Basic HTTP Authentication

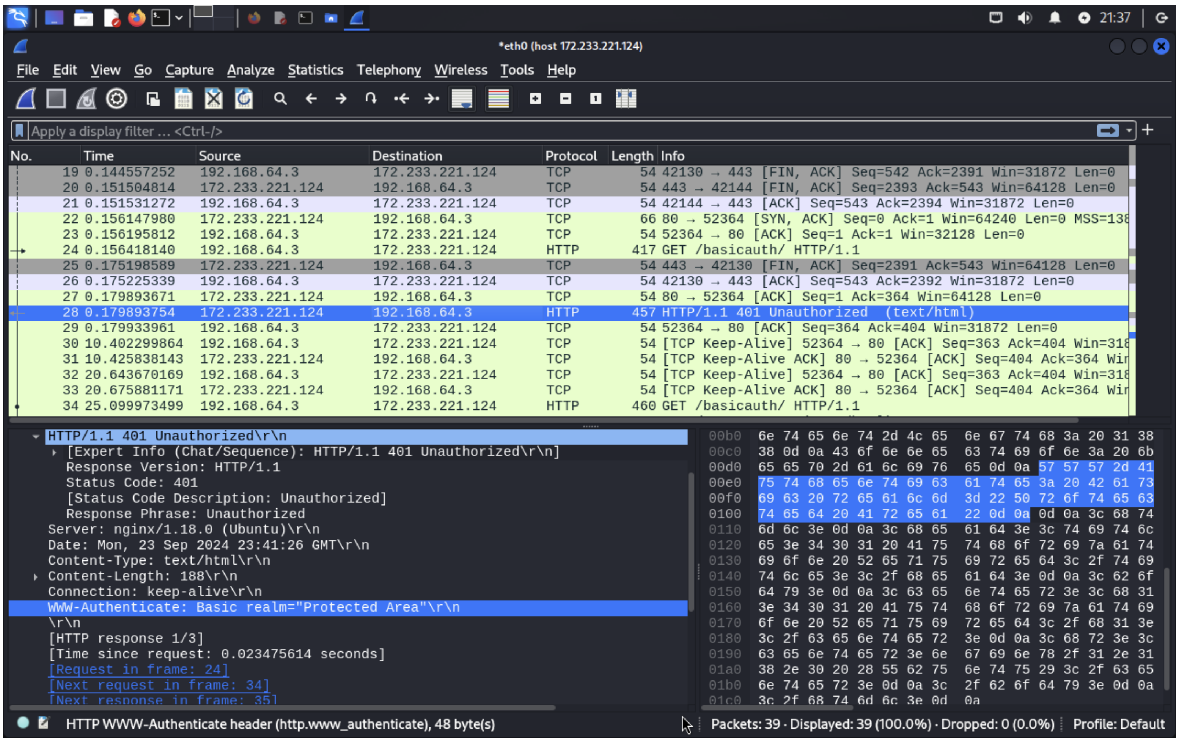
Let's say that I open a browser and navigate to http://cs338.jeffondich.com/basicauth/. What interactions occur between my browser and Jeff's server? We can answer this question by using Wireshark.

First, the browser sends two SYN packets to the server at port 443 (the standard HTTPS port) in order to initiate two TCP handshakes. The server responds with two SYN-ACK packets, and the browser responds to those with two ACK packets. Now, two TCP connections have been established between the browser and the server.

Next, the browser sends two "client hello" messages to initiate two TLS handshakes–one for each TCP connection. The server responds with two "server hello" messages. Aside from some "application data" packets, not much happens here because we aren't using HTTPS. The connections eventually close.

Now, we see more relevant interactions occur. The browser sends a SYN packet to the server at port 80 (the standard HTTP port) to initiate a TCP handshake. The server responds with a SYN-ACK packet, and the browser responds to that with an ACK packet. Now, a TCP connection has been established between the browser and the server.

Now that the browser has a TCP connection to port 80 on the server, the browser sends an HTTP GET request to the server in order to request the `/basicauth/` folder. In response, the server sends an HTTP 401 (Unauthorized) response to the browser.



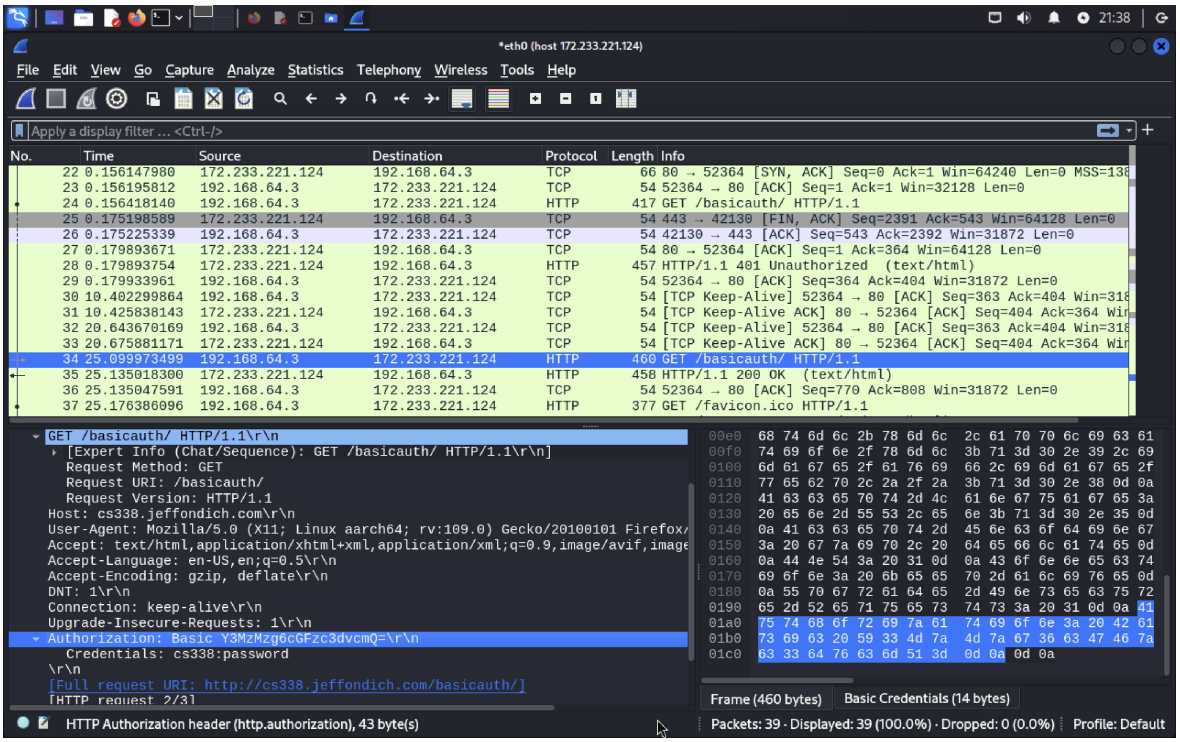
This response has a `WWW-Authenticate` header, specifying that the realm is named `Protected Area` and it is a basic realm (i.e. using the "basic" HTTP authentication scheme). It's good to know the realm name–if there are any other pages on this website that are in the same realm, then the login credentials for this page will also work for those pages!

Essentially, this 401 response and the `WWW-Authenticate` header means that the server will not let me access `/basicauth/` unless:

1. I log in and the browser sends my credentials to the server.
2. The server authenticates my credentials.
3. The server determines that I am authorized to view the folder.

At this point, the browser prompts me to log in. I enter `cs338` as the username and `password` as the password.

Then, the browser sends another HTTP GET request in an attempt to access the `/basicauth/` folder from the server. However, there is one key difference between this HTTP GET request and the unsuccessful request from earlier. Unlike the previous request, this request has an `Authorization` header. This contains the credentials that I entered.



The value is `Basic Y3MzMzg6cGFzc3dvcmQ=`. "Basic" refers to the HTTP authentication scheme. The string looks like base64, so I decoded it using the base64 command.

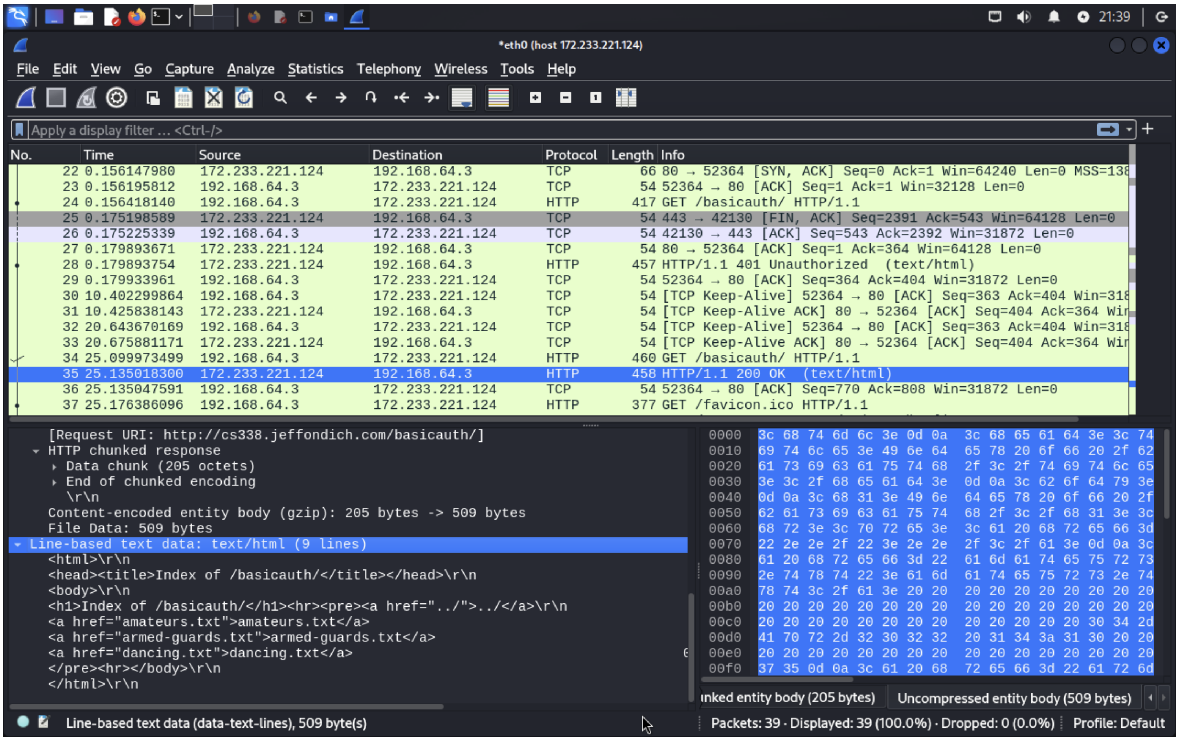
`echo "Y3MzMzg6cGFzc3dvcmQ=" | base64 -decode` returns `cs338:password`. These are the credentials that I used to log in! (From the screenshot, it looks like Wireshark automatically shows you the decoded credentials as well.) This aligns with RFC 7617, the Basic HTTP Authentication specification document, which states that the authentication scheme "transmits credentials as user-id/password pairs, encoded using Base64".

Anyway, this means that when the browser sends the username and password to the server, they are *not* encrypted–they are in clear text. They are encoded in base64, but it is trivial for an observer to decode them. Indeed, section 4 (Security Considerations) of the Basic HTTP Authentication specification document states that the "Basic authentication scheme is not a secure method of user authentication".

Essentially, the purpose of the Authorization header is for the browser to send the user's login credentials to the server. Then the server can use this information to authenticate the user's credentials, and then determine whether the user is authorized to view the resource.

In this case, the server responded with an HTTP 200 (OK) status. Fantastic! I am

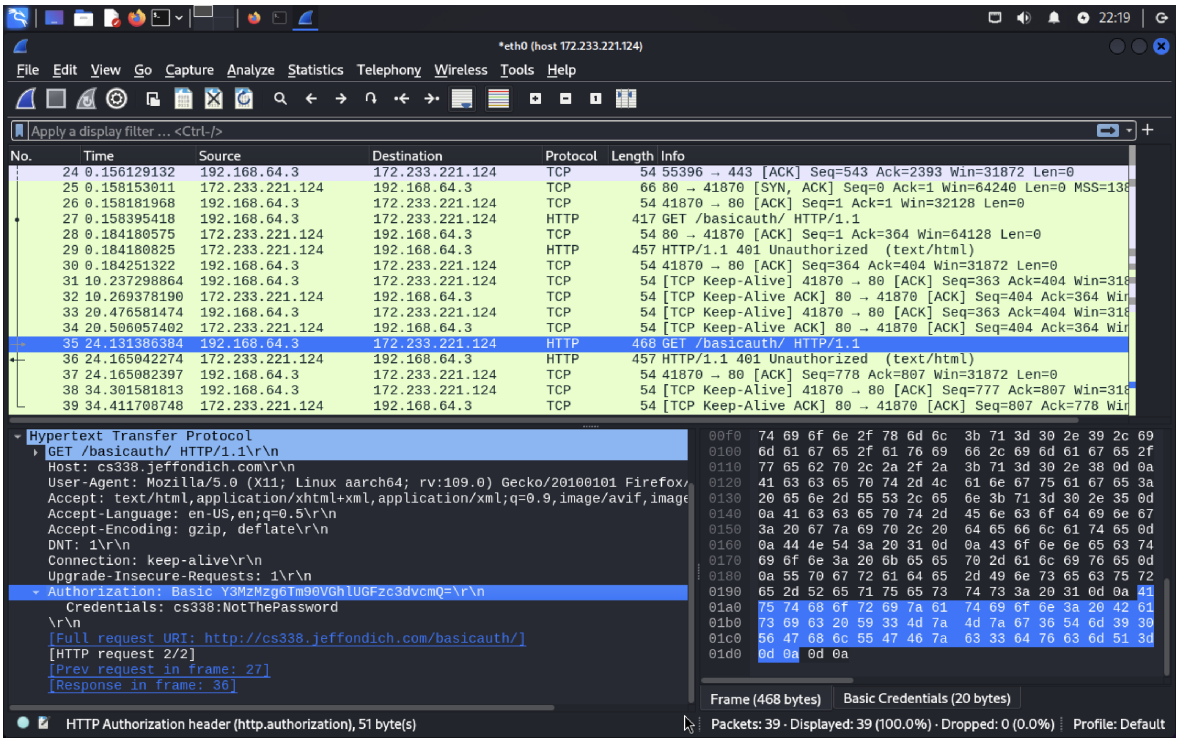
authorized to view the `/basicauth/` folder. You can see the contents of the folder in the response body, under the "line-based text data" header:



When I click on an item in the folder to access it (such as `amateurs.txt`), the browser sends an HTTP GET request to the server. Because I've already logged in, the request includes the Authorization header, with the same credentials that I logged in with. This makes sense–HTTP is a stateless protocol, so the browser will need to send my credentials to the server every time. Luckily for me, the browser remembers the credentials I provided when I first logged in, so I don't have to re-enter them every time I want to access a protected resource. This aligns with section 2.2 (Reusing Credentials) of the Basic HTTP Authentication specification document, which states that when a user has been authenticated for a particular resource and then requests to access other resources within the same protection space:

> A client MAY preemptively send the corresponding Authorization header field with requests for resources in that space without receipt of another challenge from the server.

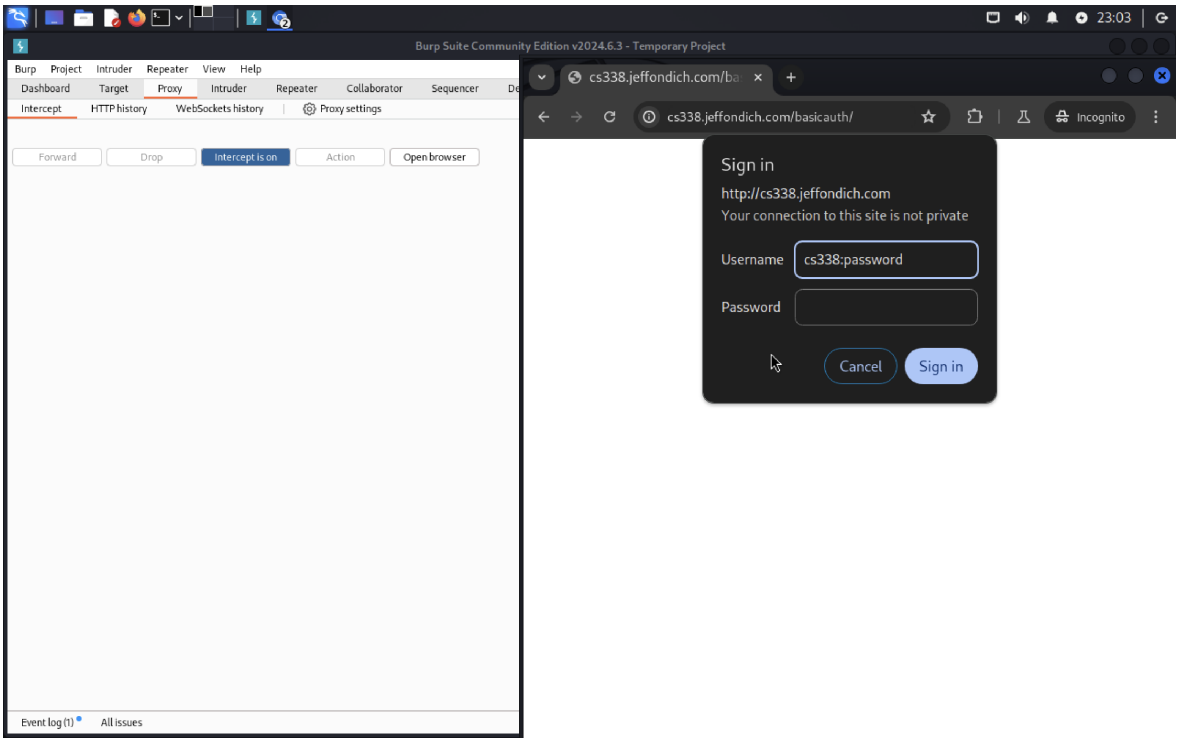
I wondered what happens when I submit an incorrect username or password. Would the server tell me why my request to access the folder was denied? I open a new private tab and type in the same username with a different password. The browser sends an HTTP GET request; the Authorization header includes my (incorrect) credentials.



`echo "Y3MzMzg6Tm90VGhlUGFzc3dvcmQ=" | base64 -decode` returns `cs338:NotThePassword`.

Once again, the server responds with an HTTP 401 (Unauthorized) status. I expected there would be an indicator of why I wasn't authorized (such as the password being incorrect) but the response contained the same information as the previous 401 response I got before I logged in.

Since the username and password are separated by a colon, I wondered what happens if I put a colon in the username. Would the browser prevent me from submitting a username with a colon as well as no password? I open Burp Suite and type this username with no password:



To my surprise, the browser doesn't prevent me from entering a username with a colon (and no password). I look at the user credentials encoded in base64 (which represents the string `cs338:password:`) and notice that if I replace the `6` at the end of the encoded string with a `=`, then this would represent the correct user credentials that I entered earlier. Sure enough, it works, and I'm able to access the folder without anyone detecting that my username was incorrect. This means that, hypothetically, if the username was `cs338` and the password was `password:`, you could successfully log in with the incorrect username `cs338:password` and no password. Both strings would be represented as `cs338:password:`, and therefore they'd have the same base64 encoding. Interesting stuff.

After using Wireshark and Burp Suite to investigate the interactions between my browser and Jeff's server, I have a better idea of how Basic HTTP Authentication works.