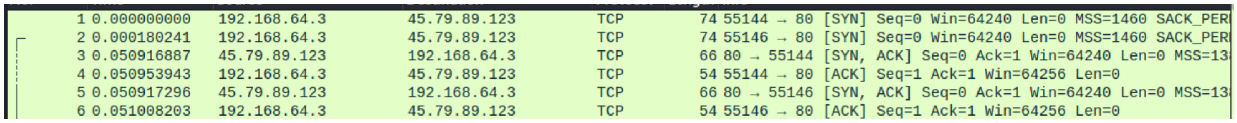
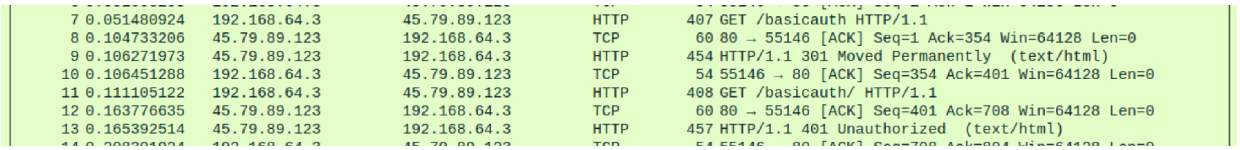
In order for my browser to first connect to Jeff's server, they perform the **TCP handshake**. My browser, which is located at the IP address 192.168.64.3, sends a **SYN** request to the server, which is located at the IP address 45.79.89.123. This can be seen in frame 1, and essentially asks the server if it wants to establish a connection between itself and the browser. The browser then sends another **SYN** request to the server, as seen in frame 2. Then, the server sends back a **SYN, ACK** in frame 3, which means that it has received the request and is willing to establish that connection. In frame 4, my browser lets the server know through an **ACK** that it has acknowledged that the connection is established. The second connection is set up likewise in frames 5 and 6. Finally, the TCP handshake is complete and my browser is connected to the server.

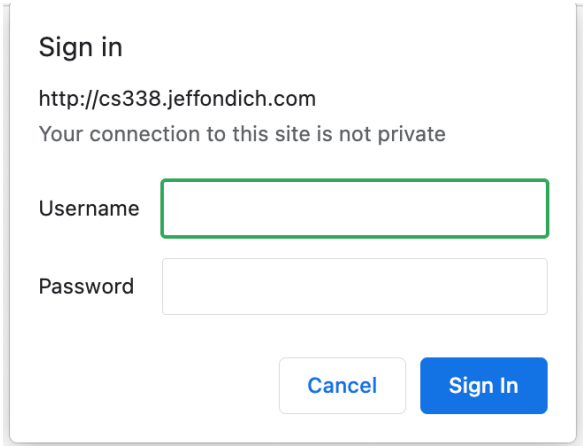


1	0.000000000	192.168.64.3	45.79.89.123	TCP	74	55144 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
2	0.000180241	192.168.64.3	45.79.89.123	TCP	74	55146 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PER
3	0.050916887	45.79.89.123	192.168.64.3	TCP	66	80 → 55144 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=13
4	0.050953943	192.168.64.3	45.79.89.123	TCP	54	55144 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
5	0.050917296	45.79.89.123	192.168.64.3	TCP	66	80 → 55146 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=13
6	0.051008203	192.168.64.3	45.79.89.123	TCP	54	55146 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0

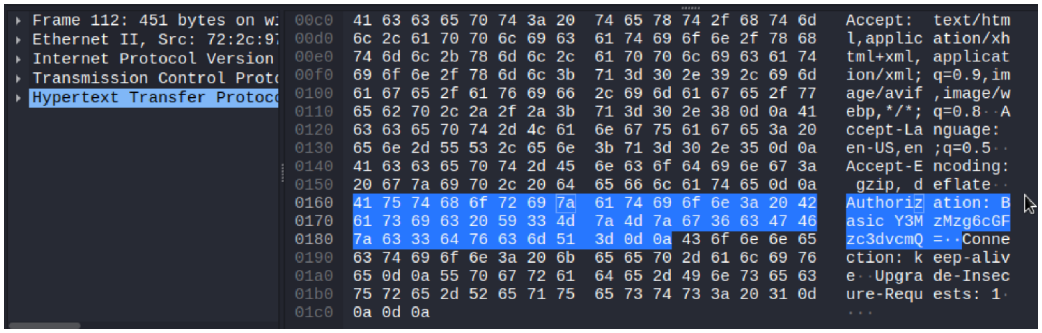
Once my browser is connected to the server, then it starts sending **HTTP requests** to retrieve the page that I have asked it to retrieve in the URL. In this case, I wanted to visit the basicauth page, since the URL inputted into the browser was http://cs338.jeffondich.com/**basicauth**. Thus, my browser sends a **GET request** to the server, asking for the /basicauth page, as seen in frame 7. Interestingly, since I entered /basicauth rather than /basicauth/ into the browser, we can see in frame 9 that the server responded with a **301 Moved Permanently error code**. This reroutes the browser to /basicauth/, and creates a new HTTP GET request for /basicauth/ in frame 11. Then, the response back from the server is a **401 Unauthorized error code** in frame 13. This prompts the browser to ask me, the user, for a username and passcode in order to authenticate who I am.



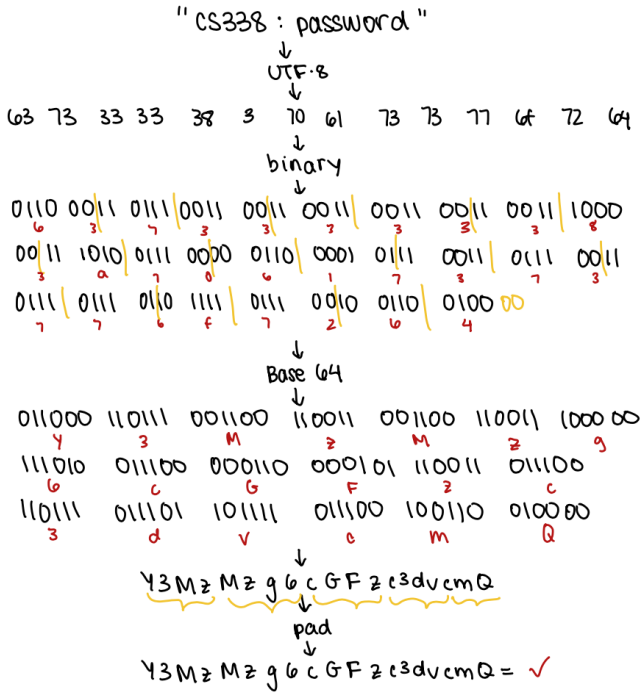
7	0.051480924	192.168.64.3	45.79.89.123	HTTP	407	GET /basicauth HTTP/1.1
8	0.104733206	45.79.89.123	192.168.64.3	TCP	60	80 → 55146 [ACK] Seq=1 Ack=354 Win=64128 Len=0
9	0.106271973	45.79.89.123	192.168.64.3	HTTP	454	HTTP/1.1 301 Moved Permanently (text/html)
10	0.106451288	192.168.64.3	45.79.89.123	TCP	54	55146 → 80 [ACK] Seq=354 Ack=401 Win=64128 Len=0
11	0.111105122	192.168.64.3	45.79.89.123	HTTP	408	GET /basicauth/ HTTP/1.1
12	0.163776635	45.79.89.123	192.168.64.3	TCP	60	80 → 55146 [ACK] Seq=401 Ack=708 Win=64128 Len=0
13	0.165392514	45.79.89.123	192.168.64.3	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)



Once I input the top secret username and password, the browser sends another GET request for /basicauth/ in frame 20. The authorization header is then updated in the packet sent by the browser to the server. This can be seen in the highlighted blue below, where wireshark tells me that my authorization is **Basic Y3MzMzg6cGFzc3dvcmQ=**, whereas before the authorization header would have been empty in the packet. Since this string of letters and numbers is not the top secret username and password I entered, this means that the browser has encrypted them.



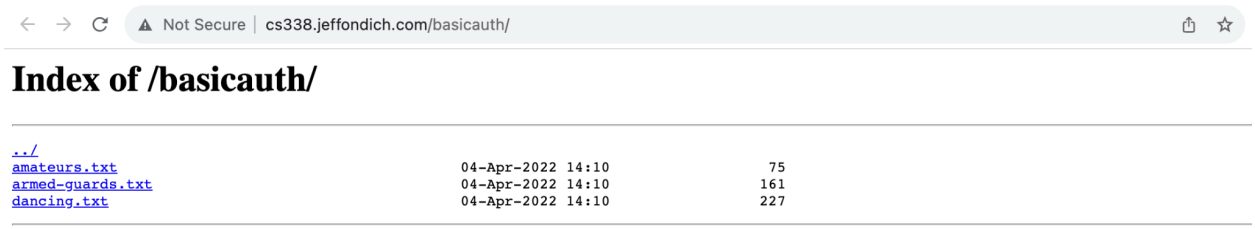
As found in the RFC 7617, this user-pass in the authorization header is constructed by concatenating the username, a ":" character, and the password (so in this case it is cs338:password), and then encoding that into an **octet sequence**, which is then encoded into **Base 64**. The RFC document told me that the octet sequence would either be a locale-specific encoding or UTF-8, and I figured that in this system it would be UTF-8. I confirmed this by checking to make sure the encoding I did resulted in the same string in the authorization header, which it was! See below for how the encoding works to take the user-pass from plain text into what is found in the authorization header.



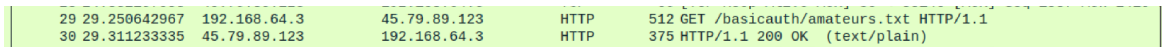
After decoding my user-pass found in the authorization header of the packet and authenticating that I am someone with access, the server responds with a **200 OK error code** in line 22, which means that I have sufficient authorization to view this page.

20	13.571690397	192.168.64.3	45.79.89.123	HTTP	451	GET /basicauth/ HTTP/1.1
21	13.625616177	45.79.89.123	192.168.64.3	TCP	60	80 → 55146 [ACK] Seq=804 Ack=1105 Win=64128 Len=0
22	13.626882806	45.79.89.123	192.168.64.3	HTTP	458	HTTP/1.1 200 OK (text/html)

Since that has been confirmed, I am now able to see in my browser the /basicauth/page, as seen below.



With this completed, I decided to do some exploring of the text files to see what they held and what the requests and responses would look like. First, I clicked on the amateurs.txt file, which sent another GET request to the server, asking for /basicauth/amateurs.txt, in frame 29. The server decided that it would be alright for me to see the file, so it sent back another **200 OK error code** with the plaintext file. This allowed me to see the text below.



29	29.250642967	192.168.64.3	45.79.89.123	HTTP	512	GET /basicauth/amateurs.txt HTTP/1.1
30	29.311233335	45.79.89.123	192.168.64.3	HTTP	375	HTTP/1.1 200 OK (text/plain)

"Amateurs hack systems, professionals hack people."

-- Bruce Schneier

I thought this was pretty cool, so I decided to examine armed-guards.txt and dancing.txt, as seen in frames 34-35 and 37-38 respectively. Again, the server had already authorized me to see these files so I was easily able to receive 200 OK error codes and view the text files.

34	49.238764556	192.168.64.3	45.79.89.123	HTTP	516	GET /basicauth/armed-guards.txt HTTP/1.1
35	49.296934072	45.79.89.123	192.168.64.3	HTTP	462	HTTP/1.1 200 OK (text/plain)
36	49.296977450	192.168.64.3	45.79.89.123	TCP	54	55146 → 80 [ACK] Seq=2339 Ack=2266 Win=64128 Len=0
37	55.485990203	192.168.64.3	45.79.89.123	HTTP	511	GET /basicauth/dancing.txt HTTP/1.1
38	55.541196889	45.79.89.123	192.168.64.3	HTTP	528	HTTP/1.1 200 OK (text/plain)

"The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards."

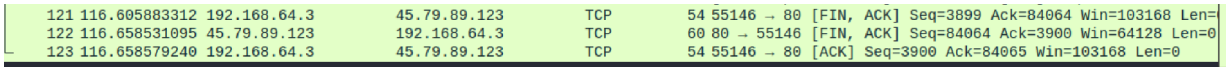
-- Gene Spafford

```
"Given a choice between dancing pigs and security, users will pick dancing pigs every time."

    -- Edward Felten and Gary McGraw, Securing Java (John Wiley & Sons, 1999)

[See also https://en.wikipedia.org/wiki/Dancing_pigs]
```

After a little more messing around, I decided to exit the page, which prompted my browser to send a **FIN,ACK** to the server, saying that they could terminate the connection. This is seen in frame 121. The server responded likewise in frame 122, saying that it had received the message from the browser and would terminate the connection. Finally, the browser told the server that it had received its communication and my time in the secrets folder was over.



121	116.605883312	192.168.64.3	45.79.89.123	TCP	54	55146 → 80 [FIN, ACK] Seq=3899 Ack=84064 Win=103168 Len=
122	116.658531095	45.79.89.123	192.168.64.3	TCP	60	80 → 55146 [FIN, ACK] Seq=84064 Ack=3900 Win=64128 Len=0
123	116.658579240	192.168.64.3	45.79.89.123	TCP	54	55146 → 80 [ACK] Seq=3900 Ack=84065 Win=103168 Len=0