

Assignment 3 - HTTP Basic Authentication - Noah Lee

This paper goes into detail about my observations of HTTP, TLS and TCP traffic on the <http://cs338.jeffondich.com/basicauth> web server using network traffic on my browser, wireshark, burpsuite, curl and wget. This is the exact order in which I used these resources to investigate the connections. The majority of my findings is taken from the wireshark section - in which I discuss all the packets and steps in more detail.

Viewing the network traffic on the browser:

The screenshot shows a web browser window with the address bar displaying 'cs338.jeffondich.com/basicauth/'. The page content is titled 'Index of /basicauth/' and lists three files: [../](#), [amateurs.txt](#), [armed-guards.txt](#), and [dancing.txt](#). To the right of the file names, there is a table with three columns: 'Date', 'Size', and 'Time'. The table contains three rows of data:

Date	Size	Time
04-Apr-2022 14:10	75	
04-Apr-2022 14:10	161	
04-Apr-2022 14:10	227	

The Chrome DevTools Network tab is open, showing a list of requests. The first two requests are marked as 'failed' and the third is successful. The table below summarizes the requests:

Name	Status	Type	Initiator	Size	Time
basicauth/	(failed)...	docum...	Other	215 B	54 ms
basicauth/	(failed)...	docum...	Other	215 B	29 ms
basicauth/	200	docum...	Other	404 B	29 ms

At the bottom of the DevTools window, there is a message: 'Enhanced 'Never pause here''. The message states: 'The 'Never pause here' option can now "cancel" DOM, XMLHttpRequest, CSP violation breakpoints, exceptions and promise rejections from built-in functions, and more.'

It appears that the client is attempting to load the page - and requests the HTML from the server. However, maybe because a 'AUTH' flag is not true, the server is not sending the HTML. This can be seen in the second failed basicauth/ request when I intentionally but an incorrect password and a similar error to when the page was opened occurred. Only when I input the correct authentication details, was the HTML finally send to the client's browser.

Name	X	Headers	Preview	Response	Initiator	>>
✖ basicauth/						
✖ basicauth/						
📄 basicauth/						
		▼ General				
		Request URL:	http://cs338.jeffondich.com/basicauth/			
		Request Method:	GET			
		Status Code:	🔴 401 Unauthorized			
		Remote Address:	172.233.221.124:80			
		Referrer Policy:	strict-origin-when-cross-origin			
		▼ Response Headers <input type="checkbox"/> Raw				
		Connection:	keep-alive			
		Content-Length:	590			
		Content-Type:	text/html			
		Date:	Sun, 22 Sep 2024 14:47:08 GMT			
		Server:	nginx/1.18.0 (Ubuntu)			
		Www-Authenticate:	Basic realm="Protected Area"			
3 requests	834 B transferred					

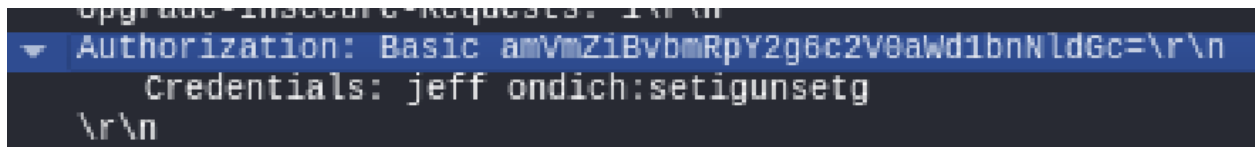
Name	X	Headers	Preview	Response	Initiator	>>
✖ basicauth/						
✖ basicauth/						
📄 basicauth/						
		▼ General				
		Request URL:	http://cs338.jeffondich.com/basicauth/			
		Request Method:	GET			
		Status Code:	🟢 200 OK			
		Remote Address:	172.233.221.124:80			
		Referrer Policy:	strict-origin-when-cross-origin			
		▼ Response Headers <input type="checkbox"/> Raw				
		Connection:	keep-alive			
		Content-Encoding:	gzip			
		Content-Type:	text/html			
		Date:	Sun, 22 Sep 2024 14:47:14 GMT			
		Server:	nginx/1.18.0 (Ubuntu)			
		Transfer-Encoding:	chunked			
3 requests	834 B transferred	▼ Request Headers <input type="checkbox"/> Raw				

These were the details of the HTTP requests in more detail. We can see that they were GET requests to <http://cs338.jeffondich.com/basicauth/>. And, when the page started or there was an incorrect password, we got the status code 401 Unauthorized - preventing the GET request from going through. Everything else looks the same except for the content-length (which is gzip for authenticated) and there is a Www-Authenticate response header compared to a Transfer-Encoding header in the authenticated request.

Using WireShark:

- TCP handshake:
 - The initial communication between the browser (client) and the server happens over HTTP, as shown in the logs. The client sends [SYN] packets to the server to initiate the connection (TCP 3-way handshake).
- GET request 301 redirection
 - After the TLS session is established, the browser attempts to access a resource via HTTP (*/basicauth*). The server responds with a '**301 Moved Permanently**' message, perhaps this is happening because I missed the '/' at the end of the URL.
 - HTTP: GET /basicauth HTTP/1.1
 - HTTP: HTTP/1.1 301 Moved Permanently
- Request blocked by authorization
 - The browser makes another GET request to /basicauth but this time over HTTPS. Since this /basicauth is protected, the server responds with a '**401 Unauthorized**' message, indicating that authentication is required first to continue HTML transfer.
 - HTTP: GET /basicauth/ HTTP/1.1
 - HTTP: HTTP/1.1 401 Unauthorized
- Putting in the wrong credentials
 - I put the wrong credentials again on purpose to see what would happen.

- After the '**401 Unauthorized**' response, the browser prompts the user for a username and password. Once the user enters the credentials, the browser sends a new GET request to the same resource. This time, the request includes an **Authorization** header that contains the Base64-encoded username and password.
- This aligns with the authentication reading which claims that basic authentication HTTP schemes transmits credentials as user-id/password pairs - encoded with Base-64. This is obviously not secure as someone could just decode the information easily, but paired with TLS as it is in this example, this supposedly makes it more secure - however I can still see this credential information on wireshark - which is not great from a security point of view - I am curious as to why TLS seems to be not working for my wireshark project (and I will show how it doesn't work later on).



A screenshot of a Wireshark packet capture. The packet list on the left shows a packet with a blue highlight. The packet details pane on the right shows the 'Authorization' header expanded, displaying 'Basic amVmZiBvbmRpdY2g6c2V8aWd1bnNldGc=\r\n'. Below this, the 'Credentials' field is expanded, showing 'jeff ondich:setigunsetg\r\n'.

- I believe 'jeff ondich: setigunsetg' is the exact text that I put into my browser - and that this is a translation of the Base64 text that follows 'Basic'.
- The server decoded the credentials and responded with the same '**401 Unauthorized**' packet as before. Upon giving correct credentials, the server responded with '**HTTP/1.1 200 OK**'
- **About the authentication header:**
 - We can see that the authentication header is composed by the web browser by concatenating userID and password into a single string 'userID:password'.
 - The browser then encodes this with Base64 to get an unintelligible string.
 - It is then appended to the packet with the authorization header.
 - The Base64 encoding is not a security measure and only to make it possible to transport binary data.
 - TLS is meant to encrypt this header - preventing hackers from getting credential information - it is viewable in wireshark perhaps because wireshark has access to the decryption keys.
 - The web server does the decoding and responds with 401 (if not correct) or 200 (if correct).

5. Clicking on a link in the server

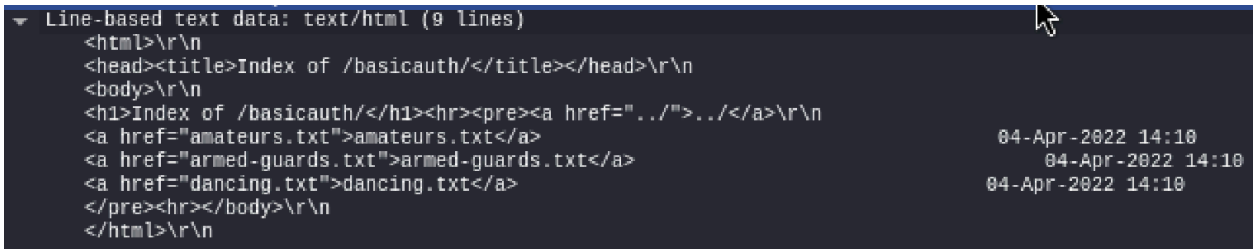
- When clicking on a link in the secure server such as 'GET /basicauth/amateurs.txt', we can see that there is still an authentication header that persists in the HTTP packet. Perhaps this authentication header is stored on the user's cache because when the site is reloaded on the browser, no authentication is needed again and the header is still there.

6. About encryption (TLS) - IGNORE FOR NOW I HAD HTTPS IN MY URL BUT THIS IS MY INITIAL THOUGHTS ON ENCRYPTING DATA

- Throughout the logs, many TLS calls back and forth between the browser and the web server can be seen - initiated by the web browser with 'Client Hello' (meaning the client

is sending the TLS encryption key(s)) and responded by the server with 'Server Hello' (server is agreeing to the encryption key). A new TLS connection seems to be created everytime a new resource is loaded like the HTML of /basicauth and the HTML of the other links like 'basicauth/amateurs.txt'. From what I see online, this may be the client and the browser sending keys and formalizing the encryption algorithm with each other (<https://www.internetsociety.org/deploy360/tls/basics/>).

- Even though the data is encrypted, I am confused as to why I can still see the data in the payload, given that it should be encrypted.

A screenshot of a Wireshark packet capture window. The top bar indicates 'Line-based text data: text/html (9 lines)'. The main pane displays the raw HTML content of a packet. The HTML includes a title 'Index of /basicauth/', a body with a heading 'Index of /basicauth/', and a list of links: 'amateurs.txt', 'armed-guards.txt', and 'dancing.txt'. To the right of the HTML content, three timestamps are visible: '04-Apr-2022 14:10' repeated three times.

```
<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>\r\n<a href="armed-guards.txt">armed-guards.txt</a>\r\n<a href="dancing.txt">dancing.txt</a>\r\n</pre><hr></body>\r\n</html>\r\n
```

- I still see this header and all the payload data, maybe WireShark is doing something under the hood to decrypt it for me or intercepting the packets before they can be encrypted.

Conclusion

- The password is sent from the browser to the server, it is Base64-encoded.
- The browser does not perform the password checking itself; the password is sent to the server for verification.
- The encryption (if it were to happen) during the transmission is handled by the TLS protocol, which encrypts all communication between the client and server after a successful TLS handshake.

Using Burpsuite:

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes
1	http://cs338.jeffondich.com	GET	/basicauth			301	400	HTML		301 Moved Permanently	
2	http://cs338.jeffondich.com	GET	/basicauth/			401	805	HTML		401 Authorization Req...	
3	http://cs338.jeffondich.com	GET	/basicauth/			401	805	HTML		401 Authorization Req...	
4	http://cs338.jeffondich.com	GET	/basicauth/			200	666	HTML		Index of /basicauth/	
5	http://cs338.jeffondich.com	GET	/favicon.ico			404	728	HTML	ico	404 Not Found	
6	http://cs338.jeffondich.com	GET	/basicauth/amateurs.txt			200	321	text	txt		

Request
Pretty Raw Hex
1 GET /basicauth/ HTTP/1.1
2 Host: cs338.jeffondich.com
3 Cache-Control: max-age=0
4 Authorization: Basic bm9haGx1ZTpwYXNzd29yZA==
5 Accept-Language: en-US
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
9 Accept-Encoding: gzip, deflate, br
10 Connection: keep-alive
11
12

Response
Pretty Raw Hex Render
1 HTTP/1.1 401 Unauthorized
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Tue, 24 Sep 2024 17:15:54 GMT
4 Content-Type: text/html
5 Content-Length: 590
6 Connection: keep-alive
7 WWW-Authenticate: Basic realm="Protected Area"
8
9 <html>
10 <head>
11 <title>
12 401 Authorization Required
13 </title>
14 </head>
15 <body>
16 <center>
17 <h1>
18 401 Authorization Required
19 </h1>
20 </center>
21

Inspector
Request attributes
Request headers
Response headers

This is what I observed when going through the same process as I did with WireShark of going on the page, putting the wrong credentials, putting the right credentials and then going onto amateurs.txt. Here, I only see the HTTP packets, with the same details as with WireShark. I see the new request header 'Authorization' with the 64-bit encoded credentials and the new response header 'WWW-Authenticate'. One thing to note is I do not see the authorization header for the favicon - which means **any unauthorized user could potentially steal your favicon**. Also, another thing I noticed is that the WWW-Authenticate response header goes away after the user is confirmed to be authenticated - maybe this is the way for the server to know that the client is authenticated and to stop asking.

Using Curl:

Basic curl - we just see that the website has been moved like before - probably to https.

```
curl http://cs338.jeffondich.com/basicauth
```

```
<html>
```

```
<head><title>301 Moved Permanently</title></head>
```

```
<body>
```

```
<center><h1>301 Moved Permanently</h1></center>
```

```
<hr><center>nginx/1.18.0 (Ubuntu)</center>
```

```
</body>
```

```
</html>
```

Trying curl verbose on https, but the TLS handshake is being blocked because the TLS certificate for the server can not be found.

curl -v https://cs338.jeffondich.com/basicauth

* Trying 172.233.221.124:443...

* Connected to cs338.jeffondich.com (172.233.221.124) port 443

* ALPN: curl offers h2.http/1.1

* (304) (OUT). TLS handshake. Client hello (1):

* CAfile: /etc/ssl/cert.pem

* CApath: none

* (304) (IN). TLS handshake. Server hello (2):

* (304) (IN). TLS handshake. Unknown (8):

* (304) (IN). TLS handshake. Certificate (11):

* (304) (IN). TLS handshake. CERT verify (15):

* (304) (IN). TLS handshake. Finished (20):

* (304) (OUT). TLS handshake. Finished (20):

* SSL connection using TLSv1.3 / AEAD-CHACHA20-POLY1305-SHA256

* ALPN: server accepted http/1.1

* Server certificate:

* subject: CN=jeffondich.com

* start date: Aug 30 17:32:27 2024 GMT

* expire date: Nov 28 17:32:26 2024 GMT

* subjectAltName does not match cs338.jeffondich.com

* SSL: no alternative certificate subject name matches target host name 'cs338.jeffondich.com'

* Closing connection

curl: (60) SSL: no alternative certificate subject name matches target host name 'cs338.jeffondich.com'

More details here: <https://curl.se/docs/sslcerts.html>

curl failed to verify the legitimacy of the server and therefore could not establish a secure connection to it. To learn more about this situation and how to fix it, please visit the web page mentioned above.

Adding -k to establish an insecure connection:

curl -k -v https://cs338.jeffondich.com/basicauth

* Trying 172.233.221.124:443...

* Connected to cs338.jeffondich.com (172.233.221.124) port 443

* ALPN: curl offers h2.http/1.1

* (304) (OUT). TLS handshake. Client hello (1):

* (304) (IN). TLS handshake. Server hello (2):

* (304) (IN). TLS handshake. Unknown (8):

* (304) (IN). TLS handshake. Certificate (11):

* (304) (IN). TLS handshake. CERT verify (15):

* (304) (IN). TLS handshake. Finished (20):

* (304) (OUT). TLS handshake. Finished (20):

* SSL connection using TLSv1.3 / AEAD-CHACHA20-POLY1305-SHA256

* ALPN: server accepted http/1.1

* Server certificate:

* subject: CN=jeffondich.com

* start date: Aug 30 17:32:27 2024 GMT

* expire date: Nov 28 17:32:26 2024 GMT

* issuer: C=US; O=Let's Encrypt; CN=E5

```
* SSL certificate verify ok.  
* using HTTP/1.1  
> GET /basicauth HTTP/1.1  
> Host: cs338.jeffondich.com  
> User-Agent: curl/8.4.0  
> Accept: */*  
>  
< HTTP/1.1 404 Not Found  
< Server: nginx/1.18.0 (Ubuntu)  
< Date: Tue, 24 Sep 2024 17:30:59 GMT  
< Content-Type: text/html  
< Content-Length: 162  
< Connection: keep-alive
```

We can see detailed information about the TLS connection - that there 7 observable steps to the handshake - although there is a number which may indicate 20 actual steps.

- 1) Client Hello (from browser)
- 2) Server hello (from web server)
- 3) Unknown (from web server)
- 4) Certificate (from web server)
- 5) CERT verify (from web server)
- 6) Finished (from web server)
- 7) Finished (from browser)

We do get a 404 Not Found reply - which doesn't align with the 401 Authentication Required reply we got with wireshark and burpsuite - so it could be that the url is wrong. But anyways - this information on TLS steps was interesting to note.

I just realised it is redirecting because I am missing a '/' at the end of the url, when doing curl -v with this url we get:

```
curl -k -v http://cs338.jeffondich.com/basicauth/  
* Trying 172.233.221.124:80...  
* Connected to cs338.jeffondich.com (172.233.221.124) port 80  
> GET /basicauth/ HTTP/1.1  
> Host: cs338.jeffondich.com  
> User-Agent: curl/8.4.0  
> Accept: */*  
>  
< HTTP/1.1 401 Unauthorized  
< Server: nginx/1.18.0 (Ubuntu)  
< Date: Tue, 24 Sep 2024 17:39:40 GMT  
< Content-Type: text/html  
< Content-Length: 188  
< Connection: keep-alive  
< WWW-Authenticate: Basic realm="Protected Area"  
<  
<html>  
<head><title>401 Authorization Required</title></head>  
<body>  
<center><h1>401 Authorization Required</h1></center>  
<hr><center>nginx/1.18.0 (Ubuntu)</center>
```

</body>

</html>

* Connection #0 to host cs338.jeffondich.com left intact

- I lost the tls data without the https though. This same http metadata could be gotten from wireshark.

Using wget:

```
wget --server-response --verbose http://cs338.jeffondich.com/basicauth
--2024-09-24 12:38:27-- http://cs338.jeffondich.com/basicauth
Resolving cs338.jeffondich.com (cs338.jeffondich.com)... 172.233.221.124
Connecting to cs338.jeffondich.com (cs338.jeffondich.com)|172.233.221.124|:80... connected.
HTTP request sent, awaiting response...
HTTP/1.1 301 Moved Permanently
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 24 Sep 2024 17:38:27 GMT
Content-Type: text/html
Content-Length: 178
Location: http://cs338.jeffondich.com/basicauth/
Connection: keep-alive
Location: http://cs338.jeffondich.com/basicauth/ [following]
--2024-09-24 12:38:27-- http://cs338.jeffondich.com/basicauth/
Reusing existing connection to cs338.jeffondich.com:80.
HTTP request sent, awaiting response...
HTTP/1.1 401 Unauthorized
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 24 Sep 2024 17:38:27 GMT
Content-Type: text/html
Content-Length: 188
Connection: keep-alive
WWW-Authenticate: Basic realm="Protected Area"
Username/Password Authentication Failed.
```

Not much to note with wget - http metadata could have been gotten from wireshark or burpsuite.