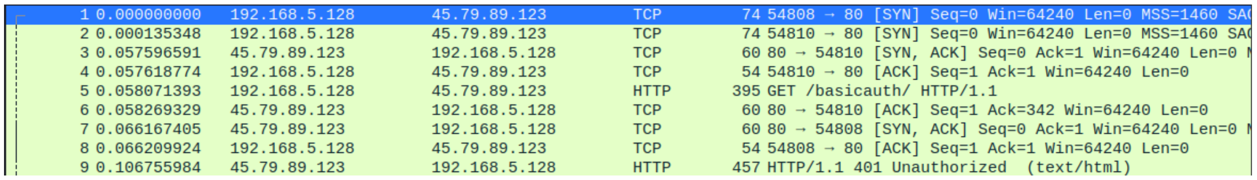
HTTP Basic Authentication Assignment

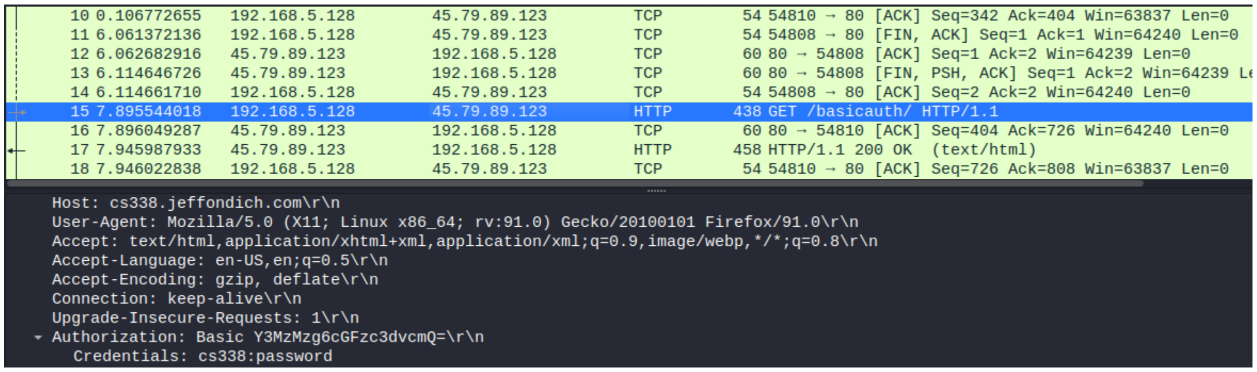
Ashok Khare
Collaborators: Emma Roskopf, Lily Haas (consulted with each other but submitted our own writeups)

One fateful day, the young computer programmer Ashok Khare sat down at his laptop to complete his basic authentication assignment. Booting up Kali and Wireshark, he set up his capture filter, then navigated to Jeff’s top-secret website and entered the super secure username and password. Seeing that Wireshark had captured the desired packets, he stopped the capture and got up to get a snack. Unfortunately, part of his soul was somehow torn off in the process and formed into the nefarious opportunist and all-around scoundrel Kohsa Erahk! Seeing Ashok distracted, Kohsa looked at the packets Wireshark had captured, hoping to glean something from them. This is what he found.



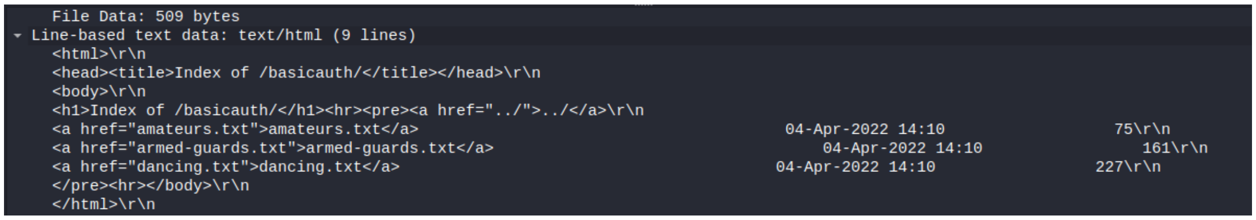
```
 1 0.000000000   192.168.5.128   45.79.89.123    TCP     74 54808 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
 2 0.000135348   192.168.5.128   45.79.89.123    TCP     74 54810 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
 3 0.057596591   45.79.89.123    192.168.5.128   TCP     60 80 → 54810 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 M
 4 0.057618774   192.168.5.128   45.79.89.123    TCP     54 54810 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
 5 0.058071393   192.168.5.128   45.79.89.123    HTTP   395 GET /basicauth/ HTTP/1.1
 6 0.058269329   45.79.89.123    192.168.5.128   TCP     60 80 → 54810 [ACK] Seq=1 Ack=342 Win=64240 Len=0
 7 0.066167405   45.79.89.123    192.168.5.128   TCP     60 80 → 54808 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 M
 8 0.066209924   192.168.5.128   45.79.89.123    TCP     54 54808 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
 9 0.106755984   45.79.89.123    192.168.5.128   HTTP   457 HTTP/1.1 401 Unauthorized  (text/html)
```

First, he noticed that two TCP connections to the server had been made: Connection A between client port 54808 and server port 80, and Connection B between client port 54810 and client port 80. He also noted the HTTP GET request sent over Connection B (as indicated by the Source and Destination TCP headers displayed in Wireshark). The request, for the /basicauth/ section of Jeff’s website, was acknowledged but rejected by the server in a 401 Unauthorized response. Looking into the HTTP headers displayed in Wireshark, Kohsa noted one that read, ‘WWW-Authenticate: Basic realm=”Protected Area”’. He recognized the line from the HTTP Basic Authentication documentation (https://datatracker.ietf.org/doc/html/rfc7617#section-1.1). The documentation clarified that ‘Basic’ meant that the authentication scheme was HTTP Basic Authentication, and the ‘realm’ argument was simply an assigned value to identify the space to be protected. Thus, Kohsa surmised that the web page’s content was locked behind an HTTP Basic Authentication login.



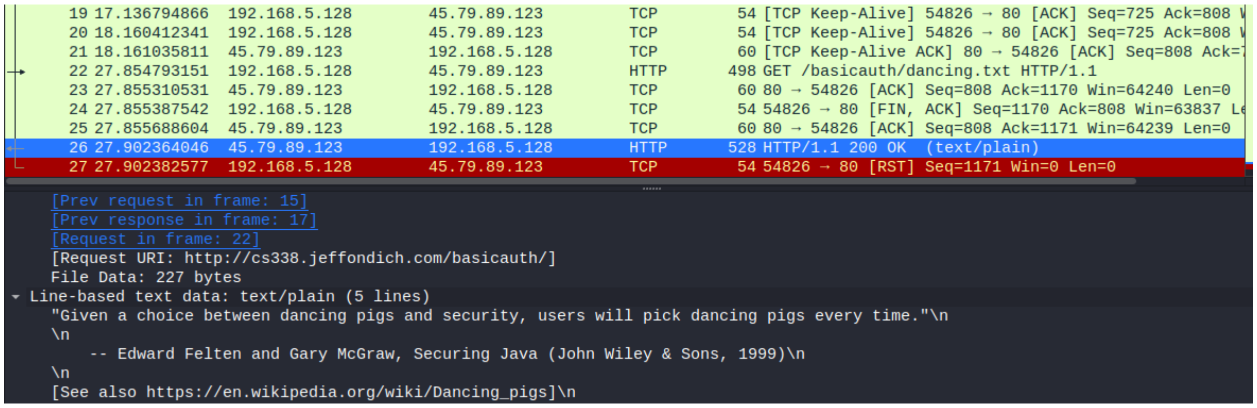
```
10 0.106772655   192.168.5.128   45.79.89.123    TCP     54 54810 → 80 [ACK] Seq=342 Ack=404 Win=63837 Len=0
11 6.061372136   192.168.5.128   45.79.89.123    TCP     54 54808 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
12 6.062682916   45.79.89.123    192.168.5.128   TCP     60 80 → 54808 [ACK] Seq=1 Ack=2 Win=64239 Len=0
13 6.114646726   45.79.89.123    192.168.5.128   TCP     60 80 → 54808 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Le
14 6.114661710   192.168.5.128   45.79.89.123    TCP     54 54808 → 80 [ACK] Seq=2 Ack=2 Win=64240 Len=0
15 7.895544018   192.168.5.128   45.79.89.123    HTTP   438 GET /basicauth/ HTTP/1.1
16 7.896049287   45.79.89.123    192.168.5.128   TCP     60 80 → 54810 [ACK] Seq=404 Ack=726 Win=64240 Len=0
17 7.945987933   45.79.89.123    192.168.5.128   HTTP   458 HTTP/1.1 200 OK  (text/html)
18 7.946022838   192.168.5.128   45.79.89.123    TCP     54 54810 → 80 [ACK] Seq=726 Ack=808 Win=63837 Len=0

    Host: cs338.jeffondich.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    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
      Credentials: cs338:password
```

Moving on, Kohsa noted the [FIN] flags being sent over Connection A. Upon seeing both systems sent [FIN] and [ACK] flags, Kohsa knew that TCP connection had been terminated, leaving just Connection B to analyze. Speaking of Connection B, he saw that another HTTP GET request was attempted over it. Upon investigating the HTTP headers attached to the request, Kohsa noticed a new one: Authorization. Looking at the corresponding value, Kohsa recognized its format from the HTTP Basic Authentication documentation (https://datatracker.ietf.org/doc/html/rfc7617#section-1.1). The documentation specified that the value for the header was created by combining a username, colon, and password, then encoding the combination in an octet sequence, and finally encoding the sequence into ASCII characters using base64. Kohsa assumed the ‘Basic’ part to just be an indicator that the Authorization value was an HTTP Basic input. This was all well and good, but it would take a lot of work to decode the login information. However, under the Authorization header was a sub-header labeled ‘Credentials’, and its value was shown in clear text to be cs338:password. Kohsa recognized this as the unencoded combined authentication value! He was jubilant! He knew the presence of this authorization information meant that there was indeed an HTTP Basic Authentication system involved with the web page, and Ashok had presumably entered his username and password into it. Then, the encoded and unencoded credentials had been sent by the browser to the server in the HTTP GET request. Now that he knew the username and password, Kohsa could now also log into Jeff’s top secret database! What a disaster!



```
    File Data: 509 bytes
▾ Line-based text data: text/html (9 lines)
    <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\n
    <a href="amateurs.txt">amateurs.txt</a>                                    04-Apr-2022 14:10                  75\r\n
    <a href="armed-guards.txt">armed-guards.txt</a>                                04-Apr-2022 14:10                 161\r\n
    <a href="dancing.txt">dancing.txt</a>                                      04-Apr-2022 14:10                 227\r\n
    </pre><hr></body>\r\n
    </html>\r\n
```

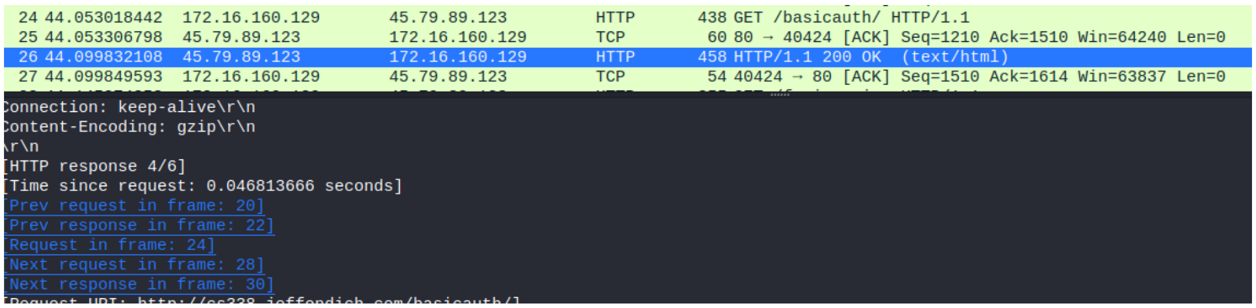
Now examining the server’s response to the HTTP request (in Frame 17, 200 OK), Kohsa noticed that the response message body contained the raw HTML of the /basicauth/ web page! Even worse! Now, by just observing the packets containing data for the pages on Jeff’s site that Ashok visited, Kohsa could also see the content of those pages, and how they were organized. No secret would be safe!



```
   19 17.136794866  192.168.5.128    45.79.89.123     TCP     54 [TCP Keep-Alive] 54826 → 80 [ACK] Seq=725 Ack=808 W
   20 18.160412341  192.168.5.128    45.79.89.123     TCP     54 [TCP Keep-Alive] 54826 → 80 [ACK] Seq=725 Ack=808 W
   21 18.161035811  45.79.89.123     192.168.5.128    TCP     60 [TCP Keep-Alive ACK] 80 → 54826 [ACK] Seq=808 Ack=
   22 27.854793151  192.168.5.128    45.79.89.123     HTTP   498 GET /basicauth/dancing.txt HTTP/1.1
   23 27.855310531  45.79.89.123     192.168.5.128    TCP     60 80 → 54826 [ACK] Seq=808 Ack=1170 Win=64240 Len=0
   24 27.855387542  192.168.5.128    45.79.89.123     TCP     54 54826 → 80 [FIN, ACK] Seq=1170 Ack=808 Win=63837 Le
   25 27.855688604  45.79.89.123     192.168.5.128    TCP     60 80 → 54826 [ACK] Seq=808 Ack=1171 Win=64239 Len=0
   26 27.902364046  45.79.89.123     192.168.5.128    HTTP   528 HTTP/1.1 200 OK  (text/plain)
   27 27.902382577  192.168.5.128    45.79.89.123     TCP     54 54826 → 80 [RST] Seq=1171 Win=0 Len=0

    [Prev request in frame: 15]
    [Prev response in frame: 17]
    [Request in frame: 22]
    [Request URI: http://cs338.jeffondich.com/basicauth/]
    File Data: 227 bytes
▾ Line-based text data: text/plain (5 lines)
    "Given a choice between dancing pigs and security, users will pick dancing pigs every time."\n
    \n
        -- Edward Felten and Gary McGraw, Securing Java (John Wiley & Sons, 1999)\n
    \n
    [See also https://en.wikipedia.org/wiki/Dancing_pigs]\n
```

Encouraged, Kohsa decided to keep at it. He saw several TCP Keep-Alive and Keep-Alive ACK packets, which he attributed to the connection between the client and server staying up, but with no data passing through it (basically leaving it in a waiting state). Then, he saw another HTTP GET request, this time for a file called ‘dancing.txt’ connected to the /basicauth/ page. Besides once again noting Ashok’s username and password clearly displayed in the Authentication (Credentials) header of the HTTP GET request, Kohsa also noted the raw contents of dancing.txt contained in the message body of the server's 200 OK response to the GET request. Again, simply by observing the data contained in the packets sent over the TCP connection, Kohsa was able to not only see which files Ashok viewed on Jeff's site, but also the explicit content of said files. Kohsa was frankly shocked at how easily he could view Jeff's secrets. He just had to read the packet details - he didn't even have to access the site himself! Finally, Kohsa saw the client send a [FIN, ACK] flag in frame 24, telling the server that it wanted to end communication. After the server's [ACK] response in the following frame, the client attempted to reset the connection, indicated by the [RST] flag in the final frame. Given that no more frames followed, Kohsa presumed that Ashok had seen all that he needed to and closed Jeff's website, cutting off the TCP connection.

Kohsa uncovered two significant security issues in his sinister spying. The first was the clear presence of the HTTP Basic Authentication information submitted in the GET request when Ashok entered his username and password. Having the credentials displayed in clear text is bad for obvious reasons - now Kohsa knows exactly what Ashok's username and password are - but having them displayed in their encrypted form is just as lethal, as Kohsa can decrypt it as long as he knows how credentials are encrypted with HTTP Basic Authentication. The second major security issue was the verbatim content of the web pages Ashok visited contained in the HTTP 200 OK responses to the GET requests. By viewing the content through the captured frames, Kohsa didn't even have to log into Jeff's website himself to see the information it held. The fact that Kohsa didn't have to access the website himself to see either the content or credentials was very problematic - it meant that anyone could see Jeff's secrets as long as they found some way to capture and access the packets sent to and from the server.



```
24 44.053018442  172.16.160.129   45.79.89.123     HTTP   438 GET /basicauth/ HTTP/1.1
25 44.053306798  45.79.89.123     172.16.160.129   TCP     60 80 → 40424 [ACK] Seq=1210 Ack=1510 Win=64240 Len=0
26 44.099832108  45.79.89.123     172.16.160.129   HTTP   458 HTTP/1.1 200 OK  (text/html)
27 44.099849593  172.16.160.129   45.79.89.123     TCP     54 40424 → 80 [ACK] Seq=1510 Ack=1614 Win=63837 Len=0
```

```
Connection: keep-alive\r\n
Content-Encoding: gzip\r\n
\r\n
[HTTP response 4/6]
[Time since request: 0.046813666 seconds]
[Prev request in frame: 20]
[Prev response in frame: 22]
[Request in frame: 24]
[Next request in frame: 28]
[Next response in frame: 30]
```

Addendum: Kohsa enlisted the help of collaborators Emma Roskopf and Lily Haas to further test the security of Jeff's website. They tried to submit several usernames and passwords using the Basic Authentication form, some of which were correct and others which were not. They thought that this might make it difficult for packet observers to figure out which username and password was the one that actually belonged to the user, but it turned out that this was not feasible. Wireshark logged the particular frame containing the HTTP request which prompted a given response (Unauthorized, OK, or another), which meant that a potential spy could see which HTTP request resulted in a particular response. Combined with the Authorization and Credentials headers in the GET request containing the inputted username and password for the login attempt, Kohsa realized he could check to see which exact GET request prompted the OK response, and then view the username and password submitted in that login attempt.