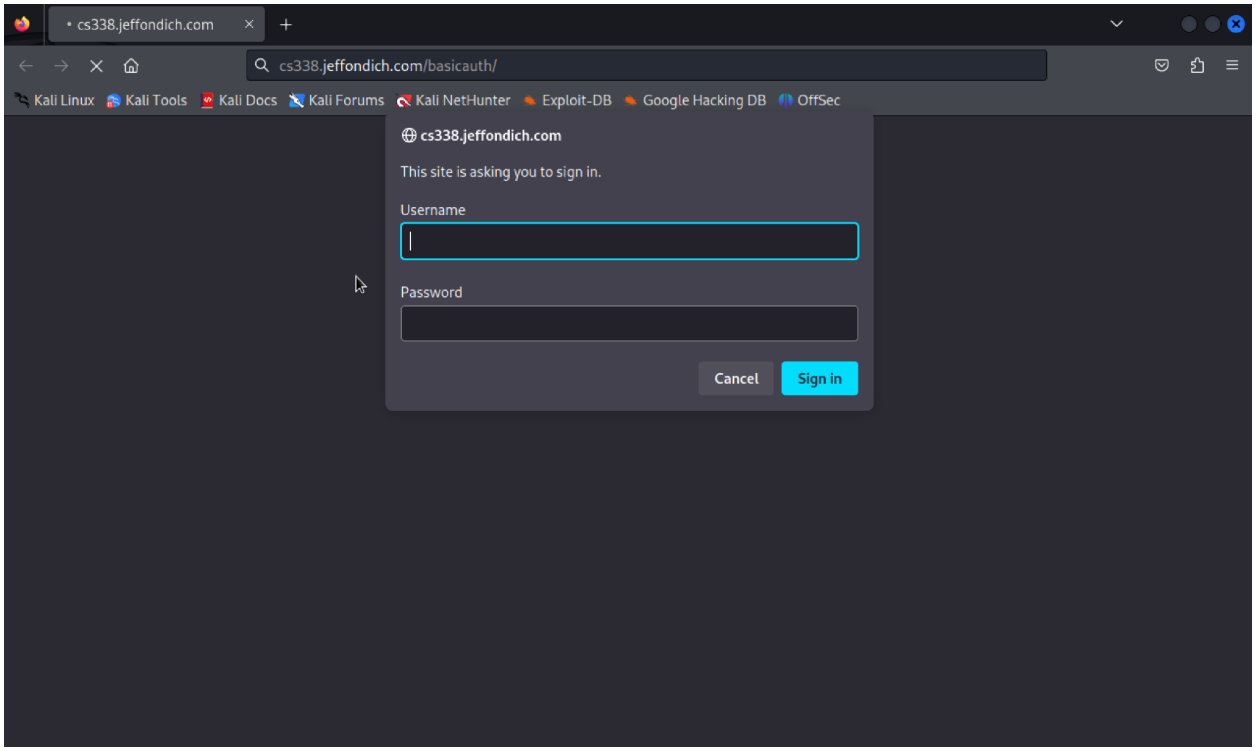
Varun Saini

HTTP's Basic Authentication: A Story

Navigating to page

Page layout

This is the dialog that appears when navigating to http://cs338.jeffondich.com/basicauth/:



Relevant frames

The following 10 frames are the result of navigating to this page:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.64.2	45.79.89.123	TCP	74	32978 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=199369778 TSecr=0 WS=128
2	0.000116423	192.168.64.2	45.79.89.123	TCP	74	32992 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=199369778 TSecr=0 WS=128
3	0.050896583	45.79.89.123	192.168.64.2	TCP	66	80 → 32992 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_PERM WS=128
4	0.050932236	192.168.64.2	45.79.89.123	TCP	54	32992 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
5	0.051399805	45.79.89.123	192.168.64.2	TCP	66	80 → 32978 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1382 SACK_PERM WS=128
6	0.051418374	192.168.64.2	45.79.89.123	TCP	54	32978 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
7	0.051642464	192.168.64.2	45.79.89.123	HTTP	408	GET /basicauth/ HTTP/1.1
8	0.101080305	45.79.89.123	192.168.64.2	TCP	60	80 → 32992 [ACK] Seq=1 Ack=355 Win=64128 Len=0
9	0.101890796	45.79.89.123	192.168.64.2	HTTP	457	HTTP/1.1 401 Unauthorized (text/html)
10	0.101982681	192.168.64.2	45.79.89.123	TCP	54	32992 → 80 [ACK] Seq=355 Ack=404 Win=64128 Len=0

Frame analysis

TCP connections

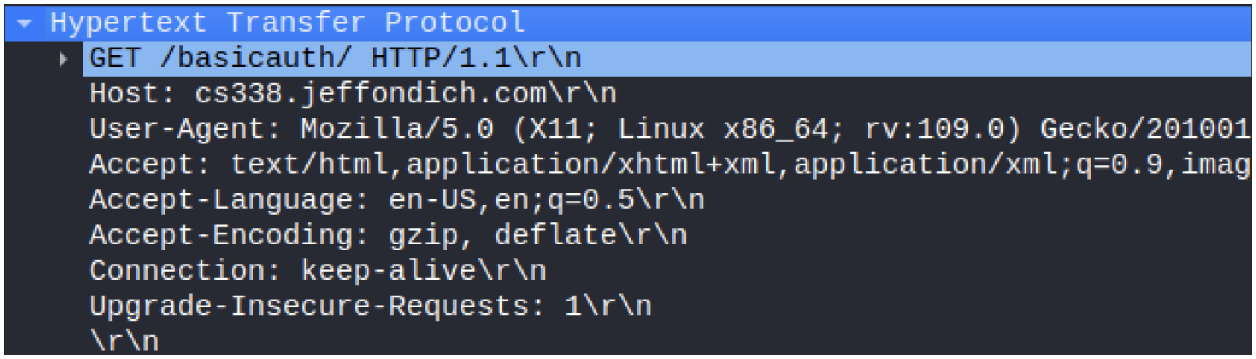
As we can see, there are two TCP connections that are initialized:

- Between client port 32978 and server port 80 (frames 1, 5, 6)
- Between client port 32992 and server port 80 (frames 2, 3, 4)

Thus, the first 6 frames are the components of two interwoven three-way TCP handshakes.

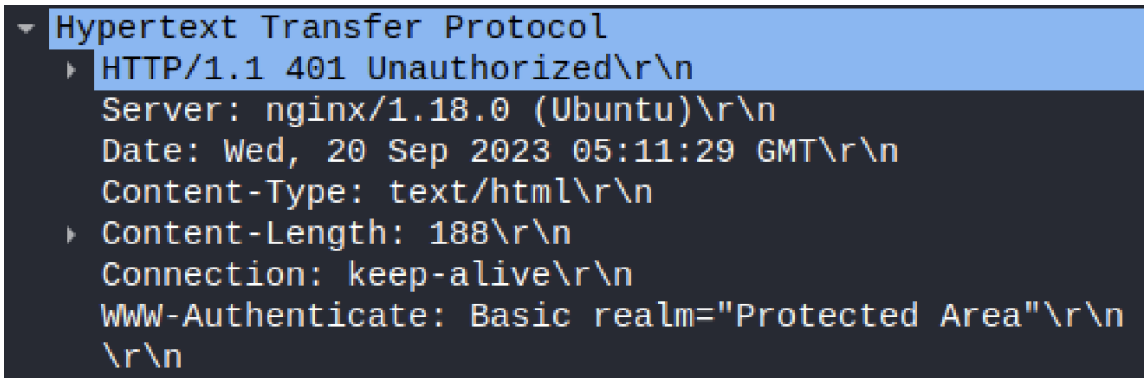
HTTP requests and responses

Frame 7 is the first HTTP frame. As we can see, this is sent from the client to the server (IP 192.168.64.2 to 45.79.89.123) and is a fairly standard GET request for "/basicauth/":



Frame 8 is an [ACK] packet sent from the server to the client acknowledging receipt of the HTTP request in frame 7.

Frame 9 is an HTTP response for the HTTP request sent in frame 7. These are the headers and body of this response (two separate images):



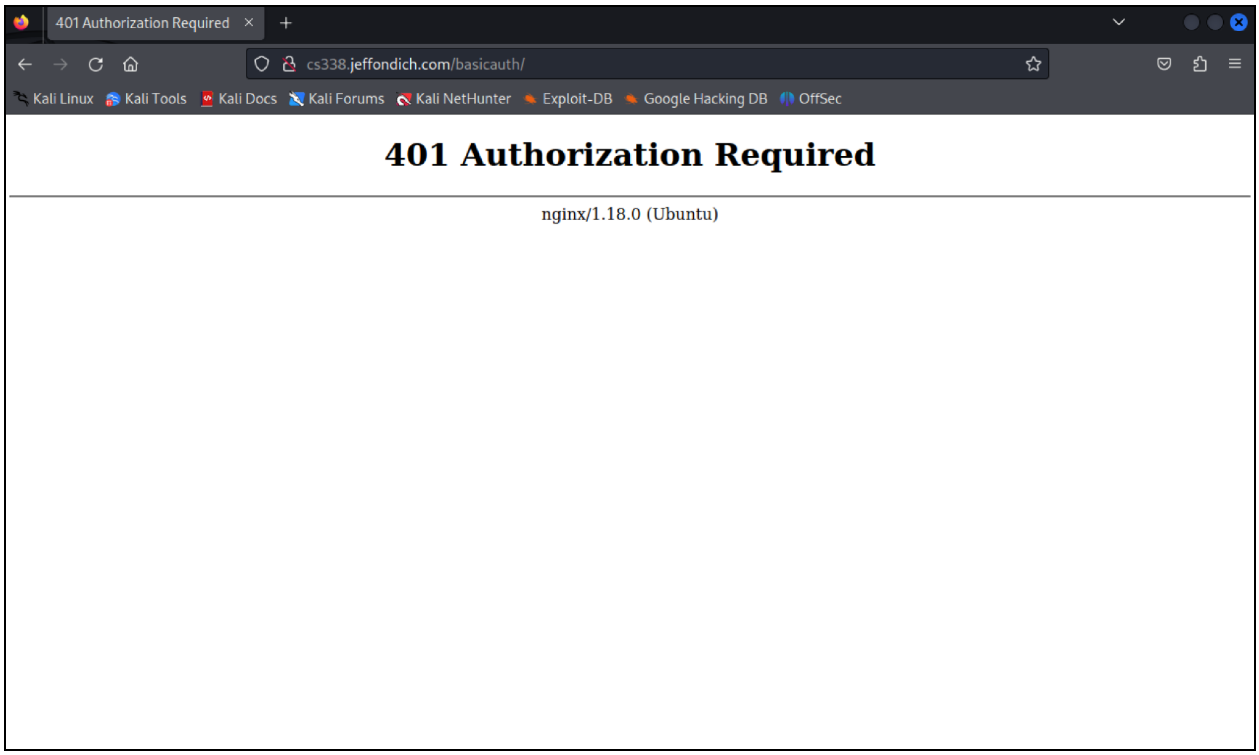
```
▾ Line-based text data: text/html (7 lines)
    <html>\r\n
    <head><title>401 Authorization Required</title></head>\r\n
    <body>\r\n
    <center><h1>401 Authorization Required</h1></center>\r\n
    <hr><center>nginx/1.18.0 (Ubuntu)</center>\r\n
    </body>\r\n
    </html>\r\n
```

As we can see, this HTTP response contains the “WWW-Authenticate” header field. As described in this section of the relevant RFC, this means that the server is replying to the client’s initial HTTP GET request with a “Basic” authentication challenge.

```
Upon receipt of a request for a URI within the protection space that
lacks credentials, the server can reply with a challenge using the
401 (Unauthorized) status code ([RFC7235], Section 3.1) and the
WWW-Authenticate header field ([RFC7235], Section 4.1).

For instance:

   HTTP/1.1 401 Unauthorized
   Date: Mon, 04 Feb 2014 16:50:53 GMT
   WWW-Authenticate: Basic realm="WallyWorld"

where "WallyWorld" is the string assigned by the server to identify
the protection space.
```

In our case, “Protected Area” is the name of the realm.

This packet also sends over the HTML for the “401 Authorization Required” page in the response body. If the client declines the challenge (by clicking “Cancel”), this body is the page that is displayed:

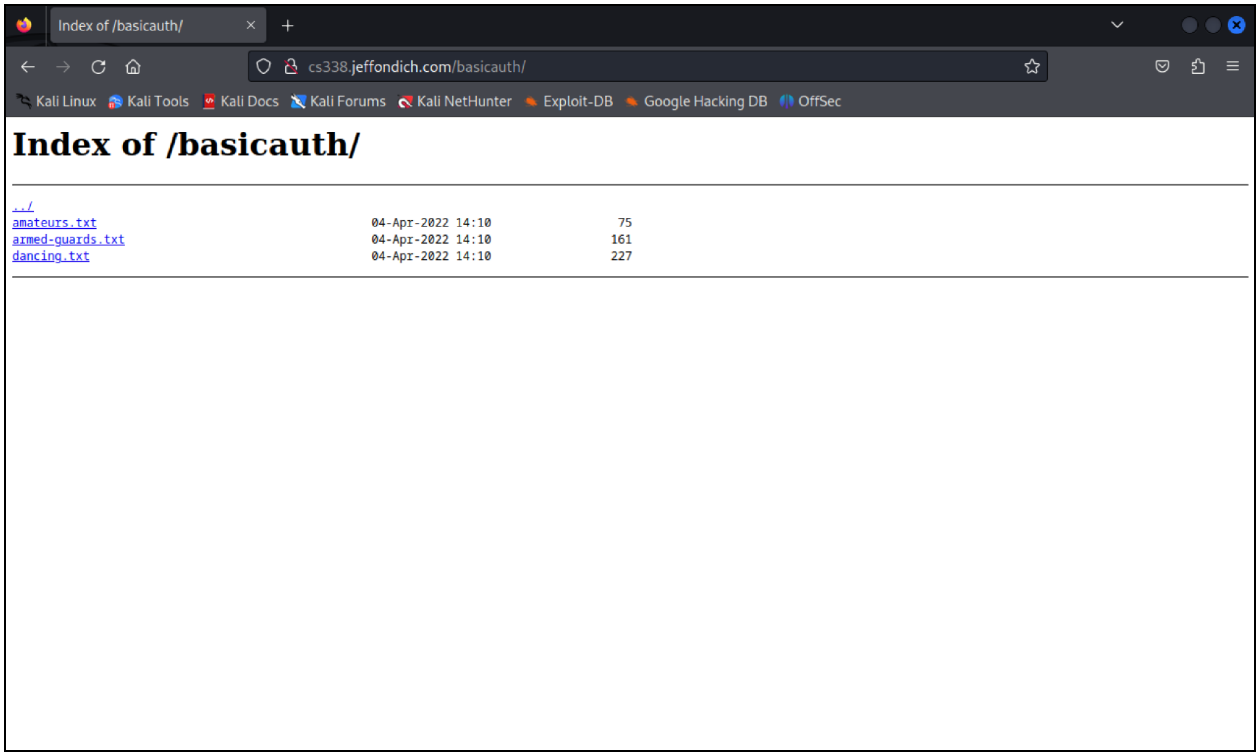


Frame 10 is an [ACK] packet sent from the client to the server acknowledging receipt of the HTTP response in frame 9.

Entering username and password

Page layout

If, instead of clicking cancel, we enter “cs338” and “password” and click “Sign In,” we unlock Jeff’s hidden treasures as pictured below:



Relevant frames

After arriving at this page, the following 7 frames are captured:

No.	Time	Source	Destination	Protocol	Length	Info
11	5.742462578	192.168.64.2	45.79.89.123	TCP	54	32978 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
12	5.798195421	45.79.89.123	192.168.64.2	TCP	60	80 → 32978 [FIN, ACK] Seq=1 Ack=2 Win=64256 Len=0
13	5.798230779	192.168.64.2	45.79.89.123	TCP	54	32978 → 80 [ACK] Seq=2 Ack=2 Win=64256 Len=0
14	8.472594002	192.168.64.2	45.79.89.123	HTTP	451	GET /basicauth/ HTTP/1.1
15	8.524139904	45.79.89.123	192.168.64.2	TCP	60	80 → 32992 [ACK] Seq=404 Ack=752 Win=64128 Len=0
16	8.525438186	45.79.89.123	192.168.64.2	HTTP	458	HTTP/1.1 200 OK (text/html)
17	8.525595651	192.168.64.2	45.79.89.123	TCP	54	32992 → 80 [ACK] Seq=752 Ack=808 Win=64128 Len=0

Frame analysis

TCP connections

Frames 11, 12, and 13 seem to be closing the TCP connection between client port 32978 and server port 80. I am not entirely sure why this happened here, but it seems like this connection was unused throughout this process. Only six frames reference the connection on this port: the three that set it up (1, 5, 6) and the three that tore it down (11, 12, 13).

HTTP requests and responses

Similar to before, we have four frames:

- Frames 14 and 16: HTTP request and response
- Frame 15 and 17: [ACK] acknowledging receipt of the previous HTTP frame

Unlike the HTTP request in Frame 7, Frame 14's HTTP request has an "Authorization" header:

```
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:109.0) Gecko/20100101 Firefox/115.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.
  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
  \r\n
```

As we can see, this is encoded using Base64: "Y3MzMzg6cGFzc3dvcmQ=".

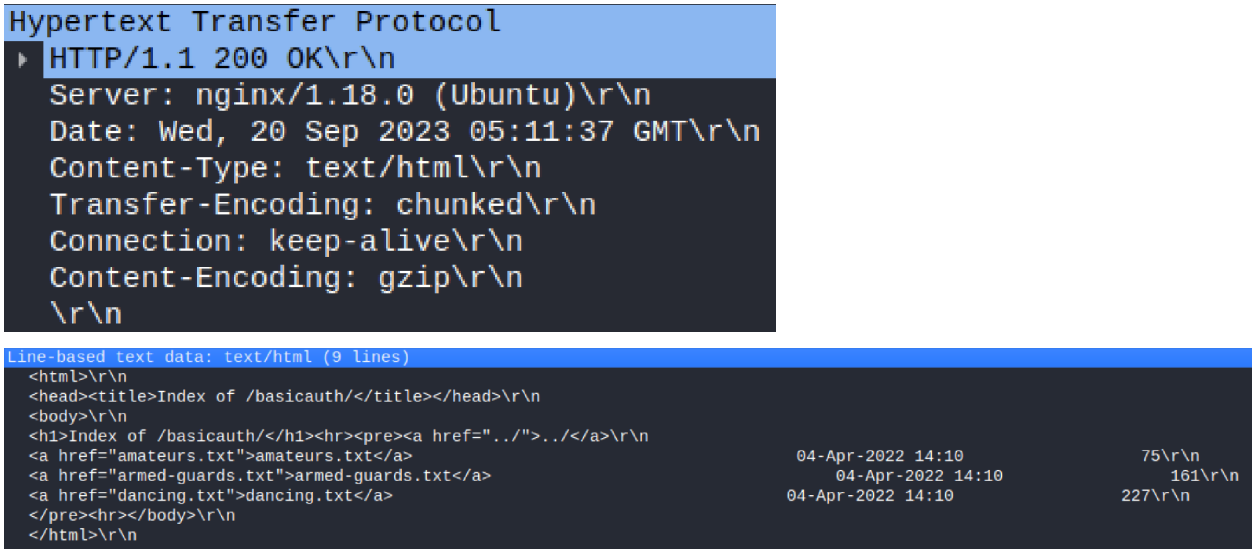
If we decode this, we get "cs338:password".

```
varunsaini — -zsh — 100×26
varunsaini@Varun-Sainis-MacBook-Pro ~ % echo Y3MzMzg6cGFzc3dvcmQ= | base64 -d
cs338:password%
varunsaini@Varun-Sainis-MacBook-Pro ~ %
```

This is exactly how the encoding is described in [this section of the relevant RFC](#).

> To receive authorization, the client
>
> 1. obtains the user-id and password from the user,
>
> 2. constructs the user-pass by concatenating the user-id, a single colon (":") character, and the password,
>
> 3. encodes the user-pass into an octet sequence (see below for a discussion of character encoding schemes),
>
> 4. and obtains the basic-credentials by encoding this octet sequence using Base64 ([RFC4648], Section 4) into a sequence of US-ASCII characters ([RFC0020]).

The HTTP response in Frame 16 looks like a normal successful response in this case since the user has been authenticated. We can see that the server returns a status code of 200 (OK), and the response body is the HTML for the hidden webpage (two pictures below).



Entering the wrong username/password

For this scenario, I am not including screenshots of captured frames, as it is fairly simple to explain what I observed.

- When we first navigated to http://cs338.jeffondich.com/basicauth/, an HTTP GET request was sent to the server (Frame 7). This request's headers did not contain an "Authorization" field.

- In response, the server sent back an HTTP response with the "WWW-Authenticate" header, meaning that it was replying with a "Basic" authentication challenge (Frame 9).

- If we press "Sign In" after entering the wrong username or password, the client sends an HTTP request to the server containing an "Authorization" field, similar to Frame 14. However, since the username and/or password are incorrect, the encoded "Authorization" field is not correct.

- As a result, the server simply responds to the client with an HTTP response that is identical to Frame 9's response (with the "WWW-Authenticate" header).

- Same as before: the client then prompts the user again for a username and password.

- This loop continues until the user either presses "Sign In" after entering the correct username and password (allowing them to see the hidden contents) or presses "Cancel" (displaying the "401 Authorization Required" page).