

Labs - Intermediate Network Analysis

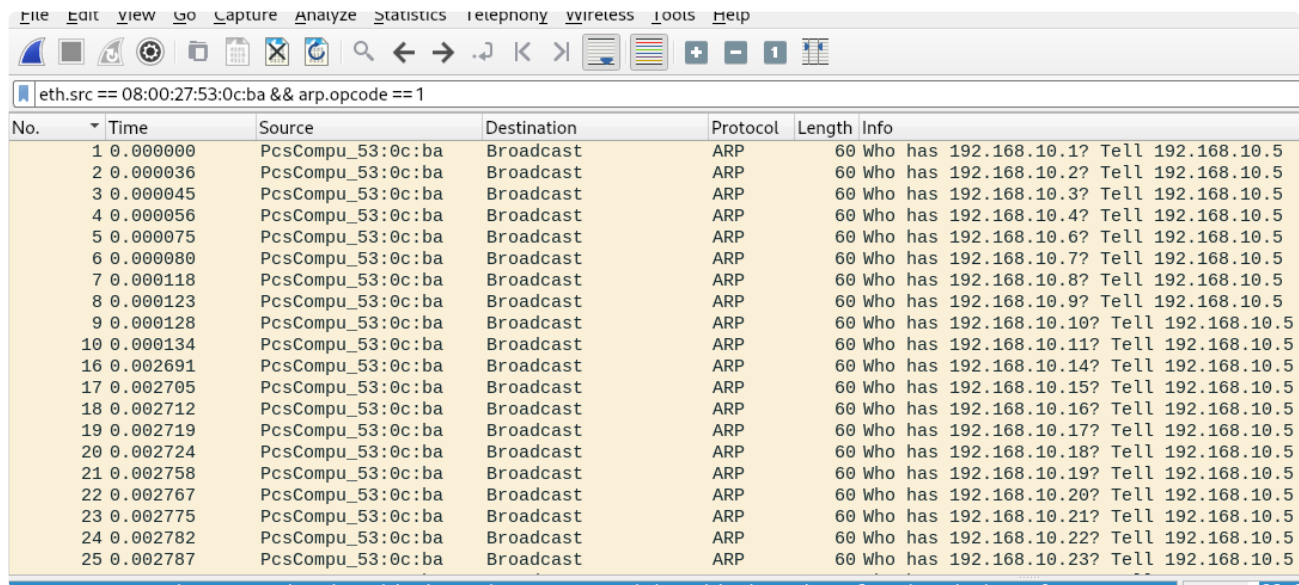
Link Layer Attack

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
```



The image shows a Wireshark packet capture window with the display filter 'eth.src == 08:00:27:53:0c:ba && arp.opcode == 1'. The packet list shows 25 ARP requests from source 08:00:27:53:0c:ba to various destinations. The packet details pane shows the selected packet (No. 10) with the following structure:

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 hierarchy 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.

Wireshark - ARP_Poison.pcapng

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

Wireshark - Protocol Hierarchy Statistics - ARP_Poison.pcapng

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	507	100.0	30420	115 k	0	0	0	507
Ethernet	100.0	507	53.3	16224	61 k	0	0	0	507
Address Resolution Protocol	100.0	507	76.7	23322	88 k	507	23322	88 k	507

-> Hence, there are 507 packets and we can see 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)
```

Wireshark - ARP_Poison.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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

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
13	0.000406	Netgear_e2:d5:c3	PcsCompu_53:0c:ba	ARP	60	192.168.10.1 is at 2c:30:33:e2:d5:c3
520	2.491827	PcsCompu_53:0c:ba	TuyaSmar_37:b9:4f	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
521	2.491854	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	Gratuitous ARP for 192.168.10.1 (Reply) (du
522	2.491859	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	Gratuitous ARP for 192.168.10.1 (Reply) (du
523	2.491863	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.6 is at 08:00:27:53:0c:ba
524	2.499813	PcsCompu_53:0c:ba	UnionMan_4d:e6:f3	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
525	2.499843	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.3 is at 08:00:27:53:0c:ba
526	2.555962	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.4 is at 08:00:27:53:0c:ba
527	2.559771	PcsCompu_53:0c:ba	ASUSTekC_8a:a6:a8	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
528	2.559795	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.8 is at 08:00:27:53:0c:ba
529	2.572048	PcsCompu_53:0c:ba	Micro-St_95:68:2a	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba

-> 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)
```

No.	Time	Source	Destination	Protocol	Length	Info
416	78.561456	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=0, FN=0, Flags=.....
417	78.565783	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=1, FN=0, Flags=.....
418	78.565801	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=0, FN=0, Flags=.....
420	78.566384	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=1, FN=0, Flags=.....
421	78.570171	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=2, FN=0, Flags=.....
422	78.572747	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=3, FN=0, Flags=.....
423	78.572834	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=2, FN=0, Flags=.....
425	78.574455	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=3, FN=0, Flags=.....
426	78.581599	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=4, FN=0, Flags=.....
427	78.583939	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=5, FN=0, Flags=.....
428	78.584316	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=4, FN=0, Flags=.....
430	78.586261	d2:4e:7e:05:43:3c	UnionMan_4d:e6:f1	802.11	26	Deauthentication, SN=5, FN=0, Flags=.....
431	78.589988	UnionMan_4d:e6:f1	d2:4e:7e:05:43:3c	802.11	26	Deauthentication, SN=6, FN=0, Flags=.....

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

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	14592	100.0	379392	37 k	0	0	0
IEEE 802.11 wireless LAN	100.0	14592	92.3	350208	34 k	14592	350208	34 k

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

-> 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 frame
```

```
▶ 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 it is likely the rogue access point.

-> We now filter for this access point (mac address) along with the arp requests, as having an arp request to the network indicates that the host likely falls 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)
```

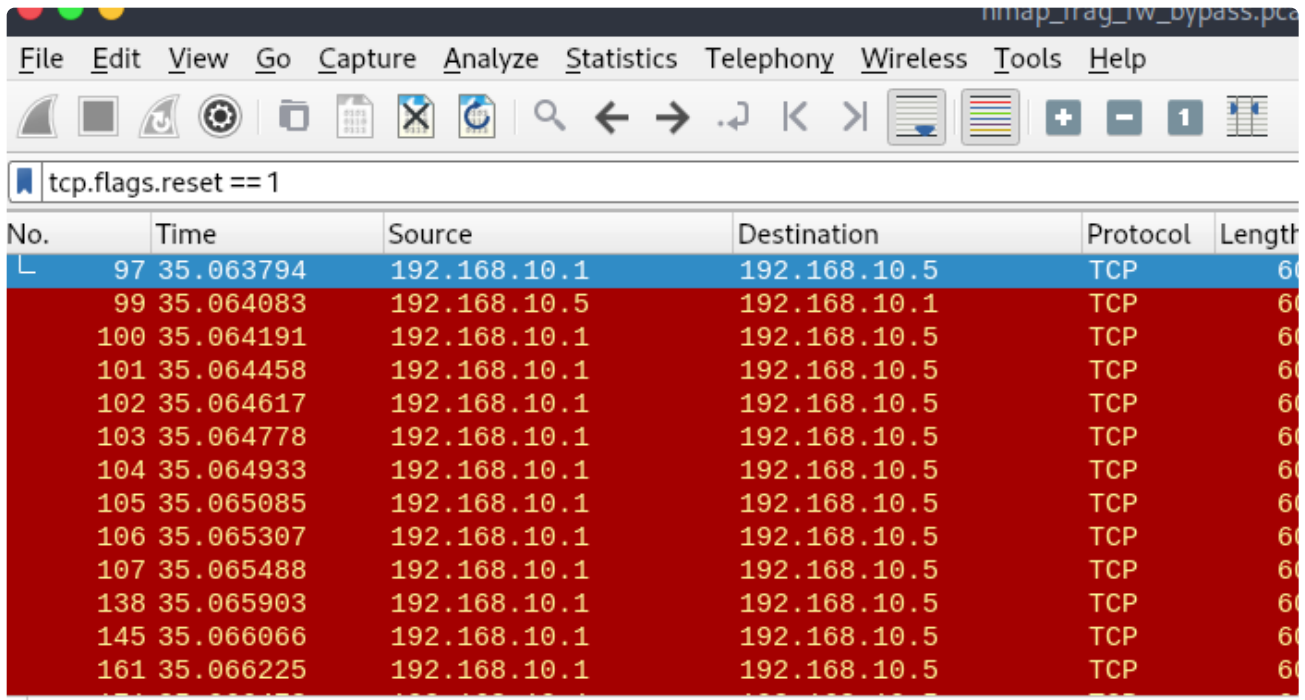
-> it has an MAC of 2c:6d:c1:af:eb:91

Detecting Network Abnormalities

Fragmentation Attacks

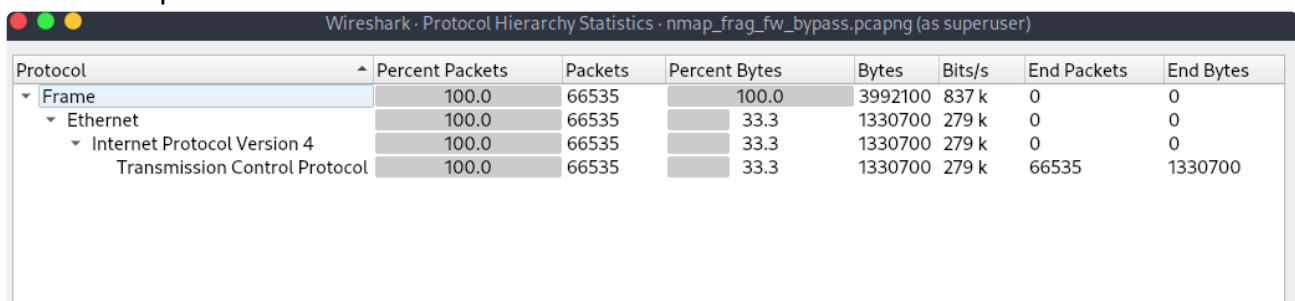
- Inspect the nmap_frag_fw_bypass.pcapng file, part of this module's resources, and enter the total count of packets that have the TCP RST flag set as your answer.
-> We did some searching up online on how to filter for rest packets, and the answer is to apply the filter tcp.flags.reset ==1, which is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

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

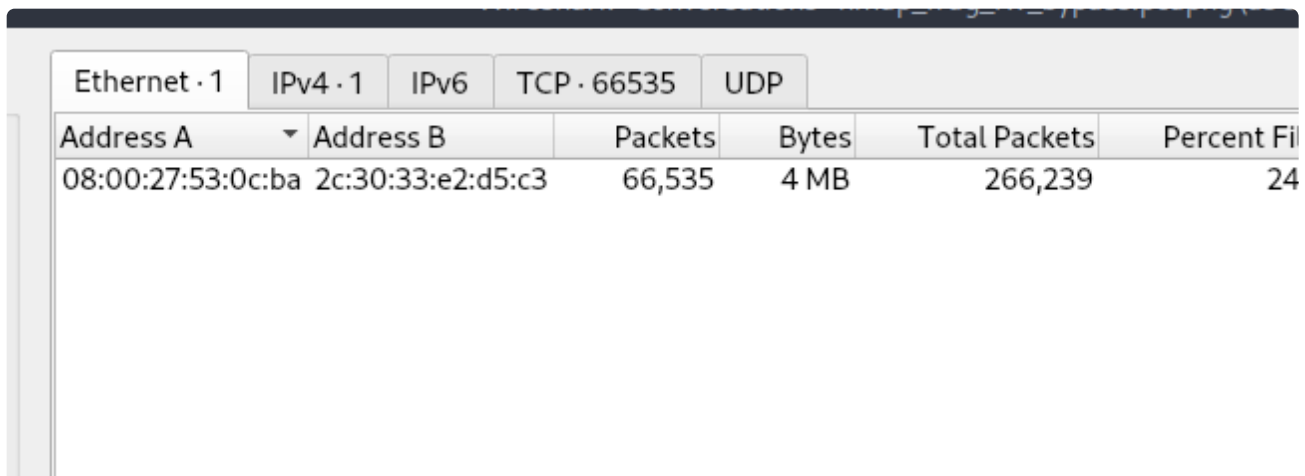


No.	Time	Source	Destination	Protocol	Length
97	35.063794	192.168.10.1	192.168.10.5	TCP	60
99	35.064083	192.168.10.5	192.168.10.1	TCP	60
100	35.064191	192.168.10.1	192.168.10.5	TCP	60
101	35.064458	192.168.10.1	192.168.10.5	TCP	60
102	35.064617	192.168.10.1	192.168.10.5	TCP	60
103	35.064778	192.168.10.1	192.168.10.5	TCP	60
104	35.064933	192.168.10.1	192.168.10.5	TCP	60
105	35.065085	192.168.10.1	192.168.10.5	TCP	60
106	35.065307	192.168.10.1	192.168.10.5	TCP	60
107	35.065488	192.168.10.1	192.168.10.5	TCP	60
138	35.065903	192.168.10.1	192.168.10.5	TCP	60
145	35.066066	192.168.10.1	192.168.10.5	TCP	60
161	35.066225	192.168.10.1	192.168.10.5	TCP	60

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



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	66535	100.0	3992100	837 k	0	0
Ethernet	100.0	66535	33.3	1330700	279 k	0	0
Internet Protocol Version 4	100.0	66535	33.3	1330700	279 k	0	0
Transmission Control Protocol	100.0	66535	33.3	1330700	279 k	66535	1330700



Ethernet · 1		IPv4 · 1	IPv6	TCP · 66535	UDP
Address A	Address B	Packets	Bytes	Total Packets	Percent Filtered
08:00:27:53:0c:ba	2c:30:33:e2:d5:c3	66,535	4 MB	266,239	24

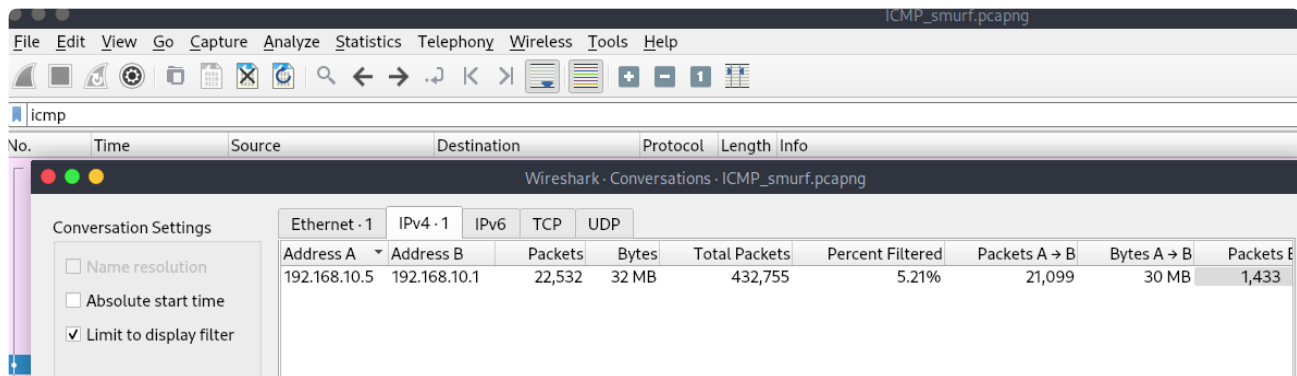
-> Hence, we would get 66,535 packets are send, indicative of a full port scan!

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

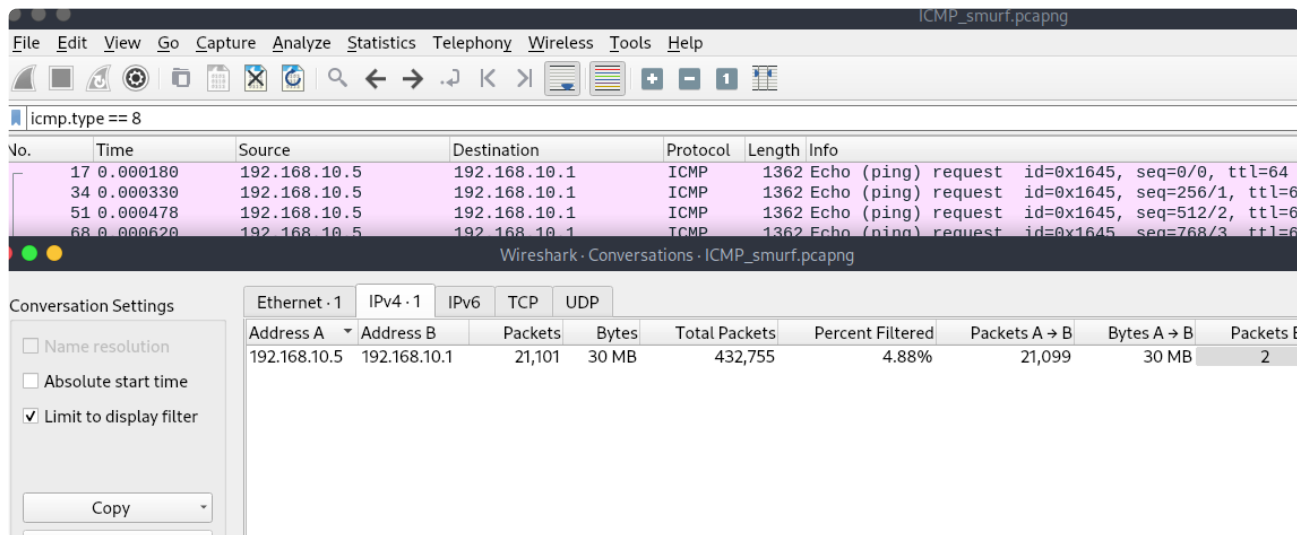
icmp



-> 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.

Ethernet · 1		IPv4 · 1	IPv6	TCP · 429	UDP		
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:

[illegible]

-> We see that it is the administrottr 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

[illegible]

-> 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"
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
http && http.request.method == "GET"						
No.	Time	Source	Destination	Protocol	Length	Info
4	0.000692	192.168.10.5	192.168.10.1	HTTP	192	GET /randomfile1 HTTP/1.1
14	0.009143	192.168.10.5	192.168.10.1	HTTP	187	GET /frand2 HTTP/1.1
24	0.038488	192.168.10.5	192.168.10.1	HTTP	194	GET /.bash_history HTTP/1.1
34	0.048448	192.168.10.5	192.168.10.1	HTTP	195	GET /.bash_history_ HTTP/1.1
44	0.060307	192.168.10.5	192.168.10.1	HTTP	188	GET /.bashrc HTTP/1.1
54	0.071501	192.168.10.5	192.168.10.1	HTTP	189	GET /.bashrc_ HTTP/1.1
64	0.082527	192.168.10.5	192.168.10.1	HTTP	187	GET /.cache HTTP/1.1
74	0.093472	192.168.10.5	192.168.10.1	HTTP	188	GET /.cache_ HTTP/1.1
84	0.104665	192.168.10.5	192.168.10.1	HTTP	188	GET /.config HTTP/1.1
94	0.115386	192.168.10.5	192.168.10.1	HTTP	189	GET /.config_ HTTP/1.1
104	0.126273	192.168.10.5	192.168.10.1	HTTP	185	GET /.cvs HTTP/1.1
114	0.137354	192.168.10.5	192.168.10.1	HTTP	186	GET /.cvs_ HTTP/1.1
124	0.149917	192.168.10.5	192.168.10.1	HTTP	191	GET /.cvsignore HTTP/1.1
134	0.159443	192.168.10.5	192.168.10.1	HTTP	192	GET /.cvsignore_ HTTP/1.1
144	0.170412	192.168.10.5	192.168.10.1	HTTP	189	GET /.forward HTTP/1.1
154	0.181592	192.168.10.5	192.168.10.1	HTTP	190	GET /.forward_ HTTP/1.1
164	0.192531	192.168.10.5	192.168.10.1	HTTP	190	GET /.git/HEAD HTTP/1.1

-> We then examine protocol hierarchy

Wireshark · Protocol Hierarchy Statistics · basic_fuzzing.pcapng						
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets
▼ Frame	100.0	204	100.0	38570	141 k	0
▼ Ethernet	100.0	204	7.4	2856	10 k	0
▼ Internet Protocol Version 4	100.0	204	10.6	4080	14 k	0
▼ Transmission Control Protocol	100.0	204	82.0	31634	115 k	0
Hypertext Transfer Protocol	100.0	204	65.1	25106	91 k	204

-> 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 hierarchy

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

Wireshark - Protocol Hierarchy Statistics - CRLF_and_host_header

Protocol	Percent Packets	Packets	Percent Bytes	Bytes
Frame	100.0	7	100.0	3843
Ethernet	100.0	7	2.6	98
Internet Protocol Version 4	100.0	7	3.6	140
Transmission Control Protocol	100.0	7	93.8	3605
Hypertext Transfer Protocol	100.0	7	88.0	3381
Line-based text data	100.0	7	54.8	2107

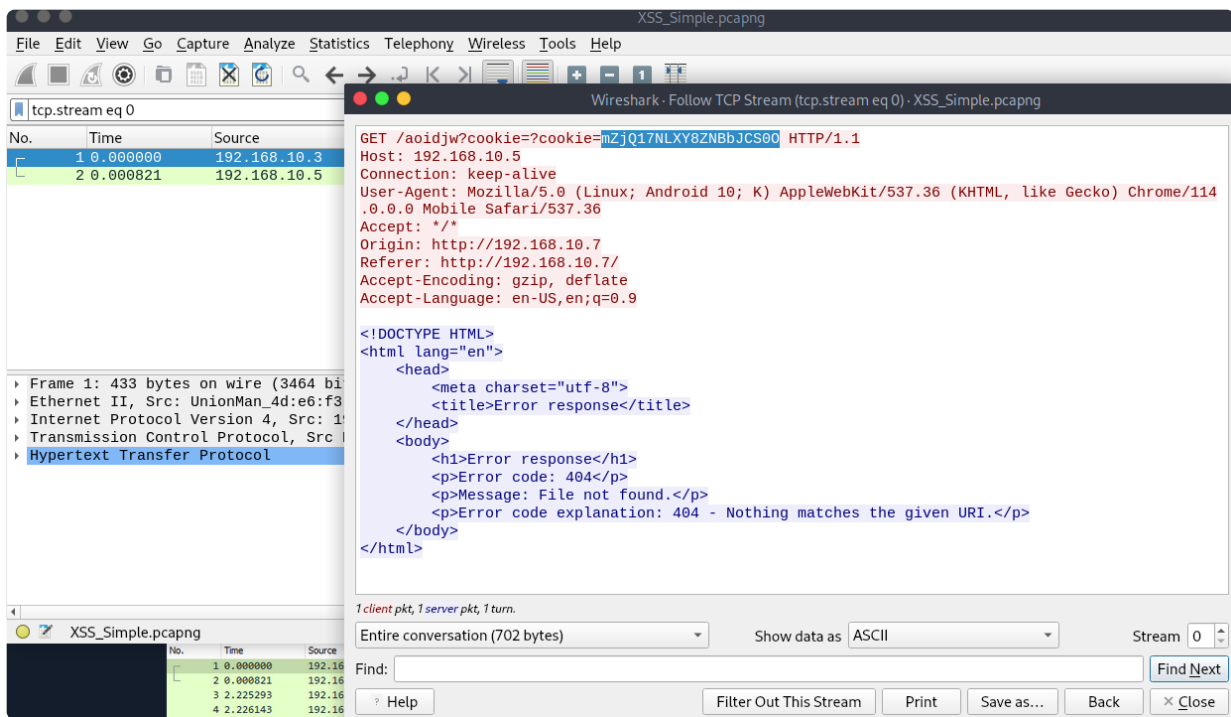
Display filter: http.response.code == 400

-> 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
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
2	0.000001000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
14	0.000013000	192.168.10.56	192.168.10.23	TLSv1.2	364	Client Hello
17	0.000016000	192.168.10.56	192.168.10.23	TLSv1.2	364	Client Hello
28	0.000033000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
29	0.000034000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
54	0.000060000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
57	0.000063000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
60	0.000066000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
61	0.000067000	192.168.10.56	192.168.10.23	TLSv1.2	364	Client Hello
79	0.000086000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello
84	0.000091000	192.168.10.56	192.168.10.23	TLSv1	364	Client Hello

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

Wireshark · Protocol Hierarchy Statistics · SSL_renegotiation_edited.pcapng

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

Display filter: ssl.record.content_type == 22 && tls.handshake.type == 1

? Help Copy Close

-> 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

No.	Time	Source	Destination	Protocol	Length	Info
7	3.440643	192.168.10.5	192.168.10.1	DNS	121	Standard query 0x0000 A htb.com TXT
8	3.453112	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
9	4.605353	192.168.10.5	192.168.10.1	DNS	121	Standard query 0x0000 A htb.com TXT
10	4.621037	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
11	106.499241	192.168.10.5	192.168.10.1	DNS	203	Standard query 0x0000 A htb.com TXT
12	106.512126	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
13	205.099424	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT
14	205.113726	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
15	206.062238	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT
16	206.073552	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
17	207.254310	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT
18	207.264705	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
19	208.351203	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT
20	208.363496	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
21	209.423461	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT
22	209.442150	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
23	210.503027	192.168.10.5	192.168.10.1	DNS	104	Standard query 0x0000 A htb.com TXT

Queries	0000 2c 30 33 e2 d5 c3 08 00 27 53 0c ba 08 00 45 00 ,03....'S....E
htb.com: type A, class IN	0010 00 bd 00 01 00 00 40 11 e4 d8 c0 a8 0a 05 c0 a8@.....
Name: htb.com	0020 0a 01 00 35 00 35 00 a9 8d b6 00 00 01 00 00 01 ...-5.....
[Name Length: 7]	0030 00 01 00 00 00 00 03 68 74 62 03 63 6f 6d 00 00h tb.com...
[Label Count: 2]	0040 01 00 01 03 68 74 62 03 63 6f 6d 00 00 10 00 01htb.com.....
Type: A (Host Address) (1)	0050 00 00 00 0a 00 75 74 56 54 42 61 55 31 45 79 50ut/ TBaU1EyV
Class: IN (0x0001)	0060 58 68 61 53 46 70 72 56 6a 4e 6f 63 6c 64 45 54 XhaSFprV jNocldET
Answers	0070 6e 4e 6b 62 56 4a 58 54 31 63 78 61 55 30 77 62 nNkbVJXT 1cxaU0wb
htb.com: type TXT, class IN	0080 33 70 58 56 6d 68 4c 59 54 46 6e 05 55 31 58 65 3pXvmhLY TFneU1Xe
Name: htb.com	0090 46 6c 4e 4d 55 70 32 57 56 5a 6f 54 31 70 74 54 FlnMUp2W ZoT1ptT
Type: TXT (Text strings) (16)	00a0 6b 6c 54 62 58 68 72 55 30 5a 4a 4d 56 64 45 54 kLTbXhrU 0ZJMvdET
Class: IN (0x0001)	00b0 6b 4e 6a 4d 58 42 59 55 6d 35 77 59 51 70 58 52 kNjMxByU m5wYQpXR
Time to live: 10 (10 seconds)	00c0 45 4a 4d 51 32 63 39 50 51 6f 3d EJMQ2c9P Qo=
Data length: 117	
TXT Length: 116	
TXT: VTBaU1EyVXhaSFprVjNocldETnNkbVJXT1cxaU0wb3pXvmhLYTFneU1XeFlnMUp2WZoT1ptTk1TbXhrU0ZJMvdETkNjMxByUm5wYQpXREJMQ2c9PQo=	
[Response In: 12]	

-> Then we triple decode as follows:

```
echo
'VTBaU1EyVXhaSFprVjNocldETnNkbVJXT1cxaU0wb3pXvmhLYTFneU1XeFlnMUp2WZoT1p
tTk1TbXhrU0ZJMvdETkNjMxByUm5wYQpXREJMQ2c9PQo=' | base64 -d | base64 -d |
base64 -d
```

```
[*]$ echo 'VTBaU1EyVXhaSFprVjNocldETnNkbVJXT1cxaU0wb3pXvmhLYTFneU1XeFlnMUp2
WZoT1ptTk1TbXhrU0ZJMvdETkNjMxByUm5wYQpXREJMQ2c9PQo=' | base64 -d | base64 -d |
base64 -d
HTB{Would_you_forward_me_this_pretty_please}
```

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)

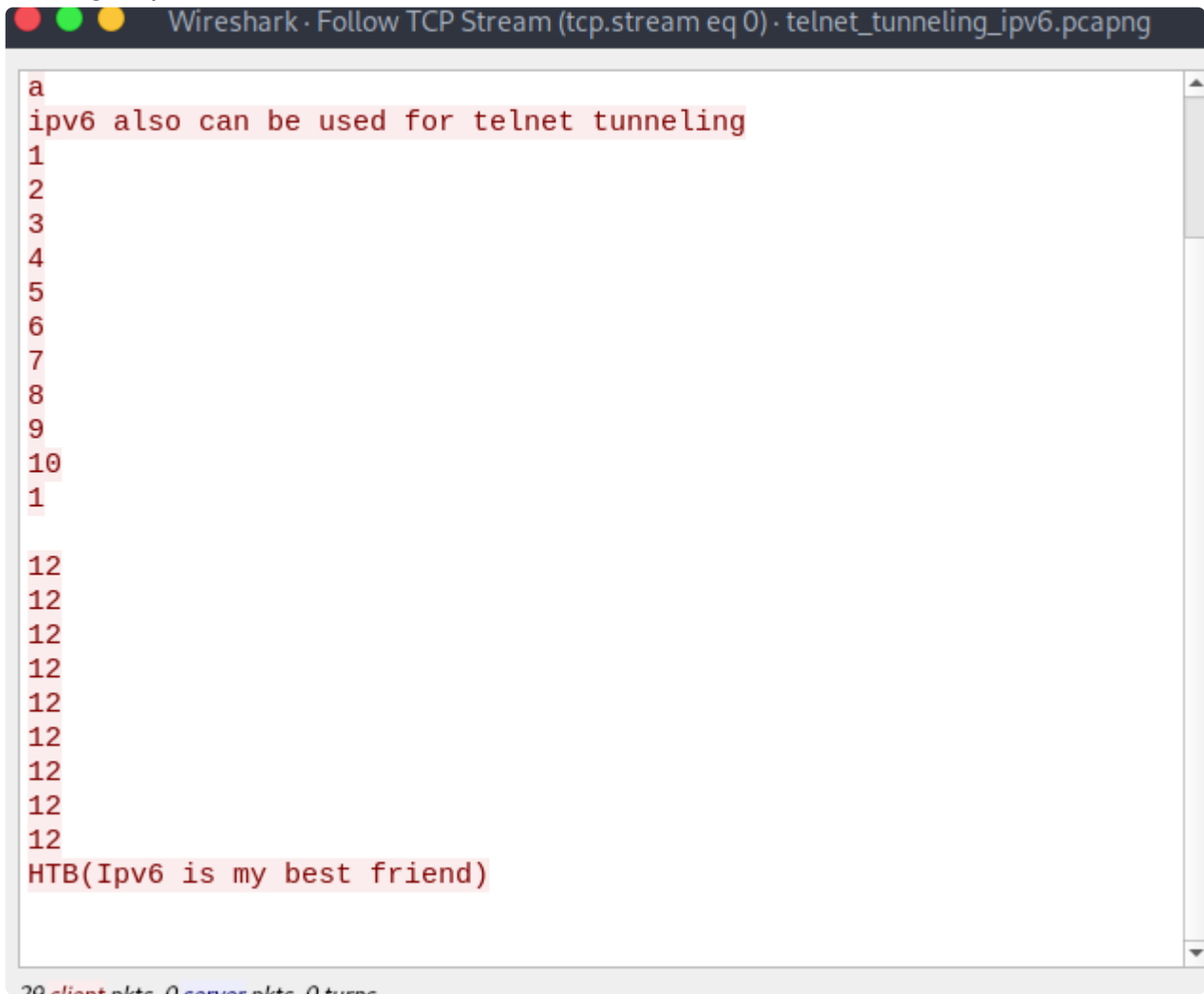
-> We first examine the file and filter for source and destination ip (as that is the host and starts the communication for telnet)

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

((ipv6.src_host == fe80::c9c8:ed3:1b10:f10b) or (ipv6.dst_host == fe80::c9c8:ed3:1b10:f10b)) and telnet							
No.	Time	Source	Destination	Protocol	Length	Info	
1	0.000000	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
3	0.327967	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
5	0.630079	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
7	0.939934	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
9	1.272028	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
11	2.219247	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
15	11.093126	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	130	Telnet	
17	19.777904	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
19	20.191486	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
21	20.643738	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
23	21.032943	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
25	21.443061	fe80::c9c8:ed3:1b10...	fe80::468a:5bff:fe9...	TELNET	89	Telnet	
Frame 1: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0							
Ethernet II, Src: PcsCompu_53:0c:ba (08:00:27:53:0c:ba), Dst: fe80::468a:5bff:fe9...							
Internet Protocol Version 6, Src: fe80::c9c8:ed3:1b10:f10b, Dst: fe80::468a:5bff:fe9...							
Transmission Control Protocol, Src Port: 512, Dst Port: 23							
Telnet							
Data: a\r\n							
				0000	44 8a 5b 95 68 2a 08 00	27 53 0c ba 8	
				0010	12 7b 00 23 06 40 fe 80	00 00 00 00 0	
				0020	0e d3 1b 10 f1 0b fe 80	00 00 00 00 0	
				0030	5b ff fe 95 68 2a c8 16	00 17 dc f5 9	
				0040	9a 33 80 18 01 fb c9 a1	00 00 01 01 0	
				0050	2d 1c 22 83 cc e4 61 0d	0a	

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

the flag required

A screenshot of the Wireshark network protocol analyzer. The title bar reads "Wireshark · Follow TCP Stream (tcp.stream eq 0) · telnet_tunneling_ipv6.pcapng". The main display area shows a list of packets on the left, with packet 1 selected. The packet details pane on the right shows the "Hypertext Transfer Protocol" section, which contains the text "HTB(Ipv6 is my best friend)". The packet list on the left shows a sequence of packets: 1 (Hypertext Transfer Protocol), 2-10 (Hypertext Transfer Protocol), 11 (Hypertext Transfer Protocol), 12-18 (Hypertext Transfer Protocol), and 19 (Hypertext Transfer Protocol). The packet 19 is highlighted in red, indicating it is the selected packet. The packet details pane shows the "Hypertext Transfer Protocol" section with the text "HTB(Ipv6 is my best friend)". The packet list on the left shows a sequence of packets: 1 (Hypertext Transfer Protocol), 2-10 (Hypertext Transfer Protocol), 11 (Hypertext Transfer Protocol), 12-18 (Hypertext Transfer Protocol), and 19 (Hypertext Transfer Protocol). The packet 19 is highlighted in red, indicating it is the selected packet.

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.

The image displays a Wireshark packet capture analysis of a DNS query and response. The packet list shows a standard query (No. 1) and a standard response (No. 2). The packet details pane shows the query for 'vaaaakardli.pirate.sea' and the response containing the IP address '10.0.2.20'. The packet bytes pane shows the raw data in hexadecimal and ASCII.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.20	10.0.2.20	DNS	82	Standard query 0x12b0 NULL vaaaakardli.pirate.sea
2	0.000093	10.0.2.20	10.0.2.20	DNS	103	Standard query response 0x12b0 NULL vaaaakardli.pirate.sea NULL vaaaakardli.pirate.sea
3	0.000897	10.0.2.20	10.0.2.20	DNS	103	Standard query response 0x30df NULL laegpump1lhbpz2yndief1jwlkjcgyw.pirate.sea
4	0.001176	10.0.2.20	10.0.2.20	DNS	144	Standard query response 0x30df NULL laegpump1lhbpz2yndief1jwlkjcgyw.pirate.sea NULL ...
5	0.006692	10.0.2.20	10.0.2.20	DNS	88	Standard query 0x4f0e NULL yrbio2.pirate.sea OPT
6	0.006891	10.0.2.20	10.0.2.20	DNS	137	Standard query response 0x4f0e NULL yrbio2.pirate.sea NULL yrbio2.pirate.sea
7	0.007103	10.0.2.20	10.0.2.20	DNS	123	Standard query 0x6d3d NULL z103AA-AAahhh-Drink-mal-ein-J0germeister-.pirate.sea OPT
8	0.007233	10.0.2.20	10.0.2.20	DNS	166	Standard query response 0x6d3d NULL z103AA-AAahhh-Drink-mal-ein-J0germeister-.pirate.sea ...
9	0.007348	10.0.2.20	10.0.2.20	DNS	132	Standard query 0x8b6c NULL z104AA-La-fl0te-na0ve-fran0aise-est-retir0-0-cr0te.pirate...
10	0.007460	10.0.2.20	10.0.2.20	DNS	184	Standard query response 0x8b6c NULL z104AA-La-fl0te-na0ve-fran0aise-est-retir0-0-cr0te...
11	0.007567	10.0.2.20	10.0.2.20	DNS	138	Standard query 0xa99b NULL z105AABbCdDeFFgGhHijJkKlMmNnOopPqRrStTuVvWwXyZz.p...
12	0.007677	10.0.2.20	10.0.2.20	DNS	196	Standard query response 0xa99b NULL z105AABbCdDeFFgGhHijJkKlMmNnOopPqRrStTuVvWw...

Packet Details:

- Frame 40: 323 bytes on wire (2584 bits), 323 bytes captured (2584 bits) on interface 0
- Ethernet II, Src: PcsCompu.9c:0e:b4 (08:00:27:9c:0e:b4), Dst: PcsCompu.c7:6e:ba (08:00:27:9c:0e:b4)
- Internet Protocol Version 4, Src: 10.0.2.20, Dst: 10.0.2.20
- User Datagram Protocol, Src Port: 44639, Dst Port: 53
- Domain Name System (query)
 - Transaction ID: 0xc8e9
 - Flags: 0x0100 Standard query
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 1
 - Queries
 - 1. Standard query type A for vaaaakardli.pirate.sea

Packet Bytes:

```

0000  08 00 27 c7 6e ba 08 00 27 9c 0e b4 08 00 00 00  :...n...:
0010  01 35 00 00 40 00 40 01 21 87 0a 00 02 1e 0a 00  :.5.@!.....:
0020  02 14 ae 5f 00 35 01 21 4d ef c8 e9 01 00 00 01  :.2.14.ae.5f.00.35.01.21.4d.ef.c8.e9.01.00.00.01:
0030  00 00 00 00 00 00 01 3e 72 64 06 05 04 34 c6 c8  :.....01.3e.72.64.06.05.04.34.c6.c8:
0040  6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55 78  :.c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55.78:
0050  6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55 78  :.c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55.78:
0060  6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55 78  :.c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55.78:
0070  6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55 78  :.c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55.78:
0080  78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55  :.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55:
0090  78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55  :.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55:
00a0  78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da 55  :.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da.55:
00b0  55 78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da  :.55.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da:
00c0  55 78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da  :.55.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da:
00d0  55 78 6c c3 e0 ef 34 da 55 78 6c c3 e0 ef 34 da  :.55.78.6c.c3.e0.ef.34.da.55.78.c.c3.e0.ef.34.da:

```

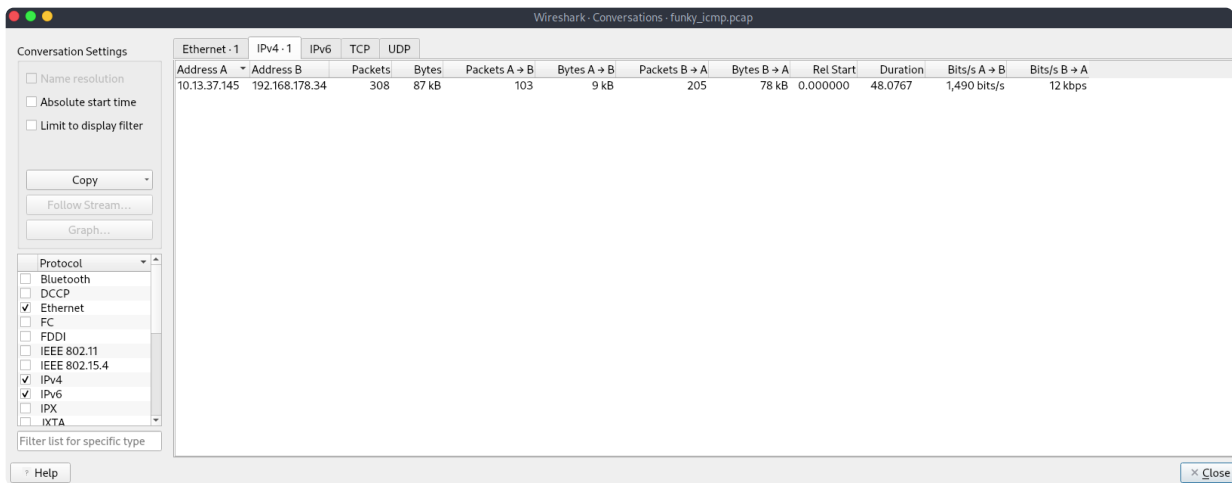
- Looking at the some of the request being made, we see the following:

[illegible]

- 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	60	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	42	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	3.009696	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	4.238674	192.168.178.34	10.13.37.145	ICMP	102	Echo (ping) reply id=0x711d, seq=0/0, ttl=128 (request in 11)
13	4.239634	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 hierarchy.

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

No.	Time	Source	Destination	Protocol	Length	Info
165	4.252543	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
162	4.252334	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
160	4.252156	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
158	4.251983	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
156	4.251819	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
154	4.251636	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
152	4.251412	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
150	4.251239	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
148	4.251032	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
142	4.250513	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
138	4.250282	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
136	4.250194	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
131	4.249642	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
128	4.249419	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
123	4.248935	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
118	4.248591	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
110	4.248053	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128
105	4.247781	192.168.178.34	10.13.37.145	ICMP	1514	Echo (ping) reply id=0x711d, seq=0/0, ttl=128

```

▶ Frame 165: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▶ Ethernet II, Src: VMware_e8:3e:63 (00:50:56:e8:3e:63), Dst: VMware_c2:2c:40 (00:0c:29:
▶ Internet Protocol Version 4, Src: 192.168.178.34, Dst: 10.13.37.145
▼ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0xaeb3 [correct]
  [Checksum Status: Good]
  Identifier (BE): 28957 (0x711d)
  Identifier (LE): 7537 (0x1d71)
  Sequence Number (BE): 0 (0x0000)
  Sequence Number (LE): 0 (0x0000)
▼ Data (1500 bytes)
  Data: 450005dccc3654000400657b40a0003010a0003020050baa11cceeef2bb34d6098010001d...
  [Length: 1500]

```

-> 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.