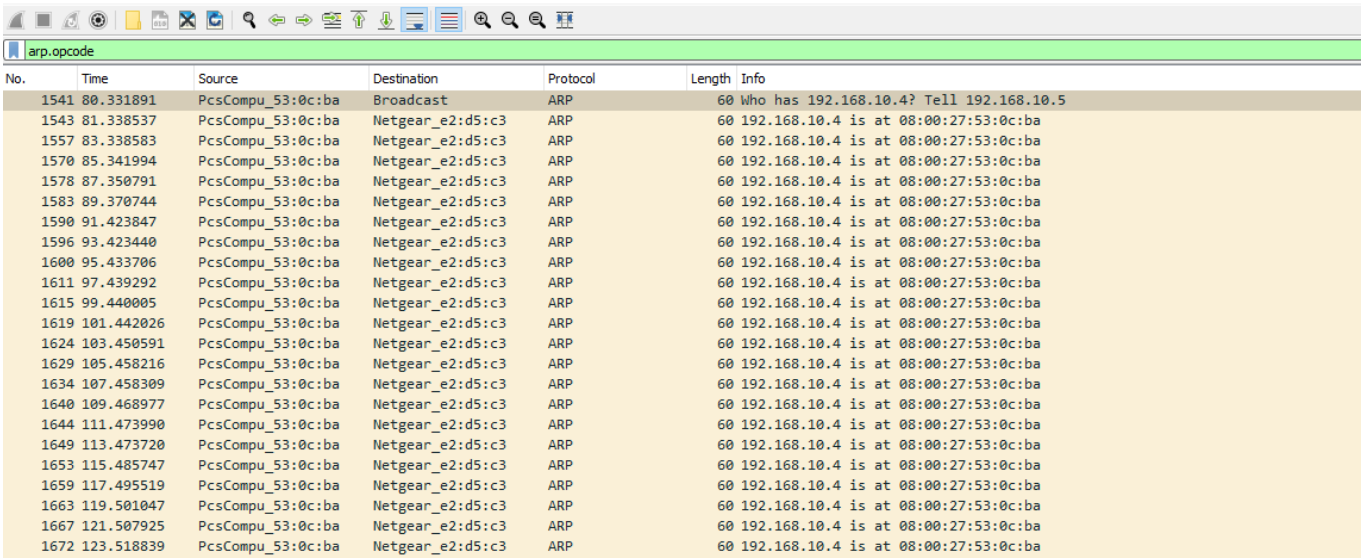
Wireshark + TCPDUMP + Tshark cheat sheet

[cheatsheet-81]

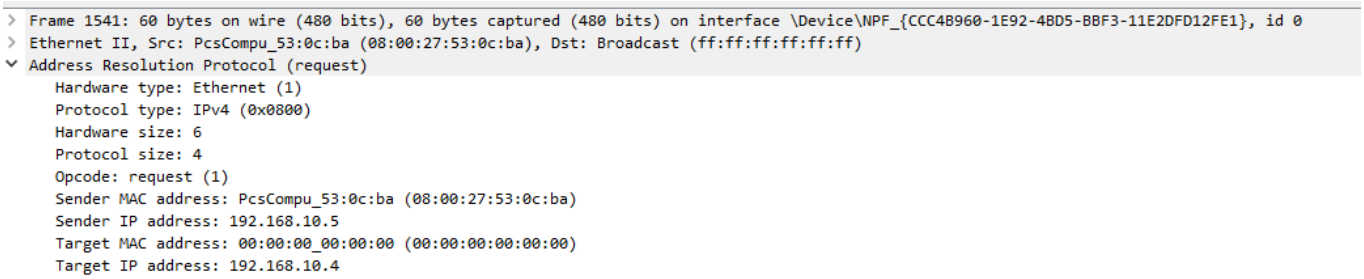
ARP Spoofing & Abnormality Detection

`arp.opcode`



Identifying The Original IP Addresses

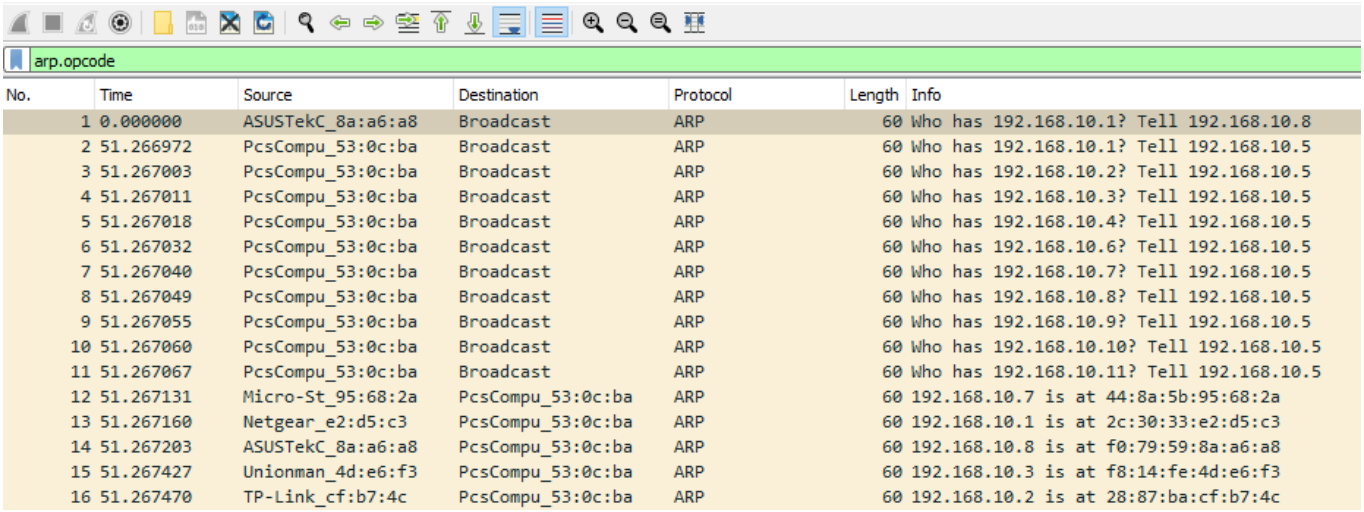
`(arp.opcode) && ((eth.src == 08:00:27:53:0c:ba) || (eth.dst == 08:00:27:53:0c:ba))`



in this case, we might instantly note that the MAC address `08:00:27:53:0c:ba` was initially linked to the IP address `192.168.10.5`, but this was recently switched to `192.168.10.4`. This transition is indicative of a deliberate attempt at ARP spoofing or cache poisoning.

ARP Scanning & Denial-of-Service

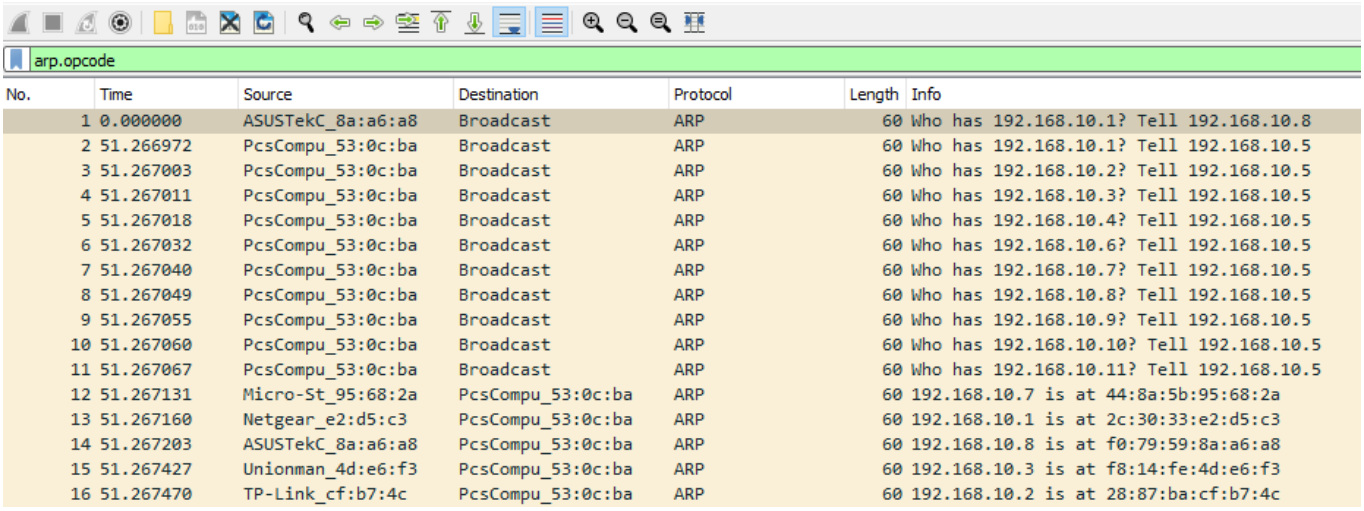
`arp.opcode`



arp.opcode

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	ASUSTekC_8a:a6:a8	Broadcast	ARP	60	Who has 192.168.10.1? Tell 192.168.10.8
2	51.266972	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.1? Tell 192.168.10.5
3	51.267003	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.2? Tell 192.168.10.5
4	51.267011	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.3? Tell 192.168.10.5
5	51.267018	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.4? Tell 192.168.10.5
6	51.267032	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.6? Tell 192.168.10.5
7	51.267040	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.7? Tell 192.168.10.5
8	51.267049	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.8? Tell 192.168.10.5
9	51.267055	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.9? Tell 192.168.10.5
10	51.267060	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.10? Tell 192.168.10.5
11	51.267067	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.11? Tell 192.168.10.5
12	51.267131	Micro-St_95:68:2a	PcsCompu_53:0c:ba	ARP	60	192.168.10.7 is at 44:8a:5b:95:68:2a
13	51.267160	Netgear_e2:d5:c3	PcsCompu_53:0c:ba	ARP	60	192.168.10.1 is at 2c:30:33:e2:d5:c3
14	51.267203	ASUSTekC_8a:a6:a8	PcsCompu_53:0c:ba	ARP	60	192.168.10.8 is at f0:79:59:8a:a6:a8
15	51.267427	Unionman_4d:e6:f3	PcsCompu_53:0c:ba	ARP	60	192.168.10.3 is at f8:14:fe:4d:e6:f3
16	51.267470	TP-Link_cf:b7:4c	PcsCompu_53:0c:ba	ARP	60	192.168.10.2 is at 28:87:ba:cf:b7:4c

It's possible to detect that indeed ARP requests are being propagated by a single host to all IP addresses in a sequential manner. This pattern is sympto



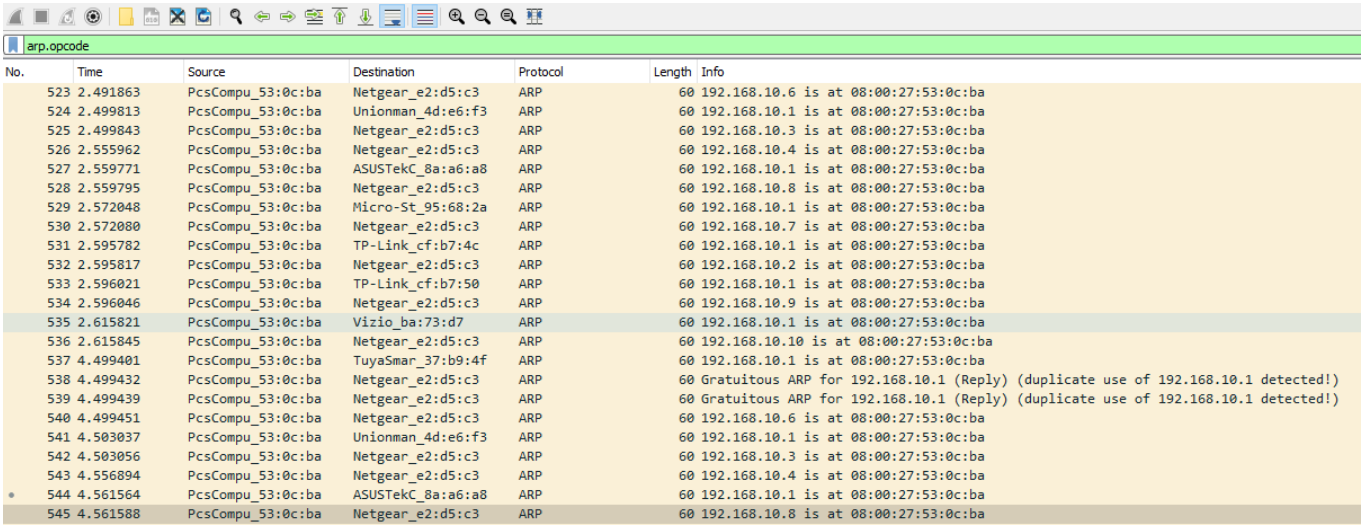
arp.opcode

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	ASUSTekC_8a:a6:a8	Broadcast	ARP	60	Who has 192.168.10.1? Tell 192.168.10.8
2	51.266972	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.1? Tell 192.168.10.5
3	51.267003	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.2? Tell 192.168.10.5
4	51.267011	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.3? Tell 192.168.10.5
5	51.267018	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.4? Tell 192.168.10.5
6	51.267032	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.6? Tell 192.168.10.5
7	51.267040	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.7? Tell 192.168.10.5
8	51.267049	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.8? Tell 192.168.10.5
9	51.267055	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.9? Tell 192.168.10.5
10	51.267060	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.10? Tell 192.168.10.5
11	51.267067	PcsCompu_53:0c:ba	Broadcast	ARP	60	Who has 192.168.10.11? Tell 192.168.10.5
12	51.267131	Micro-St_95:68:2a	PcsCompu_53:0c:ba	ARP	60	192.168.10.7 is at 44:8a:5b:95:68:2a
13	51.267160	Netgear_e2:d5:c3	PcsCompu_53:0c:ba	ARP	60	192.168.10.1 is at 2c:30:33:e2:d5:c3
14	51.267203	ASUSTekC_8a:a6:a8	PcsCompu_53:0c:ba	ARP	60	192.168.10.8 is at f0:79:59:8a:a6:a8
15	51.267427	Unionman_4d:e6:f3	PcsCompu_53:0c:ba	ARP	60	192.168.10.3 is at f8:14:fe:4d:e6:f3
16	51.267470	TP-Link_cf:b7:4c	PcsCompu_53:0c:ba	ARP	60	192.168.10.2 is at 28:87:ba:cf:b7:4c

matic of ARP scanning and is a common feature of widely-used scanners such as `Nmap`.

Identifying Denial-of-Service

`arp.opcode`



arp.opcode

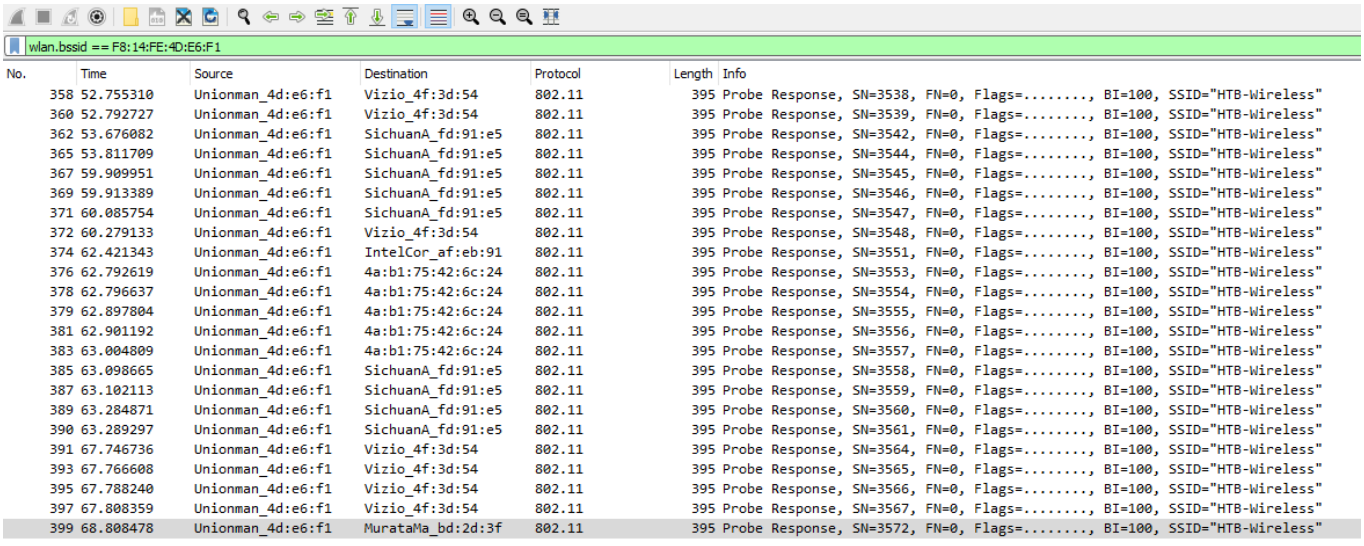
No.	Time	Source	Destination	Protocol	Length	Info
523	2.491863	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.6 is at 08:00:27:53:0c:ba
524	2.499813	PcsCompu_53:0c:ba	Unionman_4d:e6:f3	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
525	2.499843	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.3 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
530	2.572080	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.7 is at 08:00:27:53:0c:ba
531	2.595782	PcsCompu_53:0c:ba	TP-Link_cf:b7:4c	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
532	2.595817	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.2 is at 08:00:27:53:0c:ba
533	2.596021	PcsCompu_53:0c:ba	TP-Link_cf:b7:50	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
534	2.596046	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.9 is at 08:00:27:53:0c:ba
535	2.615821	PcsCompu_53:0c:ba	Vizio_ba:73:d7	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
536	2.615845	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.10 is at 08:00:27:53:0c:ba
537	4.499401	PcsCompu_53:0c:ba	TuyaSmar_37:b9:4f	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
538	4.499432	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	Gratuitous ARP for 192.168.10.1 (Reply) (duplicate use of 192.168.10.1 detected!)
539	4.499439	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	Gratuitous ARP for 192.168.10.1 (Reply) (duplicate use of 192.168.10.1 detected!)
540	4.499451	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.6 is at 08:00:27:53:0c:ba
541	4.503037	PcsCompu_53:0c:ba	Unionman_4d:e6:f3	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
542	4.503056	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.3 is at 08:00:27:53:0c:ba
543	4.556894	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.4 is at 08:00:27:53:0c:ba
544	4.561564	PcsCompu_53:0c:ba	ASUSTekC_8a:a6:a8	ARP	60	192.168.10.1 is at 08:00:27:53:0c:ba
545	4.561588	PcsCompu_53:0c:ba	Netgear_e2:d5:c3	ARP	60	192.168.10.8 is at 08:00:27:53:0c:ba

we may witness the duplicate allocation of `192.168.10.1` to client devices. This indicates that the attacker is attempting to corrupt the ARP cache of these victim devices with the intention of obstructing traffic in both directions.

802.11 Denial of Service (DeAuth)

limit our view to traffic from our AP's BSSID (`MAC`)

```
wlan.bssid == xx:xx:xx:xx:xx:xx
```

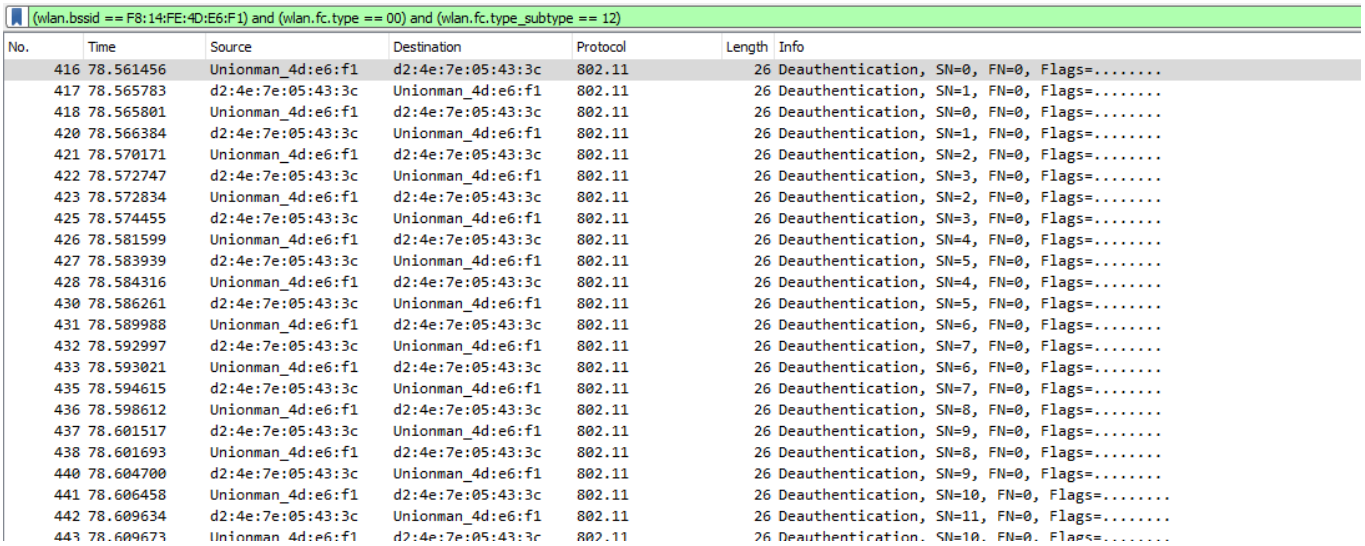


wlan.bssid == F8:14:FE:4D:E6:F1

No.	Time	Source	Destination	Protocol	Length	Info
358	52.755310	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3538, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
360	52.792727	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3539, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
362	53.676082	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3542, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
365	53.811709	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3544, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
367	59.909951	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3545, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
369	59.913389	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3546, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
371	60.085754	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3547, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
372	60.279133	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3548, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
374	62.421343	Unionman_4d:e6:f1	IntelCor_af:eb:91	802.11	395	Probe Response, SN=3551, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
376	62.792619	Unionman_4d:e6:f1	4a:b1:75:42:6c:24	802.11	395	Probe Response, SN=3553, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
378	62.796637	Unionman_4d:e6:f1	4a:b1:75:42:6c:24	802.11	395	Probe Response, SN=3554, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
379	62.897804	Unionman_4d:e6:f1	4a:b1:75:42:6c:24	802.11	395	Probe Response, SN=3555, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
381	62.901192	Unionman_4d:e6:f1	4a:b1:75:42:6c:24	802.11	395	Probe Response, SN=3556, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
383	63.004809	Unionman_4d:e6:f1	4a:b1:75:42:6c:24	802.11	395	Probe Response, SN=3557, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
385	63.098665	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3558, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
387	63.102113	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3559, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
389	63.284871	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3560, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
390	63.289297	Unionman_4d:e6:f1	SichuanA_fd:91:e5	802.11	395	Probe Response, SN=3561, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
391	67.746736	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3564, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
393	67.766608	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3565, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
395	67.788240	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3566, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
397	67.808359	Unionman_4d:e6:f1	Vizio_4f:3d:54	802.11	395	Probe Response, SN=3567, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"
399	68.808478	Unionman_4d:e6:f1	MurataMa_bd:2d:3f	802.11	395	Probe Response, SN=3572, FN=0, Flags=........, BI=100, SSID="HTB-Wireless"

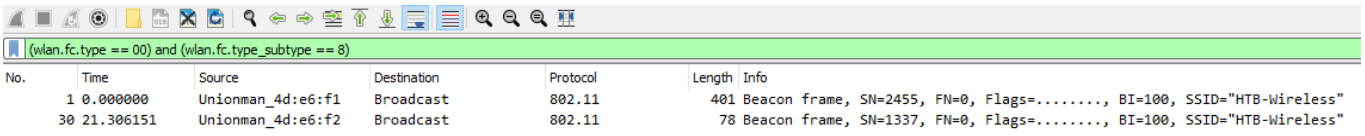
Suppose we wanted to take a look at the deauthentication frames from our BSSID or an attacker pretending to send these from our BSSID, we could use the following Wireshark filter:

- `(wlan.bssid == xx:xx:xx:xx:xx:xx) and (wlan.fc.type == 00) and (wlan.fc.type_subtype == 12)`

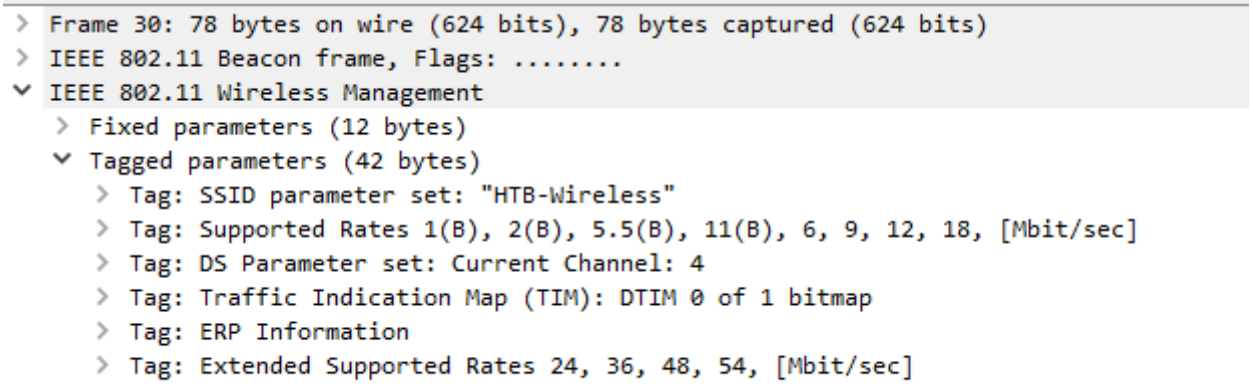


Rogue Access Point & Evil-Twin Attacks

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



No RSN

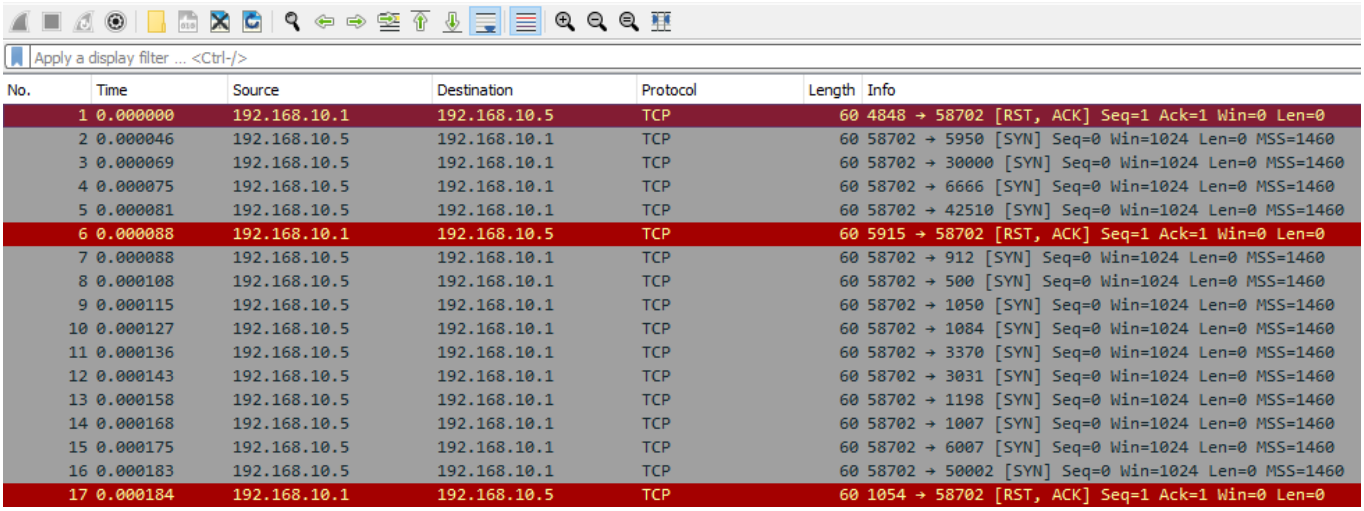


IP Time-to-Live Attacks

Low TTL Value. Example 3

```
Internet Protocol Version 4, Src: 192.168.10.5, Dst: 192.168.10.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x7312 (29458)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
  v Time to Live: 3
      > [Expert Info (Note/Sequence): "Time To Live" only 3]
    Protocol: TCP (6)
    Header Checksum: 0xaf67 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.10.5
    Destination Address: 192.168.10.1
```

Excessive SYN Flags



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

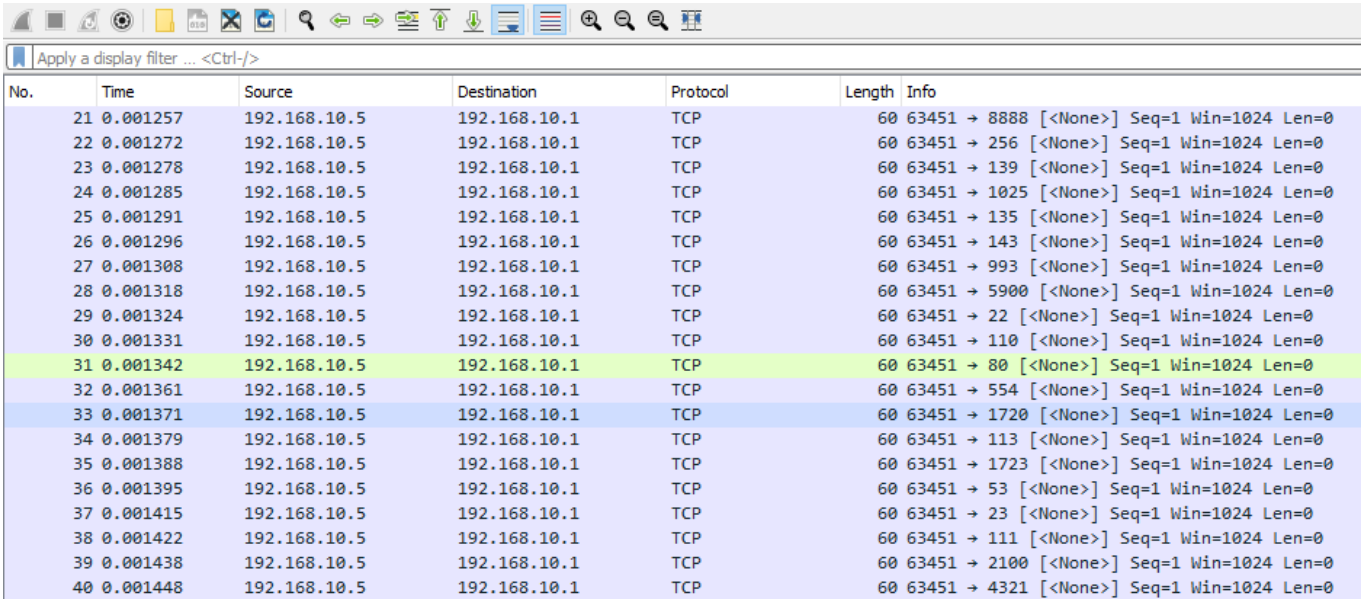
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.1	192.168.10.5	TCP	60	4848 → 58702 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2	0.000046	192.168.10.5	192.168.10.1	TCP	60	58702 → 5950 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
3	0.000069	192.168.10.5	192.168.10.1	TCP	60	58702 → 30000 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
4	0.000075	192.168.10.5	192.168.10.1	TCP	60	58702 → 6666 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
5	0.000081	192.168.10.5	192.168.10.1	TCP	60	58702 → 42510 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
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
7	0.000088	192.168.10.5	192.168.10.1	TCP	60	58702 → 912 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
8	0.000108	192.168.10.5	192.168.10.1	TCP	60	58702 → 500 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
9	0.000115	192.168.10.5	192.168.10.1	TCP	60	58702 → 1050 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10	0.000127	192.168.10.5	192.168.10.1	TCP	60	58702 → 1084 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
11	0.000136	192.168.10.5	192.168.10.1	TCP	60	58702 → 3370 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
12	0.000143	192.168.10.5	192.168.10.1	TCP	60	58702 → 3031 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
13	0.000158	192.168.10.5	192.168.10.1	TCP	60	58702 → 1198 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
14	0.000168	192.168.10.5	192.168.10.1	TCP	60	58702 → 1007 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
15	0.000175	192.168.10.5	192.168.10.1	TCP	60	58702 → 6007 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
16	0.000183	192.168.10.5	192.168.10.1	TCP	60	58702 → 50002 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
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

1. `SYN Scans` - In these scans the behavior will be as we see, however the attacker will pre-emptively end the handshake with the RST flag.
2. `SYN Stealth Scans` - In this case the attacker will attempt to evade detection by only partially completing the TCP handshake.

No Flags

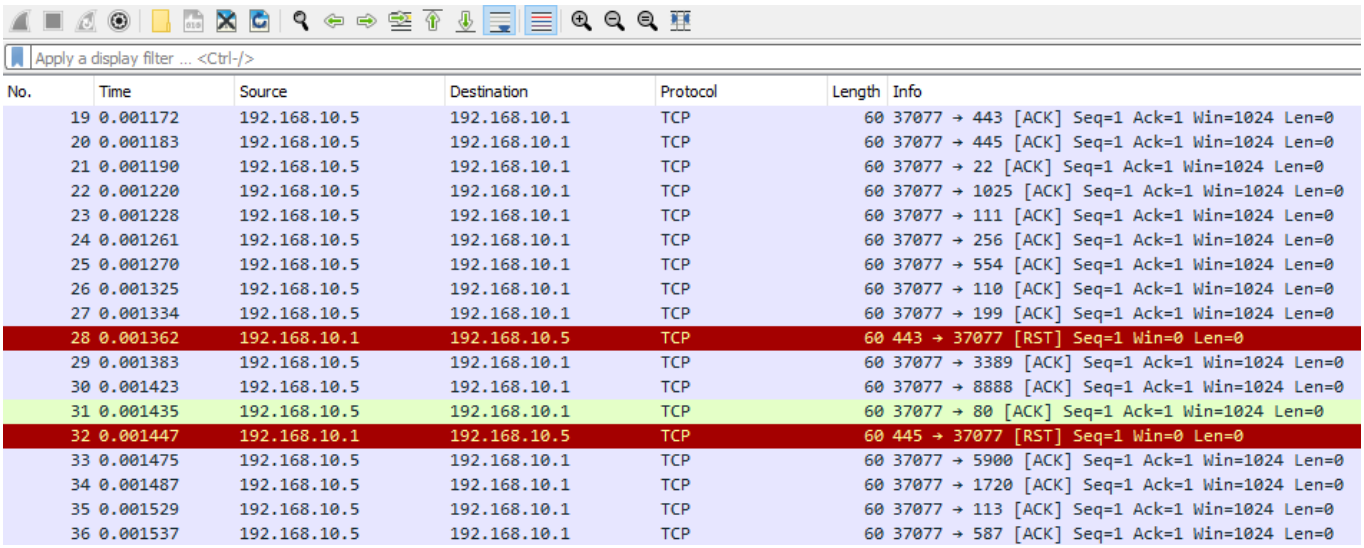
TCP Connection with Null Flags

1. `If the port is open` - The system will not respond at all since there is no flags.
2. `If the port is closed` - The system will respond with an RST packet.



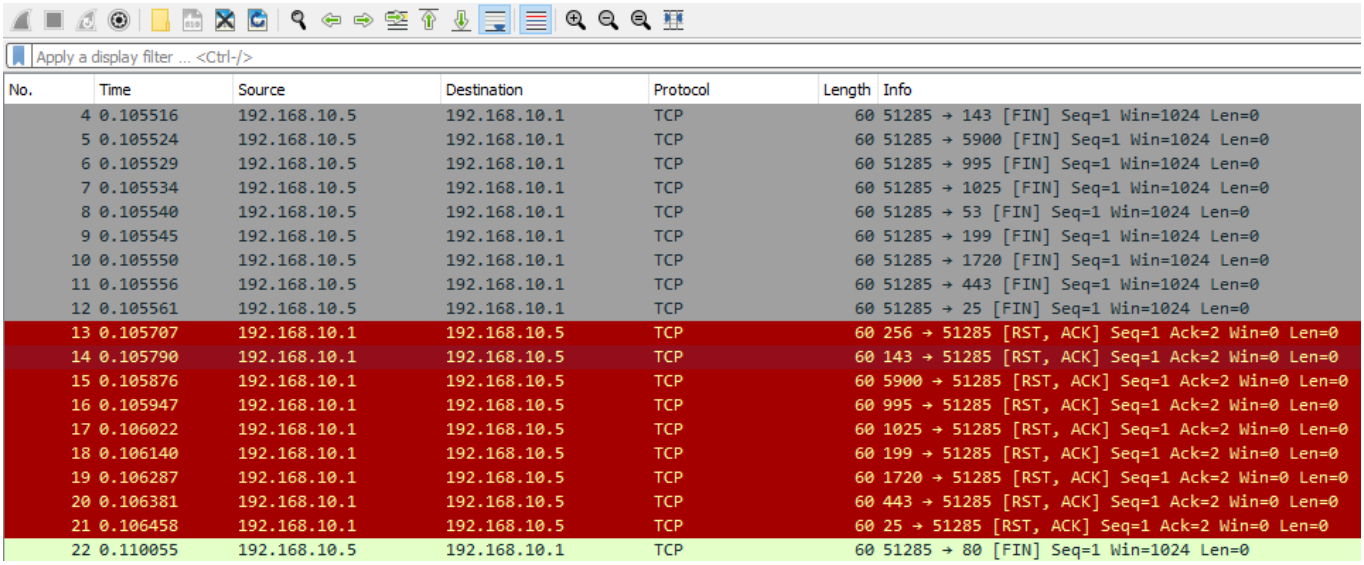
Too Many ACKs

1. `If the port is open` - The affected machine will either not respond, or will respond with an RST packet.
2. `If the port is closed` - The affected machine will respond with an RST packet.



Excessive FINs

1. `If the port is open` - Our affected machine simply will not respond.
2. `If the port is closed` - Our affected machine will respond with an RST packet.

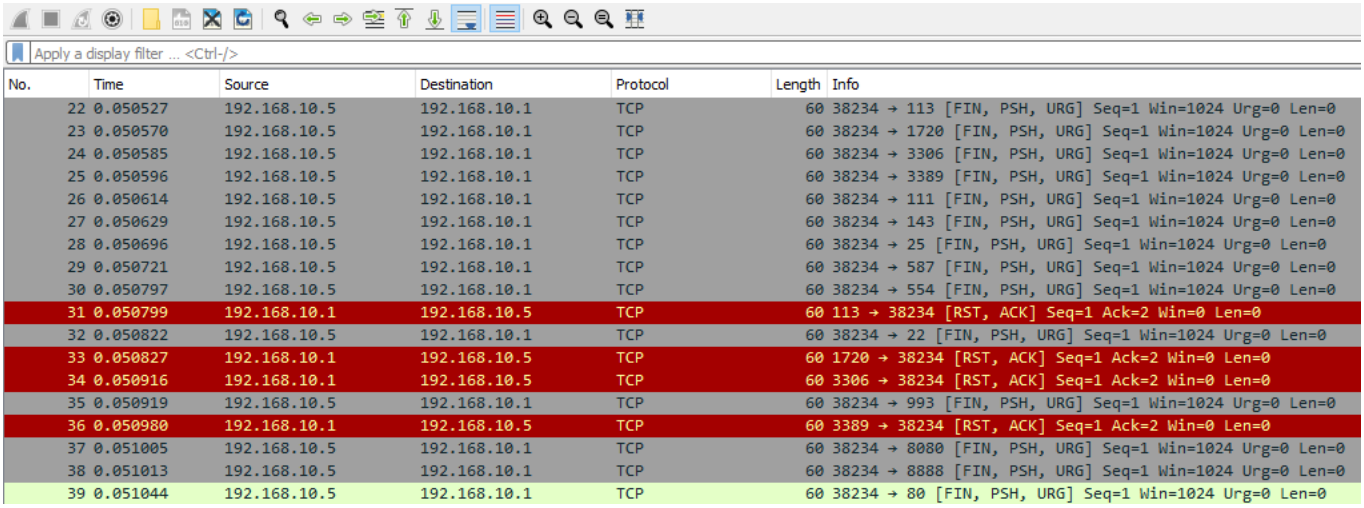


Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
4	0.105516	192.168.10.5	192.168.10.1	TCP	60	51285 → 143 [FIN] Seq=1 Win=1024 Len=0
5	0.105524	192.168.10.5	192.168.10.1	TCP	60	51285 → 5900 [FIN] Seq=1 Win=1024 Len=0
6	0.105529	192.168.10.5	192.168.10.1	TCP	60	51285 → 995 [FIN] Seq=1 Win=1024 Len=0
7	0.105534	192.168.10.5	192.168.10.1	TCP	60	51285 → 1025 [FIN] Seq=1 Win=1024 Len=0
8	0.105540	192.168.10.5	192.168.10.1	TCP	60	51285 → 53 [FIN] Seq=1 Win=1024 Len=0
9	0.105545	192.168.10.5	192.168.10.1	TCP	60	51285 → 199 [FIN] Seq=1 Win=1024 Len=0
10	0.105550	192.168.10.5	192.168.10.1	TCP	60	51285 → 1720 [FIN] Seq=1 Win=1024 Len=0
11	0.105556	192.168.10.5	192.168.10.1	TCP	60	51285 → 443 [FIN] Seq=1 Win=1024 Len=0
12	0.105561	192.168.10.5	192.168.10.1	TCP	60	51285 → 25 [FIN] Seq=1 Win=1024 Len=0
13	0.105707	192.168.10.1	192.168.10.5	TCP	60	256 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
14	0.105790	192.168.10.1	192.168.10.5	TCP	60	143 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
15	0.105876	192.168.10.1	192.168.10.5	TCP	60	5900 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
16	0.105947	192.168.10.1	192.168.10.5	TCP	60	995 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
17	0.106022	192.168.10.1	192.168.10.5	TCP	60	1025 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
18	0.106140	192.168.10.1	192.168.10.5	TCP	60	199 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
19	0.106287	192.168.10.1	192.168.10.5	TCP	60	1720 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
20	0.106381	192.168.10.1	192.168.10.5	TCP	60	443 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
21	0.106458	192.168.10.1	192.168.10.5	TCP	60	25 → 51285 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
22	0.110055	192.168.10.5	192.168.10.1	TCP	60	51285 → 80 [FIN] Seq=1 Win=1024 Len=0

Just too many flags

1. `If the port is open` - The affected machine will not respond, or at least it will with an RST packet.
2. `If the port is closed` - The affected machine will respond with an RST packet.

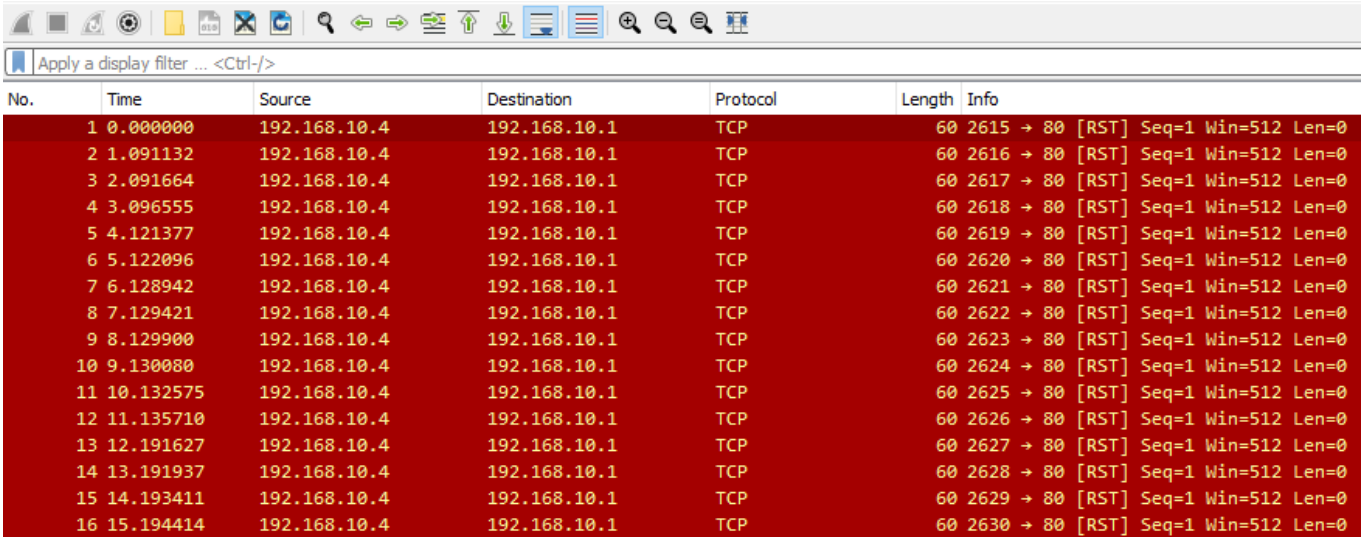


Apply a display filter ... <Ctrl-/>

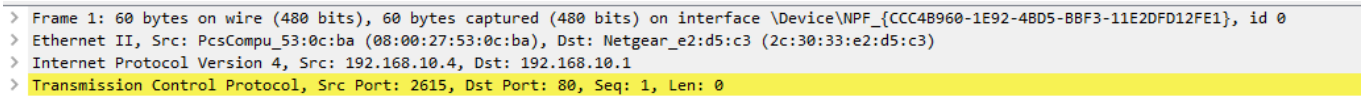
No.	Time	Source	Destination	Protocol	Length	Info
22	0.050527	192.168.10.5	192.168.10.1	TCP	60	38234 → 113 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
23	0.050570	192.168.10.5	192.168.10.1	TCP	60	38234 → 1720 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
24	0.050585	192.168.10.5	192.168.10.1	TCP	60	38234 → 3306 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
25	0.050596	192.168.10.5	192.168.10.1	TCP	60	38234 → 3389 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
26	0.050614	192.168.10.5	192.168.10.1	TCP	60	38234 → 111 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
27	0.050629	192.168.10.5	192.168.10.1	TCP	60	38234 → 143 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
28	0.050696	192.168.10.5	192.168.10.1	TCP	60	38234 → 25 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
29	0.050721	192.168.10.5	192.168.10.1	TCP	60	38234 → 587 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
30	0.050797	192.168.10.5	192.168.10.1	TCP	60	38234 → 554 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
31	0.050799	192.168.10.1	192.168.10.5	TCP	60	113 → 38234 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
32	0.050822	192.168.10.5	192.168.10.1	TCP	60	38234 → 22 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
33	0.050827	192.168.10.1	192.168.10.5	TCP	60	1720 → 38234 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
34	0.050916	192.168.10.1	192.168.10.5	TCP	60	3306 → 38234 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
35	0.050919	192.168.10.5	192.168.10.1	TCP	60	38234 → 993 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
36	0.050980	192.168.10.1	192.168.10.5	TCP	60	3389 → 38234 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0
37	0.051005	192.168.10.5	192.168.10.1	TCP	60	38234 → 8080 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
38	0.051013	192.168.10.5	192.168.10.1	TCP	60	38234 → 8888 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0
39	0.051044	192.168.10.5	192.168.10.1	TCP	60	38234 → 80 [FIN, PSH, URG] Seq=1 Win=1024 Urg=0 Len=0

TCP Connection Termination

1. `The attacker will spoof the source address to be the affected machine's`
2. `The attacker will modify the TCP packet to contain the RST flag to terminate the connection`
3. `The attacker will specify the destination port to be the same as one currently in use by one of our machines.`



One way we can verify that this is indeed a TCP RST attack is through the physical address of the transmitter of these TCP RST packets. Suppose, the IP address 192.168.10.4 is registered to aa:aa:aa:aa:aa:aa in our network device list, and we notice an entirely different MAC sending these like the following.



TCP Connection Hijacking

The attacker will need to block ACKs from reaching the affected machine in order to continue the hijacking. They do this either through delaying or blocking the ACK packets. As such, this attack is very commonly employed with ARP poisoning, and we might notice the following in our traffic analysis.

```
[TCP Retransmission] 23 → 36212 [PSH, ACK]
[TCP Retransmission] 23 → 36212 [PSH, ACK]
[TCP Retransmission] 23 → 36212 [PSH, ACK]
```

ICMP Tunneling

`icmp`

Since ICMP tunneling is primarily done through an attacker adding data into the data field for ICMP, we can find it by looking at the contents of data per request and reply.

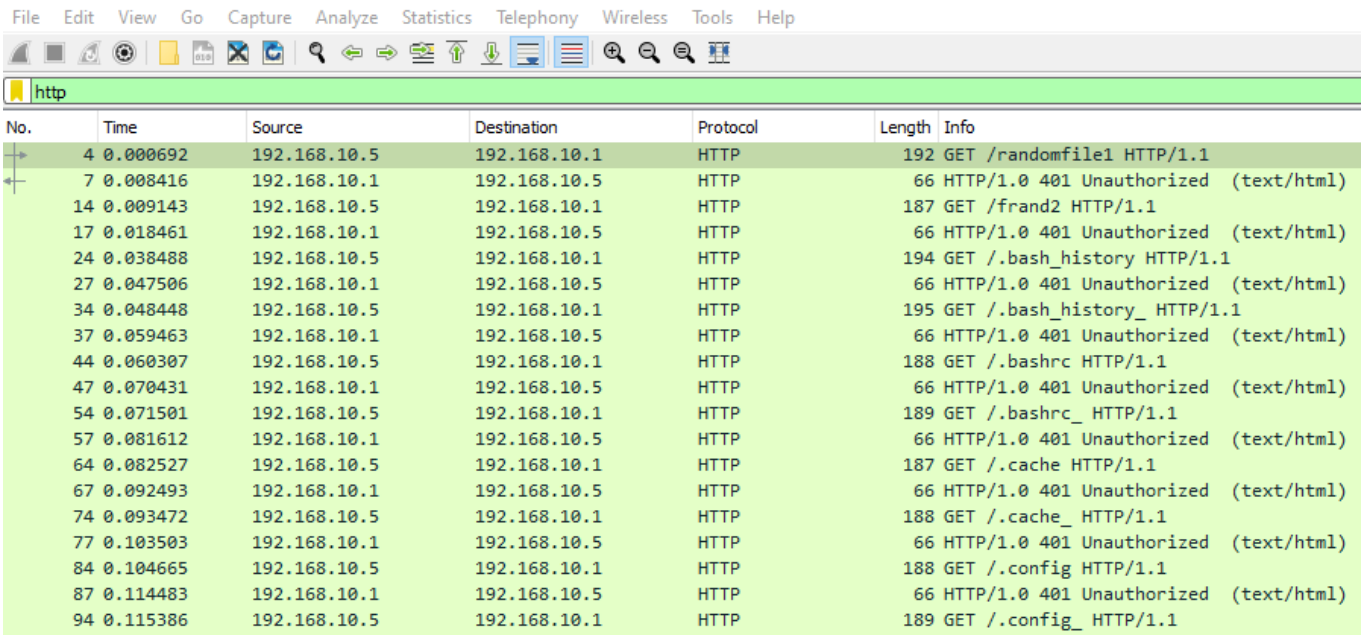
Normal Data byets may be 48, abnormal 3000 and more:

```
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x4ab7 [correct]
    [Checksum Status: Good]
    Identifier (BE): 0 (0x0000)
    Identifier (LE): 0 (0x0000)
    Sequence Number (BE): 0 (0x0000)
    Sequence Number (LE): 0 (0x0000)
    [Response frame: 66]
  Data (38000 bytes)
      Data: 557365726e616d653a20726f6f743b2050617373776f72643a2050617373776f72643132…
      [Length: 38000]
```

Finding Directory Fuzzing

Come on man you know it

`http.request`



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
7	0.008416	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
14	0.009143	192.168.10.5	192.168.10.1	HTTP	187	GET /frand2 HTTP/1.1
17	0.018461	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
24	0.038488	192.168.10.5	192.168.10.1	HTTP	194	GET /.bash_history HTTP/1.1
27	0.047506	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
34	0.048448	192.168.10.5	192.168.10.1	HTTP	195	GET /.bash_history_ HTTP/1.1
37	0.059463	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
44	0.060307	192.168.10.5	192.168.10.1	HTTP	188	GET /.bashrc HTTP/1.1
47	0.070431	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
54	0.071501	192.168.10.5	192.168.10.1	HTTP	189	GET /.bashrc_ HTTP/1.1
57	0.081612	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
64	0.082527	192.168.10.5	192.168.10.1	HTTP	187	GET /.cache HTTP/1.1
67	0.092493	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
74	0.093472	192.168.10.5	192.168.10.1	HTTP	188	GET /.cache_ HTTP/1.1
77	0.103503	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
84	0.104665	192.168.10.5	192.168.10.1	HTTP	188	GET /.config HTTP/1.1
87	0.114483	192.168.10.1	192.168.10.5	HTTP	66	HTTP/1.0 401 Unauthorized (text/html)
94	0.115386	192.168.10.5	192.168.10.1	HTTP	189	GET /.config_ HTTP/1.1

Analyzing Code 400s and Request Smuggling

`http.response.code == 400`

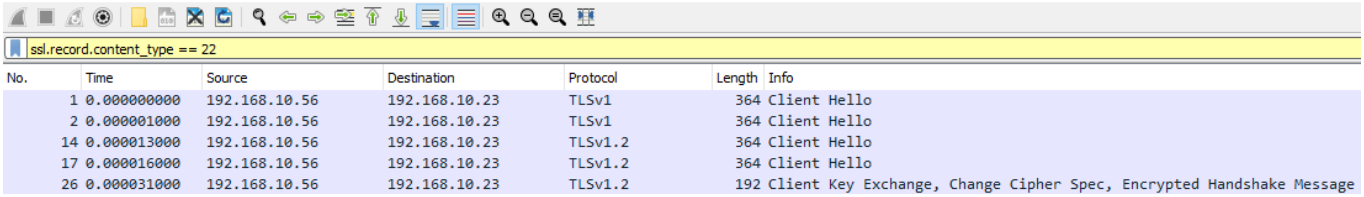
```
GET /login.php?id=1%20HTTP/1.1%0d%0aHost:%20192.168.10.5%0d%0a%0d%0aGET%20/uploads/cmd.php%20HTTP/1.1%0d%0aHost:%20127.0.0.1:8080%0d%0a%0d%0a HTTP/1.13a8080%0d%0a%0d%0a%20 HTTP/
1.1
Host: 192.168.10.5
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.5672.93 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://192.168.10.7/
```

SSL Renegotiation Attacks

In order to find irregularities in handshakes, we can utilize TCP dump and Wireshark as we have done before. In order to filter to only handshake messages we can use this filter in Wireshark.

```
ssl.record.content_type == 22
```

The content type 22 specifies handshake messages only



When we are looking for SSL renegotiation attacks, we can look for the following.

1. `Multiple Client Hellos` - This is the most obvious sign of an SSL renegotiation attack. We will notice multiple client hellos from one client within a short period like above. The attacker repeats this message to trigger renegotiation and hopefully get a lower cipher suite.
2. `Out of Order Handshake Messages` - Simply put, sometimes we will see some out of order traffic due to packet loss and others, but in the case of SSL renegotiation some obvious signs would be the server receiving a client hello after completion of the handshake.

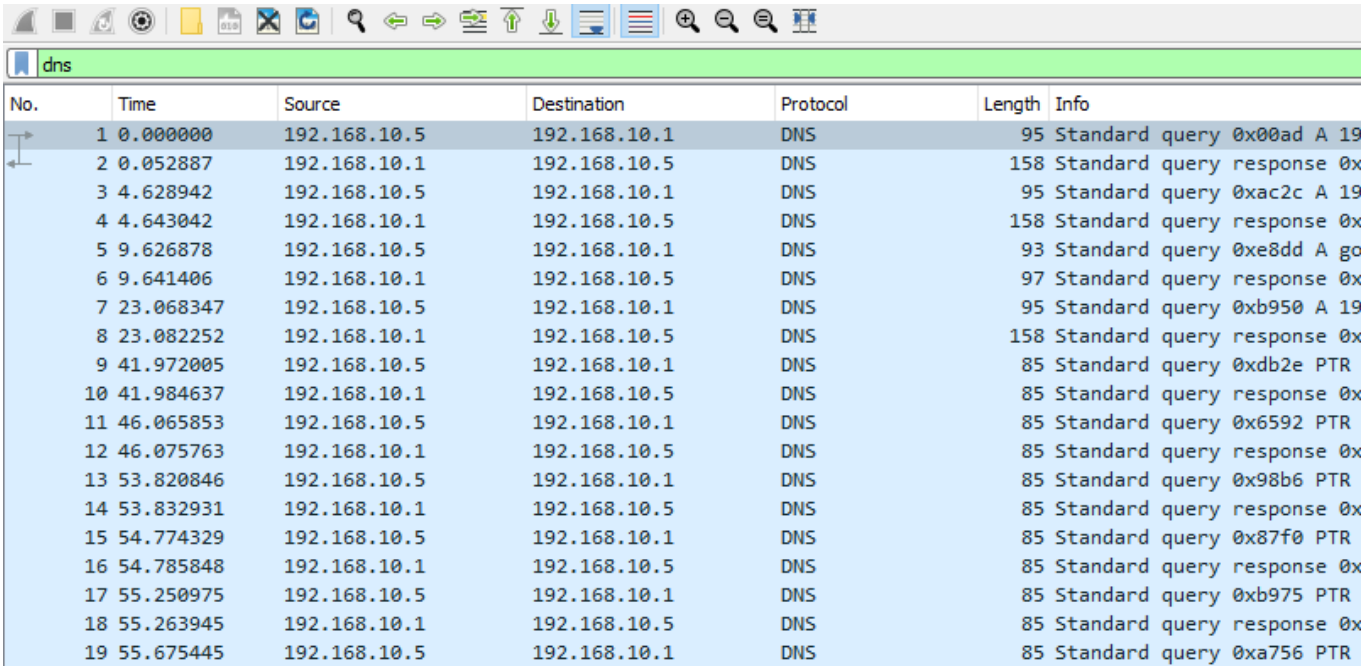
An attacker might conduct this attack against us for the following reasons

1. `Denial of Service` - SSL renegotiation attacks consume a ton of resources on the server side, and as such it might overwhelm the server and cause it to be unresponsive.
2. `SSL/TLS Weakness Exploitation` - The attacker might attempt renegotiation to potentially exploit vulnerabilities with our current implementation of cipher suites.
3. `Cryptanalysis` - The attacker might use renegotiation as a part of an overall strategy to analyze our SSL/TLS patterns for other systems.

DNS Enumeration Attempts

notice a significant amount of DNS traffic from one host

```
dns
```



dns

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.5	192.168.10.1	DNS	95	Standard query 0x00ad A 19
2	0.052887	192.168.10.1	192.168.10.5	DNS	158	Standard query response 0x
3	4.628942	192.168.10.5	192.168.10.1	DNS	95	Standard query 0xac2c A 19
4	4.643042	192.168.10.1	192.168.10.5	DNS	158	Standard query response 0x
5	9.626878	192.168.10.5	192.168.10.1	DNS	93	Standard query 0xe8dd A go
6	9.641406	192.168.10.1	192.168.10.5	DNS	97	Standard query response 0x
7	23.068347	192.168.10.5	192.168.10.1	DNS	95	Standard query 0xb950 A 19
8	23.082252	192.168.10.1	192.168.10.5	DNS	158	Standard query response 0x
9	41.972005	192.168.10.5	192.168.10.1	DNS	85	Standard query 0xdb2e PTR
10	41.984637	192.168.10.1	192.168.10.5	DNS	85	Standard query response 0x
11	46.065853	192.168.10.5	192.168.10.1	DNS	85	Standard query 0x6592 PTR
12	46.075763	192.168.10.1	192.168.10.5	DNS	85	Standard query response 0x
13	53.820846	192.168.10.5	192.168.10.1	DNS	85	Standard query 0x98b6 PTR
14	53.832931	192.168.10.1	192.168.10.5	DNS	85	Standard query response 0x
15	54.774329	192.168.10.5	192.168.10.1	DNS	85	Standard query 0x87f0 PTR
16	54.785848	192.168.10.1	192.168.10.5	DNS	85	Standard query response 0x
17	55.250975	192.168.10.5	192.168.10.1	DNS	85	Standard query 0xb975 PTR
18	55.263945	192.168.10.1	192.168.10.5	DNS	85	Standard query response 0x
19	55.675445	192.168.10.5	192.168.10.1	DNS	85	Standard query 0xa756 PTR

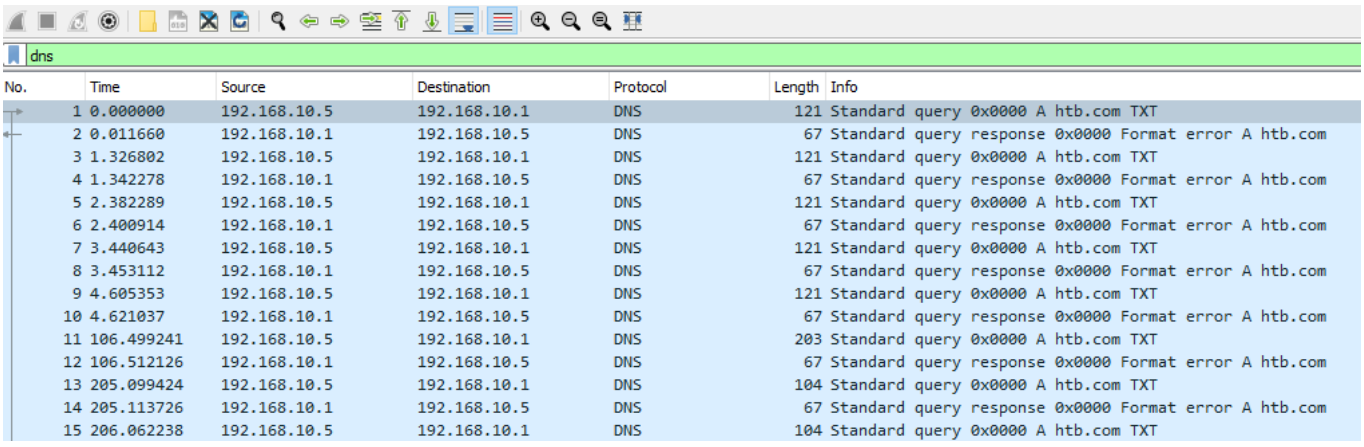
We might even notice this traffic concluded with something like `ANY`:

```
32 93.113760    192.168.10.5    192.168.10.1    DNS    121 Standard query 0x2e28 ANY 192.168.10.1 OPT
38 103.122114   192.168.10.5    192.168.10.1    DNS    121 Standard query 0x90c9 ANY 192.168.10.1 OPT
45 113.128223   192.168.10.5    192.168.10.1    DNS    121 Standard query 0x9c2a ANY 192.168.10.1 OPT
50 113.209771   192.168.10.1    192.168.10.5    DNS     97 Standard query response 0x2e28 Refused ANY 192.168.10.1
```

Finding DNS Tunneling

look for TXT record

`dns`



dns

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.5	192.168.10.1	DNS	121	Standard query 0x0000 A htb.com TXT
2	0.011660	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
3	1.326802	192.168.10.5	192.168.10.1	DNS	121	Standard query 0x0000 A htb.com TXT
4	1.342278	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
5	2.382289	192.168.10.5	192.168.10.1	DNS	121	Standard query 0x0000 A htb.com TXT
6	2.400914	192.168.10.1	192.168.10.5	DNS	67	Standard query response 0x0000 Format error A htb.com
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

```
Domain Name System (query)
    Transaction ID: 0x0000
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  > Queries
  v Answers
    v htb.com: type TXT, class IN
          Name: htb.com
          Type: TXT (Text strings) (16)
          Class: IN (0x0001)
          Time to live: 10 (10 seconds)
          Data length: 117
          TXT Length: 116
          TXT: VTBaU1EyVXhaSFprVjNocldETnNkbVJXT1cxaU0wb3pXVmhLYTFneU1XeFlNMUp2WVZoT1ptTklTbXhrU0ZJMVdETkNjMXBYUm5wYQpXREJMQ2c9PQo=
    [Response In: 12]
```

Traditional Telnet Traffic

Suppose we were to open Wireshark, we might notice some telnet communications originating from Port 23. In this case, we can always inspect this traffic further.

```
tcp.port == 23
```

```
> Frame 6: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface \Device\NPF_{CCC4B960-1E92-4BD5-BBF3-11E2DFD12FE1}, id 0
> Ethernet II, Src: PcsCompu_53:0c:ba (08:00:27:53:0c:ba), Dst: Micro-St_95:68:2a (44:8a:5b:95:68:2a)
> Internet Protocol Version 4, Src: 192.168.10.5, Dst: 192.168.10.7
> Transmission Control Protocol, Src Port: 59694, Dst Port: 23, Seq: 9, Ack: 1, Len: 62
v Telnet
    Data: telnet is unencrypted, so we can find things a little easier\r\n
```

IDS / IPS

Run suricate with rules, -l for dir location + -k none for none sig

```
sudo suricata -r /home/htb-student/pcaps/covenant.pcap -l . -k none
```

suricate rules dir

```
ls -la /etc/suricata/rules/
```

To load a cutome rule

```
nano /etc/suricata/suricata.yaml
```

Add rule.local to rule-files

Live Traffic

```
sudo  tcpreplay -i ens160 /home/htb-student/pcaps/suspicious.pcap
```

Suricata Rule Development Example 1: Detecting PowerShell Empire

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"ET MALWARE Possible
PowerShell Empire Activity Outbound"; flow:established,to_server;
content:"GET"; http_method; content:"/"; http_uri; depth:1;
pcre:"/^(?:login\/process|admin\/get|news)\.php$/RU"; content:"session=";
http_cookie; pcre:"/^(?:[A-Z0-9+/]{4})*(?:[A-Z0-9+/]{2}==|[A-Z0-9+/]{3}=|[A-
Z0-9+/]{4})$/CRi"; content:"Mozilla|2f|5.0|20 28|Windows|20|NT|20|6.1";
http_user_agent; http_start; content:".php|20|HTTP|2f|1.1|0d 0a|Cookie|3a
20|session="; fast_pattern; http_header_names; content:!"Referer";
content:!"Cache"; content:!"Accept"; sid:2027512; rev:1;)
```

Suricata Rule Development Example 2: Detecting Covenant

```
alert tcp any any -> $HOME_NET any (msg:"detected by body"; content:"
<title>Hello World!</title>"; detection_filter: track by_src, count 4 ,
seconds 10; priority:1; sid:3000011;)
```

Suricata Rule Development Example 3: Detecting Covenant (Using Analytics)

```
alert tcp $HOME_NET any -> any any (msg:"detected by size and counter";
dsize:312; detection_filter: track by_src, count 3 , seconds 10; priority:1;
sid:3000001;)
```

Suricata Rule Development Example 4: Detecting Sliver

```
alert tcp any any -> any any (msg:"Sliver C2 Implant Detected";
content:"POST";
pcre:"/\/(php|api|upload|actions|rest|v1|oauth2callback|authenticate|oauth2|
```

```
oauth|auth|database|db|namespaces)(.*?)
((login|signin|api|samples|rpc|index|admin|register|sign-up)\.php)\?[a-z_]
{1,2}=[a-z0-9]{1,10}/i"; sid:1000007; rev:1;)
```

Suricata Rule Development Example 5: Detecting Dridex (TLS Encrypted)

```
alert tls $EXTERNAL_NET any -> $HOME_NET any (msg:"ET MALWARE ABUSE.CH SSL
Blacklist Malicious SSL certificate detected (Dridex)";
flow:established,from_server; content:"|16|"; content:"|0b|"; within:8;
byte_test:3,<,1200,0,relative; content:"|03 02 01 02 02 09 00|";
fast_pattern; content:"|30 09 06 03 55 04 06 13 02|"; distance:0;
pcre:"/^[A-Z]{2}/R"; content:"|55 04 07|"; distance:0; content:"|55 04 0a|";
distance:0; pcre:"/^.{2}[A-Z][a-z]{3,}\s(?:[A-Z][a-z]{3,}\s)?(?:[A-Z](?:[A-
Za-z]{0,4}?[A-Z]|(?:\.[A-Za-z]){1,3})|[A-Z]?[a-z]+|[a-z](?:\.[A-Za-z])
{1,3})\.?[01]/Rs"; content:"|55 04 03|"; distance:0;
byte_test:1,>,13,1,relative; content:!"www."; distance:2; within:4;
pcre:"/^.{2}(?P<CN>(?:(?:\d?[A-Z]?|[A-Z]?\d?)(?:[a-z]{3,20}|[a-z]{3,6}[0-9_]
[a-z]{3,6})\.){0,2}?(?:\d?[A-Z]?|[A-Z]?\d?)[a-z]{3,}(?:[0-9_-][a-z]{3,})?\.
(?!com|org|net|tv)[a-z]{2,9})[01].*?(?P=CN)[01]/Rs"; content:!"|2a 86 48 86
f7 0d 01 09 01|"; content:!"GoDaddy"; sid:2023476; rev:5;)
```

Suricata Rule Development Example 6: Detecting Sliver (TLS Encrypted)

```
alert tls any any -> any any (msg:"Sliver C2 SSL"; ja3.hash;
content:"473cd7cb9faa642487833865d516e578"; sid:1002; rev:1;)
```

calc the https hash as follow

```
ja3 -a --json /home/htb-student/pcaps/sliverenc.pcap
```

Snort Rule Development Example 1: Detecting Ursnif (Inefficiently)

```
alert tcp any any -> any any (msg:"Possible Ursnif C2 Activity";
flow:established,to_server; content:"/images/", depth 12; content:"_2F";
content:"_2B"; content:"User-Agent|3a 20|Mozilla/4.0 (compatible|3b| MSIE
```

```
8.0|3b| Windows NT"; content:!"Accept"; content:!"Cookie|3a|";
content:!"Referer|3a|"; sid:1000002; rev:1;)
```

Snort Rule Development Example 2: Detecting Cerber

```
alert udp $HOME_NET any -> $EXTERNAL_NET any (msg:"Possible Cerber Check-
in"; dsize:9; content:"hi", depth 2, fast_pattern; pcre:"/^[af0-9]{7}$/R";
detection_filter:track by_src, count 1, seconds 60; sid:2816763; rev:4;)
```

Snort Rule Development Example 3: Detecting Patchwork

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"OISF TROJAN Targeted
AutoIt FileStealer/Downloader CnC Beacon"; flow:established,to_server;
http_method; content:"POST"; http_uri; content:".php?profile=";
http_client_body; content:"ddager=", depth 7; http_client_body;
content:"&r1=", distance 0; http_header; content:!"Accept"; http_header;
content:!"Referer|3a|"; sid:10000006; rev:1;)
```