

Ruben Boero and Vanessa Heynes

What queries are sent from the browser, and what responses does it receive?

Browser sends a query to the server, asking for the /basicauth/ page:

```
40      0.242953295      172.16.15.128      45.79.89.123      HTTP      417
GET /basicauth/ HTTP/1.1
```

Note, in this query, the client does not include the WWW-Authenticate header in the HTTP request.

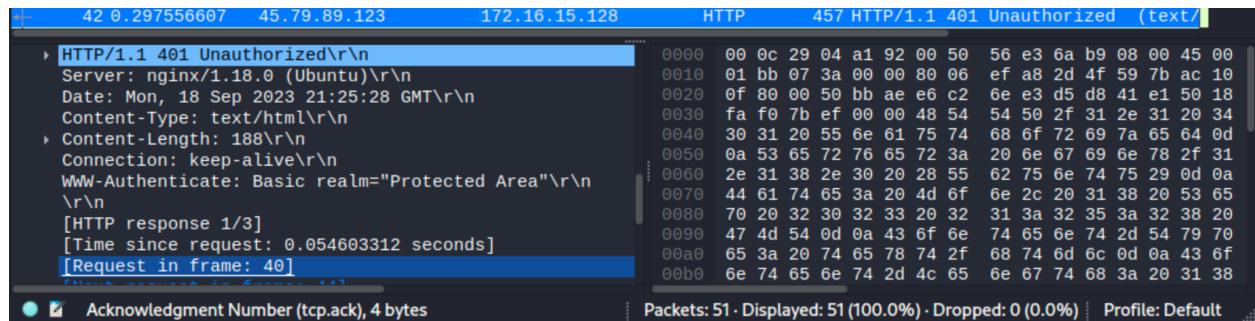
In response, server acknowledges the client's request:

```
41      0.243207591      45.79.89.123      172.16.15.128      TCP      60      80
→ 48046 [ACK] Seq=1 Ack=364 Win=64240 Len=0
```

But refuses to send the page without authorizing the user first:

```
42      0.297556607      45.79.89.123      172.16.15.128      HTTP      457
HTTP/1.1 401 Unauthorized (text/html)
```

The 401 status code is how nginx communicates to the client that they have been denied authorization.



Here, the server is saying that the requested page is protected by Basic HTTP security

After the password is typed by the user, what sequence of queries and responses do you see?

The password is sent by the client to the server:

```
44      7.247785473      172.16.15.128      45.79.89.123      HTTP      460
GET /basicauth/ HTTP/1.1
```

No.	Time	Source	Destination	Protocol	Length	Info
37	0.230541393	172.16.15.128	45.79.89.123	TCP	54	39794 → 443 [ACK] Seq=1066 Ack=52
38	0.242740167	45.79.89.123	172.16.15.128	TCP	60	80 → 48046 [SYN, ACK] Seq=0 Ack=1
39	0.242765001	172.16.15.128	45.79.89.123	TCP	54	48046 → 80 [ACK] Seq=1 Ack=1 Win=
40	0.242953295	172.16.15.128	45.79.89.123	HTTP	417	GET /basicauth/ HTTP/1.1
41	0.243207591	45.79.89.123	172.16.15.128	TCP	60	80 → 48046 [ACK] Seq=1 Ack=364 Wi
42	0.297556607	45.79.89.123	172.16.15.128	HTTP	457	HTTP/1.1 401 Unauthorized (text/
43	0.297630983	172.16.15.128	45.79.89.123	TCP	54	48046 → 80 [ACK] Seq=364 Ack=404
44	7.247785473	172.16.15.128	45.79.89.123	HTTP	460	GET /basicauth/ HTTP/1.1
45	7.248124978	45.79.89.123	172.16.15.128	TCP	60	80 → 48046 [ACK] Seq=404 Ack=770
46	7.309787275	45.79.89.123	172.16.15.128	HTTP	458	HTTP/1.1 200 OK (text/html)

```

    ▶ GET /basicauth/ HTTP/1.1\r\n
    Host: cs338.jeffondich.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0)
    Accept: text/html,application/xhtml+xml,application/xml
    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
    ▶ Authorization: Basic Y3MzMg6cGFzc3dvcmQ=\r\n
      Credentials: cs338:password
    ```

```

```

0000 00 50 56 e3 6a b9 00 0c 29 04 a1 92 08 00 45 00
0010 01 b2 37 40 00 40 06 64 a8 ac 10 0f 80 2d 4f
0020 59 7b bb ae 00 50 d5 d8 41 e1 e6 c2 70 76 50 18
0030 f9 5d 44 00 00 47 45 54 20 2f 62 61 73 69 63
0040 61 75 74 68 2f 20 48 54 54 50 2f 31 2e 31 0d 0a
0050 48 6f 73 74 3a 20 63 73 33 33 38 2e 6a 65 66 66
0060 6f 6e 64 69 63 68 2e 63 6f 6d 0d 0a 55 73 65 72
0070 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f
0080 35 2e 30 20 28 58 31 31 3b 20 4c 69 6e 75 78 20
0090 61 61 72 63 68 36 34 3b 20 72 76 3a 31 30 39 2e
00a0 30 29 20 47 65 63 6b 6f 2f 32 30 31 30 30 31 30
00b0 31 20 46 69 72 65 66 6f 78 2f 31 31 35 2e 30 0d

```

Note, in this HTTP request, the client includes the necessary WWW-Authenticate header.

The server acknowledges that the password has been received:

```
45 7.248124978 45.79.89.123 172.16.15.128 TCP 60 80
→ 48046 [ACK] Seq=404 Ack=770 Win=64240 Len=0
```

And that the password is correct:

```
46 7.309787275 45.79.89.123 172.16.15.128 HTTP 458
HTTP/1.1 200 OK (text/html)
```

No.	Time	Source	Destination	Protocol	Length	Info
46	7.309787275	45.79.89.123	172.16.15.128	HTTP	458	HTTP/1.1 200 OK (text/html)
47	7.309816067	172.16.15.128	45.79.89.123	TCP	54	48046 → 80 [ACK] Seq=770 Ack=808
48	7.365140723	172.16.15.128	45.79.89.123	HTTP	377	GET /favicon.ico HTTP/1.1

```

 ▶ Frame 46: 458 bytes on wire (3664 bits), 458 bytes captured
 ▶ Ethernet II, Src: VMware_e3:6a:b9 (00:50:56:e3:6a:b9), Dst: 172.16.15.128 (00:0c:29:04:a1:92)
 ▶ Internet Protocol Version 4, Src: 45.79.89.123, Dst: 172.16.15.128
 ▶ Transmission Control Protocol, Src Port: 80, Dst Port: 445
 ▶ Hypertext Transfer Protocol
 ▶ HTTP/1.1 200 OK\r\n
 Server: nginx/1.18.0 (Ubuntu)\r\n

```

```

0000 00 0c 29 04 a1 92 00 50 56 e3 6a b9 08 00 45 00
0010 01 bc 07 3e 00 00 80 06 ef a3 2d 4f 59 7b ac 10
0020 0f 80 00 50 bb ae e6 c2 70 76 d5 d8 43 77 50 18
0030 fa f0 5c f1 00 00 48 54 54 50 2f 31 2e 31 20 32
0040 30 30 20 4f 4b 0d 0a 53 65 72 76 65 72 3a 20 6e
0050 67 69 6e 78 2f 31 2e 31 38 2e 30 20 28 55 62 75
0060 6e 74 75 29 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c

```

Notice that the header contains "OK", rather than "Unauthorized" like in packet 42 shown above.

The server sends the previously requested /basicauth/ HTML to the client:

No.	Time	Source	Destination	Protocol	Length	Info
46	7.309787275	45.79.89.123	172.16.15.128	HTTP	458	HTTP/1.1 200 OK (text/html)
47	7.309816067	172.16.15.128	45.79.89.123	TCP	54	48046 → 80 [ACK] Seq=770 Ack=808
48	7.365140723	172.16.15.128	45.79.89.123	HTTP	377	GET /favicon.ico HTTP/1.1
49	7.365501020	45.79.89.123	172.16.15.128	TCP	60	80 → 48046 [ACK] Seq=808 Ack=1093
50	7.419076691	45.79.89.123	172.16.15.128	HTTP	383	HTTP/1.1 404 Not Found (text/html)
51	7.419104649	172.16.15.128	45.79.89.123	TCP	54	48046 → 80 [ACK] Seq=1093 Ack=113

```

 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>..
 amateurs.txt
 armed-guards.txt
 dancing.txt
 </pre><hr></body>\r\n
 </html>\r\n

```

```

0000 00 0c 29 04 a1 92 00 50 56 e3 6a b9 08 00 45 00
0010 01 bc 07 3e 00 00 80 06 ef a3 2d 4f 59 7b ac 10
0020 0f 80 00 50 bb ae e6 c2 70 76 d5 d8 43 77 50 18
0030 fa f0 5c f1 00 00 48 54 54 50 2f 31 2e 31 20 32
0040 30 30 20 4f 4b 0d 0a 53 65 72 76 65 72 3a 20 6e
0050 67 69 6e 78 2f 31 2e 31 38 2e 30 20 28 55 62 75
0060 6e 74 75 29 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c
0070 20 31 38 20 53 65 70 20 32 30 32 33 20 32 31 3a
0080 32 35 3a 33 35 20 47 4d 54 0d 0a 43 6f 6e 74 65
0090 6e 74 2d 54 79 70 65 3a 20 74 65 78 74 2f 68 74

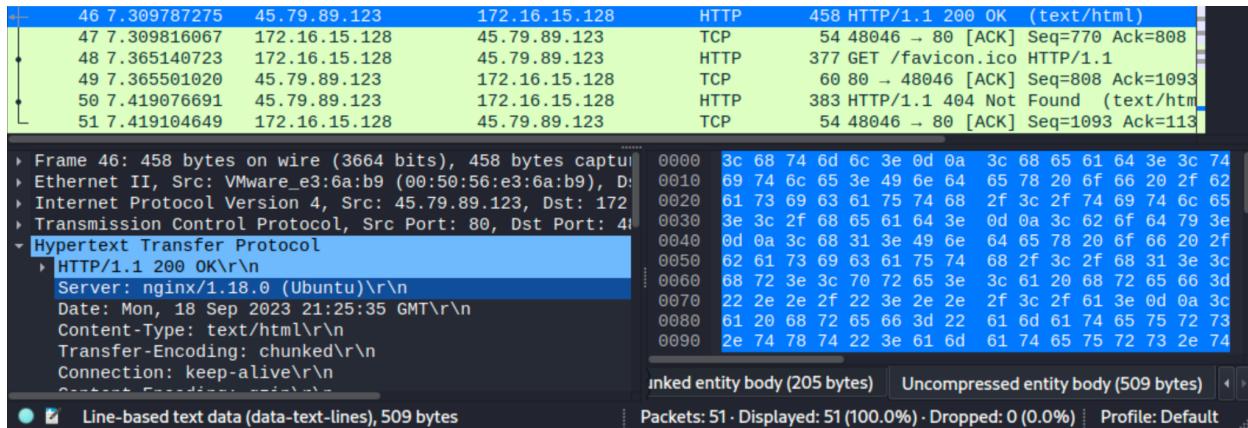
```

The completeness of the conversation capture (tcp.completeness) : 51 · Displayed: 51 (100.0%) · Dropped: 0 (0.0%) · Profile: Default

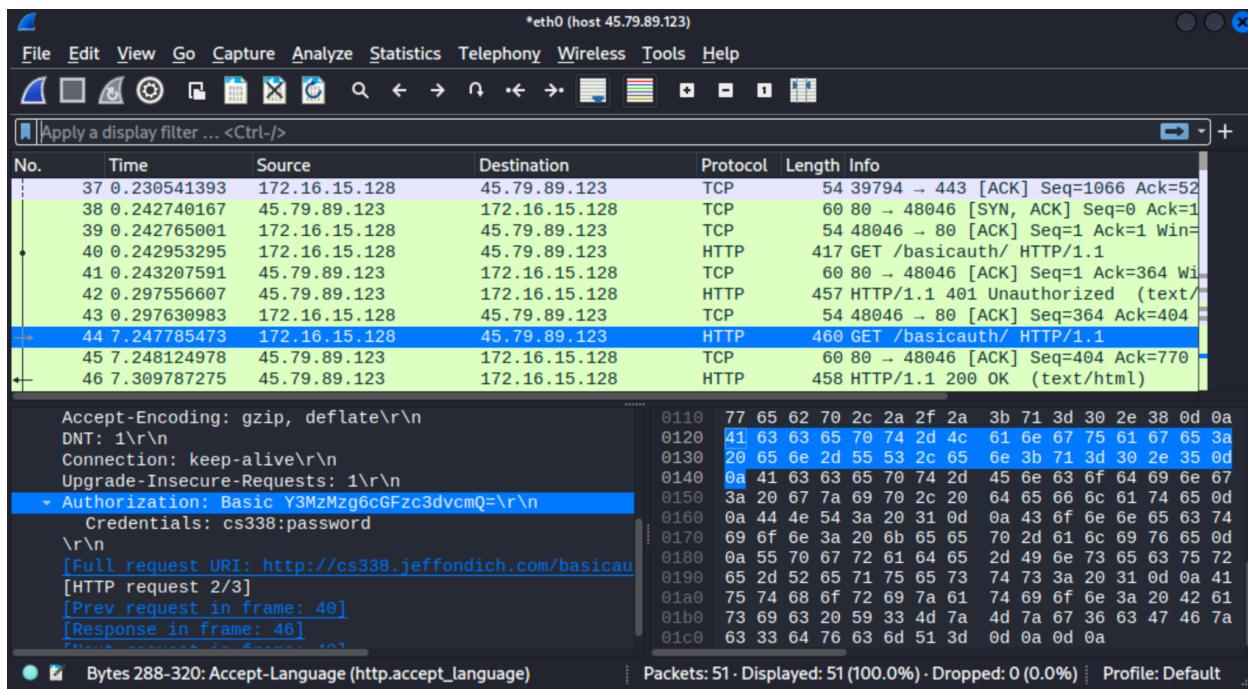
Finally, the server then sends the favicon of the page.

**Is the password sent by the browser to the server, or does the browser somehow do the password checking itself?**

The server—running nginx—is doing the checking:



We came to this conclusion because the OK message comes from the server, not from the client.



Screenshot shows the client sending the password to the server.

The password is sent by the client to the server in this packet:

```
44 7.247785473 172.16.15.128 45.79.89.123 HTTP 460
GET /basicauth/ HTTP/1.1
```

Additionally, we found some [nginx documentation/examples](#) that makes it seem like the checking is happening within the server.

We can see that the client must ask the user for the necessary username and password from [section 2 of the Basic Authentication section of the RFC](#). Because the client needs to gather the necessary information from the user, this suggests that the client is the one sending the authorizing information to the server.

### If the former, is the password sent in clear text or is it encrypted or something else?

The password is sent in clear text encoded with base64. The username and password are separated by a colon character:

```
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
 Credentials: cs338:password
```

### If it's encrypted, where did the encryption key come from?

It's not encrypted.

### How does what you observe via Wireshark connect to the relevant sections of the HTTP and HTTP Basic Authentication specification documents?

What we observed was directly connected to the section of the HTTP information under the 2.0 header.

- The username and password are sent unencrypted in plain text using base64 encoding.
- The username and password are sent as 1 string where the username and password are separated by a comma character.
- The HTTP headers were formatted in the same way as the documents describe.

Generally, the HTTP messages sent back and forth between the server and the client match the HTTP documentation.

### How is the password verified? By whom?

The password is verified by nginx from the server side.

Sources:

- <https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-subrequest-authentication/>
- <https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication/>

## Where is the authorized header?

The authorization header is found in packet 42:

Packet 42 details pane (selected):

- HTTP/1.1 401 Unauthorized
- Server: nginx/1.18.0 (Ubuntu)
- Date: Mon, 18 Sep 2023 21:25:28 GMT
- Content-Type: text/html
- Content-Length: 188
- Connection: keep-alive
- WWW-Authenticate: Basic realm="Protected Area"

Packet 42 bytes pane:

Hex	Dec	ASCII
0100	74 65 64 20 41 72 65 61 22 0d 0a 0d 0a 3c 68 74	T
0110	6d 6c 3e 0d 0a 3c 68 65 61 64 3e 3c 74 69 74 6c	d n . 0 a 3 c 6 8 6 5 6 1 6 4 3 e 3 c 7 4 6 9 7 4 6 c
0120	65 3e 34 30 31 20 41 75 74 68 6f 72 69 7a 61 74	5 . 3 4 3 0 3 1 2 0 4 1 7 5 7 4 6 8 6 f 7 2 6 9 7 a 6 1 7 4
0130	69 6f 6e 20 52 65 71 75 69 72 65 64 3c 2f 74 69	9 6 f 6 e 2 0 5 2 6 5 7 1 7 5 6 9 7 2 6 5 6 4 3 c 2 f 7 4 6 9
0140	74 6c 65 3e 3c 2f 68 65 61 64 3e 0d 0a 3c 62 6f	T 6 c 6 5 3 e 3 c 2 f 6 8 6 5 6 1 6 4 3 e 0 d 0 a 3 c 6 2 6 f
0150	64 79 3e 0d 0a 3c 63 65 6e 74 65 72 3e 3c 68 31	6 4 7 9 3 e 0 d 0 a 3 c 6 3 6 5 6 e 7 4 6 5 7 2 3 e 3 c 6 8 3 1
0160	3e 34 30 31 20 41 75 74 68 6f 72 69 7a 61 74 69	3 e 3 4 3 0 3 1 2 0 4 1 7 5 7 4 6 8 6 f 7 2 6 9 7 a 6 1 7 4 6 9
0170	6f 6e 20 52 65 71 75 69 72 65 64 3c 2f 68 31 3e	6 f 6 e 2 0 5 2 6 5 7 1 7 5 6 9 7 2 6 5 6 4 3 c 2 f 6 8 3 1 3 e
0180	3c 2f 63 65 6e 74 65 72 3e 0d 0a 3c 68 72 3e 3c	3 c 2 f 6 3 6 5 6 e 7 4 6 5 7 2 3 e 0 d 0 a 3 c 6 8 7 2 3 e 3 c
0190	63 65 6e 74 65 72 3e 6e 67 69 6e 78 2f 31 2e 31	6 3 6 5 6 e 7 4 6 5 7 2 3 e 6 e 6 7 6 9 6 e 7 8 2 f 3 1 2 e 3 1
01a0	38 2e 30 20 28 55 62 75 6e 74 75 29 3c 2f 63 65	3 8 2 e 3 0 2 0 2 8 5 5 6 2 7 5 6 e 7 4 7 5 2 9 3 c 2 f 6 3 6 5
01b0	6e 74 65 72 3e 0d 0a 3c 2f 62 6f 64 79 3e 0d 0a	6 e 7 4 6 5 7 2 3 e 0 d 0 a 3 c 2 f 6 2 6 f 6 4 7 9 3 e 0 d 0 a

Here we can see that the server sends an HTTP response to the client. In this packet the authorized header can be found. The information in this packet tells the client that they are not yet an authorized user (status 401), as well as providing the client with information about the type of authentication that is being used (Basic)

Source:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization>