Network IDS

- ICMP ping detection

- TCP SYN connection attempts and half-open connection detection

- SYN scan detection

- Basic alerting

- Running on PCAP files

- Simple unit tests

1. Project Structure

network\_ids/

├── ids.py # Main IDS detection code

├── test\_ids.py # Unit tests

├── sample\_normal.pcap # Sample normal traffic PCAP (you provide)

├── sample\_scan.pcap # Sample scan traffic PCAP (you provide)

└── README.md # Basic documentation

2. `ids.py` — Main IDS Code

python

from scapy.all import rdpcap, IP, ICMP, TCP

from collections import defaultdict

import time

class NetworkIDS:

def \_\_init\_\_(self, syn\_scan\_threshold=10, icmp\_flood\_threshold=20, syn\_flood\_threshold=20):

self.syn\_scan\_threshold = syn\_scan\_threshold

self.icmp\_flood\_threshold = icmp\_flood\_threshold

self.syn\_flood\_threshold = syn\_flood\_threshold

def detect\_icmp\_pings(self, packets):

icmp\_counts = defaultdict(int)

alerts = []

for pkt in packets:

if ICMP in pkt and pkt[ICMP].type in [8, 0]: # Echo request or reply

src = pkt[IP].src

icmp\_counts[src] += 1

alerts.append(f"ICMP Ping detected from {src} to {pkt[IP].dst}, Type: {pkt[ICMP].type}")

# Detect ICMP flood

for src, count in icmp\_counts.items():

if count > self.icmp\_flood\_threshold:

alerts.append(f"ICMP Flood detected from {src}, Count: {count}")

return alerts

def detect\_tcp\_syn(self, packets):

syn\_packets = defaultdict(int)

syn\_ack\_packets = defaultdict(int)

alerts = []

for pkt in packets:

if TCP in pkt and IP in pkt:

flags = pkt[TCP].flags

src = pkt[IP].src

dst = pkt[IP].dst

dport = pkt[TCP].dport

if flags == 'S': # SYN

syn\_packets[(src, dst, dport)] += 1

elif flags == 'SA': # SYN-ACK

syn\_ack\_packets[(dst, src, dport)] += 1

for conn, syn\_count in syn\_packets.items():

syn\_ack\_count = syn\_ack\_packets.get(conn, 0)

half\_open = syn\_count - syn\_ack\_count

if half\_open > 0:

alerts.append(f"Half-open TCP connection detected from {conn[0]} to {conn[1]}:{conn[2]}, Count: {half\_open}")

return alerts

def detect\_syn\_scan(self, packets):

syn\_counts = defaultdict(set)

alerts = []

for pkt in packets:

if TCP in pkt and IP in pkt and pkt[TCP].flags == 'S':

src = pkt[IP].src

dport = pkt[TCP].dport

syn\_counts[src].add(dport)

for src, ports in syn\_counts.items():

if len(ports) > self.syn\_scan\_threshold:

alerts.append(f"Possible SYN scan detected from {src}, scanned ports: {len(ports)}")

return alerts

def analyze\_pcap(self, pcap\_file):

print(f"Analyzing PCAP file: {pcap\_file}")

packets = rdpcap(pcap\_file)

alerts = []

alerts.extend(self.detect\_icmp\_pings(packets))

alerts.extend(self.detect\_tcp\_syn(packets))

alerts.extend(self.detect\_syn\_scan(packets))

if alerts:

print("Alerts:")

for alert in alerts:

print(f" - {alert}")

else:

print("No suspicious activity detected.")

return alerts

if \_\_name\_\_ == "\_\_main\_\_":

import argparse

parser = argparse.ArgumentParser(description="Lightweight Network IDS")

parser.add\_argument("pcap", help="PCAP file to analyze")

args = parser.parse\_args()

ids = NetworkIDS()

ids.analyze\_pcap(args.pcap)

3. `test\_ids.py` — Basic Unit Tests

`python

import unittest

from scapy.all import IP, ICMP, TCP

from ids import NetworkIDS

class TestNetworkIDS(unittest.TestCase):

def setUp(self):

self.ids = NetworkIDS(syn\_scan\_threshold=2, icmp\_flood\_threshold=2, syn\_flood\_threshold=2)

def test\_icmp\_ping\_detection(self):

# Create ICMP echo request packets

packets = []

for \_ in range(3):

pkt = IP(src="1.1.1.1", dst="2.2.2.2")/ICMP(type=8)

packets.append(pkt)

alerts = self.ids.detect\_icmp\_pings(packets)

self.assertTrue(any("ICMP Ping detected" in alert for alert in alerts))

self.assertTrue(any("ICMP Flood detected" in alert for alert in alerts))

def test\_tcp\_syn\_detection(self):

packets = []

# SYN packets without SYN-ACK (half-open)

for \_ in range(3):

pkt = IP(src="3.3.3.3", dst="4.4.4.4")/TCP(flags='S', dport=80)

packets.append(pkt)

alerts = self.ids.detect\_tcp\_syn(packets)

self.assertTrue(any("Half-open TCP connection detected" in alert for alert in alerts))

def test\_syn\_scan\_detection(self):

packets = []

# SYN packets to multiple ports from same source

for port in [22, 80, 443]:

pkt = IP(src="5.5.5.5", dst="6.6.6.6")/TCP(flags='S', dport=port)

packets.append(pkt)

alerts = self.ids.detect\_syn\_scan(packets)

self.assertTrue(any("Possible SYN scan detected" in alert for alert in alerts))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

4. `README.md` — Basic Documentation

# Lightweight Network IDS

Overview

This project implements a lightweight Network Intrusion Detection System (IDS) that analyzes PCAP files to detect:

- ICMP ping activity and ICMP floods

- TCP SYN connection attempts and half-open connections

- SYN scan patterns across multiple ports

Usage

Run the IDS on a PCAP file:

python ids.py sample\_scan.pcap

Alerts

The IDS prints alerts for suspicious activities detected in the PCAP.

Testing

Run unit tests with:

python -m unittest test\_ids.py

Next Steps

- Add detection for NULL and FIN scans

- Implement real-time packet capture and analysis

- Improve threshold tuning and false positive reduction

5. How to Run

- Install Scapy: `pip install scapy`

- Place your PCAP files (`sample\_normal.pcap`, `sample\_scan.pcap`) in the project folder.

- Run IDS on PCAP:

bash

python ids.py sample\_scan.pcap

- Run tests:

bash

python -m unittest test\_ids.py

POC on the above code

# Theory Explanation of the Network IDS Project

1.Objective

The goal of this Network Intrusion Detection System (IDS) is to monitor network traffic (from PCAP files) and detect suspicious activities such as:

- ICMP ping requests and floods

- TCP connection attempts and half-open connections

- SYN scan patterns indicating port scanning

This helps identify potential reconnaissance or attack attempts on a network.

2.Key Network Concepts

- ICMP (Internet Control Message Protocol): Used for diagnostic or control purposes, e.g., ping (echo request/reply).

- TCP (Transmission Control Protocol): Connection-oriented protocol using a three-way handshake (SYN, SYN-ACK, ACK) to establish connections.

- Port Scanning: Technique used by attackers to find open ports on a target by sending connection attempts to many ports.

3.Detection Logic

a. ICMP Ping Detection

- The IDS looks for ICMP packets with type 8 (echo request) or type 0 (echo reply).

- Each detected ping is logged as an alert.

- If a single source sends a large number of ICMP echo requests (above a threshold), it is flagged as an \*\*ICMP flood\*\*, which may indicate a denial-of-service (DoS) attack.

b. TCP SYN Connection Attempts and Half-Open Connections

- TCP connections start with a SYN packet from the client.

- The server responds with a SYN-ACK if the port is open.

- A half-open connection occurs when the client sends SYN packets but does not complete the handshake (no ACK received).

- The IDS counts SYN packets and SYN-ACK packets per connection tuple (source IP, destination IP, destination port).

- If SYN count exceeds SYN-ACK count, it indicates half-open connections, which may be a sign of a SYN flood attack or scanning.

c. SYN Scan Detection

- Attackers perform SYN scans by sending SYN packets to many ports on a target to identify open ports without completing the handshake.

- The IDS tracks the number of unique destination ports a source IP sends SYN packets to.

- If the count exceeds a threshold, it flags a possible SYN scan.

4.Implementation Details

- The IDS reads packets from a PCAP file using the Scapy library.

- It processes each packet to check for ICMP and TCP flags.

- Alerts are generated based on the detection logic described.

- Thresholds (e.g., number of ports scanned, number of ICMP packets) are configurable to balance sensitivity and false positives.

5.Why These Detections Matter

- ICMP Ping and Floods:Attackers use ping sweeps to discover live hosts. Floods can disrupt network availability.

- Half-Open Connections: SYN flood attacks exhaust server resources by leaving many connections half-open.

- SYN Scans: Common reconnaissance technique to map network services before launching attacks.

6.Limitations and False Positives

- Legitimate network tools (e.g., monitoring systems, vulnerability scanners) may trigger alerts.

- Thresholds need tuning based on network baseline to reduce false positives.

- The IDS does not analyze payloads or encrypted traffic.

- More advanced attacks may evade simple signature-based detection.

7. Next Steps for Improvement

- Add detection for other scan types (NULL, FIN scans).

- Implement real-time packet capture and alerting.

- Use statistical or machine learning methods to reduce false positives.

- Integrate with response systems for automated mitigation.