Ryan Son

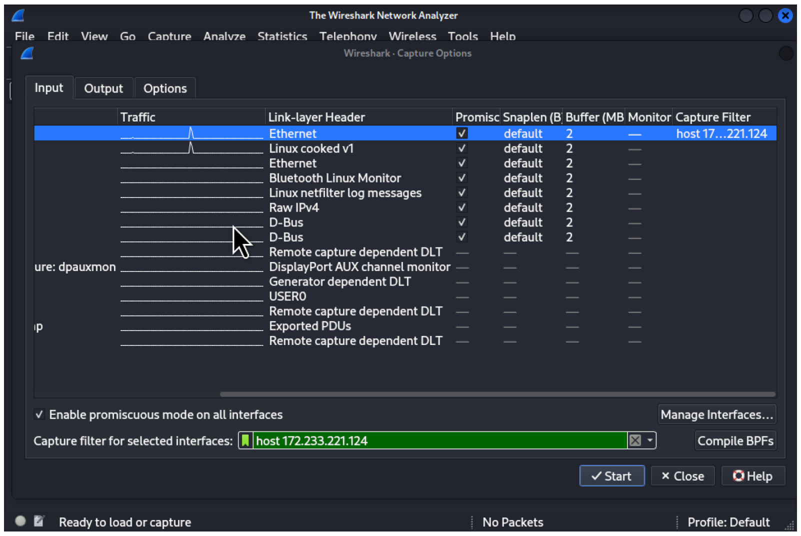
CS 338: Computer Security

Jeff Ondich

September 24, 2024

Basic Authentication Write Up

Firstly, the IP address of cs338.jeffondich.com must be identified to simply the monitoring process in Wireshark. By running "Host cs338.jeffondrich.com" in the command line, it will return with "cs338.jeffondich.com has address 172.233.221.124" which is the IP address that will be used as a Capture Filter in Wireshark. This is done by entering "host 172.233.221.124" in the Capture Options window in Wireshark.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Wireshark · Capture Options
Input
Output
Options

	Traffic	Link-layer Header	Promisc	Snaplen (B	Buffer (MB	Monitor	Capture Filter
		Ethernet	✓	default	2	—	host 17…221.124
		Linux cooked v1	✓	default	2	—	
		Ethernet	✓	default	2	—	
		Bluetooth Linux Monitor	✓	default	2	—	
		Linux netfilter log messages	✓	default	2	—	
		Raw IPv4	✓	default	2	—	
		D-Bus	✓	default	2	—	
		D-Bus	✓	default	2	—	
		Remote capture dependent DLT	—	—	—	—	
ure: dpauxmon		DisplayPort AUX channel monitor	—	—	—	—	
		Generator dependent DLT	—	—	—	—	
		USER0	—	—	—	—	
		Remote capture dependent DLT	—	—	—	—	
ıp		Exported PDUs	—	—	—	—	
		Remote capture dependent DLT	—	—	—	—	

✓ Enable promiscuous mode on all interfaces

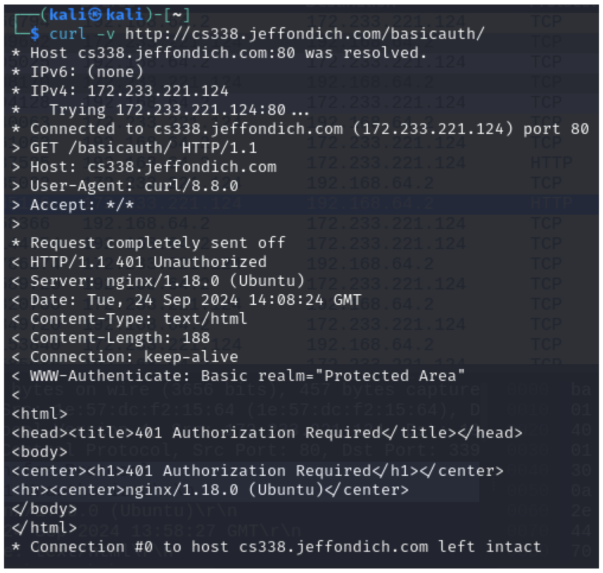
Capture filter for selected interfaces: host 172.233.221.124

Ready to load or capture | No Packets | Profile: Default

Then after pressing start, Wireshark is now configured to monitor the TCP packets that are sent and received from cs338.jeffondich.com and its associated subpages.

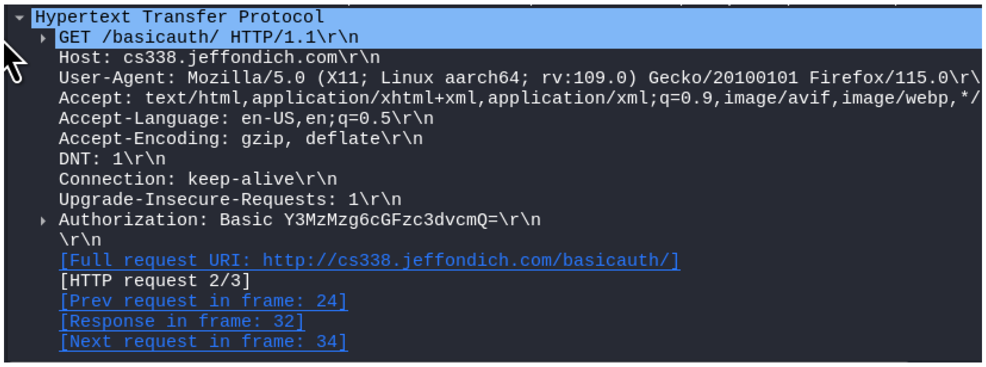
Now, opening http://cs338.jeffondich.com/basicauth/ in a Firefox private window initiates a dialog box that requests a username and password. On Wireshark, a sequence of 30 frames had been created, but it is not until frame 24 where an HTTP request is sent from the client application as "GET /basicauth/ HTTP/1.1…" and the server replies in frame 29 with

"HTTP/1.1 401 Unauthorized…" which matches the output of the command line when using "curl -v http://cs338.jeffondich.com/basicauth/".



```
┌──(kali㉿kali)-[~]
└─$ curl -v http://cs338.jeffondich.com/basicauth/
* Host cs338.jeffondich.com:80 was resolved.
* IPv6: (none)
* IPv4: 172.233.221.124
*   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/8.8.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 401 Unauthorized
< Server: nginx/1.18.0 (Ubuntu)
< Date: Tue, 24 Sep 2024 14:08:24 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
```

The line "< WWW-Authenticate: Basic realm="Protected Area"" comes from the "Basic" Hypertext Transfer Protocol authentication scheme, which details the how this authentication framework. Trying to access the website without the proper credentials will return the above "401 Authorization Required" error. Once the proper credentials are input into the box, the next frame that sends an HTTP request was 31:



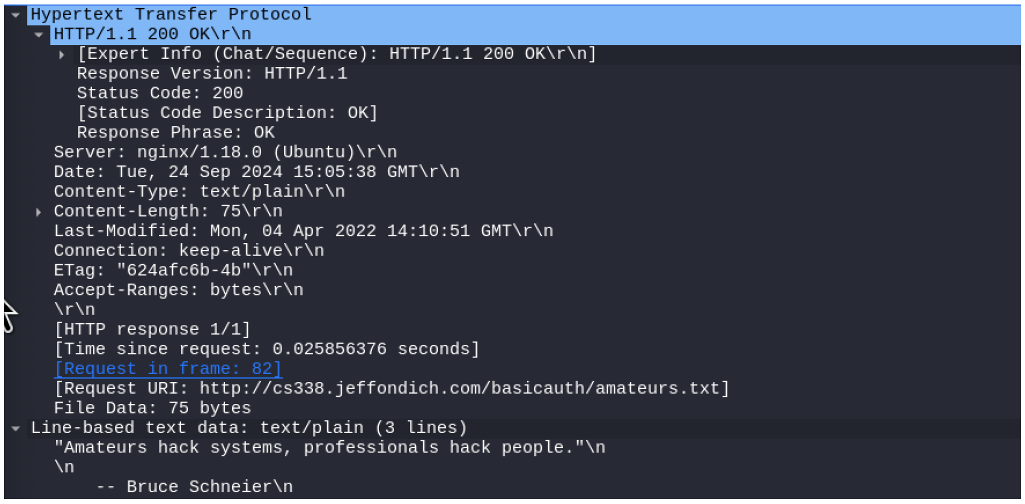
```
Hypertext Transfer Protocol
  GET /basicauth/ HTTP/1.1\r\n
  Host: cs338.jeffondich.com\r\n
  User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0\r\
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=\r\n
  \r\n
  [Full request URI: http://cs338.jeffondich.com/basicauth/]
  [HTTP request 2/3]
  [Prev request in frame: 24]
  [Response in frame: 32]
  [Next request in frame: 34]
```

An encoded version of the credentials is then sent by the client application to the server. It firstly encodes the credentials into an octet sequence and then encoding that sequence using Base64, resulting in the string "Y3MzMzg6cGFzc3dvcmQ=", as described in Section 2 of RFC 7617

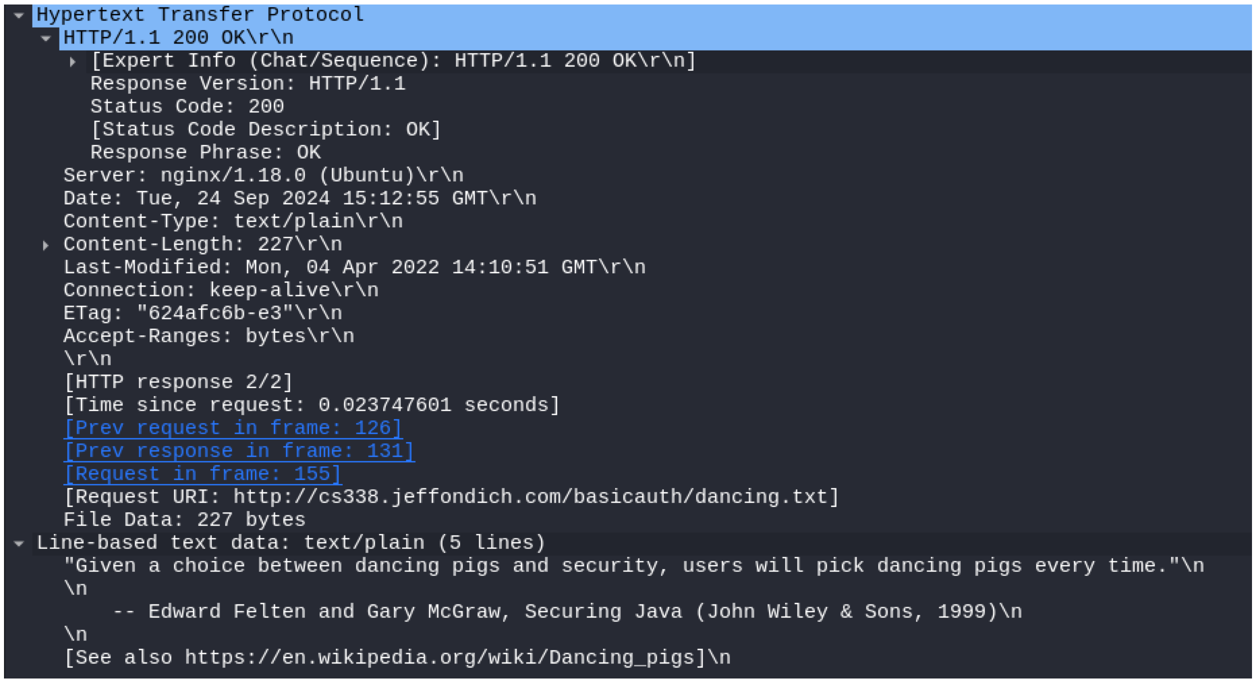
(Reschke). The client application saves this encoded credential to maintain its access to the server and will always need to occupy the “Authorization” header with the correct credential to successfully send an HTTP request to the server. In the case that the client did not save the credentials, the user would have to retype the username and password each time they clicked on a link or file that was contained in the site.

The next HTTP frame sent by the server after receiving the proper authentication was frame 32, which begins with “HTTP/1.1 200 OK…”, but an interesting change from how regular HTTP sends HTML is that this frame contains an octet encoded version of the HTML and other web contents that are displayed to the end user. This encoded information is in an HTTP chunked response.

When opening one of the files located on the website, each HTTP request sent by the client application contains the remembered “Authorization: Basic Y3MzMzg6cGFzc3dvcmQ=” to allow user access. The ensuing website output when opening “amateurs.txt” is however sent in cleartext without any encryption.


```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Response Version: HTTP/1.1
    Status Code: 200
    [Status Code Description: OK]
    Response Phrase: OK
  Server: nginx/1.18.0 (Ubuntu)\r\n
  Date: Tue, 24 Sep 2024 15:05:38 GMT\r\n
  Content-Type: text/plain\r\n
  Content-Length: 75\r\n
  Last-Modified: Mon, 04 Apr 2022 14:10:51 GMT\r\n
  Connection: keep-alive\r\n
  ETag: "624afc6b-4b"\r\n
  Accept-Ranges: bytes\r\n
  \r\n
  [HTTP response 1/1]
  [Time since request: 0.025856376 seconds]
  [Request in frame: 82]
  [Request URI: http://cs338.jeffondich.com/basicauth/amateurs.txt]
  File Data: 75 bytes
Line-based text data: text/plain (3 lines)
  "Amateurs hack systems, professionals hack people."\n
  \n
      -- Bruce Schneier\n
```

So, despite requiring authorization to access the document, if anyone is monitoring the information that is being sent by the server, then it is freely accessible by that third party. This is a severe security risk if the files contained any sensitive information. Below is the received frame 84 detailing the information inside “amateurs.txt”. This same behavior is seen when opening the other two text files on the webpage. As seen in frame 155 below:



```
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 200 OK\r\n
    ▸ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    Server: nginx/1.18.0 (Ubuntu)\r\n
    Date: Tue, 24 Sep 2024 15:12:55 GMT\r\n
    Content-Type: text/plain\r\n
  ▸ Content-Length: 227\r\n
    Last-Modified: Mon, 04 Apr 2022 14:10:51 GMT\r\n
    Connection: keep-alive\r\n
    ETag: "624afc6b-e3"\r\n
    Accept-Ranges: bytes\r\n
    \r\n
    [HTTP response 2/2]
    [Time since request: 0.023747601 seconds]
    [Prev request in frame: 126]
    [Prev response in frame: 131]
    [Request in frame: 155]
    [Request URI: http://cs338.jeffondich.com/basicauth/dancing.txt]
    File Data: 227 bytes
▾ Line-based text data: text/plain (5 lines)
    "Given a choice between dancing pigs and security, users will pick dancing pigs every time."\n
    \n
        -- Edward Felten and Gary McGraw, Securing Java (John Wiley & Sons, 1999)\n
    \n
    [See also https://en.wikipedia.org/wiki/Dancing_pigs]\n
```

This was intriguing, as if there was effort to establish some sense of encryption by encoding the authentication twice, shouldn’t the received data also require authentication by the receiving party as well? Because it is not, this information is susceptible to being viewed and manipulated before being received by the client application. In this environment, one would at most receive a false quote, but with anything more sensitive, another person could intercept social security numbers or bank account numbers. These security concerns are also reflected in Section 4 of RFC 7617. Although the credentials required to access the information on http://cs338.jeffondich.com/basicauth/ is encoded, it is not through any secure encryption methods. One note of hope is that HTTP does not hinder the ability to add more security enhancements to make data transmission across the internet more secure (Reschke).

Works Cited

Reschke, Julian. “RFC 7617: The ‘BASIC’ HTTP Authentication Scheme.” *IETF Datatracker*, Sept. 2015, datatracker.ietf.org/doc/html/rfc7617.