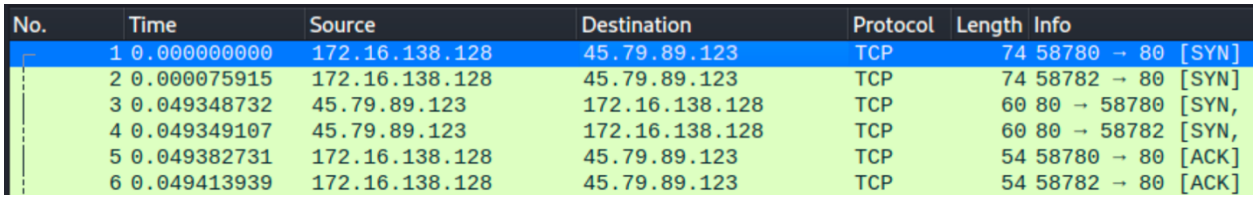
Vicente Riquelme

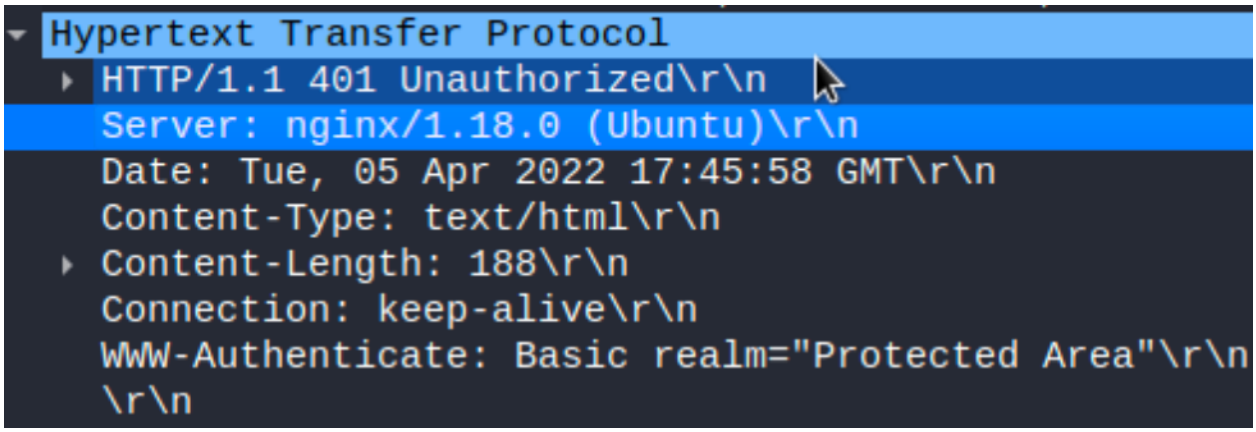
9 - Description of the sequence of events
3 - Explanation of the "Authorization" header
3 - Clarity of exposition

To begin, when the client (my computer, through firefox) attempts to access http://cs338.jeffondich.com/basicauth/. Once I click the client sends two SYN packets, attempting to start two connections over two ports. Once both connections are established through two SYN, SYN-ACK, ACK three-way handshakes, the client sent a GET /basicauth/ HTTP 1.1 request, requesting the contents of the page /basicauth/.

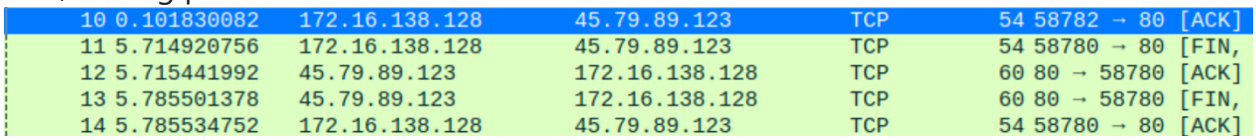


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.138.128	45.79.89.123	TCP	74	58780 → 80 [SYN]
2	0.000075915	172.16.138.128	45.79.89.123	TCP	74	58782 → 80 [SYN]
3	0.049348732	45.79.89.123	172.16.138.128	TCP	60	80 → 58780 [SYN,
4	0.049349107	45.79.89.123	172.16.138.128	TCP	60	80 → 58782 [SYN,
5	0.049382731	172.16.138.128	45.79.89.123	TCP	54	58780 → 80 [ACK]
6	0.049413939	172.16.138.128	45.79.89.123	TCP	54	58782 → 80 [ACK]

The server acknowledged this request through an ACK packet. The server then sends an HTTP 401 unauthorized packet. This packet told the client that it did not have permission to view the content. However, it also had, in the HTTP header, a WWW-Authenticate field, which contains the data for how to prompt the user for a username and password.

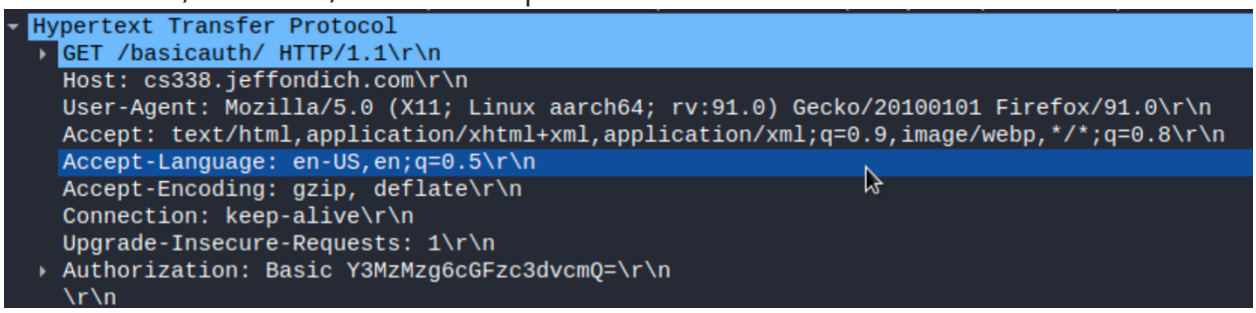


My client then acknowledged this with an ACK and sent a FIN ACK packet, to close that connection. The server then sends a FIN and ACK, to which my client responds with an ACK, closing port 58780.

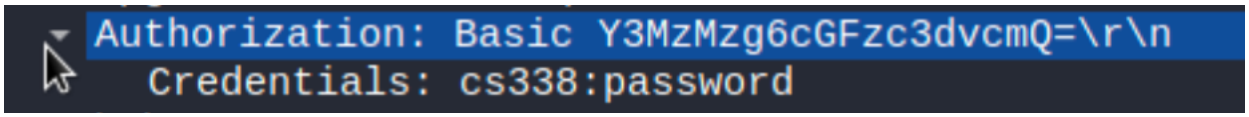


10	0.101830082	172.16.138.128	45.79.89.123	TCP	54	58782 → 80 [ACK]
11	5.714920756	172.16.138.128	45.79.89.123	TCP	54	58780 → 80 [FIN,
12	5.715441992	45.79.89.123	172.16.138.128	TCP	60	80 → 58780 [ACK]
13	5.785501378	45.79.89.123	172.16.138.128	TCP	60	80 → 58780 [FIN,
14	5.785534752	172.16.138.128	45.79.89.123	TCP	54	58780 → 80 [ACK]

It was at this point that I input the correct username and password in the browser. The browser took the username and password, joined them together with a ":" in the middle, and encoded it in base 64. This was then included in the HTTP Authorization header of another GET /basicauth/ HTTP 1.1 request.



This Authorization header is how the client can explain to the server that it is allowed to view the contents of that page by providing the input credentials and sending it along with the HTTP request. We are also able to see that this is why basic authentication is insecure. We are sending our credentials through the authentication header in very weak encoding, and thus anyone that intercepts this package would easily be able to know our credentials. This makes both this server insecure and potentially our own machine if we use the same password elsewhere. In fact, it seems that Wireshark does this conversion automatically for us.



The packet containing the authentication header received by the server and acknowledged. The server deencoded and verified that this username and password combination was correct and then sent an HTTP 1.1 200 OK back, signifying that the request could be served. It also contains the html text data in the payload which the browser displays. This are the contents of the page and we see that there are 4 clickable links.

```
    17 6.039507710   45.79.89.123          172.16.138.128        HTTP       458 HTTP/1.1 200 OK  (text/html
Ethernet II, Src: VMware_e2:93:6d (00:50:56:e2:93:6d), Dst: VMware_a5:aa:31 (00:0c:29:a5:aa:31)
Internet Protocol Version 4, Src: 45.79.89.123, Dst: 172.16.138.128
Transmission Control Protocol, Src Port: 80, Dst Port: 58782, Seq: 404, Ack: 728, Len: 404
Hypertext Transfer Protocol
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>\r\n
  <a href="amateurs.txt">amateurs.txt</a>                                    04-Apr-2022 14:10
  <a href="armed-guards.txt">armed-guards.txt</a>                                04-Apr-2022 14:10
  <a href="dancing.txt">dancing.txt</a>                                    04-Apr-2022 14:10
  </pre><hr></body>\r\n
  </html>\r\n
```

The client then sends a HTTP GET request for http://cs338.jeffondich.com/favicon.ico . We see that for this request, we were referred from /basicauth/.

```
Hypertext Transfer Protocol
  GET /favicon.ico HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
  Accept: image/webp,*/*\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Referer: http://cs338.jeffondich.com/basicauth/\r\n
  \r\n
  [Full request URI: http://cs338.jeffondich.com/favicon.ico]
  [HTTP request 3/3]
  [Prev request in frame: 15]
  [Response in frame: 23]
```

The next packet is GET /favicon HTTP 1.1, requesting that asset. Then, the client sends a FIN packet, which is ACK'ed and another FIN is sent from the server, which the client then ACKs. Thus the connection at port 58782 is closed.

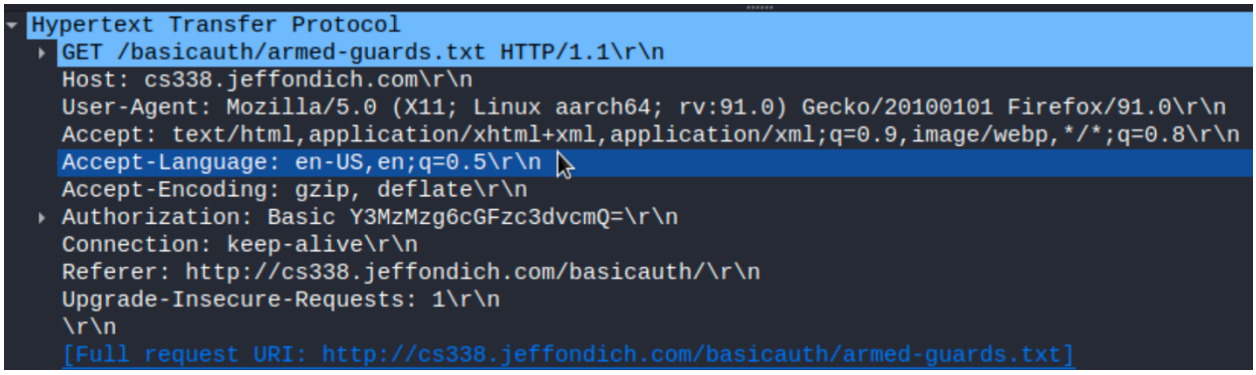
The server then sends a HTTP 404 not found, as http://cs338.jeffondich.com/favicon.ico is not a valid resource on the host site. We see that even in attempting to access it directly, there is nothing by that name on the website.

```
Hypertext Transfer Protocol
  HTTP/1.1 404 Not Found\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 404 Not Found\r\n]
```

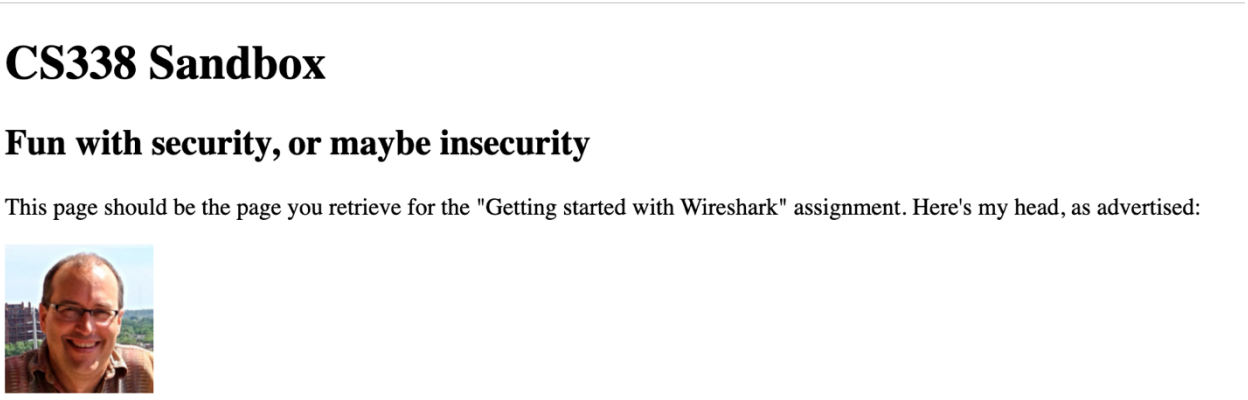
The client sends RST packet to restart the connection. There is then a new 3-way handshake which opens a new connection at port 58784.

```
    24 6.150505435   172.16.138.128        45.79.89.123          TCP         54 58782 → 80 [RST] Seq=1
    25 9.198904008   172.16.138.128        45.79.89.123          TCP         74 58784 → 80 [SYN] Seq=0
    26 9.256276843   45.79.89.123          172.16.138.128        TCP         60 80 → 58784 [SYN, ACK]
    27 9.256335258   172.16.138.128        45.79.89.123          TCP         54 58784 → 80 [ACK] Seq=1
```

Upon clicking into amateurs.txt, the client sends a GET request that contains the encoded credentials, thus the immediate response is a HTTP 200 OK and the data is sent back and displayed. We are able to click on each link, which prompts a GET request for that page. We are not denied access since the authorization header contains our encoded username and password in each request.



Clicking on the .. link sends us to http://cs338.jeffondich.com/, which does not require credentials, as there is no authorization header. Upon further testing, I saw that it was possible to access this page directly. The image below is me access the page linked without having to input credentials.



We see that this page has /basicauth/ as the referrer, but no credentials were asked. Thus, it is crucial to protect every single page on a site that is meant to be protected, as otherwise it may be possible to sidestep the authentication step.