

CSL6010 Cyber Security

Assignment 04

Name: Sumeet S Patil

Roll No: B22CS052

Objective: The aim of this lab is to analyze network traffic using Wireshark to understand protocol behavior, measure delays, and compare secure and insecure methods of data transmission, including credential exposure and encryption.

Task 01:

Task 1.1: Local FTP Server using terminal

FTP Server was started using the terminal using the command **ftp**

Upon capturing the packets using Wireshark for linux, we could notice a few essential packets as highlighted in Fig 1 for Username. It was noticed that there was no encryption of the data being transmitted and the username **test1** could be easily captured by anyone in the middle eavesdropping and could impersonate us. In the same way, password too (Fig 2) was being sent without encryption thus allowing its interception in the middle completing all the credentials needed for the attacker to impersonate others and get access to the resources illegally.

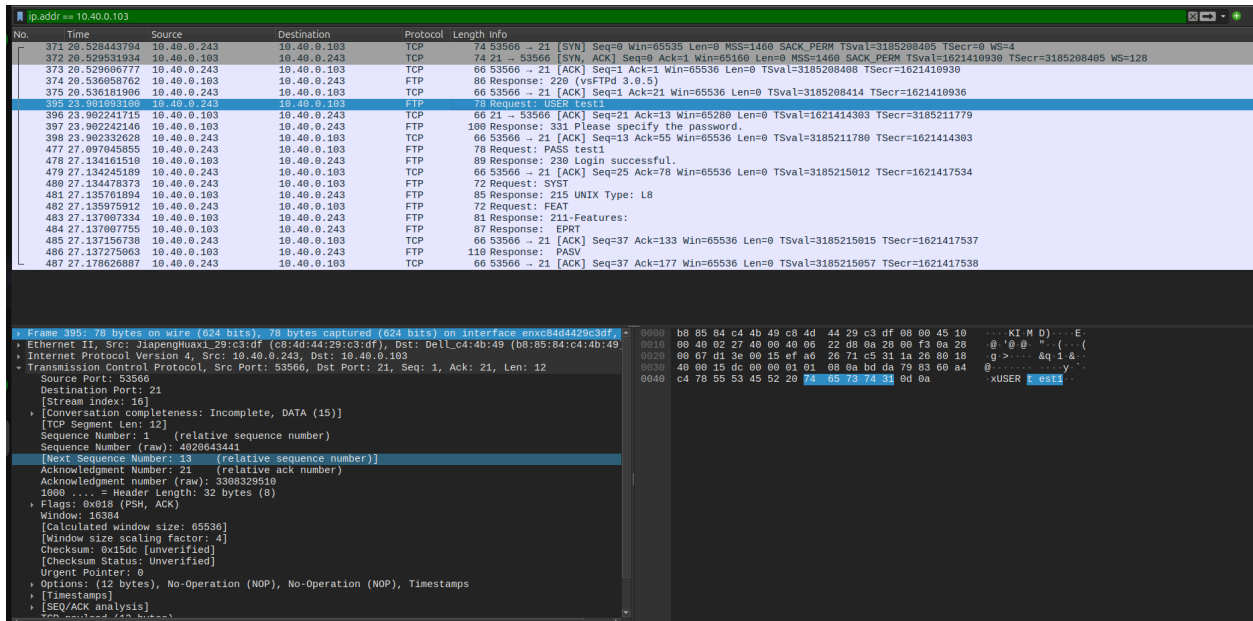


Fig 1: TCP Protocol for FTP Server (Username)

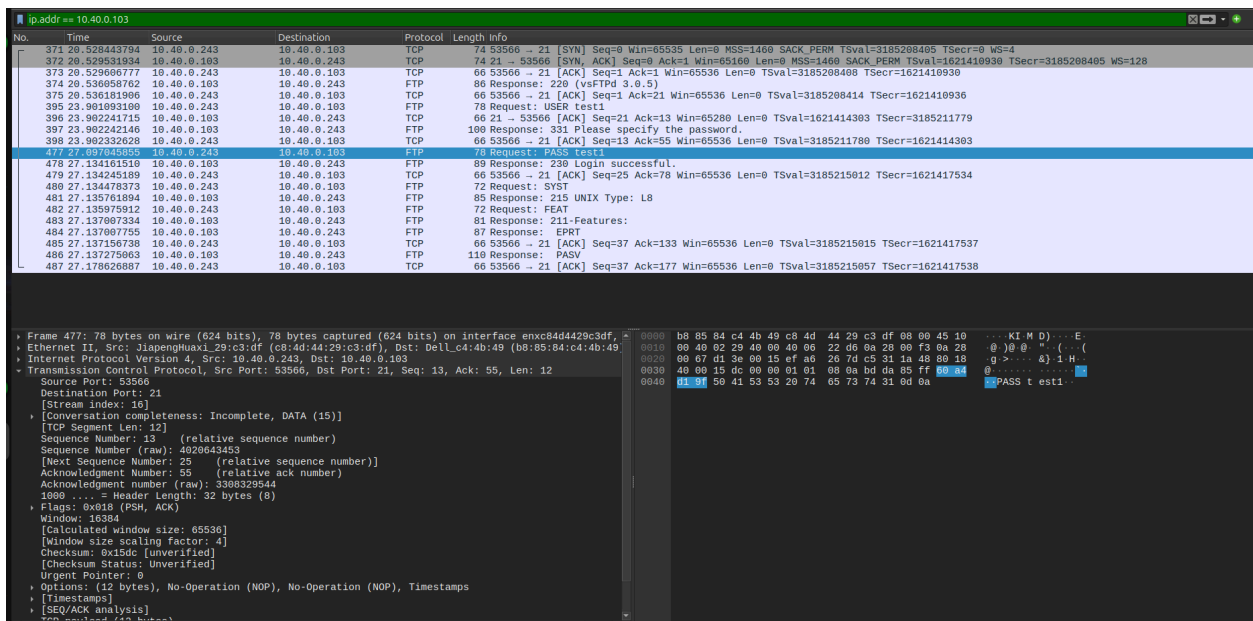


Fig 2: TCP Protocol for FTP Server (Password)

Task 1.2 FTP Server login using http

The browser Google Chrome was used to send a **GET** request to : <http://10.40.0.103> which led to the html page to login. Upon entering the credentials, we stopped capturing the packets and started observing.

Upon observation, it was found that the user name and password were url encoded (**Fig 3**), meaning part of the url of the end point corresponding to login. This is very dangerous, since the eavesdropper needn't look at the captured packet's inside data as well. The endpoint URL is quite enough to get hold of the credentials, which is quite alarming.

Task 1.3 HTTPS POST request for login

A HTTPS POST request was made to the IITJ gateway to login for Internet access and the packets' capture were stopped soon after there was a response for the server. We could notice that the POST request was actually handled by TLSv1.3, which actually maintains security, by encryption. The credentials were to be sent in the packets with labels like "Application Data" as the credentials were entered in the application layer of the network. Upon diving into further details of the packets with application data labels, we noticed that each of their data is encrypted. The encryption key is actually set first, and then the communication begins. **Fig 4** shows one such instance of an application data packet, where the encrypted data is visible. Thus, even after eavesdropping, the attacker can't get the credentials in a straightforward manner. Thus, **computational security** is achieved. In order to counter brute force attacks, I could observe a strategy, not sure whether that is the way it works. I could notice packets periodically with the description **Change of Cipher**, now these would mean probably a change of the key I guess, thus countering the brute force attacks, since after the key is changed, the ciphers also change, leading to waste of the previous information the attacker had about the old key.

No.	Time	Source	Destination	Protocol	Length	Info
1171	35.319596261	10.40.0.243	10.40.0.103	HTTP	482	GET / HTTP/1.1
1173	35.321270917	10.40.0.103	10.40.0.243	HTTP	505	HTTP/1.1 200 OK (text/html)
1185	35.403109271	10.40.0.243	10.40.0.103	HTTP	522	GET /login.html HTTP/1.1
1187	35.405343899	10.40.0.103	10.40.0.243	HTTP	1243	HTTP/1.1 200 OK (text/html)
1203	35.526702046	10.40.0.243	10.40.0.103	HTTP	432	GET /favicon.ico HTTP/1.1
1205	35.535226966	10.40.0.103	10.40.0.243	HTTP	556	HTTP/1.1 404 Not Found (text/html)
1349	42.751074045	10.40.0.103	10.40.0.243	HTTP	572	GET /login.html HTTP/1.1
1354	42.786495464	10.40.0.243	10.40.0.103	HTTP	572	GET /success.html HTTP/1.1
1355	42.788398374	10.40.0.103	10.40.0.243	HTTP	848	HTTP/1.1 200 OK (text/html)

<ul style="list-style-type: none"> Frame 1347: 570 bytes on wire (4560 bits), 570 bytes captured (4560 bits) on interface enxc84d4291 Ethernet II, Src: JlapengHuaxi 29:c3:df (c8:4d:44:29:c3:df), Dst: Dell_C4:4b:49 (b8:85:84:c4:4b:49) Internet Protocol Version 4, Src: 10.40.0.243, Dst: 10.40.0.103 Transmission Control Protocol, Src Port: 42150, Dst Port: 80, Seq: 1, Ack: 1, Len: 504 Source Port: 42150 Destination Port: 80 [Stream index: 30] [Conversation completeness: Complete, WITH DATA (31)] [TCP Segment Len: 504] Sequence Number: 1 (relative sequence number) Sequence Number (raw): 958584507 [Next Sequence Number: 505 (relative sequence number)] Acknowledgment Number: 1 (relative ack number) Acknowledgment number (raw): 2063549712 1000 = Header Length: 32 bytes (8) Flags: 0x018 (PSH, ACK) Window: 502 [Calculated window size: 64250] [Window size scaling factor: 128] Checksum: 6x17c8 [unverified] [Checksum Status: Unverified] Urgent Pointer: 0 Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps [Timestamps] [SEQ/ACK analysis] 	<pre> 0000 08 85 84 c4 4b 49 c8 4d 44 29 c3 df 08 00 45 00 ...KI-M D)....E 0010 82 2c 69 31 49 00 40 06 b9 f1 0a 28 00 f3 0a 28 ...110-0-...(-(- 0020 06 07 a4 a0 00 59 39 22 d0 b0 7b 00 c0 b6 00 18 ...g...P9" :{.... 0030 01 f6 17 c0 00 00 01 01 00 00 dd 09 3d bc 00 a8 0040 95 d2 47 45 54 20 2f 6c 0f 07 09 0e 5f 70 72 6f ...GET / login_pro 0050 03 05 73 73 2e 68 74 6d 0c 3f 75 73 05 72 6e 01 ...ess.htm /userna 0060 04 05 3d 74 05 73 74 31 20 70 61 73 77 6f 72 ...ect-test1 dpasswor 0070 64 3d 74 05 73 74 31 20 40 54 54 50 2f 31 2e 31 ...d-test1 HTTP/1.1 0080 0d 0a 48 0f 73 74 3a 20 31 30 2e 34 30 2e 30 2e ...Host: 10.40.0. 0090 21 30 23 00 0a 43 0f 0e 0e 05 63 74 09 0f 06 3a ...100-Connnection 0100 20 0b 05 65 70 2d 61 6c 60 70 05 0d 0a 55 70 0f ...keep-alive=upp 0110 72 01 64 05 2d 49 6e 73 05 63 75 72 65 2d 52 65 ...rade-Ins ecur:Re 0120 01 67 05 6e 74 3a 20 4d 0f 7a 09 0c 6c 61 2f 35 ...Agent: Mozilla/5 0130 2e 30 20 20 58 31 31 3b 20 4c 09 0e 75 70 20 78 ...0 (X11; Linux x 0140 88 39 5f 36 34 20 20 41 70 70 6c 05 6f 65 62 40 ...86.04) AppleWebKit 0150 69 74 2f 35 33 37 2e 33 30 20 20 40 54 40 46 ...10/537.3.6 (KHTML 0160 20 20 60 69 0b 05 29 47 05 63 0b 0f 29 29 43 68 ...like a cack) Ch 0170 72 0f 6d 05 2f 31 33 31 2e 30 2e 30 2e 30 20 53 ...rome/131.0.0.0 S 0180 01 06 01 72 69 2f 35 33 37 2e 33 30 0d 0a 41 03 ...afari/53.7.36-AC 0190 03 05 70 74 3a 20 74 05 70 74 2f 68 74 6d 6c 2e ...empt: te xt/html; 01a0 01 70 70 6c 09 63 61 74 09 0f 6e 2f 78 68 74 6d ...applicat ion/xhta 01b0 6c 2b 78 6d 6c 2c 61 70 70 6c 09 63 61 74 69 0f ...lxml,application 01c0 0e 2f 78 6d 6c 6b 71 31 30 2e 39 20 69 6d 0f 01 ...n/xml;pr o=0.0;limd 01d0 05 2f 61 76 69 0c 2c 69 6d 01 67 05 2f 77 05 02 ...e/avai,1 mapg/web 01e0 70 2c 69 6d 01 67 05 2f 61 70 6e 07 2c 2a 2f 28 ...p,image/ appn,7/ </pre>
--	---

Fig 3: HTTP packets for FTP Server

No.	Time	Source	Destination	Protocol	Length	Info
47	3.401821398	10.40.0.243	172.17.0.3	TCP	74	42050 -> 1003 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3859001733 TSecr=0 WS=128
48	3.402490180	172.17.0.3	10.40.0.243	TCP	74	1003 -> 42050 [SYN, ACK] Seq=0 Ack=1 Win=14400 Len=0 MSS=1460 SACK_PERM TSval=854683322 TSecr=3859001733 WS=1024
49	3.402574405	10.40.0.243	172.17.0.3	TCP	66	42050 -> 1003 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3859001734 TSecr=854683322
51	3.403321130	10.40.0.243	172.17.0.3	TLV1.3	2081	Client Hello (SNI=gateway.iitj.ac.in)
52	3.404080406	172.17.0.3	10.40.0.243	TCP	66	1003 -> 42050 [ACK] Seq=1 Ack=1449 Win=17400 Len=0 TSval=854683322 TSecr=3859001735
53	3.404089847	172.17.0.3	10.40.0.243	TCP	66	1003 -> 42050 [ACK] Seq=1 Ack=2016 Win=20400 Len=0 TSval=854683322 TSecr=3859001735
54	3.404292622	172.17.0.3	10.40.0.243	TLV1.3	165	Hello Retry Request, Change Cipher Spec
55	3.404319425	10.40.0.243	172.17.0.3	TCP	66	42050 -> 1003 [ACK] Seq=2016 Ack=100 Win=64256 Len=0 TSval=3859001736 TSecr=854683322
56	3.404399121	10.40.0.243	172.17.0.3	TLV1.3	895	Change Cipher Spec, Client Hello (SNI=gateway.iitj.ac.in)
57	3.405845289	172.17.0.3	10.40.0.243	TLV1.3	322	Server Hello, Application Data, Application Data
58	3.406040372	10.40.0.243	172.17.0.3	TLV1.3	124	Application Data
59	3.406104055	10.40.0.243	172.17.0.3	TLV1.3	1039	Application Data
60	3.407161957	172.17.0.3	10.40.0.243	TLV1.3	321	Application Data
62	3.500093734	172.17.0.3	10.40.0.243	TLV1.3	2083	Application Data, Application Data
63	3.500198352	10.40.0.243	172.17.0.3	TCP	66	42050 -> 1003 [ACK] Seq=3076 Ack=2629 Win=68224 Len=0 TSval=3859001772 TSecr=854683322
64	3.500302020	10.40.0.243	172.17.0.3	TCP	66	42050 -> 1003 [FIN, ACK] Seq=3076 Ack=2629 Win=68224 Len=0 TSval=3859001775 TSecr=854683322
65	3.504496027	172.17.0.3	10.40.0.243	TCP	66	1003 -> 42050 [ACK] Seq=2629 Ack=3077 Win=26624 Len=0 TSval=854683326 TSecr=3859001775
66	3.552382429	10.40.0.243	172.17.0.3	TCP	74	42062 -> 1003 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3859001824 TSecr=0 WS=128
67	3.553137290	172.17.0.3	10.40.0.243	TCP	74	1003 -> 42062 [SYN, ACK] Seq=0 Ack=1 Win=14400 Len=0 MSS=1460 SACK_PERM TSval=854683331 TSecr=3859001824 WS=1024
68	3.553298160	10.40.0.243	172.17.0.3	TCP	66	42062 -> 1003 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3859001825 TSecr=854683331
69	3.554280970	10.40.0.243	172.17.0.3	TLV1.3	2081	Client Hello (SNI=gateway.iitj.ac.in)
70	3.555144771	172.17.0.3	10.40.0.243	TCP	66	1003 -> 42062 [ACK] Seq=1 Ack=1449 Win=17400 Len=0 TSval=854683331 TSecr=3859001826
71	3.555145423	172.17.0.3	10.40.0.243	TCP	66	1003 -> 42062 [ACK] Seq=1 Ack=2016 Win=20400 Len=0 TSval=854683331 TSecr=3859001826
72	3.555346382	172.17.0.3	10.40.0.243	TLV1.3	165	Hello Retry Request, Change Cipher Spec
73	3.555493385	10.40.0.243	172.17.0.3	TCP	66	42062 -> 1003 [ACK] Seq=2016 Ack=100 Win=64256 Len=0 TSval=3859001827 TSecr=854683331

<ul style="list-style-type: none"> Frame 58: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface enxc84d4291c3df Ethernet II, Src: JlapengHuaxi 29:c3:df (c8:4d:44:29:c3:df), Dst: Cisco_35:48:e4 (26:31:24:35:48:e4) Internet Protocol Version 4, Src: 10.40.0.243, Dst: 172.17.0.3 Transmission Control Protocol, Src Port: 42050, Dst Port: 1003, Seq: 2045, Ack: 356, Len: 50 Transport Layer Security TLV1.3 Record Layer: Application Data Protocol: Application Data Opaque Type: Application Data (23) Version: TLS 1.2 (0x0303) Length: 53 Encrypted Application Data: 2004068c42190cd2753033a9e210537a7b7c7f7bb332a206228534eb547dbb49044 	<pre> 0000 26 31 24 35 48 e4 e4 26 44 29 c3 df 08 00 45 00 ...SSH- W D)....E 0010 00 0e 0f c5 40 00 40 06 13 9a 0a 28 00 f3 0a 11 ...no 0 0 :{.... 0020 00 03 a4 42 03 eb 8f 07 5d ad 0a 70 d9 6d 00 18 ...B... : p m... 0030 01 f5 b7 0f 00 00 01 01 00 0a e0 03 d1 8a 32 f1 : 2 0040 0e ba 1f 03 03 00 35 20 04 05 0e 42 19 bc 00 2f : f b... 0050 03 03 3a 9e 21 85 37 a7 b7 cf 7b 32 3a 2a 02 ... : (2-b 0060 20 53 eb 54 7d bb 49 04 45 90 db 38 5e 29 ef 09 ...S T) : E...B... 0070 fa e3 9f c2 e3 a9 3c e6 3d 51 df 6d ... : c... : Q... </pre>
--	--

Fig 4: HTTPS packets to IITJ gateway

Task 02:

TCP handshake delay is basically the delay caused due to TCP handshake for establishing the TCP connection between two machines.

Usually TCP handshake delay is equal to the sum of time between syn ,syn ack signals and syn ack , ack signals. Mathematically in Wireshark,
TCP handshake delay = (SYN ACK TS) - (SYN TS) + (ACK TS) - (SYN ACK TS)
Where TS = timestamp

Now, the TCP handshake is a very crucial part of the TCP/IP protocol, since this establishes the TCP connection between two machines correctly.

Impact of TCP handshake delay on network performance:

1. Increase in its delay will slow down the communication between machines, especially for short lived connections like HTTP requests.
2. Websites relying on the multiple TCP connections will have too much slowed down page loading, since resources are fetched using different TCP connections, each beginning with a handshake.

How handshake delays can be analyzed:

1. If there is too much delay in the syn and ack signals, it can be due to congestion on the network, due to packet losses or transmission delays
2. Also, the server might be quite slow, thus having the issue of processing time and in turn the delay.

Methods to analyze application level delays in wireshark:

1. For HTTP, we can set the filter **http.request || http.response**, thus the time between these for a particular source and destination IP, the delays can be known.
2. Same way for DNS queries and responses, filters set to **dns** would help analyze.

Ways to optimize:

1. If websites have a long load time, then we can reduce the handshakes by keeping **persistent TCP connections**.
2. Having **local caching** of DNS data (Domain name,IP) pairs to eliminate dns requests can help boost the communication by reducing latency.
3. Finally, using HTTP/2 or HTTP/3 can help in reducing the application level latency, since they are known for faster set up times.

Task 03:

The pcap downloaded is: [File](#)

Looking at the capture for a bird eye view, it was clearly visible that it was a UDP attack. To confirm it, I applied the filter “udp” and noticed that out of the total 13804 packets captured, 9657 (70%) were displayed after the filter (Fig 5). Thus, it was confirmed to be a UDP attack. The next task(s) were to determine:

- a. How to differentiate between legitimate and attack traffic
- b. The attack IP address(es)
- c. Analyzing its impact of the services of the server

To differentiate legitimate traffic from attack traffic, I went to **Statistics->Conversations->UDP**

Then sorting by number of packets sent, there was a clear picture who the attacker was as seen in **Fig 6**. The IP **192.168.0.2** sent 7477 UDP packets to an IP **192.168.0.1**, thus being the sole attackers, since the other sources hardly had above 50 packets. Thus, by applying this method, we could achieve both a and b part of the remaining tasks. To confirm the victim machine, I applied the filter **ip.dst == 192.168.0.1** helping me find out that around 8912 packets were sent to it, thus confirming it being the victim.

Now, coming to the last sub part of this question, to analyze the impact, I searched for other IP sources interacting with the server and I got 192.168.0.37. Then applying a filter to check the difference between the request sent and the response received, it was found out to be 2 s, while the same IP when interacting with another machine 115.0.0.2 receives response in 0.2 s, this actually shows the difference (Fig 8). Thus, the flood was successful in hampering the services of the server, slowing down the response time to nearly 10 times.

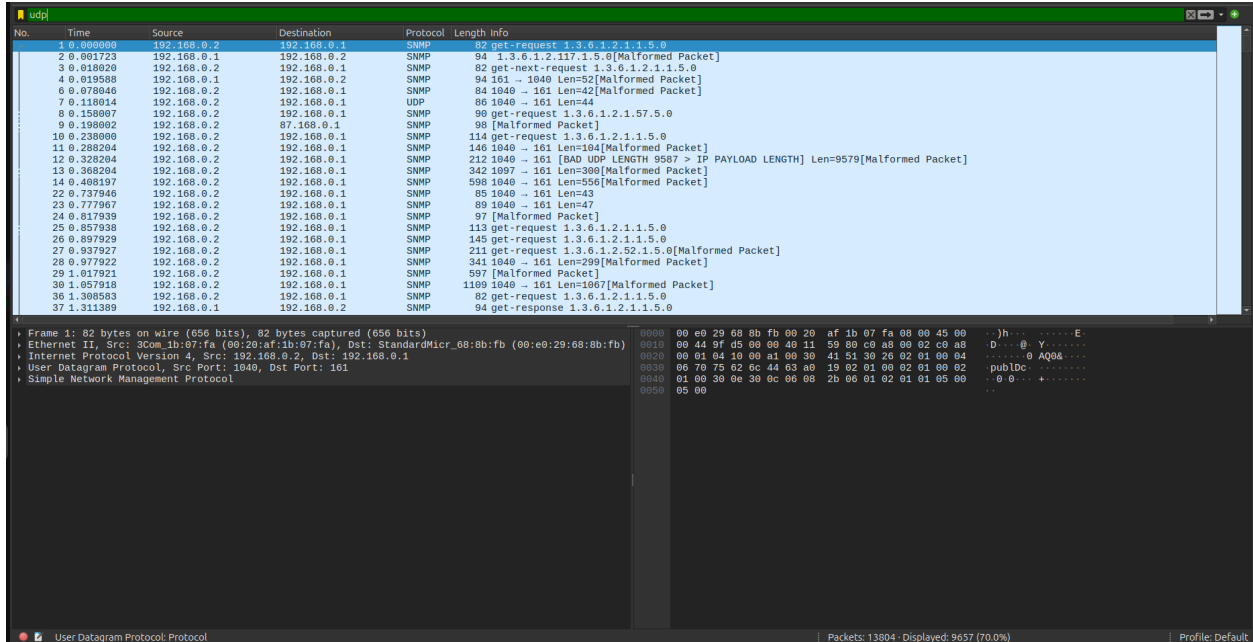


Fig 5: UDP Filter

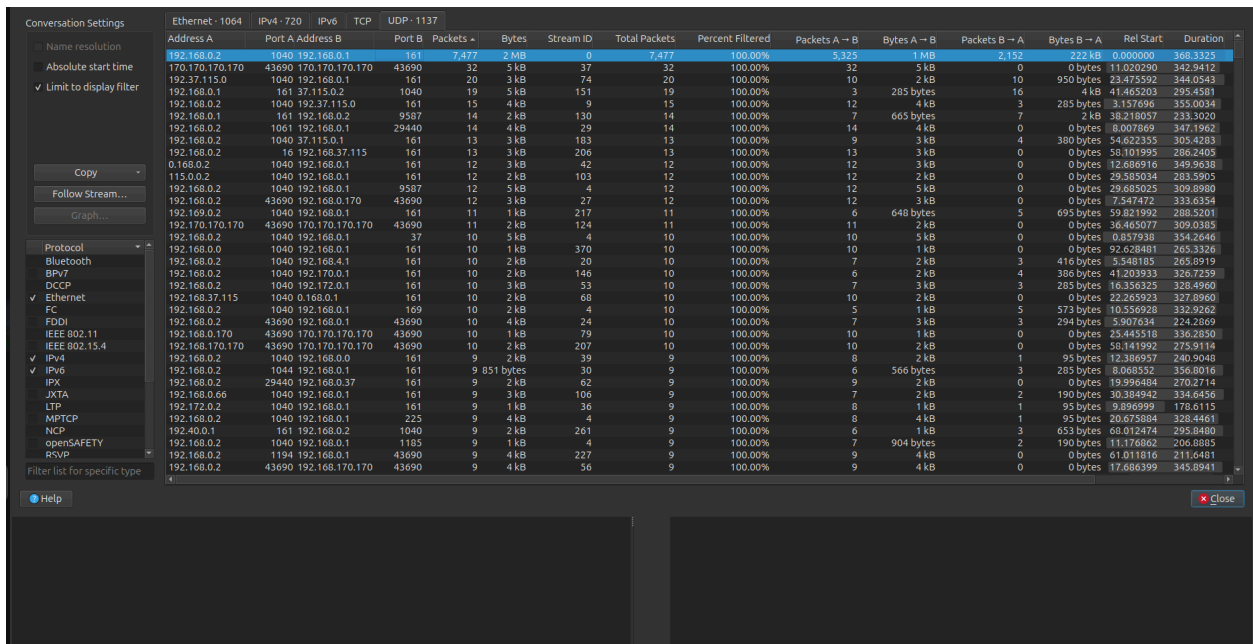


Fig 6: UDP attack statistics

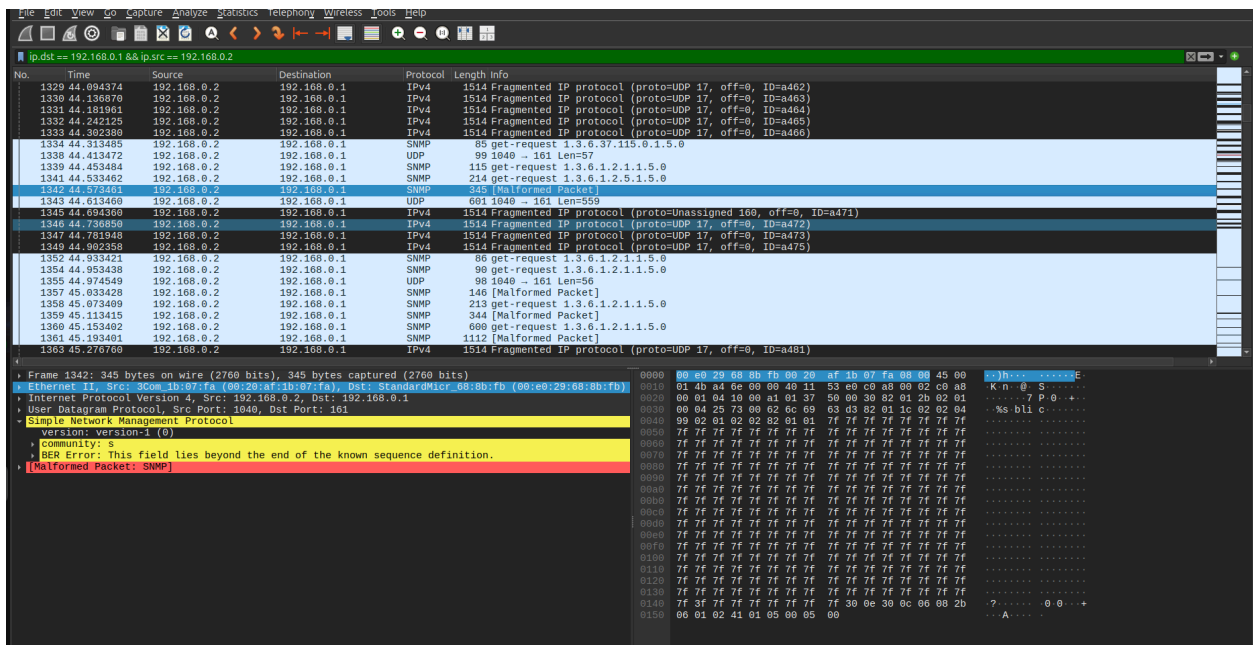


Fig 7: The Data of the packet is actually very much corrupted

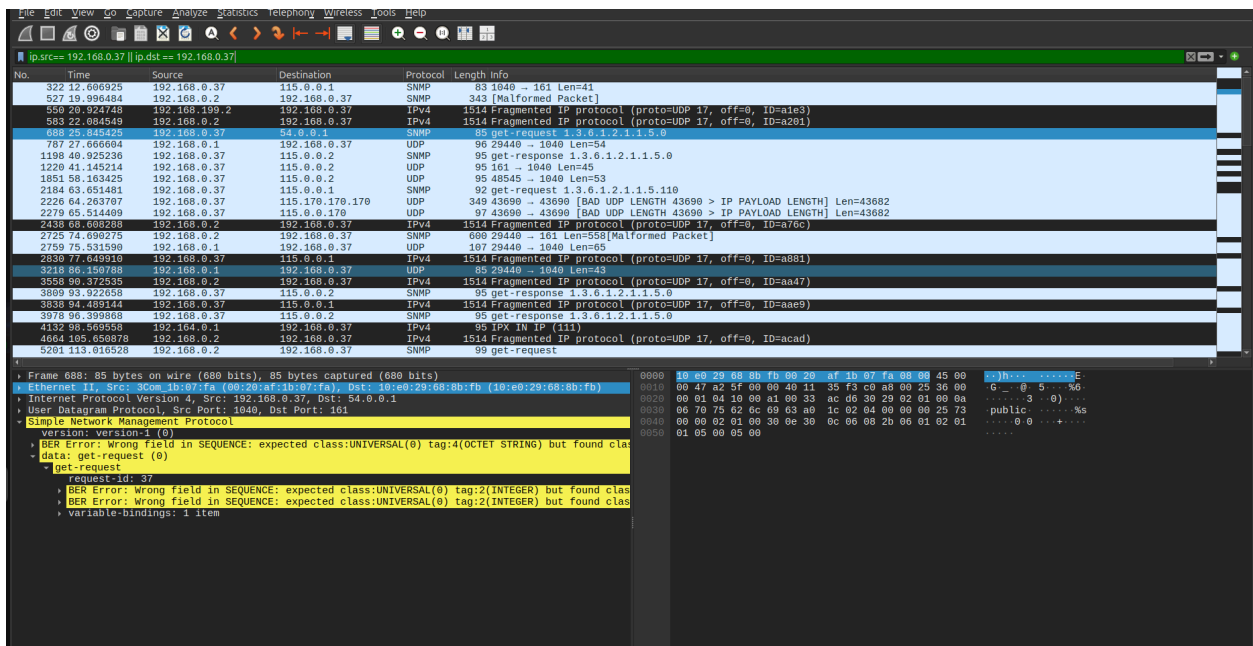


Fig 8: Impact of UDP flood