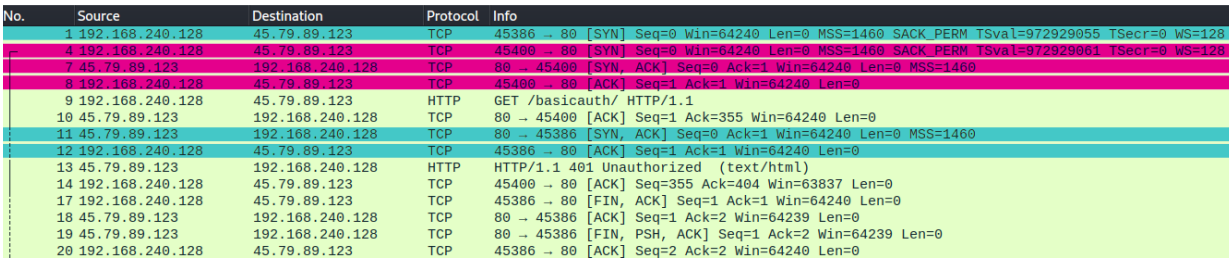
The Story of an HTTP Basic Authentication

Cole Weinstein

Connecting to a web page is a complex process, requiring exact steps through a specific course of events, and this task is made even more difficult when a consideration of authentication and authorization becomes involved. However, the HTTP Basic Authentication begins simply, as all network connections do, with a TCP handshake between the client and server. For the rest of this story, we'll consider the client to be my Kali virtual machine and the server to be the webpage http://cs338.jeffondich.com/basicauth/. In Figure 1, we can ultimately see that two TCP handshakes occur to establish a connection with the server: one between port 45386 on the client and port 80 on the server, as shown in frames 1, 11, and 12, and a second between port 45400 on the client and port 80 on the server, as shown in frames 4, 7, and 8.



No.	Source	Destination	Protocol	Info
1	192.168.240.128	45.79.89.123	TCP	45386 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=972929055 TSecr=0 WS=128
4	192.168.240.128	45.79.89.123	TCP	45400 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=972929061 TSecr=0 WS=128
7	45.79.89.123	192.168.240.128	TCP	80 → 45400 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
8	192.168.240.128	45.79.89.123	TCP	45400 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
9	192.168.240.128	45.79.89.123	HTTP	GET /basicauth/ HTTP/1.1
10	45.79.89.123	192.168.240.128	TCP	80 → 45400 [ACK] Seq=1 Ack=355 Win=64240 Len=0
11	45.79.89.123	192.168.240.128	TCP	80 → 45386 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
12	192.168.240.128	45.79.89.123	TCP	45386 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
13	45.79.89.123	192.168.240.128	HTTP	HTTP/1.1 401 Unauthorized (text/html)
14	192.168.240.128	45.79.89.123	TCP	45400 → 80 [ACK] Seq=355 Ack=404 Win=63837 Len=0
17	192.168.240.128	45.79.89.123	TCP	45386 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
18	45.79.89.123	192.168.240.128	TCP	80 → 45386 [ACK] Seq=1 Ack=2 Win=64239 Len=0
19	45.79.89.123	192.168.240.128	TCP	80 → 45386 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
20	192.168.240.128	45.79.89.123	TCP	45386 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0

Figure 1: Initial connection between client and server. Two TCP handshakes are depicted; one among frames 1, 11, and 12 (highlighted in blue), and another among frames 4, 7, and 8 (highlighted in pink).

Within Figure 1, we can also see the results of the initial connection. An HTTP GET request is made by the client in frame 9 for the directory /basicauth/ on the server, and is met with an HTTP "401 Unauthorized" error in frame 13. We can see that the Hypertext Transfer Protocol has its own header in the packet, as the response is an HTTP response, and delving into this header shows us two important pieces of information.

To start, we see the line

```
HTTP/1.1 401 Unauthorized
```

which explains to the client why the requested content could not be returned. Then, we see the line

```
WWW-Authenticate: Basic realm="Protected Area"\r\n
```

which clarifies to the client that the content could not be returned because only specific users are authorized to access it. Both of these items exist as outlined in section 2 of the RFC 7617 document which defines the protocol behind HTTP Basic Authentication (pg. 3).

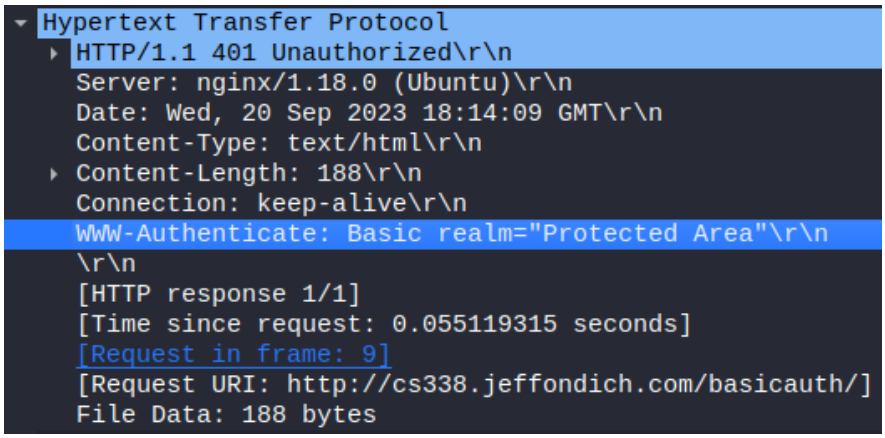


Figure 2: HTTP Header in packet 13 from Figure 1. Highlighted in light blue is the HTTP status code, and highlighted in blue is the WWW-Authenticate header field.

At this point, the browser prompts me for a username and password to return to the server (Figure 3).

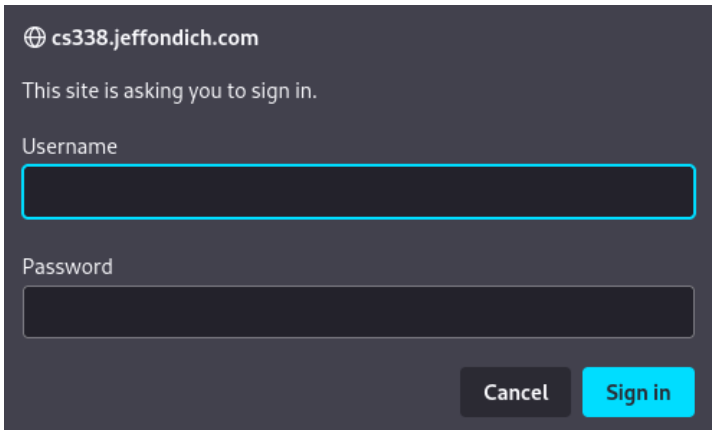


Figure 3: Prompt for password authentication made by browser.

After I type in the username and password, the browser client sends the server another HTTP GET request, but this time includes the username and password in the request. Looking at the HTTP header, we can see that a new item exists, named "Authorization" and followed by a sequence of characters. While it's not initially obvious, this string is actually just a base64 encoding of the username followed by a colon followed by the password. In this way, both of the credentials in their entirety can be sent to the server, and the server knows exactly which part is the username and which part is the password.

```
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
```

In a terminal window:

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

Clearly, the user credentials are not encrypted so anyone viewing these packets would easily be able to read our login info. However, the extra data in the packet is enough, and as we see in frame 680 of Figure 4 the server responds with an HTTP response of "200 OK". These actions again follow the steps outlined in section 2 of the RFC 7617 document (pg. 4).

No.	Source	Destination	Protocol	Info
675	192.168.240.128	45.79.89.123	TCP	46906 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=974385214 TSecr=0 WS=128
676	45.79.89.123	192.168.240.128	TCP	80 → 46906 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
677	192.168.240.128	45.79.89.123	TCP	46906 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
678	192.168.240.128	45.79.89.123	HTTP	GET /basicauth/ HTTP/1.1
679	45.79.89.123	192.168.240.128	TCP	80 → 46906 [ACK] Seq=1 Ack=398 Win=64240 Len=0
680	45.79.89.123	192.168.240.128	HTTP	HTTP/1.1 200 OK (text/html)
681	192.168.240.128	45.79.89.123	TCP	46906 → 80 [ACK] Seq=398 Ack=405 Win=63836 Len=0
682	192.168.240.128	45.79.89.123	HTTP	GET /favicon.ico HTTP/1.1
683	45.79.89.123	192.168.240.128	TCP	80 → 46906 [ACK] Seq=405 Ack=712 Win=64240 Len=0
684	45.79.89.123	192.168.240.128	HTTP	HTTP/1.1 404 Not Found (text/html)
685	192.168.240.128	45.79.89.123	TCP	46906 → 80 [ACK] Seq=712 Ack=734 Win=63836 Len=0

Figure 4: Second connection between client and server. HTTP GET request with Authorization made by the client highlighted in orange. 200 OK response made by the server highlighted in green. (We also see another TCP handshake at the top of the screen due to time-out while taking screenshots.)

Consequently, the browser loads the webpage and everyone is happy. If we inspect the HTTP GET requests made to access the files in /basicauth/, we see the same "Authorization" tag holding the same field data sent by the browser in each of these requests, added from the browser's cache storage so that the user doesn't have to enter a username and password every time they want to access a different page. Even if they leave the site and come back, they still won't be prompted for a password (provided that the credentials are still stored by the browser).