INTERNSHIP PROJECT REPORT

ON

NIDS: NETWORK INTRUSION DETECTION SYSTEM

USING SNORT

AT

INFOTACT SOLUTIONS

Electronic city Bengaluru ,Karnataka, PIN 560100

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE TWO-MONTH CYBERSECURITY INTERNSHIP PROGRAMME

SUBMITTED BY:

Thariq Hussain A (Lead)

Anurag V K

Roshiya Suresh

Unmesh Bharat

MENTOR:

Mr. Vasudev Jha

Acknowledgment

We would like to express our sincere gratitude **to Infotact Solutions** for providing us the opportunity to work on the project titled "**Network Intrusion Detection System (NIDS) using Snort."**
This internship has been an invaluable learning experience that helped us gain practical exposure to real-world cybersecurity practices and hands-on implementation of intrusion detection systems.

We extend my heartfelt thanks to our mentor **Mr Vasudev Jha** and the entire **Infotact Solutions team** for their constant guidance, encouragement, and technical support throughout the project duration. Their expertise and feedback greatly contributed to my understanding and skill development.

We would also like to thank our teammates for their cooperation and teamwork, which played a vital role in successfully completing this phase of the project.

Finally, We are grateful to **Infotact Solutions** for creating a professional and supportive environment that enhanced both my technical knowledge and practical learning experience in cybersecurity.

Table Of Content

Abstract

This project presents the implementation of a **Network Intrusion Detection System (NIDS) using Snort** in a controlled lab environment to monitor network traffic and detect malicious activities. Initially, Snort was installed and configured using default rule sets to understand its architecture and baseline detection capabilities through simulated attacks such as TCP SYN scans and ICMP floods.

In the later phase, **custom Snort rules were developed and tuned** to detect advanced network and application-layer attacks while reducing alert noise. The results demonstrate that Snort, when properly configured and enhanced with custom rules, is an effective intrusion detection solution suitable for practical SOC and blue team security monitoring.

Objective

The primary objective of this project is to design, configure, and implement a Network Intrusion Detection System (NIDS) using Snort, an open-source intrusion detection tool, to monitor network traffic, detect malicious activities, and generate real-time alerts for improved security visibility.

Through this project, the intern aims to:

- Understand the fundamental working of Intrusion Detection Systems (IDS) and their role in modern cybersecurity operations.
- Install and configure Snort in a controlled lab environment using Ubuntu.
- Monitor and analyse live network traffic to identify anomalies, reconnaissance activities, and intrusion patterns.
- Simulate real-world network and application-layer attacks (such as ICMP floods, port scans, SSH brute-force attempts, and web-based attacks) to evaluate Snort's detection capabilities.
- Develop, implement, and validate custom Snort rules to detect advanced attack scenarios beyond default signatures.
- Perform rule tuning and thresholding to reduce alert noise and false positives, improving the accuracy and relevance of generated alerts.
- Analyse Snort alert logs and rule structures to understand how detection logic maps to observed attack behaviour.

The project ultimately focuses on enhancing practical knowledge in intrusion detection, rule-based threat identification, alert tuning, and incident analysis, thereby bridging the gap between theoretical cybersecurity concepts and real-world defensive and SOC-oriented implementations.

WEEK 1:

1. Introduction

With the rapid growth of cyber threats, intrusion detection has become a vital component of modern network security. This report presents the design, implementation, and evaluation of a **Network Intrusion Detection System (NIDS) using Snort**, carried out as part of a two-month Cyber Security Internship.

The project was implemented in a controlled virtual lab environment using an Ubuntu system as the monitored target and a Kali Linux system as the attacker. During the initial phase, Snort was installed and configured with default rule sets to understand its architecture, traffic inspection process, and baseline detection capabilities. In the later phase, custom Snort rules were developed and tuned to detect a wide range of network-level and application-level attacks. Through real-time attack simulations, alert analysis, and log inspection, the project demonstrates Snort's effectiveness as a NIDS when enhanced with custom rule development for practical security monitoring.

2. Lab Environment Setup

To build a controlled and realistic cybersecurity testing environment, I set up a small virtual lab consisting of two main virtual machines and a few essential tools.

A, Virtual Machines

Ubuntu Desktop 22.04.5 LTS: This machine serves as the primary target system and has **Snort** installed. Snort acts as our Intrusion Detection System (IDS), monitoring network traffic and helping us identify suspicious activities.

Kali Linux: This machine functions as the **attacker's workstation**. It comes preloaded with a wide range of penetration testing tools, making it ideal for simulating real-world attacks against the Ubuntu server.

B, Virtualization Platform

Both machines are hosted on **VirtualBox/VMware**, running in **Bridged Networking Mode**. This configuration ensures that each VM behaves as if it were a separate device connected to the same local network, allowing them to communicate seamlessly with each other just like physical machines would.

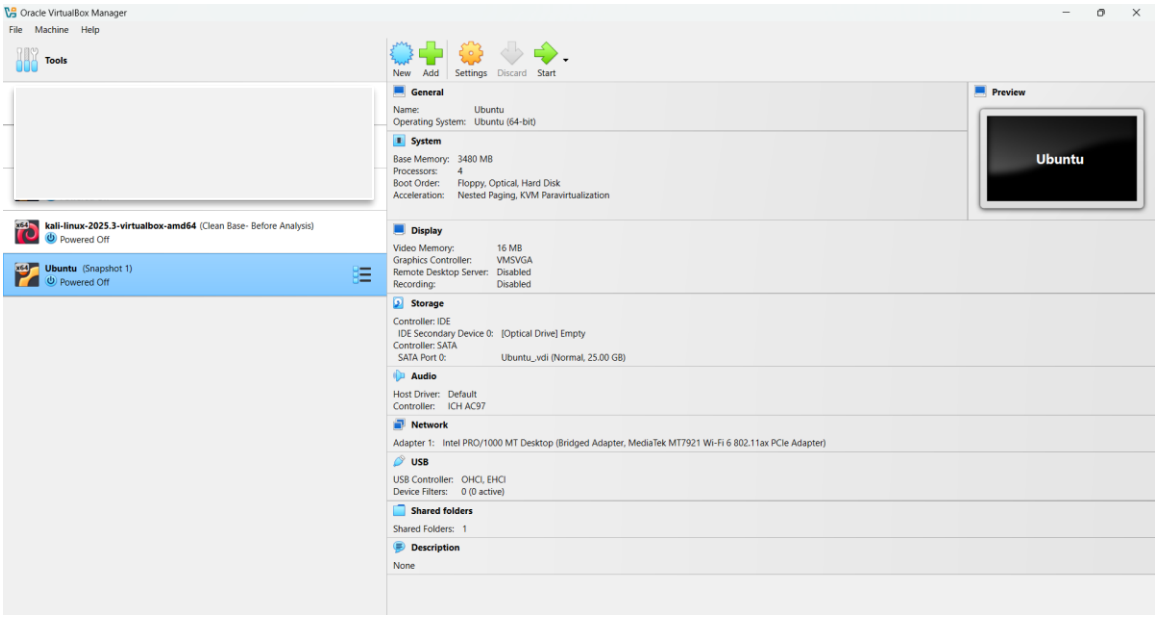
C, Tools Utilized

Snort: Used for real-time traffic monitoring and intrusion detection.
Nmap: A powerful network scanner for discovering open ports and services on the target machine.

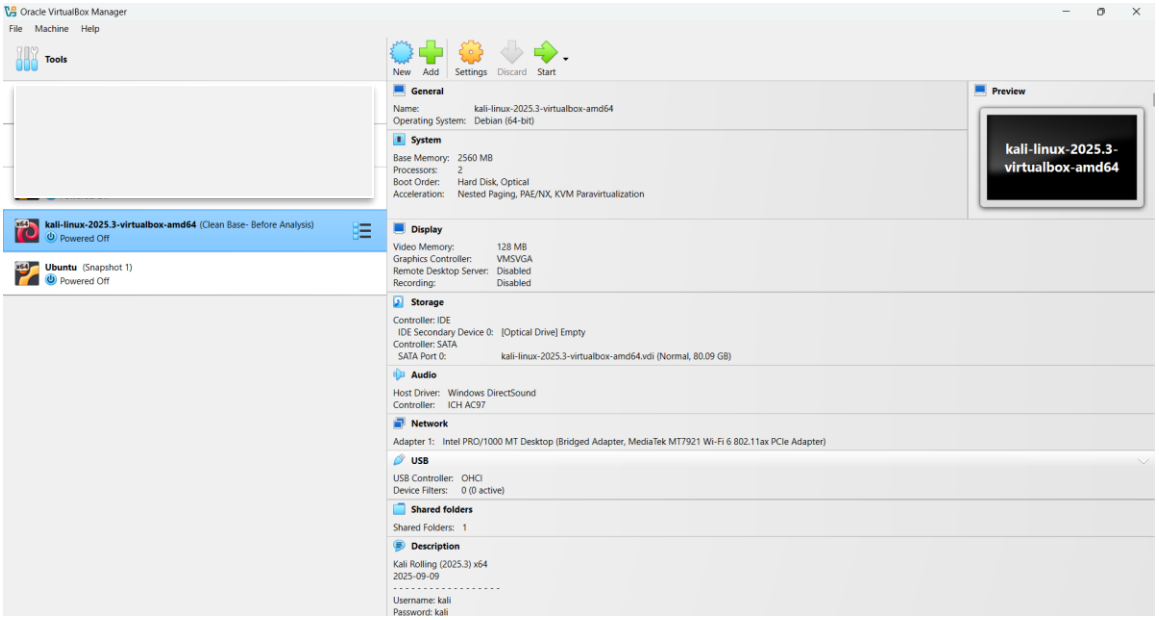
3. Implementation

Virtual Machines: Ubuntu Desktop (Snort installed), Kali Linux (Attacker machine)
Virtualization: VMware in Bridged Mode

Ubuntu (Snort installed):-



Kali Linux (Attacker machine):-

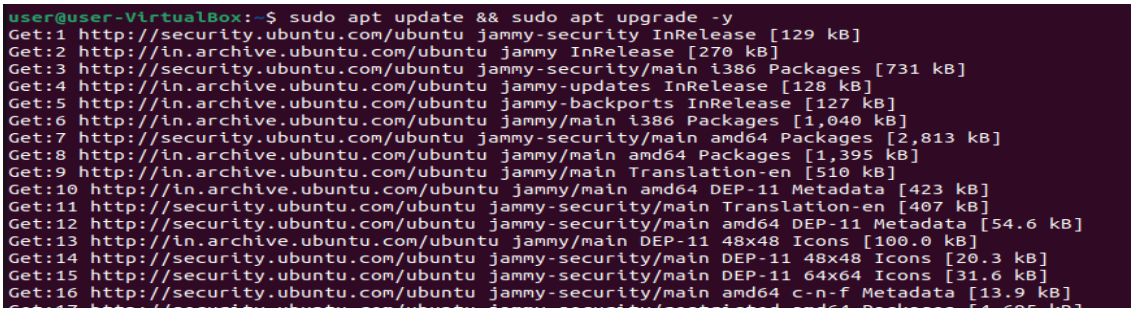


4. Snort Setup and Initial Verification

4.1 System Update and Upgrade

Before installing Snort, the system repositories and packages were updated using the following command to ensure the system had the latest dependencies:

sudo apt update && sudo apt upgrade -y

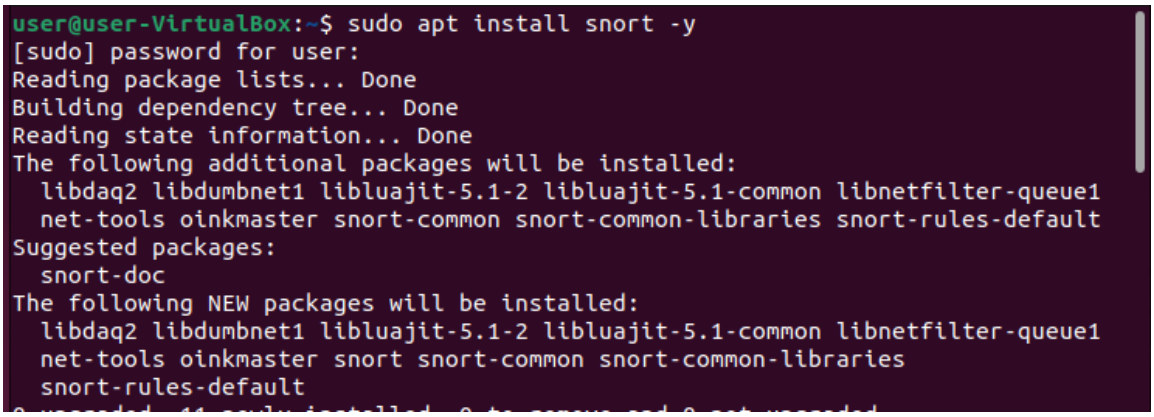


This step ensures all existing packages are up-to-date and compatible IMwith new installations.

4.2 Installing Snort and Dependencies

Snort was installed along with its required dependencies such as libdumbnet1, libluajit, and snort-common.
The installation command used was:

sudo apt install snort -y



This installs:

- Snort binary and service

- Default rule sets (snort-rules-default)
- Configuration files under /etc/snort/
- Logging directory under /var/log/snort/

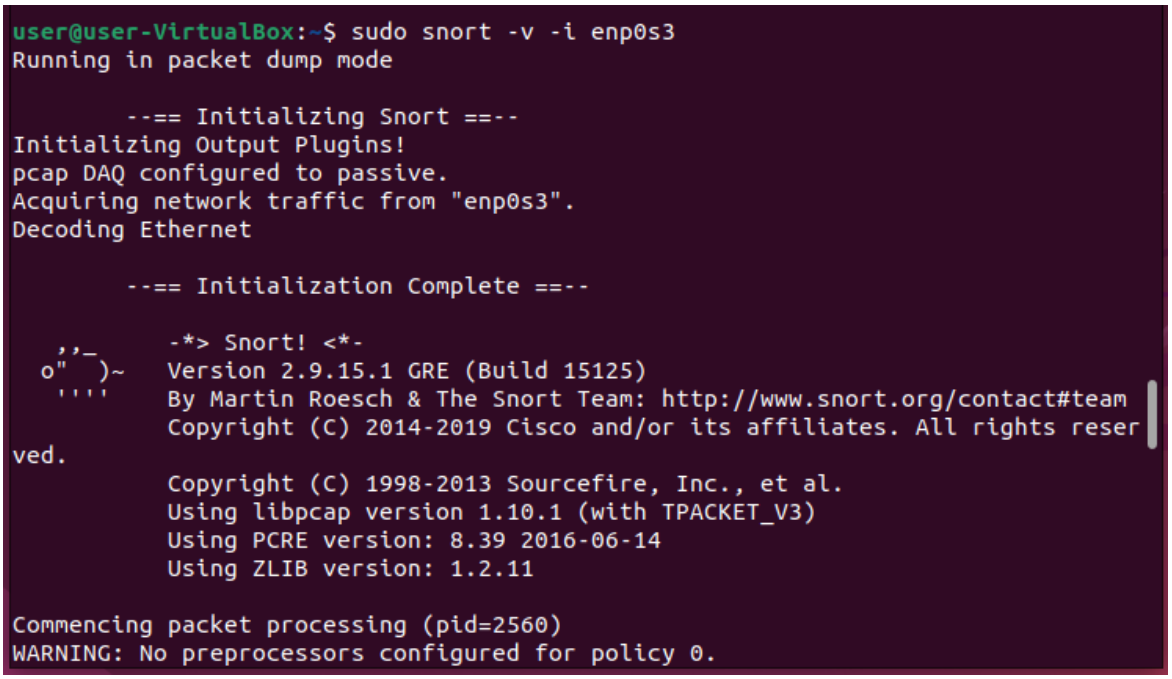
4.3. Installation verification

To confirm:

snort -v -i enp0s3

-v – verbose

-i – interface (enp0s3)



confirmed the installed Snort version.

To Verify:

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

-T – Test mode

-c – config file path

```
user@user-VirtualBox:~$ sudo snort -T -c /etc/snort/snort.conf
[sudo] password for user:
Running in Test mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
Initializing Plug-ins!
Parsing Rules file "/etc/snort/snort.conf"
PortVar 'HTTP_PORTS' defined :  [ 80:81 311 383 591 593 901 1220 1414 1741 1830
2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 7777
7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300
8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080 50002
55555 ]
PortVar 'SHELLCODE_PORTS' defined :  [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined :  [ 1024:65535 ]
PortVar 'SSH_PORTS' defined :  [ 22 ]
PortVar 'FTP_PORTS' defined :  [ 21 2100 3535 ]
PortVar 'SIP_PORTS' defined :  [ 5060:5061 5600 ]
PortVar 'FILE_DATA_PORTS' defined :  [ 80:81 110 143 311 383 591 593 901 1220 14
14 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:71
45 7510 7777 7779 8000 8008 8014 8028 8080 8085 8088 8090 8118 8123 8180:8181 82
43 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444
 41080 50002 55555 ]
```

```
           Using libpcap version 1.10.1 (with TPACKET_V3)
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

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

Snort successfully validated the configuration!
Snort exiting
user@user-VirtualBox:~$
```

Returned a successful configuration validation message

Week 2:

5. Snort Config Directory Overview

After successful installation, Snort's main configuration files are located in the directory:

cd /etc/snort/

This directory contains all essential files that define how Snort operates, processes packets, and manages rule sets.

Snort.conf

The main configuration file of Snort. It defines network variables, rule paths, logging methods, and preprocessing modules. All Snort operations depend on the parameters defined here

rules

Directory containing default and custom Snort rules. These rules define what traffic patterns or signatures Snort should detect and alert on.

classification.config

Maps alerts to predefined categories (e.g., "Attempted Information Leak", "Network Trojan Activity") for better readability in logs.

reference.config

Links rule references to external documentation such as CVE, Bugtraq, or Emerging Threats databases.

threshold.conf

Used for event filtering and thresholding to reduce false positives by limiting repetitive alerts.

file_magic.conf

Contains file-type detection patterns based on magic numbers. Helps Snort identify payload types in files.

unicode.map

Handles character encoding normalization, preventing evasion through Unicode encoding tricks.

Community-sid-msg.map

Maintain mappings between rule SIDs (Signature IDs) and their descriptive messages. Useful for correlating alerts to rules.

Snort.debian.conf

Debian-specific service configuration file controlling how Snort runs as a daemon on Ubuntu systems.

6. Configuring Monitored IP range & interface selection in Snort

6.1 Purpose: tell Snort what network(s) you consider “local” so rules that reference $HOME_NET work properly.

Backup before editing Snort configuration

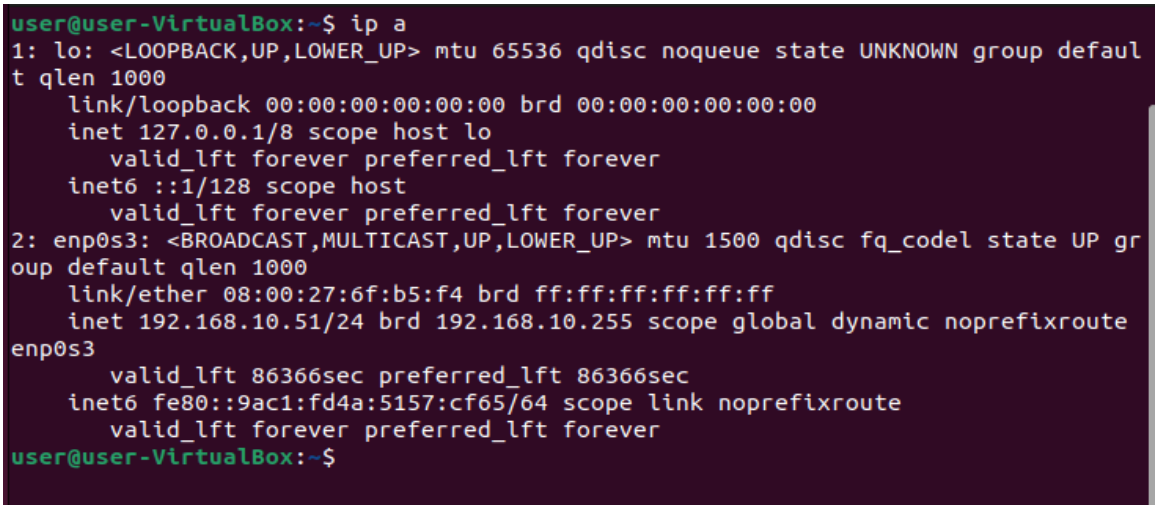
Backing up Snort config before making changes. This prevents accidental downtime if a syntax error or bad rule stops Snort from starting, and makes it easy to revert or compare changes.

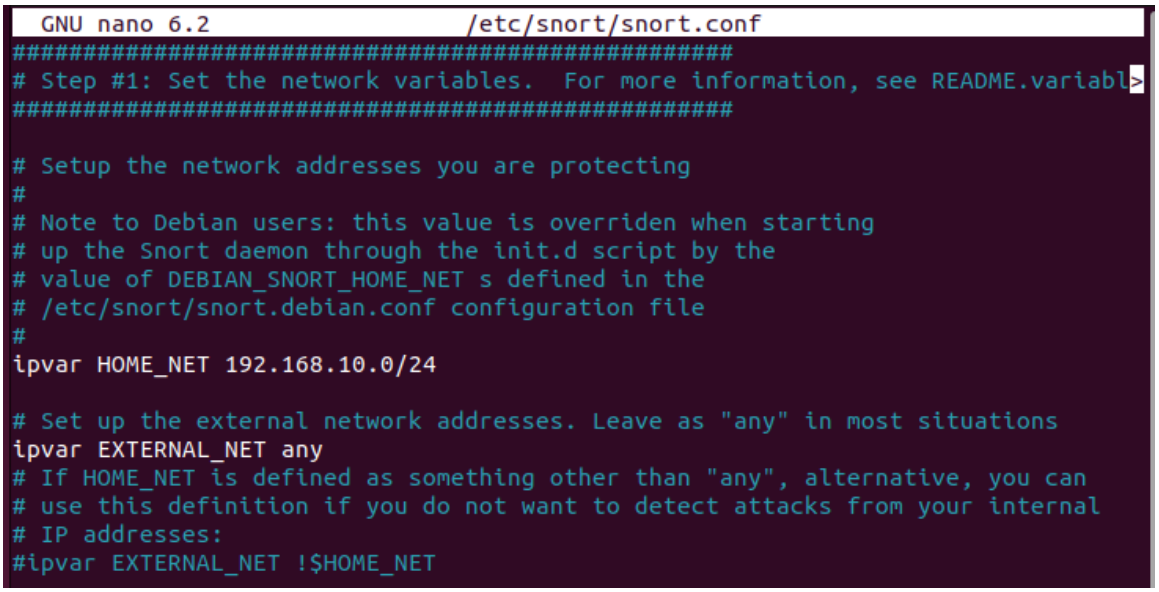
sudo cp /etc/snort/snort.conf /etc/snort/snort.conf.bak

Edit snort.conf and set HOME_NET

sudo nano /etc/snort/snort.conf

Find the var HOME_NET line and change it to the monitored range you want (depends on ip)





Save and exit.

Validate with : **sudo snort -T -c /etc/snort/snort.conf**

If successful, Snort will report parsing the rules and no fatal errors. Capture that output for the report.

6.2 Run Snort with selected interface and HOME_NET

Start Snort to monitor live traffic:

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

-A console → display alerts directly on terminal.

-q → quiet mode (suppresses banner)

-c → specifies the configuration file to use.

-i → interface to monitor.

Week 3:

7. Simulated Attacks & Snort Alert Review

Tasks covered:

- Simulate attacks (ICMP flood, SYN)
- Observe Snort alerts (live)
- Understand Snort alert formats
- Review alert logs

Commands used:

Start Snort (victim – console alerts):

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

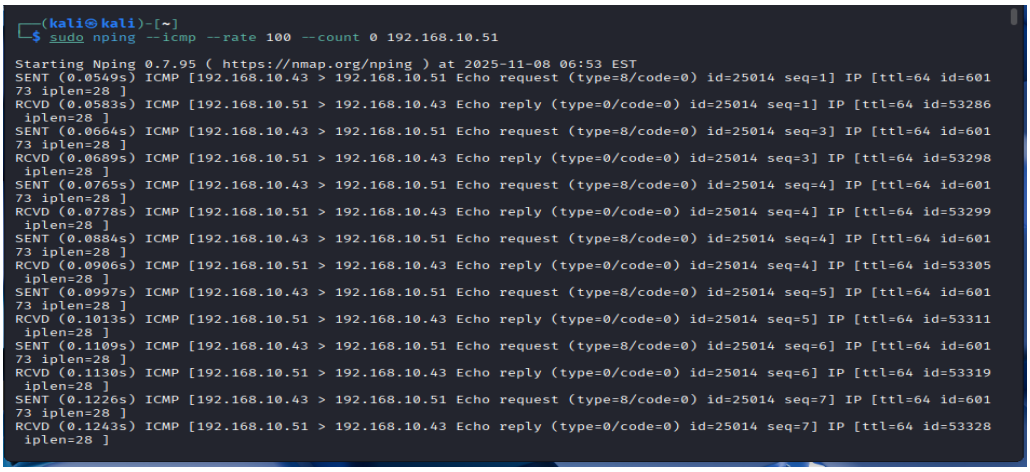
ICMP flood (attacker – nping):

sudo nping --icmp --rate 100 --count 0 192.168.10.51

TCP SYN scan (attacker – nmap):

sudo nmap -sS -Pn 192.168.10.51

7.1 a) ICMP Flood (Attacker Output)

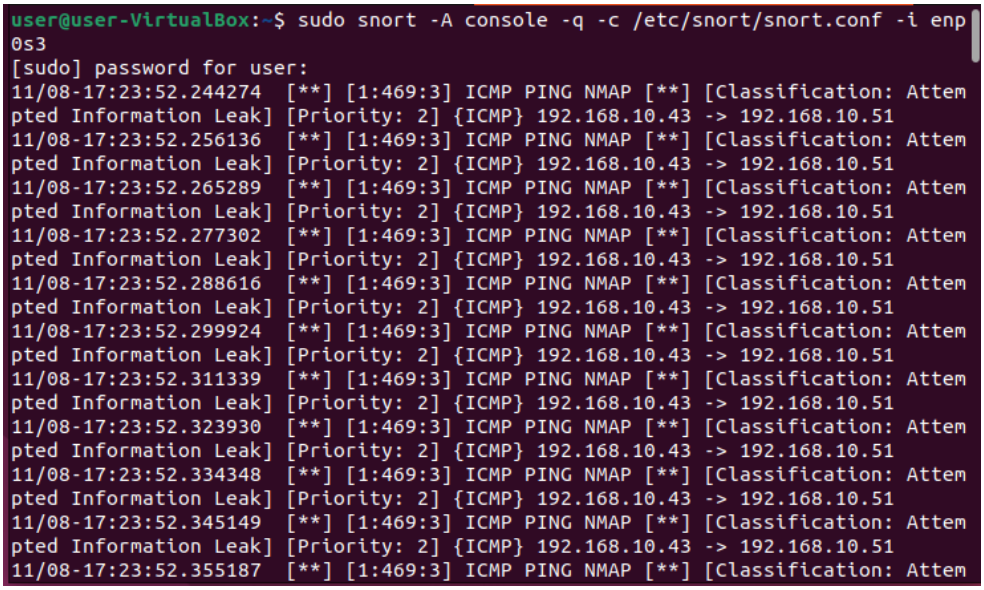


Executed an ICMP ping flood from the attacker machine.

Attacker – nping --icmp --rate 100 --count 0 192.168.10.51 running, sending rapid ICMP echo requests.

7.1 b) Snort Console – ICMP Alerts (Live View)

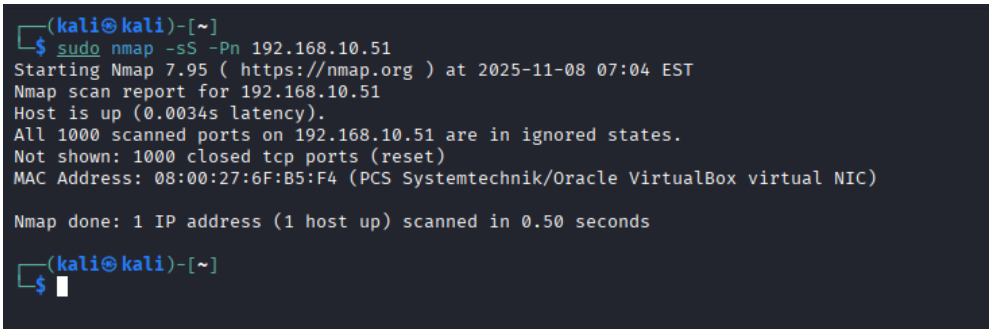
While the flood was running, Snort displayed repeated ICMP alerts such as “ICMP PING NMAP”. Each entry shows timestamp, [gid:sid:rev], msg, classification, priority, and packet direction.



Victim – Snort console showing live ICMP alerts during ping flood.

7.2 a) TCP SYN scan (Attacker Output)

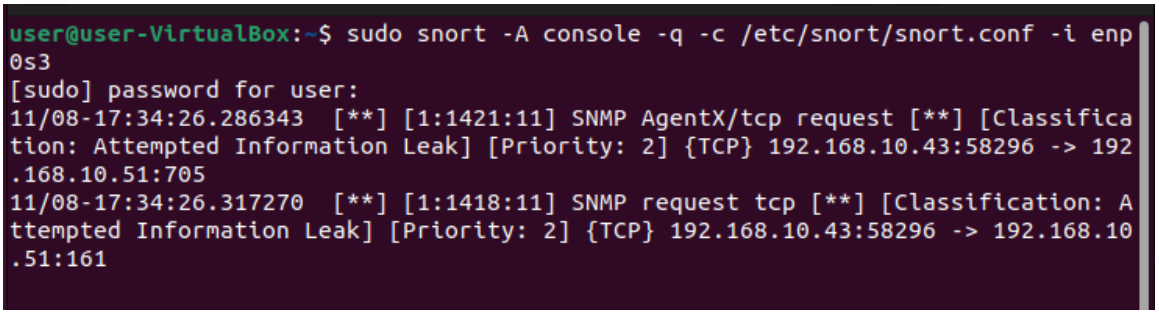
The attacker terminal shows the nmap SYN scan command and results, confirming the scan execution that Snort detected.



Attacker – nmap -sS -Pn 192.168.10.51 scan output.

7.2 b) Snort Console – TCP SYN Scan Alerts

After the ICMP test, a TCP SYN scan was performed using nmap. Snort triggered TCP/SNMP-related alerts indicating port scanning activity.

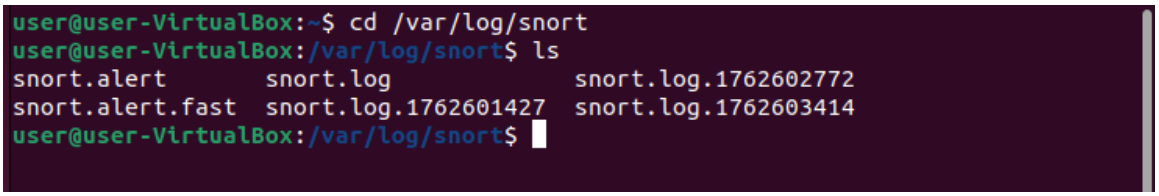


Victim – Snort console showing TCP/SYN alerts triggered by nmap -sS -Pn.

7.3 Snort Log Directory Structure

After running Snort, the log directory (/var/log/snort) contained multiple files:

- snort.alert – main detailed alert log
- snort.alert.fast – summary alert log
- snort.log.<timestamp> – binary packet capture logs for each session



Screenshot of /var/log/snort showing alert and binary log files.

7.4 Log File Review and Verification

Viewed alerts to confirm they matched live console events. The files were opened using cat and tail commands. Binary log files can be read with Snort or tcpdump or even Wireshark for packet-level analysis.

7.4.1 To review alert files in readable text form, the following command was used:

- sudo cat /var/log/snort/snort.alert.fast
- sudo tail -f /var/log/snort/snort.alert.fast (real time)

```
user@user-VirtualBox:~$ cat /var/log/snort/snort.alert.fast
11/08-15:58:55.960782  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [
Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.10.40:4
2151 -> 239.255.255.250:1900
11/08-15:59:07.787762  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [
Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.10.40:5
6654 -> 239.255.255.250:1900
11/08-16:12:27.845748  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {UDP} 0.0.0.0:68 -> 255.255.255.255:
67
11/08-16:12:27.865882  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {IPV6-ICMP} :: -> ff02::16
11/08-16:12:27.925622  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {IPV6-ICMP} :: -> ff02::16
11/08-16:12:27.931632  [**] [1:527:8] BAD-TRAFFIC same SRC/DST [**] [Classificat
ion: Potentially Bad Traffic] [Priority: 2] {IPV6-ICMP} :: -> ff02::1:ff6c:37d7
11/08-16:16:52.234846  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [
Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.10.37:4
5401 -> 239.255.255.250:1900
11/08-16:17:52.344012  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [
Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.10.37:4
5401 -> 239.255.255.250:1900
11/08-16:18:52.250329  [**] [1:1917:6] SCAN UPnP service discover attempt [**] [
Classification: Detection of a Network Scan] [Priority: 3] {UDP} 192.168.10.37:4
```

This displayed summarized alerts generated by Snort for both attacks, confirming successful detection of the ICMP flood and TCP SYN scan attempts.

7.4.2 To analyze binary packet logs, the following commands were executed:

- sudo snort -r /var/log/snort/snort.log.<timestamp>

The snort -r command decoded and displayed Snort's interpreted log view, showing ICMP echo requests and TCP connection attempts flagged during the attack simulation

Snort.log.1762602772 (ICMP echo requests)

```
Commencing packet processing (pid=2488)
WARNING: No preprocessors configured for policy 0.
11/08-17:23:52.244274 192.168.10.43 -> 192.168.10.51
ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28
Type:8  Code:0  ID:25014   Seq:1  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
11/08-17:23:52.256136 192.168.10.43 -> 192.168.10.51
ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28
Type:8  Code:0  ID:25014   Seq:3  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
11/08-17:23:52.265289 192.168.10.43 -> 192.168.10.51
ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28
Type:8  Code:0  ID:25014   Seq:4  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
11/08-17:23:52.277302 192.168.10.43 -> 192.168.10.51
ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28
Type:8  Code:0  ID:25014   Seq:4  ECHO
```

Snort.log.1762603414 (TCP SYN)

```
Commencing packet processing (pid=2640)
WARNING: No preprocessors configured for policy 0.
11/08-17:34:26.286343 192.168.10.43:58296 -> 192.168.10.51:705
TCP TTL:47 TOS:0x0 ID:37860 IpLen:20 DgmLen:44
******S* Seq: 0xF3C38956  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) => MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

WARNING: No preprocessors configured for policy 0.
11/08-17:34:26.317270 192.168.10.43:58296 -> 192.168.10.51:161
TCP TTL:37 TOS:0x0 ID:11578 IpLen:20 DgmLen:44
******S* Seq: 0xF3C38956  Ack: 0x0  Win: 0x400  TcpLen: 24
TCP Options (1) => MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+


===============================================================================
Run time for packet processing was 0.729 seconds
Snort processed 2 packets.
Snort ran for 0 days 0 hours 0 minutes 0 seconds
   Pkts/sec:            2
===============================================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       790528
```

8. Snort Log Structure Format

Snort generates alert logs for every detected intrusion attempt or suspicious network activity.
These logs are stored in /var/log/snort/ and contain structured information about each alert.

A typical alert format includes the **generator ID, rule ID, classification, priority, timestamp, source/destination IPs**, and **protocol details**.
The example below illustrates the structure of an ICMP ping flood alert detected during testing.

Example: ICMP echo request

[**] [1:469:3] ICMP PING NMAP [**]

[Classification: Attempted Information Leak] [Priority: 2]

11/09-17:23:52.244274 192.168.10.43 -> 192.168.10.51

ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28

Type:8 Code:0 ID:25014 Seq:1 ECHO

[**]	Marks the beginning and end of an alert entry
[1:469:3]	**Generator ID : Signature ID : Revision**
ICMP PING NMAP	**Rule message** that triggered the alert
[Classification: Attempted Information Leak]	Type of suspicious activity (based on classification.conf)
[Priority: 2]	Alert severity (1 = highest, 3 = low)
11/09-17:23:52.244274	**Timestamp** (when alert was generated)
192.168.10.43 -> 192.168.10.51	**Source and destination IP addresses**
ICMP TTL:64 TOS:0x0 ID:60173 IpLen:20 DgmLen:28	Basic IP header details

Type:8 Code:0 ID:25014 Seq:1 ECHO	**Protocol-specific details** (here, ICMP ping request)

This structured output allows analysts to quickly identify the **type of attack**, **involved hosts**, and **threat severity**, enabling effective network response.

Week 4:

9. Understanding of Snort Rules

Snort uses text rule files (stored under /etc/snort/rules/) to identify suspicious traffic. Each rule describes a detection condition and an action (e.g., alert, log, pass). In this section we inspect the default rule location, the files Snort includes at startup, analyze a sample rule field-by-field, and show how to map a Snort alert to the originating rule (via the rule SID).

Step 1: Locating Default Snort Rules

Snort stores its predefined detection rules inside the directory:

ls /etc/snort/rules/

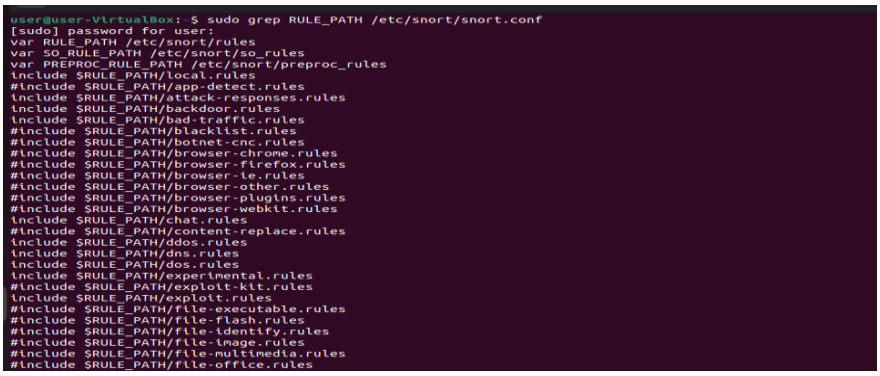
This command lists all the available rule files. Each file corresponds to a specific protocol or threat category (e.g., icmp.rules, ftp.rules, dos.rules, etc.).

Step 2: Viewing How Rules Are Loaded

The main configuration file /etc/snort/snort.conf contains the path of these rules.

Command used:

sudo grep RULE_PATH /etc/snort/snort.conf



Observation:
Snort dynamically loads these rule files using the include directive defined inside snort.conf.
This helps it automatically read protocol-specific detection logic during startup.

Step 3: Inspecting a Default Rule File

To read a particular rule set, open one of the files (for example, ICMP):

sudo cat /etc/snort/rules/icmp.rules | head -n 20

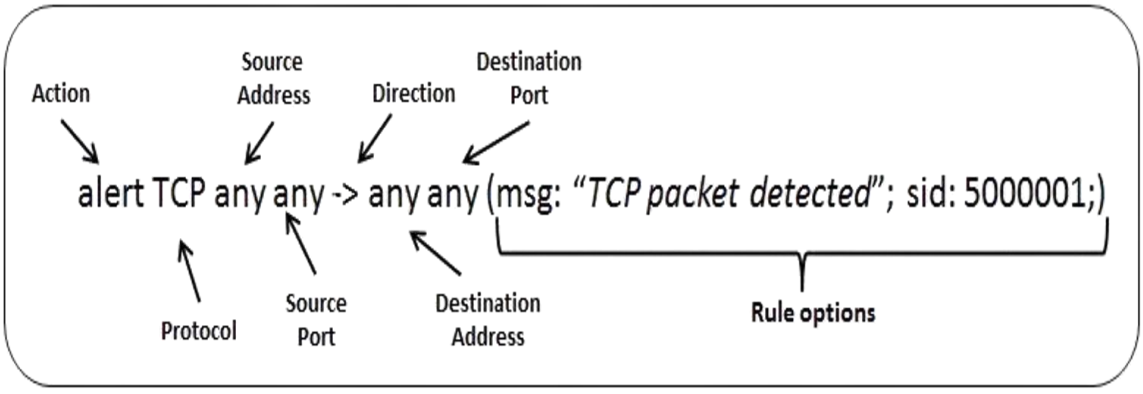
```
user@user-VirtualBox:~$ sudo cat /etc/snort/rules/icmp.rules
# Copyright 2001-2005 Sourcefire, Inc. All Rights Reserved
#
# This file may contain proprietary rules that were created, tested and
# certified by Sourcefire, Inc. (the "VRT Certified Rules") as well as
# rules that were created by Sourcefire and other third parties and
# distributed under the GNU General Public License (the "GPL Rules").  The
# VRT Certified Rules contained in this file are the property of
# Sourcefire, Inc. Copyright 2005 Sourcefire, Inc. All Rights Reserved.
# The GPL Rules created by Sourcefire, Inc. are the property of
# Sourcefire, Inc. Copyright 2002-2005 Sourcefire, Inc. All Rights
# Reserved.  All other GPL Rules are owned and copyrighted by their
# respective owners (please see www.snort.org/contributors for a list of
# owners and their respective copyrights).  In order to determine what
# rules are VRT Certified Rules or GPL Rules, please refer to the VRT
# Certified Rules License Agreement.
#
#
# $Id: icmp.rules,v 1.25.2.1.2.2 2005/05/16 22:17:51 mwatchinski Exp $
#------------
# ICMP RULES
#------------
#
# Description:
# These rules are potentially bad ICMP traffic.  They include most of the
# ICMP scanning tools and other "BAD" ICMP traffic (Such as redirect host)
#
# Other ICMP rules are included in icmp-info.rules

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP ISS Pinger"; itype:8; content:"ISSPNGRQ"; depth:32; reference:arachnids,158
; classtype:attempted-recon; sid:465; rev:3;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP L3retriever Ping"; icode:0; itype:8; content:"ABCDEFGHIJKLMNOPQRSTUVWABCDEF
GHI"; depth:32; reference:arachnids,311; classtype:attempted-recon; sid:466; rev:4;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Nemesis v1.1 Echo"; dsize:20; icmp_id:0; icmp_seq:0; itype:8; content:"|00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|"; reference:arachnids,449; classtype:attempted-recon; sid:467; rev:3;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP PING NMAP"; dsize:0; itype:8; reference:arachnids,162; classtype:attempted-
recon; sid:469; rev:3;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP icmpenum v1.1.1"; dsize:0; icmp_id:666 ; icmp_seq:0; id:666; itype:8; refer
ence:arachnids,450; classtype:attempted-recon; sid:471; rev:3;)
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP redirect host"; icode:1; itype:5; reference:arachnids,135; reference:cve,19
```

ICMP rule file content with alert icmp entries.

Observation:
The rules contain clear detection logic — keywords like alert, msg, sid, and rev define how Snort triggers alerts when suspicious ICMP packets are seen (e.g., ping floods or scans).

Step 4: Understanding Rule Structure



Example default rule extracted from TCP.rules:

Part	Meaning
alert	Action: Snort will generate an alert
TCP	Protocol inspected
$EXTERNAL_NET any -> $HOME_NET any	Traffic direction and scope
(msg:"TCP packet detected")	Message displayed when matched
sid:5000001	Unique Snort rule ID

NOTE: The rev field is not mandatory.

Step 5: Checking Variable Definitions

Variables like $HOME_NET and $EXTERNAL_NET are defined in snort.conf.
To view them:

sudo nano /etc/snort/snort.conf

```
###################################################
# 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 any

# 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

# List of sql servers on your network
ipvar SQL_SERVERS $HOME_NET

# List of telnet servers on your network
ipvar TELNET_SERVERS $HOME_NET
```

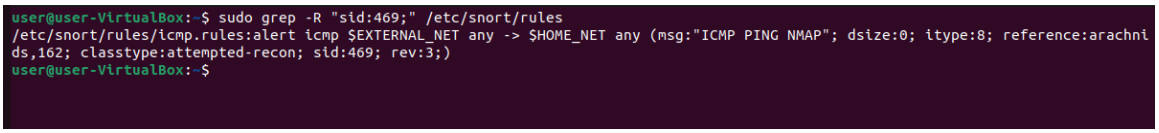
Showing $HOME_NET and $EXTERNAL_NET definitions inside snort.conf.

Observation:
These variables allow rules to stay generic and reusable across different networks.

Step 6: Mapping Alerts to Rules

When Snort runs in live mode and generates an alert, each alert includes a Signature ID (SID).
To find which rule produced it:

sudo grep -R "sid:469;" /etc/snort/rules/ -n



Output mapping SID 469 to the corresponding line in icmp.rules.

Observation:
This helps analysts trace alerts back to the rule that triggered them for verification or tuning.

WEEK 5-8 (2ND Month)

10, CUSTOM RULES

Rule No: 1 – SYN Scan Detection

Description:
This custom Snort rule is designed to detect TCP SYN scan activity, which is commonly used during the reconnaissance phase to identify open ports on a target system.

Rule Used:
A TCP alert rule with the SYN (S) flag set and stateless inspection to identify scan attempts targeting the internal network.

```
#1) Generic SYN scan
alert tcp any any -> $HOME_NET 1:20,24:79,81:109,111:142,144:3388,3390:65535 (msg:"SCAN: SYN scan detected";flags:S;flow:stateless;sid:4000001; rev:1;)
```

Attack Simulation:
A SYN scan was launched from Kali Linux (attacker) against an Ubuntu system (victim) using Nmap.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sS 192.168.10.51
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-20 05:45 EST
Nmap scan report for 192.168.10.51
Host is up (0.00083s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
23/tcp open  telnet
MAC Address: 08:00:27:6F:B5:F4 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.72 seconds
```

Detection Outcome:
Snort successfully captured and generated real-time alerts for multiple SYN packets sent to different destination ports, confirming effective detection of port scanning activity.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/23-20:59:49.314845  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:3306
11/23-20:59:49.314845  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:8888
11/23-20:59:49.314846  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:995
11/23-20:59:49.314846  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:993
11/23-20:59:49.314846  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:445
11/23-20:59:49.320708  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:443
11/23-20:59:49.314846  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:445
11/23-20:59:49.320708  [**] [1:4000001:1] SCAN: SYN scan detected [**] [Priority: 0] {TCP} 192.168.10.43:44333 -> 192.168.10.51:443
```

Rule No: 2 – FIN Scan Detection

Description:
This custom Snort rule detects TCP FIN scan activity, a stealth scanning technique commonly used by attackers to evade basic firewall and IDS detection during reconnaissance.

Rule Used:
A TCP alert rule configured to trigger on packets with the FIN (F) flag set using stateless inspection for effective scan identification.

```
# 2) FIN Scan Detection
alert tcp any any -> $HOME_NET any (msg:"Nmap FIN Scan"; flags:F; flow:stateless;sid:10000302; rev:1;)
```

Attack Simulation:
A FIN scan was performed from Kali Linux (attacker) against an Ubuntu system (victim) using Nmap.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sF 192.168.10.51
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-23 07:27 EST
Nmap scan report for 192.168.10.51
Host is up (0.0019s latency).
Not shown: 991 closed tcp ports (reset)
PORT     STATE         SERVICE
21/tcp   open|filtered ftp
22/tcp   open|filtered ssh
23/tcp   open|filtered telnet
80/tcp   open|filtered http
110/tcp  open|filtered pop3
143/tcp  open|filtered imap
993/tcp  open|filtered imaps
995/tcp  open|filtered pop3s
3389/tcp open|filtered ms-wbt-server
MAC Address: 08:00:27:6F:B5:F4 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
```

Detection Outcome:
Snort successfully generated real-time alerts for FIN-flagged TCP packets targeting multiple ports, confirming accurate detection of FIN scan attempts.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/23-17:57:17.414498  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:80
11/23-17:57:17.414498  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:80
11/23-17:57:17.414499  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:3306
11/23-17:57:17.414499  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:3306
11/23-17:57:17.414499  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:587
11/23-17:57:17.414499  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:587
11/23-17:57:17.416248  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:21
11/23-17:57:17.416248  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:21
11/23-17:57:17.416248  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:21
11/23-17:57:17.416248  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:1720
11/23-17:57:17.416248  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:1720
11/23-17:57:17.416248  [**] [1:100007332:1] RDP: Real Conection  Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:3389
11/23-17:57:17.416248  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:3389
11/23-17:57:17.416248  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:3389
11/23-17:57:17.416248  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:199
11/23-17:57:17.416248  [**] [1:621:7] SCAN FIN [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:50716 -> 192.168.10.51:199
11/23-17:57:17.416248  [**] [1:10000302:1] Nmap FIN Scan [**] [Priority: 0] {TCP} 192.168.10.43:50716 -> 192.168.10.51:110
```

Rule No: 3 – Xmas Scan Detection

Description:
This custom Snort rule detects TCP Xmas scan activity, a stealth reconnaissance technique where multiple TCP flags are set to identify open or filtered ports.

Rule Used:
A TCP alert rule configured to trigger on packets with FIN, PSH, and URG (FPU) flags using stateless inspection.

```
# 3) Xmas Scan Detection
alert tcp any any -> $HOME_NET any (msg:"Nmap Xmas Scan"; flags:FPU; flow:stateless;sid:10000013; rev:1;)
```

Attack Simulation:
An Xmas scan was executed from Kali Linux (attacker) against an Ubuntu system (victim) using Nmap.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sX 192.168.10.51
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-23 07:29 EST
Nmap scan report for 192.168.10.51
Host is up (0.0026s latency).
```

Detection Outcome:
Snort generated real-time alerts for TCP packets containing Xmas scan flag combinations, confirming successful detection of the attack.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/23-17:59:04.017856  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:995
11/23-17:59:04.017856  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:995
11/23-17:59:04.017856  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:113
11/23-17:59:04.017856  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:113
11/23-17:59:04.017856  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:1723
11/23-17:59:04.017856  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:1723
11/23-17:59:04.017856  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:8080
11/23-17:59:04.017856  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:8080
11/23-17:59:04.017857  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:993
11/23-17:59:04.017857  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:993
11/23-17:59:04.017857  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:445
11/23-17:59:04.017857  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:445
11/23-17:59:04.017857  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:587
11/23-17:59:04.017857  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:587
11/23-17:59:04.017857  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:256
11/23-17:59:04.017857  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:256
11/23-17:59:04.018600  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:199
11/23-17:59:04.018600  [**] [1:1228:7] SCAN nmap XMAS [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.43:57214 -> 192.168.10.51:199
11/23-17:59:04.018600  [**] [1:10000013:1] Nmap Xmas Scan [**] [Priority: 0] {TCP} 192.168.10.43:57214 -> 192.168.10.51:135
```

Rule No: 4 – ICMP Detection

Description:
This custom Snort rule detects ICMP echo request (ping) activity, which is commonly used for host discovery and network reconnaissance.

Rule Used:
An ICMP alert rule configured to trigger on ICMP type 8 (echo request) packets targeting the internal network.

```
# 4) ICMP ping scan
alert icmp any any -> $HOME_NET any (msg:"SCAN: ICMP echo request";itype:8;sid:4000005; rev:1;)
```

Attack Simulation:
ICMP ping requests were sent from Kali Linux (attacker) to an Ubuntu system (victim).

```
┌──(kali㉿kali)-[~]
└─$ ping 192.168.10.51
PING 192.168.10.51 (192.168.10.51) 56(84) bytes of data.
64 bytes from 192.168.10.51: icmp_seq=1 ttl=64 time=4.12 ms
64 bytes from 192.168.10.51: icmp_seq=2 ttl=64 time=4.55 ms
64 bytes from 192.168.10.51: icmp_seq=3 ttl=64 time=16.7 ms
64 bytes from 192.168.10.51: icmp_seq=4 ttl=64 time=1.25 ms
64 bytes from 192.168.10.51: icmp_seq=5 ttl=64 time=2.54 ms
64 bytes from 192.168.10.51: icmp_seq=6 ttl=64 time=1.41 ms
```

Detection Outcome:
Snort successfully captured and generated real-time alerts for each ICMP echo request, confirming effective detection of ICMP-based reconnaissance.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
11/20-16:15:13.678818  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:14.682365  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:15.680674  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:16.685535  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:17.687519  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:18.690219  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:19.691595  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:20.694789  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:21.697305  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
11/20-16:15:22.699130  [**] [1:4000005:1] SCAN: ICMP echo request [**] [Priority: 0] {ICMP} 192.168.10.43 -> 192.168.10.51
```

Rule No: 5 – SSH Login Attempt Detection

Description:
This custom Snort rule detects SSH login attempts to identify unauthorized or suspicious access attempts against systems running SSH services.

Rule Used:
A TCP alert rule configured to monitor flag: S packets targeting port 22, indicating an SSH connection attempt.

```
# 5) SSH Login Attempt Detected
alert tcp any any -> $HOME_NET 22 (msg:"SSH Login Attempt"; flags:S; sid:1000001; rev:1;)
```

Attack Simulation:
An SSH login attempt was initiated from Kali Linux (attacker) to an Ubuntu system (victim) using invalid credentials.

```
┌──(kali㉿kali)-[~]
└─$ ssh user@192.168.10.51
user@192.168.10.51's password:
Permission denied, please try again.
user@192.168.10.51's password:
Permission denied, please try again.
user@192.168.10.51's password:
user@192.168.10.51: Permission denied (publickey,password).
```

Detection Outcome:
Snort successfully generated real-time alerts for each SSH connection attempt, confirming effective detection of SSH access attempts.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/23-18:01:25.814836  [**] [1:1000001:1] SSH Login Attempt [**] [Priority: 0] {TCP} 192.168.10.43:42362 -> 192.168.10.51:22
11/23-18:01:39.195980  [**] [1:1000001:1] SSH Login Attempt [**] [Priority: 0] {TCP} 192.168.10.43:48086 -> 192.168.10.51:22
11/23-18:01:46.660707  [**] [1:1000001:1] SSH Login Attempt [**] [Priority: 0] {TCP} 192.168.10.43:57886 -> 192.168.10.51:22
11/23-18:01:50.058717  [**] [1:1000001:1] SSH Login Attempt [**] [Priority: 0] {TCP} 192.168.10.43:58286 -> 192.168.10.51:22
11/23-18:02:04.883585  [**] [1:1000001:1] SSH Login Attempt [**] [Priority: 0] {TCP} 192.168.10.43:41966 -> 192.168.10.51:22
```

Rule No: 6 – SSH Brute Force Attempt Detection

Description:
This custom Snort rule is designed to detect SSH brute force attacks by identifying multiple SSH connection attempts originating from the same source within a limited time period.

Rule Explanation:
The rule monitors TCP traffic directed to the SSH service on port 22 and focuses on established connections to the server. By applying a detection filter, it tracks repeated connection attempts from a single source IP and triggers an alert when the number of attempts exceeds the defined threshold within 60 seconds. This behavior is indicative of automated brute force login attempts.

```
# 6) SSH Bruteforce Attempt
alert tcp any any -> $HOME_NET 22 (msg:"SSH Brute Force Attempt"; flow:to_server,established; detection_filter:track by_src, count 5, seconds 60; sid:1000002;
```

Attack Simulation:
An SSH brute force attack was launched from Kali Linux (attacker) against an Ubuntu system (victim) using the Hydra tool.

```
┌──(kali㉿kali)-[~]
└─$ hydra -l user -p pass.txt ssh://192.168.10.51
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-
binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-11-20 02:46:55
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 1 task per 1 server, overall 1 task, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking ssh://192.168.10.51:22/
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-11-20 02:46:58
```

Detection Outcome:
Snort successfully generated real-time alerts after detecting multiple SSH login attempts from the same source, confirming effective brute force attack detection.

```
11/25-17:46:36.349090  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56414 -> 192.168.10.51:22
11/25-17:46:33.346296  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:42612 -> 192.168.10.51:22
11/25-17:46:37.719346  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56372 -> 192.168.10.51:22
11/25-17:46:34.356309  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:42644 -> 192.168.10.51:22
11/25-17:46:36.301207  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56404 -> 192.168.10.51:22
11/25-17:46:36.914153  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56344 -> 192.168.10.51:22
11/25-17:46:45.782449  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56372 -> 192.168.10.51:22
11/25-17:46:40.454226  [**] [1:1000002:1] SSH Brute Force Attempt [**] [Priority: 0] {TCP} 192.168.10.43:56414 -> 192.168.10.51:22
```

Rule No: 7 – Telnet Connection Attempt Detection

Description:
This custom Snort rule is designed to detect Telnet connection attempts, which are considered insecure due to the lack of encryption and are commonly targeted by attackers for unauthorized access.

Rule Explanation:
The rule monitors TCP traffic directed to port 23, which is the default port used by the Telnet service. Any connection attempt targeting this port triggers an alert, allowing the detection of insecure or unauthorized Telnet access attempts to systems within the internal network.

```
# 7) Telnet login attempt to  port (ONLY port 23)
alert tcp any any -> $HOME_NET 23 (msg:"TELNET login attempt"; sid:1000003; rev:1;)
```

Attack Simulation:
A Telnet connection attempt was initiated from **Kali Linux (attacker)** to an **Ubuntu system (victim)** using the Telnet client.

```
┌──(kali㉿kali)-[~]
└─$ telnet 192.168.10.51 23
Trying 192.168.10.51 ...
Connected to 192.168.10.51.
Escape character is '^]'.
Ubuntu 22.04.5 LTS
user-VirtualBox login:
```

Detection Outcome:
Snort successfully generated real-time alerts upon detecting Telnet connection attempts, confirming effective monitoring and detection of insecure Telnet access.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/25-17:54:00.664918  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
11/25-17:54:00.668282  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
11/25-17:54:00.752239  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
11/25-17:54:00.753657  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
11/25-17:54:00.756201  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
11/25-17:54:00.759318  [**] [1:1000003:1] TELNET login attempt [**] [Priority: 0] {TCP} 192.168.10.43:53976 -> 192.168.10.51:23
```

Rule No: 8 – FTP Connection Attempt Detection

Description:
This custom Snort rule detects FTP connection and login attempts, as FTP transmits credentials in plain text and is commonly targeted for unauthorized access.

Rule Explanation:
The rule monitors TCP traffic directed to port 21, the default port used by the FTP service. Any connection or login attempt to this port triggers an alert, allowing the detection of potentially insecure or unauthorized FTP access attempts within the internal network.

```
# 8) FTP login attempt to port (ONLY port 21)
alert tcp any any -> $HOME_NET 21 (msg:"FTP login attempt"; sid:100005; rev:1;)
```

Attack Simulation:
An FTP connection attempt was initiated from Kali Linux (attacker) to an Ubuntu system (victim) using the FTP client.

```
┌──(kali㉿kali)-[~]
└─$ ftp 192.168.10.51
Connected to 192.168.10.51.
220 (vsFTPd 3.0.5)
Name (192.168.10.51:kali): ^C
```

Detection Outcome:
Snort successfully generated real-time alerts upon detecting FTP connection attempts, confirming effective monitoring of FTP-based access.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/25-17:44:47.183127  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:43928 -> 192.168.10.51:21
11/25-17:44:47.187532  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:43928 -> 192.168.10.51:21
11/25-17:44:47.244095  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:43928 -> 192.168.10.51:21
11/25-17:45:05.600167  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:57748 -> 192.168.10.51:21
11/25-17:45:05.601323  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:57748 -> 192.168.10.51:21
11/25-17:45:05.635188  [**] [1:100005:1] FTP login attempt [**] [Priority: 0] {TCP} 192.168.10.43:57748 -> 192.168.10.51:21
```

Rule No: 9 – SQL Injection Detection

Description:

This custom Snort rule is designed to detect SQL injection attempts targeting web applications by identifying malicious input patterns commonly used to manipulate backend database queries**.**

Rule Explanation:

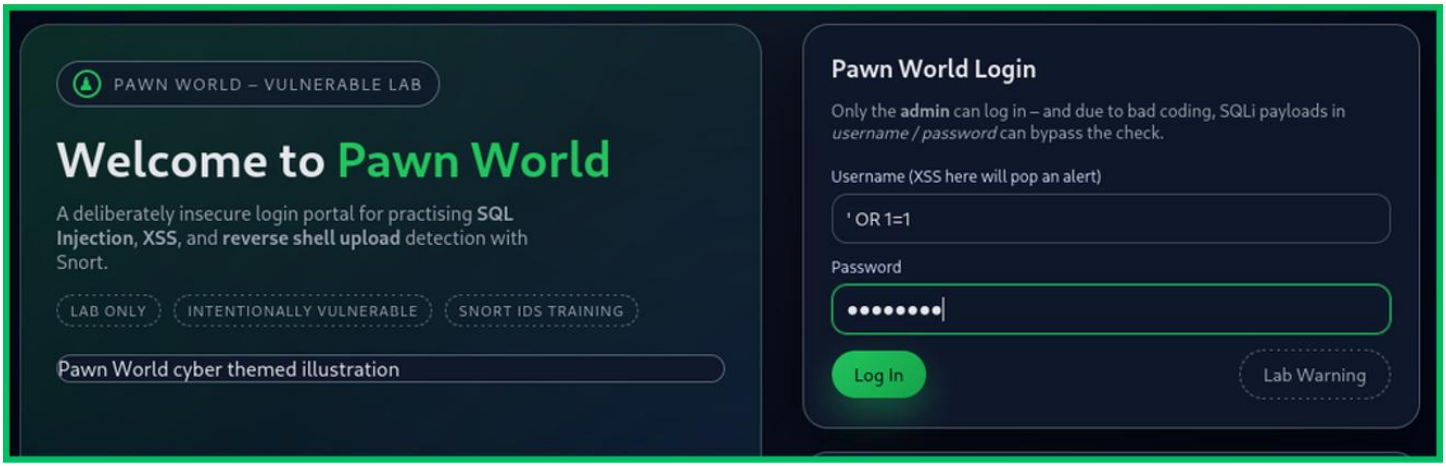
The rule monitors HTTP traffic directed to port 80 and inspects request content for SQL injection payloads. In this implementation, a URL-encoded SQL injection string is used instead of a plain OR 1=1 payload, as direct SQL keywords are easily flagged by basic filters. Encoding the payload allows the attack to resemble real-world techniques used to bypass simple input validation while still being detected by Snort through content inspection.

```
# 9) SQL Injection Attempt
alert tcp any any -> $HOME_NET 80 (msg:"SQL Injection Attempt Detected";content:"username=%27+OR+1%3D1";sid:100008902; rev:1;)
```

Threat Simulation:

The SQL injection attack was simulated from Kali Linux (attacker) using the curl command with a URL-encoded payload. To demonstrate a real-life scenario, a deliberately vulnerable web application named Pawn World was developed and hosted on the victim system. The injection was performed against the login functionality of this application to emulate practical web-based SQL injection behaviour rather than a theoretical test case.

```
┌──(kali㉿kali)-[~]
└─$ curl -X POST "http://192.168.10.51/index.php" \
     -d "username=%27+OR+1%3D1&password=%27+OR+1%3D1"
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Pawn World - Vulnerable Login</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        * { box-sizing: border-box; margin: 0; padding: 0; }
        body {
            font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
            background: radial-gradient(circle at top, #101827, #020617 60%);
            color: #e5e7eb;
            min-height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            padding: 20px;
        }
```



Detection Outcome:

Snort successfully generated real-time alerts for the encoded SQL injection attempts originating from the attacker system, confirming effective detection of web-based SQL injection activity against the vulnerable application.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
11/25-10:24:23.280014  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:37584 -> 192.168.10.51:80
11/25-10:24:53.791534  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:53334 -> 192.168.10.51:80
11/25-10:25:10.152225  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:52316 -> 192.168.10.51:80
11/25-10:25:34.819758  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:52378 -> 192.168.10.51:80
11/25-10:25:35.499892  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:52394 -> 192.168.10.51:80
11/25-10:25:36.849155  [**] [1:100008902:1] SQL Injection Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:52396 -> 192.168.10.51:80
```

Rule No: 10 – Directory Traversal Detection

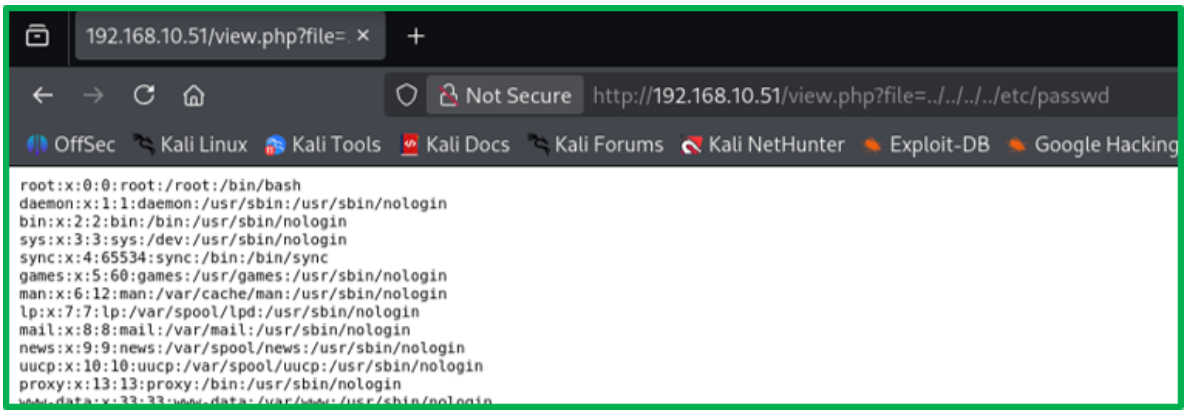
Description:
This custom Snort rule is designed to detect directory traversal attacks, which attempt to access sensitive system files by manipulating file path parameters in web applications.

Rule Explanation:
The rule monitors HTTP traffic directed to port 80 and inspects the requested URI for directory traversal patterns such as ../. These patterns indicate an attempt to escape the intended web directory structure and access restricted files on the server. Case-insensitive URI inspection ensures detection even when attackers try to obfuscate the payload.

```
# 10) Directory Traversal Attack  Detection
alert tcp any any -> $HOME_NET 80 (msg:"Directory Traversal Attempt Detected";content:"../"; http_uri;nocase;sid:10000301; rev:1;)
```

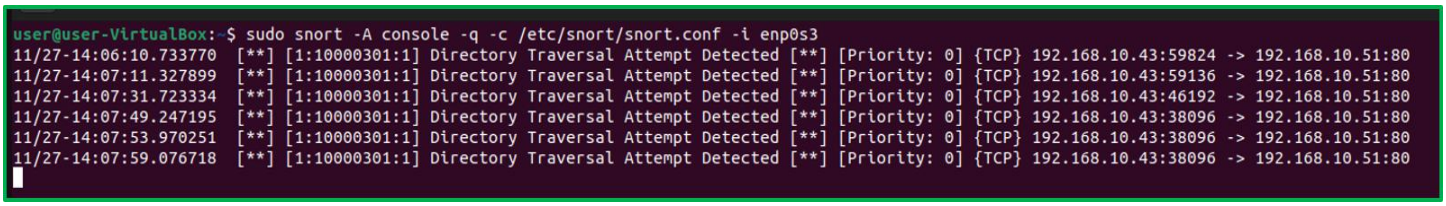
Threat Simulation:
The directory traversal attack was simulated from Kali Linux (attacker) using the curl command with a crafted URL containing traversal sequences to access the /etc/passwd file. In addition to the command-line test, the same attack was demonstrated through a web browser against a deliberately vulnerable web application hosted on the victim system, providing a real-world scenario of how directory traversal vulnerabilities are exploited in web applications.

```
┌──(kali㉿kali)-[~]
└─$ curl "http://192.168.10.51/view.php?file=../../../../etc/passwd"
<pre>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```



Detection Outcome:

Snort successfully generated real-time alerts for the directory traversal attempts, confirming effective detection of unauthorized file access attempts against the vulnerable web service.



Rule No: 11 – Sensitive File Access Detection

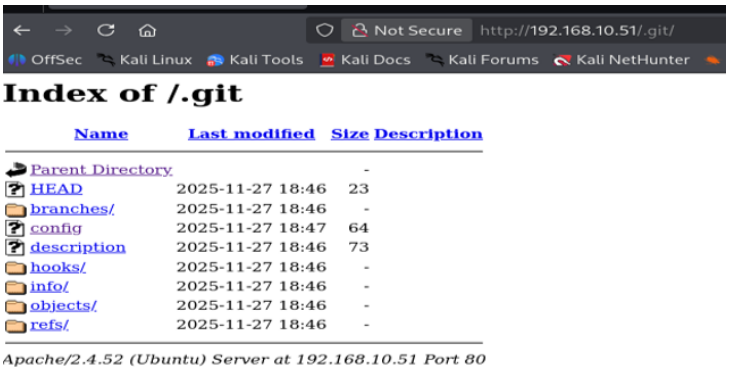
Description:
This custom Snort rule is designed to detect unauthorized access attempts to sensitive files and directories exposed through web applications, specifically version control repositories such as the .git directory.

Rule Explanation:
The rule monitors HTTP traffic directed to port 80 and inspects requested URIs for access to sensitive paths like .git. Such directories often contain repository metadata, configuration files, and commit history, which can lead to serious information disclosure if exposed publicly. By detecting requests targeting .git, the rule helps identify insecure deployments and potential data leakage attempts.

```
# 11) Sensitive File Access Detection
alert tcp any any -> $HOME_NET 80 (msg:"Sensitive Repository Access - .git Directory";content:"/.git"; http_uri;sid:1000300; rev:1;)
```

Threat Simulation:
The attack was simulated from Kali Linux (attacker) using the curl command to directly request the .git directory from the web server. The same exposure was also demonstrated through a web browser, where directory listing of the .git repository was accessible, representing a realistic misconfiguration scenario in web applications.

```
┌──(kali㉿kali)-[~]
└─$ curl -v http://192.168.10.51/.git/
*   Trying 192.168.10.51:80...
* Connected to 192.168.10.51 (192.168.10.51) port 80
* using HTTP/1.x
> GET /.git/ HTTP/1.1
> Host: 192.168.10.51
> User-Agent: curl/8.15.0
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 04 Dec 2025 06:43:09 GMT
< Server: Apache/2.4.52 (Ubuntu)
< Vary: Accept-Encoding
< Content-Length: 2289
< Content-Type: text/html;charset=UTF-8
```



Detection Outcome:

Snort successfully generated real-time alerts for each attempt to access the .git directory, confirming effective detection of sensitive file and repository exposure.

```
user@user-VirtualBox:~$ sudo nano /etc/snort/rules/local.rules
[sudo] password for user:
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
11/27-18:53:51.835424  [**] [1:1000300:1] Sensitive Repository Access - .git Directory [**] [Priority: 0] {TCP} 192.168.10.43:58678 -> 192.168.10.51:80
11/27-18:54:14.137697  [**] [1:1000300:1] Sensitive Repository Access - .git Directory [**] [Priority: 0] {TCP} 192.168.10.43:39582 -> 192.168.10.51:80
11/27-18:54:56.401548  [**] [1:1000300:1] Sensitive Repository Access - .git Directory [**] [Priority: 0] {TCP} 192.168.10.43:51210 -> 192.168.10.51:80
11/27-18:55:01.820866  [**] [1:1000300:1] Sensitive Repository Access - .git Directory [**] [Priority: 0] {TCP} 192.168.10.43:51214 -> 192.168.10.51:80
11/27-18:55:19.046313  [**] [1:1000300:1] Sensitive Repository Access - .git Directory [**] [Priority: 0] {TCP} 192.168.10.43:59292 -> 192.168.10.51:80
```

Rule No: 12 – SQL Scanning Detection

Description:
This custom Snort rule is designed to detect automated SQL scanning activity commonly performed using tools such as SQLMap, which are used to identify and exploit SQL injection vulnerabilities in web applications.

Rule Explanation:
The rule monitors HTTP traffic directed to port 80 and inspects HTTP headers for signatures associated with SQLMap, such as the presence of the sqlmap identifier. Automated scanners often leave recognizable fingerprints in request headers, and detecting these patterns allows early identification of reconnaissance and vulnerability scanning activity before exploitation occurs.

```
# 12) SQL Scan Detection
alert tcp any any -> $HOME_NET 80 (msg:"SQLMap Scan Detected";content:"sqlmap"; http_header; nocase;sid:900500; rev:1;)
```

Threat Simulation:
The SQL scanning activity was simulated from Kali Linux (attacker) using the SQLMap tool in batch mode against a web application hosted on the victim system. This represents a real-world scenario where attackers use automated tools to probe applications for SQL injection vulnerabilities.

Detection Outcome:

Snort successfully generated real-time alerts for SQLMap scan attempts detected in HTTP traffic, confirming effective identification of automated SQL scanning behavior.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for user:
11/27-19:21:54.892871  [**] [1:900500:1] SQLMap Scan Detected [**] [Priority: 0] {TCP} 192.168.10.43:34924 -> 192.168.10.51:80
11/27-19:21:55.388680  [**] [1:900500:1] SQLMap Scan Detected [**] [Priority: 0] {TCP} 192.168.10.43:34928 -> 192.168.10.51:80
11/27-19:22:04.879050  [**] [1:900500:1] SQLMap Scan Detected [**] [Priority: 0] {TCP} 192.168.10.43:35758 -> 192.168.10.51:80
11/27-19:22:05.377895  [**] [1:900500:1] SQLMap Scan Detected [**] [Priority: 0] {TCP} 192.168.10.43:35768 -> 192.168.10.51:80
```

Rule No: 13 – Command Injection Attempt Detection

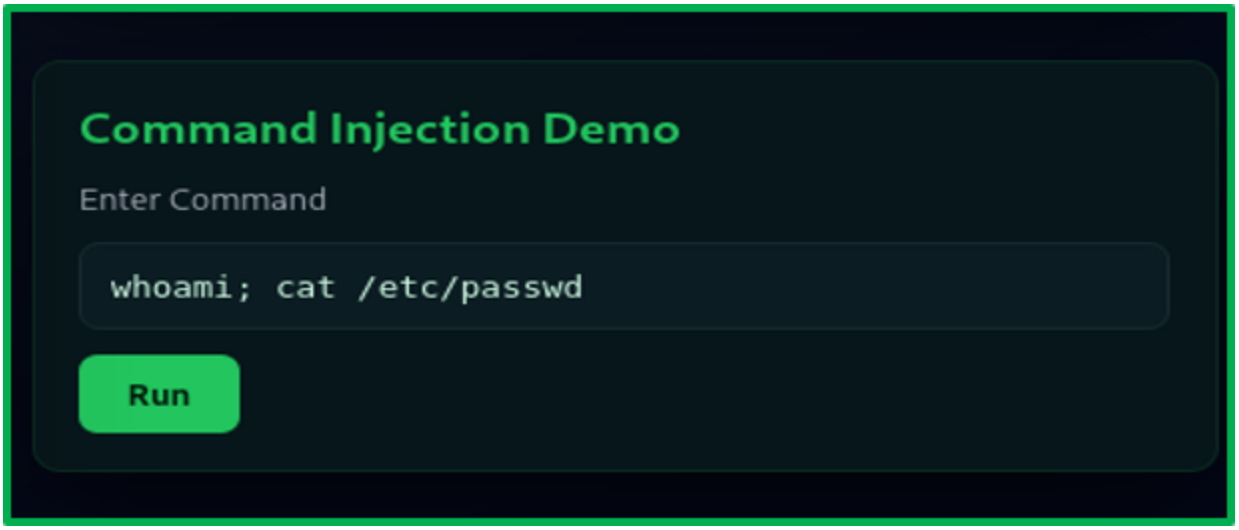
Description:
This custom Snort rule is designed to detect command injection attempts against web applications, where attackers try to execute operating system commands through vulnerable input fields.

Rule Explanation:
The rule monitors HTTP traffic directed to port 80 and inspects request content for a predefined command injection indicator string. This approach is used to identify command execution attempts submitted through web requests, simulating how attackers exploit insecure server-side command handling in real-world applications.

```
# 13) Command Injection Attempt
alert tcp any any -> $HOME_NET 80 (msg:"Command Injection Attempt";content:"INJECT_DEMO";sid:1001300; rev:1;)
```

Threat Simulation:
The command injection attack was simulated from Kali Linux (attacker) using the curl command to send a crafted POST request containing command input. In addition, a deliberately vulnerable web application with a command execution interface was used to demonstrate how attackers can execute system-level commands such as whoami and file access commands through a web interface, reflecting a realistic exploitation scenario.

```
┌──(kali㉿kali)-[~]
└─$ curl -X POST -d "cmd=whoami;INJECT_DEMO" http://192.168.10.51/
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Pawn World - Vulnerable Login</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        * { box-sizing: border-box; margin: 0; padding: 0; }
        body {
            font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI"
            background: radial-gradient(circle at top, #101827, #020617 60%);
            color: #e5e7eb;
```



Detection Outcome:

Snort successfully generated real-time alerts for each command injection attempt detected in the HTTP traffic, confirming effective detection of command execution attempts against the vulnerable web application.

```
user@user-VirtualBox:/etc/snort$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
12/04-10:32:35.071111  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:38288 -> 192.168.10.51:80
12/04-10:32:40.343567  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:48424 -> 192.168.10.51:80
12/04-10:32:44.863161  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:48424 -> 192.168.10.51:80
12/04-10:33:51.914987  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:37880 -> 192.168.10.51:80
12/04-10:33:54.862356  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:37894 -> 192.168.10.51:80
12/04-10:33:55.539080  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:37904 -> 192.168.10.51:80
12/04-10:33:56.102416  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:53348 -> 192.168.10.51:80
12/04-10:33:56.596301  [**] [1:1001300:1] Command Injection Attempt [**] [Priority: 0] {TCP} 192.168.10.43:53354 -> 192.168.10.51:80
```

Rule No: 14 – XSS Detection

Description:

This custom Snort rule is designed to detect Cross-Site Scripting (XSS) attempts, where attackers inject malicious client-side scripts into web application inputs to execute in a victim's browser.

Rule Explanation:

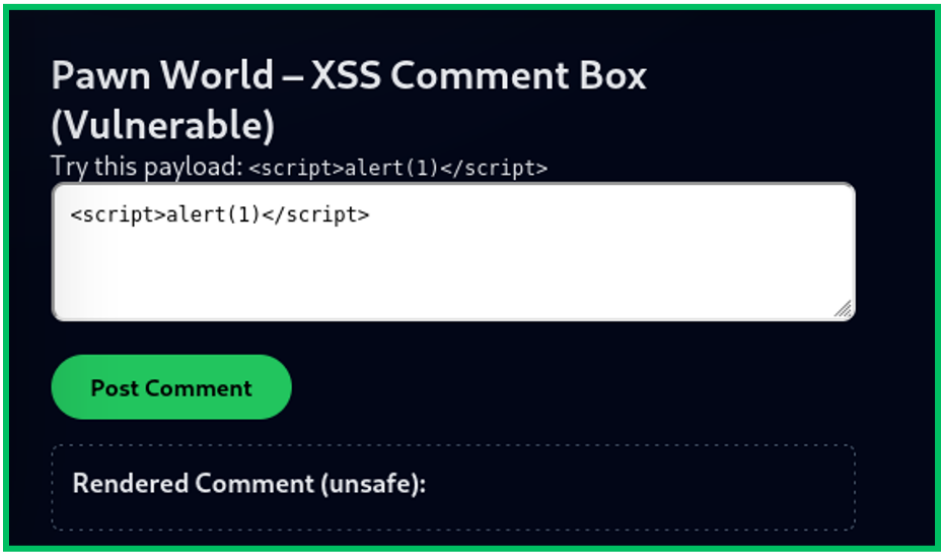
The rule monitors HTTP traffic directed to port 80 and inspects request content for encoded XSS payload patterns, specifically script injection attempts. Encoding is commonly used by attackers to bypass basic input validation, and detecting these patterns helps identify malicious script injection before it can impact end users.

```
# 14) XSS Attempt
alert tcp any any -> $HOME_NET 80 (msg:"XSS Attempt Detected";content:"xss_comment=%3Cscript%3Ealert%281%29%3C%2Fscript%3E";sid:100008920; rev:2;)
```

Threat Simulation:

The XSS attack was simulated from Kali Linux (attacker) using the curl command with an encoded script payload. In addition, a deliberately vulnerable web application named Pawn World was used, where the payload was injected through a comment input field to demonstrate a realistic stored/reflected XSS scenario in a web interface.

```
┌──(kali㉿kali)-[~]
└─$ curl -X POST http://192.168.10.51/index.php \
  -d "xss_comment=%3Cscript%3Ealert%281%29%3C%2Fscript%3E"
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Pawn World - Vulnerable Login</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        * { box-sizing: border-box; margin: 0; padding: 0; }
        body {
            font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans
            background: radial-gradient(circle at top, #101827, #020617 60%);
            color: #e5e7eb;
            min-height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            padding: 20px;
        }
        .wrapper {
            max-width: 1100px;
            width: 100%;
            display: grid;
            grid-template-columns: minmax(0, 1.2fr) minmax(0, 1fr);
```



Detection Outcome:

Snort successfully generated real-time alerts for the XSS injection attempts detected in HTTP traffic, confirming effective detection of cross-site scripting activity against the vulnerable application.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
11/25-15:19:53.849669  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:49126 -> 192.168.10.51:80
11/25-15:20:12.600421  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:42978 -> 192.168.10.51:80
11/25-15:20:13.290431  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:42984 -> 192.168.10.51:80
11/25-15:20:13.946519  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:42994 -> 192.168.10.51:80
11/25-15:20:14.593976  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:43000 -> 192.168.10.51:80
11/25-15:21:04.017647  [**] [1:100008920:2] XSS Attempt Detected [**] [Priority: 0] {TCP} 192.168.10.43:38234 -> 192.168.10.51:80
```

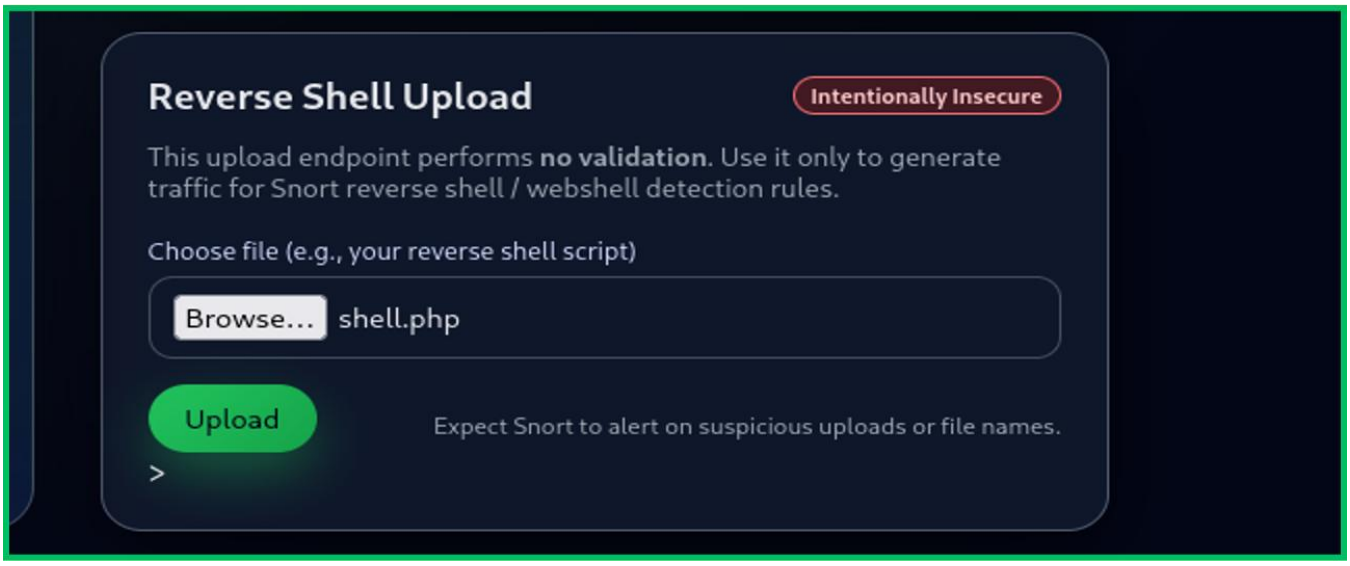
Rule No: 15 – File Upload Detection

Description:
This custom Snort rule is designed to detect suspicious file upload attempts, particularly those involving server-side script files that may be used to gain unauthorized access or establish reverse shells.

Rule Explanation:
The rule monitors HTTP traffic directed to port 80 and inspects request content for file upload indicators such as the filename parameter combined with script file extensions like .php. This pattern is commonly associated with malicious file uploads, where attackers attempt to upload web shells or reverse shell scripts to the server. Detecting these characteristics helps identify potentially dangerous uploads at an early stage.

```
# 15) Reverse Shell Attempt
alert tcp any any -> $HOME_NET 80 (msg:"PHP File Upload Detected (possible reverse shell)";content:"filename=\"";content:".php"; distance:0; within:15;sid:100000910; rev:2;)
```

Threat Simulation:
The attack was simulated using a deliberately vulnerable file upload functionality in the web application, where a PHP-based reverse shell file was uploaded without validation. This setup represents a real-world scenario in which insecure file upload mechanisms can be abused to gain remote access to the server.



Detection Outcome:

Snort successfully generated real-time alerts for the suspicious PHP file upload attempts, confirming effective detection of potentially malicious file upload activity.

```
user@user-VirtualBox:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
11/25-10:20:08.189818  [**] [1:100008910:1] PHP File Upload Detected (possible reverse shell) [**] [Priority: 0] {TCP} 192.168.10.43:42778 -> 192.168.10.51:80
11/25-10:20:58.175271  [**] [1:100008910:1] PHP File Upload Detected (possible reverse shell) [**] [Priority: 0] {TCP} 192.168.10.43:49202 -> 192.168.10.51:80
11/25-10:21:11.740090  [**] [1:100008910:1] PHP File Upload Detected (possible reverse shell) [**] [Priority: 0] {TCP} 192.168.10.43:34364 -> 192.168.10.51:80
11/25-10:22:14.120299  [**] [1:100008910:1] PHP File Upload Detected (possible reverse shell) [**] [Priority: 0] {TCP} 192.168.10.43:52252 -> 192.168.10.51:80
11/25-10:22:14.765203  [**] [1:100008910:1] PHP File Upload Detected (possible reverse shell) [**] [Priority: 0] {TCP} 192.168.10.43:52258 -> 192.168.10.51:80
```

11. Final Observation and Result

Final Observation

The Snort Intrusion Detection System (IDS) was successfully deployed and configured in a controlled lab environment, including proper network interface selection, monitored IP range configuration, and rule integration. Initial verification confirmed that Snort was actively monitoring network traffic and generating alerts as expected.

Through a series of simulated attack scenarios, Snort effectively detected multiple categories of malicious activities, including network reconnaissance, authentication attacks, insecure service access, and web application–based attacks. The implementation of custom Snort rules enhanced detection capability beyond default signatures, allowing the system to identify real-world attack techniques such as port scanning, brute force attempts, SQL injection, command injection, cross-site scripting, directory traversal, sensitive file exposure, automated scanning, and malicious file uploads.

Real-time alert generation was consistently observed during all attack simulations. Alerts contained accurate details such as source IP, destination IP, protocol, port, timestamp, and alert message, demonstrating correct rule triggering and log generation. The Snort log structure and console output provided clear visibility into detected events, enabling effective analysis and correlation of attack behaviour.

Additionally, the use of deliberately vulnerable web applications and encoded payloads closely reflected real-life attack scenarios, validating Snort's effectiveness in detecting both manual and automated attack techniques commonly used by attackers.

Result

The results of this implementation confirm that Snort, when properly configured and supplemented with custom rules, functions as an effective Intrusion Detection System for monitoring network and application-level threats. All simulated attacks were successfully detected, and corresponding alerts were generated in real time without false negatives.

This project demonstrates a comprehensive understanding of Snort architecture, rule creation, traffic inspection, alert analysis, and attack detection methodologies. The successful detection of diverse attack vectors highlights the importance of custom rule development in strengthening network security monitoring and improving threat visibility in real-world environments

12. Conclusion

This project successfully demonstrated the deployment, configuration, and practical implementation of the Snort Intrusion Detection System in a controlled laboratory environment. Starting from environment setup and Snort installation to rule configuration and alert analysis, each phase was executed systematically to understand the core functionality of Snort as a network security monitoring tool.

Through extensive attack simulations, including network reconnaissance, authentication attacks, insecure service access, and web application–based vulnerabilities, Snort proved to be effective in detecting malicious activities in real time. The development and implementation of custom Snort rules significantly enhanced detection accuracy and allowed the identification of realistic attack patterns that are commonly observed in real-world environments.

The project also emphasized the importance of proper rule creation, traffic inspection, and log analysis in identifying threats at both the network and application layers. By using encoded payloads, automated scanning tools, and deliberately vulnerable web applications, the experiments closely reflected practical attack scenarios, strengthening the reliability and relevance of the results.

Overall, this implementation confirms that Snort is a powerful and flexible IDS when properly configured and supported with custom detection rules. The knowledge gained through this project provides a strong foundation in intrusion detection, security monitoring, and incident analysis, which are essential skills for cybersecurity and SOC operations.

13. References

Snort Official Documentation, *Snort.org*, https://www.snort.org/documents

Cisco Talos Intelligence Group, *Snort Rules and Updates*, https://www.talosintelligence.com/

Ubuntu Documentation, *Ubuntu Official Help Portal*, https://help.ubuntu.com/