Helana Solomon

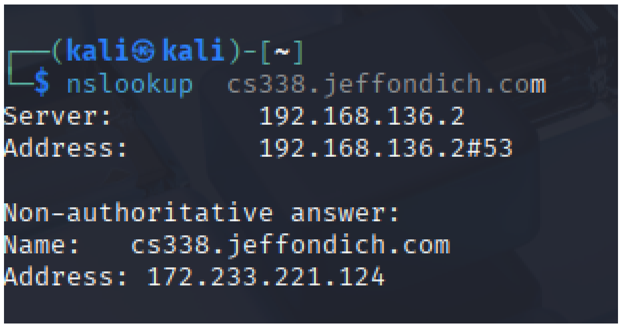
Websites/Sources specifically to understand the TLS protocol and what information was passed during each step:

https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/

https://www.youtube.com/watch?v=j9QmMEWmcfo

Description of the Task:

In this investigation, I looked into the interactions between the browser and the cs338.jeffondich.com's nginx server. To do so, I first examined the communication that happened between the client (my computer) and the server (cs338.jeffondich.com).
To do this, I used wireshark to examine the packets that were sent to and from the server. I first found the IP address to use when sending the command nslookup <website> to find the ip address of the website we are trying to analyze.

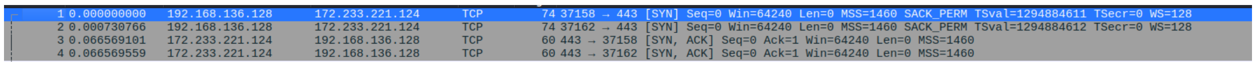


```
┌──(kali㉿kali)-[~]
└─$ nslookup  cs338.jeffondich.com
Server:         192.168.136.2
Address:        192.168.136.2#53

Non-authoritative answer:
Name:   cs338.jeffondich.com
Address: 172.233.221.124
```

After setting up the right ip address to analyze on wireshark, I go through the process of accessing the website on firefox and logging in with my credentials.

When examining the packets sent to and from the server on wireshark, I first find that the TCP Protocol is established. The client sends a message to a server saying it wants to initialize a connection, the server acknowledges the client's message and sends a message back to the client saying it wants to connect.



```
1 0.000000000   192.168.136.128   172.233.221.124   TCP   74 37158 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1294884611 TSecr=0 WS=128
2 0.000730766   192.168.136.128   172.233.221.124   TCP   74 37162 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1294884612 TSecr=0 WS=128
3 0.066569101   172.233.221.124   192.168.136.128   TCP   60 443 → 37158 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4 0.066569559   172.233.221.124   192.168.136.128   TCP   60 443 → 37162 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
```

TLS (Transport Layer Security): It is used by client-server applications to communicate with a server across a network without anyone eavesdropping. Since we want the information passed through the network without anyone accessing the information. In the case of this, we don't want anyone accessing the valid username and password to access the website.

There are three main steps in the TLS Protocol: Encryption, Authentication and Integrity.

The TLS handshake is then started by the client (my computer). The client first sends a message to the server saying Client Hello, telling the server that this is the client that initialized this TLS protocol. Then, it sends the "Application Data" which contains a list of possible ciphers the server can choose from when it can decode the public key.

```
13	0.041720186 192.168.136.128	172.233.221.124	TLSv1.3	712	Client Hello (SNI=cs338.jeffondich.com)

15	0.053926529 192.168.136.128	172.233.221.124	TLSv1.3	78	Application Data
```

The server acknowledges the response and responds with Server Hello, indicating that this is the correct server the client should be communicating to establish a connection. The server then says Change Cipher Spec - which tells the client which cipher it has chosen to use to encrypt their data. Within this data, the client also sends the public key to the client.

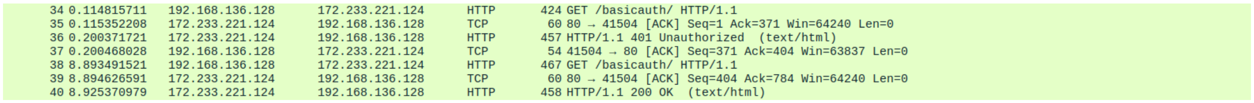
```
19	0.060654016 172.233.221.124	192.168.136.128	TLSv1.3	1436	Server Hello, Change Cipher Spec, Application Data

21	0.062325022 172.233.221.124	192.168.136.128	TLSv1.3	1074	Application Data, Application Data, Application Data
```

The client then sends the information about the encrypted key to the server. This information is within the "Application Data." This establishes the communication pathway so the information that is stored between the computers is secure.

```
25	0.076099694 192.168.136.128	172.233.221.124	TLSv1.3	78	Application Data
```

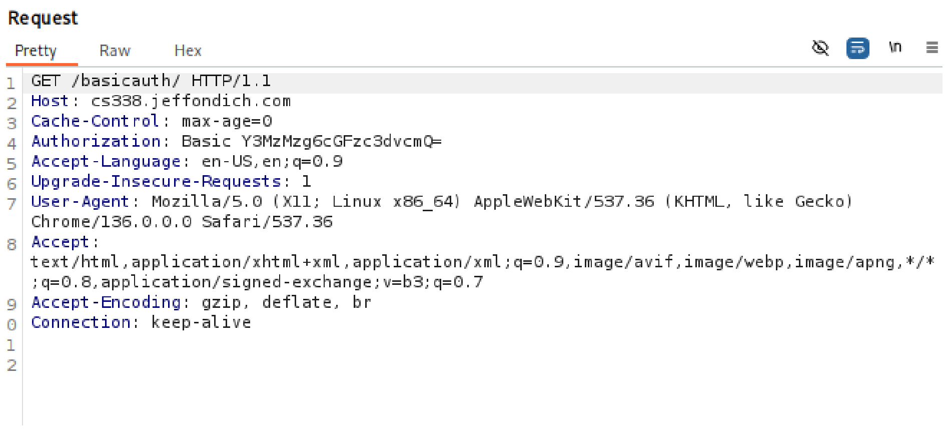
Authorization Header:



```
     34 0.114815711   192.168.136.128       172.233.221.124       HTTP       424 GET /basicauth/ HTTP/1.1
     35 0.115352208   172.233.221.124       192.168.136.128       TCP         60 80 → 41504 [ACK] Seq=1 Ack=371 Win=64240 Len=0
     36 0.200371721   172.233.221.124       192.168.136.128       HTTP       457 HTTP/1.1 401 Unauthorized  (text/html)
     37 0.200468028   192.168.136.128       172.233.221.124       TCP         54 41504 → 80 [ACK] Seq=371 Ack=404 Win=63837 Len=0
     38 8.893491521   192.168.136.128       172.233.221.124       HTTP       467 GET /basicauth/ HTTP/1.1
     39 8.894626591   172.233.221.124       192.168.136.128       TCP         60 80 → 41504 [ACK] Seq=404 Ack=784 Win=64240 Len=0
     40 8.925370979   172.233.221.124       192.168.136.128       HTTP       458 HTTP/1.1 200 OK  (text/html)
```

This is an example of what a GET request sent to the server would look like after the secure communication channel has been established. The GET request is sent by the client, but it receives an 401 unauthorized error message (line 35 in the image above) because the server is expecting the client to input credentials. The credentials needed to access the information on the server is then sent through the "Authorization" header (line

4 in the image below), which includes the encrypted credentials to get into the server (the website). This authorization header is included in the second GET request sent by the client (line 38 in the image above). Once this process is complete, the server responds with a “200” message (line 40 below), indicating that the information has been successfully sent to the client.



```
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/136.0.0.0 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
0 Connection: keep-alive
1
2
```