Introduction

For my Cyber Attack I have chosen the TCP SYN flood. A SYN flood is a type of DoS (Denial of Service) attack that takes advantage of the TCP 3 way handshake. The TCP 3 way handshake is the process by which a TCP connection between a server and client is established (Hsu et al., 2016). It is engaged by the client with a SYN packet, the server receives the SYN packet and sends a SYN-ACK packet back, reserving its memory for establishing a TCP connection with the client. Finally the client sends an ACK packet back to the server, establishing a TCP connection where they can exchange data (Hsu et al., 2016).

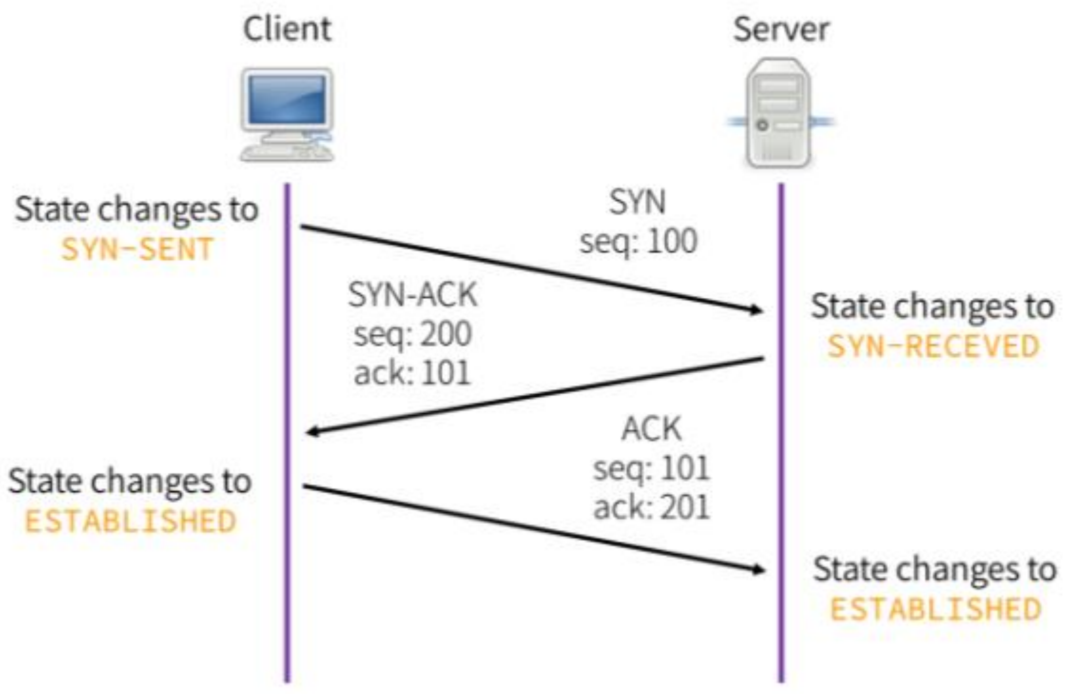
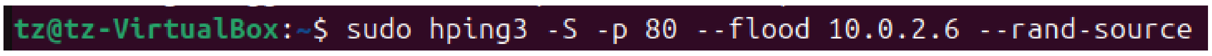

Client
Server
State changes to SYN-SENT
SYN seq: 100
SYN-ACK seq: 200 ack: 101
State changes to SYN-RECEVED
ACK seq: 101 ack: 201
State changes to ESTABLISHED
State changes to ESTABLISHED

Figure 1: TCP 3-Way-Handshake

(Hsu et al., 2016)

A SYN flood works by targeting the fact that after receiving a SYN packet, the client reserves resources in the wait for the ACK reply back. The server resources are limited and it can only reserve so many connections, and until the timeout occurs, nothing can be done with those reserved resources. If enough SYN packets are received without any replies, the servers resource limit is exceeded and it becomes unable to handle any new clients (Hsu et al., 2016). IPS have the ability to detect this by detecting an abnormal amount of SYN packets, tracking the amount of packets per second and then, if a certain threshold is passed, generating an alert. Snort IDS is ineffective at dealing with these attacks as it is very easy for the attacker to spoof their IP address, making each SYN packet appear as though it is from a different client. Since Snort IDS can only target protocols and IP addresses, the most the IDS can do is limit/block TCP traffic, impacting server availability and playing into the aim of the DOS attack when used by itself.

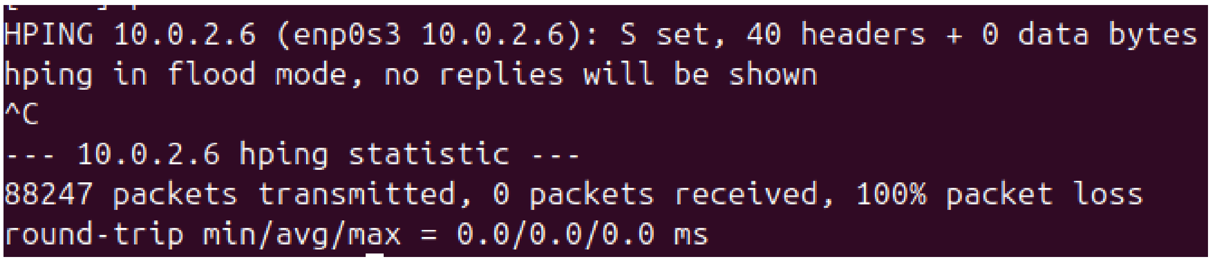
<u>Simulation</u>



```
tz@tz-VirtualBox:~$ sudo hping3 -S -p 80 --flood 10.0.2.6 --rand-source
```

Figure 2: Attack Command

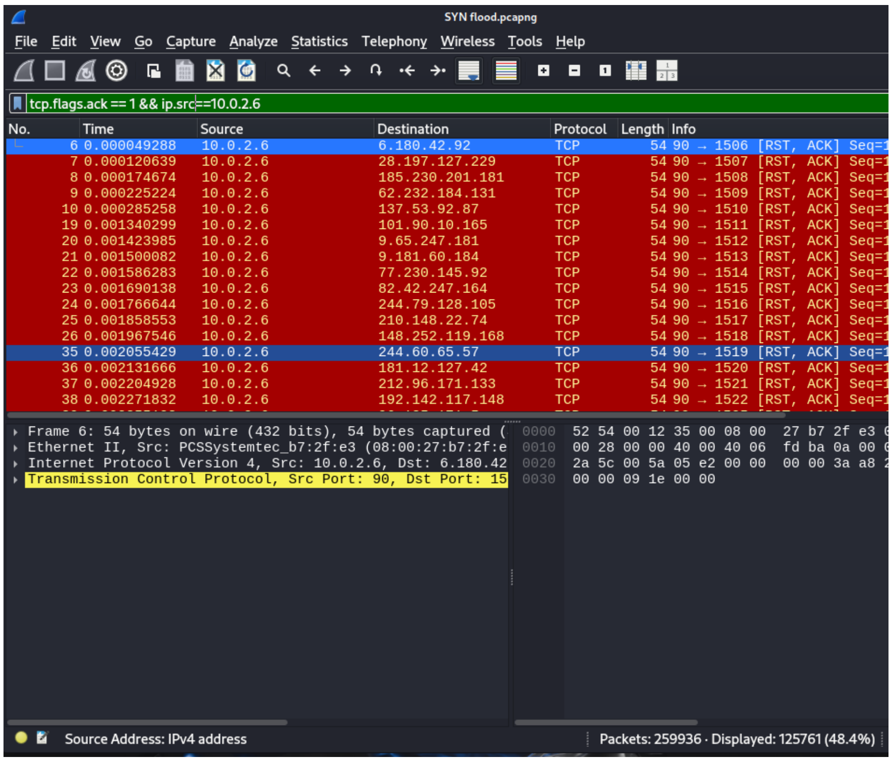
Hping3 is the tool being used, -S stands for SYN packets, -p stands for targeted port, 80 is the targeted port (HTTP), 10.0.2.6 is the destination ip, --rand-source spoofs the source IP.



```
HPING 10.0.2.6 (enp0s3 10.0.2.6): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.2.6 hping statistic ---
88247 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 3: Attack Command After Completion

If the attack is executed correctly, ratio of outgoing ACK packets to incoming ACK packets should be heavily skewed to outgoing showing lots of incoming SYN packets with no reply.


SYN flood.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.flags.ack == 1 && ip.src==10.0.2.6

No.  Time         Source     Destination      Protocol Length Info
   6 0.000049288  10.0.2.6   6.180.42.92      TCP      54 90 → 1506 [RST, ACK] Seq=1
   7 0.000120639  10.0.2.6   28.197.127.229   TCP      54 90 → 1507 [RST, ACK] Seq=1
   8 0.000174674  10.0.2.6   185.230.201.181  TCP      54 90 → 1508 [RST, ACK] Seq=1
   9 0.000225224  10.0.2.6   62.232.184.131   TCP      54 90 → 1509 [RST, ACK] Seq=1
  10 0.000285258  10.0.2.6   137.53.92.87     TCP      54 90 → 1510 [RST, ACK] Seq=1
  19 0.001340299  10.0.2.6   101.90.10.165    TCP      54 90 → 1511 [RST, ACK] Seq=1
  20 0.001423985  10.0.2.6   9.65.247.181     TCP      54 90 → 1512 [RST, ACK] Seq=1
  21 0.001500082  10.0.2.6   9.181.60.184     TCP      54 90 → 1513 [RST, ACK] Seq=1
  22 0.001586283  10.0.2.6   77.230.145.92    TCP      54 90 → 1514 [RST, ACK] Seq=1
  23 0.001690138  10.0.2.6   82.42.247.164    TCP      54 90 → 1515 [RST, ACK] Seq=1
  24 0.001766644  10.0.2.6   244.79.128.105   TCP      54 90 → 1516 [RST, ACK] Seq=1
  25 0.001858553  10.0.2.6   210.148.22.74    TCP      54 90 → 1517 [RST, ACK] Seq=1
  26 0.001967546  10.0.2.6   148.252.119.168  TCP      54 90 → 1518 [RST, ACK] Seq=1
  35 0.002055429  10.0.2.6   244.60.65.57     TCP      54 90 → 1519 [RST, ACK] Seq=1
  36 0.002131666  10.0.2.6   181.12.127.42    TCP      54 90 → 1520 [RST, ACK] Seq=1
  37 0.002204928  10.0.2.6   212.96.171.133   TCP      54 90 → 1521 [RST, ACK] Seq=1
  38 0.002271832  10.0.2.6   192.142.117.148  TCP      54 90 → 1522 [RST, ACK] Seq=1

▸ Frame 6: 54 bytes on wire (432 bits), 54 bytes captured (    0000  52 54 00 12 35 00 08 00  27 b7 2f e3
▸ Ethernet II, Src: PCSSystemtec_b7:2f:e3 (08:00:27:b7:2f:e    0010  00 28 00 00 40 00 40 06  fd ba 0a 00
▸ Internet Protocol Version 4, Src: 10.0.2.6, Dst: 6.180.42    0020  2a 5c 00 5a 05 e2 00 00  00 00 3a a8
▸ Transmission Control Protocol, Src Port: 90, Dst Port: 15    0030  00 00 09 1e 00 00

Source Address: IPv4 address                    Packets: 259936 · Displayed: 125761 (48.4%)
```

Figure 4: Outgoing ACK packets

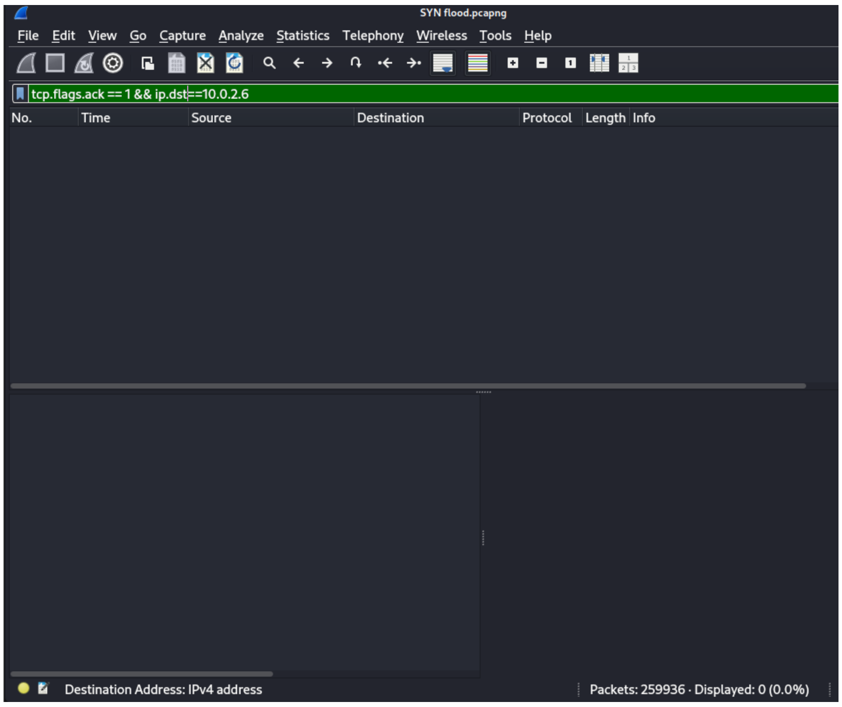
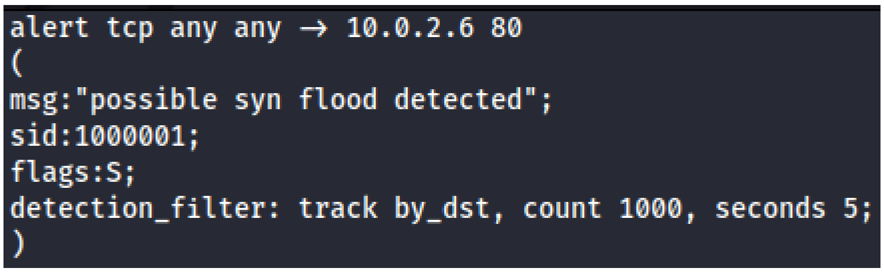

SYN flood.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.flags.ack == 1 && ip.dst==10.0.2.6
No. Time Source Destination Protocol Length Info
Destination Address: IPv4 address
Packets: 259936 · Displayed: 0 (0.0%)

Figure 5: Incoming ACK packets

The ratio of outgoing ACK to incoming ACK is 1:0, showing this is clearly a SYN flood attack.

<u>IDS Rules</u>

To detect a SYN flood I used the snort rule



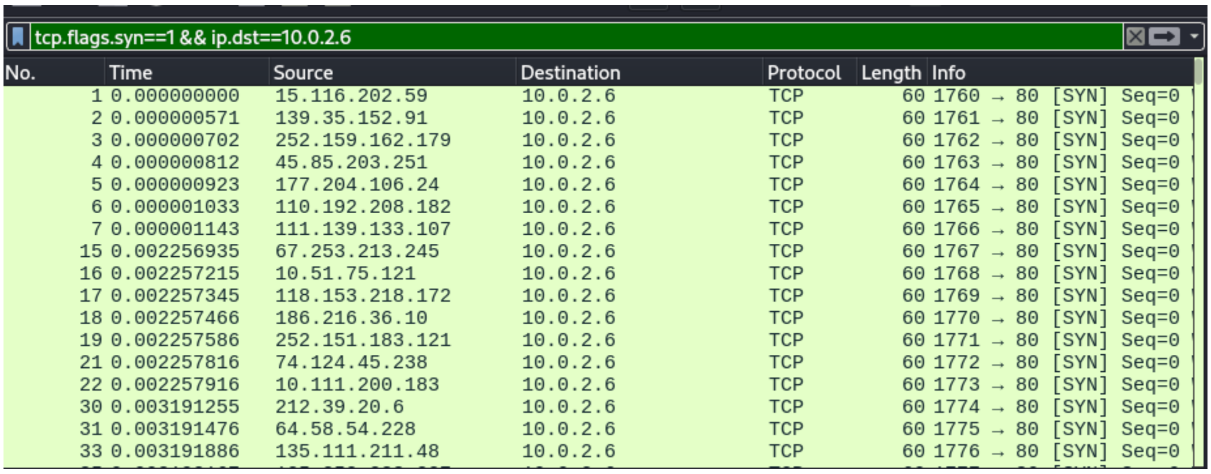
```
alert tcp any any → 10.0.2.6 80
(
msg:"possible syn flood detected";
sid:1000001;
flags:S;
detection_filter: track by_dst, count 1000, seconds 5;
)
```

Figure 6: Snort Rule

This snort rules issues an alert for the TCP protocol when any source and any protocol going towards the destination 10.0.2.6 and port 80 (http). When issued the message "possible syn flood detected" is outputted and the rule ID is 1000001. It is triggered when SYN packets specifically surpass the count of 1000 within 5 seconds, all towards the same destination of 10.0.2.6

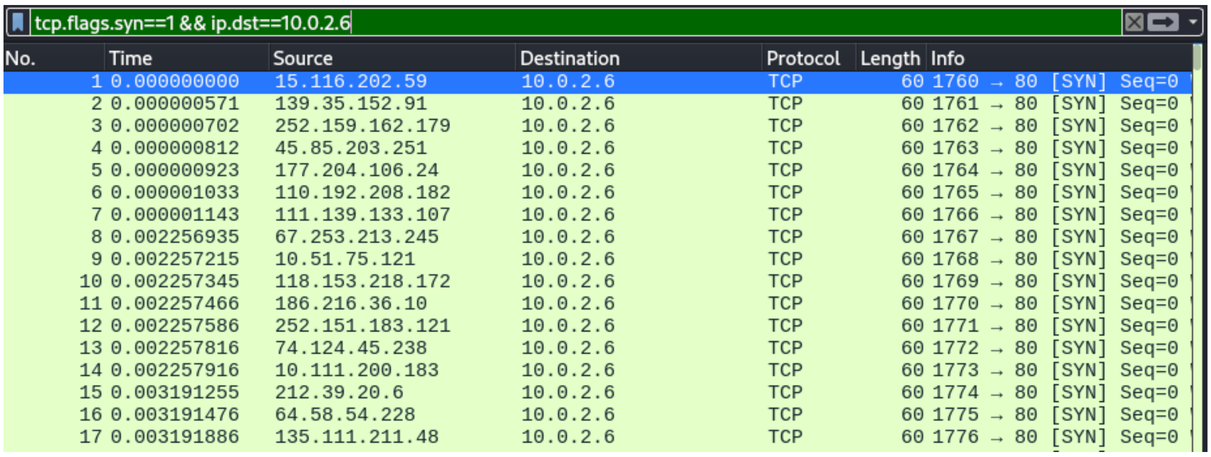
<u>Validation</u>

I could not get the pcap generation for snort working so I had to compare the console output and Wireshark output manually. I did this by exporting all the packets that fit the filter tcp.flag.syn==1 && ip.dst==10.0.2.6 as well as copy pasting all of the snort rule outputs from the console into a text file to make it easy to search it. Exporting the SYN packets is needed because without doing this, the packet number becomes meaningless without it as it includes packets from the host being sent out as we can see here.



```
tcp.flags.syn==1 && ip.dst==10.0.2.6
No.  Time         Source            Destination  Protocol  Length  Info
   1 0.000000000  15.116.202.59     10.0.2.6     TCP       60 1760 → 80 [SYN] Seq=0
   2 0.000000571  139.35.152.91     10.0.2.6     TCP       60 1761 → 80 [SYN] Seq=0
   3 0.000000702  252.159.162.179   10.0.2.6     TCP       60 1762 → 80 [SYN] Seq=0
   4 0.000000812  45.85.203.251     10.0.2.6     TCP       60 1763 → 80 [SYN] Seq=0
   5 0.000000923  177.204.106.24    10.0.2.6     TCP       60 1764 → 80 [SYN] Seq=0
   6 0.000001033  110.192.208.182   10.0.2.6     TCP       60 1765 → 80 [SYN] Seq=0
   7 0.000001143  111.139.133.107   10.0.2.6     TCP       60 1766 → 80 [SYN] Seq=0
  15 0.002256935  67.253.213.245    10.0.2.6     TCP       60 1767 → 80 [SYN] Seq=0
  16 0.002257215  10.51.75.121      10.0.2.6     TCP       60 1768 → 80 [SYN] Seq=0
  17 0.002257345  118.153.218.172   10.0.2.6     TCP       60 1769 → 80 [SYN] Seq=0
  18 0.002257466  186.216.36.10     10.0.2.6     TCP       60 1770 → 80 [SYN] Seq=0
  19 0.002257586  252.151.183.121   10.0.2.6     TCP       60 1771 → 80 [SYN] Seq=0
  21 0.002257816  74.124.45.238     10.0.2.6     TCP       60 1772 → 80 [SYN] Seq=0
  22 0.002257916  10.111.200.183    10.0.2.6     TCP       60 1773 → 80 [SYN] Seq=0
  30 0.003191255  212.39.20.6       10.0.2.6     TCP       60 1774 → 80 [SYN] Seq=0
  31 0.003191476  64.58.54.228      10.0.2.6     TCP       60 1775 → 80 [SYN] Seq=0
  33 0.003191886  135.111.211.48    10.0.2.6     TCP       60 1776 → 80 [SYN] Seq=0
```

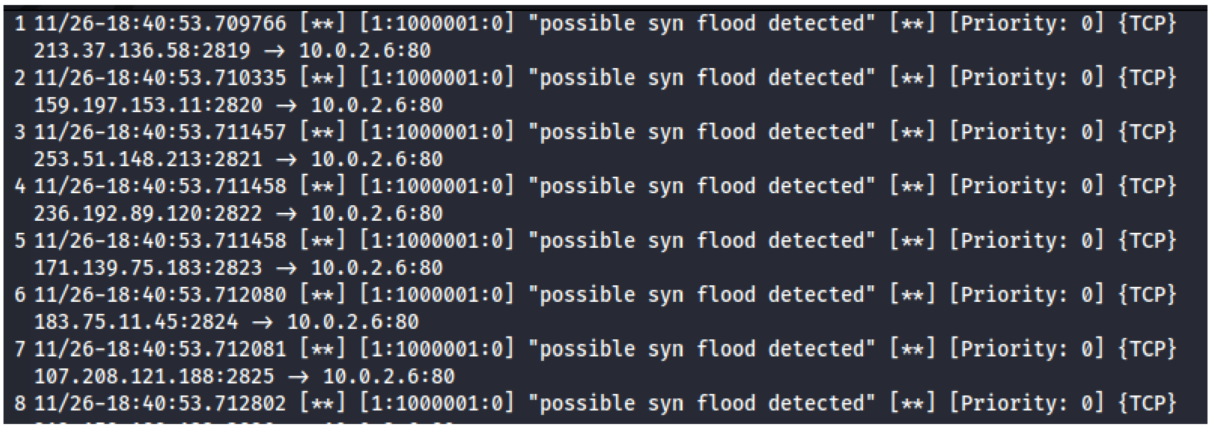
Figure 7: We can see the SYN packets have sudden big jumps in packet number



```
tcp.flags.syn==1 && ip.dst==10.0.2.6
No.  Time         Source            Destination  Protocol  Length  Info
   1 0.000000000  15.116.202.59     10.0.2.6     TCP       60 1760 → 80 [SYN] Seq=0
   2 0.000000571  139.35.152.91     10.0.2.6     TCP       60 1761 → 80 [SYN] Seq=0
   3 0.000000702  252.159.162.179   10.0.2.6     TCP       60 1762 → 80 [SYN] Seq=0
   4 0.000000812  45.85.203.251     10.0.2.6     TCP       60 1763 → 80 [SYN] Seq=0
   5 0.000000923  177.204.106.24    10.0.2.6     TCP       60 1764 → 80 [SYN] Seq=0
   6 0.000001033  110.192.208.182   10.0.2.6     TCP       60 1765 → 80 [SYN] Seq=0
   7 0.000001143  111.139.133.107   10.0.2.6     TCP       60 1766 → 80 [SYN] Seq=0
   8 0.002256935  67.253.213.245    10.0.2.6     TCP       60 1767 → 80 [SYN] Seq=0
   9 0.002257215  10.51.75.121      10.0.2.6     TCP       60 1768 → 80 [SYN] Seq=0
  10 0.002257345  118.153.218.172   10.0.2.6     TCP       60 1769 → 80 [SYN] Seq=0
  11 0.002257466  186.216.36.10     10.0.2.6     TCP       60 1770 → 80 [SYN] Seq=0
  12 0.002257586  252.151.183.121   10.0.2.6     TCP       60 1771 → 80 [SYN] Seq=0
  13 0.002257816  74.124.45.238     10.0.2.6     TCP       60 1772 → 80 [SYN] Seq=0
  14 0.002257916  10.111.200.183    10.0.2.6     TCP       60 1773 → 80 [SYN] Seq=0
  15 0.003191255  212.39.20.6       10.0.2.6     TCP       60 1774 → 80 [SYN] Seq=0
  16 0.003191476  64.58.54.228      10.0.2.6     TCP       60 1775 → 80 [SYN] Seq=0
  17 0.003191886  135.111.211.48    10.0.2.6     TCP       60 1776 → 80 [SYN] Seq=0
```

Figure 8:The jumps are now gone

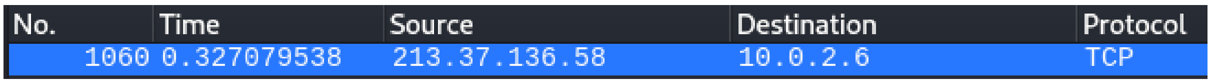
First of all, we can see that the rule is clearly being triggered



```
1 11/26-18:40:53.709766 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  213.37.136.58:2819 -> 10.0.2.6:80
2 11/26-18:40:53.710335 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  159.197.153.11:2820 -> 10.0.2.6:80
3 11/26-18:40:53.711457 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  253.51.148.213:2821 -> 10.0.2.6:80
4 11/26-18:40:53.711458 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  236.192.89.120:2822 -> 10.0.2.6:80
5 11/26-18:40:53.711458 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  171.139.75.183:2823 -> 10.0.2.6:80
6 11/26-18:40:53.712080 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  183.75.11.45:2824 -> 10.0.2.6:80
7 11/26-18:40:53.712081 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
  107.208.121.188:2825 -> 10.0.2.6:80
8 11/26-18:40:53.712802 [**] [1:1000001:0] "possible syn flood detected" [**] [Priority: 0] {TCP}
```

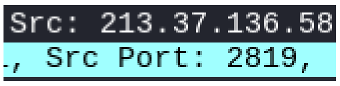
Figure 9: Snort rule being triggered

If the rule has worked correctly, the first snort rule trigger should be approximately the 1000th SYN packet, as we can see from the above image, the IP address of the first rule trigger is 213.37.136.58 with the port number 2819. Searching the exported packets with this IP address returns this



No.	Time	Source	Destination	Protocol
1060	0.327079538	213.37.136.58	10.0.2.6	TCP

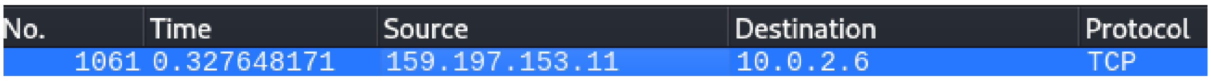
With this port



```
Src: 213.37.136.58
., Src Port: 2819,
```

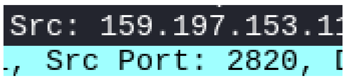
Clearly showing a match. The time is also clearly under 5 seconds, while the packet number might not match 1000 exactly it is still close enough to show that the rule is functioning.

Checking frame number 1061 gives us this packet



No.	Time	Source	Destination	Protocol
1061	0.327648171	159.197.153.11	10.0.2.6	TCP

With this port



```
Src: 159.197.153.1
., Src Port: 2820,
```

Matching the second snort rule hit. This shows the rule keeps hitting every time there is a SYN packet if the threshold within the time limit is hit, which is the expected behaviour.