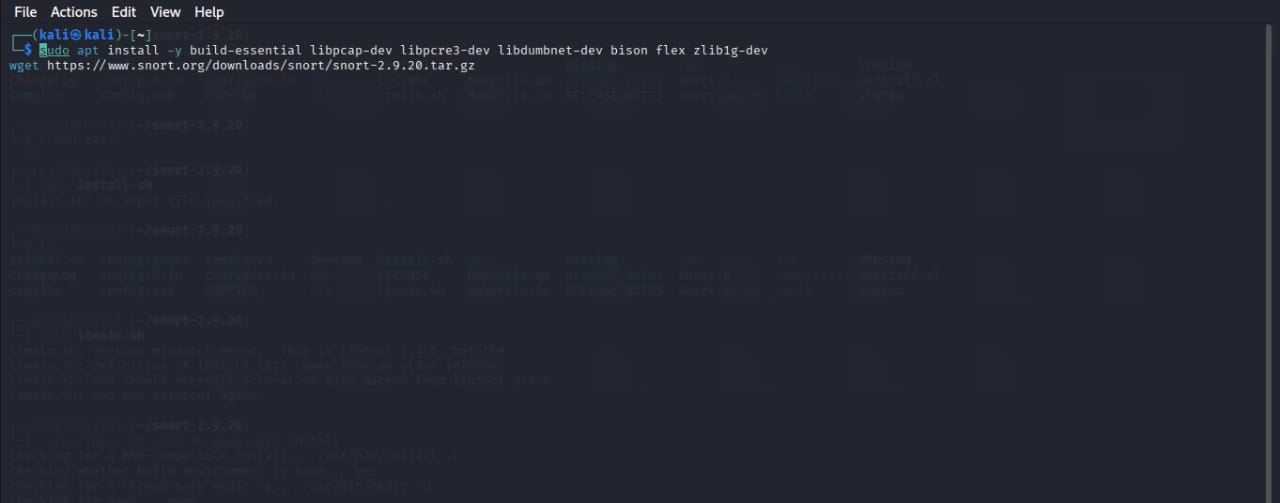
**Overall Summary**

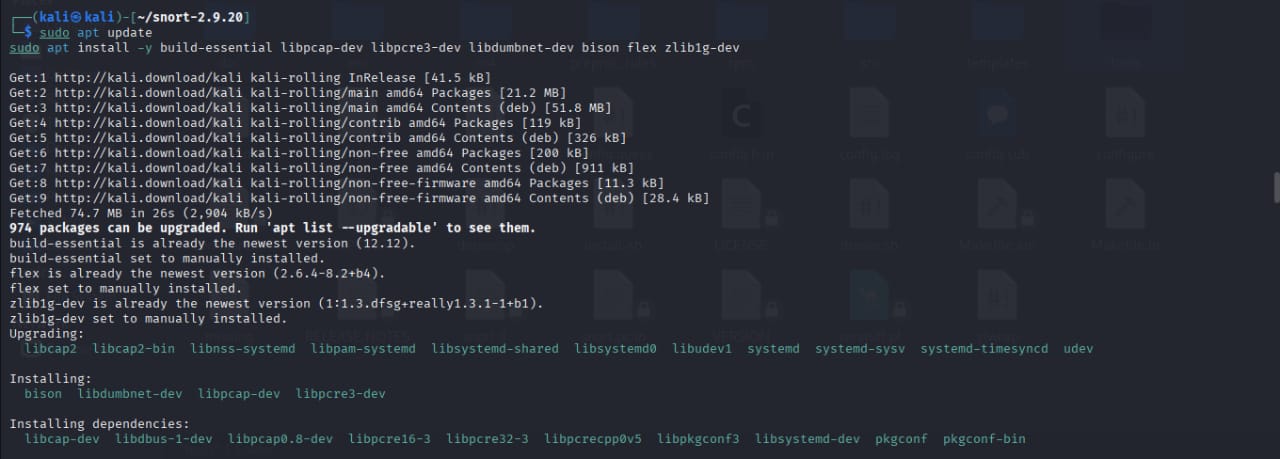
**Step-by-Step Process Explanation**

**Part 1: Preparation & Installation:**

**Objective: Install Snort along with all of its required dependencies.**

**Installing Dependencies:**



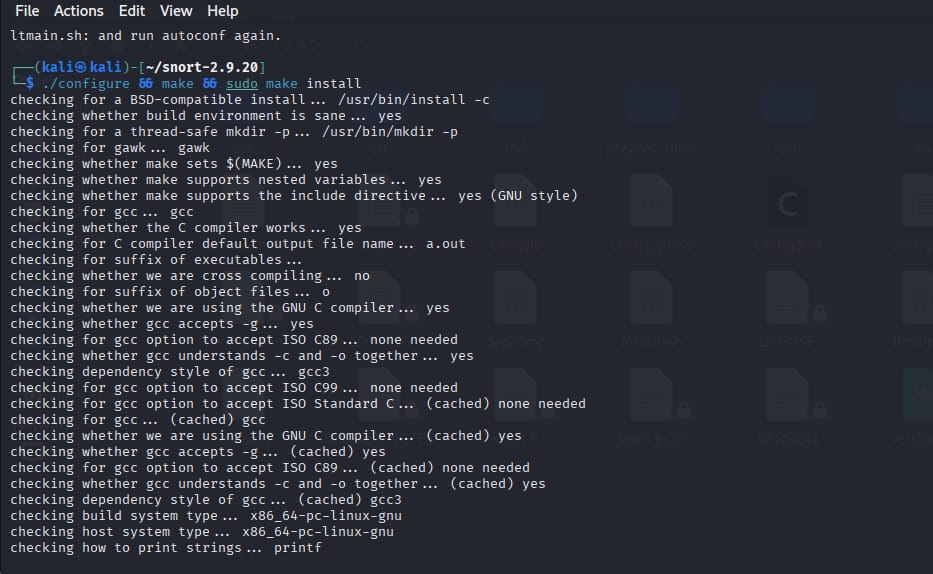
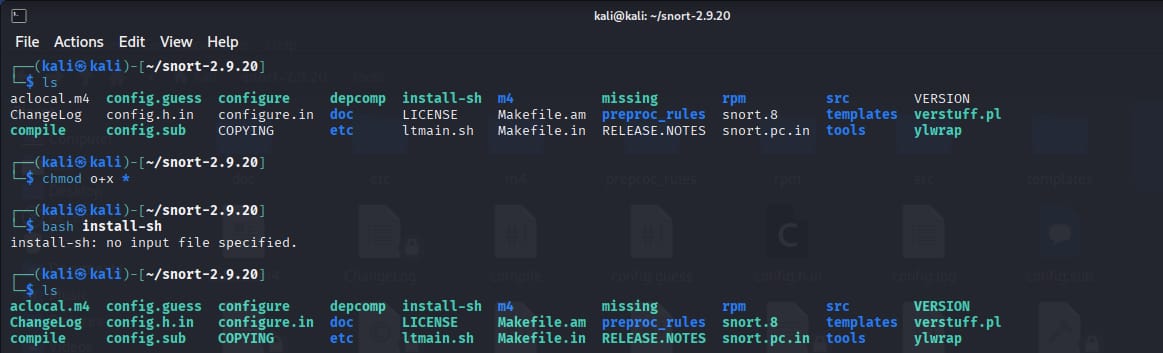


* We attempts to install the libraries that will be necessary to compile Snort (build-essential, libpcap-dev, libpcre3-dev, libdumbnet-dev, bison, flex, zlib1g-dev). There is a typo in the first line of commands (libpcrs3-dev instead of libpcre3-dev, zlibig-dev instead of zlib1g-dev).
* We fix the command (sudo apt update && sudo apt install -y .) and successfully install the dependencies, such as bison, libdumbnet-dev, libpcap-dev, and libpcre3-dev.

1. **Downloading & Extracting Snort:**

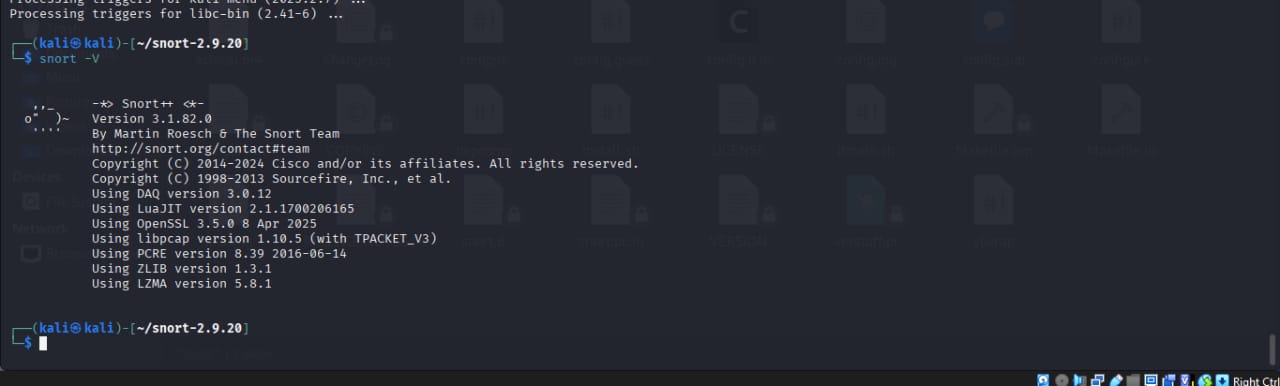
* We download the source code for Snort version \*2.9.20\* (an older version) using wget from the official Snort website.

1. **Building the Configuration:**



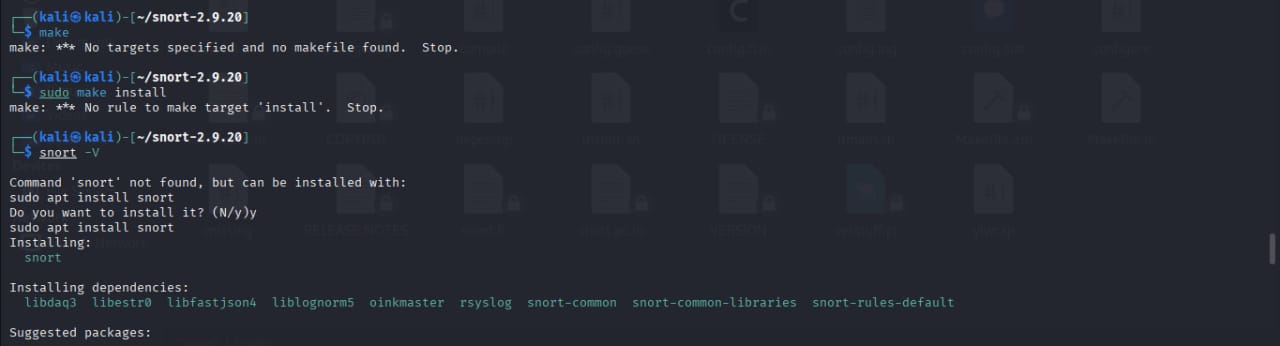
* They change directories into the unzipped snort-2.9.20 and list its contents (ls). The contents include typical files of a source code package built around the GNU Autotools system (e.g., configure, Makefile.in).
* They execute the configure script (Screenshot 3). The script tests the system for every required library, tool, and configuration in preparation for successful compilation. Successful checks for the C compiler (gcc), build environment, and other essential components are indicated in the output.

4. **Compilation and Installation Issues:**



* Following configuration, the general practice is to execute make to build the code and then sudo make install for installation.
* The make command fails with "No targets specified and no makefile found." This indicates the configure script in Screenshot 3 may not have completed successfully or was interrupted, not leaving a Makefile behind.

1. **Package Manager Installation (Fallback):**



* Confronted with a compilation problem, the user resorts to source installation but instead employs the package manager for Kali. They enter snort -v, and upon failure, we accept the suggestion to install it (sudo apt install snort).
* This loads \*Snort 3.1.82.0\* (many versions newer), as verified by issuing snort -v in Screenshot 6. The version, build details, and libraries it loads (e.g., Libpcap, PCRE, OpenSSL) are indicated in the output.

**Part 2: Writing Custom Rules:**

**Objective: Write a collection of custom rules to identify particular malicious activities.**

**alert icmp any any -> any any (msg:"ICMP test rule fired"; sid:1000001; rev:1;)**

**alert tcp any any -> $HOME\_NET 22 (msg:"SSH Brute-Force Attempt Detected";**

**flow:to\_server,established;**

**detection\_filter:track by\_src, count 5, seconds 60;**

**sid:1000002; rev:1;)**

**alert ip any any -> $HOME\_NET any (msg:"Possible Port Scan";**

**detection\_filter:track by\_src, count 15, seconds 30;**

**sid:1000011; rev:1;)**

**alert ip any any -> $HOME\_NET any (msg:"GENERIC Buffer Overflow Attempt - NOP Sled Detected";**

**content:"|90 90 90 90 90 90 90 90|";**

**depth:100;**

**sid:1000012; rev:1;)**

We also modifies the local rules file (/etc/snort/rules/custom.rules) through the nano text editor. They insert a number of rules with distinct Signature IDs (SIDs) beginning from 1000001.

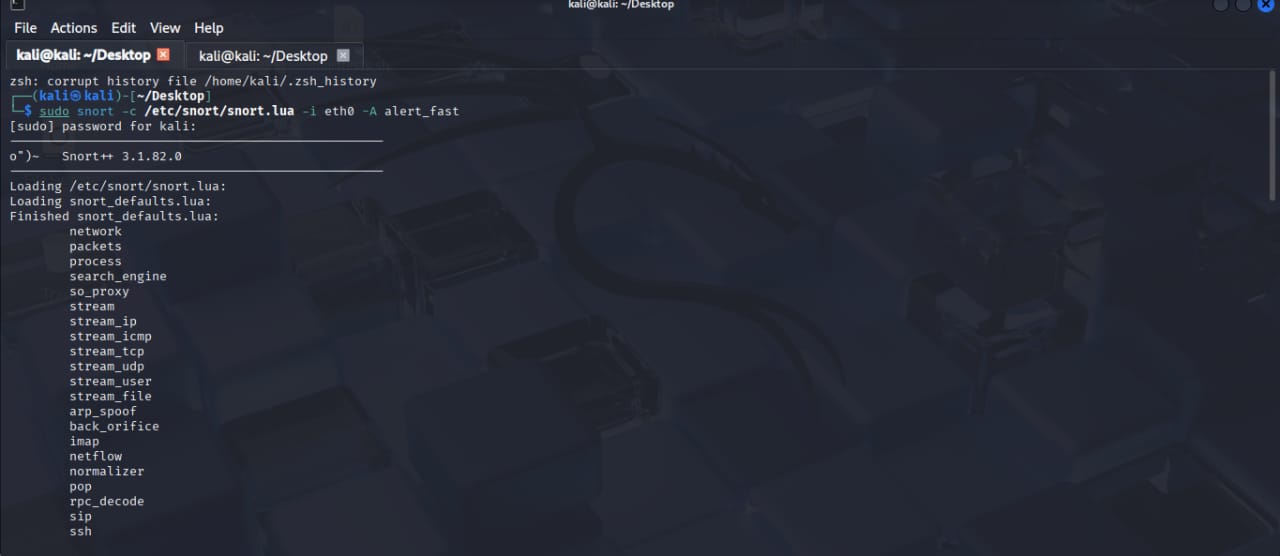
**Main Rules Created:**

* ICMP Ping Sweep (SID 1000001):\* Warns if ICMP ping requests have been received from a single source IP.
* SSH Brute-Force (SID 1000002):\* Alerts when 5 TCP connections to port 22 (SSH) from the same source IP are attempted within 60 seconds.
* Port Scan (SID 1000011):\* Generates an alarm if 15 packets to any port are sent by a single source IP within 30 seconds.
* Buffer Overflow (SID 1000012):\* Notifies if a string of NOP instructions (\\x90\\x90.), typical of exploit payloads, is found in a packet.

**Part 3: Testing the Rules:**

**Objective: Confirm that Snort is executed and that the custom rules properly generate alerts.**

1. **Starting Snort:**





\* Snort using a particular command:

bash

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

\* -c /etc/snort/snort.lua\*\*: Draws on the Snort 3 configuration file (.lua rather than the traditional .conf).

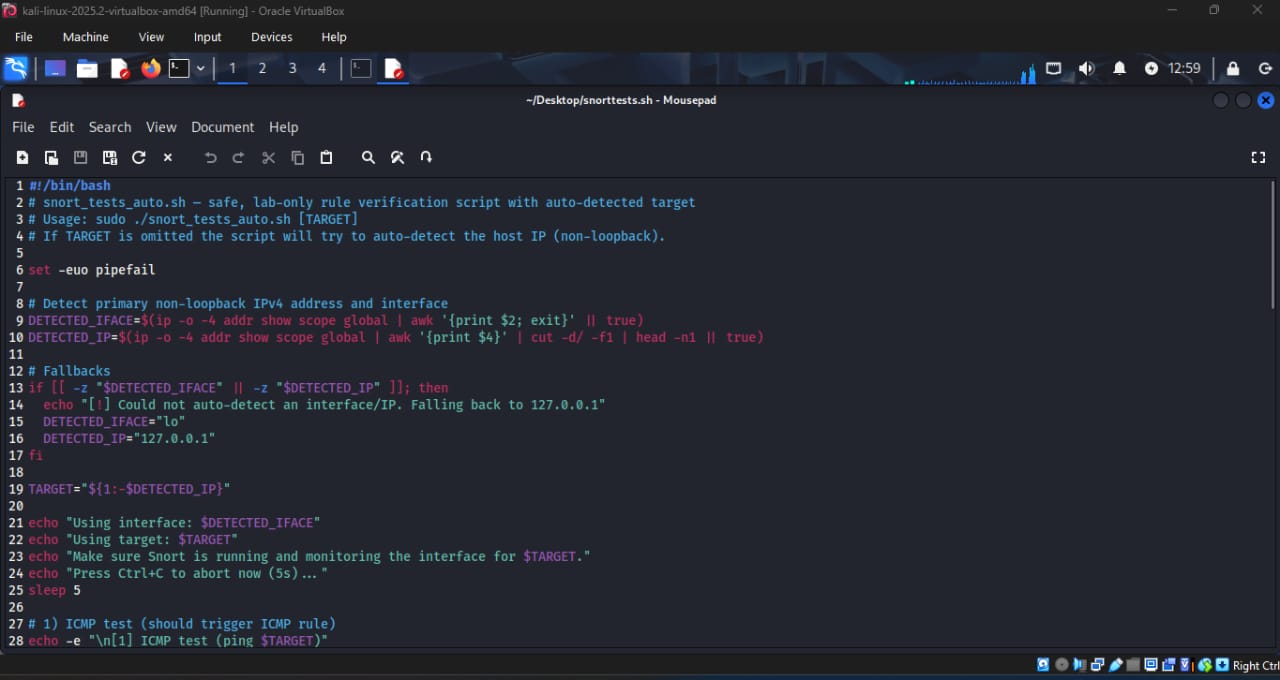
\* -i eth0\*\*: It listens on interface eth0.

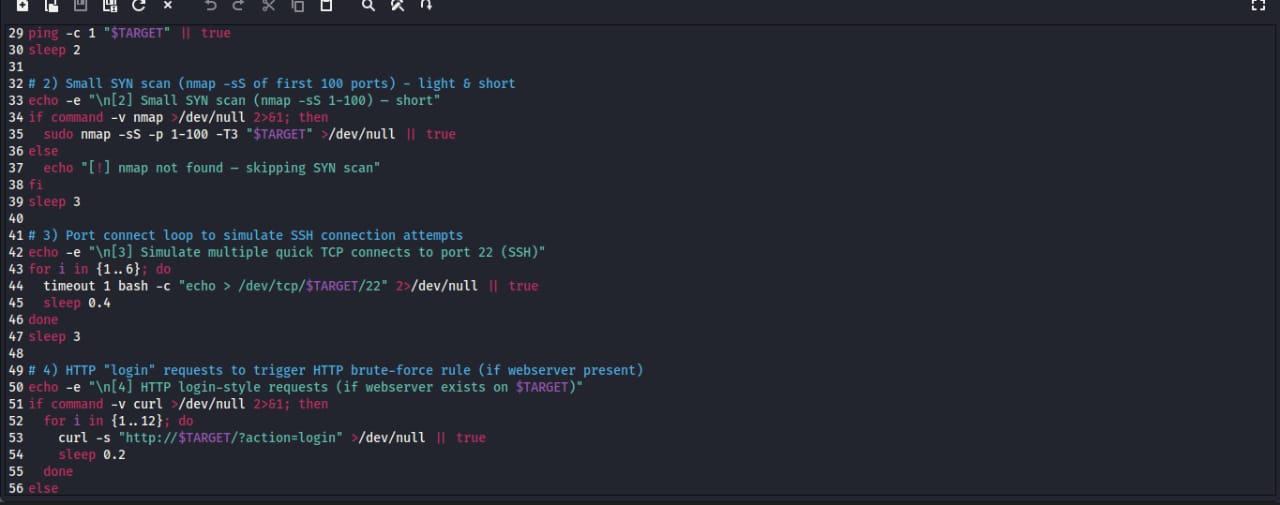
\* -A alert\_fast\*\*: It sets the alert mode to alert\_fast, which outputs alerts rapidly to the console in a plain format.

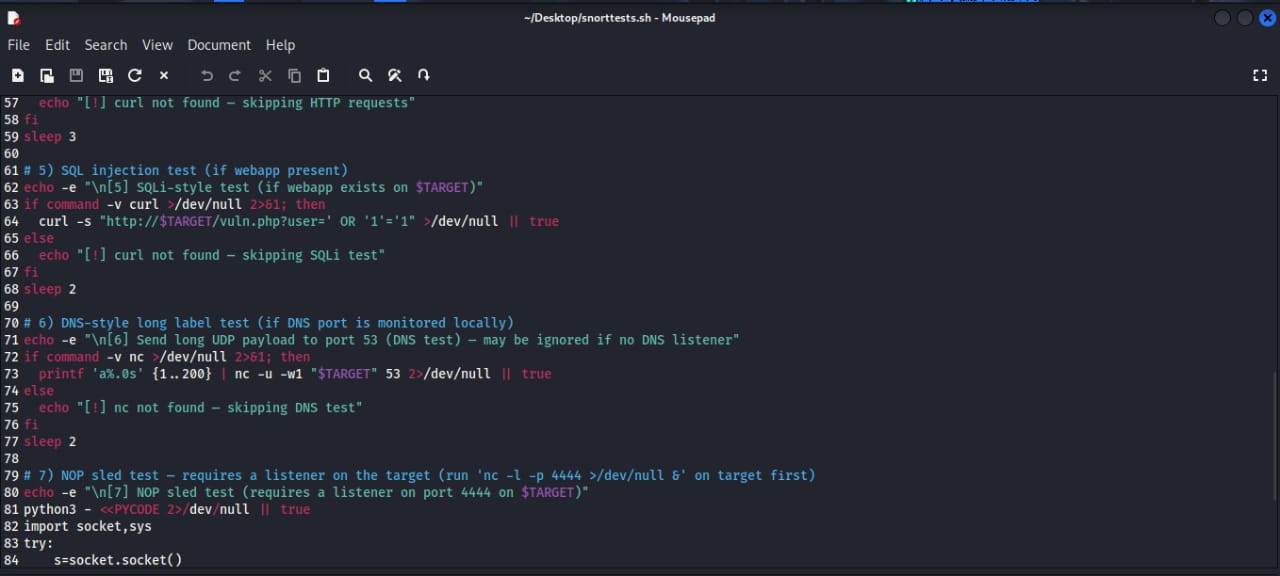
\* The output indicates that Snort initializes successfully, loads 208 rules, and starts processing packets on eth0.

1. **Automated Test Script Creation**:









* The user codes a complex Bash script (snort\_tests\_auto.sh) to automate network traffic generation that is supposed to activate the custom rules.
* The script automatically determines the machine's IP address and interface.

1.) **It proceeds to execute a set of tests**:

* Test 1: Pings using ICMP (should invoke the Ping Sweep rule, SID 1000005).
* Test 2: Performs a light nmap SYN scan to ports 1-100 (should invoke the Port Scan rule, SID 1000011, and SYN Flood, SID 1000010).
* Test 3: Attempts several quick connections to port 22/SSH (should invoke the SSH Brute-Force rule, SID 1000001).
* Test 4: Submits several HTTP requests with "/?action=login" (should invoke the HTTP Brute-Force rule, SID 1000002).
* Test 5: Issues an HTTP request containing a SQL injection payload (should invoke the SQLi rule, SID 1000003).
* Test 6: Issues a long UDP packet to port 53/DNS.
* Test 7: Utilizes Python to transmit a packet with NOP-sled bytes to port 4444 (should invoke the Buffer Overflow rule, SID 1000012).

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 (ping, nmap scan, etc.).
* The last message tells the user to monitor the Snort console (running in a separate terminal) for alerts that match the SIDs they authored.

**Conclusion**

This process illustrates an end-to-end workflow for a budding security analyst or network defender:

1. \*Installation:\* Getting a capable IDS/IPS tool installed.

2. \*Configuration:\* Crafting custom, relevant detection logic specific to particular threats.

3. \*Validation:\* Systematically testing the detection features to confirm that the system performs as designed prior to using it in an actual network environment. The automated test script is a professional feature that enables simple re-testing upon any modifications.