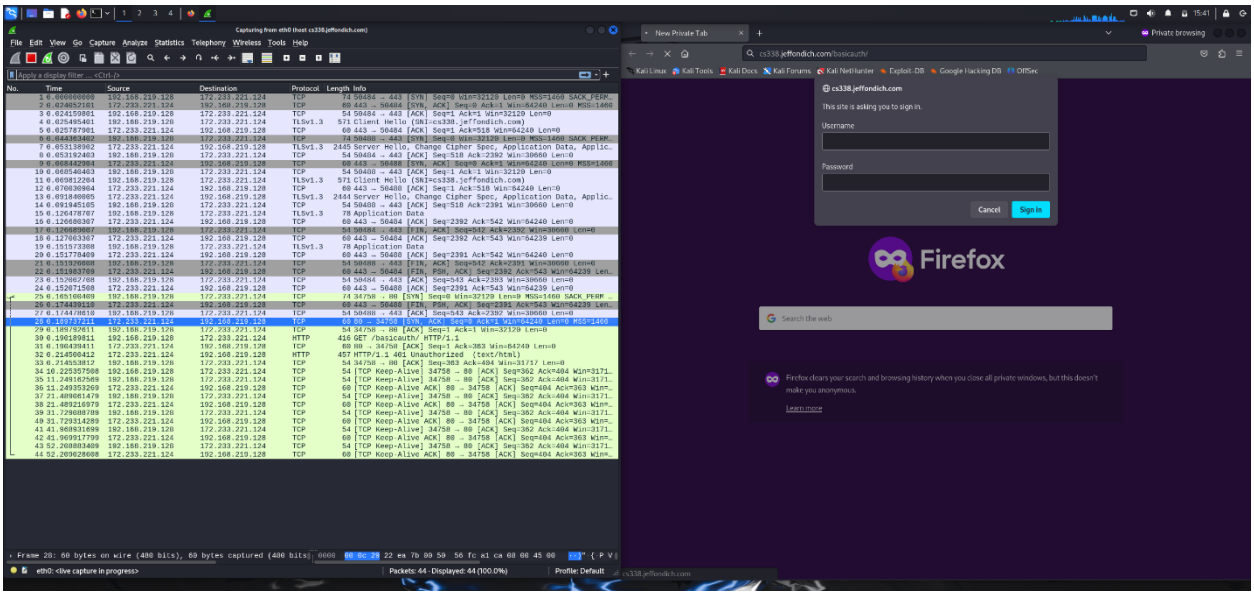
User authentication when accessing webservers

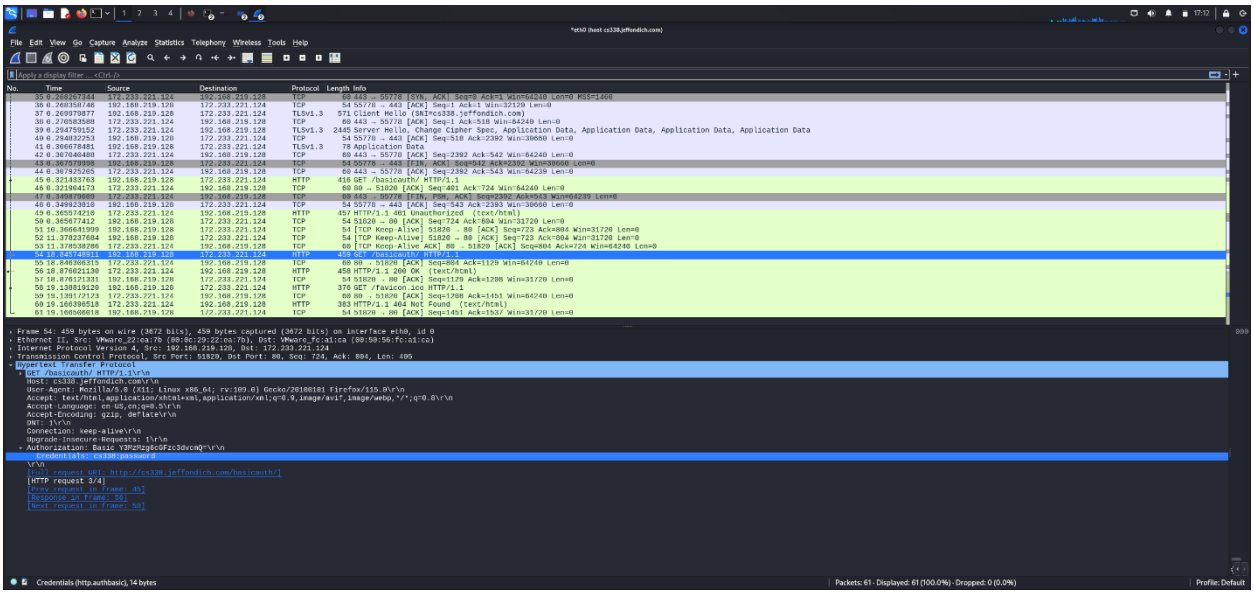
Most websites we visit nowadays want us to create some kind of account, often blocking access to content behind certain kinds of accounts. Certain webservers have easily configurable authentication abilities built in. In this case, we will be looking at Nginx.

Here, we can see our first attempt to go to the website http://cs338.jeffondich.com/basicauth/



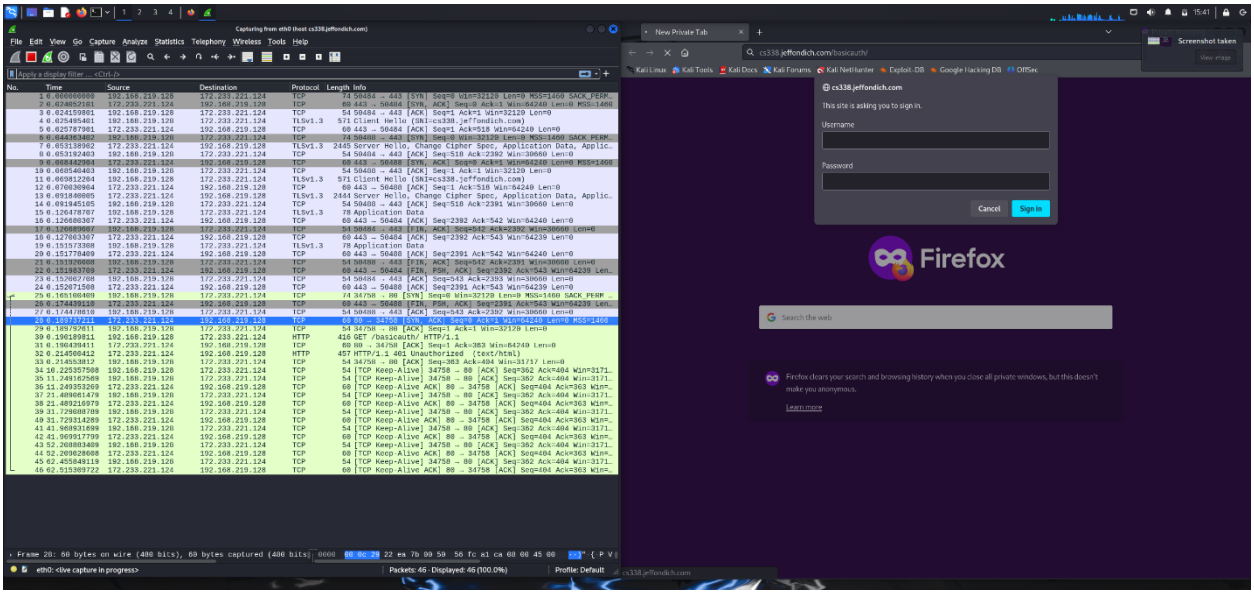
On the left, we have wireshark: a program that lets us see network traffic. That is, we can see what signals our computer is sending and receiving to the webserver that hosts cs338.jeffondich.com. When we try to go to http://cs338.jeffondich.com/basicauth/, we don't even have the chance to load the page before the password popup appears. In frames number 1-3 and 7-10, we can see our computer opening up TCP connections with the webserver. Normally, all of this information would be unencrypted because of we are using http to access the website instead of https. However, possibly do to the password prompt, these are secure.

Once we enter a password however, it is not send via an encrypted channel:

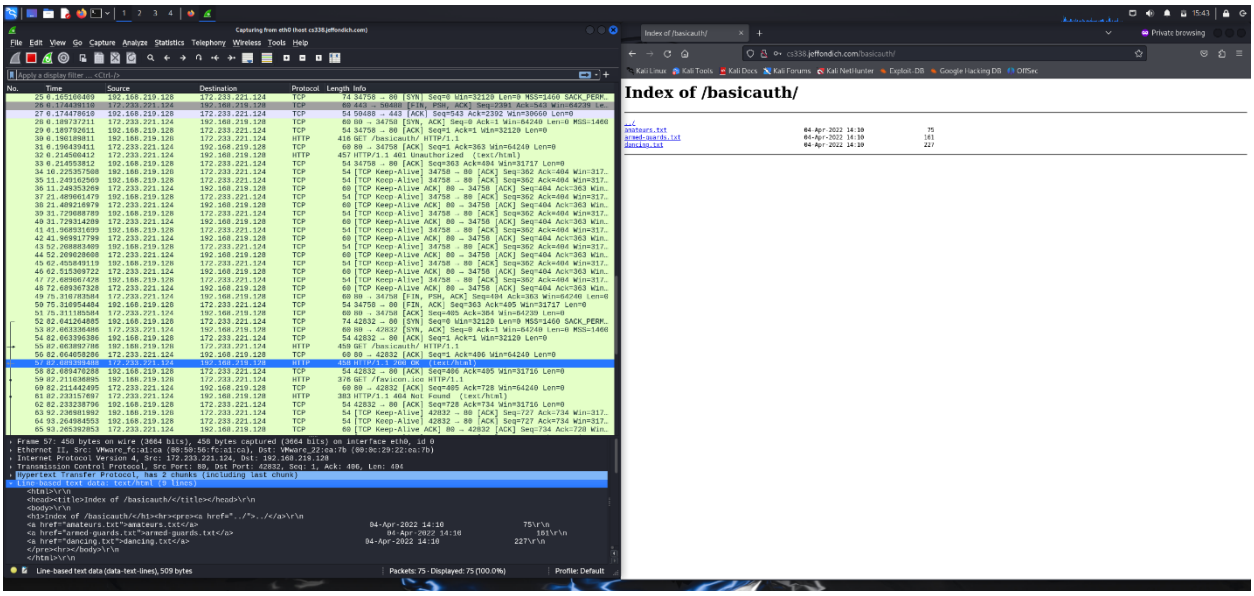


Here we can clearly see the username and password we used to access the website. Some research into Nginx shows that the basic authentication it provides only uses base64 encoding which is easily unencoded without any need for an encryption key.

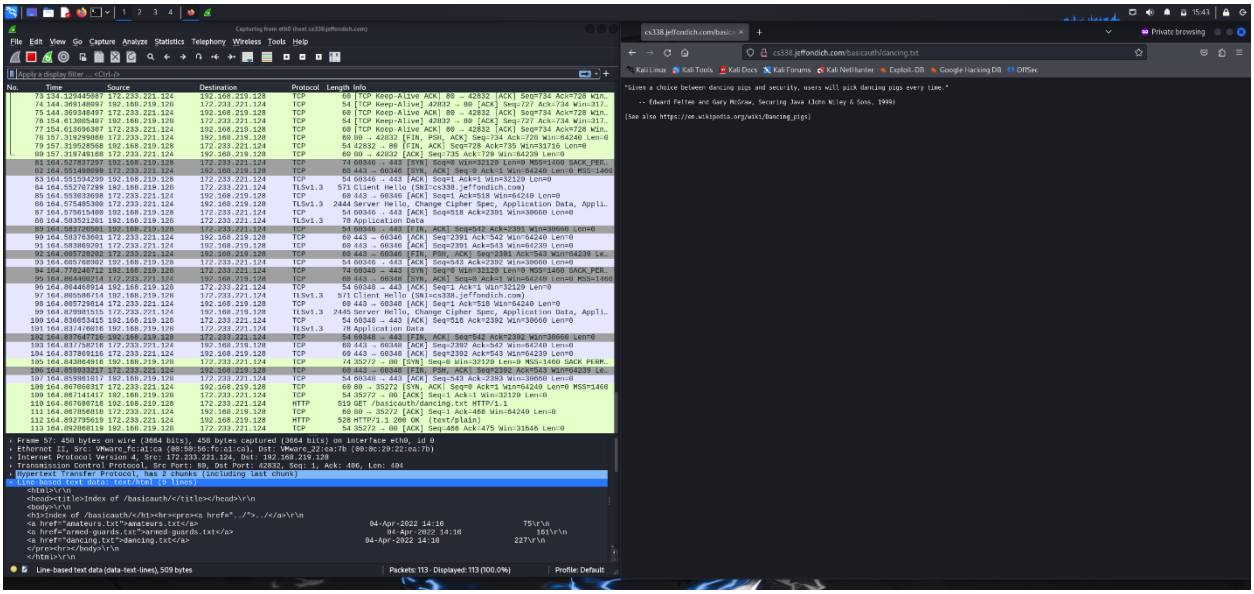
Attempts to access the website with incorrect credentials will get a reply from the server saying that access is unauthorized. See Frame 30 for an example.



Once we have successfully entered our credentials, we can access the desired page. On Frame 57, our HTTP GET request goes through.

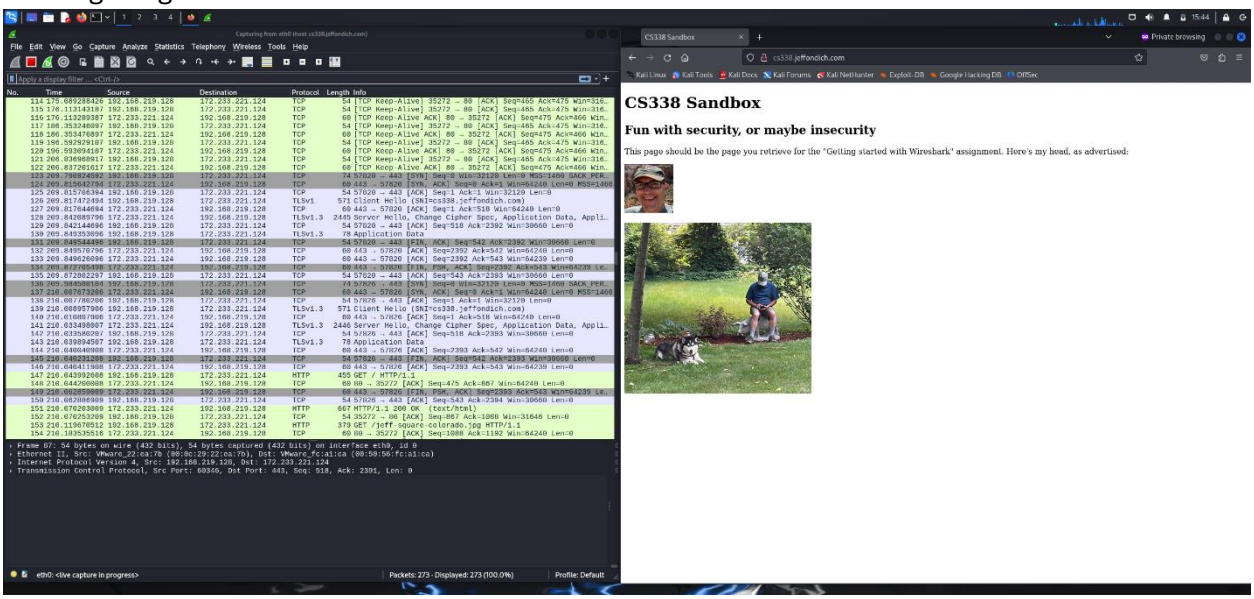


As we try to access other pages on the website, our credentials are checked again. Thankfully, our browser can do this for us, without having to reenter anything. The grey lines are a repeat of the same requests and responses that we got when first accessing the /basicauth path.



Each new page on the website goes through this process again. Interestingly, pressing the back arrow can avoid this . This likely means that the connection is remembered from out previous access of the page and there is no need to go back to the server to update the page. This is probably done through browser caching. Since private browsing windows promise that they are not caching, I assume that this is done only in memory, although browsers have been known to lie about not

caching things before.



Hopefully this illustrates some of the insecurity with Nginx's basic authentication and provides a good reason to use more secure methods of authentication. I am still unclear as to why there are some communications being secured with TLS but the password is not one of them. The application data sent over is always 78 bytes, so perhaps more investigation can be done into what is being sent.