**Name: Mauricio I. Reyes Villanueva**
**Due: 09/24/2024**

**Prompt:**

Tell the story between the client/server interactions that occur. What happens? When? Who does what? Why did they do it that way? etc...

Your job for this assignment is to use suitable tools (wireshark, burp suite, curl,...) to help you tell the story of what happens when you access my secrets folder via a web browser. You should provide as much detail as you can figure out about the interactions between the browser and cs338.jeffondich.com's nginx server. What queries are sent from the browser, and what responses does it receive? After the password is typed by the user, what sequence of queries and responses do you see? Is the password sent by the browser to the server, or does the browser somehow do the password checking itself? If the former, is the password sent in clear text or is it encrypted or something else? If it's encrypted, where did the encryption key come from? How does what you observe via Wireshark connect to the relevant sections of the HTTP and HTTP Basic Authentication specification documents? etc.

**Start**

Disclaimer! Yesterno is not a real person!

To tell a story related to the interactions between a client & server, we must first start out by describing a setting for which such an interaction will take place. To start, we will have our main character, Yesterno. Yesterno wants to visit Jeff's super secret folder, which he just so happened to know about through a wandering sticky note, that contained the url, username, and even password to this site. (NOTE: If you're ever to find such a sticky note in real life please make sure to report it to the owner of the sticky note and do not do what Yesterno does! Now back on to the story!) Yesterno decides to open up his web browser and types in the url found in the stick note: http://cs338.jeffondich.com/basicauth/ and clicks enter.

To start things up, the browser parses the link into "http", "jeffondich.com", "/basicauth/" and does a DNS lookup, which in short translates "jeffondich.com", our host, into an ip address.

From here Yesterno's browser goes on to establish a connection with the host server, via a TCP 3-way handshake. The handshake can be broken down as follows

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 192.168.64.2 | 172.233.221.124 | TCP | 74 | 36352 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=518200985 TSecr=0 WS=128 |
| 2 | 0.025333990 | 172.233.221.124 | 192.168.64.2 | TCP | 66 | 80 → 36352 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1386 SACK_PERM WS=128 |
| 3 | 0.025382115 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 36352 → 80 [ACK] Seq=1 Ack=1 Win=32128 Len=0 |

1. The client initiates a connection by sending the server a SYN (SYNCHRONIZE) packet, which contains a sequence number (the one shown above is simply a relative sequence number). This sequence number usually starts out as a randomly large number and is incremented as packets are exchanged and is used to identify the connection.
2. The server receives the SYN packet, and goes on to return their own SYN ACK (SYNCHRONIZE ACKNOWLEDGE) packet which is simply consisting of the servers own sequence number (for connectio identification) and their acknowledge that they received the previous packet.
3. Finally, the client increments its sequence number by one, and returns an ACK packet which acknowledge receiving the servers latest packet.

From here, the browser creates the following HTTP GET request

```
GET /basicauth/ HTTP/1.1
Host: cs338.jeffondich.com
Accept-Language: en-US 2.
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Sa
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

1. The request line (first line of the header) starts off by specifying the request method, which in this case is an HTTP GET request, requesting for the /basicauth/ page, using HTTP/1.1 which is the HTTP version from the host: cs338.jeffondich.com.
2. The header then goes on to specify that the client would prefer to accept an en-US language.
3. The third line, Upgrade-Insecure-Requests key, consists of a boolean (T=1, F=0) which tells the server that the client prefers an encrypted and authenticated response, and can handle secure HTTPS connections.
4. The User-Agent identifies the software program that was used to send the request (or at least that's what it's supposed to do, but this can be modified at any point).
5. The Accept and Accept-Encoding keys aim to specify what sort of file type and encoding we're willing to accept in our response respectively.
6. Finally, Connection key aims to give the client the opportunity to hint at how the connection should be used, which in this case is set to "keep-alive."

Overall, many of the HTTP GET requests that follow will comform to a similar guideline, but may very from request to request. From here, the HTTP GET request is sent out from the senders machine, and eventually (pretty much instantaneously) reaches the server. The server then goes on to provide the following HTTP response:

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 24 Sep 2024 20:33:03 GMT
Content-Type: text/html
Content-Length: 590
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>
```

1. Some new things worth noting include the fact that we received a 401 status code in the first line, as is stated, this indicates that we're not authorized to view the content on this page until we become "authorized" via authentication.
2. The rest of the response header then consists of some more detailed information, such as the host server, the full date of when the response was sent out, the content type, length, and connection type.
3. Next we can note that the response header contains a WWW-Authenticate key. This key specifies the method that might be used to gain access to this resource. The value for this key consists of two directives, "Basic" and "realm." "Basic" indicates that the method of gaining access optionally allows charset, unlike bearer which is token68 based. Additionally, the "realm" describes the partition, since a realm allows a client to partition up the areas it protects.
4. Finally, a text/html body is provided which is used if authentication is failed and authorization is required.

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 4 | 0.027419805 | 192.168.64.2 | 172.233.221.124 | HTTP | 499 | GET /basicauth/ HTTP/1.1 |
| 5 | 0.051235934 | 172.233.221.124 | 192.168.64.2 | TCP | 54 | 80 → 36352 [ACK] Seq=1 Ack=446 Win=64128 Len=0 |
| 6 | 0.051236142 | 172.233.221.124 | 192.168.64.2 | HTTP | 859 | HTTP/1.1 401 Unauthorized (text/html) |
| 7 | 0.051326558 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 36352 → 80 [ACK] Seq=446 Ack=806 Win=31872 Len=0 |

To further elaborate on the HTTP request exchanges from earlier, there is also 2 TCP packet exchanges which occur, and can be seen via wireshark. These packet exchanges are simply packets sent either by the server or client respectively acknowledging having received the previous packet.

After this, interestingly enough though; Yesterno's TCP connection is then discontinued, but I wonder if this is due to Yesterno taking too long to respond between actions, and the machine choosing to save resources, and simply re-establish a new connection later once a new action is performed. The packets are as follows:

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 8 | 75.128655055 | 172.233.221.124 | 192.168.64.2 | TCP | 54 | 80 → 36352 [FIN, ACK] Seq=806 Ack=446 Win=64128 Len=0 |
| 9 | 75.172884890 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 36352 → 80 [ACK] Seq=446 Ack=807 Win=31872 Len=0 |

1. Here we can note that a FIN ACK packet is initiated from the server's end which specifies that they received the last issued packet (which we can see was approximately 74 seconds ago) and that they now wish to end the connection.
2. To which the client goes on to acknowledge receiving this packet and cuts the connection.

Yesterno then presents his newly found credentials to the browser, and the following packet exchanges are made:

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 10 | 425.078022191 | 192.168.64.2 | 172.233.221.124 | TCP | 74 | 40158 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=518626063 TSecr=0 WS=128 |
| 11 | 425.105056219 | 172.233.221.124 | 192.168.64.2 | TCP | 66 | 80 → 40158 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1386 SACK_PERM WS=128 |
| 12 | 425.105100761 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 40158 → 80 [ACK] Seq=1 Ack=1 Win=32128 Len=0 |
| 13 | 425.108151435 | 192.168.64.2 | 172.233.221.124 | HTTP | 568 | GET /basicauth/ HTTP/1.1 |
| 14 | 425.131913205 | 172.233.221.124 | 192.168.64.2 | TCP | 54 | 80 → 40158 [ACK] Seq=1 Ack=515 Win=64128 Len=0 |
| 15 | 425.134935004 | 172.233.221.124 | 192.168.64.2 | HTTP | 458 | HTTP/1.1 200 OK (text/html) |
| 16 | 425.134962671 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 40158 → 80 [ACK] Seq=515 Ack=405 Win=31872 Len=0 |
| 17 | 500.173121999 | 172.233.221.124 | 192.168.64.2 | TCP | 54 | 80 → 40158 [FIN, ACK] Seq=405 Ack=515 Win=64128 Len=0 |
| 18 | 500.217384715 | 192.168.64.2 | 172.233.221.124 | TCP | 54 | 40158 → 80 [ACK] Seq=515 Ack=406 Win=31872 Len=0 |

1. 10 through 12 are simply re-establishing a connection as discussed previously.
2. 17 through 18 are sending acknowledgment and connection finalization packets.
3. More interestingly packets 13 through 16 are where all the actions happening, specifically packets 13 and 15.

First up, we have the client sending a GET request:

```
GET /basicauth/ HTTP/1.1
Host: cs338.jeffondich.com
Cache-Control: max-age=0
Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Sa
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

The server which is once more requesting for the /basicauth/ page, but this time it provides the users authorization value, which is of type basic, and a Base64 encoded (for lack of a better term) cs338:password response, where the colon separatest the username and password. Although it's referred to as "encoded", Base64 is simply translating the text into something that is more machine readable for parsing. As such, there is no security encoding, and is basically as good as sending the user's credentials in plain text! It was bad enough that Jeff's sticky note wandered off into Yesterno's hands, but now they've even been exposed to a further area of compromise via the world wide web traffic through what's pretty much consistent of plain text credentials!

As a side note the GET request also contains a cache-control header requesting for the response to be of max-age=0, meaning no-cache (which is another cache-control option), or i.e. the response becomes stale from the get-go.

From here on Yesterno finally receives a worthy HTTP response!

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Tue, 24 Sep 2024 20:40:08 GMT
Content-Type: text/html
Connection: keep-alive
Content-Length: 509

<html>
<head><title>Index of /basicauth/</title></head>
<body>
<h1>Index of /basicauth/</h1><hr><pre><a href="../">../</a>
<a href="amateurs.txt">amateurs.txt</a>                                        04-Apr-2022 14:10                  75
<a href="armed-guards.txt">armed-guards.txt</a>                                    04-Apr-2022 14:10
<a href="dancing.txt">dancing.txt</a>                                         04-Apr-2022 14:10                 227
</pre><hr></body>
</html>
```

In short, this response simply contains our header body, with a new header which we haven't seen before named "Content-Length" which simply specifies the length of the body returned, which in this case is 509, and the actual body which as we can see is html for the webpage.

The webpage contains three links which lead to other subdirectories, which in this case consist of some text documents. Following any one of these would generate a new GET request, and response, as well as lead to a further exchange of TCP packets.

**End**

**Citations:**


https://www.linode.com/docs/guides/http-get-request/
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade-Insecure-Requests
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Keep-Alive
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control