

Lab 03 – Network Sniffing

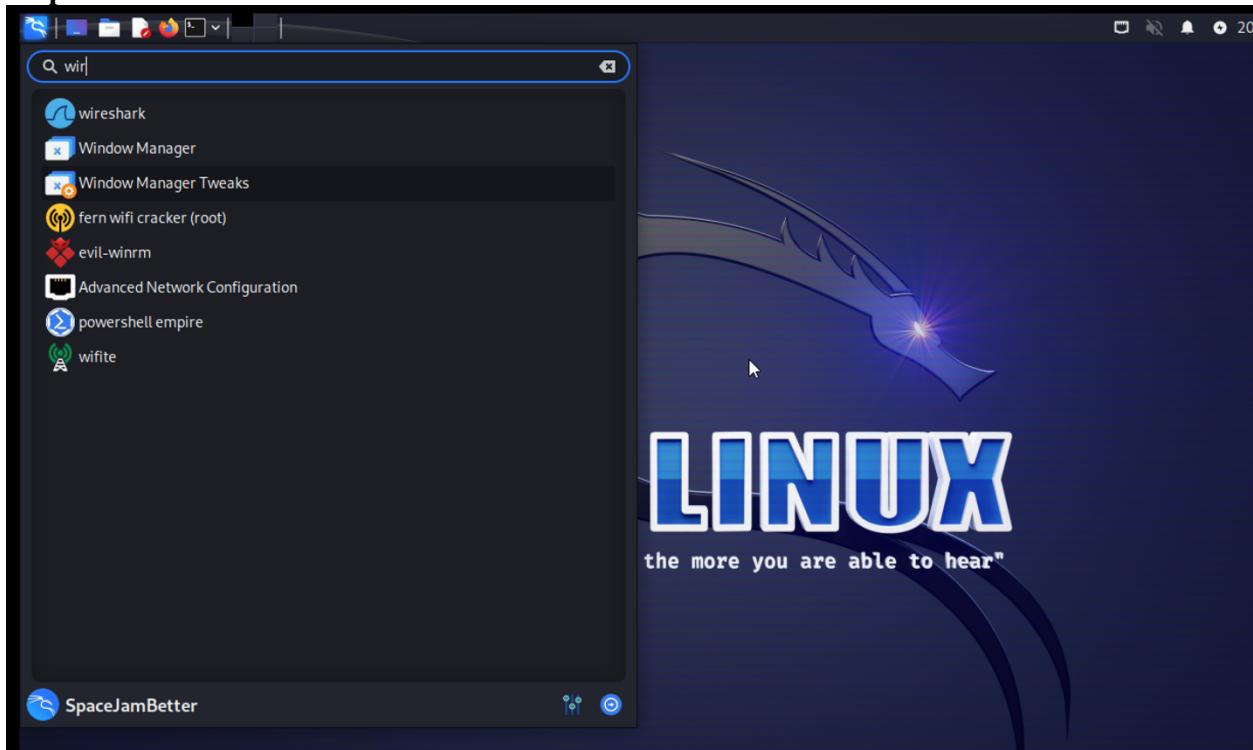
David Murillo Santiago
Professor Pugh
IS-3513
26 October 2023

INTRODUCTION

In this lab, I will learn the fundamentals of network packet analysis through Wireshark. I will acquire essential skills in capturing, filtering, and dissecting network packets, which are critical for understanding network traffic and ensuring security.

PROCESS

Step 1: Download Wireshark



To begin, I needed to ensure that I had Wireshark. Because I am using a Kali Linux, Wireshark already comes downloaded with my machine.

Step 2: Packet Capture

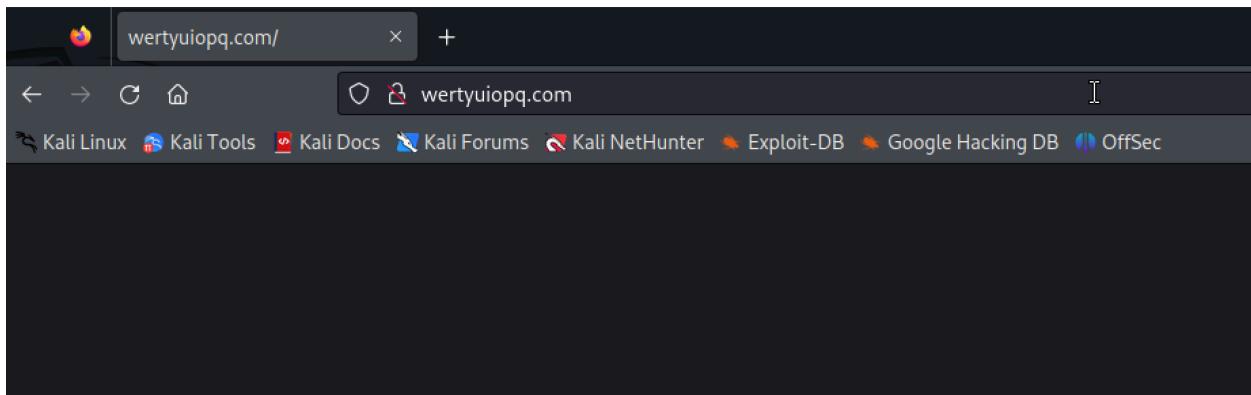
In this step, I began capturing packets on Wireshark and I searched for different websites.

The screenshot shows the Wireshark interface with a list of captured network packets. The table has columns: No., Time, Source, Destination, Protocol, Length, and Info. The 'Info' column displays detailed hex and ASCII data for each packet. The first few rows show TCP connections between various IP addresses.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.022664491	192.229.211.108	192.168.64.2	TCP	66	[TCP ACKed ur
6	0.024528546	23.220.161.15	192.168.64.2	TCP	66	[TCP ACKed ur
7	0.256742918	192.168.64.2	23.220.161.15	TCP	66	56644 → 80 [A
8	0.278901281	23.220.161.15	192.168.64.2	TCP	66	[TCP ACKed ur
9	1.280052097	192.168.64.2	3.161.247.212	TCP	66	37058 → 80 [A
10	1.305401526	3.161.247.212	192.168.64.2	TCP	66	[TCP ACKed ur
11	1.536775907	192.168.64.2	142.250.114.94	TCP	66	39564 → 80 [A
12	1.579879378	142.250.114.94	192.168.64.2	TCP	66	[TCP ACKed ur
13	3.076191077	192.168.64.2	192.229.211.108	TCP	66	53870 → 80 [A
14	3.102339348	192.229.211.108	192.168.64.2	TCP	66	[TCP ACKed ur

Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0 at 00:0c:29:14:7d:08 (Intel PRO/100 MT Desktop), transmitted on interface eth0 at 00:0c:29:14:7d:08 (Intel PRO/100 MT Desktop) at 19:45:18.000000000 UTC
Ethernet II, Src: Kali Linux (00:0c:29:14:7d:08), Dst: (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 192.168.64.2 (192.168.64.2), Dst: 192.229.211.108 (192.229.211.108)
Transmission Control Protocol, Src Port: 5760 (5760), Dst Port: 80 (80)

Screenshot revealing that I began the packet capture.

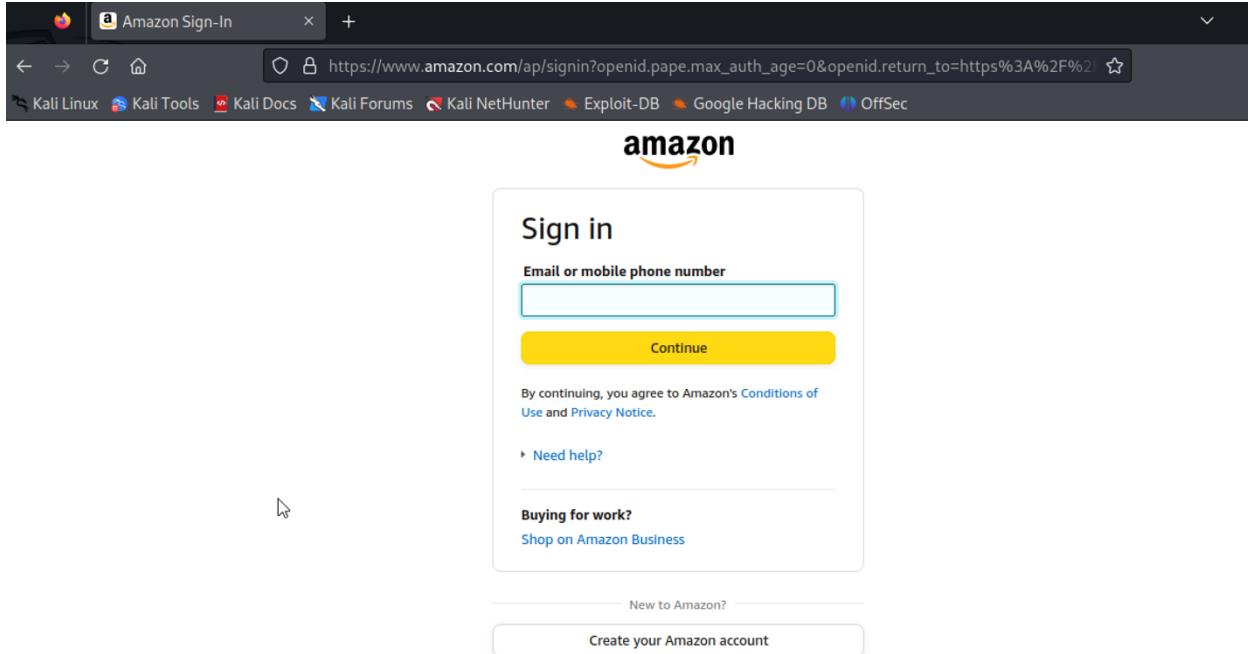


Next, I searched up a random website that does not exist. I searched up <http://wertyuiopq.com>. The search did not take me anywhere as the link was made up, therefore I was left on a blank, dark screen.

```
(spacejambetter㉿kali)-[~]
$ ping www.google.com
PING www.google.com (142.250.138.99) 56(84) bytes of data.
64 bytes from rw-in-f99.1e100.net (142.250.138.99): icmp_seq=1 ttl=103 time=19.8 ms
64 bytes from rw-in-f99.1e100.net (142.250.138.99): icmp_seq=2 ttl=103 time=19.9 ms
64 bytes from rw-in-f99.1e100.net (142.250.138.99): icmp_seq=3 ttl=103 time=27.4 ms
^C
--- www.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 19.822/22.366/27.360/3.531 ms

(spacejambetter㉿kali)-[~]
$
```

Next, I used the terminal to ping the google server. By pinging google, I found that their servers are responsive. Because I am using a Linux terminal, I needed to manually stop pinging google, so I entered “control + c” which ended the ping.



Next, I went on the Amazon website and clicked on “sign in.” After this, I ended the packet capture.

Step 2: DNS packet analysis

In this step I applied a filter on Wireshark that only displays the captured DNS packets. I was specifically looking for the DNS response to the fake website I attempted to search.

No.	Time	Source	Destination	Protocol	Length	Info
415	22.235876907	192.168.64.2	192.168.64.1	DNS	74	Standard query 0x2be7 A www.google.com
416	22.235891782	192.168.64.2	192.168.64.1	DNS	74	Standard query 0xdfef5 AAAA www.google.com
417	22.237189470	192.168.64.1	192.168.64.2	DNS	170	Standard query response 0x2be7 A www.google.com A 1
418	22.237189512	192.168.64.1	192.168.64.2	DNS	102	Standard query response 0xdfef5 AAAA www.google.com
421	22.257334096	192.168.64.2	192.168.64.1	DNS	87	Standard query 0x986d PTR 99.138.250.142.in-addr.arpa
422	22.258205061	192.168.64.1	192.168.64.2	DNS	120	Standard query 0x986d PTR 99.138.250.142.in-addr.arpa
451	23.267851599	192.168.64.2	192.168.64.1	DNS	87	Standard query 0xb61d PTR 99.138.250.142.in-addr.arpa
452	23.269652545	192.168.64.1	192.168.64.2	DNS	120	Standard query 0xb61d PTR 99.138.250.142.in-addr.arpa
530	33.081570541	192.168.64.2	192.168.64.1	DNS	78	Standard query 0xd94c A huntress.ctf.games
531	33.082481223	192.168.64.1	192.168.64.2	DNS	94	Standard query 0xd94c A huntress.ctf.games
598	35.173595498	192.168.64.2	192.168.64.1	DNS	74	Standard query 0x3a77 A wertyuiopq.com
599	35.212278517	192.168.64.1	192.168.64.2	DNS	90	Standard query 0x3a77 A wertyuiopq.com A 1
726	46.478861506	192.168.64.2	192.168.64.1	DNS	74	Standard query 0x8816 A www.amazon.com
730	46.505155994	192.168.64.1	192.168.64.2	DNS	169	Standard query response 0x8816 A www.amazon.com CNAME
753	47.326578056	192.168.64.2	192.168.64.1	DNS	81	Standard query 0xc8d1 A completion.amazon.com
769	47.339624505	192.168.64.1	192.168.64.2	DNS	97	Standard query 0xc8d1 A completion.amazon.com
1192	48.598887397	192.168.64.2	192.168.64.1	DNS	131	Standard query 0x9e69 A p3epspumangb5dvijf52er7wdi..
1195	48.599010565	192.168.64.2	192.168.64.1	DNS	131	Standard query 0x2076 AAAA p3epspumangb5dvijf52er7wdi..
1196	48.600018833	192.168.64.1	192.168.64.2	DNS	207	Standard query 0x2076 AAAA p3epspumangb5dvijf52er7wdi..
1197	48.602662294	192.168.64.2	192.168.64.1	DNS	131	Standard query 0xe501 A p3epspumangb5dvijf52er7wdi..
1211	48.622568175	192.168.64.1	192.168.64.2	DNS	179	Standard query 0x9e69 A p3epspumangb5dvijf52er7wdi..

I entered DNS next to the Display Filter to only view DNS packets. This screenshot reveals all the captured DNS packets which includes the google, amazon, and fake website searches I made during the traffic capture.

530 33.081570541	192.168.64.2	192.168.64.1	DNS	78 Standard query 0xd94c A huntress.ctf.games
531 33.082481223	192.168.64.1	192.168.64.2	DNS	94 Standard query response 0xd94c A huntress.ctf.games
598 35.173595498	192.168.64.2	192.168.64.1	DNS	74 Standard query 0x3a77 A wertyuiopq.com
599 35.212278517	192.168.64.1	192.168.64.2	DNS	90 Standard query 0x3a77 A wertyuiopq.com A 1

Screenshot of the DNS query and response from my fake website search.

```
▶ Frame 599: 90 bytes on wire (720 bits), 90 bytes captured (720 bits)
▶ Ethernet II, Src: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64), Dst: 42:Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.2
▶ User Datagram Protocol, Src Port: 53, Dst Port: 60371
└ Domain Name System (response)
    Transaction ID: 0x3a77
    Flags: 0x8100 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
    ┌ Queries
    ┌ Answers
    ┌ [Request In: 598]
    ┌ [Time: 0.038683019 seconds]
```

Next, I investigated the DNS response and found that there were no errors. This is odd because, if the website I searched is fake, the DNS response should include a “NXDOMAIN” or a “No such name.”

```
└ Additional RRs: 0
  ┌ Queries
  ┌ Answers
  ┌ [Request In: 598]
  ┌ [Time: 0.038683019 seconds]
```

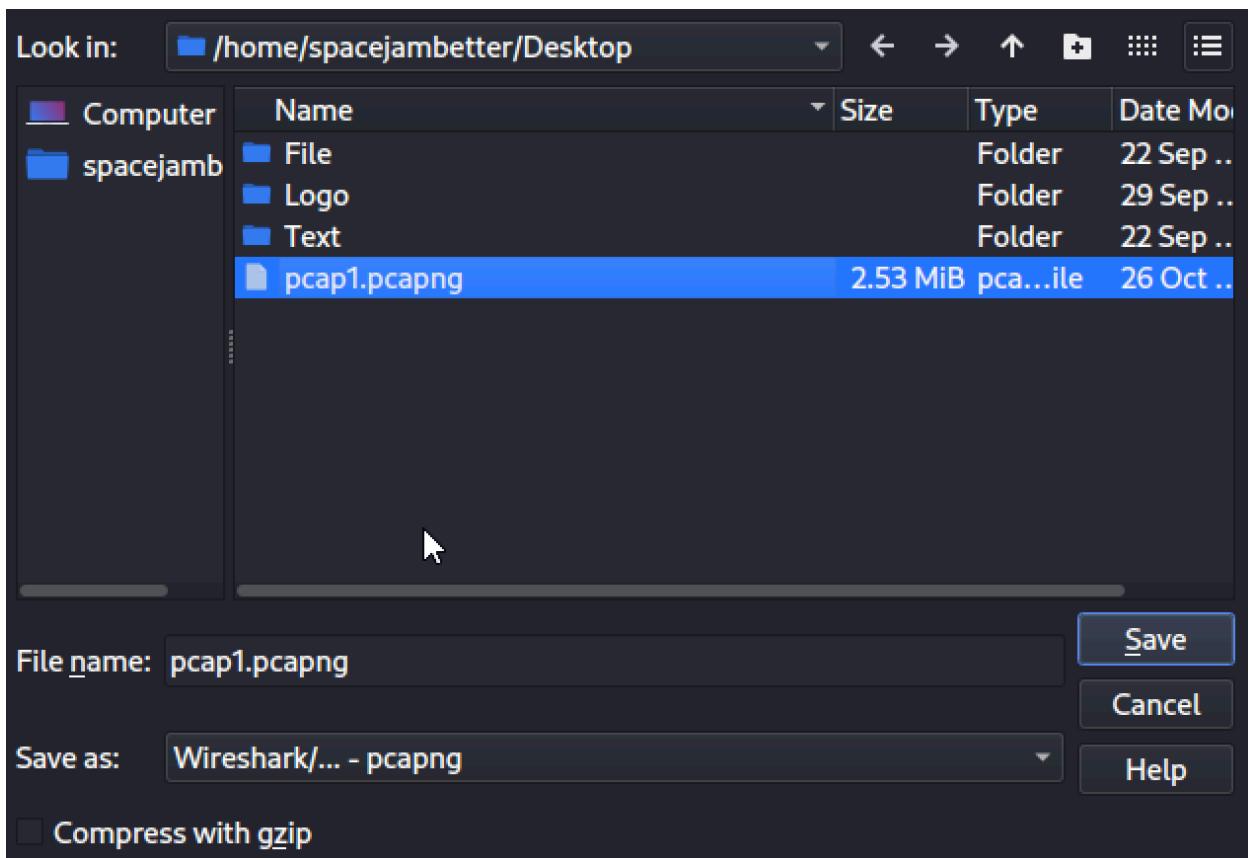
wertyuiopq.com: type A, class IN, addr 143.244.220.150

Name: wertyuiopq.com
Type: A (Host Address) (1)
Class: IN (0x0001)
Time to live: 77 (1 minute, 17 seconds)
Data length: 4
Address: 143.244.220.150

Next, I investigated further and found that the “fake” website I had searched for was in fact real. The highlighted portion of the screenshot reveal the link, type, class, and address of the site.

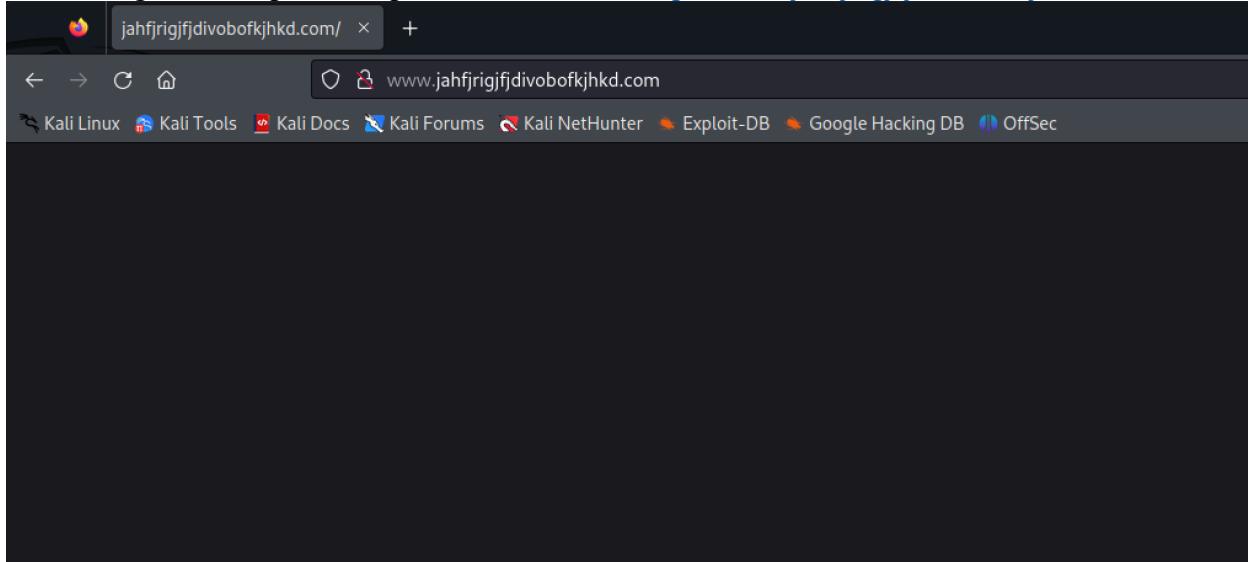
Once I realized the “fake” website was real, I decided to use the fake link provided in the instructions to guarantee that the website is fake.

First, I saved the packet capture to ensure the integrity of the remaining data which I must analyze.



Screenshot revealing that I saved the packet capture as "pcap1" on my Desktop.

Next, I began a new packet capture and searched <http://www.jahfjrigjfjdivobofkjhkd.com/>.



Screenshot of me searching for the new, provided link.

No.	Time	Source	Destination	Protocol	Length	Info
10	5.552936960	192.168.64.2	192.168.64.1	DNS	91	Standard query 0x21d8 A www.jahfjrigjfjdivobofkjhkd
11	5.586204080	192.168.64.1	192.168.64.2	DNS	107	Standard query response 0x21d8 A www.jahfjrigjfjdiv
56	25.866130312	192.168.64.2	192.168.64.1	DNS	74	Standard query 0xdeaa A www.google.com
57	25.887656681	192.168.64.1	192.168.64.2	DNS	170	Standard query response 0xdeaa A www.google.com A 1
76	26.508864043	192.168.64.2	192.168.64.1	DNS	74	Standard query 0xab7e A www.google.com
77	26.508865459	192.168.64.2	192.168.64.1	DNS	74	Standard query 0x5c7c AAAA www.google.com
78	26.509794195	192.168.64.1	192.168.64.2	DNS	170	Standard query response 0xab7e A www.google.com A 1
79	26.522097960	192.168.64.1	192.168.64.2	DNS	102	Standard query response 0x5c7c AAAA www.google.com

Frame 11: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface eth0, id 0
 Ethernet II, Src: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64), Dst: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee)
 Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.2
 User Datagram Protocol, Src Port: 53, Dst Port: 45962
 Domain Name System (response)

0000	42	e0	d8
0010	00	5d	84
0020	40	02	00
0030	00	01	00
0040	72	69	67
0050	6b	64	03
0060	01	00	00

Screenshot revealing the new DNS packets that were captured.

▼ Domain Name System (response)
Transaction ID: 0x21d8
▶ Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
▼ Queries
▼ www.jahfjrigjfjdivobofkjhkd.com: type A, class IN
Name: www.jahfjrigjfjdivobofkjhkd.com
[Name Length: 31]
[Label Count: 3]
Type: A (Host Address) (1)
Class: IN (0x0001)

Once again, the DNS response was "No error."

To understand the meaning of "No error" I used the following website by BlueCat Networks: "The top four DNS response codes and what they mean" <https://bluecatnetworks.com/blog/the-top-four-dns-response-codes-and-what-they-mean/>

According to the site, "NOERROR" means that the DNS query was successfully completed, meaning that the DNS query for a domain name resolved to an IP address without any errors. One reason that the DNS response is "No error" instead of "no such name" is because my machine is configured to handle DNS queries for non-existent domains differently, possibly redirecting or modifying the DNS responses to provide alternative results or search pages rather than returning a standard "No such name" response. To test this I entered very fictitious and long URLs in my browser and observed the subsequent DNS responses.

→ 4562 467.833755786 192.168.64.2	192.168.64.1	DNS	89	Standard query 0x22e5 A www.fnerofoernfoenofnweofwe
4563 467.833797452 192.168.64.2	192.168.64.1	DNS	89	Standard query 0x34e7 AAAA www.fnerofoernfoenofnweofwe
4564 467.861290065 192.168.64.1	192.168.64.2	DNS	89	Standard query response 0x34e7 AAAA www.fnerofoernfoenofnweofwe
4565 467.861478607 192.168.64.1	192.168.64.2	DNS	105	Standard query response 0x22e5 A www.fnerofoernfoenofnweofwe

Frame 4565: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface eth0, id 0
 Ethernet II, Src: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64), Dst: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee)
 Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.2
 User Datagram Protocol, Src Port: 53, Dst Port: 43533
 Domain Name System (response)
 Transaction ID: 0x22e5
 ▶ Flags: 0x8180 Standard query response, No error

0000	42	e0	d8
0010	00	5b	25
0020	40	02	00
0030	00	01	00
0040	66	67	66
0050	6f	66	77
0060	00	00	40

I searched up www.fnerofoernfoenofnweofwe.net, a site which I made up by mashing keys, and the DNS response was still "No error."

No.	Name	Source	Destination	Protocol	Length Info
4250	366.942046449	192.168.64.2	192.168.64.1	DNS	100 Standard query 0xce0a A www.thisnameisimpossiblelong9887
4251	366.942062949	192.168.64.2	192.168.64.1	DNS	100 Standard query 0x250c AAAA www.thisnameisimpossiblelong9
4252	366.975825151	192.168.64.1	192.168.64.2	DNS	116 Standard query response 0xce0a A www.thisnameisimpossibl
4254	367.062934410	192.168.64.1	192.168.64.2	DNS	100 Standard query response 0x250c AAAA www.thisnameisimposs
4391	424.062966332	192.168.64.2	192.168.64.1	DNS	74 Standard query 0x7ce9 A www.google.com
4392	424.064204834	192.168.64.1	192.168.64.2	DNS	90 Standard query response 0x7ce9 A www.google.com A 142.25
4419	426.388092454	192.168.64.2	192.168.64.1	DNS	74 Standard query 0x7318 A www.offsec.com
4420	426.388781788	192.168.64.2	192.168.64.1	DNS	74 Standard query 0xbbe6 AAAA www.offsec.com

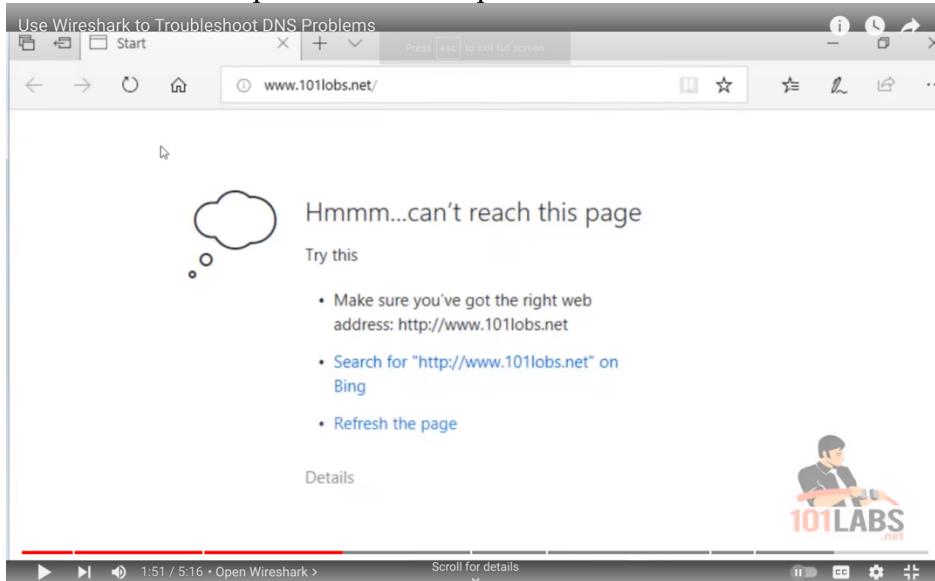
Frame 4254: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface eth0, id 0
Ethernet II, Src: a2:78:17:b2:6d:64 (a2:78:17:b2:6d:64), Dst: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee)
Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.2
User Datagram Protocol, Src Port: 53, Dst Port: 34874
Domain Name System (response)
Transaction ID: 0x250c
Flags: 0x8180 Standard query response, No error

0000 42 e0 d
0010 00 56 f
0020 40 02 0
0030 00 00 0
0040 61 6d 6
0050 6f 6e 6
0060 00 1c 0

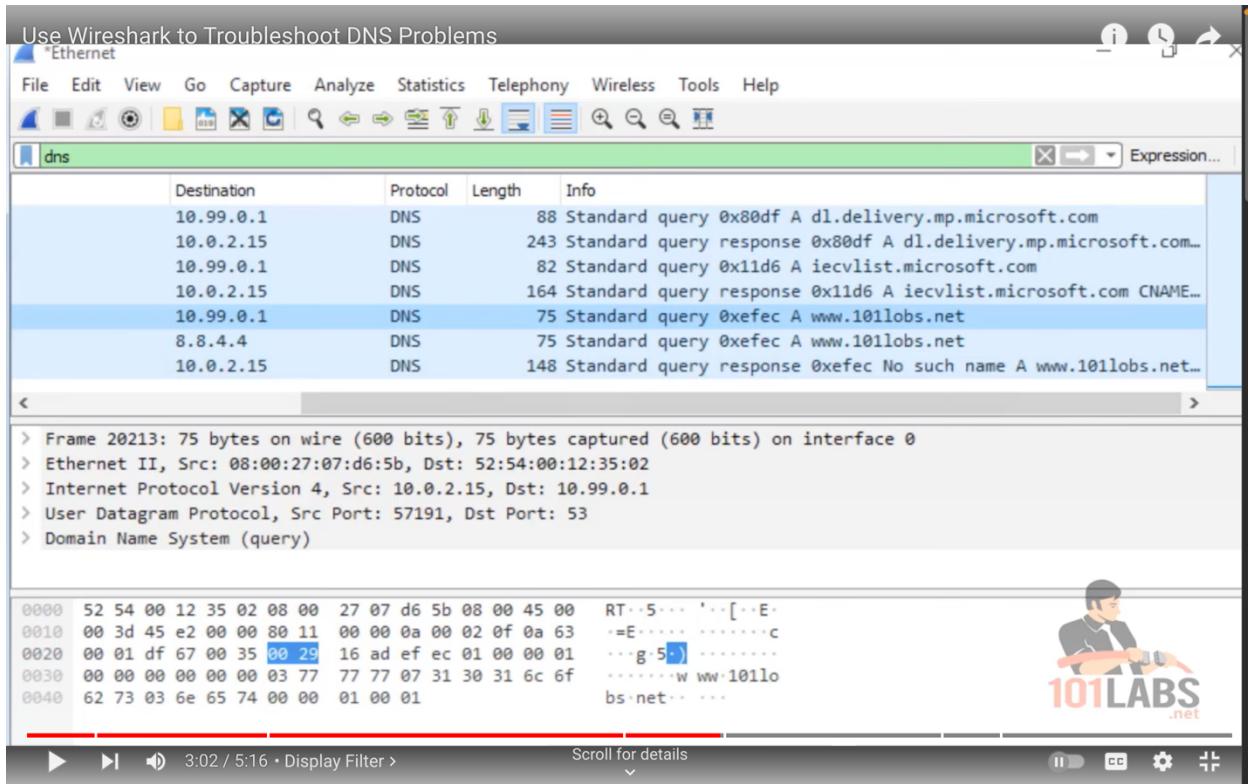
Next, I entered another fake link www.thisnameisimpossiblelong988776544563.net and the DNS response was once again “No error.”

Finally, I watched the following YouTube video by Paul Browning:

<https://www.youtube.com/watch?v=vJsOteThRYA>. This video gave me a reference for how my machine should respond to the DNS packet.

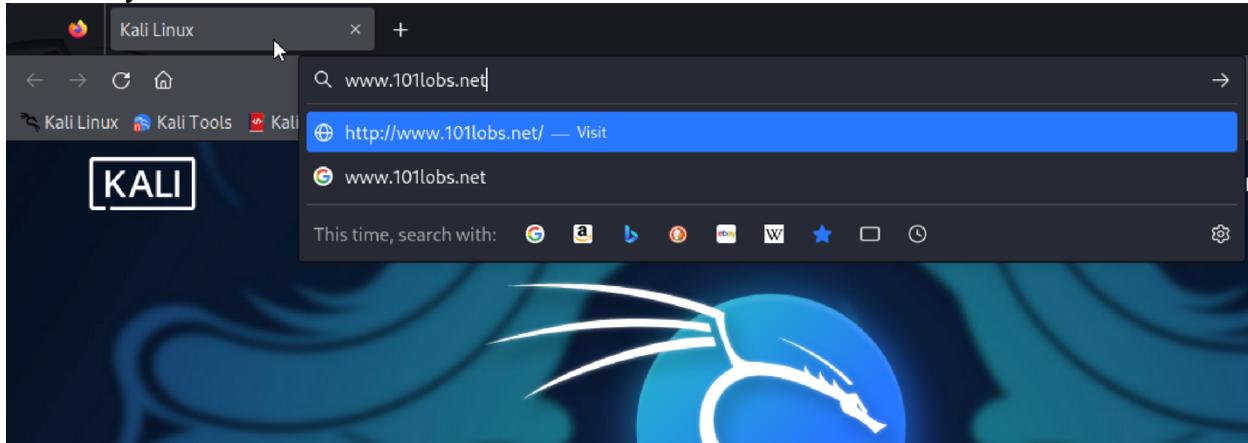


In the video, Browning searches www.101labs.net/

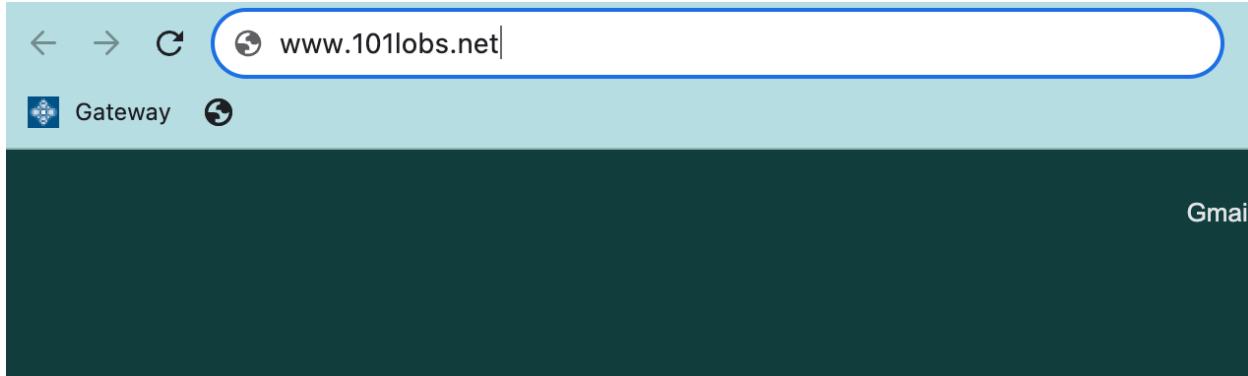


And as shown on the last packet, Browning receives a DNS response of “No such name.”

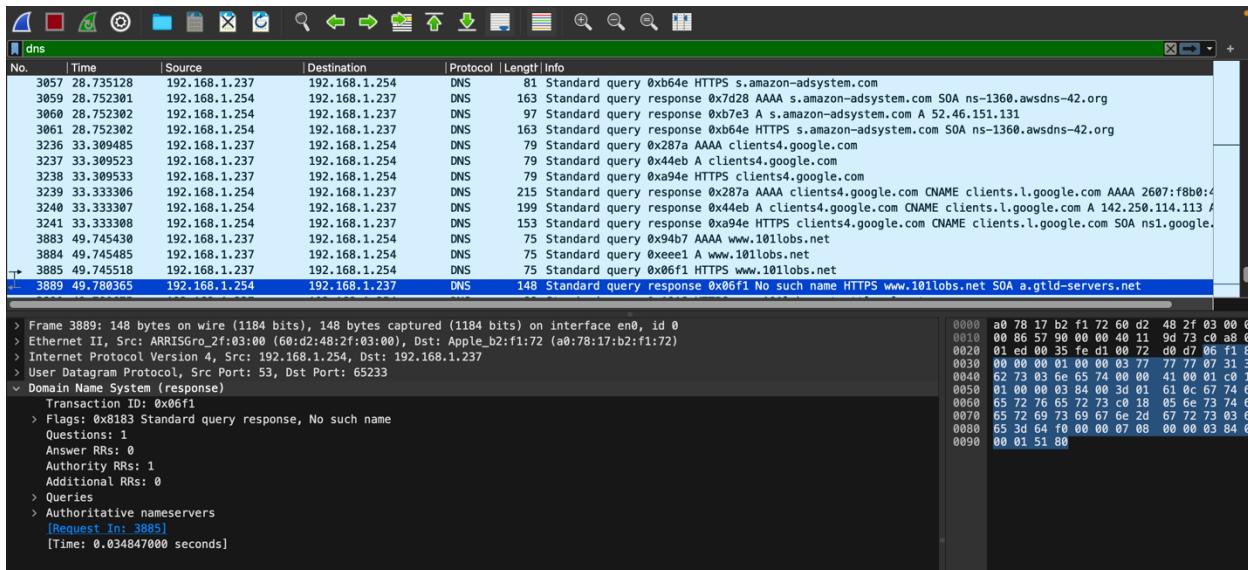
I attempted this process on both my Linux machine and my Mac to see if they would react differently.



I searched www.101labs.net on my Linux machine.



I then searched for the same site on my Mac.

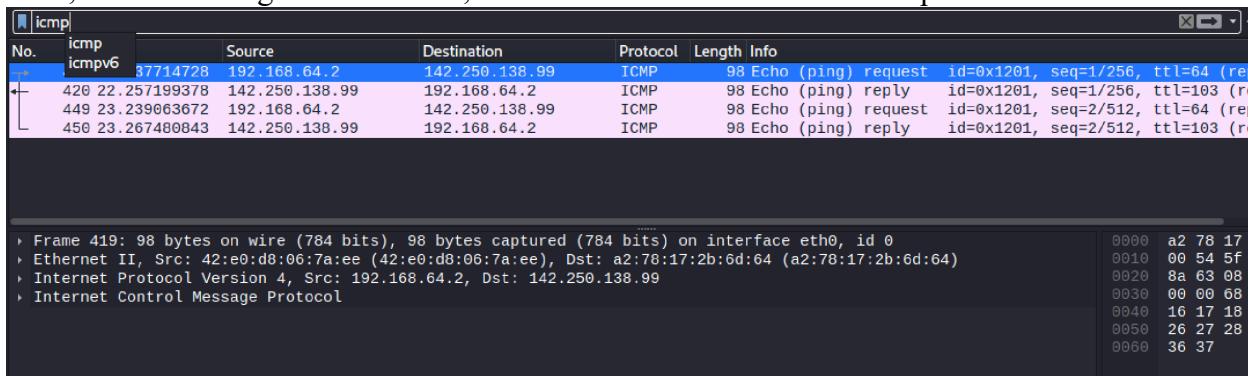


And finally, I got a “No such name” DNS response on Wireshark.

Because my two machines respond differently, it is likely that they are using separate DNS resolvers or DNS configuration settings, which can result in varying DNS response behaviors when querying the same domain.

Step 3: ICMP packet analysis

Next, after removing the DNS filter, I entered ICMP to filter for ICMP packets.



Here are the ICMP packets that were captured. There are multiple ICMP request and replies because I did not stop pinging immediately, therefore it repeated the ping.

icmp							
No.	Time	Source	Destination	Protocol	Length	Info	
→	419 22.237714728	192.168.64.2	142.250.138.99	ICMP	98	Echo (ping) request id=0	
←	420 22.257199378	142.250.138.99	192.168.64.2	ICMP	98	Echo (ping) reply id=0	
↓	449 23.239063672	192.168.64.2	142.250.138.99	ICMP	98	Echo (ping) request id=0	
↓	450 23.267480843	142.250.138.99	192.168.64.2	ICMP	98	Echo (ping) reply id=0	

```

Frame 419: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
Ethernet II, Src: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee), Dst: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64)
Internet Protocol Version 4, Src: 192.168.64.2, Dst: 142.250.138.99
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xa608 [correct]
        [Checksum Status: Good]
    Identifier (BE): 4609 (0x1201)
    Identifier (LE): 274 (0x0112)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
        [Response frame: 420]
    Timestamp from icmp data: Oct 26, 2023 21:09:30.000000000 CDT
        [Timestamp from icmp data (relative): 0.237958545 seconds]
    Data (48 bytes)

```

Looking into the first ICMP request, the ICMP type is 8, which according to IBM's website "ICMP type and code IDs" <https://www.ibm.com/docs/en/qsip/7.4?topic=applications-icmp-type-code-ids>, means that the ICMP packet is an echo packet.

icmp							
No.	Time	Source	Destination	Protocol	Length	Info	
↑	419 22.237714728	192.168.64.2	142.250.138.99	ICMP	98	Echo (ping) request id=0	
↑	420 22.257199378	142.250.138.99	192.168.64.2	ICMP	98	Echo (ping) reply id=0	
↓	449 23.239063672	192.168.64.2	142.250.138.99	ICMP	98	Echo (ping) request id=0	
↓	450 23.267480843	142.250.138.99	192.168.64.2	ICMP	98	Echo (ping) reply id=0	

```

Frame 420: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface eth0, id 0
Ethernet II, Src: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64), Dst: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee)
Internet Protocol Version 4, Src: 142.250.138.99, Dst: 192.168.64.2
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xae08 [correct]
        [Checksum Status: Good]
    Identifier (BE): 4609 (0x1201)
    Identifier (LE): 274 (0x0112)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
        [Request frame: 419]
        [Response time: 19.485 ms]
    Timestamp from icmp data: Oct 26, 2023 21:09:30.000000000 CDT
        [Timestamp from icmp data (relative): 0.257443195 seconds]
    Data (48 bytes)

```

After the echo request packet, the following packet type was type 0, meaning echo reply. Because there was a reply, that proves that Google's servers are open and reachable.

Step 4: TLS

Next, I removed the ICMP filter and searched for the first TLS x.x. "Client Hello" that is associated with www.amazon.com. To find the TLS associated with amazon, I first needed to

find amazon's IP address. To do this I filtered for DNS packets again and found the Amazon query and response.

No.	Time	Source	Destination	Protocol	Length	Info
769	47.339624505	192.168.64.1	192.168.64.2	DNS	97	Standard query response 0xc8d1 A completion.amazon.com A
1192	48.598887397	192.168.64.2	192.168.64.1	DNS	131	Standard query 0x9e69 A p3epsnumangb5dviif52er7wdi.appspot.com
1195	48.599010565	192.168.64.2	192.168.64.1	DNS	131	Standard query 0x2076 AAAA n3epsnumangb5dviif52er7wdi.appspot.com
Frame 769: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface eth0, id 0						
Ethernet II, Src: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64), Dst: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee)						
Internet Protocol Version 4, Src: 192.168.64.1, Dst: 192.168.64.2						
User Datagram Protocol, Src Port: 53, Dst Port: 35074						
Domain Name System (response)						
Transaction ID: 0xc8d1						
Flags: 0x8180 Standard query response, No error						
Questions: 1						
Answer RRs: 1						
Authority RRs: 0						
Additional RRs: 0						
Queries						
completion.amazon.com: type A, class IN						
Answers						
completion.amazon.com: type A, class IN, addr 44.215.119.143						
[Request In: 753]						
[Time: 0.01904649 seconds]						

Once I found the DNS response associated with Amazon, I scrolled to the answers section and found Amazon's IP: 44.215.119.143.

With this IP was able to specify my filter to specifically TLS packets associated with Amazon's IP. I used Wikipedia's website "Wireshark/Display filter -Wikiversity"

https://en.wikiversity.org/wiki/Wireshark/Display_filter to learn how to filter by IP address on Wireshark.

ip.addr == 44.215.119.143 and tls						
No.	Time	Source	Destination	Protocol	Length	Info
780	47.371938554	192.168.64.2	44.215.119.143	TLSv1.2	583	Client Hello
814	47.429888498	44.215.119.143	192.168.64.2	TLSv1.2	2962	Server Hello
816	47.429888915	44.215.119.143	192.168.64.2	TLSv1.2	2962	Certificate, Certificate Status
818	47.429888915	44.215.119.143	192.168.64.2	TLSv1.2	292	Server Key Exchange, Server Hello Done
820	47.438118180	192.168.64.2	44.215.119.143	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, E
840	47.486116996	44.215.119.143	192.168.64.2	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Me
1092	48.282079166	192.168.64.2	44.215.119.143	TLSv1.2	989	[TCP Previous segment not captured] , Application Data
1111	48.341088838	44.215.119.143	192.168.64.2	TLSv1.2	445	[TCP ACKed unseen segment] , Application Data
2346	52.536489500	192.168.64.2	44.215.119.143	TLSv1.2	97	Encrypted Alert
Frame 780: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface eth0, id 0						
Ethernet II, Src: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee), Dst: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64)						
Internet Protocol Version 4, Src: 192.168.64.2, Dst: 44.215.119.143						
Transmission Control Protocol, Src Port: 33208, Dst Port: 443, Seq: 1, Ack: 1, Len: 517						
Transport Layer Security						

To locate Amazon's 'Client Hello,' I used Wireshark's display filter. First, I filtered for Amazon's IP address, which was 44.215.119.143, to isolate packets associated with Amazon's server. Then, I specified 'tls' in the display filter to narrow down the results to TLS-encrypted packets. All together I entered "ip.addr == 44.215.119.143 and tls." Doing this, I was able to successfully filter for TLS packets associated with Amazon's IP address, and I was able to find the initial "Client Hello."

Frame 780: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface eth0, id 0
Ethernet II, Src: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee), Dst: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64)
Internet Protocol Version 4, Src: 192.168.64.2, Dst: 44.215.119.143
Transmission Control Protocol, Src Port: 33208, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
Transport Layer Security
TLSv1.2 Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 512
Handshake Protocol: Client Hello

Next, on the "Client hello" packet, I expanded the "Transport Layer Security" segment.

```
Cipher Suites Length: 34
Cipher Suites (17 suites)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa8)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

And I then expanded the “Cipher Suites” segment to view the list of cipher suites supported.

Step 5: Hex Cheat Sheet

In this step, I reviewed.

Hex File Header and ASCII Equivalent

File headers are used to identify a file by examining the first 4 or 5 bytes of its hexadecimal content.

Filetype	Start	Start ASCII Translation
ani	52 49 46 46	RIFF
au	2E 73 6E 64	snd
bmp	42 4D F8 A9	BM
bmp	42 4D 62 25	BMp%
bmp	42 4D 76 03	BMv
cab	4D 53 43 46	MSCF
dll	4D 5A 90 00	MZ
Excel	D0 CF 11 E0	
exe	4D 5A 50 00	MZP (inno)
exe	4D 5A 90 00	MZ
flv	46 4C 56 01	FLV
gif	47 49 46 38 39 61	GIF89a
gif	47 49 46 38 37 61	GIF87a
gz	1F 8B 08 08	
ico	00 00 01 00	
jpeg	FF D8 FF E1	
jpeg	FF D8 FF E0	JFIF
jpeg	FF D8 FF FE	JFIF
Linux bin	7F 45 4C 46	ELF
png	89 50 4E 47	PNG
msi	D0 CF 11 E0	
mp3	49 44 33 2E	ID3
mp3	49 44 33 03	ID3
OFT	4F 46 54 32	OFT2
PPT	D0 CF 11 E0	
PDF	25 50 44 46	%PDF
rar	52 61 72 21	Rar!
sfw	43 57 53 06/08	cws
tar	1F 8B 08 00	
tgz	1F 9D 90 70	
Word	D0 CF 11 E0	
wmv	30 26 B2 75	
zip	50 4B 03 04	PK

According to the hex cheat sheet, the PDF header values are 25, 50, 44, and 46.

Step 6: Lab 3 Packet Capture

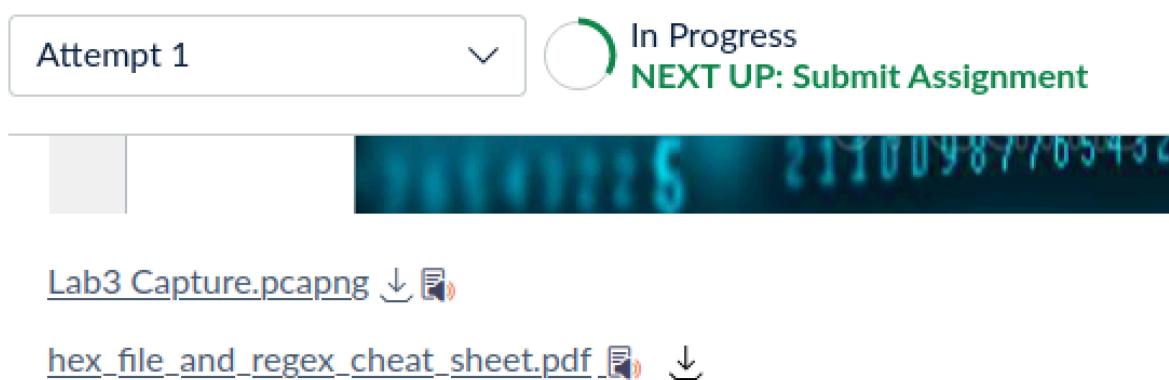
In this step, I needed to download the “Lab3 Capture” file and open it.

Lab 3

Due: Mon Oct 30, 2023 11:59pm

Attempt 1  In Progress

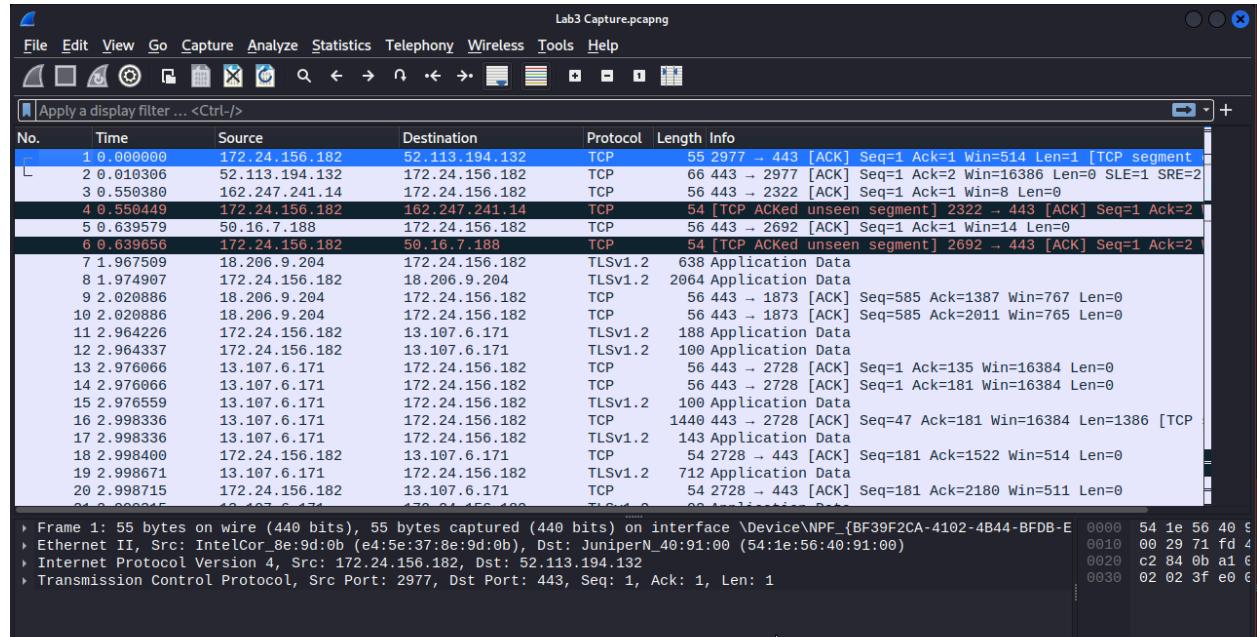
NEXT UP: Submit Assignment



[Lab3 Capture.pcapng](#) 

[hex_file_and_regex_cheat_sheet.pdf](#) 

To begin, I opened Canvas and found the assignment. Once I opened the Lab 3 assignment, I was able to download the “Lab3 Capture” file.



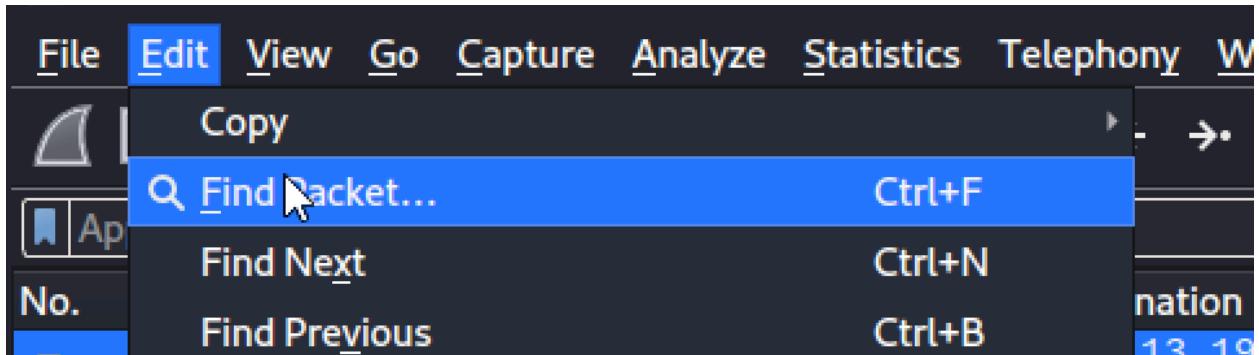
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.24.156.182	52.113.194.132	TCP	55	2977 → 443 [ACK] Seq=1 Ack=1 Win=1 Len=1 [TCP segment]
2	0.010366	52.113.194.132	172.24.156.182	TCP	66	443 → 2977 [ACK] Seq=1 Ack=2 Win=16386 Len=0 SRE=2
3	0.550380	162.247.241.14	172.24.156.182	TCP	56	443 → 2322 [ACK] Seq=1 Ack=1 Win=8 Len=0
4	0.550449	172.24.156.182	162.247.241.14	TCP	54	[TCP ACKed unseen segment] 2322 → 443 [ACK] Seq=1 Ack=2
5	0.639579	50.16.7.188	172.24.156.182	TCP	56	443 → 2692 [ACK] Seq=1 Ack=1 Win=14 Len=0
6	0.639656	172.24.156.182	50.16.7.188	TCP	54	[TCP ACKed unseen segment] 2692 → 443 [ACK] Seq=1 Ack=2
7	1.967589	18.206.9.204	172.24.156.182	TLSv1.2	638	Application Data
8	1.974907	172.24.156.182	18.206.9.204	TLSv1.2	2064	Application Data
9	2.020886	18.206.9.204	172.24.156.182	TCP	56	443 → 1873 [ACK] Seq=585 Ack=1387 Win=767 Len=0
10	2.020886	18.206.9.204	172.24.156.182	TCP	56	443 → 1873 [ACK] Seq=585 Ack=2011 Win=765 Len=0
11	2.964226	172.24.156.182	13.107.6.171	TLSv1.2	188	Application Data
12	2.964337	172.24.156.182	13.107.6.171	TLSv1.2	100	Application Data
13	2.976066	13.107.6.171	172.24.156.182	TCP	56	443 → 2728 [ACK] Seq=1 Ack=135 Win=16384 Len=0
14	2.976066	13.107.6.171	172.24.156.182	TCP	56	443 → 2728 [ACK] Seq=1 Ack=181 Win=16384 Len=0
15	2.976559	13.107.6.171	172.24.156.182	TLSv1.2	100	Application Data
16	2.998336	13.107.6.171	172.24.156.182	TCP	1440	443 → 2728 [ACK] Seq=47 Ack=181 Win=16384 Len=1386 [TCP]
17	2.998336	13.107.6.171	172.24.156.182	TLSv1.2	143	Application Data
18	2.998400	172.24.156.182	13.107.6.171	TCP	54	2728 → 443 [ACK] Seq=181 Ack=1522 Win=514 Len=0
19	2.998671	13.107.6.171	172.24.156.182	TLSv1.2	712	Application Data
20	2.998715	172.24.156.182	13.107.6.171	TCP	54	2728 → 443 [ACK] Seq=181 Ack=2180 Win=511 Len=0
21	2.998945	13.107.6.171	172.24.156.182	TLSv1.2	200	Application Data

Frame 1: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface \Device\NPF_{BF39F2CA-4102-4B44-BFDB-E 0000 54 1e 56 40 9
Ethernet II, Src: IntelCor_8e:9d:0b (e4:5e:37:8e:9d:0b), Dst: JuniperN_40:91:00 (54:1e:56:40:91:00)
Internet Protocol Version 4, Src: 172.24.156.182, Dst: 52.113.194.132
Transmission Control Protocol, Src Port: 2977, Dst Port: 443, Seq: 1, Ack: 1, Len: 1

Next, I opened the packet capture file.

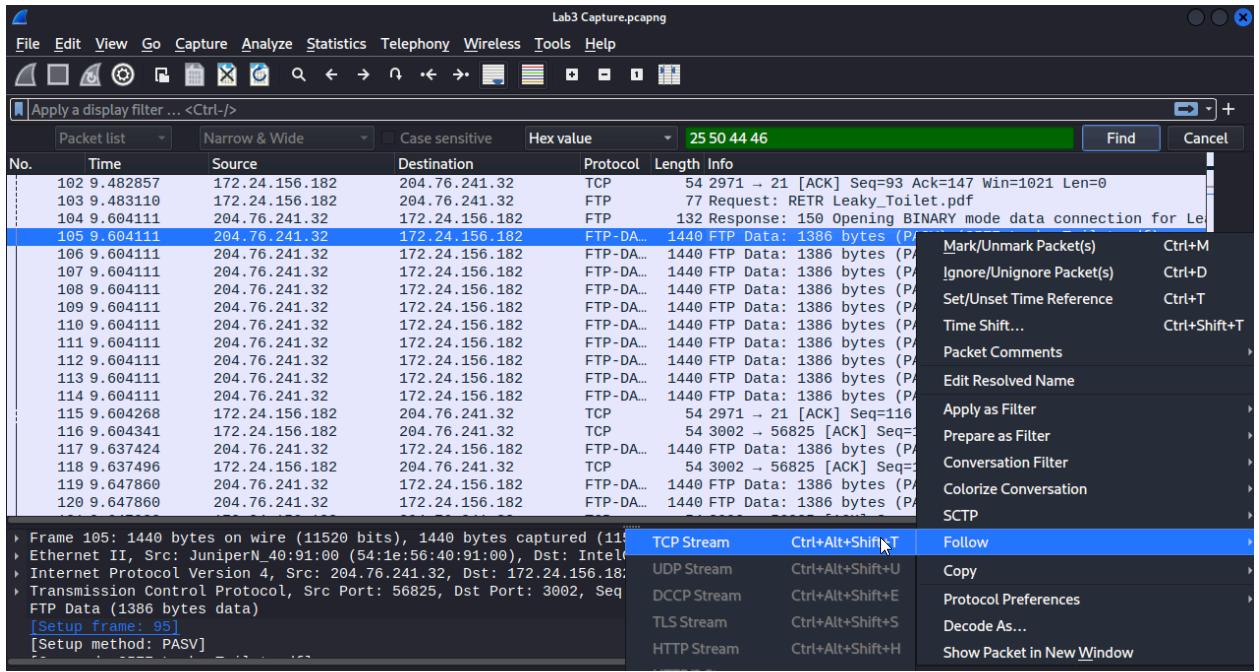


Selected Edit, then "Find Packet..."

The screenshot shows the Wireshark interface with the following details:

- Panels:** The top navigation bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, and Wireless. Below it is a toolbar with various icons. The main area has two panels: 'Packet list' on the left and 'Details' on the right.
- Display Filter:** The display filter bar at the top of the packet list pane shows the filter '25 50 44 46'.
- Packet List:** The packet list pane shows several network frames. Frame 105 is selected, showing details like Source (204.76.241.32), Destination (172.24.156.182), Protocol (FTP), and Length (1440). The Info column shows the content of the selected frame: 'FTP Data: 1386 bytes (PASV) (SIZE Leaky_Toilet.pdf)'.
- Details:** The details pane on the right displays the raw hex and ASCII data for the selected frame. The hex dump shows the sequence of bytes, and the ASCII dump shows the corresponding characters.
- Status Bar:** The status bar at the bottom right shows 'nation 13 19'.

And I changed the display filter to "Hex value" and entered the header values from the cheat sheet (25, 50, 44, 46) and clicked 'Find.'



I then selected the found packet and followed its TCP stream.

Wireshark - Follow TCP Stream (tcp.stream eq 14) · Lab3 Capture.pcapng

```
%PDF-1.6
%...
54 0 obj
<</Filter/FlateDecode/First 9/Length 333/N 2/Type/ObjStm>>stream
h..Q.n.0.....=.....DQ..n..hpZ..#c..k.....Y..xg....jh
P....Q...${Y7.J....!....@0...y.$0'IV}.>I....6..R.c.A..P....F...u'c%Y].....5.5..p}.T...
....h..W....97..~X?3d.c...?!.5.U.6...Y#Or.....\....(
..t.....O.....2..m..q&m..)7f.....0.`....B<#.gh.Gg...Y3...Ks$.zC
y3....6...>E..D...{.~....n.n..c.....p..?..$.....d
endstream
endobj[REDACTED]
55 0 obj
<</Filter/FlateDecode/First 46/Length 542/N 7/Type/ObjStm>>stream
h...Qk.@.....G.0.....n..F.Q.....;i...'s..0...|.'...bP
..)P1h$H...'ZB..8..@9m...*.M>..6.]....}.0]...M...,.T...(F.Ey]6.v>...yNk ....Y..k[..J"
..n..K...S.z..y...A....c....w.1.c..r1.%k
..k.l...'.`....F.z....c<>..6m...,0..r.].bq.....f...~
..w.Q...
G.H....Bsi...S....N.B`..Zk.....#....9.eH...?V..=.a.A....k.t..6.Q..@F.
..`..A.V.k9.&.-..d8...Mt..W...'.Q..l.&.lmWe...d....r..o.;..7#....Y...k1.'.....Y.....
.C....4....._....~.0o....5..P...../.).(*.'....`.....9K.Zm<b)n-.Aw..=.....
P...d?gy...j4...G..KY.Q.....(.....
endstream
endobj[REDACTED]
1 0 obj
<</Metadata 2 0 R/OCProperties<</D<</OFF[]/Order[18 0 R]/RBGroups[]>>/OCGs[18 0 R]>>/P
ages 3 0 R/Type/Catalog>
endobj[REDACTED]
2 0 obj

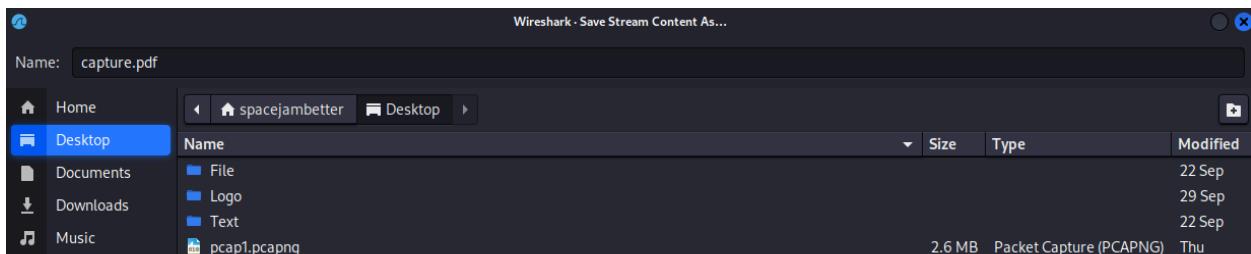
```

Packet 105. 0 client pkt(s), 574 server pkt(s), 0 turn(s). Click to select.

Entire conversation (789 kB) Show data as ASCII Stream 14 Find Next

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

Screenshot of the TCP stream.



I altered the "Show data as" section of the TCP stream and selected "Raw," and I saved it as a pdf on my Desktop named "capture.pdf."

After saving the capture as a pdf, I opened my terminal to open the file.

```
(spacejambetter㉿kali)-[~]
$ cd Desktop

(spacespacejambetter㉿kali)-[~/Desktop]
$ ls
File Logo Text capture.pdf pcap1.pcapng pcap2.pcapng
No. Time Source Destination
(spacespacejambetter㉿kali)-[~/Desktop]
$ open capture.pdf
```

Screenshot of me using the terminal to open the capture.pdf file.



Screenshot of the contents of the capture.pdf file. It appears to be an infographic on fixing leaky toilets. Very helpful!

Questions for the Lab

1. What is the Domain Name System (DNS)? What would happen if the DNS didn't function properly?

DNS is a protocol designed to convert human-friendly names to IP addresses. This facilitates the access of websites because, without DNS, users would have to memorize a site's IP address to access the website. If DNS were to not function properly, accessing website would be difficult as most people do not memorize their favorite website's IP address. Another potential issue with DNS is that if it is mapped incorrectly, users could be redirected to the wrong address. Regardless of the issue, DNS problems make it difficult to access a desired site.

2. What is the purpose of ICMP?

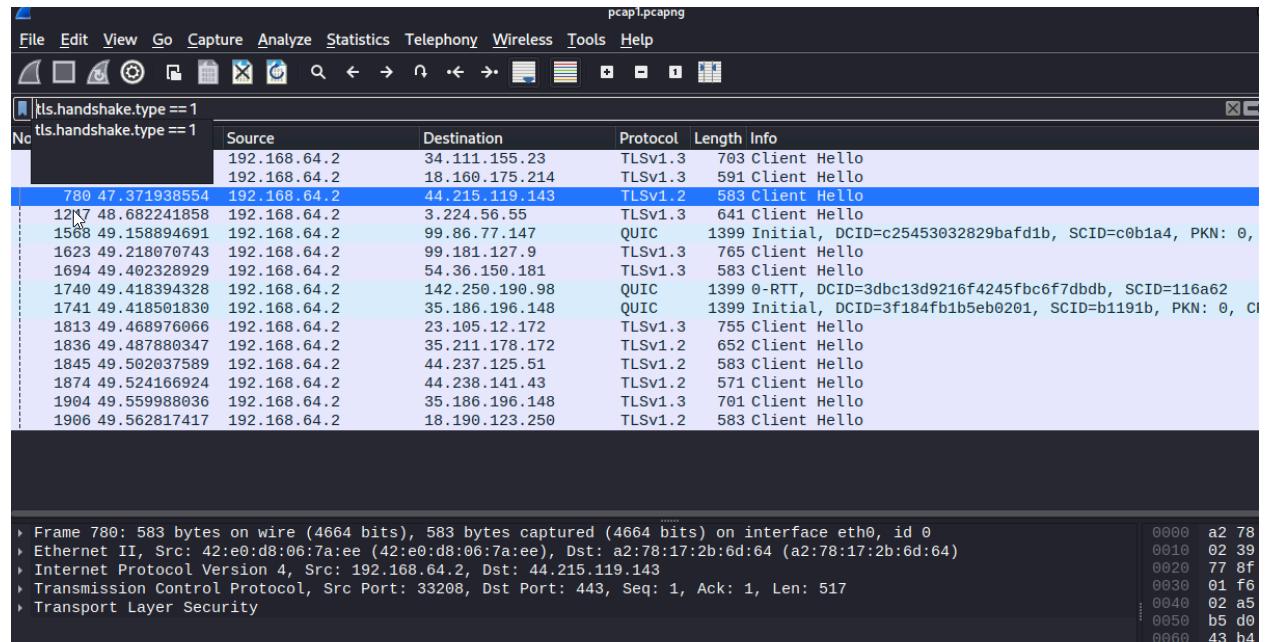
ICMP is used for network diagnostics and analyzing the network status. By using ICMP echo requests (pinging), one could view the availability of a specific port or server. In this lab, I pinged the google server to verify if it was available and reachable.

3. What is TLS? Which version did you observe in your packet capture?

TLS is a cryptographic protocol designed to secure and encrypt communication over a network. In this lab, I observed TLSv1.2. According to Gigamon Blog's article, "What is TLS 1.2 and Why Should You (Still) Care?" <https://blog.gigamon.com/2021/07/14/what-is-tls-1-2-and-why-should-you-still-care/>, TLSv1.2 provides increased security by using a single hash and both symmetric and asymmetric cryptography.

4. Is there a Wireshark filter that could have saved you time in finding the “Client Hello.”

I had no problem finding the ‘Client Hello’ for Amazon because I used the “ip.addr == 44.215.119.143 and tls” filter to find TLS packets associated with Amazon’s IP. But there is a better way to search for “Client Hello” packets. I read InsidePacket’s webpage “Wireshark Filter for SSL Traffic” <https://davidwzhang.com/2018/03/16/wireshark-filter-for-ssl-traffic/> and learned that I could filter for SSL “Client Hello” packets using the following filter “ssl.handshake.type == 1.” Because SSL is similar to TLS (they are both cryptographic protocols that use a handshake), I decided to see if the filter works the same way if I replace SSL with TLS.

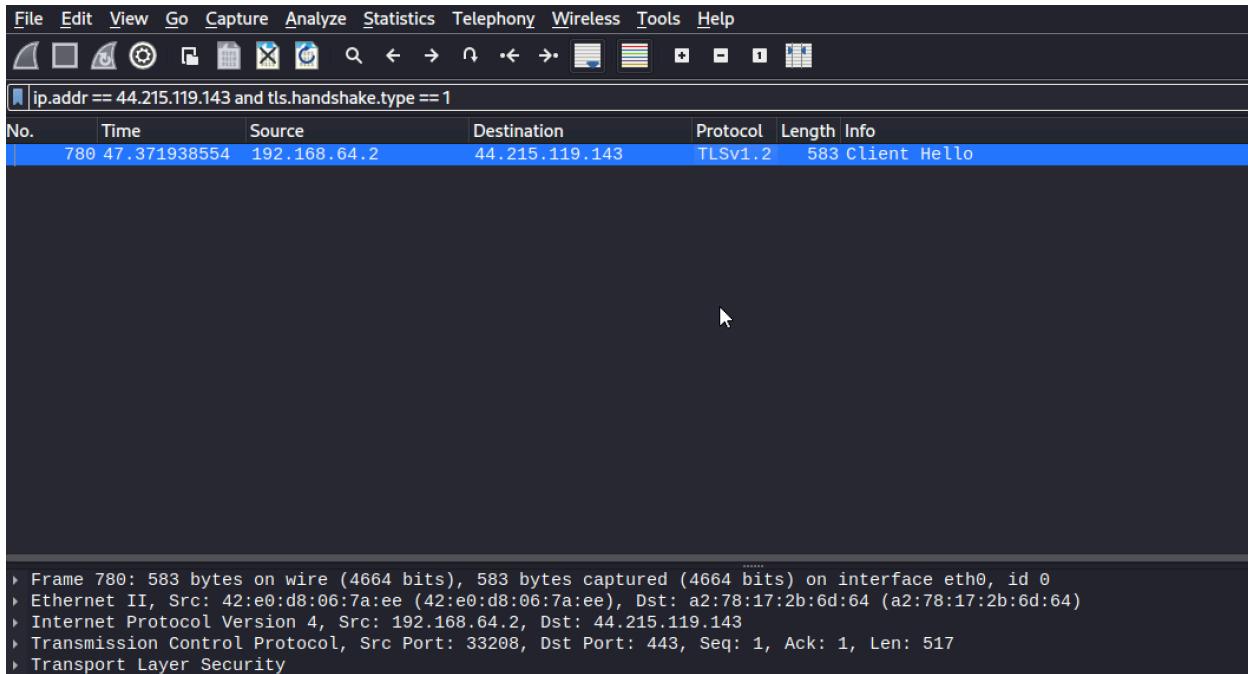


No	Source	Destination	Protocol	Length	Info
1	192.168.64.2	34.111.155.23	TLSv1.3	703	Client Hello
2	192.168.64.2	18.160.215.214	TLSv1.3	591	Client Hello
3	780 47.371938554	192.168.64.2	44.215.119.143	583	Client Hello
4	121 47.682241858	192.168.64.2	3.224.56.55	641	Client Hello
5	1568 49.158894691	192.168.64.2	99.86.77.147	1399	Initial, DCID=c25453032829bafdf1b, SCID=c0b1a4, PKN: 0, C
6	1623 49.218070743	192.168.64.2	99.181.127.9	765	Client Hello
7	1694 49.402328929	192.168.64.2	54.36.150.181	583	Client Hello
8	1740 49.418394328	192.168.64.2	142.250.190.98	1399	0-RTT, DCID=3dbc13d9216f4245fb6f7dbdb, SCID=116a62
9	1741 49.418501830	192.168.64.2	35.186.196.148	1399	Initial, DCID=3f184fb1b5eb0201, SCID=b1191b, PKN: 0, C
10	1813 49.468976066	192.168.64.2	23.105.12.172	755	Client Hello
11	1836 49.487880347	192.168.64.2	35.211.178.172	652	Client Hello
12	1845 49.502037589	192.168.64.2	44.237.125.51	583	Client Hello
13	1874 49.524166924	192.168.64.2	44.238.141.43	571	Client Hello
14	1904 49.559988036	192.168.64.2	35.186.196.148	701	Client Hello
15	1906 49.562817417	192.168.64.2	18.190.123.250	583	Client Hello

Frame 780: 583 bytes on wire (4664 bits), 583 bytes captured (4664 bits) on interface eth0, id 0
Ethernet II, Src: 42:e0:d8:06:7a:ee (42:e0:d8:06:7a:ee), Dst: a2:78:17:2b:6d:64 (a2:78:17:2b:6d:64)
Internet Protocol Version 4, Src: 192.168.64.2, Dst: 44.215.119.143
Transmission Control Protocol, Src Port: 33208, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
Transport Layer Security

I entered “tls.handshake.type == 1” and successfully filtered for TLS “Client Hello” packets.

This is an effective way to filter for “Client Hello” packets for the entire network traffic, but if you’re looking specific “Client Hello” packets to an IP address, there is another way.



I entered both the IP filter and the TLS filter to find the Client Hello associated with Amazon's IP.

5. What is FTP? How are you able to capture the contents of a .pdf using Wireshark in this example?

FTP (file transfer protocol) is a protocol designed to transfer files across the internet. Using Wireshark, I was able to capture an FTP packet labeled FTP-Data. Because FTP (File Transfer Protocol) is designed to transfer files across the internet, I knew that there was some sort of file transfer occurring. More specifically, I identified the type of file being transferred as a PDF because I filtered for the Hex Header values associated with PDFs (25 50 44 46). Next, I analyzed the file by following the TCP stream and converting the data to raw. When data is converted to “raw,” it is converted into hexadecimal values which could be interpreted by a computer. Once the data was converted to raw, I was able to export the file as a pdf and open the image.

LIMITATIONS/CONCLUSION

In this lab I learned about Network sniffing through Wireshark. I learned about several protocols like FTP, ICMP, DNS, and TLS, and I learned about useful Wireshark filters which could facilitate traffic analysis. One limitation I had during the lab was I was unable to recreate the “no such name” DNS response on my Kali Linux virtual computer. It is likely that the Wireshark is configured differently on the machine as I was able to recreate the “no such name” through Wireshark on my Mac.

REFERENCES

BlueCat Networks: “The top four DNS response codes and what they mean”

<https://bluecatnetworks.com/blog/the-top-four-dns-response-codes-and-what-they-mean/>

I used this website to understand the different DNS responses and what they mean.

IBM: “ICMP type and code IDs”

<https://www.ibm.com/docs/en/qsip/7.4?topic=applications-icmp-type-code-ids>

I used this website to understand the different type of ICMP types and what they mean.

Paul Browning: “Use Wireshark to Troubleshoot DNS Problems”

<https://www.youtube.com/watch?v=vJsOteThRYA>

This video gave me a reference to how my machine should react to a fake URL.

Gigamon Blog: “What is TLS 1.2 and Why Should You (Still) Care?”

<https://blog.gigamon.com/2021/07/14/what-is-tls-1-2-and-why-should-you-still-care/>

I used this article to understand TLSv1.2 and how it works.

Wikipedia: “Wireshark/Display filter -Wikiversity”

https://en.wikiversity.org/wiki/Wireshark/Display_filter

I used this website to learn how to filter by IP address on Wireshark.