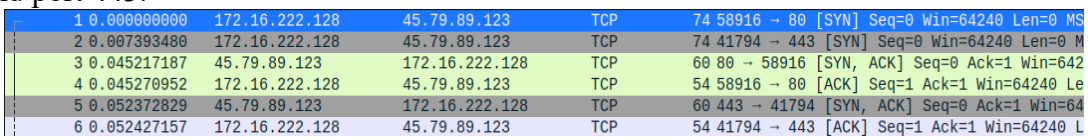
John Woods

A Basic Auth Story

The process begins with the TCP handshake: my machine sending SYN packets to the server, and the server responding with SYN ACK, with the handshake being completed by my machine sending an ACK. Notably, my machine appears to connect to two separate ports, port 80 and port 443.



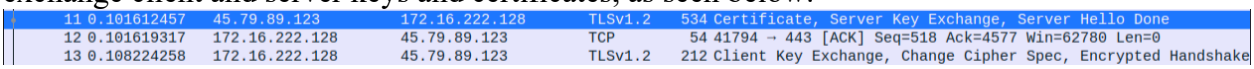
1	0.000000000	172.16.222.128	45.79.89.123	TCP	74 58916 → 80 [SYN] Seq=0 Win=64240 Len=0 MS
2	0.007393480	172.16.222.128	45.79.89.123	TCP	74 41794 → 443 [SYN] Seq=0 Win=64240 Len=0 M
3	0.045217187	45.79.89.123	172.16.222.128	TCP	60 80 → 58916 [SYN, ACK] Seq=0 Ack=1 Win=642
4	0.045270952	172.16.222.128	45.79.89.123	TCP	54 58916 → 80 [ACK] Seq=1 Ack=1 Win=64240 Le
5	0.052372829	45.79.89.123	172.16.222.128	TCP	60 443 → 41794 [SYN, ACK] Seq=0 Ack=1 Win=64
6	0.052427157	172.16.222.128	45.79.89.123	TCP	54 41794 → 443 [ACK] Seq=1 Ack=1 Win=64240 L

Port 80 is used for general plaintext data transmission, whereas port 443 allows secure network data transmission (https://www.ssl2buy.com/wiki/port-80-http-vs-port-443-https#:~:text=The%20main%20difference%20between%20Port,a%20non%2DHTTPS%20web%20page)

Next, my machine sends a “Client Hello,” which contains the address cs338.jeffondich.com. The hello message includes the TLS version I’m running, as well as some other information (https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/)

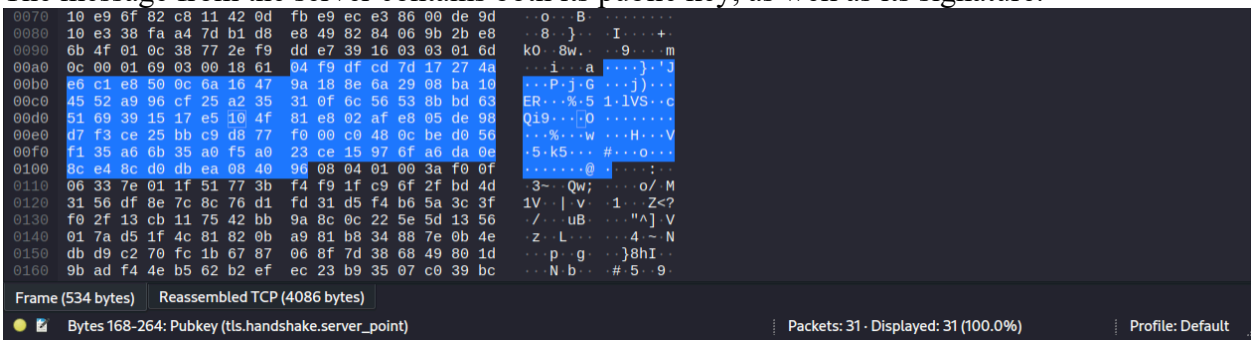
This Hello is acknowledged by the server, and it responds with a Server Hello, which contains the phrase “let’s encrypt!” a few times, alongside the address accioapps.com, which (I believe) is the website for an app of Jeff’s given that the copyright points to Ultralingua Inc.

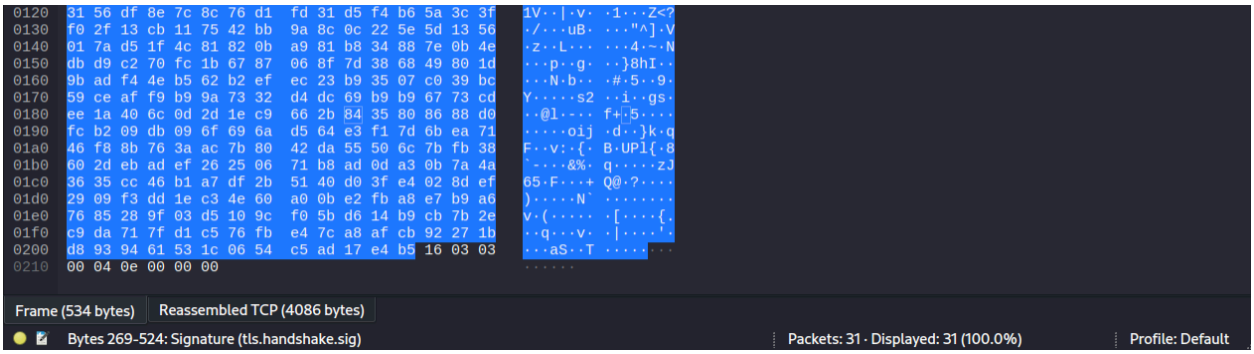
The server hello also establishes the certificate of the server. Additionally, the client and server exchange client and server keys and certificates, as seen below:



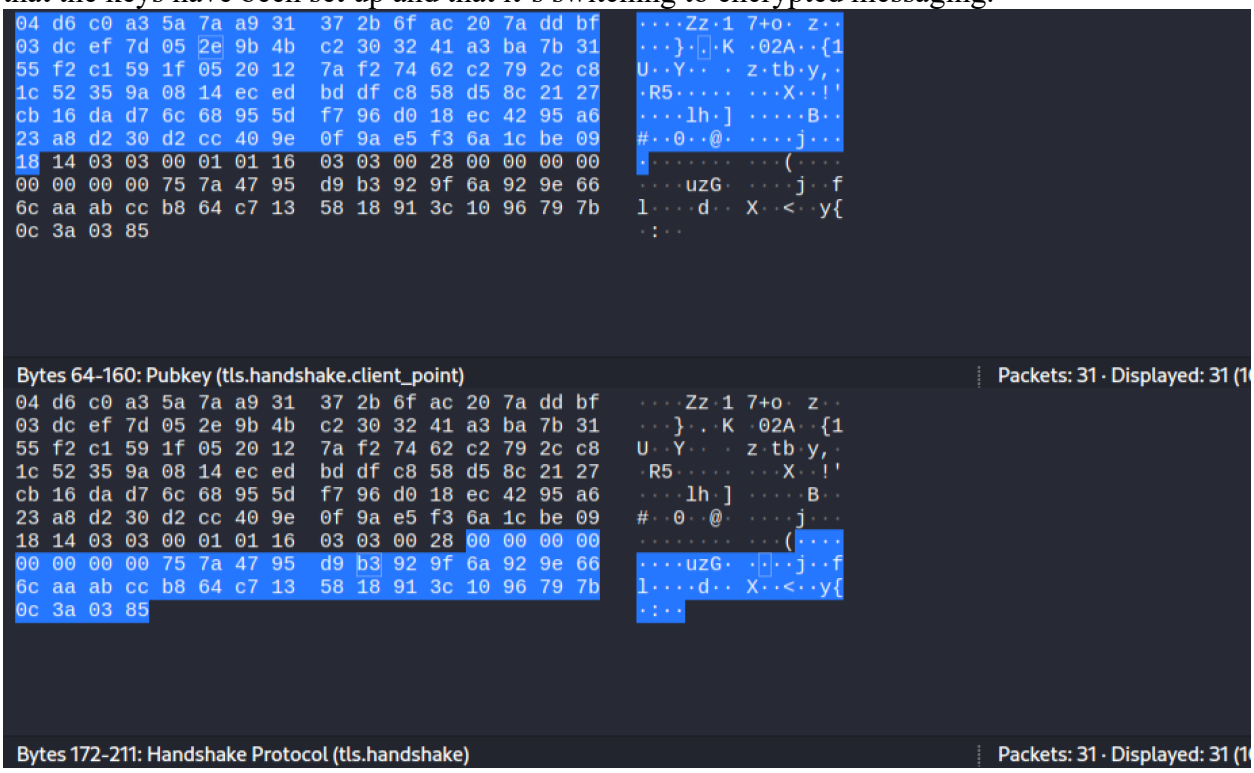
11	0.101612457	45.79.89.123	172.16.222.128	TLSv1.2	534 Certificate, Server Key Exchange, Server Hello Done
12	0.101619317	172.16.222.128	45.79.89.123	TCP	54 41794 → 443 [ACK] Seq=518 Ack=4577 Win=62780 Len=0
13	0.108224258	172.16.222.128	45.79.89.123	TLSv1.2	212 Client Key Exchange, Change Cipher Spec, Encrypted Handshake

The message from the server contains both its public key, as well as its signature.

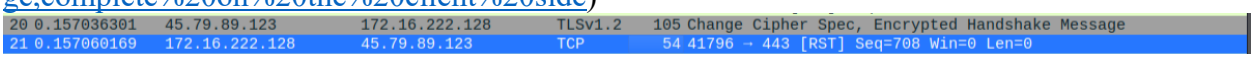




Meanwhile, the client key message only contains its pubkey, as well as its handshake protocol, and the message is marked as a Change Cypher Spec message, which is letting the server know that the keys have been set up and that it's switching to encrypted messaging.



Notably, these messages have occurred over port 443, since security is necessary until this point. Additionally, the server sends a Change Cypher Spec message, of its own. This is followed by the RST packet from the client, letting the server know that it's done using port 443, and since the encryption keys have been set up, they can both switch to using port 80. This message also contains data that lets both parties ensure the handshake hasn't been tampered with. (https://www.thesslstore.com/blog/explaining-ssl-handshake/#:~:text=The%20%E2%80%9CChange%20Cipher%20Spec%E2%80%9D%20message,complete%20on%20the%20client%20side)



From there on, all messages seem to be encrypted, and only use port 80.

```
21 0.157060169   172.16.222.128   45.79.89.123     TCP   54 41796 → 443 [RST] Seq=708 Win=0 Len=0
22 0.158021041   45.79.89.123     172.16.222.128   TCP   60 80 → 58924 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
23 0.158057114   172.16.222.128   45.79.89.123     TCP   54 58924 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
24 5.058202067   172.16.222.128   45.79.89.123     TCP   54 58920 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
25 5.058483310   45.79.89.123     172.16.222.128   TCP   60 80 → 58920 [ACK] Seq=1 Ack=2 Win=64239 Len=0
26 5.103099675   45.79.89.123     172.16.222.128   TCP   60 80 → 58920 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
27 5.103118925   172.16.222.128   45.79.89.123     TCP   54 58920 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
28 6.058513834   172.16.222.128   45.79.89.123     TCP   54 58924 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
29 6.058878331   45.79.89.123     172.16.222.128   TCP   60 80 → 58924 [ACK] Seq=1 Ack=2 Win=64239 Len=0
30 6.103679581   45.79.89.123     172.16.222.128   TCP   60 80 → 58924 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
31 6.103707502   172.16.222.128   45.79.89.123     TCP   54 58924 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0

rnet Protocol Version 4, Src: 172.16.222.128, Dst: 45.79.89.123

00 50 56 f2 f3 24 00 0c  29 38 74 c0 08 00 45 00   ·PV··$·· )8t···E·
00 28 00 00 40 00 40 06  29 75 ac 10 de 80 2d 4f   ·(··@·@· )u····-O
59 7b e6 2c 00 50 ca 86  16 4c 17 1d c1 d9 50 10   Y{·,·P·· ·L····P·
fa f0 03 42 00 00                                  ···B··
```

Finally, my browser requests the basicauth folder, and is denied with a 401 unauthorized error, and we begin the authorization process.

```
24 0.161333164   172.16.64.129   45.79.89.123     HTTP   403 GET /basicauth/ HTTP/1.1
25 0.161600876   45.79.89.123     172.16.64.129    TCP     60 80 → 46306 [ACK] Seq=1 Ack=350 Win=64240 Len=0
26 0.207128663   45.79.89.123     172.16.64.129    HTTP   457 HTTP/1.1 401 Unauthorized  (text/html)
```

Note that the auth error also contains the WWW-Authenticate header which informs my browser about what authorization schemes will be accepted (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization).

```
65 65 70 2d 61 6c 69 76  65 0d 0a 57 57 57 2d 41   eep-aliv e··WWW-A
75 74 68 65 6e 74 69 63  61 74 65 3a 20 42 61 73   uthentic ate: Bas
69 63 20 72 65 61 6c 6d  3d 22 50 72 6f 74 65 63   ic realm ="Protec
74 65 64 20 41 72 65 61  22 0d 0a 0d 0a 3c 68 74   ted Area "····<ht
6d 6c 3e 0d 0a 3c 68 65  61 64 3e 3c 74 69 74 6c   ml>··<he ad><titl
65 3e 34 30 31 20 41 75  74 68 6f 72 69 7a 61 74   e>401 Au thorizat
69 6f 6e 20 52 65 71 75  69 72 65 64 3c 2f 74 69   ion Requ ired</ti
74 6c 65 3e 3c 2f 68 65  61 64 3e 0d 0a 3c 62 6f   tle></he ad>··<bo
64 79 3e 0d 0a 3c 63 65  6e 74 65 72 3e 3c 68 31   dy>··<ce nter><h1
3e 34 30 31 20 41 75 74  68 6f 72 69 7a 61 74 69   >401 Aut horizati
6f 6e 20 52 65 71 75 69  72 65 64 3c 2f 68 31 3e   on Requi red</h1>
3c 2f 63 65 6e 74 65 72  3e 0d 0a 3c 68 72 3e 3c   </center >··<hr><
63 65 6e 74 65 72 3e 6e  67 69 6e 78 2f 31 2e 31   center>n ginx/1.1

Bytes 219-266: WWW-Authenticate (http.www_authenticate)   Packets: 74 · Displayed: 74 (100.0%) · Marked: 4 (5.4%)   Profile: Default
```

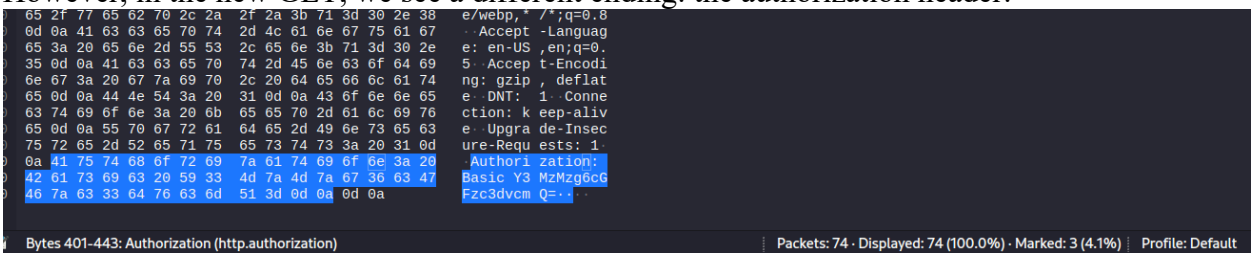
Next, I input the extremely secure username and password that the server requested, and we start the “second phase” of the connection. Once again, we see the TCP handshake, and my browser requests the folder again.

```
48 112.281775872 172.16.64.129   45.79.89.123     TCP    74 46308 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3575…
49 112.327170023 45.79.89.123    172.16.64.129    TCP    60 80 → 46308 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
50 112.327221916 172.16.64.129   45.79.89.123     TCP    54 46308 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
51 112.327571178 172.16.64.129   45.79.89.123     HTTP  446 GET /basicauth/ HTTP/1.1
```

This GET, however, is different from the last. In the screenshot below, notice that the GET request from before I entered my credentials ends with the request line.

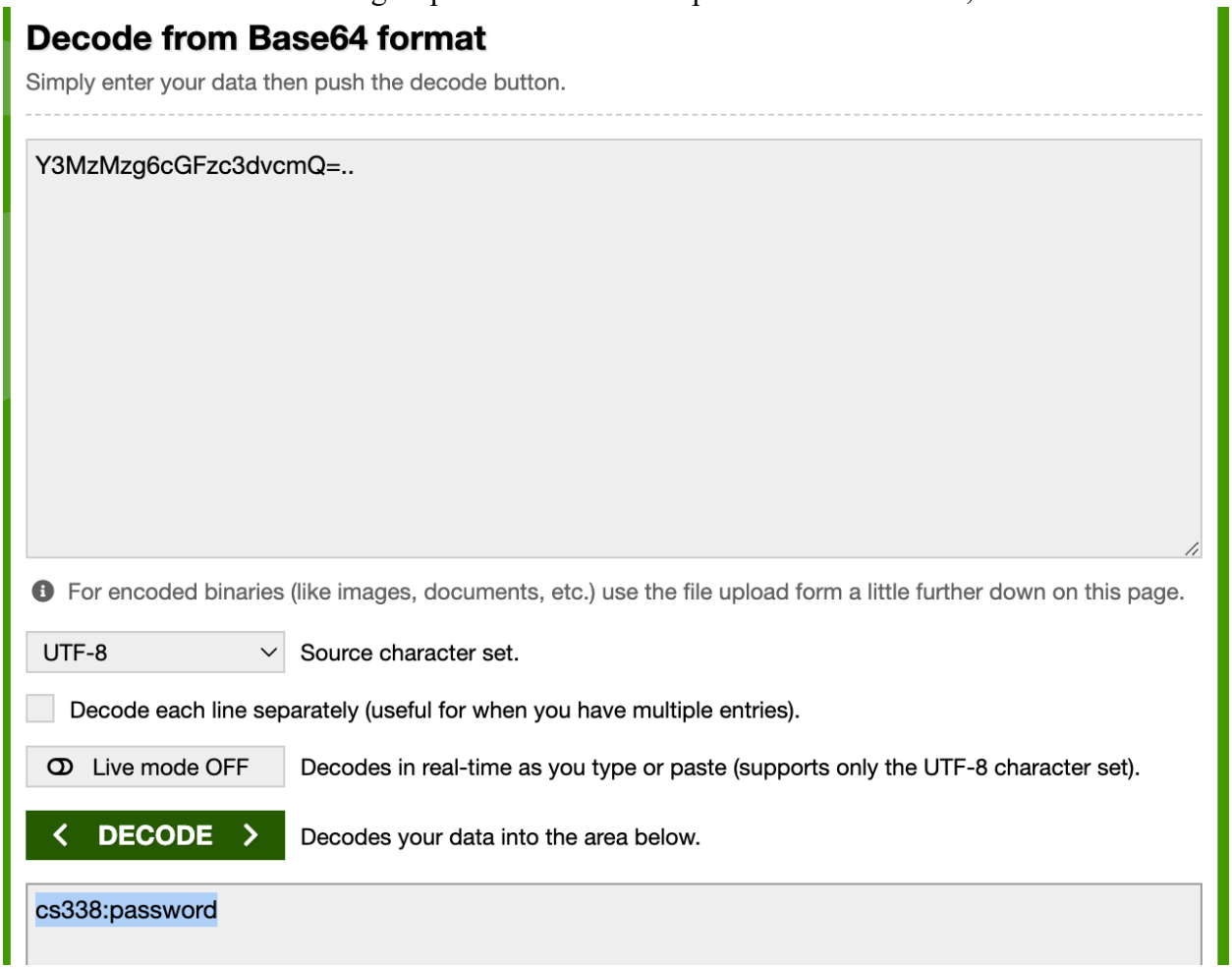
```
6c 2b 78 6d 6c 2c 61 70  70 6c 69 63 61 74 69 6f   l+xml,ap plicatio
6e 2f 78 6d 6c 3b 71 3d  30 2e 39 2c 69 6d 61 67   n/xml;q= 0.9,imag
65 2f 77 65 62 70 2c 2a  2f 2a 3b 71 3d 30 2e 38   e/webp,* /*;q=0.8
0d 0a 41 63 63 65 70 74  2d 4c 61 6e 67 75 61 67   ··Accept -Languag
65 3a 20 65 6e 2d 55 53  2c 65 6e 3b 71 3d 30 2e   e: en-US ,en;q=0.
35 0d 0a 41 63 63 65 70  74 2d 45 6e 63 6f 64 69   5··Accep t-Encodi
6e 67 3a 20 67 7a 69 70  2c 20 64 65 66 6c 61 74   ng: gzip , deflat
65 0d 0a 44 4e 54 3a 20  31 0d 0a 43 6f 6e 6e 65   e··DNT:  1··Conne
63 74 69 6f 6e 3a 20 6b  65 65 70 2d 61 6c 69 76   ction: k eep-aliv
65 0d 0a 55 70 67 72 61  64 65 2d 49 6e 73 65 63   e··Upgra de-Insec
75 72 65 2d 52 65 71 75  65 73 74 73 3a 20 31 0d   ure-Requ ests: 1·
0a 0d 0a                                           ···

Request line (http.request.line), 30 bytes   Packets: 74 · Displayed: 74 (100.0%) · Marked: 3 (4.1%)   Profile: Default
```

However, in the new GET, we see a different ending: the authorization header:



This authorization header not only contains the information about what authorization type my browser is choosing to use, it also contains the password and username in plain text (!!!). Although it's been converted into base64, the information is all available, so therefore using HTTPS or TLS is essentially mandatory when using basic authentication (https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication#basic_authentication_scheme).

To prove that this is truly the username and password in plaintext, we can simply take the string that is sent following the "Basic" declaration, (Y3MzMzg6cGFzc3dvcmQ=..) and input it into a base64 conversion tool. Doing so provides us with the promised information, as seen below:

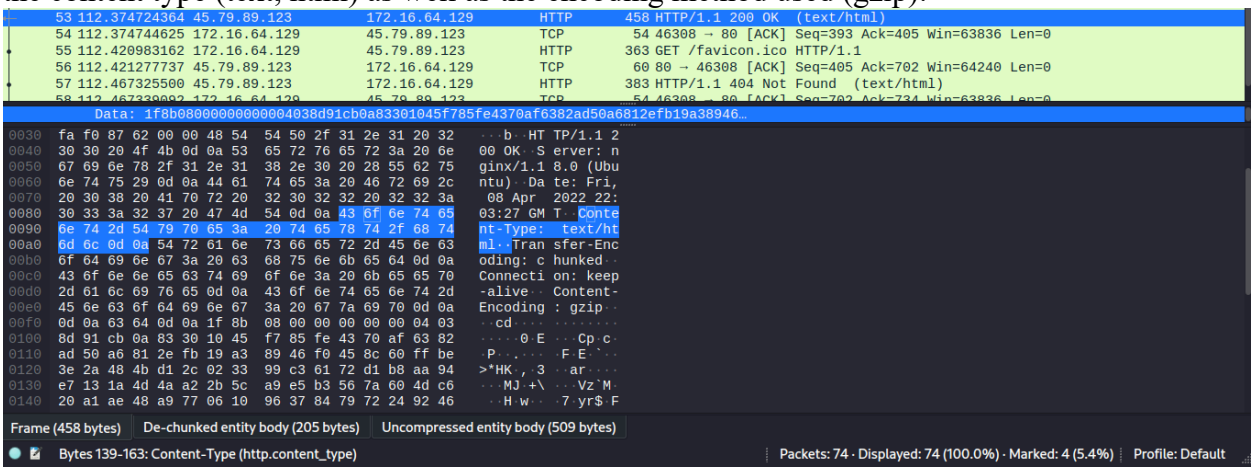


(courtesy of base64decode.org)

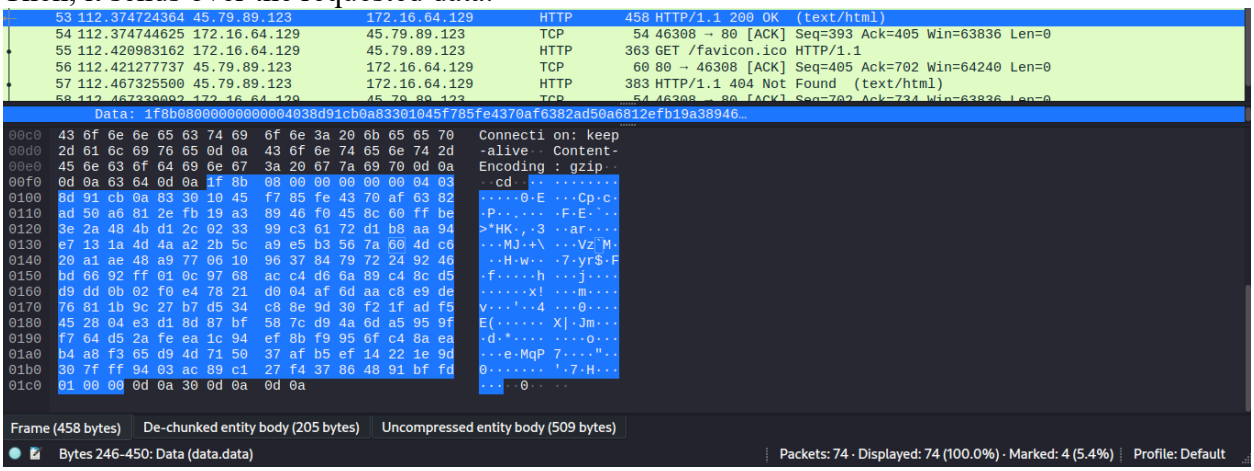
Notice that the username and password are concatenated with a colon, as specified by the HTTP page on the Basic authentication scheme (https://datatracker.ietf.org/doc/html/rfc7617#section-2). The scheme page also notes that users are known to use "weak passwords," but thankfully we've avoided this pitfall.

```
Servers and proxies implementing Basic authentication need to store
user passwords in some form in order to authenticate a request.
These passwords ought to be stored in such a way that a leak of the
password data doesn't make them trivially recoverable.  This is
especially important when users are allowed to set their own
passwords, since users are known to choose weak passwords and to
reuse them across authentication realms.  While a full discussion of
```

The server acknowledges this request. Then, after the server verifies the credentials, it sends back a 200 OK message, and this contains the requested information. The message first contains the content type (text, html) as well as the encoding method used (gzip):



Then, it sends over the requested data:



After this is done, my machine ACKs the message, and also requests the favicon for the page, though the server is apparently unable to provide and sends along a 404 not found message, alongside an ACK of the request.

```
    51 112.327571178 172.16.64.129        45.79.89.123         HTTP      446 GET /basicauth/ HTTP/1.1
    52 112.327983533 45.79.89.123         172.16.64.129        TCP        60 80 → 46308 [ACK] Seq=1 Ack=393 Win=64240 Len=0
    53 112.374724364 45.79.89.123         172.16.64.129        HTTP      458 HTTP/1.1 200 OK  (text/html)
    54 112.374744625 172.16.64.129        45.79.89.123         TCP        54 46308 → 80 [ACK] Seq=393 Ack=405 Win=63836 Len=0
    55 112.420983162 172.16.64.129        45.79.89.123         HTTP      363 GET /favicon.ico HTTP/1.1
    56 112.421277737 45.79.89.123         172.16.64.129        TCP        60 80 → 46308 [ACK] Seq=405 Ack=702 Win=64240 Len=0
    57 112.467325500 45.79.89.123         172.16.64.129        HTTP      383 HTTP/1.1 404 Not Found  (text/html)
```

My client ACKs this error, and that's essentially it! If we request any of the text files in the folder, we see a very similar request that's authenticated in the same way, with the authorization header at the bottom.

```
▸ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
00a0  20 47 65 63 6b 6f 2f 32  30 31 30 30 31 30 31 20   Gecko/2 0100101
00b0  46 69 72 65 66 6f 78 2f  39 31 2e 30 0d 0a 41 63  Firefox/ 91.0··Ac
00c0  63 65 70 74 3a 20 74 65  78 74 2f 68 74 6d 6c 2c  cept: te xt/html,
00d0  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 68 74 6d  applicat ion/xhtm
00e0  6c 2b 78 6d 6c 2c 61 70  70 6c 69 63 61 74 69 6f  l+xml,ap plicatio
00f0  6e 2f 78 6d 6c 3b 71 3d  30 2e 39 2c 69 6d 61 67  n/xml;q= 0.9,imag
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 41 75 74 68 6f  e··DNT:  1··Autho
0160  72 69 7a 61 74 69 6f 6e  3a 20 42 61 73 69 63 20  rization : Basic
0170  59 33 4d 7a 4d 7a 67 36  63 47 46 7a 63 33 64 76  Y3MzMzg6 cGFzc3dv
0180  63 6d 51 3d 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e  cmQ=··Co nnection
0190  3a 20 6b 65 65 70 2d 61  6c 69 76 65 0d 0a 55 70  : keep-a live··Up
01a0  67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52  grade-In secure-R
01b0  65 71 75 65 73 74 73 3a  20 31 0d 0a 0d 0a        equests:  1····

HTTP Authorization header (http.authorization), 43 bytes   Packets: 147 · Displayed: 147 (100.0%) · Marked: 5 (3.4%)   Profile: Default
```

For the text files, the server simply responds with the direct cleartext contents:

```
▸ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
00a0  20 47 65 63 6b 6f 2f 32  30 31 30 30 31 30 31 20   Gecko/2 0100101
00b0  46 69 72 65 66 6f 78 2f  39 31 2e 30 0d 0a 41 63  Firefox/ 91.0··Ac
00c0  63 65 70 74 3a 20 74 65  78 74 2f 68 74 6d 6c 2c  cept: te xt/html,
00d0  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 68 74 6d  applicat ion/xhtm
00e0  6c 2b 78 6d 6c 2c 61 70  70 6c 69 63 61 74 69 6f  l+xml,ap plicatio
00f0  6e 2f 78 6d 6c 3b 71 3d  30 2e 39 2c 69 6d 61 67  n/xml;q= 0.9,imag
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 41 75 74 68 6f  e··DNT:  1··Autho
0160  72 69 7a 61 74 69 6f 6e  3a 20 42 61 73 69 63 20  rization : Basic
0170  59 33 4d 7a 4d 7a 67 36  63 47 46 7a 63 33 64 76  Y3MzMzg6 cGFzc3dv
0180  63 6d 51 3d 0d 0a 43 6f  6e 6e 65 63 74 69 6f 6e  cmQ=··Co nnection
0190  3a 20 6b 65 65 70 2d 61  6c 69 76 65 0d 0a 55 70  : keep-a live··Up
01a0  67 72 61 64 65 2d 49 6e  73 65 63 75 72 65 2d 52  grade-In secure-R
01b0  65 71 75 65 73 74 73 3a  20 31 0d 0a 0d 0a        equests:  1····

HTTP Authorization header (http.authorization), 43 bytes   Packets: 147 · Displayed: 147 (100.0%) · Marked: 5 (3.4%)   Profile: Default
```

From there on, the only messages are keepalive exchanges, which occur until the server eventually loses its patience and sends along a FIN, to which my client responds with a FIN ACK, responded to once more with the server's ACK.

```
    63 133.615469055 45.79.89.123         172.16.64.129        TCP        60 [TCP Keep-Alive ACK] 80 → 46308 [ACK] Seq=734 Ack=702 Win=64240 Len=0
    64 143.854483096 172.16.64.129        45.79.89.123         TCP        54 [TCP Keep-Alive] 46308 → 80 [ACK] Seq=701 Ack=734 Win=63836 Len=0
    65 143.854901772 45.79.89.123         172.16.64.129        TCP        60 [TCP Keep-Alive ACK] 80 → 46308 [ACK] Seq=734 Ack=702 Win=64240 Len=0
    66 154.093849876 172.16.64.129        45.79.89.123         TCP        54 [TCP Keep-Alive] 46308 → 80 [ACK] Seq=701 Ack=734 Win=63836 Len=0
    67 154.094118027 45.79.89.123         172.16.64.129        TCP        60 [TCP Keep-Alive ACK] 80 → 46308 [ACK] Seq=734 Ack=702 Win=64240 Len=0
    68 164.334550711 172.16.64.129        45.79.89.123         TCP        54 [TCP Keep-Alive] 46308 → 80 [ACK] Seq=701 Ack=734 Win=63836 Len=0
    69 164.334951565 45.79.89.123         172.16.64.129        TCP        60 [TCP Keep-Alive ACK] 80 → 46308 [ACK] Seq=734 Ack=702 Win=64240 Len=0
    70 174.573921310 172.16.64.129        45.79.89.123         TCP        54 [TCP Keep-Alive] 46308 → 80 [ACK] Seq=701 Ack=734 Win=63836 Len=0
    71 174.574166991 45.79.89.123         172.16.64.129        TCP        60 [TCP Keep-Alive ACK] 80 → 46308 [ACK] Seq=734 Ack=702 Win=64240 Len=0
    72 177.455969281 45.79.89.123         172.16.64.129        TCP        60 80 → 46308 [FIN, PSH, ACK] Seq=734 Ack=702 Win=64240 Len=0
    73 177.456254858 172.16.64.129        45.79.89.123         TCP        54 46308 → 80 [FIN, ACK] Seq=702 Ack=735 Win=63836 Len=0
    74 177.456662759 45.79.89.123         172.16.64.129        TCP        60 80 → 46308 [ACK] Seq=735 Ack=703 Win=64239 Len=0
```

And so, the client and server lived happily ever after ☺