

Network IDS and Packet Sniffer

I. Introduction

This project addresses the critical need for internal network security monitoring and data forensics capabilities within any organization or virtual lab environment. The objective was to design and implement a lightweight, customizable **Network Intrusion Detection System (NIDS)** and packet analyzer using Python. This tool demonstrates the foundational principles of **network security**, **real-time traffic analysis**, and structured **digital evidence collection**. The solution leverages Python's powerful ecosystem to actively monitor network traffic, identify potential threats like port scanning and flooding, and securely log all relevant data for later forensic investigation.

II. Abstract

This study details the development of a Python-based NIDS that utilizes the **Scapy** library for raw packet capture and parsing, providing granular access to Layer 3 and Layer 4 protocol fields. The system is designed with a focus on data persistence, writing detailed packet metadata to a flat file (packet_log.txt) and a structured **SQLite** database (packets.db). The core innovation is the integrated **Anomaly Detection Module**, which employs customized threshold-based logic to flag suspicious behaviors, including **Port Scans** and **Traffic Floods**, generating real-time alerts. The project successfully validates the ability to create robust, customized security tools in a controlled environment, directly supporting incident response and forensic readiness.

III. Tools Used

Tool/Technology	Role in Project	Screenshot Reference
Python 3	Core development language and environment.	04_IDE_Project_Structure.png
Scapy	Primary library for packet sniffing and protocol parsing (TCP, UDP, ICMP).	N/A (Embedded in sniffer.py logic)
SQLite3	Embedded relational database for structured, queryable data storage (packets.db).	01_Live_Traffic_Summary.png
netifaces	Used to automatically determine the default network interface for sniffing.	N/A
Kali Linux VM	Development and testing environment (required for root access for sniffing).	04_Flood_Simulation_Ping.png

IV. Steps Involved in Building the Project

The project followed a structured development approach, emphasizing modularity for scalability:

- Database and Logging Setup:** The `setup_database()` function initialized the SQLite structure for persistent, structured logging. Separately, a flat file logger (`log_packet`) was implemented for simple, timestamped record-keeping, essential for rapid initial triage during a forensic timeline reconstruction.
- Protocol Parsing with Scapy:** The `process_packet()` function was designed to intercept all packets on the network interface. It specifically extracted Layer 3 (IP Source/Destination) and Layer 4 (Port, Protocol, **TCP Flags**) data. This is evident in the detailed log file structure.

- **Demonstration: 03_Packet_Data_Persistence.png** shows the logged data, including crucial TCP Flags (e.g., FA for FIN/ACK), proving successful L4 parsing.
3. **Real-Time Anomaly Detection (IDS Logic):** Two threshold-based detectors were implemented using Python's defaultdict for real-time tracking:
- **Port Scan:** Tracks unique destination ports attempted by a source IP. A count exceeding **10** triggers an alert, simulating detection of tools like **Nmap**.
 - **Flood Attack:** Tracks the total number of packets from a source IP within a **10-second window**. A count exceeding **100** triggers an alert.
 - **Demonstration: 02_Anomaly_Alerts_Log.png** confirms the successful triggering of both "Port scan detected" and "Possible flood detected" alerts.
4. **Reporting and Concurrency:** The `live_summary()` function queries the SQLite database every 10 seconds to generate top talkers and protocol usage. The use of Python's **threading** ensured this reporting ran concurrently with the sniffing process, maintaining real-time analysis without dropping packets.
- **Demonstration: 01_Live_Traffic_Summary.png** displays the final output, showing the Protocol Breakdown (ICMP, TCP, UDP) and **Top 5 Source/Destination IPs** aggregated from the database. The initial **ping** command was used to test and trigger the ICMP-related Flood detection, as shown in **04_Flood_Simulation_Ping.png**.

V. Conclusion

The development of the Custom Network IDS and Packet Sniffer was a successful exercise in applying **Python scripting** to practical **cybersecurity** challenges. The project validates proficiency in core concepts critical for a career in security and forensics: **network protocol analysis**, **real-time data processing**, and the maintenance of a verifiable **chain of custody** through robust logging.

This system functions as a robust proof-of-concept for the initial stages of a **Network Forensics** investigation—identifying an incident (via alerts), preserving the state (via logs/DB), and enabling subsequent analysis (via summary reporting). This experience solidifies my fundamental knowledge in tools like **Nmap** (simulated detection) and **Wireshark** (programmatic packet analysis), directly aligning with the prerequisites for working with government or intelligence agencies in the **ethical hacking** and **digital forensics** domains.