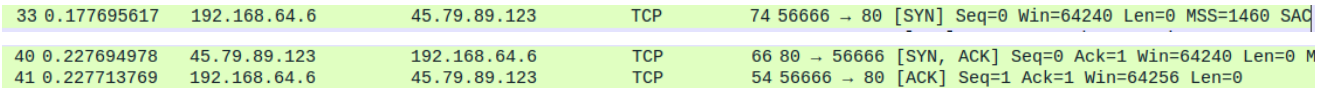
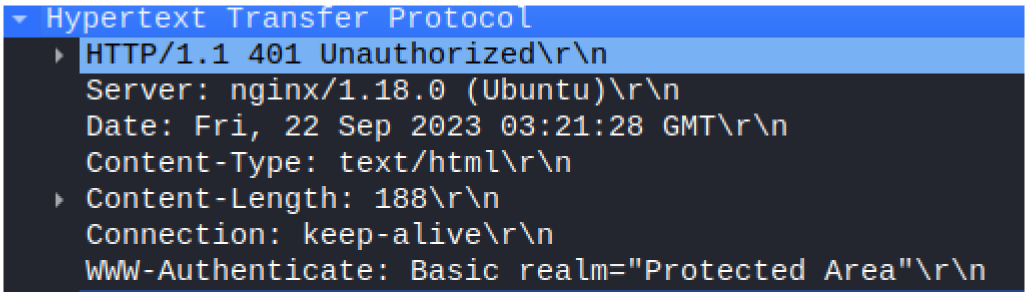
Timeline

In the beginning, there was nothing. Then, there was a TCP connection initiated by my browser at IP `192.168.64.6` on port `56666` to connect to Jeff's server at IP `45.79.89.123`. Because the intention of the TCP connection was to later send an HTTP request, all the TCP packets talks to the default HTTP port `80` on Jeff's server.



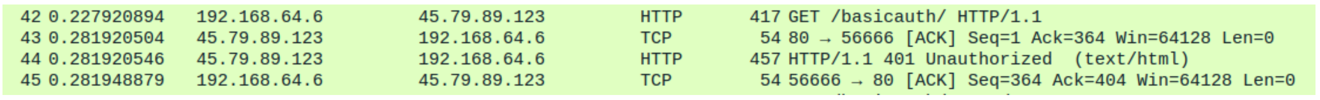
```
33 0.177695617   192.168.64.6      45.79.89.123     TCP    74 56666 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC

40 0.227694978   45.79.89.123      192.168.64.6     TCP    66 80 → 56666 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 M
41 0.227713769   192.168.64.6      45.79.89.123     TCP    54 56666 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
```

All seemed well until my browser asked to see some secretive content on Jeff's server by sending an HTTP GET request to the URL `http://cs338.jeffondich.com/basicauth/`. Because no credential is provided, the browser received a `401` response code, indicating that the client is not authorized to access the wanted resource (https://datatracker.ietf.org/doc/html/rfc7231#section-6.1).



```
▾ Hypertext Transfer Protocol
  ▸ HTTP/1.1 401 Unauthorized\r\n
    Server: nginx/1.18.0 (Ubuntu)\r\n
    Date: Fri, 22 Sep 2023 03:21:28 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
```

But the server did more than just denial, it also specified how to unlock its secrecy. In server's HTTP response header, there was a field named `WWW-Authenticate` that tells the client what type of authentication is needed. In this case, the value `Basic` referred to the 'Basic' HTTP Authentication Scheme specified in RFC 7617 (https://datatracker.ietf.org/doc/html/rfc7617#page-3).
What's also included in the field is the required parameter `realm`. According to the RFC 7617, a realm, also known as a protected space, is defined by the canonical root URI that can partition the access right of protected resources on the server. In our case, we were trying to access the realm under `http://cs338.jeffondich.com/basicauth/` with realm name `Protected Area`.

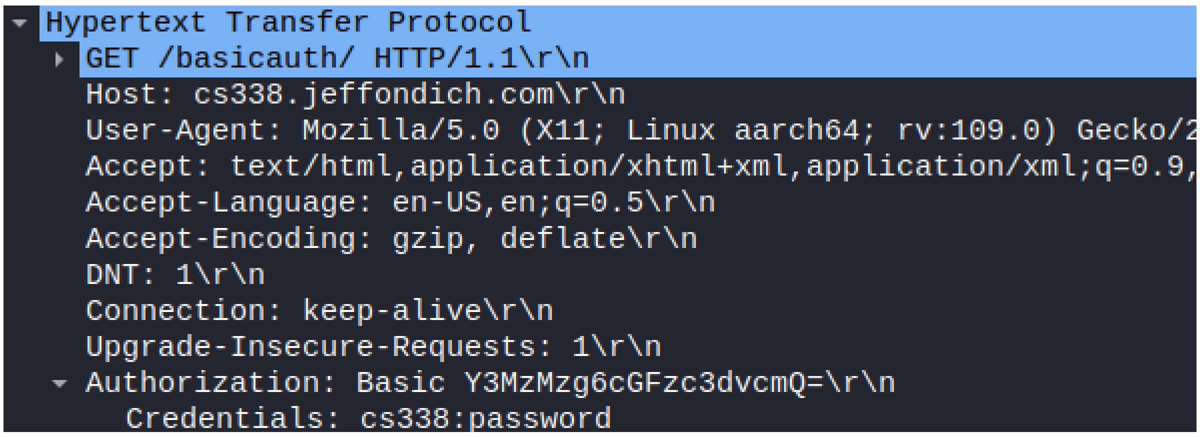


```
42 0.227920894   192.168.64.6      45.79.89.123     HTTP   417 GET /basicauth/ HTTP/1.1
43 0.281920504   45.79.89.123      192.168.64.6     TCP     54 80 → 56666 [ACK] Seq=1 Ack=364 Win=64128 Len=0
44 0.281920546   45.79.89.123      192.168.64.6     HTTP   457 HTTP/1.1 401 Unauthorized  (text/html)
45 0.281948879   192.168.64.6      45.79.89.123     TCP     54 56666 → 80 [ACK] Seq=364 Ack=404 Win=64128 Len=0
```

After acknowledging receiving this information with a TCP packet in `ACK` flag, the browser client prompt user a window to input user-id and password to satisfy the 'Basic' authentication scheme. This authentication scheme requires the client to do the following:

```
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.
```

As the RFC described, the browser will get user-id and password from the prompt window inputs, concatenate them with `:` , encode them with Base64, and send them in the next HTTP request header.

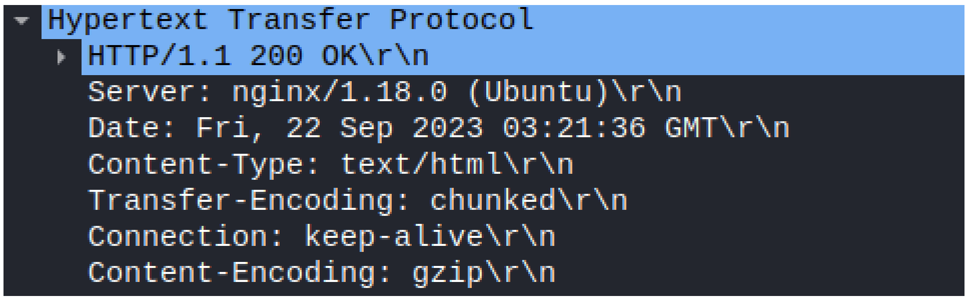


```
▾ Hypertext Transfer Protocol
  ▸ GET /basicauth/ HTTP/1.1\r\n
    Host: cs338.jeffondich.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/2
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
  ▾ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
      Credentials: cs338:password
```

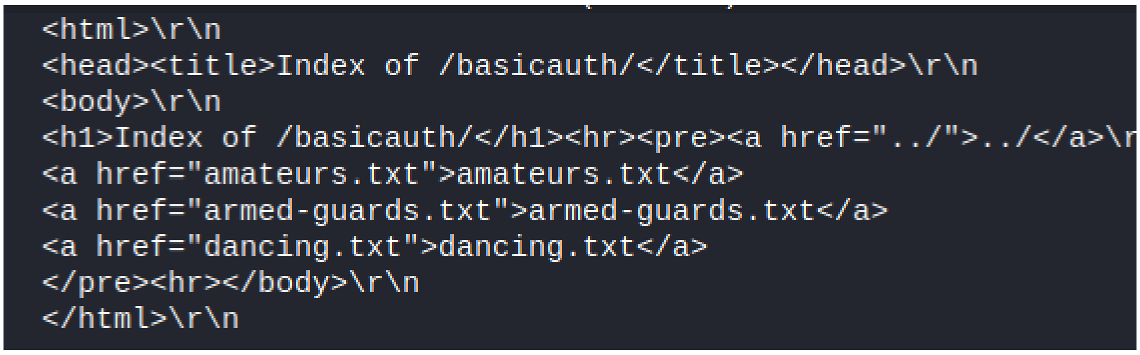
From this example, we can see that the next HTTP request header included a `Authorization` field. It confirmed that it's following the `Basic` authentication scheme, and attached the Base64 string `Y3MzMzg6cGFzc3dvcmQ=`, which decoded to the user-id:password. It's worth noticing that Base64 is an encoding not encryption, meaning that user-id and password are basically sent through the internet in plaintext. Any eavesdropper tapping into the network can easily acquire this pair of user-pass. This is why the Security Considerations section of RFC 7617 highlighted that the 'Basic' HTTP authentication scheme SHOULD NOT be used to protect anything sensitive or valuable.
We can deduce from this that in our interaction with `http://cs338.jeffondich.com/basicauth/`, the password's correctness is checked by the server. If client can already determine the correctness of the password, it won't need to send the password as plaintext to the server. This is further proven when we input the wrong password of which we shall see the reason later. From now on, our story can diverge a little.

If the user inputs the correct user-id and password

In this case, we got the access to the protected resource. The server sent the content we want with a `200` success status code. We now see that the content we want is an HTML document with some basic layouts and links.

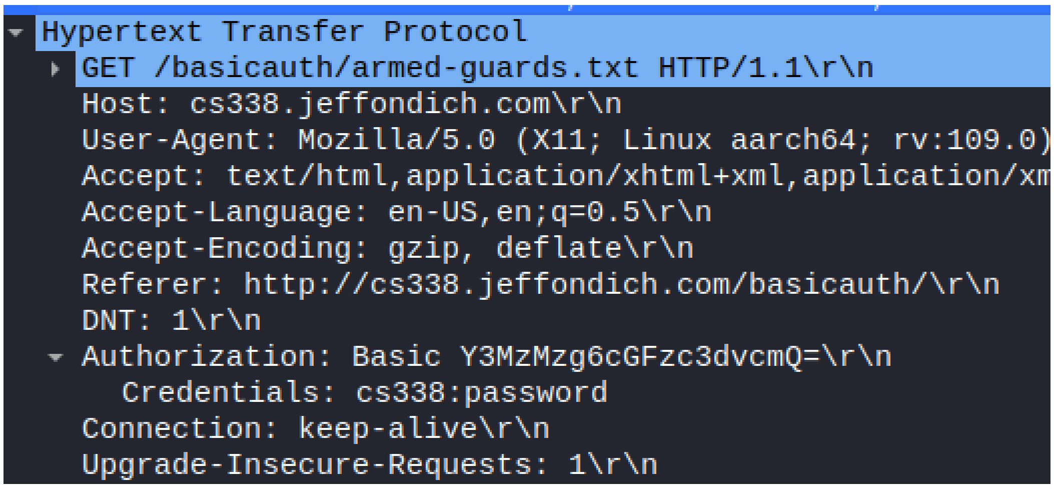


```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Fri, 22 Sep 2023 03:21:36 GMT\r\n
  Content-Type: text/html\r\n
  Transfer-Encoding: chunked\r\n
  Connection: keep-alive\r\n
  Content-Encoding: gzip\r\n
```



```
<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
<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
</html>\r\n
```

When we click any links to visit any page under `/basicauth/`, the authorization header will still be included so that we don't have to input user-id and passcode again for resource under the same protection space.



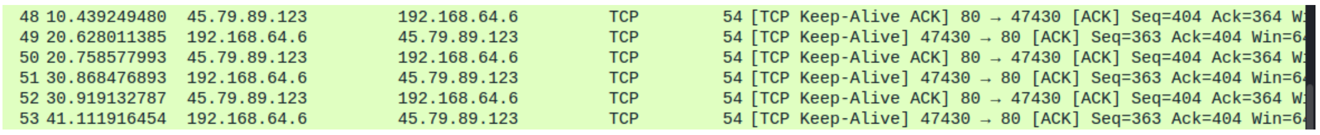
```
Hypertext Transfer Protocol
  GET /basicauth/armed-guards.txt HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0)
  Accept: text/html,application/xhtml+xml,application/xm
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Referer: http://cs338.jeffondich.com/basicauth/\r\n
  DNT: 1\r\n
  Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
    Credentials: cs338:password
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
```

This "caching" behavior is specified in section 6.2 in RFC 7617

> Existing HTTP clients and user agents typically retain authentication
> information indefinitely. HTTP does not provide a mechanism for the
> origin server to direct clients to discard these cached credentials,
> since the protocol has no awareness of how credentials are obtained or managed by the user agent.

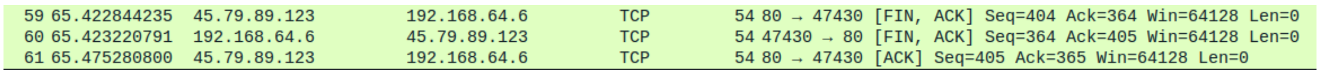
If the user just doesn't input anything

In this annoying but actually common case, the browser will keep sending special TCP Keep Alive packets to tell the server keep the TCP connection open and continue to wait for user response.



```
48 10.439249480  45.79.89.123    192.168.64.6    TCP    54 [TCP Keep-Alive ACK] 80 → 47430 [ACK] Seq=404 Ack=364 Wi
49 20.628011385  192.168.64.6    45.79.89.123    TCP    54 [TCP Keep-Alive] 47430 → 80 [ACK] Seq=363 Ack=404 Win=64
50 20.758577993  45.79.89.123    192.168.64.6    TCP    54 [TCP Keep-Alive ACK] 80 → 47430 [ACK] Seq=404 Ack=364 Wi
51 30.868476893  192.168.64.6    45.79.89.123    TCP    54 [TCP Keep-Alive] 47430 → 80 [ACK] Seq=363 Ack=404 Win=64
52 30.919132787  45.79.89.123    192.168.64.6    TCP    54 [TCP Keep-Alive ACK] 80 → 47430 [ACK] Seq=404 Ack=364 Wi
53 41.111916454  192.168.64.6    45.79.89.123    TCP    54 [TCP Keep-Alive] 47430 → 80 [ACK] Seq=363 Ack=404 Win=64
```

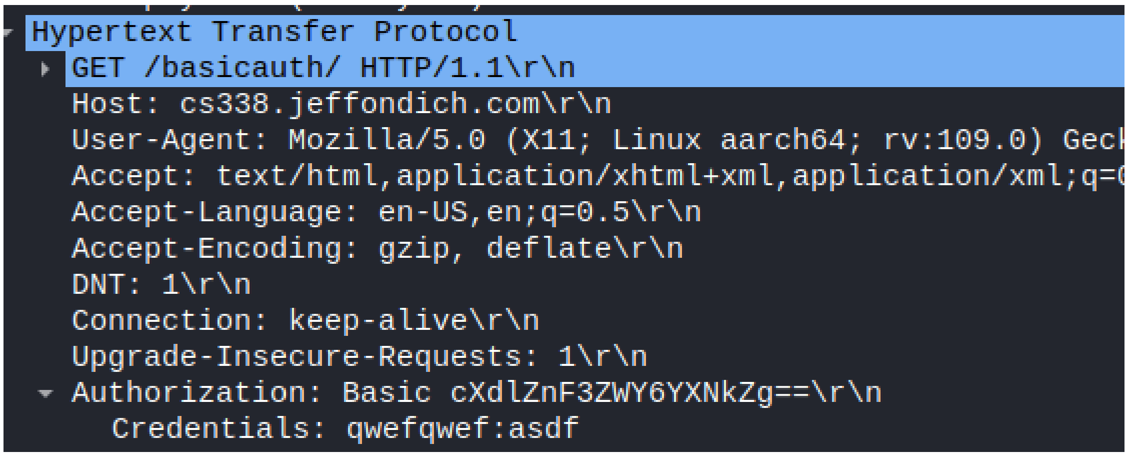
If at some point user closed the browser, TCP connection will be terminated by the server using TCP packet with `FIN` flag.



```
59 65.422844235  45.79.89.123    192.168.64.6    TCP    54 80 → 47430 [FIN, ACK] Seq=404 Ack=364 Win=64128 Len=0
60 65.423220791  192.168.64.6    45.79.89.123    TCP    54 47430 → 80 [FIN, ACK] Seq=364 Ack=405 Win=64128 Len=0
61 65.475280800  45.79.89.123    192.168.64.6    TCP    54 80 → 47430 [ACK] Seq=405 Ack=365 Win=64128 Len=0
```

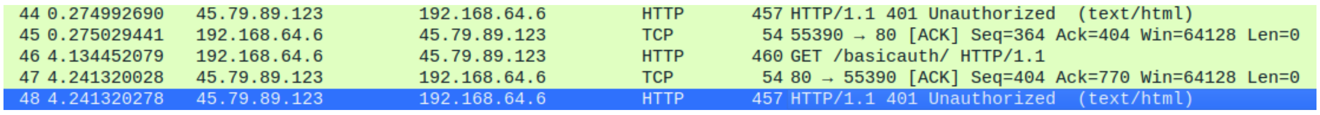
If the user entered the wrong user-id or password

The browser will still construct the user-pass and send it along, which shows it really doesn't check the password itself.



```
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:109.0) Gec
  Accept: text/html,application/xhtml+xml,application/xml;q=
  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 cXdlZnF3ZWY6YXNkZg==\r\n
    Credentials: qwefqwef:asdf
```

Upon receiving the wrong user-pass, the server will just ask with the same `401` unauthorized response again, and the browser prompt the user-id and password input once more.



```
44 0.274992690   45.79.89.123    192.168.64.6    HTTP   457 HTTP/1.1 401 Unauthorized  (text/html)
45 0.275029441   192.168.64.6    45.79.89.123    TCP     54 55390 → 80 [ACK] Seq=364 Ack=404 Win=64128 Len=0
46 4.134452079   192.168.64.6    45.79.89.123    HTTP   460 GET /basicauth/ HTTP/1.1
47 4.241320028   45.79.89.123    192.168.64.6    TCP     54 80 → 55390 [ACK] Seq=404 Ack=770 Win=64128 Len=0
48 4.241320278   45.79.89.123    192.168.64.6    HTTP   457 HTTP/1.1 401 Unauthorized  (text/html)
```