

Basic Auth Reverse Engineering and Vulnerabilities

The first request

The first request is made when entering the URL: `http://cs338.jeffondich.com/basicauth/`

When this is requested, a TCP handshake commences with between the server (Jeff's computer) and the client (Burps browser on my Mac UTM VM). After the hand shake the request below is sent:

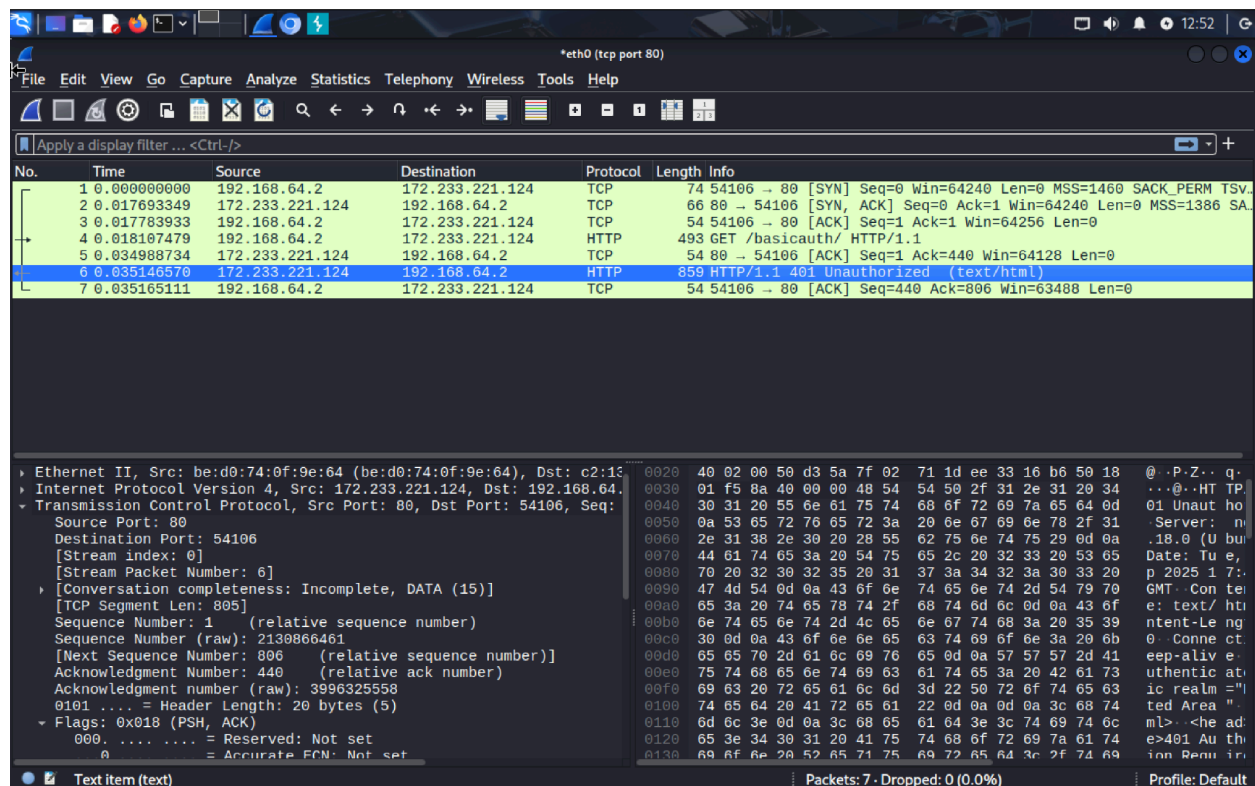
> GET /basicauth/ HTTP/1.1

The server recognizes the request with an ACK packet then sends its own HTTP packet.

> HTTP/1.1 401 Unauthorized (text/html)

Finally the client (my computer) recognizes the response with an ACK TCP packet.

A picture of this communication can be seen below.



What this means

Our client tried to access the endpoint `/basicauth/`. This endpoint requires the user to be authenticated. As this was the first time opening the endpoint (Had not authorized before) the user had not been authenticated to the server. This means that the server responded with a 401 Unauthorized error and did not send the content at `/basicauth/`. The next question to answer is why does this cause the sign up alert to appear in the browser. At this point we are not able to see any client side code using `inspect element`. With no client side code calling the alert to appear I assume that the browser's code defaults to showing this sign up alert after receiving a 401 Unauthorized error.

Second Request - Logging In

The next step is to complete the login form which is presented through the page alert. We are given the username: `'cs338'` and the password: `'password'` after inputting these values I then clicked the "Sign In" button.

After sending the request there is one important thing to note: The request headers. In this case, our client requests:

```
> GET /basicauth/
```

containing the header `"Authorization: Basic Y3MzMzg6cGFzc3dvcmQ="`. Due to the `=` at the end of the string

I immediately think that this string is base64 encoded. Decoding this string we then find:

```
"Authorization: Basic cs338:password"
```

This is our username and password which we had just submitted! This can be seen in the image below:

Filter settings: Hiding CSS, image and general binary content												
#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS
7	http://cs338.jeffondich.com	GET	/basicauth/			401	805	HTML		401 Authorization Req...		172.233.221.124
8	http://cs338.jeffondich.com	GET	/basicauth/									172.233.221.124

Request

```

1 GET /basicauth/ HTTP/1.1
2 Host: cs338.jeffondich.com
3 Cache-Control: max-age=0
4 Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=
5 Accept-Language: en-US,en;q=0.9
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
9 Accept-Encoding: gzip, deflate, br
10 Connection: keep-alive
11
12

```

Inspector

Request header

Name	Value
Authorization	Basic Y3MzMzg6cGFzc3dvcmQ=

Previous request

I then thought to look at the headers in the previous request and found an accompanying response header within the previous server error response: "WWW-Authenticate: Basic realm="Protected Area"". This connects the dots of my previous question. Our browser received the Unauthenticated error with this given header. This header told our browser if you want to access the endpoint /basicauth/ login using Basic authentication. This then triggered the browser to open the basic authentication alert.

At this point I now released the login request to the server for a response.

Servers response to GET /basicauth/ with login and password

After releasing the request another TCP handshake commences (Probably because I waited so long between request, if done quickly connection: keep-alive would have avoided this subsequent TCP handshake). This new HTTP packet is sent with the Authorization header outlined above. The server responds with ACK and then a HTTP/1.1 200 OK response! This HTTP packet contains an index of the /basicauth folder on the client. This means that /basicauth does not have any content at its root, but instead is a folder which contains the items amateurs.txt, armed-guards.txt, and dancing.txt. The server is nice enough to format this information as an HTML file for us though. Whether or not a Index is provided is a web server configuration dependent question. Some servers may respond with 404 Not found in this case. (Thanks Jeff for providing index pages).

Final Thoughts and Questions

When sending our password to the server for authentication, (The server is checking its database to see if it has seen this key value pair before), our browser sends the information `cs338:password` following the format `<username>:<pswd>` as a base64 encoded string. This is a big problem. Base64 is not an encryption technique and can be decoded by anyone. This means if someone were to be intercepting all packets with the header `"WWW-Authentication"` present. They would be able to see your own personal username and password. This of course is a problem.

Jeff's Nginx Config

Reading how password authentication is done with Nginx, the headers present during this transaction make even more sense. `/basicauth` is a protected endpoint defined within Jeff's Nginx config. We can assume it looks as so:

```
,  
location /basicauth {  
    auth_basic          "Protected Area";  
    auth_basic_user_file /etc/apache2/.htpasswd;  
}  
,
```

Due to this config when we made a request to the `/basicauth` endpoint Nginx expected our user to be authenticated. When it was not, Nginx automatically sent a response packet with the added header ``WWW-Authenticate: Basic realm="Protected Area"`.

Doing some further speculation we can assume `WWW-Authenticate` is a set header which browsers recognizes as "The value in this pair is the method I should use to authorize the user". We see the first value being Basic. Meaning we should use Basic authentication. Finally, we are given the name of the Area or "Realm" in which the config is protected. In this case: "Protected Area".

On our next request with `Authorization: username:password` given as a header. Nginx sees the header and uses the user name to search for a match within `etc/apache2/.htpasswd` (File containing hashed username:password pairs). If nginx finds a matching pair, it forwards the request `"GET /basicauth HTTP/1.1"` to the web framework process running on some port within the computer (Or simply returns the static content found at `$NGINXROOT/basicauth`, see explanation below). This is when we see our index page response.

To clarify, I mention another web server once or twice in my explanation. This is often the case for Nginx to be used as a proxy for another web process running on another port. (Nginx is good at static files, rate limiting, firewalls, etc where other web processes (Eg. Unicorn) are not, they excel in running frameworks efficiently). Although Nginx can function stand-alone by serving static files, many opt for it to be a proxy and static server only. Much of the content on Jeff's

webpage is static. Therefore, it is entirely possible Nginx is the only web process running on the server.

Incorrect username:password

I was curious if Nginx's response would differ from the initial request if an incorrect username and password is sent. Long story short, it does not. The Server sends the same HTTP/1.1 401 Unauthorized response to the client.