Mary Blanchard and Kaitlyn Peterson

HTTP's Basic Authentication: A Story

The first thing that we saw in Wireshark when we began the process of accessing Jeff's secret website was a series of DNS queries, which turned the link into an IP address. Having the IP address allowed the client to initiate a TCP connection.

| | | | | | | |
|---|---|---|---|---|---|---|
| 39 | 35.792970269 | 192.168.12.129 | 192.168.12.2 | DNS | 80 | Standard query 0xe972 A cs338.jeffondi |
| 40 | 35.793025475 | 192.168.12.129 | 192.168.12.2 | DNS | 80 | Standard query 0xab70 AAAA cs338.jeffor |
| 41 | 35.795599805 | 192.168.12.2 | 192.168.12.129 | DNS | 390 | Standard query response 0xe972 A cs338. |
| 42 | 35.806923805 | 192.168.12.129 | 192.168.12.2 | DNS | 80 | Standard query 0x3f3f A cs338.jeffondi |
| 43 | 35.806967753 | 192.168.12.129 | 192.168.12.2 | DNS | 80 | Standard query 0xee38 AAAA cs338.jeffor |
| 44 | 35.809844238 | 192.168.12.2 | 192.168.12.129 | DNS | 390 | Standard query response 0x3f3f A cs338. |
| 45 | 35.823352532 | 192.168.12.2 | 192.168.12.129 | DNS | 159 | Standard query response 0xab70 AAAA cs3 |
| 46 | 35.823452606 | 192.168.12.2 | 192.168.12.129 | DNS | 159 | Standard query response 0xee38 AAAA cs3 |

Beginning at frame 47, there are two TCP handshakes. The client initiates two connections, both from client port 54312. One goes to the server's port 80 (signifying an HTTP connection) and the other to server's port 443 (HTTPS connection). It does so by sending [SYN] flags to both these server locations using TCP. The servers acknowledge this and send back [SYN][ACK]. Finally, the client sends [ACK] to both server ports. The TCP handshake is complete and two connections are established.

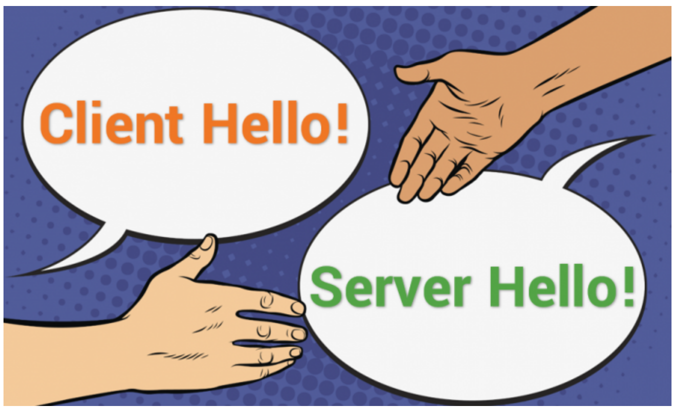
| | | | | | | |
|---|---|---|---|---|---|---|
| 47 | 35.823707983 | 192.168.12.129 | 45.79.89.123 | TCP | 74 | 54312 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14 |
| 48 | 35.823838741 | 192.168.12.129 | 45.79.89.123 | TCP | 74 | 38680 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1 |
| 49 | 35.868902554 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 80 → 54312 [SYN, ACK] Seq=0 Ack=1 Win=64240 L |
| 50 | 35.868955650 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 54312 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |
| 51 | 35.868902870 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [SYN, ACK] Seq=0 Ack=1 Win=64240 |
| 52 | 35.869008324 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 38680 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0 |

Next, wireshark displays frames 53 through 64 in pink! These frames use TCP and TLS protocols to attempt to establish a secure and encrypted connection through the server's port 443. Frame 53 is the client sending the server a "Client Hello." The server acknowledges this and sends back a "Server Hello." After this, a process similar to the TCP handshake occurs, where the TLS protocol allows the client and server to exchange keys that they could use in an encrypted connection. However, we (the client) send an "[Encrypted Alert](#)," which is the beginning of the TLS connection termination process. The client follows it with a [FIN] flag, and the server acknowledges this. However, the server does not send a [FIN] back, and tries to send another key in frame 66. Since this is unexpected, frame 67 shows the client response, which includes the [RST] flag because the client is no longer expecting to be in the secure connection.

| | | | | | | |
|---|---|---|---|---|---|---|
| 53 | 35.871274681 | 192.168.12.129 | 45.79.89.123 | TLSv1.2 | 571 | Client Hello |
| 54 | 35.871558663 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [ACK] Seq=1 Ack=518 Win=64240 Len=0 |
| 55 | 35.917266021 | 45.79.89.123 | 192.168.12.129 | TLSv1.2 | 4150 | Server Hello |
| 56 | 35.917300703 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 38680 → 443 [ACK] Seq=518 Ack=4097 Win=61320 Len=0 |
| 57 | 35.918801608 | 45.79.89.123 | 192.168.12.129 | TLSv1.2 | 534 | Certificate, Server Key Exchange, Server Hello Done |
| 58 | 35.918815482 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 38680 → 443 [ACK] Seq=518 Ack=4577 Win=62780 Len=0 |
| 59 | 35.925839604 | 192.168.12.129 | 45.79.89.123 | TLSv1.2 | 212 | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message |
| 60 | 35.926086043 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [ACK] Seq=4577 Ack=676 Win=64240 Len=0 |
| 61 | 35.926169946 | 192.168.12.129 | 45.79.89.123 | TLSv1.2 | 85 | Encrypted Alert |
| 62 | 35.926312288 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 38680 → 443 [FIN, ACK] Seq=707 Ack=4577 Win=62780 Len=0 |
| 63 | 35.926337040 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [ACK] Seq=4577 Ack=707 Win=64240 Len=0 |
| 64 | 35.926523330 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [ACK] Seq=4577 Ack=708 Win=64239 Len=0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 64 | 35.926523330 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 443 → 38680 [ACK] Seq=4577 Ack=708 Win=64239 Len=0 |
| 65 | 35.928054785 | 192.168.12.129 | 45.79.89.123 | TCP | 74 | 54314 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T |
| 66 | 35.971808239 | 45.79.89.123 | 192.168.12.129 | TLSv1.2 | 105 | Change Cipher Spec, Encrypted Handshake Message |
| 67 | 35.971826558 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 38680 → 443 [RST] Seq=708 Win=0 Len=0 |

We believe that all of the [TLS handshaking]() and disconnecting occurs because we used an incognito browser to access the webpage, but the webpage does not use HTTPS or TLS. Because the browser is incognito, it tries to establish a secure connection by default. We used wireshark to see how an incognito browser connected to a different website and saw a very similar TLS handshake happen. However, because Jeff's website is HTTP, the secure connection is terminated and the browser tries to access it through port 80 instead of port 443.




After the secured connection fails to establish, a new TCP handshake happens to initiate a connection between the client port 54314 and server port 80 (HTTP). This is a new client port, and is separate from the connection we established earlier. Using the new port, the client makes a GET request for the webpage in frame 70 with the [HTTP protocol](). This GET request does not have an authorization header, since this is before we have typed in the password. The server acknowledges the request in TCP, and then sends an HTTP packet that says the GET request is unauthorized, and includes the HTML for the 401 Authorization Required page.

| | | | | | | |
|---|---|---|---|---|---|---|
| 70 | 35.973078731 | 192.168.12.129 | 45.79.89.123 | HTTP | 403 | GET /basicauth/ HTTP/1.1 |
| 71 | 35.973363961 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 80 → 54314 [ACK] Seq=1 Ack=350 Win=6424( |
| 72 | 36.018729690 | 45.79.89.123 | 192.168.12.129 | HTTP | 457 | HTTP/1.1 401 Unauthorized (text/html) |
| 73 | 36.018747651 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 54314 → 80 [ACK] Seq=350 Ack=404 Win=63ε |

At this point in our connection, we got several duplicate packets, which wireshark displayed in red. They each had a [[TCP Dup ACK]() 1#1] flag. Ultimately, we ignored these because they are not necessarily a part of our basic authentication connection.

| | | | | |
|---|---|---|---|---|
| 74 37.375198321 | 192.168.12.129 | 54.192.58.25 | TCP | 54 [TCP Dup ACK 5#3] 46946 → 443 [ACK] |
| 75 37.375310232 | 192.168.12.129 | 142.250.190.67 | TCP | 54 [TCP Dup ACK 6#3] 35420 → 80 [ACK] |
| 76 37.375312284 | 192.168.12.129 | 72.21.91.29 | TCP | 54 [TCP Dup ACK 7#3] 60410 → 80 [ACK] |
| 77 37.375689518 | 54.192.58.25 | 192.168.12.129 | TCP | 60 [TCP Dup ACK 8#3] [TCP ACKed unseen |
| 78 37.375689764 | 142.250.190.67 | 192.168.12.129 | TCP | 60 [TCP Dup ACK 9#3] [TCP ACKed unseen |
| 79 37.375689805 | 72.21.91.29 | 192.168.12.129 | TCP | 60 [TCP Dup ACK 10#3] [TCP ACKed unsee |
| 80 38.147335846 | 192.168.12.129 | 54.192.58.23 | TCP | 54 [TCP Dup ACK 11#3] 45326 → 443 [ACK |
| 81 38.147799390 | 54.192.58.23 | 192.168.12.129 | TCP | 60 [TCP Dup ACK 12#3] [TCP ACKed unsee |

After the first GET request fails to authenticate, the client sends a TCP frame with the [FIN] flag from client port 54312, which was the original connection. The server acknowledges it, and that connection is terminated.

| | | | | |
|---|---|---|---|---|
| 82 40.872146557 | 192.168.12.129 | 45.79.89.123 | TCP | 54 54312 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 |
| 83 40.872486443 | 45.79.89.123 | 192.168.12.129 | TCP | 60 80 → 54312 [ACK] Seq=1 Ack=2 Win=64239 Len=0 |
| 84 40.917416154 | 45.79.89.123 | 192.168.12.129 | TCP | 60 80 → 54312 [FIN, PSH, ACK] Seq=1 Ack=2 Win=6 |
| 85 40.917438940 | 192.168.12.129 | 45.79.89.123 | TCP | 54 54312 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0 |

Then we see some TCP Keep-Alive requests, presumably while the website waits for us to put in our username and password. We also see a few more Dup packets.

In frame 99, the client sends a GET request through HTTP. This request has an authorization header. Within this header, we can see the credentials, which are readable to us as "cs338:password" because Wireshark interprets them for us. The actual data sent over the network is the bytes highlighted at the bottom of our screenshot. This is just the byte representation of the user credentials which have been encoded using base 64, per the Basic HTTP Authentication Scheme (section 2).

Within the Hypertext Transfer Protocol header of frame 99:

```
    Upgrade-Insecure-Requests: 1\r\n
  ▾ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
      Credentials: cs338:password
    \r\n
    [Full request URI: http://cs338.jeffondich.com/basicauth/]
    [HTTP request 2/3]
    [Prev request in frame: 70]
    [Response in frame: 101]
    [Next request in frame: 103]

0100  65 2f 77 65 62 70 2c 2a  2f 2a 3b 71 3d 30 2e 38   e/webp,* /*;q=0.8
0110  0d 0a 41 63 63 65 70 74  2d 4c 61 6e 67 75 61 67   ··Accept -Languag
0120  65 3a 20 65 6e 2d 55 53  2c 65 6e 3b 71 3d 30 2e   e: en-US ,en;q=0.
0130  35 0d 0a 41 63 63 65 70  74 2d 45 6e 63 6f 64 69   5··Accep t-Encodi
0140  6e 67 3a 20 67 7a 69 70  2c 20 64 65 66 6c 61 74   ng: gzip , deflat
0150  65 0d 0a 44 4e 54 3a 20  31 0d 0a 43 6f 6e 6e 65   e··DNT:  1··Conne
0160  63 74 69 6f 6e 3a 20 6b  65 65 70 2d 61 6c 69 76   ction: k eep-aliv
0170  65 0d 0a 55 70 67 72 61  64 65 2d 49 6e 73 65 63   e··Upgra de-Insec
0180  75 72 65 2d 52 65 71 75  65 73 74 73 3a 20 31 0d   ure-Requ ests: 1·
0190  0a 41 75 74 68 6f 72 69  7a 61 74 69 6f 6e 3a 20   ·Authori zation: 
01a0  42 61 73 69 63 20 59 33  4d 7a 4d 7a 67 36 63 47   Basic Y3 MzMzg6cG
01b0  46 7a 63 33 64 76 63 6d  51 3d 0d 0a 0d 0a         Fzc3dvcm Q=····
```

To be clear, the user credentials were encoded, not encrypted. Anyone who intercepts the GET request with the credentials, can see that it's encoded using the Basic Authentication scheme, and use a simple base 64 decoder program to get the original credential string back. As section 4, Security Considerations, of the Basic Authentication scheme documentation states, "This scheme is not considered to be a secure method of user authentication unless used in conjunction with some external secure system such as TLS."

After the server receives our GET request, it acknowledges the request, authenticates our credentials, and then sends the actual HTML of the restricted website via HTTP. The client acknowledges, and then we send a GET request for a favicon, which does not exist, so the server sends back a 404 error. After this, nothing else interesting happens until we close the webpage, which we did not capture via Wireshark, since we've seen it before.

| | | | | | | |
|---|---|---|---|---|---|---|
| 99 | 50.525998754 | 192.168.12.129 | 45.79.89.123 | HTTP | 446 | GET /basicauth/ HTTP/1.1 |
| 100 | 50.526258423 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 80 → 54314 [ACK] Seq=404 Ack=742 Win=64240 Len=0 |
| 101 | 50.572869689 | 45.79.89.123 | 192.168.12.129 | HTTP | 458 | HTTP/1.1 200 OK (text/html) |
| 102 | 50.572886619 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 54314 → 80 [ACK] Seq=742 Ack=808 Win=63837 Len=0 |
| 103 | 50.637765866 | 192.168.12.129 | 45.79.89.123 | HTTP | 363 | GET /favicon.ico HTTP/1.1 |
| 104 | 50.638062086 | 45.79.89.123 | 192.168.12.129 | TCP | 60 | 80 → 54314 [ACK] Seq=808 Ack=1051 Win=64240 Len=0 |
| 105 | 50.683759267 | 45.79.89.123 | 192.168.12.129 | HTTP | 383 | HTTP/1.1 404 Not Found (text/html) |
| 106 | 50.683775946 | 192.168.12.129 | 45.79.89.123 | TCP | 54 | 54314 → 80 [ACK] Seq=1051 Ack=1137 Win=63837 Len=0 |