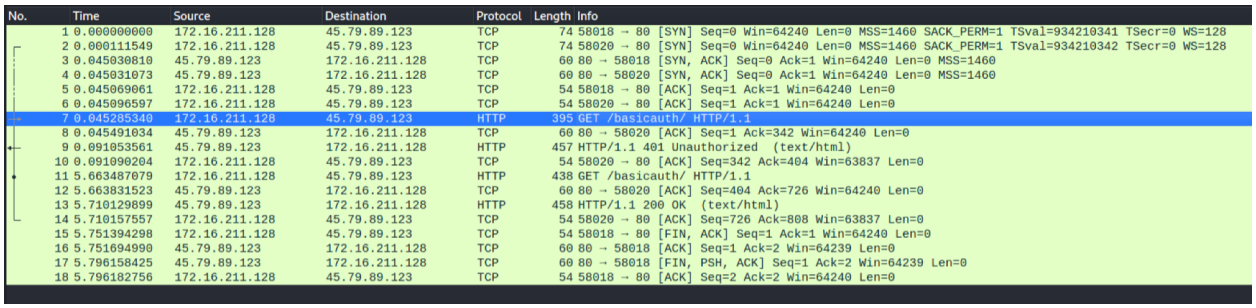
To understand the mechanism behind HTTP Basic Authentication, there are three cases that I am considering:

Initial request, correct credentials:
Here, access to the website http://cs338.jeffondich.com/basicauth/ is initiated through a new browser tab with no previous access to the website, and the credentials are not logged in browser memory. Furthermore, the password is entered correctly within the first attempt.

Here are the frames exchanged between the server and the client within this interaction.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.211.128	45.79.89.123	TCP	74	58018 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=934210341 TSecr=0 WS=128
2	0.000111549	172.16.211.128	45.79.89.123	TCP	74	58020 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=934210342 TSecr=0 WS=128
3	0.045030810	45.79.89.123	172.16.211.128	TCP	60	80 → 58018 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4	0.045031073	45.79.89.123	172.16.211.128	TCP	60	80 → 58020 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.045069061	172.16.211.128	45.79.89.123	TCP	54	58018 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.045096597	172.16.211.128	45.79.89.123	TCP	54	58020 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	0.045285340	172.16.211.128	45.79.89.123	HTTP	395	GET /basicauth/ HTTP/1.1
8	0.045491034	45.79.89.123	172.16.211.128	TCP	60	80 → 58020 [ACK] Seq=1 Ack=342 Win=64240 Len=0
9	0.091053561	45.79.89.123	172.16.211.128	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
10	0.091090204	172.16.211.128	45.79.89.123	TCP	54	58020 → 80 [ACK] Seq=342 Ack=404 Win=63837 Len=0
11	5.663487079	172.16.211.128	45.79.89.123	HTTP	438	GET /basicauth/ HTTP/1.1
12	5.663831523	45.79.89.123	172.16.211.128	TCP	60	80 → 58020 [ACK] Seq=404 Ack=726 Win=64240 Len=0
13	5.710129899	45.79.89.123	172.16.211.128	HTTP	458	HTTP/1.1 200 OK (text/html)
14	5.710157557	172.16.211.128	45.79.89.123	TCP	54	58020 → 80 [ACK] Seq=726 Ack=808 Win=63837 Len=0
15	5.751394298	172.16.211.128	45.79.89.123	TCP	54	58018 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
16	5.751694990	45.79.89.123	172.16.211.128	TCP	60	80 → 58018 [ACK] Seq=1 Ack=2 Win=64239 Len=0
17	5.796158425	45.79.89.123	172.16.211.128	TCP	60	80 → 58018 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
18	5.796182756	172.16.211.128	45.79.89.123	TCP	54	58018 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0

Figure 1: Frames sent within case 1 interaction

Since we are only interested in the HTTP session, let's ignore the TCP protocols. The first frame with HTTP protocol is Frame 7 is a request from the client to the server, as the source IP address is 172.16.211.128 and the destination IP address is 45.79.89.123, which is the IP address of the host cs338.jeffondich.com. The following figure shows the header attached to this HTTP protocol.

```
Hypertext Transfer Protocol
  GET /basicauth/ HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  \r\n
  [Full request URI: http://cs338.jeffondich.com/basicauth/]
  [HTTP request 1/2]
  [Response in frame: 9]
  [Next request in frame: 11]
```

Figure 2: HTTP header, Frame 7

We can verify that the host is cs338.jeffondich.com, as shown in the second line. We have the following information in the header:

- The request is GET /basicauth/ HTTP/1.1, which is in line with what we were trying to do when we accessed the website.

- User-Agent identifies the Operating System and the Browser of the web-server, which in this case is Mozilla Firefox as expected.
- Accept indicates the content types that the clients can understand. In this header, the Accept request HTTP header says that the clients can read text files, applications, and images.
- Accept-Language is English (US) and English.
- Accept-Encoding indicates the content encoding (in this case, gzip which is a compression algorithm) that the client can understand.

As we can see, Frame 7 is a request header that is sent to the server, from the client, to indicate a request for access to the website /basicauth/ on the host cs338.jeffondich.com. This is a standard request HTTP header with nothing interesting yet except for the following:

- Response in frame: 9. This line indicates that the response is sent with frame 9.
- Next request in frame: 11. This line indicates that the next request is sent within frame 11.

If we look at Frame 9, we see the following.

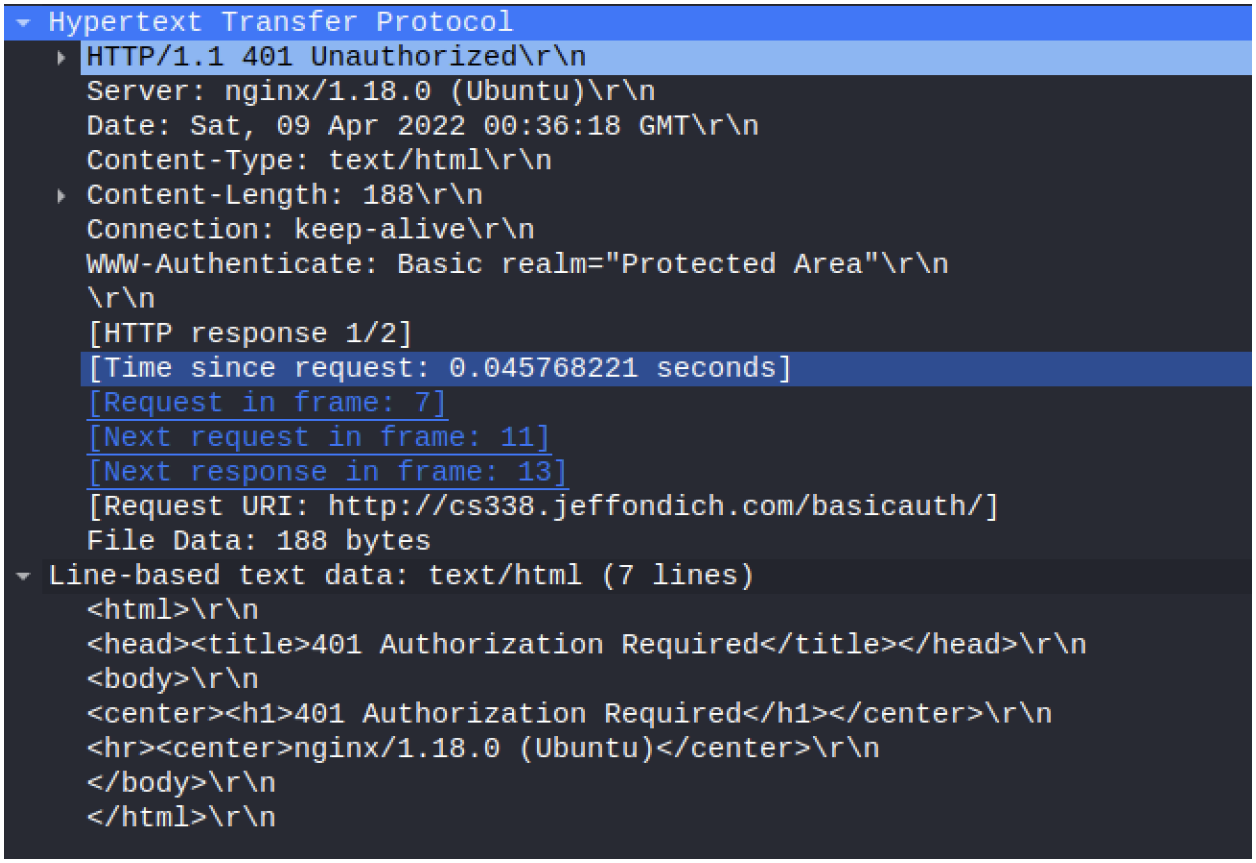


Figure 3: HTTP header, Frame 9

Here is where something interesting happens. This is the beginning of what is referred to in RFC 7617 as the 'Basic' Authentication Scheme. Here, we see response HTTP/1.1 401

Unauthorized, which lets us know that we do not have authorization to access this website (for now). Let's take a look at what goes on in this line:

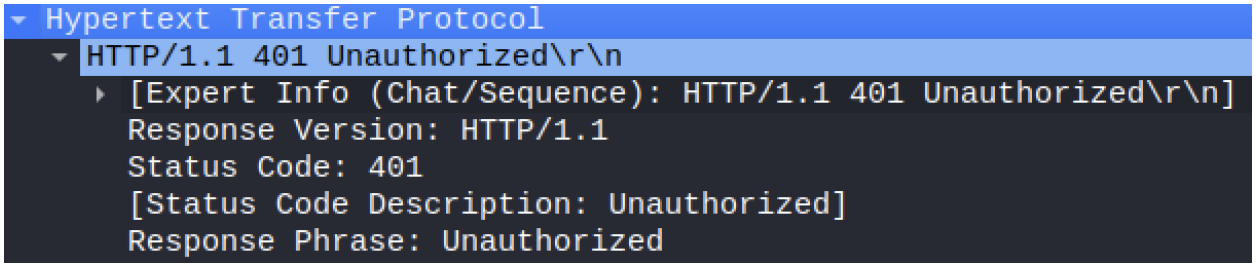


Figure 4: HTTP/1.1 401 Unauthorized information

Here, we can see that the response version and the Status Code (both in numerical value and description) are recorded. This is a response to a request for a protected source.
There is also a challenge recorded in this response header, as seen in Figure 3: WWW-Authenticate: Basic realm="Protected Area". Here, just like what is described in RFC 7617, the scheme name "Basic" and the authentication parameter 'realm' is recorded. This line outlines the challenge that the client must fulfill to authenticate itself with the server. The client can do so by including an Authorization request header with credentials, often in the form of presenting a password prompt to the user and then issuing the request including the correct Authorization header. Based on the request header in Figure 2 and response header in Figure 3, we can expect this request to be issued 11 ([Next request in frame 11])/

And this is what we see in Frame 11, after the user has entered the password.

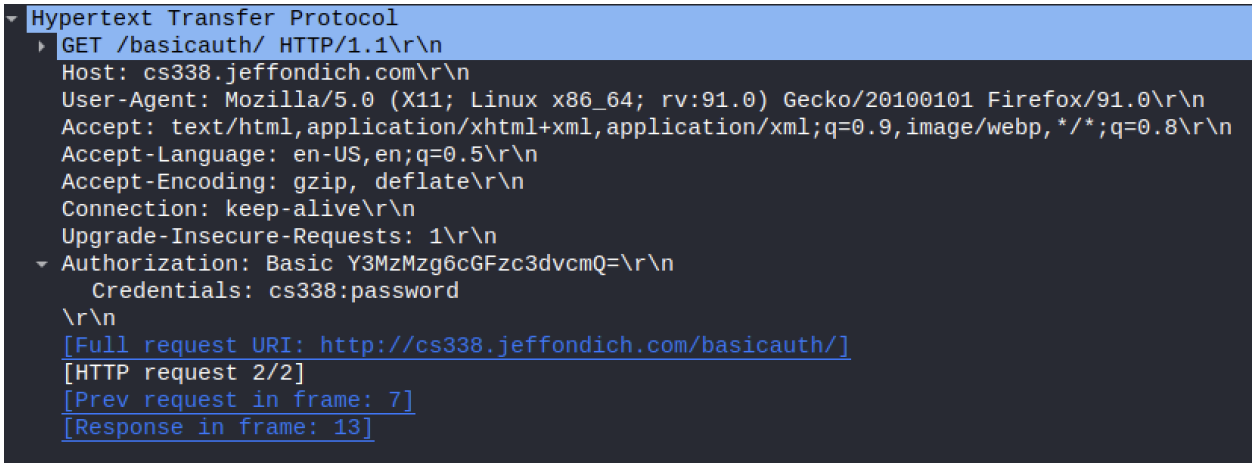


Figure 5: HTTP header, Frame 11

Here we see the same method, path, and protocol as the request in frame 7: GET /basicauth/ HTTP/1.1. Everything else is the same, except for Authorization: Basic y3mZmZg6cGFzc3dvmQ=. This authorization includes the scheme name "Basic" and a long string. Underneath, we can see that the credentials are recorded in the form

user-name:password (with a ':' to separate them), which is then encoded into the long string above in base64. It

Now, we expect the next HTTP frame to respond to the request and verify the credentials. Now, it is hard to see whether or not the browser checks the password itself before sending it to the server or the password is just sent to the server, but this is why we are studying three separate cases. We will come back to this after looking at case 2. However, in this particular case, we can see that according to figure 5, the response header is included in Frame 13.

And this is Frame 13:

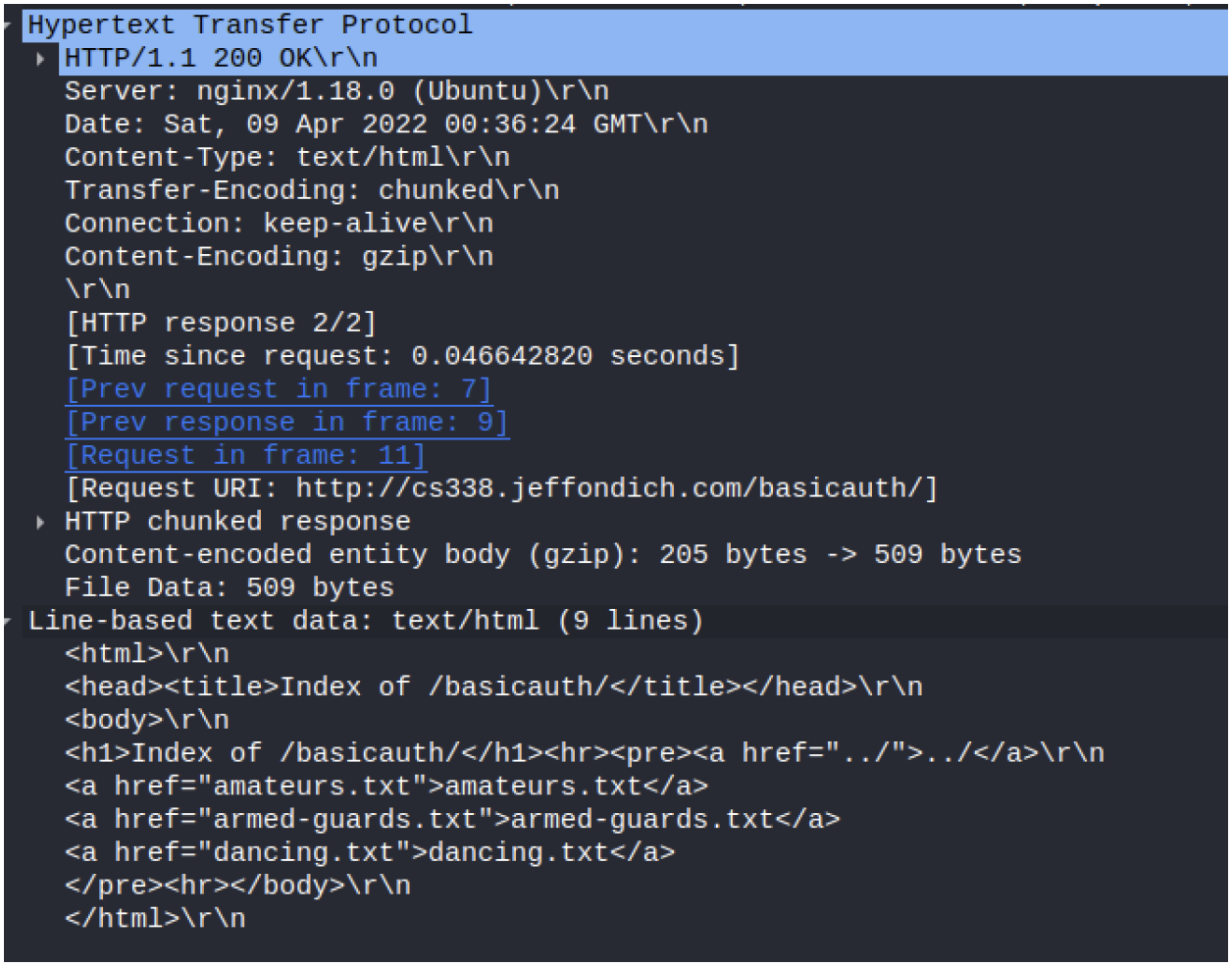


Figure 6: HTTP header, Frame 13

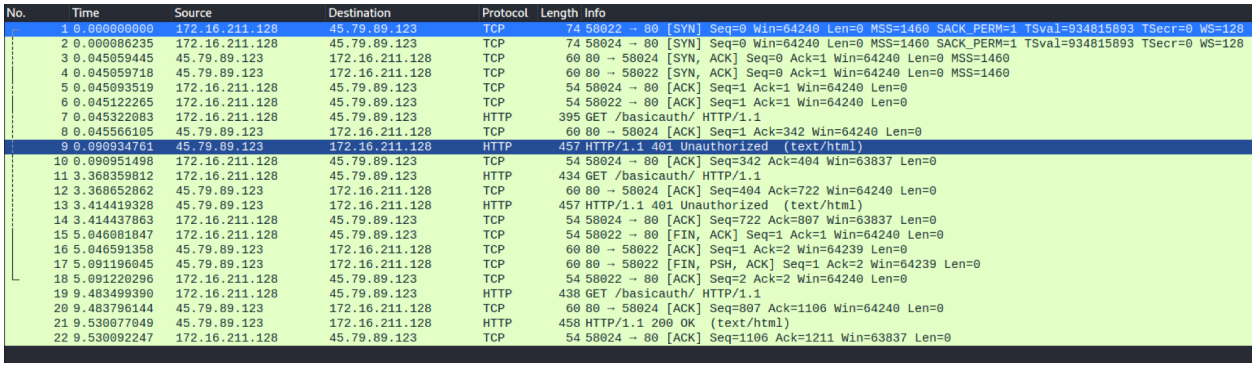
The first line included the protocol and status code 200 OK, which grants the client access to the protected website. Server and date are self-explanatory. Content-Type lets the user know the content type of the website. We can verify that this response header is in response to request in frame 11 as predicted above in the line [Request in frame: 11].
This seems pretty standard and straightforward: user requests access to website, server requests authentication by sending the status code "401 Unauthorized" to let the client know

that credentials are requested. Client then sent back the request for the website, but now with authorization credentials. Server verifies this request and grants access through the status code "200 OK".

Now, this is in a perfect world where the user enters the correct credentials on first try. This leads us to case 2.

Initial request, incorrect credentials:
Here, just like in the first scenario, access to the website http://cs338.jeffondich.com/basicauth/ is initiated through a new browser tab with no previous access to the website, and the credentials are not logged in browser memory. However, unlike the first scenario, the credentials are incorrect in the first attempt. Here, we are limiting our incorrect credentials to one try only. The case where the user enters multiple incorrect credentials before getting it correctly follows the same pattern.

Here are the frames exchanged between the server and the client within this interaction.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.211.128	45.79.89.123	TCP	74	58022 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=934815893 TSecr=0 WS=128
2	0.000086235	172.16.211.128	45.79.89.123	TCP	74	58024 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=934815893 TSecr=0 WS=128
3	0.045059445	45.79.89.123	172.16.211.128	TCP	60	80 → 58024 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4	0.045059718	45.79.89.123	172.16.211.128	TCP	60	80 → 58022 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.045093519	172.16.211.128	45.79.89.123	TCP	54	58024 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.045122265	172.16.211.128	45.79.89.123	TCP	54	58022 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	0.045322083	172.16.211.128	45.79.89.123	HTTP	395	GET /basicauth/ HTTP/1.1
8	0.045566105	45.79.89.123	172.16.211.128	TCP	60	80 → 58024 [ACK] Seq=1 Ack=342 Win=64240 Len=0
9	0.090934761	45.79.89.123	172.16.211.128	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
10	0.090951498	172.16.211.128	45.79.89.123	TCP	54	58024 → 80 [ACK] Seq=342 Ack=404 Win=63837 Len=0
11	3.368359812	172.16.211.128	45.79.89.123	HTTP	434	GET /basicauth/ HTTP/1.1
12	3.368652862	45.79.89.123	172.16.211.128	TCP	60	80 → 58024 [ACK] Seq=404 Ack=722 Win=64240 Len=0
13	3.414419328	45.79.89.123	172.16.211.128	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
14	3.414437863	172.16.211.128	45.79.89.123	TCP	54	58024 → 80 [ACK] Seq=722 Ack=807 Win=63837 Len=0
15	5.046081847	172.16.211.128	45.79.89.123	TCP	54	58022 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
16	5.046591358	45.79.89.123	172.16.211.128	TCP	60	80 → 58022 [ACK] Seq=1 Ack=2 Win=64239 Len=0
17	5.091196045	45.79.89.123	172.16.211.128	TCP	60	80 → 58022 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
18	5.091220296	172.16.211.128	45.79.89.123	TCP	54	58022 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
19	9.483499390	172.16.211.128	45.79.89.123	HTTP	438	GET /basicauth/ HTTP/1.1
20	9.483796144	45.79.89.123	172.16.211.128	TCP	60	80 → 58024 [ACK] Seq=807 Ack=1106 Win=64240 Len=0
21	9.530077049	45.79.89.123	172.16.211.128	HTTP	458	HTTP/1.1 200 OK (text/html)
22	9.530092247	172.16.211.128	45.79.89.123	TCP	54	58024 → 80 [ACK] Seq=1106 Ack=1211 Win=63837 Len=0

Figure 7: Frames sent within case 2 interaction

Here, we can see that the interaction follows the interaction in case 1 nicely, except that in Frame 13, instead of seeing HTTP/1.1 200 OK like the first case, we get HTTP/1.1 401 Unauthorized. This is a response header, so let's look at its corresponding request header which is in Frame 11:

```
Hypertext Transfer Protocol
  GET /basicauth/ HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /basicauth/ HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /basicauth/
    Request Version: HTTP/1.1
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Authorization: Basic aGVsbG86aGVsbG8=\r\n
    Credentials: hello:hello
  \r\n
  [Full request URI: http://cs338.jeffondich.com/basicauth/]
  [HTTP request 2/3]
  [Prev request in frame: 7]
  [Response in frame: 13]
  [Next request in frame: 19]
```

Figure 8: HTTP header, Frame 11

Here, we can see that the only thing that is different from Frame 11 in the first case is the Authorization section. Here, the credentials is hello:hello, which is then encoded in base64. The response to this is requested in Frame 13, as indicated above.

```
Hypertext Transfer Protocol
  HTTP/1.1 401 Unauthorized\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 401 Unauthorized\r\n]
    Response Version: HTTP/1.1
    Status Code: 401
    [Status Code Description: Unauthorized]
    Response Phrase: Unauthorized
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Sat, 09 Apr 2022 00:46:27 GMT\r\n
  Content-Type: text/html\r\n
  Content-Length: 188\r\n
  Connection: keep-alive\r\n
  WWW-Authenticate: Basic realm="Protected Area"\r\n
  \r\n
  [HTTP response 2/3]
  [Time since request: 0.046059516 seconds]
  [Prev request in frame: 7]
  [Prev response in frame: 9]
  [Request in frame: 11]
  [Next request in frame: 19]
  [Next response in frame: 21]
  [Request URI: http://cs338.jeffondich.com/basicauth/]
  File Data: 188 bytes
Line-based text data: text/html (7 lines)
  <html>\r\n
  <head><title>401 Authorization Required</title></head>\r\n
  <body>\r\n
  <center><h1>401 Authorization Required</h1></center>\r\n
  <hr><center>nginx/1.18.0 (Ubuntu)</center>\r\n
  </body>\r\n
  </html>\r\n
```

Figure 7: HTTP header, Frame 13

Here the server sent back an Unauthorized status code, informing the client that the client still does not have authorization for this website. We see once again the challenge which informs that the client needs to prompt the user to enter another credential. This is reflected in the Line-based text data section where Authorization Required is the content of the frame. When entering the correct credentials, as shown in frame 19:

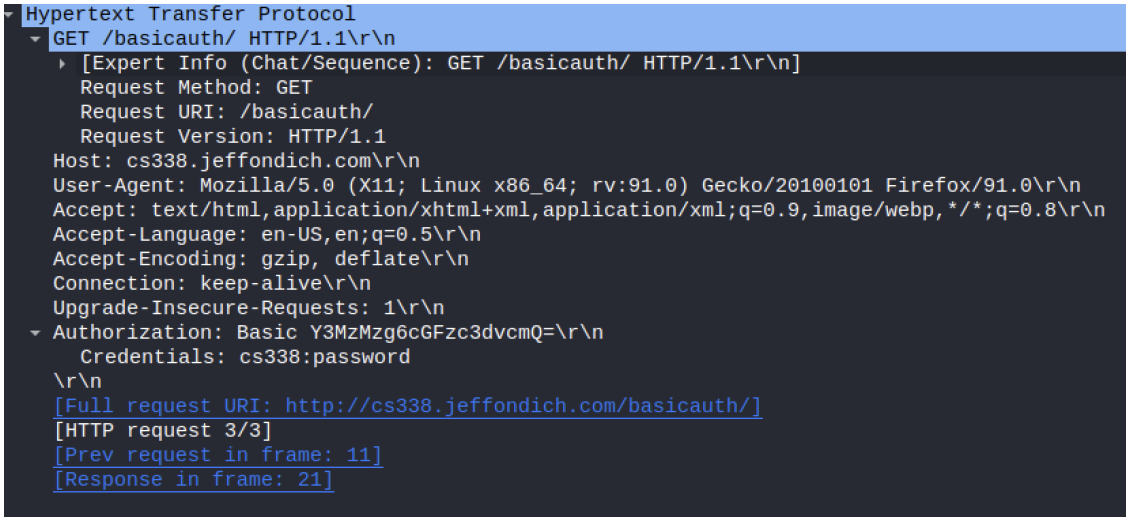


Figure 8: HTTP header, Frame 19

server responds with 200 OK status code in Frame 21:

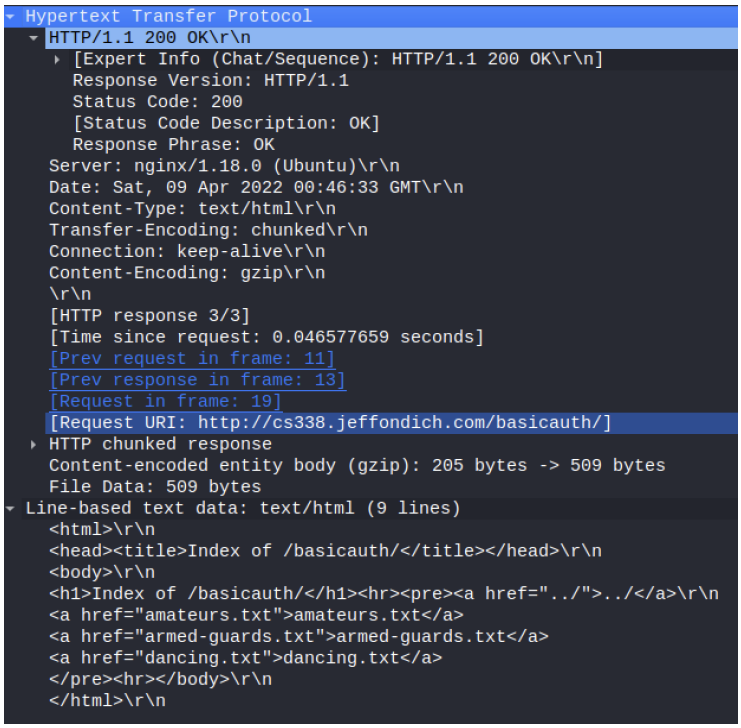


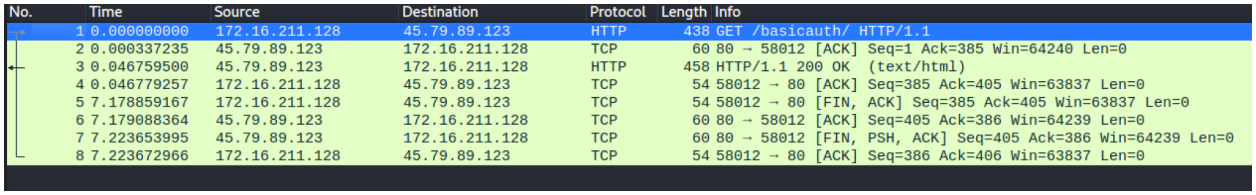
Figure 9: HTTP Header, Frame 21

Here, we can see that despite the credentials, the client sends the server the credentials and it is up to the server to verify them and respond appropriately. And obviously, the password is not encrypted. It is sent in clear text, with the additional encoding to base64. The next case is a bit trivial, but it is interesting to see nonetheless.

Post-authorization request

Here, access to the website http://cs338.jeffondich.com/basicauth/ is initiated through the same browser that has previously accessed the website successfully, and the credentials are logged in browser memory.

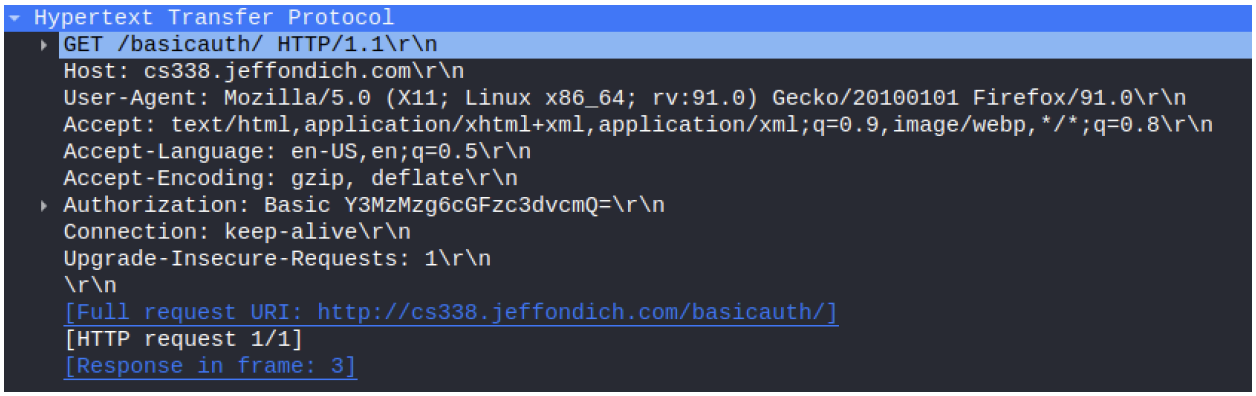
Here are the frames exchanged between the server and the client within this interaction.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.211.128	45.79.89.123	HTTP	438	GET /basicauth/ HTTP/1.1
2	0.000337235	45.79.89.123	172.16.211.128	TCP	60	80 → 58012 [ACK] Seq=1 Ack=385 Win=64240 Len=0
3	0.046759500	45.79.89.123	172.16.211.128	HTTP	458	HTTP/1.1 200 OK (text/html)
4	0.046779257	172.16.211.128	45.79.89.123	TCP	54	58012 → 80 [ACK] Seq=385 Ack=405 Win=63837 Len=0
5	7.178859167	172.16.211.128	45.79.89.123	TCP	54	58012 → 80 [FIN, ACK] Seq=385 Ack=405 Win=63837 Len=0
6	7.179088364	45.79.89.123	172.16.211.128	TCP	60	80 → 58012 [ACK] Seq=405 Ack=386 Win=64239 Len=0
7	7.223653995	45.79.89.123	172.16.211.128	TCP	60	80 → 58012 [FIN, PSH, ACK] Seq=405 Ack=386 Win=64239 Len=0
8	7.223672966	172.16.211.128	45.79.89.123	TCP	54	58012 → 80 [ACK] Seq=386 Ack=406 Win=63837 Len=0

Figure 10: Frames sent within case 3 interaction

Here, we can see a lack of request for authorization. We don't see any 401 Unauthorized status code, and the exchange happens as if there is no need for authentication (which corresponds to the lack of prompt for password and credentials on the client's end when we access the website). There are a total of 2 HTTP request and response, with the request header in Frame 1:



```
▾ Hypertext Transfer Protocol
  ▸ GET /basicauth/ HTTP/1.1\r\n
    Host: cs338.jeffondich.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
  ▸ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://cs338.jeffondich.com/basicauth/]
    [HTTP request 1/1]
    [Response in frame: 3]
```

Figure 11: HTTP header, Frame 1

and the response header in Frame 3:

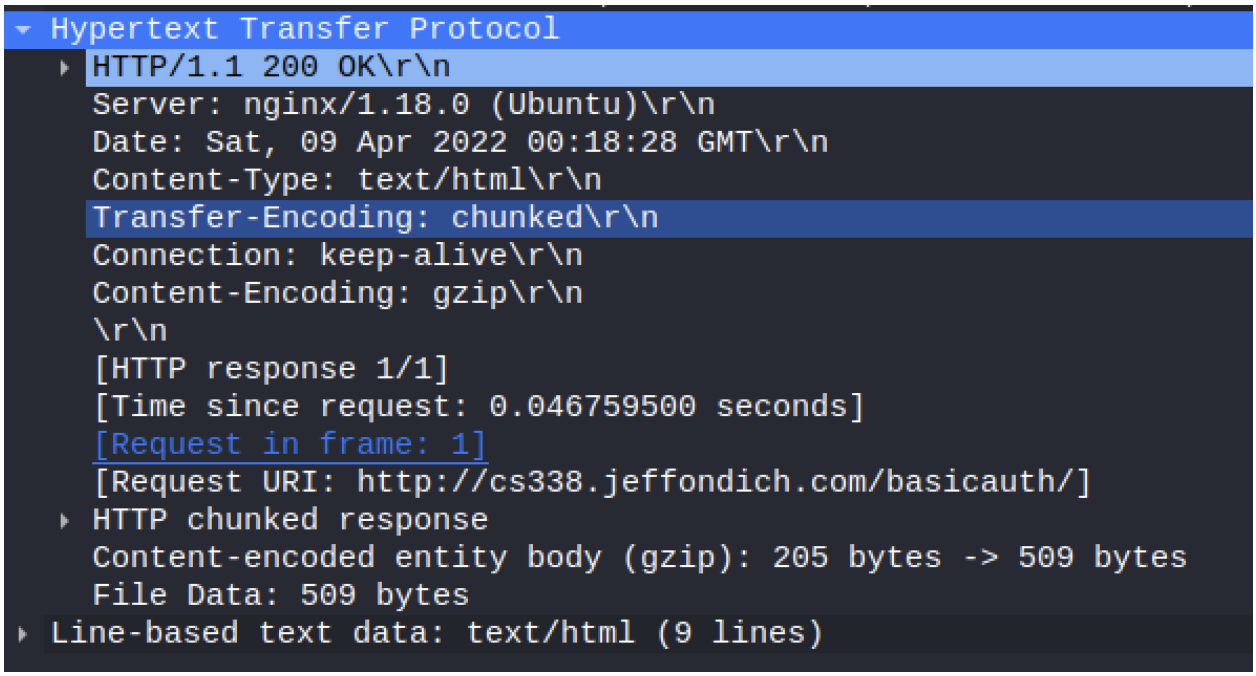


Figure 12: HTTP header, Frame 3

The line-based text data information confirms that we successfully accessed the website.

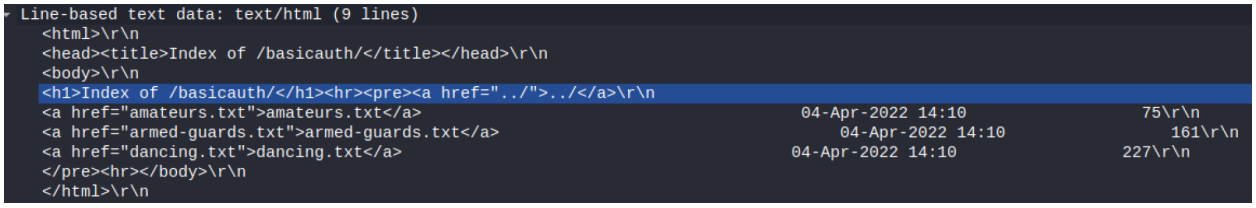


Figure 13: Frame 3 Line-based text data

As we can clearly see, there is a lack of request for authentication if the website has been accessed successfully before using the same browser session.

Reference:

1. https://datatracker.ietf.org/doc/html/rfc7617