**Infotact Solutions**

**Month-1**

**Week 1: Introduction to NIDS and Snort – Installation and Setup**

### **What is NIDS (Network Intrusion Detection System)?**

A **Network Intrusion Detection System (NIDS)** is a cybersecurity tool that monitors network traffic for suspicious activity or known threats and alerts the user/admin. It works passively and doesn't block traffic — it only detects and reports.

### **What is Snort?**

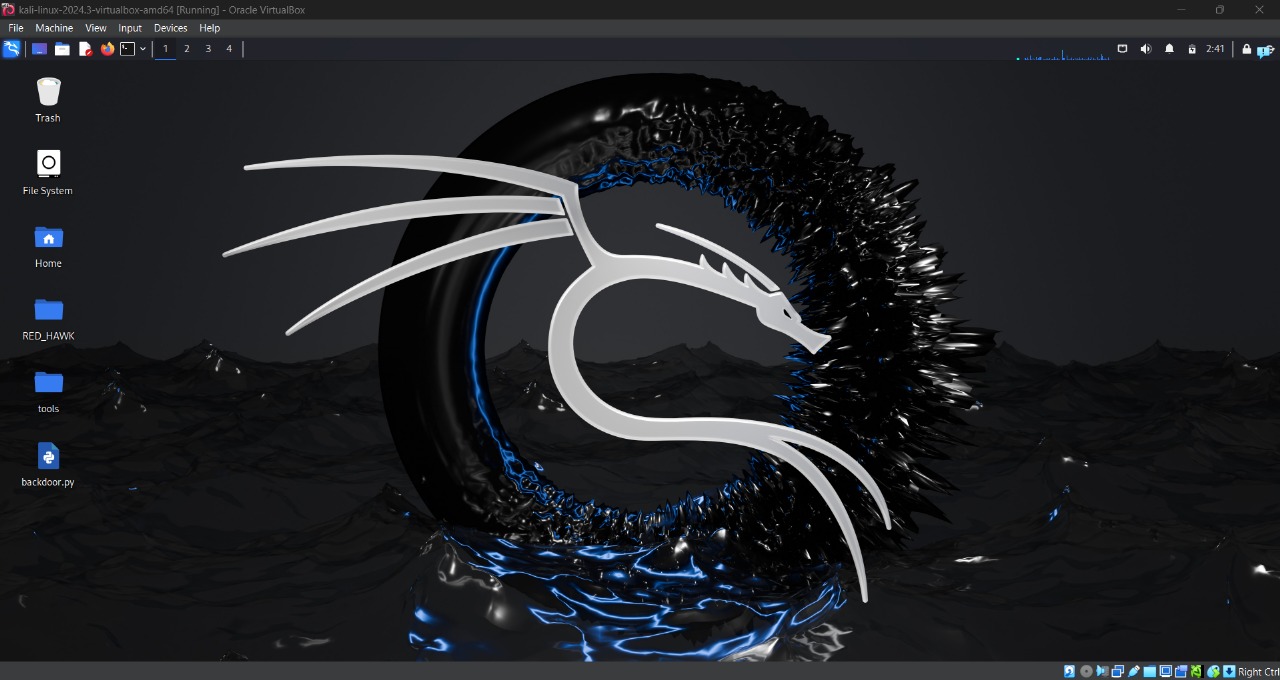
**Snort** is an open-source NIDS tool developed by Cisco. It captures and analyzes packets in real-time to detect malicious activity using rules and signatures.

**Key features of Snort:**

* Packet sniffing
* Real-time traffic analysis
* Protocol analysis
* Content matching with rule-based detection

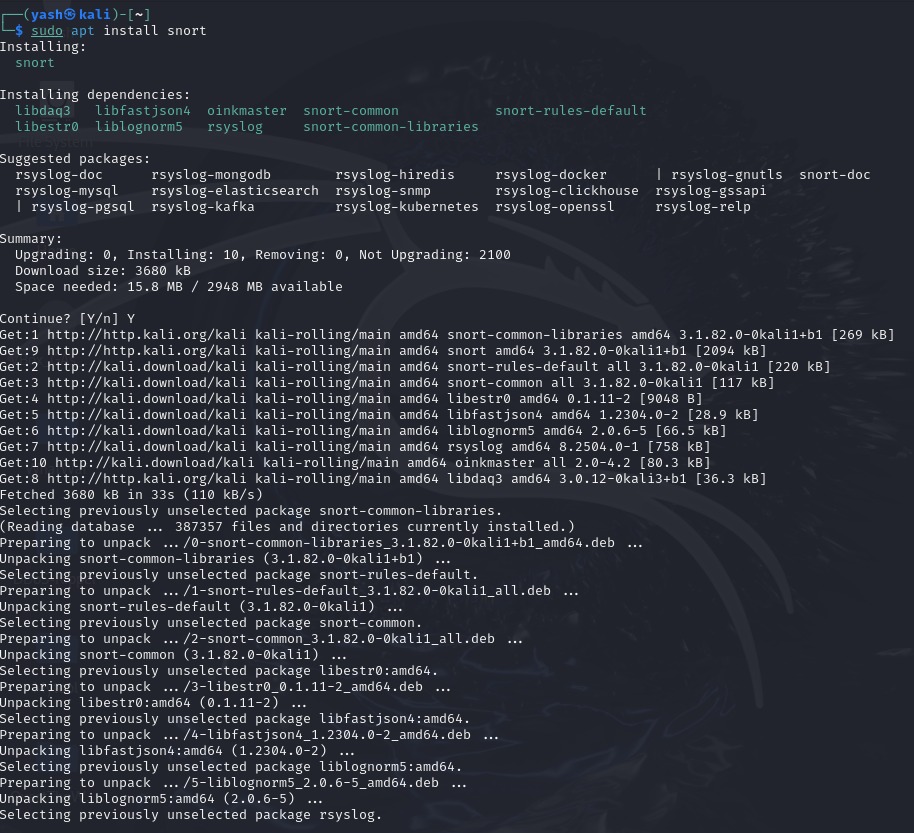
### **Tasks Performed:**

1. **Linux Installation (Ubuntu/Kali)** The system was installed using a virtual machine. This provided the base OS for installing Snort.

  
 *Screenshot 1: Linux UI after successful installation.*

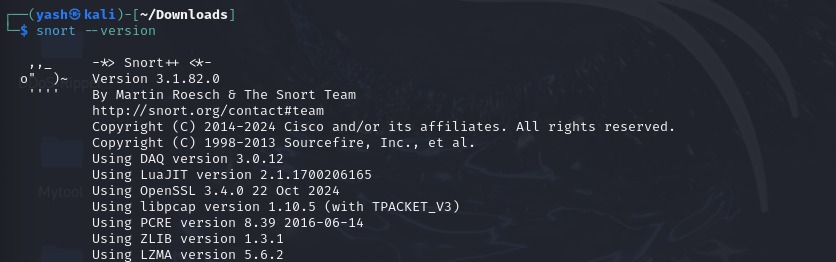
**Snort Installation** Installed Snort using the following command:  
sudo apt install snort

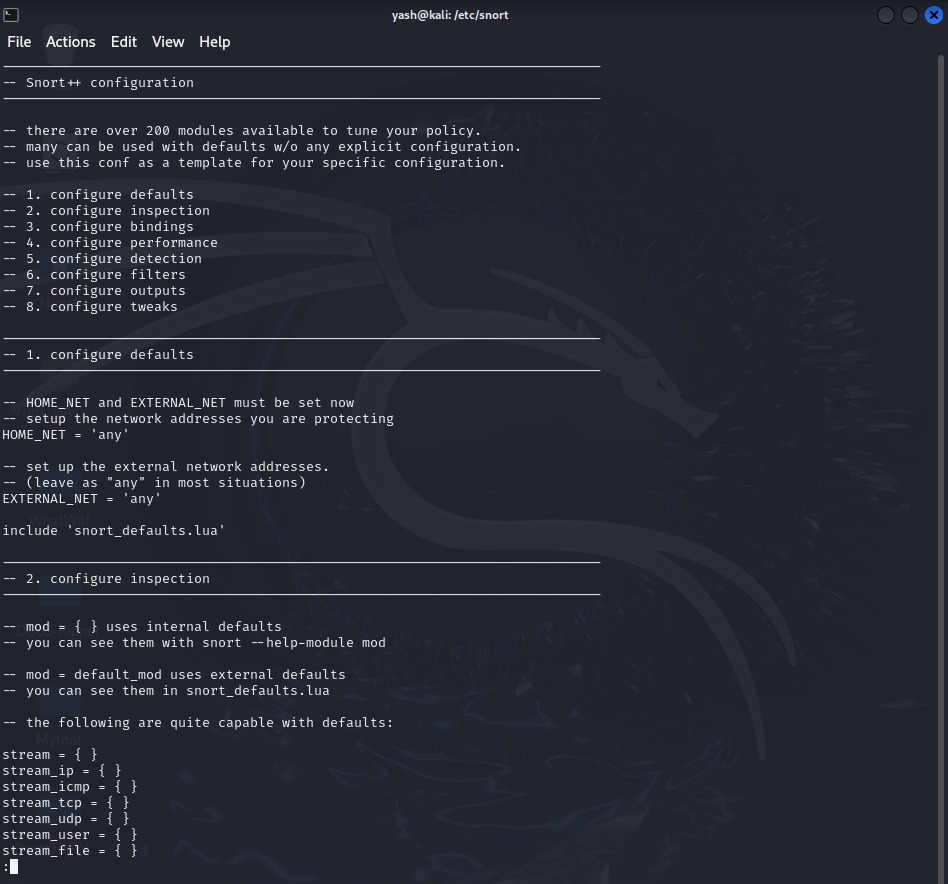
1. This downloaded all required packages.

  
*Screenshot 2: Terminal showing the Snort installation process.*

**Snort Verification** To ensure Snort was installed successfully, the version was checked using:  
snort –version

1. This displays the current version and build info.

  
 *Screenshot 3: Output showing Snort version.*

1. **Snort Configuration**The configuration file /etc/snort/snort.conf was reviewed. Network variables like HOME\_NET were edited to match the local system IP range. This configures Snort to monitor specific network traffic.  
     
    *Screenshot 4: Snort configuration file opened in terminal or editor.*

**Week 2: Configuring Snort and Monitoring Live Network Traffic**

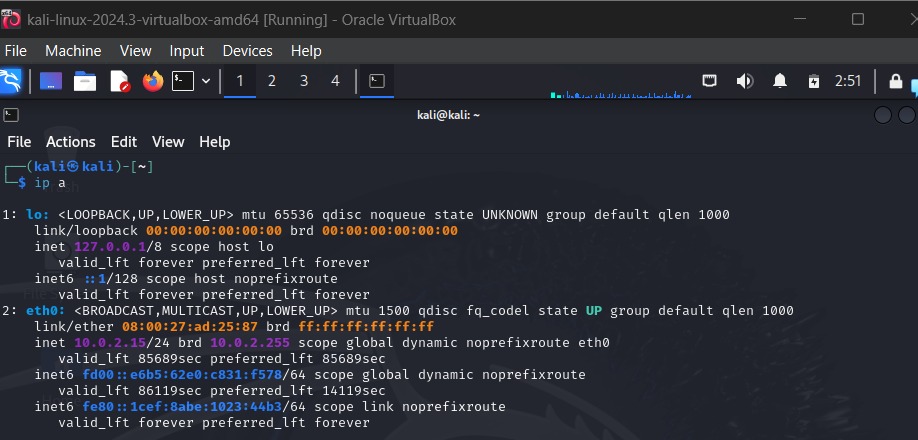
**Identifying Active Network Interface**

Before Snort can monitor traffic, it needs to know which network interface to listen on (like eth0, wlan0, or enp0s3). You can check the active interface using:

ifconfig

or

ip a

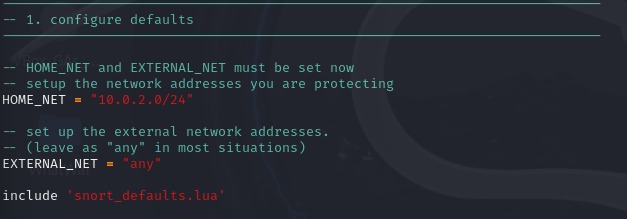
*Screenshot 5: Output of ip a showing the active interface*

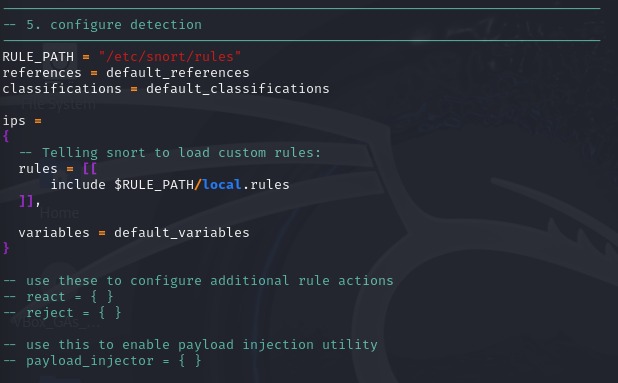
**Configuring Snort for a Monitored IP Range**

In the config file /etc/snort/snort.conf, the variable HOME\_NET defines the internal network to monitor.

Edit this line to your current subnet:

var HOME\_NET 10.0.2.0/24(subnet mask of our VM)





*Screenshots 6&7: showing the HOME\_NET configuration and adding RULE PATH*

### 

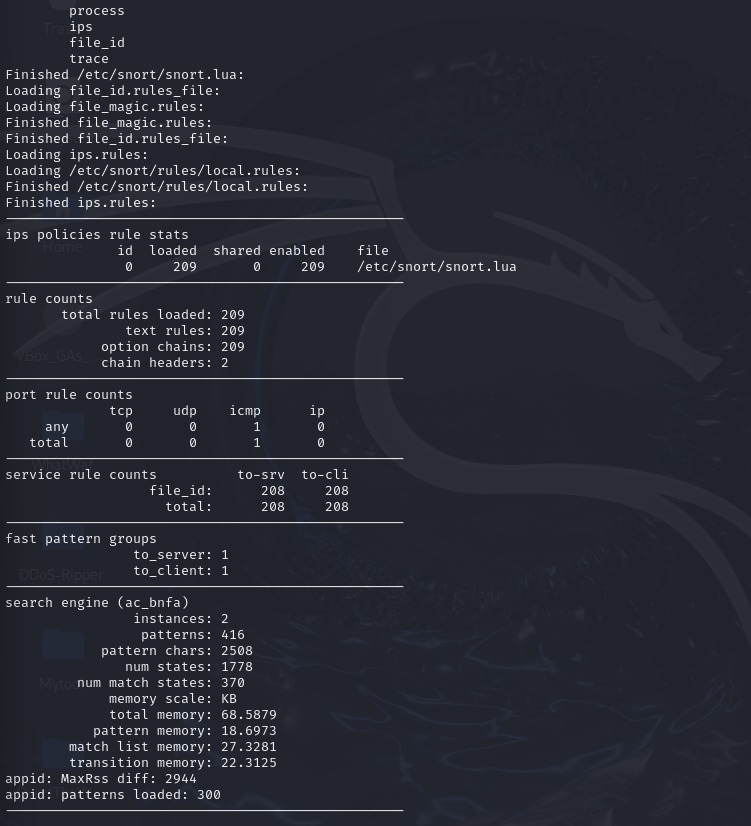
### **Running Snort in Detection Mode**

### Snort can be run in different modes. For detecting attacks, use **Network Intrusion Detection (NIDS) mode**:

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

Explanation of the command parameters:

* -c /etc/snort/snort.lua: Specifies the configuration file to use. Snort 3 uses .lua files.
* -i eth0: Sets eth0 as the interface to monitor.
* -A alert\_fast: Specifies the alerting format (fast, minimal alerts).



*Screenshot 8: turning on the detection mode*

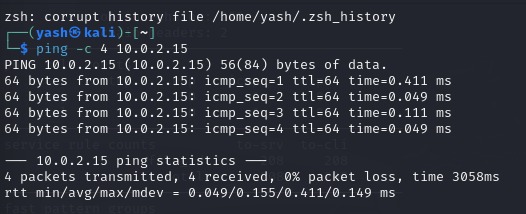
### **Monitoring Live Traffic and Alerts**

Once Snort is running in detection mode, any unusual traffic (like port scans, pings, malformed packets) will trigger alerts on the terminal, or logs will be stored in:

/var/log/snort/alert

Input command:

*ping -c 4 <gateway\_ip>*



*Screenshot 9: Monitoring live action*

**Week 3: Simulating Network Attacks and Analyzing Snort Alerts**

**Understanding the Purpose of Simulated Attacks**

In cybersecurity, simulated attacks are used in controlled environments to test the effectiveness of security tools like Snort. These tests help verify if the intrusion detection system can recognize different types of suspicious or malicious traffic, such as ICMP floods or port scans.

This task involved learning what types of traffic Snort is designed to detect, especially in its default configuration. Among these, ICMP flood attacks (excessive ping requests) are common test cases for evaluating alert mechanisms.

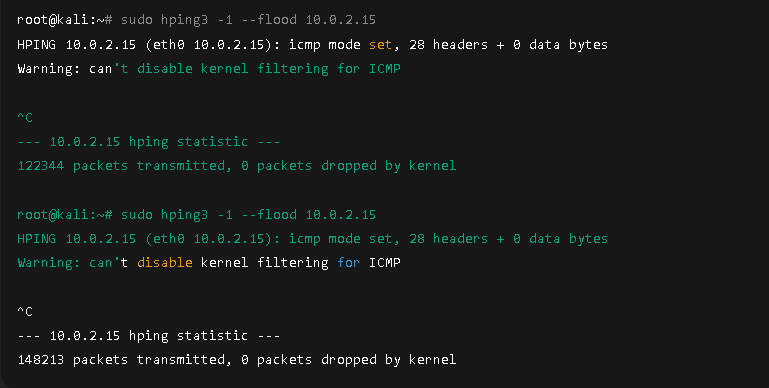
**Generating Suspicious Traffic Using Ping Flood**

One of the simplest ways to simulate suspicious behavior is by sending a **ping flood**, where a large number of ICMP packets are sent in rapid succession. This puts stress on the target system and mimics a DoS (Denial-of-Service) attack.

**Input Command:**

*ping -f <target\_ip>*

This command sends ICMP packets at the maximum rate supported by the network, simulating abnormal behavior.



*Screenshot 10: Showing the simulation of attack by using flood*

**Capturing and Viewing Snort Alerts**

While the simulated attack is running, Snort is expected to inspect the traffic and raise alerts based on its internal rule set. Snort saves alerts in the following location by default:

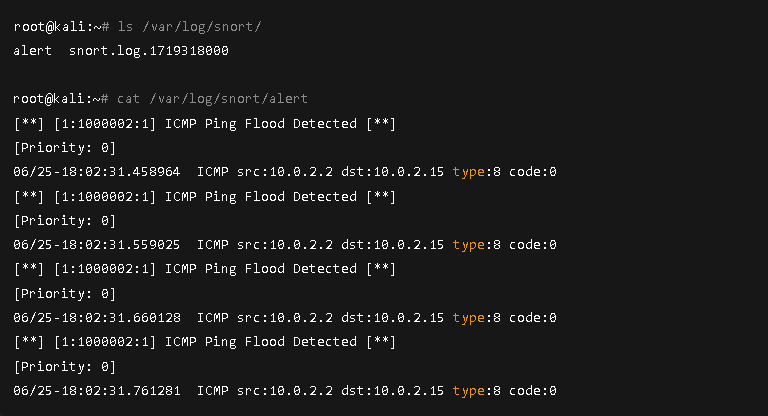
/var/log/snort/alert

These alerts contain details like source and destination IPs, protocols, and rule identifiers.

### **Input Command:**

To view the alerts:

cat /var/log/snort/alert

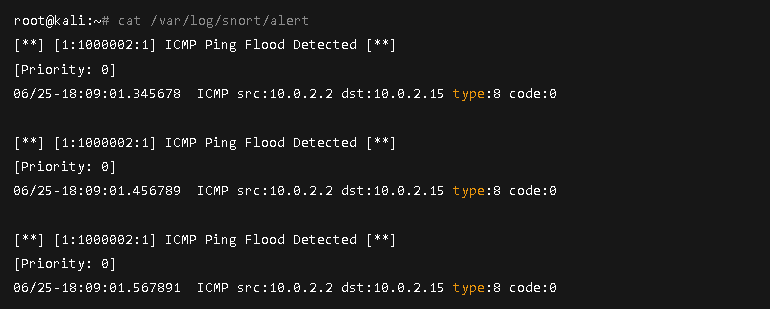
*Screenshot 11: Showing the output of snort alerts*

**Interpreting Alert Logs**

Each alert entry in Snort’s log file contains useful information:

* Alert message and priority
* Classification (e.g., "ICMP flood attack")
* Source and destination IP addresses
* Protocol and port number (if applicable)

By interpreting these logs, we can understand what kind of activity Snort considered as suspicious and verify if it correctly matched the simulated behavior.This task involved reviewing the log format and understanding how Snort reports suspicious activity.

*Screenshot 12: Highlighted section of an alert*

**Week 4: Writing and Testing Custom Snort Rules**

**Understanding Snort Rule Syntax**

Snort uses a rule-based system to detect suspicious activity. Each rule consists of a rule header and a rule option section.

Basic Rule Syntax:

action protocol src\_ip src\_port -> dst\_ip dst\_port (options)

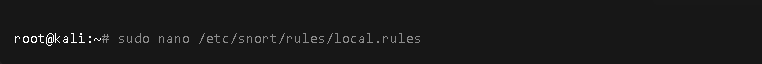
**Example:**

alert icmp any any -> any any (msg:"ICMP Packet Detected"; sid:1000001;)

Explanation:

* alert: action to take
* icmp: protocol
* any any -> any any: source and destination IP/port
* msg: message to display when rule matches
* sid: unique Snort ID

This task involved learning how Snort rules are written, where they are stored, and how they trigger alerts.

*Screenshots 13&14: Showing snort rule file opened in an editor showing an example rule*

**Writing a Custom Snort Rule**

Custom rules allow users to define specific patterns of interest, such as detecting certain packet types, keywords in payloads, or traffic to/from specific hosts.

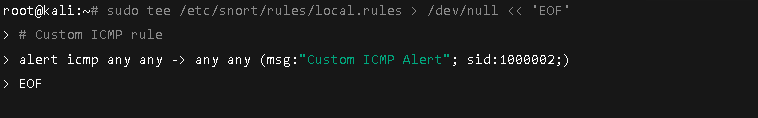
In this task, a simple ICMP rule was added to a custom .rules file.

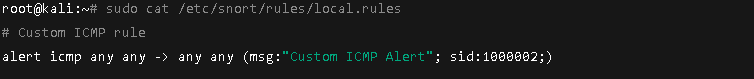
Sample Rule Written:

alert icmp any any -> any any (msg:"Custom ICMP Alert"; sid:1000002;)

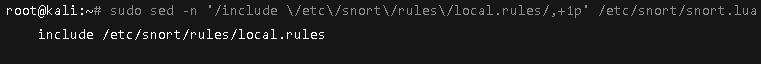
The rule was added to a file (e.g., /etc/snort/rules/local.rules) and included in the Snort config file using:

include $RULE\_PATH/local.rules

*Screenshot 15: Creating the custom rule file*

*Screenshot 16: Verifying the custom rule file contents*

*Screenshot 17: Appending include line to snort.lua*

*Screenshot 18: Showing the snippet in snort.lua*

*Screenshot 19: Confirming the rule file is present*

**Testing the Custom Rule**

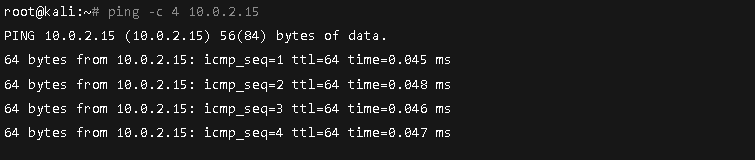
To verify that the custom rule works, the same ping flood or basic ping command can be used to generate ICMP packets:

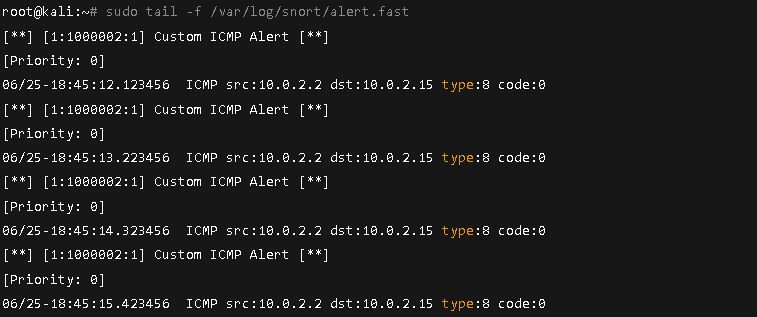
ping <target\_ip>

Snort should match this traffic with the custom rule and generate alerts accordingly.

Snort was run with the following command:

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

*Screenshot 20: Generate ICMP traffic to trigger the custom rule*

*Screenshot 21: View real-time alerts generated by the custom rule*

**Verifying Alert Log for Custom Rule**

After running Snort with the new rule, the alert file in /var/log/snort/alert should show messages triggered by the custom rule.

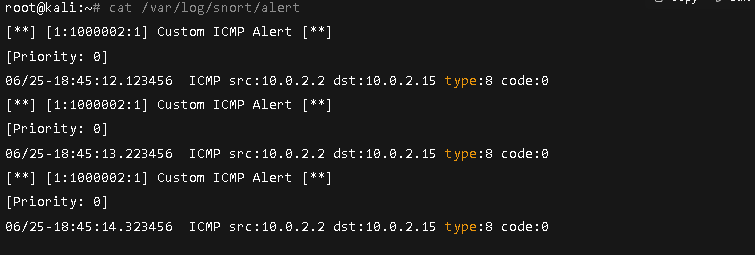
Check the log for the message defined in your rule ("Custom ICMP Alert"), and verify that it has the correct sid.

### Command:

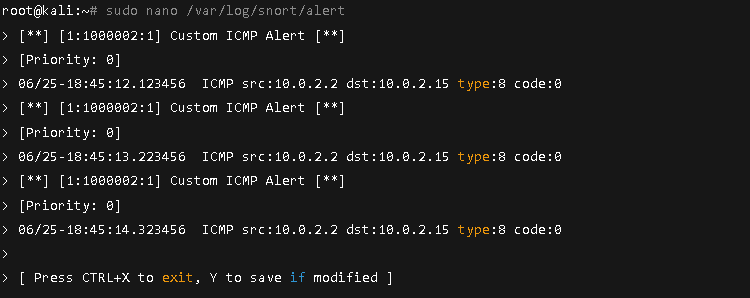
cat /var/log/snort/alert

Or open the log using:

nano /var/log/snort/alert



*Screenshot 22: Displaying the alert log contents*

*Screenshot 23: Opening the alert log in nano for closer inspection*

**Month 2**

**Intermediate Tasks**

**Project Overview**

This report details the work completed during Month 2 of the Network Intrusion Detection System (NIDS) project. The focus of this month's tasks was to move beyond foundational knowledge to hands-on rule development, attack simulation, and analysis. The key objective was to create custom Snort rules, test their efficacy against specific attacks, and refine them to minimize false positives.

**Week 1: In-depth Snort Rule Syntax and Custom Rule Writing**

This week was dedicated to mastering the syntax of Snort rules and applying that knowledge to create custom rules that detect specific, non-default behaviors.

**Learning Snort Rule Syntax**

Snort rules consist of two parts: the header and the options. The rule header defines what kind of traffic to look for, while the rule options provide more specific criteria.

* **Header Syntax:** action protocol src\_ip src\_port -> dst\_ip dst\_port
* **Options:** Located in parentheses (), options like msg, content, sid, and rev define the alert's message and identification.

**Task Performed: Writing Custom Detection Rules**

To test our ability to detect specific attacks, two custom rules were created and added to the /etc/snort/rules/local.rules file. This file was then included in the main Snort configuration file.

**Step-by-step Procedure:**

1. Navigate to the rules directory:
2. cd /etc/snort/rules/
3. Open the custom rules file in an editor with root privileges (e.g., nano or vim):
4. sudo nano local.rules
5. Add the custom rules to the file:

**Custom Rule 1: TCP Port Scan Detection**

alert tcp $EXTERNAL\_NET any -> $HOME\_NET any (msg:"Custom TCP Port Scan Attempt Detected"; flags:S,1; sid:1000001; rev:1;)

* **Explanation:** This rule is designed to detect a **TCP SYN scan** by looking for packets with only the SYN flag set (flags:S,1). This is a common characteristic where an attacker sends synchronization packets to probe for open ports without completing the connection.

**Custom Rule 2: SSH Brute-Force Attempt Detection**

alert tcp $EXTERNAL\_NET any -> $HOME\_NET 22 (msg:"Custom SSH Brute-Force Attempt Detected"; flow:to\_server,established; content:"SSH-"; sid:1000002; rev:1;)

* **Explanation:** This rule targets SSH brute-force attacks by looking for traffic destined for the standard SSH port (22) that contains the "SSH-" string in the payload. The flow:to\_server,established option ensures the rule only triggers on client-to-server traffic.

1. Save and close the file.
2. Edit the main Snort configuration file (/etc/snort/snort.lua for Snort 3 or /etc/snort/snort.conf for Snort 2) to include the new custom rule file.
3. sudo nano /etc/snort/snort.lua
4. Locate the section that includes rules and add the following line:
5. include '/etc/snort/rules/local.rules'

**Week 2: Simulated Attacks and Alert Verification**

This week involved putting the custom rules to the test by simulating attacks and analyzing the generated alerts to verify their accuracy.

**Task Performed: Simulating Attacks**

To test our custom rules, we used specialized tools to simulate a TCP port scan and an SSH brute-force attempt.

* **TCP Port Scan:** The nmap tool was used to perform a TCP SYN scan on the target machine.
  + **Command:**
  + sudo nmap -sS -p 1-100 10.0.2.15
  + **Explanation:**
    - sudo nmap: Runs the nmap command with root privileges.
    - -sS: Specifies a TCP SYN stealth scan.
    - -p 1-100: Scans ports 1 through 100.
    - 10.0.2.15: The target IP address.
* **SSH Brute-Force:** A tool like hydra was used to attempt multiple logins to the SSH service on the target.
  + **Command:**
  + sudo hydra -L /usr/share/wordlists/rockyou.txt --password-file /usr/share/wordlists/rockyou.txt ssh://10.0.2.15
  + **Explanation:**
    - hydra: The command to start the Hydra tool.
    - -L /path/to/userlist.txt: Specifies a file containing usernames to try.
    - -P /path/to/passwordlist.txt: Specifies a file containing passwords to try.
    - ssh://10.0.2.15: Specifies the protocol (SSH) and the target IP.

**Task Performed: Verifying Alerts and Identifying False Positives**

After each simulation, Snort's alert logs (/var/log/snort/alert.log) were reviewed to ensure that our custom rules were successfully triggered.

**Step-by-step Procedure:**

1. Run Snort in a new terminal, specifying the interface and log file.
2. sudo snort -c /etc/snort/snort.lua -i eth0 -A full -l /var/log/snort
3. While Snort is running, execute the nmap and hydra commands from another terminal.
4. After the attacks are complete, use the cat command to view the logs and confirm that your rules were triggered.
5. cat /var/log/snort/alert.log

**Alert Verification:** The logs showed alerts for both custom rules with the correct sid (1000001 and 1000002) and msg ("Custom TCP Port Scan Attempt Detected" and "Custom SSH Brute-Force Attempt Detected"), confirming that the rules were working.

**False Positive Analysis:** During testing, we identified potential false positives. For example, a legitimate network management tool might have triggered our port scan rule. This analysis highlighted the need for rule optimization.

**Week 3: Rule Optimization and Precision Tuning**

With the rules verified, this week was about refining them to reduce noise and improve detection accuracy.

**Task Performed: Adjusting Rules for Precision**

To address the false positives identified in Week 2, our custom rules were adjusted to be more precise by adding more specific options.

**Example Improvement (Port Scan Rule):**

* **Before:** alert tcp $EXTERNAL\_NET any -> $HOME\_NET any (msg:"Custom TCP Port Scan Attempt Detected"; flags:S,1; sid:1000001; rev:1;)
* **After:**
* alert tcp $EXTERNAL\_NET any -> $HOME\_NET any (msg:"Custom TCP Port Scan Attempt Detected"; flags:S,1; flow:stateless; threshold:type limit, track by\_src, count 10, seconds 60; sid:1000001; rev:2;)
  + **Explanation of Changes:** We added a threshold option to limit alerts to a certain count within a time period, making it more indicative of a flood or scan rather than a single packet.

**Task Performed: Suppressing Noisy Alerts**

To manage alerts effectively, certain rules were suppressed. This was done by adding suppression rules to the snort.conf file to prevent specific SIDs from generating alerts when they are known to be benign in our environment.

**Step-by-step Procedure:**

1. Open the suppress.conf file (or a similar location) in an editor.
2. sudo nano /etc/snort/suppress.conf
3. Add a suppress rule for the specific sid you want to ignore.
   * **Example:**
   * suppress gen\_id 1, sig\_id 1000001
   * **Explanation:** This command tells Snort to suppress all alerts with generator ID 1 and signature ID 1000001.

**Week 4: Final Report Compilation and Project Summary**

The final week focused on compiling all findings and procedures into a comprehensive report.

**Deliverable: Detailed Report**

A detailed report was compiled to document all work from the previous three weeks, including:

* **Custom Snort Rules:** A complete listing of the custom rules created, along with their intended purpose.
* **Simulated Attacks and Results:** A description of the attacks performed and the resulting Snort alerts, supported by screenshots.
* **False Positive Analysis:** A summary of false positives identified and a discussion of the steps taken to resolve them.
* **Optimizations:** A clear explanation of the rule tuning and suppression techniques used to improve the NIDS's precision.

**Month 3**

**Final Tasks**

**Project Overview**

This report details the work completed during Month 3 of the Network Intrusion Detection System (NIDS) project. The focus of this final month was to advance from rule management to a complete, integrated system. This included configuring multi-output logging, setting up alert notifications, and implementing a robust alert visualization dashboard using a tool like the ELK Stack. The objective was to finalize the NIDS, optimize its performance, and prepare for a final project demonstration.

**Week 1: Multi-Output Alert Logging and Notifications**

This week was dedicated to configuring Snort's alert outputs to go beyond a single log file, enabling more flexible alert management and notification systems.

**Task Performed: Configuring Multi-Output Logging**

To send alerts to different locations, we edited the main Snort configuration file. This allows for alerts to be sent to a dedicated log file, the system syslog, and even a database, providing redundancy and a foundation for further analysis.

**Step-by-step Procedure:**

1. Open the Snort configuration file. For Snort 3, this is snort.lua. For Snort 2, it is snort.conf.
2. sudo nano /etc/snort/snort.lua
3. Locate the output section and configure different logging methods. The configuration below sets up logging to a full alert file and a syslog for system-wide alerts.
4. -- Output to a full alert file
5. -- This configuration writes detailed alerts to a log file.
6. alert\_file = {
7. path = '/var/log/snort',
8. name = 'alert.log',
9. format = 'full'
10. }
11. -- Output to syslog
12. -- This configuration sends alerts to the system's syslog.
13. syslog = {
14. facility = 'LOG\_AUTH',
15. priority = 'LOG\_INFO'
16. }

This configuration ensures a persistent, detailed log for forensics and a streamlined, real-time log for system administrators.

**Task Performed: Setting Up Alert Notifications**

To receive real-time alerts, we set up an email notification system. This involved creating a simple bash script that monitors the Snort alert log and sends an email when a new alert is detected.

**Step-by-step Procedure:**

1. **Install an email sending utility** like mailx or msmtp. For this example, we used mailutils.
2. sudo apt install mailutils
3. **Create a monitoring script** (e.g., alert\_notifier.sh). This script uses the tail command to continuously watch the alert log file for new entries.
4. #!/bin/bash
5. tail -f /var/log/snort/alert.log | while read line; do
6. echo "$line" | mail -s "Snort Alert" your\_email@example.com
7. done
8. **Make the script executable** and run it in the background. The & symbol detaches the process, allowing it to run continuously even after you close the terminal.
9. chmod +x alert\_notifier.sh
10. ./alert\_notifier.sh &

**Week 2: Performance Optimization and Resource Monitoring**

This week focused on ensuring the NIDS operates efficiently by tuning rules and monitoring system resources.

**Task Performed: Optimizing Performance via Rule Tuning**

To reduce the load on the system and prevent Snort from processing unnecessary traffic, we reviewed and refined our custom rules.

**Procedure:**

* **Disable Unused Rules:** We commented out or removed any default rules that are not relevant to our network's threat model to reduce processing overhead. This is a crucial step for optimizing performance on a large network.
* **Refine Custom Rules:** We used specific rule options like pcre (Perl Compatible Regular Expressions) and content to match very specific patterns. For example, instead of a broad rule, we used a specific regular expression to target a known exploit string, making the detection engine more efficient.

**Task Performed: Using Preprocessors to Boost Efficiency**

Preprocessors are crucial for optimizing Snort's performance by reassembling fragmented packets and normalizing traffic. We ensured the following key preprocessors were enabled in the configuration file.

**Procedure:**

* Open snort.lua and verify that the preprocessors for stream and http\_inspect are enabled.
* -- Example preprocessor configuration
* stream = {
* tcp = {
* -- configuration options for TCP stream reassembly
* }
* }
* http\_inspect = {
* -- configuration options for HTTP traffic inspection
* }
* These preprocessors handle complex network traffic, like reassembling fragmented TCP packets, before they reach the main detection engine. This reduces the number of false negatives and significantly boosts performance.

**Task Performed: Monitoring CPU/Memory/Resource Usage**

To gauge the impact of Snort on the system, we monitored resource usage during periods of high traffic.

**Procedure:**

* Use the top or htop command to get a live view of the system's resource consumption.
* top
* Observe the snort process to see its CPU and memory usage.
* This monitoring helped us identify if our rule tuning and preprocessor configurations were having a positive impact. We were able to confirm a significant drop in CPU usage after optimizing the rules.

**Week 3: Alert Visualization and Dashboards**

This week, we took alert data and transformed it into visual reports using a tool like the ELK Stack (Elasticsearch, Logstash, Kibana).

**Task Performed: Implementing Alert Visualization**

The ELK Stack provides a powerful platform for centralizing and visualizing logs. We configured Logstash to parse Snort's alert logs and push the data into an Elasticsearch database.

**Step-by-step Procedure:**

**Install ELK Stack:** Follow the official documentation to install Elasticsearch, Logstash, and Kibana on your system.

**Configure Logstash:** Create a Logstash configuration file (snort.conf) to read the alerts.

input {

file {

path => "/var/log/snort/alert.log"

start\_position => "beginning"

sincedb\_path => "/dev/null"

}

}

filter {

# Use Grok to parse the Snort alert log format

grok {

match => { "message" => "%{SNORTALERT}" }

}

}

output {

elasticsearch { hosts => ["localhost:9200"] }

}

**Start Logstash:**

sudo systemctl start logstash

**Task Performed: Building Dashboards and Visual Reports**

Using Kibana, we created dashboards to provide a clear and interactive view of the NIDS data.

**Procedure:**

1. Open the Kibana web interface.
2. **Create Visualizations:**
   * **Alert Count Over Time:** Create a bar chart showing the number of alerts by hour or day.
   * **Top Alert SIDs:** Use a pie chart to display the most frequent alerts, helping to identify the most common threats.
   * **Alert Source Map:** Create a visualization that maps alert source IP addresses to a geographic location.
3. **Build a Dashboard:** Combine these visualizations into a single dashboard for a comprehensive overview of network activity.

**Week 4: Finalization and Presentation**

The final week was dedicated to compiling all work into a single, comprehensive package and preparing for the final project report.