INTERNSHIP REPORT

Name: Annu
University: Geeta University
Course: Bsc Forensic Science
Internship Duration: June 1st 2025 – July 1st 2025
Company: CodeAlpha
Domain: Cyber Security

Objectives

My primary objective for this internship were to:

To gain practical knowledge in the field of cybersecurity through real-time learning and online modules.
To understand key concepts like cyber threats, vulnerabilities, ethical hacking, and data protection.
To develop skills in analyzing and identifying potential cybersecurity risks.
To enhance my technical knowledge related to network security and digital forensics.
To build a strong foundation that complements my background in forensic science.

Task and Responsibilities

Task 1: Basic Network Sniffer

- **Objective:**
  - To design and implement a Python-based network sniffer on a Linux system using Scapy, aimed at capturing and analyzing real-time network traffic.
  - The goal was to understand how data packets travel across a network and extract essential metadata such as IP addresses, protocol types, and payload information.

- **Responsibilities & Activities:**

Environment Setup:

Verified the installation of Python 3 and the Scapy library using terminal commands:

python3 --version to confirm Python installation.

pip3 install scapy --break-system-packages to install Scapy.

pip3 show scapy to verify the installation and version.

Script Development:

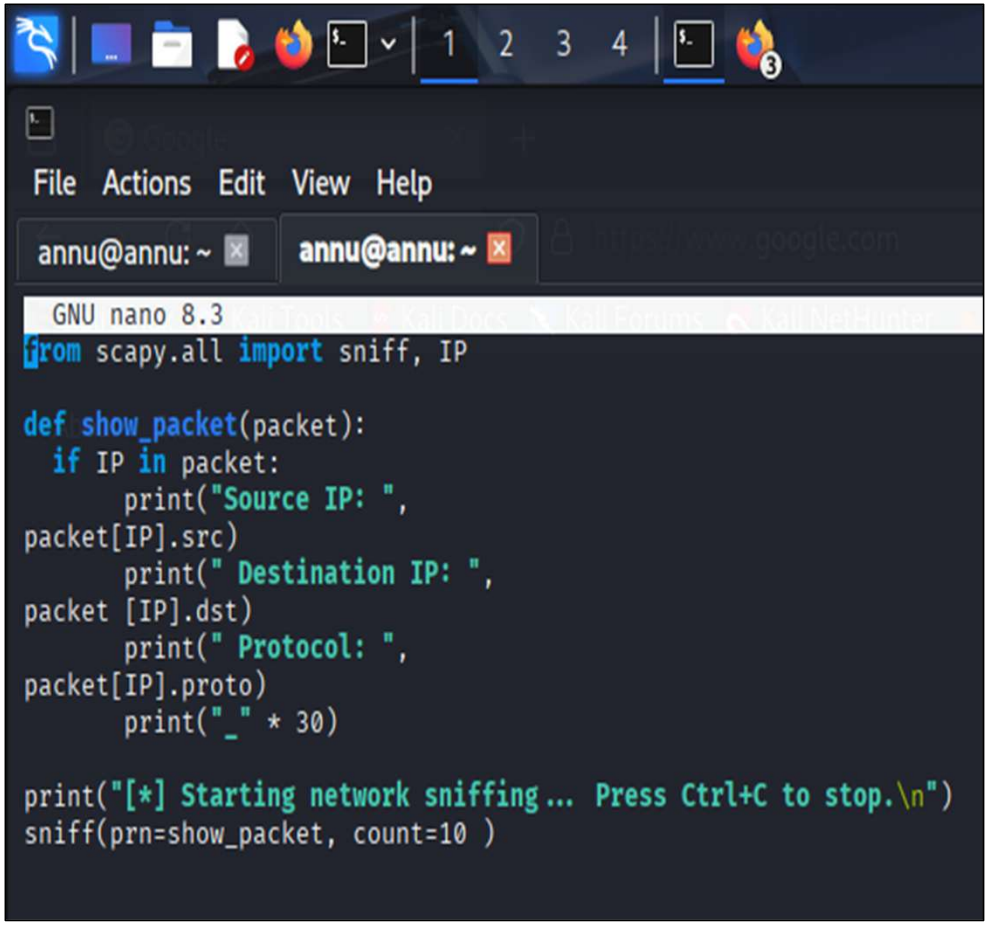
Created a Python script (sniffer.py) using the nano text editor.

Utilized Scapy to develop a custom packet sniffer capable of.

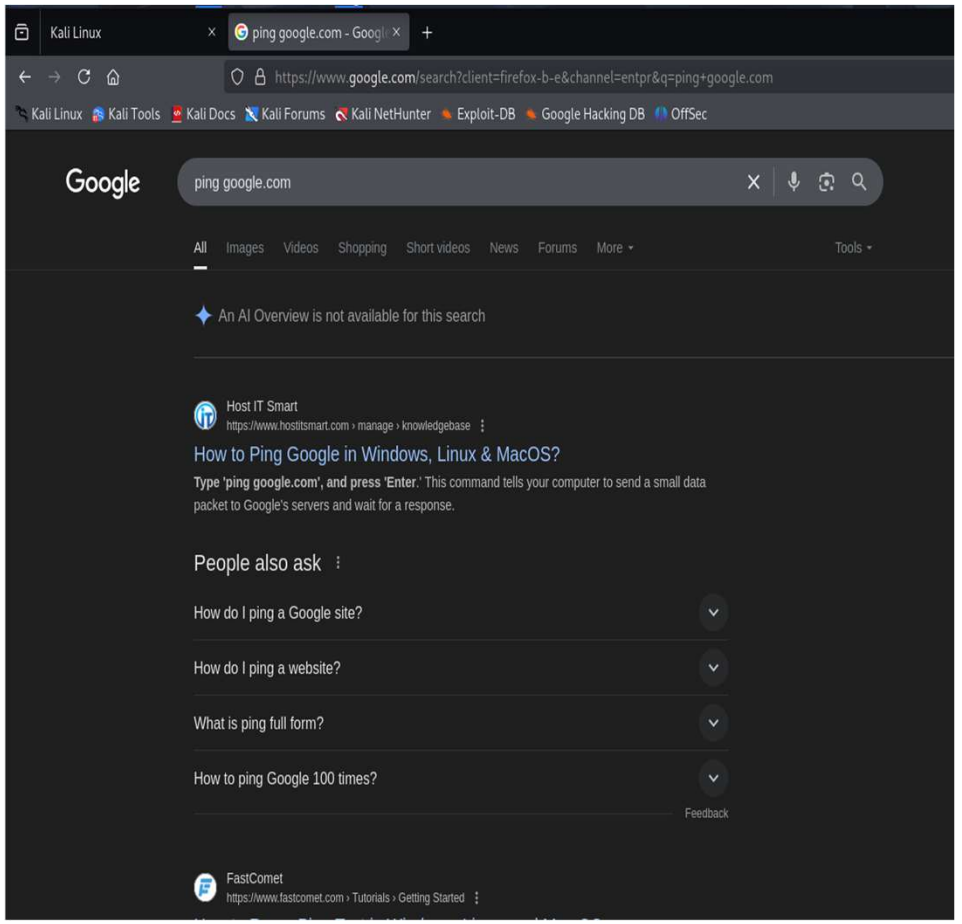
Monitoring live network traffic.

Capturing each packet and extracting: Source IP address ,Destination IP address, Protocol type ,Payload data (summary).

Scapy Script for Packet Sniffing



Web Traffic Generation via Browser



Sample code execution:

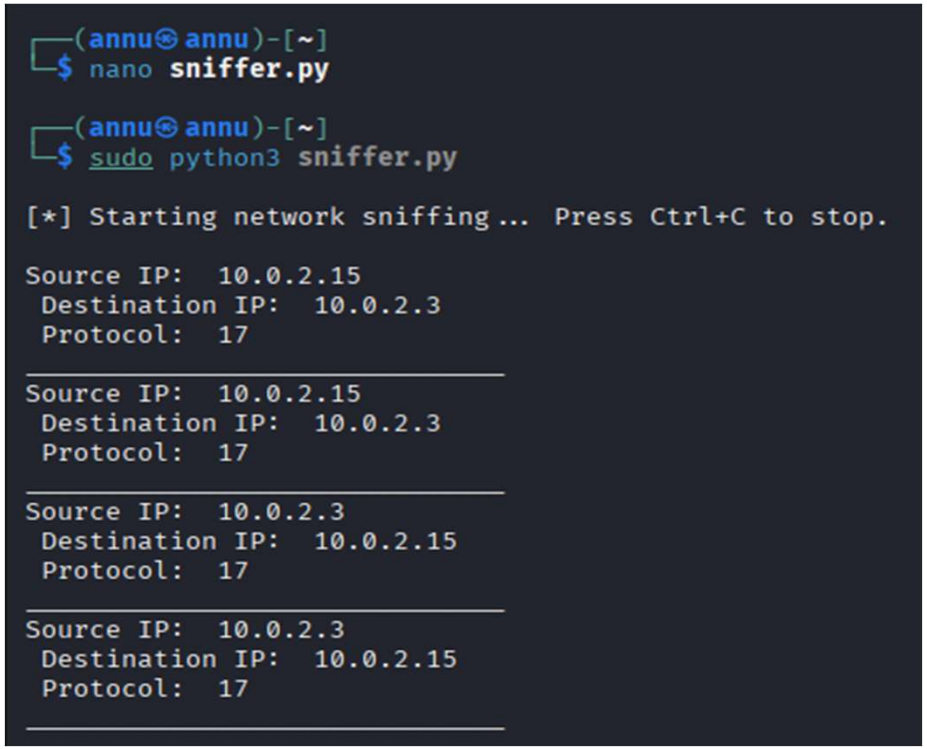
sudo python3 sniffer.py

Packet Traffic Generation:

Used browsers like Firefox or Chrome to initiate web traffic during script execution.
This allowed for the capture of dynamic packets in real time, enhancing the quality of the output and analysis.

Outcome:

Successfully built a basic yet functional network packet sniffer. The tool effectively captured and displayed live packet information, providing a clearer understanding of:Packet structure and flow.Protocol behavior (e.g., TCP, UDP, ICMP).The practical use of libraries like Scapy in cybersecurity and forensic investigations.



Task 4: Network Intrusion Detection System (Snort)

- **Objective**: Set up Snort to detect and log ICMP ping requests.

- **Responsibilities and Activities:**

Step 1: Snort Installation & Verification

- **Installation Command:**

  sudo apt update && sudo apt install snort –y

  apt update: Updates package lists to ensure you install the latest Snort version**.**

  -y: Automatically confirms installation prompts.

- **Verify Installation:**

```
┌──(annu㉿annu)-[~]
└─$ snort --v
o")~   Snort++ 3.1.82.0

Network Policy : policy id 0 :

Inspection Policy : policy id 0 :

pcap DAQ configured to passive.

host_cache
    memcap: 33554432 bytes

Snort successfully validated the configuration (with 0 warnings).
o")~   Snort exiting
```

Step 2 : Snort Configuration Deep Dive

Edit Configuration File:

sudo nano /etc/snort/your_config.conf

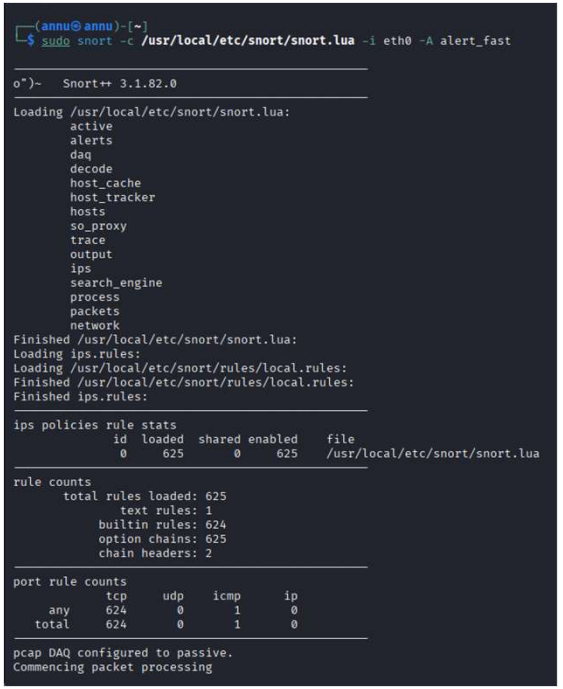
Add ICMP rule :

```
alert icmp any any → any any (msg:"ICMP Ping Detected"; sid:1000001;
```

Run Snort:

Flags Explained:

- **-A console :** Print alerts to terminal (real - time).
- **-q :** Quiet mode (reduces clutter)
- **- c /etc/snort.conf :** Path to your config file.
- **-I eth0 :** Network interface (check with ifconfig or ip a)

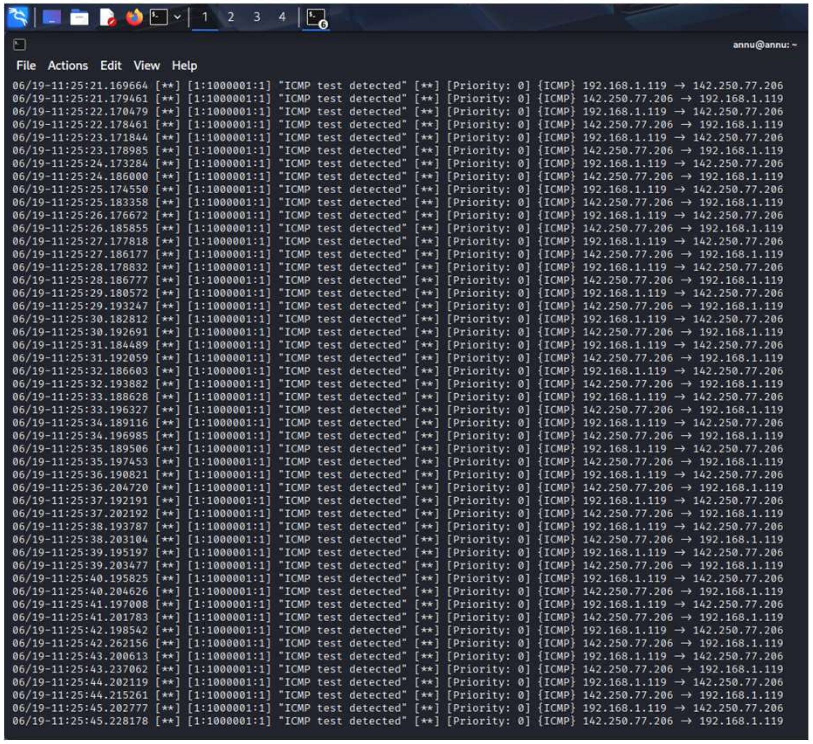


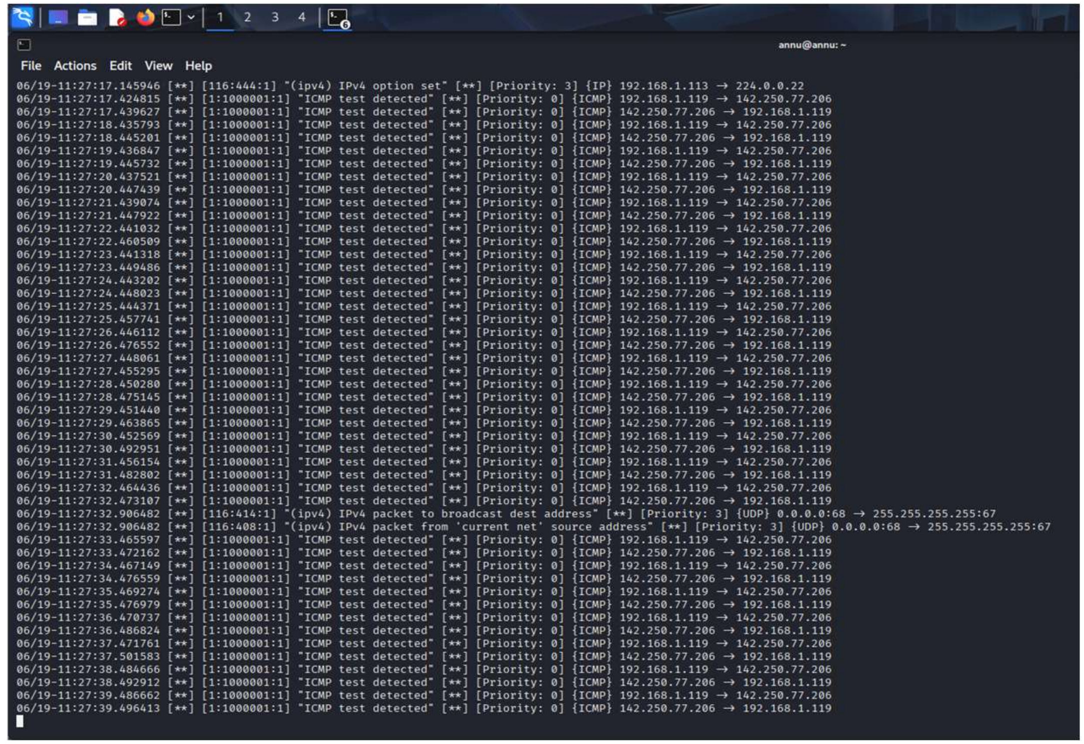
Trigger ICMP traffic and observe alerts:

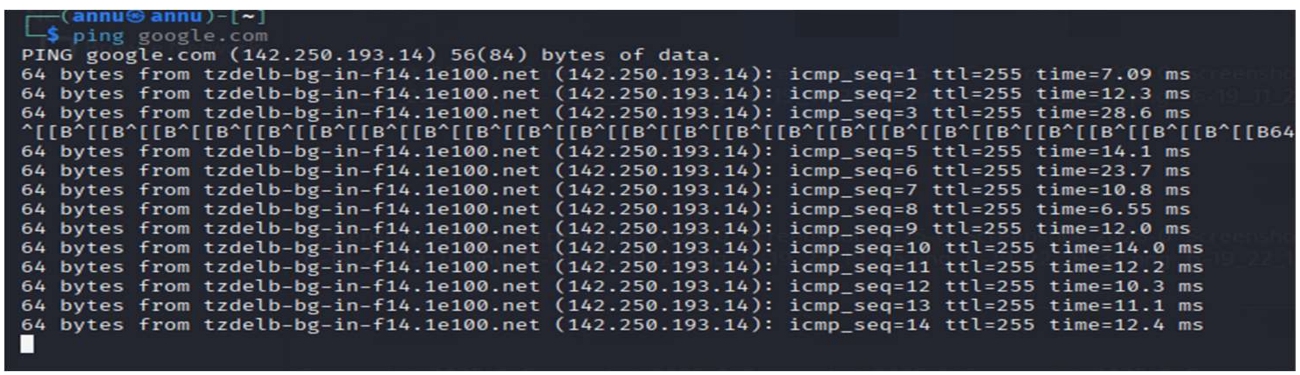
Purpose

Goal: Verify that Snort detects and logs ICMP (ping) requests in real-time.

Why ICMP?: Simple to test, commonly allowed in networks, and reveals basic IDS functionality


File Actions Edit View Help
annu@annu: ~


File Actions Edit View Help
annu@annu: ~


┌──(annu㉿annu)-[~]
└─$ ping google.com
PING google.com (142.250.193.14) 56(84) bytes of data.
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=1 ttl=255 time=7.09 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=2 ttl=255 time=12.3 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=3 ttl=255 time=28.6 ms
^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B64
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=5 ttl=255 time=14.1 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=6 ttl=255 time=23.7 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=7 ttl=255 time=10.8 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=8 ttl=255 time=6.55 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=9 ttl=255 time=12.0 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=10 ttl=255 time=14.0 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=11 ttl=255 time=12.2 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=12 ttl=255 time=10.3 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=13 ttl=255 time=11.1 ms
64 bytes from tzdelb-bg-in-f14.1e100.net (142.250.193.14): icmp_seq=14 ttl=255 time=12.4 ms

Key Details in Alert:

[**] … [**] : Rule triggered (matches your snort.conf rule).
10.129.84.57 -> 192.168.1.2 : Sorce -> destination IP.
{ICMP} : Confirms ICMP protocol detection.

Lessons Learned:

Task 1: Network Sniffer with Python:

1. Protocol Analysis:
Learned to decode raw packets (TCP/UDP/ICMP) using scapy, including:
Header structures (IP/MAC addresses, ports).
Payload extraction (limited to non-encrypted traffic).
Challenge: Handling fragmented packets required custom reassembly logic.

2. Tool Limitations:
scapy is powerful but slow for high-speed traffic (switched to pyshark for better performance).
Encrypted traffic (HTTPS) necessitated MITM techniques for deeper inspection.

3. Security Implications:
Raw packet capture requires root privileges (sudo), highlighting the need for strict permission controls.

Task 4: Snort IDS Implementation:

1. Rule Optimization:
Discovered that overly broad rules (e.g., alert icmp any any any any) generate false positives. Refined to:
alert icmp $EXTERNAL NET any SHOME NET any (msg: "Suspicious Ping; dsize>128; sid: 100 0002;)
Used dsize to filter large ICMP packets (common in ping floods).

2. Log Management:
Learned to rotate logs (logrotate) to prevent disk exhaustion during prolonged monitoring.

3. Real-World Gap:
Snort alone cannot block traffic (only detects). Integrated with iptables for automated blocking:
iptables -A INPUT -p icmp recent name ICMP ATTACK-set-J

Cross-Task Insights:

1. Complementary Tools:
Snort (detection) + Python sniffer (analvsis) provide lavered visibility.
wed psize to mertargetслар раскіна (common in png noods.

2. Ethical Boundaries:
Both tasks reinforced the importance of only monitoring authorized networks.

3. Professional Growth:
Coding: Improved Python skills for packet manipulation.
DevOps: Automated Snort deployment with Ansible for scalability.
Documentation: Created a rule-cheat sheet for future projects.

Acknowledgement:

I acknowledge CodeAlpha for providing learning opportunities and practical experience in technology and programming domains. Their contributions to skill development and hands-on project experience are valuable for aspiring developers and tech professionals.