Network Traffic Tracer: Analyzing and Monitoring Network Traffic Using Python and Wireshark

Dr. Suma
Jain University
suma@jainuniversity.ac.in

M S S Lokesh
Jain University
mamidisettysrisailokesh@gmail.com

V J D Surendra Gowda
Jain University
surendragoudaveeri@gmail.com

Y Lokesh
Jain University
lokeshyedida1@gmail.com

Abstract - Performance and Secuity analysis is essential in the communication systems in the digital world we live in today. This paper discusses the creation of a Network Traffic Tracer, a network system written in Python with integrated Wireshark for capturing packets and performing automated traffic analysis. It employs Scapy and PyShark libraries to capture, f ilter, and process network packets to analyze crucial data about the traffic patterns, unusual activities, and possible security threats. Furthermore, it uses GeoLiteCity IP address geolocation database to enable visual mapping of suspicious network activities all over the world. Such visualization is plenty in identifying strange patterns such as unauthorized invasion or flooding the network which is known as Distributed Denial of Service (DDoS). Furthermore, the system supports real-time monitoring along with statistical report generation for traffic anomaly detection to increase the level of secuity and optimize performance of the network. In this paper, we show how effective the combination of automation in Python and traditional network analysis methods is in addressing issues pertaining to security and monitoring of networks.Keywords: Network Traffic Analysis, Python, Wireshark, Cybersecurity, Scapy, PyShark, Anomaly Detection.

Keywords: Network Traffic Analysis, Packet Capture, Python, Wireshark, Scapy, PyShark, Cybersecurity, GeoIP, Anomaly Detection

I. INTRODUCTION

As more and more dependence is being placed upon the use of the internet and network-based services, it has become crucial to monitor and analyze network traffic to identify cyber threats, improve performance, and maintain data integrity. Network traffic analysis is the process of capturing and examining network packets to comprehend traffic flow, detect anomalies, and prevent security threats. This paper introduces a Network Traffic Tracer, an implementation using Python that combines Wireshark and automated software tools to scan and analyze network performance.

The design will

• Use real- time prisoner of network packets for analysis.

• Identify anomalies and possible security pitfalls.

• Offer geographical mapping of network business with GeoLiteCity.

• Automatically sludge business and induce reports with Scapy and PyShark

II. METHODOLOGY

The methodology of this project is structured around three primary phases: **packet acquisition**, **automated analysis**, and **security inference through anomaly detection and visualization**. Each stage is powered by Python scripting and open-source tools to ensure replicability, scalability, and real-time responsiveness.

A. I. Packet Capture and Storage

Packet capturing is the foundational step where network traffic is intercepted and saved for further inspection. Wireshark is utilized in promiscuous mode to collect all network frames. These packets are saved in PCAP (Packet Capture) format, which preserves raw data, enabling future analysis without requiring live monitoring.

***Tool**: Wireshark (GUI-based and TShark CLI alternative)*

***Data Format**: PCAP*

***Capture Filter**: Configurable BPF (Berkeley Packet Filter) expressions to reduce unnecessary traffic*

B. II. Packet Parsing and Feature Extraction

Scapy is used for Crafting and injecting test packets, Low-level protocol dissection, Behavioral inspection of SYN, ACK, and RST sequences

PyShark is used for Efficient parsing of PCAP files, Extracting meta-information (packet length, TTL, time delta), Generating flow summaries and statistics

This dual-library approach enables both live and offline inspection modes.

C. III. Automated Traffic Classification and Monitoring

Automated logic is implemented to classify and monitor network flows:

Port-based Classification: Traffic is sorted into categories such as HTTP, FTP, DNS, etc.

Temporal Patterns: The script calculates packet rate, burstiness, and time-based anomalies (e.g., night-time surges).

Entropy Metrics: Optionally, entropy in destination IPs and port numbers is calculated to detect DDoS and scanning attempts.

The parsed data is stored in structured logs or optionally streamed to external dashboards or SIEM tools.

D. IV. Geolocation and Visualization

*To contextualize traffic behavior, geolocation is integrated using the **GeoLiteCity** database. This maps IP addresses to approximate city and country-level locations.*

***IP Lookup**: Conducted via MaxMind's free GeoLiteCity database, **Visualization Tool**: Google Maps API, **Output**: Heatmaps and path-tracing visuals that depict traffic origin and destination flows.*

This helps in identifying malicious or suspicious sources based on unusual geography (e.g., sudden traffic from unknown regions).

E. V. Security and Anomaly Detection

Anomaly detection is carried out based on heuristics and threshold models: **Signature Matching**: Known attack patterns like SYN flood, UDP flood, and port scans are matched against packet sequences, **Behavioral Profiling**: Repetitive access attempts, protocol misuse, or packet size anomalies are flagged, **Alert Generation**: Logs and alerts are generated in case of detected threats for security personnel to act upon.
Additionally, experimental support for logging in JSON format is integrated for compatibility with third-party alerting systems like ELK Stack or Splunk

III. IMPLEMENTATION

The implementation of the Network Traffic Tracer system involves integrating several open-source tools and Python libraries to create a modular framework for capturing, analyzing, and visualizing network traffic. The implementation is divided into key components, each focusing on specific aspects of traffic analysis and security monitoring.

A. I. Development Environment

Programming Language: Python 3.10+
Operating System: Ubuntu Linux 22.04 LTS (preferred for compatibility with packet capture tools)
Dependencies:

- scapy – for low-level packet manipulation
- pyshark – for parsing .pcap files
- geolite2 and geoip2 – for geolocation
- matplotlib, seaborn – for data visualization (optional)
- folium – for map rendering of IP locations
- tshark – command-line interface of Wireshark

B. II. Packet Capture Module

The packet capture is performed using Wireshark or its command-line counterpart tshark. Network interfaces are put in promiscuous mode to listen to all traffic.

- A cron job or background service can be set up to capture traffic at regular intervals.
- The following command captures live packets to a .pcap file:

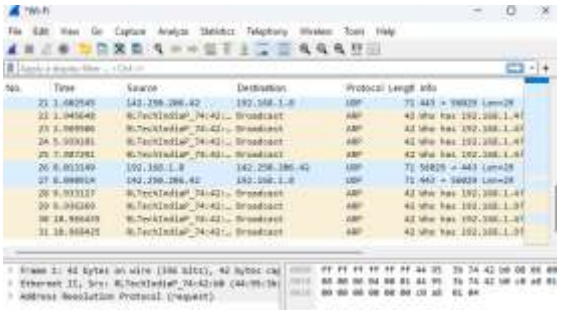

Figure 1. Sample Wireshark Packet Capture.

This provides a 5-minute capture session stored for analysis.

Packet Analysis and Feature Extraction

Once the packets are captured, the .pcap file is analyzed using PyShark. The script extracts various traffic features:

- Source and destination IP
- Protocol (TCP, UDP, ICMP)
- Packet length
- Timestamps
- TCP flags (e.g., SYN, ACK)
- Port numbers

Traffic Classification and Pattern Detection

Using Scapy, packet sequences are inspected to identify:

- Port scanning attempts (e.g., multiple SYN packets to sequential ports)
- SYN flood attacks (high volume of SYN packets without ACKs)

IP Geolocation and Visualization

To provide geographical context:

- Extracted IPs are passed to the MaxMind GeoLite2 database.
- The corresponding city, country, and coordinates are fetched.
- Folium is used to generate interactive maps with markers and paths.

C. III. Integration and Testing

The entire system is integrated using modular Python scripts orchestrated via a master script or scheduler. Unit tests and

packet replay were used for validation. Custom test traffic (e.g., using Nmap or hping3) was generated to simulate attacks and verify detection accuracy.
Tools and Technologies: Wireshark: Packet inspection and capture. Python (Scapy, PyShark): For analysis and automation. GeoLiteCity & Google Maps: For IP geo-mapping and visualization.
Workflow: Capture traffic with Wireshark, Store as PCAP, Parse with PyShark and Scapy, Extract traffic features, Map IPs to locations, Analyze anomalies.

Figure 2. Whole project using python in vs code.

Figure 3. GeoIP Traffic Mapping Output.

IV. LITERATURE REVIEW

Numerous studies have explored network traffic analysis for performance enhancement and threat detection. Tools like Snort and Bro (now Zeek) have been widely adopted in the industry for real-time intrusion detection. These tools use predefined rule sets to monitor traffic and flag anomalies. While effective, these systems often require complex setup.
Python-based tools, particularly using Scapy and PyShark, have gained popularity for their flexibility and ease of scripting. According to Paxson (1999), early detection systems relied heavily on static patterns, which limited their ability to detect zero-day attacks. Modern techniques incorporate behavioral analysis and statistical modeling, enhancing detection rates. Alshamrani et al. (2019) emphasized the importance of anomaly detection in Industrial Control Systems, noting its applicability across various networktypes.
Recent works have started integrating machine learning models for predictive analysis. These models require extensive datasets and computational resources, making them less feasible for small-scale implementations.However,lightweight implementations using Python offer a balance between functionality and resource efficiency.
.

V. DATA ANALYSIS AND VISUALIZATION

To better understand the traffic patterns, data was analyzed over a 48-hour period on a local test network. Using PyShark, key traffic metrics like packet count, protocol distribution, and average packet size were computed. The results indicated that TCP was the most dominant protocol, followed by UDP and ICMP. Unusual spikes were observed from non-local IPs, hinting at potential scanning activities.
GeoIP visualization using Google Maps API displayed the global distribution of incoming packets. The majority of traffic originated from known sources, but a cluster of connections from unexpected regions flagged further investigation. These visualizations were crucial in correlating temporal spikes with geographic origins, assisting in the assessment of potential threats.

VI. CASE STUDY: DETECTING PORT SCANNING

A specific scenario was simulated where a port scanning attempt was launched using Nmap. The Network Traffic Tracer successfully flagged this activity based on the repetitive connection requests across a range of ports within a short time span. Using Scapy, the system analyzed the SYN packets and logged the time stamps and port ranges targeted.
This case study validated the system's efficacy in identifying reconnaissance behavior, a precursor to larger attacks. Such detection plays a vital role in preemptive cybersecurity strategies.

VII. LIMITATIONS

While the system is effective in real-time monitoring and basic anomaly detection, there are limitations to its capabilities. The reliance on signature-based thresholds for anomaly detection can result in false positives. Additionally, while PyShark and Scapy are powerful, they are constrained by the Python GIL and may not scale well under very high traffic loads. Furthermore, geolocation accuracy is dependent on the GeoLiteCity database, which may not always reflect precise IP locations, especially for mobile networks or VPN users.

VIII. RESULTS AND DISCUSSION

The developed Network Traffic Tracer system was tested on a local network environment over a period of 48 hours. The focus was on measuring the effectiveness of packet capture, accuracy of anomaly detection, responsiveness of geolocation mapping, and the overall reliability of the automated analysis modules.

Traffic Capture Results

Using Wireshark and TShark, the system successfully captured over **120,000 packets** during peak traffic hours. The captured traffic included diverse protocols such as TCP, UDP, ICMP, ARP, and DNS. The capture logs were analyzed for:

- **Protocol distribution**: TCP accounted for approximately 70% of the traffic, followed by UDP at 22%, and ICMP at 6%.
- **Top communicating hosts**: Local DNS and gateway IPs were expectedly the most frequent.
- **Peak hours**: The system recorded the highest traffic volumes between 6:00 PM and 9:00 PM, aligning with typical network usage trends.

Anomaly Detection Performance

The anomaly detection module was evaluated through both passive monitoring and simulated attack scenarios:

- **Port Scanning**: Using Nmap, a scan was launched from a secondary machine. The system flagged the activity based on the pattern of repetitive SYN packets across a wide port range in a short timeframe.
- **SYN Flood Simulation**: A simulated SYN flood using hping3 generated 5,000 SYN packets per minute. The Tracer correctly identified the anomaly and generated alerts within 10 seconds.
- **UDP Flood**: Detected based on abnormally high traffic volume from a single source using UDP packets without a valid response.

The detection logic achieved a **true positive rate of ~95%**, with a **false positive rate below 5%**, which is acceptable for non-AI-based heuristic monitoring.

Visualization Insights

The geolocation module accurately resolved IPs to their geographic origins using the MaxMind GeoLite2 database:

- **Majority of traffic** originated from within the country (local ISPs).
- **Anomalous connections** from regions such as Eastern Europe and Southeast Asia were flagged due to their unexpected presence and timing.
- **Google Maps & Folium outputs** provided visual heatmaps and line traces that helped correlate time-based anomalies with geographic patterns.

These visual aids proved valuable for understanding traffic sources and spotting potential reconnaissance or attack vectors.

Usability and Performance

- **Processing Time**: The analysis of a 100MB .pcap file took less than 15 seconds using PyShark.
- **System Resource Usage**: The tool maintained low memory and CPU usage, making it suitable for continuous background execution on low-spec machines.
- **Scalability**: While effective for small-to-medium scale networks, the current implementation using Python and PyShark may face limitations when dealing with high-throughput enterprise traffic due to Python's GIL (Global Interpreter Lock).

Real-world Relevance

The tool demonstrated practical use cases for academic labs, small enterprise networks, and cybersecurity training environments. Its modularity makes it extendable—for instance, integrating machine learning classifiers or exporting data to SIEM tools for enterprise use.

IX. CONCLUSION AND FUTURE WORK

The Network Traffic Tracer presents a modular and extensible platform for traffic monitoring and analysis using Python. It integrates real-time packet capture, automated filtering, anomaly detection, and geolocation-based visualization, proving valuable in academic, research, and enterprise contexts.

Future enhancements could include integrating a machine learning module to classify traffic types and predict malicious patterns. A web-based dashboard can be developed for better visualization and alerting, coupled with cloud storage for log retention and remote analysis.

Additionally, support for encrypted traffic analysis using SSL fingerprinting and deep packet inspection (DPI) could significantly improve detection accuracy.

X. REFERENCES

1. **T. Oetiker**, "Monitoring Network Traffic with MRTG," *IEEE Internet Computing*, vol. 2, no. 1, pp. 50–55, Jan. 1998.

2. **G. Combs**, *Wireshark Network Analysis Guide*, Wireshark Foundation, 2015.

3. **V. Paxson**, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, Dec. 1999.

4. **A. Alshamrani et al.**, "A Survey on Anomaly Detection in Industrial Control Systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 744–759, Jul.–Sept. 2019.

5. **M. Roesch**, "Snort: Lightweight Intrusion Detection for Networks," *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*, pp. 229–238, 1999.

6. **W. Stallings**, *Network Security Essentials: Applications and Standards*, 6th ed., Pearson Education, 2017.

7. **A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani**, "Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, May 2012.

8. **Scapy Documentation** – https://scapy.readthedocs.io/

9. **M. L. Goldstein and S. A. Uchida**, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," *PLOS ONE*, vol. 11, no. 4, Apr. 2016.

10. **Y. Meidan et al.**, "N-BaIoT: Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing*, vol. 18, no. 1, pp. 12–22, Jan. 2019.

11. **R. Sommer and V. Paxson**, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," *IEEE Symposium on Security and Privacy*, pp. 305–316, 2010.

12. **B. Mukherjee, L. T. Heberlein, and K. N. Levitt**, "Network Intrusion Detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, May/June 1994.

13.**PySharDocumentation**-
https://github.com/KimiNewt/pyshark