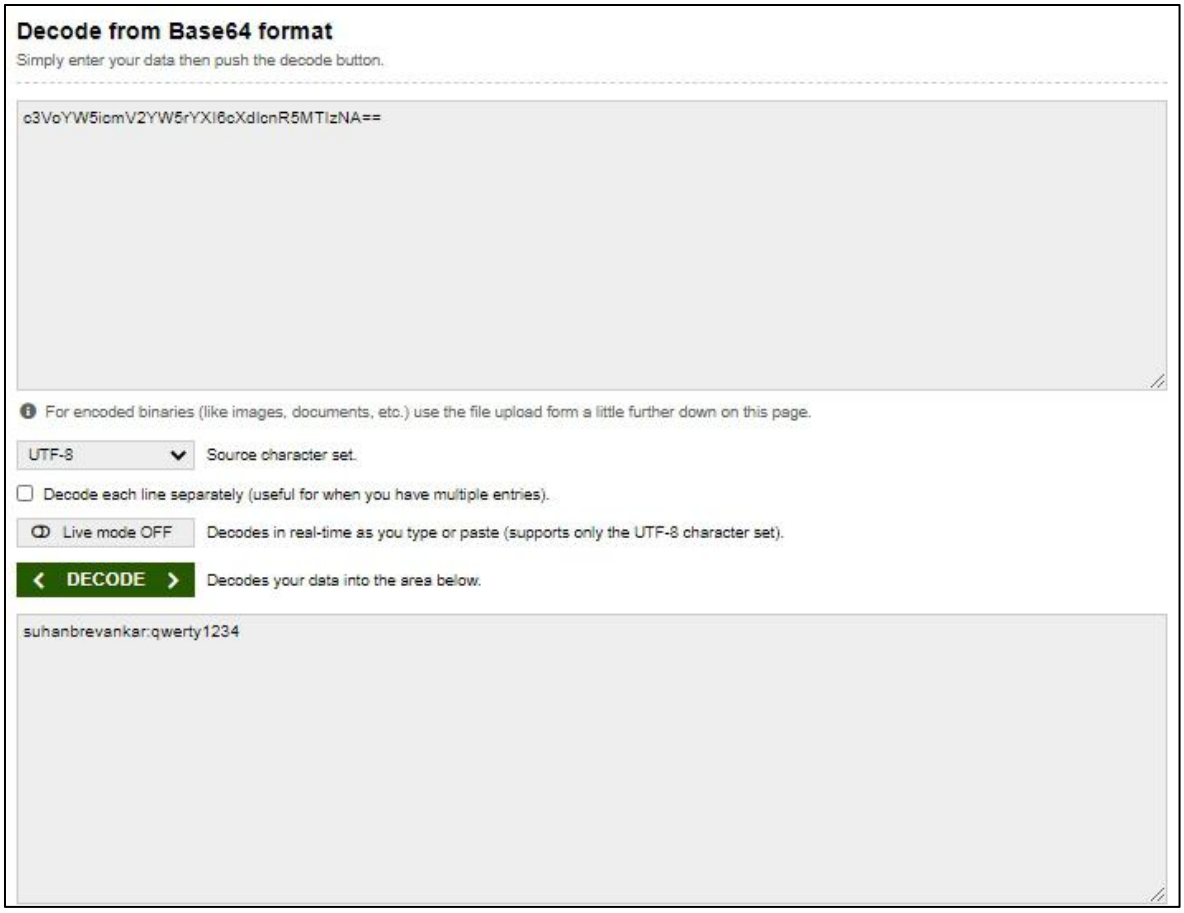
	A	B	C	D	E	F	G
1	**BASE WORD**	**DECIMAL VALUE**	**BINARY VALUES (6 bits)**	**BINARY VALUE (combined to 1 byte)**	**ASCII CODE**	**CORRESPONDING CHARACTER**	
2	c	28	011100	01110011	115	s	
3	3	55	110111	01110101	117	u	
4	V	21	010101	01101000	104	h	
5	o	40	101000	01100001	97	a	
6	Y	24	011000	01101110	110	n	
7	W	22	010110	01100010	98	b	
8	5	57	111001	01110010	114	r	
9	i	34	100010	01100101	101	e	
10	c	28	011100	01110110	118	v	
11	m	38	100110	01100001	101	e	
12	V	21	010101	01101110	110	n	
13	2	54	110110	01110101	107	k	
14	Y	24	011000	01100001	97	a	
15	W	22	010110	01110010	114	r	
16	5	57	111001	00111010	58	:	
17	r	53	110101	01110001	113	q	
18	Y	24	011000	01110111	119	w	
19	X	23	010111	01100101	101	e	
20	I	8	001000	01110010	114	r	
21	6	58	111010	01110100	116	t	
22	c	28	011100	01111001	121	y	
23	X	23	010111	00110001	49	1	
24	d	29	011101	00110010	50	2	
25	l	37	100101	00110011	51	3	
26	c	28	011100	00110100	52	4	
27	n	39	100111	000000			
28	R	17	010001				
29	5	57	111001				
30	M	12	001100				
31	T	19	010011				
32	I	8	001000				
33	z	51	110011				
34	N	13	001101				
35	A	0	000000				
36	=						
37	=			FINAL RESULT = suhanbrevenkar:qwerty1234			
38							
39							
40							
56							

Sheet1

When the bits are combined to form a byte, the last row which shows four zeroes shall be isolated from the 132-bit digit and would remain solitary as it was used to extend the total number of bits, making 128 bits divisible by six.

Hence the codeword was successfully decoded to, suhanbrevankar:qwerty1234.

The image below shows a small verification of this:



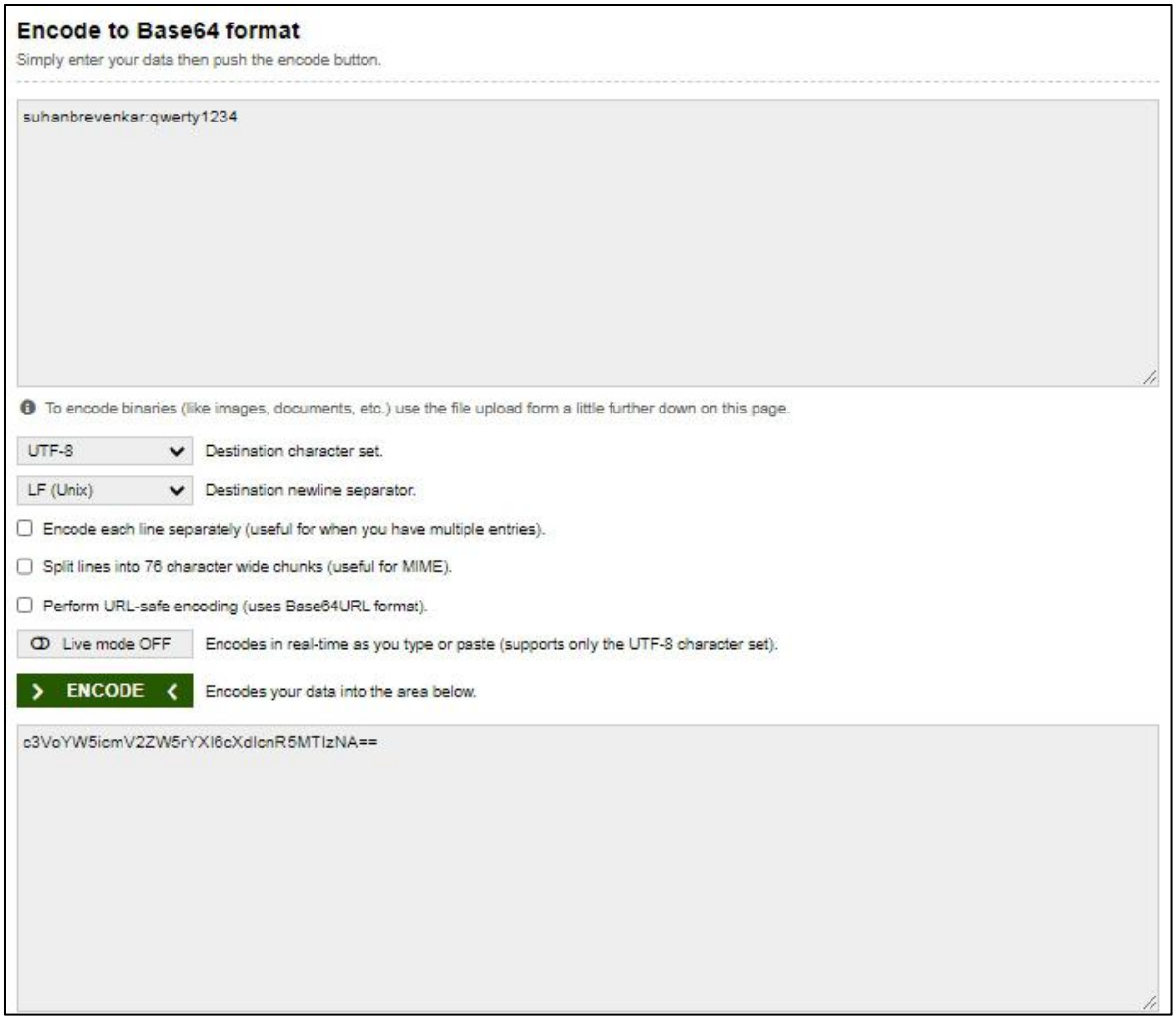
In a reversed fashion (Base64 → Plain text i.e., encoding):

If we reckon the same algorithm in an upside-down fashion, there are slight alterations to be done.

CORRESPONDING CHARACTER	ASCII CODE	BINARY VALUE (combined to 1 byte)	BINARY VALUES (6 bits)	DECIMAL VALUE	BASE WORD
s	115	01110011	011100	28	c
u	117	01110101	110111	55	3
h	104	01101000	010101	21	V
a	97	01100001	101000	40	o
n	110	01101110	011000	24	Y
b	98	01100010	010110	22	W
r	114	01110010	111001	57	5
e	101	01100101	100010	34	i
v	118	01110110	011100	28	c
e	101	01100001	100110	38	m
n	110	01101110	010101	21	V
k	107	01110101	110110	54	2
a	97	01100001	011000	24	Y
r	114	01110010	010110	22	W
:	58	00111010	111001	57	5
q	113	01110001	110101	53	r
w	119	01110111	011000	24	Y
e	101	01100101	010111	23	X
r	114	01110010	001000	8	I
t	116	01110100	111010	58	6
y	121	01111001	011100	28	c
1	49	00110001	010111	23	X
2	50	00110010	011101	29	d
3	51	00110011	100101	37	l
4	52	00110100	011100	28	c
		000000	100111	39	n
			010001	17	R
			111001	57	5
			001100	12	M
			010011	19	T
			001000	8	I
			110011	51	z
			001101	13	N
			000000	0	A
					=
	FINAL RESULT = c3VoYW5icmV2YW5rYXI6cXdlcnR5MTIzNA==				=

In the second row, the last column, we see additional four zeroes. This is for the very reason that the binary digit must be divisible by six (which are made solitary in decoding). In the last column, there's additional two equals' symbol. To make the length of the string divisible by four, the two equals were added.

And, just for a bijou verification:

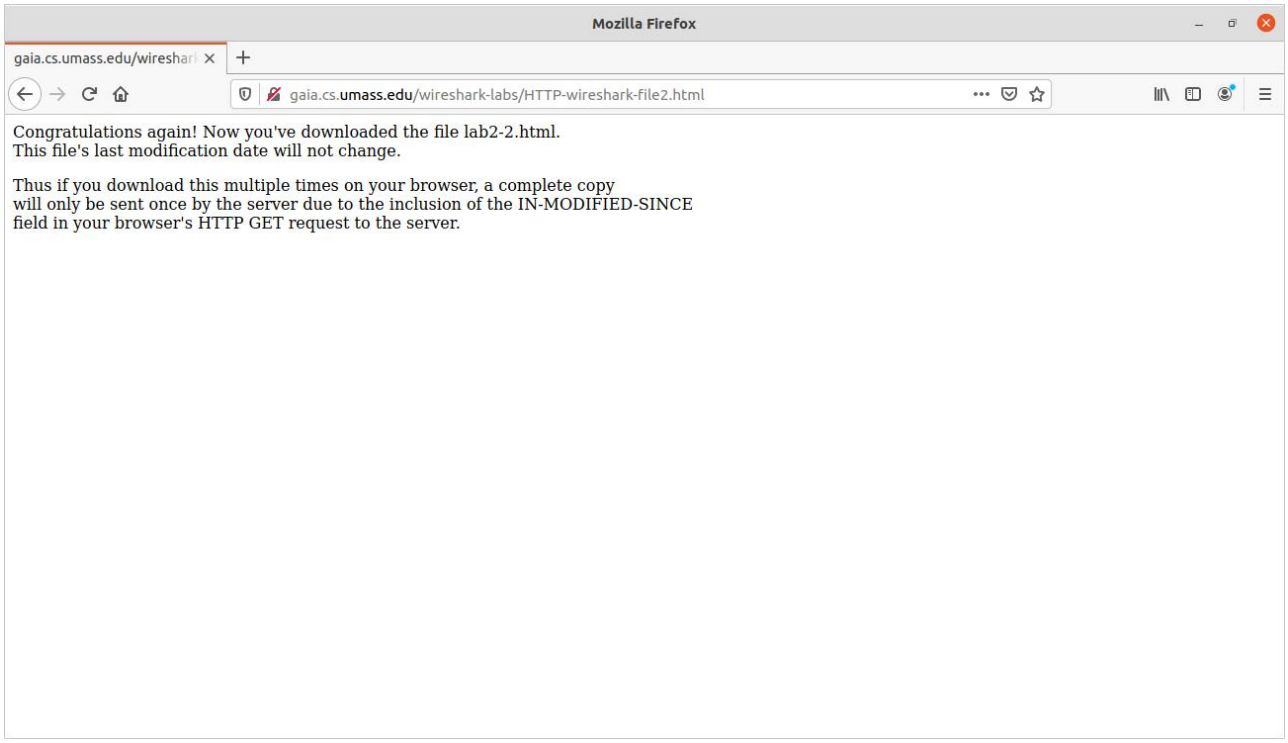


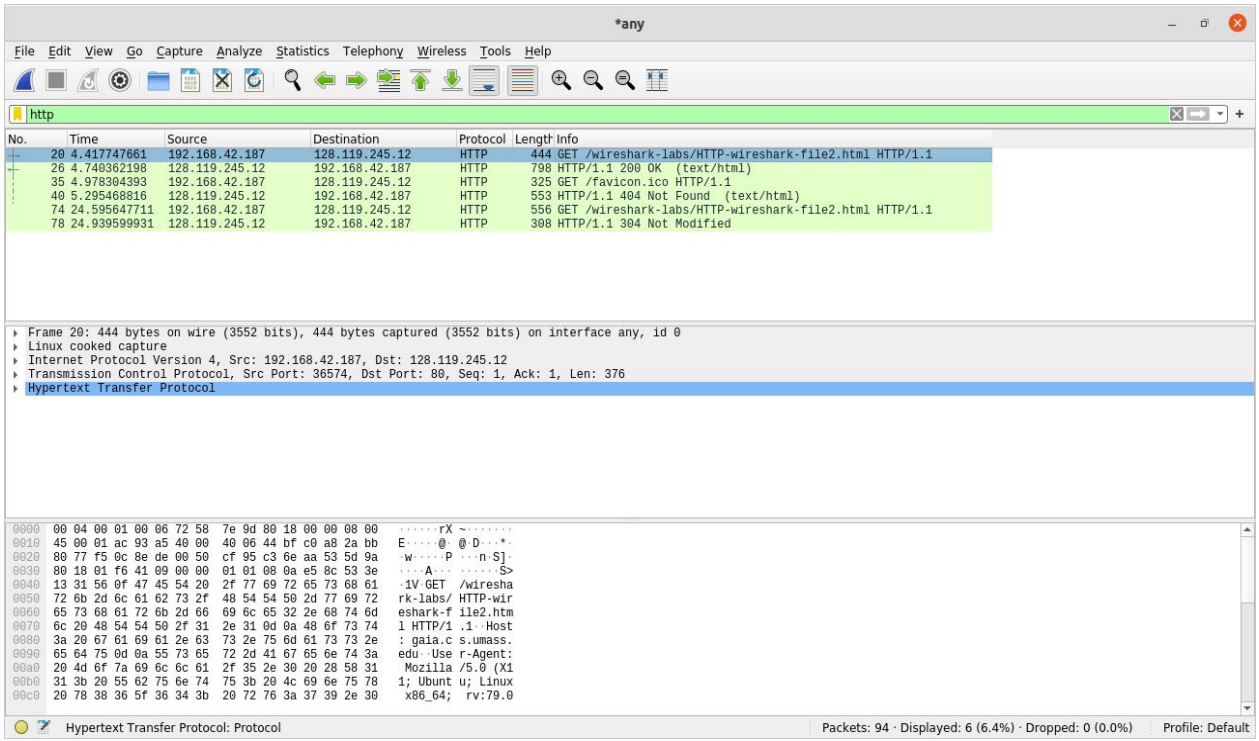
There are not many exclamatory results from the Wireshark capture. All that could be observed is that the cookies are set successfully. The set-cookie attribute under the HTTP header in the TCP stream manifests this.

Conditional Get: If-Modified-Since

Here, an attempt is being made to look for any modification in a file, which is monitored with the Wireshark tool.

First, the history cache of the browser was expunged. And then, the website was accessed.



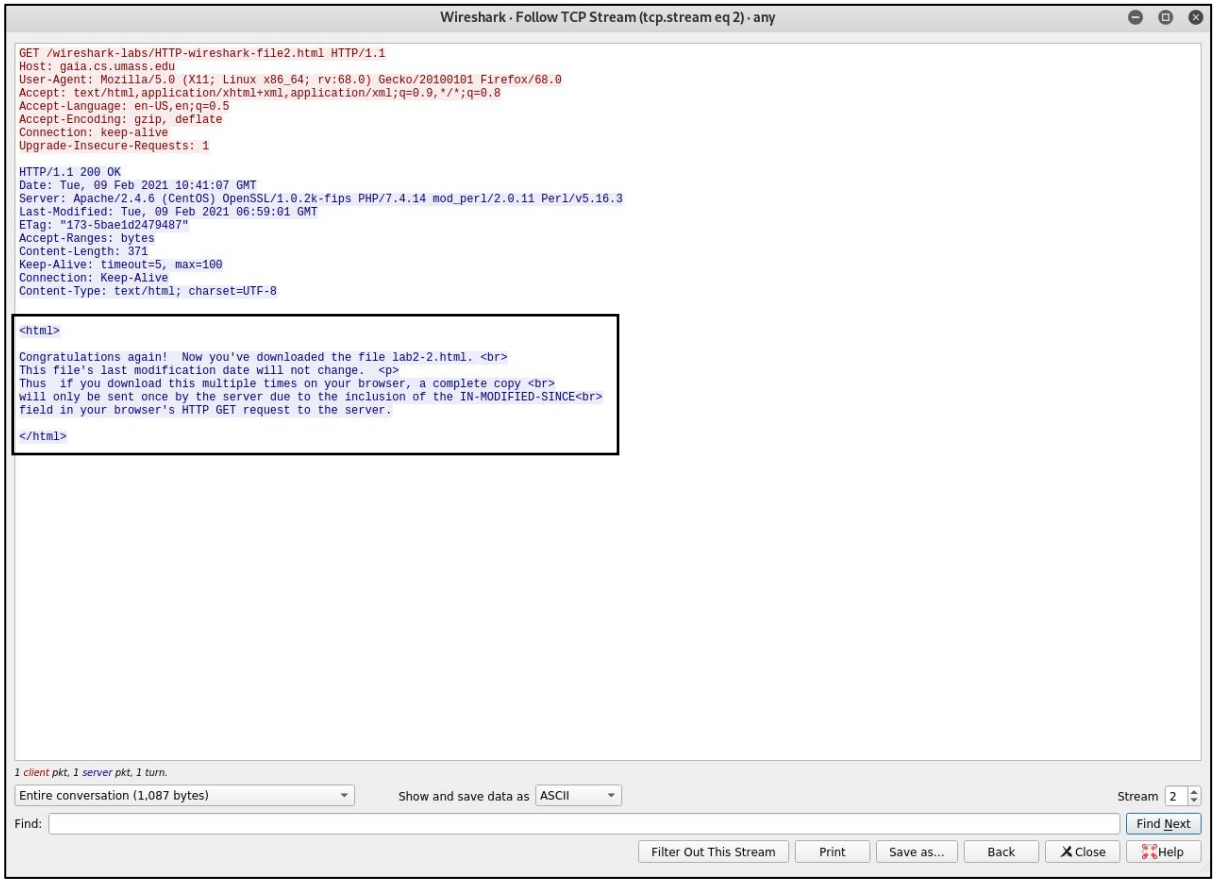


These were the results provided by the Wireshark tool.

This leads to some questions:

1. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?

Here’s the image of the TCP stream:



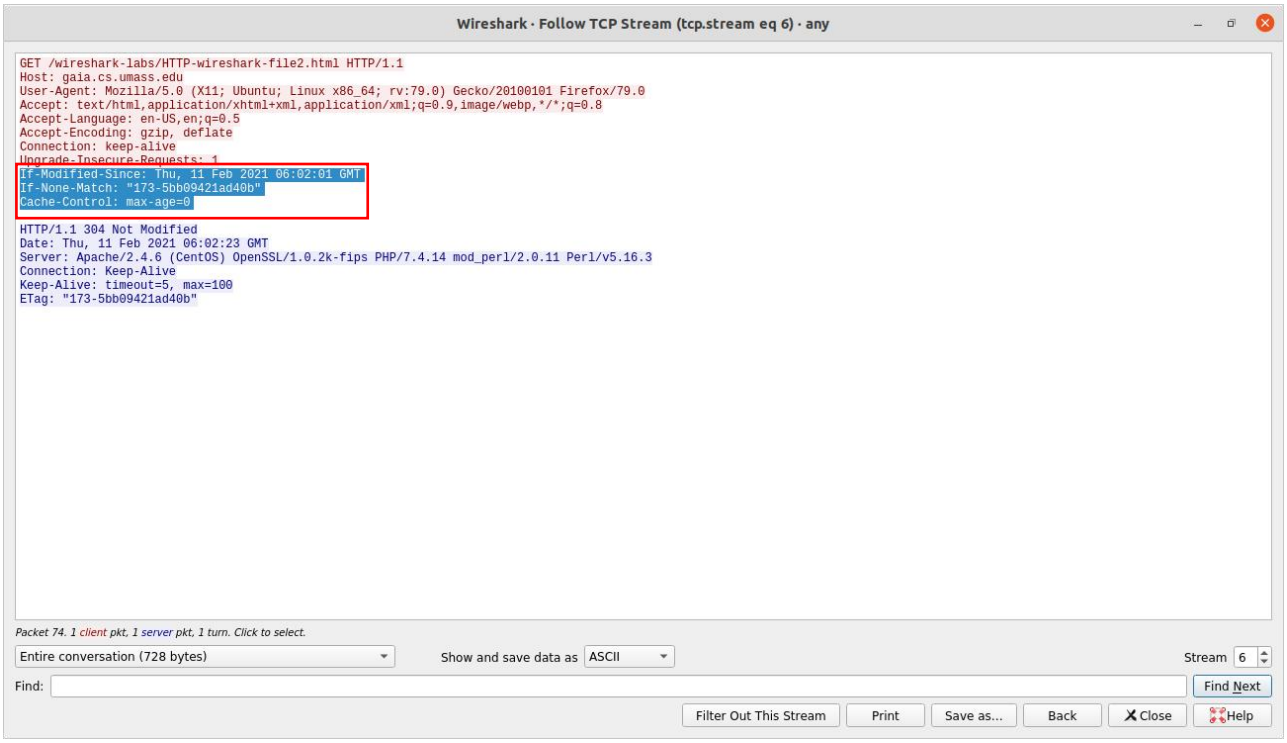
Alas, it’s a no. There’s no `If-modified-since` in the first HTTP GET request.

2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?

Undoubtedly, yes. The server did return the contents of the file. Under the HTTP response, we can see a bijou code, which manifests that the server explicitly returned the contents of the file.

3. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?

Taking a rapid glance at this image,



It's undoubtedly claimed that the "`IF-MODIFIED-SINCE:`" line is in the HTTP GET request. It shows the latest date and time that the user has visited the website. This is the information followed by the "`IF-MODIFIED-SINCE:`" header.

4. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

In response to the second HTTP GET request, the server returned a status code of `304` signifying the phrase, "`Not modified`". No, the server did not return the contents of the file. The file has been by no means modified and hence retrieved the old file from the browser's cache memory. The scenario would have been altered if the file had been modified. The server would have of course returned the contents of the file.

If the same experiment is repeated with a webpage that contains some images in it:

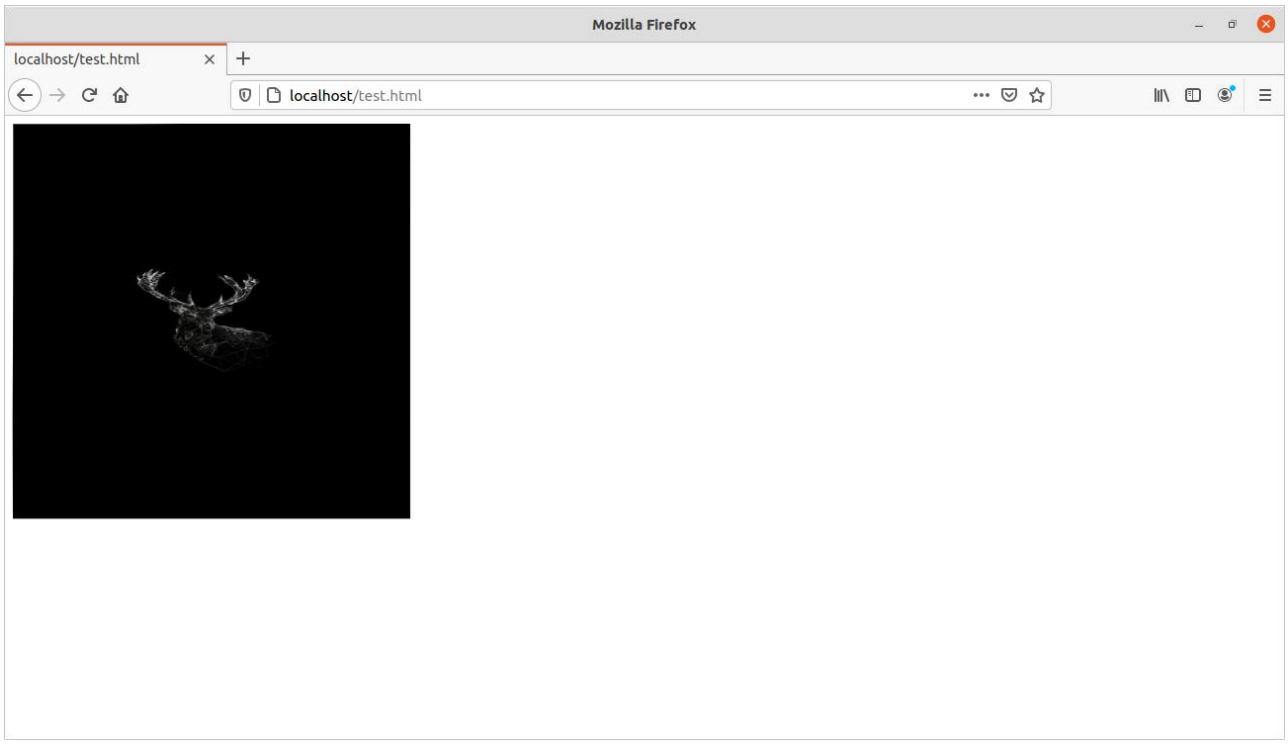
The HTML program:

Open *test.html /var/www/html Save

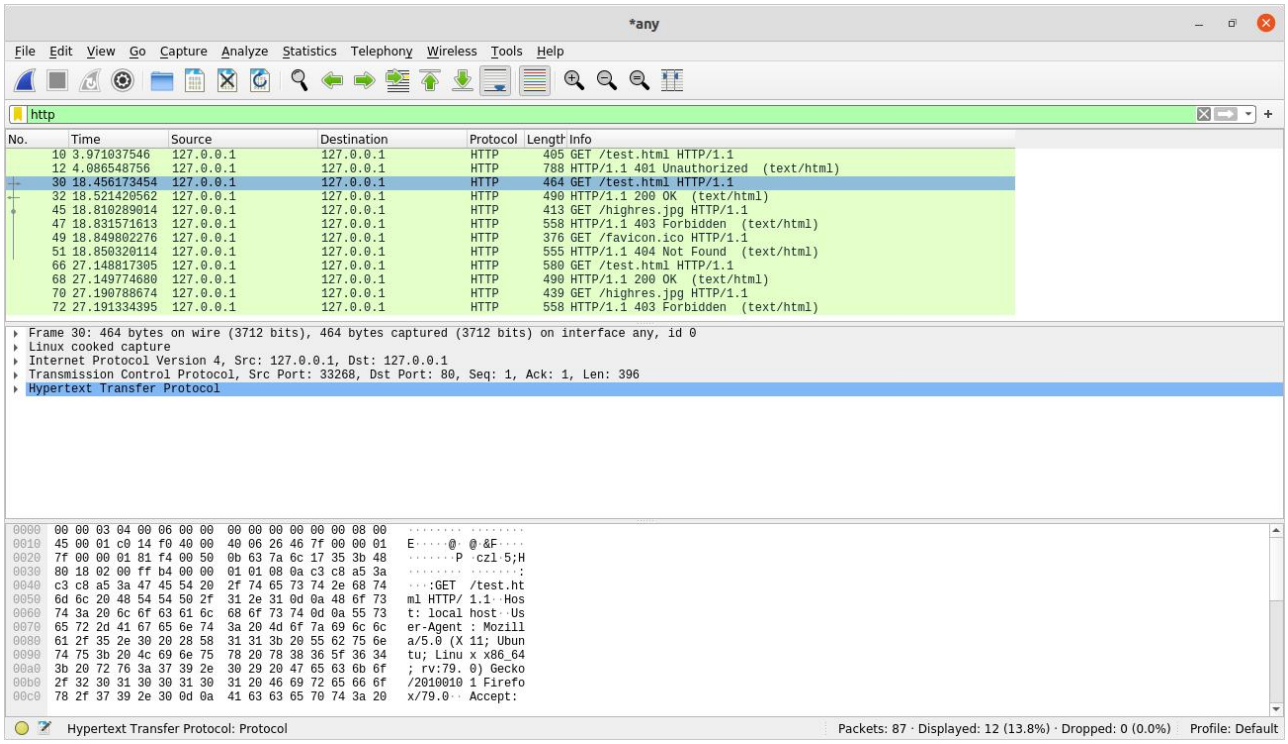
```
1 <!DOCTYPE html>
2 <html>
3 <image src="highres.jpg" height="400" width="400"/>
4 </html>
5
```

HTML Tab Width: 8 Ln 5, Col 1 INS

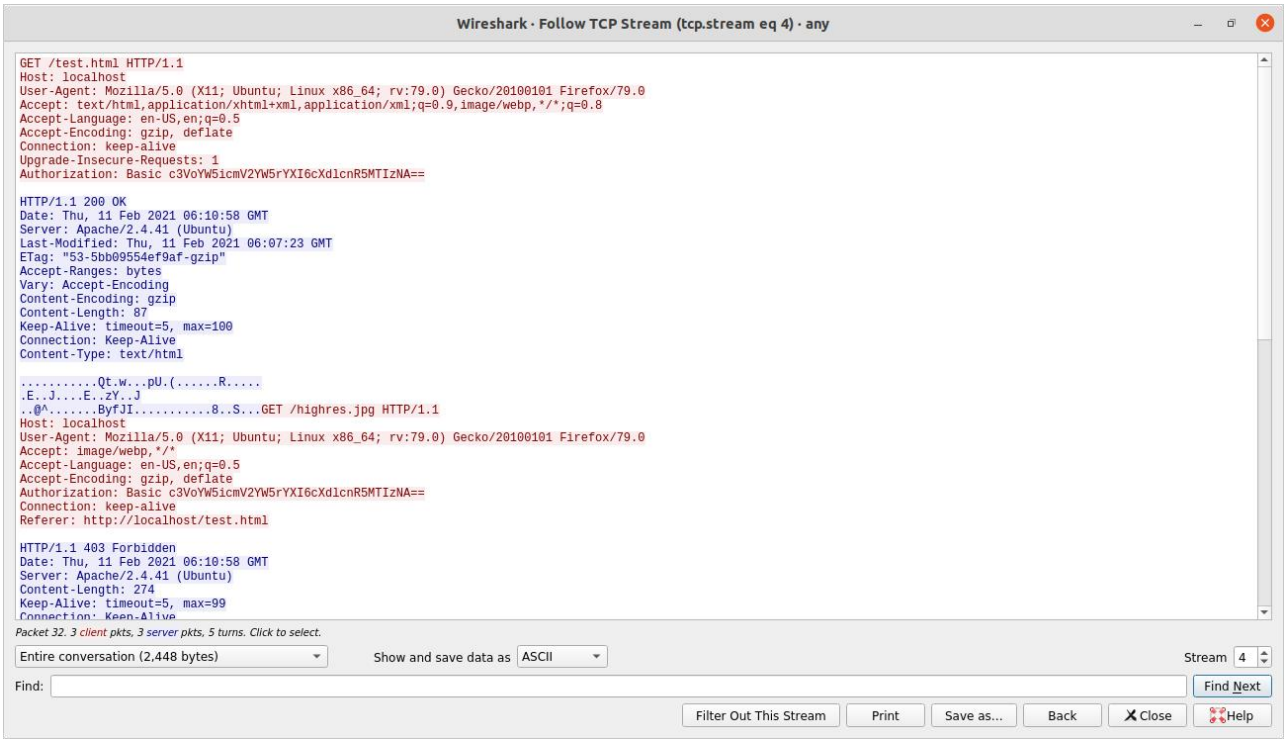
The webpage:



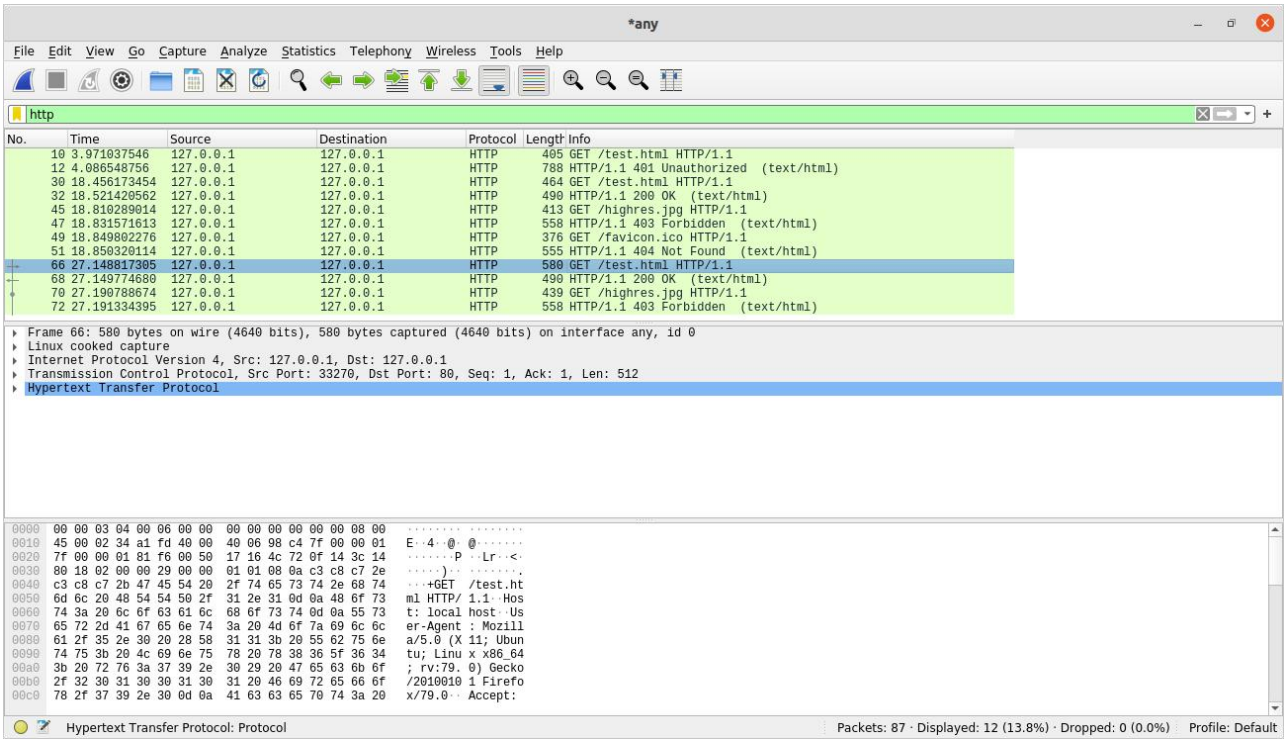
Wireshark results:

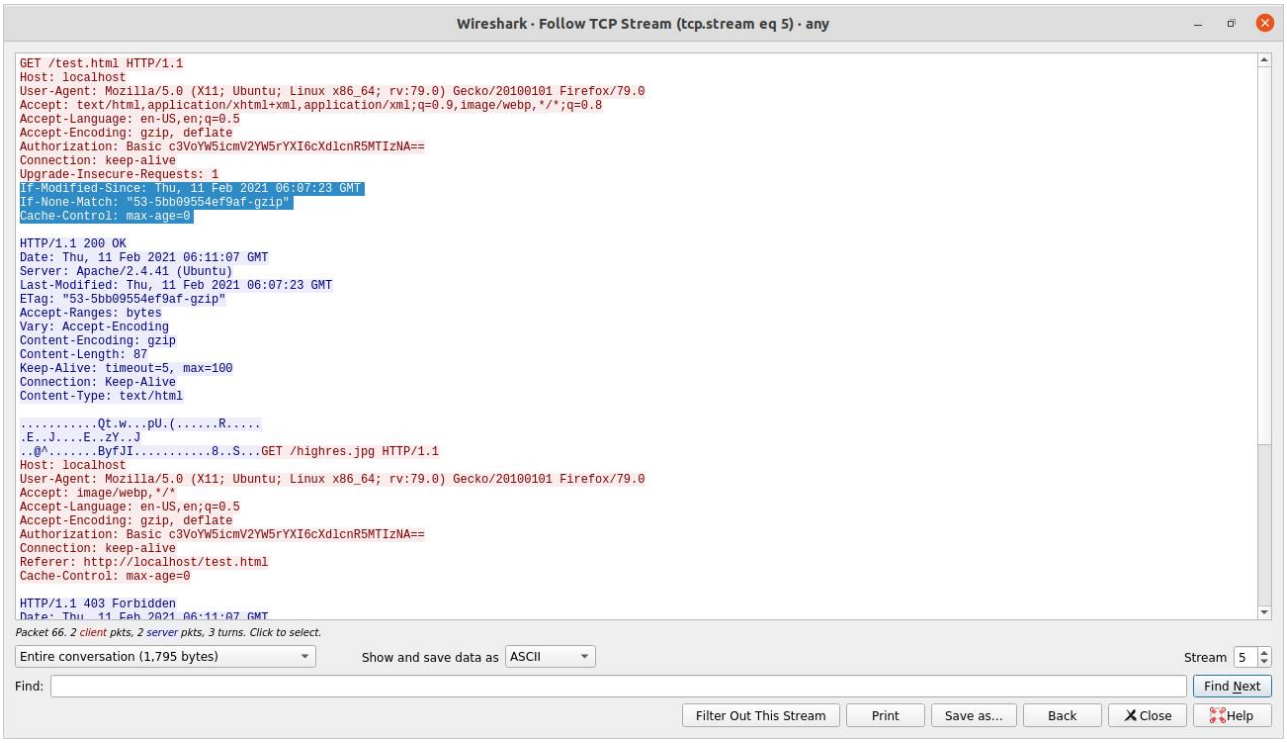


The first response:



There is an If-modified-since line and returned the details of the images.




Wireshark · Follow TCP Stream (tcp.stream eq 5) · any
GET /test.html HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Basic c3VoYW5icmV2YW5rYXI6cXdlcnR5MTIzNA==
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Thu, 11 Feb 2021 06:07:23 GMT
If-None-Match: "53-5bb09554ef9af-gzip"
Cache-Control: max-age=0
HTTP/1.1 200 OK
Date: Thu, 11 Feb 2021 06:11:07 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Thu, 11 Feb 2021 06:07:23 GMT
ETag: "53-5bb09554ef9af-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 87
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
..........Qt.W...pU.(......R.....
.E..J....E..zY..J
..@^.......ByfJI..........8..S...GET /highres.jpg HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Basic c3VoYW5icmV2YW5rYXI6cXdlcnR5MTIzNA==
Connection: keep-alive
Referer: http://localhost/test.html
Cache-Control: max-age=0
HTTP/1.1 403 Forbidden
Date: Thu, 11 Feb 2021 06:11:07 GMT
Packet 66. 2 client pkts, 2 server pkts, 3 turns. Click to select.
Entire conversation (1,795 bytes)
Show and save data as
ASCII
Stream
5
Find:
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

****