

HTTP's Basic Authentication: A Story

In this assignment, I will be telling the somewhat detailed story about how a user signs in and gains access to <http://cs338.jeffondich.com/basicauth/>.

TCP Handshake

The story begins with the user inputting the url into their browser. This prompts the fabled TCP handshake, in this case opening two TCP connections. The browser initiates contact with the website's server, sending two [syn] packets requesting communications to be opened:

0.000000000	192.168.64.4	45.79.89.123	TCP	74 53334 → 80 [SYN]
0.000044656	192.168.64.4	45.79.89.123	TCP	74 53336 → 80 [SYN]

The server responds by sending a [SYN] packet back to the client browser, stating that it wants to communicate as well. It also sends an [ACK] package, signaling to the client that it received its previous message.

The client responds to this by sending its own [ACK] package to acknowledge the server's urge to talk. This interaction happens twice, once for each TCP connection that is opened:

0.051586175	45.79.89.123	192.168.64.4	TCP	66 80 → 53334 [SYN,
0.051606337	192.168.64.4	45.79.89.123	TCP	54 53334 → 80 [ACK]
0.051586258	45.79.89.123	192.168.64.4	TCP	66 80 → 53336 [SYN,
0.051616418	192.168.64.4	45.79.89.123	TCP	54 53336 → 80 [ACK]

Thus two TCP connections are established.

HTTP Basic Authentication

Now that the TCP connections exist, the browser wants access to the website. The browser requests access from the website with its first GET request:

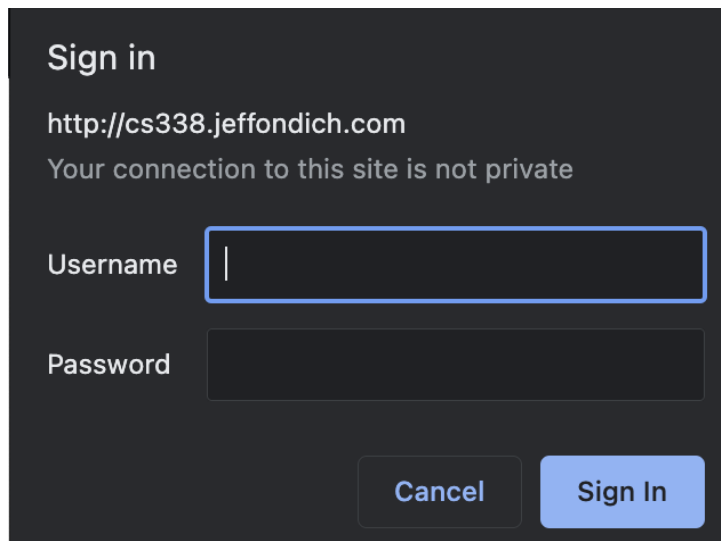
```
0.051832743 192.168.64.4 45.79.89.123 HTTP 409 GET /basicauth/ HTTP/1.1
```

Unfortunately for the browser client, it has not yet signed into the website for access. The server determines that the client does not have access, and returns a 401 message stating such:

```
0.103966292 45.79.89.123 192.168.64.4 HTTP
```

```
▼ 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: Wed, 20 Sep 2023 04:22:19 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
```

Upon receiving this message, the browser realizes that it must ask its user for authorization credentials. This prompts the browser to display a window asking for username and password:



Sign in

http://cs338.jeffondich.com

Your connection to this site is not private

Username

Password

The user inputs the username and password (cs338 and password) and hits the sign in button. The client constructs the “user pass” which starts out as the username and password concatenated together by a colon. The user pass is encoded into an octet sequence then into Base64. (Source: <https://datatracker.ietf.org/doc/html/rfc7617>)

At this point, the browser once again requests access from the website with a GET /basicauth/, this time including the aforementioned user pass under Authorization:

```
27.766999450 192.168.64.4 45.79.89.123 HTTP 452 GET /basicauth/ HTTP/1.1
```

```
▼ 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/109.0\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
      Connection: keep-alive\r\n
      Upgrade-Insecure-Requests: 1\r\n
      Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
        Credentials: cs338:password
```

Upon receiving this GET request, the server will examine the encoded user pass and determine that the user has inputted valid credentials for gaining access. It will acknowledge this by sending a 200 ok message back to the client along with the html for the website:

```
27.832530616 45.79.89.123 192.168.64.4 HTTP 458 HTTP/1.1 200 OK (text/html)
```

```
▼ 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>
  <a href="armed-guards.txt">armed-guards.txt</a>
  <a href="dancing.txt">dancing.txt</a>
  </pre><hr></body>\r\n
  </html>\r\n
```

The user is now able to see the website!

Index of /basicauth/

../		
amateurs.txt	04-Apr-2022 14:10	75
armed-guards.txt	04-Apr-2022 14:10	161
dancing.txt	04-Apr-2022 14:10	227

The final notable interaction (still of dramatically lesser importance) is the browser requesting the page's favicon from the server:

```
27.863671543 192.168.64.4 45.79.89.123 HTTP 369 GET /favicon.ico HTTP/1.1
```

The server attempts to deliver the favicon but for whatever reason is unable to find it. As a response it returns an error 404 message:

```
27.918173313 45.79.89.123 192.168.64.4 HTTP 383 HTTP/1.1 404 Not Found
```

Causing the browser to display its default favicon, in my case chrome displaying:



(Source:

<https://www.concretecms.com/about/blog/web-design/what-why-where-and-how-favicons#:~:text=If%20a%20favicon%20is%20missing%2C%20the%20browser%20will%20default%20to,in%20your%20history%20or%20tabs.>)

And thus concludes a somewhat detailed story about how a user signs in and gains access to <http://cs338.jeffondich.com/basicauth/>.