

Names: PJ Sangvong , Luisa Escosteguy

## HTTP Basic Authentication

In this paper, we describe the interactions between the browser and cs338.jeffondich.com's nginx server, and detail how nginx password protection works.

### Sequence of events description

#### 1. DNS queries

When we go to <http://cs338.jeffondich.com/basicauth/> for the first time, we can see DNS standard queries. Those are issued to get the IP address of the domain name. The client (Kali) sends those DNS queries to the router's address (192.168.8.2), which responds back.

Source	Destination	Protocol	Length	Info
192.168.8.128	192.168.8.2	DNS	80	Standard query 0x782f A cs338.jeffondich.com
192.168.8.128	192.168.8.2	DNS	80	Standard query 0x0bcb AAAA cs338.jeffondich.com
192.168.8.2	192.168.8.128	DNS	390	Standard query response 0x782f A cs338.jeffondich.com
192.168.8.2	192.168.8.128	DNS	159	Standard query response 0x0bcb AAAA cs338.jeffondich.com

The DNS query response includes the IP address of our host: 45.79.89.123. The type A indication means that the hostname corresponds to an IPv4 address, and class IN refers to the internet. The type AAAA is related to the IPv6 address of the server.

```
▼ Answers
  ▼ cs338.jeffondich.com: type A, class IN, addr 45.79.89.123
    Name: cs338.jeffondich.com
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 5 (5 seconds)
    Data length: 4
    Address: 45.79.89.123
```

#### 2. TCP Handshake

The image below shows the TCP handshake between the client (Kali) and the server (<http://cs338.jeffondich.com/basicauth/>). On the first two lines, Kali is trying to connect to the server by sending a [SYN] packet. In the following two lines, the server is responding back to Kali by acknowledging the connection (through [ACK]) and also trying to establish a connection with Kali by sending a [SYN] packet. In the last two lines, Kali is responding back to the server by saying that the connection has been established ([ACK] packet).

192.168.8.128	45.79.89.123	TCP	74	43104 → 80	[SYN] Seq=0 Win=64240 Len=0 MSS=1460
192.168.8.128	45.79.89.123	TCP	74	43106 → 80	[SYN] Seq=0 Win=64240 Len=0 MSS=1460
45.79.89.123	192.168.8.128	TCP	60	80 → 43104	[SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
45.79.89.123	192.168.8.128	TCP	60	80 → 43106	[SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
192.168.8.128	45.79.89.123	TCP	54	43104 → 80	[ACK] Seq=1 Ack=1 Win=64240 Len=0
192.168.8.128	45.79.89.123	TCP	54	43106 → 80	[ACK] Seq=1 Ack=1 Win=64240 Len=0

The reason the frames double (2 [SYN], 2 [SYN, ACK], and 2 [ACK]) is that for each pair, one has a Complete status and another has an Incomplete status. This means that Kali and the server try to communicate with each other and fail on the first attempt, so they retry again and complete it on the second attempt.

```

Transmission Control Protocol, Src Port: 43106, Dst Port: 80
Source Port: 43106
Destination Port: 80
[Stream index: 1]
[Conversation completeness: Incomplete, DATA (15)]

Transmission Control Protocol, Src Port: 43104, Dst Port: 80
Source Port: 43104
Destination Port: 80
[Stream index: 0]
[Conversation completeness: Complete, NO_DATA (23)]

```

### 3. Displaying the page

After the TCP handshake, Kali sends a GET request for the page to be displayed. The server then responds with a [ACK] packet, saying that it has received the request and is about to send the response over. In the next frame, the server sends over the content that the user requests if the login credentials are valid; but here, since we are not yet authorized to look at the actual page, the server sends over a HTTP response with code 401(Unauthorized) and a login page for us to type in our username and password as the content.

11	0.139397187	192.168.8.128	45.79.89.123	HTTP	395 GET /basicauth/ HTTP/1.1
12	0.149558833	45.79.89.123	192.168.8.128	TCP	60 80 → 43106 [ACK] Seq=1 Ack=342 Win=64240 Len=0
13	0.196633715	45.79.89.123	192.168.8.128	HTTP	457 HTTP/1.1 401 Unauthorized (text/html)

```

Hypertext Transfer Protocol
HTTP/1.1 401 Unauthorized\r\n
Content-Type: text/html\r\n

```

### 4. After the password is typed by the user

Once we type the correct username and password, we are now authorized to access the content of the page. The authorization happens on the server-side. The browser does this by sending an encrypted username and password over to the server (more on this below) along with the GET request for the homepage. If the credentials are correct, the server sends back the content of the page. We also see some ACK packets in between, which just means that the server/client received the requests/responses.

```

19 8.965676969 192.168.8.128 45.79.89.123 HTTP 438 GET /basicauth/ HTTP/1.1
20 8.966271485 45.79.89.123 192.168.8.128 TCP 60 80 → 43106 [ACK] Seq=404 Ack=726
21 9.019693323 45.79.89.123 192.168.8.128 HTTP 458 HTTP/1.1 200 OK (text/html)
22 9.019746985 192.168.8.128 45.79.89.123 TCP 54 43106 → 80 [ACK] Seq=726 Ack=808

\r\n
[HTTP response 2/3]
[Time since request: 0.054016354 seconds]
[Prev request in frame: 11]
[Prev response in frame: 13]
[Request in frame: 19]
[Next request in frame: 23]
[Next response in frame: 25]
[Request URI: http://cs338.jeffondich.com/basicauth/]
HTTP chunked response
Content-encoded entity body (gzip): 205 bytes -> 509 bytes
File Data: 509 bytes
e-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>
<a href="armed-guards.txt">armed-guards.txt</a>
<a href="dancing.txt">dancing.txt</a>
</pre><hr></body>\r\n
</html>\r\n
04-Apr-2022 14:10
04-Apr-2022 14:10
04-Apr-2022 14:10

```

## 5. Extra: When the credentials are wrong

We also observed what happens when we typed the wrong login and password. The first part is the same: the client sends a GET request to the host to get the page /basicauth/, and as it is not authorized, the server responds with the login page. Once we type a wrong credential, the client sends the same GET request to the host, but this time includes the Authorization header with a wrong credential. Again, the server acknowledges, but since the credential is incorrect, the server responds back with the HTTP response with code 401(Unauthorized) and sends back the login page again.

```

5376159 192.168.8.128 45.79.89.123 HTTP 395 GET /basicauth/ HTTP/1.1
5858270 45.79.89.123 192.168.8.128 TCP 60 80 → 43118 [ACK] Seq=1 Ack=342
3339648 45.79.89.123 192.168.8.128 HTTP 457 HTTP/1.1 401 Unauthorized (tex
3362349 192.168.8.128 45.79.89.123 TCP 54 43118 → 80 [ACK] Seq=342 Ack=40
434548 192.168.8.128 45.79.89.123 HTTP 434 GET /basicauth/ HTTP/1.1
2381752 45.79.89.123 192.168.8.128 TCP 60 80 → 43118 [ACK] Seq=404 Ack=72
0729840 45.79.89.123 192.168.8.128 HTTP 457 HTTP/1.1 401 Unauthorized (tex
0760147 192.168.8.128 45.79.89.123 TCP 54 43118 → 80 [ACK] Seq=722 Ack=80
5626880 192.168.8.128 45.79.89.123 TCP 54 43116 → 80 [FIN, ACK] Seq=1 Ack

Hypertext Transfer Protocol
▶ GET /basicauth/ HTTP/1.1\r\n
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 ZGV30mZld2Z3ZQ==\r\n
Credentials: dew:fewfwe

```

## "Authorization" header

When we typed the password and pressed “sign-in”, the client sent a GET request that contained the Authorization header with the password encrypted in base 64 (the hash after “Basic”). We verified that this is the correct encryption by typing Y3MzMzg6cGFzc3dvcmQ= in

the decode section of this [Base64 Decoder](#) website. Since base64 is standardized encryption, the client doesn't have to tell the server what the encryption key is. Both the server and client just have to know that the credentials are being sent over with base64 encryption. This is probably a standardized protocol for how credentials are supposed to be sent over through HTTP.

```
.79.89.123      HTTP      446 GET /basicauth/ HTTP/1.1
*****
▶ Transmission Control Protocol, Src Port: 43078, Dst Port: 80, Seq: 3050, Ack: 3148,
▼ Hypertext Transfer Protocol
  ▶ GET /basicauth/ HTTP/1.1\r\n
    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
    DNT: 1\r\n
  ▼ Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
    Credentials: cs338:password
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
```

Throughout this exercise, we have observed how the Authentication Credentials are used in the header of HTTP requests. We saw in action how the server would only return a protected page when the GET request for the page has valid credentials inside the Authorization Header field. Finally, we observed all three steps of the TCP handshake.

## Citations

[https://nginx.org/en/docs/http/nginx\\_http\\_auth\\_basic\\_module.html](https://nginx.org/en/docs/http/nginx_http_auth_basic_module.html)