CS338 Assgnment 4

Kuo Wang

April 2022

When I paste the address and click "go" on the browser, the browser connects to a DNS server first to exchange the cs338.jeffondich.com domain name for an IP address: 45.79.89.123. Both an A and AAAA record requests are made, and both return the same result. AAAA records, for additional knowledge, is able to resolve IPv6 addresses as a newer version and eventual replacement of A records.

```
1   0.000000000    172.16.34.128 172.16.34.2   DNS    80     Standard query 0x39c9 A cs338.
    jeffondich.com
2   0.000071937    172.16.34.128 172.16.34.2   DNS    80     Standard query 0x9ec5 AAAA cs338.
    jeffondich.com
3   0.000746153    172.16.34.2   172.16.34.128 DNS    96     Standard query response 0x39c9 A
    cs338.jeffondich.com A 45.79.89.123
4   0.001683400    172.16.34.2   172.16.34.128 DNS    80     Standard query response 0x9ec5
    AAAA cs338.jeffondich.com
```

Now that the browser has the IP address, it initiates the standard DNS handshake with the given IP address. The first SYN TCP packet is sent from client port 50880 to server port 80.

```
5   0.001907226    172.16.34.128 45.79.89.123  TCP    74     50880    80 [SYN] Seq=0 Win=64240
    Len=0 MSS=1460 SACK_PERM=1 TSval=2223634628 TSecr=0 WS=128
```

Almost simultaneously, another TCP SYN packet is sent from port 50882 to port 80. Apparently the web browser sends multiple packets through different ports to increase load speed with parallelism.

```
6   0.007191050    172.16.34.128 45.79.89.123  TCP    74     50882    80 [SYN] Seq=0 Win=64240
    Len=0 MSS=1460 SACK_PERM=1 TSval=2223634634 TSecr=0 WS=128
```

Continuing on, the TCP handshake through both ports carry on. For the transactions between client port 50880 to server port 80, SYN is met with SYN,ACK from the server, then the client sends an ACK as response. A connection is established here.

```
7   0.092932245    45.79.89.123  172.16.34.128 TCP    60     80    50880 [SYN, ACK] Seq=0 Ack=1
     Win=64240 Len=0 MSS=1460
8   0.092964556    172.16.34.128 45.79.89.123  TCP    54     50880    80 [ACK] Seq=1 Ack=1 Win
    =64240 Len=0
```

The same goes for 50882 for initial connection with TCP.

```
6   0.007191050   172.16.34.128 45.79.89.123  TCP    74     50882    80 [SYN] Seq=0 Win=64240
    Len=0 MSS=1460 SACK_PERM=1 TSval=2223634634 TSecr=0 WS=128
11  0.100864836   45.79.89.123  172.16.34.128 TCP    60     80     50882 [SYN, ACK] Seq=0 Ack=1
     Win=64240 Len=0 MSS=1460
12  0.100890806   172.16.34.128 45.79.89.123  TCP    54     50882    80 [ACK] Seq=1 Ack=1 Win
    =64240 Len=0
```

Then, through 50880, the client posts an HTTP GET request to the server for the path /basicauth/. The server responds with a TCP ACK packet signaling that it received the packet (HTTP uses TCP as underlying protocol).

```
9   0.093187171   172.16.34.128 45.79.89.123  HTTP   395    GET /basicauth/ HTTP/1.1
10  0.093278765   45.79.89.123  172.16.34.128 TCP    60     80     50880 [ACK] Seq=1 Ack=342
    Win=64240 Len=0
```

Then, the server sends a HTTP protocol packet with status code 401 in its header to tell the browser that it is not authorized to get a response from the server.

```
13  0.245854333   45.79.89.123  172.16.34.128 HTTP   457    HTTP/1.1 401 Unauthorized (text/
    html)
```

The browser sees this, sends an ACK to signal the server, and asks me for credentials.

```
14  0.245885012   172.16.34.128 45.79.89.123  TCP    54     50880    80 [ACK] Seq=342 Ack=404
    Win=63837 Len=0
```

I key in the right credentials, and click submit. This triggers the browser to submit another HTTP GET request with an Authorization header...

```
21  3.564746561   172.16.34.128 45.79.89.123  HTTP   438    GET /basicauth/ HTTP/1.1
```

...which includes the credentials in base64 text. "Basic" lets any application involved in the process know that the HTTP authorization scheme is basic: just a pair of username/password. Then, "Y3MzMzg6cGFzc3dvcmQ=" is the combination itself in Base64 format, which can be decoded by anyone with a simple tool into the string "cs338:password" in the format of ¡username¿:¡password¿. The string is used by the server to verify identity and grant access. Finally, "\r\n" just signals the end of the header by a way outlined by RFC standards.

```
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
   Credentials: cs338:password
\r\n
```

Base64 encoding can be decoded into plain string by anyone; there is no secrets involved in the decoding process. This is a problem, because essentially the lack of end-to-end encryption means that whoever gets the package can know the credentials; therefore, the system is prone to packet sniffing exploits.

Anyway, the server now has a GET request from the client with the right credentials. The server sends to the client an ACK so it knows that the packet was received. Afterwards, as the server verified the credentials, my browser and I are let in through the invitation of an HTTP packet with status code 200 (OK) along with the data I requested. Of course, the browser reciprocated the HTTP packet with another ACK packet.

```
21  3.564746561   172.16.34.128 45.79.89.123  HTTP   438    GET /basicauth/ HTTP/1.1
22  3.565075098   45.79.89.123  172.16.34.128 TCP    60     80    50880 [ACK] Seq=404 Ack=726
    Win=64240 Len=0
23  3.650702440   45.79.89.123  172.16.34.128 HTTP   458    HTTP/1.1 200 OK (text/html)
24  3.650721025   172.16.34.128 45.79.89.123  TCP    54     50880    80 [ACK] Seq=726 Ack=808
    Win=63837 Len=0
```

The data is an HTML page that links to three other sites through hyperlinks. This is found in the HTTP data packet as Line-based text data.

Simultaneously, 50882 is also working in parallel. As the webpage is loaded, the favicon is requested by the browser through the 50882 port, which was kept alive. The favicon must not have been set up to require authorization, because 50882 neither asked for credential nor contain authorization header. Since there isn't a favicon, a RST is sent from the client after the HTTP 404 response.

```
25  3.738839869   172.16.34.128 45.79.89.123  HTTP   355    GET /favicon.ico HTTP/1.1
27  3.739008701   45.79.89.123  172.16.34.128 TCP    60     80    50882 [ACK] Seq=1 Ack=302
    Win=64240 Len=0
29  3.932877042   45.79.89.123  172.16.34.128 HTTP   383    HTTP/1.1 404 Not Found (text/html)
30  3.932905446   172.16.34.128 45.79.89.123  TCP    54     50882    80 [RST] Seq=303 Win=0
    Len=0
```

So now, I am at the web page which is successfully loaded. The client and the server regularly exchange TCP keep-alive packets to ensure that the connection exists and further actions are possible.

```
35  13.757264492  172.16.34.128 45.79.89.123  TCP    54     [TCP Keep-Alive] 50880  80 [ACK]
    Seq=725 Ack=808 Win=63837 Len=0
36  14.781214430  172.16.34.128 45.79.89.123  TCP    54     [TCP Keep-Alive] 50880  80 [ACK]
    Seq=725 Ack=808 Win=63837 Len=0
37  14.781522898  45.79.89.123  172.16.34.128 TCP    60     [TCP Keep-Alive ACK] 80  50880 [
    ACK] Seq=808 Ack=726 Win=64240 Len=0
```

When I click on a link, my browser sends to the server through port 50880 (client) to 80 (server) a new HTTP GET request. The GET request also contains the same authorization header as before, so access to the destination file is also restricted. As the first page finished loading, the browser reset the connection from its port 50880. I closed the tab, and that might have triggered the RST.

```
42  32.805532807  172.16.34.128 45.79.89.123  HTTP   499    GET /basicauth/amateurs.txt HTTP
    /1.1
44  32.805791820  45.79.89.123  172.16.34.128 TCP    60     80    50880 [ACK] Seq=808 Ack=1171
     Win=64240 Len=0
45  32.805791941  45.79.89.123  172.16.34.128 TCP    60     80    50880 [ACK] Seq=808 Ack=1172
     Win=64239 Len=0
46  32.911529940  45.79.89.123  172.16.34.128 HTTP   375    HTTP/1.1 200 OK (text/plain)
47  32.911564155  172.16.34.128 45.79.89.123  TCP    54     50880    80 [RST] Seq=1172 Win=0
    Len=0
```

Accessing all of amateurs.txt, dancing.txt, and armed-guards.txt follow the same procedure, just with a new connection established from a different client port to the same server port. The new connection itself went through the regular TCP handshake procedures to establish.

Afterwards, the processes are always in the order of client sending HTTP GET to the server with path to the desired file, server sends back ACK, server sends back HTTP response with code 200 and desired file, followed by client ACK. The connection was never reset.

As expected, TCP keep alive transactions are present and sporadic.

```
62  37.796176669   172.16.34.128 45.79.89.123  HTTP    503     GET /basicauth/armed-guards.txt
    HTTP/1.1
63  37.796432336   45.79.89.123  172.16.34.128 TCP     60      80     50922 [ACK] Seq=1129 Ack
    =1620 Win=64240 Len=0
64  37.910365110   45.79.89.123  172.16.34.128 HTTP    462     HTTP/1.1 200 OK (text/plain)
65  37.910383646   172.16.34.128 45.79.89.123  TCP     54      50922    80 [ACK] Seq=1620 Ack
    =1537 Win=63837 Len=0
66  38.492003610   172.16.34.128 45.79.89.123  TCP     54      [TCP Keep-Alive] 50924   80 [ACK]
    Seq=301 Ack=330 Win=63911 Len=0
67  39.159604036   172.16.34.128 45.79.89.123  HTTP    498     GET /basicauth/dancing.txt HTTP
    /1.1
68  39.159769722   45.79.89.123  172.16.34.128 TCP     60      80     50922 [ACK] Seq=1537 Ack
    =2064 Win=64240 Len=0
69  39.343699813   45.79.89.123  172.16.34.128 HTTP    528     HTTP/1.1 200 OK (text/plain)
70  39.343718959   172.16.34.128 45.79.89.123  TCP     54      50922    80 [ACK] Seq=2064 Ack
    =2011 Win=63837 Len=0
```

Apparantly, all three files have restricted access since the GET requests have authorization headers, and attempts at accessing them with no authorization proved futile.