Computer Networks Lab

Week 3

Vishal R

PES1UG19CS571

I Section

February 13, 2021

Understanding the working of HTTP Headers

---

1 Password Authentication

1.1 Creating a Password

To have basic HTTP Authentications, we need to create a password file. This can be done by executing the command `$ sudo htpasswd -c /etc/apache2/.htpasswd PES1UG19CS571`. But before we begin, we need to install `apache2` server which will be required later in the experiment. `Apache2` can be installed by executing the command `$ sudo apt-get install apache2`.

Once `apache2` is installed the previous command is executed in the terminal. Upon executing the command, it asks us to enter a new password for the username `PES1UG19CS571`. This password is required to access any file on the `apache` server.

We can view the authentication by typing the command
`$ sudo cat /etc/apache2/.htpasswd` in terminal.

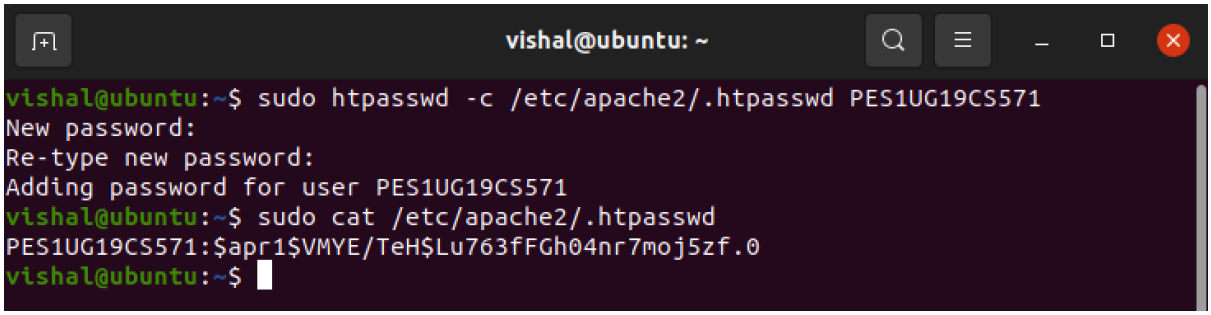

Figure 1: Shows creation of password for a new user for HTTP Authentications.

1.2 Configuring `Apache` to handle HTTP Authentications

To enable HTTP authentications, we need to change a few things in the `000-default.conf` file in apache server. We can open this file by typing the command `$ sudo gedit /etc/apache2/sites-available/000-default.conf` in terminal.

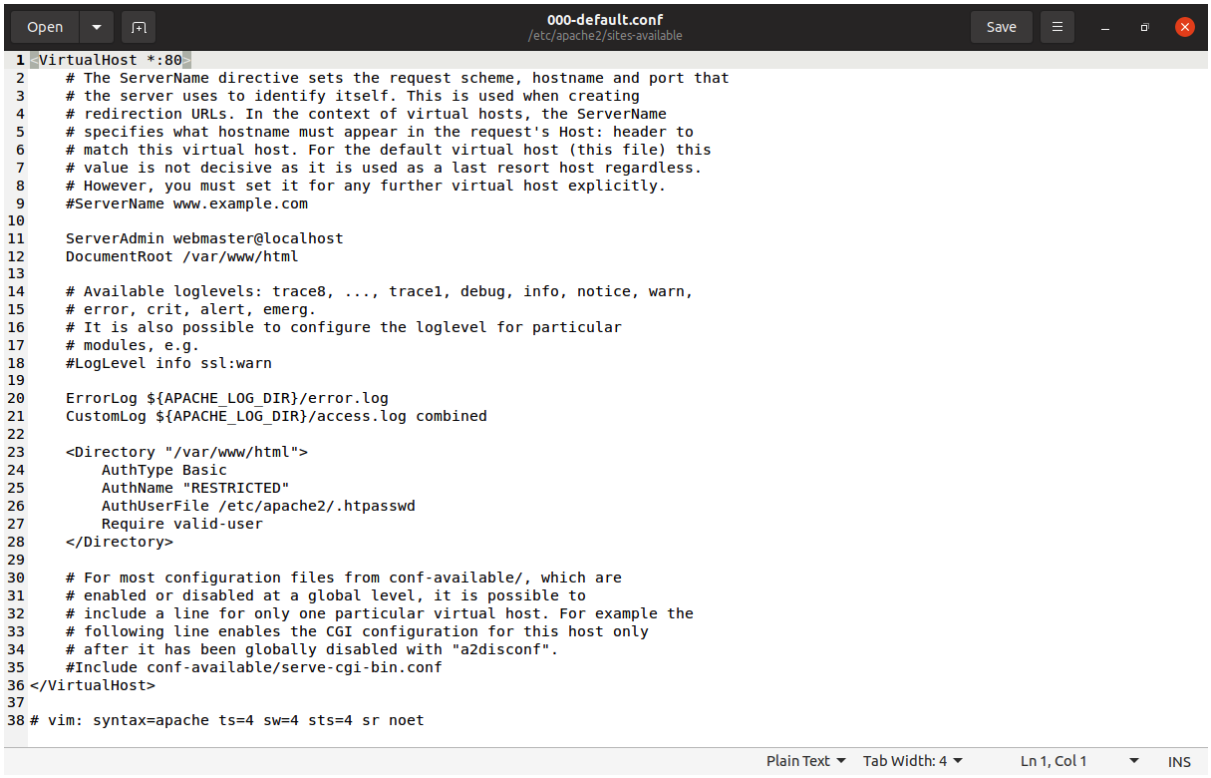

Figure 2: Shows changes made in `000-default.conf` file to make server support HTTP Authentications.

In the above image, lines 23 to 28 were added. These lines configures the server to provide access to its contents only after it receives an authentication from the user.

After all these steps, we need to restart the server so that it will use the new configuration settings that we have set. To restart the server we will type `$ sudo systemctl restart apache2` in terminal.

1.3 Accessing `localhost` in browser

To see if `apache2` server is running properly, we can go to the browser and type `localhost` in the search bar.

If the server asks for username and password, then we can conclude that all the server configurations have been done properly.

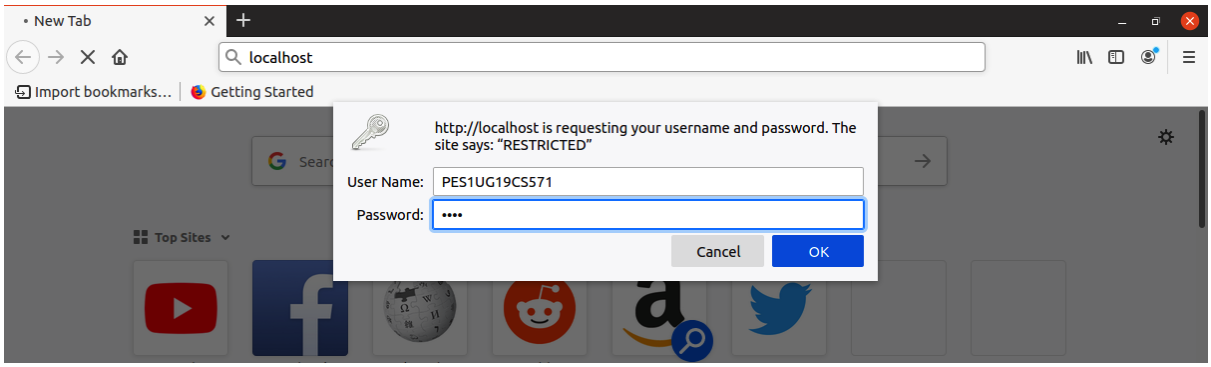

Figure 3: Accessing `localhost` from web browser.

From the above figure, we can say that our server configuration is proper and now we can proceed further. After entering our credentials, server will display the default `apache` page.

1.4 Capturing packets in Wireshark

We will open wireshark and monitor the 'any' interface with HTTP filter enabled.

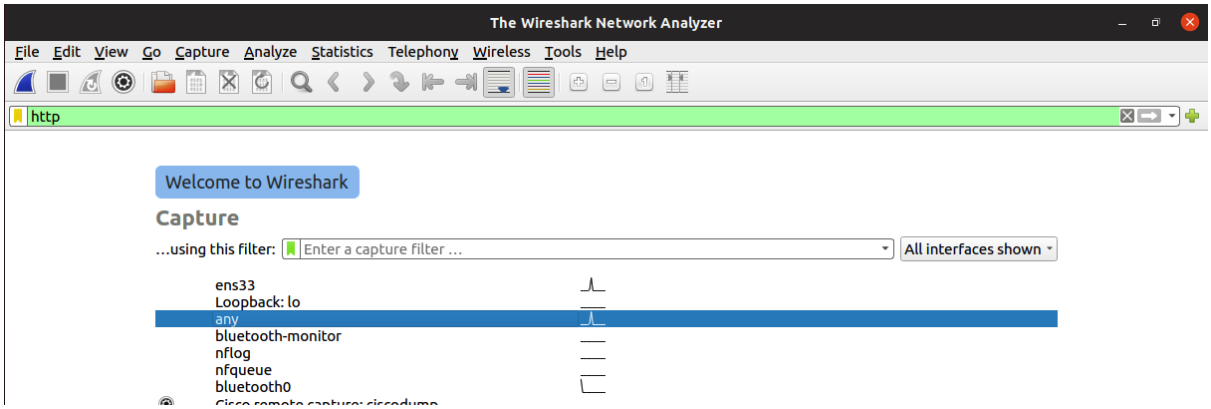

Figure 4: Selecting 'any' interface in wireshark with HTTP filter enabled.

We will empty the browser's cache and again search for `localhost` in browser.

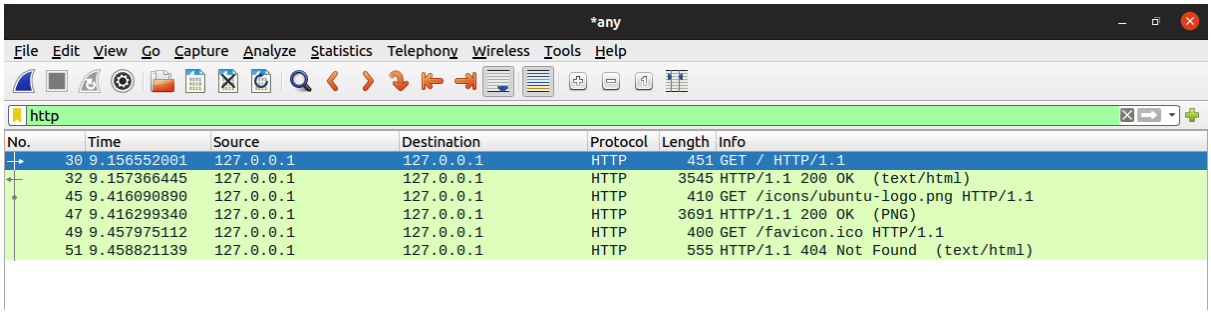

Figure 5: Packets captured on searching for `localhost` in browser.

We will select the GET request packet made to the server and follow its TCP Stream.

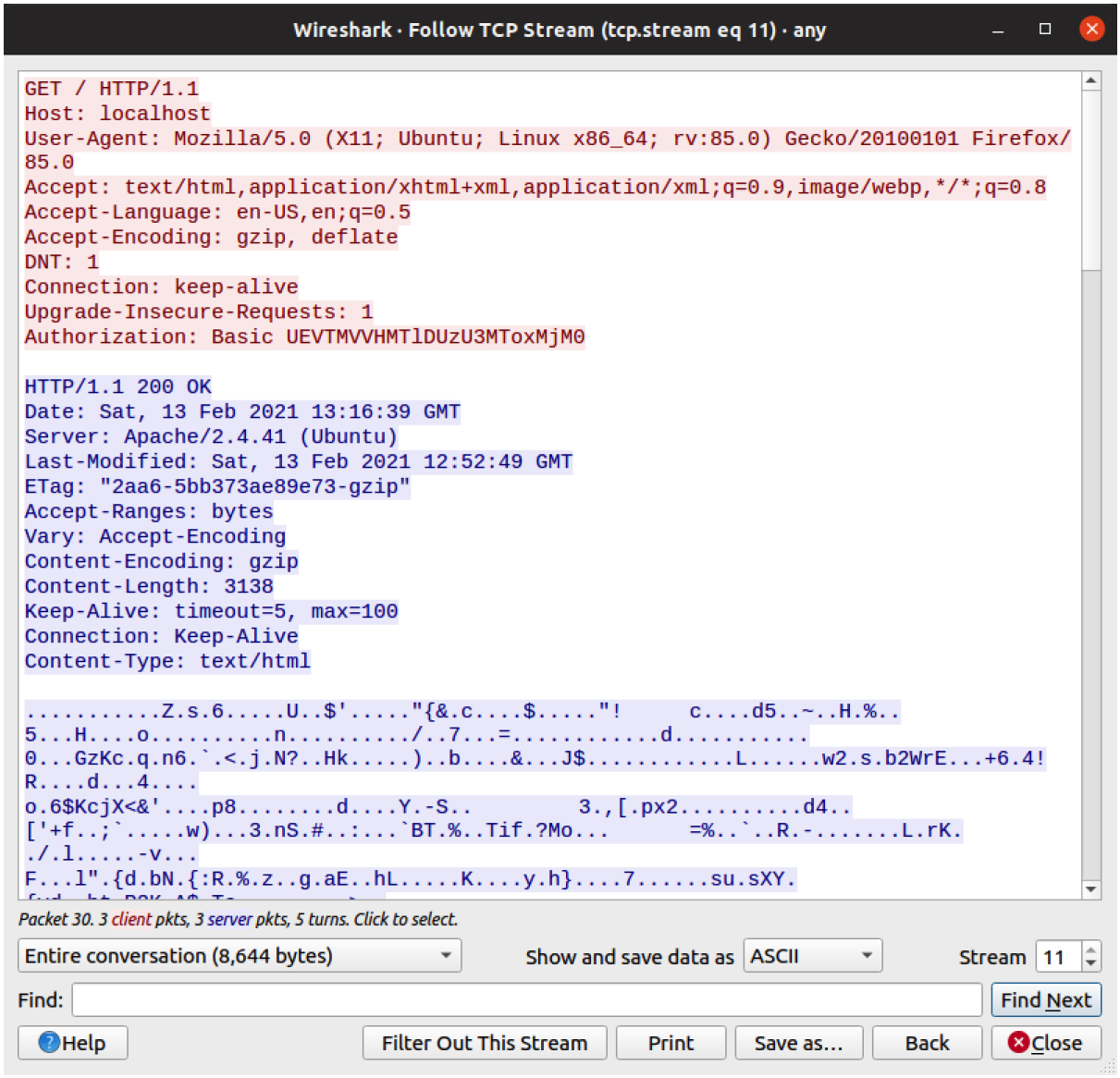

Figure 6: Following the TCP stream of GET request packet.

Important thing to note here is the `Authorization` field in the TCP Stream. `Basic` indicates that it is using basic encryption algorithm (Base64) to encrypt user's credentials. The text after `Basic` is the encrypted credentials of the user.

1.5 Understanding Base64 Algorithm

Base64 encode and decode algorithm converts any data into plain text and vice versa.

1.5.1 Base64 Encoding

Encoding is done in few simple steps.

- Convert each character in the input string to its equivalent binary value. The binary value is obtained by converting the ASCII value of the character to binary.

  PES1UG19CS571:1234 would be encoded as follows :

  P - 01010000
  E - 01000101
  S - 01010011
  1 - 00110001
  U - 01010101
  G - 01000111
  1 - 00110001
  9 - 00111001
  C - 01000011
  S - 01010011
  5 - 00110101
  7 - 00110111
  1 - 00110001
  : - 00111010
  1 - 00110001
  2 - 00110010
  3 - 00110011
  4 - 00110100

- Now we will concatenate all the binary values together to get one big number.

  01010000010001010101001100110001010101010100011100110001001110001...

- Divide this giant number into chunks of 6 binary digits as follows.

  010100 000100 010101 010011 001100 ....

- Add 00 in beginning of every chunk and convert each chunk into its decimal equivalent as follows :

  00010100 - 20
  00000100 - 04

00010101 - 21
00010011 - 19
00001100 - 12
.
... and so on.

- Now replace these decimal values with their corresponding alphabets. The alphabet set consists of all characters indexed from 0 i.e A = 0, B = 1, C = 2, D = 3 ... and so on.
- Hence PES1UG19CS571:1234 in Base64 encode will result in

UEVTMVVHMTlDUzU3MToxMjM0

1.5.2 Base64 Decoding

Decoding a Base64 encoded string is very simple and can be done as follows.

- Split the Base64 encoded string character by character.

  U
  E
  V
  T
  M
  ... and so on

- Convert the alphabets into its decimal equivalents. If A = 0, B = 1, C = 2 , ... then

  U - 20
  E - 4
  V - 21
  T - 19
  M - 12
  ... and so on.

- Convert these decimal numbers into its equivalent binary value.

  20 - 00010100
  04 - 00000100
  21 - 00010101
  ... and so on.

- Remove the first two 0's from each binary value and concatenate all the values into one big value.

  010100000100010101010011 ...

- Divide the above string into chunks of 8 as follows

  01010000
  01000101

01010011
... and so on.

- Converting this binary number into decimal format will give us the ASCII value. Based on the ASCII value we can convert it into alphabets.

$$01010000 \mapsto 80(ASCII) \mapsto P$$
$$01000101 \mapsto 69(ASCII) \mapsto E$$
$$01010011 \mapsto 83(ASCII) \mapsto S$$
$$\ldots \text{ so on.}$$

- Concatenating all letters, we get back PES1UG19CS571:1234.
  Thus we have successfully decoded the credentials.

UEVTMVVHMTlDUzU3MToxMjM0 ↦ PES1UG19CS571:1234

2 Setting Cookies

2.1 Setting Cookies using PHP

Cookies can be set by using a small `.php` script. The `setcookie(key,value,expire_time);` method can be used to set cookies.

We will create a php file called `abc.php` in the directory `/var/www/html` and the following contents were added in it.

Figure 7: Contents of `abc.php` script.

Here we are setting a cookie with key as Username and its value as Vishal. This cookie will expire after 1 day. We will set another cookie with key SRN and its value as PES1UG19CS571.

Before we proceed further, we need to make sure that we have PHP locally installed otherwise we will not be able to execute php files. To check if we have PHP installed, we can type `php -v` in terminal. If we do not have php installed, we need to install it using the command `$ sudo apt-get install php`.

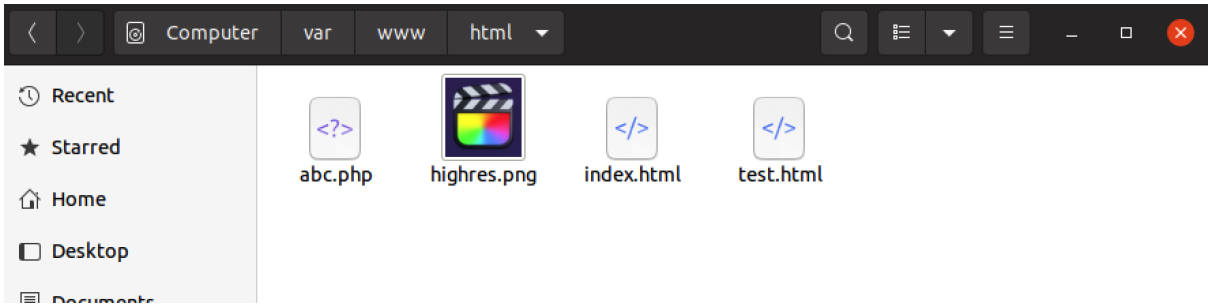

Figure 8: Contents of `/var/www/html` directory.

Lastly, we need to restart our apache server to reflect the changes. This can be done by typing the command `$ sudo systemctl restart apache2`.

2.2 Capturing Packets in Wireshark

Wireshark can be used to capture packets sent in the network. We will open a browser and type `localhost/abc.php` in the search bar.

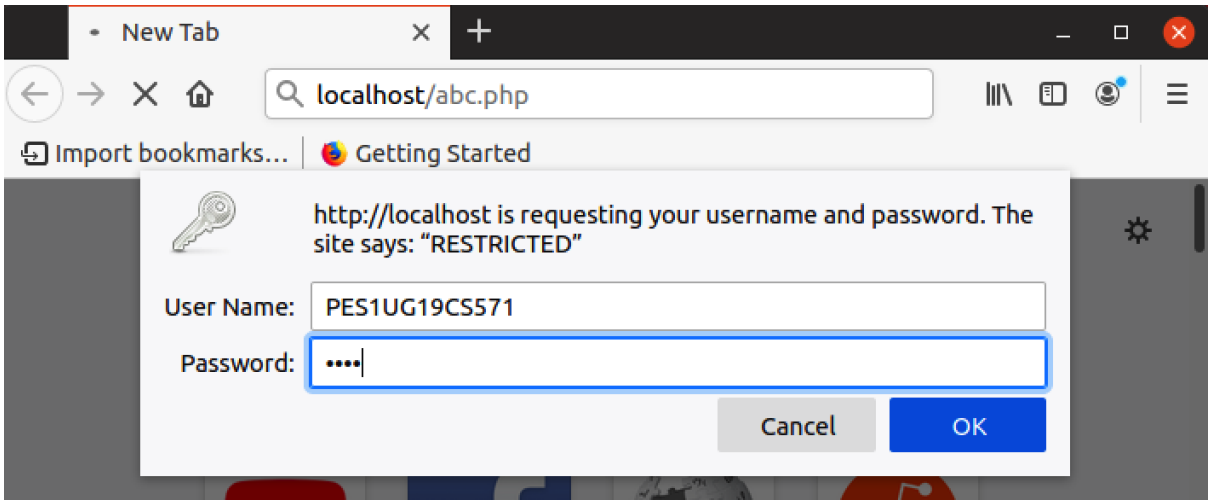

Figure 9: Making GET request to `localhost/abc.php`.

Before we click 'OK' we will open wireshark and monitor 'any' interface with HTTP filter enabled.

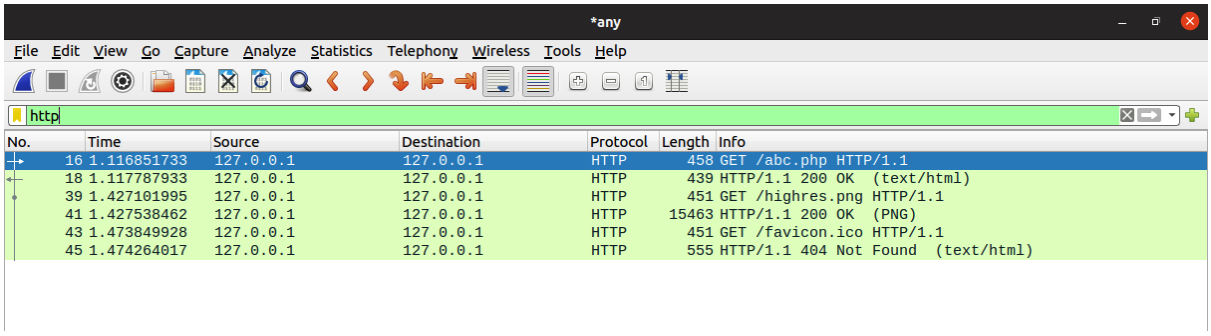

Figure 10: Packets captured in wireshark.

We will select the first GET request packet and follow its TCP Stream.

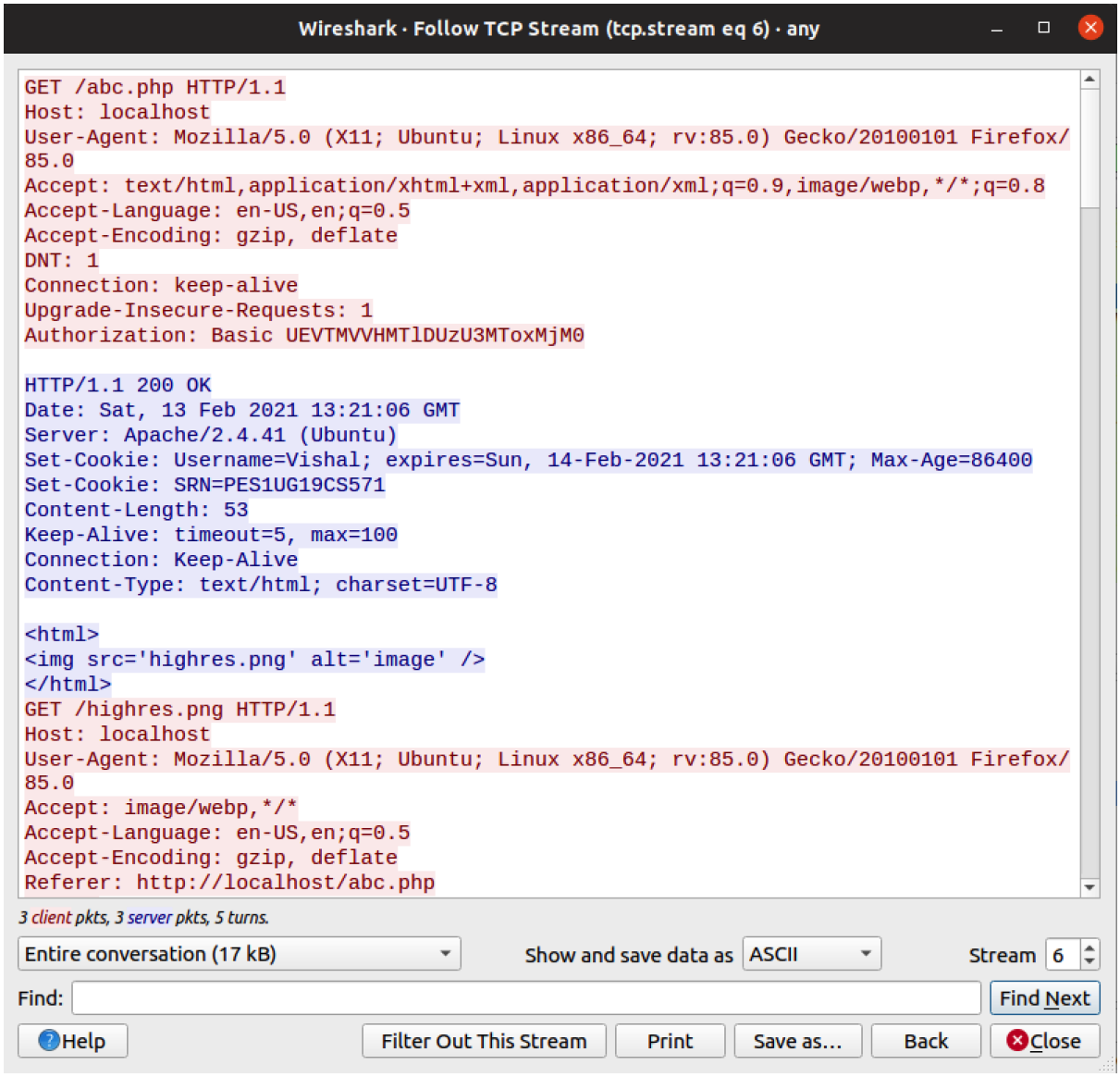

Figure 11: TCP Stream of first GET request packet.

The cookie's key, value and expiry date can be view in the `Set-Cookie` field of the TCP Stream. This shows that the cookie has been successfully created and is stored in browser.

3 Conditional GET

A conditional HTTP response is one that carries only the resource if it has been modified since the last GET request from the client.

The `IF-Modified-Since` HTTP header is one way to implement conditional GET. The server checks this header value and resends the resource if it has been modified. If the resource was not modified, we will get a `304 Not Modified` status code from server.

3.1 Repeated GET requests for HTML Page

With wireshark opened in capture mode and browser's cache cleared, the link http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html was opened in browser.

Immediately, we will request for the same page again by clicking the reload button.

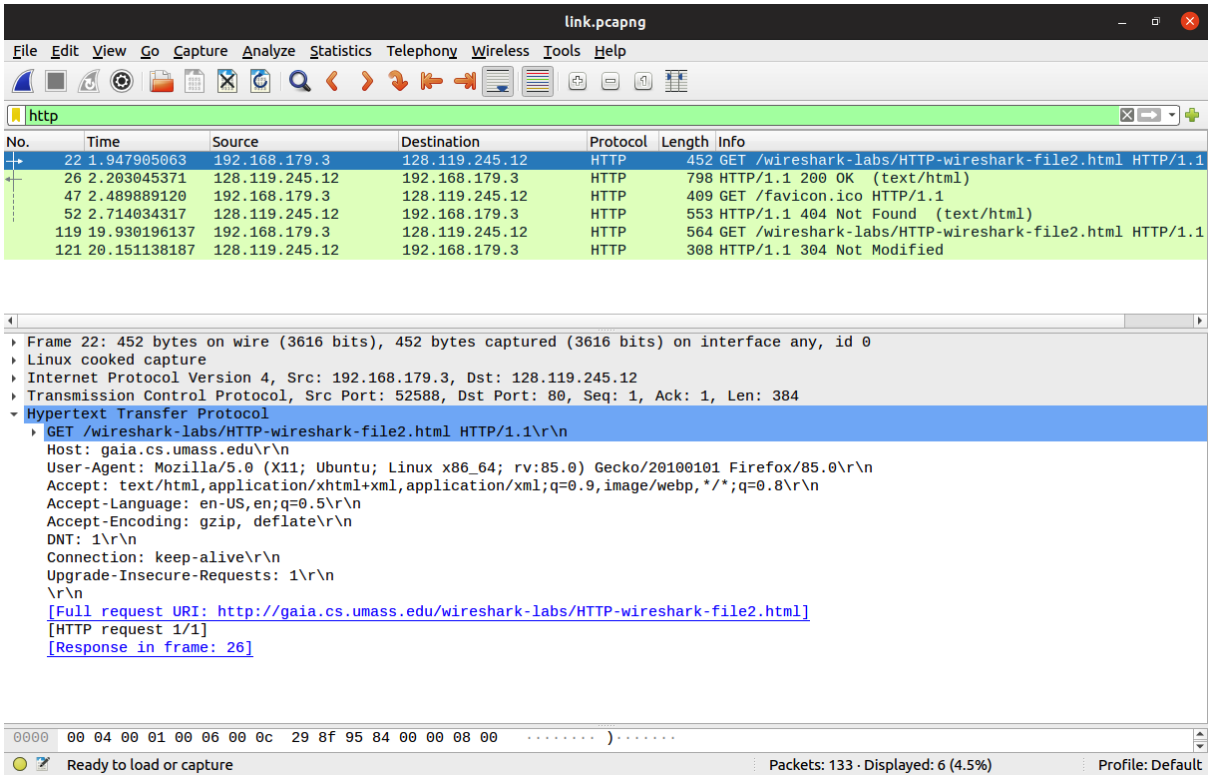

Figure 12: Packet capture in wireshark.

On examining the packet contents, we can observe that there is **no** `IF-Modified-Since` field in the HTTP header.

We will inspect the server's response now and see if the server has sent the contents of html file that we requested for the first time.

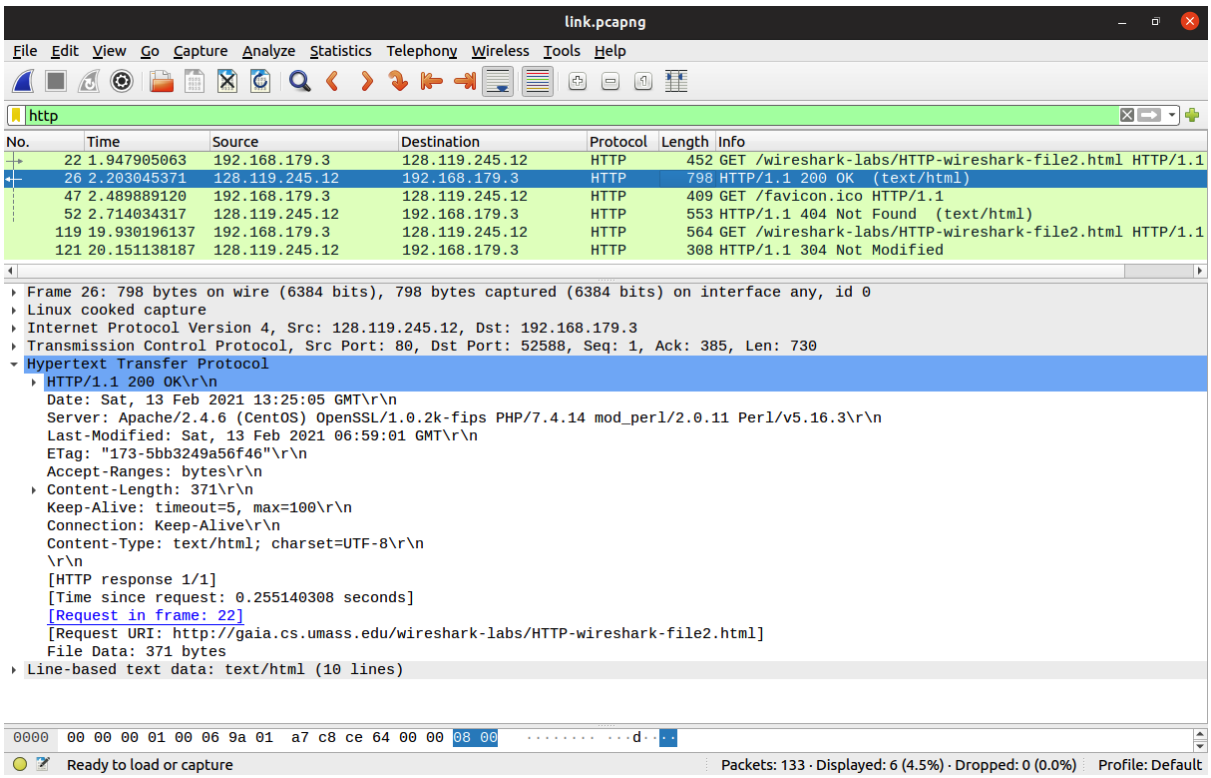

Figure 13: Packet capture in wireshark.

We can see that server has **explicitly returned the contents of html file. We can tell this by seeing the text/html tab in wireshark.**

We will inspect the second GET request we made.

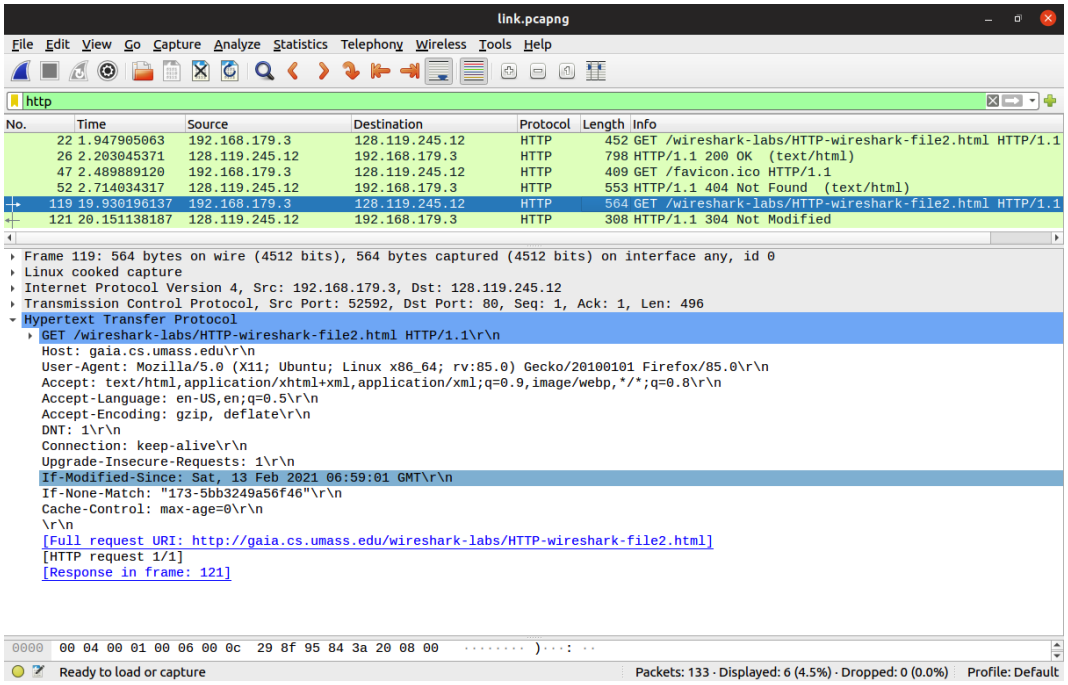

Figure 14: Inspecting second GET request packet in wireshark.

From the above figure, we can see that `IF-Modified-Since` header is present in the request packet. The information against this header is a timestamp that indicates the time at which the last get request was made by the client. This will indicate to server to send the resource if the resource has been modified since that timestamp. If the resource has not been modified, the browser will load from cache.

Finally, we will inspect the second response we got from the server.

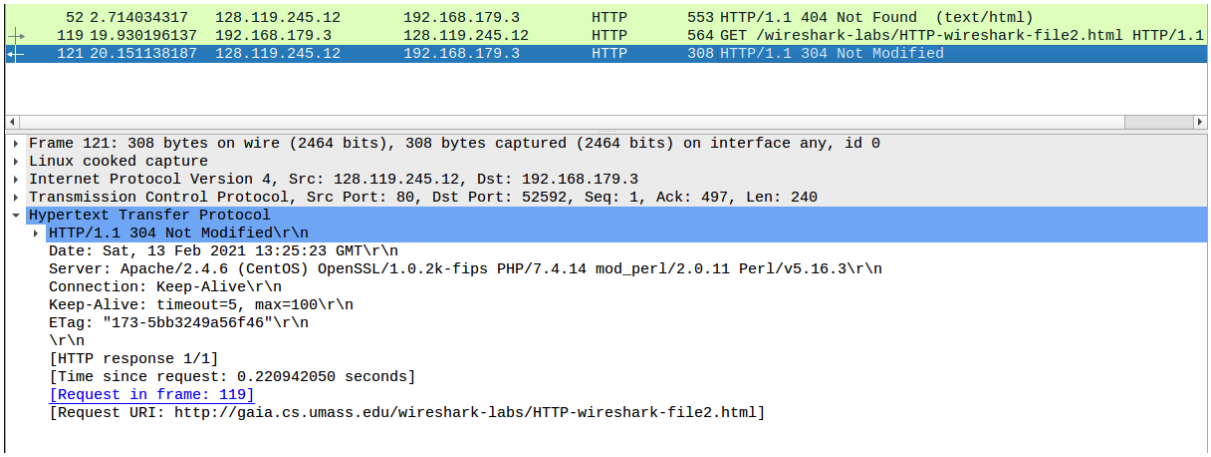

Figure 15: Inspecting second response packet in wireshark.

The status code sent back to us is **304 Not Modified**. This indicates that the resource has not been modified since the last GET request made by client.

Also, the server has not sent any html content which means that the resource hasn't been modified and the browser will load the resource from its cache.

3.2 Conditional GET on Localhost

We can perform the same steps on an html file present in localhost. The following html file was created and added to the `/var/www/html/` directory.

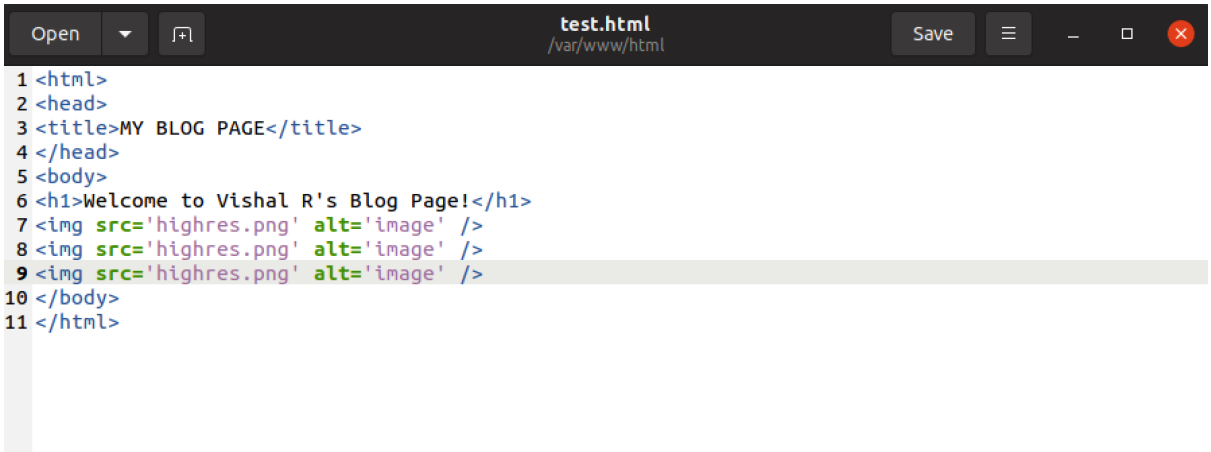

Figure 16: Contents of `test.html`.

In browser, `localhost/test.html` was entered.

Figure 17: `test.html` loaded in browser.

The the page was quickly refreshed and the following packets were captured in wireshark.

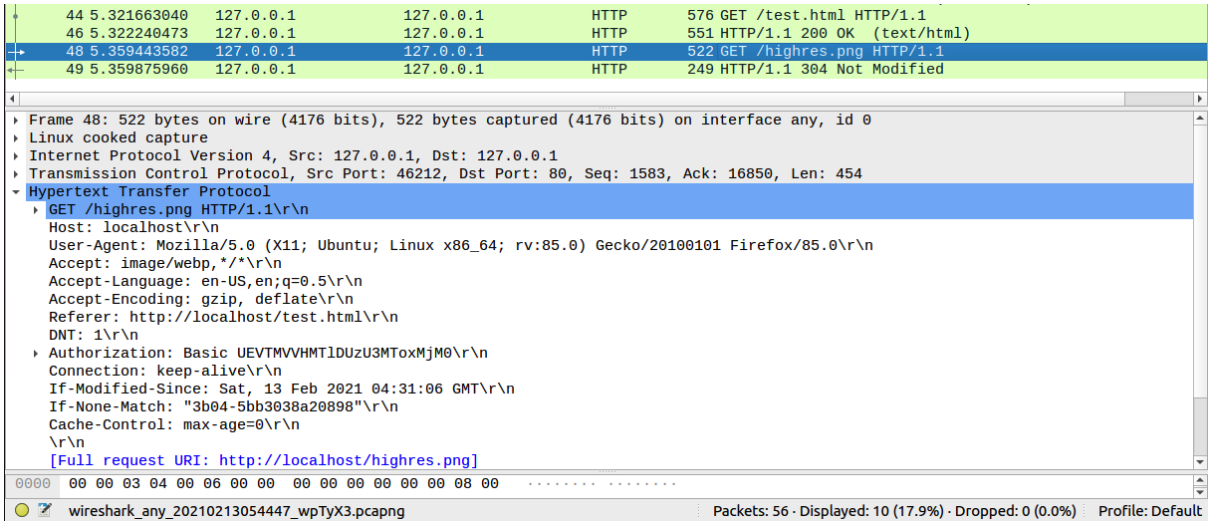

Figure 18: Packet Caputre in Wireshark.

As you can see from the figure, the first time page is requested by client, the resources are cached by browser. When we made the second GET request, we got a response as **304 Not Modified** indicating that the resource has not been modified since the last GET request made by the client. If the resource had been modified, the server would send the contents to client.