Requesting the Authentication:

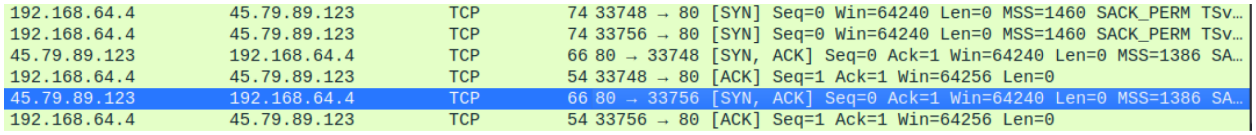
The connection starts with the HTTP three-way handshake. The client sends the server a [SYN] packet to connect to it. It roughly means “Hey, let’s talk.” In this case, the source port of the client was 39380, and the destination port was 80 (HTTP port of the server). The server replies with a [SYN, ACK] packet requesting to talk in both directions and acknowledging the reception of the packet from the client. It roughly means “Yeah let’s talk, but let’s talk in both directions. I also want to send you information.” Finally, the client acknowledges the reception of the packet sent by the server with an [ACK] packet.

I know this because the first lines in Wireshark contain:

192.168.64.4	45.79.89.123	TCP	[SYN]
45.79.89.123	192.168.64.4	TCP	[SYN, ACK]
192.168.64.4	45.79.89.123	TCP	[ACK]

However, as described in the previous assignment, two connections are initiated. In this assignment, however, only one of the connections makes requests and receives responses. That is the connection I will be following in the description below.

The screenshot below shows both connections being initiated:

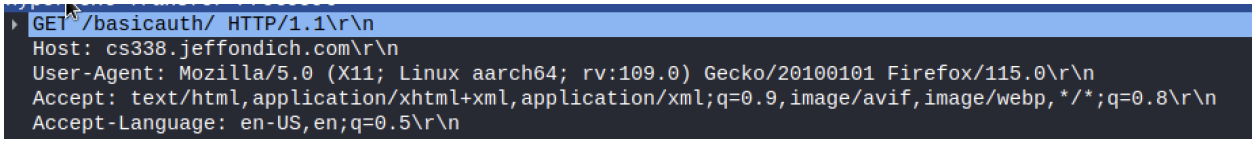


After the three-way handshake, the communication between the client and the server initiates. The very first thing the client does is request a page through a GET command. The client will request the page /basicauth/. I know this is the next step as the next line in Wireshark is:

192.168.64.4	45.79.89.123	HTTP	GET /basicauth/ HTTP/1.1

What is more, looking at the header of the request, I know that the name of the host is cs338.jeffondich.com, as one of the parameters is “Host: cs338.jeffondich.com.” The user agent is Mozilla/5.0 (User-Agent: Mozilla/5.0…), and the accepted languages are English and American English. I know this because inside the packet there is the following information:

Accept-Language: en-US, en;



The q=0.5 right beside language, according to stack overflow means relative quality factor and specifies which language the user would prefer, on a scale from 0 to 1. What is more, it can also be seen that the browser is waiting for an HTML application, an XML application or something that uses a mix of both. This can be seen by the line with the parameter Accept.

After that, the server then acknowledges the packet sent by the user with an [ACK] Packet. This can be seen because the next line in Wireshark is

45.79.89.123	192.168.64.4	TCP	[ACK]

Only after that, the server will return an HTTP packet to the user. The very next package sent by the server is:

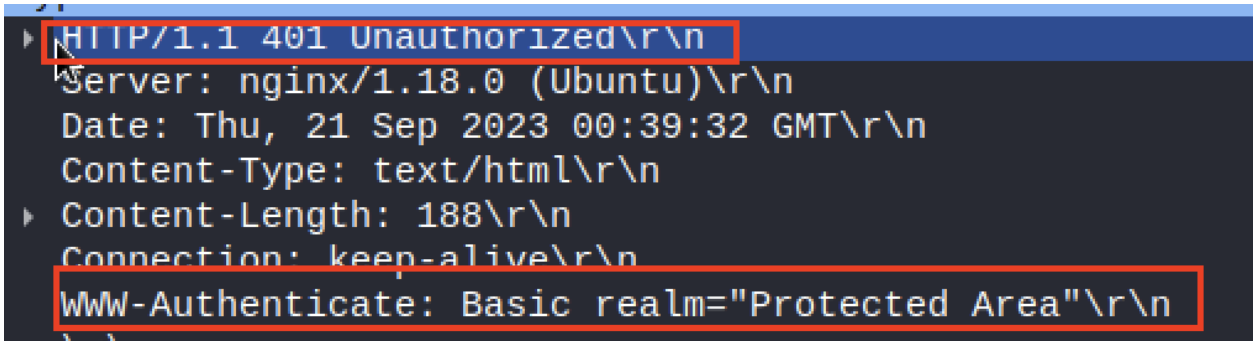
45.79.89.123 192.168.64.4 HTTP HTTP/1.1 401 Unauthorized

Let's study this packet in detail because it is what led the browser to ask for the user ID and password.

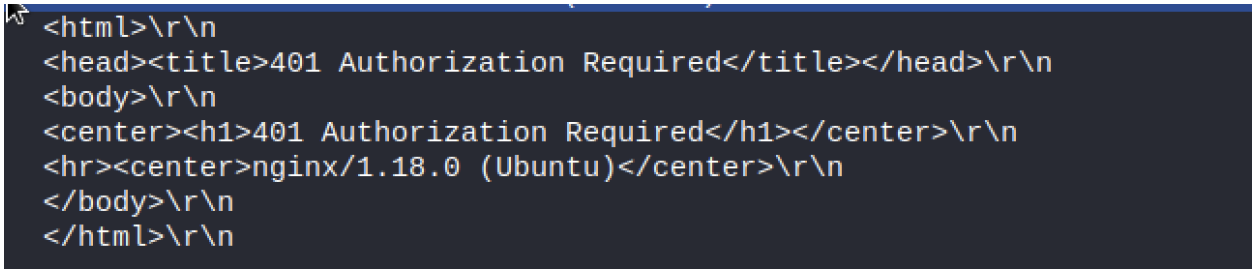
First of all, the error code sent by HTTP is 401 Unauthorized. According to Wikipedia, this means that authentication is required to access that page, but it has not yet been provided, or the user failed the authentication.

The HTTP also includes the field WWW-Authenticate: Basic realm="Protected Area"

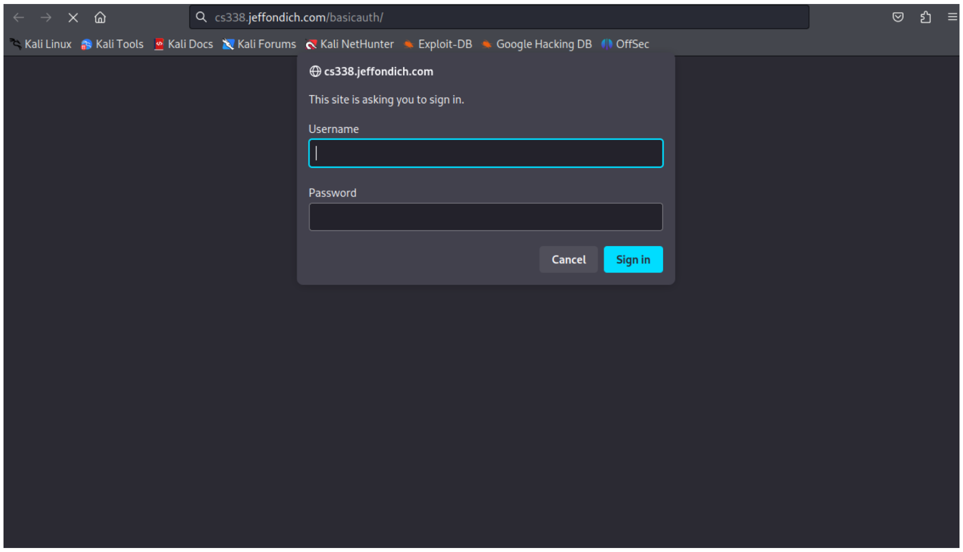
According to Wikipedia, this is required for HTTP headers with code 401. This is further confirmed by the official documentation of the 'Basic' HTTP Authentication. As seen in the documentation, and also described in class, the status code 401 combined with the WWW-Authenticate field in the request prompts the browser to ask the user for authentication.



The payload from this packet contains an HTML page that says that the user must be authorized to proceed. As seen in the screenshot below:



However, this was not rendered by the browser, as seen in the screenshot below. I assume that before rendering the web page, the browser saw the 401 status code and the WWW-Authenticate field and requested the user-ID and password. Other browsers might render the page before asking for authentication though.



After that, the browser seems to close one of the HTTP connections as there are [FIN, ACK] packets.

192.168.64.4	45.79.89.123	TCP	54 35138 → 80 [ACK] Seq=356 Ack=404 Win=64128 Len=0
192.168.64.4	45.79.89.123	TCP	54 35128 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
45.79.89.123	192.168.64.4	TCP	54 80 → 35128 [FIN, ACK] Seq=1 Ack=2 Win=64256 Len=0

This is followed by a series of [TCP KEEP ALIVE] and [TCP KEEP ALIVE ACK] packets. This, according to Imperva, indicates to the browser not to immediately close the connection after the first request (which is the default behavior). This way, the browser will wait a while to see if there are other requests and process them with the same connection. However, for this specific interaction, no other requests were made. So, after this series of [TCP KEEP ALIVE] and [TCP KEEP ALIVE ACK] packets, the second connection is closed with [FIN, ACK] packets. This can be seen in the following screenshots.

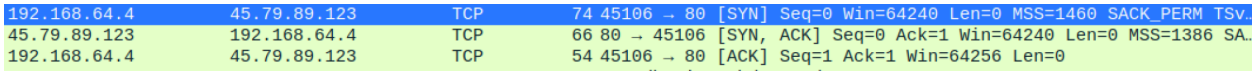
192.168.64.4	45.79.89.123	TCP	54 33748 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
45.79.89.123	192.168.64.4	TCP	54 80 → 33748 [FIN, ACK] Seq=1 Ack=2 Win=64256 Len=0
192.168.64.4	45.79.89.123	TCP	54 33748 → 80 [ACK] Seq=2 Ack=2 Win=64256 Len=0
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
192.168.64.4	45.79.89.123	TCP	54 [TCP Keep-Alive] 33756 → 80 [ACK] Seq=355 Ack=404 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54 [TCP Keep-Alive ACK] 80 → 33756 [ACK] Seq=404 Ack=356 Win=641…
45.79.89.123	192.168.64.4	TCP	54 80 → 33756 [FIN, ACK] Seq=404 Ack=356 Win=64128 Len=0
192.168.64.4	45.79.89.123	TCP	54 33756 → 80 [FIN, ACK] Seq=356 Ack=405 Win=64128 Len=0

Observe that after the first connection was closed the browser waited a while to close the second connection.

Now I will analyze what happens once the user types the password. In order to clean my prompt, I reloaded the Wireshark screen before typing the username and password. This, it will be easier to spot the packets I am interested in.

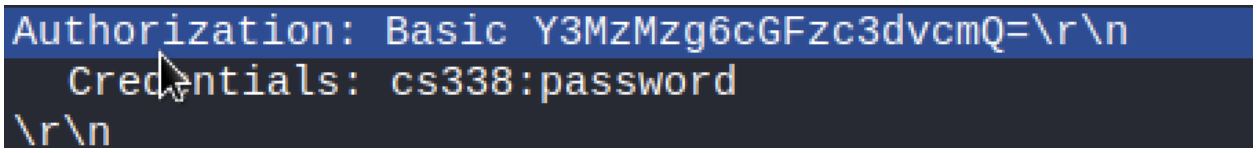
Successful Attempt:

I proceeded to type the correct username and password and pressed submit. As soon as I did a new connection to the the server was initiated. This makes sense, as the last connection to the server was closed before this interaction, as seen from the [FIN, ACK] packages described above. What is more, this new interaction starts with a three-way handshake initiated by the client. However, this time a single connection was initialized. This can be seen in the screenshot below:



After the connection was initialized, the client proceeded to request the page /basicauth/ just like in the last interaction. This can be seen as the very next packet sent was:

However, the contents of the HTTP packet are a bit different this time. First of all a new field can be seen in the packet. We can now see Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=
The string after Basic represents the username followed by a single colon followed by the password in base64, as described in the official documentation. This can be further confirmed by pasting the above string in a base64 decoder, the result will be cs338:password. Therefore, just as described in the official documentation, and as explained in class, the HTTP Basic authentication Scheme is not encrypted. All it does is convert the text typed by the user to Base64, which is by no means safe. Any computer which intercepts this packet can know exactly the username and password. To make matters worse, even Wireshark converts the value to a human-readable string.
The new field added and the Wireshark conversion to a human-readable string can be seen in the screenshot below:



This packet is then sent to the server. This can be validated as the next packet is an [ACK] packet sent from the server to the user.

The server then proceeds to check if the password sent is the correct one.
The validation happens on the server side. I know that, as the user sent the username and password to the server in the GET request as described above.
After seeing that the username and the password match, the server returns to the user the desired web page. This can be seen as the next packet sent by the server is

The packet has code 200, which, according to Wikipedia, means a successful HTTP request. This packet also contains the usual bookkeeping information, such as Server: nginx/1.18.0 (Ubuntu), Date: Thu, 21 Sep 2023 02:13:40 GMT, Content-Type: text/html, and others. These fields represent the type of server, date, and the contents of the file, but don't seem to contain any mention of the password.

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Thu, 21 Sep 2023 02:13:40 GMT\r\n
  Content-Type: text/html\r\n
  Transfer-Encoding: chunked\r\n
  Connection: keep-alive\r\n
  Content-Encoding: gzip\r\n
  \r\n
  [HTTP response 1/2]
  [Time since request: 0.054360781 seconds]
  [Request in frame: 4]
  [Next request in frame: 8]
  [Next response in frame: 10]
```

The request also contains the HTML of the page, which was rendered on the browser.

```
Line-based text data: text/html (9 lines)
  <html>\r\n
  <head><title>Index of /basicauth/</title></head>\r\n
  <body>\r\n
  <h1>Index of /basicauth/</h1><hr><pre><a href="../">../</a>
  <a href="amateurs.txt">amateurs.txt</a>
  <a href="armed-guards.txt">armed-guards.txt</a>
  <a href="dancing.txt">dancing.txt</a>
  </pre><hr></body>\r\n
  </html>\r\n
```

Index of /basicauth/

../		
amateurs.txt	04-Apr-2022 14:10	75
armed-guards.txt	04-Apr-2022 14:10	161
dancing.txt	04-Apr-2022 14:10	227

After all of that, I successfully got access to the page. But it was not done loading just yet. Immediately after the package with the HTML was received, the user acknowledges that through an [ACK] packet.

Then the user requests the icon of the page, through GET /favicon.ico. The server acknowledges the request through an [ACK] packet but returns the status 404 Not Found, meaning that the icon of the page was not found in the server. This can be validated by looking at the following Wireshark packets:

45.79.89.123	192.168.64.4	HTTP	458	HTTP/1.1 200 OK (text/html)
192.168.64.4	45.79.89.123	TCP	54	45106 → 80 [ACK] Seq=399 Ack=405 Win=64128 Len=0
192.168.64.4	45.79.89.123	HTTP	369	GET /favicon.ico HTTP/1.1
45.79.89.123	192.168.64.4	TCP	54	80 → 45106 [ACK] Seq=405 Ack=714 Win=64128 Len=0
45.79.89.123	192.168.64.4	HTTP	383	HTTP/1.1 404 Not Found (text/html)

After this, the page was successfully loaded and no other queries were made to the server. The server tries to keep the connection alive for a while, but soon finalizes it through [FIN] packets.

45.79.89.123	192.168.64.4	HTTP	383	HTTP/1.1 404 Not Found (text/html)
192.168.64.4	45.79.89.123	TCP	54	45106 → 80 [ACK] Seq=714 Ack=734 Win=64128 Len=0
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
192.168.64.4	45.79.89.123	TCP	54	[TCP Keep-Alive] 45106 → 80 [ACK] Seq=713 Ack=734 Win=64128 L…
45.79.89.123	192.168.64.4	TCP	54	[TCP Keep-Alive ACK] 80 → 45106 [ACK] Seq=734 Ack=714 Win=641…
45.79.89.123	192.168.64.4	TCP	54	80 → 45106 [FIN, ACK] Seq=734 Ack=714 Win=64128 Len=0
192.168.64.4	45.79.89.123	TCP	54	45106 → 80 [FIN, ACK] Seq=714 Ack=735 Win=64128 Len=0

Last but not least, let's see what happens when we try to access one of the pages on the website.

I tried accessing amateurs.txt. I will describe the series of packets sent and received during this interaction. Similar to what was described above, the first thing that happened was the client trying to establish a connection to the server. This is necessary as the previous connection was finalized, as shown above.

As further proof that the connection was established, the client executed the three-way handshake. However, this time the client established two connections to the server.

45.79.89.123	TCP	74	44250 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSv…
45.79.89.123	TCP	74	44266 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSv…
192.168.64.4	TCP	66	80 → 44250 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1386 SA…
45.79.89.123	TCP	54	44250 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
192.168.64.4	TCP	66	80 → 44266 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1386 SA…
45.79.89.123	TCP	54	44266 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0

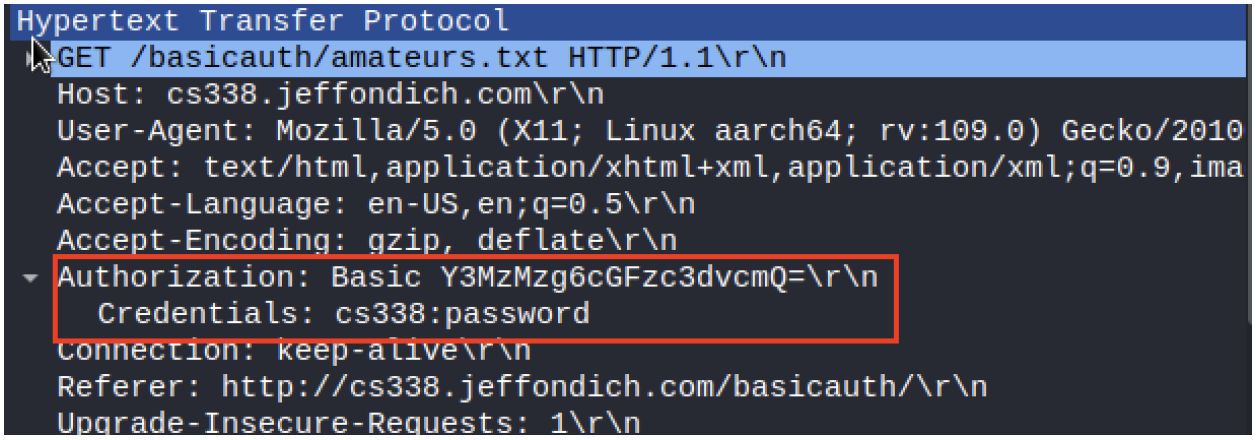
Immediately after that, the user requests the desired page, through GET /basicauth/amateurs.txt, as can be seen in the following packet

192.168.64.4	45.79.89.123	HTTP	513	GET /basicauth/amateurs.txt HTTP/1.1

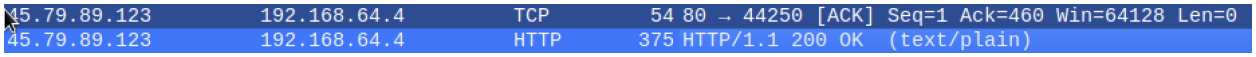
As we are already authenticated, the packet sent in this request also contains Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=

This is very similar to the GET packet the browser sent right after we typed the correct username and password for the first time. Both of them contain the same value for the Authorization field. This makes me believe that the username and password were stored in the Browser, not in the server.

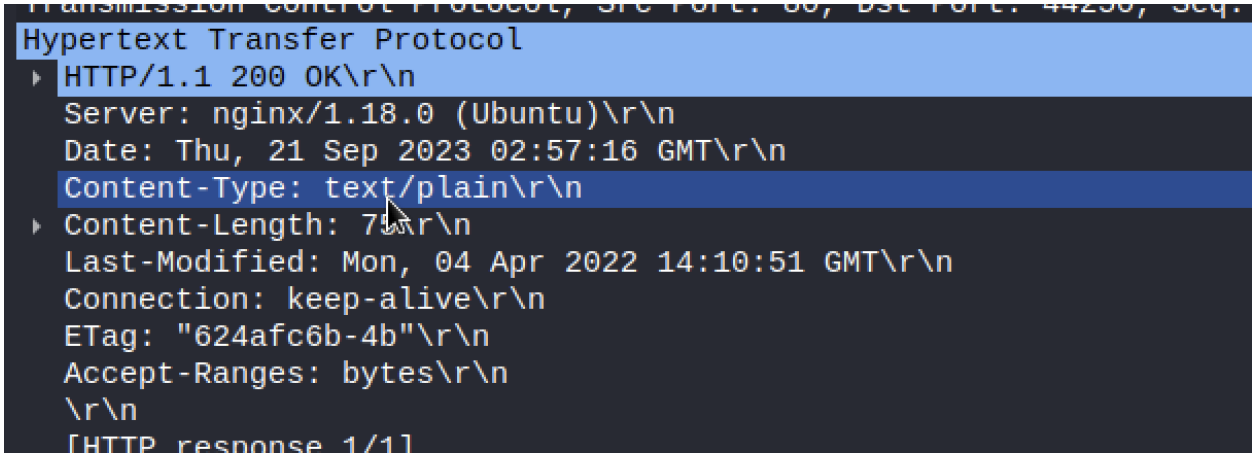
The contents of the Packet can be seen in the following screenshot:



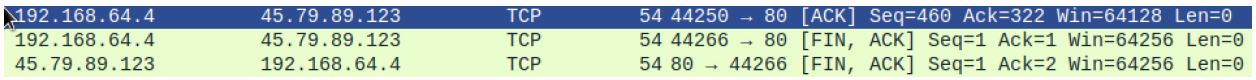
The page basicauth/amateurs.txt is password protected, and as specified by the Basic HTTP Authentication documentation, I was able to access it without having to type my password again. This happened because the browser added my credentials to the request.
After the request was sent, the server sends an [ACK] packet, confirming that it received the user request and returns the desired page. The latter can be confirmed as the server also sent an HTTP packet containing the status code 200 OK



This packet also does not contain any information relating to the user-id and password, and most of its data is bookkeeping information:



After that packet was received, the user confirms they received it through an [ACK] packet and then the connection is finalized.



Sources:

Base64 decode and encode - online. Base64 Decode. (n.d.). https://www.base64decode.org/

Reschke, J. (2015, September 30). *RFC 7617: The “BASIC” HTTP authentication scheme*. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc7617

What is HTTP keep alive: Benefits of Connection Keep alive: Imperva. Learning Center. (n.d.). https://www.imperva.com/learn/performance/http-keep-alive/

What is Q=0.5 in accept HTTP headers?*. Stack Overflow. (2011, December 18). https://stackoverflow.com/questions/8552927/what-is-q-0-5-in-accept-http-headers

Wikimedia Foundation. (2023, August 12). *List of HTTP status codes*. Wikipedia. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes