

The screenshot shows a Wireshark packet capture of a network session between a client (172.16.127.128) and a server (45.79.89.123). The capture starts with a TCP handshake: a [SYN] request in frame 1, a [SYN, ACK] response in frame 2, a [FIN, ACK] request in frame 3, and a [FIN, ACK] response in frame 4. In frame 7, the client sends a GET request for /basicauth/. The server responds in frame 8 with a 200 OK status and a text/html body. In frame 9, the client sends another GET request for /basicauth/. The server responds in frame 10 with a 404 Not Found status and a text/html body. The capture continues with various other requests and responses, including a 403 Forbidden response in frame 11 and a 404 Not Found response in frame 12.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.127.128	45.79.89.123	TCP	74	36152 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3549338393 TSecr=0 WS=128
2	0.000073396	172.16.127.128	45.79.89.123	TCP	74	36154 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3549338393 TSecr=0 WS=128
3	0.044960260	45.79.89.123	172.16.127.128	TCP	60	80 → 36152 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4	0.044960523	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.04495676	172.16.127.128	45.79.89.123	TCP	54	36152 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.045045506	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	0.045074185	172.16.127.128	45.79.89.123	HTTP	385	GET /basicauth/ HTTP/1.1
8	0.045463660	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=1 Ack=342 Win=64240 Len=0
9	0.090956493	45.79.89.123	172.16.127.128	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
10	0.090973781	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=342 Ack=484 Win=63837 Len=0
11	0.046148439	172.16.127.128	45.79.89.123	TCP	54	36152 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
12	0.046645778	45.79.89.123	172.16.127.128	TCP	60	80 → 36152 [ACK] Seq=1 Ack=2 Win=64239 Len=0
13	0.091152781	45.79.89.123	172.16.127.128	TCP	60	80 → 36152 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
14	0.091174461	172.16.127.128	45.79.89.123	TCP	54	36152 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
15	0.071217615	172.16.127.128	45.79.89.123	HTTP	438	GET /basicauth/ HTTP/1.1
16	0.071565251	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=484 Ack=726 Win=64240 Len=0
17	0.117081291	45.79.89.123	172.16.127.128	HTTP	458	HTTP/1.1 200 OK (text/html)
18	0.117700225	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=726 Ack=808 Win=63837 Len=0
19	0.171485623	172.16.127.128	45.79.89.123	HTTP	355	GET /favicon.ico HTTP/1.1
20	0.172071488	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=808 Ack=1287 Win=64240 Len=0
21	0.217980977	45.79.89.123	172.16.127.128	HTTP	383	HTTP/1.1 404 Not Found (text/html)
22	0.218060575	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1027 Ack=1137 Win=63837 Len=0
23	0.237080533	172.16.127.128	45.79.89.123	HTTP	434	GET / HTTP/1.1
24	0.237398469	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=1137 Ack=1407 Win=64240 Len=0
25	0.403341819	45.79.89.123	172.16.127.128	HTTP	639	HTTP/1.1 200 OK (text/html)
26	0.403360807	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1407 Ack=1722 Win=63837 Len=0
27	0.429447618	172.16.127.128	45.79.89.123	HTTP	354	GET /jeff_square_head.jpg HTTP/1.1
28	0.429748934	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=1722 Ack=1707 Win=64240 Len=0
29	0.477175577	45.79.89.123	172.16.127.128	TCP	14534	80 → 36154 [PSH, ACK] Seq=1722 Ack=1707 Win=64240 Len=14480 [TCP segment of a reassembled PDU]
30	0.477222105	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1707 Ack=16282 Win=55480 Len=0
31	0.524267638	45.79.89.123	172.16.127.128	TCP	29014	80 → 36154 [PSH, ACK] Seq=16282 Ack=1707 Win=64240 Len=28960 [TCP segment of a reassembled PDU]
32	0.524420463	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1707 Ack=45162 Win=45260 Len=0
33	0.570835117	45.79.89.123	172.16.127.128	HTTP	36949	HTTP/1.1 200 OK (JPEG JFIF image)
34	0.570856623	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=1707 Ack=2057 Win=39420 Len=0
35	0.593333399	172.16.127.128	45.79.89.123	HTTP	409	GET /basicauth/amateurs.txt HTTP/1.1
36	0.59625952	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=2057 Ack=2152 Win=64240 Len=0
37	0.595796395	45.79.89.123	172.16.127.128	HTTP	375	HTTP/1.1 200 OK (text/plain)
38	0.595811912	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=2152 Ack=82378 Win=65535 Len=0
39	0.604147183	172.16.127.128	45.79.89.123	HTTP	583	GET /basicauth/armed-guards.txt HTTP/1.1
40	0.604166667	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=82378 Ack=2601 Win=64240 Len=0
41	0.748958818	45.79.89.123	172.16.127.128	HTTP	462	HTTP/1.1 200 OK (text/plain)
42	0.748612291	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=2601 Ack=82786 Win=65535 Len=0
43	0.594659512	172.16.127.128	45.79.89.123	HTTP	498	GET /basicauth/dancing.txt HTTP/1.1
44	0.594948955	45.79.89.123	172.16.127.128	TCP	60	80 → 36154 [ACK] Seq=82786 Ack=3845 Win=64240 Len=0
45	0.641195191	45.79.89.123	172.16.127.128	HTTP	528	HTTP/1.1 200 OK (text/plain)
46	0.64120777	172.16.127.128	45.79.89.123	TCP	54	36154 → 80 [ACK] Seq=3845 Ack=83260 Win=65535 Len=0

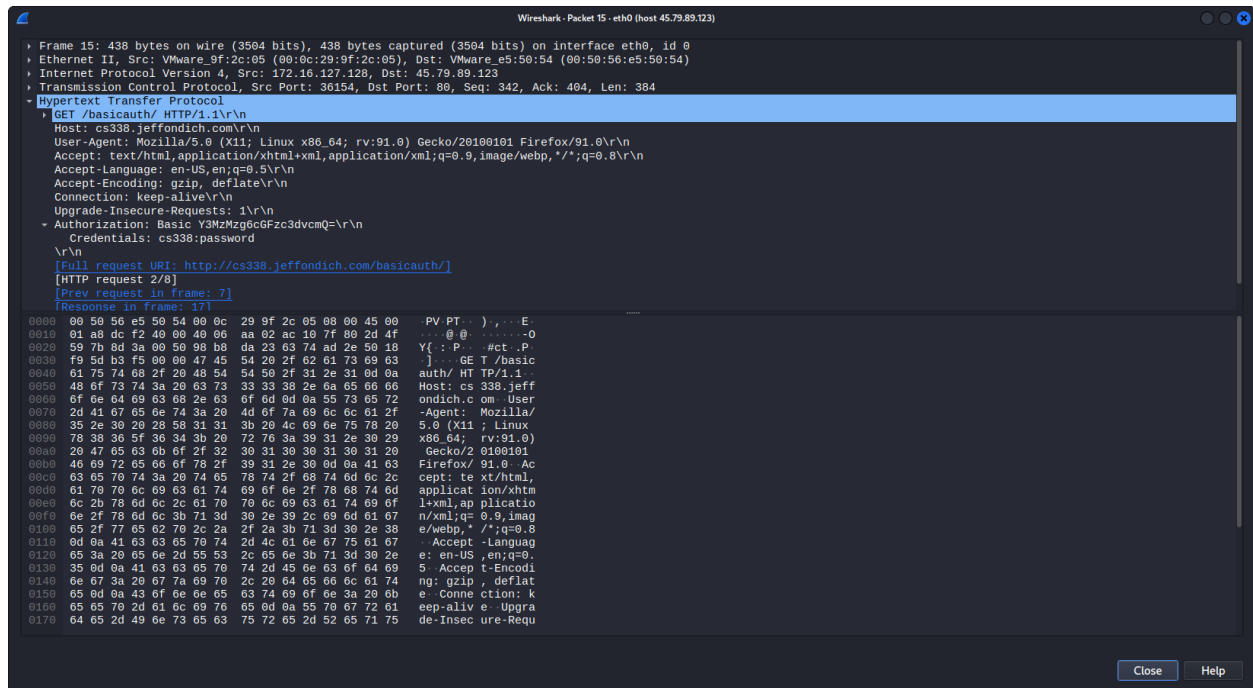
Above is an overview screenshot of the full client-server interaction when visiting the website <http://cs338.jeffondich.com/basicauth/>.

The first 6 frames detail the TCP handshake between the client and server. For whatever reason (perhaps multi-threading) the client sent two [SYN] requests, which, in turn, were answered by two [SYN, ACK] requests, and responded to by the client's own two [ACK] requests. Hence, the TCP handshake occurred twice, and this was consistent between multiple packet collection sessions.

In frame 7, however, things get interesting, the client is requesting “GET /basicauth/ HTTP/1.1”, this seems like the client requesting the mechanism for authentication with the server. The server acknowledges this in frame 8, and then sends “Unauthorized (text/html)” in frame 9. Following this, communications are a combination of [ACK] and [FIN] between the client and

server, indicating that communications could very well end here should proper authentication not be sent by the client.

Yet, low and behold, the client sends its credentials in frame 15.



The image shows a Wireshark packet capture window. The top pane displays the packet list, with frame 15 selected. The middle pane shows the packet details for the selected frame, highlighting the 'Authorization' header in the 'Hypertext Transfer Protocol' section. The bottom pane shows the raw packet data in hexadecimal and ASCII. The 'Authorization' header value is 'Basic Y3MzMzg6cGFzc3dvcmQ=\r\n', which is the base64-encoded string 'cs338:jeffondich.com' followed by a carriage return and a newline character.

```
Frame 15: 438 bytes on wire (3504 bits), 438 bytes captured (3504 bits) on interface eth0, id 0
Ethernet II, Src: VMware_Bf:2c:05 (00:0c:29:9f:2c:05), Dst: VMware_e5:50:54 (00:50:56:e5:50:54)
Internet Protocol Version 4, Src: 172.16.127.128, Dst: 45.79.89.123
Transmission Control Protocol, Src Port: 36154, Dst Port: 80, Seq: 342, Ack: 404, Len: 384
Hypertext Transfer Protocol
  GET /basicauth/ HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; 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
  Connection: keep-alive\r\n
  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 2/0]
[Prev request in frame: 7]
[Response in frame: 17]
```

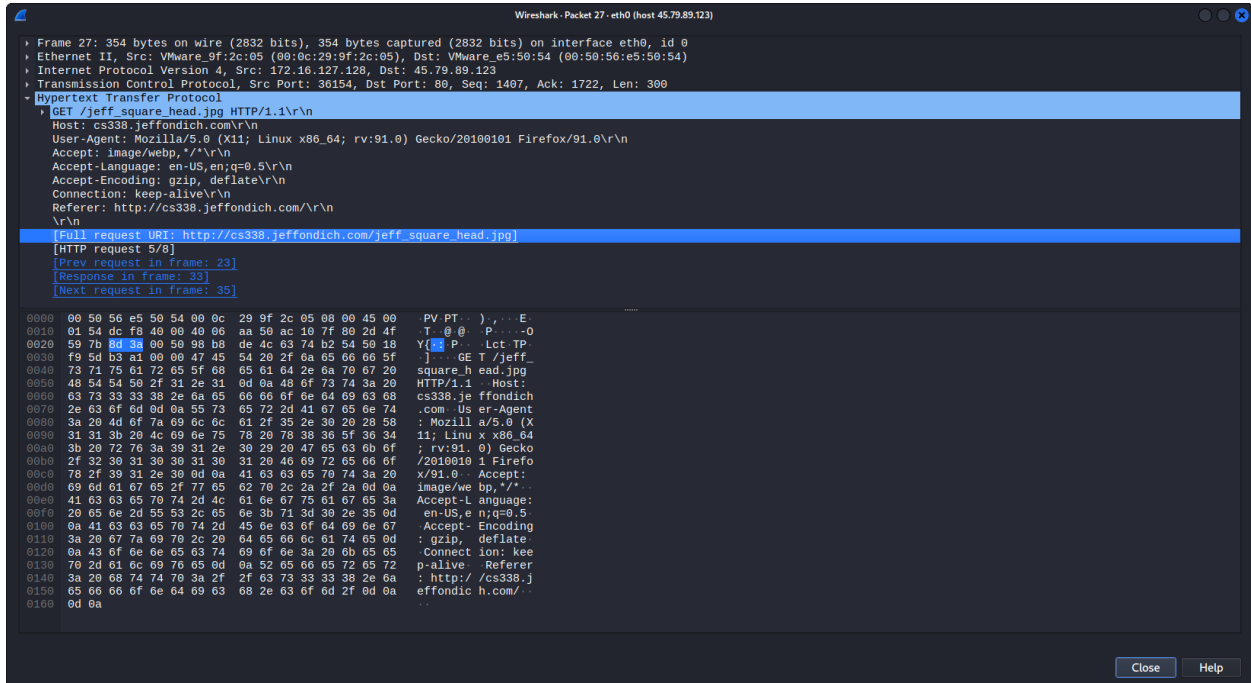
These credentials are found in the payload of the packet, under the HTTP headers, are encoded in base64, (not encrypted, no keys were visibly sent, and this observation matches my expectations from the Authentication's documentation) and are delivered to the server as plain text. I was expecting to have to dig around a little deeper to find these credentials, but apparently Basic HTTP Authentication is so readable that wireshark saw the characters Y3MzMzg6cGFzc3dvcmQ=\r\n and felt enlightened enough to decode them for me then and there, putting the username:password under its own little tab.

After this, the server sends an [ACK] followed by an OK (text/html) in frame 17 signaling that it got the client's message and found it acceptable. And... we're in.

Following this authentication sequence, things seem to happen in the manner of standard TCP communication and HTTP GET requests. I opened

each of the subsections of the website, looking at the server's sensitive information, such as jeff_square_head.jpg.

Imagine, if I (a malicious third-party) had intercepted these packets, I would have unauthorized access to this highly privy information.



The image shows a Wireshark packet capture window titled "Wireshark - Packet 27: eth0 (host 45.79.89.123)". The packet list on the left shows "Frame 27: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits) on interface eth0, id 0". The packet details pane shows the following structure:

- Frame 27: 354 bytes on wire (2832 bits), 354 bytes captured (2832 bits) on interface eth0, id 0
- Ethernet II, Src: VMware 9f:2c:05 (00:0c:29:9f:2c:05), Dst: VMware e5:50:54 (00:50:56:e5:50:54)
- Internet Protocol Version 4, Src: 172.16.127.128, Dst: 45.79.89.123
- Transmission Control Protocol, Src Port: 36154, Dst Port: 80, Seq: 1487, Ack: 1722, Len: 380
- Hypertext Transfer Protocol
 - GET /jeff_square_head.jpg HTTP/1.1\r\n
 - Host: cs338.jeffondich.com\r\n
 - User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20190101 Firefox/91.0\r\n
 - Accept: image/webp,*/*\r\n
 - Accept-Language: en-US,en;q=0.5\r\n
 - Accept-Encoding: gzip, deflate\r\n
 - Connection: keep-alive\r\n
 - Referer: http://cs338.jeffondich.com/\r\n
 - \r\n

The packet bytes pane shows the raw data of the packet, including the HTTP request line and headers in hexadecimal and ASCII format.