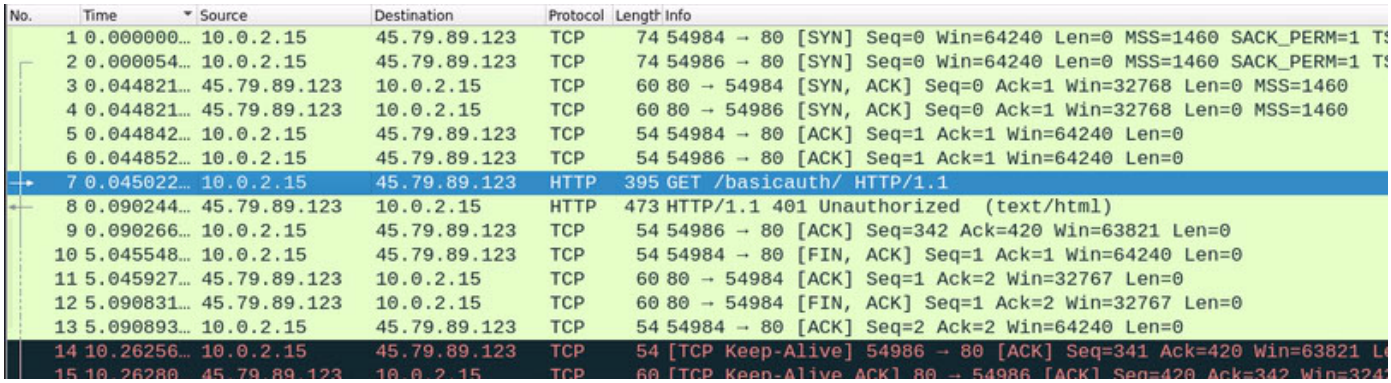
HTTP Insecurity

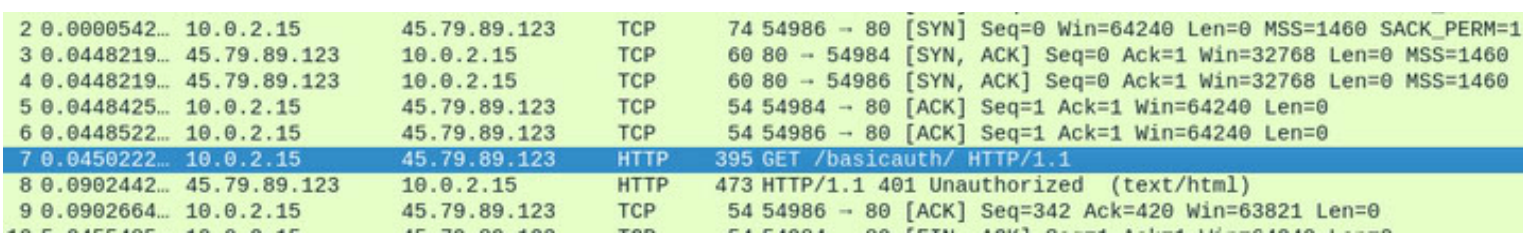
By Kaeden Berg and Yasmeen Awad

How secure is HTTP's basic authentication? What's this fancy HTTP*S*? Jeff set up a basic authentication on his personal website for us to hack around with, and we loaded up wireshark to see how secure our conversations with his server would be. First, we request the page. Our computer starts with a standard three way handshake on port 54984 followed by a GET request. Our first SYN seemed to time out, and our browser attempted to open a second connection on port 54986. We received SYN, ACK for both connections, so our browser closed 54984 and communicated through 54968 instead.



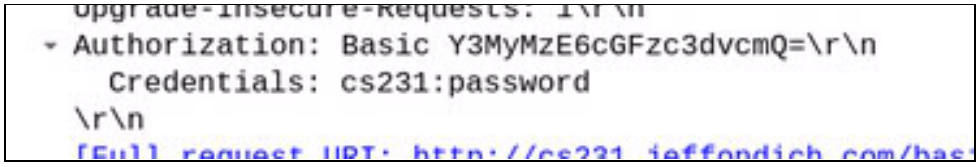
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000…	10.0.2.15	45.79.89.123	TCP	74	54984 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TS
2	0.000054…	10.0.2.15	45.79.89.123	TCP	74	54986 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TS
3	0.044821…	45.79.89.123	10.0.2.15	TCP	60	80 → 54984 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
4	0.044821…	45.79.89.123	10.0.2.15	TCP	60	80 → 54986 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
5	0.044842…	10.0.2.15	45.79.89.123	TCP	54	54984 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.044852…	10.0.2.15	45.79.89.123	TCP	54	54986 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	0.045022…	10.0.2.15	45.79.89.123	HTTP	395	GET /basicauth/ HTTP/1.1
8	0.090244…	45.79.89.123	10.0.2.15	HTTP	473	HTTP/1.1 401 Unauthorized (text/html)
9	0.090266…	10.0.2.15	45.79.89.123	TCP	54	54986 → 80 [ACK] Seq=342 Ack=420 Win=63821 Len=0
10	5.045548…	10.0.2.15	45.79.89.123	TCP	54	54984 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
11	5.045927…	45.79.89.123	10.0.2.15	TCP	60	80 → 54984 [ACK] Seq=1 Ack=2 Win=32767 Len=0
12	5.090831…	45.79.89.123	10.0.2.15	TCP	60	80 → 54984 [FIN, ACK] Seq=1 Ack=2 Win=32767 Len=0
13	5.090893…	10.0.2.15	45.79.89.123	TCP	54	54984 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
14	10.26256…	10.0.2.15	45.79.89.123	TCP	54	[TCP Keep-Alive] 54986 → 80 [ACK] Seq=341 Ack=420 Win=63821 Le
15	10.26280…	45.79.89.123	10.0.2.15	TCP	60	[TCP Keep-Alive ACK] 80 → 54986 [ACK] Seq=420 Ack=342 Win=324

Upon sending a GET request to /basicauth/, we were immediately sent a 401 response stating we were "unauthorized". The nginx server can authenticate users based on IP addresses, and since we aren't administrators and have never been to this website, we aren't on the list of approved IP addresses.



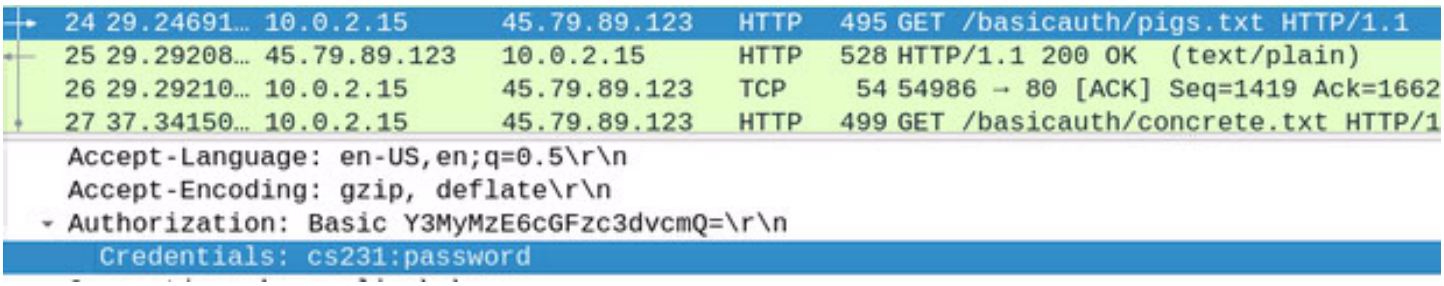
2	0.0000542…	10.0.2.15	45.79.89.123	TCP	74	54986 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
3	0.0448219…	45.79.89.123	10.0.2.15	TCP	60	80 → 54984 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
4	0.0448219…	45.79.89.123	10.0.2.15	TCP	60	80 → 54986 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
5	0.0448425…	10.0.2.15	45.79.89.123	TCP	54	54984 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.0448522…	10.0.2.15	45.79.89.123	TCP	54	54986 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	0.0450222…	10.0.2.15	45.79.89.123	HTTP	395	GET /basicauth/ HTTP/1.1
8	0.0902442…	45.79.89.123	10.0.2.15	HTTP	473	HTTP/1.1 401 Unauthorized (text/html)
9	0.0902664…	10.0.2.15	45.79.89.123	TCP	54	54986 → 80 [ACK] Seq=342 Ack=420 Win=63821 Len=0

The connection continues with a TCP Keep-Alive and another GET request for the page. This request finally allows us to send in our credentials. We type in the very secure credentials, username: cs231, password: password. And immediately we saw our own credentials in plain text!

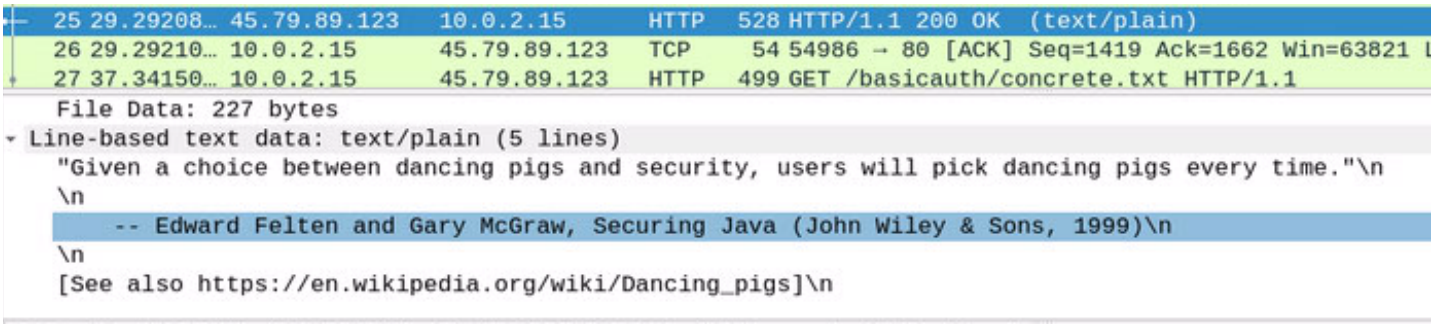


Now, username and password are actually encoded with Base64, that's the "Basic Y3MyMZE…" bit, but can easily be uncoded. Wireshark is able to display them in plain text for us right below. This code is sent to the browser, which seems to save the credentials and send them upon each request. We can see our password re-sent in this format whenever we request a new webpage. For example, see below when we send a GET request to /basicauth/pigs.txt, we were not prompted to enter the password, but the HTTP request still contains our credentials. We believe this is because the browser saves these credentials.

See how on line 24 below, we send a GET request for /basicauth/pigs.txt. We haven't entered our username and password, yet they show up as our authorization in our GET packet to the server. We can reasonably assume that the browser (the only other party involved in this transaction) is saving and then sending our credentials when needed.



We can see the html data through the session (as it is sent in plain text, rather than being encrypted, seen in the HTTP 200 OK response below), which defeats the purpose of needing credentials, as anyone listening in, regardless of whether they are authenticated or not, can view the pages being sent to our browser. In order to properly secure this system, the HTML text should be sent encrypted, and decrypted locally.



The two most important pieces of this system, the credentials and the private data, are both sent insecurely. This should not be surprising, as HTTP is a protocol focused on transferring data, it leaves the security to other protocols. The documentation states this, it warns that this system is insecure if not paired with “some external secure system”, such as the Transport Layer Security Protocol (TLS). If you package the two together, you would get HTTPS, which is essentially HTTP with encryption. HTTPS encrypts data before sending and decrypts upon receiving, so that anyone listening in without the proper key will only see gibberish.

In order to authenticate with HTTPS instead, you have to purchase and install a SSL certificate for your web hosting account. SSL stands for Secure Socket Layer, which is the protocol which was originally developed for secure HTTP communication, and has been replaced by TLS. These certificates are unique for each service, and have public and private keys that are used to encrypt this data. Carleton has HTTPS set up, but for this exercise Jeff went back to HTTP to allow us to see how insecure it is.