Nathan Hayes-Rich and Jake Martens, basic-authentication.pdf, 4/8/2021

Frames 1-7: The Three-way Handshake
In the first seven frames the client sends two messages to the server to initiate the SYN, ACK sequence and establish a connection. The server responds with SYN, ACK to acknowledge that this message has been received and to open a connection in the opposite direction. Finally, the client responds with an ACK to accept, and it is now possible to exchange information between the two. These frames are provided below:

```
    1 0.000000000   172.18.210.251     45.79.89.123      TCP    74    34790 → 80 [SYN]
    2 0.000021400   172.18.210.251     45.79.89.123      TCP    74    34792 → 80 [SYN]
    3 0.044377000   45.79.89.123       172.18.210.251    TCP    74    80 → 34792 [SYN,
ACK]
    4 0.044407200   172.18.210.251     45.79.89.123      TCP    66    34792 → 80 [ACK]
    6 0.044552400   45.79.89.123       172.18.210.251    TCP    74    80 → 34790 [SYN,
ACK]
    7 0.044560900   172.18.210.251     45.79.89.123      TCP    66    34790 → 80 [ACK]
```

Frames 5-9: HTTP GET Request and Response
In frame 5, the client sends a message to the server requesting authentication to access the webpage. During our observation, this happened before receiving the final ACK in the three-way handshake. The server then responds with ACK to acknowledge the request, followed by an HTTP response from the server in frame 9. Frame 9 contains a 401 error message, which indicates that the client is not authorized to access the webpage.

```
    5 0.044526000   172.18.210.251     45.79.89.123      HTTP    415    GET /basicauth/
HTTP/1.1
    8 0.089194700   45.79.89.123       172.18.210.251    TCP     66     80 → 34792 [ACK]
    9 0.089568400   45.79.89.123       172.18.210.251    HTTP    485    HTTP/1.1 401
```

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.14.0 (Ubuntu)
Date: Wed, 07 Apr 2021 21:18:03 GMT
Content-Type: text/html
Content-Length: 204
Connection: keep-alive
WWW-Authenticate: Basic realm="Protected Area"
```

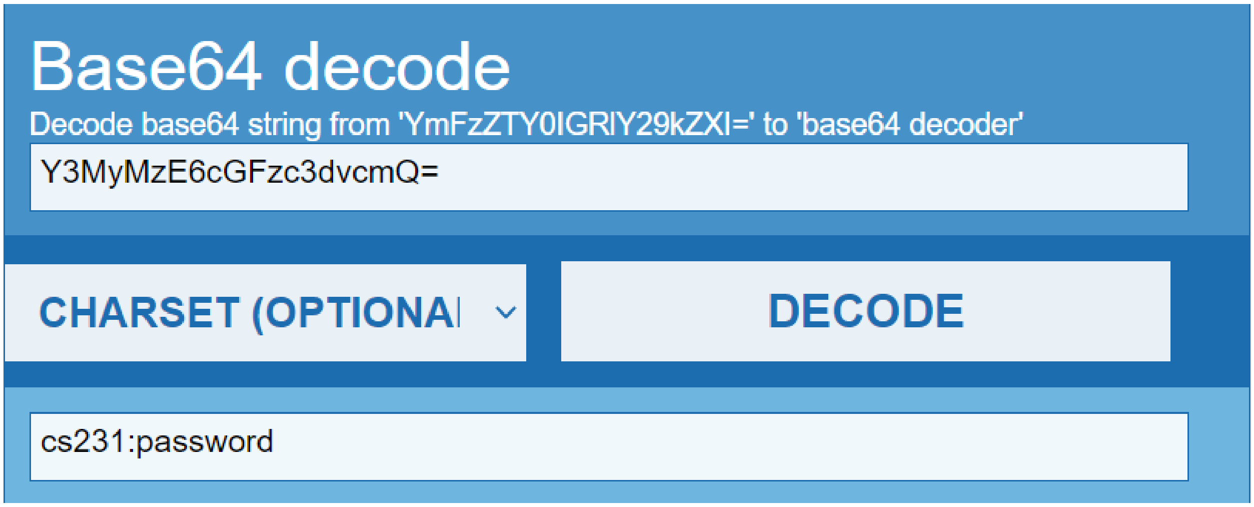
Above is the HTTP message included in frame 9 (excluding the message body). The Basic keyword refers to the authentication scheme being used here. The realm refers to the page that requires login credentials for access. The charset parameter is not used in this instance.

Frames 10-12: Client Login Credentials

Having received the error message from the server, the client acknowledges its receipt in frame 10. From the perspective of the user attempting to access this page, the browser opens a separate login prompt box that contains two text boxes for the user to input a username and password (Step 1 on page 5 of RFC 7617). After receiving the username and password, the browser concatenates them with a colon in between, and then encodes them into base64 (steps 2, 3 and 4 on page 5 of RFC 7617). In frame 11, the client sends an HTTP response containing the text inputted by the user with the same GET /basicauth/ message as in frame 5. In Wireshark, we can observe the contents of what the client sends the server, which includes the encoded username and password. This text is highlighted below next to “Authorization: Basic”. Wireshark automatically decodes this string for the observer, and an online decoder further confirms that the original text, in this case, reads “cs231:password”. The protocol does not encrypt these credentials, so any observer can relatively easily decode this information.

```
10 0.089579600    172.18.210.251       45.79.89.123        TCP     66     34792 → 80 [ACK]
11 4.974754400    172.18.210.251       45.79.89.123        HTTP     458    GET /basicauth/
HTTP/1.1
```

```
        GET /basicauth/ HTTP/1.1
        Host: cs231.jeffondich.com
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        Authorization: Basic Y3MyMzE6cGFzc3dvcmQ=
```



Base64 decode

Decode base64 string from 'YmFzZTY0IGRlY29kZXI=' to 'base64 decoder'

Y3MyMzE6cGFzc3dvcmQ=

cs231:password

Base64 decode courtesy of https://www.base64decode.net/

The protocol uses base 64 not as a security protocol, but to convert characters that are incompatible with HTTP into a format that is compatible. As RFC 7617 states in section 4, the

Basic Authentication Scheme is not for security purposes, but should be used alongside some other system that can encrypt the data. (https://stackoverflow.com/questions/4070693/what-is-the-purpose-of-base-64-encoding-and-why-it-used-in-http-basic-authentica)

Once the server receives the message containing the user-inputted login information from the client, it responds with an ACK message to confirm. At this point, the server has not yet responded to whether the information is correct.

```
12 5.019654700    45.79.89.123         172.18.210.251       TCP      66     80 → 34792 [ACK]
```

Frame 13: Server Response to HTTP Request

In frame 13, the server responds to the client with the actual HTTP response (200 OK), indicating that the client request was processed successfully. Thus, the server received the base64 encoded password from the client, decoded and successfully compared it with the one stored locally to the server. The HTTP headers contain basic information about the response, including the encoding of the HTML data. Included in the message body of the HTTP response is the actual HTML text of the webpage, encoded into several octets. Highlighted below are the raw chunk data transferred from the server. Having received the 200 OK and chunked data, the client decodes the chunked data into the actual lines representing the webpage and displays it to the user.

```
13 5.020644300    45.79.89.123         172.18.210.251       HTTP     487    HTTP/1.1 200 OK (text/html)
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Wed, 07 Apr 2021 21:07:30 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Content-Encoding: gzip
```

```
0100   0d 0a 1f 8b 08 00 00 00 00 00 00 03 ad 91 c1 0a
0110   c2 30 0c 86 ef 82 ef 50 7a 77 59 27 db 54 62 ef
0120   1e 7c 88 6e cb ec 60 5a a9 11 f5 ed 5d a7 c2 64
0130   08 0a f6 50 9a 34 7f 3e fe 04 2d ef 5b 3d 9d a0
0140   25 53 69 e4 86 5b d2 9b 43 45 57 e1 6a 01 85 39
0150   35 a5 39 b3 05 84 c7 1f 42 5f d9 29 0a 57 dd 44
0160   b1 2b 5d eb fc 5a 5e 6c c3 24 fb 4e ea 43 03 ab
0170   34 5a af f1 e8 49 a3 11 d6 53 bd 96 51 04 52 77
0180   17 82 09 e2 57 da ec 0d d3 d9 9f 22 be b2 d4 c3
0190   28 14 8a ef ce 3c 9e 6d 8d 9f 25 71 a2 84 ca 56
```

```
01a0   e9 62 2c cb d3 01 b4 74 87 d2 13 d3 03 3a 8c fe
01b0   0a 55 99 1a 40 8f cd ee e9 f2 f5 fa 01 16 0c bd
01c0   b9 cc 57 f1 72 e4 32 49 f2 0e 08 fd dc c3 02 20
01d0   ac 2e 4c 1b 9e db bf 03 a7 21 72 8a 06 02 00 00
01e0   0d 0a 30 0d 0a 0d 0a
```

Frame 14: Client Acknowledgement
In frame 14, the client acknowledges to the server that it has received the HTTP 200 OK.

```
14 5.020656500    172.18.210.251       45.79.89.123         TCP      66     34792 → 80 [ACK]
```

Frames 15-18: Client Request / Server Response for favicon
In frames 15-18 (which are omitted here as these frames are unrelated to the authentication process itself) the client sends an HTTP GET request for the favicon.ico. The server acknowledges the request and sends an HTTP 404 response (indicating to the client that the resource does not exist). The client then acknowledges the response.

Frames 19-24: FIN ACK Sequence
Finally, having either closed the browser or tab locally, the client initiates a FIN, ACK sequence for both of its open TCP connections (Frames 19 and 22). The server responds appropriately (FIN, ACK) to each (Frames 20 and 23, respectively), and the client acknowledges that the connection has been closed in each case (Frames 21 and 24, respectively).

```
    19 5.796545500    172.18.210.251       45.79.89.123         TCP      66     34790 → 80 [FIN,
ACK]
    20 5.841221300    45.79.89.123         172.18.210.251       TCP      66     80 → 34790 [FIN,
ACK]
    21 5.841261200    172.18.210.251       45.79.89.123         TCP      66     34790 → 80 [ACK]
    22 13.814131700   172.18.210.251       45.79.89.123         TCP      66     34792 → 80 [FIN,
ACK]
    23 13.858763400   45.79.89.123         172.18.210.251       TCP      66     80 → 34792 [FIN,
ACK] Seq=1188 Ack=1003 Win=64256 Len=0 TSval=2672284210 TSecr=1014769850
    24 13.858790300   172.18.210.251       45.79.89.123         TCP      66     34792 → 80 [ACK]
Seq=1003 Ack=1189 Win=64128 Len=0 TSval=1014769895 TSecr=2672284210
```