Event Log of Connecting and Logging Into

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

2022/04/08

Author: Lily Li

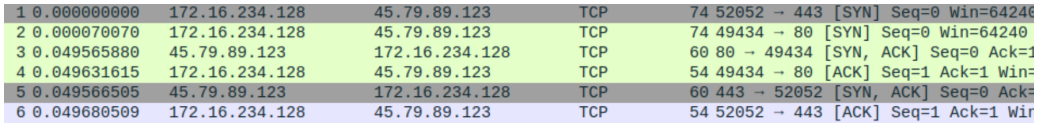
Conferred with: Jeff Ondich

Other Sources:

1. https://tls.ulfheim.net/
2. https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:online-data-security/xcae6f4a7ff015e7d:secure-internet-protocols/a/transport-layer-security-protocol-tls
3. https://developer.mozilla.org/en-US/docs/Web/HTTP

1: TCP Handshake (server port 80 & port 443):

The first thing that happened was the TCP handshake, a three-way handshake to establish connection between client and server, which happened between client port 49434 and server port 80 (for HTTP) and client port 52052 and server port 443 (for HTTPS) of the server



1	0.000000000	172.16.234.128	45.79.89.123	TCP	74	52052 → 443 [SYN] Seq=0 Win=64240
2	0.000070070	172.16.234.128	45.79.89.123	TCP	74	49434 → 80 [SYN] Seq=0 Win=64240
3	0.049565880	45.79.89.123	172.16.234.128	TCP	60	80 → 49434 [SYN, ACK] Seq=0 Ack=1
4	0.049631615	172.16.234.128	45.79.89.123	TCP	54	49434 → 80 [ACK] Seq=1 Ack=1 Win=
5	0.049566505	45.79.89.123	172.16.234.128	TCP	60	443 → 52052 [SYN, ACK] Seq=0 Ack=
6	0.049680509	172.16.234.128	45.79.89.123	TCP	54	52052 → 443 [ACK] Seq=1 Ack=1 Win

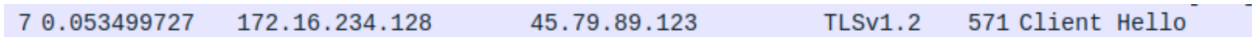
2: TLS Handshake (server port 443):

Interestingly, although Jeff said he did not set up the domain for enabling HTTPS, a TLS session took place between client port 52052 and server port 443 (for HTTPS) right after the TCP handshake:

The Transport Layer Security (TLS) protocol is built upon the TCP/IP as a session layer. Therefore, the TLS Handshake happens after the 3-Way TCP handshake is complete. TLS adds a layer of security on top of the TCP/IP transport protocol for securely sending private data.

A: Client Hello

The client informs the server that it desires a TLS connection instead of the standard insecure connection with the **Client Hello** Handshake Protocol, and provides ***preferred cryptographic Information***



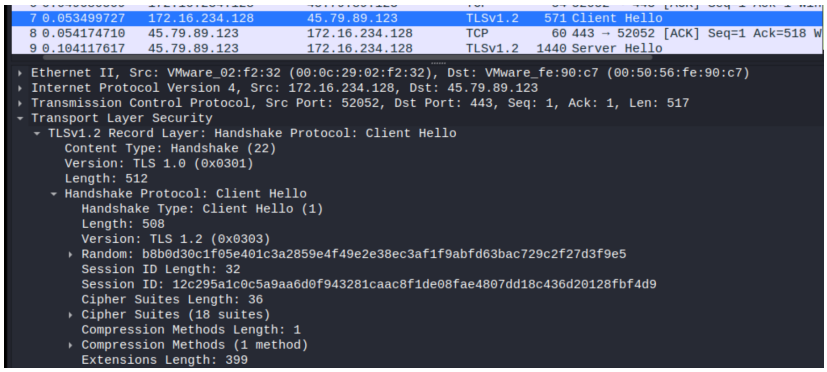
7	0.053499727	172.16.234.128	45.79.89.123	TLSv1.2	571	Client Hello

According to the screen shot, the client provides:

- Protocol version: *TLS 1.2*
- Client random data: *b8b0d30c1f05e401c3a2859e4f49…*
- Session ID: *12c295a1c0c5a9aa6d0f943281ca…*
- Cipher suites: 18 *suites…*

  An ordered list of which cryptographic methods it will support for key exchange, encryption with that exchanged key, and message authentication
- Compression methods

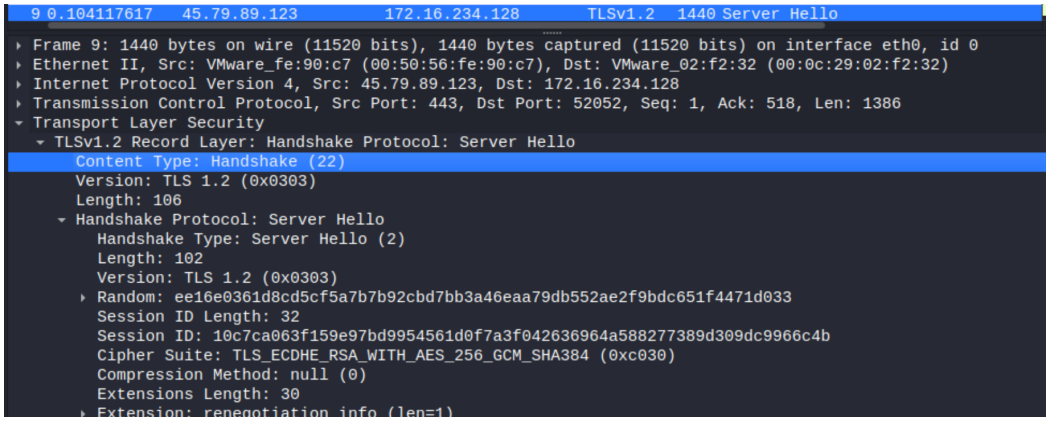
- Extensions



B: Server Hello

The server supports the requested TLS protocol version and cryptographic methods, and responds with a **Server Hello** confirmation, with its selected protocol version, cipher suite, and compression method, as in the screenshot below:

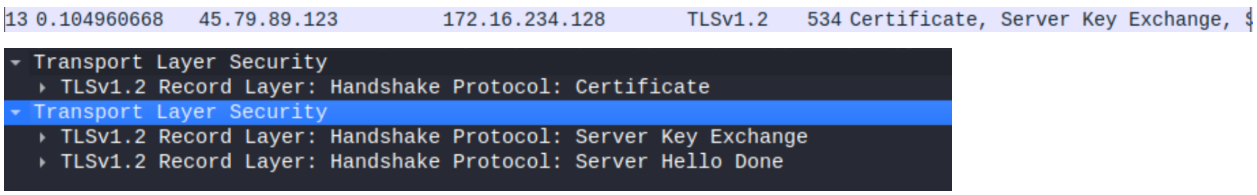
- Protocol version: *TLS 1.2*
- Server random data: *ee16e0361d8cd5cf5a7b7b92cbd7bb3a46ea…*
- Session ID:*10c7ca063f159e97bd9954561d0f7a3…*
- **Selected Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)**
- Compression method
- Extensions



C: Server Certificate, Server Key Exchange, Server Hello Done

Then the server sends over a packet containing:

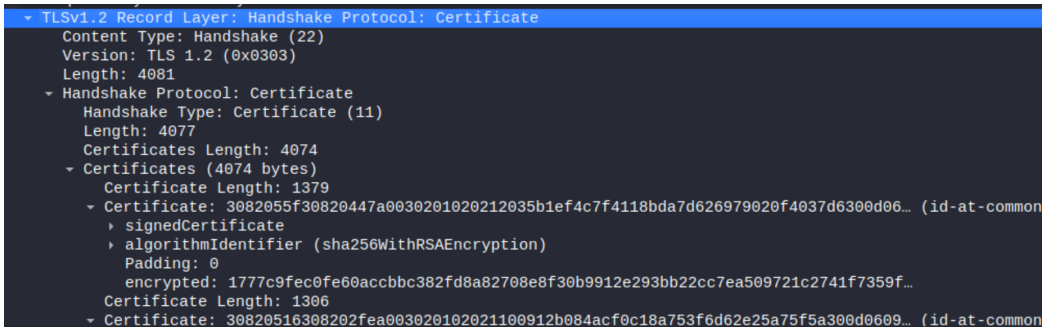
Server Certificate, Server Key Exchange, Server Hello Done



i.Server Certificate

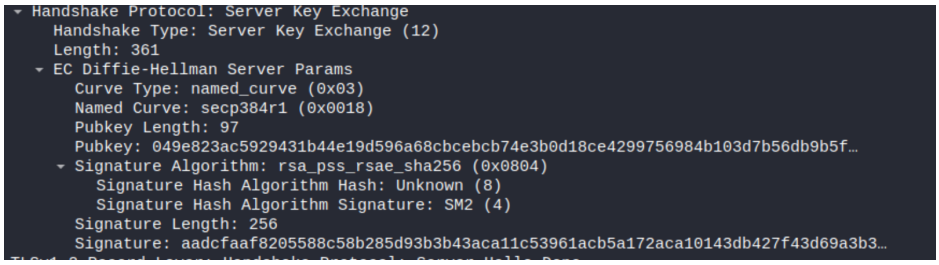
The server's digital certificate is the server's way of certifying ownership of a public key, issued by CA. The client may abort the connection if it doesn't believe in the certificate.The *certificate* contains:

1. Handshake header (type and length)
2. Certification Length
3. Encrypted: hostname, public key, proof from a trusted third party that the owner of this hostname holds the private key for this public key, dates… etc



ii.Server Key Exchange

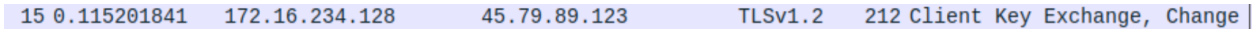
Server provides its **public key**. Both the server and the client will have a keypair of public and private keys, and will send the other party their public key. The shared encryption key will then be generated using a combination of each party's private key and the other party's public key.

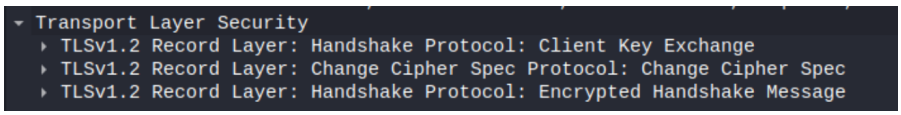


iii.Server Hello Done

The server indicates that it is done with half of the handshake.

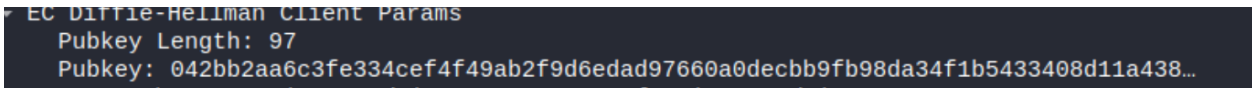
C: Client Key Exchange, Client Cipher Spec, Encrypted Handshake Message





i.Client Key Exchange

Similar to server key exchange, the client provides it's **public key**



ii.Client Cipher Spec

The client indicates that it has calculated the shared encryption keys and that all following messages from the client will be encrypted with the client write key.

iii.Encrypted Handshake Message

To verify that the handshake was successful, the encrypts **verification data** with the client's write key and sends it over

D: Encrypted Alert (Close of Session)

```
 17 0.118775906   172.16.234.128      45.79.89.123        TLSv1.2    85 Encrypted Alert
```

"Encrypted Alert" is used in for notifying to close the connection (end of session)

E: Serve Change Cipher Spec, Encrypted Handshake Message

```
22 0.169536180   45.79.89.123        172.16.234.128      TLSv1.2   105 Change Cipher Spec, Encrypted Hand
```

i.Server Cipher Spec

Similar to Client Cipher Spec, indicates that all following messages will be encrypted with the server write key.

ii.Encrypted Handshake Message

Similar to that of a client.

Question 01:

It is observed that frame 6(TCP ACK from Client) and frame 7 (Client Hello) has the same TCP sequence number, but different IP Identification number. Is it that IP broke the same TCP packet into 2 because it is too long? Similar phenomenon can also be observed in frame 9 (Server Hello) and the frame right before it.

```
 6 0.049680509   172.16.234.128      45.79.89.123        TCP         54 52052 → 443 [ACK] Seq=1 Ack=1 Win
 7 0.053499727   172.16.234.128      45.79.89.123        TLSv1.2    571 Client Hello
```

3: HTTP Requests/Responses (server port 80):

After the TLS handshake, the client begins sending HTTP requests to port 80. ***NOTE*** that although we just spend an excessive effort establishing a TLS encryption method on port 443, it does not affect port 80. Therefore, the HTTP requests are **NOT ENCRYPTED** 🤣

In summary, what happens next is the client will fail to gain access to the resource without authentication, and will gain access to the website once credentials are entered.

A: Client First Attempts Get Request

The client first attempts to get resources of the webpage from the server (send a HTTP request), without entering our credentials. The request begins in the following format, followed by other header fields:

Method (GET) URI(/basicauth/) Protocol Version(HTTP/1.1)

```
26 0.178852723   172.16.234.128      45.79.89.123        HTTP      404 GET /basicauth/ HTTP/1.1
```

```
Hypertext Transfer Protocol
  GET /basicauth/ HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,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
  \r\n
  [Full request URI: http://cs338.jeffondich.com/basicauth/]
  [HTTP request 1/3]
```

B: Server's HTTP Response: Unauthorized

The Server responds with a **401** (unauthorized error code), indicating that access is token missing. A textual version of the reason is also provided in the status line, followed by other header fields, and the *Line-based text data.*

i.Header Fields

```
Transmission Control Protocol, Src Port: 80, Dst Port: 49436, Seq: 1, A
Hypertext Transfer Protocol
  HTTP/1.1 401 Unauthorized\r\n
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Fri, 08 Apr 2022 15:12:53 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
  \r\n
  [HTTP response 1/3]
  [Time since request: 0.056941597 seconds]
  [Request in frame: 26]
```

Headers worth noting:

1: *Connection:*

The value of the connection header, keep-alive, means that it allows a single TCP connection to remain open for subsequent requests to the same server to be done. So you can enter the credentials and try requesting for the service again.

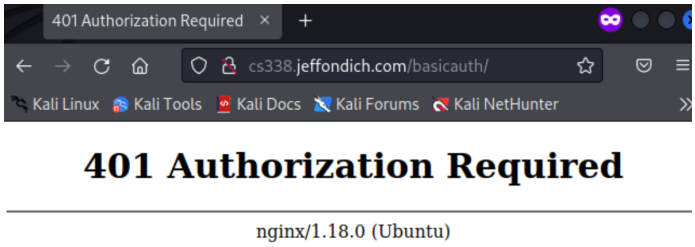
2: WWW-Authenticate:

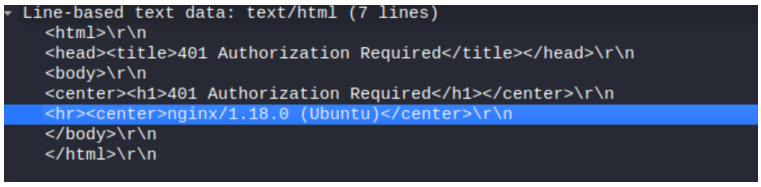
When A server using HTTP authentication will respond with a 401, it must include at least one WWW-Authenticate header and at least one *challenge*, dicates what authentication schemes can be used to access the resource.

```
28 0.235794320   45.79.89.123        172.16.234.128      HTTP      457 HTTP/1.1 401 Unauthorized  (text/H
```

ii.Line-based text data

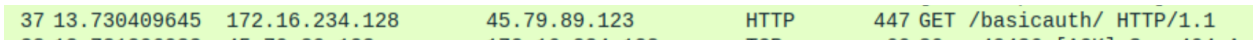
Along with the HTTP response a line-based text data was also sent, the HTML of the page that will show if you refuse to provide authentication (Press cancel on the pop out window)

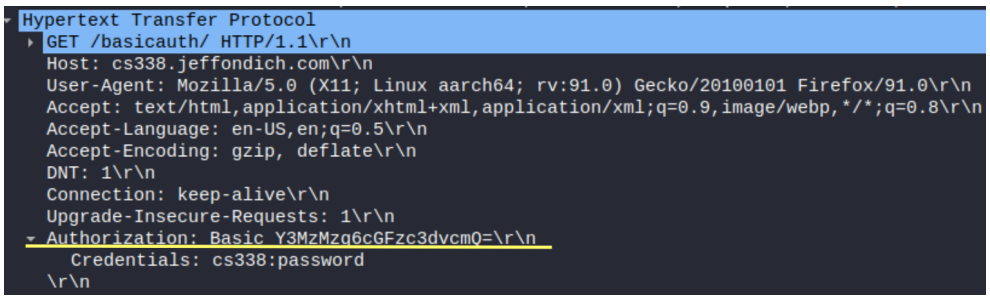




C: Client Sends Get Request With Authorization:

After you enter the password, the client will send another GET request to the server, which is very similar to the first one, but with the **Authorization Header**

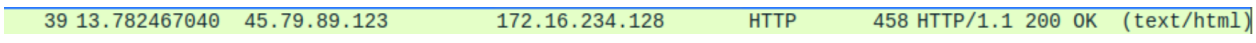




Wireshark decodes the credentials for us, but the original message is encoded. (***NOTE***: the credentials are **NOT encrypted** because we are using HTTP on port 80, which is a definite *Security Concern*!)

D: Server Responds with OKAY Status and Sends Resources!

The server sends a HTTP 200 OK success status to state that the request has succeeded. Along with the HTML of the webpage, the resources we required! **Now we have logged in to the Webpage!**



```
▸ HTTP/1.1 200 OK\r\n
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Fri, 08 Apr 2022 15:13:06 GMT\r\n
  Content-Type: text/html\r\n
  Transfer-Encoding: chunked\r\n
  Connection: keep-alive\r\n
  Content-Encoding: gzip\r\n
  \r\n
  [HTTP response 2/3]
  [Time since request: 0.052057395 seconds]
  [Prev request in frame: 26]
  [Prev response in frame: 28]
  [Request in frame: 37]
  [Next request in frame: 41]
  [Next response in frame: 43]
  [Request URI: http://cs338.jeffondich.com/favicon.ico]
▸ HTTP chunked response
  Content-encoded entity body (gzip): 205 bytes -> 509 bytes
  File Data: 509 bytes
▾ 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
```

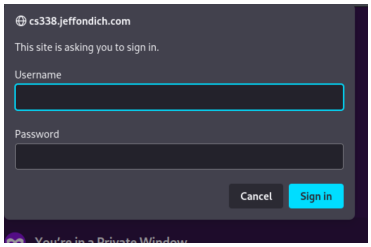
E: Failed to get an icon

The Client also attempts to GET an outer-sourced icon that is used in the webpage, but the server was not able to find it. Most probably is because Jeff built this webpage too long ago and the icon is no longer available on its original source….

```
 41 13.853875245  172.16.234.128   45.79.89.123     HTTP   364 GET /favicon.ico HTTP/1.1
 42 13.854191817  45.79.89.123     172.16.234.128   TCP     60 80 → 49436 [ACK] Seq=808 Ack=105
 43 13.905927388  45.79.89.123     172.16.234.128   HTTP   383 HTTP/1.1 404 Not Found  (text/ht
```

Question 02:

We know where the HTML for the cancel webpage comes from, but where does the HTML for the pop-up window come from to enter a password? Is it built in the browser?



3: TCP Keep Alive (server port 80):

In the middle of the session, the Client also sends the server a TCP Keep-Alive message just to check if the server is still listening (alive). The server responds with acknowledgement, 'I'm still here' [TCP Keep-Alive ACK].

```
172.16.234.128   45.79.89.123     TCP   54 [TCP Keep-Alive] 49436 → 80 [ACK] Seq=350 Ack=404
172.16.234.128   45.79.89.123     TCP   54 [TCP Keep-Alive] 49436 → 80 [ACK] Seq=350 Ack=404
45.79.89.123     172.16.234.128   TCP   60 [TCP Keep-Alive ACK] 80 → 49436 [ACK] Seq=404 Ack
```