Task 03: Network Security Monitoring and Analysis

1. Introduction

This document presents the implementation, analysis, and findings of Network Security Monitoring and Analysis. The primary goal of this task is to gain hands-on understanding of both **passive** and **active** network security monitoring techniques using widely adopted, industry-standard tools such as **Wireshark, Snort,** and **Scapy.**

In modern Security Operations Center **(SOC)** environments, continuous monitoring of network traffic is essential for detecting malicious activity, identifying abnormal behavior, and responding to potential threats in a timely manner. This task simulates real-world **SOC** workflows by combining traffic capture and inspection, intrusion **detection,** and **controlled** attack simulation within a lab environment.

Through this task, **passive** network traffic analysis was performed using **Wireshark** to observe **protocol** behavior and identify **suspicious** patterns. **Active** intrusion **detection** was implemented using **Snort** configured in **IDS** mode with custom **detection rules.** Additionally, **Scapy** was used to generate crafted network **packets** to simulate common attack techniques such as SYN scans and port scanning, enabling **validation** of **IDS detections.**

Overall, this exercise bridges theoretical concepts of network security with practical implementation, providing a clear understanding of how network-based attacks manifest at the **packet** level and how defensive tools can be configured to detect and analyze such threats effectively.

2. Objective

The primary objective of this task is to perform comprehensive network security monitoring by combining **passive** traffic analysis, **active** intrusion **detection,** and **controlled** attack simulation using industry-standard security tools. This task focuses on analyzing network traffic using **Wireshark** for **passive** monitoring, configuring **Snort** as an Intrusion **Detection** System **(IDS)** for real-time threat **detection,** and generating test traffic using **Scapy** to validate **detection** mechanisms.

Through this exercise, the task aims to develop a practical understanding of how network-based attacks and **suspicious** activities appear at the **packet** level and how they can be identified using security monitoring tools. Specific emphasis is placed on recognizing common network security threats such as TCP SYN scans, port scanning activities, and **reconnaissance** behavior, which are frequently observed during the initial stages of cyber attacks.

Additionally, this task seeks to strengthen analytical skills by correlating observations from multiple tools, enabling accurate interpretation of network events and improving the ability to respond to potential threats in a Security Operations Center **(SOC)** environment.

3. Environment Setup

The environment for this task was designed to closely resemble a practical network security lab while remaining simple and efficient to manage. A virtualized setup was used to safely perform network monitoring, intrusion detection**,** and attack simulation without impacting any production systems.

3.1 Host Machine

Operating System: Windows
The Windows host system was used to run the virtual machine and manage the overall lab environment.

3.2 Virtual Machine

Operating System: Kali Linux
A Kali Linux virtual machine was used as the primary analysis and testing platform. Kali Linux was chosen due to its extensive support for security tools and pre-configured libraries required for network analysis and penetration testing activities.

3.3 Tools and Technologies Used

The following tools were used during the execution of this task:

- **Wireshark**
  Used for **passive** network traffic capture and analysis. **Wireshark** enabled

inspection of **packets** at various **protocol** layers and identification of traffic patterns and anomalies.

- **Snort**
  Configured as an Intrusion **Detection** System **(IDS)** to **actively** monitor network traffic and generate **alerts** based on predefined and custom **detection rules.**
- **Scapy**
  A Python-based **packet** manipulation library used to generate custom network **packets. Scapy** was utilized to simulate SYN **packets** and port scanning behavior in order to test and validate **Snort IDS rules.**

This environment ensured safe experimentation while providing a realistic platform to understand and analyze network security monitoring workflows commonly used in Security Operations Center **(SOC)** environments.

4. Network Traffic Analysis Using Wireshark

Wireshark was used to perform **passive** network traffic analysis to observe **packet-level** communication, identify **protocol** usage, and detect any potentially **suspicious** patterns without **actively** interacting with the network. This section documents the installation, traffic capture, filtering, and analysis results obtained using **Wireshark.**

4.1 Installation and Verification

Wireshark was installed on the Kali Linux virtual machine using the system package manager. The installation was verified by executing the tshark -v command, which confirmed that the **Wireshark** command-line utilities were correctly installed. Additionally, the **Wireshark** graphical user interface (GUI) was launched successfully, and available network interfaces were visible, confirming that the tool was ready for **packet** capture and analysis.

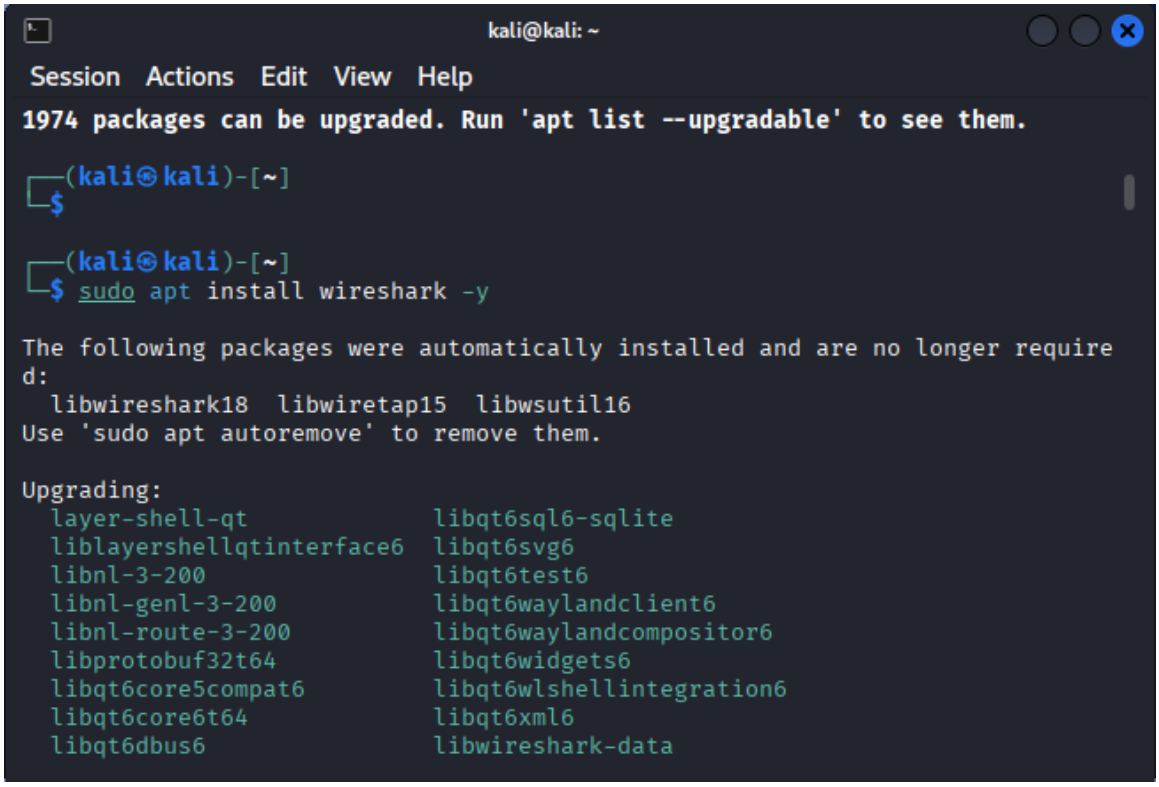

Figure 1: Wireshark installation

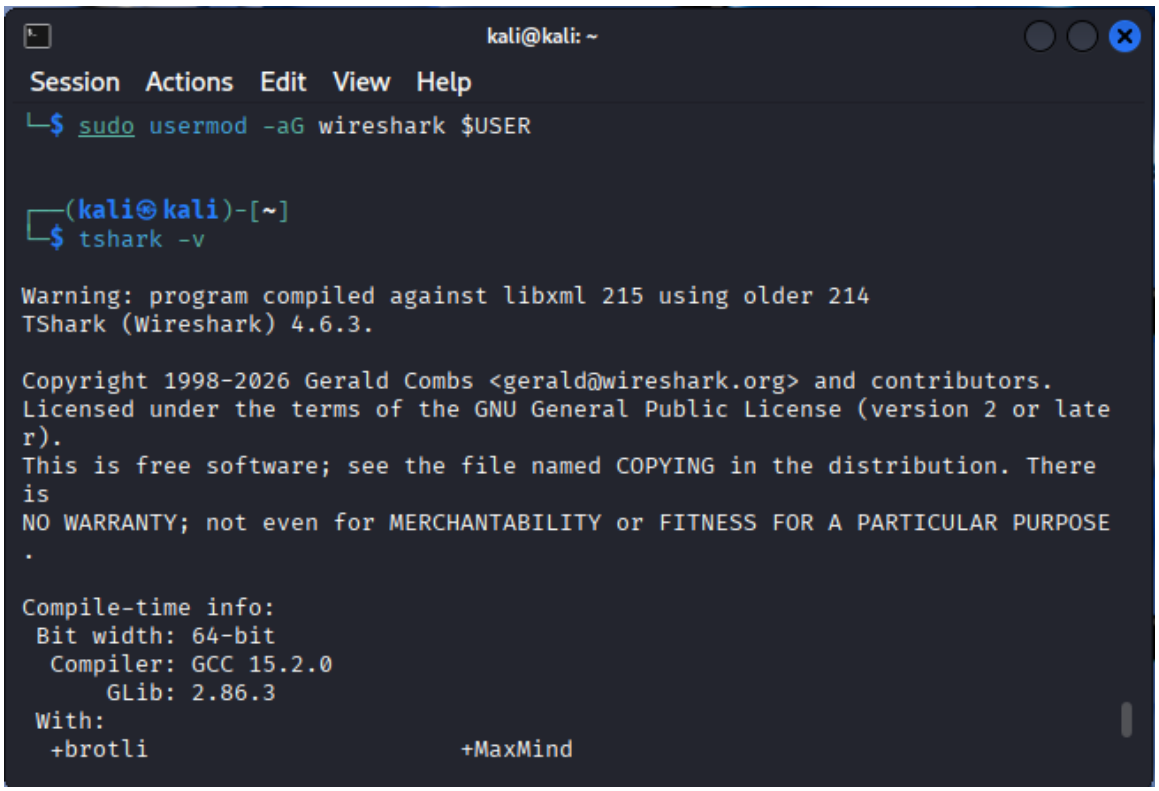

Figure 2: Wireshark installation verification / tshark version output]

4.2 Traffic Capture

Network traffic was captured on the eth0 network interface, which was the **active** interface on the system during the task. The capture was allowed to run for several minutes to collect a representative sample of normal background traffic and user-generated network activity.

The captured traffic included routine communication such as web requests, DNS lookups, and system-level network operations. The **packet** capture file was saved in .pcapng format for further analysis and documentation.

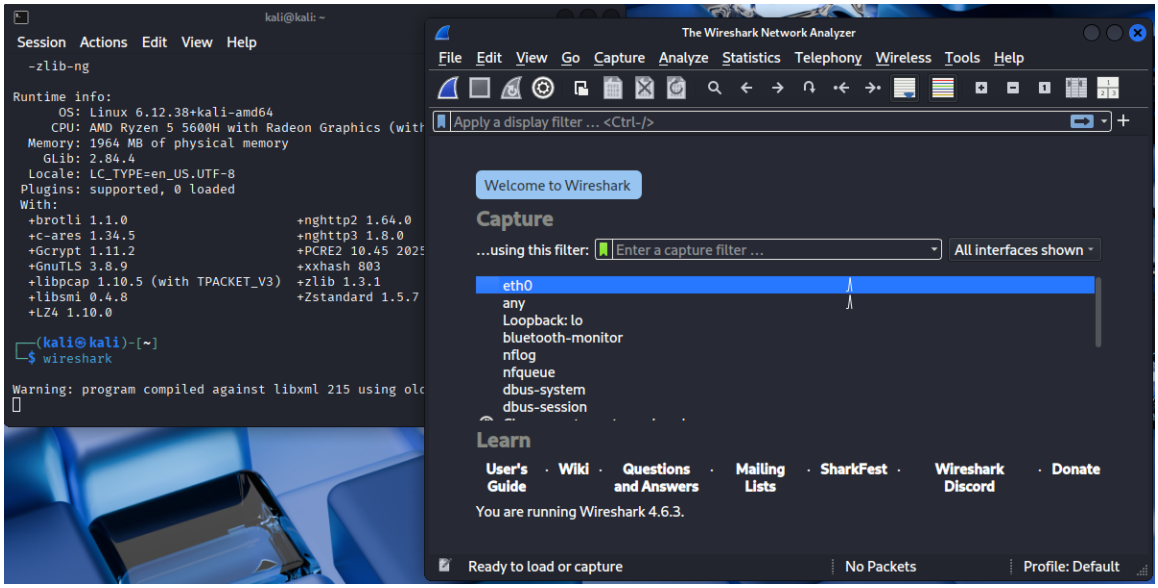

Figure 3: Live network traffic capture on eth0 interface

4.3 Packet Filtering and Protocol Analysis

To analyze the captured traffic effectively, **Wireshark** display filters were applied to isolate **packets** based on **protocol** type. The following **protocol-based** filters were used:

- TCP
- UDP
- ICMP

Analysis of the filtered traffic identified the following **protocols** and their purposes:

- **Transmission Control Protocol (TCP):**
  TCP **packets** constituted a significant portion of the captured traffic. TCP is a connection-oriented **protocol** used for reliable communication and is commonly

associated with services such as HTTP, HTTPS, and SSH. TCP **packet** sequences showed standard connection initiation behavior.

- **User Datagram Protocol (UDP):**
  UDP traffic was observed primarily during DNS communication. UDP is a connectionless **protocol** that enables faster data transmission without session establishment, commonly used for name resolution and other lightweight network services.
- **Internet Control Message Protocol (ICMP):**
  ICMP **packets** were observed during diagnostic communication, such as echo requests and replies. ICMP is used for network diagnostics and error reporting and is commonly generated during connectivity checks.

This **protocol-level** analysis helped establish a baseline understanding of normal network communication on the system.

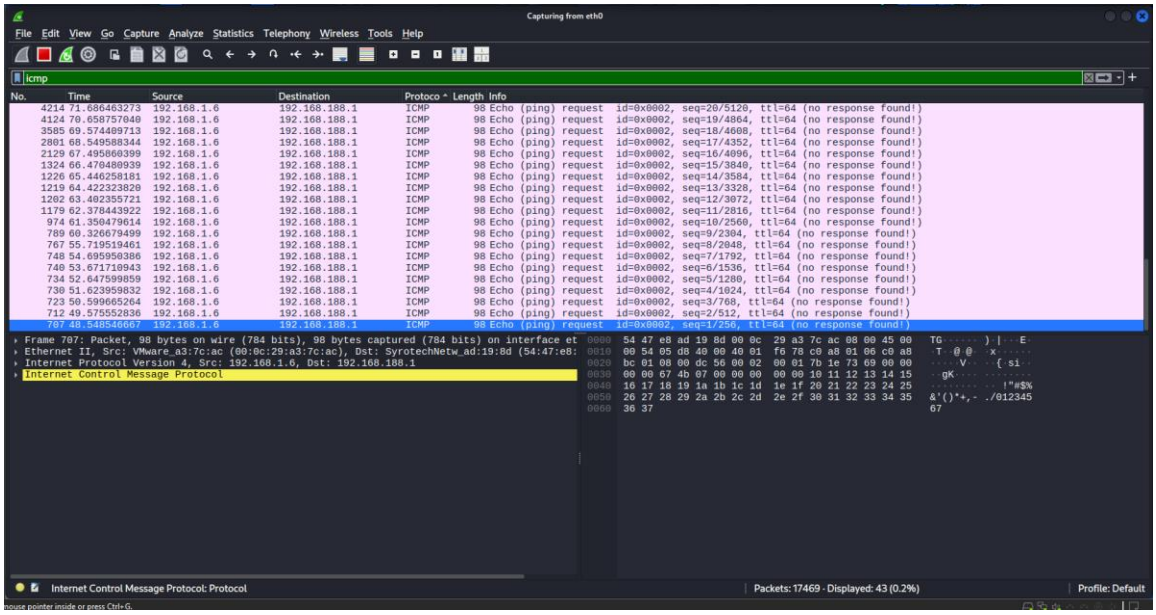

Figure 4: Wireshark ICMP Filter

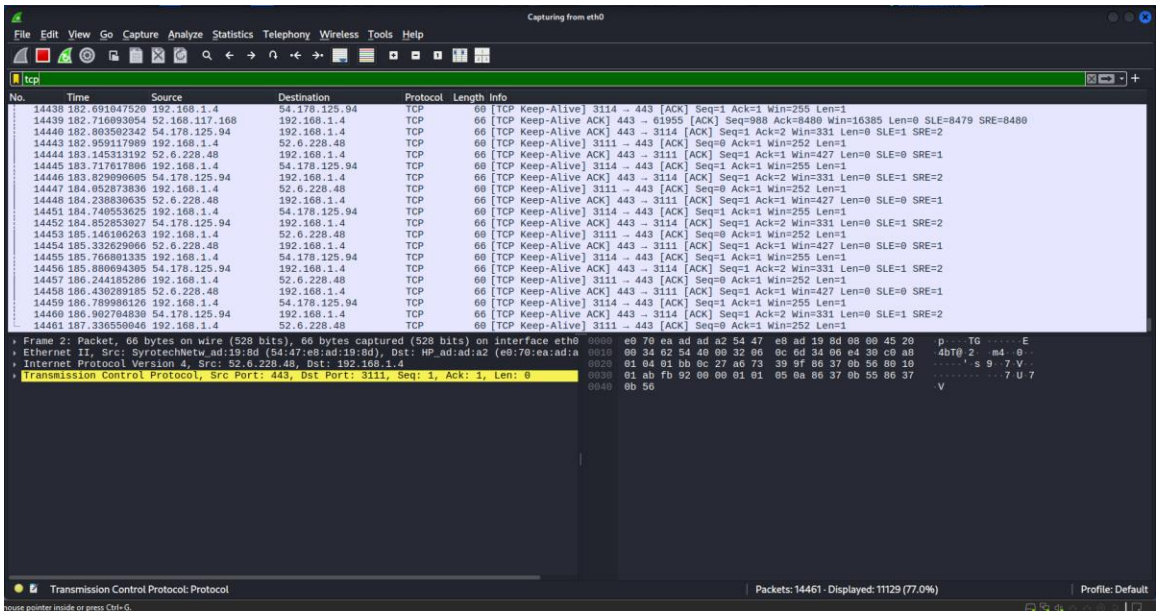

Figure 5: Wireshark TCP Filter

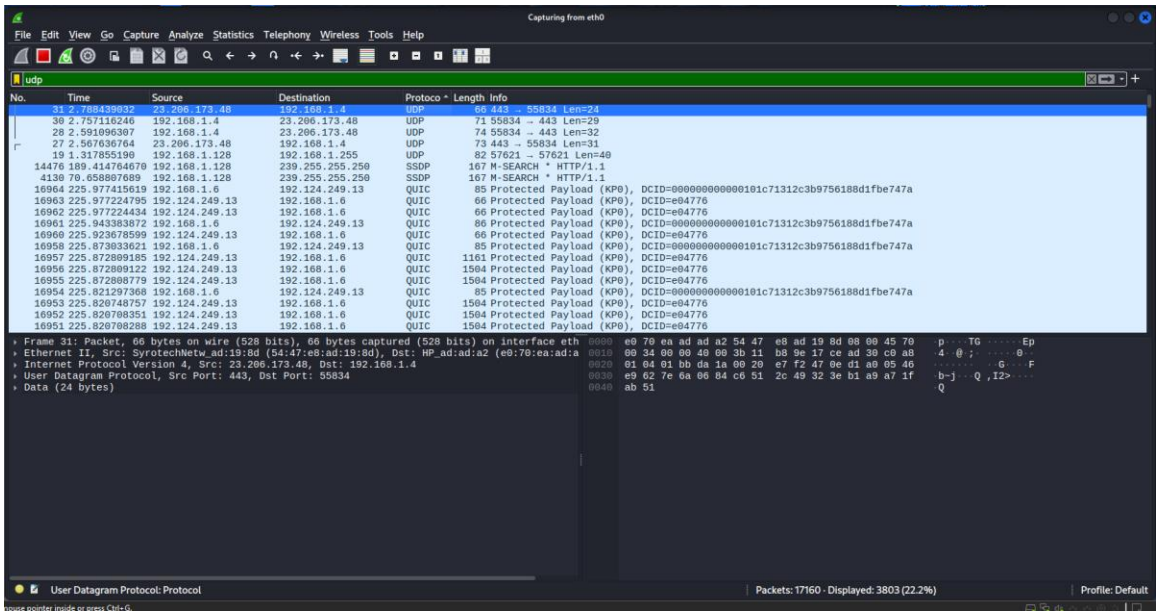

Figure 6: Wireshark UCP Filter

4.4 Identification of Suspicious Traffic Patterns

To identify potential **reconnaissance** behavior, a SYN-only TCP **packet** filter was applied to the captured traffic. This filter isolated TCP **packets** with the SYN flag set, which represent initial connection attempts.

The filtered results showed repeated SYN **packets** originating from the same source, indicating multiple connection initiation attempts. While SYN **packets** are a normal part

of TCP communication, repeated SYN **packets** without corresponding session completion can be indicative of scanning or probing activity.

Wireshark's statistical and visualization features, including I/O graphs, were used to review **packet** distribution over time. These tools assisted in observing the frequency and concentration of SYN **packets** during the capture period without making assumptions beyond the observed data.

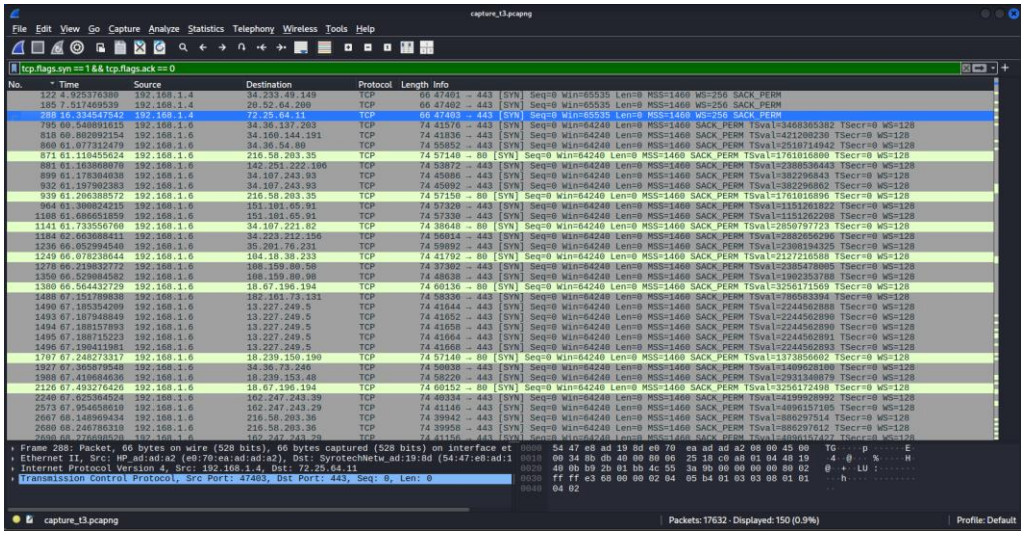

Figure 7: SYN-only packet filter results

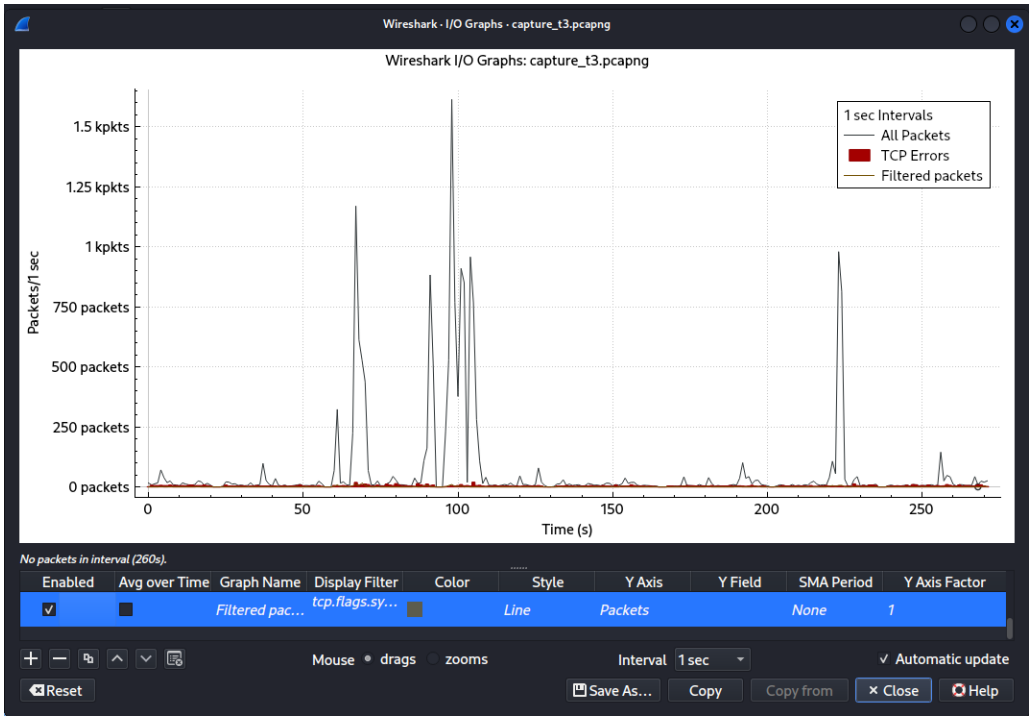

Figure 8: Wireshark I/O graph view

Summary of Wireshark Analysis

The **Wireshark** analysis successfully demonstrated how **passive** network traffic monitoring can be used to identify **protocol** usage, establish baseline communication patterns, and detect early indicators of **suspicious** behavior such as repeated TCP SYN **packets.** These findings satisfied the requirements for **protocol** analysis and **suspicious** pattern identification as defined in the task objectives.

5. Intrusion Detection Using Snort

Snort was used in this task to perform **active** network security monitoring by inspecting live network traffic and generating **alerts** based on defined **detection rules.** Unlike **Wireshark,** which provides **passive** visibility into captured traffic, **Snort** operates as an Intrusion **Detection** System **(IDS)** capable of analyzing **packets** in real time and raising **alerts** when traffic matches **suspicious** patterns.

In this phase of the task, **Snort** was primarily used to observe background network behavior on the system and identify **suspicious** activity through **alerting,** without **actively** blocking or modifying traffic.

5.1 Snort Installation and Configuration

Snort was installed on the Kali Linux system and configured to operate in **IDS** mode, allowing continuous inspection of network traffic without enforcing **packet** drops or blocking actions. The version used for this task was **Snort** 3 **(Snort++),** which uses Lua-based configuration files instead of the traditional **snort.conf** file used in earlier versions. The system was configured using a bridged network adapter, resulting in the same internal IP address (192.168.1.4) being observed for both the Kali Linux virtual machine and the Windows host. This setup allowed **Snort** to monitor real background network traffic generated by normal system activity.

Custom local **rules** were included in the **Snort** configuration to ensure **detection** of specific TCP-based behaviors. The configuration was validated using **Snort's** built-in test mode, confirming that all required modules, **rule** files, and dependencies were successfully loaded without errors.

Running **Snort** in **IDS** mode enabled monitoring of the **active** network interface and ensured that **alerts** would be generated whenever observed traffic matched the defined **detection rules.**

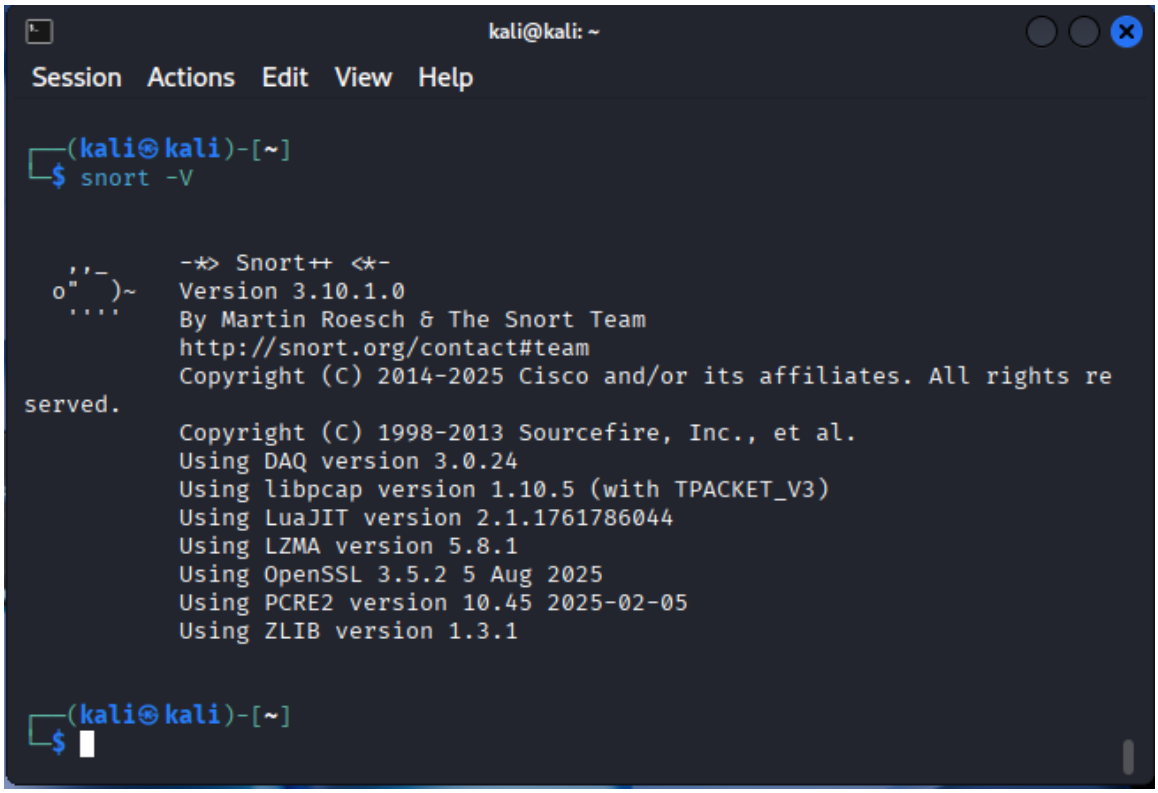

Figure 9: Snort configuration validation showing successful rule loading

5.2 Custom IDS Rules

To detect **suspicious** network behavior, two custom Intrusion **Detection** System **(IDS) rules** were implemented. These **rules** were designed to identify **reconnaissance-style** activities, which commonly occur during the early stages of network attacks.

The following **rules** were added:

- **SYN Packet Detection Rule**

  This rule detects TCP **packets** with the SYN flag set, indicating attempts to initiate new TCP connections. While SYN **packets** are a normal part of TCP communication, repeated SYN **packets** can indicate probing or abnormal connection behavior.

- **Port Scan Detection Rule**

  This rule detects TCP SYN **packets** targeting multiple destination ports within a defined range. Such behavior is characteristic of port scanning, where a system attempts to identify open ports or **active** services.

Both **rules** were stored in the local **rules** file and explicitly included in the **Snort** configuration to ensure they were **active** during traffic inspection.

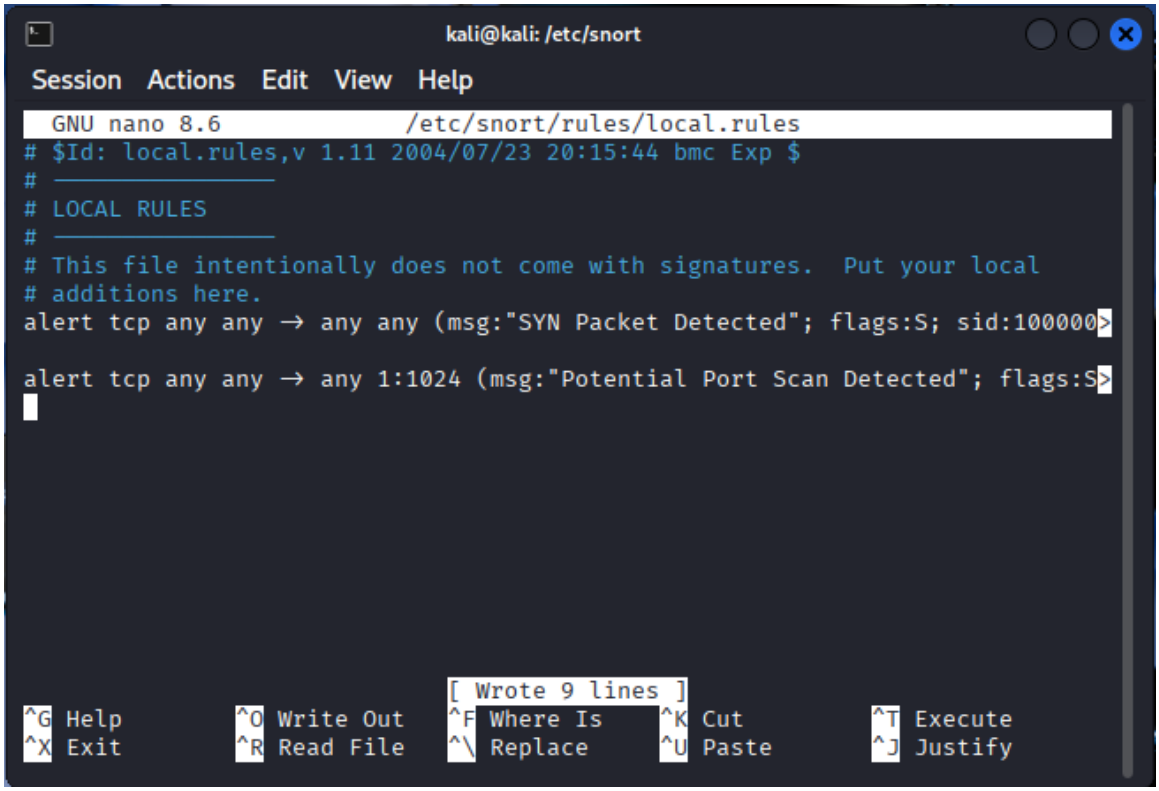

Figure 10: local.rules file showing custom IDS rules

5.3 Snort Alert Analysis

After configuration, **Snort** was executed in **IDS** mode and allowed to monitor live network traffic on the **active** interface. During this period, **Snort** analyzed normal background network traffic generated by the system.

Snort generated multiple **alerts** corresponding to the configured **detection rules.** The **alerts** confirmed **detection** of:

- TCP SYN **packets,** indicating repeated connection initiation attempts
- Potential port scanning activity, identified through multiple SYN **packets** targeting different destination ports
- The **alert** output showed repeated activity originating from the internal IP address 192.168.1.4, which is consistent with the bridged network setup used during the task. **Alerts** included detailed information such as source and destination IP addresses, destination ports (including commonly used service ports), **protocol** type, and the **rule** identifiers responsible for the **detection.**

These **alerts** demonstrate that even during observation of background network behavior, **Snort** was able to identify patterns consistent with **reconnaissance-style** traffic. This

confirmed that the **IDS** was functioning correctly and that the custom **rules** were effective in detecting **suspicious** TCP connection behavior.

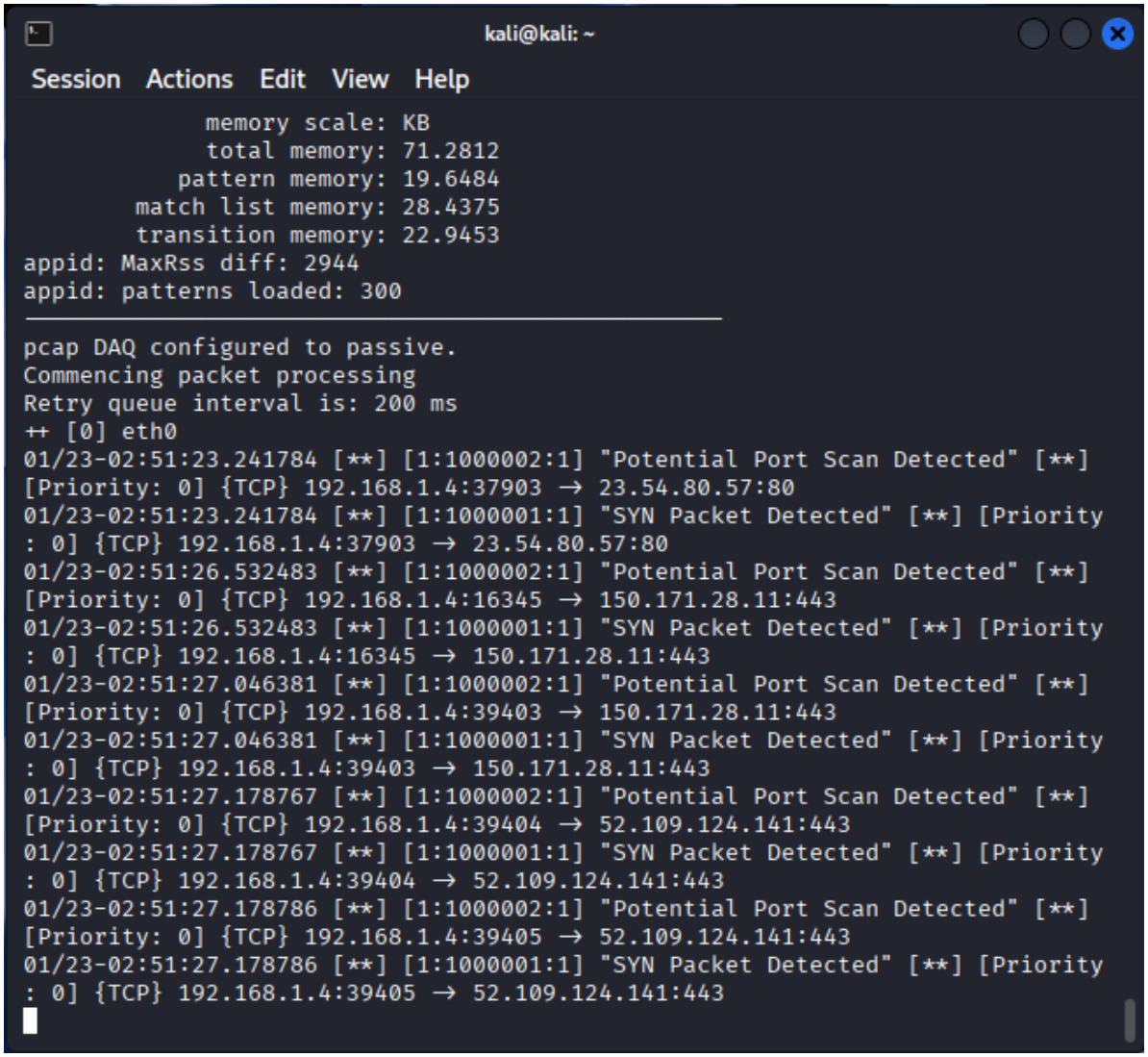

Figure 11: Snort alerts displayed in console during IDS operation

Summary of Snort Detection

The **Snort-based** intrusion **detection** phase successfully demonstrated real-time **alerting** capabilities during live network monitoring. By detecting repeated TCP SYN **packets** and potential port scanning behavior in background traffic, **Snort** fulfilled the task requirement of **active** intrusion **detection** and validated the effectiveness of the configured **IDS rules.**

6. Packet Generation Using Scapy

Scapy was used in this task to perform controlled network packet generation with the objective of simulating reconnaissance-style network activity and validating the effectiveness of the configured Snort IDS rules. While Wireshark and Snort are primarily used for traffic observation and detection, Scapy enables direct construction and transmission of custom packets at the protocol level, making it suitable for intentional testing of intrusion detection mechanisms in a controlled lab environment.
In this phase, Scapy was used specifically for intentional detection testing, as required by the task guidelines.

6.1 Scapy Script Overview

Scapy was used in this task to perform **controlled** network **packet** generation with the objective of simulating **reconnaissance-style** network activity and validating the effectiveness of the configured **Snort IDS rules.** While **Wireshark** and **Snort** are primarily used for traffic observation and **detection, Scapy** enables direct construction and transmission of custom **packets** at the **protocol** level, making it suitable for intentional testing of intrusion **detection** mechanisms in a **controlled** lab environment.
In this phase, **Scapy** was used specifically for intentional **detection** testing, as required by the task guidelines.

6.1 Scapy Script Overview

A Python script was developed using the **Scapy** library to generate TCP SYN **packets** targeting multiple destination ports. This traffic pattern closely resembles a basic TCP SYN scan, which is a common **reconnaissance** technique used to identify open ports and running services on a target system.
The script was designed with the following objectives:

1. Generate TCP SYN **packets** at the **packet** level
2. Target multiple destination ports sequentially
3. Operate in a **controlled** and safe manner by targeting the local system
4. Trigger the custom **Snort IDS rules** configured for SYN **packet detection** and port scan **detection**
5. To maintain realistic traffic behavior and avoid excessive **packet** flooding, a short delay was introduced between successive **packet** transmissions. This ensured

that the generated traffic was clearly observable and could be accurately analyzed by the **IDS.**

6.2 Scapy Script Code

The following Python script was used to generate test traffic for this task. The script is structured, well-documented, and includes comments explaining the logic behind **packet** generation and port selection. The script sends TCP SYN **packets** to commonly used service ports in order to simulate **reconnaissance** behavior.
623091ef-085e-4e1c-8957-0df3065…

The following is the python script: **test_packets.py:**

```
"""
test_packets.py
This script uses Scapy to generate TCP SYN packets in order to
simulate basic network attacks such as SYN scanning / port scanning.
The generated traffic is intended to trigger Snort IDS rules
configured to detect:
1. SYN packets
2. Potential port scan activity
"""
from scapy.all import IP, TCP, send
from typing import List
import time

def generate_syn_packets(target_ip: str, ports: List[int]) -> None:
    """
    Sends TCP SYN packets to multiple destination ports.

    Args:
        target_ip (str): Target IP address.
        ports (List[int]): List of destination ports to scan.
    """
    print("[*] Starting SYN packet generation...")

    for port in ports:
        packet = IP(dst=target_ip) / TCP(dport=port, flags="S")
        send(packet, verbose=False)
        print(f"[+] Sent SYN packet to port {port}")
        time.sleep(1)

    print("[*] SYN packet generation completed.")
```

```
if __name__ == "__main__":
    # Target is localhost for safe testing
    TARGET_IP = "127.0.0.1"

    # Common ports used to simulate a port scan
    TARGET_PORTS = [21, 22, 25, 80, 443]

    generate_syn_packets(TARGET_IP, TARGET_PORTS)
```

The script was executed with elevated privileges, as raw **packet** transmission is required to craft and send custom TCP **packets** using **Scapy.**

6.3 Scapy Execution and Validation

Upon execution of the **Scapy** script, TCP SYN **packets** were successfully transmitted to multiple destination ports, including ports commonly **associated** with network services (e.g., 21, 22, 25, 80, and 443). The script output confirmed the successful transmission of each SYN **packet.**

At the same time, **Snort** was running in **IDS** mode on the monitored interface. As shown in the **alert** output, **Snort** immediately generated **alerts** corresponding to the configured **detection rules,** including:

- “SYN Packet Detected”
- “Potential Port Scan Detected”

The **alerts** clearly show that the SYN **packets** generated by the **Scapy** script directly triggered the **IDS rules. Alert** details included the source IP address, destination IP address, destination port, **protocol** type, and **rule** identifiers, confirming correct **rule** execution and **detection.**

This behavior demonstrates that the **Scapy-generated** traffic accurately simulated **reconnaissance-style** activity and that the **Snort IDS** was functioning as intended during live traffic inspection.

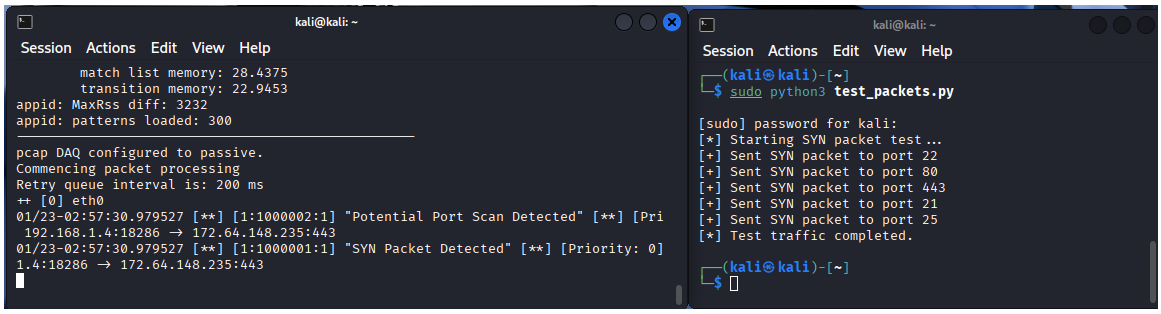

Figure 12: Scapy script execution output + Snort Alerts

Summary of Packet Generation Phase

The **packet** generation phase successfully demonstrated the use of **Scapy** for intentional attack simulation and **IDS rule validation.** By generating TCP SYN-based scanning traffic and confirming its **detection** by **Snort,** this phase fulfilled the task requirement of using **Scapy** to test intrusion **detection rule** effectiveness in a **controlled** lab environment.

7. Correlation and Results Analysis

This section presents a consolidated analysis of results obtained from **Wireshark, Snort,** and **Scapy,** demonstrating how **passive** traffic observation, **active** intrusion **detection,** and intentional traffic generation complement each other in identifying **suspicious** network behavior.

7.1 Correlation Between Wireshark and Snort Observations

Wireshark was used to perform **passive** traffic analysis and establish visibility into **protocol-level** communication on the monitored network interface. During analysis, repeated TCP SYN **packets** were observed, indicating multiple connection initiation attempts. These observations highlighted potential **reconnaissance-style** behavior but did not generate **alerts** on their own, as **Wireshark** functions strictly as a monitoring and analysis tool.

Snort, operating in **IDS** mode on the same network interface, **actively** inspected live traffic and generated **alerts** when similar TCP SYN patterns were detected. The **Snort alerts** corresponding to “SYN **Packet** Detected” and “Potential Port Scan Detected” aligned directly with the SYN **packet** activity observed in **Wireshark.** This confirmed that the **suspicious** patterns identified during **passive** analysis were actionable and detectable through **rule-based** intrusion **detection.**

7.2 Correlation Between Scapy-Generated Traffic and Snort Alerts

Scapy was used to intentionally generate TCP SYN **packets** targeting multiple destination ports in a **controlled** lab environment. The execution of the **Scapy** script resulted in the immediate transmission of SYN **packets** to commonly used service ports. While the **Scapy** script was running, **Snort** generated real-time **alerts** corresponding to the configured **IDS rules.** The **alerts** clearly showed that the **Scapy-generated** SYN **packets** directly triggered both the SYN **packet detection rule** and the port scan **detection rule.** This provided clear **validation** that the **IDS rules** were functioning as intended and were capable of detecting **reconnaissance-style** traffic under **controlled** conditions.

7.3 End-to-End Validation of Detection Workflow

The combined results from all three tools demonstrate a complete end-to-end network security monitoring workflow:

- **Wireshark** provided **passive** visibility into network traffic and helped identify **suspicious** patterns at the **packet** level.
- **Snort actively** detected and **alerted** on those patterns in real time using custom **IDS rules.**
- **Scapy** enabled intentional traffic generation to validate and confirm **IDS** effectiveness.
- The correlation of observations across these tools confirms that the detected **alerts** were not false positives but were directly linked to observable **packet** behavior. This layered approach illustrates how **passive** analysis, **active detection,** and **controlled** testing together enhance the reliability and accuracy of network security monitoring.

8. Challenges Faced and Solutions

During the execution of this task, several technical and conceptual challenges were encountered. These challenges are common when working with real-world network security tools and provided valuable learning opportunities. Each issue was identified, analyzed, and resolved as part of the task workflow.

8.1 Snort 3 Configuration Complexity

One of the primary challenges faced was configuring **Snort** 3 **(Snort++),** which uses Lua-based configuration files instead of the traditional **snort.conf** format. Initially, difficulties were encountered while loading custom **IDS rules** due to incorrect **rule** inclusion syntax.

This issue was resolved by identifying the correct method for including local **rule** files within the **Snort** 3 configuration structure and validating the configuration using **Snort's** built-in test mode. Successful **validation** confirmed that the **rules** were loaded correctly and enabled effective intrusion **detection.**

8.2 Raw Socket Permission Issues with Scapy

While executing the **Scapy** script to generate custom TCP SYN **packets,** a permission-related error was encountered. This occurred because **Scapy** requires access to raw **sockets** in order to craft and transmit custom network **packets.**

The issue was resolved by executing the **Scapy** script with elevated privileges using sudo. Once proper permissions were granted, the script executed successfully and generated the intended network traffic.

8.3 Interpreting Protocol-Level Network Behavior

Another challenge involved accurately interpreting **protocol-level** behavior, particularly distinguishing between normal TCP connection attempts and potentially **suspicious** patterns such as repeated SYN **packets.** This required careful application of **Wireshark** filters and repeated examination of captured traffic.

Through iterative filtering and analysis, a clearer understanding of TCP communication patterns and **reconnaissance** indicators was achieved. This improved the ability to identify meaningful anomalies and correlate them with **IDS alerts.**

Summary of Challenges and Resolutions

All encountered challenges were successfully resolved through configuration **validation,** permission management, and systematic traffic analysis. Overcoming these challenges contributed to a deeper practical understanding of network security monitoring tools and workflows.

9. Key Learnings and Insights

This task provided valuable practical insight into network security monitoring by combining **passive** analysis, **active detection,** and **controlled** traffic generation. The

hands-on nature of the task reinforced several important concepts relevant to real-world security operations.

One of the key learnings was the clear distinction between **passive** and **active** network monitoring. Using **Wireshark** demonstrated how **passive** tools allow visibility into network communication without influencing traffic flow, making them suitable for baseline analysis and pattern observation. In contrast, **Snort** illustrated how **active** monitoring enables real-time inspection and **alerting** based on predefined **detection rules.**

The task also provided a deeper understanding of how Intrusion **Detection** Systems **(IDS)** identify **reconnaissance** and scanning behavior. By analyzing TCP SYN **packets** and port scan patterns, it became clear how seemingly normal connection attempts can become **suspicious** when observed in repetition or across multiple destination ports. This reinforced the importance of **rule-based detection** in identifying early-stage attack activity.

Another significant learning outcome was gaining practical experience with **packet-level** traffic analysis. Working directly with **protocols** such as TCP, UDP, and ICMP improved understanding of how network communication operates at a low level and how attacks manifest within **packet** structures rather than at the application layer alone.

Finally, the task highlighted the importance of correlating multiple security tools to achieve accurate and reliable threat **detection. Wireshark, Snort,** and **Scapy** each served a distinct role, and correlating their outputs helped confirm findings, reduce false assumptions, and validate **detection** mechanisms. This layered approach reflects real-world **SOC** practices, where multiple data sources are analyzed together to form actionable conclusions.

10. Sources Consulted

- **Wireshark Official Documentation** - https://www.wireshark.org/docs/
- **Snort User Manual** - https://docs.snort.org/
- **Scapy Official Documentation** - https://scapy.readthedocs.io/
- **Linux Manual Pages** - https://man7.org/linux/man-pages/

11. Conclusion

This task successfully demonstrated the practical application of network security monitoring and intrusion **detection** techniques using industry-standard tools. By combining **Wireshark** for **passive** traffic analysis, **Snort** for **active** intrusion **detection,** and **Scapy** for **controlled** traffic generation, the task provided a comprehensive view of how **suspicious** network behavior can be observed, detected, and validated.

Through **packet-level** analysis and **rule-based detection,** common **reconnaissance** patterns such as repeated TCP SYN **packets** and port scanning behavior were identified and analyzed. The results confirmed the effectiveness of using a layered approach, where **passive** monitoring supports visibility, **active detection** provides real-time **alerts,** and **controlled** testing validates **detection** mechanisms.

Overall, this task strengthened practical understanding of network security workflows and demonstrated how multiple tools can be used together to achieve accurate and reliable threat **detection** in a **controlled** environment.