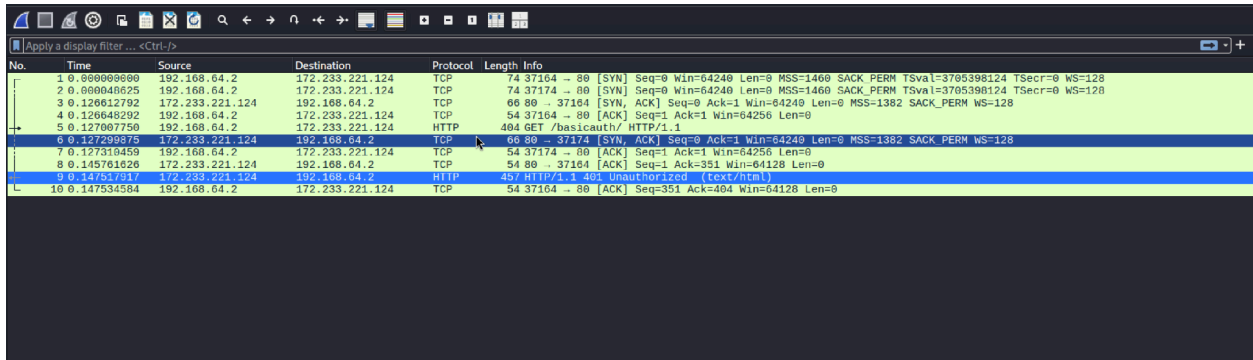


The story begins when the browser first begins talking to jeffondich.com, before any authentication credentials are provided. Here, you can see a (quite small) picture of what this interaction looks like.



A screenshot of the Wireshark network protocol analyzer. The packet list pane shows several packets. Packet 6 is selected, which is a GET request for /basicauth/ from 192.168.64.2 to 172.233.221.124. Packet 7 is the corresponding SYN-ACK response. Packet 8 is an ACK response. Packet 9 is the HTTP 401 Unauthorized response, which is highlighted in blue. The packet details pane for packet 9 shows the status code 401 and the 'WWW-Authenticate: Basic realm="Protected Area"' header.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.64.2	172.233.221.124	TCP	74	37164 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3705398124 TSecr=0 WS=128
2	0.000046	192.168.64.2	172.233.221.124	TCP	74	37174 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3705398124 TSecr=0 WS=128
3	0.126612	172.233.221.124	192.168.64.2	TCP	60	80 → 37164 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_PERM WS=128
4	0.126648	192.168.64.2	172.233.221.124	TCP	54	37164 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
5	0.127067	192.168.64.2	172.233.221.124	HTTP	484	GET /basicauth/ HTTP/1.1
6	0.127299	172.233.221.124	192.168.64.2	TCP	66	80 → 37174 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_PERM WS=128
7	0.127319	192.168.64.2	172.233.221.124	TCP	54	37174 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
8	0.145761	172.233.221.124	192.168.64.2	TCP	54	80 → 37164 [ACK] Seq=1 Ack=351 Win=64128 Len=0
9	0.147537	172.233.221.124	192.168.64.2	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
10	0.147534	192.168.64.2	172.233.221.124	TCP	54	37164 → 80 [ACK] Seq=351 Ack=404 Win=64128 Len=0

Picture of initial HTTP request where the server responds with code 401

In this photo, we can see the browser opens two TCP connections with two corresponding handshakes and then queries the /basicauth/ file on server. The server responds with a 401 code. This tells the browser that it should look for a WWW-Authenticate header¹ in the response. This header will tell the browser necessary details like what system to use for password verifying. On top of having the error 401, no actual website data is sent. This is how the client is prevented from seeing the data before any password is entered. Looking for that header in Wireshark, we see:



A screenshot of the Wireshark packet details pane for the selected packet 9 (HTTP 401 Unauthorized). The pane shows the structure of the HTTP response, including the status line, headers, and body. The 'WWW-Authenticate: Basic realm="Protected Area"' header is highlighted in blue. Below the headers, there are links to view the request in frame 5, the time since request, the request URI, the full request URI, and the file data.

```
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 401 Unauthorized\r\n
    Server: nginx/1.18.0 (Ubuntu)\r\n
    Date: Mon, 22 Sep 2025 20:27:11 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
    [Request in frame: 5]
    [Time since request: 0.020510167 seconds]
    [Request URI: /basicauth/]
    [Full request URI: http://cs338.jeffondich.com/basicauth/]
    File Data: 188 bytes
```

Contents of error code 401 that reveals how the browser communicates the authentication scheme

¹ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/401>

Essentially, this tells the browser that the authentication scheme is Basic², meaning that the browser should get a simple username and password to enter the “realm,” which is named “Protected Area” in this example (though it can be named other things). I then cleared Wireshark’s output and entered the username and password. When I clicked “sign in” Wireshark got the following data.

1	0.000000000	192.168.64.2	172.233.221.124	TCP	74	57710 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3706092779 Tsecr=0 WS=128
2	0.022045932	172.233.221.124	192.168.64.2	TCP	66	80 → 57710 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_PERM WS=128
3	0.022079557	192.168.64.2	172.233.221.124	TCP	54	57710 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
4	0.022209682	192.168.64.2	172.233.221.124	HTTP	447	GET /basicauth/ HTTP/1.1
5	0.041279826	172.233.221.124	192.168.64.2	TCP	54	80 → 57710 [ACK] Seq=1 Ack=394 Win=64128 Len=0
6	0.043099032	172.233.221.124	192.168.64.2	HTTP	458	HTTP/1.1 200 OK (text/html)
7	0.043106823	192.168.64.2	172.233.221.124	TCP	54	57710 → 80 [ACK] Seq=394 Ack=405 Win=64128 Len=0

Sequence of interactions between client and server when the correct username password pair is entered

Here we can observe that the browser opens a new TCP connection, as the first 3 tags are the handshake. Something that is interesting is that this new TCP connection is not consistently opened, and the browser sometimes uses the old connection instead. My educated guess is that this is related to how long the connection is idle, in which case the browser may close the connection, but I have no strong conclusions. I’m especially uncertain because the 401 response that told the browser it needed a password said that it should keep the connection alive. Nonetheless, the connection was closed. Finding the exact cause seems beyond the scope of this assignment. After making an HTTP request over the connection, however, the browser responds with a 200 OK response code, instead of the Unauthorized code. This is because, examining the Browser’s GET request closer, it included a new HTTP header that was not there originally: Authorization. See the below screenshot for an example of the Authorization header in context.

² https://en.wikipedia.org/wiki/Basic_access_authentication

```

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:128.0) Gecko/20100101 Firefox/128.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
  Priority: u=0, i\r\n
  Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
\r\n
[Response in frame: 6]
[Full request URI: http://cs338.jeffondich.com/basicauth/]

```

Contents of HTTP request sent in above photo. Notice the Authorization token that is not normally present in HTTP requests.

I recognized that this was likely a base64 encoded string (and also got a hint from the wikipedia article on Basic authentication), so I ran it through the base64 command to decode it.

```

$ echo "Y3MzMzg6cGFzc3dvcmQ=" | base64 -d
cs338:password

```

Authorization token contents decoded from base64 to regular text

This shows that the credentials are encoded, but not encrypted because we do not need to take any steps beyond decoding the authentication. I would assume that the server does a similar process to decode the username and password pair on the receiving end and when it sees a match, serves the content to the user. The parsing and subsequent checking that the password and username match seem to be handled entirely by the server.

We can tell that the difference between getting a 401 Unauthorized code and 200 OK code lies in the correct value for the Authorization header because when you first access the page, there is no Authorization token in the header. Similarly, when you enter the wrong combination, you still get a 401 Unauthorized error, even though the Authorization header is present. Though this is not shown, it looks nearly identical to the initial HTTP request without the Authorization token shown at all, even the WWW-Authenticate header stays the same.

Thus, for my explanation of the Authorization header, I would say that the Authorization header is a header that is only present when the browser knows that the server is expecting a

form of authorization. It begins with the Authorization scheme (in this case, Basic authorization) and follows with whatever code. For basic authorization, the code is <username>:<password> encoded into base64 text, but not encrypted. What I have observed regarding this header is consistent with what is seen in the Wikipedia article on Basic Authentication.