Labs - Intermediate Network Analysis

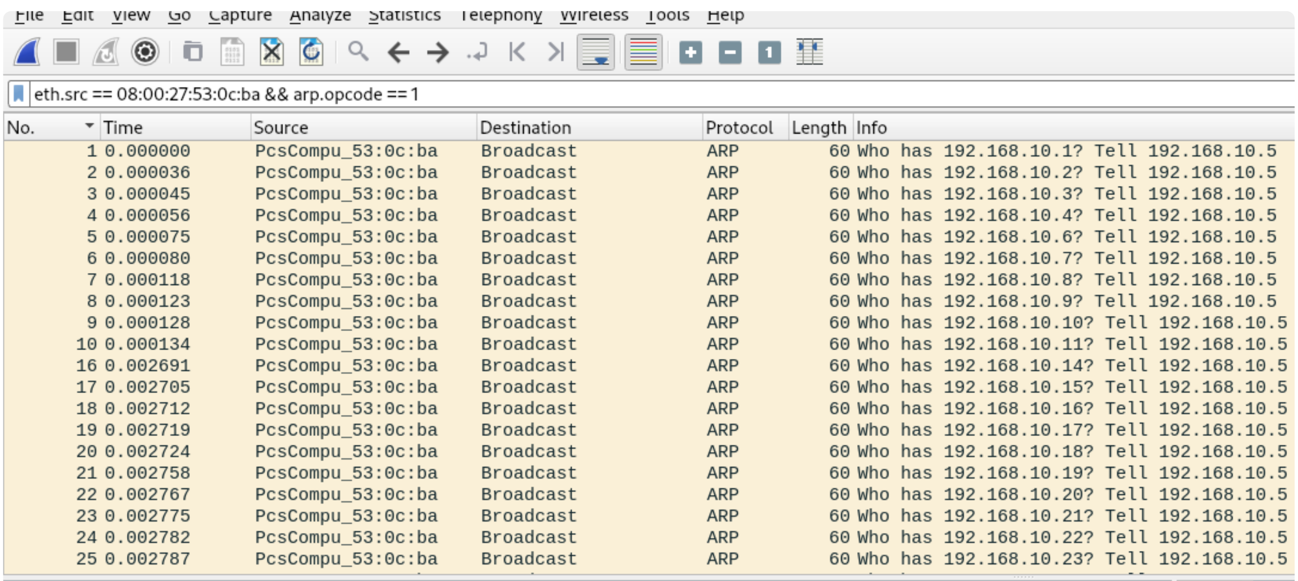
Link Layer Attack

ARP Spoofing & Abnormality Detection

Question

- Inspect the ARP_Poison.pcapng file, part of this module's resources, and submit the total count of ARP requests (opcode 1) that originated from the address 08:00:27:53:0c:ba as your answer.
  -> We filter for source mac address 08:00:27:53:0c:ba and all ARP requests associated with it on the display filter

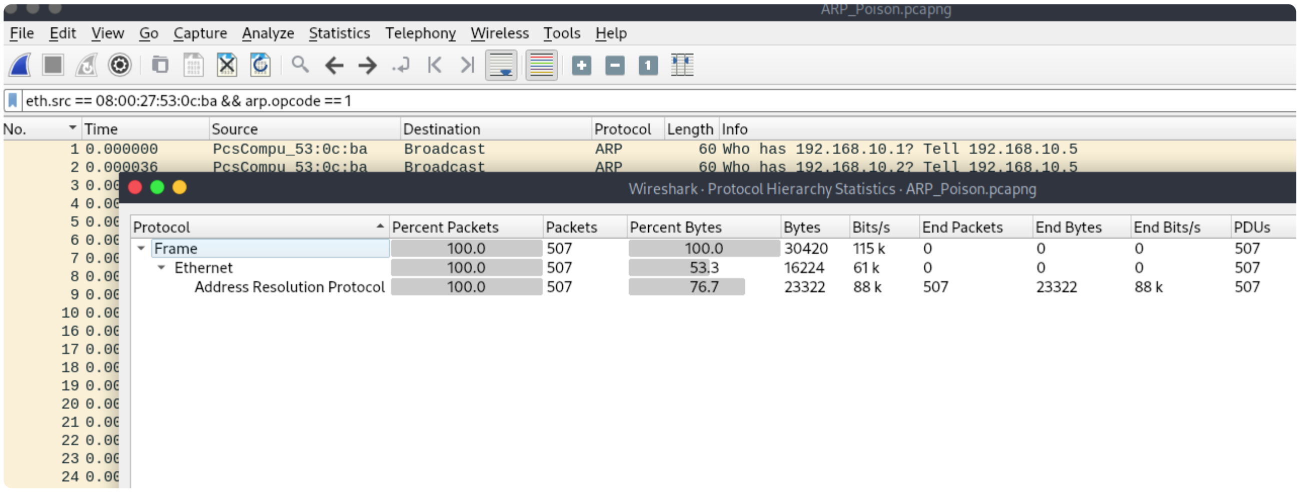
```
eth.src == 08:00:27:53:0c:ba && arp.opcode == 1
```


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

eth.src == 08:00:27:53:0c:ba && arp.opcode == 1

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.1? Tell 192.168.10.5
2	0.000036	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.2? Tell 192.168.10.5
3	0.000045	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.3? Tell 192.168.10.5
4	0.000056	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.4? Tell 192.168.10.5
5	0.000075	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.6? Tell 192.168.10.5
6	0.000080	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.7? Tell 192.168.10.5
7	0.000118	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.8? Tell 192.168.10.5
8	0.000123	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.9? Tell 192.168.10.5
9	0.000128	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.10? Tell 192.168.10.5
10	0.000134	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.11? Tell 192.168.10.5
16	0.002691	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.14? Tell 192.168.10.5
17	0.002705	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.15? Tell 192.168.10.5
18	0.002712	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.16? Tell 192.168.10.5
19	0.002719	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.17? Tell 192.168.10.5
20	0.002724	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.18? Tell 192.168.10.5
21	0.002758	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.19? Tell 192.168.10.5
22	0.002767	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.20? Tell 192.168.10.5
23	0.002775	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.21? Tell 192.168.10.5
24	0.002782	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.22? Tell 192.168.10.5
25	0.002787	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.23? Tell 192.168.10.5

-> We look at protocol hierachy to see the total number of packets, as we can see the packet no. changed from 10 to 16 so looking at it wouldn't do.



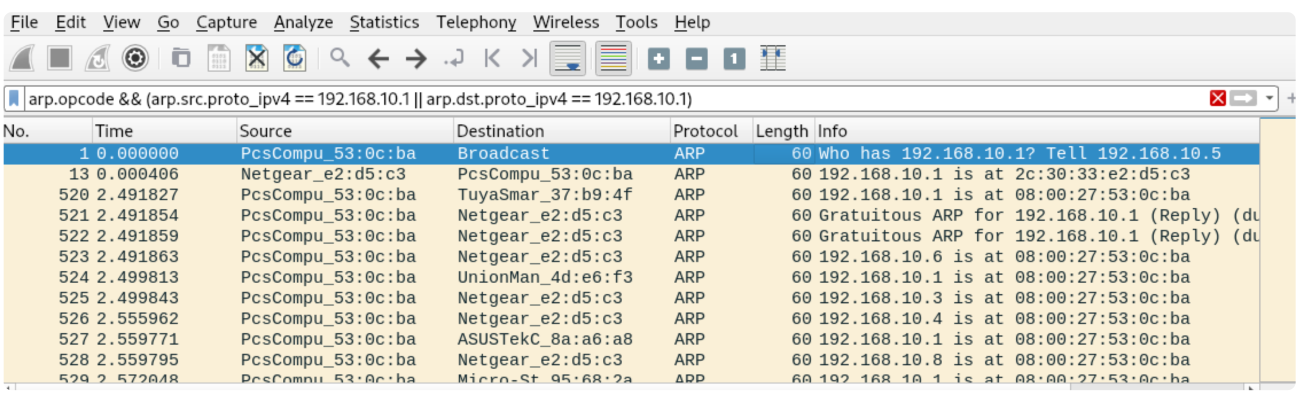
-> Hence, there are 507 packets and we can what the host is doing, asking for the arp address of every host, what good can it do?

ARP Scanning & Denial-of-Service

Question

- Inspect the ARP_Poison.pcapng file, part of this module's resources, and submit the first MAC address that was linked with the IP 192.168.10.1 as your answer.
  -> We set an display filter with the associated ip address 192.168.10.1

```
arp.opcode && (arp.src.proto_ipv4 == 192.168.10.1 || arp.dst.proto_ipv4
== 192.168.10.1)
```



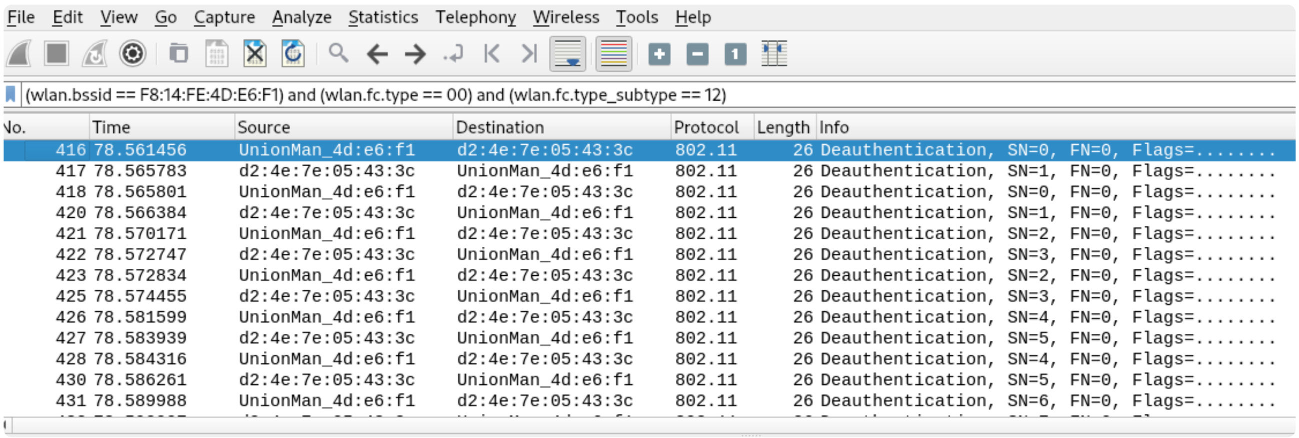
-> Hence we see that it has an mac address of 2c:30:33:e2:d5:c3

802.11 Denial of Service

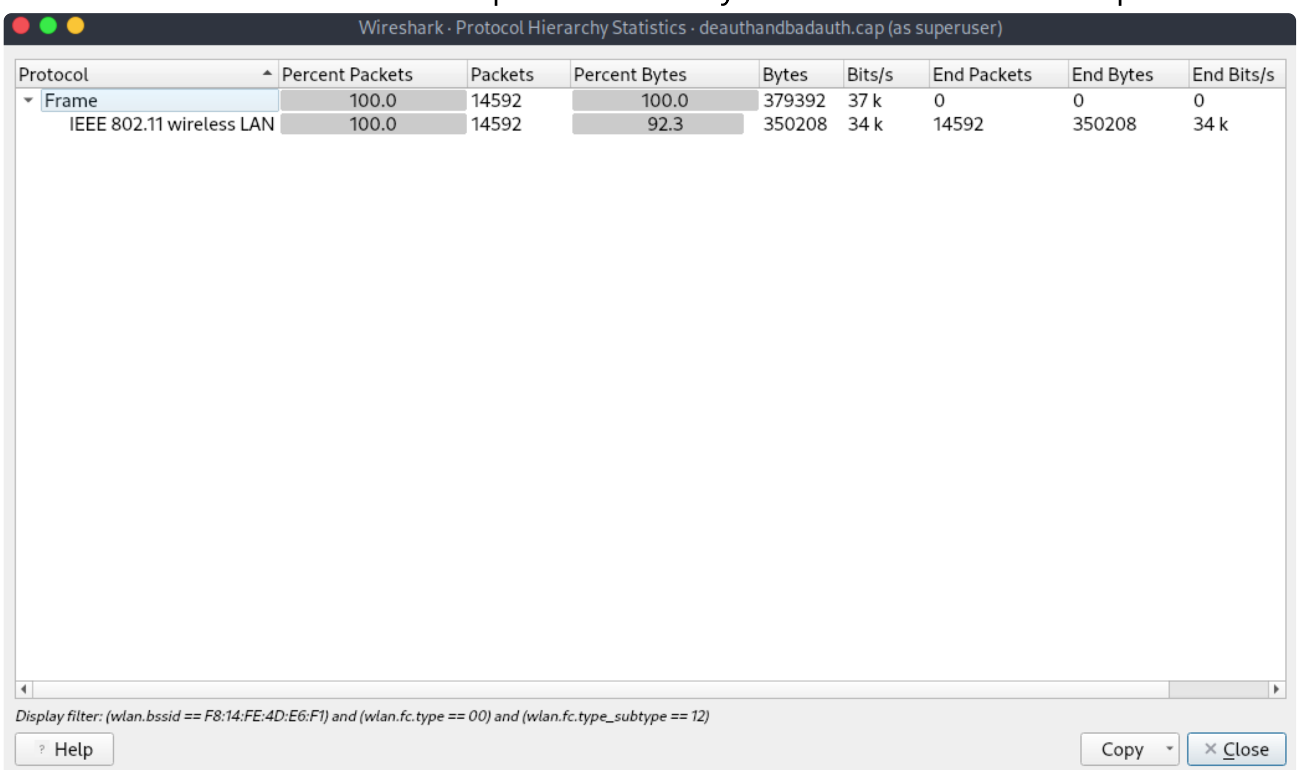
Question

- Inspect the deauthandbadauth.cap file, part of this module's resources, and submit the total count of deauthentication frames as your answer.
  -> We look at total count of deauthentication frames (given the mac address of access point) for management frame with deauthentication type.

```
(wlan.bssid == F8:14:FE:4D:E6:F1) and (wlan.fc.type == 00) and
(wlan.fc.type_subtype == 12)
```



-> Next we look at the statistics -> protocol hierachy to see the total number of packets:



-> Hence, we have 14592 deauthentication packets.

Rogue Access Point & Evil-Twin Attacks

Question

- Inspect the rogueap.cap file, part of this module's resources, and enter the MAC address of the Evil Twin attack's victim as your answer.
  -> We first look for beacon frame, which contains information about the network

```
(wlan.fc.type == 00) and (wlan.fc.type_subtype == 8)
```

```
30 21.306151    UnionMan_4d:e6:f2    Broadcast    802.11    78 Beacon fram
```

```
▸ Frame 30: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▸ IEEE 802.11 Beacon frame, Flags: ........
▾ IEEE 802.11 Wireless Management
  ▾ Fixed parameters (12 bytes)
      Timestamp: 102402
      Beacon Interval: 0.102400 [Seconds]
    ▸ Capabilities Information: 0x0401
  ▾ Tagged parameters (42 bytes)
    ▸ Tag: SSID parameter set: "HTB-Wireless"
    ▸ Tag: Supported Rates 1(B), 2(B), 5.5(B), 11(B), 6, 9, 12, 18, [Mbit/sec]
    ▸ Tag: DS Parameter set: Current Channel: 4
    ▸ Tag: Traffic Indication Map (TIM): DTIM 0 of 1 bitmap
    ▸ Tag: ERP Information
    ▸ Tag: Extended Supported Rates 24, 36, 48, 54, [Mbit/sec]
```

-> We see that the mac address of f8:14:fe:4d:e6:f2 does not have RSN (Robust Security Information), which shows that is is likely the rogue access point.

-> We now filter for this access point (mac address) along with the arp reqeusts, as having an arp reqeust to the network indicates that the host likely fall victim under the evil-twin attack:

```
(wlan.bssid == f8:14:fe:4d:e6:f2) && arp.opcode
```

(wlan.bssid == f8:14:fe:4d:e6:f2) && arp.opcode

No.	Time	Source	Destination	Protocol	Length	Info
179	44.853222	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
180	44.855353	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
181	45.673839	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
182	45.677012	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
200	46.658241	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
201	46.660397	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169.254.63.254? (ARP Probe)
205	47.657928	IntelCor_af:eb:91	Broadcast	ARP	60	ARP Announcement for 169.254.63.254
206	47.660230	IntelCor_af:eb:91	Broadcast	ARP	60	ARP Announcement for 169.254.63.254

-> Hence, we see that one of the host has fall into it.

-> Examining the first packet, we see that

No.	Time	Source	Destination	Protocol	Length	Info
179	44.853222	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
180	44.855353	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
181	45.673839	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
182	45.677012	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
200	46.658241	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
201	46.660397	IntelCor_af:eb:91	Broadcast	ARP	60	Who has 169
205	47.657928	IntelCor_af:eb:91	Broadcast	ARP	60	ARP Announc
206	47.660230	IntelCor_af:eb:91	Broadcast	ARP	60	ARP Announc

```
▸ Frame 179: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▾ IEEE 802.11 Data, Flags: .......T
    Type/Subtype: Data (0x0020)
  ▸ Frame Control Field: 0x0801
    .000 0000 0011 0000 = Duration: 48 microseconds
    Receiver address: UnionMan_4d:e6:f2 (f8:14:fe:4d:e6:f2)
    Transmitter address: IntelCor_af:eb:91 (2c:6d:c1:af:eb:91)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Source address: IntelCor_af:eb:91 (2c:6d:c1:af:eb:91)
    BSS Id: UnionMan_4d:e6:f2 (f8:14:fe:4d:e6:f2)
    STA address: IntelCor_af:eb:91 (2c:6d:c1:af:eb:91)
    .... .... .... 0000 = Fragment number: 0
    0000 0000 1101 .... = Sequence number: 13
▸ Logical-Link Control
▸ Address Resolution Protocol (ARP Probe)
```

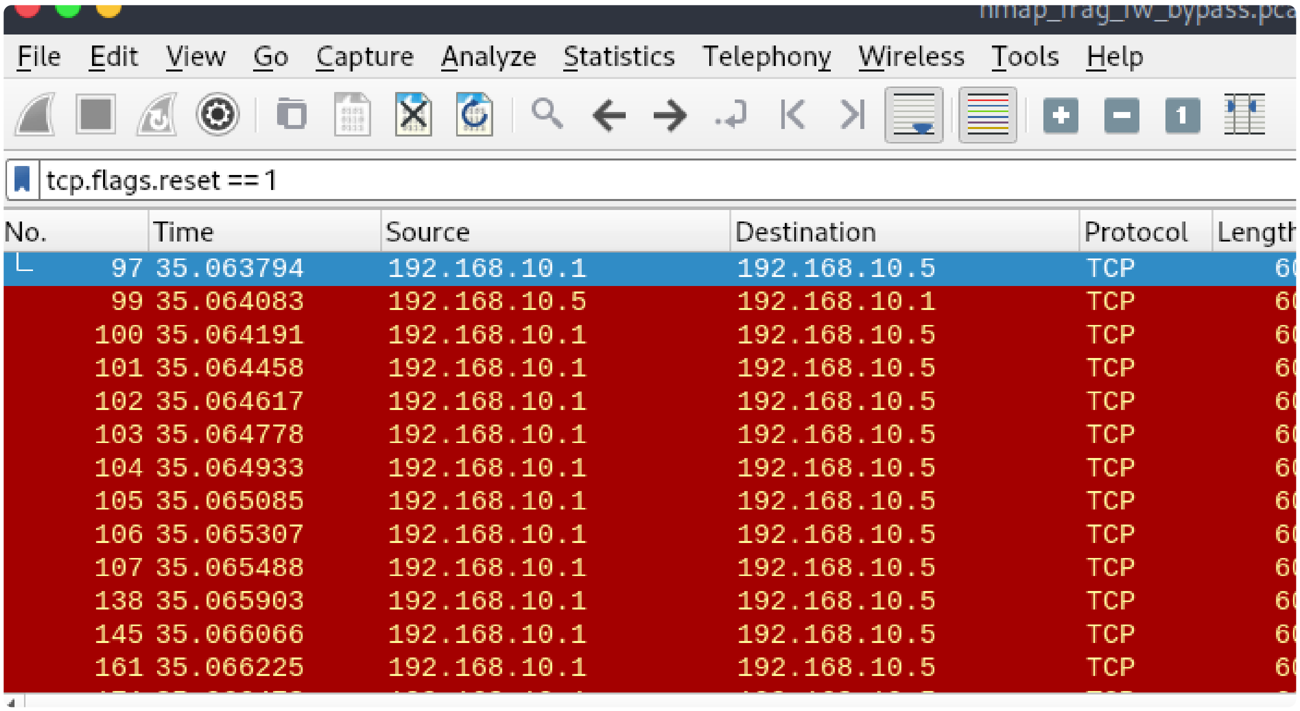
-> it has an MAC of 2c:6d:c1:af:eb:91

Detecting Network Abnormalities

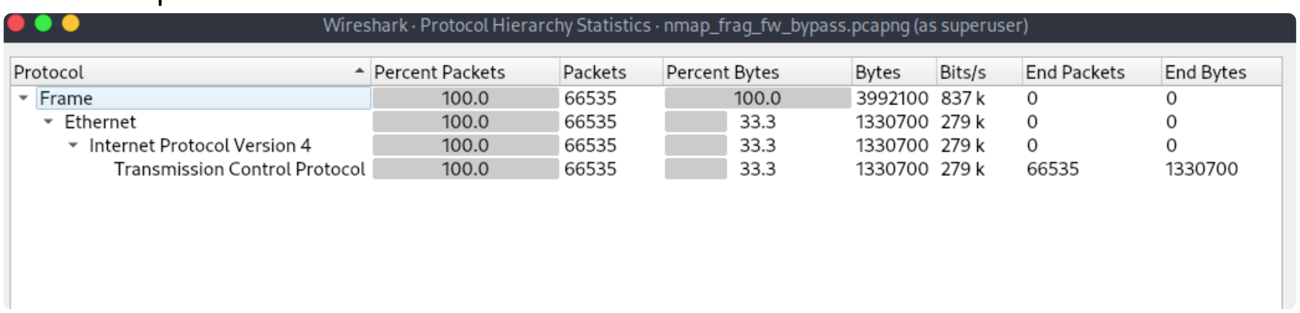
Fragmentation Attacks

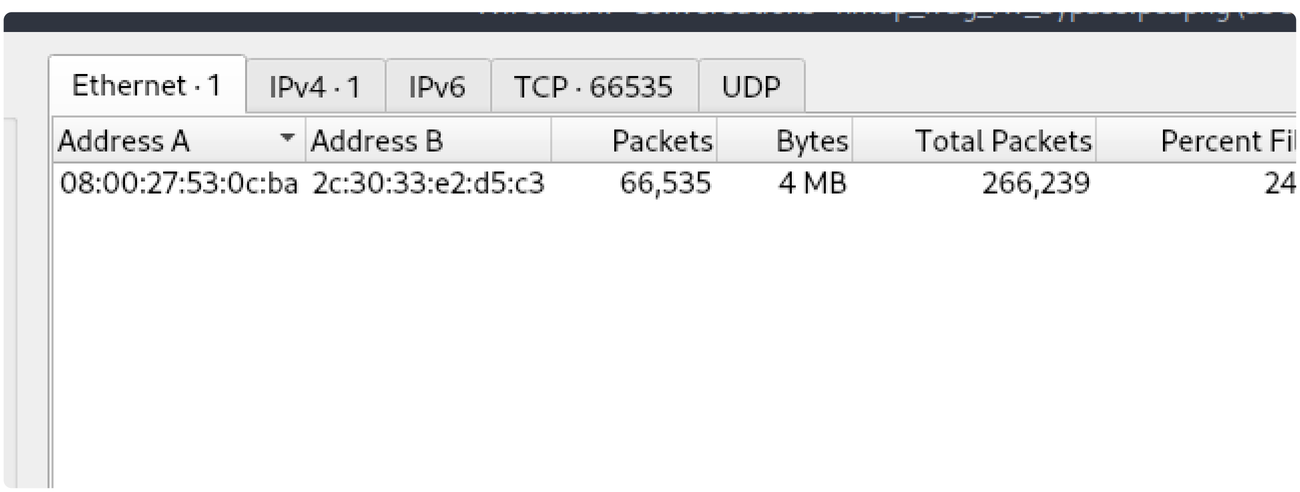
- Inspect the nmap_frag_fw_bypass.pcapng file, part of this module's resources, and enter the total count of packets that have the TCP RST flag set as your answer.
  -> We did some searching up online on how to filter for rest packets, and the answer is to apply the filter tcp.flags.rest ==1, which is shown below

```
tcp.flags.reset == 1
```



-> Looking at protocol hierarchy or conversations in the statistics tab would show the number of packets:





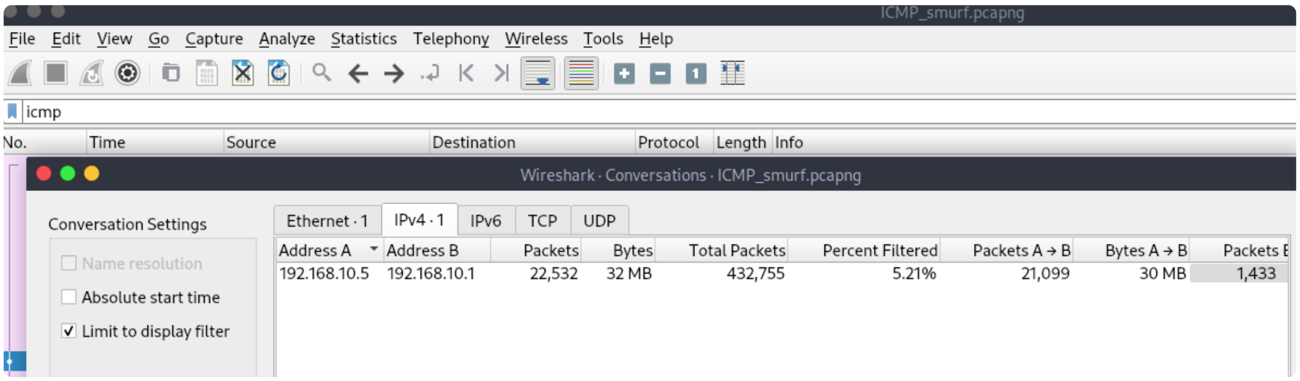
-> Hence, we would get 66,535 packets are send, indicative of a full port scan!

IP Source & Destination Spoofing Attacks

Question

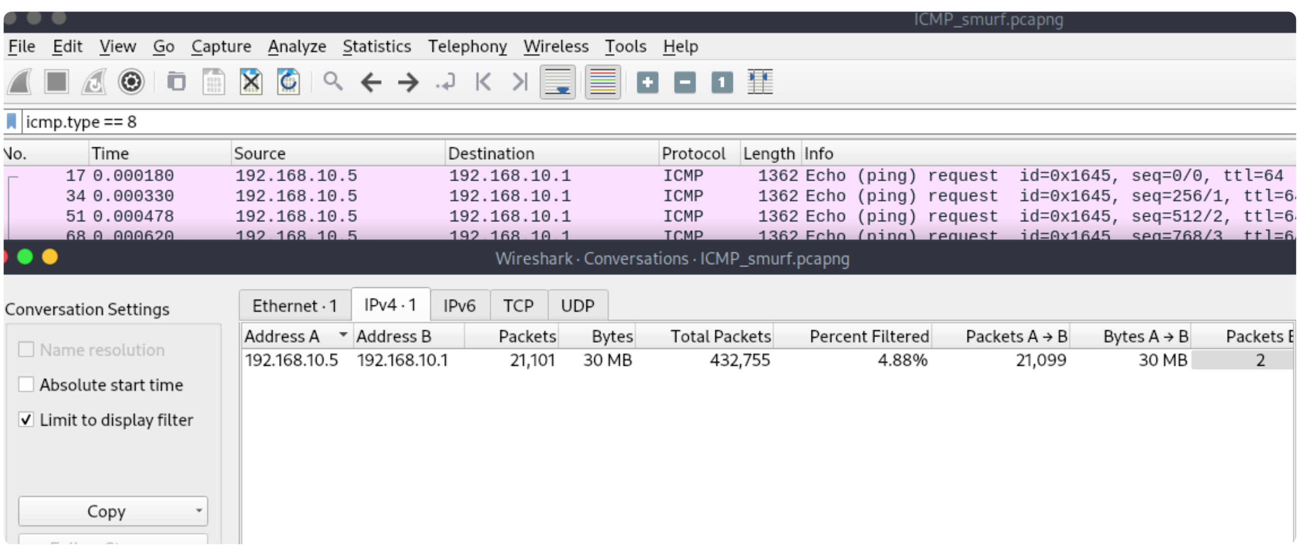
- Inspect the ICMP_smurf.pcapng file, part of this module's resources, and enter the total number of attacking hosts as your answer.
  -> We look at the ICMP protocol and examine the conversation

```
icmp
```



-> We see that there are only 2 addresses, so one of them should be the victim's host.
-> We now examine who is sending the ICMP request message, using icmp.type == 8 and examine the conversation tab

```
icmp.type == 8
```



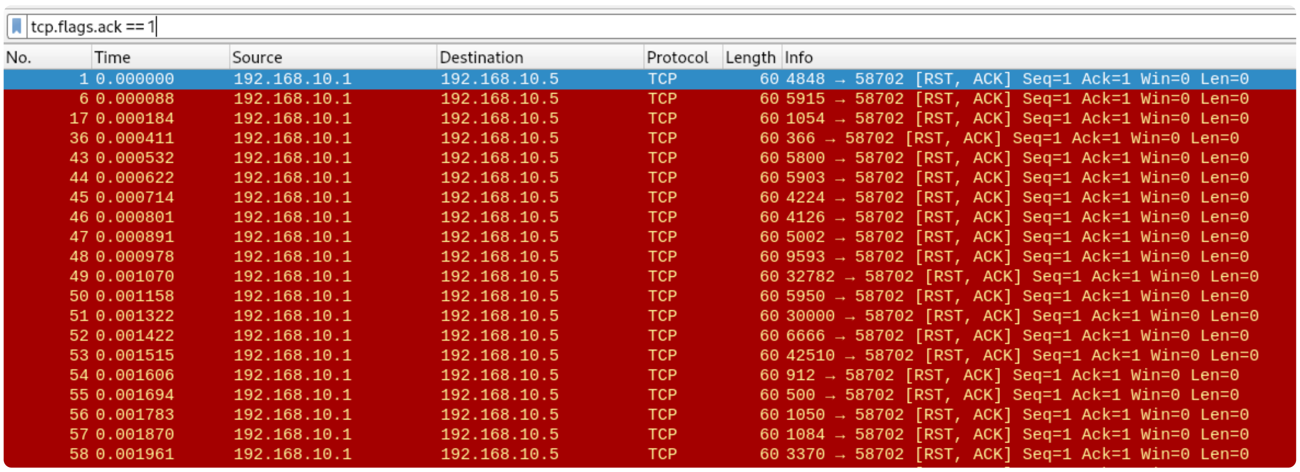
-> We see that the communications are from 192.168.10.5, which the attacker spoofed in attempted to crash 192.168.10.1, the gateway.
-> There is only 1 ip address sending the packets to 192.168.10.1, so we only have 1 attacking host.

TCP Handshake Abnormalities

Question

- Inspect the nmap_syn_scan.pcapng file, part of this module's resources, and enter the total count of packets that have the TCP ACK flag set as your answer.
  -> We first filter for packets with ack flag set as follows:

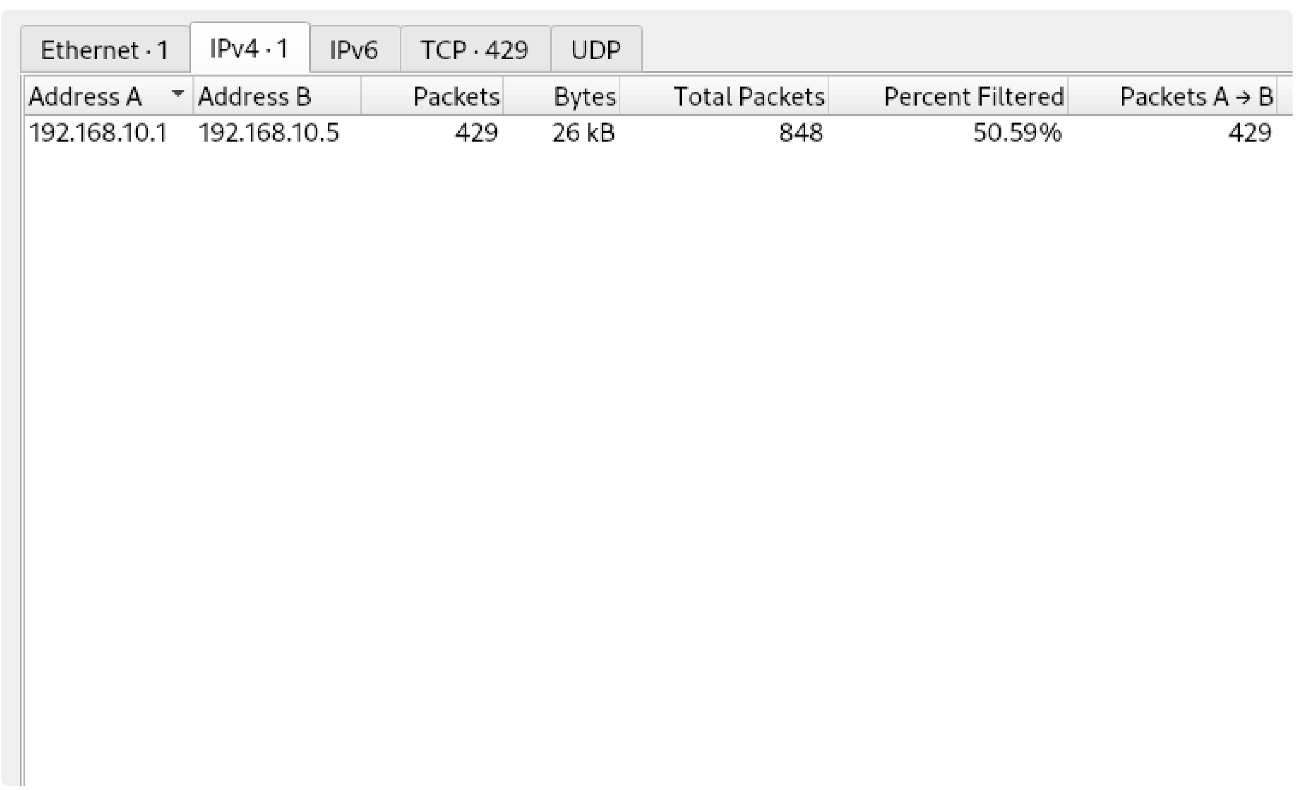
```
tcp.flags.ack == 1
```



tcp.flags.ack == 1

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.1	192.168.10.5	TCP	60	4848 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	0.000088	192.168.10.1	192.168.10.5	TCP	60	5915 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.000184	192.168.10.1	192.168.10.5	TCP	60	1054 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36	0.000411	192.168.10.1	192.168.10.5	TCP	60	366 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
43	0.000532	192.168.10.1	192.168.10.5	TCP	60	5800 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
44	0.000622	192.168.10.1	192.168.10.5	TCP	60	5903 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
45	0.000714	192.168.10.1	192.168.10.5	TCP	60	4224 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
46	0.000801	192.168.10.1	192.168.10.5	TCP	60	4126 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
47	0.000891	192.168.10.1	192.168.10.5	TCP	60	5002 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
48	0.000978	192.168.10.1	192.168.10.5	TCP	60	9593 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
49	0.001070	192.168.10.1	192.168.10.5	TCP	60	32782 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
50	0.001158	192.168.10.1	192.168.10.5	TCP	60	5950 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
51	0.001322	192.168.10.1	192.168.10.5	TCP	60	30000 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
52	0.001422	192.168.10.1	192.168.10.5	TCP	60	6666 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
53	0.001515	192.168.10.1	192.168.10.5	TCP	60	42510 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
54	0.001606	192.168.10.1	192.168.10.5	TCP	60	912 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
55	0.001694	192.168.10.1	192.168.10.5	TCP	60	500 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
56	0.001783	192.168.10.1	192.168.10.5	TCP	60	1050 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
57	0.001870	192.168.10.1	192.168.10.5	TCP	60	1084 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
58	0.001961	192.168.10.1	192.168.10.5	TCP	60	3370 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

-> Then we look at statistics -> Conversations to look at the number of packet wit ACK flag set.

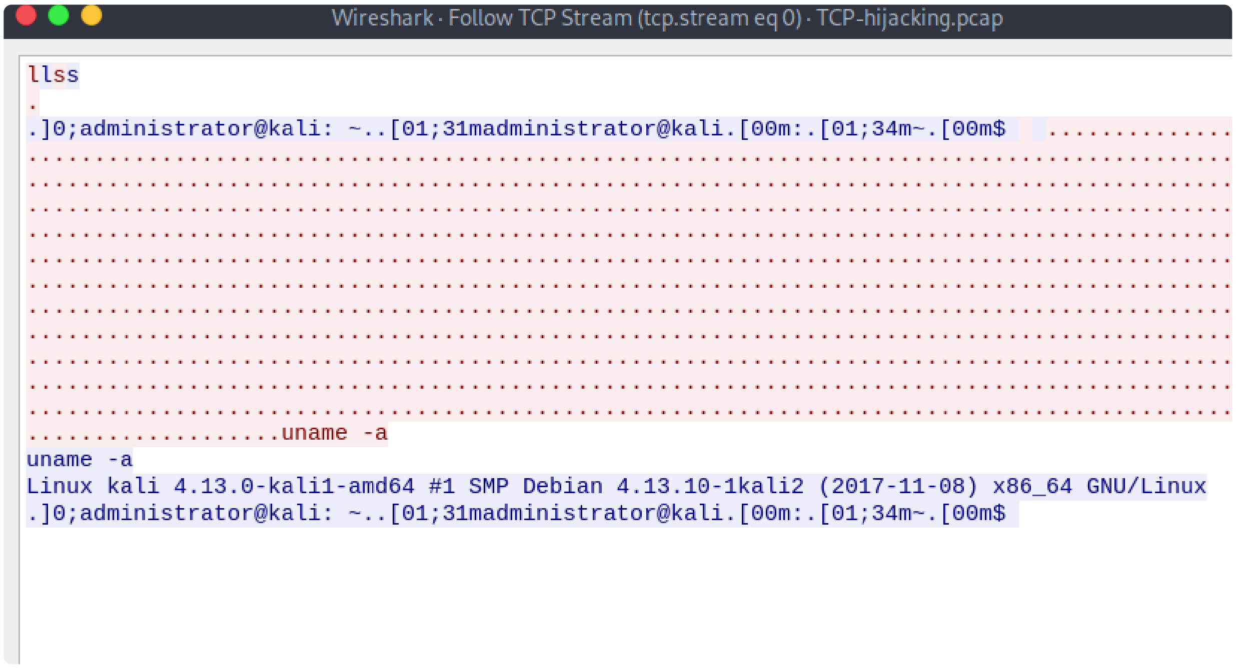


Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered	Packets A → B
192.168.10.1	192.168.10.5	429	26 kB	848	50.59%	429

-> We see that there are 429 packets with ACK set.

TCP Connection Resets & Hijacking

Question

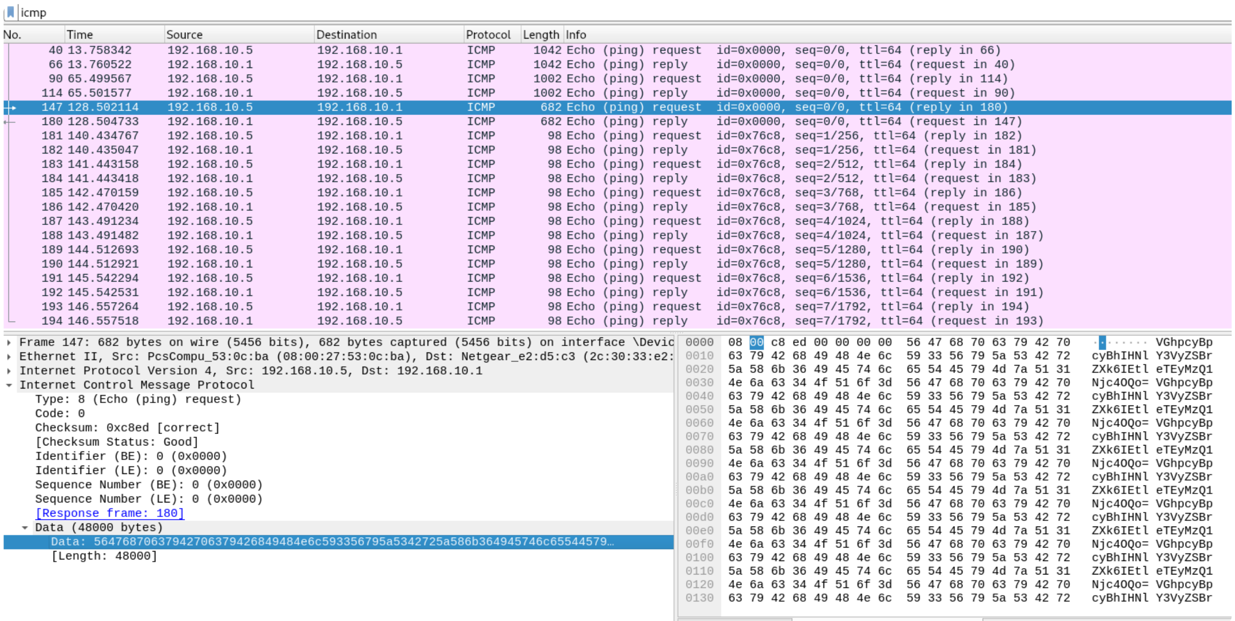
- Inspect the TCP-hijacking.pcap file, part of this module's resources, and enter the username that has been used through the telnet protocol as your answer.
  -> We follow the stream for the telnet protocol:
  

  -> We see that it is the administrotr user that has been used through the telnet protocol.

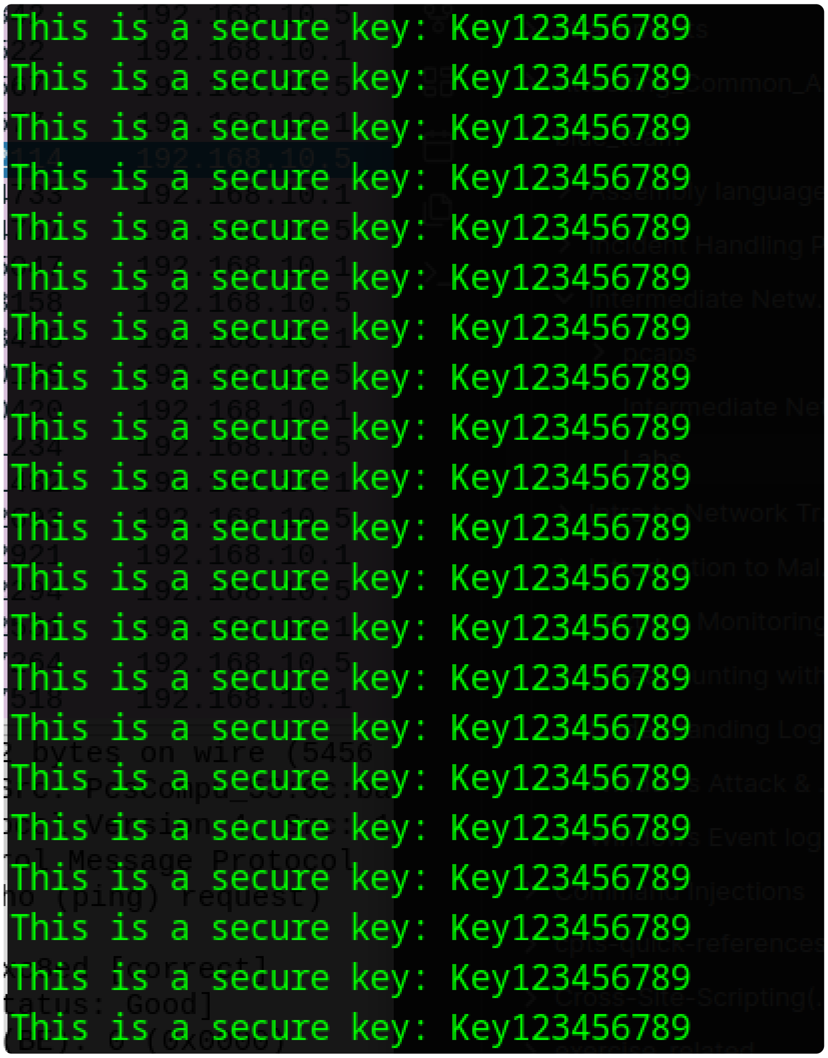
ICMP Tunneling

Question

- Enter the decoded value of the base64-encoded string that was mentioned in this section as your answer.
  -> We first find the packets that's encoded



-> Then we copy Data as printable text, paste it to linux and decode it:



-> And so the answer is Key123456789

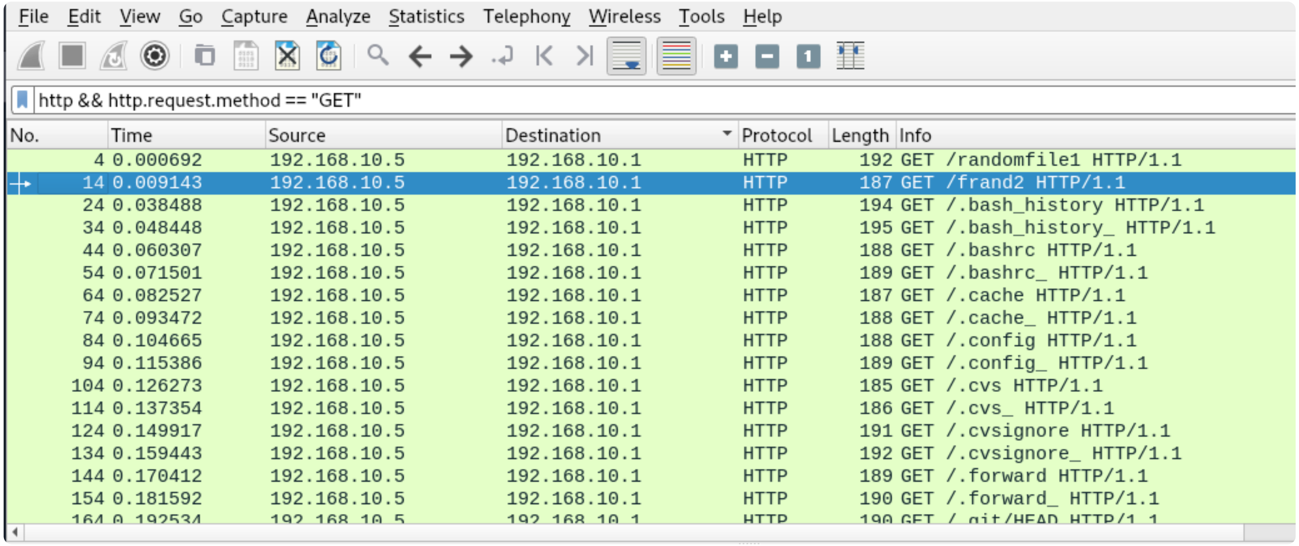
Application Layer Attacks

HTTP/HTTPs Service Enumeration

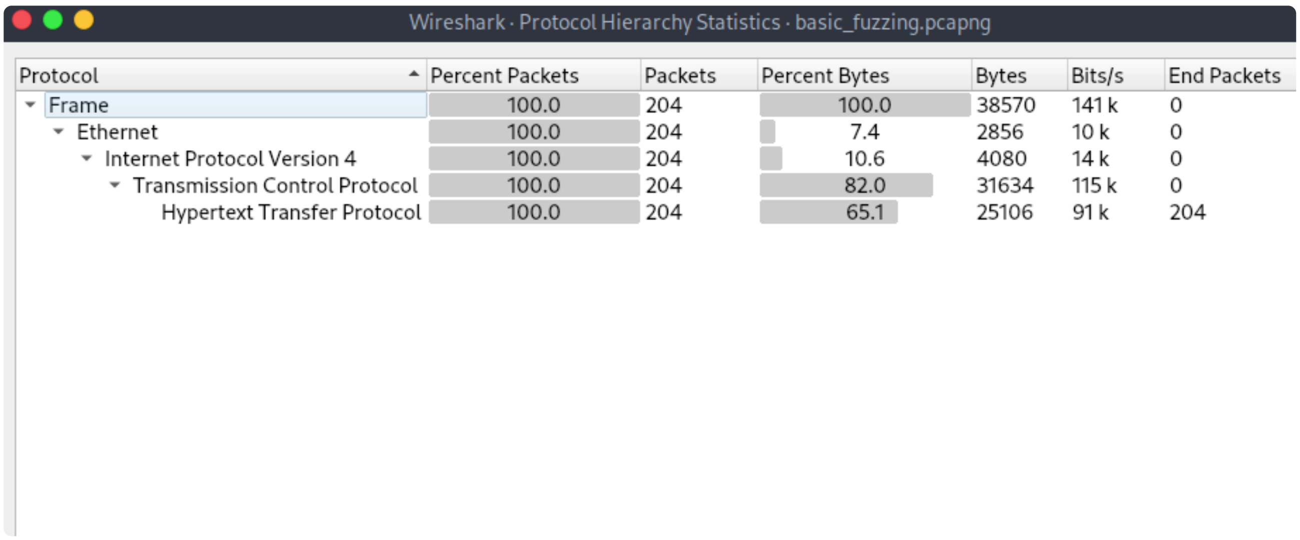
Question

- Inspect the basic_fuzzing.pcapng file, part of this module's resources, and enter the total number of HTTP packets that are related to GET requests against port 80 as your answer.
  -> We filter for http get request method

```
http && http.request.method == "GET"
```



-> We then examine protocol hierarchy



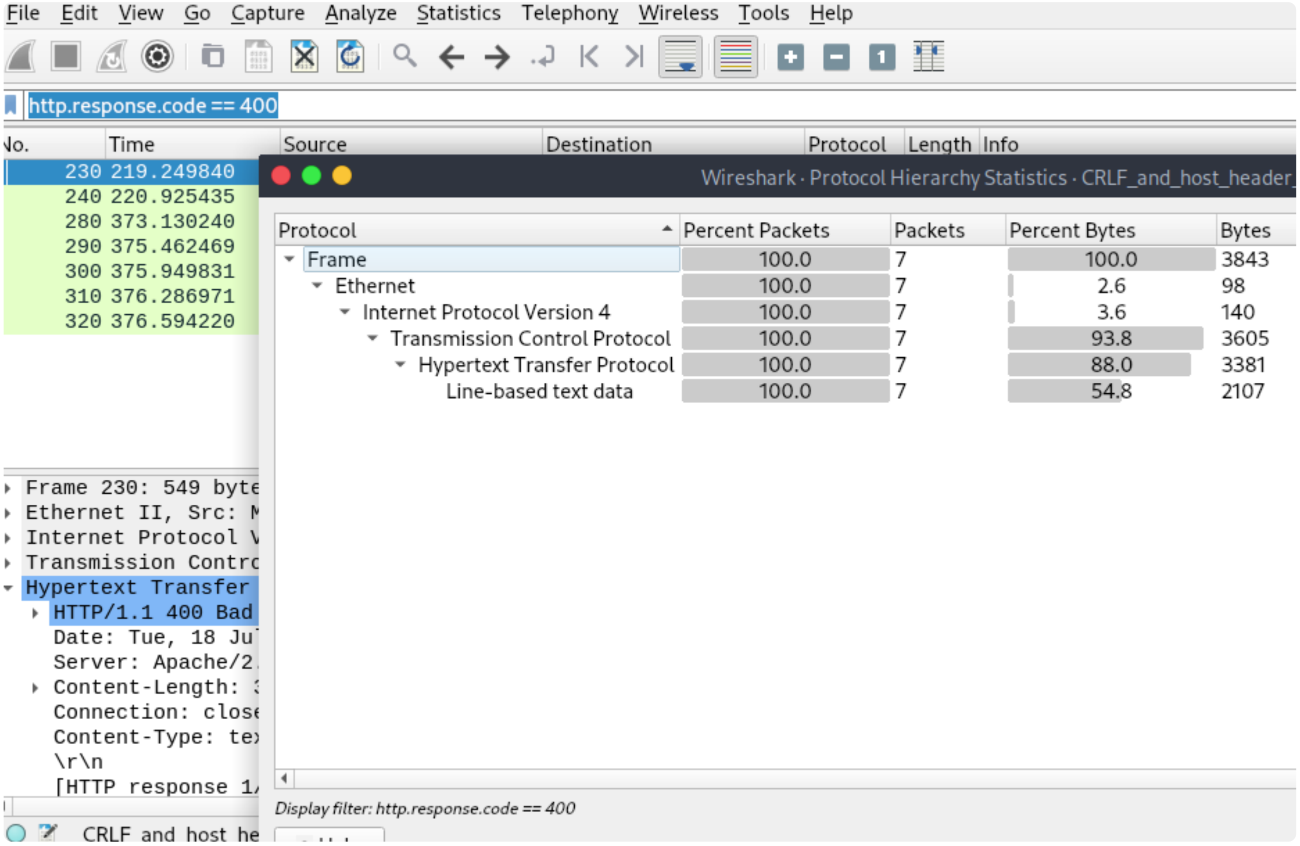
-> So there are 204 packets

Strange HTTP Headers

Question

- Inspect the CRLF_and_host_header_manipulation.pcapng file, part of this module's resources, and enter the total number of HTTP packets with response code 400 as your answer.
  -> We apply the filter on response code 400 and view protocol hierachy

```
http.response.code == 400
```

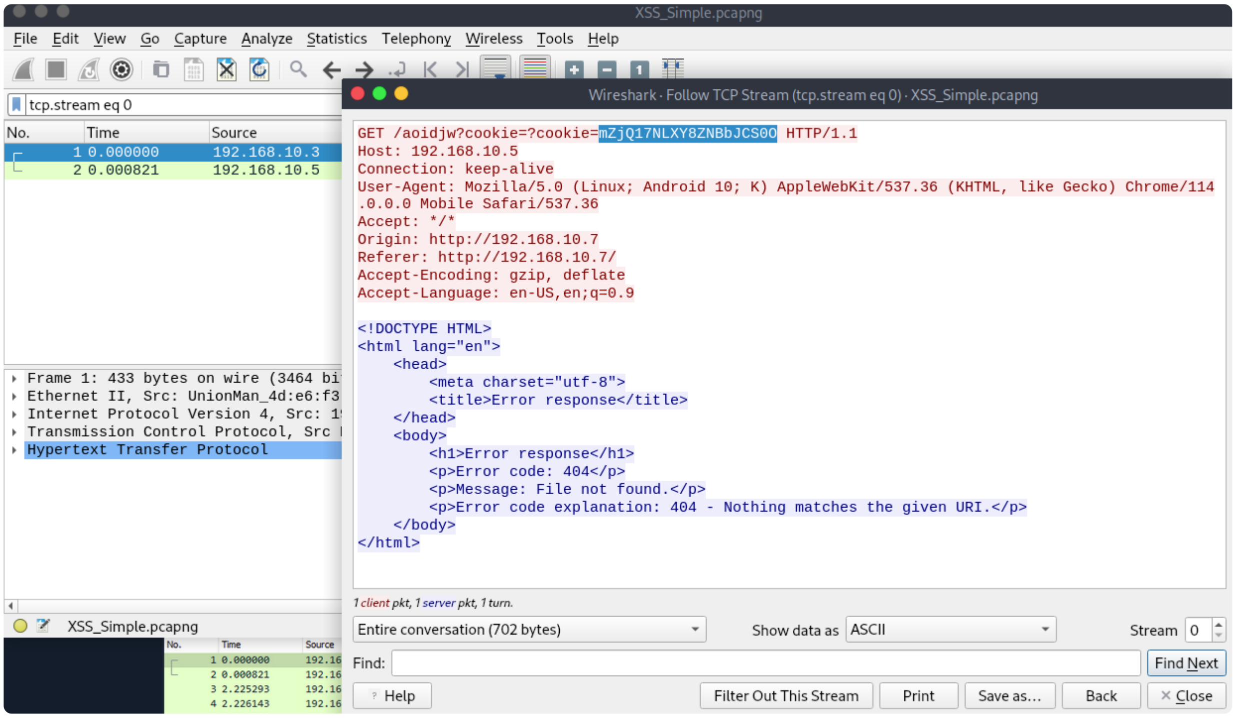


-> So there is a total of 7 packets.

Cross-Site Scripting (XSS) & Code Injection Detection

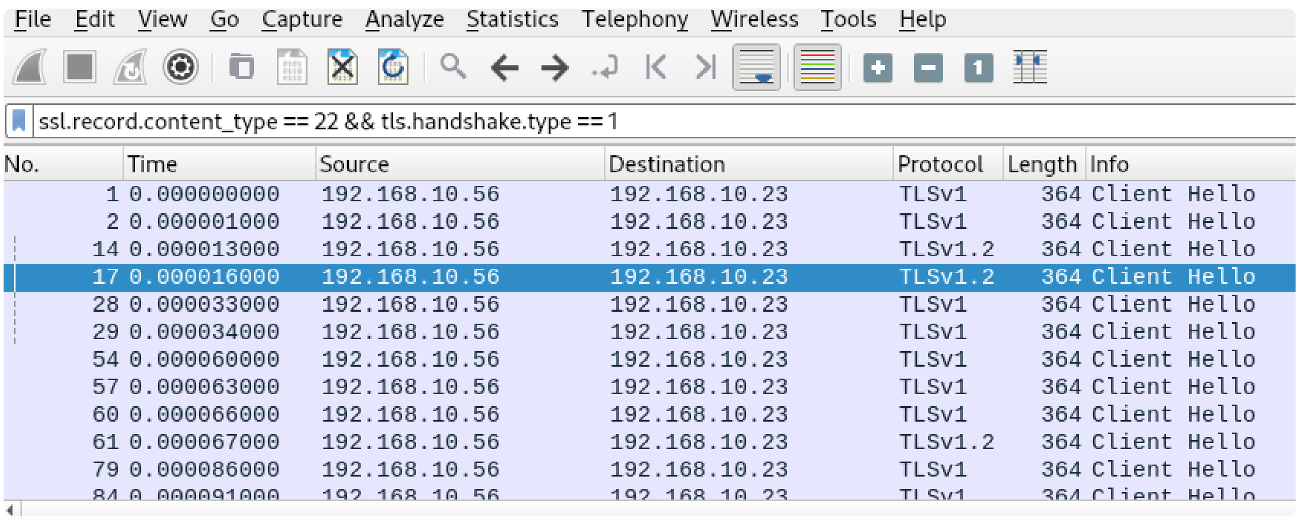
Question

- Inspect the first packet of the XSS_Simple.pcapng file, part of this module's resources, and enter the cookie value that was exfiltrated as your answer.
  -> We followed the stream for the first packet

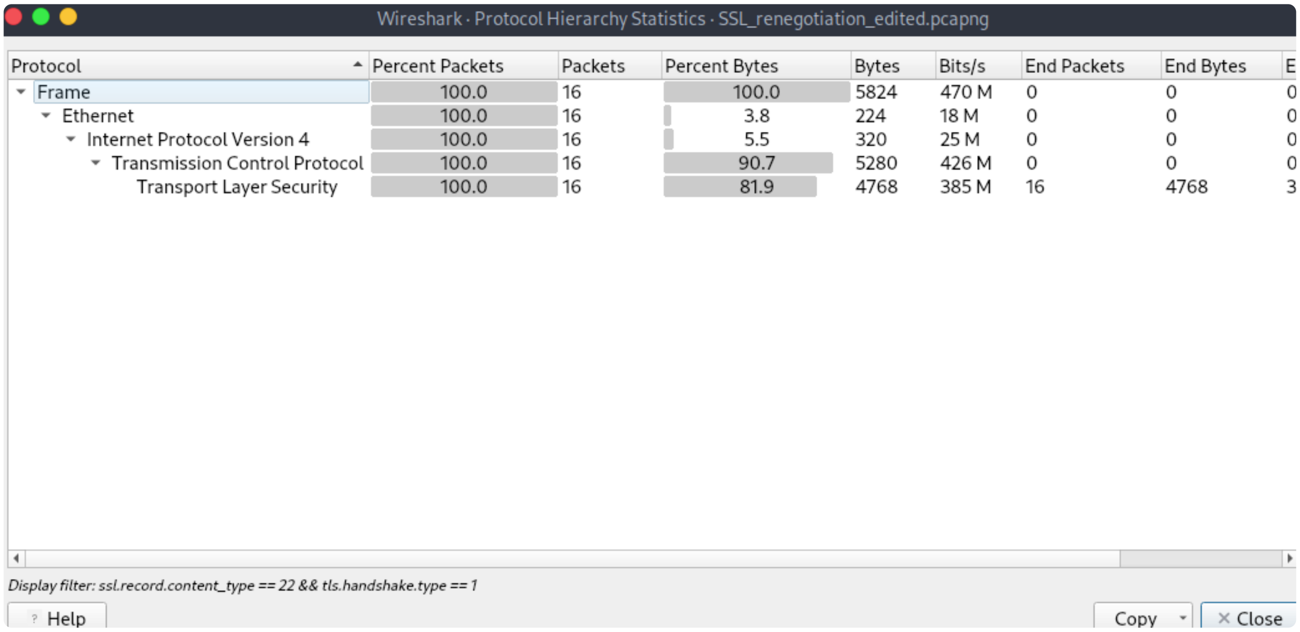


-> Hence, we obtain the cookie required

```
ssl.record.content_type == 22 && tls.handshake.type == 1
```



-> We then look into protocol hierachy and see the number of packets



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	E
Frame	100.0	16	100.0	5824	470 M	0	0	0
Ethernet	100.0	16	3.8	224	18 M	0	0	0
Internet Protocol Version 4	100.0	16	5.5	320	25 M	0	0	0
Transmission Control Protocol	100.0	16	90.7	5280	426 M	0	0	0
Transport Layer Security	100.0	16	81.9	4768	385 M	16	4768	3

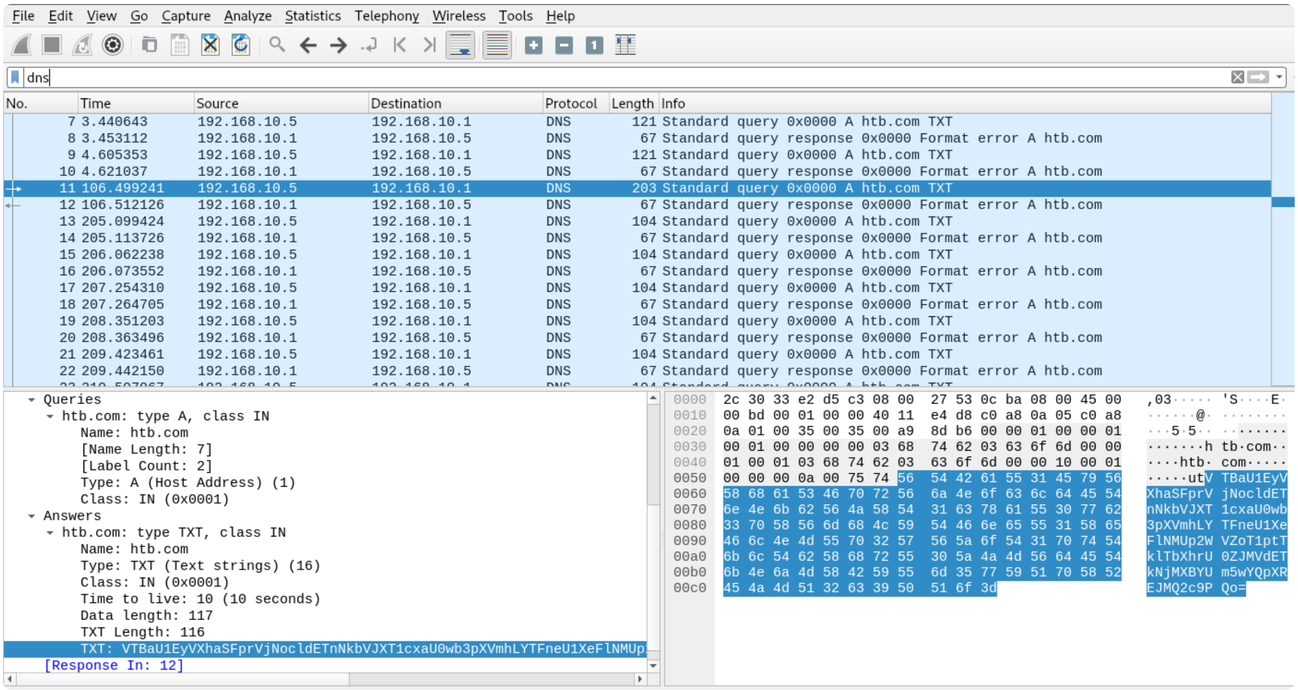
-> So we have 16 packets in total.

Peculiar DNS Traffic

Question

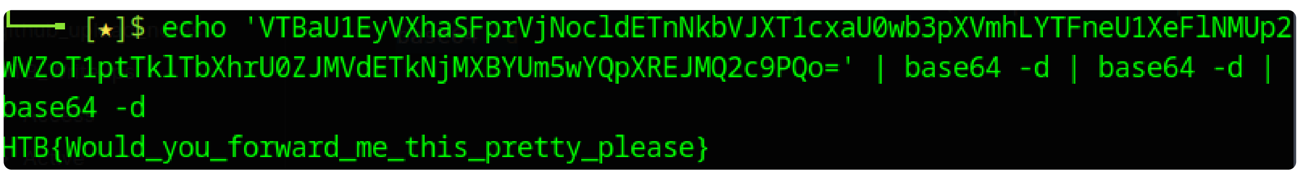
- Enter the decoded value of the triple base64-encoded string that was mentioned in this section as your answer. Answer format: HTB{___}
  -> We first found the value of triple-encoded base64 string with capture filter

```
dns
```



-> Then we triple decode as follows:

```
echo
'VTBaU1EyVXhaSFprVjNocldETnNkbVJXT1cxaU0wb3pXVmhLYTFneU1XeFlNMUp2WVZoT1p
tTklTbXhrU0ZJMVdETkNjMXBYUm5wYQpXREJMQ2c9PQo=' | base64 -d | base64 -d |
base64 -d
```



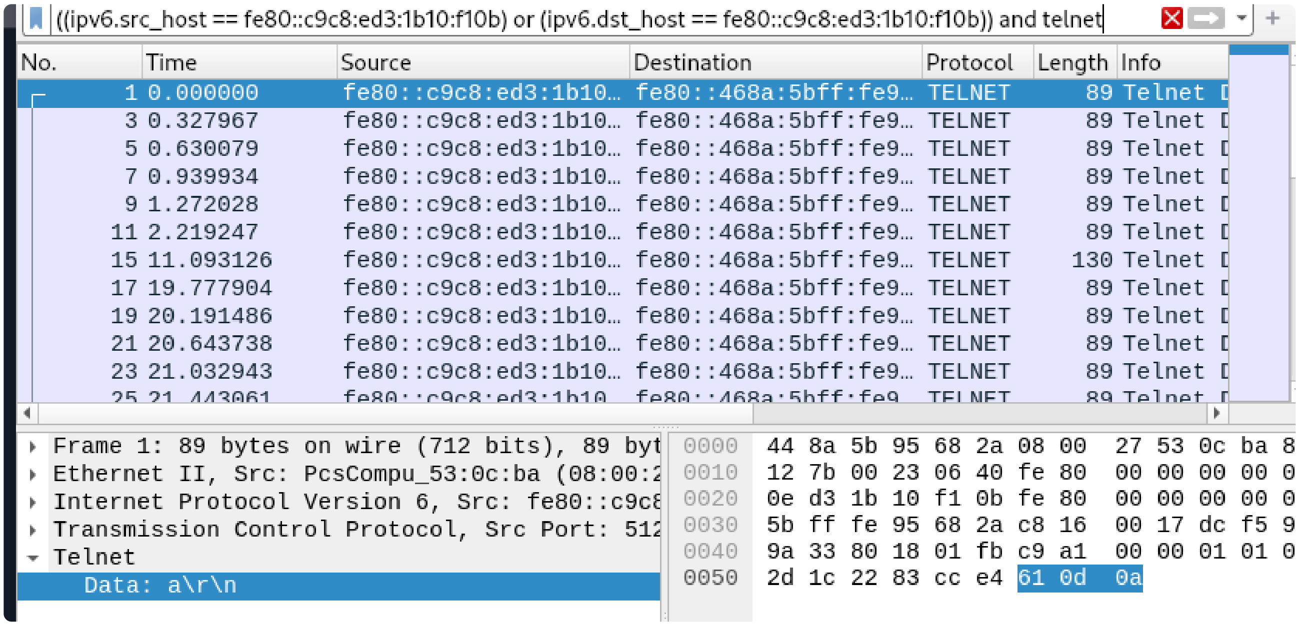
Strange Telnet & UDP Connections

Question

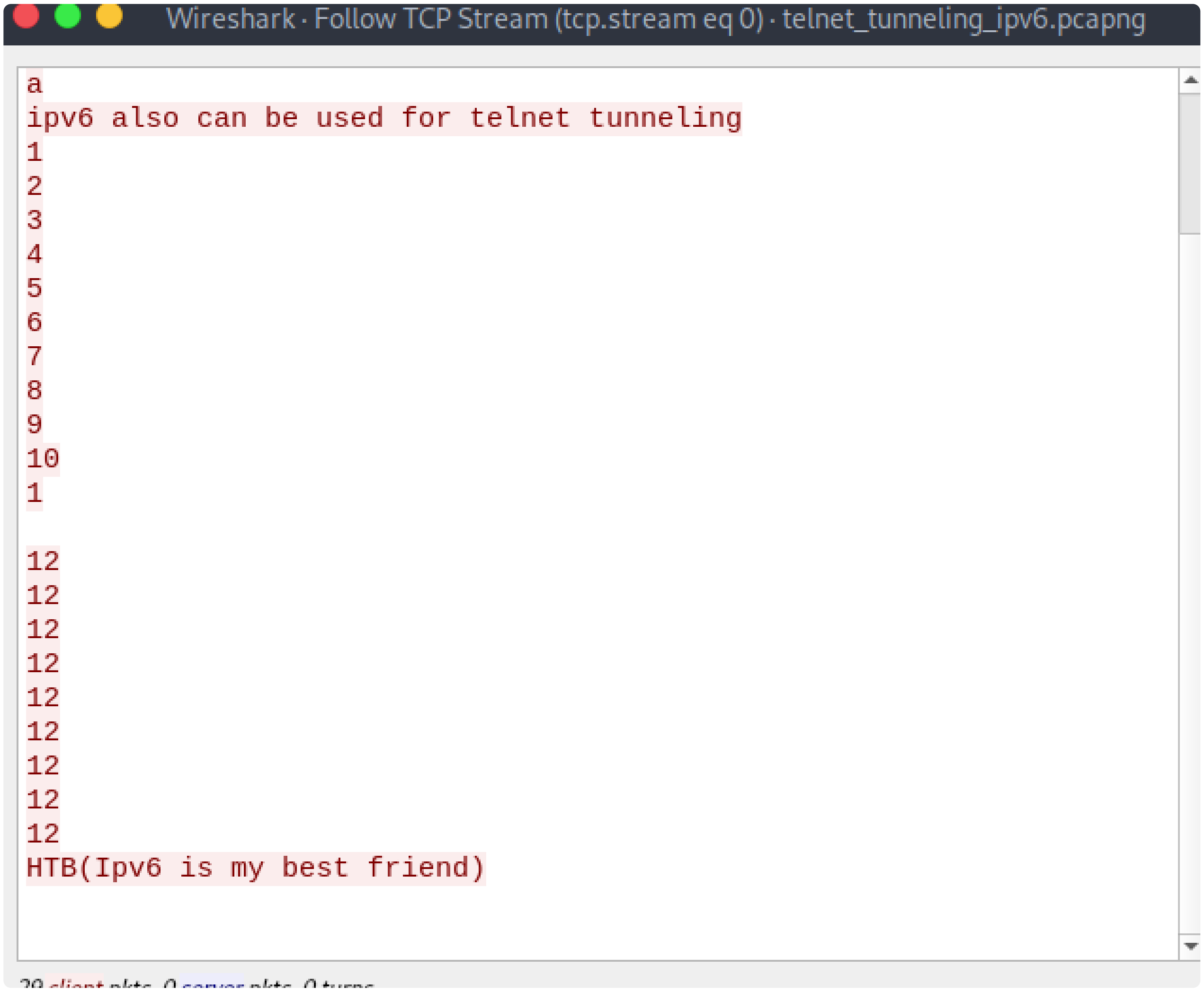
- Inspect the telnet_tunneling_ipv6.pcapng file, part of this module's resources, and enter the hidden flag as your answer. Answer format: HTB(___) (Replace all spaces with underscores)

-> We first examine the file and filter for source and destination ip (as that is the host and starts the communication for telnet)

```
((ipv6.src_host == fe80::c9c8:ed3:1b10:f10b) or (ipv6.dst_host ==
fe80::c9c8:ed3:1b10:f10b)) and telnet
```



-> We then follow the tcp stream to examine the content for communication and we found the flag required



Skills Assessment

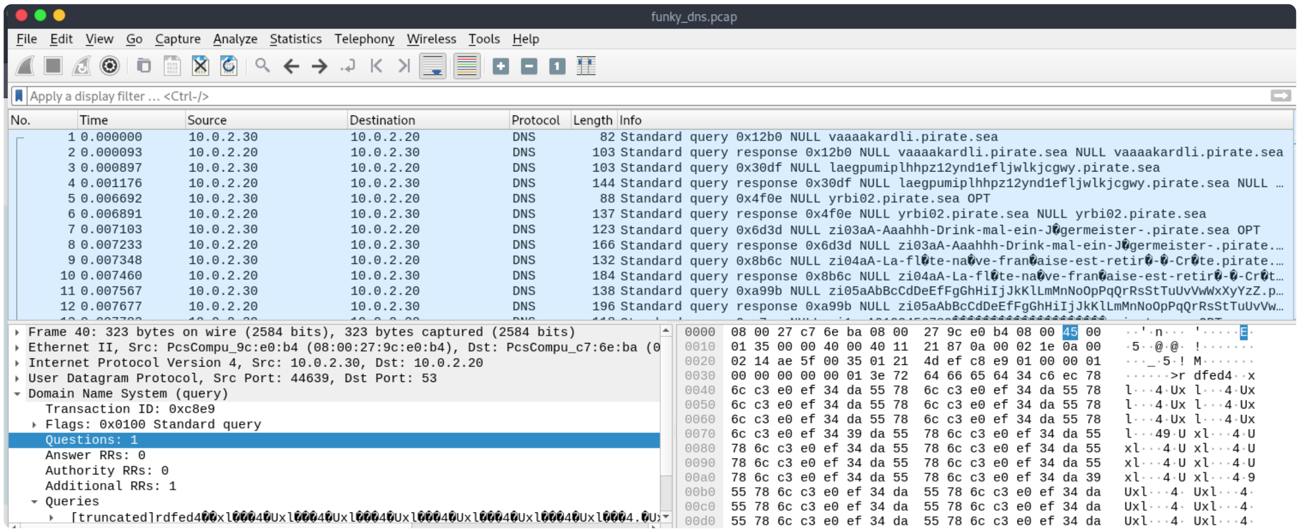
Scenario

- As a Security Operations Center (SOC) analyst, you were recently provided with two PCAP (Packet Capture) files named `funky_dns.pcap` and `funky_icmp.pcap`.
- Inspect the `funky_dns.pcap` and `funky_icmp.pcap` files, part of this module's resources, to identify if there are certain patterns and behaviors within these captures that deviate from what is typically observed in routine network traffic.
- Then, answer the questions below.

Question

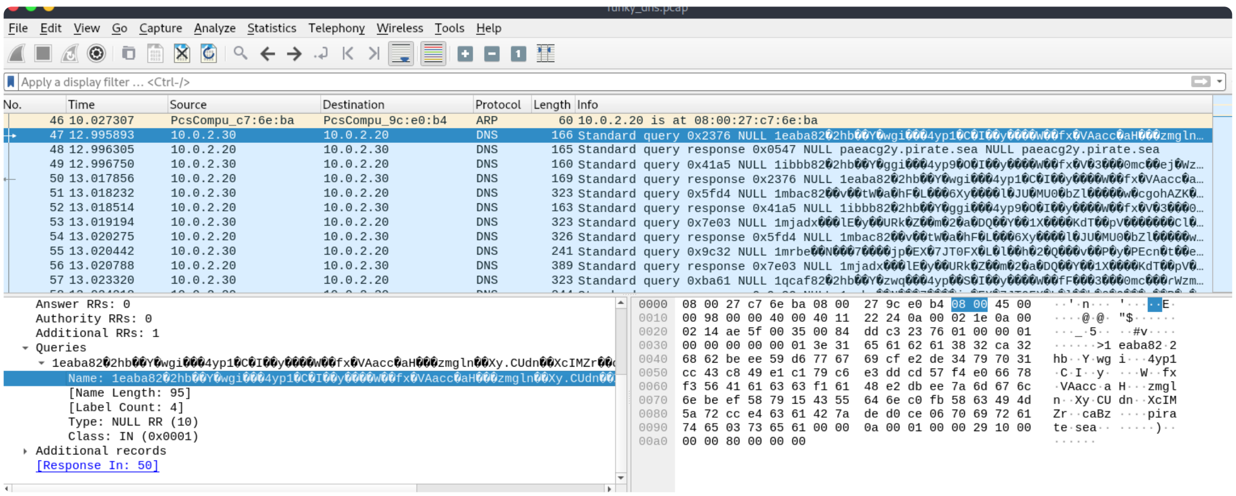
-> Inspect the funky_dns.pcap file, part of this module's resources, and enter the related attack as your answer.

Answer format: "DNS Flooding", "DNS Amplification", "DNS Tunneling"



-> We see that lots of dns query being made from 10.0.2.30 to 10.0.2.20

- Looking at the some of the request being made, we see the following:

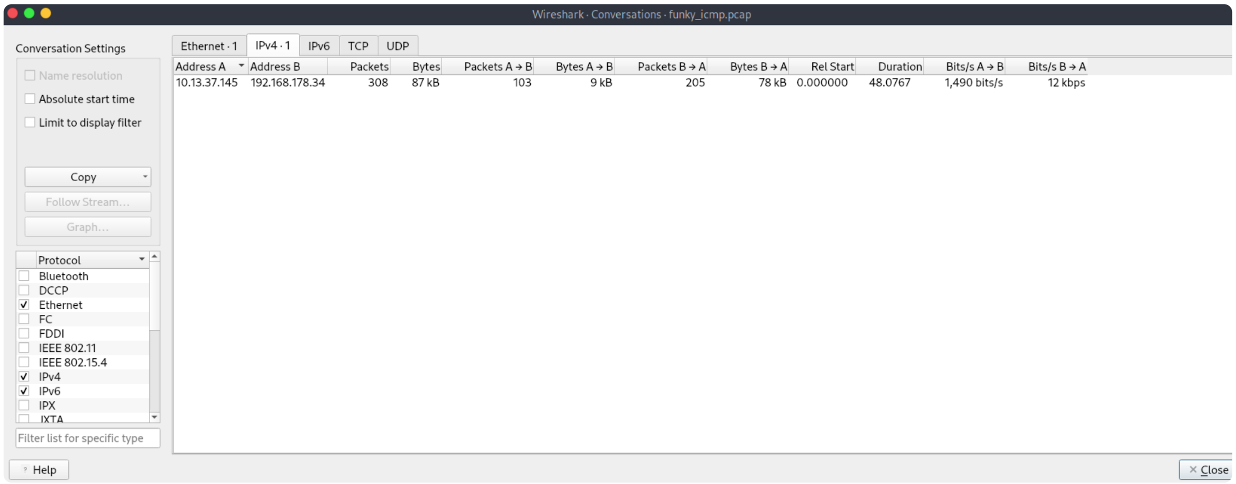


-> We see that alot of encryption messages being send from the host and the responder. This is likely an indication of DNS tunneling with some encryption.

- Inspect the funky_icmp.pcap file, part of this module's resources, and enter the related attack as your answer. Answer format: "ICMP Flooding", "ICMP Tunneling", "ICMP SMURF Attack"

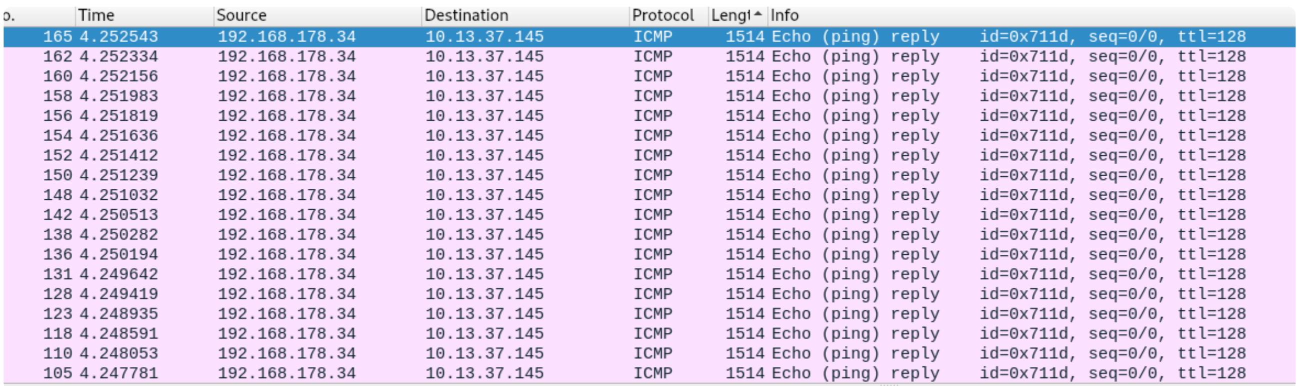
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.13.37.145	192.168.178.34	ICMP	42	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 2)
2	0.001140	192.168.178.34	10.13.37.145	ICMP	60	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 1)
3	1.002454	10.13.37.145	192.168.178.34	ICMP	42	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 4)
4	1.003488	192.168.178.34	10.13.37.145	ICMP	60	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 3)
5	2.005242	10.13.37.145	192.168.178.34	ICMP	42	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 6)
6	2.005768	192.168.178.34	10.13.37.145	ICMP	60	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 5)
7	3.007954	10.13.37.145	192.168.178.34	ICMP	42	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 8)
8	3.009669	192.168.178.34	10.13.37.145	ICMP	60	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 7)
9	4.011939	10.13.37.145	192.168.178.34	ICMP	42	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 10)
10	4.013297	192.168.178.34	10.13.37.145	ICMP	60	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 9)
11	4.238309	10.13.37.145	192.168.178.34	ICMP	102	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 12)
12	4.238874	192.168.178.34	10.13.37.145	ICMP	102	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 11)
13	4.239034	192.168.178.34	10.13.37.145	ICMP	102	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
14	4.239119	10.13.37.145	192.168.178.34	ICMP	94	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 16)
15	4.239212	10.13.37.145	192.168.178.34	ICMP	221	Echo (ping) request id=0x711d, seq=0/0, ttl=64 (reply in 17)
16	4.239411	192.168.178.34	10.13.37.145	ICMP	94	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 14)
17	4.239428	192.168.178.34	10.13.37.145	ICMP	221	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 15)
18	4.239696	192.168.178.34	10.13.37.145	ICMP	94	Echo (ping) reply id=0x711d, seq=0/0, ttl=128

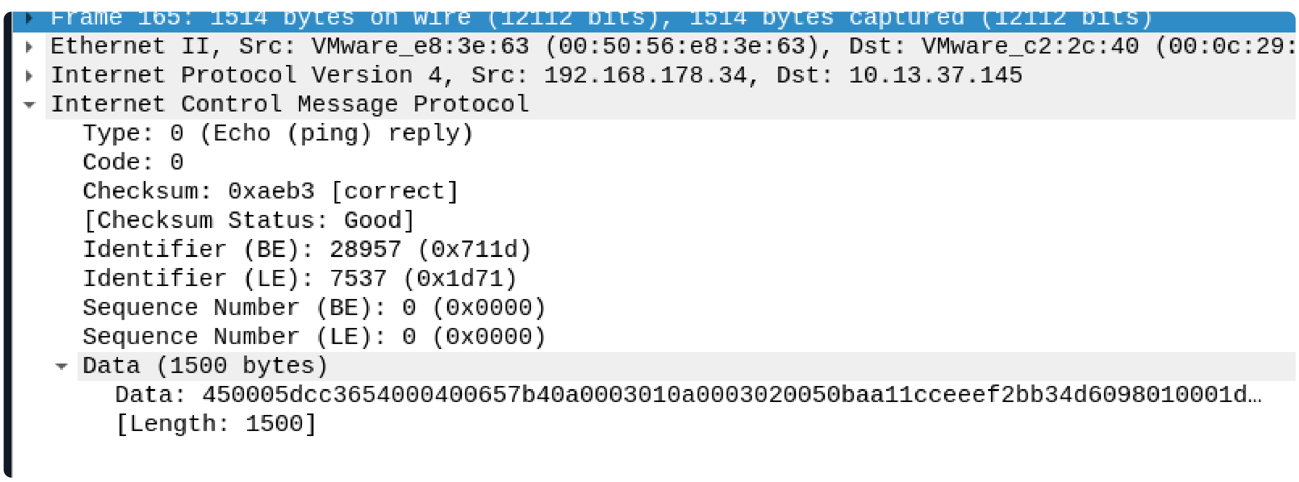
-> We see that the source address 10.13.37.145 is sending alot of icmp messages to destination 192.168.178.34.



-> We can confirm this by looking at protocol hierachy.

-> Notice that there are some ICMP packets with large data size:





-> Hence, we can say that it is ICMP tunneling.

Extra:

-> It can't really be ICMP flooding or ICMP smurfing because of the amount of packets is send to too little. According to some website, they usually have thousands and millions of incoming packets per seconds.

-> ICMP flooding is about blocking incoming bandwidth of the target address while ICMP smurfing is about spoofing a source IP address and causing DDOS on that source IP address spoofed.