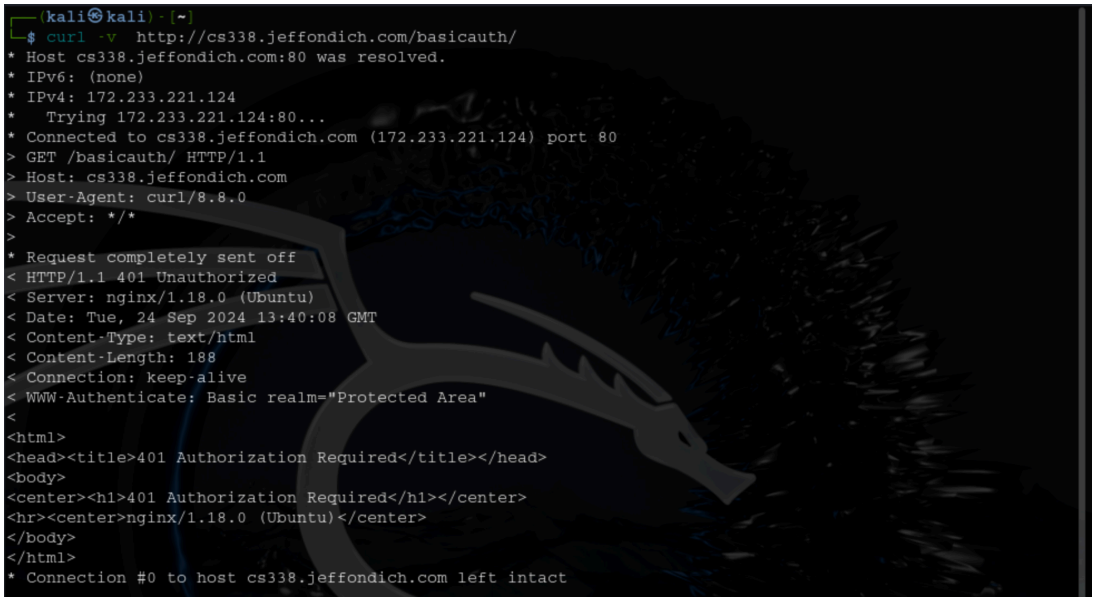
HTTP Basic Authentication Investigation

Investigating with Curl

Observations



Because curl simply does a GET request for the page at /basicauth, it is met with a "`401 Unauthorized`" response. I wonder what kind of request has to be made to enable authentication to happen. Could one authenticate through netcat for instance? Or is it only possible through a browser window?

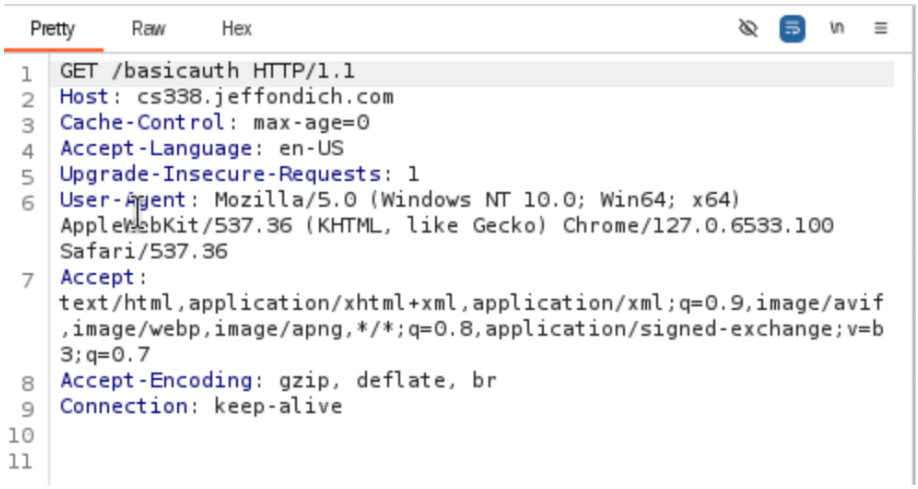
There were two things that stood out to me:

- `* Request completely sent off`
    - I was unsure what that meant but trying to google it wasn't much help. From what I can gather it doesn't seem to be anything of note
- `WWW-Authenticate: Basic realm="Protected Area"`
    - I wanted to take note of this header before I accessed the webpage as I have a feeling that this might be the name on the dialog window that pops up asking for authentication. Or it could just be a setting that means authentication is required. (Update: it's not)

Investigating with Wireshark and Burpsuite

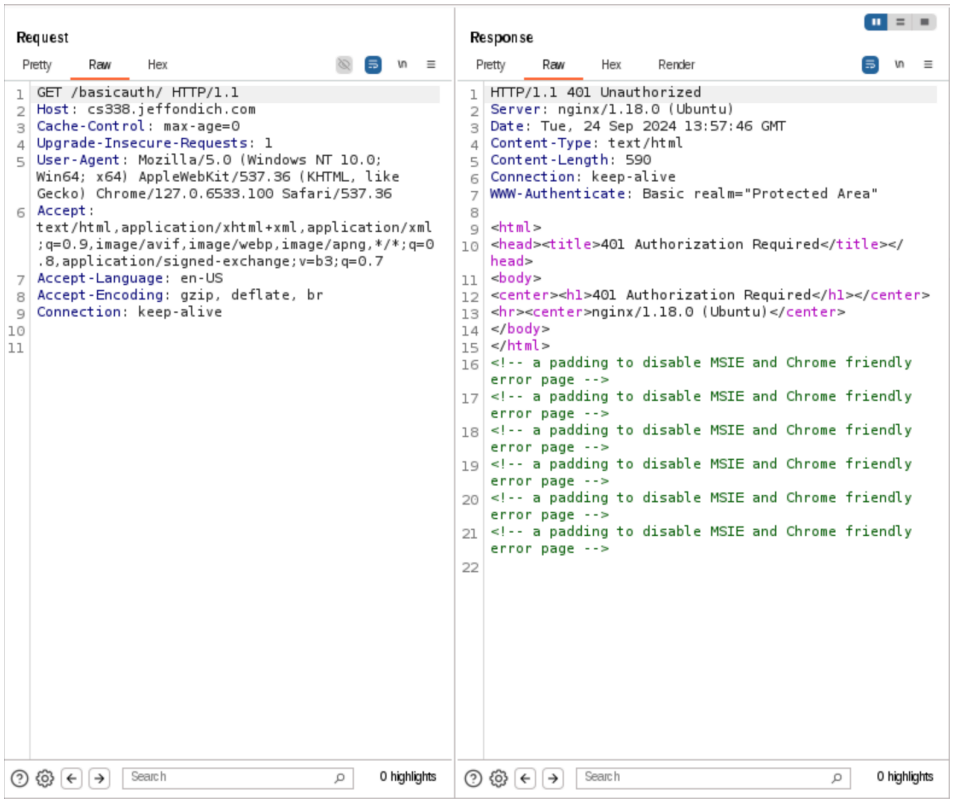
Main Observations

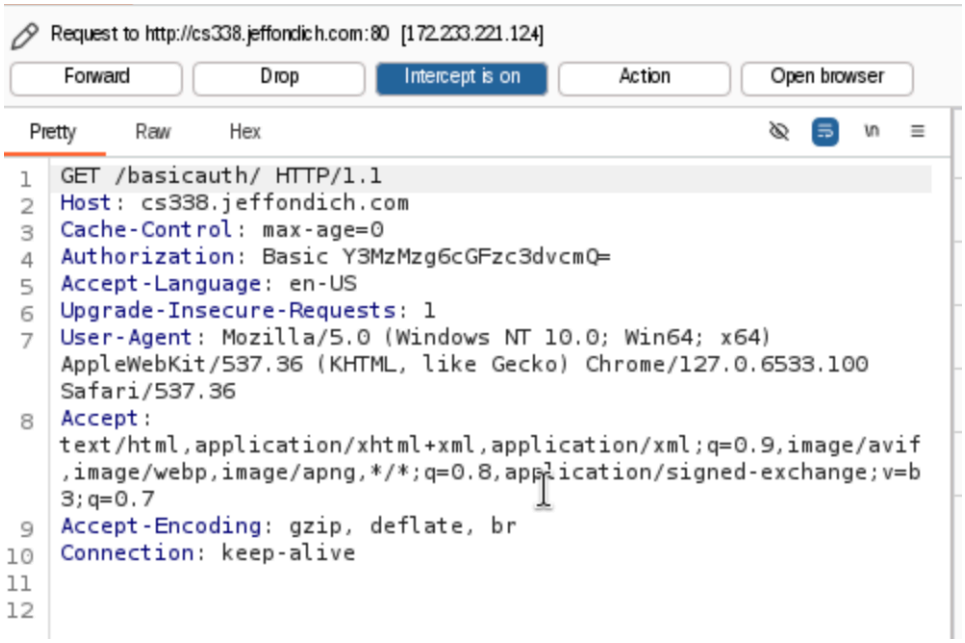
I decided to combine Wireshark with Burpsuite at first to see what that would reveal. The first thing I noticed is that there seemed to be an issue switching from https to http. Perhaps I clicked the buttons in the wrong order, but it took several forwards and Continue to sites before I was finally able to access the web page. However, because humans are slow and because I had Wireshark loaded as well, I can kind of tell which of the requests was actually sent using http.



I noticed that several GET requests were made for the web page. I wonder perhaps if this initial GET request was met with a 401 Error which instructed the browser to send a new GET request with the appropriate Authorization header.

Update: It was! The initial GET request was met with a 401 Unauthorized response. This prompted the browser to send the request again but with the Authorization header included.

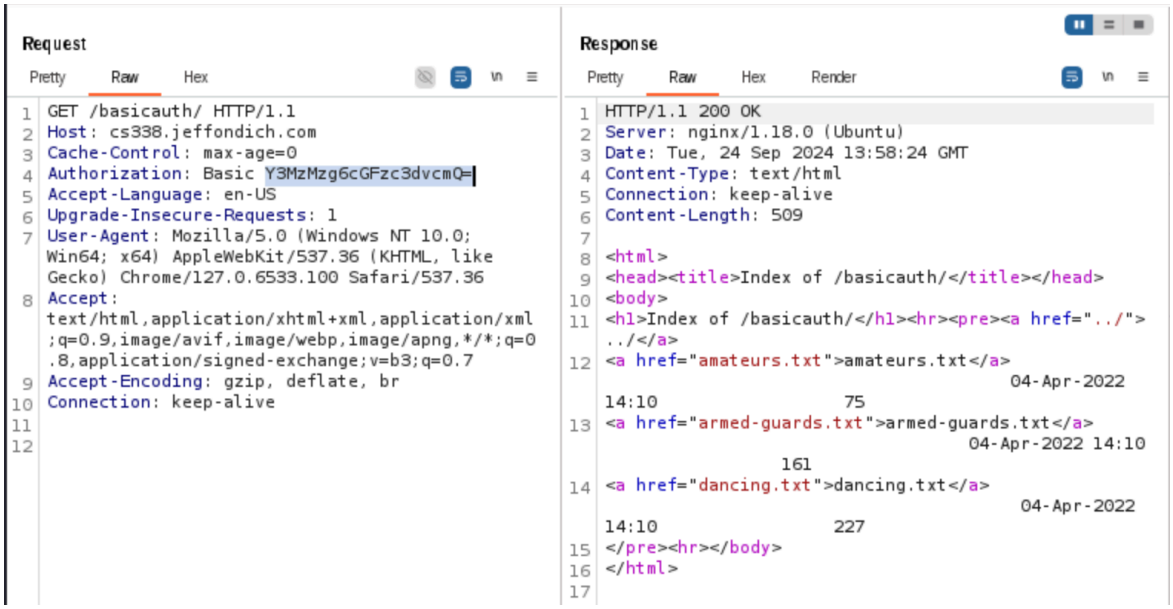




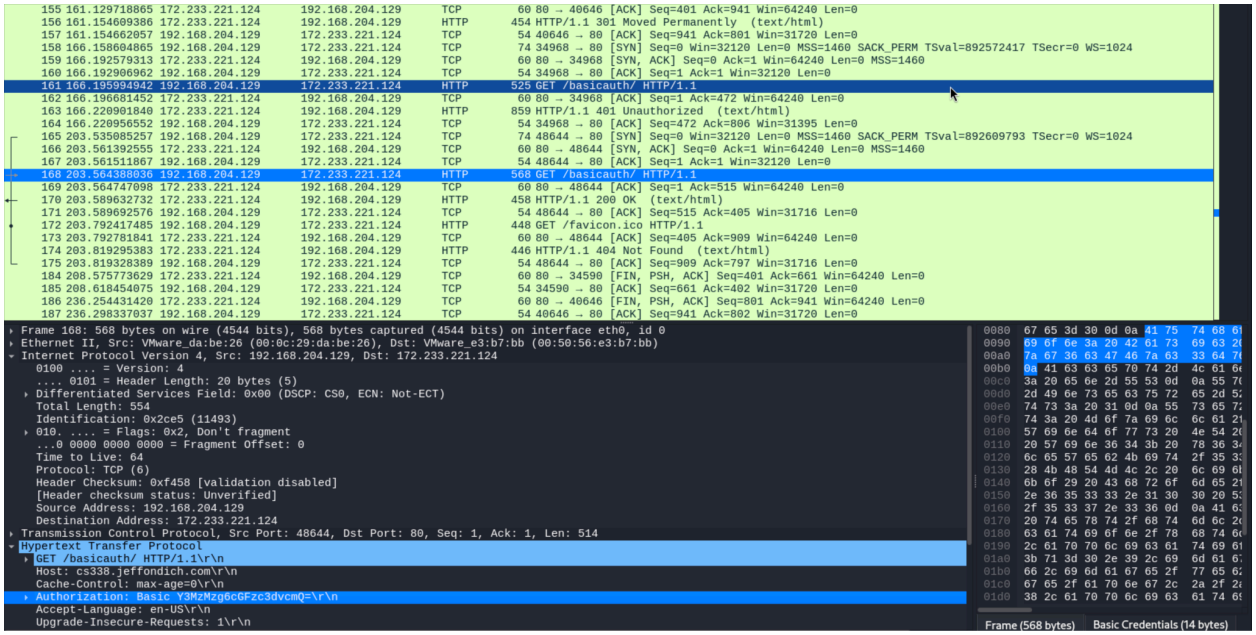
This is the second GET request made by the browser. Most of it is the same except for the added "`Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=`"
I'm assuming this prompted the server to ask for authentication. However I'm not sure what "Basic Y3MzMzg6cGFzc3dvcmQ=" means in this context.

Another update: Knowing how the basic HTTP authentication protocol works, I decided to decode “Y3MzMzg6cGFzc3dvcmQ=” through base64. It is in fact the username and password!

This reveals that the server does the authenticating because it sends a 200 OK response after that.



I was able to see with Wireshark that after the 401 Unauthorized Error, the browser initiated another TCP handshake. It then made a GET request with the username and password I entered in the header.



What I'm unsure about is who initiated the pop up. At this point it seems like it was the browser. But it seems like it should be a feature of the website so I'm not sure.

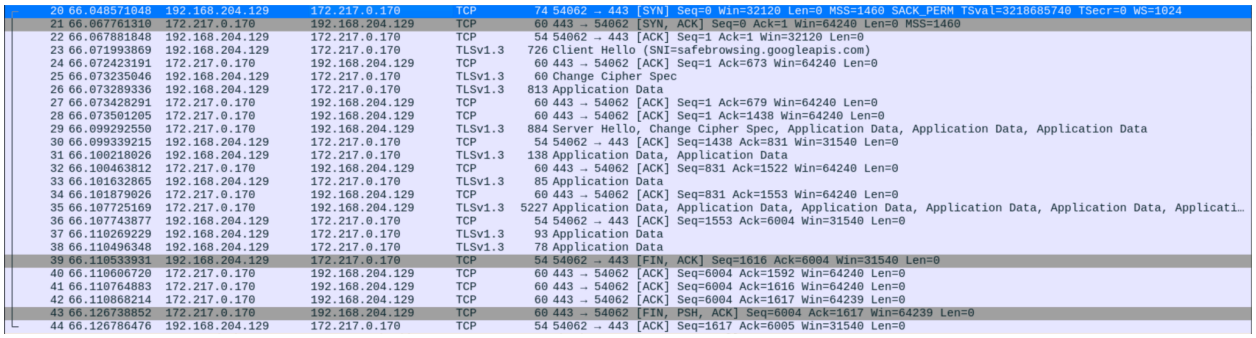
What is clear is that the server does the authenticating. I'm just not sure if the pop-up is a feature of the web page or the browser.

Additional Observations

I decided to look at all the traffic rather than just traffic to cs338.jeffondich.com, I learned that even just between your browser and your machine, there is a lot of communication. There is a TCP connection set up and they agree upon a cipher for TLS.

I opened up an incognito window and I found a lot of requests between 172.217.0.170, which after running that through ARIN, I learned was google.
Usually whenever I'm looking at Wireshark, I filter out my results but it was interesting to see what other communication is happening between the browser and Kali. Especially when you are presented with one of those "This website is insecure pages"

I also saw mentions of *safebrowsing.googleapis.com* which drew my attention. I was curious about whether that applied to just the incognito windows or if it was always present.



20	66.048571048	192.168.204.129	172.217.0.170	TCP	74	54062 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3218685740 TSecr=0 WS=1024
21	66.067761310	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
22	66.067881848	192.168.204.129	172.217.0.170	TCP	54	54062 → 443 [ACK] Seq=1 Ack=1 Win=32120 Len=0
23	66.071993869	192.168.204.129	172.217.0.170	TLSv1.3	726	Client Hello (SNI=safebrowsing.googleapis.com)
24	66.072423191	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=1 Ack=673 Win=64240 Len=0
25	66.073235046	192.168.204.129	172.217.0.170	TLSv1.3	60	Change Cipher Spec
26	66.073289336	192.168.204.129	172.217.0.170	TLSv1.3	813	Application Data
27	66.073428291	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=1 Ack=679 Win=64240 Len=0
28	66.073501205	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=1 Ack=1438 Win=64240 Len=0
29	66.099292550	172.217.0.170	192.168.204.129	TLSv1.3	884	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data
30	66.099339215	192.168.204.129	172.217.0.170	TCP	54	54062 → 443 [ACK] Seq=1438 Ack=831 Win=31540 Len=0
31	66.100218026	192.168.204.129	172.217.0.170	TLSv1.3	138	Application Data, Application Data
32	66.100463812	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=831 Ack=1522 Win=64240 Len=0
33	66.101632865	192.168.204.129	172.217.0.170	TLSv1.3	85	Application Data
34	66.101879026	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=831 Ack=1553 Win=64240 Len=0
35	66.107725169	172.217.0.170	192.168.204.129	TLSv1.3	5227	Application Data, Application Data, Application Data, Application Data, Application Data, Applicati…
36	66.107743877	192.168.204.129	172.217.0.170	TCP	54	54062 → 443 [ACK] Seq=1553 Ack=6004 Win=31540 Len=0
37	66.110269229	192.168.204.129	172.217.0.170	TLSv1.3	93	Application Data
38	66.110496348	192.168.204.129	172.217.0.170	TLSv1.3	78	Application Data
39	66.110533931	192.168.204.129	172.217.0.170	TCP	54	54062 → 443 [FIN, ACK] Seq=1616 Ack=6004 Win=31540 Len=0
40	66.110606720	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=6004 Ack=1592 Win=64240 Len=0
41	66.110764883	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=6004 Ack=1616 Win=64240 Len=0
42	66.110868214	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [ACK] Seq=6004 Ack=1617 Win=64239 Len=0
43	66.126738852	172.217.0.170	192.168.204.129	TCP	60	443 → 54062 [FIN, PSH, ACK] Seq=6004 Ack=1617 Win=64239 Len=0
44	66.126786476	192.168.204.129	172.217.0.170	TCP	54	54062 → 443 [ACK] Seq=1617 Ack=6005 Win=31540 Len=0

Useful acronyms:

- SNI: Server Name Identification
    - Client indicates which host it is trying to connect with.

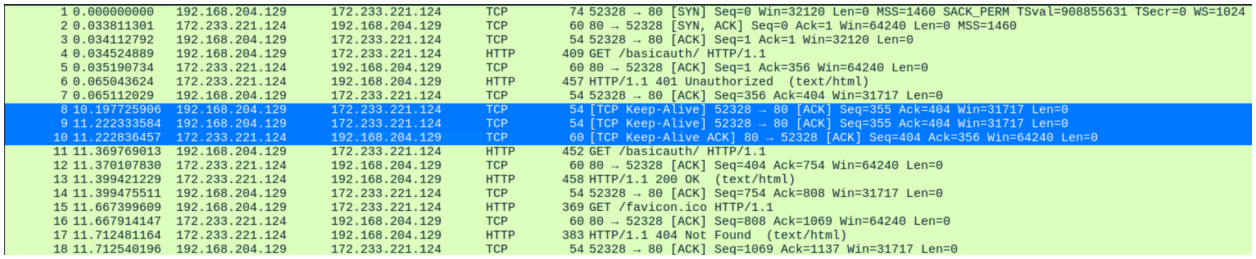
Investigating with Wireshark

Observations

Since my first experiments with both Wireshark and Burpsuite included a little noise and a few unanswered questions, I decided to look at each one on its own.

One thing I noticed this time that I didn't before was the presence of TCP Keep Alive packets.

I'm assuming the browser was prompting the server to not close the connection as I typed the username and password.



1	0.000000000	192.168.204.129	172.233.221.124	TCP	74 52328 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=908855631 TSecr=0 WS=1024
2	0.033811301	172.233.221.124	192.168.204.129	TCP	60 80 → 52328 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.034112792	192.168.204.129	172.233.221.124	TCP	54 52328 → 80 [ACK] Seq=1 Ack=1 Win=32120 Len=0
4	0.034524889	192.168.204.129	172.233.221.124	HTTP	409 GET /basicauth/ HTTP/1.1
5	0.035190734	172.233.221.124	192.168.204.129	TCP	60 80 → 52328 [ACK] Seq=1 Ack=356 Win=64240 Len=0
6	0.065043624	172.233.221.124	192.168.204.129	HTTP	457 HTTP/1.1 401 Unauthorized (text/html)
7	0.065112029	192.168.204.129	172.233.221.124	TCP	54 52328 → 80 [ACK] Seq=356 Ack=404 Win=31717 Len=0
8	10.197725906	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 52328 → 80 [ACK] Seq=355 Ack=404 Win=31717 Len=0
9	11.222333584	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 52328 → 80 [ACK] Seq=355 Ack=404 Win=31717 Len=0
10	11.222836457	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 52328 [ACK] Seq=404 Ack=356 Win=64240 Len=0
11	11.369769013	192.168.204.129	172.233.221.124	HTTP	452 GET /basicauth/ HTTP/1.1
12	11.370107830	172.233.221.124	192.168.204.129	TCP	60 80 → 52328 [ACK] Seq=404 Ack=754 Win=64240 Len=0
13	11.399421229	172.233.221.124	192.168.204.129	HTTP	458 HTTP/1.1 200 OK (text/html)
14	11.399475511	192.168.204.129	172.233.221.124	TCP	54 52328 → 80 [ACK] Seq=754 Ack=808 Win=31717 Len=0
15	11.667399609	192.168.204.129	172.233.221.124	HTTP	369 GET /favicon.ico HTTP/1.1
16	11.667914147	172.233.221.124	192.168.204.129	TCP	60 80 → 52328 [ACK] Seq=808 Ack=1069 Win=64240 Len=0
17	11.712481164	172.233.221.124	192.168.204.129	HTTP	383 HTTP/1.1 404 Not Found (text/html)
18	11.712540196	192.168.204.129	172.233.221.124	TCP	54 52328 → 80 [ACK] Seq=1069 Ack=1137 Win=31717 Len=0

I decided to also try using the wrong password and it once again just showed a series of GET requests each with the wrong encoding. What's interesting is that after you decide to cancel the sign in, your browser still has to send another GET request before returning with the 401 error. It has me wondering what other response is possible.

13	17.691102906	172.233.221.124	192.168.204.129	TCP	60 80 → 33886 [ACK] Seq=807 Ack=1180 Win=64240 Len=0
14	17.727373545	172.233.221.124	192.168.204.129	HTTP	457 HTTP/1.1 401 Unauthorized (text/html)
15	17.727410539	192.168.204.129	172.233.221.124	TCP	54 33886 → 80 [ACK] Seq=1180 Ack=1210 Win=31717 Len=0
16	27.772014777	192.168.204.129	172.233.221.124	HTTP	484 GET /basicauth/ HTTP/1.1
17	27.772404878	172.233.221.124	192.168.204.129	TCP	60 80 → 33886 [ACK] Seq=1210 Ack=1610 Win=64240 Len=0
18	27.801792286	172.233.221.124	192.168.204.129	HTTP	457 HTTP/1.1 401 Unauthorized (text/html)
19	27.801848402	192.168.204.129	172.233.221.124	TCP	54 33886 → 80 [ACK] Seq=1610 Ack=1613 Win=31717 Len=0
20	31.229317576	192.168.204.129	172.233.221.124	HTTP	369 GET /favicon.ico HTTP/1.1
21	31.229630899	172.233.221.124	192.168.204.129	TCP	60 80 → 33886 [ACK] Seq=1613 Ack=1925 Win=64240 Len=0
22	31.259939235	172.233.221.124	192.168.204.129	HTTP	383 HTTP/1.1 404 Not Found (text/html)
23	31.259971480	192.168.204.129	172.233.221.124	TCP	54 33886 → 80 [ACK] Seq=1925 Ack=1942 Win=31717 Len=0
24	41.384657616	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
25	42.409216335	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
26	42.409934423	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0
27	52.648549028	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
28	52.648982747	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0
29	62.889759072	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
30	62.890752614	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0
31	73.129654237	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
32	73.130346248	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0
33	83.368740828	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
34	83.369294986	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0
35	93.608702132	192.168.204.129	172.233.221.124	TCP	54 [TCP Keep-Alive] 33886 → 80 [ACK] Seq=1924 Ack=1942 Win=31717 Len=0
36	93.609408514	172.233.221.124	192.168.204.129	TCP	60 [TCP Keep-Alive ACK] 80 → 33886 [ACK] Seq=1942 Ack=1925 Win=64240 Len=0

Investigating with Burpsuite

Observations

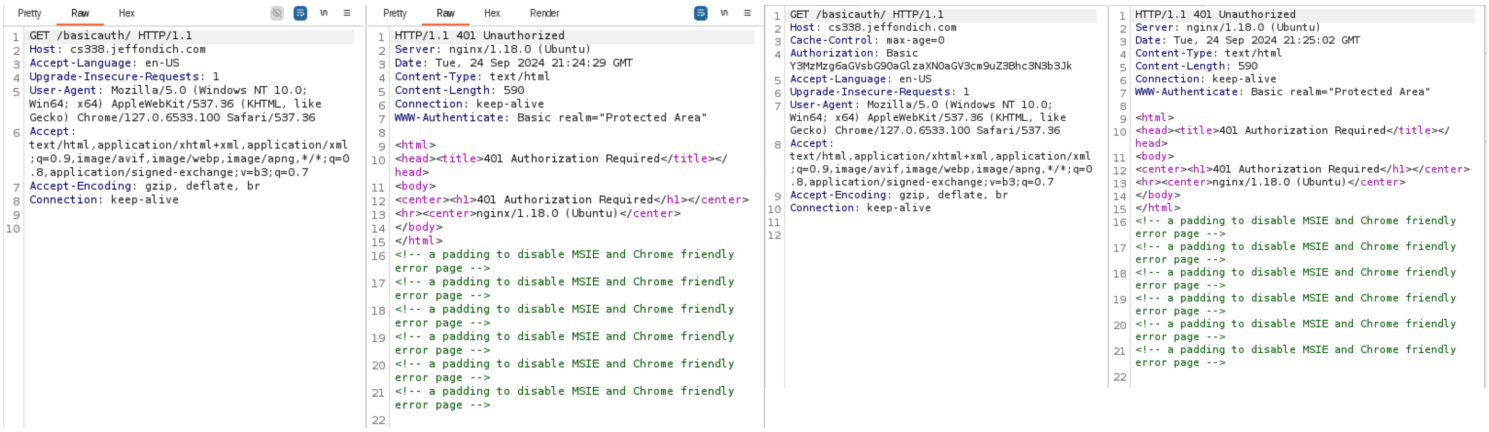
Initial observations running everything again. This time there were 2 GET requests. The first is just a normal GET request and the second is a GET request with the username and password “encoded” in base64.

Which leads me to believe that the browser prompts for the username and password, then uses that information to generate a second GET request to authenticate with the server.

Additional Observations

Motivating question #1: **what would happen if I entered the password and username wrong?**

The browser continues to prompt for your password indefinitely. Or at least until something on either end crashes. I was curious because if the sign in prompt is generated by the browser, how would it know when to stop. Especially since the response from the server is the same, the browser would understandably have the same response: to prompt for a sign-in.



It seems that a stopping procedure would have to be implemented on the server side. For instance only allowing 2 sign-in attempts before sending a 403 error or something. Unless of course you press cancel and the browser recognizes you no longer wish to attempt to sign in.

Motivating question #1: what would happen if I edited the initial GET request to include the Authorization header?

Knowing that the initial GET request and 401 Unauthorized response simply prompts the browser to ask for the username and password to reformat its GET request, I want to see what happens when you just add it in.

I'm assuming it'll bypass the ask for the username and password and just send you straight through. Which is likely what happens when the browser caches that information in the first place.

That is exactly what happened.