Snort-IDS Project Report

ABSTRACT

This project involves the development and simulation of a Network Intrusion Detection System (NIDS) using Snort 2.9. The NIDS is host-based and deployed on an Ubuntu Server running services such as MySQL and SSH. The primary objective of the system is to monitor network activity and generate alerts based on predefined rules for suspicious behaviors, such as port scanning or brute-force attempts against services like SSH.

The alerts generated by Snort are logged and then forwarded to the ELK Stack—comprising Elasticsearch, Logstash, and Kibana—for parsing, analysis, and visualization. This integration enables the creation of interactive dashboards that provide deep insights into network activities and security incidents. These visualizations aid in real-time threat detection and response, while also contributing to threat intelligence and overall network situational awareness.

Table of Contents

Introduction

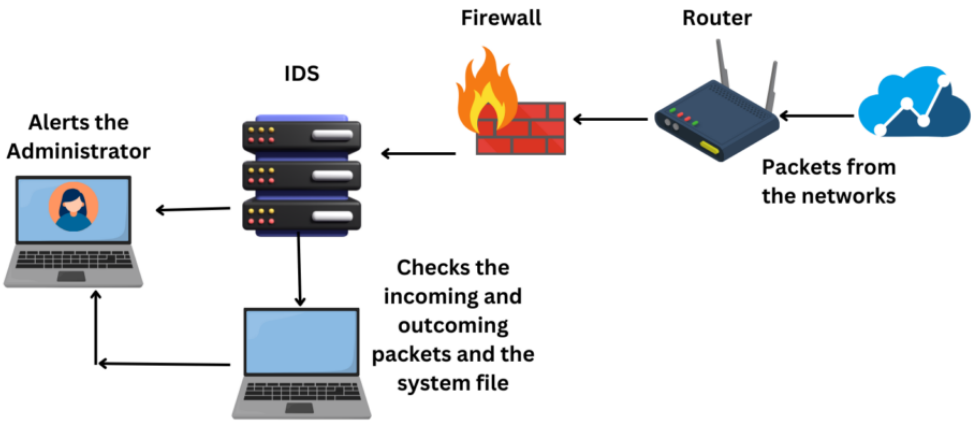
In the modern digital ecosystem, cybersecurity plays a crucial role in defending systems against a wide range of threats—from unauthorized access and malware infections to data theft and system exploitation. As cyberattacks grow more frequent and sophisticated, there is a pressing need for intelligent and proactive defense mechanisms that can detect anomalies and respond effectively in real-time.

One of the critical layers in this defense strategy is the **Host-based Intrusion Detection System (HIDS)**. Unlike Network-based Intrusion Detection Systems (NIDS), which monitor traffic across a network, a HIDS operates directly on a host machine—such as a server or workstation. It inspects system-level activities including file integrity, process execution, system calls, and log files to detect suspicious behavior or unauthorized changes.

This project uses **Snort 2.9**—traditionally known for network intrusion detection—in a host-based configuration on a **Ubuntu Server** running key services like **SSH** and **MySQL**. Snort is configured to monitor system logs and trigger alerts based on predefined rules for suspicious activity, such as brute-force login attempts or port scans. This makes it an effective tool for monitoring critical services on the host and ensuring their integrity.

Alongside intrusion detection, log monitoring and visualization play a vital role in the cybersecurity lifecycle. Logs generated by security tools, operating systems, and network services contain valuable information that can be used to detect anomalies, reconstruct attack timelines, and perform forensic analysis. However, due to the large volume and complexity of log data, manual inspection is often impractical.

Host Intrusion Detection System (HIDS)



Objectives

The objectives of the project are:

- Configure and Install snort 2.9 on Ubuntu Server. The configuration would include writing different snort rules for generating alerts and logs for malicious activities like port scanning and brute-force attacks. Further, the snort should be placed between network and the host so that all incoming traffic could be monitored.

- A separate dedicated Ubuntu Desktop should be created on which ELK Stack should be installed and configured for log analysis and visualization process.

- Simulation of attack on Ubuntu Server to monitor the performance of IDS and log monitoring machine.

- Creating relevant dashboards for log monitoring.

Literature review

1) Why Snort?

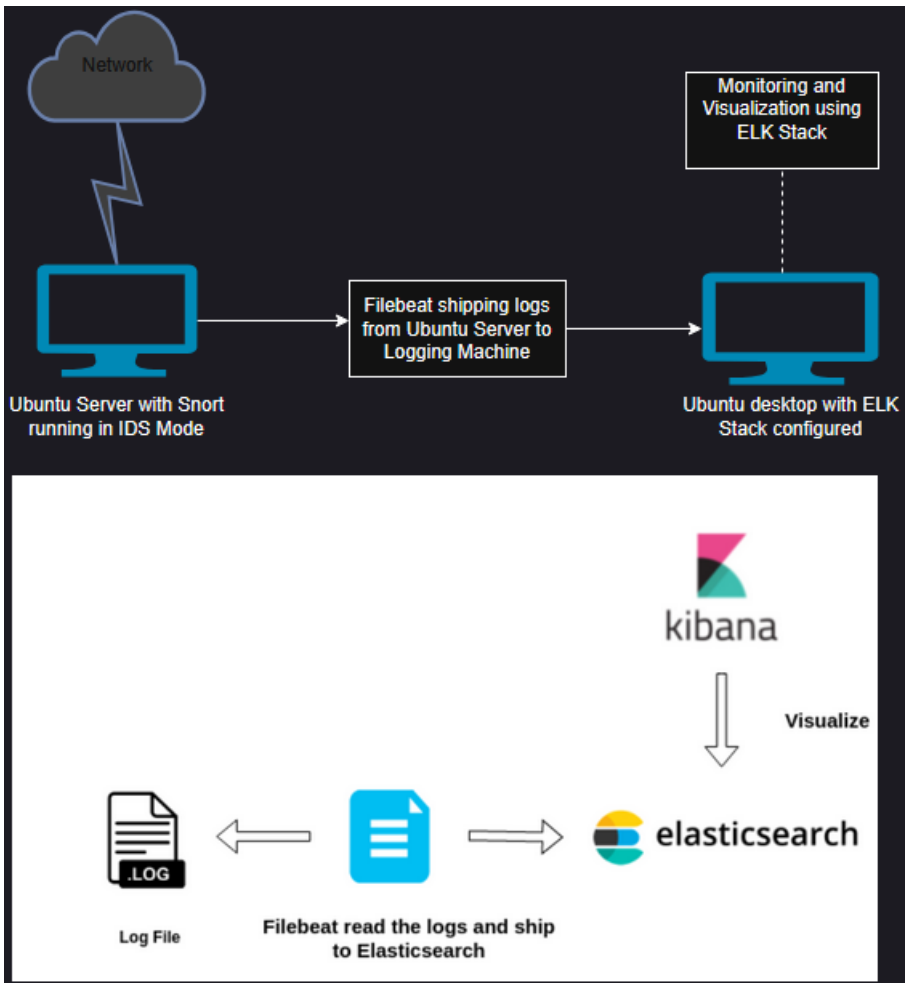
The rising frequency and complexity of cyber threats have led to an increased reliance on Intrusion Detection Systems (IDS) for network and host security. Several studies emphasize the role of IDS in detecting malicious activities such as port scanning, brute-force attacks, and unauthorized access attempts. Snort, an open-source IDS, has gained widespread popularity due to its flexibility and rule-based detection capabilities. Originally designed as a Network-based IDS (NIDS), it has also been successfully deployed as a Host-based IDS (HIDS) for monitoring individual servers and services like SSH and MySQL.

2) Why ELK Stack?

In recent years, the integration of intrusion detection logs with log management and visualization platforms like the ELK Stack (Elasticsearch, Logstash, and Kibana) has improved incident response and forensic analysis. According to existing research, visual dashboards and real-time log analysis enhance situational awareness and reduce the mean time to detect (MTTD) security threats. Various implementations have demonstrated that coupling Snort with ELK allows administrators to efficiently parse, store, and visualize alerts, making it easier to detect patterns and trends in attack behavior.

Thus, the properly deployed Snort + ELK combination can prove to be a powerful and cost-effective factor in security posture of an IT infrastructure.

Architecture



The architecture would consist of :

1) Ubuntu 22.04.5 LTS: It would have snort installed and it would be running in IDS mode to monitor the network traffic against the set of defined rules. Specifications of the server are:

   OS: Ubuntu 22.04.5 LTS
   Base memory: 2048mb
   Processors: 2
   Storage: 25gb
   Network adapter: Host-Only Adapter (Static IP assigned)

2) Ubuntu 22.04.5 LTS Desktop: This machine would receive the snort logs and alerts from server using filebeat and the logs would be processed, analyzed and visualized using ELK Stack installed in it. Specifications of the machine are:

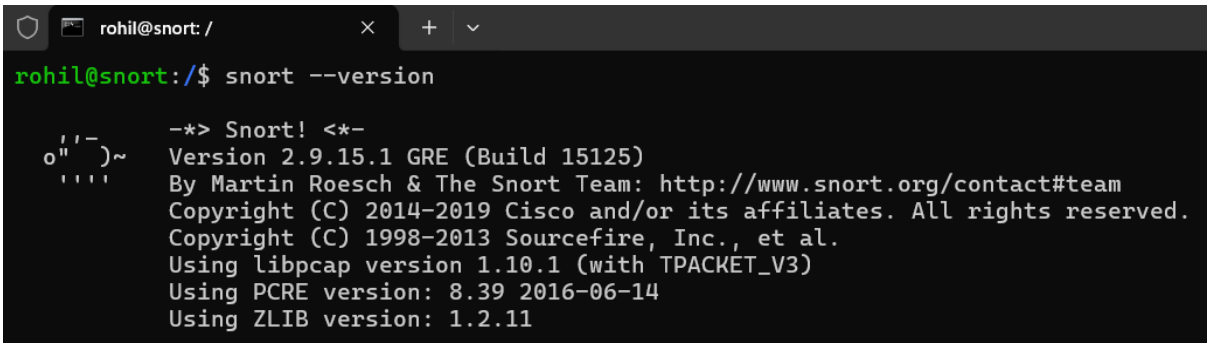
   OS: Ubuntu 22.04.5 LTS
   Base Memory: 4096 MB
   Processors: 3
   Storage: 45gb
   Network Adapter: Host-Only Adapter (Static IP assigned)

Implementation of Snort IDS

➢ Setting up Snort for port scanning detection

A. Following the initial installation of Ubuntu Server 22.04, the Snort intrusion detection system was installed using the command sudo apt-get install snort. This command initiated the installation process on the server.

B. Upon successful installation, the Snort version was verified by executing the command snort --version, confirming that the software was correctly installed and operational.



After the installation of Snort, configuration was carried out by navigating to the directory /etc/snort/, where the primary configuration file, snort.conf, is located.

1. To define the scope of network monitoring, network variables were configured within the snort.conf file. The configuration file was accessed using the command **sudo nano snort.conf**. Within the file, the variable **HOME_NET** was assigned the IP address of the server whose network traffic was to be monitored:

   ipvar HOME_NET 192.168.56.103

2. Additionally, the variable **EXTERNAL_NET** was configured to represent the external network from which potential malicious traffic could originate. In this project, **EXTERNAL_NET** was set to the entire network address range to allow

Snort to monitor traffic from any external device attempting to communicate with the server.

In case of this project, the value of **EXTERNAL_NET** has been assigned as entire network address range to monitor traffic from any device on network to the server.

3. To enhance detection accuracy, an optional variable named **ROUGE** was also defined. This variable was assigned the IP address of a designated attacker machine, which was used during testing to simulate malicious activity. This setup allowed for more targeted detection by Snort.

(Snort 2.9 has default preprocessor which comprises of rules to generate alerts when certain types of port scanning is detected. The preprocessor is called sfportscan preprocessor. To read more about this preprocessor, visit: https://www.snort.org/faq/readme-sfportscan)

4. Within the snort.conf configuration file, further tuning was performed to enable port scan detection capabilities. Specifically, the sfportscan preprocessor, which is responsible for detecting port scanning activities, was configured.

5. Upon locating the section in the configuration file related to sfportscan, it was observed that the corresponding preprocessor rule was commented out by default. This rule was uncommented and modified to ensure it matched the required configuration for effective port scan detection:

```
# Portscan detection.  For more information, see README.sfportscan
#preprocessor sfportscan: proto  { all } memcap { 10000000 } sense_level { low }
preprocessor sfportscan: proto { all } \
    scan_type { all } \
    sense_level { medium } \
    #target_ports {22 3306}
    logfile { /var/log/snort/portscan1.log }
```

The argument logfile is optional. The scan type is set to all to detect any type of scan performed and sense_level is set to medium to maintain accuracy and prevent false positives.

Save changes and exit.

➢ Setting up Snort for SSH Brute-Force detection

1. As part of configuring Snort to detect specific types of suspicious activity, a custom intrusion detection rule was added. This was accomplished by navigating to the /etc/snort/rules directory and opening the local.rules file.

   Within this file, the following rule was appended:

   alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Multiple SSH Login Attempts Detected"; flags:S; threshold:type both, track by_src, count 5, seconds 60; sid:1000001; rev:1;)

   This rule is designed to generate an alert when five or more TCP connection attempts to port 22 (SSH) are detected from the same external IP address within a 60-second window. The use of the flags:S condition ensures that only SYN packets (indicative of new connection attempts) are counted.

   The threshold keyword is used to mitigate the risk of flooding the alert system with repeated notifications, while still ensuring that potentially malicious behavior is flagged promptly. The threshold values were carefully chosen to strike a balance—too high a value may result in missed detections, whereas too low a threshold may lead to false positives. The configuration assumes that legitimate SSH users are unlikely to initiate multiple failed connection attempts within such a short timeframe.

   This rule enhances Snort's capability to detect brute-force SSH login attempts, which are commonly used in early stages of unauthorized access attempts.

2. The Snort configuration file (snort.conf) was reopened, and the line: **#include $RULE_PATH/local.rules** was located. If this line was commented out, it was uncommented to ensure that the custom rules defined in local.rules would be active. The changes were then saved and the editor exited.
3. To verify the correctness of the configuration, the following command was executed: **sudo snort -T -c /etc/snort/snort.conf** . If no errors were returned, the configuration was considered valid.

4. Subsequently, Snort was configured to start automatically during system startup using the command: **sudo systemctl enable snort**
5. Finally, to initiate Snort in Intrusion Detection System (IDS) mode, the following command was executed in the terminal: **sudo snort -A console -q -c /etc/snort/snort.conf -i <interface>**

   Here is the use of different parameters passed in the command are:

   1) **sudo:** To execute this command with highest privileges because snort would sniff the packets to monitor them which would require the highest privileges.
   2) **snort:** To invoke the snort service.
   3) **-A console:** To set the alert mode to console.
   4) **-q:** To run snort in quiet mode i.e it won't run in verbose mode.
   5) **-c /etc/snort/snort.conf:** Path to the snort config file which we want to use.
   6) **-i enp0s3:** To define the network interface whose traffic we want snort to monitor.

➢ Simulating Malicious Activity for Snort Performance Verification

To evaluate Snort's effectiveness in detecting malicious activities such as port scanning and brute-force login attempts, simulated attacks were conducted from an external system referred to as the attack box. This attack box is a Kali Linux-based virtual machine (VM) configured within the same network address space as the target Ubuntu server.

By maintaining both systems on the same subnet, direct communication was ensured, allowing for the generation and monitoring of network traffic under controlled test conditions.

- **Attack Simulation using Port Scanning**

1) Nmap was used to perform Port Scanning on Server

```
┌──(kali㉿kali)-[~]
└─$ nmap 192.168.56.103
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-08 14:54 +09
Nmap scan report for 192.168.56.103
Host is up (0.00087s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
3306/tcp open  mysql
MAC Address: 08:00:27:34:E1:47 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 17.09 seconds

┌──(kali㉿kali)-[~]
└─$
```

2) The Attack VM completed the scan and showed the result
3) Thereafter, the terminal on server was checked to ensure that IDS was detecting it.

```
rohil@snort: ~
rohil@snort:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
04/08-05:54:24.296347  [**] [1:1418:11] SNMP request tcp [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP}
 192.168.56.102:43983 -> 192.168.56.103:161
04/08-05:54:24.325676  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attempted Information Leak] [Priority: 2
] {TCP} 192.168.56.102:43983 -> 192.168.56.103:705
```

4) Upon verifying, it was observed that Snort detected the malicious network activity and generated an alert message on terminal windows.

   These alerts indicate that the scanning host (192.168.56.102) attempted to probe SNMP-related ports on the server (192.168.56.103):

   - **Port 161** is the default port for **SNMP (Simple Network Management Protocol)**.
   - **Port 705** is associated with **SNMP AgentX**, which allows different SNMP sub-agents to communicate with a master agent.

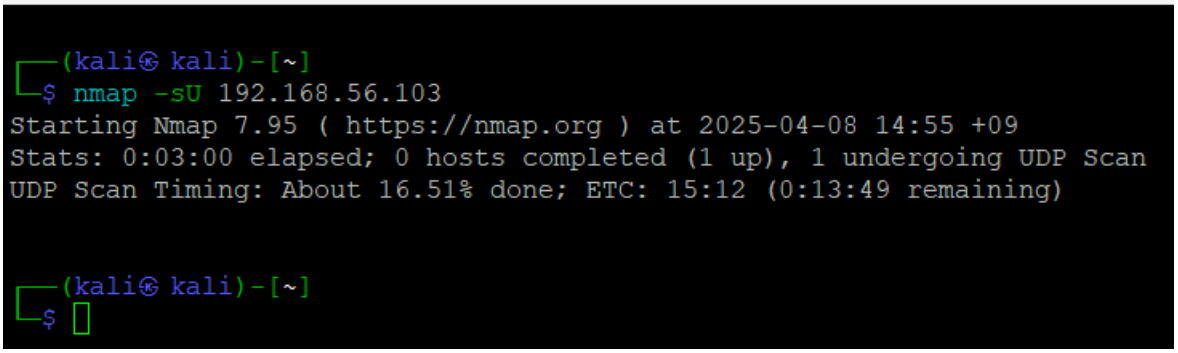
   Snort triggered the rules with signature IDs 1418 and 1421, which are designed to detect potential **information gathering attempts** through SNMP. These are classified as **"Attempted Information Leak"**, meaning the scan is likely trying to enumerate system details or configurations via SNMP.

This detection confirmed that the Snort was effectively monitoring the server and flagging suspicious traffic that could lead to reconnaissance or exploitation.

- **UDP and Ping Sweep Scan Simulation**

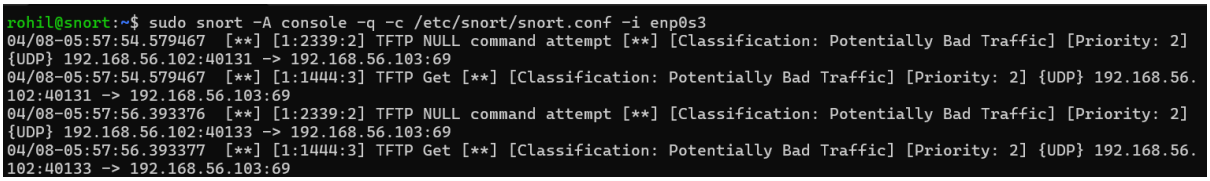
Using the attack VM, different types of Scans like UDP and Ping Sweep Scans were performed to verify that Snort was detecting it all

1) Performing UDP Scan:



As there were no UDP services running on the server, it did not display any output. Therefore, the scan was terminated midway as it was time intensive and goal of the simulation was that whether this port probing attempt is detected in server or not and as the UDP scan (despite being terminated) will probe the ports in target environment, it should be detected.

2) Verification in Server to check the detection,



Breaking the alert messages for explanation:

During a UDP scan (likely using tools such as nmap -sU), multiple alerts were generated by Snort on the monitored Ubuntu server. These alerts correspond to activity directed at **port 69**, which is associated with the **Trivial File Transfer Protocol (TFTP)**. TFTP is a simple, UDP-based file transfer protocol that is known for minimal security features, making it a common target for reconnaissance.

Key Alerts Identified:

1. **TFTP NULL command attempt**
    - This alert was triggered when the scanning host sent malformed or empty TFTP packets (likely without a valid operation code). These types of probes are generally used to detect the presence of a TFTP service without initiating a full file transfer.
    - **Classification:** Potentially Bad Traffic
    - **Priority:** 2

2. **TFTP Get**

- Indicates that the scanner sent a **read request (RRQ)** to the TFTP service. This is a typical behavior observed in scans trying to assess whether anonymous or unauthenticated file access is possible.
- **Classification:** Potentially Bad Traffic
- **Priority:** 2

Common Attributes:

- **Source IP:** 192.168.56.102 (scanner/attacker)
- **Destination IP:** 192.168.56.103 (Snort-monitored server)
- **Destination Port:** 69 (TFTP)
- **Protocol:** UDP
- **Time Stamps:** Alerts are closely timed, indicating **automated scanning**

Snort's Role:

Snort successfully flagged these as **potentially malicious UDP packets**, classifying them under information gathering or suspicious behavior. These alerts, although not indicating a full-blown attack, are important indicators of possible **reconnaissance**, often the first step in the cyber kill chain.

Similarly, now performing Stealth Scan and Ping Sweep Scan to Monitor the Snort:

1) Stealth Scan Simulation

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sS 192.168.56.103
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-08 15:21 +09
Nmap scan report for 192.168.56.103
Host is up (0.00048s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
3306/tcp open  mysql
MAC Address: 08:00:27:34:E1:47 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 17.15 seconds
```

2) Detection in snort:

```
rohil@snort:~$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s3
[sudo] password for rohil:
04/08-06:21:37.238245  [**] [1:1418:11] SNMP request tcp [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP}
 192.168.56.102:39499 -> 192.168.56.103:161
04/08-06:21:37.240227  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attempted Information Leak] [Priority: 2
] {TCP} 192.168.56.102:39499 -> 192.168.56.103:705
```

3) Ping Sweep Scan Simulation

```
┌──(kali㉿kali)-[~]
└─$ nmap -sP 192.168.56.103 --disable-arp-ping
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-08 15:24 +09
Nmap scan report for 192.168.56.103
Host is up (0.00063s latency).
MAC Address: 08:00:27:34:E1:47 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Nmap done: 1 IP address (1 host up) scanned in 16.85 seconds
```

4) Detection in Snort:

Note regarding Snort:

ARP requests do not generate alerts in Snort, since Snort operates at the IP layer and above.

Disabling ARP makes Nmap send ICMP echo requests, which are visible to Snort and can trigger alerts for reconnaissance or ping sweep attempts.

This is essential for validating that Snort is correctly detecting host discovery techniques used by attackers.

1) Attack Simulation by Brute-Forcing SSH

The ideal rule for detecting Brute-Force attempt on SSH service is to monitor the multiple login attempts on SSH service in small amount of time from same network origin.

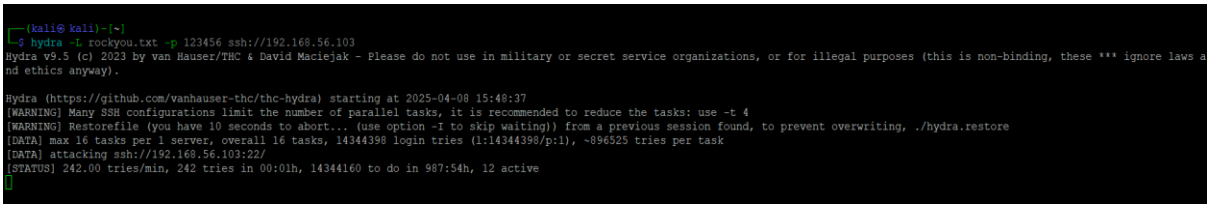
The rule written for this purpose was:

alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"Multiple SSH Login Attempts Detected"; flags:S; threshold:type both, track by_src, count 5, seconds 60; sid:1000001; rev:1;)

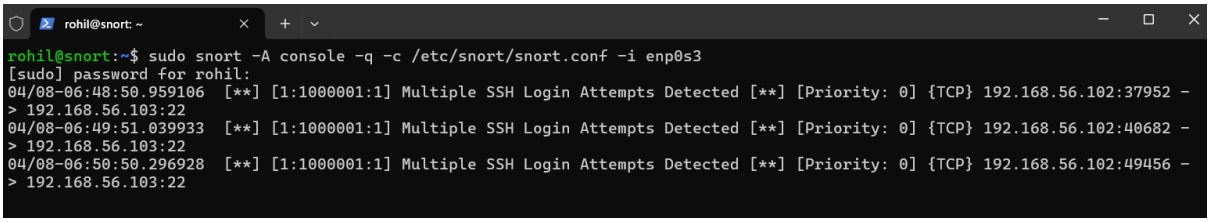
This rule was added in Local Rules file of snort. (Mentioned in earlier steps)

Simulation:

1) First, Snort was started in IDS Mode in Server
2) Then, the Hydra tools was used in attack VM to Brute Force SSH Service. The command used in Hydra was: **sudo hydra -L rockyou.txt -p 12345 ssh://192.168.56.103**



3) Verification in Snort IDS:



Hence, the snort was successfully detecting multiple SSH login attempts originating from same network address.

The rule written to detect such malicious activity can always be tweaked and tuned to meet the specific infrastructure and security requirements.

Logs in Snort

Various types of logging options in snort are:

a) **Unified2**:
   Binary format logs for tools like Barnyard2.

b) **Fast Alert (snort.alert.fast)**:
   Simple, human-readable alert log for quick review.

c) **Full Alert (snort.alert.full)**:
   Detailed logs with packet payload in hex/ASCII.

d) **Binary Packet Logging (-b)**:
   Logs raw packets in pcap format for Wireshark analysis.

e) **Console Logging (-A console)**:
   Displays alerts in real-time on terminal.

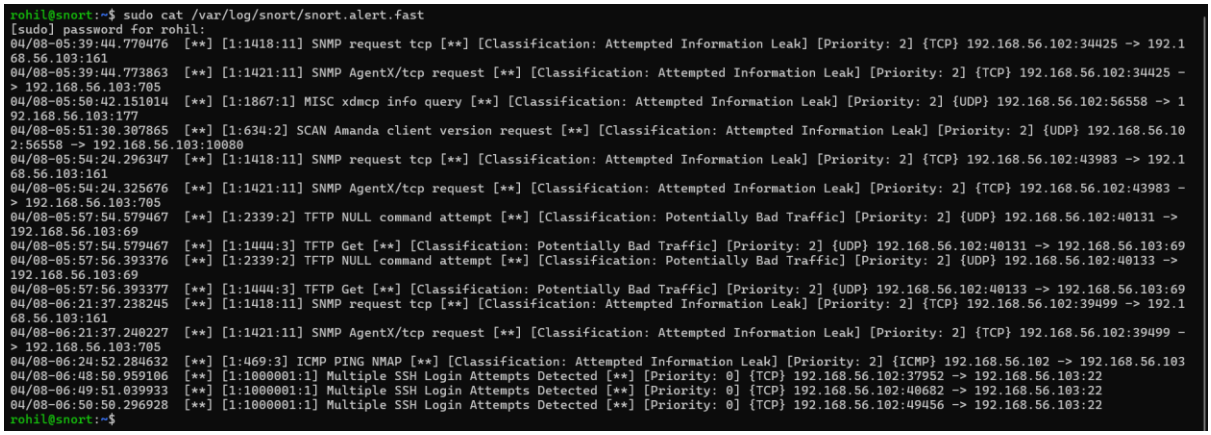
f) **UNIX Socket Alerting**:
   Sends alerts via UNIX sockets for external tools.

g) **Syslog Output** (optional):
   Forwards alerts to system log for centralized logging.

For the convenience, The Fast Alert logs were used as they are in Human-Readable format and can be easily parsed and managed.
(Note: The types of logging can be selected according to infrastructure and security needs. For demo purpose, we have used Fast Alert logs)

The snort logs are stored in location: /var/log/snort

Example of Fast Alert Logs:



(Advanced versions of Snort have more options available for logging like JSON logs etc.)

Integrating Logs with ELK Stack

The ELK Stack is a powerful log management and analysis platform composed of Elasticsearch, Logstash, and Kibana. It helps collect, parse, store, and visually analyze logs in real-time, aiding in threat detection, network and system monitoring.

The ELK Stack can be configured on any host i.e server or desktop.

For this project, the configuration was carried out on Ubuntu 22.04 Desktop which would be a dedicated system for Log Monitoring.

Before configuring, the following pre-requisites of the ELK Stack should be verified:

- A system running Ubuntu 20.04 / 22.04
- A user account with sudo or administrative privileges
- Terminal or command line access
- You have Java version 8 or 11 installed (preferably 8)

First, Elasticsearch was configured:

A. Configuring Elasticsearch:

➢ **Installation**

1) Added the elastic repository:
   wget -qO — https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
2) Once the Elastic GPG key was imported, the **apt-transport-https** package was installed with the following **apt-get** command:
   # sudo apt-get install apt-transport-https
3) Added the Elastic repository to the system's repository list by executing the following command:
   # echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee –a /etc/apt/sources.list.d/elastic-7.x.list

4) Updated the repository using **sudo apt-get update** and installed elasticsearch using command: **sudo apt-get install elasticsearch**.

- **Configuring the elasticsearch after installation:**

1) Opened the elasticsearch configuration file from given path:
   sudo /etc/nano/elasticsearch/elasticsearch.yml

2) Assigned the value of network.host to the value of the device's IP on which ELK Stack was supposed to run. (Consider assigning the static IP address value)
3) Assigned http.port to 9200.
4) The security feature was disabled by assigning **xpack.security.enabled: false** and **xpack.security.enrollment.enabled: false** as it can increase overhead on configuration.
5) As part of the configuration process, the cluster.initial_master_nodes parameter was set to a preferred node name. This step is essential for bootstrapping the cluster and defining the initial master-eligible node during the Elasticsearch setupSave file and exit editor.
6) Executed the command **sudo systemctl start elasticsearch.**
7) Executed the command **sudo systemctl enable elasticsearch** to start elasticsearch on boot.
8) Used command **curl -X "localhost:9200"** to test the configuration.

Hence, elasticsearch was configured.

B. Installation and Configuration of Kibana.

(Note: Always installed Kibana after and before installing Elasticsearch and Logstash respectively)

1) Kibana was installed on the server using the following command: **sudo apt-get install kibana**.
2) Following installation, the Kibana configuration file located at /etc/kibana/kibana.yml was opened using: **sudo nano /etc/kibana/kibana.yml.**
3) In the configuration file, the following parameters were set:

- server.port was set to 5601.
- server.host was assigned the static IP address of the machine.
- elasticsearch.hosts was also set to the same static IP, as Elasticsearch is hosted on the same server.

4) After making these changes, the file was saved and closed.
5) To start Kibana and enable it to run on system boot, the following commands were executed: **sudo systemctl start kibana** and **sudo systemctl enable kibana**
6) Kibana was then accessed through a web browser using the URL: **http://<ip-address-of-machine/localhost>:5601**

C. Installation and Configuration of Logstash

1) Logstash was installed on the server using the following command: **sudo apt-get install logstash.**
2) Once installed, the Logstash service was enabled to start automatically at system boot using**: sudo systemctl enable logstash.**
3) The service was then started with the command: **sudo systemctl start logstash**.

D. Logstash Pipe Configuration

To process and forward logs to Kibana, a Logstash pipeline was configured as part of the project. The pipeline consists of three main components: an input file to ingest logs, a filter file to parse the logs, and an output file to forward the parsed data to Elasticsearch (for Kibana visualization).

1. **Input File** – Defines the source from which logs are ingested.
2. **Filter File** – Specifies the log parsing rules and formatting logic.
3. **Output File** – Sends the processed logs to the appropriate Elasticsearch instance, enabling visualization in Kibana.

To configure the Logstash Pipeline, the given files were created in **/etc/logstash/conf.d**

- Input file: **01-beats-input.conf**
  Purpose: To read logs which would be sent by filebeat from server.
  Content of file:

```
input {
  beats {
    port => 5044
    host => "192.168.56.109"  # Listen on all network interfaces
  }
}
```

- Filter file: **10-snort-filter.conf**
  Purpose: to write grok filter to parse snort fast alert logs into meaningful fields
  Content of file:

```
    filter {
  grok {
    match => {
      "message" => [
        "%{MONTHNUM:month}/%{MONTHNUM:day}-
%{TIME:time}\s+\[\*\*\] \[%{INT:gid}:%{INT:sid}:%{INT:rev}\]
%{DATA:alert_msg} \[\*\*\](?: \[Classification: %{DATA:classification}\])?
\[Priority: %{INT:priority}\] \{%{DATA:protocol}\}
(?:%{IPV6:src_ip}|%{IPV4:src_ip})(?::%{INT:src_port})? ->
(?:%{IPV6:dest_ip}|%{IPV4:dest_ip})(?::%{INT:dest_port})?"
      ]
    }
  }

  mutate {
    add_field => {
      "fixed_year" => "2025"
      "logdate" => "%{fixed_year}-%{month}-%{day} %{time}"
    }
  }
```

```
  date {
    match => ["logdate", "yyyy-MM-dd HH:mm:ss.SSSSSS"]
    target => "@timestamp"
    timezone => "UTC"
  }

  mutate {
    convert => {
      "priority" => "integer"
      "gid" => "integer"
      "sid" => "integer"
      "rev" => "integer"
    }
    remove_field => ["month", "day", "time", "fixed_year", "logdate",
"host", "agent"]
  }
}
```

- Create file: **30-elasticsearch-output.conf**
  Purpose: Send parsed logs to elasticsearch/kibana
  Content of file:

```
  output {
    elasticsearch {
      hosts => ["http://localhost:9200"]
      index => "snort-alerts-%{+YYYY.MM.dd}"
    }
  }
```

Hence, the logstash was configured.

E. Configuration of Filebeat on Ubuntu Server

1) Filebeat was installed on the system using the following command: **sudo nano /etc/filebeat/filebeat.yml**.
2) Inside the file, the input and output to the logstash was defined.
   Content of file:

```
filebeat.inputs:
 - type: log
   enabled: true
   paths:
     - /var/log/snort/snort.alert.fast


output.logstash:
  hosts: ["192.168.56.109:5044"]
```

3) After installation, the Filebeat service was started and configured to run on system startup using the following commands: **sudo systemctl start filebeat** and **sudo systemctl enable filebeat**

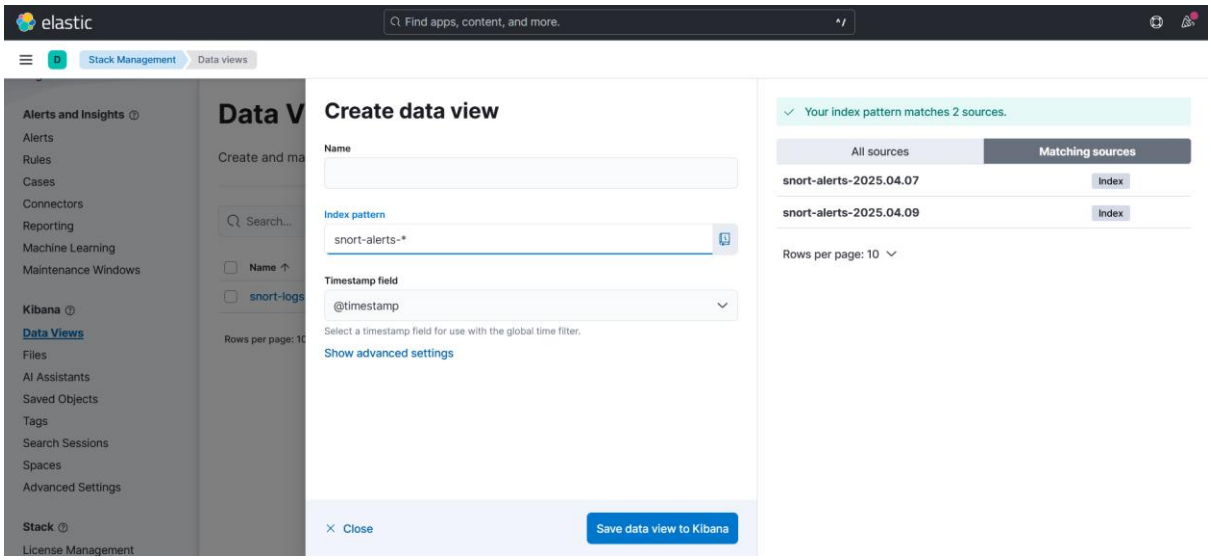
Thus, the filebeat was configured to ship the logs to logstash.

Viewing Logs in ELK Stack

To view the logs in ELK Stack, the Index Pattern must be createad first. In the output of logstash file, I have already created an index pattern consisting of date:
index => "snort-alerts-%{+YYYY.MM.dd}"

A. Accessing Kibana and configuring Index Pattern

1. Kibana was accessed through a web browser using the address: **http://<ip-address>:5601**.
2. From the Kibana homepage, the following navigation path was used: Management → Stack Management → Data Views
3. For initiating data view creation, Management → Stack Management → Data Views.
4. In the **Index pattern** field, the following value was entered: **snort-alerts-***. This pattern matches all indices created by Logstash using the defined output format.
5. The field @timestamp was selected as the default time field for filtering and visualizing time-series data.



B. Viewing Logs in Kibana

After successfully creating a Data View in Kibana based on the Logstash-generated indices, the following steps were performed to view and analyze the logs collected from the Snort Intrusion Detection System:

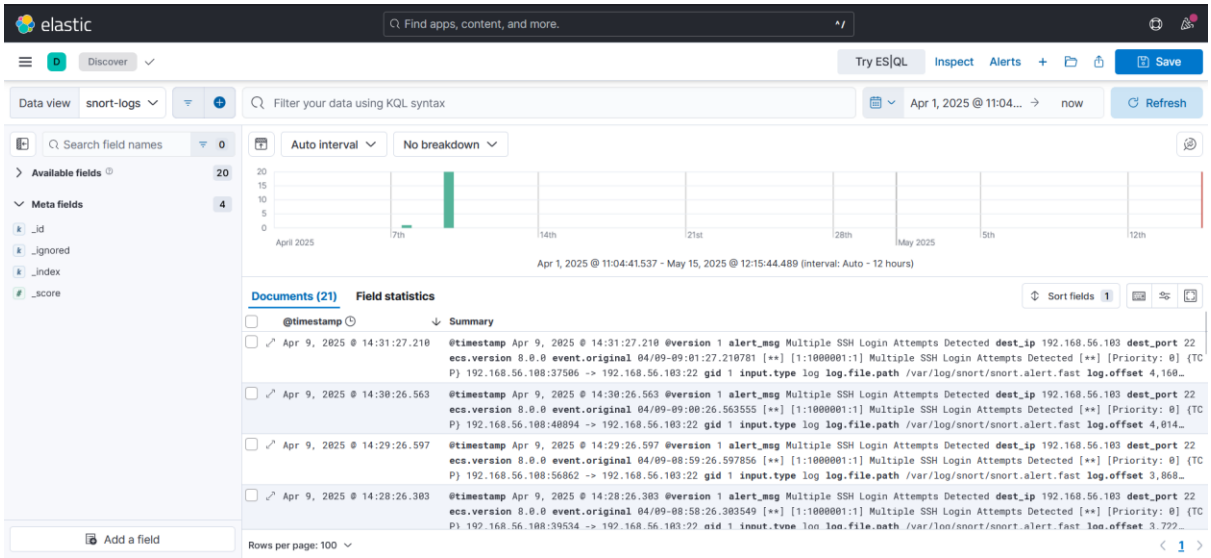
1) In Kibana dashboard, the following path was navigated: **Kibana → Discover**

2) To configure the data view, in the Discover panel, the previously created Data View (snort-alerts-*) was selected from the drop-down list. This enabled Kibana to fetch and display all log entries matching that index pattern.

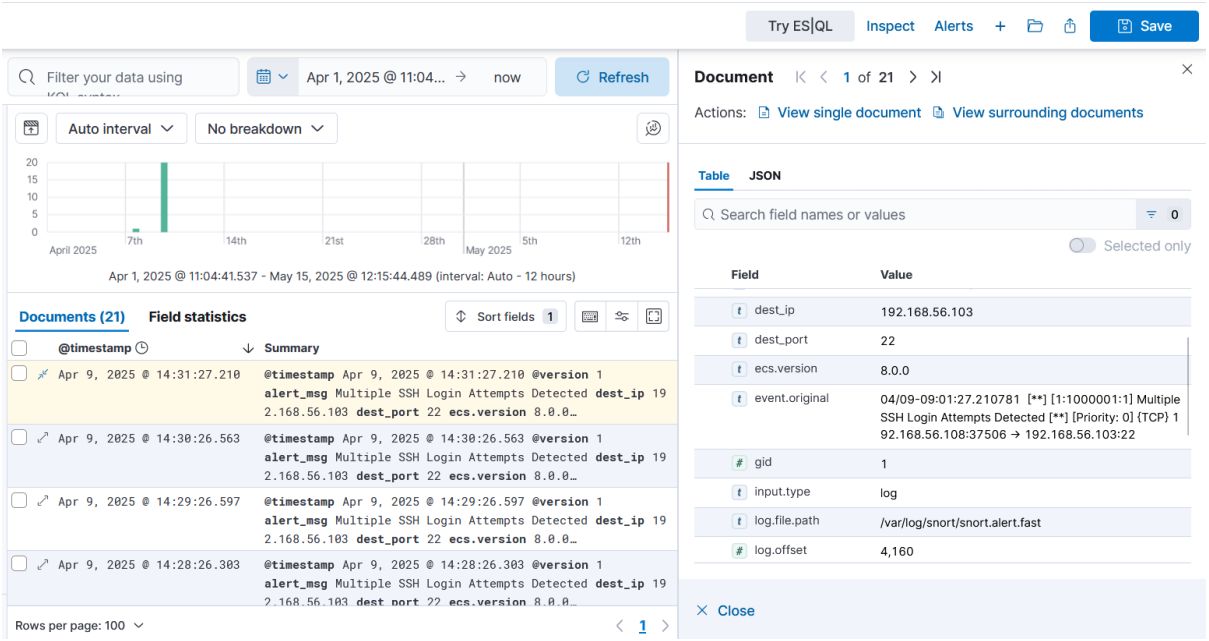
   Thus, the logs were displayed on the Kibana interface. The time stamps can be manually configured to view the logs according to it.

   It is also possible to inspect each individual logs by enlarging it and viewing it's relevant fields like timestamp, source/destination IPs, and snort alert messages.

Viewing Logs in Kibana interface:



Inspecting a single log:



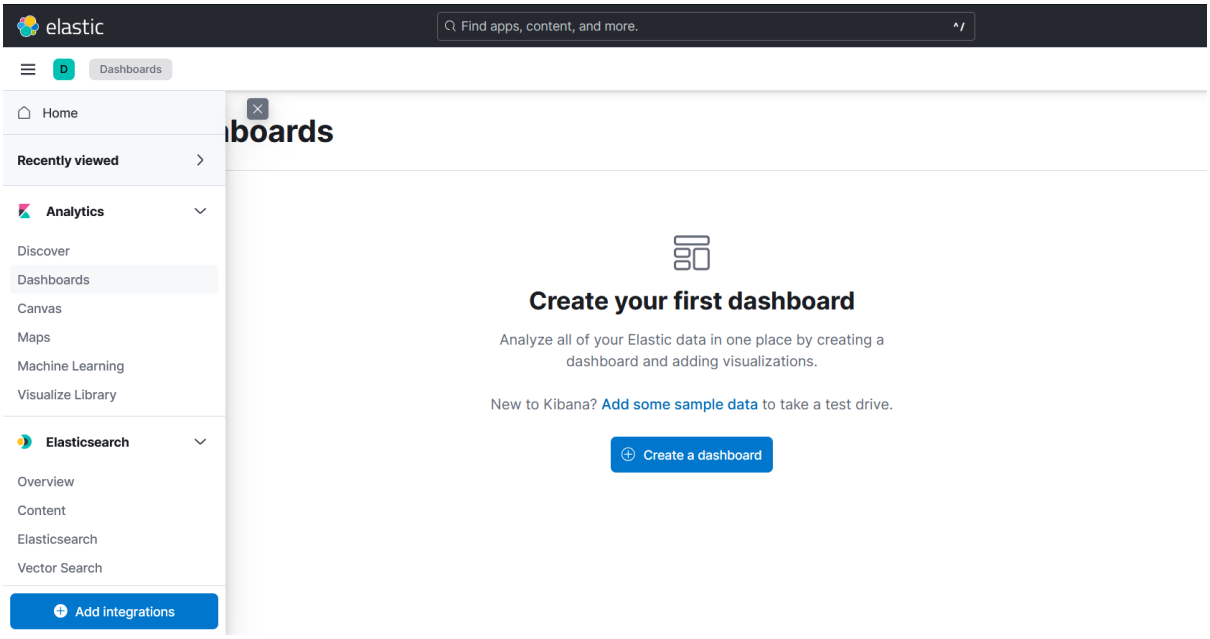
In this log, the different fields can be viewed like:

- Destination IP
- Destination Port
- Event.original: It displays Snort Fast Alert Message. Etc.

C. Creating a Dashboard in Kibana

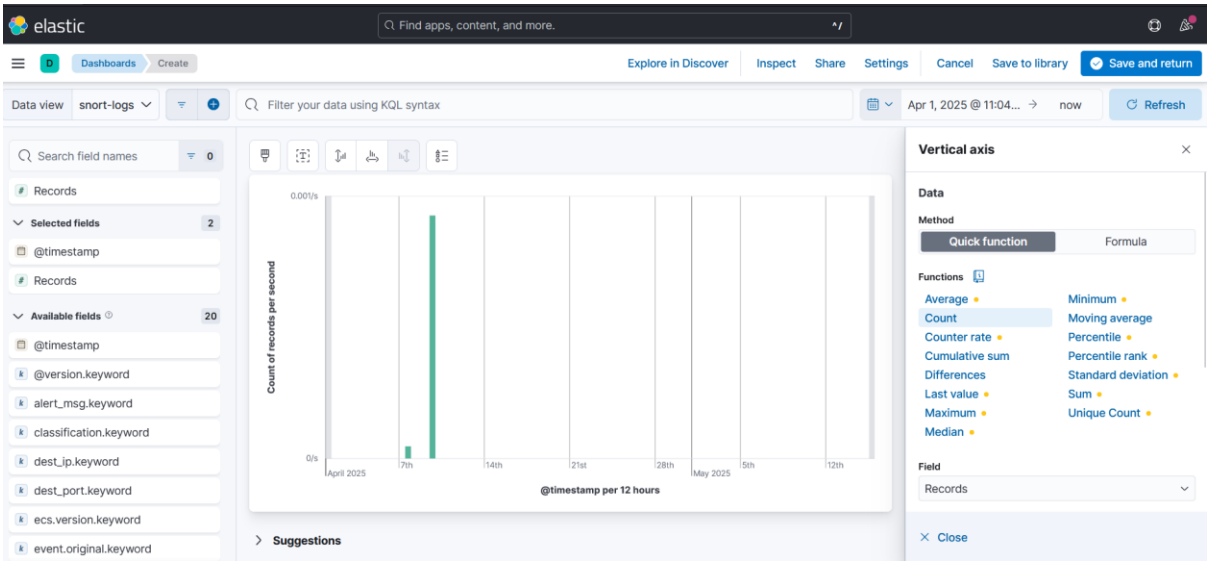
To provide a centralized and visual overview of the Snort alert data collected via the ELK Stack, a dashboard was created in Kibana. The dashboard allows for real-time monitoring and pattern analysis through a series of configurable visual components. The following steps were followed to create and configure the dashboard:

1. In Kibana → Dashboard, the dashboard was created by clicking on Create a Dashboard.

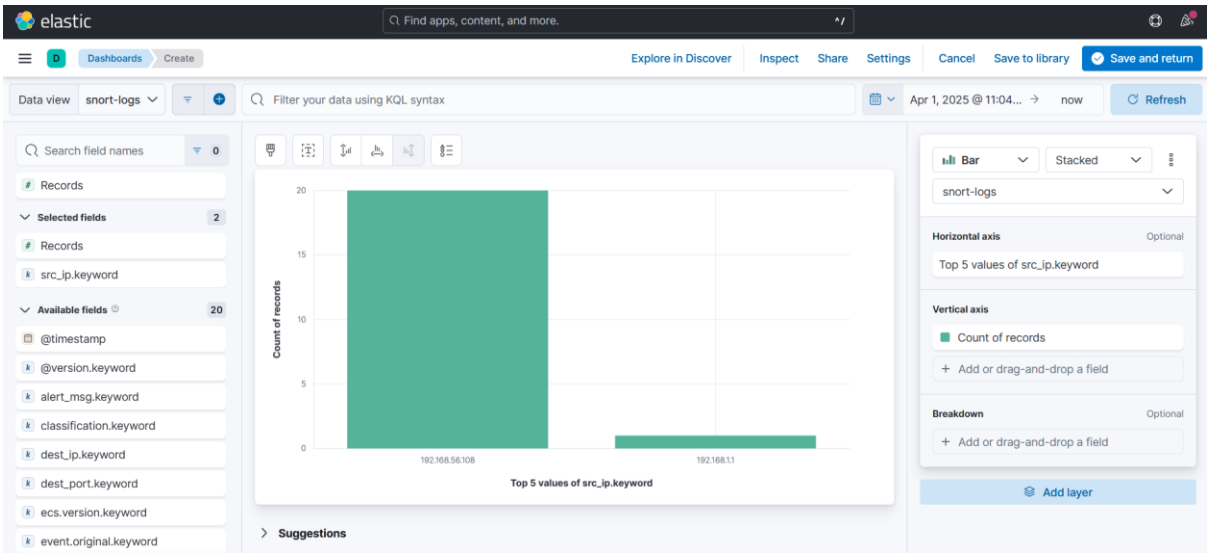


2. The **"Create dashboard"** button was clicked to initiate a new dashboard. A blank dashboard canvas was presented, allowing custom visualizations to be added.
3. The relevant visualizations were added like:
    - Frequency of alerts generated over time
    - Top source IP addresses
    - Top destination IP addresses
    - Most targeted ports
    - Most frequently generated alert messages

For example: Creation of visualization for Frequency of alerts generated over time:

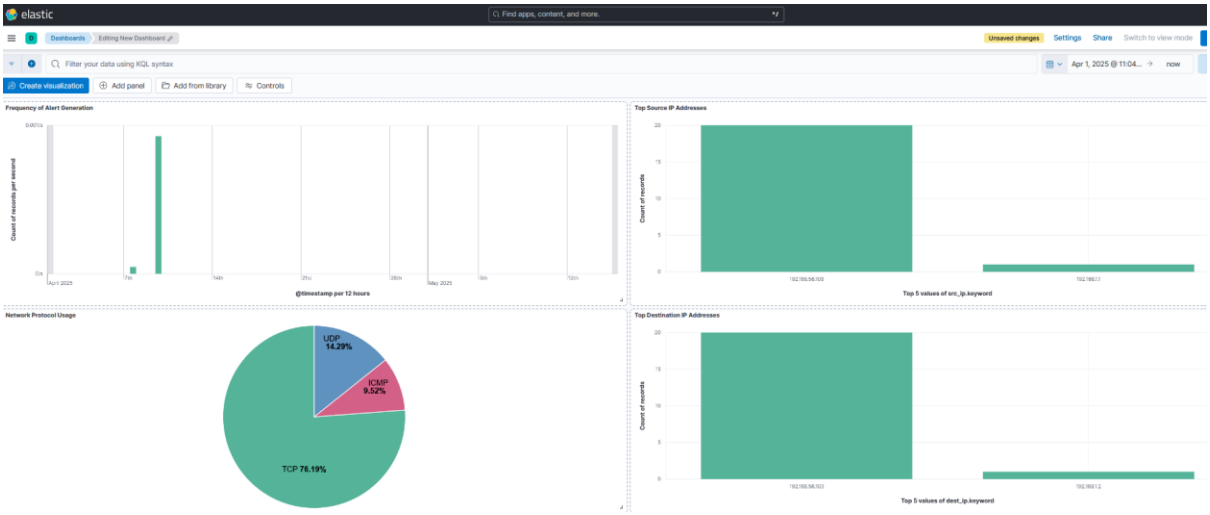


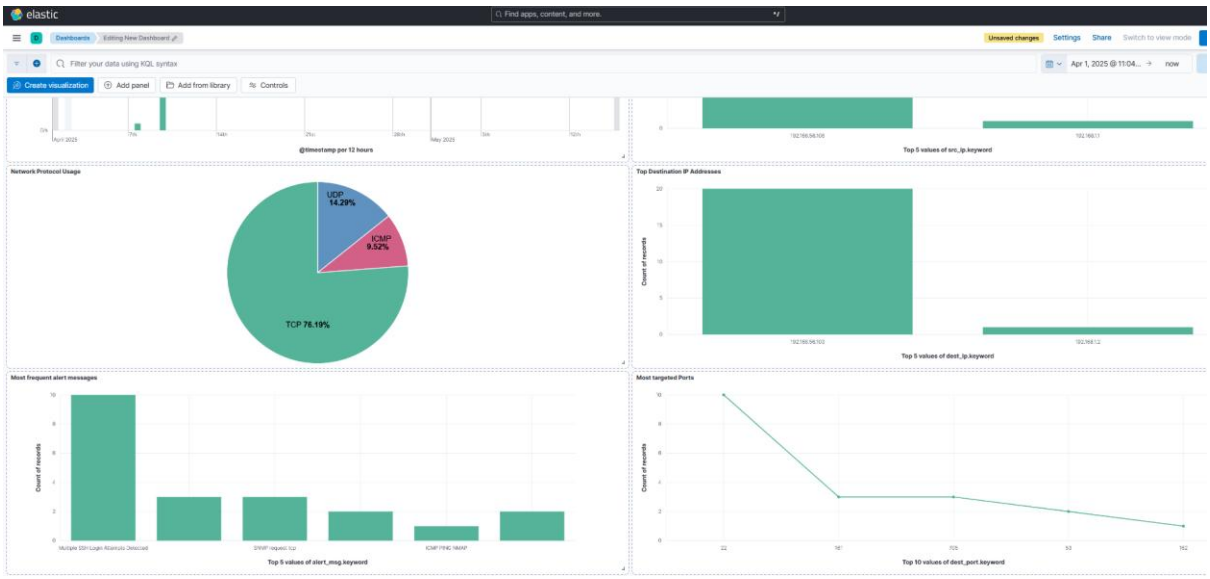
Visualization of Top Source IP Addresses:



Multiple visualizations were created following this process and later compiled into a Kibana dashboard for centralized security monitoring and analysis of Snort alerts.

Full view of Dashboard:




elastic
Network Protocol Usage
UDP 14.29%
ICMP 9.52%
TCP 76.19%
Top Destination IP Addresses
Most frequent alert messages
Most targeted Ports

Challenges Faced and Solutions

Some of the challenges faced in this project were:

1) Failure in installation of ELK Stack.

Solution:

The ELK Stack is version sensitive. So ensure that the prerequisites are satisfied completely. For example: The JDK version should be Java 8 and not higher or lower than it.

2) Snort is unable to detect the malicious activities like Port Scanning or SSH Brute-Force attempt.

Solution:

To ensure proper detection, ensure that IP variables are more explicit rather than general address range i.e. it it is ensured that Attack VM is on 192.168.56.108 then $HOME_NET should be set to 192.168.56.108 rather then any, 192.168.56.0/24 etc.c

Also cross verify the rules i.e. sensitivity levels, threshold, protocols etc.

3) Elasticsearch fails to start

Solution:

One possible reason for Elasticsearch failing to start could be the JVM Heap allocation. The heap allocation should not be implicitly changed by user to ensure that it is dynamically configured during runtime.

Conclusion

This project successfully demonstrated the implementation of a network-based intrusion detection system using **Snort** in conjunction with the **ELK Stack (Elasticsearch, Logstash, and Kibana)** for comprehensive log management, alert generation, and real-time visualization.

The process began with the installation and configuration of **Snort** on an Ubuntu Server 22.04 system, where network variables and preprocessor rules were customized to define the detection scope. A custom alert rule was written to identify brute force login attempts, showcasing Snort's flexibility in rule-based traffic analysis.

Following this, **Logstash** was configured to collect and parse Snort-generated alert logs and forward them to **Elasticsearch**, where they were indexed for querying and analysis. **Filebeat** was also deployed to support log forwarding and ensure a consistent data flow to the stack.

In the final phase, **Kibana** was used to visualize the alert data. Several visualizations were created—highlighting alert frequency, top source and destination IPs, frequently targeted ports, and most common alert messages. These were consolidated into an interactive **dashboard**, providing real-time insights into potential threats and helping in the detection of suspicious activity.

Simulated attacks (e.g., port scanning, brute force SSH attempts) were executed from a Kali Linux virtual machine to validate Snort's detection capabilities. The alerts generated confirmed successful traffic monitoring and rule enforcement.

Overall, this project demonstrated a practical, end-to-end deployment of an IDS and SIEM environment, reinforcing the importance of proactive monitoring in modern cybersecurity infrastructure. The approach outlined is scalable and can be adapted for more complex networks with additional rules, automation, and alert handling mechanisms in production environments.