SNORT Intrusion Detection System

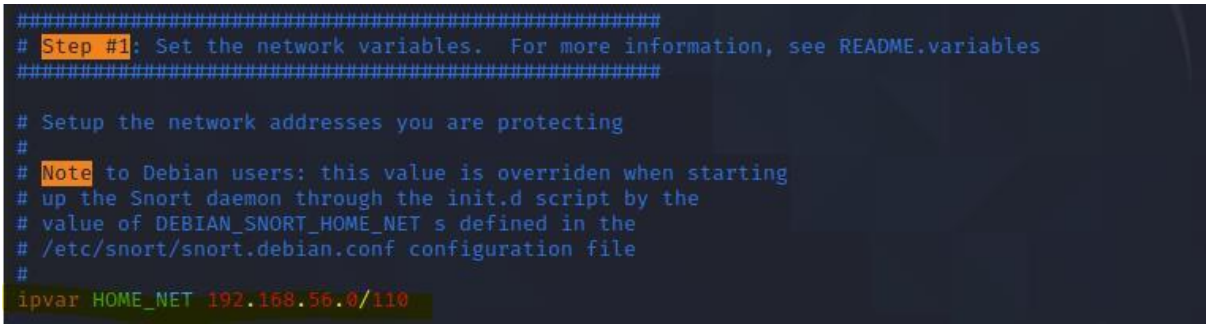
LAB SETUP

1. Following Virtual Machines were installed on the virtual box to complete this lab:
   - Kali Linux VM as an attacker
   - Metasploitable VM as a victim

2. The IP address of both machines are:
   - Attacker's machine- 192.168.56.102
   - Victim's machine- 192.168.56.104

3. Installed snort on kali Linux VM using given commands: *sudo apt-get update*
   sudo apt-get install snort

4. Updated given configuration in snort.conf in kali VM
   sudo vi /etc/snort/snort.conf



LAB TASKS

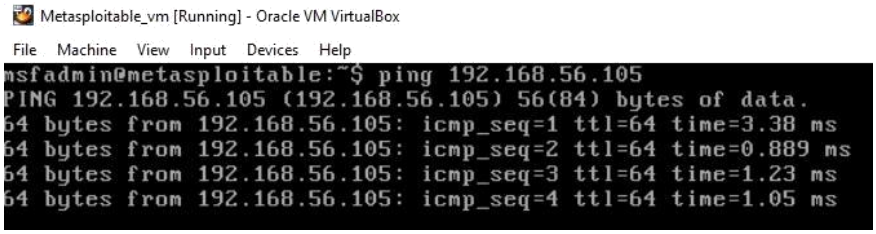
Task 1: Use Snort as a Packet Sniffer.

Demonstrate how snort can be used as a packet sniffer.

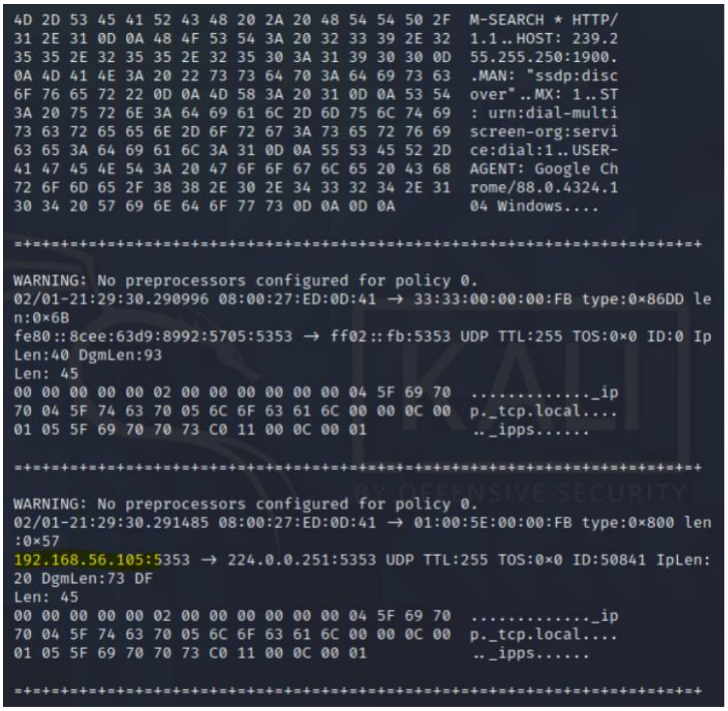
On the command line type snort –vde and press ENTER. This command and options will run Snort as a Packet Sniffer

For this task, ping a different ip address and observe the snort output

- Executed given command from attacker's VM
  snort -vde
- And parallelly executed given command from victim's VM to different IP address (192.168.56.105)
  ping 192.168.56.105



- Ping command shows the ability of the source IP (192.168.56.104) to reach a specified destination IP (192.168.56.105).



- In the above snapshot, observed that the packet (192.168.56.105) is transmitting from the victim's machine to a different IP address (192.168.56.105) that has been sniffed by the attacker's VM.

Task 2: Run Snort as IDS to detect ping scans

Write custom rules to detect the ping scan, write configuration file, test the rule and check the logs

- **The custom rule** for snort as IDS to detect the ping scans is shown in a given snapshot:

```
alert icmp any any → $HOME_NET any (msg:"Ping scan test"; sid:1000001; rev:1;)
```

- **Updated the configuration file** with include and rule file path, refer below screenshot:

```
# set my snort rules
include $RULE_PATH/kraticarules.rules
```

- Executed given command on one terminal of attacker's machine and observed given read/detected rules:
  sudo snort -A console -c /etc/snort/snort.conf -i eth0

```
+++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
1 Snort rules read
    1 detection rules
    0 decoder rules
    0 preprocessor rules
1 Option Chains linked into 1 Chain Headers
+++++++++++++++++++++++++++++++++++++++++++++++++++
```

One snort rule was read or detected in rule chains because only one rule is enabled in **kraticarules.rules** file.

To test the rule:

- by executing the below command on the attacker's VM: *sudo snort -T -c /etc/snort/snort.conf*
  Here -T means testing

```
        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.15.1 GRE (Build 15125)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2019 Cisco and/or its affiliates. All rights reserv
ed.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.9.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.1  <Build 1>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: appid  Version 1.1  <Build 5>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>

Snort successfully validated the configuration!
Snort exiting
```

Observed that the snort has successfully validated the configuration.

- On the second terminal of the attacker's machine, run the command:
  sudo nmap -sn 192.168.56.104 -disable-arp-ping

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sn 192.168.56.104 -disable-arp-ping
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-02 04:37 EST
Nmap scan report for 192.168.56.104
Host is up (0.00074s latency).
MAC Address: 08:00:27:BF:FC:39 (Oracle VirtualBox virtual NIC)
Nmap done: 1 IP address (1 host up) scanned in 16.63 seconds
```

- Used -disable-arp-ping in the above command because it is disabling arp packets and it will filter only the ICMP packet as defined in rules.

- After running the snort as IDS, observed the **given logs** where packets are processing from source IP address (192.168.56.102) to destination IP address (192.168.56.104)

```
Commencing packet processing (pid=4315)
02/02-04:51:56.742509  [**] [1:1000001:1] "Ping scan test" [**] [Priority: 0] {IC
MP} 192.168.56.102 → 192.168.56.104
02/02-04:51:56.742509  [**] [1:469:3] ICMP PING NMAP [**] [Classification: Attemp
ted Information Leak] [Priority: 2] {ICMP} 192.168.56.102 → 192.168.56.104
02/02-04:51:56.743996  [**] [1:1000001:1] "Ping scan test" [**] [Priority: 0] {IC
MP} 192.168.56.102 → 192.168.56.104
```

- Hence, observed that snort rules which are set for ping scan have successfully validated the configuration.

Task 3: Run Snort as IDS to detect port scans

Write custom rules to detect various types of port scans. Recall the nmap scans including SYN scan, FIN scan and XMAS scan

STEP 1: Use Nmap to launch a SYN scan attack, create a rule to detect the attack, test the rule and check the logs

- **Created rule** to detect SYN scan attack

  alert tcp any any -> $HOME_NET any (flags:S; msg:"Test SYN scan attack"; sid: 1000002; rev:1;)

```
alert tcp any any → $HOME_NET any (flags:S; msg:"Test SYN scan attack"; sid: 100
0002; rev:1;)
```

- **Tested the rule** by executing below command from another terminal of attacker's machine: *sudo nmap -sS 192.168.56.104*

```
└─$ sudo nmap -sS 192.168.56.104
[sudo] password for kali:
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-02 16:53 EST
Nmap scan report for 192.168.56.104
Host is up (0.00073s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
```

- **Logs** which are formed after executing "*sudo snort -A console -c /etc/snort/snort.conf -i eth0*" command and parallel running nmap SYN scan command are as follows:

  02/02-16:53:17.254443 [**] [1:1000002:1] Test SYN scan attack [**] [Priority: 0] {TCP} 192.168.56.102:55928 -> 192.168.56.104:19
  02/02-16:53:17.254785 [**] [1:1000002:1] Test SYN scan attack [**] [Priority: 0] {TCP} 192.168.56.102:55928 -> 192.168.56.104:24444
  02/02-16:53:17.255116 [**] [1:1000002:1] Test SYN scan attack [**] [Priority: 0] {TCP} 192.168.56.102:55928 -> 192.168.56.104:9102
  02/02-16:53:17.255447 [**] [1:1000002:1] Test SYN scan attack [**] [Priority: 0] {TCP} 192.168.56.102:55928 -> 192.168.56.104:7004
  02/02-16:53:17.256182 [**] [1:1000002:1] Test SYN scan attack [**] [Priority: 0] {TCP} 192.168.56.102:55928 -> 192.168.56.104:1099

STEP2: Use Nmap to launch a FIN scan attack, create a rule to detect the attack, test the rule and check the logs

In a FIN scan the attacker is searching for open ports using only the FIN flag. This is notifying the target that is wants to tear down a connection, even though no connection is present. Any packet not containing a SYN, RST or ACK will result in a returned RST if the port is closed and no response if the port is open

- **Created rule** to detect FIN scan attack
  alert tcp any any -> $HOME_NET any (flags:F; msg:"Detect FIN scan attack"; sid: 1000003; rev:1;)

```
alert tcp any any → $HOME_NET any (flags:F; msg:"Detect FIN scan attack"; sid: 1
000003; rev:1;)
```

- **Tested the rule** by executing the below command from another terminal of the attacker's machine:
  sudo nmap -sF 192.168.56.104

```
└─$ sudo nmap -sF 192.168.56.104
[sudo] password for kali:
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-02 18:16 EST
Nmap scan report for 192.168.56.104
Host is up (0.00044s latency).
Not shown: 977 closed ports
PORT     STATE         SERVICE
21/tcp   open|filtered ftp
22/tcp   open|filtered ssh
23/tcp   open|filtered telnet
25/tcp   open|filtered smtp
53/tcp   open|filtered domain
80/tcp   open|filtered http
111/tcp  open|filtered rpcbind
139/tcp  open|filtered netbios-ssn
445/tcp  open|filtered microsoft-ds
512/tcp  open|filtered exec
513/tcp  open|filtered login
514/tcp  open|filtered shell
1099/tcp open|filtered rmiregistry
1524/tcp open|filtered ingreslock
2049/tcp open|filtered nfs
2121/tcp open|filtered ccproxy-ftp
3306/tcp open|filtered mysql
5432/tcp open|filtered postgresql
5900/tcp open|filtered vnc
6000/tcp open|filtered X11
```

  Observed no response received (even after retransmissions) for few ports when setting TCP FIN bit.

- **Logs** which are formed after executing "*sudo snort -A console -c /etc/snort/snort.conf -i eth0*" command and parallel running nmap FIN scan command are as follows:

  02/02-18:16:24.897414 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:3323
  02/02-18:16:24.897723 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:40911
  02/02-18:16:24.898060 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:5679
  02/02-18:16:24.898396 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:3493
  02/02-18:16:24.898736 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:2382
  02/02-18:16:24.899068 [**] [1:1000003:1] "Detect FIN Scan attack" [**] [Priority: 0] {TCP} 192.168.56.102:52648 -> 192.168.56.104:10180

STEP3: Use Nmap to launch a XMAS scan attack, create a rule to detect the attack, test the rule and check the logs

In an XMAS scan, the attacker is searching for open ports using the FIN, PSH and URG flags; As mentioned back in Step 2, any combination of these three flags will result in a returned RST if the port is closed and no response if the port is open.

- **Created rule** to detect XMAS scan attack
  alert tcp any any -> $HOME_NET any (flags:FPU; msg:"Detect XMAS scan attack"; sid: 1000004; rev:1;)

```
alert tcp any any → $HOME_NET any (flags:FPU; msg:"Detect XMAS scan attack"; sid
: 1000004; rev:1;)
~
```

- **Tested the rule** by executing below command from another terminal of attacker's machine:
  sudo nmap -sX 192.168.56.104

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sX 192.168.56.104
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-02 18:26 EST
Nmap scan report for 192.168.56.104
Host is up (0.00048s latency).
Not shown: 977 closed ports
PORT     STATE         SERVICE
21/tcp   open|filtered ftp
22/tcp   open|filtered ssh
23/tcp   open|filtered telnet
25/tcp   open|filtered smtp
53/tcp   open|filtered domain
80/tcp   open|filtered http
111/tcp  open|filtered rpcbind
139/tcp  open|filtered netbios-ssn
445/tcp  open|filtered microsoft-ds
512/tcp  open|filtered exec
513/tcp  open|filtered login
514/tcp  open|filtered shell
1099/tcp open|filtered rmiregistry
1524/tcp open|filtered ingreslock
2049/tcp open|filtered nfs
2121/tcp open|filtered ccproxy-ftp
3306/tcp open|filtered mysql
5432/tcp open|filtered postgresql
5900/tcp open|filtered vnc
6000/tcp open|filtered X11
```

Observed few ports which sets the FIN, PSH, and URG flags, lighting the packet up like a Christmas tree.

- **Logs** which are formed after executing "*sudo snort -A console -c /etc/snort/snort.conf -i eth0*" command and parallel running nmap XMAS scan command are as follows:

  02/02-18:28:58.401076 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:16993
  02/02-18:28:58.401154 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:1147
  02/02-18:28:58.401230 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:49161
  02/02-18:28:58.401353 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:5911
  02/02-18:28:58.401438 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:5225
  02/02-18:28:58.401517 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:5061
  02/02-18:28:58.401590 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:5810
  02/02-18:28:58.401666 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:3077
  02/02-18:28:58.401759 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:1096
  02/02-18:28:58.401838 [**] [1:1000004:1] Detect XMAS scan attack [**] [Priority: 0] {TCP} 192.168.56.102:45280 -> 192.168.56.104:6389

Task 4: Write your own custom rule to detect ANY other attack on the victim machine. Demonstrate the attack and show how snort is capable of detecting this attack

For this task, created rule to detect **UDP flood attack** which attacker tries to do on the victim's machine.

A UDP flood attack comprise of flooding UDP target ports with UDP packets on a victim machine. The victim host or UDP program can slow down or go down if enough UDP packets are delivered to the destination UDP port. A spoofed or random IP address should be set as the source IP address. The destination UDP port in the victim host should be set to the number of an available UDP port.

- **Created rule** to detect UDP flood attack
  alert udp any any -> $HOME_NET any (msg:"Detect UDP flood attack"; sid: 1000005; rev:1;)

```
alert udp any any → $HOME_NET any (msg: "Detect UDP Flood attack"; sid:1000005; rev:1
;)
```

- **Tested the rule** by executing below command from another terminal of attacker's machine:
  sudo nmap -sU -p 80 192.168.56.104

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sU -p 80 192.168.56.104
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-02 21:09 EST
Nmap scan report for 192.168.56.104
Host is up (0.0011s latency).

PORT   STATE  SERVICE
80/udp closed http
MAC Address: 08:00:27:BF:FC:39 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 16.82 seconds
```

```
Commencing packet processing (pid=1931)
02/02-21:09:49.367917  [**] [1:1000005:1] Detect UDP Flood attack [**] [Priority: 0] {
UDP} 192.168.56.102:39366 → 192.168.56.104:80
```

Observed one IP address of port 80 was detected when scanned through UDP scan.

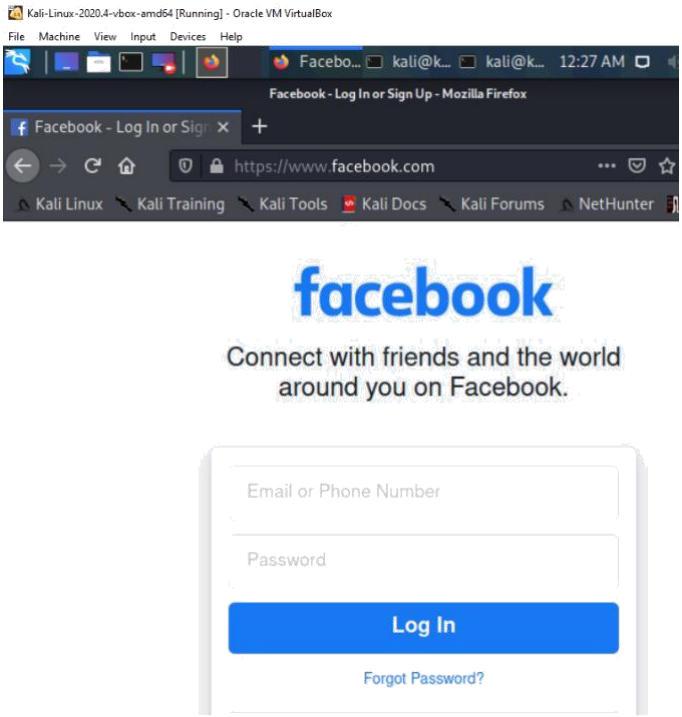
Task 5: Write a rule that will fire when you browse to facebook.com from the machine Snort is running on; it should look for any outbound TCP request to facebook.com and alert on it.

- **Created rule** that will fire when browsing facebook.com on browser where snort is running.

```
alert tcp any any → $EXTERNAL_NET any (content: "facebook.com"; sid:100000
6; rev:1; msg: "facebook.com detected";)
```

The content used in the rule is facebook.com instead of its IP because it will fire only when facebook.com browsed on browser.

- The given snapshot shows facebook.com to be open properly on the firefox browser.



- As the facebook.com hits on the browser of the machine where snort rule is running, then the alert was triggered.

```
Commencing packet processing (pid=1132)
02/03-00:27:33.688174  [**] [1:1000006:1] facebook.com detected [**] [Prior
ity: 0] {TCP} 157.240.3.35:80 → 10.0.2.15:44422
```

Task6: Answer the following questions

Q1: State how each of following real rules from the snort home page work:

i. **alert icmp any any -> any any (msg:"ICMP Source Quench"; itype: 4; icode: 0;)**

This rule specifies ICMP protocol where the message has been mentioned as ICMP Source Quench with type 4 and code 0, which is meant to be the **mechanism for congestion control with the network layer.**

icode: The icode keyword is used to detect the code field in the ICMP packet header. If code field is 0, it is a network redirect ICMP packet.

itype: The itype keyword is used to detect attacks that use the type field in the ICMP packet header. “4” is for the source quence.

The keyword “any” can be define as the source address, destination address, source and destination port in this rule. The arrow in this rule defines that traffic is flowing in one direction.

“alert” can be define as the action to be perform.

ii. **alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-CGI view-source access";flags: A+; content:"/view source?../../../../../../../etc/passwd"; nocase;reference:cve,CVE-1999-0174;)**

This rule specifies that snort has detected traffic exploiting vulnerabilities in web-based applications on servers.

The CVE-1999-0174 in the view-source CGI program allows remote attackers to read arbitrary files via a .. (dot dot) attack.

The “flags: A+” is define as “Acknowledge flag with AND” where bits are set inside the TCP header of a packet.

The “nocase” keyword is used in combination with the content keyword. It has no arguments. Its only purpose is to make a case insensitive search of a pattern within the data part of a packet.

“Content” keyword is used to find these signatures in the packet.

The reference keyword can add a reference to information present on other systems available on the Internet.

“alert” can be define as the action to be perform and protocol to be used as “tcp” in this rule.

The keyword “any” can be define as the source port. The EXTERNAL_NET is the source IP address, HTTP_SERVERS is the destination IP address and 80 is the destination port. The arrow in this rule defines that traffic is flowing in one direction.

Q2: Develop your own snort signature to capture DNS queries directed against the host that you choose to connect to via HTTPS. Make sure that your snort rule references the DNS data and not simply IP address of the server

The host used for building up the snort signature is “facebook.com”. Refer below snort signature created to capture DNS queries against facebook.com to connect via HTTPS.

alert udp any any -> any any (msg:"DNS query detected"; sid: 1; content:"|9d f0 03 23|"; reference:url,https://www.facebook.com/; rev:1)

DNS primarily uses the User Datagram Protocol (UDP) on port number 53 to serve requests. DNS queries consist of a single UDP request from the client followed by a single UDP reply from the server.

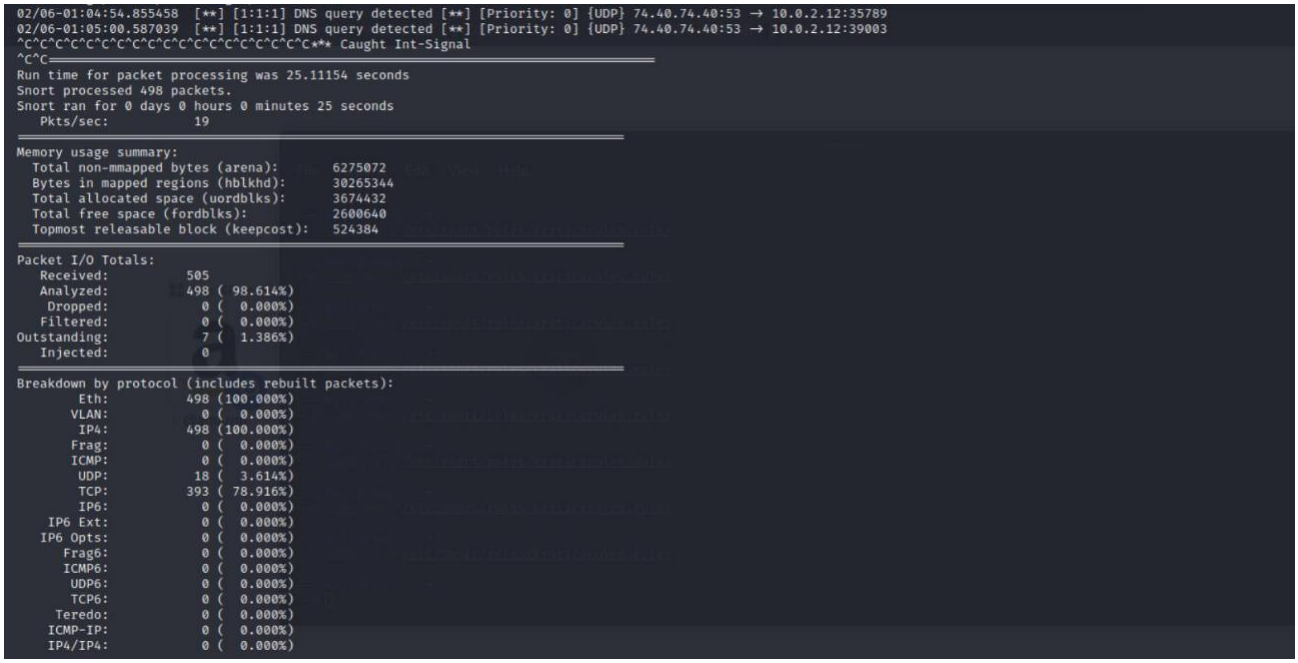
content:"|9d f0 03 23|"

This is the host IP address (157.240.3.3.5) which we get after doing nslookup of www.facebook.com and converted it to hexadecimal.

```
┌──(kali㉿kali)-[~]
└─$ nslookup www.facebook.com
Server:         192.152.0.1
Address:        192.152.0.1#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 157.240.3.35
Name:   star-mini.c10r.facebook.com
Address: 2a03:2880:f101:83:face:b00c:0:25de
```

Below are the logs which we got after executing the snort rule for DNS queries.



Observation:

The total I/O packets received and analyzed are 505 and 498. The runtime for packet processing was 25.11 seconds