Name: Yikwenmein Victor Magheng
Email address:yikwenmeinvictor1995@gmail.com
Personal code number:19950218-T614

Course: Network and System Security(ET1595)

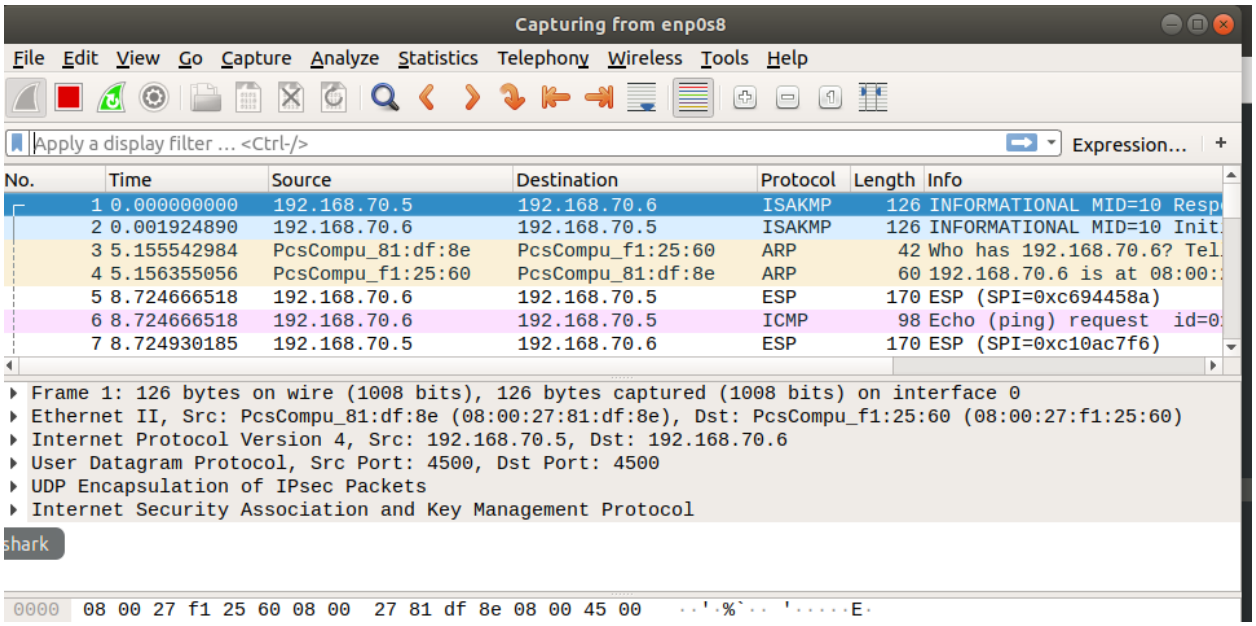
Lab_3: IDS

Task 1: Check connectivity

For this task, after installing the basic software (Metasploit on server-B and Snort on server-A), I started both servers and pinged server-A from server-B

```
student@serverB:~$ ping 192.168.70.5
PING 192.168.70.5 (192.168.70.5) 56(84) bytes of data.
64 bytes from 192.168.70.5: icmp_seq=1 ttl=64 time=0.437 ms
64 bytes from 192.168.70.5: icmp_seq=2 ttl=64 time=1.73 ms
64 bytes from 192.168.70.5: icmp_seq=3 ttl=64 time=1.47 ms
64 bytes from 192.168.70.5: icmp_seq=4 ttl=64 time=1.83 ms
^C
--- 192.168.70.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 0.437/1.369/1.832/0.554 ms
student@serverB:~$
```

On wireshark, we observe the following on interface enp0s8



Task 2: Detect incoming pings

For this task, I initially added the following rule

`alert icmp 192.168.70.6 192.168.70.5 (msg:"ICMP message detected"; sid:2000001)` to `/etc/snort/rules/local.rules` on Server A as on the lab guide but it raises the following error

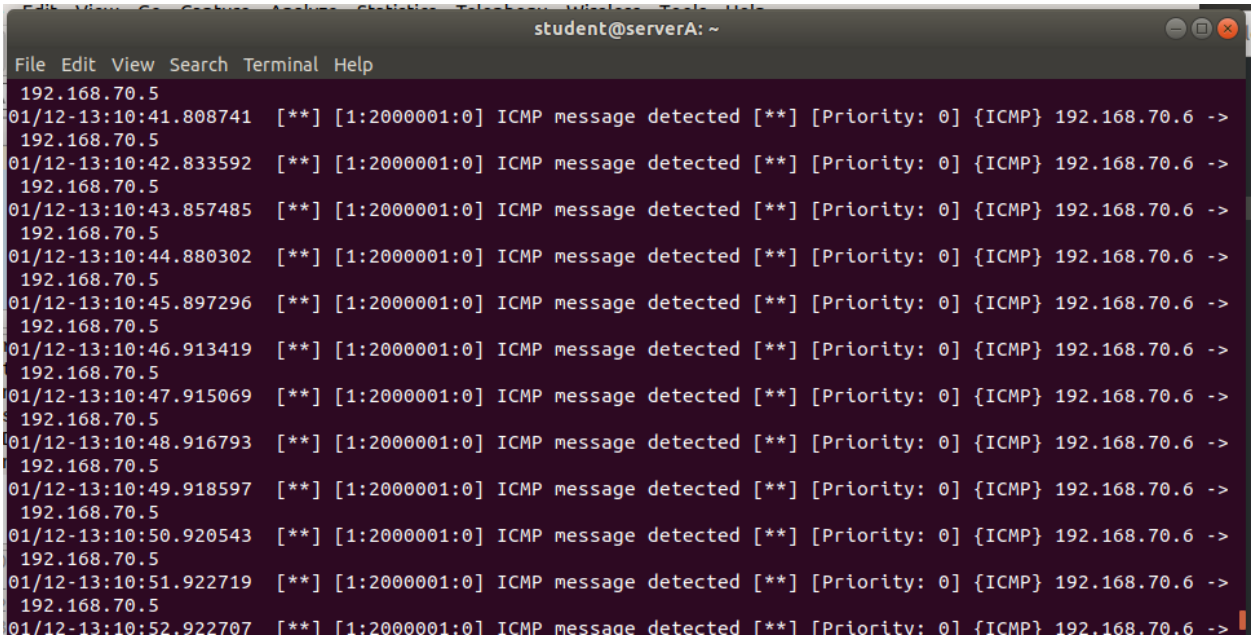
```
++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
ERROR: /etc/snort/rules/local.rules(7) Illegal direction specifier:
(msg:"ICMP
Fatal Error, Quitting..
```

When i execute `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf`

I then resorted to

```
alert icmp 192.168.70.6 any -> 192.168.70.5 any (msg:"ICMP message
detected"; sid:2000001)
```

and `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf` went successful and I could see matching alerts from snort on Server A after pinging 192.168.70.5 from server B, matching alerts ason the following screenshot were observed.



Task 3: Detect TCP port scanning

To get this done, I started both servers A and B. On server B,I ran `sudo msfconsole` To get metasploit started. The next set of commands were

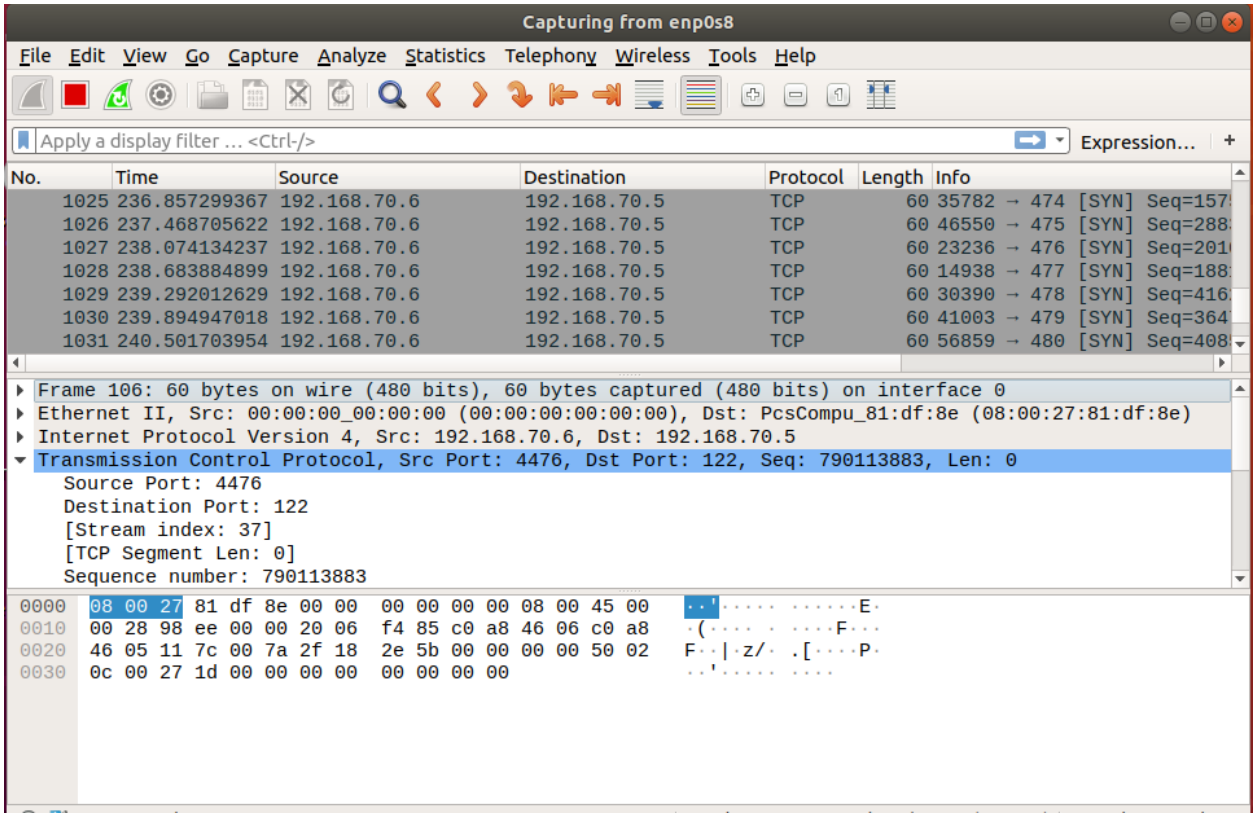
```
use auxiliary/scanner/portscan/syn
```

```
set RHOSTS 192.168.70.5
set INTERFACE enp0s8
set PORTS 1-500
```

Executing `run` showed the following open ports on server A(192.168.70.5)

```
msf6 auxiliary(scanner/portscan/syn) > run

[+]  TCP OPEN 192.168.70.5:22
[+]  TCP OPEN 192.168.70.5:80
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/syn) >
```

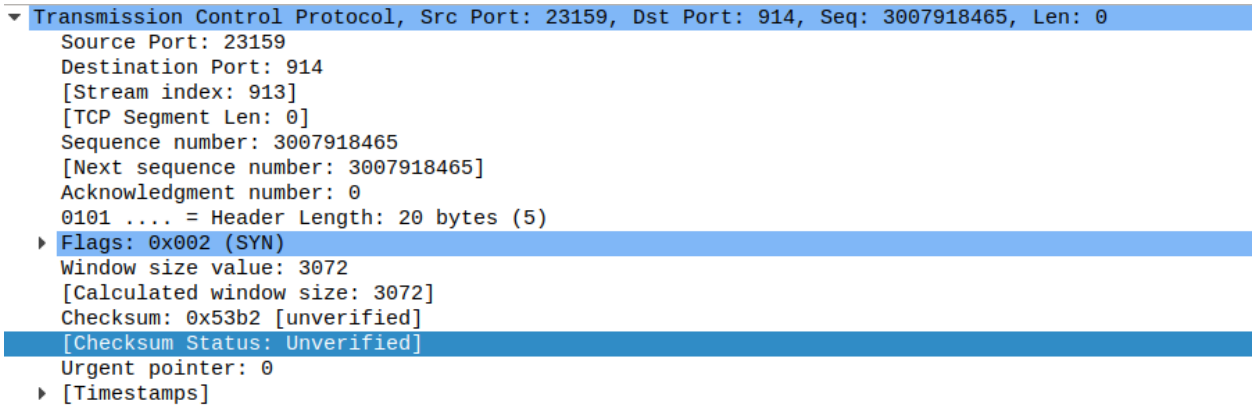
Checking through wireshark on server A, pairs of TCP packets – a SYN request from Server B to Server A and replies are observed.



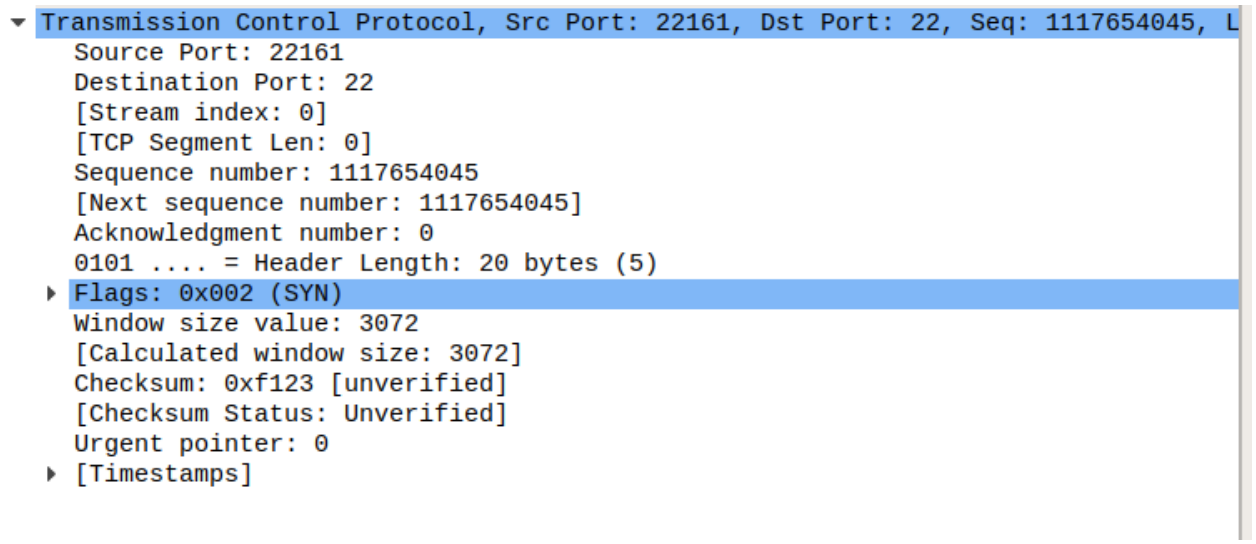
As on the previous screenshot, the open ports found on server A are

1. port `22` and it's service is `ssh`
2. `Port 80 used for HTTP in the world wide web`

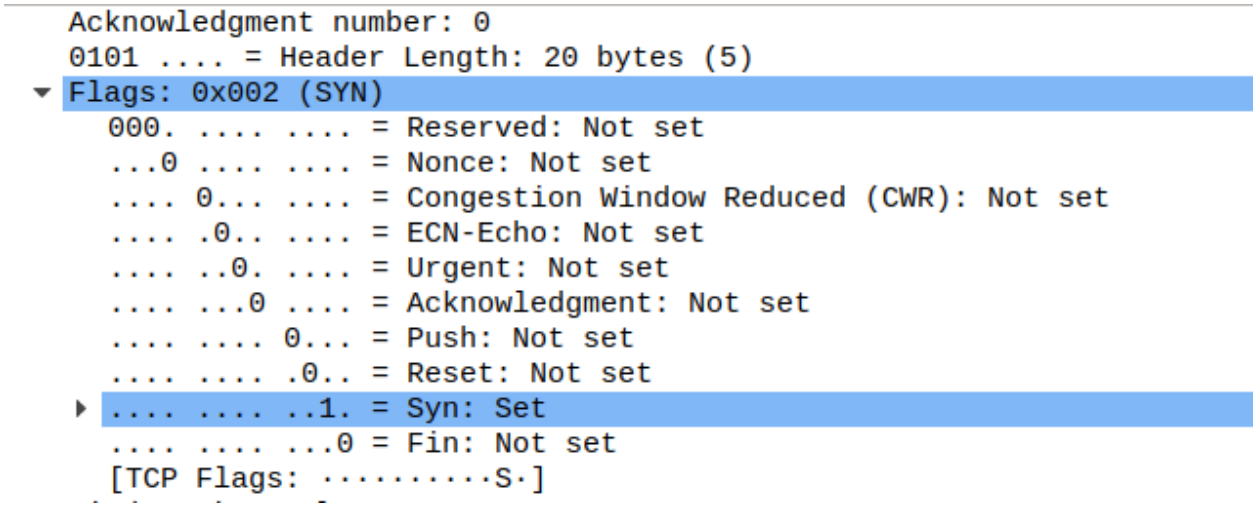
A TCP header for random close port is as follows



The TCP header for an open port(22) is as shown



My observation hasn't seen any big difference between the two TCP headers. Even the flags are the same(all bits aren't set except for the Syn bit)



In this type of scanning(SYN scanning), the hostile client attempts to set up a TCP/IP connection with a server at every possible port. This is done by sending a SYN (synchronization) packet, as if to initiate a three-way handshake, to every port on the server. If the server responds with a SYN/ACK (synchronization acknowledged) packet from a particular port, it means the port is open. Then the hostile client sends an RST (reset) packet. As a result, the server assumes that there has been a communications error, and that the client has decided not to establish a connection. The open port nevertheless remains open and vulnerable to exploitation. If the server responds with an RST (reset) packet from a particular port, it indicates that the port is closed and cannot be exploited.

On server B, `telnet 192.168.70.5` returns immediately with the following output

```
Trying 192.168.70.5...
telnet: Unable to connect to remote host: Connection refused
```

Comparing TCP headers for scanning and Telnet packets.

Comparing TCP headers from the telnet packets going towards Server A to the TCP headers from the scanning packets going in the same direction we see that Scanning packets all have 0 as an acknowledge number while for Telnet packets, the acknowledgement number is set and is not zero. For example, the acknowledgment number is 613435349. Scanning packets , only the SYN flag is set while for telnet packets, we have both ACK and PSH flags set. The IP header for scanning packets has a TTL of 32 while that for the telnet packets is 64. The screenshot below depicts headers for port scanning packets

```
▾ Internet Protocol Version 4, Src: 192.168.70.6, Dst: 192.168.70.5
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x07ed (2029)
  ▸ Flags: 0x0000
    Time to live: 32
    Protocol: TCP (6)
    Header checksum: 0x8587 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.70.6
    Destination: 192.168.70.5
▾ Transmission Control Protocol, Src Port: 62745, Dst Port: 496, Seq: 1018700379, Len: 0
    Source Port: 62745
    Destination Port: 496
    [Stream index: 495]
    [TCP Segment Len: 0]
    Sequence number: 1018700379
    [Next sequence number: 1018700379]
    Acknowledgment number: 0
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x002 (SYN)
    Window size value: 3072
    [Calculated window size: 3072]
    Checksum: 0x4069 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ [Timestamps]
```

The screenshot below depicts the header for telnet packets.

```
▸ Frame 7: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_f1:25:60 (08:00:27:f1:25:60), Dst: PcsCompu_81:df:8e (08:00:27:81:df:8e)
▾ Internet Protocol Version 4, Src: 192.168.70.6, Dst: 192.168.70.5
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 93
    Identification: 0x8291 (33425)
  ▸ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xaaad [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.70.6
    Destination: 192.168.70.5
▾ Transmission Control Protocol, Src Port: 22, Dst Port: 36202, Seq: 3597188789, Ack: 3428659232, Len: 41
    Source Port: 22
    Destination Port: 36202
    [Stream index: 0]
    [TCP Segment Len: 41]
    Sequence number: 3597188789
    [Next sequence number: 3597188830]
    Acknowledgment number: 3428659232
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
```

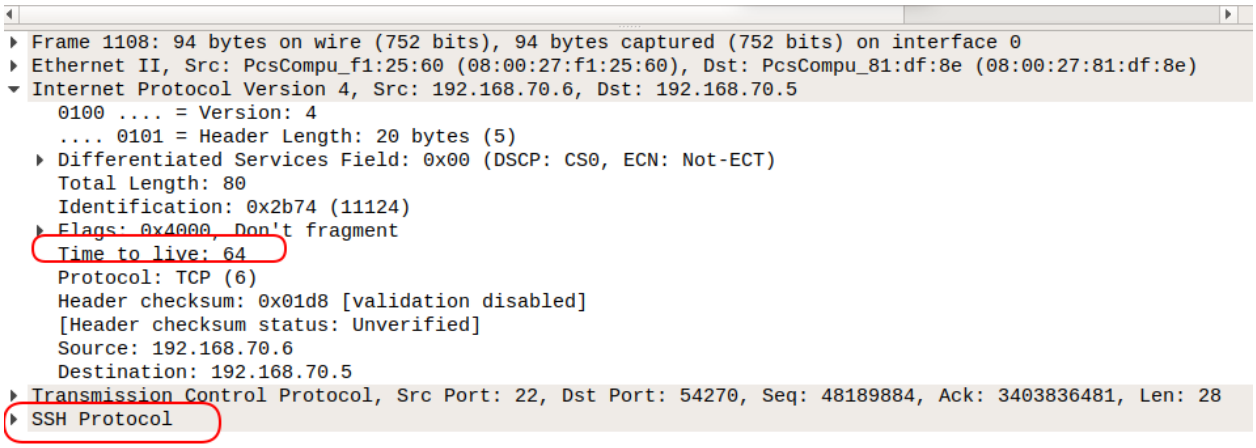
```
    Window size value: 227
    [Calculated window size: 29056]
    [Window size scaling factor: 128]
    Checksum: 0x6257 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (41 bytes)
▸ SSH Protocol
```

Confirming with SSH,

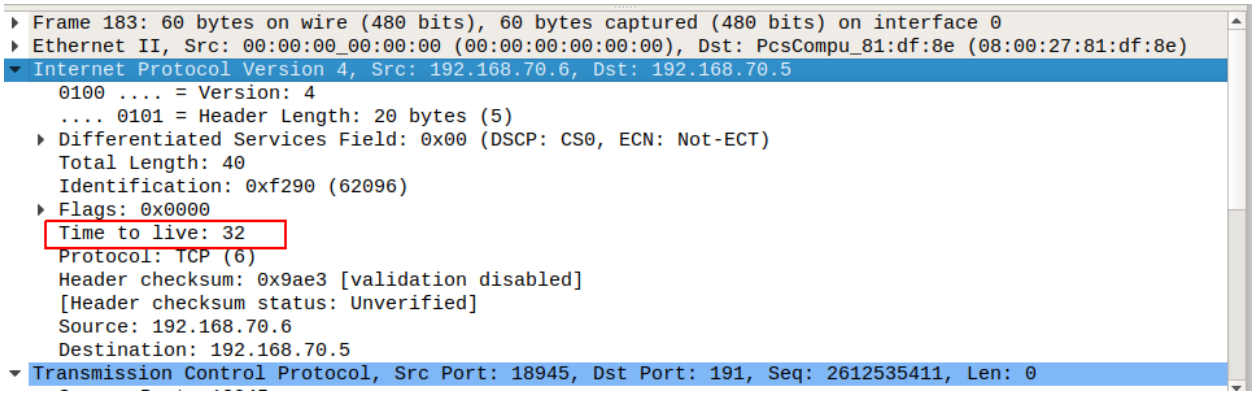
```
student@serverB:~$ ssh 192.168.70.5
The authenticity of host '192.168.70.5 (192.168.70.5)' can't be established.
ECDSA key fingerprint is SHA256:QSfKuPhpg3hiOu85R2Dp/FlndnSsfrbJIzASBpqXdWY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.70.5' (ECDSA) to the list of known hosts.
student@192.168.70.5's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Introducing self-healing high availability clusters in MicroK8s.
   Simple, hardened, Kubernetes for production, from RaspberryPi to DC.

     https://microk8s.io/high-availability

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

642 packages can be updated.
454 updates are security updates.

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Nov 22 17:09:36 2020 from 192.168.60.111
student@serverA:~$ exit
logout
Connection to 192.168.70.5 closed.
student@serverB:~$ telnet 192.168.70.5 22
```

```
▾ Internet Protocol Version 4, Src: 192.168.70.6, Dst: 192.168.70.5
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 80
    Identification: 0xf223 (61987)
  ▸ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0x3b28 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.70.6
    Destination: 192.168.70.5
▾ Transmission Control Protocol, Src Port: 22, Dst Port: 36224, Seq: 2947405957, Ack: 1376781616, Len: 28
    Source Port: 22
    Destination Port: 36224
    [Stream index: 0]
    [TCP Segment Len: 28]
    Sequence number: 2947405957
    [Next sequence number: 2947405985]
    Acknowledgment number: 1376781616
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 271
    [Calculated window size: 34688]
```

We can as well observe a TTL of 64 and **PSH,ACK** flags set.



For scanning packets, the TCP header has already been shown above(comparing TCP headers for closed and open ports). The IP header for port scanning is as follows



The conclusion that can be drawn is that TCP header for port scanning packets don’t have acknowledgement numbers (has 0 as acknowledgement number) hence flag bit for acknowledgement number is not set while those for benign traffic have acknowledgement numbers and flag bit for acknowledgement number is set to 1. The IP header for port scanning packets have TTL value of 32 while those of benign traffic have a TTL value of 64.

snort rule for port scanning

```
alert tcp 192.168.70.6 any -> 192.168.70.5 any (flags:S; ttl:=32;
msg:"Detecting TCP port scanning"; sid:2000002)
```

After executing `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf`, the results are as follows

```
05/09-00:01:31.313196  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:7392 -> 192.168.70.5:1
05/09-00:01:31.926088  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:44623 -> 192.168.70.5:2
05/09-00:01:32.534977  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:63973 -> 192.168.70.5:3
05/09-00:01:33.146380  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:16628 -> 192.168.70.5:4
05/09-00:01:33.754080  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:43840 -> 192.168.70.5:5
05/09-00:01:34.365318  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:9610 -> 192.168.70.5:6
05/09-00:01:34.977468  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:19895 -> 192.168.70.5:7
05/09-00:01:35.604546  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:53379 -> 192.168.70.5:8
05/09-00:01:36.220090  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:55442 -> 192.168.70.5:9
05/09-00:01:36.825692  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:28724 -> 192.168.70.5:10
05/09-00:01:37.432073  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:57791 -> 192.168.70.5:11
05/09-00:01:38.038457  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:4001 -> 192.168.70.5:12
05/09-00:01:38.644371  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:28586 -> 192.168.70.5:13
05/09-00:01:39.252066  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:26187 -> 192.168.70.5:14
05/09-00:01:39.858936  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:14990 -> 192.168.70.5:15
05/09-00:01:40.469602  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:30674 -> 192.168.70.5:16
05/09-00:01:41.078715  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:8081 -> 192.168.70.5:17
05/09-00:01:41.687407  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:57857 -> 192.168.70.5:18
05/09-00:01:42.302476  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:50511 -> 192.168.70.5:19
05/09-00:01:42.922680  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:36108 -> 192.168.70.5:20
05/09-00:01:43.532545  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:21346 -> 192.168.70.5:21
05/09-00:01:44.146153  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:46373 -> 192.168.70.5:22
05/09-00:01:44.755363  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:16600 -> 192.168.70.5:23
05/09-00:01:45.363415  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:4612 -> 192.168.70.5:24
05/09-00:01:45.980434  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:24392 -> 192.168.70.5:25
05/09-00:01:46.593771  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:3729 -> 192.168.70.5:26
05/09-00:01:47.206569  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:25022 -> 192.168.70.5:27
05/09-00:01:47.821549  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:43690 -> 192.168.70.5:28
05/09-00:01:48.432382  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:5031 -> 192.168.70.5:29
05/09-00:01:49.044182  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:24096 -> 192.168.70.5:30
05/09-00:01:49.649866  [**] [1:2000002:0] Detecting TCP port scanning [**] [Priority: 0] {TCP} 192.168.70.6:14468 -> 192.168.70.5:31
Show Applications
```

To verify that the rule does not generate alerts for benign traffic(telnet and ssh),I interrupted port scanning with ctrl+c and then run telnet 192.168.70.5 and/or ssh 192.168.70.5 and at this time, we don't see alerts as we have on the previous screenshot. The terminal looks like this

```
 3] {UDP} 192.168.70.1:36468 -> 239.255.255.250:1900
05/09-10:53:54.350090  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:36468 -> 239.255.255.250:1900
05/09-10:53:55.352850  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:36468 -> 239.255.255.250:1900
05/09-10:53:56.356021  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:36468 -> 239.255.255.250:1900
05/09-10:55:53.432946  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:52482 -> 239.255.255.250:1900
05/09-10:55:54.434638  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:52482 -> 239.255.255.250:1900
05/09-10:55:55.436514  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:52482 -> 239.255.255.250:1900
05/09-10:55:56.438199  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:52482 -> 239.255.255.250:1900
05/09-10:57:53.497068  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:35289 -> 239.255.255.250:1900
05/09-10:57:54.498353  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:35289 -> 239.255.255.250:1900
05/09-10:57:55.499269  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:35289 -> 239.255.255.250:1900
05/09-10:57:56.501076  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:35289 -> 239.255.255.250:1900
05/09-10:59:53.560855  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:58658 -> 239.255.255.250:1900
05/09-10:59:54.562420  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:58658 -> 239.255.255.250:1900
05/09-10:59:55.563423  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:58658 -> 239.255.255.250:1900
05/09-10:59:56.564372  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [Classification: Detection of a Network Scan] [Priority:
 3] {UDP} 192.168.70.1:58658 -> 239.255.255.250:1900
```

Explanation of the snort rule used.

Rule headers

- alert: Rule action. Snort will generate an alert when the set condition is met.
- Tcp: Signifies the TCP/IP protocol that the rule must match
- 192.168.70.6: specifies the source IP for snort to look at.
- any: any is for the source port. Snort will look at all ports
- -> : Specifies the direction. From source to destination
- 192.168.70.5: specifies the destination IP address.
- any: Destination port. Snort will look at all ports on the protected network.

Rule options:

- msg:"Detecting TCP port scanning" is a message with the alert.
- flags:S - The "flags" keyword is used to find out which flag bits are set inside the TCP header of a packet. In this case, we are looking for packets having the SYN flag set in the TCP packet header.
- ttl:=32 -This specifies that snort should check the IP time-to-live value of 32.
- sid:2000002 – Snort rule ID to uniquely identify snort rules. Remember all numbers smaller than 1,000,000 are reserved; this is why we are starting with 1,000,001. (You may use any number, as long as it's greater than 1,000,000.)

The rule will detect only port scanning but not benign traffic because the options used for the rule have been carefully chosen and are peculiar only to port scanning packets.

Task 4: Detect DoS attack

For this task, metasploit is launched on server B and then the following commands to set up the attack.

```
use auxiliary/dos/tcp/synflood
set RHOSTS 192.168.70.5      //target's IP
set SHOST 192.168.70.6       //attacker's IP
exploit                      // to launch the attack
```

```
msf6 auxiliary(dos/tcp/synflood) > show options

Module options (auxiliary/dos/tcp/synflood):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE  enp0s8           no        The name of the interface
   NUM                         no        Number of SYNs to send (else unlimited)
   RHOSTS     192.168.70.5     yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT      80               yes       The target port
   SHOST      192.168.70.6     no        The spoofable source address (else randomizes)
   SNAPLEN    65535            yes       The number of bytes to capture
   SPORT                       no        The source port (else randomizes)
   TIMEOUT    500              yes       The number of seconds to wait for new data

msf6 auxiliary(dos/tcp/synflood) > exploit
[*] Running module against 192.168.70.5

[*] SYN flooding 192.168.70.5:80...
```

Observing on wireshark, we see a high frequency of SYN packets.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.70.6	192.168.70.5	TCP	60	22570 → 80 [SYN] Seq=3193346708 Win=1174 Len=0
2	0.001224641	192.168.70.6	192.168.70.5	TCP	60	5688 → 80 [SYN] Seq=3956048916 Win=3420 Len=0
3	0.002437832	192.168.70.6	192.168.70.5	TCP	60	86 → 80 [SYN] Seq=403216010 Win=1236 Len=0
4	0.003520740	192.168.70.6	192.168.70.5	TCP	60	33294 → 80 [SYN] Seq=704193228 Win=2535 Len=0
5	0.004541618	192.168.70.6	192.168.70.5	TCP	60	32470 → 80 [SYN] Seq=3153411475 Win=2027 Len=0
6	0.005625956	192.168.70.6	192.168.70.5	TCP	60	29768 → 80 [SYN] Seq=3507171490 Win=2919 Len=0
7	0.006626962	192.168.70.6	192.168.70.5	TCP	60	56264 → 80 [SYN] Seq=3769059986 Win=3533 Len=0
8	0.007668555	192.168.70.6	192.168.70.5	TCP	60	22486 → 80 [SYN] Seq=3629459267 Win=3078 Len=0
9	0.008800388	192.168.70.6	192.168.70.5	TCP	60	17006 → 80 [SYN] Seq=2942756725 Win=2638 Len=0
10	0.009768686	192.168.70.6	192.168.70.5	TCP	60	58012 → 80 [SYN] Seq=1990582585 Win=2900 Len=0
11	0.010693376	192.168.70.6	192.168.70.5	TCP	60	2434 → 80 [SYN] Seq=4193229942 Win=925 Len=0
12	0.011735288	192.168.70.6	192.168.70.5	TCP	60	57029 → 80 [SYN] Seq=624883091 Win=1463 Len=0
13	0.012673619	192.168.70.6	192.168.70.5	TCP	60	37406 → 80 [SYN] Seq=1814103424 Win=997 Len=0
14	0.013701736	192.168.70.6	192.168.70.5	TCP	60	5343 → 80 [SYN] Seq=3762087813 Win=3470 Len=0
15	0.014934530	192.168.70.6	192.168.70.5	TCP	60	36088 → 80 [SYN] Seq=775059905 Win=3842 Len=0
16	0.016050224	192.168.70.6	192.168.70.5	TCP	60	45858 → 80 [SYN] Seq=1731168113 Win=1341 Len=0
17	0.017375419	192.168.70.6	192.168.70.5	TCP	60	63954 → 80 [SYN] Seq=47146851 Win=3400 Len=0
18	0.018405335	192.168.70.6	192.168.70.5	TCP	60	15429 → 80 [SYN] Seq=285179622 Win=3210 Len=0
19	0.019432436	192.168.70.6	192.168.70.5	TCP	60	2217 → 80 [SYN] Seq=4925621 Win=3174 Len=0

Observing TCP/IP headers, there are not many features to pick from as IP TTL has varying lengths. The only observations are the high frequency of packets and that packets have only the SYN flag set.

```
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: PcsCompu_81:df:8e (08:00:27:81:df:8e)
▾ Internet Protocol Version 4, Src: 192.168.70.6, Dst: 192.168.70.5
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x151a (5402)
  ▸ Flags: 0x0000
    Time to live: 236
    Protocol: TCP (6)
    Header checksum: 0xac59 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.70.6
    Destination: 192.168.70.5
▾ Transmission Control Protocol, Src Port: 10689, Dst Port: 80, Seq: 259051382, Len: 0
    Source Port: 10689
    Destination Port: 80
    [Stream index: 126]
    [TCP Segment Len: 0]
    Sequence number: 259051382
    [Next sequence number: 259051382]
    Acknowledgment number: 0
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x002 (SYN)
    Window size value: 1179
    [Calculated window size: 1179]
    Checksum: 0x94f3 [unverified]
```

A possible snort rule for detecting DoS attack is seen below

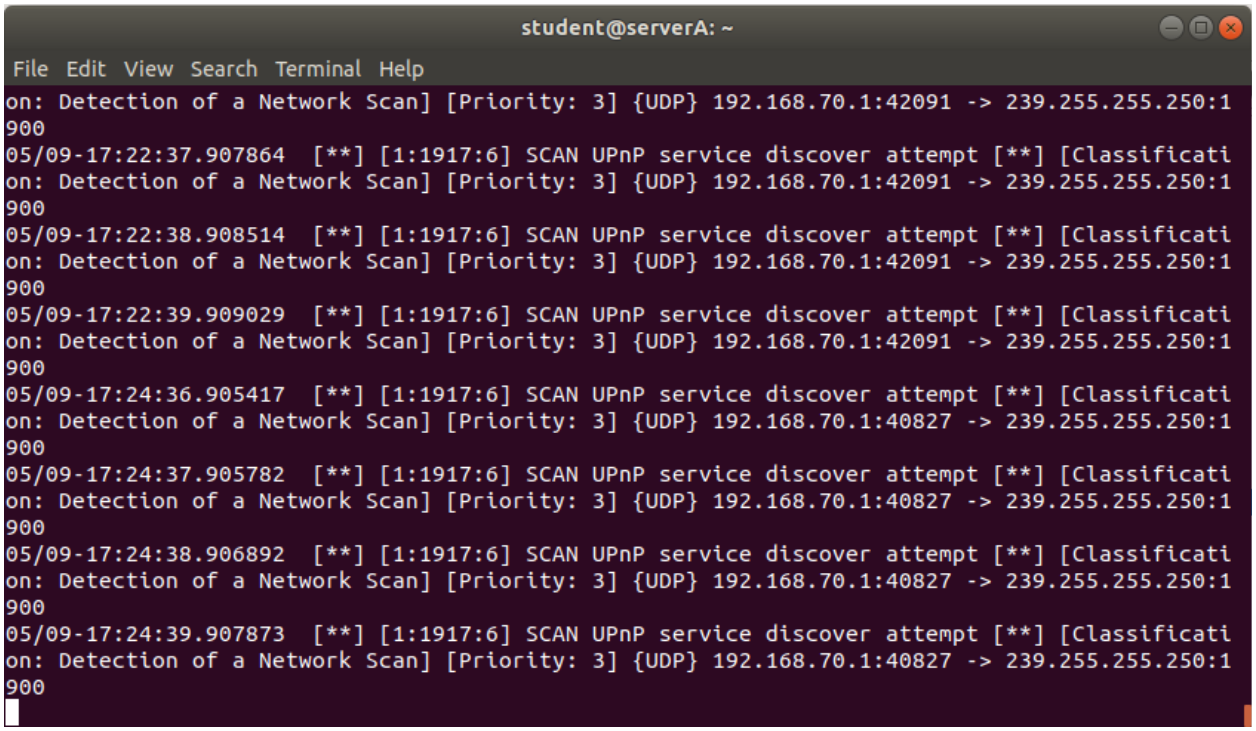
```
alert tcp 192.168.70.6 any -> 192.168.70.5 80 (msg:"TCP SYN Flooding DoS attack detected"; flags:S; threshold: type threshold, track by_dst, count 1000 , seconds 60; sid: 2000003; rev:1;)
```

After executing `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf`, while the exploit is launched on server B is ongoing, we can observe this results

```
student@serverA: ~
File  Edit  View  Search  Terminal  Help
y: 0] {TCP} 192.168.70.6:16920 -> 192.168.70.5:80
05/09-16:51:32.441400  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:27820 -> 192.168.70.5:80
05/09-16:51:33.073165  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:61536 -> 192.168.70.5:80
05/09-16:51:33.714160  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:21843 -> 192.168.70.5:80
05/09-16:51:34.383811  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:50684 -> 192.168.70.5:80
05/09-16:51:35.018331  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:56204 -> 192.168.70.5:80
05/09-16:51:35.639915  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:11047 -> 192.168.70.5:80
05/09-16:51:36.299288  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:12363 -> 192.168.70.5:80
05/09-16:51:36.948477  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:3466 -> 192.168.70.5:80
05/09-16:51:37.604015  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:63345 -> 192.168.70.5:80
05/09-16:51:38.257846  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:14803 -> 192.168.70.5:80
05/09-16:51:38.915388  [**] [1:2000003:1] TCP SYN Flooding DoS attack detected [**] [Priorit
y: 0] {TCP} 192.168.70.6:58810 -> 192.168.70.5:80
```

To verify that the rule does not generate alerts for benign traffic(telnet and ssh),I interrupted the exploit attack on server B with `ctrl+c` and then

run telnet 192.168.70.5 and/or ssh 192.168.70.5 and at this time, we don't see alerts as we have on the previous screenshot. The terminal looks like this



Explanation of the snort rule used.

Rule headers

- alert: Rule action. Snort will generate an alert when the set condition is met.
- Tcp: Signifies the TCP/IP protocol that the rule must match
- 192.168.70.6: specifies the source IP for snort to look at.
- any: any is for the source port. Snort will look at all ports
- -> : Specifies the direction. From source to destination
- 192.168.70.5: specifies the destination IP address.
- 80: Destination port. Snort will look at port 80

Rule options:

- msg:"TCP SYN Flooding DoS attack detected" – Snort will include this message with the alert.
- flags:S - The "flags'' keyword is used to find out which flag bits are

set inside the TCP header of a packet. In this case, we are looking for packets having the SYN flag set in the TCP packet header.

- The "threshold" keyword means that this rule logs every event on this SID during a 60 second interval. So, if less than 1000 events occur in 60 seconds, nothing gets logged. Once an event is logged, a new time period starts.
- The "track" by_dst keyword means track by destination IP.
- The "count" keyword means count number of events.
- The "seconds" keyword means time period over which count is accrued.
- sid:2000003 – Snort rule ID to uniquely identify snort rules. Remember all numbers smaller than 1,000,000 are reserved; this is why we are starting with 1,000,001. (You may use any number, as long as it's greater than 1,000,000.)
- rev:1 – Revision number. This option allows for easier rule maintenance.

The rule will detect only DDoS attacks but not benign traffic because the options used for the rule have been carefully chosen and are peculiar only to DDoS attacks

Task 5: Detect incoming rogue SSH connections

For this task, metasploit is launched on server B and then the following commands to set up the attack

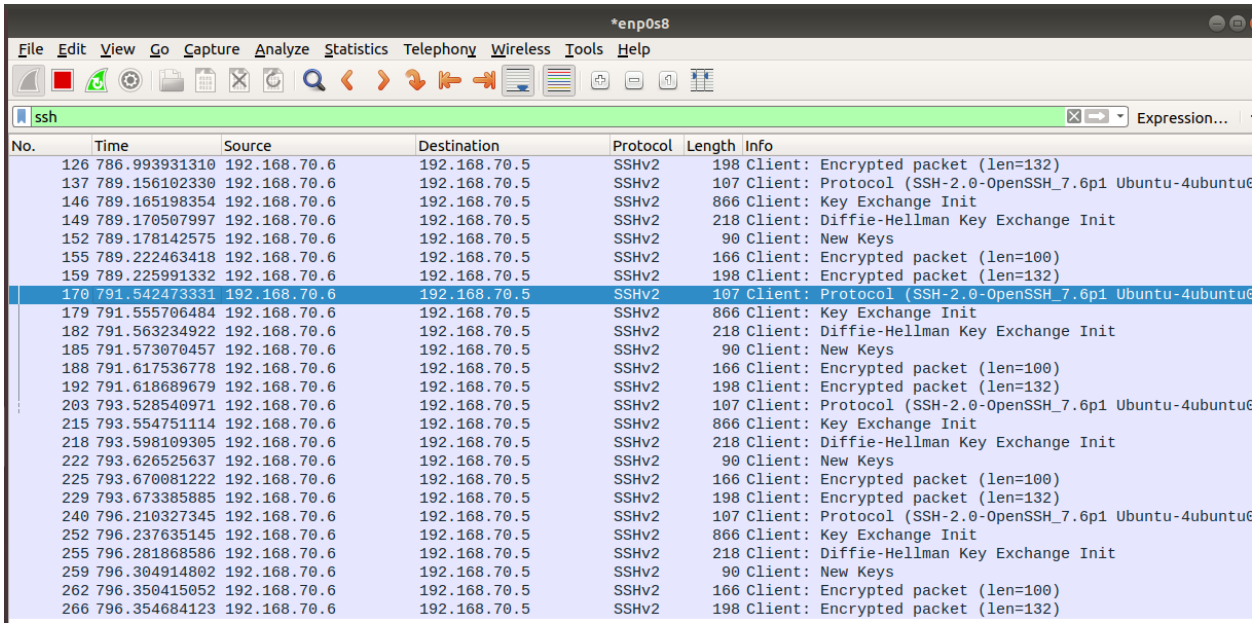
```
use auxiliary/scanner/ssh/ssh_login
set RHOSTS 192.168.70.5
```

`set USERPASS_FILE Desktop/passwords.txt` // passwords.txt has been created withinserver's B Desktop

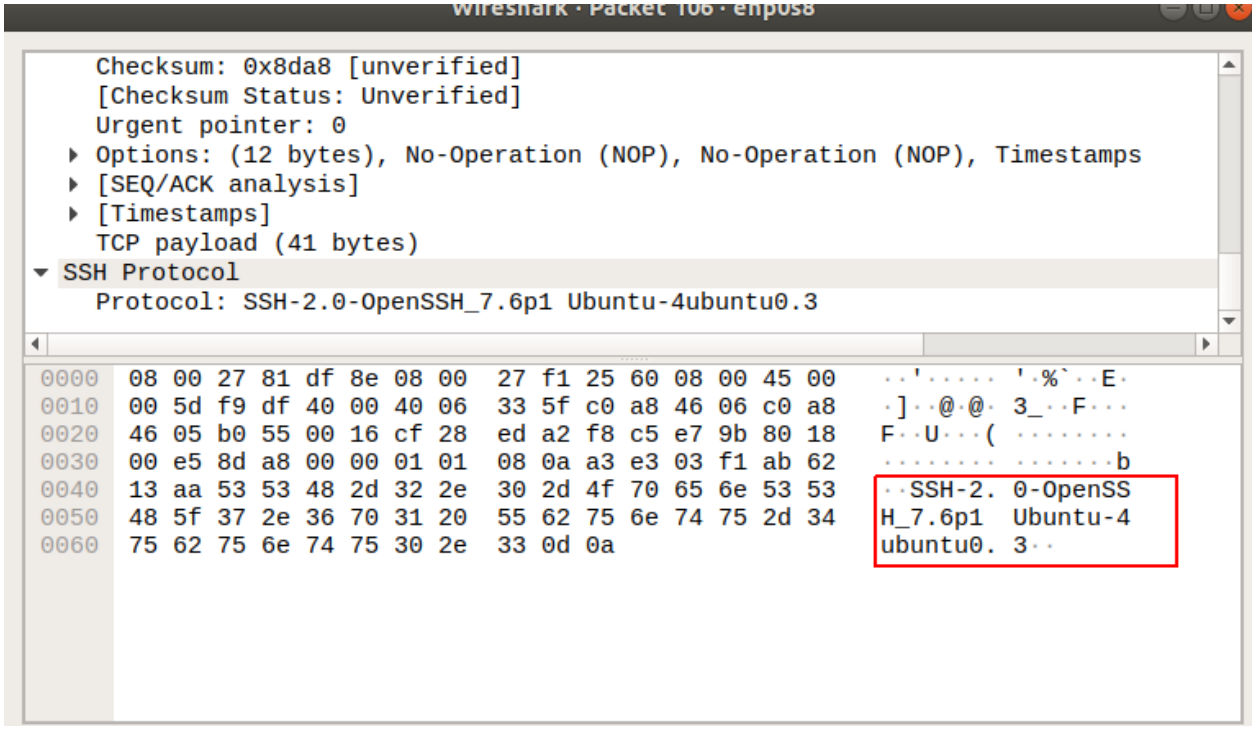
```
run
```

```
Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting         Required  Description
   ----              ---------------         --------  -----------
   BLANK_PASSWORDS   false                   no        Try blank passwords for all users
   BRUTEFORCE_SPEED  5                       yes       How fast to bruteforce, from 0 to 5
   DB_ALL_CREDS      false                   no        Try each user/password couple stored in the current database
   DB_ALL_PASS       false                   no        Add all passwords in the current database to the list
   DB_ALL_USERS      false                   no        Add all users in the current database to the list
   PASSWORD                                  no        A specific password to authenticate with
   PASS_FILE                                 no        File containing passwords, one per line
   RHOSTS            192.168.70.5            yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path
>'
   RPORT             22                      yes       The target port
   STOP_ON_SUCCESS   false                   yes       Stop guessing when a credential works for a host
   THREADS           1                       yes       The number of concurrent threads (max one per host)
   USERNAME                                  no        A specific username to authenticate as
   USERPASS_FILE     Desktop/passwords.txt   no        File containing users and passwords separated by space, one pair per line
   USER_AS_PASS      false                   no        Try the username as the password for all users
   USER_FILE                                 no        File containing usernames, one per line
   VERBOSE           false                   yes       Whether to print output for all attempts

msf6 auxiliary(scanner/ssh/ssh_login) >
```

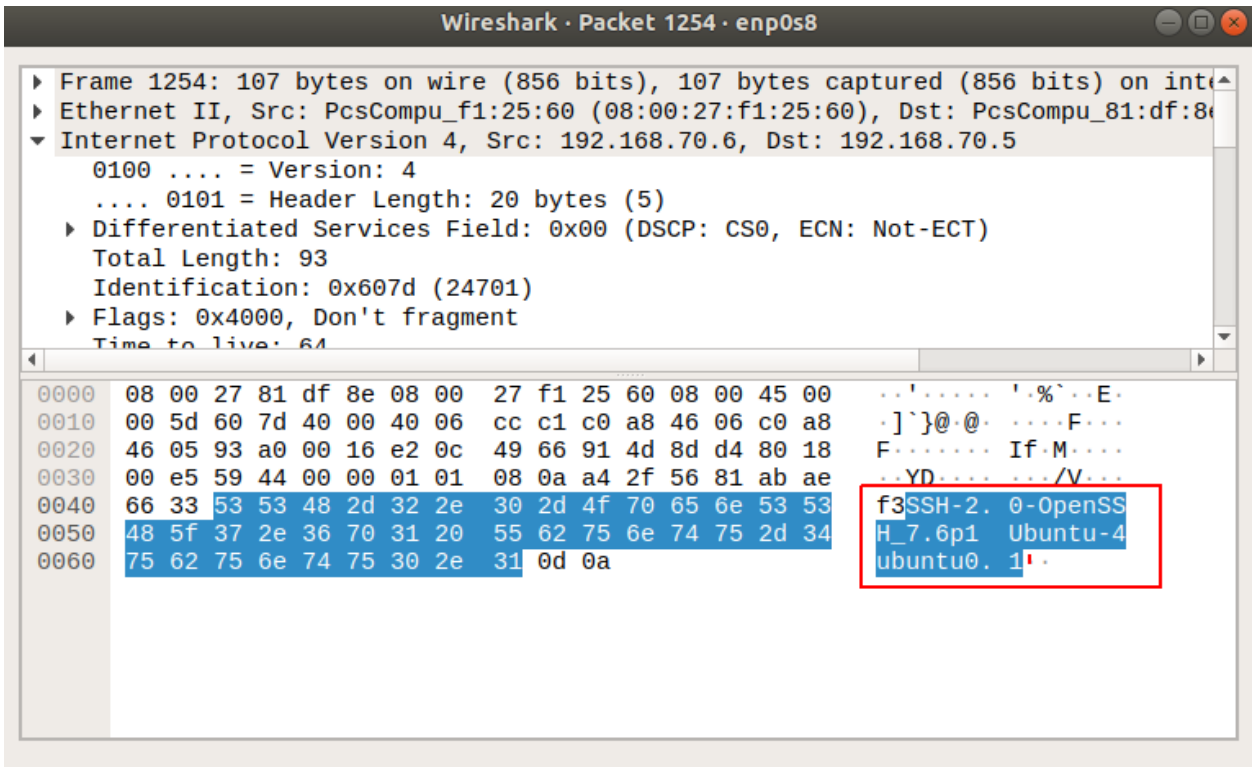
Observing on wireshark, we see the following



For the malicious traffic, the data exchange for the SSH handshake is as on the screenshot below.



For the benign SSH traffic, the data exchanged is as seen below

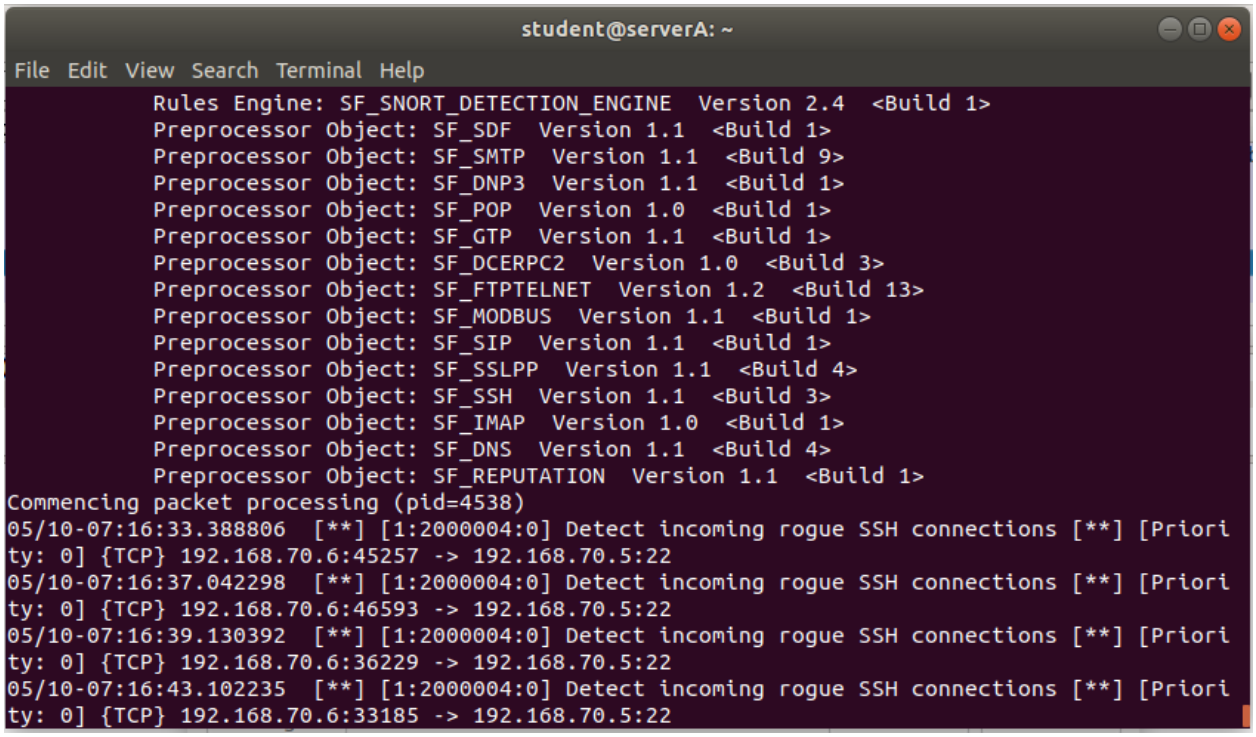


The observation here is that for the malicious traffic, the data exchange for the SSH handshake is “SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3” and done several times while for the benign SSH traffic, it is “SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.1” done just once.

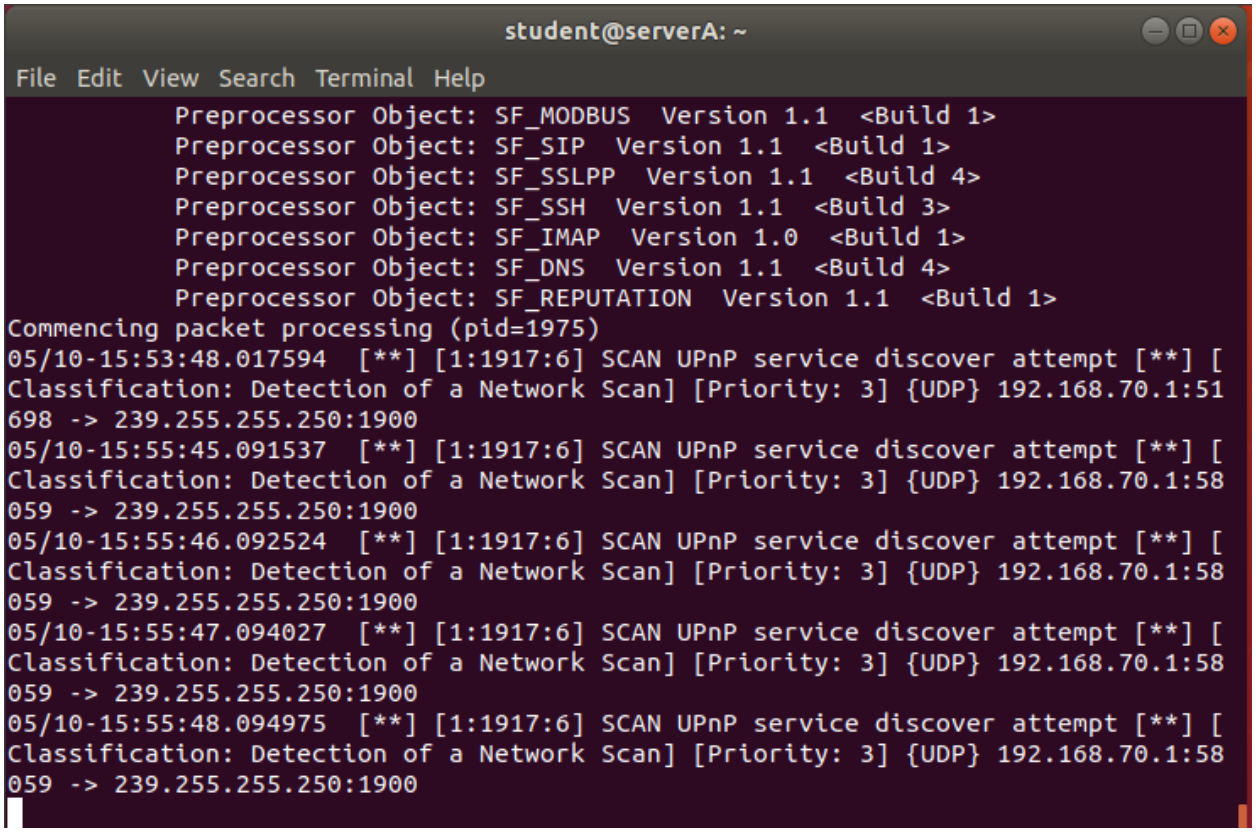
So a possible snort rule for detecting the malicious SSH handshake is as follows;

```
alert tcp 192.168.70.6 any -> 192.168.70.5 22 (msg: "Detect incoming rogue SSH connections"; content:"SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3"; nocase; depth:40; threshold:type threshold,track by_src,count 5,seconds 10; sid:2000004;)
```

After executing `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf`, while the exploit is launched repeatedly on server B is ongoing, we can observe this results



Verifying with benign SSH traffic(ssh 192.168.70.5) from server B to server A, we don’t observe any alerts with"Detect incoming rogue SSH connections" message after executing `sudo snort -i enp0s8 -A console -c /etc/snort/snort.conf`



This confirms that the snort rule detects only malicious connection attempts but not benign SSH traffic(ssh 192.168.70.5) from server B.

Explanation of the snort rule used.

Rule headers

- alert: Rule action. Snort will generate an alert when the set condition is met.
- Tcp: Signifies the TCP/IP protocol that the rule must match
- 192.168.70.6: specifies the source IP for snort to look at.
- any: any is for the source port. Snort will look at all ports
- -> : Specifies the direction. From source to destination
- 192.168.70.5: specifies the destination IP address.
- 22: Destination port. Snort will look at port 22

Rule options:

- msg: "Detect incoming rogue SSH connections" – Snort will include this message with the alert.
- content:"SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3: It allows the rule to check if the string pattern "SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3" matches any content within the packet's payload and generate alerts if there's a match.
- The nocase keyword tells the rule to look for the given pattern ignoring case. nocase modifies the previous content keyword in the rule.
- The depth keyword allows the rule to specify how far into a packet Snort should search for the specified pattern. depth modifies the previous `content' keyword in the rule.A depth of 40 would tell Snort to only look for the specified pattern within the first 40 bytes of the payload.
- The "threshold" keyword means that this rule logs every event on this SID during a 10 second interval. So, if less than 5 events occur in 10 seconds, nothing gets logged. Once an event is logged, a new time period starts.
- The "track" by_src keyword means track by source IP.
- The "count" keyword means count number of events.
- The "seconds" keyword means time period over which count is accrued.
- sid:2000004 – Snort rule ID to uniquely identify snort rules. Remember all numbers smaller than 1,000,000 are reserved; this is why we are starting with 1,000,001. (You may use any number, as long as it's greater than 1,000,000.)

The rule will detect only incoming rogue SSH connections but not benign SSH traffic because the options used for the rule have been carefully chosen and are peculiar only to rogue SSH connections