**PROJECT REPORT**

**“NIDS Rule Creation and Testing Lab”**

**(Project – 1)**



**Submitted By,**

Dazy

Harsh Singh Rao

Dewoo P Kumar

Willie Peace Eseme

**Trainer:** Vasudev Jha

**Date:** 05/10/2025

**Duration:** 4 Weeks

**Summary**

The purpose of this project was to develop and test a robust set of custom rules for a Network Intrusion Detection System (NIDS) to identify and flag common cyber-attacks in real-time, reducing the mean time to detect threats within a network.

Here we used Snort, a powerful open-source NIDS that can monitor network traffic in real-time and trigger alerts when suspicious activity is detected.

This project demonstrates the detection of various attack types such as brute-force attempts, SQL injection, port scans, buffer overflows, and denial-of-service attacks using custom rules and an automated test script.

**Tools & Technologies Used**

* **Operating System:** Kali Linux
* **Tool:** Snort 3.1.82.0
* **Dependencies Installed:** build-essential, libpcap-dev, libpcre3-dev, libdumbnet-dev, bison, flex, zlib1g-dev
* **Network Interface Used:** eth0

**1.SETUP AND INSTALLATION**

**Kali Linux**

The project was carried out on Kali Linux, a Debian-based penetration testing distribution widely used in cybersecurity. Kali provides an extensive library of preinstalled tools, supports advanced networking features, and integrates easily with open-source security software such as Snort.

**Snort**

Snort is an open-source Network Intrusion Detection System. It can analyze network traffic in real time and generate alerts when suspicious patterns are detected. Snort supports multiple rule types, making it suitable for detecting brute-force attempts, SQL injection, port scans, DoS, and other common network attacks.

**Dependencies**

Before installing Snort, the required libraries and build tools must be present in the system. These dependencies allow Snort to compile, run efficiently, and handle different protocols:

* build-essential → Provides gcc, make, and other compiler tools.
* libpcap-dev → Enables packet capture from live network interfaces.
* libpcre3-dev → Adds support for regular expression pattern matching in rules.
* libdumbnet-dev → Lightweight networking library for low-level operations.
* bison and flex → Used by Snort to parse and tokenize protocol headers.
* zlib1g-dev and liblzma-dev → Support compressed data streams.
* libssl-dev → Provides SSL/TLS libraries for encrypted traffic handling.
* autoconf, automake, libtool, pkg-config, ethtool → Build helpers and network configuration tools (needed for compiling from source).
* **Installation Steps**
  1. **Update System Packages**

Update the package list before trying to installing the snort.

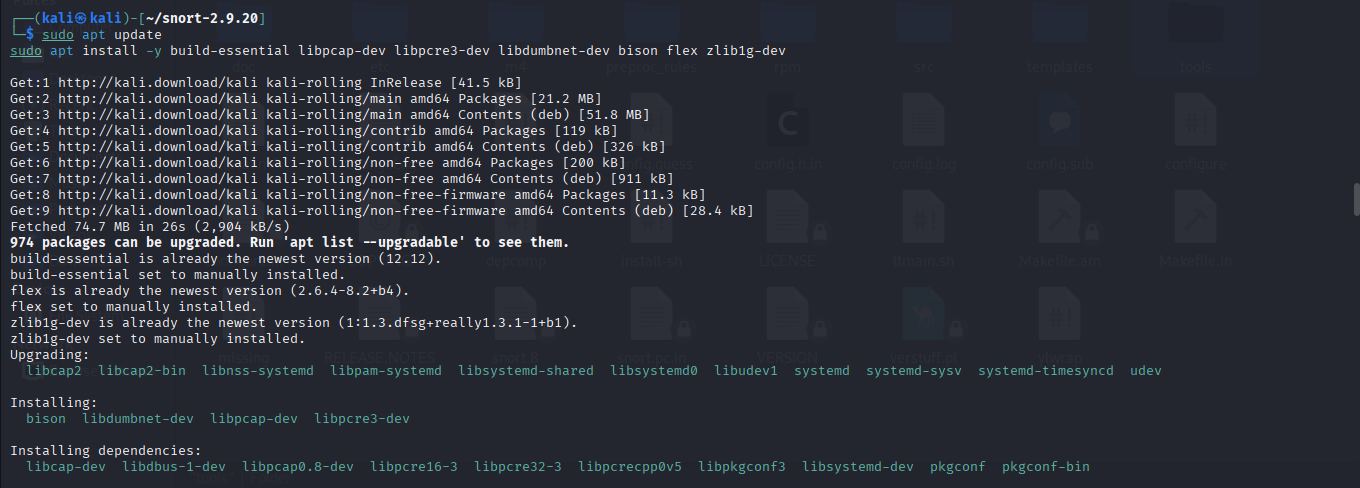
sudo apt update

* 1. **Install Dependencies and Snort**

Once the system packages are updated, we can install dependencies and snort.

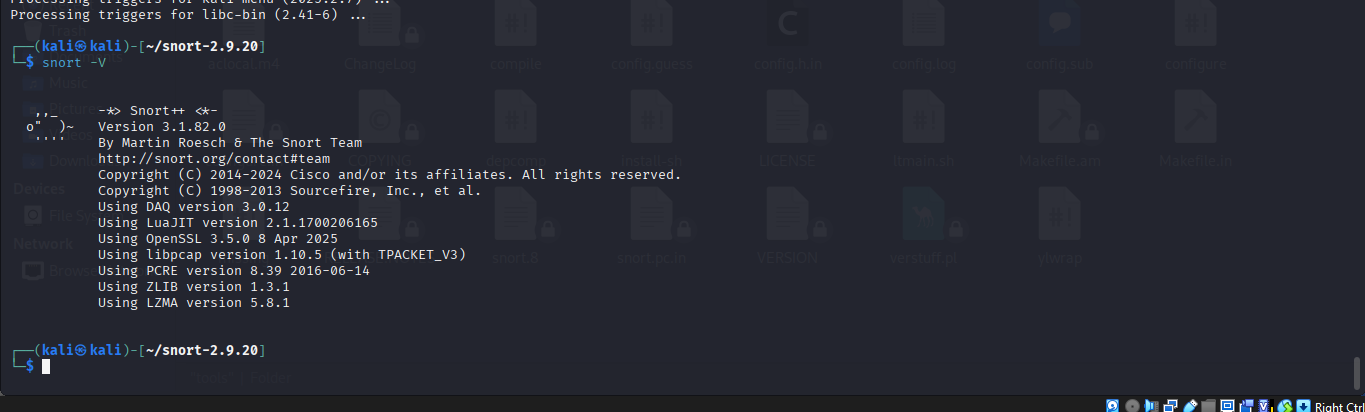
We can Install Snort by:

Sudo apt install -y snort



* 1. **Verify Installation**

Verify snort is installed and check its version .



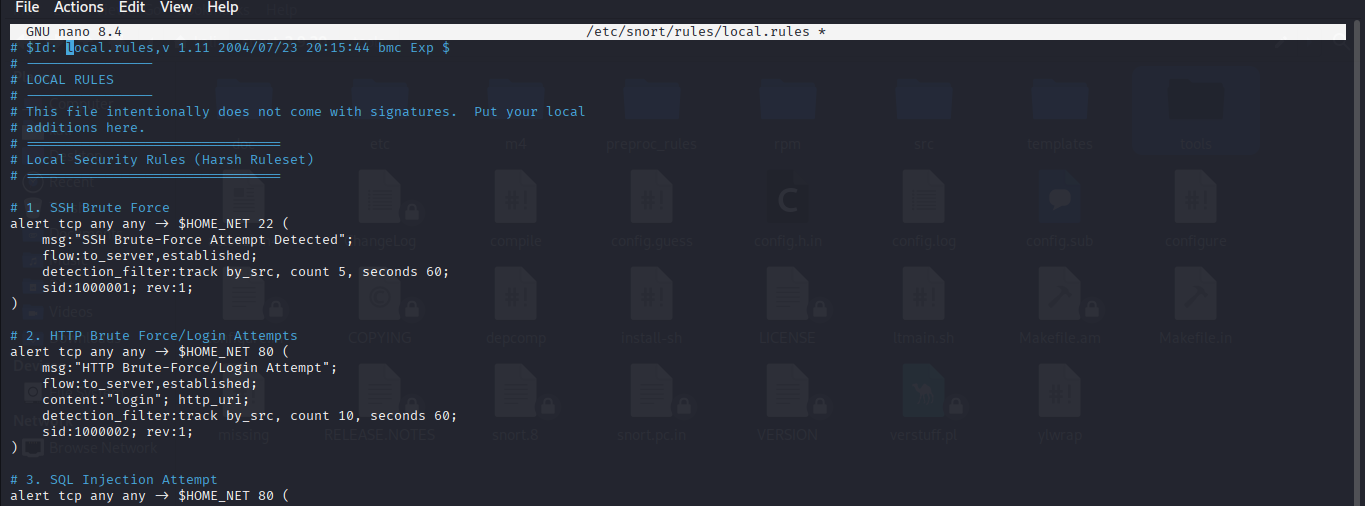
**2. WRITING CUSTOM RULES**

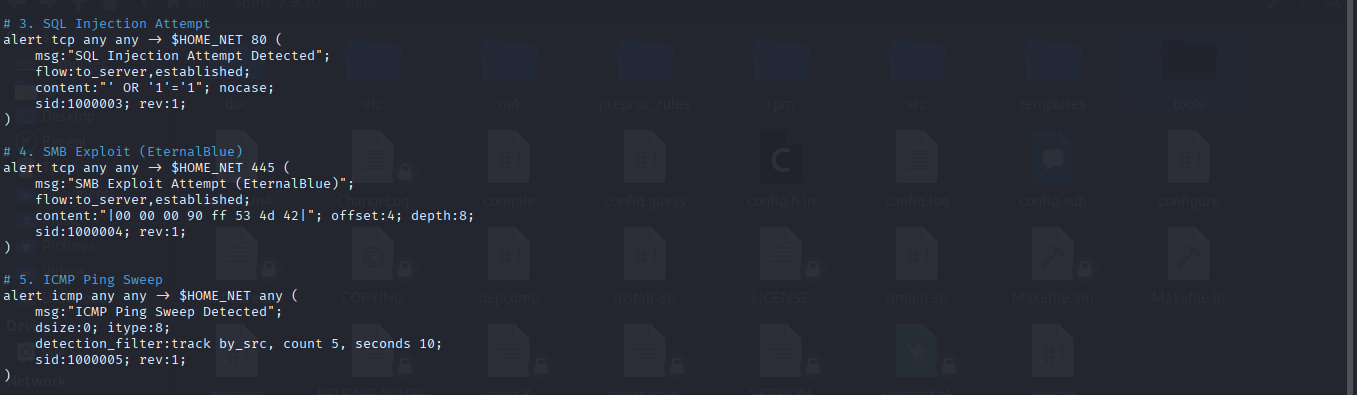
Here we write a collection of snort custom rules to identify particular malicious activities.

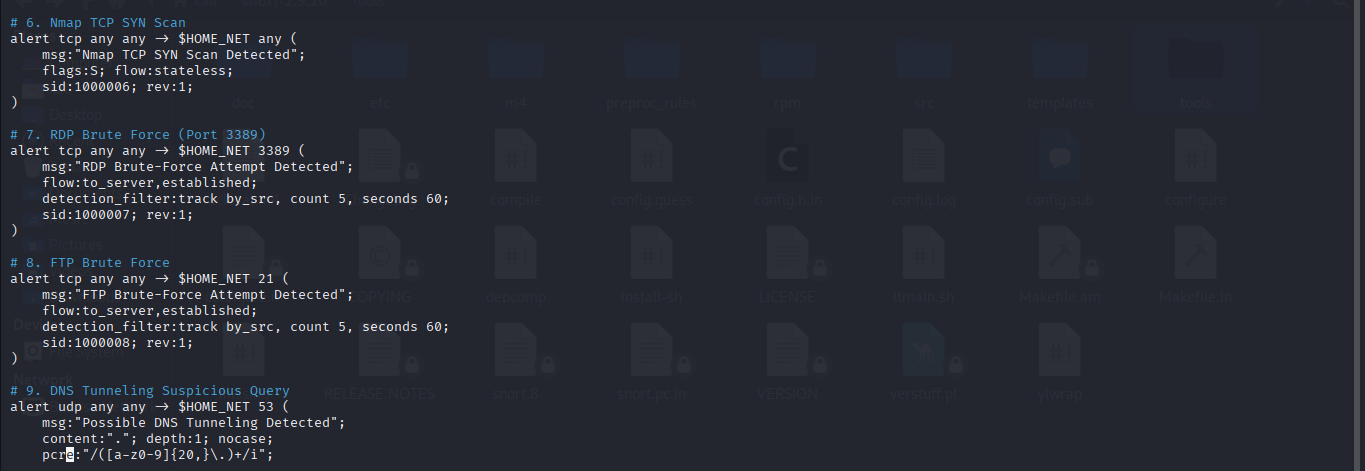
Custom rules allow Snort to detect environment-specific patterns that are not covered by the default rule set.

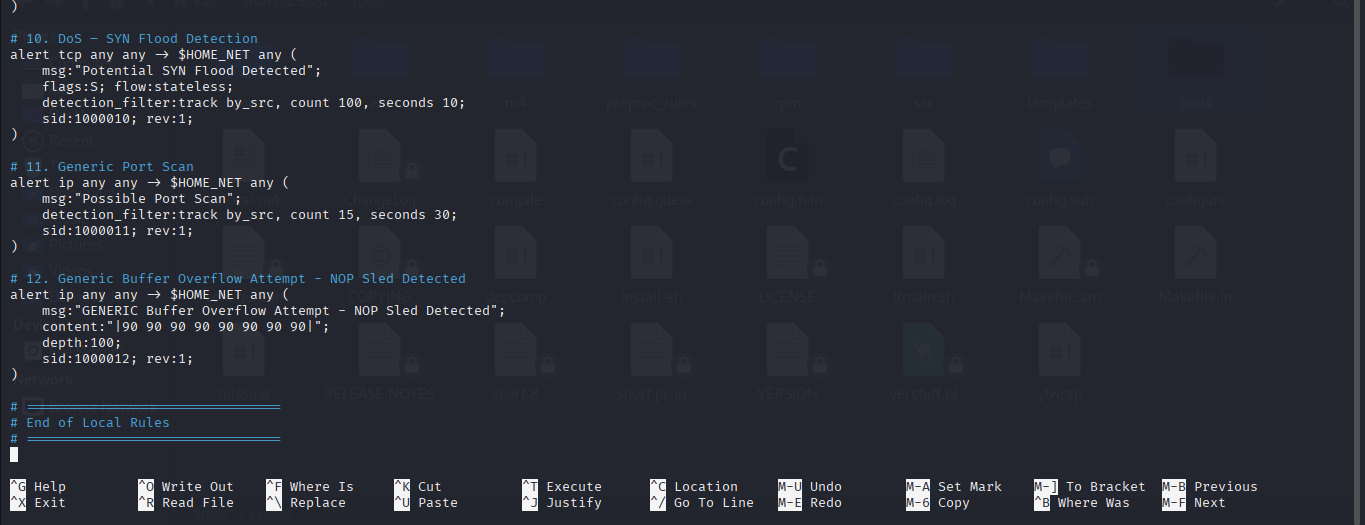
The snort custom rules is placed inside local.rules.

Sudo nano /etc/snort/rules/local.rules









**Explanation:**

* **SSH Brute-Force (SID 1000001):**

The rule monitors incoming TCP traffic to port 22 on our protected network and will generate an alert, when a single source IP makes 5 or more established connections to an SSH server within 60 seconds.

* **HTTP Brute-Force/Login (SID 1000002):**

The rule monitors TCP traffic to port 80 and will generate an alert, when a single source IP sends HTTP requests with "login" in the request URL to your protected hosts and makes 10 or more established connections within 60 seconds.

* **SQL Injection (SID 1000003):**

The rule watches TCP traffic to port 80 and will generate an alert, when an established connection to the server contains the case‑insensitive string ' OR '1'='1, indicating a likely SQL‑injection attempt.

* **SMB Exploit (EternalBlue) (SID 1000004):**

Monitors TCP traffic to port 445 and will generate an alert, when an established, server‑bound SMB packet contains the byte sequence 00 00 00 90 FF 53 4D 42 at offset 4, indicating a likely EternalBlue-style SMB exploit attempt.

* **ICMP Ping Sweep (SID 1000005):**

The rule watches for ICMP echo (ping) requests (ICMP type 8) with 0-byte payload to any protected host and will generate an alert, when a single source sends 5 or more such pings within 10 seconds (indicating a ping sweep).

* **Nmap TCP SYN Scan (SID 1000006):**

The rule monitors TCP packets sent to any port on our protected hosts and will generate an alert, when it sees TCP packets with the SYN flag in a stateless context indicating a possible Nmap TCP SYN scan.

* **RDP Brute Force (SID 1000007):**

The rule alerts when a single IP makes 5 or more RDP (TCP 3389) connection attempts within 60 seconds, indicating a possible RDP brute-force attack.

* **FTP Brute Force (SID 1000008):**

The rule alerts when a single IP makes 5 or more FTP (TCP 21) connection attempts within 60 seconds, indicating a possible FTP brute-force attack.

* **DNS Tunneling Suspicious Query (SID 1000009):**

The rule monitors UDP traffic to port 53 (DNS) and will generate an alert, when a DNS query contains long, unusual subdomains, which may indicate DNS tunneling or data exfiltration attempts.

* **SYN Flood (SID 1000010):**

The rule monitors TCP traffic to any port on our protected hosts and will generate an alert, when a single source sends 100 or more SYN packets within 10 seconds, indicating a possible SYN flood DoS attack.

* **Port Scan (SID 1000011):**

The rule monitors all IP traffic to your protected hosts and will generate an alert, when a single source sends 15 or more packets within 30 seconds, indicating a possible generic port scan.

* **Buffer Overflow (SID 1000012):**

Alerts when any IP packet to your protected hosts contains an 8‑byte NOP sled (90 90 90 90 90 90 90 90) within the first 100 bytes, indicating a likely buffer‑overflow/exploit shellcode

attempt.

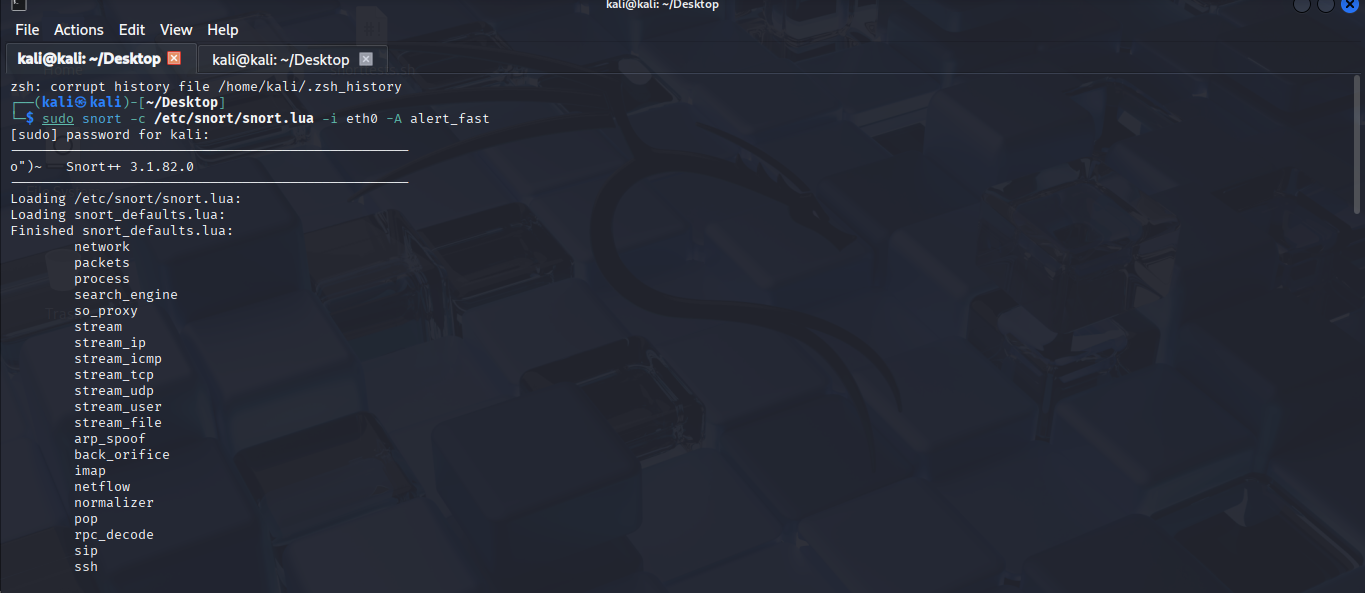
**3.TEST THE RULE**

Here we will check that the snort is running and the custom rules properly generate alerts.

1. **Starting the snort**

Run Snort in console mode to watch for alerts in real-time.

sudo snort -c /etc/snort/snort.lua -i eth0 -A alert\_fast



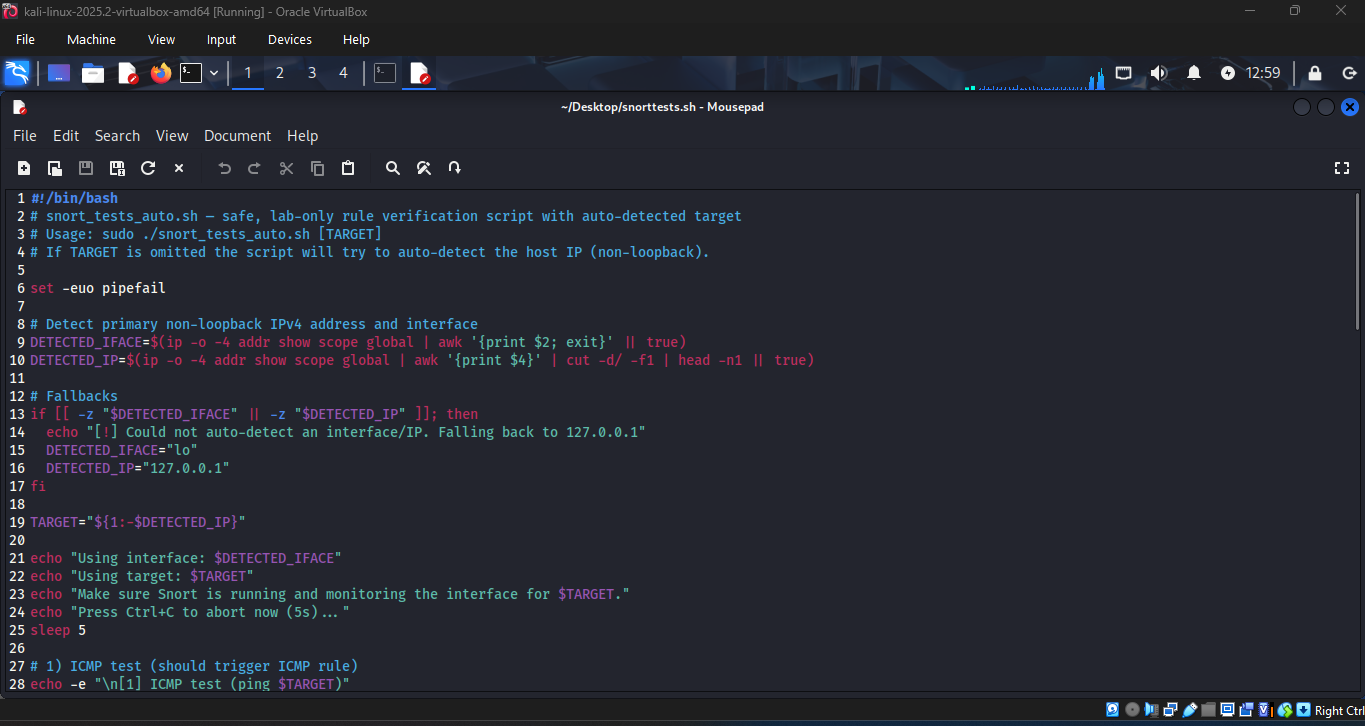


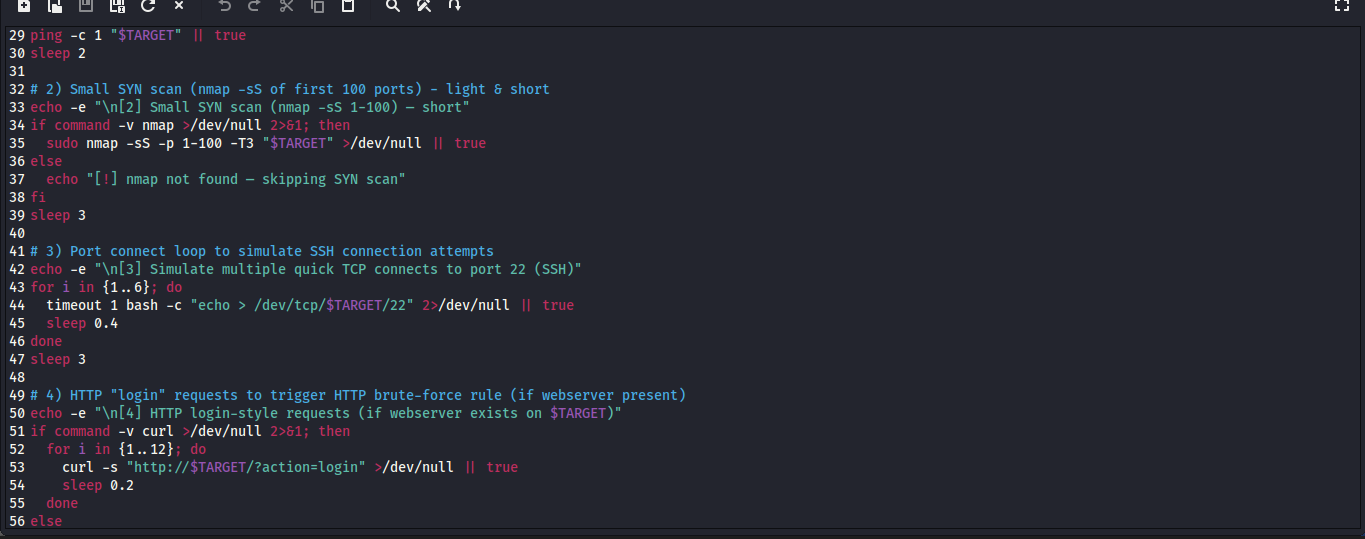
This output indicates that the snort initializes successfully,loads 208 rules, and starts processing packets on eth0.

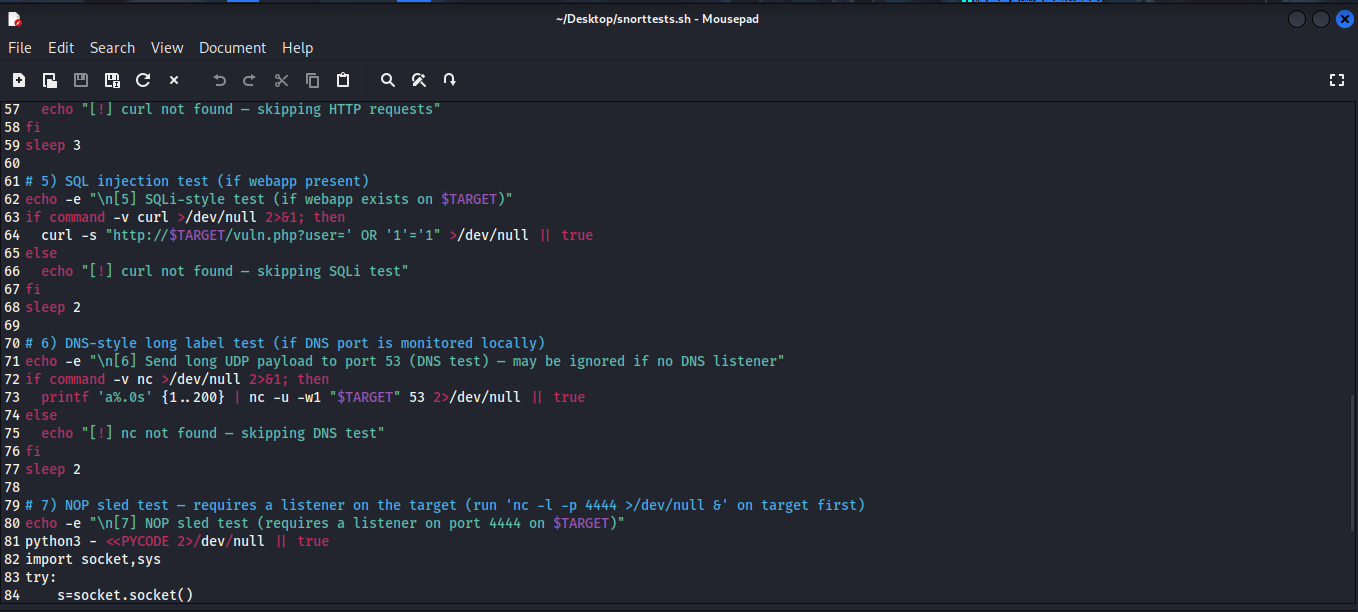
1. **Automated Test Script Creation**

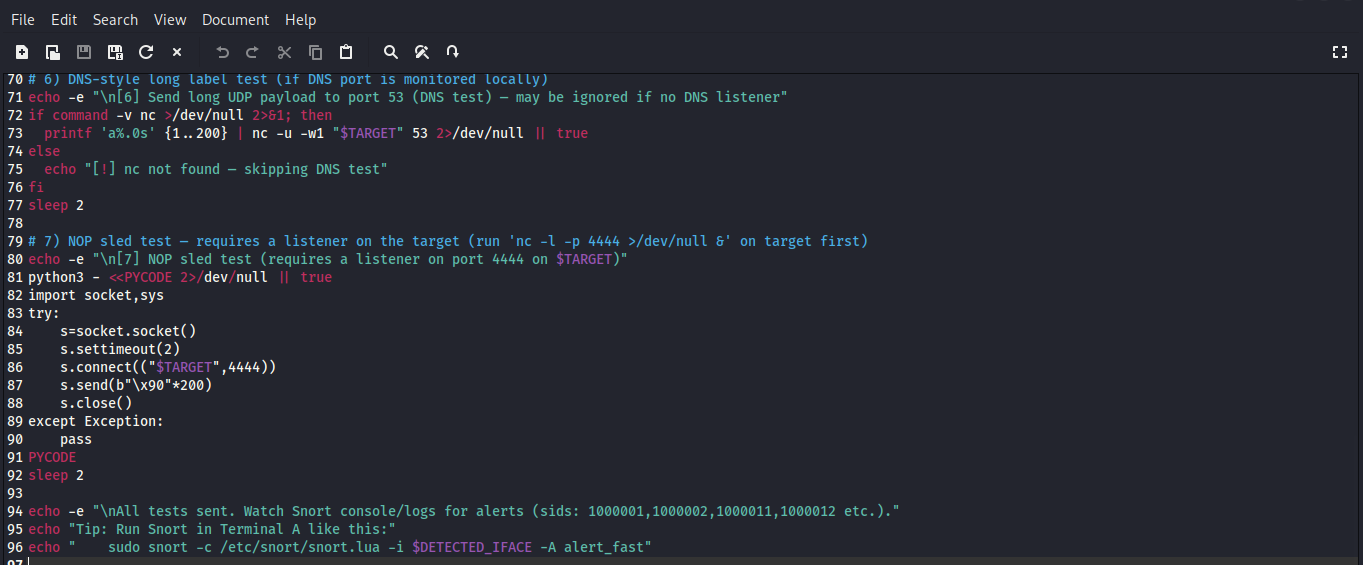
Here we generated a Bash Script to automate network traffic generation that is supposed to activate the custom rules and also the script automatically determines the machine’s IP address and network interface.







****

****

**Explanation of Script:**

This script is a safe, lab-only Snort rule verification tool that automatically picks a target and sends a sequence of test packets/requests designed to trigger common Snort rules. It’s meant for local testing only, not for attacking real systems.

It auto-detects our non-loopback IP , then runs 7 lightweight tests that generate traffic Snort can detect so we can verify those specific SIDs are firing.

* **Test 1 — ICMP ping**

ping -c 1 "$TARGET" 🡪 Sends one ICMP echo request to trigger ICMP-related rules.

* **Test 2 — Small SYN scan**

Uses nmap -sS -p 1-100 -T3 "$TARGET" 🡪 To send a short stealth SYN scan of ports 1–100 to simulate scanning behaviour.

* **Test 3 — Multiple quick TCP connects to SSH**

A loop performs 6 quick TCP connects to port 22 via bash’s /dev/tcp/host/port to simulate multiple SSH connection attempts that triggers brute-force detection thresholds.

* **Test 4 — HTTP "login" requests**

If curl exists, sends 12 quick http://$TARGET/?action=login requests to simulate HTTP login attempts that match rules checking http\_uri for login.

* **Test 5 — SQLi-style request**

Sends a single curl request with user=' OR '1'='1 in the query string to simulate the literal SQLi string your SQLi rule looks for.

* **Test 6 — DNS-style long UDP payload**

If nc (netcat) is installed, it sends ~200 a characters via UDP to port 53 to simulate long/abnormal DNS queries that can indicate DNS tunneling.

* **Test 7 — NOP sled test**

Runs a short-embedded Python snippet to connect to port 4444 and send 200 \x90 bytes (NOP), requires a listener on the target. This exercises NOP-sled detection rules.

* **Finish**

Prints a final message telling you which SIDs to watch for and suggests how to run Snort to view alerts.

1. **Executing the Test**

* The user runs the script (sudo ./snort\_tests\_auto.sh). The output indicates the script executing each test step by step .
* The last message tells the user to monitor the Snort console for alerts that match the SIDs they authored.



**4.VERIFY THE ALERT**

Watch the console where Snort is running. After a few seconds of the attack, you will see the alert message.

