Kevin Huang
CSS 537
Lab 3: SNORT Intrusion Detection System
02/05/2022

Introduction

In this lab, I explored the SNORT intrusion detection system tool. I learned how to create custom rules and analyze the log files. For the lab, I used a Kali Linux VM for my victim and have installed SNORT and SEED labs machine to be the attacker.

Task 1: Use Snort as a Packet Sniffer

For this task, I used my SEED labs VM (10.0.2.5) to ping my Kali Linux VM while running 'snort -vde'. I see that the snort program sniffs out the ping request.

Screenshot 1.1: SEED VM sends ping

```
[02/01/22]seed@VM:~$ ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.200 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.199 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.199 ms
^C
--- 10.0.2.15 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.199/0.199/0.200/0.000 ms
```

Screenshot 1.2: Kali VM snort program sniffs ping

```
┌──(kali㉿kali)-[~]
└─$ sudo snort -vde
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
Acquiring network traffic from "eth0".
Decoding Ethernet

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=4466)
WARNING: No preprocessors configured for policy 0.
02/01-21:58:32.228779 08:00:27:FB:A2:84 → 08:00:27:50:4C:14 type:0×800 len:0×62
10.0.2.5 → 10.0.2.15 ICMP TTL:64 TOS:0×0 ID:25282 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:10   Seq:1  ECHO
58 F3 F9 61 00 00 00 00 28 35 0B 00 00 00 00 00  X..a....(5......
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Task 2: Run Snort as IDS to detect ping scans

For this task, I edited the HOME_NET address to my target VM and also included my custom lab3.rules. The rule is to only log ICMP requests going to the target VM. I tested this by pinging the target VM and then trying to telnet into the target VM. This rule was successful and only the ICMP request was logged.

Screenshot 2.1: Updated HOME_NET address

```
###################################################
# Step #1: Set the network variables.  For more information, see README.variables
###################################################

# Setup the network addresses you are protecting
#
# Note to Debian users: this value is overriden when starting
# up the Snort daemon through the init.d script by the
# value of DEBIAN_SNORT_HOME_NET s defined in the
# /etc/snort/snort.debian.conf configuration file
#
ipvar HOME_NET 10.0.2.15

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any
```

Screenshot 2.2: include only lab3.rules

```
# Path to your rules files (this can be a relative path)
# Note for Windows users:  You are advised to make this an absolute path,
# such as:  c:\snort\rules
var RULE_PATH /etc/snort/rules
var SO_RULE_PATH /etc/snort/so_rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules

include $RULE_PATH/lab3.rules
```

Screenshot 2.3: lab3.rules

```
  GNU nano 5.9                                lab3.rules *
#task 2: detect ping scans
alert icmp any any → 10.0.2.15 any (msg:"Ping detected"; sid:1000001; rev:1;)
```

Screenshot 2.4: attacker VM pinging and then telnet into 10.0.2.15

```
[02/01/22]seed@VM:~$ ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.207 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.264 ms
^C
--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1023ms
rtt min/avg/max/mdev = 0.207/0.235/0.264/0.028 ms
[02/01/22]seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
telnet: Unable to connect to remote host: Connection refused
```

Screenshot 2.4: Running snort

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/01-23:04:38.005959  [**] [1:1000001:1] Ping detected [**] [Priority: 0] {ICMP} 10.0.2.5 -
> 10.0.2.15
02/01-23:04:39.029969  [**] [1:1000001:1] Ping detected [**] [Priority: 0] {ICMP} 10.0.2.5 -
> 10.0.2.15
^C** Caught Int-Signal
```

Screenshot 2.5: Captured only ICMP requests

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1643774665
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1643774665".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=21338)
WARNING: No preprocessors configured for policy 0.
02/01-23:04:38.005959 10.0.2.5 → 10.0.2.15
ICMP TTL:64 TOS:0×0 ID:7777 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:15   Seq:1  ECHO
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

WARNING: No preprocessors configured for policy 0.
02/01-23:04:39.029969 10.0.2.5 → 10.0.2.15
ICMP TTL:64 TOS:0×0 ID:7796 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:15   Seq:2  ECHO
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

===============================================================================
Run time for packet processing was 0.45 seconds
Snort processed 2 packets.
Snort ran for 0 days 0 hours 0 minutes 0 seconds
   Pkts/sec:           2
===============================================================================
```

Task 3: Run Snort as IDS to detect port scans

Step 1: I made a custom rule in lab3.rules to detect TCP SYN scans. I used the SEED VM to launch a SYN scan attack on the Kali VM using “nmap -sS -Pn 10.0.2.15”. Snort was able to detect the scans.

Screenshot 3.1: rule for with “S” flag for TCP SYN scan

```
#task 3, step 1: SYN scan attack
alert tcp any any → 10.0.2.15 any (flags:S; msg:"TCP SYN detected"; sid:1000001; rev:1;)
```

Screenshot 3.2: Kali VM is running snort

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -A console -q -c /etc/snort/snort.conf
02/02-16:31:43.156281  [**] [1:1000001:1] TCP SYN detected [**] [Priority: 0] {TCP} 10.0.2.5
:47380 → 10.0.2.15:1723
02/02-16:31:43.156281  [**] [1:1000001:1] TCP SYN detected [**] [Priority: 0] {TCP} 10.0.2.5
:47380 → 10.0.2.15:443
02/02-16:31:43.156281  [**] [1:1000001:1] TCP SYN detected [**] [Priority: 0] {TCP} 10.0.2.5
:47380 → 10.0.2.15:113
```

Screenshot 3.3: SEED VM uses nmap to launch TCP SYN scan attack

```
Nmap done: 1 IP address (1 host up) scanned in 0.34 seconds
[02/02/22]seed@VM:~$ sudo nmap -sS -Pn 10.0.2.15
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-02 16:31 EST
Nmap scan report for 10.0.2.15
Host is up (0.000082s latency).
All 1000 scanned ports on 10.0.2.15 are closed
MAC Address: 08:00:27:50:4C:14 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.32 seconds
```

Screenshot 3.4: Logs show successful detection of SYN

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1643837498
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1643837498".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=35720)
WARNING: No preprocessors configured for policy 0.
02/02-16:31:43.156281 10.0.2.5:47380 → 10.0.2.15:1723
TCP TTL:42 TOS:0x0 ID:49992 IpLen:20 DgmLen:44
******S* Seq: 0x2C892EF3  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) ⇒ MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/02-16:31:43.156281 10.0.2.5:47380 → 10.0.2.15:443
TCP TTL:48 TOS:0x0 ID:5603 IpLen:20 DgmLen:44
******S* Seq: 0x2C892EF3  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) ⇒ MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/02-16:31:43.156281 10.0.2.5:47380 → 10.0.2.15:113
TCP TTL:45 TOS:0x0 ID:14110 IpLen:20 DgmLen:44
******S* Seq: 0x2C892EF3  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) ⇒ MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Step 2: I made a custom rule in lab3.rules to detect TCP FIN scans. I used the SEED VM to launch a FIN scan attack on the Kali VM using "nmap -sF -Pn 10.0.2.15". Snort was able to detect the scans.

Screenshot 3.5: rule for "F" flag for TCP FIN scan

```
#task 3, step 2: FIN scan attack
alert tcp any any → 10.0.2.15 any (flags:F; msg:"TCP FIN detected"; sid:1000001; rev:1;)
```

Screenshot 3.6: SEED VM launch TCP FIN scan attack

```
[02/02/22]seed@VM:~$ sudo nmap -sF -Pn 10.0.2.15
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-02 16:46 EST
Nmap scan report for 10.0.2.15
Host is up (0.00022s latency).
All 1000 scanned ports on 10.0.2.15 are closed
MAC Address: 08:00:27:50:4C:14 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.35 seconds
```

Screenshot 3.7: Kali VM running snort detects

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/02-16:46:07.371007  [**] [1:1000001:1] TCP FIN detected [**] [Priority: 0] {TCP} 10.0.2.5
:37151 → 10.0.2.15:554
02/02-16:46:07.371007  [**] [1:1000001:1] TCP FIN detected [**] [Priority: 0] {TCP} 10.0.2.5
:37151 → 10.0.2.15:8888
```

Screenshot 3.8: Snort log shows successful capture of packet with FIN set

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1643838298
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1643838298".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=39157)
WARNING: No preprocessors configured for policy 0.
02/02-16:46:07.371007 10.0.2.5:37151 → 10.0.2.15:554
TCP TTL:53 TOS:0x0 ID:58679 IpLen:20 DgmLen:40
*******F Seq: 0x76DC6562  Ack: 0x0  Win: 0x400  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/02-16:46:07.371007 10.0.2.5:37151 → 10.0.2.15:8888
TCP TTL:38 TOS:0x0 ID:13722 IpLen:20 DgmLen:40
*******F Seq: 0x76DC6562  Ack: 0x0  Win: 0x400  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Step 3: I made a custom rule in lab3.rules to detect TCP XMAS scans. I used the SEED VM to launch a XMAS scan attack on the Kali VM using “nmap -sX -Pn 10.0.2.15”. Snort was able to detect the scans.

Screenshot 3.9: rule for “FPU” flags for TCP XMAS scan

```
#task 3, step 3: XMAS scan attack
alert tcp any any → 10.0.2.15 any (flags:FPU; msg:"TCP XMAS detected"; sid:1000001; rev:1;)
```

Screenshot 3.10: SEED VM launch nmap TCP XMAS scan attack

```
[02/02/22]seed@VM:~$ sudo nmap -sX -Pn 10.0.2.15
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-02 16:55 EST
Nmap scan report for 10.0.2.15
Host is up (0.000091s latency).
All 1000 scanned ports on 10.0.2.15 are closed
MAC Address: 08:00:27:50:4C:14 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.31 seconds
```

Screenshot 3.10: Kali VM running snort detects XMAS scan

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/02-16:55:12.499291  [**] [1:1000001:1] TCP XMAS detected [**] [Priority: 0] {TCP} 10.0.2.5:43212
 → 10.0.2.15:554
02/02-16:55:12.499292  [**] [1:1000001:1] TCP XMAS detected [**] [Priority: 0] {TCP} 10.0.2.5:43212
 → 10.0.2.15:3389
```

Screenshot 3.11: Snort log shows successful capture of TCP packets with “UPF” flags

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1643838889
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1643838889".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=41257)
WARNING: No preprocessors configured for policy 0.
02/02-16:55:12.499291 10.0.2.5:43212 → 10.0.2.15:554
TCP TTL:50 TOS:0x0 ID:62280 IpLen:20 DgmLen:40
**U*P**F Seq: 0x152C29D5  Ack: 0x0  Win: 0x400  TcpLen: 20  UrgPtr: 0x0
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/02-16:55:12.499292 10.0.2.5:43212 → 10.0.2.15:3389
TCP TTL:39 TOS:0x0 ID:17470 IpLen:20 DgmLen:40
**U*P**F Seq: 0x152C29D5  Ack: 0x0  Win: 0x400  TcpLen: 20  UrgPtr: 0x0
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Task 4:

I made a custom rule in lab3.rules to detect TCP ACK scans. I used the SEED VM to launch a ACK scan attack on the Kali VM using “nmap -sA -Pn 10.0.2.15”. Snort was able to detect the scans.

Screenshot 4.1: rule with “A” flag for ACK scan attacks

```
#task 4: ACK scan attack
alert tcp any any → 10.0.2.15 any (flags:A; msg:(TCP ACK detected"; sid:1000001; rev:1;)
```

Screenshot 4.2: SEED VM using nmap for TCP ACK scan attack

```
[02/02/22]seed@VM:~$ sudo nmap -sA -Pn 10.0.2.15
Starting Nmap 7.80 ( https://nmap.org ) at 2022-02-02 18:02 EST
Nmap scan report for 10.0.2.15
Host is up (0.00018s latency).
All 1000 scanned ports on 10.0.2.15 are unfiltered
MAC Address: 08:00:27:50:4C:14 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.36 seconds
```

Screenshot 4.3: Kali VM detects ACK packets with snort

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/02-18:02:02.618532  [**] [1:1000001:1] (TCP ACK detected" [**] [Priority: 0] {TCP} 10.0.2.5:4438
0 → 10.0.2.15:80
02/02-18:02:02.618533  [**] [1:1000001:1] (TCP ACK detected" [**] [Priority: 0] {TCP} 10.0.2.5:4438
0 → 10.0.2.15:113
```

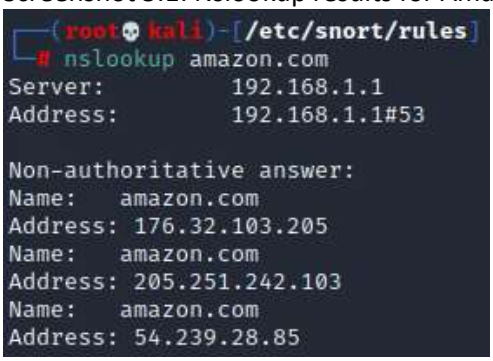
Screenshot 4.4: Snort log report shows successful detection of ACK packets

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1643842909
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1643842909".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=55199)
WARNING: No preprocessors configured for policy 0.
02/02-18:02:02.618532 10.0.2.5:44380 → 10.0.2.15:80
TCP TTL:39 TOS:0x0 ID:40961 IpLen:20 DgmLen:40
***A**** Seq: 0x0  Ack: 0xBF7EF1EB  Win: 0x400  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/02-18:02:02.618533 10.0.2.5:44380 → 10.0.2.15:113
TCP TTL:49 TOS:0x0 ID:39408 IpLen:20 DgmLen:40
***A**** Seq: 0x0  Ack: 0xBF7EF1EB  Win: 0x400  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

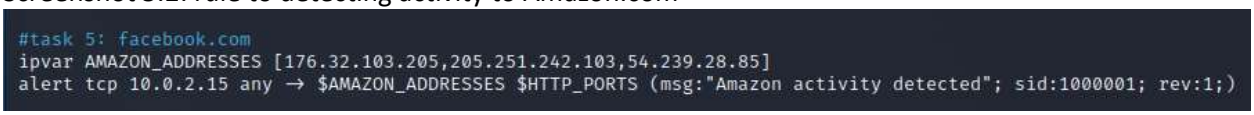
Task 5:

For this task, I was supposed to write a rule that would detect when you browse facebook.com. I was having problems detecting any results. I asked Professor Geetha about my rule and it seemed like I had the correct idea. I tried using amazon.com instead of facebook.com and I was able to successfully detect outbound TCP request. I used nslookup to find the addresses for amazon.com. In my rule, I did any port from the Kali VM to Amazon addresses with HTTP ports (specified in the snort config file). The other way that I got it to work was use the IP address of facebook.com directly. I was able to detect the outbound TCP request this way.

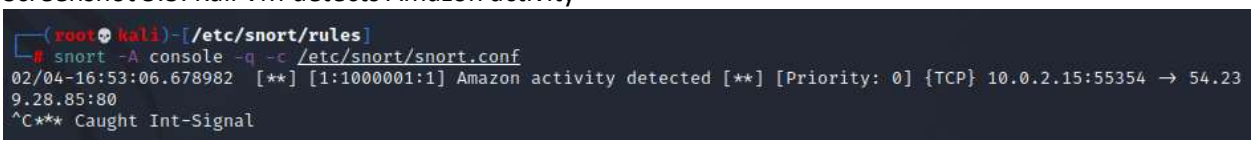
Screenshot 5.1: Nslookup results for Amazon.com



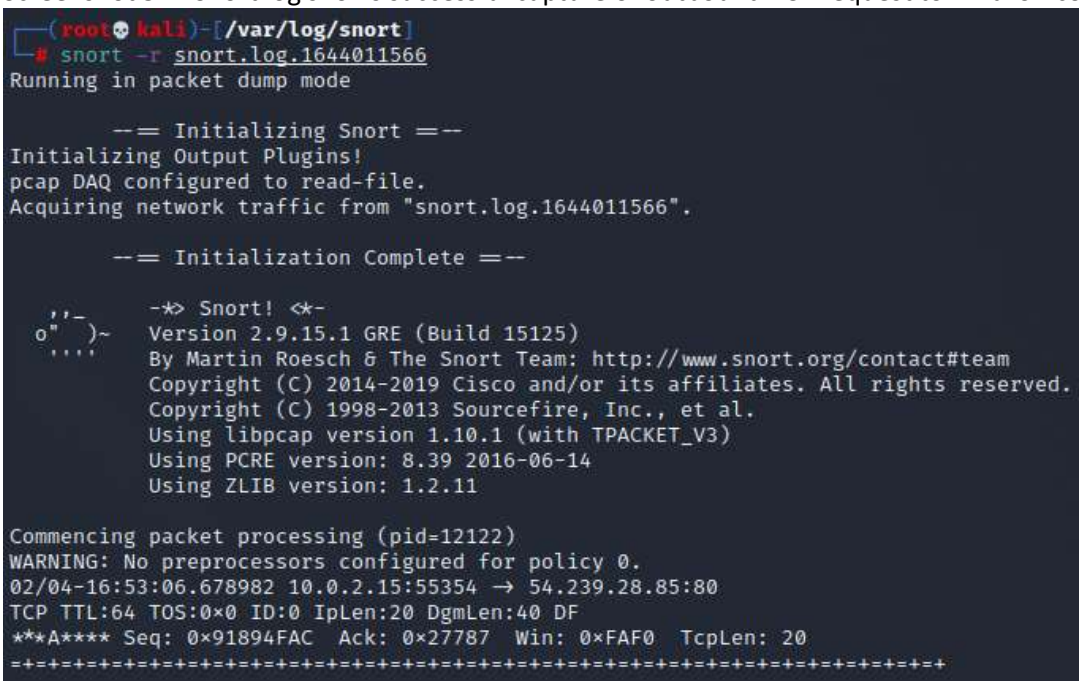
Screenshot 5.2: rule to detecting activity to Amazon.com



Screenshot 5.3: Kali VM detects Amazon activity



Screenshot 5.4: Snort log shows successful capture of outbound TCP request to Amazon.com



Screenshot 5.5: Rule for detecting activity to Facebook.com

```
#task 5: facebook.com
#ipvar AMAZON_ADDRESSES [176.32.103.205,205.251.242.103,54.239.28.85]
#alert tcp 10.0.2.15 any → $AMAZON_ADDRESSES $HTTP_PORTS (msg:"Amazon activity detected"; sid:1000001; rev:1;)
alert tcp 10.0.2.15 any → 157.240.3.35 $HTTP_PORTS (msg: "Facebook activity detected"; sid:1000002; rev:1;)
```

Screenshot 5.6: Outbound TCP activity captured

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/05-17:42:50.932883  [**] [1:1000002:1] Facebook activity detected [**] [Priority: 0] {TCP} 10.0.2.15:56352 → 157.240.3.35:80
^C*** Caught Int-Signal
```

Screenshot 5.7: Snort log shows successful capture of TCP activity to Facebook's IP address

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1644100953
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1644100953".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=3305)
WARNING: No preprocessors configured for policy 0.
02/05-17:42:50.932883 10.0.2.15:56352 → 157.240.3.35:80
TCP TTL:64 TOS:0×0 ID:0 IpLen:20 DgmLen:40 DF
***A**** Seq: 0×B70852A1  Ack: 0×198C  Win: 0×FAF0  TcpLen: 20
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Task 6:

Q1:

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

This rule generates an alert and then logs the packet. This rule is for any and all packets on the network with any source address and port and any destination address and port. The type of ICMP packets that will be detected is type 4 code 0, which is a source quench request to decrease the traffic rate. If detected, a “ICMP Source Quench” message will be alerted.

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 generates an alert and then logs the packet. The rule is for packets coming from any addresses listed in $EXTERNAL_NET any port to the destination addresses listed in $HTTP_SERVERS with port 80. The rule looks for "/view source?../../../../../../../etc/passwd" in the packet payload, and not case sensitive. The “A+” flag is for packets with the ACK flag plus any other additional bits set. If detected, it will prompt the “WEB-CGI view-source access” message. The rule also references an external attack identification system “cve” and the specific one pertaining to this attack: “CVE-1999-0174”.

Q2:

In this task, we needed to capture DNS queries directed against a host of our choice. The first thing I did was to check if I can write rules to detect all DNS queries. DNS uses port 53 so that’s where the destination port will be. When I did nslookup with this general rule, only the responses were captured. When I changed the rule to detect DNS queries/replies for Amazon.com, I was not able to capture any packets at all even though Wireshark shows the packet being sent.

Screenshot 6.1: Rules for detection of all DNS queries and responses

```
#task 6: DNS
alert udp 10.0.2.15 any → any 53 (msg:"DNS query detected"; sid:1000001; rev:1;)
alert udp any 53 → 10.0.2.15 any (msg:"DNS reply detected"; sid:1000002; rev:1;)
```

Screenshot 6.2: Using nslookup to generate a DNS query

```
┌──(kali㉿kali)-[~]
└─$ nslookup amazon.com
Server:         192.168.1.1
Address:        192.168.1.1#53

Non-authoritative answer:
Name:   amazon.com
Address: 54.239.28.85
Name:   amazon.com
Address: 205.251.242.103
Name:   amazon.com
Address: 176.32.103.205
```

Screenshot 6.3: Only the DNS replies are getting captured

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
02/05-18:01:29.687494  [**] [1:1000002:1] DNS reply detected [**] [Priority: 0] {UDP} 192.168.1.1:53 → 10.0.2.15:56955
02/05-18:01:29.699694  [**] [1:1000002:1] DNS reply detected [**] [Priority: 0] {UDP} 192.168.1.1:53 → 10.0.2.15:46053
^C** Caught Int-Signal
```

Screenshot 6.4: Snort log shows DNS replies only

```
┌──(root💀kali)-[/var/log/snort]
└─# snort -r snort.log.1644102086
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to read-file.
Acquiring network traffic from "snort.log.1644102086".

        --== 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 reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=8418)
WARNING: No preprocessors configured for policy 0.
02/05-18:01:29.687494 192.168.1.1:53 -> 10.0.2.15:56955
UDP TTL:255 TOS:0x0 ID:765 IpLen:20 DgmLen:104
Len: 76
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
02/05-18:01:29.699694 192.168.1.1:53 -> 10.0.2.15:46053
UDP TTL:255 TOS:0x0 ID:766 IpLen:20 DgmLen:117
Len: 89
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Screenshot 6.5: rules for detecting DNS queries and replies for Amazon.com

```
#task 6: DNS
#alert udp 10.0.2.15 any -> any 53 (msg:"DNS query detected"; sid:1000001; rev:1;)
#alert udp any 53 -> 10.0.2.15 any (msg:"DNS reply detected"; sid:1000002; rev:1;)
alert udp 10.0.2.15 any -> any 53 (msg:"Amazon DNS query detected"; content:"amazon.com"; nocase; sid:1000001; rev:1;)
alert udp any 53 -> 10.0.2.15 any (msg:"Amazon DNS reply detected"; content:"amazon.com"; nocase; sid:1000002; rev:1;)
```

Screenshot 6.6: Wireshark shows the DNS request and replies completing

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	192.168.1.1	DNS	70	Standard query 0x8dc9 A amazon.com
2	0.012710982	192.168.1.1	10.0.2.15	DNS	118	Standard query response 0x8dc9 A amazon.com A 54.239.28.85 A 205.251.242.103 A 176
3	0.013134136	10.0.2.15	192.168.1.1	DNS	70	Standard query 0xcb6c AAAA amazon.com
4	0.026173785	192.168.1.1	10.0.2.15	DNS	131	Standard query response 0xcb6c AAAA amazon.com SOA dns-external-master.amazon.com

Screenshot 6.7: No packets captured with rule

```
┌──(root💀kali)-[/etc/snort/rules]
└─# snort -A console -q -c /etc/snort/snort.conf
```