**Week: #3**

**Understand working of HTTP Headers**

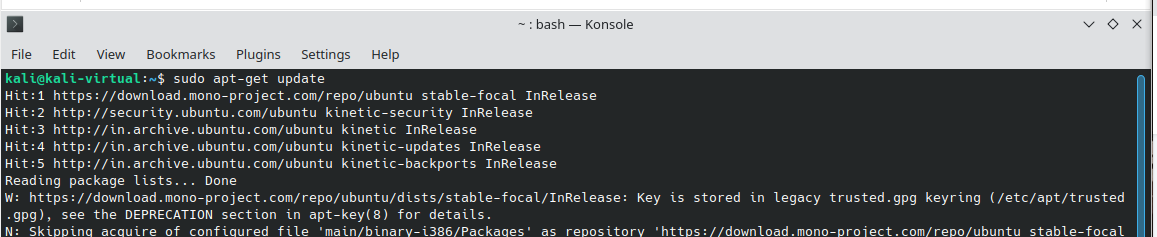
**Name = Rachappa**

**Srn = PES1UG19CS359**

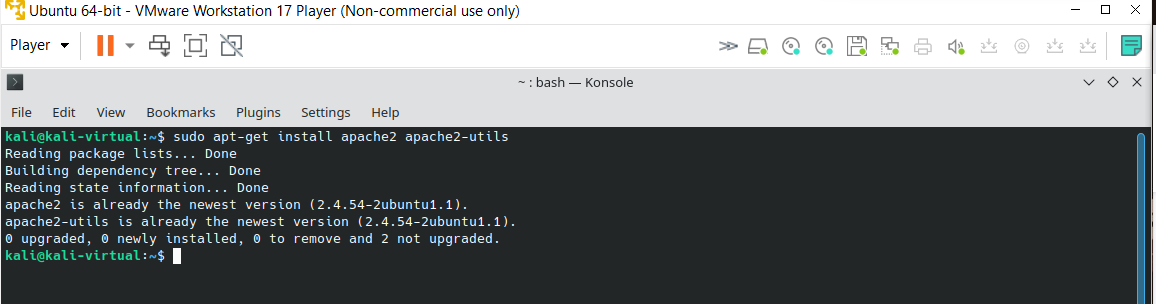
**ROLL NO = 1**

**CLASS = A**

**sudo apt-get update**

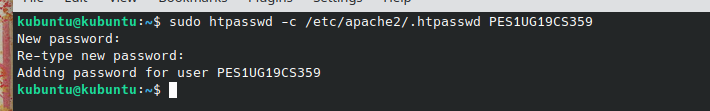


**sudo apt-get install apache2 apache2-utils**

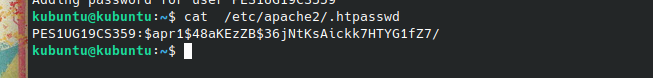


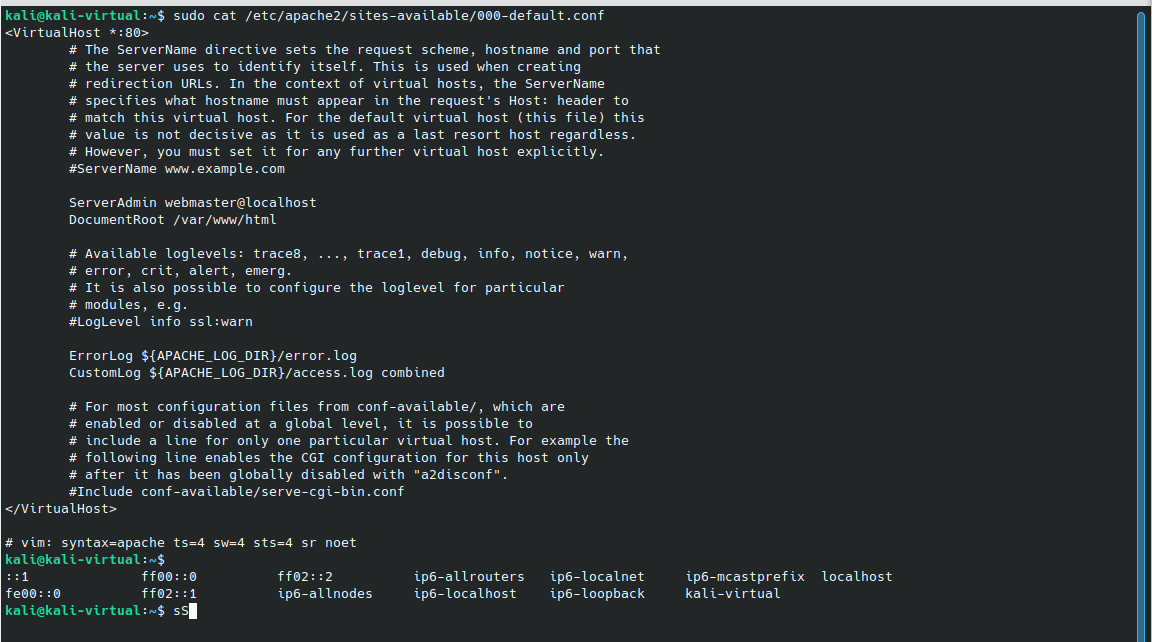
--> Provide username and password to set authentication

**sudo htpasswd -c /etc/apache2/.htpasswd ANY\_USERNAME**



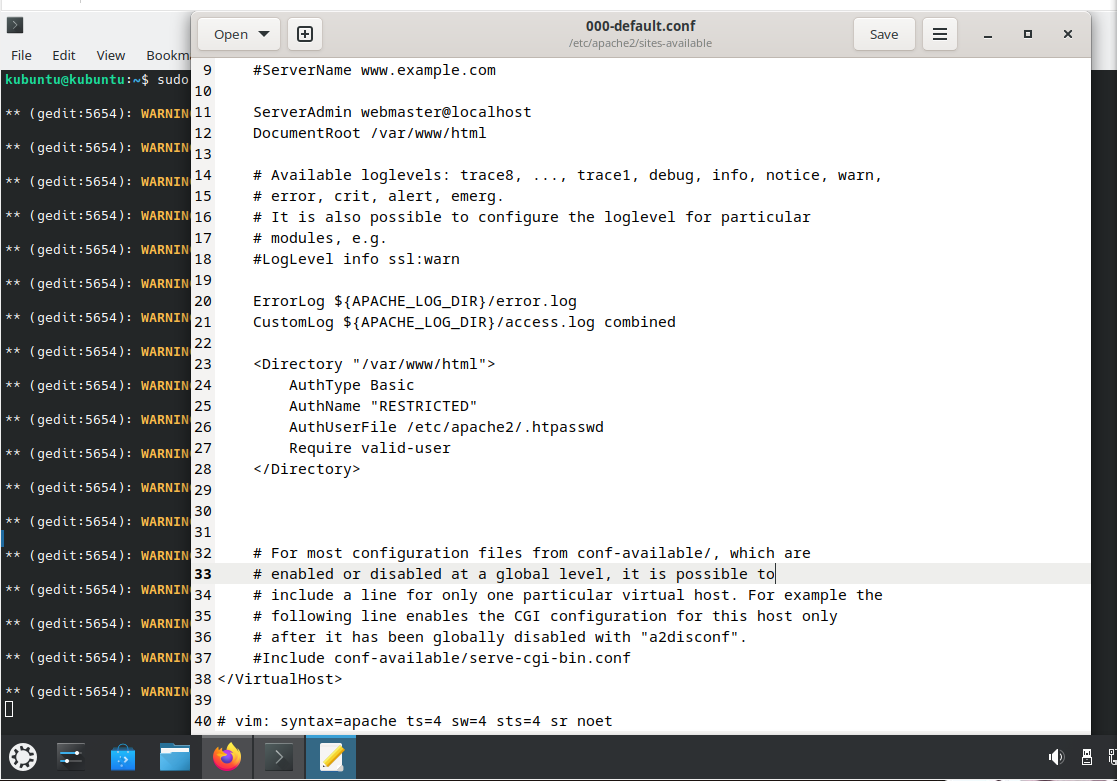
**sudo cat /etc/apache2/.htpasswd**

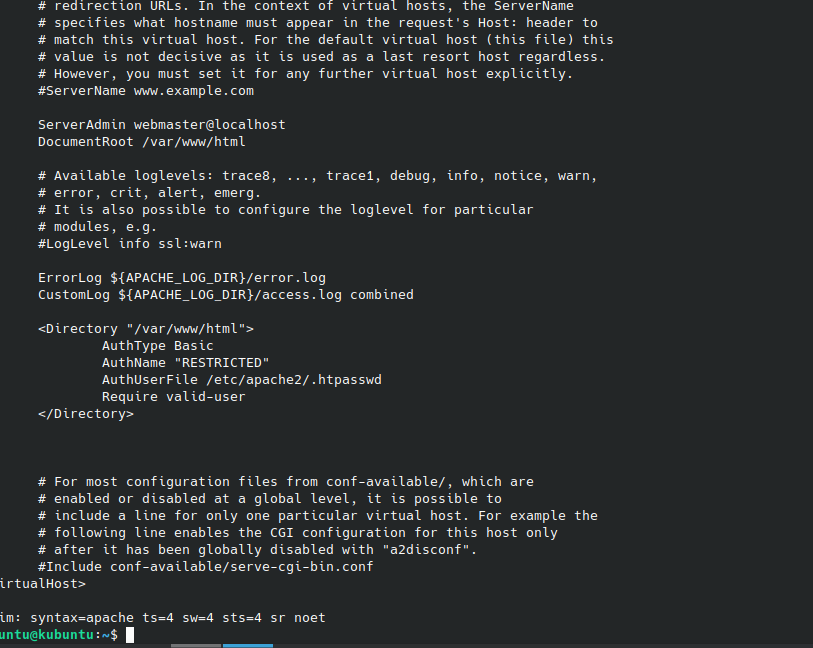




--> Opening the file for setting authentication

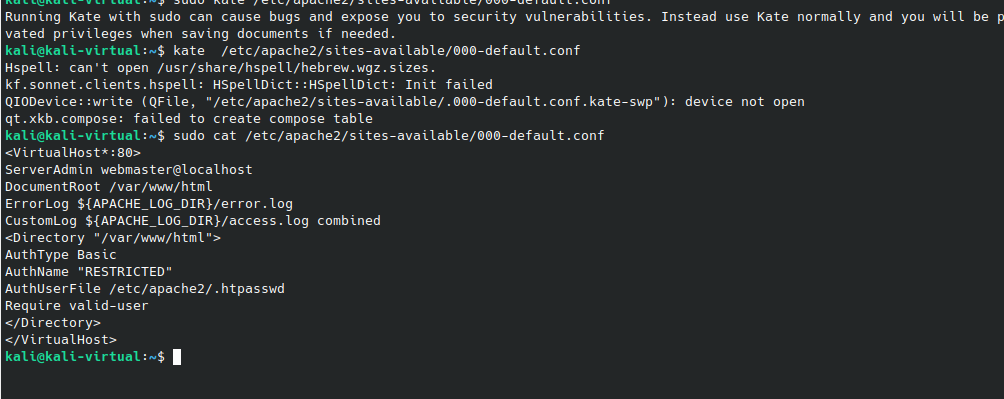
**sudo nano /etc/apache2/sites-available/000-default.conf**





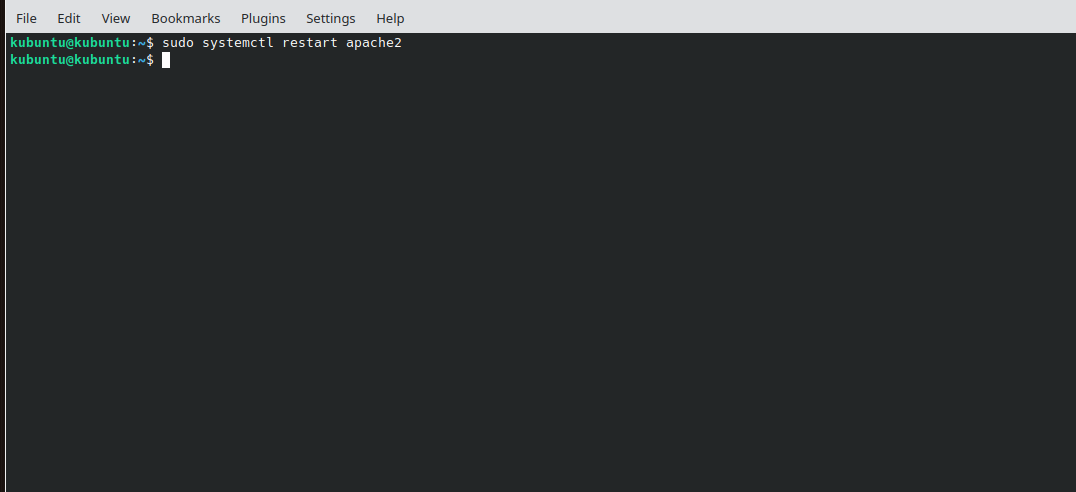
--> Opening the file for setting authentication

**sudo nano /etc/apache2/sites-available/000-default.conf**



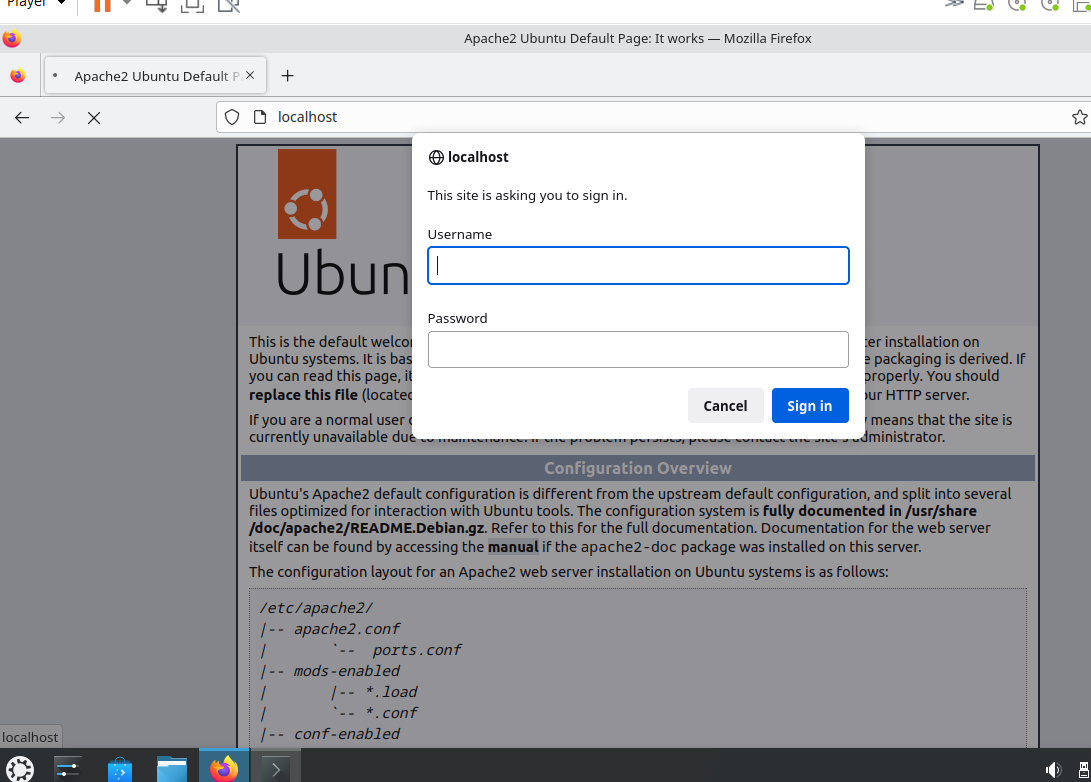
. Password policy implementation is done by restarting the server as:

**sudo service apache2 restart**

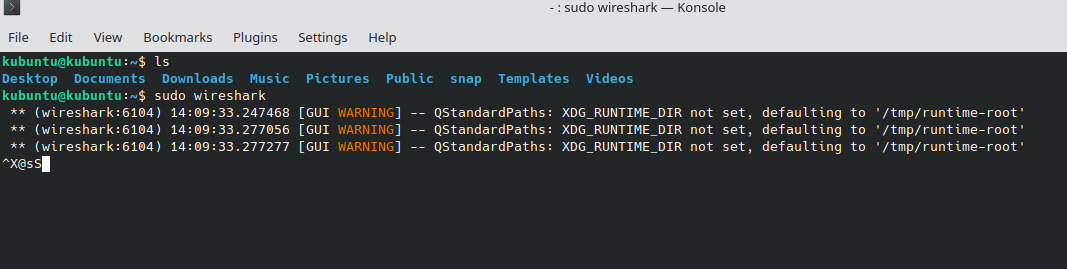


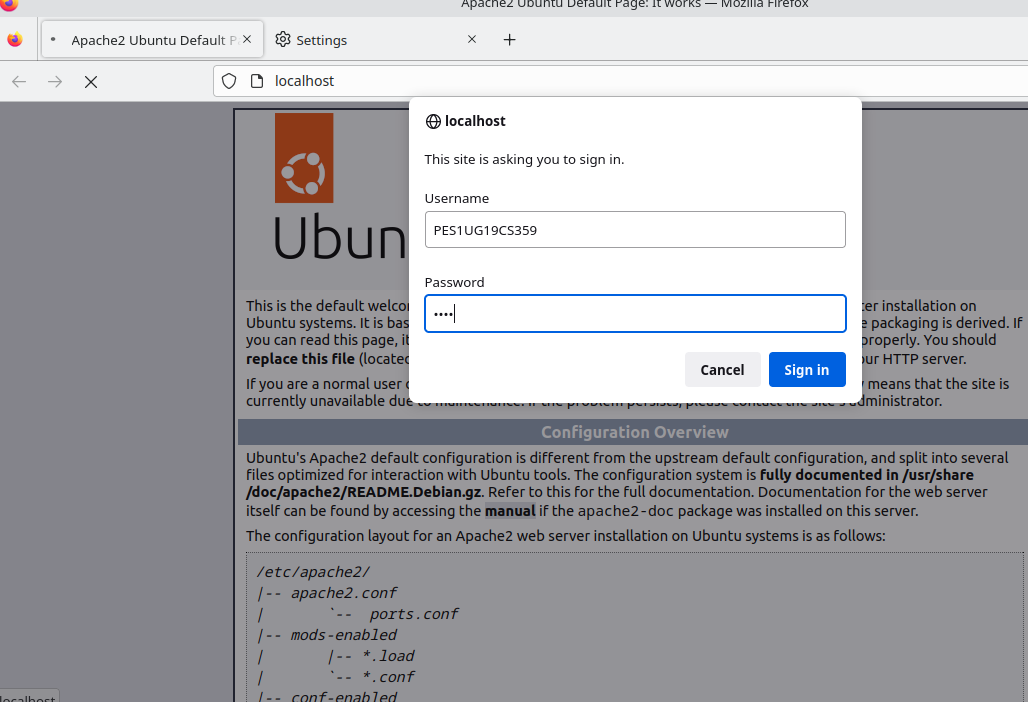
4. The localhost is then accessed using the Firefox browser requiring a username and a password

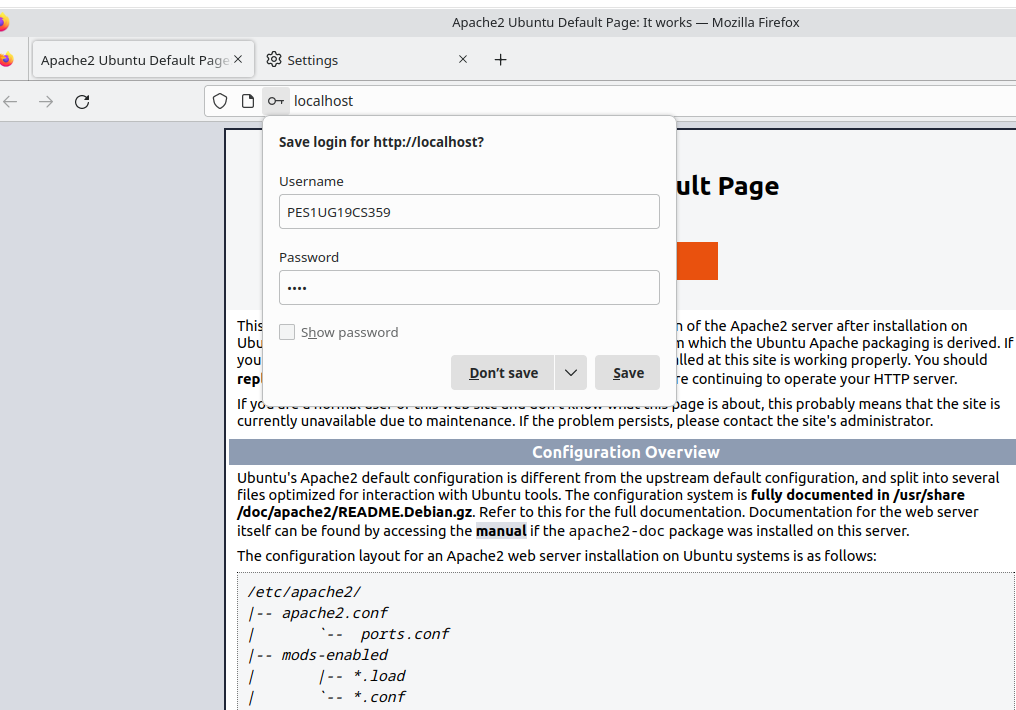
set during the authentication phase.

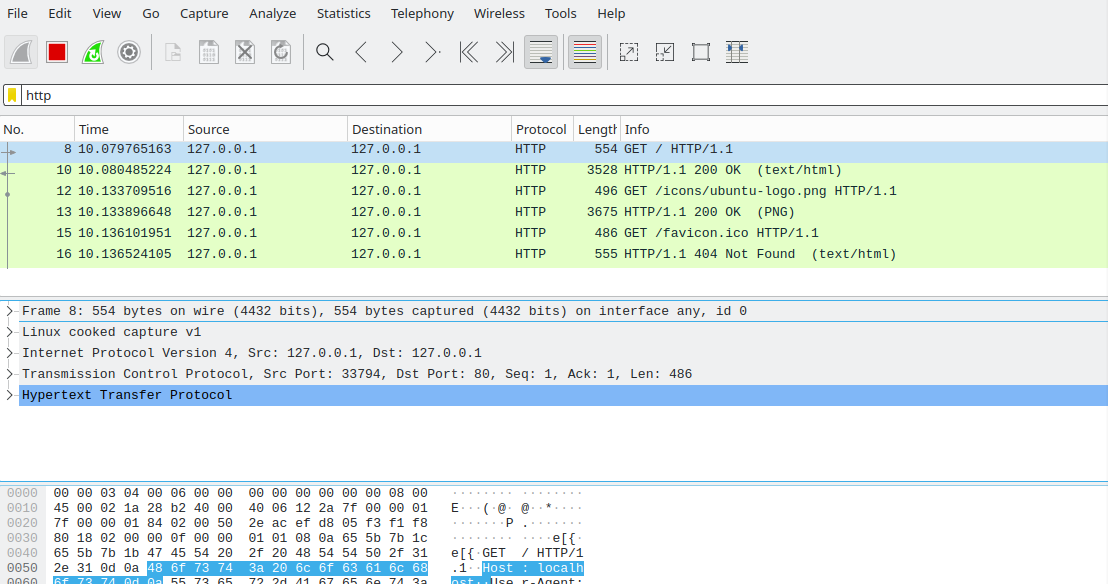


5. Wireshark is used to capture the packets sent upon the network.





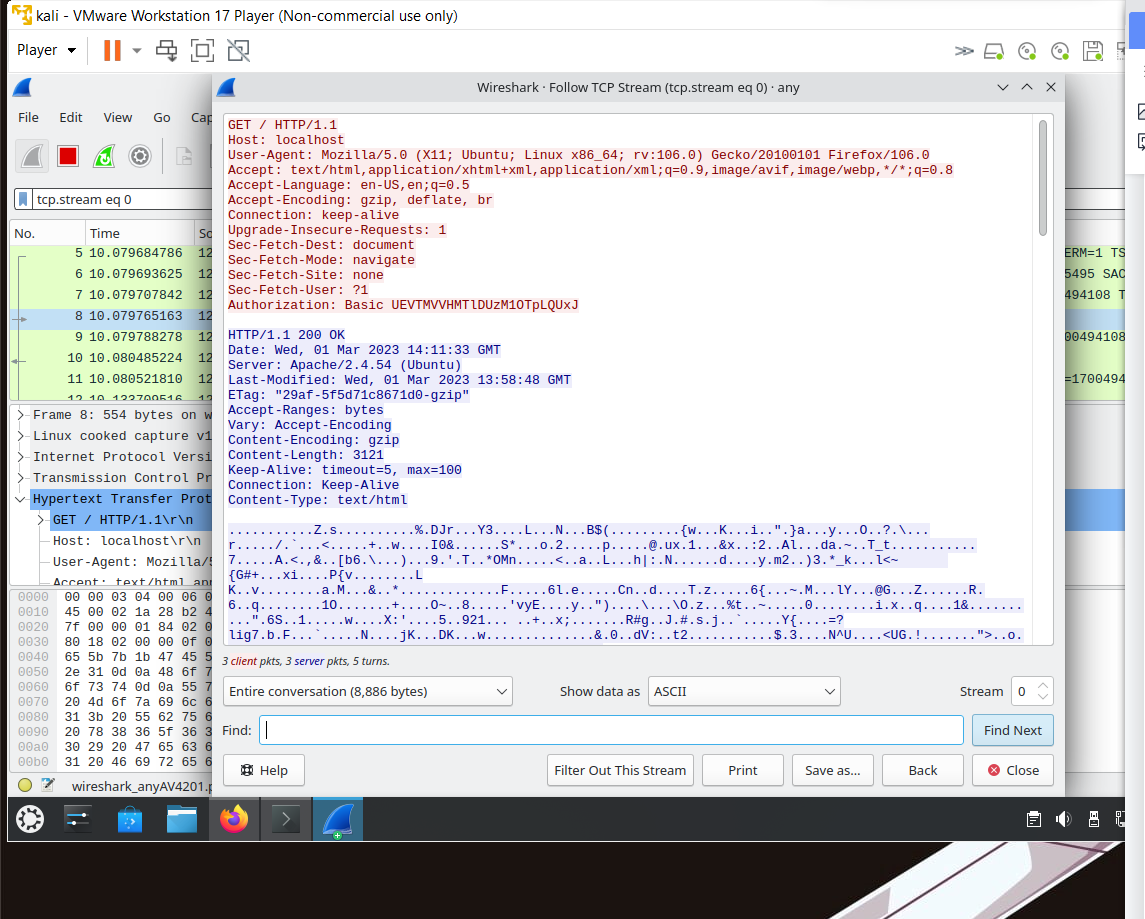




6. Using the “follow TCP stream” on the HTTP message segment the password was retrieved

which was encrypted by the base64 algorithm and decryption could be done with same

algorithm



## Understanding Base64 Algorithm

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

* + 1. **Base64 Encoding**

Encoding is done in few simple steps.

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

PES1UG19CS359:1234 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

3 - 00110011

5 - 00110101

9 - 00111001

: - 00111010

1 - 00110001

2 - 00110010

3 - 00110011

4 - 00110100

Now we will concatenate all the binary values together to get one big number. 0101000001000101010100110011000101010101010001110011000100111001*...*

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 equiv- alent 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 PES1UG19CS359:1234 in Base64 encode will result in

**UEVTMVVHMTlDUzU3M10TpLQUxJ**

* + 1. **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 *›→* 80(*ASCII*) *›→ P*

01000101 *›→* 69(*ASCII*) *›→ E*

01010011 *›→* 83(*ASCII*) *›→ S*

... so on.

Concatenating all letters, we get back PES1UG19CS571:1234.

Thus we have successfully decoded the credentials.

**UEVTMVVHMTlDUzU3M10TpLQUxJ** *›→* **PES1UG19CS5359:1234**

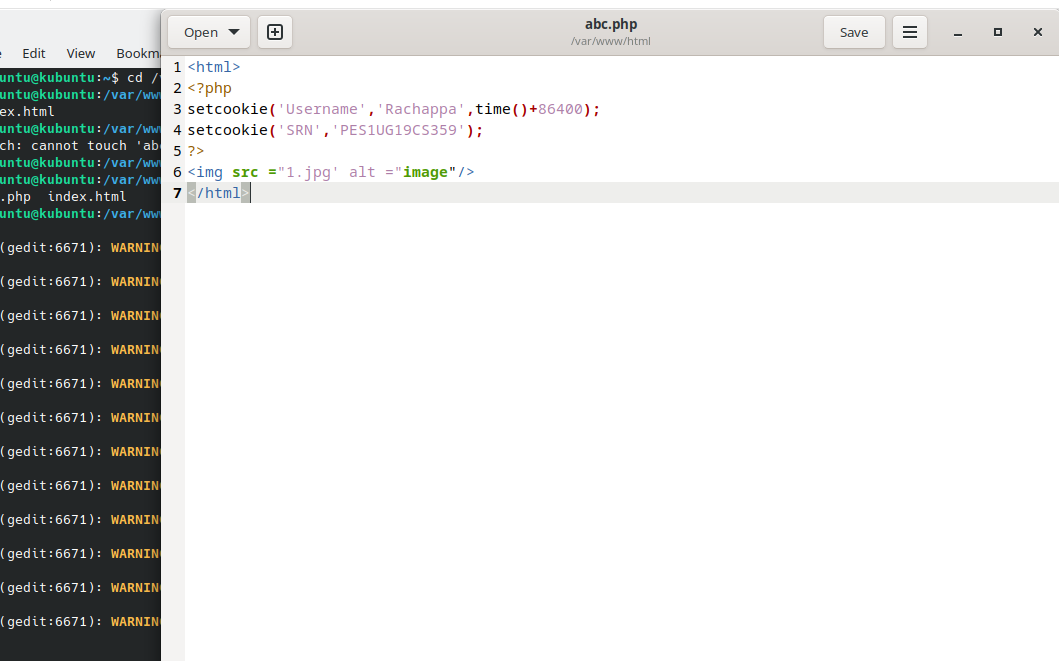
**Steps of Execution (Cookie Setting)**

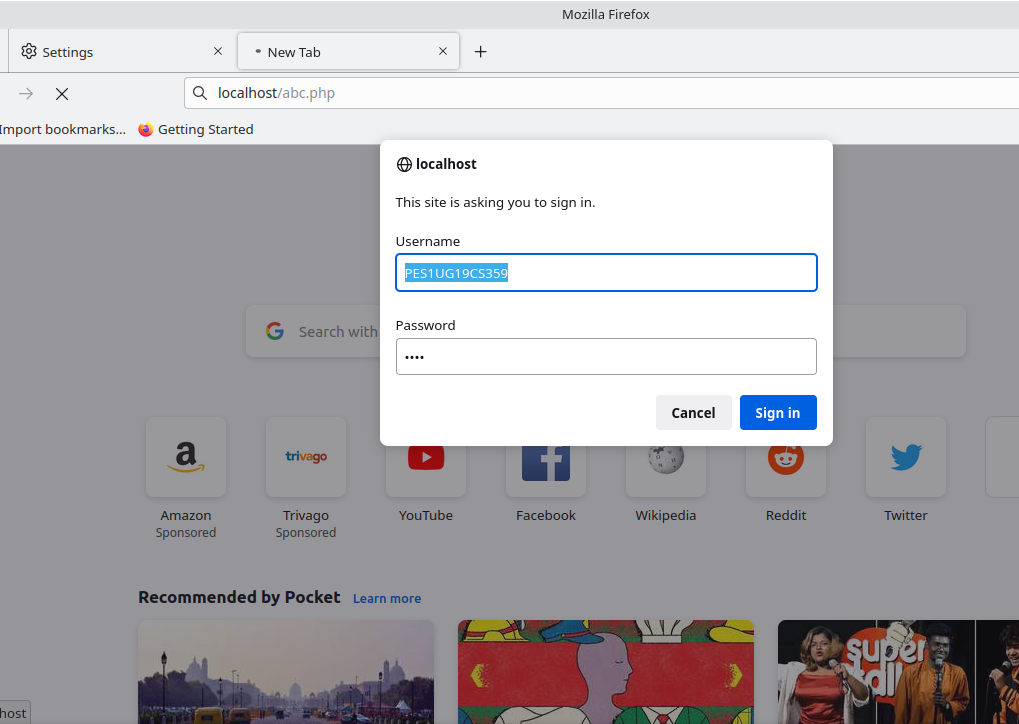
1. A PHP file to set the cookie is created which also contains an image in it (placed under the

HTML directory) to be accessed once the cookie is set. The following code helped to set the

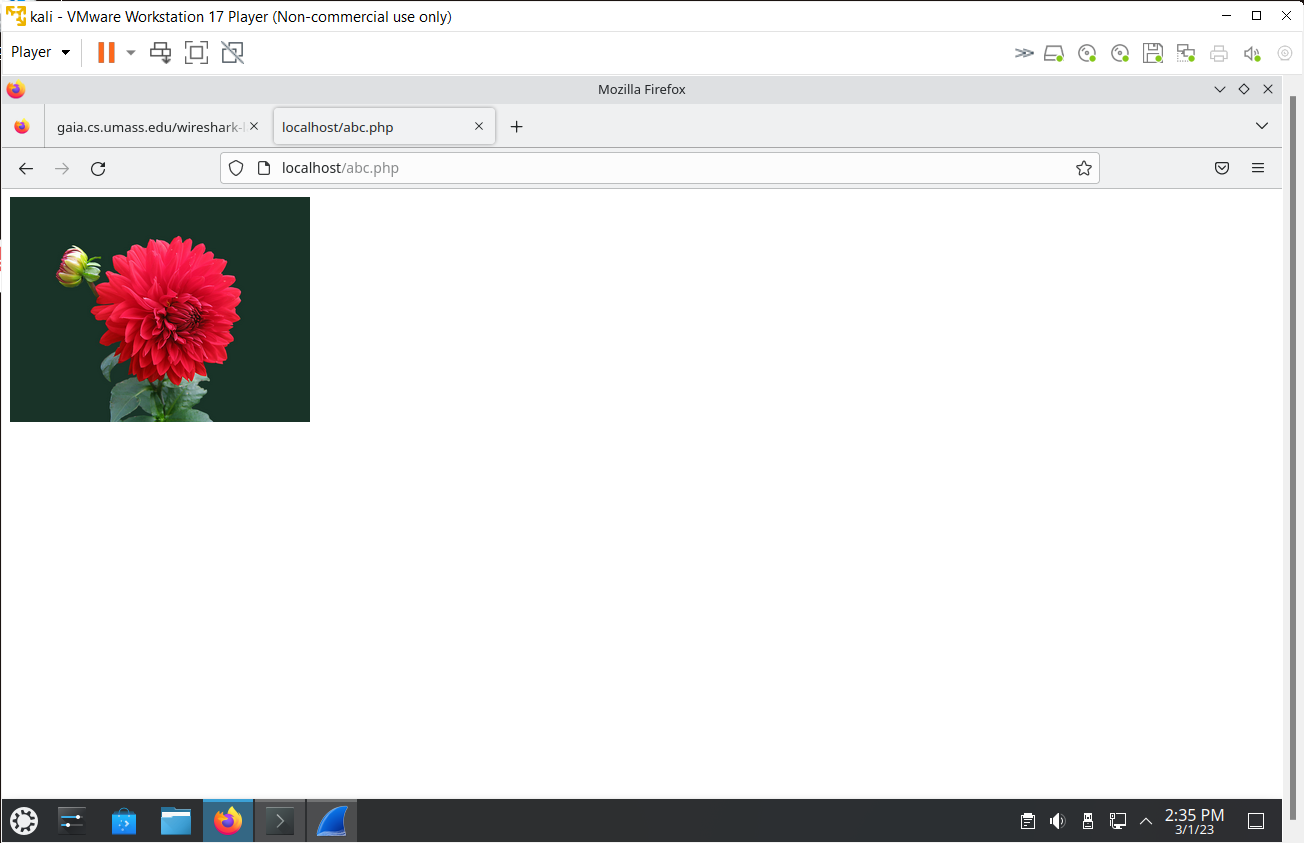
cookie:

## Setting Cookies using PHP

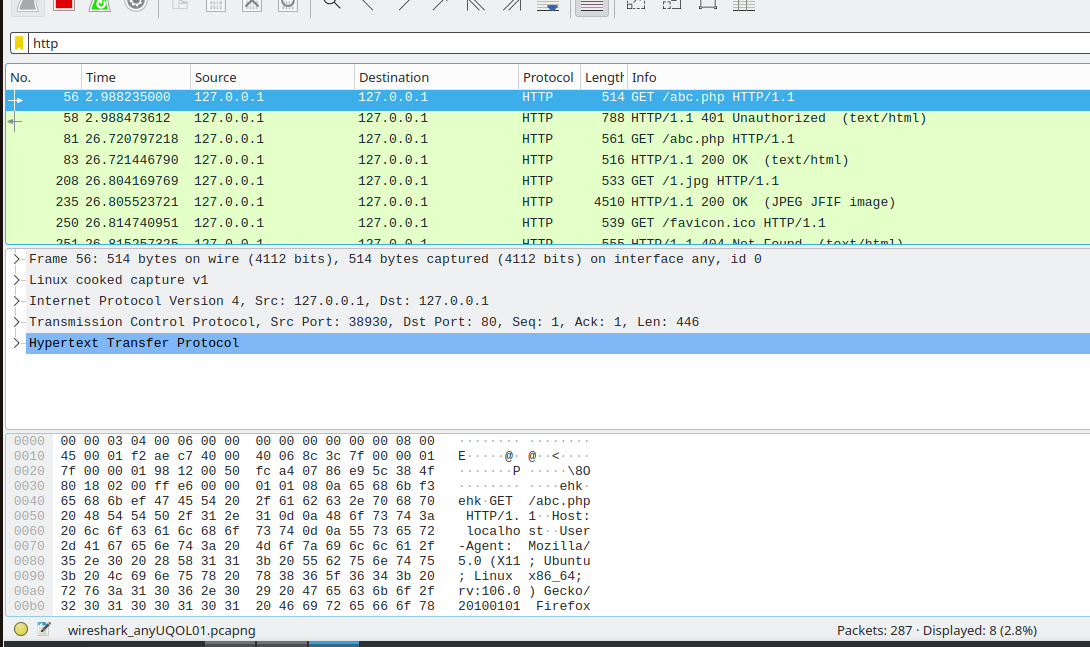


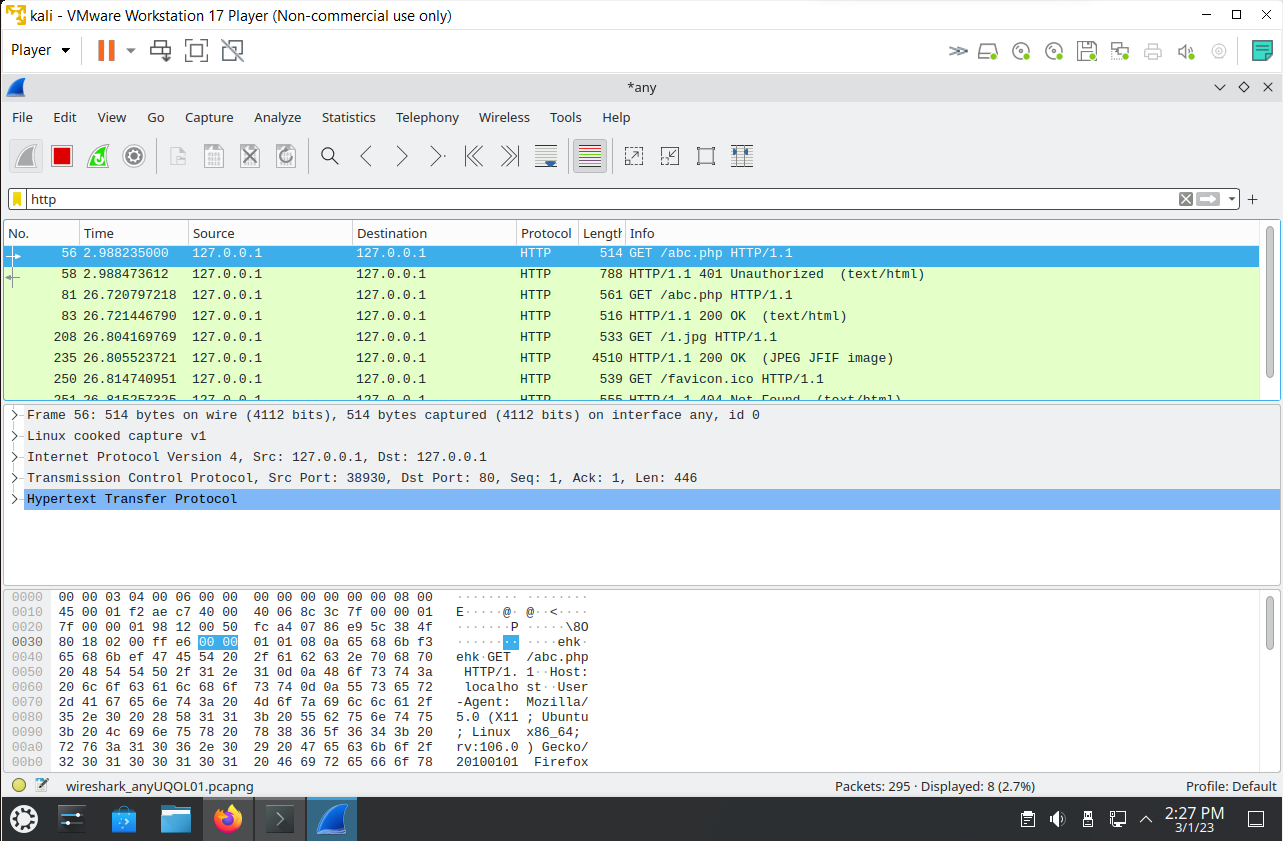


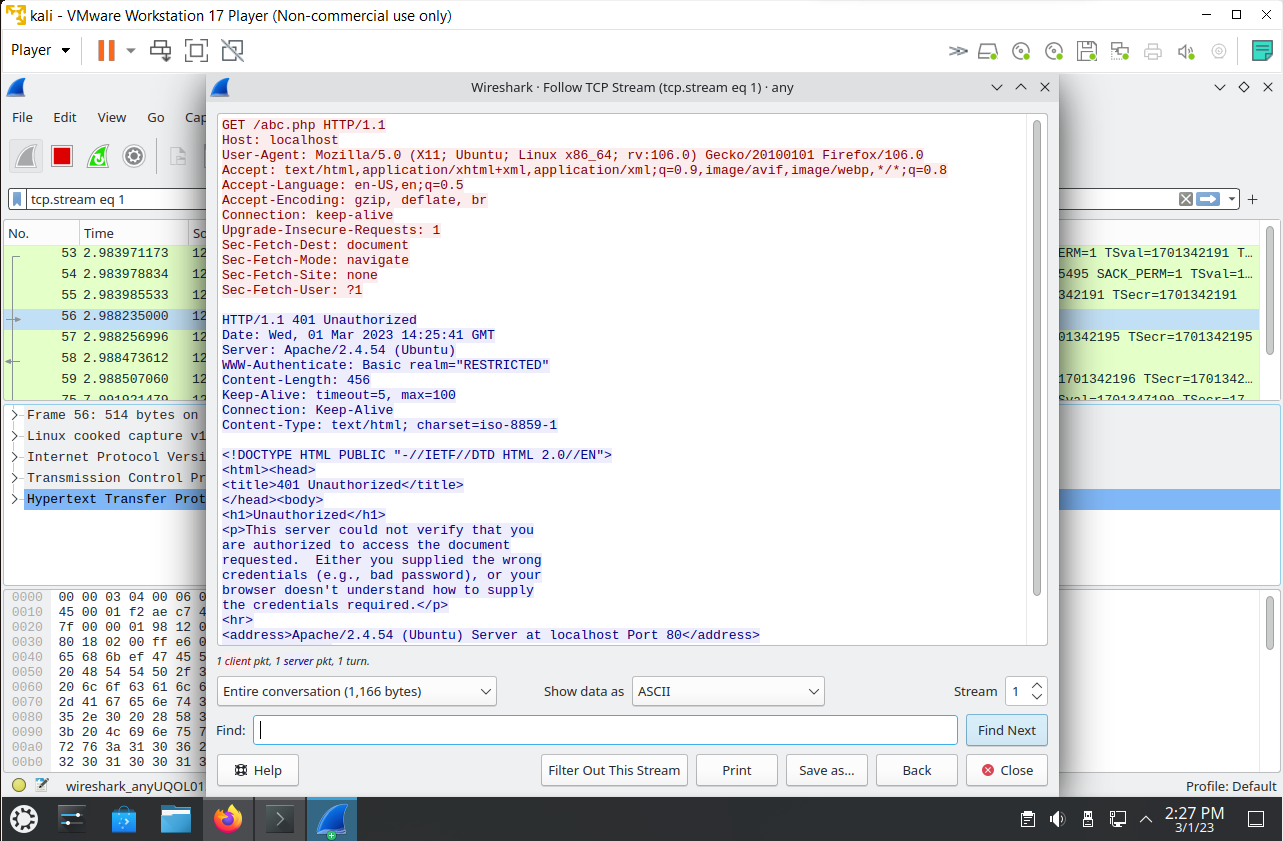
2. The combined file saved with a .php extension is placed under **/var/www/html** for accessing.



## Capturing Packets in Wireshark







**Conditional Get: If-Modified-Since**

Before performing the steps below, make sure your browser’s cache is empty. (To do this under

Firefox, select Tools -> Clear Recent History and check the Cache box). Now do the following:➢ Start up your web browser, and make sure your browser’s cache is cleared, as discussed

above.

➢ Start up the Wireshark packet sniffer.

➢ Enter the following URL into your browser http://gaia.cs.umass.edu/wireshark

labs/HTTP-wireshark-file2.html

➢ Your browser should display a very simple five-line HTML file.

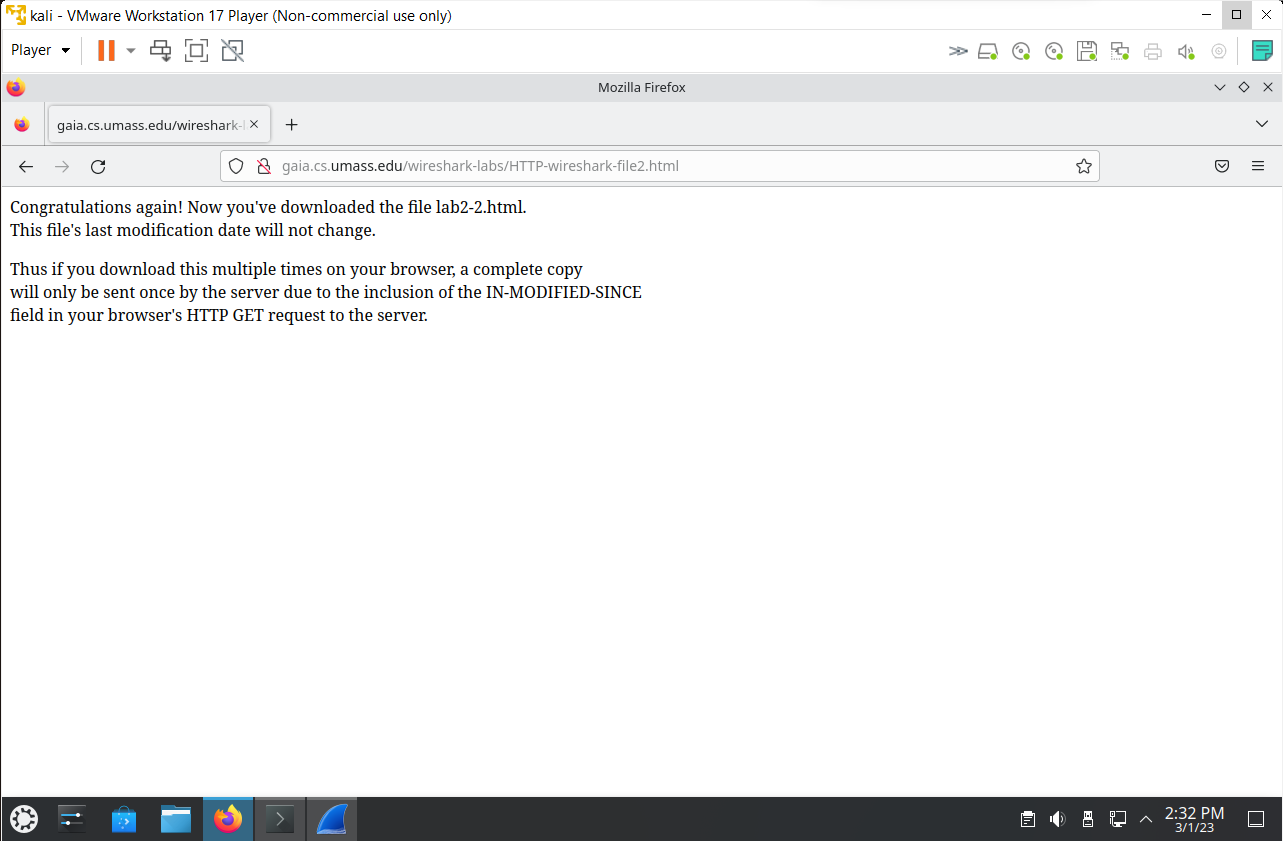
➢ Quickly enter the same URL into your browser again (or simply select the refresh button

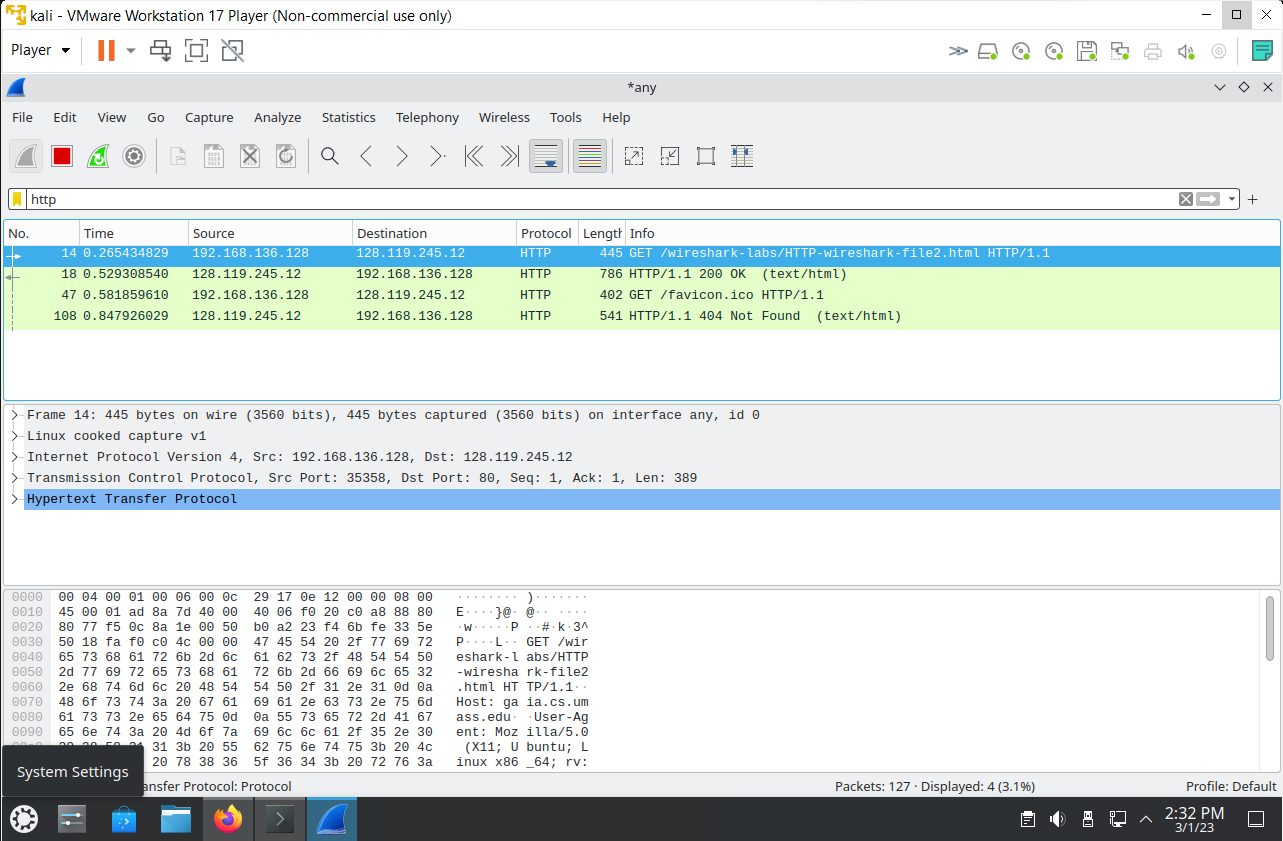
on your browser)

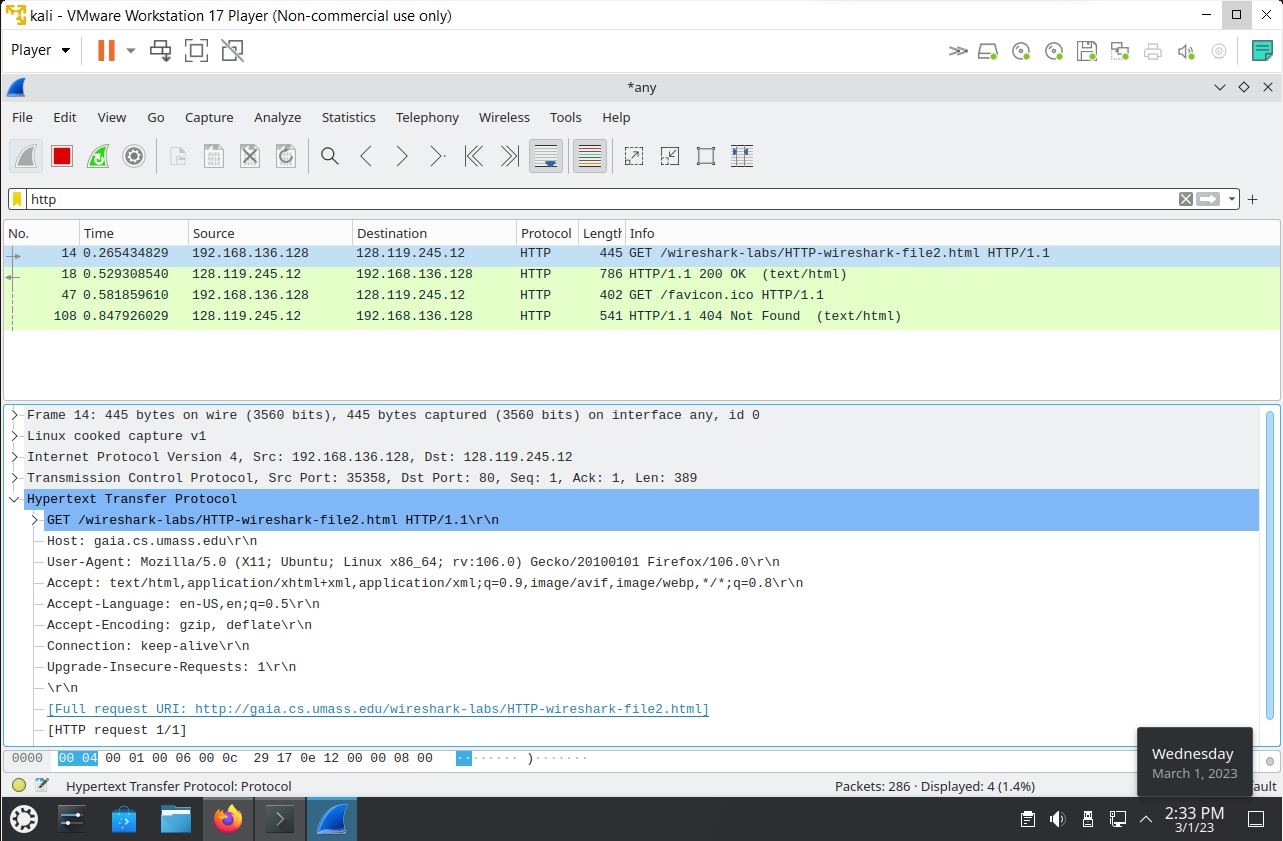
➢ Stop Wireshark packet capture, and enter “http” in the display-filter-specification

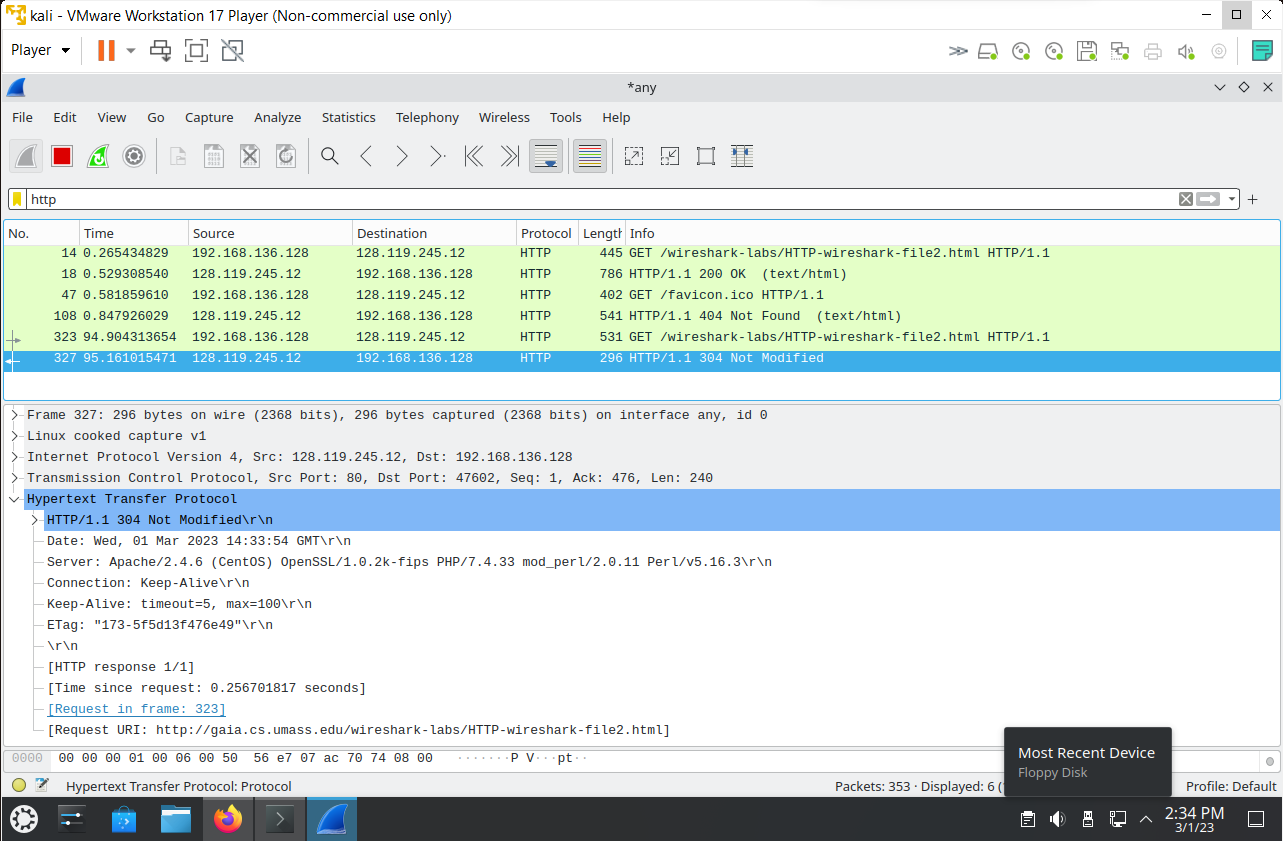
window, so that only captured HTTP messages will be displayed later in the packet

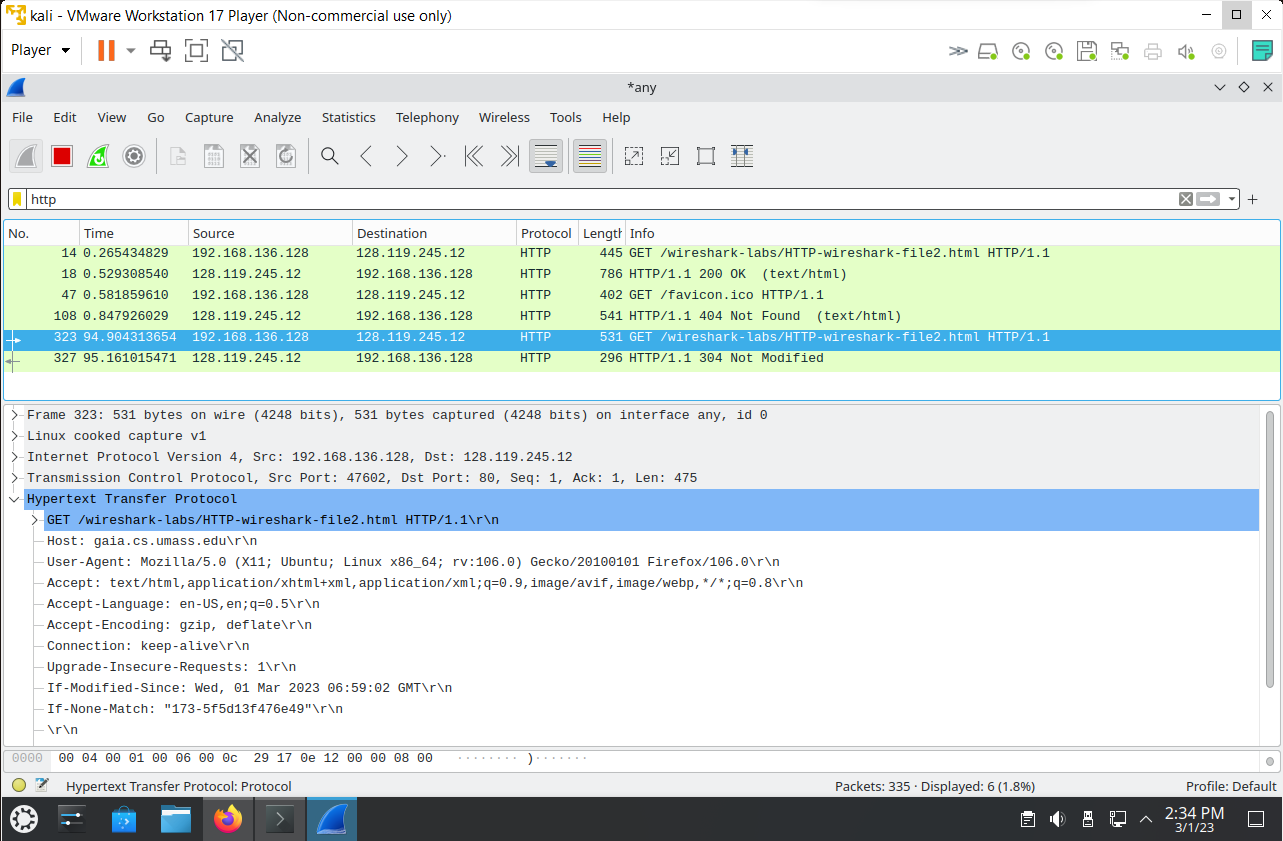
listing window.











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.