SOUTHEAST MISSOURI
STATE UNIVERSITY · 1873®

.

Assignment Week 6

Date: 09/28/2024

Course: Info Sec in System Admins (CY520-01)
Professor: Dr. George Li
Author:
Sartaj Jamal Chowdhury

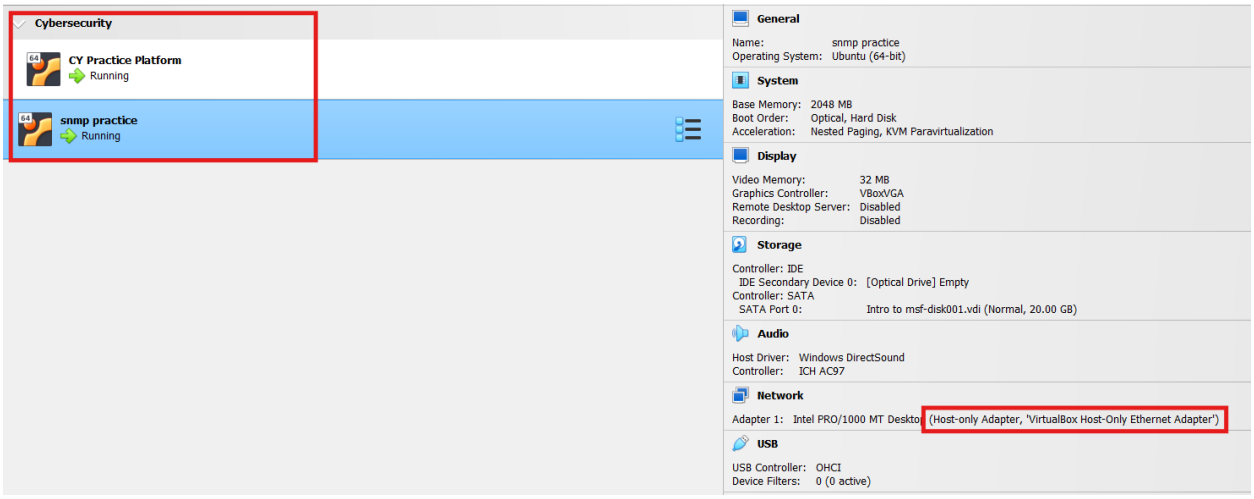
Contents

Problem Statement

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

Environment

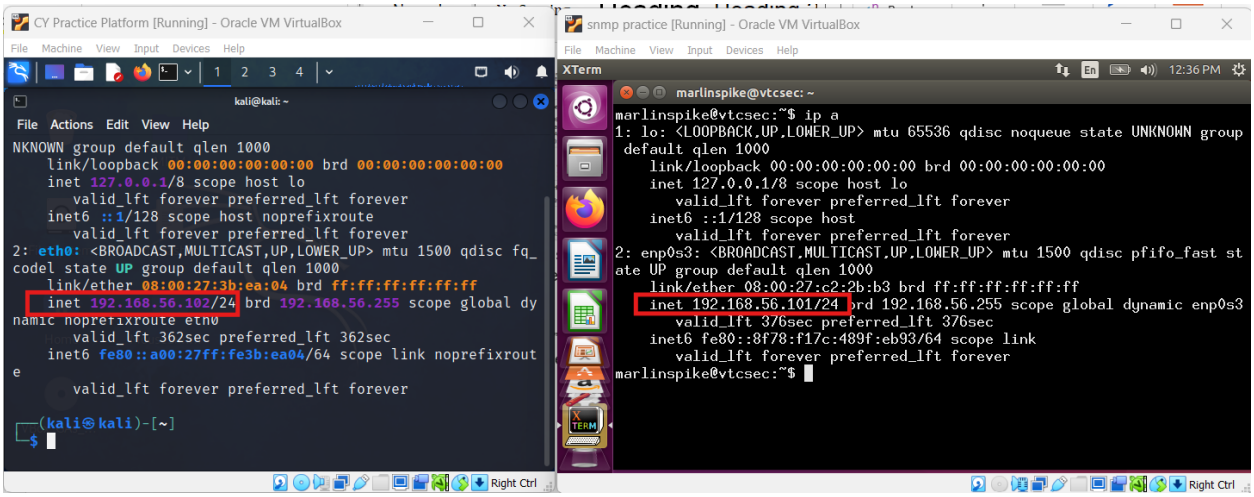
We utilize 2 virtual machines.



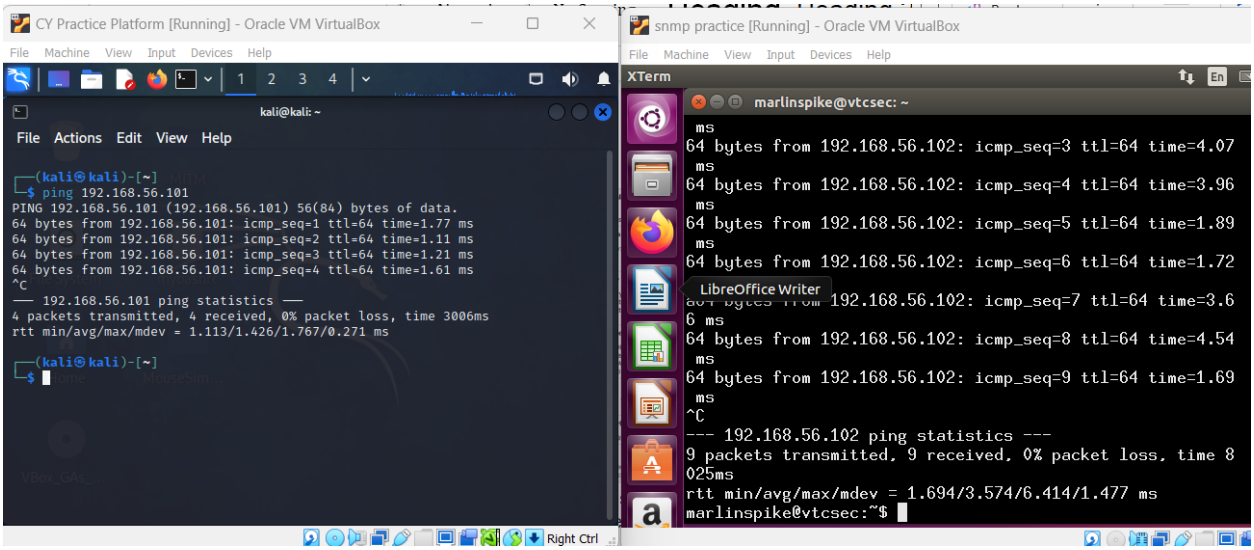
“snmp practice” machine is an Ubuntu v16.04 machine. We use this as the Target Machine. “CY Practice Platform” machine is a Kali 2023.3 machine. We use this as the Attacking Machine.

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.



Flood.py: Attack Simulation Script

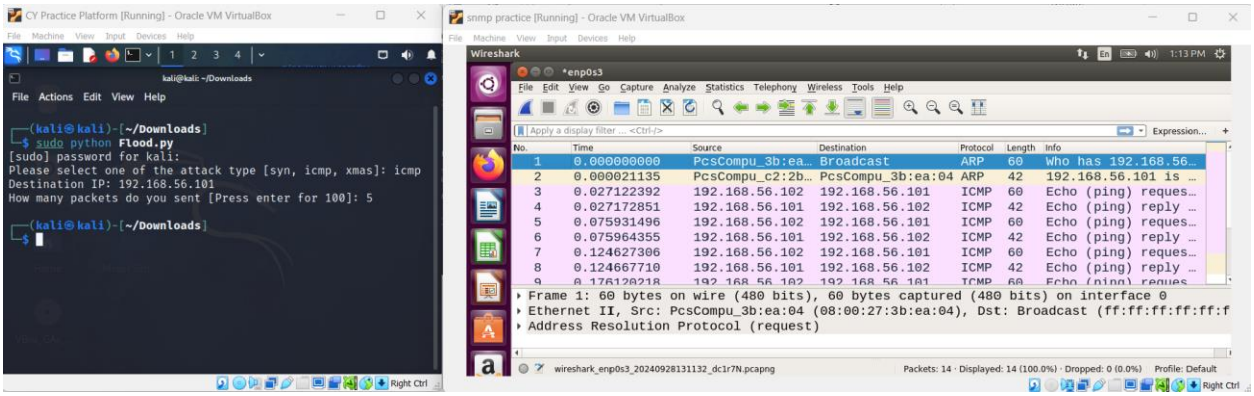
```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat Flood.py
#/usr/bin/python3
#@EmreOvunc
#@0×00
#Source - https://github.com/EmreOvunc/Icmp-Syn-Flood
#Requirements - pip3 install scapy
from scapy.all import *


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.

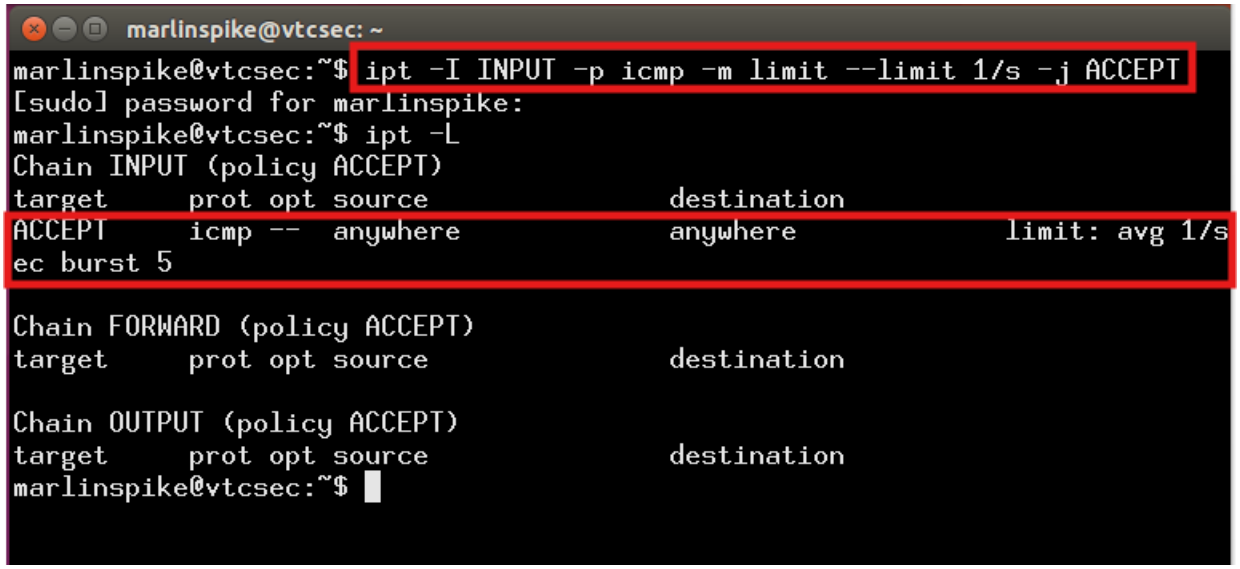
IPTables on the Target Machine

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

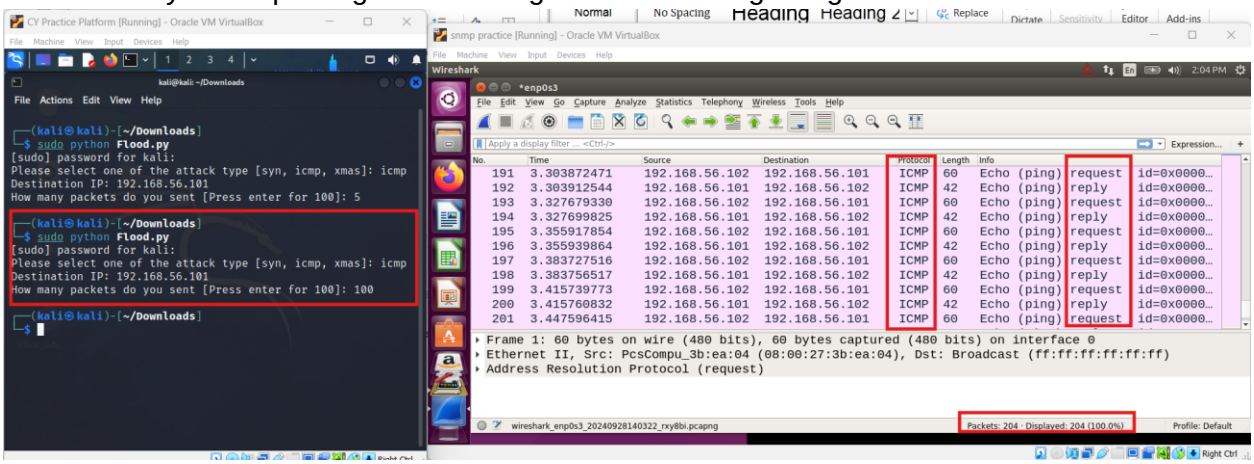


Here, we insert a new IPTable rule:
ipt -I INPUT -p icmp -m limit –limit 1/s -j ACCEPT

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:



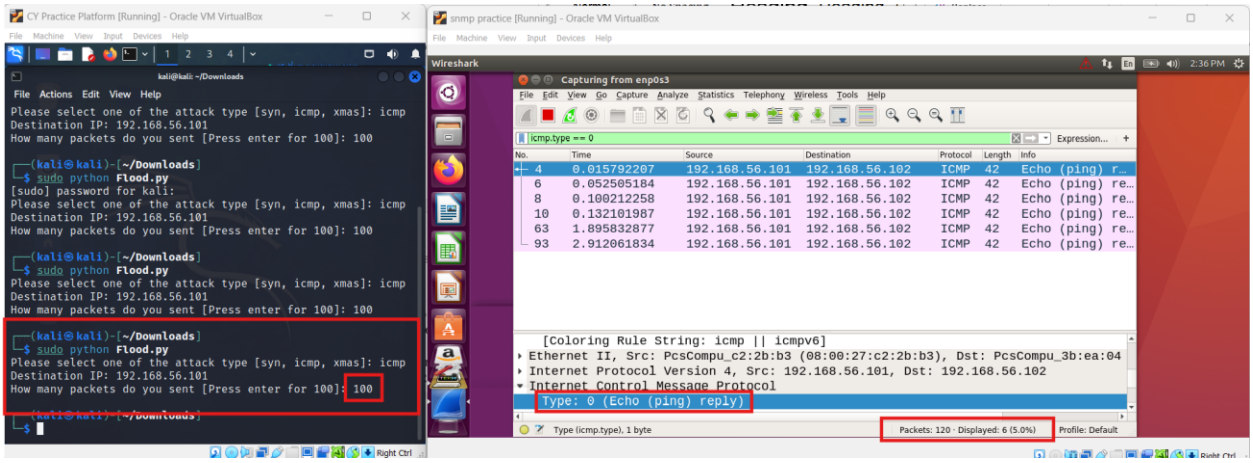
Here, we can see that for the 100 ICMP packets we sent from the attacking machine, all were received in the target machine, and all were also replied to.

```
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

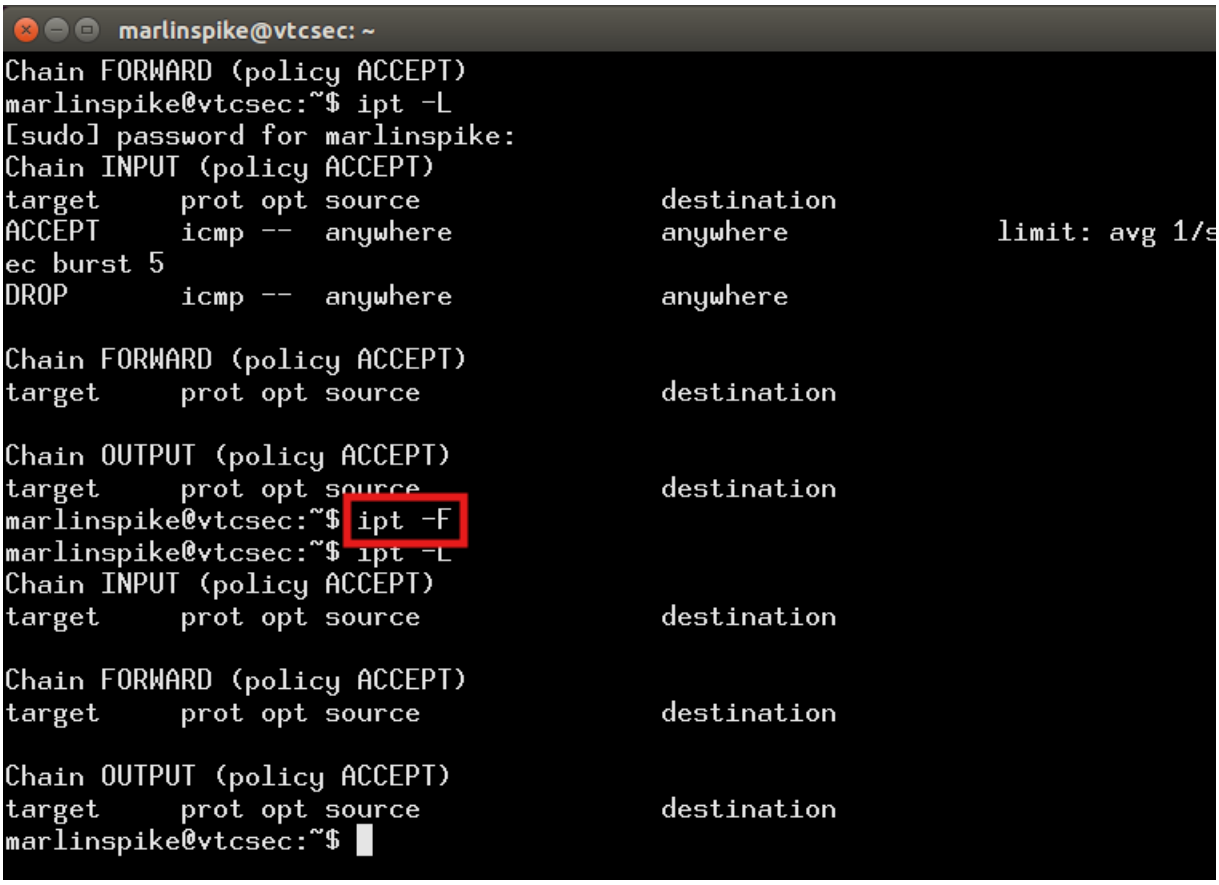
This rule helps drop any extra icmp packets.

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 (ipt -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 (ipt -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.

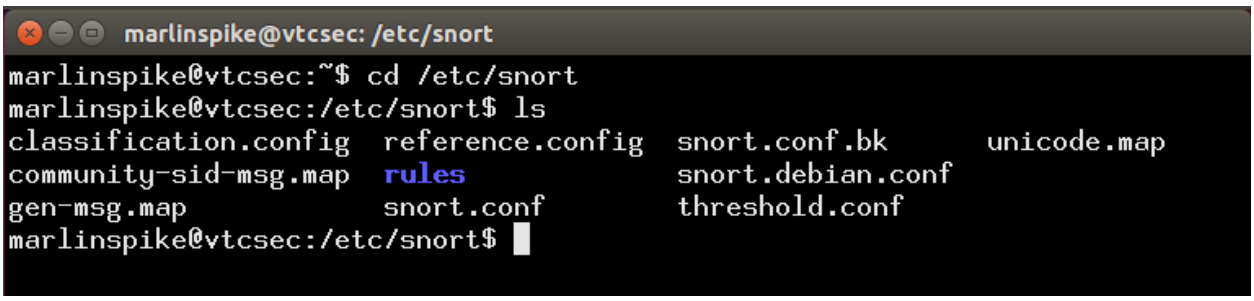


We use the command:
Ipt -F

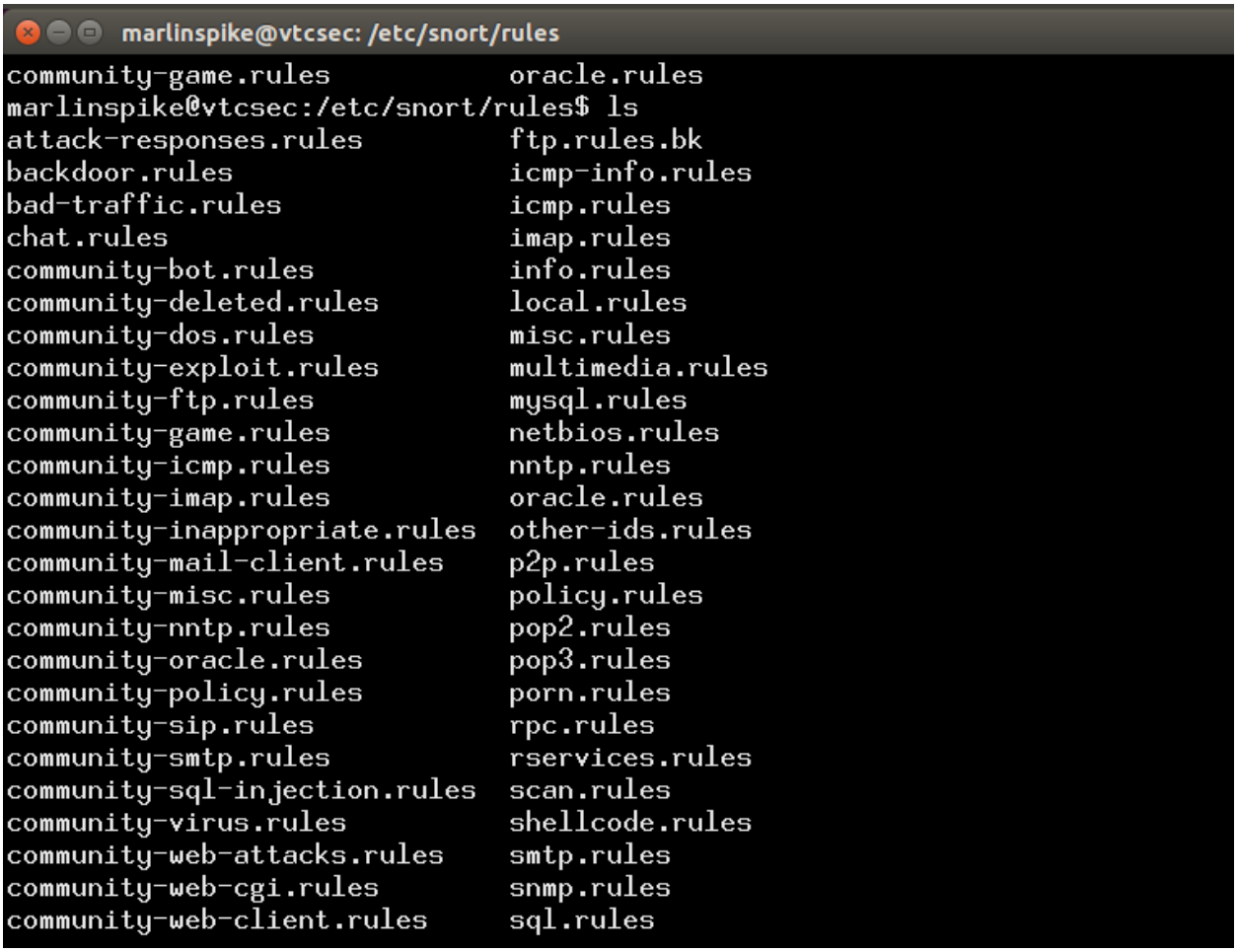
All the rules are deleted, and it goes back to the original settings.

Snort Intrusion Detection System (IDS)

We locate snort in the following directory:
cd /etc/snort

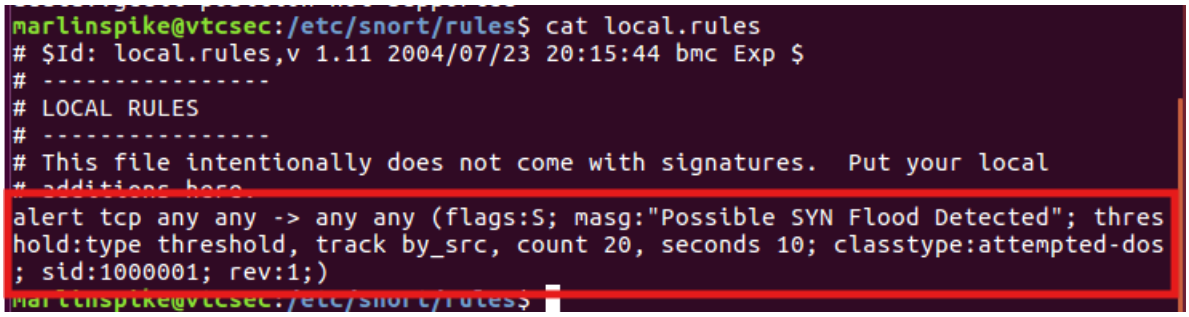


In the directory rules we can see that there are so many rules:



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

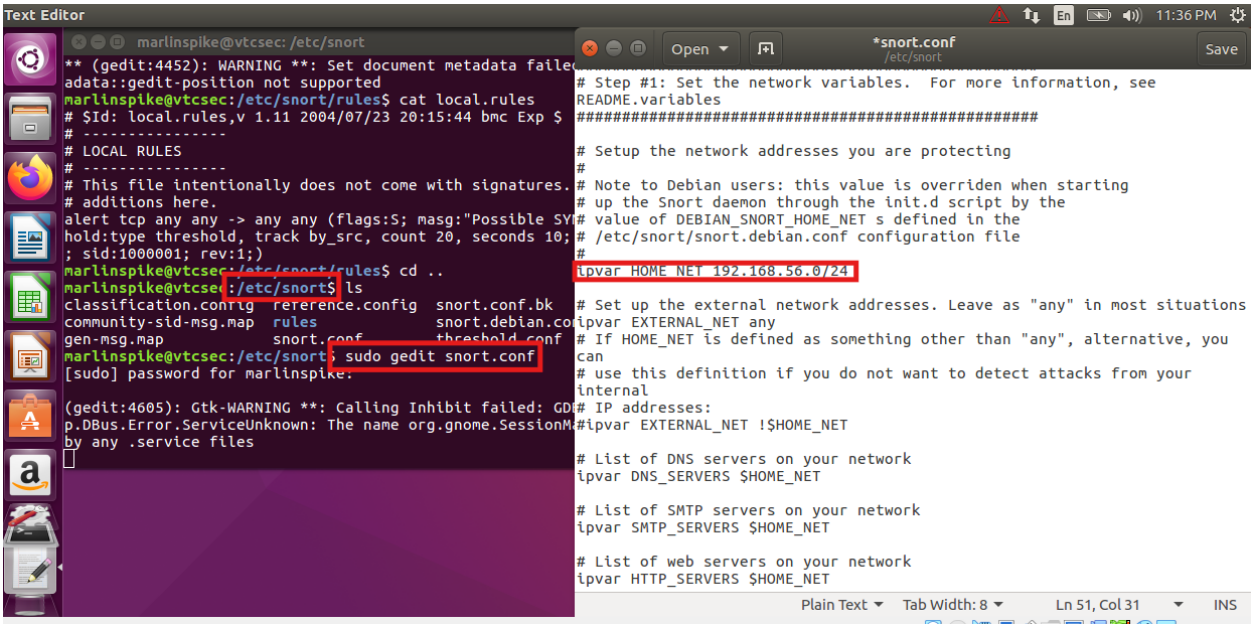
In the local.rules file in snort>rules, we add the following rule configuration:



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;)

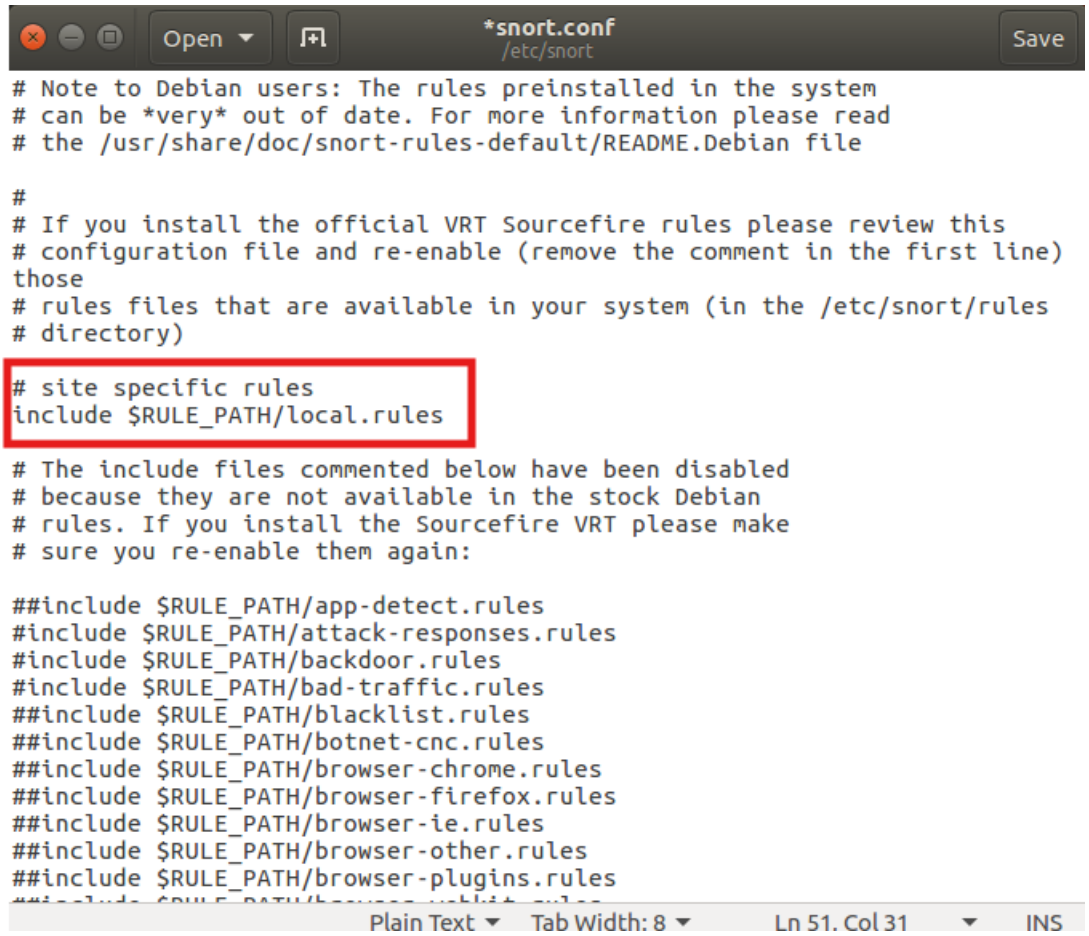
This Snort rule is designed to detect potential SYN flood attacks. A SYN flood occurs when many SYN requests are sent to overwhelm a server. The rule tracks SYN packets from a single source IP and triggers an alert if more than 20 SYN packets are detected in 10 seconds from that source, indicating a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

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

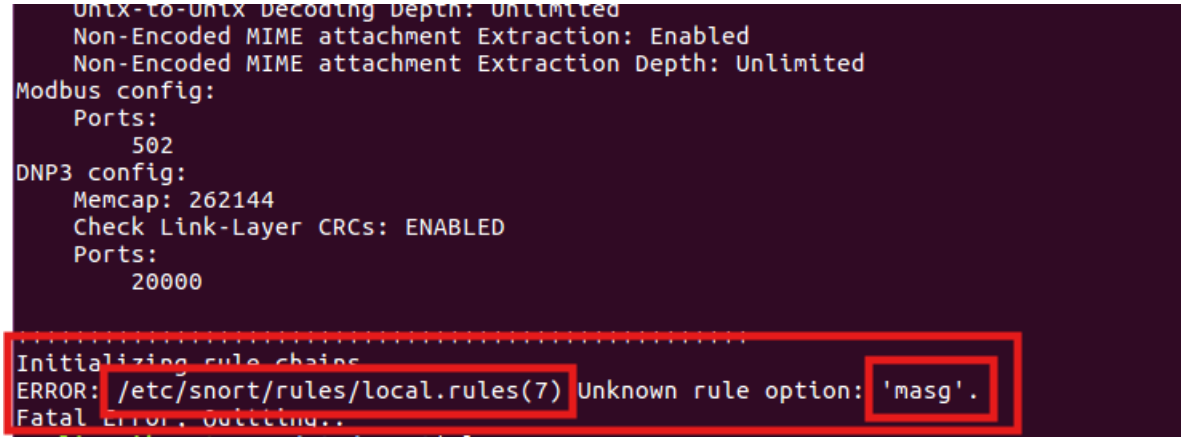


1. We change the ipvar HOME_NET value to the subnet value of the attacking machine.

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.

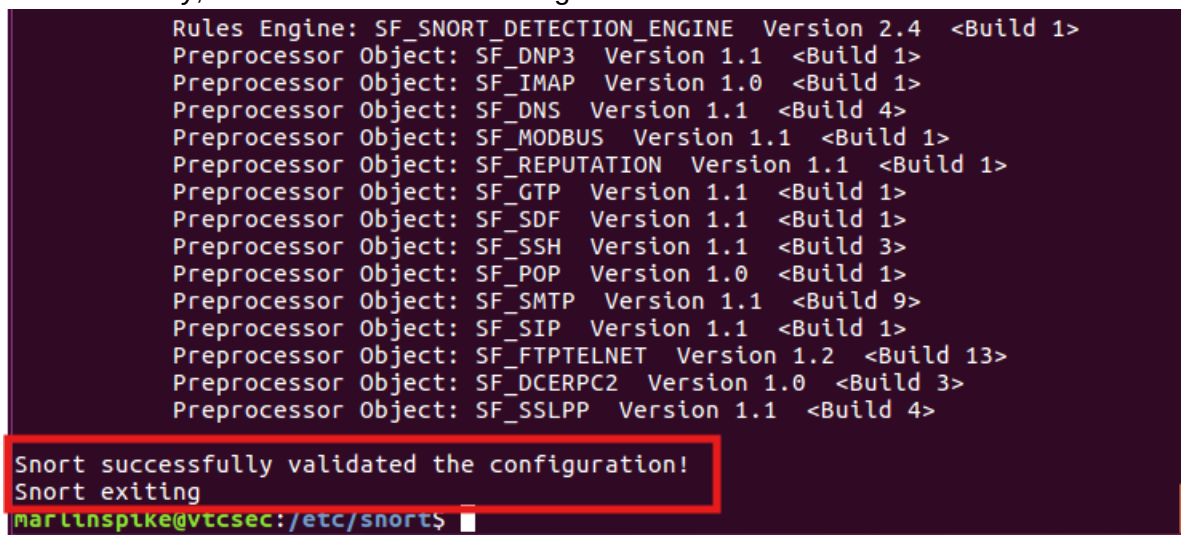


Next, we can test if the “snort.conf” i.e. the snort configuration is working or not. We use the command:
sudo snort -T -c /etc/snort/snort.conf



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.

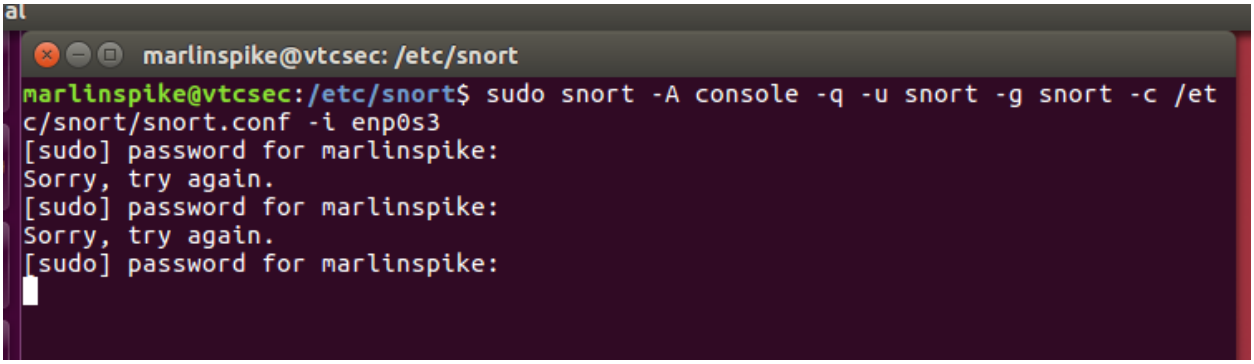
On the next try, we see that the snort configuration test was successful:



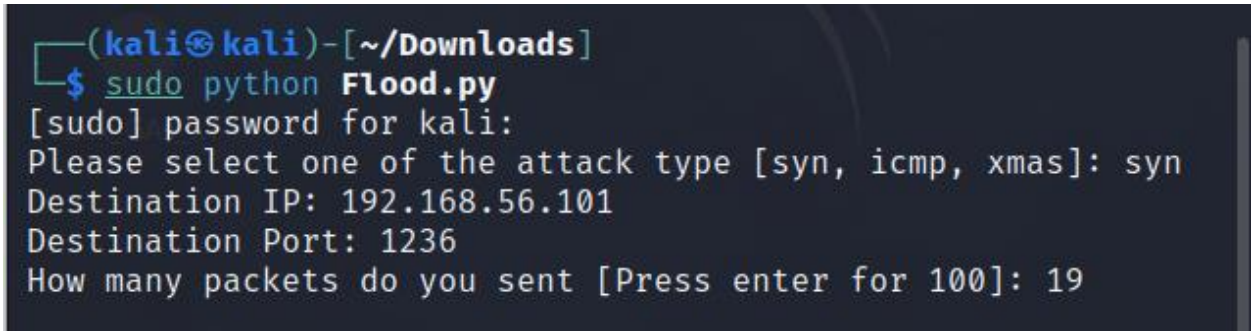
Now let us run the snort configuration with the command:
sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i enp0s3

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.

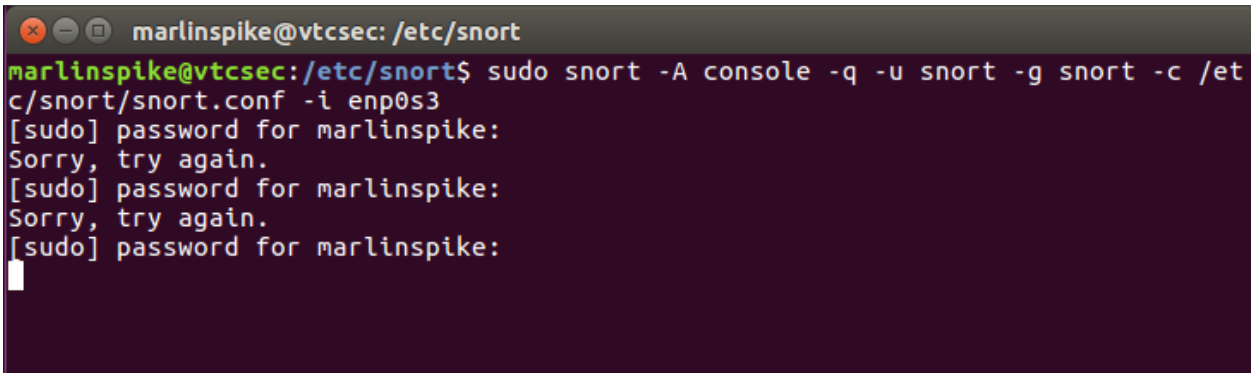
First, we run the command:



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

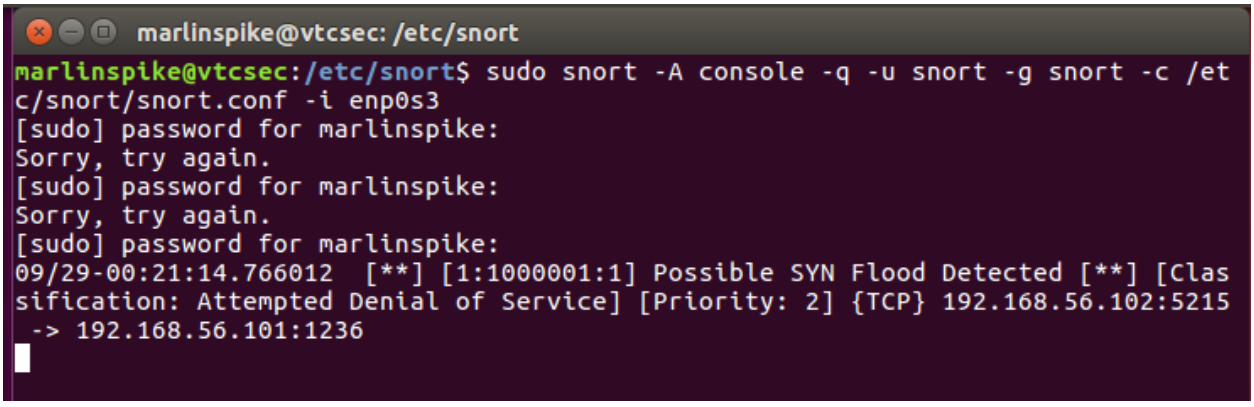


When we send 19 SYN packets.



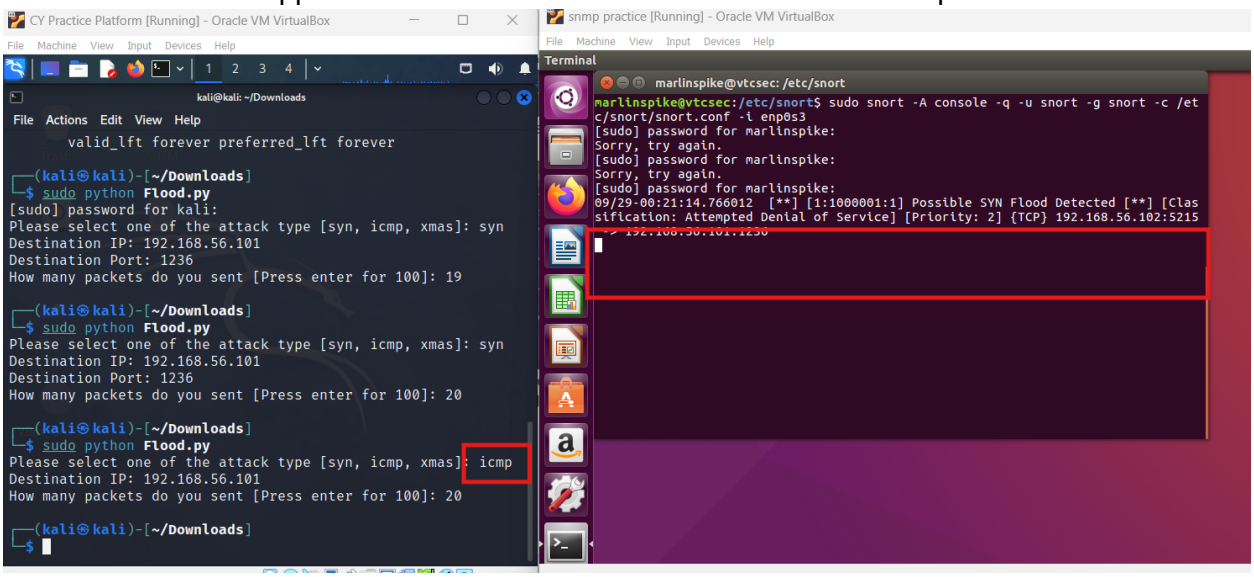
The Snort IDS detects nothing.

Then, we sent 20 SYN Packets.



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.

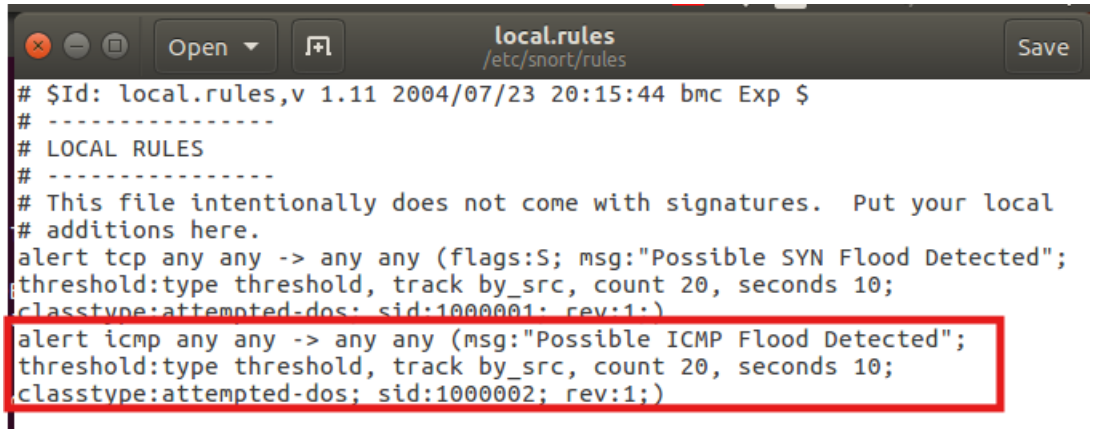
Let us check what happens when we initiate an ICMP flood attack of 20 packets.



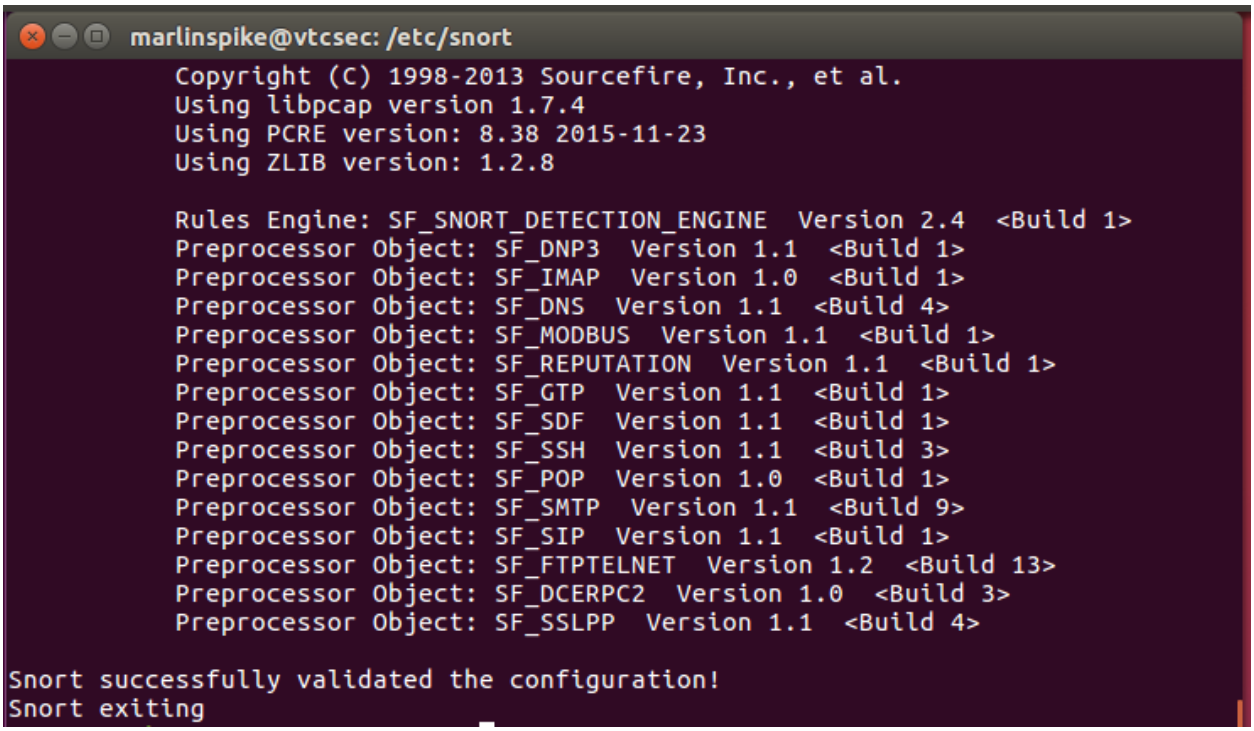
That does not trigger anything in the Snort IDS.

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;)



Now, let us test the configuration:

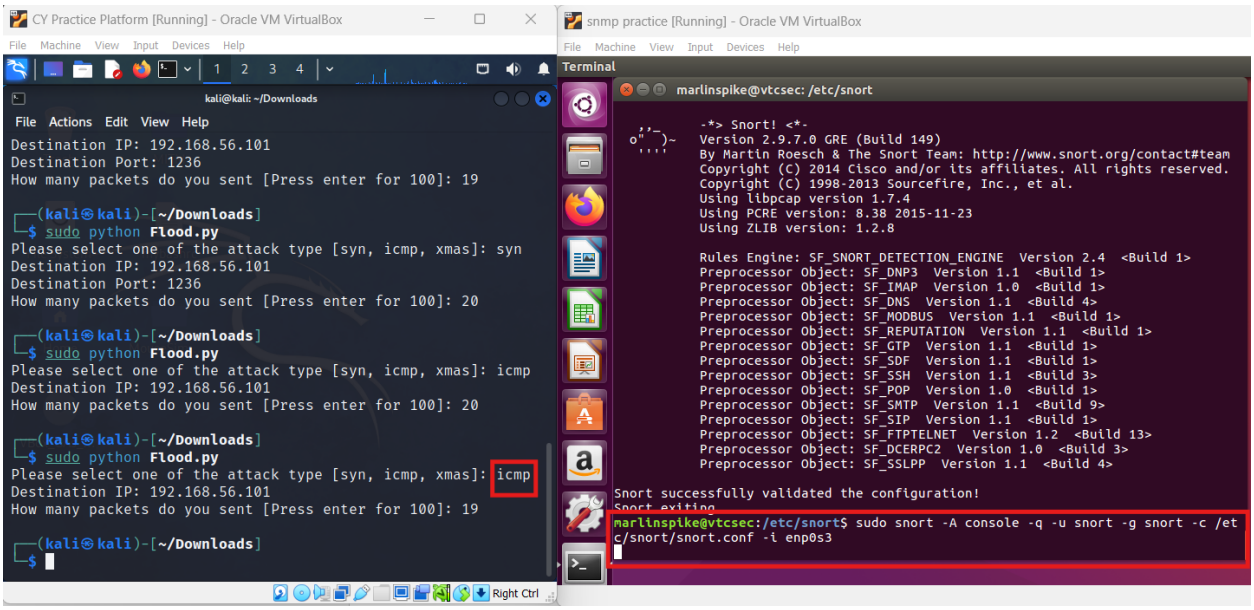


Test successful.

Let us run the configuration:

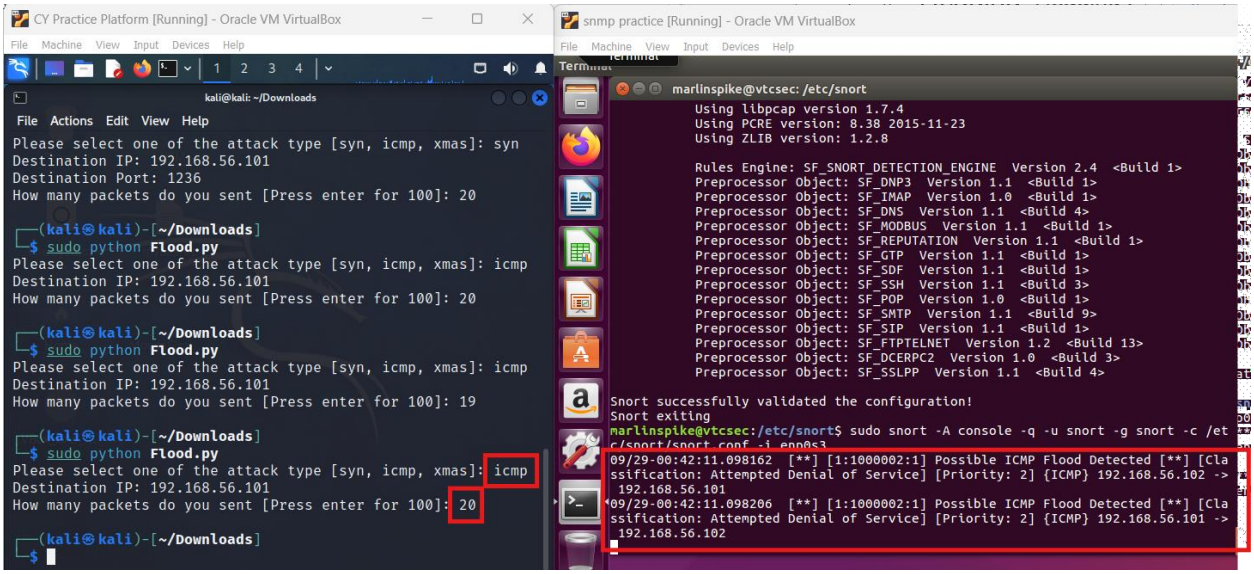
```
marlinspike@vtcsec:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /et
c/snort/snort.conf -i enp0s3
```

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



No incident detected in the Snort IDS.

Let us do the ICMP flood attack with 20 packets:



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.