Title: Network Security Monitoring and Analysis

1. Introduction

This task focuses on capturing network packets using Wireshark and analyzing the captured traffic for protocols such as TCP, UDP, and ICMP. The analysis specifically involves identifying repeated SYN packets, SYN scan behavior, and abnormal traffic spikes. Then setup an Intrusion Detection System (IDS) using Snort. The IDS is configured with a basic rule which includes these two rules:

1. Detection of TCP SYN packets
2. Detection of port scanning activity

Finally , Develop a Python script using Scapy to generate test traffic that intentionally triggers the configured Snort IDS rules.

2. Tools Required

- Operating system: Kali Linux
- Python 3
- Scapy library
- Snort
- Wireshark

3. Wireshark Traffic Analysis

Setup

1. Set the Wireshark interface into a loopback
2. Use nmap scan like -sT, -sU, -sS
3. Which sends tcp, udp and icmp packets

Traffic Analysis

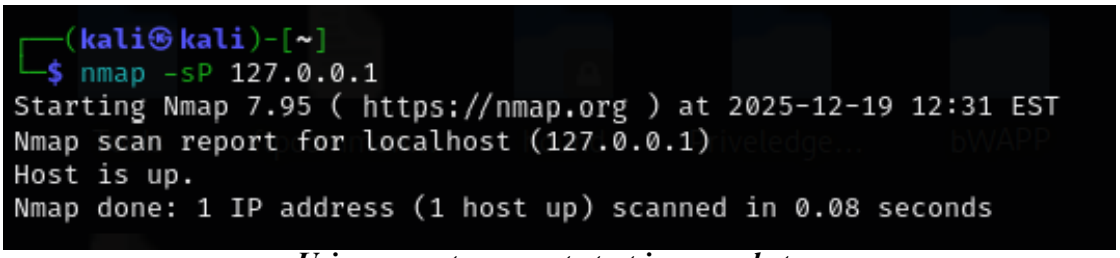
In Wireshark navigate to the Statistics → Protocol Hierarchy protocol and the packets count along with the percentages

Protocol	Packets
TCP	204
UDP	22
ICMP	129

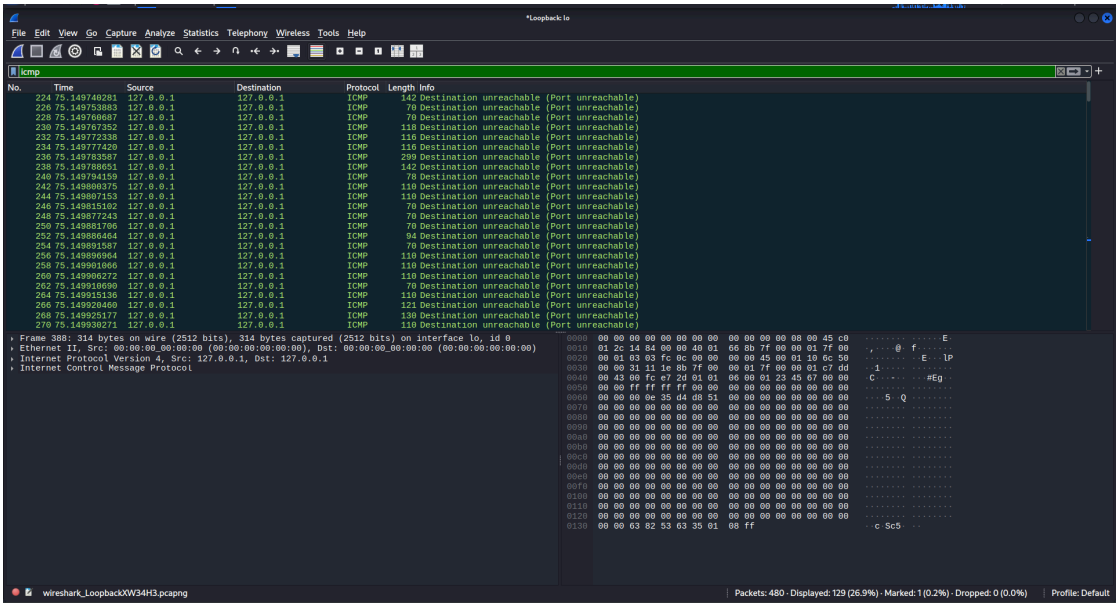
The captured packets are saved in a file, named capture.pcap for analysis

4. Analysis Result

1. The analysis result of ICMP protocol in the captured packets



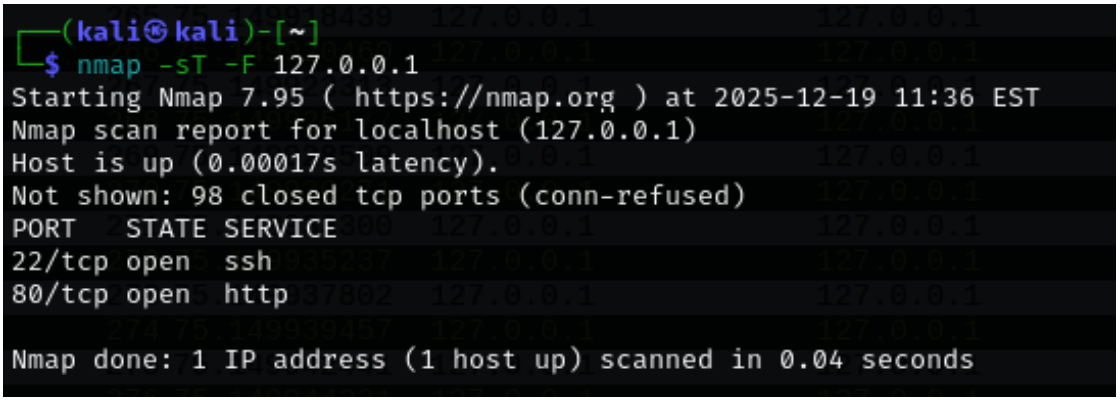
Using nmap to generate test icmp packets



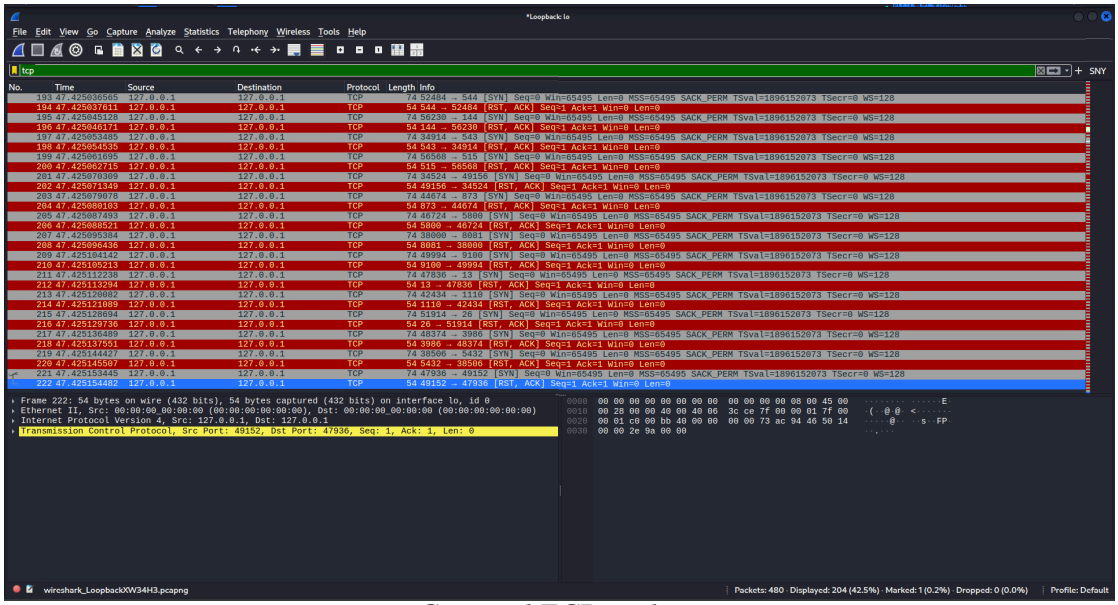
Captured ICMP packets

The above result shows the ICMP packets which was captured by wireshark , the icmp traffic is generated by the nmap for testing purpose

2. The analysis result of TCP protocol in the captured packets



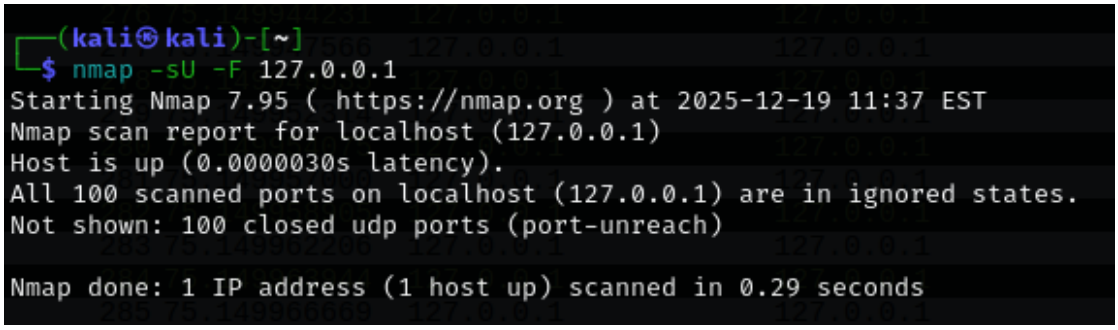
Using nmap to generate test TCP packets



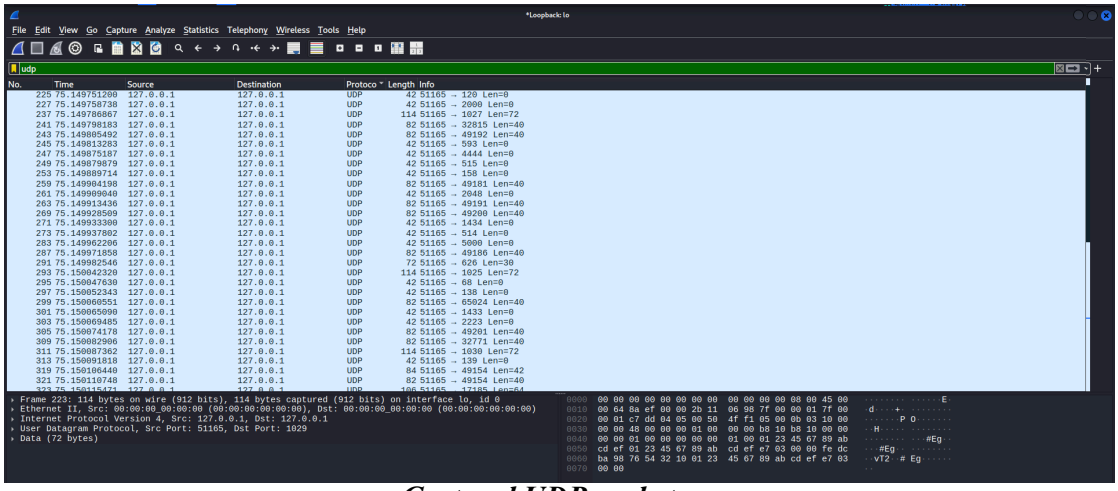
Captured TCP packets

The above result shows the TCP packets which was captured by wireshark , the TCP traffic is generated by the nmap for testing purpose, Where it show the abnormal spikes like a repeated SYN , SYN-ACK and RST-ACK

3. The analysis result of UDP protocol in the captured packets



Using nmap to generate test UDP packets



Captured UDP packets

The above result shows the UDP packets which was captured by wireshark , the UDP traffic is generated by the nmap for testing purpose.

5. Snort

1. Installation

- sudo apt install snort

2. Configurations

- **File path : /etc/snort/snort.lua**
- The configuration file is **snort.lua**

3. Setting up Rules

- **File path : /etc/snort/rules/local.rules**
- **Adding the two given rules :**
  a. alert tcp any any -> any any (msg:"SYN Packet Detected"; flags:S; sid:1000001;)
  b. alert tcp any any -> any any (msg:"SYN Packet Detected"; flags:S; sid:1000001;)

```
┌──(kali㉿kali)-[/unishare/CTF/Py-script]
└─$ cat /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ------------------
# LOCAL RULES
# ------------------
# Rule 1 - detect any SYN packet
alert tcp any any → any any (msg:"SYN Packet Detected"; flags:S; sid:1000001; rev:1;)

# Rule 2 - simple potential port-scan detection on low ports
alert tcp any any → any 1:1024 (msg:"Potential Port Scan"; flags:S; sid:1000002; rev:1;)

# This file intentionally does not come with signatures.  Put your local
# additions here.
```

Snort rules setup

6. Running snort

Start the Snort (IDS) :

To start the Snort : **sudo snort -i lo -c /etc/snort/snort.lua -R/etc/snort/rules/local.rules -A alert_fast**

- -i = To select Interface
- -c = To specify config file
- -R = To specify the rules file
- -A = To specify to the alert output

Snort Started

The snort has started and waiting for the detection

7. Scapy

Developed a Scapy script in python that sends SYN packets to various ports on the Snort-monitored interface which is on **loopback** so both rules fire.

Python Code:

```
#!/usr/bin/env python3
from scapy.all import IP, TCP, send

def main():
    target_ip = input("Enter target IP (Snort-monitored host): ").strip()
    if not target_ip:
        print("[-] No target IP provided, exiting.")
        return

    ports = [22, 80, 443, 8080, 4444]
    src_ip = "127.0.0.1"
    src_port = 44444

    print("Starting SYN packet generation ...")
    for dport in ports:
        ip_layer = IP(src=src_ip, dst=target_ip)
        tcp_layer = TCP(sport=src_port, dport=dport, flags="S", seq=100)
        packet = ip_layer / tcp_layer
        print(f"Sending SYN from {src_ip}:{src_port} to {target_ip}:{dport}")
        send(packet, verbose=0)
    print("Finished sending test packets.")

if __name__ == "__main__":
    main()
```

Script Structure

1. Ask User for setting up the target
2. Send SYN packets to the target ip

```
┌──(kali㉿kali)-[/unishare/CTF/Py-script]
└─$ sudo python3 test_packets.py
Enter target IP (Snort-monitored host): 127.0.0.1
[+] Starting SYN packet generation ...
[+] Sending SYN from 127.0.0.1:44444 to 127.0.0.1:22
[+] Sending SYN from 127.0.0.1:44444 to 127.0.0.1:80
[+] Sending SYN from 127.0.0.1:44444 to 127.0.0.1:443
[+] Sending SYN from 127.0.0.1:44444 to 127.0.0.1:8080
[+] Sending SYN from 127.0.0.1:44444 to 127.0.0.1:4444
[+] Finished sending test packets.
```

Script Output

8. Snort output

The Snot detected the test traffic send by the Scapy script which ensure that the script is sending the SNY traffic. By this we can verify our rules are working fine by detecting the SYN traffic.

```
--------------------------------------------------
fast pattern groups
               to_server: 1
               to_client: 1
--------------------------------------------------
search engine (ac_bnfa)
                instances: 2
                 patterns: 438
          pattern chars: 2602
             num states: 1832
        num match states: 392
            memory scale: KB
            total memory: 71.2812
          pattern memory: 19.6484
       match list memory: 28.4375
       transition memory: 22.9453
appid: MaxRss diff: 3072
appid: patterns loaded: 300
--------------------------------------------------
pcap DAQ configured to passive.
Commencing packet processing
Retry queue interval is: 200 ms
++ [0] lo
12/19-13:42:26.359901 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:26.359901 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:26.389987 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:26.389987 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:26.416451 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:26.416451 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:26.456348 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:8080
12/19-13:42:26.489051 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:4444
12/19-13:42:37.124777 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:37.124777 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:37.148240 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:37.148240 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:37.171303 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:37.171303 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:37.203338 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:8080
12/19-13:42:37.257333 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:4444
12/19-13:42:43.713391 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:43.713391 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:22
12/19-13:42:43.735817 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:43.735817 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:80
12/19-13:42:43.772611 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:43.772611 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:443
12/19-13:42:43.807172 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:8080
12/19-13:42:43.838316 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:20 → 127.0.0.1:4444
12/19-13:48:55.376082 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:22
12/19-13:48:55.376082 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:22
12/19-13:48:55.396558 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:80
12/19-13:48:55.396558 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:80
12/19-13:48:55.424568 [**] [1:1000002:1] "Potential Port Scan" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:443
12/19-13:48:55.424568 [**] [1:1000001:1] "SYN Packet Detected" [**] [Priority: 0] {TCP} 127.0.0.1:44444 → 127.0.0.1:443
```

12. Key Learnings

- Learned how to capture and analyze network traffic using Wireshark, applying display filters to focus on the required packets for better analysis.
- Gained knowledge about Snort and how to configure it to detect specific traffic by defining custom rules in the local.rules file.
- Developed a Python script using Scapy to generate test traffic for the Snort IDS, successfully triggering alerts that match the two specified rules.

13. Conclusion

This task provided hands-on experience across three essential layers of network security: traffic observation, traffic control, and intrusion detection systems. By scripting test packets with Scapy, defining and enforcing Snort rules, and then running the Scapy script, it became clear how an IDS detects or blocks requests based on those rules. These practical exercises reinforced theoretical networking concepts and showed how automation and proper configuration significantly improve both offensive testing and defensive hardening in a lab environment and in real-world networks.

14. References

- Wireshark - https://www.wireshark.org/
- Wireshark traffic analysis - https://medium.com/@jcm3/wireshark-traffic-analysis-part-1-tryhackme-walkthrough-a9bec11d94d9
- https://www.snort.org/
- Snort 3 Documentation - https://docs.snort.org/start/help
- Scapy - https://scapy.readthedocs.io/en/latest/usage.html