Introduction

In this assignment, I explore HTTP Basic Authentication, a widely used authentication mechanism for a long time. Technically, HTTP Basic Authentication transmits credentials as user ID/password pairs, encoded using base64. But its simplicity also exposes significant security risks, such as transmitting credentials in an easily decodable format.

I will analyze the interaction between the client (a web browser) and the server using Wireshark. My local machine acts as the client, using a web browser to send HTTP requests to the remote server at `jeffondich.com`. The server (remote) processes those requests, including authentication, and responds accordingly. Using tools, Wireshark, I can capture and decode the traffic between the client and the server, offering a detailed look at the entire authentication process.

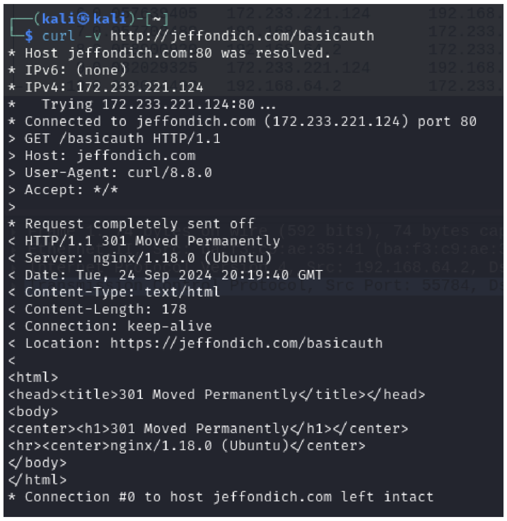
The goal of this assignment is to observe and describe the sequence of events that occurs when I (or any user) attempts to access a protected resource, identify how the browser handles authentication, and examine how the server responds. In particular, I will focus on understanding the `Authorization` header sent by the browser, which contains the user's credentials encoded in Base64, and discuss its security weaknesses. Through this analysis, I will connect my observations to the HTTP specification and evaluate the potential risks of using Basic Authentication.

Methods

1. **Step 1**: Obtain the IP address of the server for capturing in Wireshark.

   Run the following command:

   ```
   curl -v http://jeffondich.com/basicauth
   ```



```
┌──(kali㉿kali)-[~]
└─$ curl -v http://jeffondich.com/basicauth
* Host jeffondich.com:80 was resolved.
* IPv6: (none)
* IPv4: 172.233.221.124
*   Trying 172.233.221.124:80 ...
* Connected to jeffondich.com (172.233.221.124) port 80
> GET /basicauth HTTP/1.1
> Host: jeffondich.com
> User-Agent: curl/8.8.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 301 Moved Permanently
< Server: nginx/1.18.0 (Ubuntu)
< Date: Tue, 24 Sep 2024 20:19:40 GMT
< Content-Type: text/html
< Content-Length: 178
< Connection: keep-alive
< Location: https://jeffondich.com/basicauth
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx/1.18.0 (Ubuntu)</center>
</body>
</html>
* Connection #0 to host jeffondich.com left intact
```

Figure 1: Curl command

2. **Step 2**: In Wireshark, use the IP address of the host to capture packets:

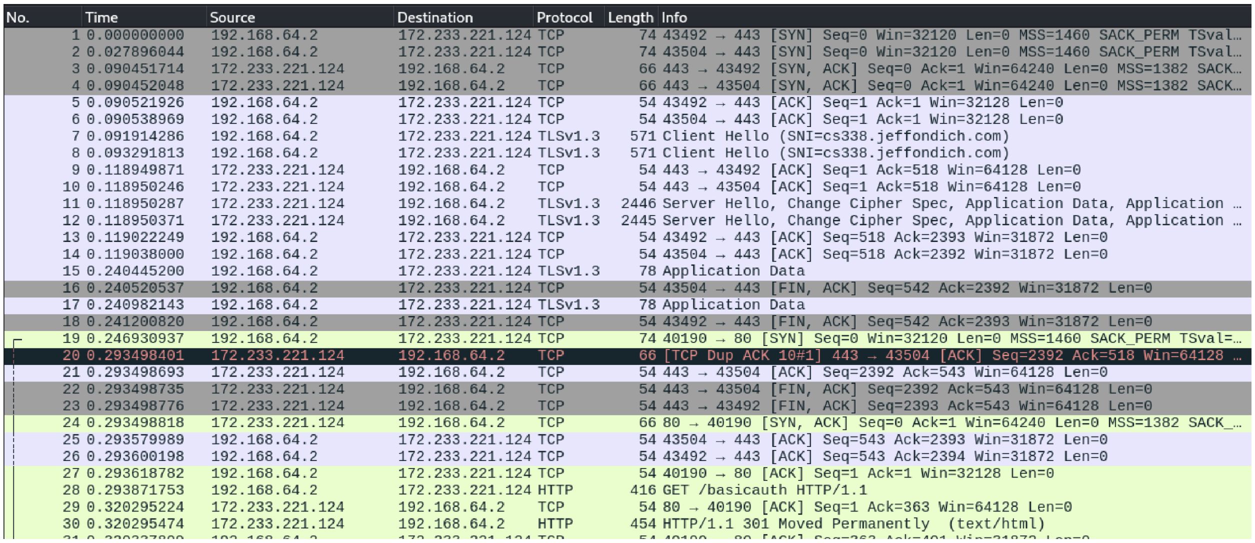
   ```
   host 172.233.221.124
   ```

3. **Step 3**: Clear the search history in Firefox (I have done this in incognito mode but haven't deleted the history; it didn't work) and search for:

   ```
   http://cs338.jeffondich.com/basicauth
   ```

   This will prompt for a username and password to sign in. The username is `cs338` and the password is `password`.

4. **Step 4**: Go back to Wireshark, where you will detect multiple frames, showing the sequence of events as follows:

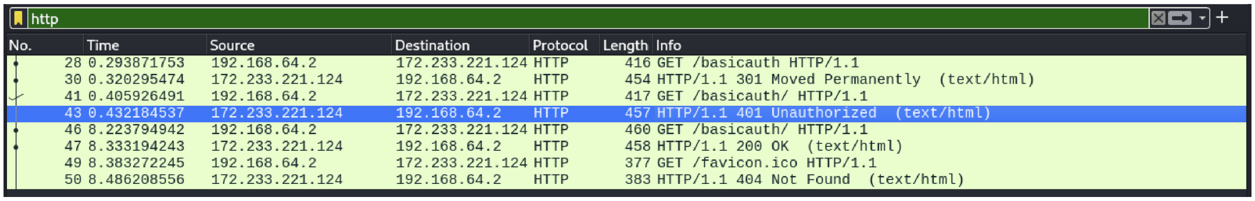


```
No.   Time         Source           Destination      Protocol Length Info
    1 0.000000000  192.168.64.2     172.233.221.124  TCP        74 43492 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval…
    2 0.027896044  192.168.64.2     172.233.221.124  TCP        74 43504 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval…
    3 0.090451714  172.233.221.124  192.168.64.2     TCP        66 443 → 43492 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK…
    4 0.090452048  172.233.221.124  192.168.64.2     TCP        66 443 → 43504 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK…
    5 0.090521926  192.168.64.2     172.233.221.124  TCP        54 43492 → 443 [ACK] Seq=1 Ack=1 Win=32128 Len=0
    6 0.090538969  192.168.64.2     172.233.221.124  TCP        54 43504 → 443 [ACK] Seq=1 Ack=1 Win=32128 Len=0
    7 0.091914286  192.168.64.2     172.233.221.124  TLSv1.3   571 Client Hello (SNI=cs338.jeffondich.com)
    8 0.093291813  192.168.64.2     172.233.221.124  TLSv1.3   571 Client Hello (SNI=cs338.jeffondich.com)
    9 0.118949871  172.233.221.124  192.168.64.2     TCP        54 443 → 43492 [ACK] Seq=1 Ack=518 Win=64128 Len=0
   10 0.118950246  172.233.221.124  192.168.64.2     TCP        54 443 → 43504 [ACK] Seq=1 Ack=518 Win=64128 Len=0
   11 0.118950287  172.233.221.124  192.168.64.2     TLSv1.3  2446 Server Hello, Change Cipher Spec, Application Data, Application …
   12 0.118950371  172.233.221.124  192.168.64.2     TLSv1.3  2445 Server Hello, Change Cipher Spec, Application Data, Application …
   13 0.119022249  192.168.64.2     172.233.221.124  TCP        54 43492 → 443 [ACK] Seq=518 Ack=2393 Win=31872 Len=0
   14 0.119038000  192.168.64.2     172.233.221.124  TCP        54 43504 → 443 [ACK] Seq=518 Ack=2392 Win=31872 Len=0
   15 0.240445200  192.168.64.2     172.233.221.124  TLSv1.3    78 Application Data
   16 0.240520537  192.168.64.2     172.233.221.124  TCP        54 43504 → 443 [FIN, ACK] Seq=542 Ack=2392 Win=31872 Len=0
   17 0.240982143  192.168.64.2     172.233.221.124  TLSv1.3    78 Application Data
   18 0.241200820  192.168.64.2     172.233.221.124  TCP        54 43492 → 443 [FIN, ACK] Seq=542 Ack=2393 Win=31872 Len=0
   19 0.246930937  192.168.64.2     172.233.221.124  TCP        74 40190 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=…
   20 0.293498401  172.233.221.124  192.168.64.2     TCP        66 [TCP Dup ACK 10#1] 443 → 43504 [ACK] Seq=2392 Ack=518 Win=64128 …
   21 0.293498693  172.233.221.124  192.168.64.2     TCP        54 443 → 43504 [ACK] Seq=2392 Ack=543 Win=64128 Len=0
   22 0.293498735  172.233.221.124  192.168.64.2     TCP        54 443 → 43504 [FIN, ACK] Seq=2392 Ack=543 Win=64128 Len=0
   23 0.293498776  172.233.221.124  192.168.64.2     TCP        54 443 → 43492 [FIN, ACK] Seq=2393 Ack=543 Win=64128 Len=0
   24 0.293498818  172.233.221.124  192.168.64.2     TCP        66 80 → 40190 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_…
   25 0.293579989  192.168.64.2     172.233.221.124  TCP        54 43504 → 443 [ACK] Seq=543 Ack=2393 Win=31872 Len=0
   26 0.293600198  192.168.64.2     172.233.221.124  TCP        54 43492 → 443 [ACK] Seq=543 Ack=2394 Win=31872 Len=0
   27 0.293618782  192.168.64.2     172.233.221.124  TCP        54 40190 → 80 [ACK] Seq=1 Ack=1 Win=32128 Len=0
   28 0.293871753  192.168.64.2     172.233.221.124  HTTP      416 GET /basicauth HTTP/1.1
   29 0.320295224  172.233.221.124  192.168.64.2     TCP        54 80 → 40190 [ACK] Seq=1 Ack=363 Win=64128 Len=0
   30 0.320295474  172.233.221.124  192.168.64.2     HTTP      454 HTTP/1.1 301 Moved Permanently  (text/html)
```

Figure 2: Frames

- **TCP Handshake**: The client (`192.168.64.2`) sends a SYN packet to initiate a connection with the server (`172.233.221.124`). The server responds with a SYN-ACK, and the client completes the handshake with an ACK. This establishes a TCP connection.

5. **Step 5**: Filter for HTTP traffic looking for HTTP Basic Authentication in Wireshark, and I get frames similar to the following.



http

No.	Time	Source	Destination	Protocol	Length	Info
28	0.293871753	192.168.64.2	172.233.221.124	HTTP	416	GET /basicauth HTTP/1.1
30	0.320295474	172.233.221.124	192.168.64.2	HTTP	454	HTTP/1.1 301 Moved Permanently (text/html)
41	0.405926491	192.168.64.2	172.233.221.124	HTTP	417	GET /basicauth/ HTTP/1.1
43	0.432184537	172.233.221.124	192.168.64.2	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
46	8.223794942	192.168.64.2	172.233.221.124	HTTP	460	GET /basicauth/ HTTP/1.1
47	8.333194243	172.233.221.124	192.168.64.2	HTTP	458	HTTP/1.1 200 OK (text/html)
49	8.383272245	192.168.64.2	172.233.221.124	HTTP	377	GET /favicon.ico HTTP/1.1
50	8.486208556	172.233.221.124	192.168.64.2	HTTP	383	HTTP/1.1 404 Not Found (text/html)

Figure 3: HTTP Frames

The following is an explanation of the 8 frames captured from the http filtering:

(a) **Frame 28**

- **Source**: 192.168.64.2
- **Destination**: 172.233.221.124
- **Protocol**: HTTP
- **Length**: 416
- **Info**: `GET /basicauth HTTP/1.1`

The client (192.168.64.2) sends a GET request to the server (172.233.221.124) to access the `/basicauth` page using HTTP version 1.1.

(b) **Frame 30**

- **Source**: 172.233.221.124
- **Destination**: 192.168.64.2
- **Protocol**: HTTP
- **Length**: 454
- **Info**: `HTTP/1.1 301 Moved Permanently`

The server responds with a `301` status code, indicating that the requested resource has been moved permanently to a new location. In this case, it moved to the user and password popup page.

(c) **Frame 41**

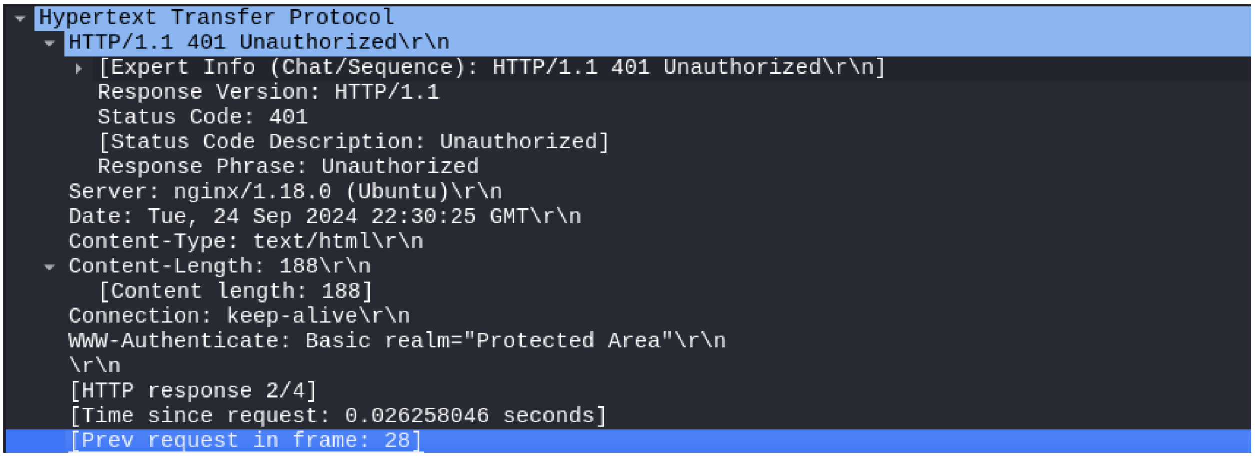
- **Source**: 192.168.64.2
- **Destination**: 172.233.221.124
- **Protocol**: HTTP
- **Length**: 417
- **Info**: `GET /basicauth/ HTTP/1.1`

The client follows the redirection and sends another GET request to `/basicauth/`.

(d) **Frame 43**

- **Source**: 172.233.221.124
- **Destination**: 192.168.64.2
- **Protocol**: HTTP
- **Length**: 457
- **Info**: `HTTP/1.1 401 Unauthorized`

The server responds with a `401 Unauthorized` status, indicating that authentication is required to access the resource.



```
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 401 Unauthorized\r\n
    ▸ [Expert Info (Chat/Sequence): HTTP/1.1 401 Unauthorized\r\n]
      Response Version: HTTP/1.1
      Status Code: 401
      [Status Code Description: Unauthorized]
      Response Phrase: Unauthorized
    Server: nginx/1.18.0 (Ubuntu)\r\n
    Date: Tue, 24 Sep 2024 22:30:25 GMT\r\n
    Content-Type: text/html\r\n
  ▾ Content-Length: 188\r\n
      [Content length: 188]
    Connection: keep-alive\r\n
    WWW-Authenticate: Basic realm="Protected Area"\r\n
    \r\n
    [HTTP response 2/4]
    [Time since request: 0.026258046 seconds]
    [Prev request in frame: 28]
```

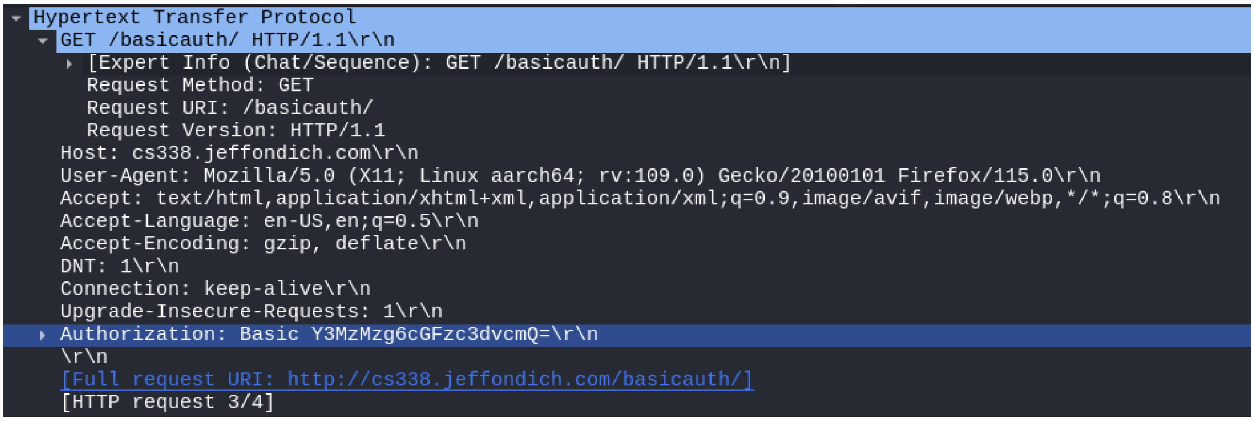
Figure 4: Frame 43 (Hypertext Transfer Protocol)

The client sends another GET request, and the server responds with a `401 Unauthorized` message (seen in Frame 43). This response includes the `Authenticate` header indicating that the server requires Basic authentication.

(e) **Frame 46**

- **Source**: 192.168.64.2
- **Destination**: 172.233.221.124
- **Protocol**: HTTP
- **Length**: 460
- **Info**: `GET /basicauth/ HTTP/1.1`

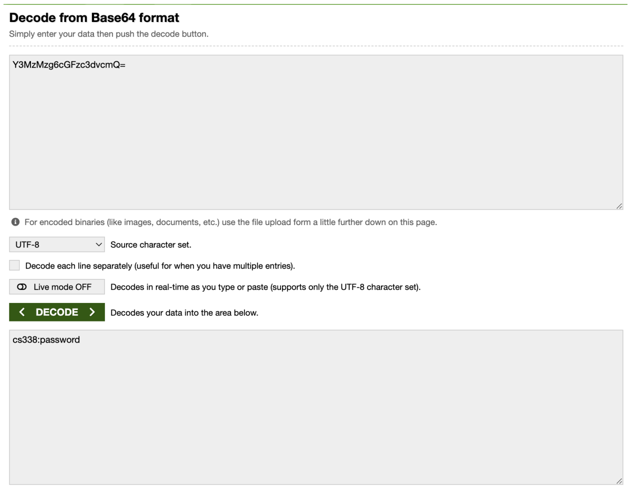
The client sends another GET request to access the `/basicauth/` resource.



```
▾ Hypertext Transfer Protocol
  ▾ GET /basicauth/ HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): GET /basicauth/ HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /basicauth/
      Request Version: HTTP/1.1
    Host: cs338.jeffondich.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
  ▸ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
    \r\n
    [Full request URI: http://cs338.jeffondich.com/basicauth/]
    [HTTP request 3/4]
```

Figure 5: Frame 46 (Hypertext Transfer Protocol)

The client retries the request, but this time it includes the Basic Authorization header. The value of this header,
`Authorization:  Basic Y3MzMzg6cGFzc3dvcmQ=`



Decode from Base64 format

Simply enter your data then push the decode button.

```
Y3MzMzg6cGFzc3dvcmQ=
```

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

□ Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE > Decodes your data into the area below.

```
cs338:password
```

Figure 6: Decode from Base64 format

A Base64-encoded string, which contains the username and password concatenated with a colon. The credentials are transmitted in human-readable text after decoding.

(f) **Frame 47**

- **Source**: 172.233.221.124
- **Destination**: 192.168.64.2
- **Protocol**: HTTP
- **Length**: 458
- **Info**: `HTTP/1.1 200 OK`

The server responds with a `200 OK` status, indicating that the resource has been successfully accessed.

(g) **Frame 49**

- **Source**: 192.168.64.2
- **Destination**: 172.233.221.124
- **Protocol**: HTTP
- **Length**: 377
- **Info**: `GET /favicon.ico HTTP/1.1`

The client requests the favicon, which is the small icon that appears in the browser tab.

(h) **Frame 50**

- **Source**: 172.233.221.124
- **Destination**: 192.168.64.2
- **Protocol**: HTTP
- **Length**: 383
- **Info**: `HTTP/1.1 404 Not Found`

The server responds with a `404 Not Found` status, indicating that the requested favicon resource does not exist on the server.

Conclusions

In conclusion, this analysis began with a series of HTTP requests sent by the client, followed by a redirection response from the server. Eventually, the server prompted for authentication using a 401 Unauthorized response, indicating that the client needed to type valid credentials. After resending the request with the Authorization header containing Base64-encoded credentials, the server granted access to the protected resource with a 200 OK response.

This exploration highlighted the core security issue of Basic Authentication: credentials are transmitted in a format that is easily decoded, making them vulnerable if intercepted. While it offers a straightforward mechanism for authentication (same for every server with http), its lack of encryption poses significant risks, especially over unsecured networks.

Through this assignment, I developed a deeper understanding of HTTP Basic Authentication, its weaknesses, and the broader client-server interaction. The use of tools like Wireshark provided an insightful look into the HTTP Basic Authentication.