

## Find 100 things that you have learnt with Wireshark analysis

- Packets 1 and 2 depicts DNS query and response respectively to DNS server

No.	Source	Destination	Length	Protocol	Time	Info
1	10.14.135.119	10.14.135.5	75	DNS	0.000000	Standard query 0x0985 A pulpfarmerd.com
2	10.14.135.5	10.14.135.119	91	DNS	0.139827	Standard query response 0x0985 A pulpfa

The client is trying to visit pulpfarmed.com. A request from client located at 10.14.135.119 is sent to the DNS server at 10.14.135.5

▼ Domain Name System (query)

- Transaction ID: 0x0985
- › Flags: 0x0100 Standard query
- Questions: 1
- Answer RRs: 0
- Authority RRs: 0
- Additional RRs: 0
- ▼ Queries
  - ▼ pulpfarmerd.com: type A, class IN
    - Name: pulpfarmerd.com
    - [Name Length: 15]
    - [Label Count: 2]
    - Type: A (Host Address) (1)
    - Class: IN (0x0001)
- [\[Response In: 2\]](#)

- Flags: 0 means standard query

Questions: Indicates the number of queries present in the packet. In this case it is 1

Answers: Indicates the number of answers in response to the query sent.

Authority RRs: Indicates the number of authority resource records sent as response.

Additional RRs: Indicates the number of additional resource records sent as response

Therefore, as per the screenshot above, 1 query was sent in the packet.

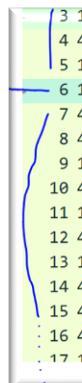
- In response to this query, you will be seeing one more packet with the same transaction ID that denotes the association of a particular query. It is the response packet. Response for our query will usually consist of IPv4 address for the domain we are trying to look for – 45.95.11.201 in our case. This is depicted in the screenshot below.

```

Domain Name System (response)
  Transaction ID: 0x0985
  > Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  < Queries
    < pulpfarmerd.com: type A, class IN
      Name: pulpfarmerd.com
      [Name Length: 15]
      [Label Count: 2]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
  < Answers
    > pulpfarmerd.com: type A, class IN, addr 45.95.11.201
  Request Tn: 11

```

4. At the beginning of every HTTP session, the TCP three-way handshake takes place. It creates a dedicated channel between the communicating hosts followed by HTTP and data packets, which are sent in and received while the session is active. For instance, you are visiting a web server located at <http://45.95.11.201> and the client at 10.14.135.119.
5. This is how a request looks. In the first line, there are three things passed on to the server as the arguments, which are HTTP method and requested resource location “/” (root directory) i.e the Host argument which is required by the HTTP/1.1 protocol requests. The value of this field is the webserver’s address that you typed in the address bar of the browser.



3	10.14.135.119	45.95.11.201	66	TCP	0.144709	60686 → 80	[SYN]	Seq=0	Win=65535	Len=0
4	45.95.11.201	10.14.135.119	58	TCP	0.478890	80 → 60686	[SYN, ACK]	Seq=0	Ack=1	Win=6
5	10.14.135.119	45.95.11.201	54	TCP	0.479114	60686 → 80	[ACK]	Seq=1	Ack=1	Win=65535
6	10.14.135.119	45.95.11.201	583	HTTP	0.479293	GET /cbfsd/B1DFRsj1bsGvKdLIj/98697/7309				
7	45.95.11.201	10.14.135.119	54	TCP	0.479488	80 → 60686	[ACK]	Seq=1	Ack=530	Win=6424
8	45.95.11.201	10.14.135.119	1430	TCP	0.801899	80 → 60686	[PSH, ACK]	Seq=1	Ack=530	Win=6
9	10.14.135.119	45.95.11.201	54	TCP	0.802117	60686 → 80	[ACK]	Seq=530	Ack=1377	Win=6
10	45.95.11.201	10.14.135.119	1430	TCP	0.804594	80 → 60686	[PSH, ACK]	Seq=1377	Ack=530	
11	10.14.135.119	45.95.11.201	54	TCP	0.804762	60686 → 80	[ACK]	Seq=530	Ack=2753	Win=6
12	45.95.11.201	10.14.135.119	1430	TCP	0.814172	80 → 60686	[PSH, ACK]	Seq=2753	Ack=530	
13	10.14.135.119	45.95.11.201	54	TCP	0.814378	60686 → 80	[ACK]	Seq=530	Ack=4129	Win=6
14	45.95.11.201	10.14.135.119	1514	TCP	0.814509	80 → 60686	[ACK]	Seq=4129	Ack=530	Win=6
15	45.95.11.201	10.14.135.119	1346	TCP	0.814514	80 → 60686	[PSH, ACK]	Seq=530	Ack=530	
16	45.95.11.201	10.14.135.119	1430	TCP	0.814553	80 → 60686	[PSH, ACK]	Seq=6881	Ack=530	
17	10.14.135.119	45.95.11.201	54	TCP	0.814711	60686 → 80	[ACK]	Seq=530	Ack=8257	Win=6

- In packet number 6, you see that the client performing a GET request as per the screenshot above.
- There are 11866 TCP packets in the pcap file.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.14.135.119	10.14.135.5	DNS	Standard query 0x0985 A pulpfarmerd.com
2	0.139827	10.14.135.5	10.14.135.119	DNS	Standard query response 0x0985 A pulpfarmerd.com A 45.95.11.201
3	0.004882	10.14.135.119	45.95.11.201	TCP	60686 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
4	0.334181	45.95.11.201	10.14.135.119	TCP	80 → 60686 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.000224	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
6	0.000179	10.14.135.119	45.95.11.201	HTTP	GET /cbfsd/B1DFRsj1bsGvKdL1j/98697/7309/33451/Pg9zYLcfzirZtPtx1Pn...

In packets 3, 4, and 5, it is clearly visible that the client and server are trying to create a dedicated channel.

- The client initiated the creation by sending a SYN packet in the 3 packet with the SEQ set to 0. Since the server was open for communication, the server responded with a SYN/ACK packet with ACK set to 1 and SEQ set to 0.
- This is followed by a confirmation sent from the client side that is seen in the packet number 5 with SEQ=1 and ACK=1. This is what is known as a three-way handshake process.
- This can be seen before any real data transfer that happens that follows the TCP approach. After the successful completion of channel creation, the client sends a GET request to access the contents of the web-root directory.

507	0.000347	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=437569 Win=65535 Len=0
508	0.002143	45.95.11.201	10.14.135.119	TCP	80 → 60686 [PSH, ACK] Seq=437569 Ack=530 Win=64240 Len=1376 [TCP ...
509	0.000265	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=438945 Win=64159 Len=0
510	0.000817	45.95.11.201	10.14.135.119	TCP	80 → 60686 [PSH, ACK] Seq=438945 Ack=530 Win=64240 Len=1376 [TCP ...
511	0.000122	45.95.11.201	10.14.135.119	TCP	80 → 60686 [ACK] Seq=440321 Ack=530 Win=64240 Len=1460 [TCP segme...
512	0.000053	45.95.11.201	10.14.135.119	HTTP	HTTP/1.1 200 OK
513	0.000139	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=440321 Win=65535 Len=0
514	0.000131	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=442998 Win=65535 Len=0
515	0.185942	10.14.135.119	45.95.11.201	TCP	60686 → 80 [RST, ACK] Seq=530 Ack=442998 Win=0 Len=0
516	5.895096	10.14.135.119	10.14.135.5	TCP	60690 → 135 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
517	0.000242	10.14.135.5	10.14.135.119	TCP	135 → 60690 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25...
518	0.000225	10.14.135.119	10.14.135.5	TCP	60690 → 135 [ACK] Seq=1 Ack=1 Win=262656 Len=0
519	0.000059	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: EPMv4 V3.0 (...)

- The server acknowledged this the packet number 512 but sent the requested content to the client's machine with the 200 OK status message after many packets. Acknowledgement should have happened immediately in the next packets. This looks suspicious.
- After a few packets were exchanged between the hosts in the later frames, the clients received a 200 OK status message, suggesting successful transfer of the malware at packet number 512. Refer below screenshot:

512	45.95.11.201	10.14.135.119	1181	HTTP	2.460193	HTTP/1.1	200	OK
513	10.14.135.119	45.95.11.201	54	TCP	2.460332	60686 → 80	[ACK]	Seq=530 Ack=440321 Win
514	10.14.135.119	45.95.11.201	54	TCP	2.460463	60686 → 80	[ACK]	Seq=530 Ack=442908 Win
TCP segment data (1127 bytes)								
> [320 Reassembled TCP Segments (442907 bytes): #8(1376), #10(1376), #12(1376), #14(1460), #15(1292), #16(1376), #18(1376), #20(1376)								
▼ Hypertext Transfer Protocol								
> HTTP/1.1 200 OK\r\n								
Date: Wed, 03 Nov 2021 17:42:13 GMT\r\n								
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.2.34\r\n								
X-Powered-By: PHP/7.2.34\r\n								
Content-Description: File Transfer\r\n								
Content-Disposition: attachment; filename="zidem6"\r\n								
Expires: 0\r\n								
Cache-Control: must-revalidate\r\n								
Pragma: public\r\n								
Content-Length: 442495\r\n								

13. Follow the TCP stream for this particular packet and you can see the below details.



```
Wireshark - Follow TCP Stream (tcp.stream eq 0) - sinkhole.pcap

GET /cbfsd/B1DFRsj1bsGvKdLIj/98697/7309/33451/Pg9zYLcfzirZtPtx1Pn64fLoWAIDvNPx41clw/
LaQAZSeiTYPcjjCble334/QdHhD0r/98/RDvuSh/zidem3?
q=RYaTpLn2leLH6rxKG0px1CME3RY&sid=UY8SVDRzRqZb&CWPJmychHi=iF0I26&sid=YGrkJjD4n&q=mbdtF5zi
KWJczkstB1W0PBT7Ia&time=DEY07nTt&q=EY7s124iZtw7zTehznnCVwHt&q=G9FdCrnm6Z6yu HTTP/1.1
Accept: */
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; WOW64; Trident/7.0;
.NET4.0C; .NET4.0E)
Host: pulpfarmerd.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 03 Nov 2021 17:42:13 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.2.34
X-Powered-By: PHP/7.2.34
Content-Description: File Transfer
Content-Disposition: attachment; filename="zidem6"
Expires: 0
Cache-Control: must-revalidate
Pragma: public
Content-Length: 442495
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/octet-stream

MZ.....@..... !..L!.
This program cannot be run in DOS mode.

$.....[...[...[...] ..\...| ..N...[...K...E.&....E.0.Q...E.!.%...E./.\...E.
7.Z...E.1.Z...E.4.Z...Rich[.....PE..d...F.`....." ..
.....n.....P.....d.....@.....
```

Packet 6. 1 client pkt, 320 server pkts, 1 turn. Click to select.

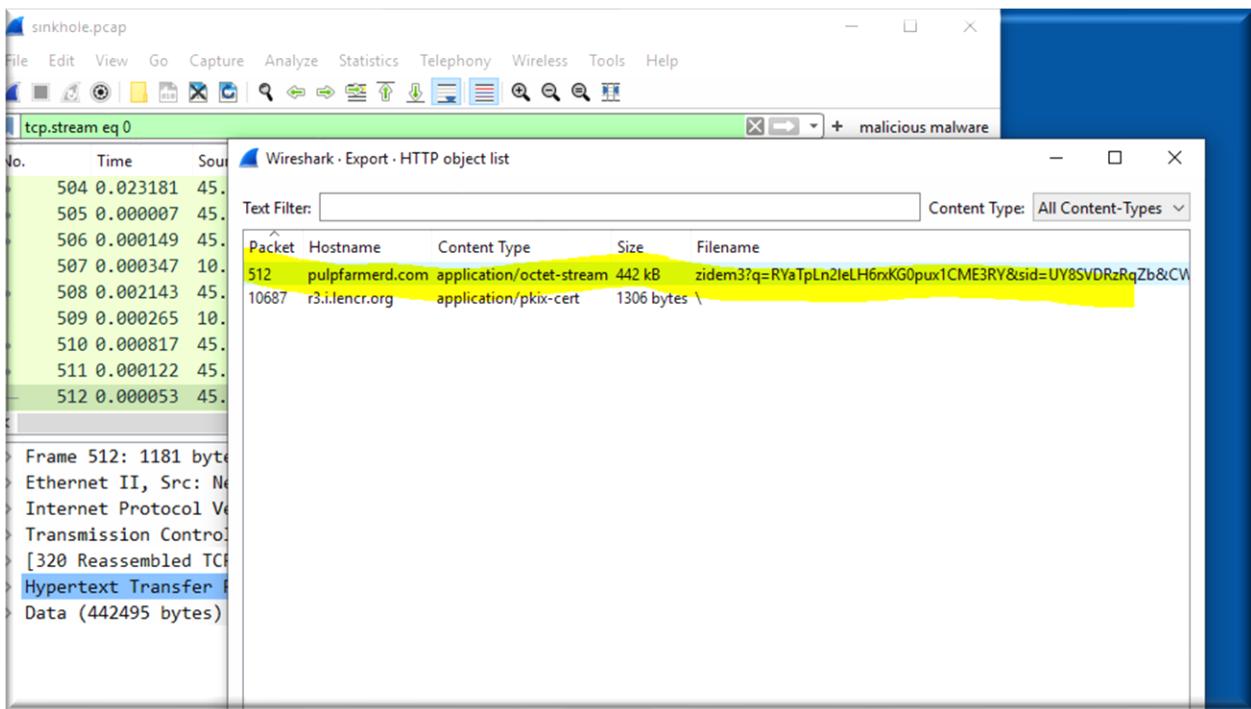
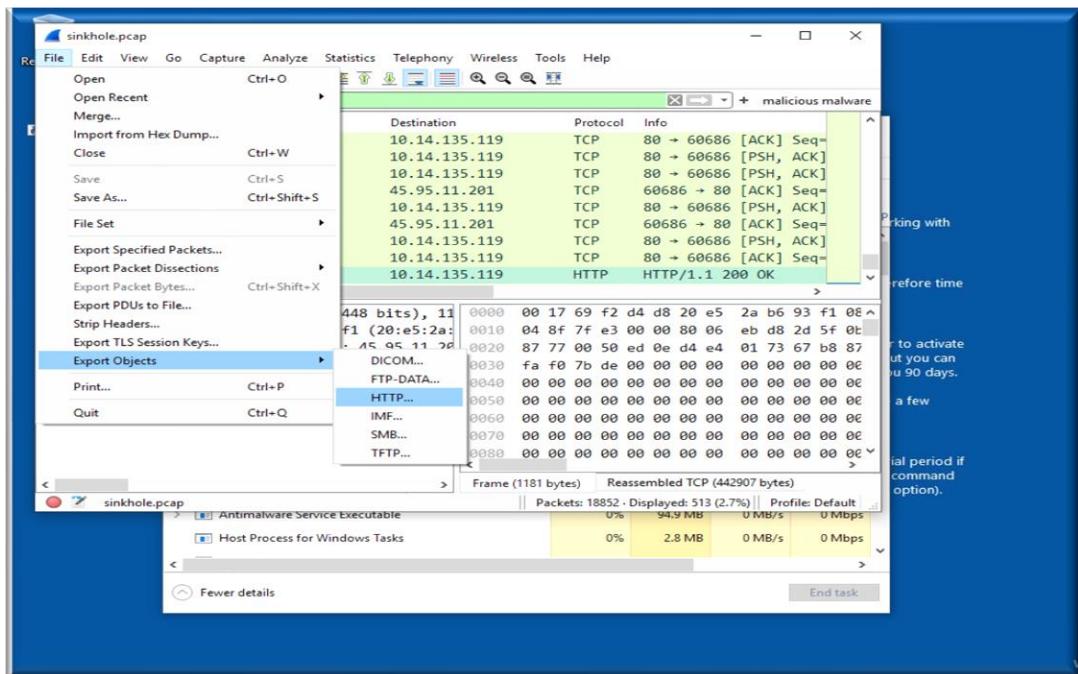
Entire conversation (443 kB) Show data as ASCII Stream 0

Find:  Filter Out This Stream Print Save as... Back Close Help

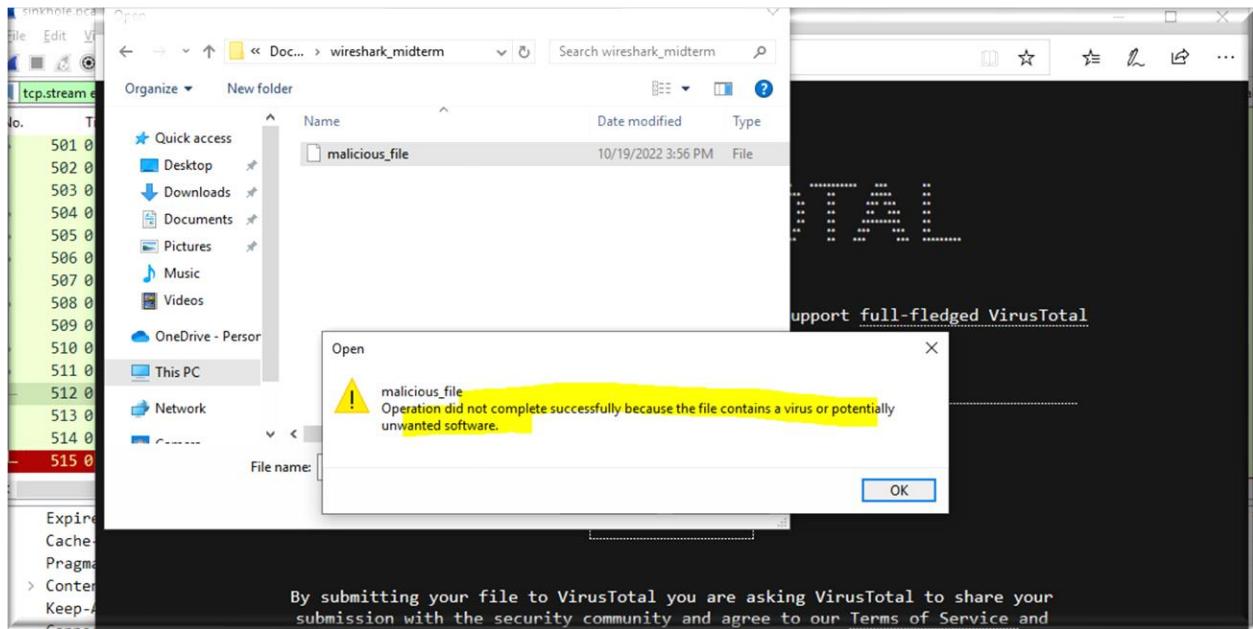
14. The file downloaded as an attachment and name is “zidem6”

We can observe that this is malicious file by witnessing the malware signature in the screenshot above which starts with “MZ....”

15. Moving on my investigation regarding the malicious file, I exported the file using Wireshark by going to File -> Export Objects -> HTTP. You will see a dialog similar to the one shown below:



16. I saved it and named it as a malicious file. I tried to open this malicious file in <http://www.virustotal.com>. It popped up with the below error message.



17. The file was deleted in sometime by the antivirus installed on my VM.
18. By this I conclude that, the octet stream content in the form of attachment was a virus file infecting the client machine located at 10.14.135.119.
19. We have 34 packets in total with HTTP protocol. Refer screenshot below:

No.	Time	Source	Destination	Protocol	Info
1364	195.765...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
1366	0.043058	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
3232	2700.72...	10.14.135.119	8.249.245.254	HTTP	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?2ea6...
3234	0.095124	8.249.245.254	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
4136	994.250...	10.14.135.119	67.26.233.254	HTTP	GET /msdownload/update/v3/static/trustedr/en/authrootstl.cab?cd10...
4138	0.089716	67.26.233.254	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
4139	0.008017	10.14.135.119	67.26.233.254	HTTP	GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.ca...
4141	0.119873	67.26.233.254	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
4918	1598.11...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
4920	0.050086	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
5720	1803.96...	10.14.135.119	209.197.3.8	HTTP	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?a30...
5722	0.089210	209.197.3.8	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
8135	3595.81...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
8137	0.034428	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
8557	894.636...	10.14.135.119	72.21.81.240	HTTP	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?b4cb...
8559	0.030900	72.21.81.240	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
8806	450.828...	10.14.135.119	72.21.81.240	HTTP	GET /msdownload/update/v3/static/trustedr/en/authrootstl.cab?5787...
8808	0.033938	72.21.81.240	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
8809	0.007965	10.14.135.119	72.21.81.240	HTTP	GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.ca...
8811	0.135805	72.21.81.240	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
18341	2254.77...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
18343	0.072808	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
18684	353.364...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
18687	0.036282	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 200 OK (application/pkix-cert)
18693	2476.57...	10.14.135.119	8.253.189.126	HTTP	GET /msdownload/update/v3/static/trustedr/en/authrootstl.cab?d698...
18695	0.026497	8.253.189.126	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
18696	0.009100	10.14.135.119	8.253.189.126	HTTP	GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.ca...
18698	0.018209	8.253.189.126	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
18339	2569.80...	10.14.135.119	104.94.77.31	HTTP	GET / HTTP/1.1
18341	0.033550	104.94.77.31	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified
18348	0.108483	10.14.135.119	8.249.3.254	HTTP	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?6c41...
18350	0.023545	8.249.3.254	10.14.135.119	HTTP	HTTP/1.1 304 Not Modified

20. I tried to follow TCP stream for every HTTP GET request and found 2 HTTP objects as shown in the screenshot below:

Packet	Hostname	Content Type	Size	Filename
512	pulpfarmerd.com	application/octet-stream	442 kB	zidem3?q=RYaTpLn2leLH6rxKG0pxu1CME3RY&sid=UY8SVDRzRqZb&CWPjmycHi=iF0l26&sid=YGrkJD4n&q=mbdtF5ziKWJczkstBIW0PBT7la&cti
10687	r3.i.lencr.org	application/pkix-cert	1306 bytes	\

1 is the malicious file and 2<sup>nd</sup> is the pkix certificate which seems to be clean.

21. TCP reset abrupt disconnection RST, ACK error in the packet line number 515.

because the server daemon is not running or because of any reason the server is not accepting our requests, then we could have seen RST ACK packets.

This error states that the server is out of service or perhaps the server is not supposed to respond to our requests.

Refer screenshot below:

No.	Source	Destination	Length	Protocol	Time	Info
512	45.95.11.201	10.14.135.119	1181	HTTP	2.460193	HTTP/1.1 200 OK
513	10.14.135.119	45.95.11.201	54	TCP	2.460332	60686 → 80 [ACK] Seq=530 Ack=440321 Win
514	10.14.135.119	45.95.11.201	54	TCP	2.460463	60686 → 80 [ACK] Seq=530 Ack=442908 Win
<b>515</b>	<b>10.14.135.119</b>	<b>45.95.11.201</b>	<b>54</b>	<b>TCP</b>	<b>2.646405</b>	<b>60686 → 80 [RST, ACK] Seq=530 Ack=442908</b>
516	10.14.135.119	10.14.135.5	66	TCP	8.541501	60690 → 135 [SYN] Seq=0 Win=64240 Len=0

```

Sequence Number: 530      (relative sequence number)
Sequence Number (raw): 1740146447
[Next Sequence Number: 530      (relative sequence number)]
Acknowledgment Number: 442908      (relative ack number)
Acknowledgment number (raw): 3571713498
0101 .... = Header Length: 20 bytes (5)
> Flags: 0x014 (RST, ACK)
Window: 0
[Calculated window size: 0]
[Window size scaling factor: -2 (no window scaling used)]
Checksum: 0x2e3e [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
> [Timestamps]

```

22. After drop, again we see 3 way TCP communication and the client sends an ACK to the server in packet 518.

514	2.460463	10.14.135.119	45.95.11.201	TCP	54 60686 → 80 [ACK] Seq=530 Ack=442908 Win=65535 Len=0
<b>515</b>	<b>2.646405</b>	<b>10.14.135.119</b>	<b>45.95.11.201</b>	<b>TCP</b>	<b>54 60686 → 80 [RST, ACK] Seq=530 Ack=442908 Win=0 Len=0</b>
516	8.541501	10.14.135.119	10.14.135.5	TCP	66 60690 → 135 [SYN] Seq=0 Win=64240 Len=0 MSS=1468 WS=256 S
517	8.541743	10.14.135.5	10.14.135.119	TCP	66 135 → 60690 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1468 WS=256 S
518	8.541968	10.14.135.119	10.14.135.5	TCP	54 60690 → 135 [ACK] Seq=1 Ack=1 Win=262656 Len=0
519	8.542027	10.14.135.119	10.14.135.5	DCERPC	214 Bind: call_id: 2, Fragment: Single, 3 context items: EPMv
520	8.542514	10.14.135.5	10.14.135.119	DCERPC	162 Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 ms
521	8.542724	10.14.135.119	10.14.135.5	EPM	222 Map request, LSARPC, 32bit NDR
522	8.543028	10.14.135.5	10.14.135.119	EPM	226 Map response, LSARPC, 32bit NDR

23. Post acknowledgement we see DCERPC protocol running on top of TCP using default TCP port of 135.

No.	Time	Source	Destination	Protocol	Info
512	0.000053	45.95.11.201	10.14.135.119	HTTP	HTTP/1.1 200 OK
513	0.000139	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=440321 Win=65535 Len=0
514	0.000131	10.14.135.119	45.95.11.201	TCP	60686 → 80 [ACK] Seq=530 Ack=442908 Win=65535 Len=0
515	0.185942	10.14.135.119	45.95.11.201	TCP	60686 → 80 [RST, ACK] Seq=530 Ack=442908 Win=0 Len=0
516	5.895096	10.14.135.119	10.14.135.5	TCP	60690 → 135 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
517	0.000242	10.14.135.5	10.14.135.119	TCP	135 → 60690 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25..
518	0.000225	10.14.135.119	10.14.135.5	TCP	60690 → 135 [ACK] Seq=1 Ack=1 Win=262656 Len=0
519	0.000059	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: EPMv4 V3.0 (...)
520	0.0000487	10.14.135.5	10.14.135.119	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 max_recv: ...
521	0.000210	10.14.135.119	10.14.135.5	EPM	Map request, LSARPC, 32bit NDR
522	0.000304	10.14.135.5	10.14.135.119	EPM	Map response, LSARPC, 32bit NDR
523	0.000589	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
524	0.000212	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=...
525	0.000190	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
526	0.000191	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: LSARPC V0.0 ...
527	0.000168	10.14.135.5	10.14.135.119	DCERPC	Bind ack: call id: 2. Fragment: Single, max xmit: 5840 max recv: ...

```
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 200
Identification: 0x83da (33754)
> 010. .... = Flags: 0x2, Don't fragment
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: TCP (6)
Header Checksum: 0x53bd [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.14.135.119
Destination Address: 10.14.135.5
▼ Transmission Control Protocol, Src Port: 60690, Dst Port: 135, Seq: 1, Ack: 1, Len: 1
  Source Port: 60690
  Destination Port: 135
  [Stream index: 1]
  [Conversation completeness: Complete, WITH_DATA (31)]
```

```
0000 a4 1f 72 c2 09 6a 00 17 69 f2 d4 d8 08 00 45
0010 00 c8 83 da 40 00 80 06 53 bd 0a 0e 87 77 0a
0020 87 05 ed 12 00 87 c3 82 1f 90 09 21 25 32 50
0030 04 02 dc ae 00 00 05 00 0e 03 10 00 00 00 a0
0040 00 00 02 00 00 00 d0 16 d0 16 00 00 00 00 03
0050 00 00 00 00 01 00 08 83 af e1 1f 5d c9 11 91
0060 08 00 2b 14 a0 fa 03 00 00 00 04 5d 88 8a eb
0070 c9 11 9f e8 08 00 2b 10 48 60 02 00 00 00 01
0080 01 00 08 83 af e1 1f 5d c9 11 91 a4 08 00 2b
0090 a0 fa 03 00 00 00 33 05 71 71 ba be 37 49 83
00a0 b5 db ef 9c cc 36 01 00 00 00 02 00 01 00 08
00b0 af e1 1f 5d c9 11 91 a4 08 00 2b 14 a0 fa 03
00c0 00 00 2c 1c b7 6c 12 98 40 45 03 00 00 00 00
00d0 00 00 01 00 00 00
```

24. Connection oriented DCE/RPC uses TCP as its transport protocol.

DCE/RPC is a specification for a remote procedure call mechanism that defines both APIs and an over-the-network protocol.

25. A DCE/RPC server's endpoint mapper (EPMAP) will listen for incoming calls. A client will call this endpoint mapper and ask for a specific interface, which will be accessed on a different connection. After that, the client can request calls to the server.

The well known TCP port for DCE/RPC EPMAP is 135. This transport is called ncacn\_ip\_tcp

No.	Time	Source	Destination	Protocol	Info
524	0.000212	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=...
525	0.000190	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
526	0.000191	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: LSARPC V0.0 ...
527	0.000168	10.14.135.5	10.14.135.119	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 max_recv: ...
528	0.000237	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
529	0.000447	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
530	0.000435	10.14.135.119	10.14.135.5	DCERPC	Alter_context: call_id: 3, Fragment: Single, 1 context items: LSA...
531	0.000169	10.14.135.5	10.14.135.119	DCERPC	Alter_context_resp: call_id: 3, Fragment: Single, max_xmit: 5840 ...
532	0.000196	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
533	0.000351	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
534	0.017704	10.14.135.5	10.14.135.119	TCP	[TCP Retransmission] 135 → 60690 [PSH, ACK] Seq=109 Ack=329 Win=2...
535	0.000174	10.14.135.119	10.14.135.5	TCP	60690 → 135 [ACK] Seq=329 Ack=281 Win=262400 Len=0 SLE=109 SRE=281
536	0.030399	10.14.135.5	10.14.135.119	TCP	[TCP Retransmission] 49667 → 60691 [PSH, ACK] Seq=389 Ack=937 Win=...
537	0.000168	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=937 Ack=573 Win=2101760 Len=0 SLE=389 SRE=...
538	0.167230	10.14.135.119	10.14.135.5	DNS	Standard query 0x3291 A microsoft.com
539	0.000238	10.14.135.5	10.14.135.119	DNS	Standard query response 0x3291 A microsoft.com A 40.76.4.15 A 40...

Call ID: 3
Alloc hint: 160
Context ID: 1
Opnum: 77
[Response in frame: 533]
Auth Info: NETLOGON Secure Channel, Packet privacy, AuthContextId(1)
Auth type: NETLOGON Secure Channel (68)
Auth level: Packet privacy (6)
Auth pad len: 0
Auth Rsrvd: 0
Auth Context ID: 1
> Secure Channel Verifier
Local Security Authority, lsa_LookupNames4
Operation: lsa_LookupNames4 (77)
[Response in frame: 533]
Encrypted stub data: a3cd7e05d8652a06f8fed2573596be2fa68c561ddce275ef28158ed8469ee6

No.	Time	Source	Destination	Protocol	Info
524	0.000212	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=...
525	0.000190	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
526	0.000191	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: LSARPC V0.0 ...
527	0.000168	10.14.135.5	10.14.135.119	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 max_recv: ...
528	0.000237	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
529	0.000447	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
530	0.000435	10.14.135.119	10.14.135.5	DCERPC	Alter_context: call_id: 3, Fragment: Single, 1 context items: LSA...
531	0.000169	10.14.135.5	10.14.135.119	DCERPC	Alter_context_resp: call_id: 3, Fragment: Single, max_xmit: 5840 ...
532	0.000196	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
533	0.000351	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
534	0.017704	10.14.135.5	10.14.135.119	TCP	[TCP Retransmission] 135 → 60690 [PSH, ACK] Seq=109 Ack=329 Win=2...
535	0.000174	10.14.135.119	10.14.135.5	TCP	60690 → 135 [ACK] Seq=329 Ack=281 Win=262400 Len=0 SLE=109 SRE=281
536	0.030399	10.14.135.5	10.14.135.119	TCP	[TCP Retransmission] 49667 → 60691 [PSH, ACK] Seq=389 Ack=937 Win=...
537	0.000168	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=937 Ack=573 Win=2101760 Len=0 SLE=389 SRE=...
538	0.167230	10.14.135.119	10.14.135.5	DNS	Standard query 0x3291 A microsoft.com
539	0.000238	10.14.135.5	10.14.135.119	DNS	Standard query response 0x3291 A microsoft.com A 40.76.4.15 A 40...

Context ID: 1
Cancel count: 0
[Opnum: 77]
[Request in frame: 532]
[Time from request: 0.000351000 seconds]
Auth Info: NETLOGON Secure Channel, Packet privacy, AuthContextId(1)
Auth type: NETLOGON Secure Channel (68)
Auth level: Packet privacy (6)
Auth pad len: 8
Auth Rsrvd: 0
Auth Context ID: 1
> Secure Channel Verifier
Local Security Authority, lsa_LookupNames4
Operation: lsa_LookupNames4 (77)
[Request in frame: 532]
Encrypted stub data: 4361308e81f6c69a04ebe1eb61801b26bd4d46fa1bd4defe3f1f7354ca4305

No.	Time	Source	Destination	Protocol	Info
525	0.000190	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
526	0.000191	10.14.135.119	10.14.135.5	DCERPC	Bind: call_id: 2, Fragment: Single, 3 context items: LSARPC V0.0 ...
527	0.000168	10.14.135.5	10.14.135.119	DCERPC	Bind_ack: call_id: 2, Fragment: Single, max_xmit: 5840 max_recv: ...
528	0.000237	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
529	0.000447	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
530	0.000435	10.14.135.119	10.14.135.5	DCERPC	Alter_context: call_id: 3, Fragment: Single, 1 context items: LSA...
531	0.000169	10.14.135.5	10.14.135.119	DCERPC	Alter_context_resp: call_id: 3, Fragment: Single, max_xmit: 5840 ...
532	0.000196	10.14.135.119	10.14.135.5	LSARPC	lsa_LookupNames4 request
533	0.000351	10.14.135.5	10.14.135.119	LSARPC	lsa_LookupNames4 response
536	0.048277	10.14.135.5	10.14.135.119	TCP	[TCP Retransmission] 49667 → 60691 [PSH, ACK] Seq=389 Ack=937 Win...
537	0.000168	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=937 Ack=573 Win=2101760 Len=0 SLE=389 SRE...
594	10.074260	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [FIN, ACK] Seq=937 Ack=573 Win=2101760 Len=0
595	0.000218	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [ACK] Seq=573 Ack=938 Win=2101504 Len=0
596	0.000072	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [FIN, ACK] Seq=573 Ack=938 Win=2101504 Len=0
597	0.000136	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=938 Ack=574 Win=2101760 Len=0

26. I think, post malware transmission from webserver, domain security policies and account database information are trying to be captured by the intruder since DCERPC /LSARPC have vulnerabilities. These are followed by PSH, ACK and eventually FIN, ACK closing the channel
27. Moving further, I checked total DNS packets and found 352 in total with standard query and response providing IP addresses to below domains:

[amazon.com](http://amazon.com)  
[arc.msn.com](http://arc.msn.com)  
[bing.com](http://bing.com)  
[client.wns.windows.com](http://client.wns.windows.com)  
[ctldl.windowsupdate.com](http://ctldl.windowsupdate.com)  
[dns.msftncsi.com](http://dns.msftncsi.com)  
[evoke-windowsservices-tas.msedge.net](http://evoke-windowsservices-tas.msedge.net)  
[google.com](http://google.com)  
[introwebsites.com](http://introwebsites.com)  
[login.live.com](http://login.live.com)  
[login.microsoftonline.com](http://login.microsoftonline.com)  
[microsoft.com](http://microsoft.com)  
[myexternalip.com](http://myexternalip.com)  
[nexusrules.officeapps.live.com](http://nexusrules.officeapps.live.com)  
[pulpfarmerd.com](http://pulpfarmerd.com)  
[r3.i.lencr.org](http://r3.i.lencr.org)  
[ris.api.iris.microsoft.com](http://ris.api.iris.microsoft.com)  
[settings-win.data.microsoft.com](http://settings-win.data.microsoft.com)  
[update.googleapis.com](http://update.googleapis.com)  
[v10.events.data.microsoft.com](http://v10.events.data.microsoft.com)  
[wpad.rusticmoment.com](http://wpad.rusticmoment.com)  
[www.amazon.com](http://www.amazon.com)  
[www.bing.com](http://www.bing.com)  
[www.microsoft.com](http://www.microsoft.com)  
[www.yahoo.com](http://www.yahoo.com)  
[x1.c.lencr.org](http://x1.c.lencr.org)  
[yahoo.com](http://yahoo.com)

28. I have checked all the above domain in virus total and found

Domain	Virus total scan
<a href="#">amazon.com</a>	Clean
<a href="#">arc.msn.com</a>	Clean
<a href="#">bing.com</a>	Clean
<a href="#">client.wns.windows.com</a>	Clean
<a href="#">ctldl.windowsupdate.com</a>	Clean
<a href="#">dns.msftncsi.com</a>	Clean
<a href="#">evoke-windowsservices-tas.msedge.net</a>	Clean
<a href="#">google.com</a>	Clean
<a href="#">introwebsites.com</a>	Malicious
<a href="#">login.live.com</a>	Clean
<a href="#">login.microsoftonline.com</a>	Clean
<a href="#">microsoft.com</a>	Clean
<a href="#">myexternalip.com</a>	Malicious
<a href="#">nexusrules.officeapps.live.com</a>	Clean
<a href="#">pulpfarmerd.com</a>	Malicious
<a href="#">r3.i.lencr.org</a>	Clean
<a href="#">ris.api.iris.microsoft.com</a>	Clean
<a href="#">settings-win.data.microsoft.com</a>	Clean
<a href="#">update.googleapis.com</a>	Clean
<a href="#">v10.events.data.microsoft.com</a>	Clean
<a href="#">wpad.rusticmoment.com</a>	Clean
<a href="#">www.amazon.com</a>	Clean
<a href="#"><u>www.bing.com</u></a>	Clean
<a href="#">www.microsoft.com</a>	Clean
<a href="#"><u>www.yahoo.com</u></a>	Clean
<a href="#">x1.c.lencr.org</a>	Clean
<a href="#">yahoo.com</a>	Clean

29. Now let's observe link layer protocol i.e. ARP packets.

*The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link-layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address.*

30. There are 730 ARP packets observed in wireshark. Few are shown below:

No.	Time	Source	Destination	Protocol	Info
602	0.000000	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
603	0.000482	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
605	47.400023	Cymphoni_f2:d4:d8	Netgear_b6:93:f1	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
1187	12.652971	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	Who has 10.14.135.5? Tell 10.14.135.119
1188	0.000314	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	10.14.135.5 is at a4:1f:72:c2:09:6a
1225	36.501037	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	Who has 10.14.135.5? Tell 10.14.135.119
1226	0.000470	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	10.14.135.5 is at a4:1f:72:c2:09:6a
1227	0.431638	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
1228	0.000257	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
1368	60.001339	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
1369	0.000218	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
1370	0.068277	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	Who has 10.14.135.5? Tell 10.14.135.119
1371	0.000226	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	10.14.135.5 is at a4:1f:72:c2:09:6a
1420	54.122462	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5

31. Let's look into 602-605 packet numbers.

No.	Time	Source	Destination	Protocol	Info
599	0.000357	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=560 Ack=6876 Win=63239 Len=0
600	0.000231	10.14.135.119	23.47.169.181	TCP	60693 → 443 [FIN, ACK] Seq=560 Ack=6876 Win=63239 Len=0
601	0.000170	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=6876 Ack=561 Win=64239 Len=0
602	14.640728	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
603	0.000482	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
604	47.132869	10.14.135.119	87.120.37.231	TCP	60694 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
605	0.267154	Cymphoni_f2:d4:d8	Netgear_b6:93:f1	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
606	0.000122	87.120.37.231	10.14.135.119	TCP	443 → 60694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
607	0.000329	10.14.135.119	87.120.37.231	TCP	60694 → 443 [ACK] Seq=1 Ack=1 Win=65535 Len=0
608	0.011259	10.14.135.119	87.120.37.231	TLSv1.2	Client Hello
609	0.000227	87.120.37.231	10.14.135.119	TCP	443 → 60694 [ACK] Seq=1 Ack=150 Win=64240 Len=0
610	0.215338	87.120.37.231	10.14.135.119	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
611	0.000290	10.14.135.119	87.120.37.231	TCP	60694 → 443 [ACK] Seq=150 Ack=899 Win=65535 Len=0
612	0.025840	10.14.135.119	87.120.37.231	TLSv1.2	Client Key Exchange, Change Cipher Spec, Finished, Handshake

```
> Frame 602: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
> Ethernet II, Src: Dell_c2:09:6a (a4:1f:72:c2:09:6a), Dst: Cymphoni_f2:d4:d8 (00:17:69:f2:d4:d8)
  Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Dell_c2:09:6a (a4:1f:72:c2:09:6a)
    Sender IP address: 10.14.135.5
    Target MAC address: Cymphoni_f2:d4:d8 (00:17:69:f2:d4:d8)
    Target IP address: 10.14.135.119
```

No.	Time	Source	Destination	Protocol	Info
599	0.000357	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=560 Ack=6876 Win=63239 Len=0
600	0.000231	10.14.135.119	23.47.169.181	TCP	60693 → 443 [FIN, ACK] Seq=560 Ack=6876 Win=63239 Len=0
601	0.000170	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=6876 Ack=561 Win=64239 Len=0
602	14.640728	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
603	0.000482	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
604	47.132869	10.14.135.119	87.120.37.231	TCP	60694 → 443 [SYN] Seq=0 Win=5535 MSS=1460 WS=256 SACK_PERM
605	0.267154	Cymphoni_f2:d4:d8	Netgear_b6:93:f1	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
606	0.000128	87.120.37.231	10.14.135.119	TCP	443 → 60694 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
607	0.000329	10.14.135.119	87.120.37.231	TCP	60694 → 443 [ACK] Seq=1 Ack=1 Win=65535 Len=0
608	0.011259	10.14.135.119	87.120.37.231	TLSv1.2	Client Hello
609	0.000227	87.120.37.231	10.14.135.119	TCP	443 → 60694 [ACK] Seq=1 Ack=150 Win=64240 Len=0
610	0.215338	87.120.37.231	10.14.135.119	TLSv1.2	Server Hello, Certificate, Server Key Exchange, Server Hello Done
611	0.000290	10.14.135.119	87.120.37.231	TCP	60694 → 443 [ACK] Seq=150 Ack=899 Win=65535 Len=0
612	0.025040	10.14.135.119	87.120.37.231	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

```

> Frame 603: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
> Ethernet II, Src: Cymphoni_f2:d4:d8 (00:17:69:f2:d4:d8), Dst: Dell_c2:09:6a (a4:1f:72:c2:09:6a)
  Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Cymphoni_f2:d4:d8 (00:17:69:f2:d4:d8)
    Sender IP address: 10.14.135.119
    Target MAC address: Dell_c2:09:6a (a4:1f:72:c2:09:6a)
    Target IP address: 10.14.135.5

```

32. Dell\_c2:09:6a and Cymphoni\_f2:d4:d8 successfully obtains one another's physical(MAC address) by broadcasting ARP request onto the network.
33. A value of 1 indicates this ARP packet is a *Request*, or a value of 2 would indicate this ARP packet is a *Response*
34. There are around 4072 packets with TLS v1.2 protocol. TCP is 3 way handshake whereas TLS v1.2 is 5 way handshake.
35. Let's go through 1 instance of TLS handshake below connecting to [www.microsoft.com](http://www.microsoft.com)

No.	Time	Source	Destination	Protocol	Info
563	0.024492	10.14.135.119	10.14.135.5	DNS	Standard query 0x83a3 A www.microsoft.com
564	0.030642	10.14.135.119	104.215.148.63	TCP	60692 → 443 [ACK] Seq=519 Ack=7305 Win=63747 Len=0
565	0.020499	10.14.135.5	10.14.135.119	DNS	Standard query response 0x83a3 A www.microsoft.com CNAME www.mic...
566	0.001658	10.14.135.119	23.47.169.181	TCP	60693 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
567	0.093716	23.47.169.181	10.14.135.119	TCP	443 → 60693 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
568	0.000260	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
569	0.000353	10.14.135.119	23.47.169.181	TLSv1.2	Client Hello
570	0.000120	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=1 Ack=203 Win=64240 Len=0
571	0.113097	23.47.169.181	10.14.135.119	TLSv1.2	Server Hello
572	0.001527	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=1377 Ack=203 Win=64240 Len=1460 [TCP segment ...]
573	0.000020	23.47.169.181	10.14.135.119	TLSv1.2	Certificate
574	0.000183	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=203 Ack=4097 Win=64240 Len=0
575	0.000189	23.47.169.181	10.14.135.119	TCP	443 → 60693 [PSH, ACK] Seq=4097 Ack=203 Win=64240 Len=1376 [TCP segment ...]
576	0.000134	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=203 Ack=5473 Win=62864 Len=0
577	0.076372	23.47.169.181	10.14.135.119	TLSv1.2	Certificate Status, Server Key Exchange, Server Hello Done
578	0.004290	10.14.135.119	23.47.169.181	TLSv1.2	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
579	0.000222	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=5874 Ack=329 Win=64240 Len=0

```

> Frame 566: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: Cymphoni_f2:d4:d8 (00:17:69:f2:d4:d8), Dst: Netgear_b6:93:f1 (20:e5)
  Internet Protocol Version 4, Src: 10.14.135.119, Dst: 23.47.169.181
    Version: 4
    Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x27ae (10158)
    Flags: 0x0 (Don't fragment)
    Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x80ac [validation disabled]
      Header checksum status: Unverified

```

36. We see the DNS request <-> response taking place yielding IP address 23.47.169.181.

37. HTTP uses TCP handshake. HTTPS uses HTTP over TLS wherein it exchanges certificates, keys and client/server Hello's. In our case it is going to be HTTPS.

38. Once the TCP three-way handshake is done. The TLS handshake will kick in with client hello.

The first step to start a TSL handshake with 'Client Hello' from 10.14.135.119

```
Content Type: Handshake (22)
Version: TLS 1.2 (0x0303)
Length: 197
└ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 193
    Version: TLS 1.2 (0x0303)
    > Random: 6182c9fe2b049cf88db91835548cc120c91f0efb0b45f88dc6c9ff80966935ef
    Session ID Length: 0
    Cipher Suites Length: 38
    > Cipher Suites (19 suites)
    Compression Methods Length: 1
    > Compression Methods (1 method)
```

39. After the server receives the client hello, it will examine the supported TLS versions and cipher suites sent by the client. The server will select the highest supported TLS version by both client and the server – TLS v1.2 in our case.

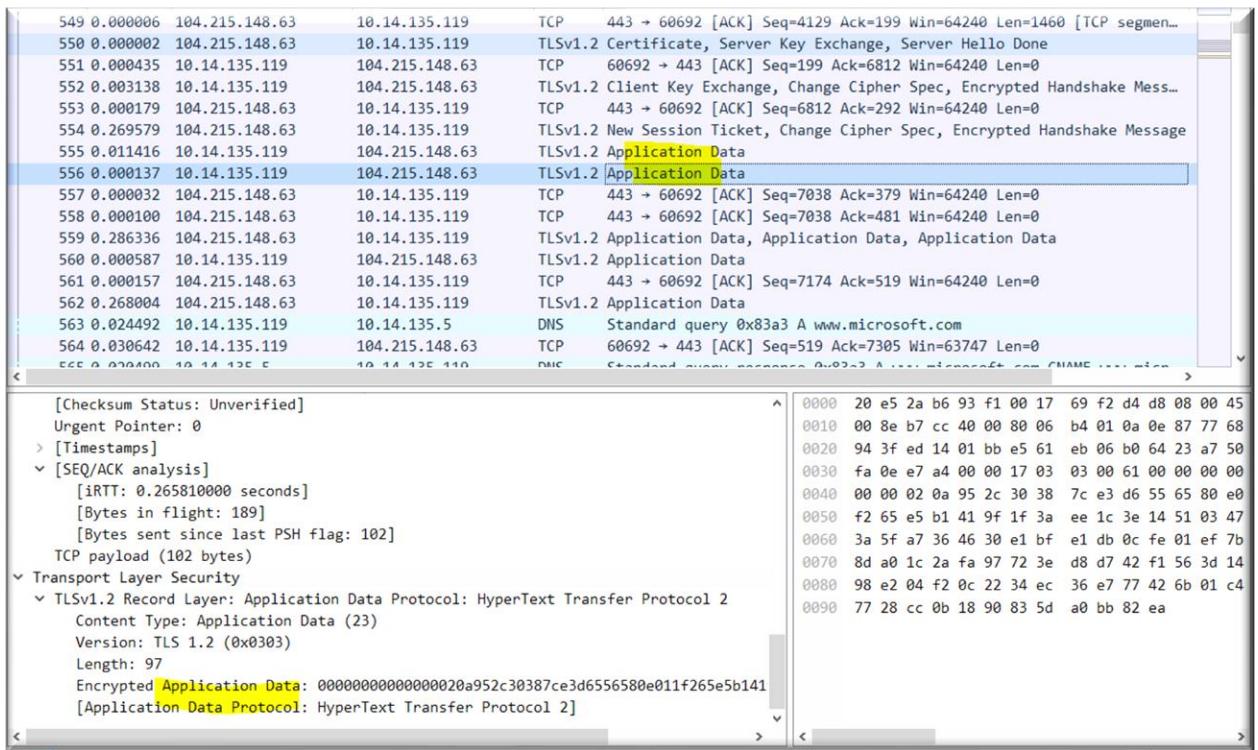
```
└ Transport Layer Security
    └ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 78
    └ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 74
        Version: TLS 1.2 (0x0303)
        > Random: 8889026397672f71a2df6a478a751bf51a7a966e41c62306444f574e47524401
        Session ID Length: 0
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
        Compression Method: null (0)
        Extensions Length: 21
```

40. Sending Certificate, Server Encryption Key, Server Hello Done to the client

41. The server will send the certificate, server encrypted key, and server hello done message to the client.

42. Sending Certificate, Server Encryption Key, Server Hello Done to the client in Wireshark

43. Receiving Client encrypted key, change cipher spec, and encrypted handshake message from client
44. After the client receives the server encrypted key. It will respond with the client encrypted key. It also sends change cipher spec. What it means is that it has enough information to start encrypted communication, and it is going to send the data with encryption from now onwards. Till now, the communication was plain text. After this, communication will happen with encryption. After the server receives the change cipher spec message, it expects encrypted data from the client.
45. The final step in TLS handshake — sending change cipher spec and the final handshake message to the client.
46. This is going to be the last message that the server is going to send, which includes change cipher spec and finishing the message. This indicates that all feature messages are going to be encrypted.
47. Now that the TLS handshake is done we can see the Encrypted application data, and it is shown in the screenshot below:



48. One more interesting thing I found is that, there is an Encrypted alert in the info section of the detailed view section. This means that wireshark cannot decrypt the data. Refer the screenshot below:

593 0.000210	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [ACK] Seq=573 Ack=938 Win=2101504 Len=0
596 0.000072	10.14.135.5	10.14.135.119	TCP	49667 → 60691 [FIN, ACK] Seq=573 Ack=938 Win=2101504 Len=0
597 0.000136	10.14.135.119	10.14.135.5	TCP	60691 → 49667 [ACK] Seq=938 Ack=574 Win=2101760 Len=0
598 12.442181	23.47.169.181	10.14.135.119	TLSv1.2	Encrypted Alert
599 0.000357	10.14.135.119	23.47.169.181	TCP	60693 → 443 [ACK] Seq=560 Ack=6876 Win=63239 Len=0
600 0.000231	10.14.135.119	23.47.169.181	TCP	60693 → 443 [FIN, ACK] Seq=560 Ack=6876 Win=63239 Len=0
601 0.000170	23.47.169.181	10.14.135.119	TCP	443 → 60693 [ACK] Seq=6876 Ack=561 Win=64239 Len=0
602 14.640728	Dell_c2:09:6a	Cymphoni_f2:d4:d8	ARP	Who has 10.14.135.119? Tell 10.14.135.5
603 0.000482	Cymphoni_f2:d4:d8	Dell_c2:09:6a	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
604 47.132869	10.14.135.119	87.120.37.231	TCP	60694 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
605 0.267154	Cymphoni_f2:d4:d8	Netgear_b6:93:f1	ARP	10.14.135.119 is at 00:17:69:f2:d4:d8
606 0.001122	87.120.37.231	10.14.135.119	TCP	443 → 60694 [SYN] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Checksum: 0x3a6d [unverified]  
[Checksum Status: Unverified]  
Urgent Pointer: 0  
> [Timestamps]  
▼ [SEQ/ACK analysis]  
[IRTT: 0.093976000 seconds]  
[Bytes in flight: 31]  
[Bytes sent since last PSH flag: 31]  
TCP payload (31 bytes)  
▼ Transport Layer Security  
▼ TLSv1.2 Record Layer: Encrypted Alert  
Content Type: Alert (21)  
Version: TLS 1.2 (0x0303)  
Length: 26  
Alert Message: Encrypted Alert

0000	00 17 69 f2 d4 d8 20 e5 2a b6 93 f1 08 00 45
0010	00 47 80 12 00 00 80 06 68 35 17 2f a9 b5 0a
0020	87 77 01 bb ed 15 70 0d 2e 7d db 8b 1b 0c 50
0030	f4 f0 3a 6d 00 00 15 03 03 00 1a f5 76 b4 85
0040	e2 43 e4 ec 6b fb 04 ff 42 fe 36 a5 98 0a 07
0050	b6 f1 25 dc 44

49. Now let's observe the LDAP authentication method from packet number 1947.
50. Authentication is supplied in the "LDAP bind" operation. There are three different authentication methods.
51. We see sasl authentication method in these packets.
52. Simple Authentication and Security Layer, SASL is an extensible framework that makes it possible to plug almost any kind of authentication into LDAP. Beside authentication, it can also provide a data security layer offering data integrity and data confidentiality services. All authentication mechanisms may not support confidentiality (encryption)
53. The first 3 packets belong to TCP 3-way handshaking.

1947 0.002128	10.14.135.119	10.14.135.5	TCP	60721 → 389 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1948 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25...
1949 0.000191	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=1 Ack=1 Win=262656 Len=0
1950 0.001145	10.14.135.119	10.14.135.5	LDAP	searchRequest(1) "<ROOT>" baseObject

54. Once the TCP connection establishes, It simply creates a search request, searchRequest(1) and search result in received. Refer below screenshots:

1947 0.002128	10.14.135.119	10.14.135.5	TCP	60721 → 389 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1948 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25...
1949 0.000191	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=1 Ack=1 Win=262656 Len=0
1950 0.001145	10.14.135.119	10.14.135.5	LDAP	searchRequest(1) "<ROOT>" baseObject
1951 0.000380	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=1 Ack=351 Win=2102272 Len=1460 [TCP segment...]
1952 0.000003	10.14.135.5	10.14.135.119	LDAP	searchResEntry(1) "<ROOT>"   searchResDone(1) success [4 results]
1953 0.000194	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1954 0.000990	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1955 0.000002	10.14.135.119	10.14.135.5	LDAP	bindRequest(3) "<ROOT>" sasl
1956 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=2812 Ack=2406 Win=2102272 Len=0
1957 0.000883	10.14.135.5	10.14.135.119	LDAP	bindResponse(3) success
1958 0.000657	10.14.135.119	10.14.135.5	LDAP	SASL GSS-API Integrity: searchRequest(4) "CN=DESKTOP-UZZYREZ,CN=...

[PDU Size: 350]  
▼ Lightweight Directory Access Protocol  
▼ LDAPMessage searchRequest(1) "<ROOT>" baseObject  
messageID: 1  
protocolOp: searchRequest (3)  
▼ searchRequest  
baseObject:  
scope: baseObject (0)  
derefAliases: neverDerefAliases (0)  
sizeLimit: 0  
timeLimit: 120  
typesOnly: False  
> Filter: (objectclass=\*)  
▼ attributes: 15 items

0000	a4 1f 72 c2 09 6a 00 17 69 f2 d4 d8 08 00 2a
0010	01 86 84 90 40 00 80 06 52 49 0a 0e 87 77 00
0020	87 05 ed 31 01 85 35 a9 9f 7c d1 ec 2b 82 50
0030	04 02 c2 bf 00 00 30 84 00 00 01 58 02 01 00
0040	84 00 00 01 4f 04 00 0a 01 00 0a 01 00 02 00
0050	02 01 78 01 01 00 87 0b 6f 62 6a 65 63 74 60
0060	61 73 73 30 84 00 00 01 2b 04 11 73 75 62 70
0070	68 65 6d 63 53 75 62 65 6e 74 72 79 04 0d 60
0080	53 65 72 76 69 63 65 4e 61 6d 65 04 0e 6e 60
0090	69 6e 67 43 6f 6e 74 65 78 74 73 04 14 64 60
00a0	61 75 6c 74 4e 61 6d 69 6e 67 43 6f 6e 74 60
00b0	74 04 13 73 63 68 65 6d 61 4e 61 6d 69 6e 60
00c0	6f 6e 74 65 78 74 04 1a 63 6f 6e 66 69 67 60

1945 0.000320	10.14.135.119	10.14.135.5	DRSUAPI	DsUnbind request
1946 0.000290	10.14.135.5	10.14.135.119	DRSUAPI	DsUnbind response
1947 0.002128	10.14.135.119	10.14.135.5	TCP	60721 → 389 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1948 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25...
1949 0.000191	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=1 Ack=1 Win=262656 Len=0
1950 0.001145	10.14.135.119	10.14.135.5	LDAP	searchRequest(1) "<ROOT>" baseObject
1951 0.000380	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=1 Ack=351 Win=2102272 Len=1460 [TCP segment...]
1952 0.000003	10.14.135.5	10.14.135.119	LDAP	searchResEntry(1) "<ROOTS>"   searchResDone(1) success [4 results]
1953 0.000194	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1954 0.000990	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1955 0.000002	10.14.135.119	10.14.135.5	LDAP	bindRequest(3) "<ROOT>" sasl
1956 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=2812 Ack=2406 Win=2102272 Len=0
1957 0.000883	10.14.135.5	10.14.135.119	LDAP	bindResponse(3) success
1958 0.000657	10.14.135.119	10.14.135.5	LDAP	SASL GSS-API Integrity: searchRequest(4) "CN=DESKTOP-U2ZYREZ,CN=C...

55. The binding operation includes user name and password.

56. In our case it is sasl authentication. Hence, it will show that SASL is used for authentication.

Refer below screenshot:

1947 0.002128	10.14.135.119	10.14.135.5	TCP	60721 → 389 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1948 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=25...
1949 0.000191	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=1 Ack=1 Win=262656 Len=0
1950 0.000145	10.14.135.119	10.14.135.5	LDAP	searchRequest(1) "<ROOT>" baseObject
1951 0.000380	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=1 Ack=351 Win=2102272 Len=1460 [TCP segment...]
1952 0.000003	10.14.135.5	10.14.135.119	LDAP	searchResEntry(1) "<ROOTS>"   searchResDone(1) success [4 results]
1953 0.000194	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1954 0.000990	10.14.135.119	10.14.135.5	TCP	60721 → 389 [ACK] Seq=351 Ack=2812 Win=262656 Len=0
1955 0.000002	10.14.135.119	10.14.135.5	LDAP	bindRequest(3) "<ROOT>" sasl
1956 0.000228	10.14.135.5	10.14.135.119	TCP	389 → 60721 [ACK] Seq=2812 Ack=2406 Win=2102272 Len=0
1957 0.000883	10.14.135.5	10.14.135.119	LDAP	bindResponse(3) success
1958 0.000657	10.14.135.119	10.14.135.5	LDAP	SASL GSS-API Integrity: searchRequest(4) "CN=DESKTOP-U2ZYREZ,CN=C...

57. The packet also contains which version of LDAP it wants to use (in this case version 3).

0000 00 17 69 f2 d4 d8 a4 1f 72 c2 09 6a 08 00 ↴
0010 05 6f 49 1c 40 00 80 06 89 d4 0a 0e 87 05 ↴
0020 87 77 01 85 ed 31 d1 ec 31 36 35 a9 a0 da ↴
0030 20 14 b5 fc 00 00 31 36 2e 38 34 30 2e 31 2
0040 31 33 37 33 30 2e 33 2e 34 2e 31 30 04 17 ↴
0050 32 2e 38 34 30 2e 31 31 33 35 36 2e 31 2
0060 2e 31 35 30 34 04 17 31 2e 32 2e 38 34 30 2
[PDU Size: 2055] TCP segment data (595 bytes)
0070 > [2 Reassembled TCP Segments (2055 bytes): #1954(1460), #1955(595)]
Lightweight Directory Access Protocol
LDAPMessage bindRequest(3) "sasl"
messageID: 3
protocolOp: bindRequest (0)
bindRequest
version: 3
name:
authentication: sasl (3)
> end
0080 0000 a4 1f 72 c2 09 6a 00 17 69 f2 d4 d8 08 00 ↴
0090 02 7b 84 93 40 00 80 06 51 51 0a 0e 87 77 ↴
00a0 0020 87 05 ed 31 01 85 35 a9 a6 8e d1 ec 36 7d ↴
00b0 04 02 22 24 00 00 d4 a2 79 4f 46 73 f2 a9 e
00c0 66 a5 44 ae c2 70 5f 2b 05 7f 4a 66 be 22 1
00d0 de cd b9 91 6e fa 89 94 f0 02 0d 0b 24 74 e
00e0 00 13 81 e0 e3 e6 92 95 9c ac 93 9a 99 f3 t
00f0 0070 1e 0a 9b 7b 1f 0c e0 01 78 d6 ed 57 a4 82 e
0100 30 82 02 05 a0 03 02 01 12 a2 82 01 fc 04 e
0110 0080 f8 bf ed 9c 94 28 92 7a de 47 79 6d c7 d3 7
0120 00a0 90 1d 8c 16 75 85 c0 0c 78 1f 9f 02 e2 cb ↴

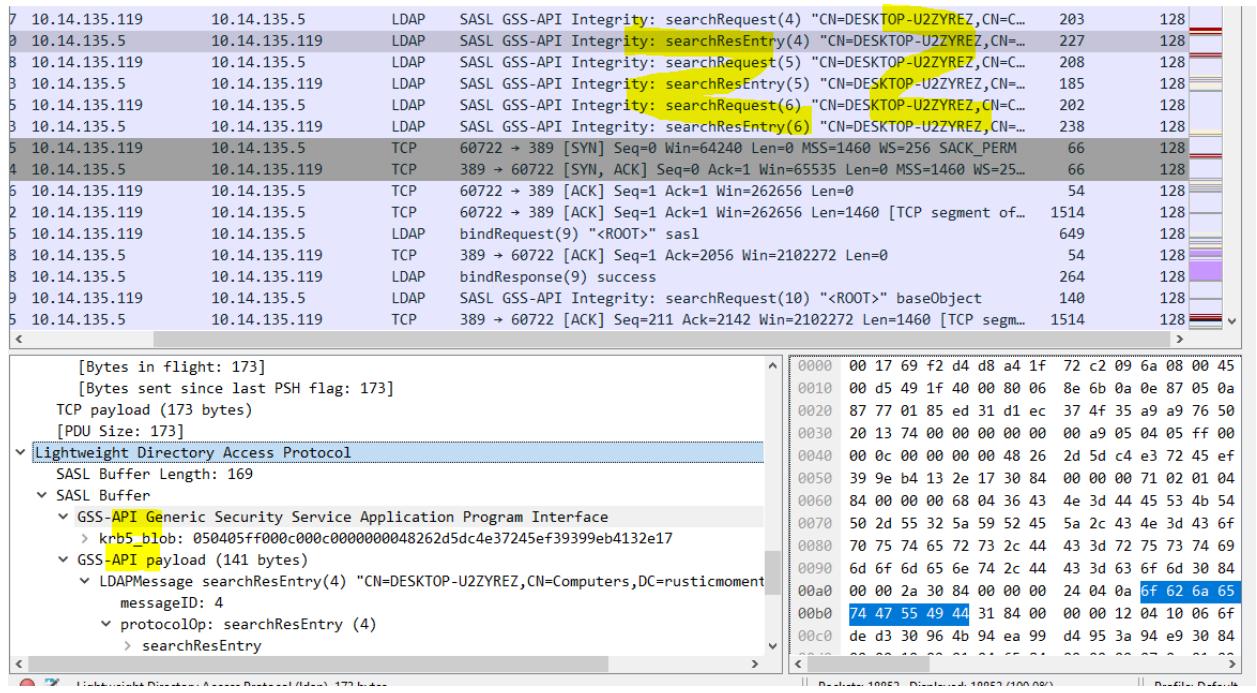
58. The server authenticates the client successfully and returns "success" as "resultCode" using bindResponse operation as shown in the screenshot above.

59. From this moment, the client can query the ldap server.

60. The client uses unbind operation to end the session

1978 0.000251 10.14.135.119 10.14.135.5 LDAP SASL GSS-API Integrity: unbindRequest

61. SASL supports multiple authentication mechanism. Few are briefly depicted in the screenshot below:



62. Now let's look into CLDAP protocol. There is one more interesting observation which I've learnt which is regarding CLDAP protocol. CLDAP is UDP based protocol and is under radar for DDoS attacks since many years in the past.

63. There are around 48 packets with cldap protocol in the provided pcap file. Refer screenshot below:

No.	Time	Source	Destination	Protocol	Info
1527	0.000000	10.14.135.119	10.14.135.5	CLDAP	searchRequest(47) "<ROOT>" baseObject
1528	0.000400	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(47) "<ROOT>" searchResDone(47) success [1 result]
2877	2295.70...	10.14.135.119	10.14.135.5	CLDAP	searchRequest(48) "<ROOT>" baseObject
2878	0.000402	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(48) "<ROOT>" searchResDone(48) success [1 result]
2937	0.347078	10.14.135.119	10.14.135.5	CLDAP	searchRequest(49) "<ROOT>" baseObject
2938	0.000291	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(49) "<ROOT>" searchResDone(49) success [1 result]
3660	985.281...	10.14.135.119	10.14.135.5	CLDAP	searchRequest(50) "<ROOT>" baseObject
3661	0.000304	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(50) "<ROOT>" searchResDone(50) success [1 result]
5557	3599.94...	10.14.135.119	10.14.135.5	CLDAP	searchRequest(51) "<ROOT>" baseObject
5558	0.000350	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(51) "<ROOT>" searchResDone(51) success [1 result]
5589	0.139514	10.14.135.119	10.14.135.5	CLDAP	searchRequest(52) "<ROOT>" baseObject
5590	0.000367	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(52) "<ROOT>" searchResDone(52) success [1 result]
6127	1174.23...	10.14.135.119	10.14.135.5	CLDAP	searchRequest(53) "<ROOT>" baseObject
6129	0.000523	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(53) "<ROOT>" searchResDone(53) success [1 result]
7284	1617.00...	10.14.135.119	10.14.135.5	CLDAP	searchRequest(54) "<ROOT>" baseObject
7285	0.000410	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(54) "<ROOT>" searchResDone(54) success [1 result]
7295	0.221670	10.14.135.119	10.14.135.5	CLDAP	searchRequest(55) "<ROOT>" baseObject

Frame 1527: 312 bytes on wire (2496 bits), 312 bytes captured (2496 bits)

Ethernet II, Src: Cymphoni\_f2:d4:d8 (00:17:69:f2:d4:d8), Dst: Dell\_c2:09:6a (a4:1f:72)

Internet Protocol Version 4, Src: 10.14.135.119, Dst: 10.14.135.5

    0100 .... = Version: 4

    ....0101 = Header Length: 20 bytes (5)

    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)

        Total Length: 298

    Identification: 0x842e (33838)

    000. .... = Flags: 0x0

    ...0 0000 0000 0000 = Fragment Offset: 0

    Time to Live: 128

    Protocol: UDP (17)

    Header Checksum: 0x92fc [validation disabled]

        [Header checksum status: Unverified]

    Source Address: 10.14.135.119

    Destination Address: 10.14.135.5

Connectionless Lightweight Directory Access Protocol: Protocol

Packets: 18852 - Displayed: 48 (0.3%)

Profile: Default

64. The communication is taking place between the client 10.14.135.119 and server 10.14.135.5
65. Looking at the UDP stream as shown in the screenshot below, I think this is a DDoS attack.

Wireshark - Follow UDP Stream (udp.stream eq 29) · sinkhole.pcap	
No.	Time
2877	0.000000 10.14.135.119 → 10.14.135.5
2878	0.000402 10.14.135.5 → 10.14.135.119
	0.....0c.....
	...
	DnsDomain..rusticmoment.com.....Host..DESKTOP-U2ZYREZ.....
	DomainGuid...N1.,=I.v.x..W.....
	..NtVer..... /.../DnsHostName. DESKTOP-U2ZYREZ.rusticmoment.com0... .Netlogon0.....0d.....
	0.....0.....Netlogon1.....N1.,=I.v.x..W..rusticmoment.com...RUSTICMOMENT-DC...RUSTICMOMENT.RUSTICMOMENT-DC..Default-First-Site-Name.^.....0.....0e.....
	.....

66. Even the port number 389 gives us the idea that there may be a possibility of an attack through this vulnerable port.

No.	Time	Source	Destination	Protocol	Info	Length
2877	0.000000	10.14.135.119	10.14.135.5	CLDAP	searchRequest(48) "<ROOT>" baseObject	269
2878	0.000402	10.14.135.5	10.14.135.119	CLDAP	searchResEntry(48) "<ROOT>" searchResDone(48) success [1 result]	244

User Datagram Protocol, Src Port: 55429, Dst Port: 389

Source Port: 55429  
Destination Port: 389  
Length: 235  
Checksum: 0xe39a [unverified]  
[Checksum Status: Unverified]  
[Stream index: 29]  
> [Timestamps]  
UDP payload (227 bytes)

Connectionless Lightweight Directory Access Protocol

LDAPMessage searchRequest(48) "<ROOT>" baseObject  
messageID: 48

protocolOp: searchRequest (3)  
> searchRequest  
[Response\_In: 2878]

```

0000 a4 1f 72 c2 09 6a 00 17 69 f2 d4 d8 08 00 45
0010 00 ff 85 15 00 00 80 11 92 40 0a 0e 87 77 0a
0020 87 05 d8 85 01 85 00 eb e3 9a 30 84 00 00 00
0030 02 01 30 63 84 00 00 00 d4 04 00 0a 01 00 0a
0040 00 02 01 00 02 01 00 01 01 00 a0 84 00 00 00
0050 a3 84 00 00 00 1e 04 09 44 6e 73 44 6f 6d 61
0060 6e 04 11 72 75 73 74 69 63 6d 6f 6d 65 6e 74
0070 63 6f 6d 2e a3 84 00 00 00 00 17 04 04 48 6f 73
0080 04 0f 44 45 53 4b 54 4f 50 2d 55 32 5a 59 52
0090 5a a3 84 00 00 00 1e 04 0a 44 6f 6d 61 69 6e
00a0 75 69 64 04 10 f7 d9 4e 31 01 2c 3d 49 84 76
00b0 78 9a 07 57 e9 a3 84 00 00 00 00 00 00 04 05 4e 74
00c0 65 72 04 04 16 00 00 20 a3 84 00 00 00 2f 04
00d0 44 6e 73 48 6f 73 74 4e 61 6d 65 04 20 44 45
00e0 4b 54 4f 50 2d 55 32 5a 59 52 45 5a 2e 72 75

```

67. LDAP over TCP is safe whereas LDAP over UDP known as CLDAP is unsecure with 389 port being vulnerable.
68. Most LDAP servers and clients use the TCP protocol, which prevents amplification because of a connection handshake that verifies the source and destination can communicate with one another. UDP does not perform this verification, so the LDAP server can be convinced to send traffic to a destination that is unverified.
69. The easiest way to solve this issue is to enable a firewall on your server that blocks the LDAP port 389 from being accessed via UDP. LDAP is most used on Windows servers running Active Directory services.
70. Furthermore, there are ICMP protocols provided in the pcap file. Unlike the Transport Control Protocol (TCP) and User Datagram Protocol (UDP), the Internet Control Message Protocol (ICMP) is not designed for carrying data.
71. While ICMP packets do have a data section, their purpose is not to wrap and carry protocols like HTTP and DNS. Instead, ICMP is designed as a low-level management protocol for the internet. It carries error messages and implements simple management functions.
72. As an error messaging protocol, the structure of an ICMP packet is designed to provide the necessary information to the recipient. Error data in ICMP is carried in two values: the type and the code. Various types and codes are portrayed below:

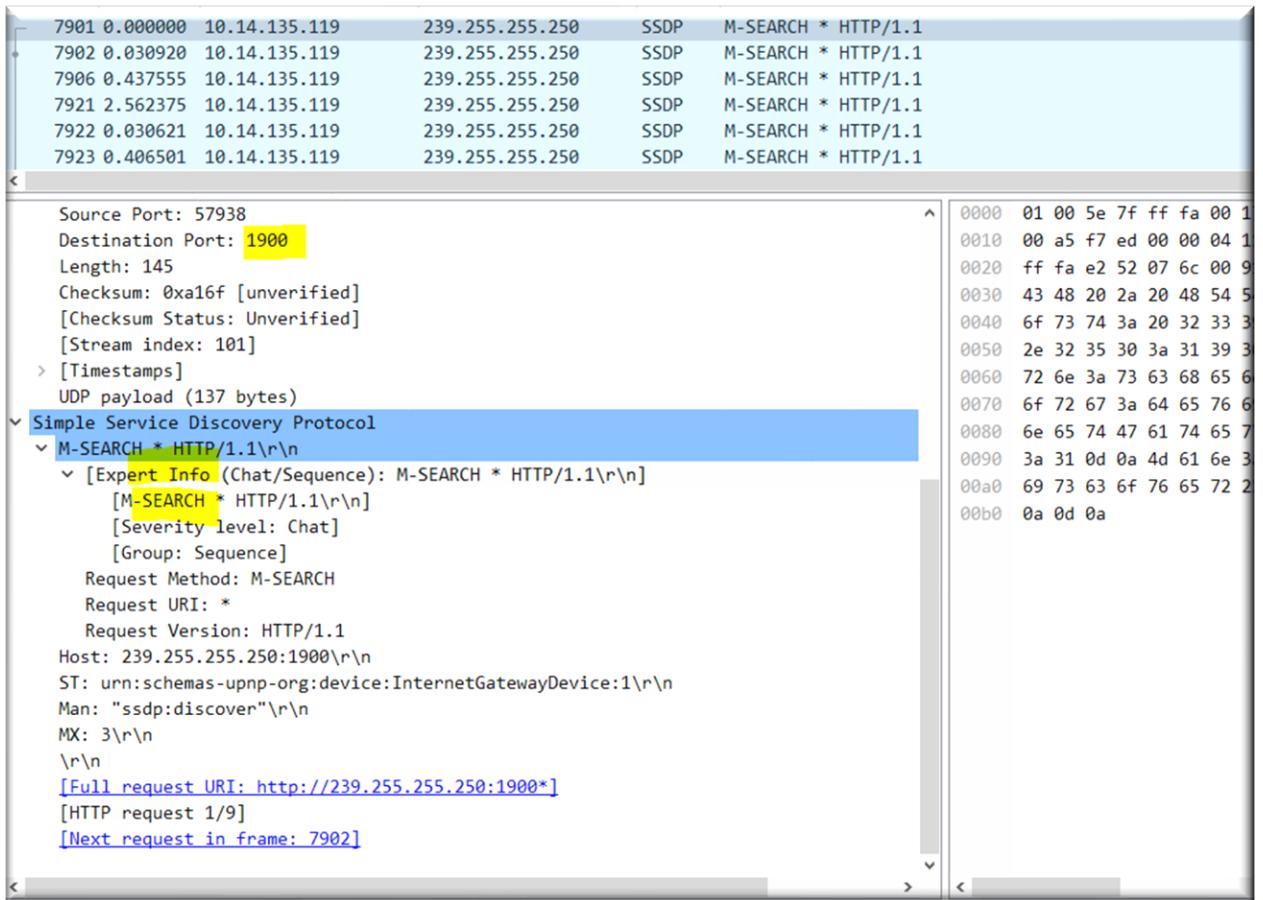
Type	Code	Description
0 – Echo Reply	0	Echo reply
3 – Destination Unreachable	0	Destination network unreachable
	1	Destination host unreachable
	2	Destination protocol unreachable
	3	Destination port unreachable
	4	Fragmentation needed and DF flag set
	5	Source route failed
5 – Redirect Message	0	Redirect datagram for the Network
	1	Redirect datagram for the host
	2	Redirect datagram for the Type of Service and Network
	3	Redirect datagram for the Service and Host
8 – Echo Request	0	Echo request
9 – Router Advertisement	0	Used to discover the addresses of operational routers
10 – Router Solicitation	0	
11 – Time Exceeded	0	Time to live exceeded in transit
	1	Fragment reassembly time exceeded
12 – Parameter Problem	0	Pointer indicates error
	1	Missing required option
	2	Bad length
13 – Timestamp	0	Used for time synchronization
14 – Timestamp Reply	0	Reply to Timestamp message

73. As shown above, the type of an ICMP packet contains the overall message that the message is intended to convey. For example, a type value of 3 means that the intended destination is unreachable.
74. For some types, there are multiple code values intended to provide additional information. For example, a type 3 ICMP message with a 0 code points to issues with the destination network, while a 1 code means that the issue is that the particular host is unreachable.
75. While many ICMP messages are designed to be sent as error messages in response to packets of other protocols, some are designed to implement standalone functionality.
76. The purpose of ping is to determine if the system at a certain IP address exists and is currently functional, and that a route to that system can be found. Typing ping into the Windows or Linux terminal will send a series of ping packets and provide a percentage value for the reachability of the destination based upon the number of ping requests that received a response.
77. The images below shows an ICMP ping request and response in Wireshark.

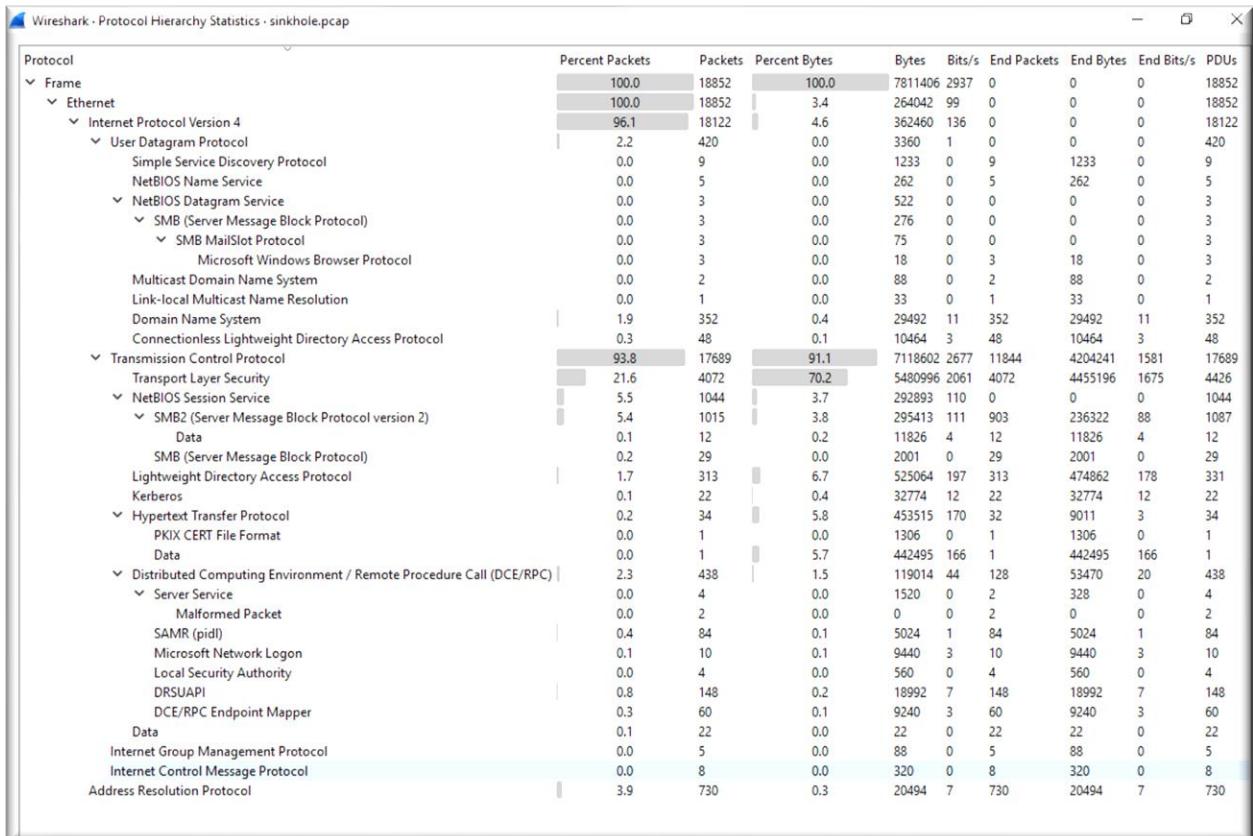
```
Destination Address: 10.14.135.5
└ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4d5a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 12700]
    > Data (32 bytes)
```

```
Destination Address: 10.14.135.119
└ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x555a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Request frame: 12699]
    [Response time: 0.239 ms]
```

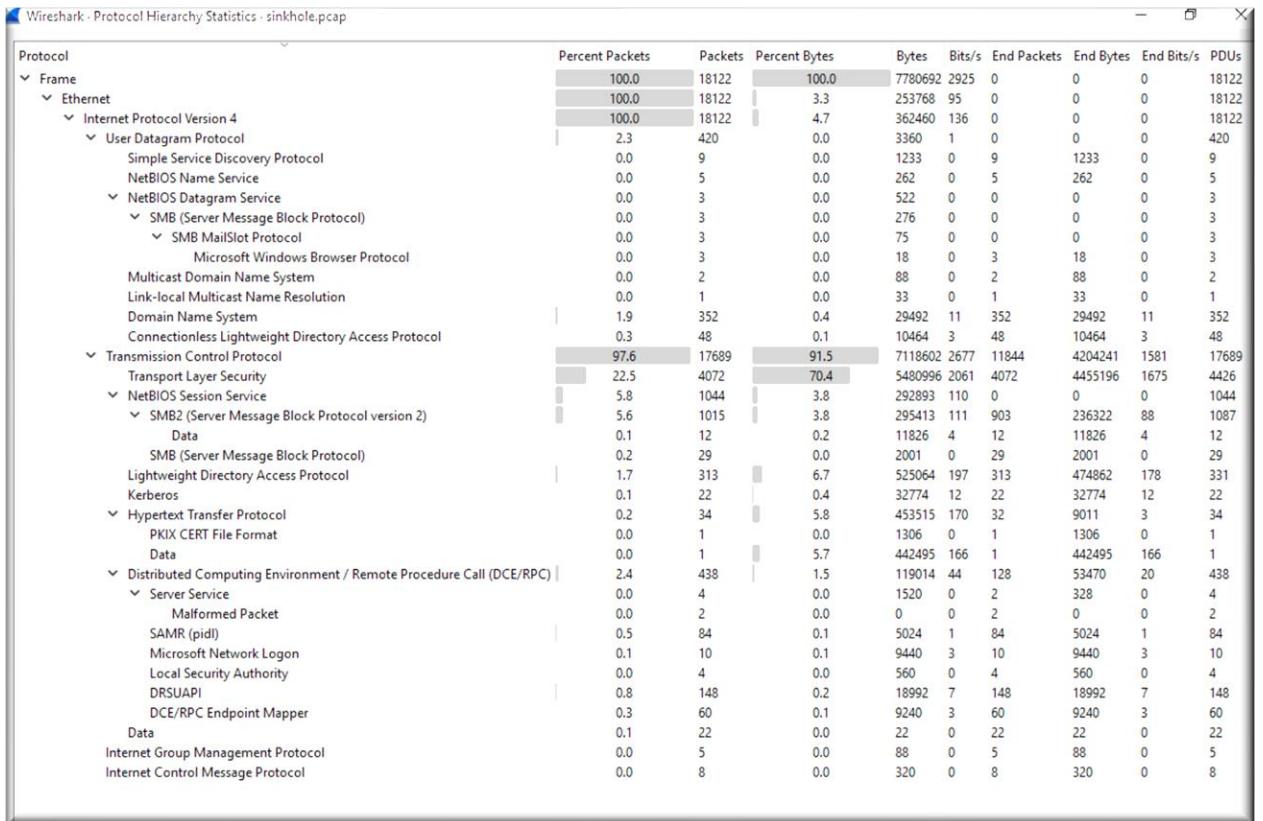
78. Other protocols which I have learnt and observed in the pcap file are:  
Internet Group Management Protocol (IGMP): IGMP is used by IP hosts to manage their dynamic multicast group membership.
79. It is also used by connected routers to discover these group members.
80. Simple Service Discovery Protocol (SSDP): The SSDP protocol can discover Plug & Play devices, with uPnP (Universal Plug and Play).
81. SSDP is HTTP like protocol and works with NOTIFY and M-SEARCH methods.
82. SSDP uses UDP transport protocol on port 1900. Refer below screenshot:



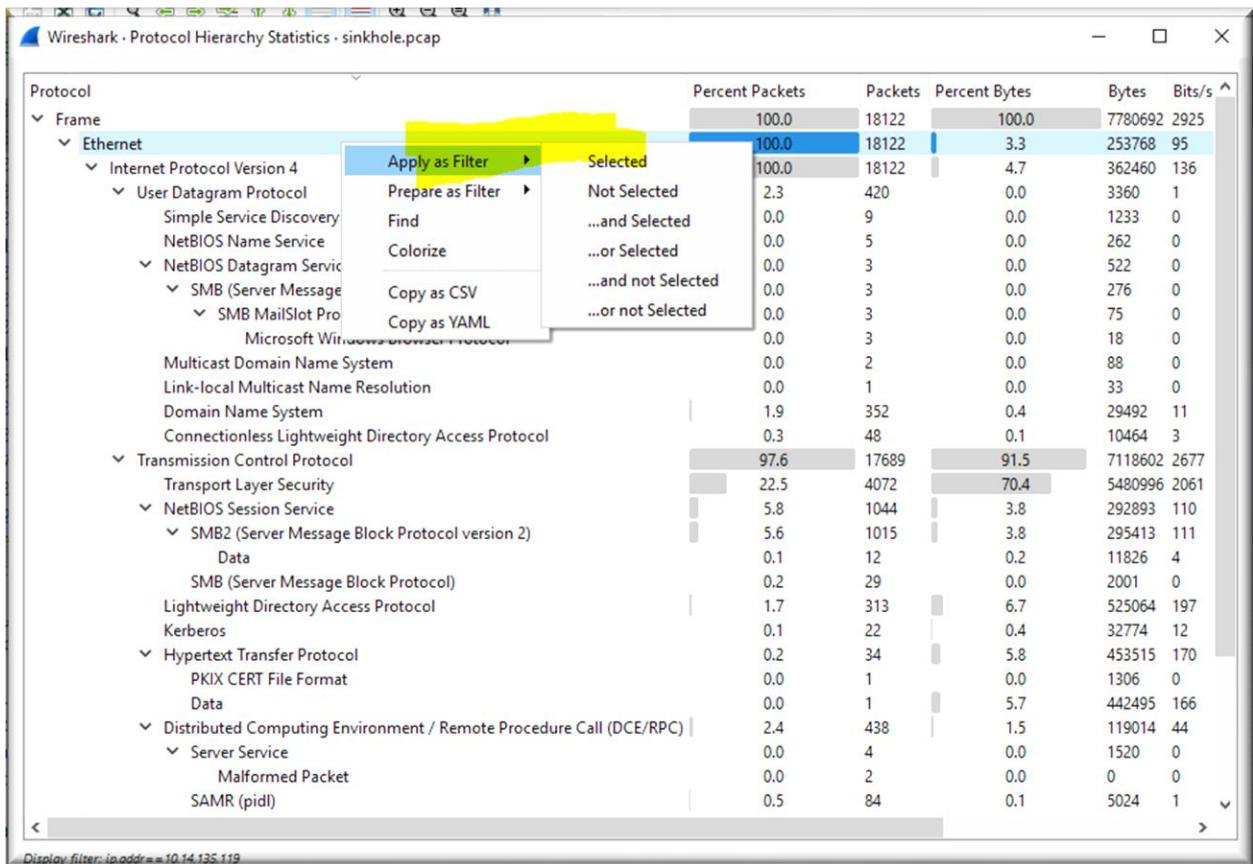
83. Now, let's observe some graphical study in wireshark "Statistics menu" which I've learnt additionally.
84. A wide range of tools related to network stats is available in the menu, which facilitate users in gaining information ranging from general info to specific protocol related info in detail.
85. general details with respect to the packets captured, filters applied, marked packets, and various other stats can be checked in the Statistics menu.
86. The Protocol Hierarchy window provides us with an overview regarding distribution of protocols used in the communication process and how to spot unusual activities in your network that do not follow the benchmark as expected. By distribution of protocols, I mean in what percentage a certain protocol has been used in the communication between two hosts, and statistics, for example, how many bytes and packets are being sent and received for every protocol, are collected easily. Any form of unusual activity can be easily figured out by matching our current traffic with the baseline created.



87. If you want to check the protocol distribution for a specific host, then before you open the Protocol Hierarchy window, apply a display filter, for example, ip.addr==10.14.135.119. The same filter will be visible at the top of the Hierarchy window just below the title bar. This makes it easy for us to figure out what kind of traffic is actually generated from a certain host, and any malicious traffic from a certain host can be easily figured out. Refer below screenshot:



88. Using the Protocol Hierarchy window, you can create filters too. Just right-click on the protocol you wish to use and then go ahead and specify the expression, as shown in the following screenshot:



89. There will be situations when a certain host in your network has been breached and you might be observing some unusual traffic associated with a particular host. In such situations, the Protocol Hierarchy window will prove worthy.
90. **Conversations** When two devices are connected to each other on the network, they are supposed to communicate; this is considered normal behavior. However, suppose you have thousands of devices connected to your network and you want to figure out the most active device that is generating too much traffic, then in that instance, the Conversations window will be quite useful.
91. To access this nice tool, click on Statistics -> Conversations. After this, you will be presented with a window like the one shown in the following screenshot, which lists various details in terms of several columns listing the packets that were transferred, the bytes that were transferred, the flow of traffic, devices' MAC addresses, and various other details. At the top, you will observe various protocols displayed individually in separate tabs, and along with each active protocol tab, you will notice a number that denotes the number of unique conversations.

Wireshark - Conversations - sinkhole.pcap

Conversation Settings

- Name resolution
- Absolute start time
- Limit to display filter

**Ethernet · 9** IPv4 · 69 IPv6 TCP · 592 UDP · 204

Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets A → B	Bytes B → A	Rel Start	Du
00:13:f3:76:a7:13	00:17:69:f2:d4:d8	24	1.520 KiB	32	75.00%	12	748 bytes	12	808 bytes	15650.999029	
00:17:69:f2:d4:d8	00:e0:18:1e:c6:6b	49	10.247 KiB	50	98.00%	26	6.072 KiB	23	4.175 KiB	15650.904204	1
00:17:69:f2:d4:d8	01:00:5e:00:00:16	5	278 bytes	5	100.00%	5	278 bytes	0	0 bytes	10820.304177	
00:17:69:f2:d4:d8	01:00:5e:00:00:fb	2	172 bytes	2	100.00%	2	172 bytes	0	0 bytes	10823.330687	
00:17:69:f2:d4:d8	01:00:5e:00:fc	1	75 bytes	1	100.00%	1	75 bytes	0	0 bytes	10823.331409	
00:17:69:f2:d4:d8	01:00:5e:ffff:fa	9	1.573 KiB	9	100.00%	9	1.573 KiB	0	0 bytes	10823.553260	1
00:17:69:f2:d4:d8	20:e5:2a:b6:93:f1	14,563	6.265 MiB	14,763	98.65%	7,031	1.425 MiB	7,532	4.840 MiB	0.144709	2127
00:17:69:f2:d4:d8	a4:1f:72:c2:09:6a	3,465	1.142 MiB	3,978	87.10%	1,721	455.071 KiB	1,744	713.925 KiB	0.000000	2123
00:17:69:f2:d4:d8	ff:ff:ff:ff:ff:ff	4	368 bytes	12	33.33%	4	368 bytes	0	0 bytes	252.760763	1539

Protocol

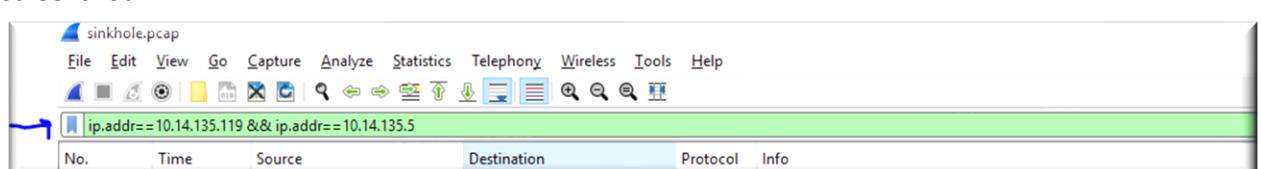
- Bluetooth
- DCCP
- Ethernet
- FC
- FDDI
- IEEE 802.11
- IEEE 802.15.4
- IPv4
- IPv6
- IPX
- JXTA
- MPTCP

Filter list for specific type

92. For example, if you are looking for the devices that generated a lot of packets and from where major data transfer has happened, then open the Conversations dialog, go to the IPv4 tab, and sort the packets column in a descending order. Here, the device listed in the first row is your answer. Take a look at the following screenshot that illustrates the same.

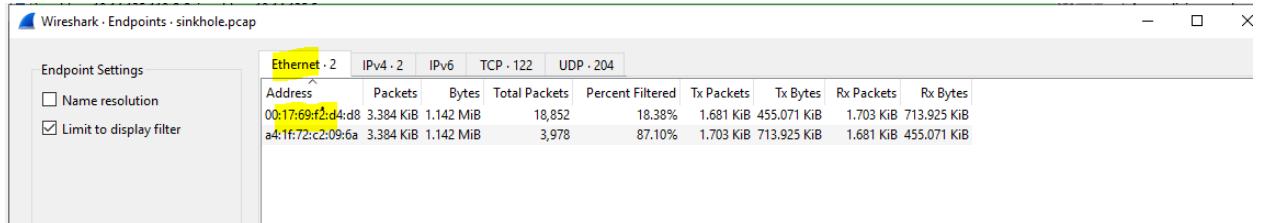
Ethernet · 9		IPv4 · 69	IPv6	TCP · 592	UDP · 204										
Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered	Packets A → B	Bytes A → B	Packets A ← B	Bytes B → A	Rel Start	Dura				
10.14.135.119	192.34.109.19	4,222	1.545 MiB	4,222	100.00%	2,111	470.664 KiB	2,111	1.085 MiB	14845.040587	1274.				
10.14.135.119	10.14.135.5	3,465	1.142 MiB	3,465	100.00%	1,721	455.071 KiB	1,744	713.925 KiB	0.000000	21109.				
10.14.135.119	87.120.37.231	2,906	1.209 MiB	2,906	100.00%	1,462	410.314 KiB	1,444	827.674 KiB	92.886362	20997.				
10.14.135.119	204.79.197.200	2,033	1.601 MiB	2,033	100.00%	837	62.251 KiB	1,196	1.540 MiB	2819.264992	16291.				
10.14.135.119	91.92.109.10	916	241.200 KiB	916	100.00%	498	168.682 KiB	418	72.519 KiB	1893.651863	5289.				
10.14.135.119	45.95.11.201	513	460.111 KiB	513	100.00%	191	10.601 KiB	322	449.511 KiB	0.144709	2.				
10.14.135.119	52.183.220.149	431	146.627 KiB	431	100.00%	183	30.945 KiB	248	115.682 KiB	137.893584	18105.				
10.14.135.119	31.13.195.145	335	90.708 KiB	335	100.00%	183	65.935 KiB	152	24.773 KiB	142.917532	1869.				
10.14.135.119	74.6.143.26	318	93.612 KiB	318	100.00%	166	18.538 KiB	152	75.074 KiB	153.541621	19749.				
10.14.135.119	13.107.21.200	267	104.610 KiB	267	100.00%	128	23.449 KiB	139	81.161 KiB	925.965680	18182.				
10.14.135.119	98.137.11.164	266	69.619 KiB	266	100.00%	142	16.188 KiB	124	53.432 KiB	153.107605	19790.				
10.14.135.119	162.219.225.118	164	43.133 KiB	164	100.00%	83	9.982 KiB	81	33.150 KiB	3127.013078	16486.				
10.14.135.119	176.32.103.205	149	50.863 KiB	149	100.00%	70	7.776 KiB	79	43.087 KiB	539.488635	19073.				
10.14.135.119	52.185.211.133	105	26.447 KiB	105	100.00%	46	7.657 KiB	59	18.790 KiB	3832.413348	12510.				
10.14.135.119	74.6.231.21	103	32.284 KiB	103	100.00%	55	6.223 KiB	48	26.062 KiB	4168.507654	11264.				
10.14.135.119	98.137.11.163	97	25.505 KiB	97	100.00%	51	5.980 KiB	46	19.524 KiB	4167.911817	11265.				
10.14.135.119	74.6.143.25	95	25.154 KiB	95	100.00%	51	5.980 KiB	44	19.174 KiB	2250.537468	9552.				
10.14.135.119	54.239.28.85	91	29.738 KiB	91	100.00%	45	4.890 KiB	46	24.849 KiB	8824.926990	4628.				
10.14.135.119	205.251.242.103	90	29.816 KiB	90	100.00%	44	4.902 KiB	46	24.914 KiB	15748.425909	5524.				
10.14.135.119	40.83.247.108	79	18.963 KiB	79	100.00%	37	6.832 KiB	42	12.131 KiB	10828.505433	10092.				
10.14.135.119	13.249.31.218	74	27.674 KiB	74	100.00%	37	3.949 KiB	37	23.725 KiB	20390.771435	882.				
10.14.135.119	52.184.215.140	70	23.256 KiB	70	100.00%	36	11.397 KiB	34	11.858 KiB	14202.511948	34.				
10.14.135.119	13.107.5.88	68	29.390 KiB	68	100.00%	30	3.654 KiB	38	25.735 KiB	1826.045253	14525.				
10.14.135.119	104.94.77.31	67	7.691 KiB	67	100.00%	36	3.189 KiB	31	4.502 KiB	198.183192	19717.				
10.14.135.119	40.126.28.11	62	32.143 KiB	62	100.00%	28	7.865 KiB	34	24.277 KiB	1477.602776	12724.				
10.14.135.119	40.126.29.5	60	31.593 KiB	60	100.00%	27	7.681 KiB	33	23.912 KiB	10828.932818	3372.				
10.14.135.119	99.86.100.195	52	18.623 KiB	52	100.00%	26	2.736 KiB	26	15.887 KiB	12231.244231	1221.				
10.14.135.119	10.14.135.91	49	10.247 KiB	49	100.00%	26	6.072 KiB	23	4.175 KiB	15650.904204	10.				
10.14.135.119	13.226.182.163	48	18.149 KiB	48	100.00%	22	2.263 KiB	26	15.887 KiB	539.979342	812.				
10.14.135.119	52.168.112.66	48	17.368 KiB	48	100.00%	21	6.442 KiB	27	10.926 KiB	1831.172482	6413.				
10.14.135.119	142.251.33.46	45	14.411 KiB	45	100.00%	22	2.440 KiB	23	11.971 KiB	3564.860624	428.				
10.14.135.119	40.126.29.9	42	25.441 KiB	42	100.00%	17	6.331 KiB	25	19.110 KiB	11726.072224	0.				
10.14.135.119	52.109.12.18	41	19.880 KiB	41	100.00%	17	1.814 KiB	24	18.065 KiB	1117.192429	119.				
10.14.135.119	104.214.104.116	38	7.913 KiB	38	100.00%	20	2.612 KiB	18	5.301 KiB	14235.819516	0.				
10.14.135.119	216.58.193.142	37	7.670 KiB	37	100.00%	19	2.503 KiB	18	5.167 KiB	14342.899167	870.				
10.14.135.119	142.250.68.174	36	7.871 KiB	36	100.00%	18	2.577 KiB	18	5.294 KiB	5388.927594	2124.				
10.14.135.119	142.251.33.3	32	9.820 KiB	32	100.00%	14	2.720 KiB	18	7.101 KiB	19658.465354	0.				
10.14.135.119	142.250.114.94	21	9.770 KiB	21	100.00%	13	2.667 KiB	18	7.103 KiB	1671.259526	0.				

93. In the first row, we can see how many packets/bytes have been sent and received by each endpoint and the total elapsed duration. If you wish to create a filter for the same, right click on the first row and then create the respective expression you are thinking about. I chose the first option, A<->B, which only shows packets that are associated with Address A and Address B:
94. The respective filter will be inserted in the Display Filter dialog, as shown in the following screenshot:

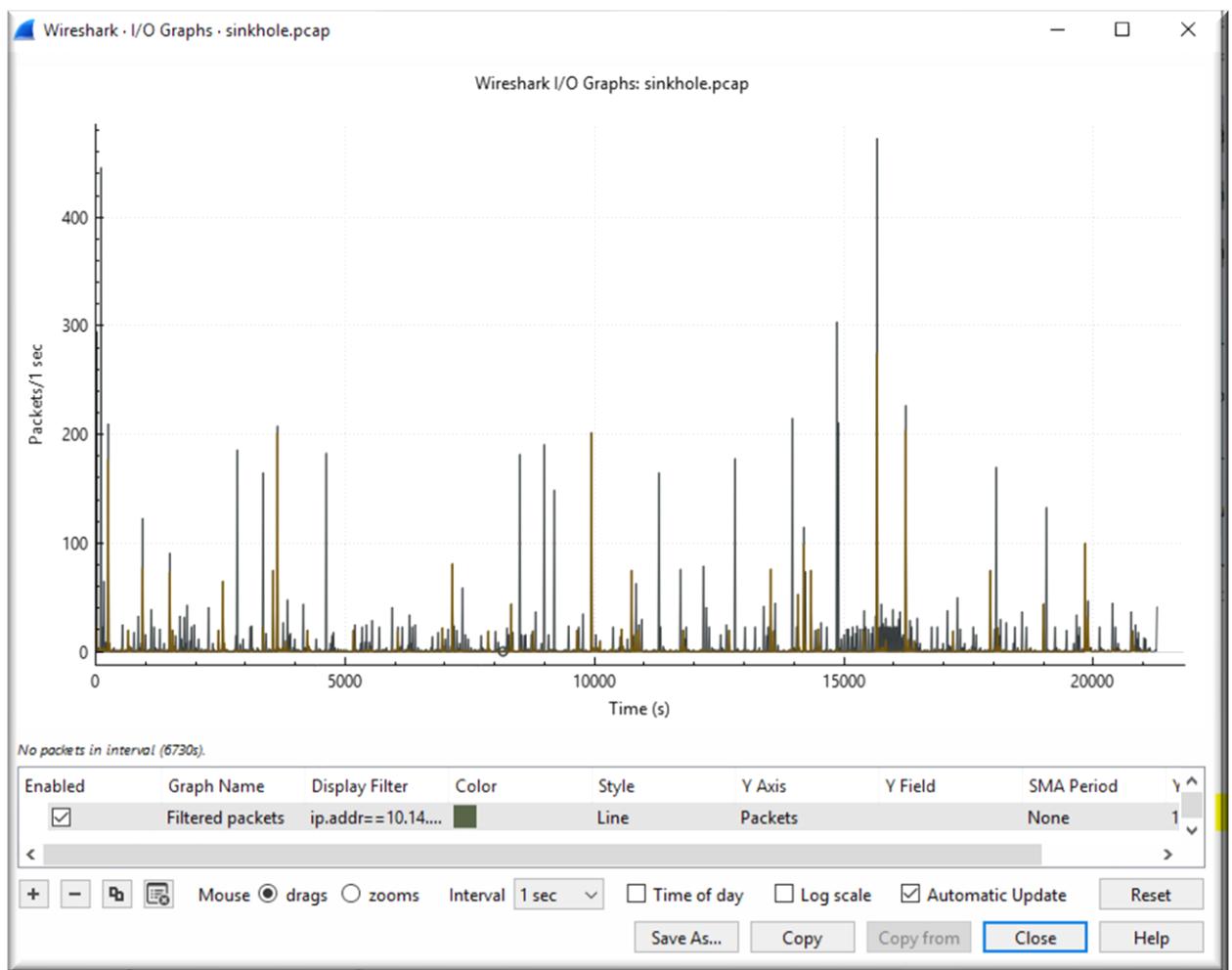


95. **Endpoints**, Two devices that share data with each other are often referred to as endpoints with reference to Wireshark. Every host is able to communicate with the help of an Network Interface Card (NIC) that holds a physical address (often termed as a MAC address), and the same address is used for communication over a local network. Devices that communicate in this

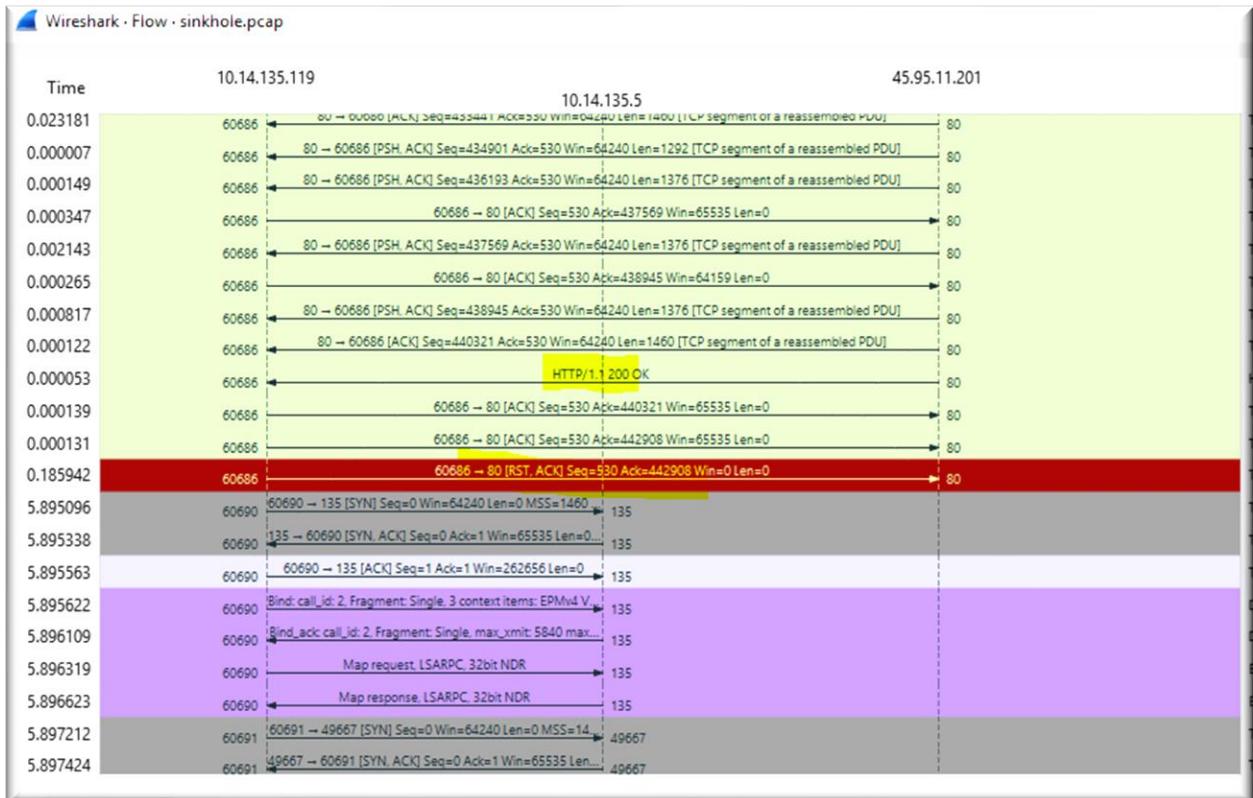
kind of infrastructure are termed as endpoints. Wireshark gives us the facility of analyzing and collecting information regarding these two devices



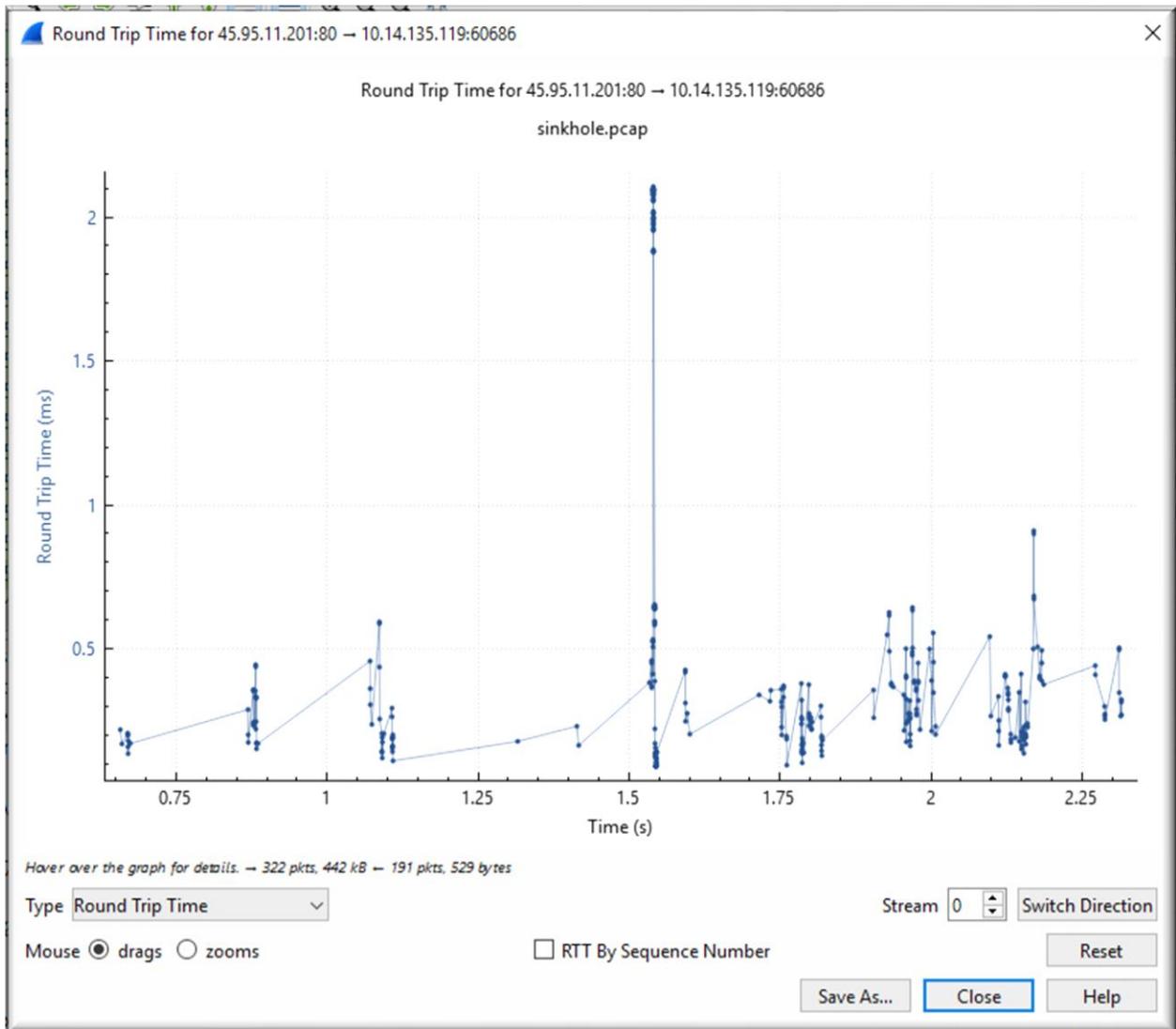
96. **IO Graphs**, is one of the basic graphs that are created using the packets available in the capture file. To create the IO graph, select any TCP packet in your capture file and then click on IO Graph under Statistics. Refer to the following screenshot:



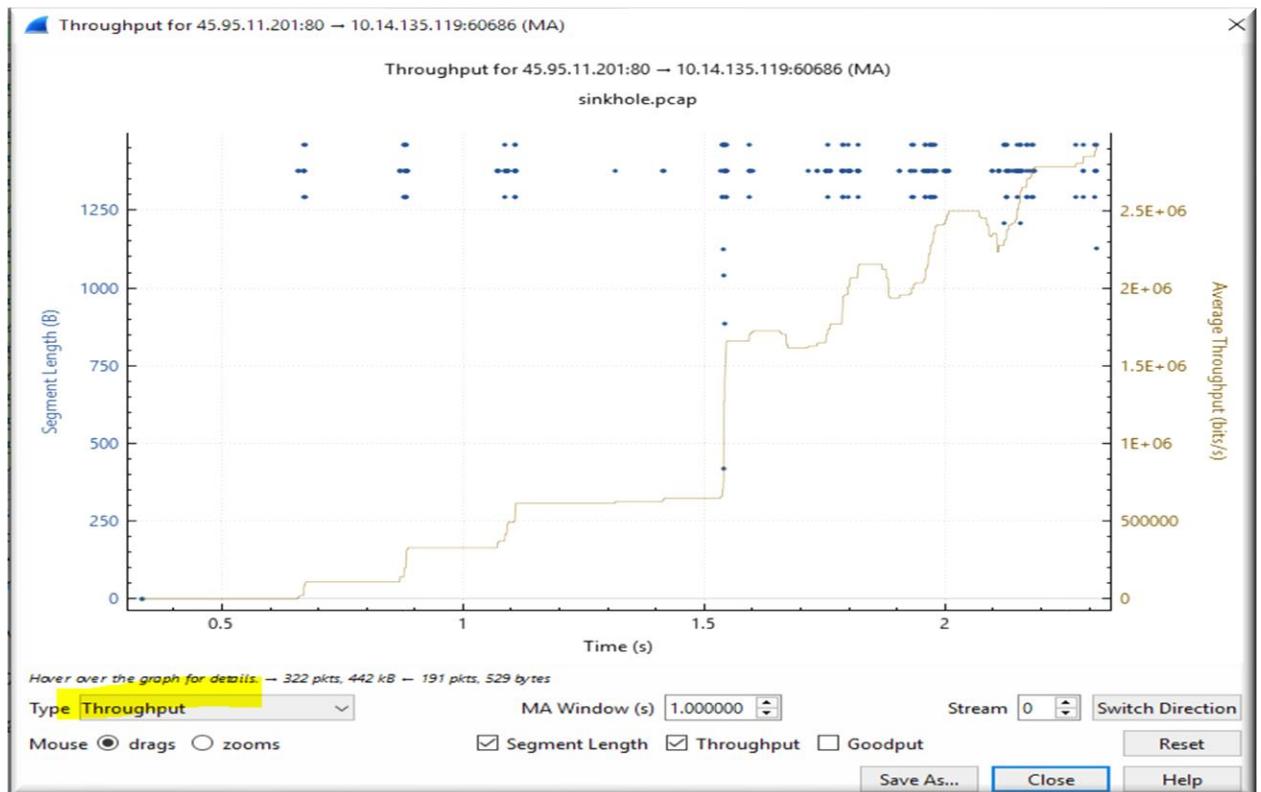
97. **Flow graph**, is one of the nicest features in Wireshark, where we are assisted with troubleshooting capabilities in scenarios like facing a lot of dropped connections, lost frames, retransmission traffic, and more. Flow graphs let us create a column-based graph, which summarizes the flow of traffic between two endpoints, and it even lets us export the results in a simple text-based format. This is the easiest way of verifying the connection between client and server. Refer below screenshot:



98. As we can see the communication above between client(10.14.135.119) and server(10.14.135.5), post receiving HTTP/1.1 200 OK from the server, wherein malicious software was transferred in the form of an attachment, we observe the packet drop immediately as highlighted in red as the client sends reset to the server (RST, ACK) due to loss in connection.
99. **Round-trip time (RTT)** is the duration in which the ACK for a packet that is sent is received, that is, for every packet sent from a host, there is an ACK received (TCP communication), which determines the successful delivery of the packet. The total time that is consumed from the transfer of the packet to the ACK for the same is called round trip time. Follow these steps to create one for yourself: Select any TCP packet in your packet list pane. Navigate to Statistics | TCP Stream Graph | Round Trip Time Graph. The x axis represents the TCP sequence number, and the y axis represents the RTT in milliseconds. Each plotted point on the graph represents the RTT of a packet. If you are not seeing anything in your graph, then you might have selected an opposite directional packet. RTT graphs are often used by network admins to identify any congestion or latency that can make your network perform slowly.



100. **Throughput graph**, is very similar to the IO graph that depicts the traffic flow. However, it is different in one important aspect that Throughput graphs depict the unidirectional traffic whereas IO graphs depict the traffic in both directions.



101. **Expert Information** The information in the Expert Infos dialog is populated by the dissectors that enable the translation of every protocol that is well known to Wireshark. Refer screenshot below:

Packet	Summary	Group	Protocol
>Error	Malformed Packet (Exception occurred)	Malformed	SRVSVC
12736	NetShareEnumAll response[Malformed Packet]	Malformed	SRVSVC
12842	NetShareEnumAll response[Malformed Packet]	Malformed	SRVSVC
> Warning	Long frame	Protocol	SRVSVC
> Warning	Long frame	Protocol	SAMR
> Warning	D-SACK Sequence	Sequence	TCP
> Warning	Connection reset (RST)	Sequence	TCP
> Note	A new tcp session is started with the same ports as an earlier session in th...	Sequence	TCP
> Note	Time To Live	Sequence	IPv4
> Note	ACK to a TCP keep-alive segment	Sequence	TCP
> Note	TCP keep-alive segment	Sequence	TCP
> Note	The acknowledgment number field is nonzero while the ACK flag is not set	Protocol	TCP
> Note	This session reuses previously negotiated keys (Session resumption)	Sequence	TLS
> Note	This frame undergoes the connection closing	Sequence	TCP
> Note	This frame initiates the connection closing	Sequence	TCP
> Note	This frame is a (suspected) retransmission	Sequence	TCP
> Chat	Formatted text	Sequence	SSDP
> Chat	Connection finish (FIN)	Sequence	TCP
> Chat	Formatted text	Sequence	HTTP
> Chat	Connection establish acknowledge (SYN+ACK)	Sequence	TCP
> Chat	Connection establish request (SYN)	Sequence	TCP

