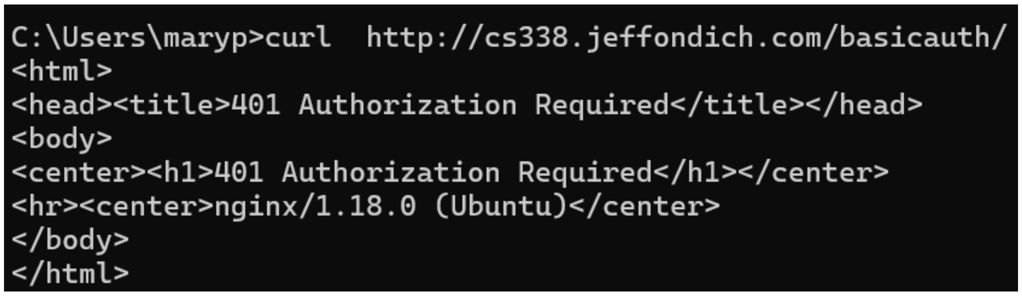
Investigating with curl

First we start by doing a curl command. Basically, curl fetches the HTML file of the specified webpage from the web server, and displays it in the terminal. Running curl with http://cs338.jeffondich.com/basicauth/ gives us an HTML file, but the content is concealed. Instead, we get a header with a 401 error code, followed by “Authorization Required”. We also get access to the server software: nginx/1.18.0 (Ubuntu).

Comment: Knowing the version of the server software can lead to security vulnerabilities. One could look up the vulnerabilities of a given software version(if any..) more easily, but if that information is unknown, it might be a little harder to exploit the server.



```
C:\Users\maryp>curl  http://cs338.jeffondich.com/basicauth/
<html>
<head><title>401 Authorization Required</title></head>
<body>
<center><h1>401 Authorization Required</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
```

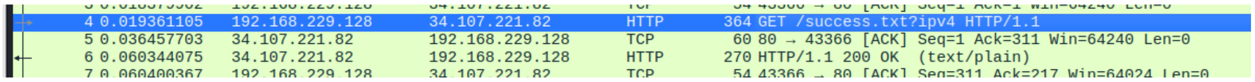
Investigating the network using Wireshark.

The first time I fired wireshark with the webpage, I got a series of [FIN,ACK] and [ACK] messages between the client server and the web server, without any TCP 3-way authentication happening. This might be due to the abrupt termination happening as a result of the network error I was encountering. I had to reset the settings of VM before continuing.

Once I reset the network and launched firefox, the first thing that happens is the TCP 3-way handshake connection being established between the client and server computers. After this connection is established, the client sends an HTTP request to the server for a document called “success.txt”. This happens before the username and password has been entered for authentication. The summary frame looks like:

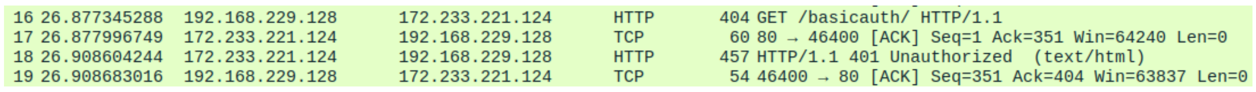
4	0.019361105	192.168.229.128	34.107.221.82	HTTP	364	GET /success.txt?ipv4 HTTP/1.1

The web server acknowledges the request and sends a plain text as response to the client computer.



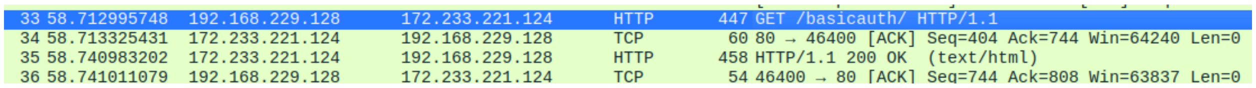
```
4 0.019361105   192.168.229.128   34.107.221.82     HTTP   364 GET /success.txt?ipv4 HTTP/1.1
5 0.036457703   34.107.221.82     192.168.229.128   TCP     60 80 → 43366 [ACK] Seq=1 Ack=311 Win=64240 Len=0
6 0.060344075   34.107.221.82     192.168.229.128   HTTP   270 HTTP/1.1 200 OK  (text/plain)
7 0.060400367   192.168.229.128   34.107.221.82     TCP     54 43366 → 80 [ACK] Seq=311 Ack=217 Win=64024 Len=0
```

We then get another set of TCP 3-way handshake between the client and the server, but happening on a different client port. (port changed from 43366 to 46400). Once this new TCP connection is established, the client sends a request for a basic authentication to the server. The web server acknowledges the request and gives an “unauthorized” reply, which the client acknowledges. At this point, we still have not entered the authentication credentials.



```
16 26.877345288  192.168.229.128     172.233.221.124     HTTP      404 GET /basicauth/ HTTP/1.1
17 26.877996749  172.233.221.124     192.168.229.128     TCP        60 80 → 46400 [ACK] Seq=1 Ack=351 Win=64240 Len=0
18 26.908604244  172.233.221.124     192.168.229.128     HTTP      457 HTTP/1.1 401 Unauthorized  (text/html)
19 26.908683016  192.168.229.128     172.233.221.124     TCP        54 46400 → 80 [ACK] Seq=351 Ack=404 Win=63837 Len=0
```

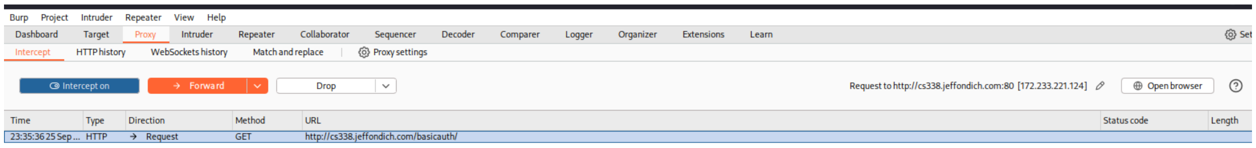
The client computer then requests another basic authentication, which the web server acknowledges, and returns a status code 200 OK indicating that the authentication was successful.



```
33 58.712995748  192.168.229.128     172.233.221.124     HTTP      447 GET /basicauth/ HTTP/1.1
34 58.713325431  172.233.221.124     192.168.229.128     TCP        60 80 → 46400 [ACK] Seq=404 Ack=744 Win=64240 Len=0
35 58.740983202  172.233.221.124     192.168.229.128     HTTP      458 HTTP/1.1 200 OK  (text/html)
36 58.741011079  192.168.229.128     172.233.221.124     TCP        54 46400 → 80 [ACK] Seq=744 Ack=808 Win=63837 Len=0
```

Once the authentication is done, the client sends a request to the web server to get the favorite icon, as it now has access to the web page.

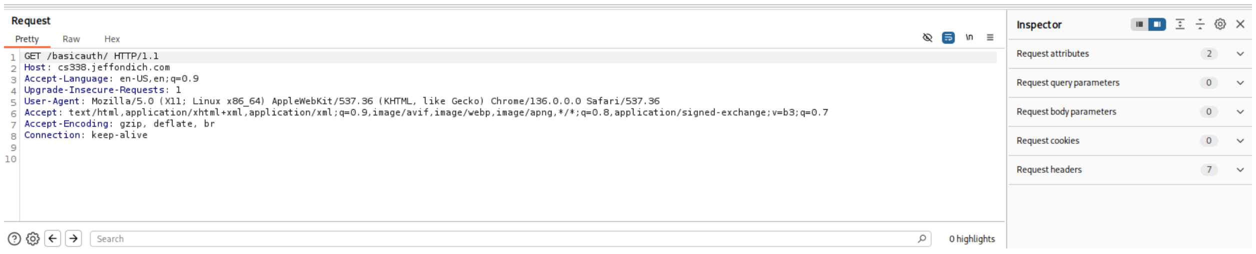
Investigating with Burpsuite



Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Open browser

Request to http://cs338.jeffondich.com:80 [172.233.221.124]

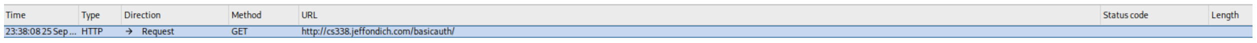
Time	Type	Direction	Method	URL	Status code	Length
23:35:36 25 Sep ...	HTTP	→ Request	GET	http://cs338.jeffondich.com/basicauth/		



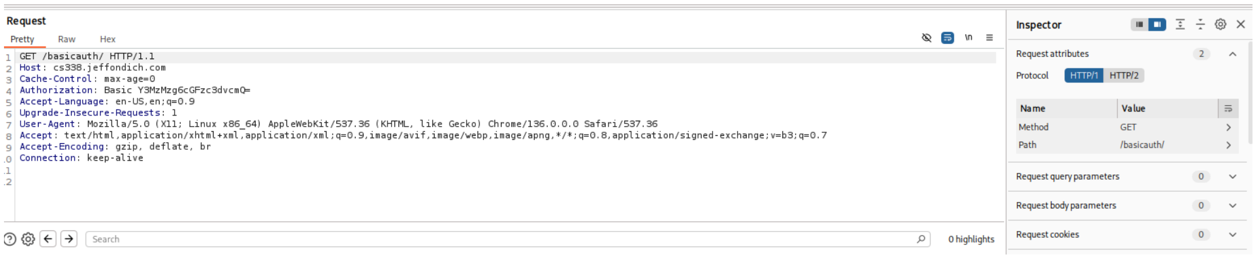
```
GET /basicauth/ HTTP/1.1
Host: cs338.jeffondich.com
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive


```

Request attributes	2
Request query parameters	0
Request body parameters	0
Request cookies	0
Request headers	7



Time	Type	Direction	Method	URL	Status code	Length
23:38:08 25 Sep ...	HTTP	→ Request	GET	http://cs338.jeffondich.com/basicauth/		



Request

Pretty Raw Hex

```
GET /basicauth/ HTTP/1.1
Host: cs338.jeffondich.com
Cache-Control: max-age=0
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Inspector

Request attributes 2

Protocol HTTP/1 HTTP/2

Name	Value
Method	GET
Path	/basicauth/

Request query parameters 0

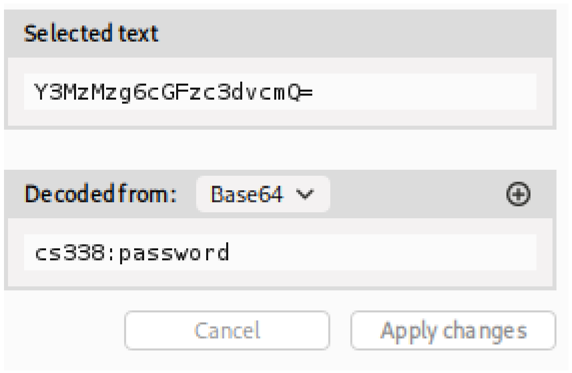
Request body parameters 0

Request cookies 0

The URL encoding sent by the browser looks like:

Authorization: Basic Y3MzMzg6cGFzc3dvcmQ

The browser does not do the basic authentication itself. It combines the username and the password, encodes it using base64 and sends it to the webserver to verify the authentication. Here is a capture of burpsuite inspection from the basicauth request sent by the browser.



Selected text

Y3MzMzg6cGFzc3dvcmQ=

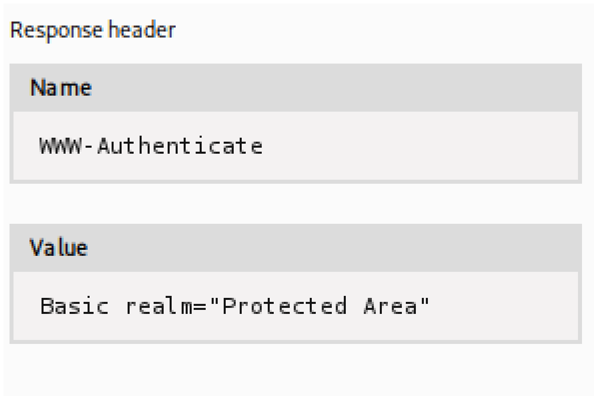
Decoded from: Base64

cs338:password

Burpsuite provides a decoder that can help decode the authentication credentials. The credentials are not encrypted using a key.

Relation to the HTTP Basic Authentication specification documents

- The format we observe the credentials transferred to the nginx server in Burpsuite matches what is specified in the HTTP Basic Authentication document. I.e user id and password encoded using base64. [ref : https://datatracker.ietf.org/doc/html/rfc7235]
- The request headers and response headers on burpsuite equally match the documentation. The response header sent by the server upon receiving an access request looks like:


Response header

Name

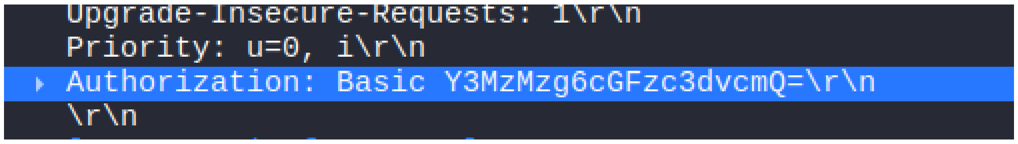
```
WWW-Authenticate
```

Value

```
Basic realm="Protected Area"
```

Where "Protected Area" is the string assigned by the server to represent a protected space [ref: https://datatracker.ietf.org/doc/html/rfc7617]

- We do get similar GET and HTTP requests on wireshark made by the browser as specified in the documentation (the screenshots are included above).
- From wireshark we can also see the authentication credentials transmitted by the web browser over the network:



```
Upgrade-Insecure-Requests: 1\r\n
Priority: u=0, i\r\n
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
\r\n
```

Summary:

- The browser request access to the webpage
- Since the webpage is protected, nginx response with 401 Unauthorized and WWW-Authenticate: Basic realm="Protected"
- The browser prompts the user for the username and password, encodes it with base64 with the format "username: password", and sends it to nginx for verification.
- Nginx decodes the credentials and checks if it is correct.
- Since the credentials are correct, nginx replies with a status code 200 OK, and grants access to the webpage.

This authentication method is not very secure, as the user's credentials can easily be accessed and decoded when transmitted over the network.