Name:Sahil Dabekar **Inter Id:339**

Proof of Concept (POC): Network IDS

Introduction

This document provides a Proof of Concept (POC) for a lightweight Network Intrusion Detection System (IDS). The IDS is designed to monitor network traffic (live or from PCAP files) and raise alerts for ICMP pings, TCP connection attempts, common scan patterns, and suspicious behaviors.

Objective

The primary objective of this POC is to build a simple IDS that demonstrates the detection of the following activities:

1. ICMP ping (echo request/reply)
2. TCP connection attempts (SYNs, half-open connections)
3. Common scan patterns (SYN/NULL/FIN scans, repeated connection attempts across many ports)
4. Suspicious behaviors (repeated ICMP floods, high-rate SYNs to multiple ports)

Methodology

The IDS leverages the Scapy library in Python to capture and analyze network packets. The detection logic includes identifying ICMP packets, tracking TCP flags for connection attempts, and detecting repetitive patterns that indicate scanning or flooding behavior.

Deliverables

1. Python code for the IDS.
2. Demonstration against at least two PCAPs (normal traffic and malicious traffic).
3. Documentation explaining the detection logic, false positive considerations, and next-step ideas.
4. Unit tests for core detection functions.

Code:

```
from scapy.all import *

from collections import defaultdict

import time


# Track counts

icmp_count = 0

syn_count = defaultdict(int)

scan_attempts = defaultdict(set)

start_time = time.time()


# Thresholds

ICMP_FLOOD_THRESHOLD = 50

SYN_FLOOD_THRESHOLD = 100
```

```
def detect_packet(pkt):
    global icmp_count

    # ICMP Ping Detection
    if pkt.haslayer(ICMP):
        if pkt[ICMP].type == 8:  # Echo Request
            print(f"[ALERT] ICMP Ping Request from {pkt[IP].src} to {pkt[IP].dst}")
        elif pkt[ICMP].type == 0:  # Echo Reply
            print(f"[INFO] ICMP Ping Reply from {pkt[IP].src} to {pkt[IP].dst}")
        icmp_count += 1
        if icmp_count > ICMP_FLOOD_THRESHOLD:
            print(f"[ALERT] Possible ICMP Flood detected from {pkt[IP].src}")

    # TCP SYN Detection
    if pkt.haslayer(TCP):
        tcp_flags = pkt[TCP].flags
        if tcp_flags == "S":  # SYN flag
            syn_count[pkt[IP].src] += 1
            print(f"[ALERT] TCP SYN attempt from {pkt[IP].src} to {pkt[IP].dst}:{pkt[TCP].dport}")
```

```
            # Check for SYN Flood
            if syn_count[pkt[IP].src] > SYN_FLOOD_THRESHOLD: print(f"[ALERT]
               Possible SYN Flood from {pkt[IP].src}")

            # Record ports for scan detection
            scan_attempts[pkt[IP].src].add(pkt[TCP].dport)
            if len(scan_attempts[pkt[IP].src]) > 20:
               print(f"[ALERT] Port scan suspected from {pkt[IP].src}")

        # Detect FIN/NULL scans
        if tcp_flags == "F" or tcp_flags == "":
            print(f"[ALERT] Possible FIN/NULL scan from {pkt[IP].src}")

def run_ids(pcap_file=None):
    if pcap_file:
        print(f"[*] Reading packets from {pcap_file}...")
        sniff(offline=pcap_file, prn=detect_packet, store=0)
    else:
        print("[*] Sniffing live traffic (press Ctrl+C to stop)...")
        sniff(prn=detect_packet, store=0)
```

```
if __name__ == "__main__":
    # Example: run live sniff or with PCAP file
    # run_ids("test_traffic.pcap")
    run_ids()
```

Next Steps

Future improvements may include:

- Adding anomaly detection using machine learning.
- Extending protocol coverage (e.g., UDP, HTTP).
- Reducing false positives by implementing threshold-based detection.
- Integration with alerting systems such as email or Slack notifications.