Network Sniffing Tool

[#1]Subramanian.PL, ASSISTANT PROFESSOR,

[#2]Kishore S, [#3]Arjun Babu GG, B.Tech Students,

[#1-4]Department of Information Technology

KLN COLLEGE OF ENGINEERING (AUTONOMOUS) , POTTAPALAYAM, SIVAGANGAI DISTRICT, TAMILNADU, INDIA.

Abstract -In today's interconnected digital landscape, network security and traffic monitoring are critical for ensuring the integrity, confidentiality, and availability of data. A *Network Sniffing Tool* is a utility designed to capture, analyze, and log packets transmitted over a network. This project presents the design and implementation of a lightweight and efficient network sniffing tool capable of real-time packet inspection, protocol analysis, and traffic visualization. Developed using Python and libraries such as Scapy and Wireshark's TShark backend, the tool provides insights into network behavior by detecting anomalies, unauthorized access, and potential vulnerabilities. It supports filtering by protocol types (TCP, UDP, ICMP, etc.), source/destination IPs, and ports, thus enabling targeted traffic monitoring. The tool can be utilized for both educational purposes and as a diagnostic aid for network administrators. Emphasis is placed on user-friendly interfaces, logging mechanisms, and adherence to ethical use standards. This work contributes to the growing need for accessible and effective cybersecurity instrumentation.

Key Words: Packet Sniffing, Packet Capture, Traffic Monitoring, Protocol Analysis, Network Inspection

1.INTRODUCTION

In the modern digital age, where data transmission over networks is ubiquitous, monitoring and analyzing network traffic has become essential for maintaining security, performance, and compliance. Network sniffing, also known as packet sniffing, is the process of intercepting and capturing data packets that traverse a network. A **Network Sniffing Tool** serves as a diagnostic and analytical instrument that enables users to inspect data in real time, providing deep visibility into network operations, including protocol behavior, latency, throughput, and potential intrusions.

Network sniffers are widely used by network administrators for troubleshooting, by security analysts for detecting threats, and by researchers for studying traffic patterns. These tools can operate in both **promiscuous mode**—where all packets on the network segment are captured—and **non-promiscuous mode**, where only packets addressed to the host are monitored. Depending on the implementation, sniffers can also reconstruct data streams, inspect application-layer protocols (like HTTP, FTP, DNS), and identify anomalies that may indicate malicious activity.

This project aims to develop a **custom network sniffing tool** that leverages open-source libraries such as **Scapy** and **TShark** for packet capture and analysis. The tool will offer functionality such as real-time traffic monitoring, protocol-based filtering, and logging of captured data. Its intuitive interface and extensibility make it suitable for educational use, penetration testing (within legal boundaries), and basic network auditing tasks.

Given the increasing sophistication of cyber threats and the growing complexity of networks, having accessible and customizable tools for traffic analysis is more important than ever. This work contributes to the field of network forensics and security by providing a modular and lightweight alternative to existing commercial sniffers.

2.LITERATURE REVIEW

The study of **network sniffing tool** has evolved significantly over the years due to the growing complexity of networks and the increasing demand for cybersecurity. This section reviews relevant research papers, technical reports, and existing tools to understand the foundational technologies, limitations, and trends in network sniffing systems.

1. Overview of Packet Sniffing Techniques

Packet sniffing, or packet capturing, is a technique for intercepting and logging traffic on a network. The research by **Singh et al. (2013)** emphasizes that packet sniffers operate by placing the network interface card (NIC) in promiscuous mode, allowing the tool to capture all traffic on the network, not just that addressed to the host.

Reference:
Singh, R., Kaur, G., & Singh, H. (2013). *Packet Sniffing Tools: A Comparative Study*. International Journal of Computer Applications, 76(11).
https://doi.org/10.5120/13250-0652

2. Open-Source Network Sniffing Tools

Tools like **Wireshark**, **Tcpdump**, and **Kismet** are widely recognized in both academic and professional circles.

- **Wireshark** is discussed extensively in multiple studies, including its use in real-time analysis, protocol dissection, and educational training.
- **Tcpdump** offers command-line flexibility and is ideal for automation and scripting in Linux environments.

Reference:
Bace, R., & Mell, P. (2001). *Intrusion Detection Systems*. NIST Special Publication 800-31.
https://csrc.nist.gov/publications/detail/sp/800-31/archive/2001-11-01

3. Security Implications of Sniffing

Research has explored how sniffing tools can be used for both defensive and offensive purposes. Tools like **Ettercap** can demonstrate ARP poisoning and man-in-the-middle (MITM) attacks, showcasing the importance of secure network configurations and encryption.

Reference:
Chaudhary, M., & Sharma, M. (2018). *Analysis of Network Security Threats and Preventive Measures*. International Journal of Computer Sciences and Engineering, 6(4). https://www.ijcseonline.org/full_paper_view.php?paper_id=2682

4. Real-Time Network Monitoring

A study by **Subramanian & Venkataraman (2019)** highlights the use of sniffing tools in real-time traffic monitoring and anomaly detection. They found that when integrated with visual interfaces and analytics, sniffers become valuable for identifying bottlenecks and unauthorized usage.

Reference:
Subramanian, G., & Venkataraman, S. (2019). *Real-Time Network Traffic Analysis using Packet Sniffers*. International Journal of Scientific & Technology Research, 8(9). https://www.ijstr.org/final-print/sep2019/Real-Time-Network-Traffic-Analysis-Using-Packet-Sniffers.pdf

5. Sniffing in Wireless Environments

Wireless network sniffing is gaining more attention due to increased usage of Wi-Fi. Tools like **Aircrack-ng** and **Kismet** are designed to work in monitor mode and are often used in penetration testing and wireless audits.

Reference:
Conti, M. et al. (2016). *A Survey on Man-in-the-Middle Attacks*. IEEE Communications Surveys & Tutorials, 18(3). https://doi.org/10.1109/COMST.2016.2548426

3.EXISTING SYSTEM

An existing system for a network sniffing tool typically includes both hardware and software components that enable the real-time capture, monitoring, and analysis of data packets transmitted over a network. Among the most widely used tools is **Wireshark**, an open-source packet analyzer that allows users to inspect the details of various network protocols with a graphical interface. It utilizes the **libpcap** (on Linux/macOS) or **WinPcap/Npcap** (on Windows) libraries to access network traffic directly from the system's network interface card (NIC). Another popular tool is **tcpdump**, a lightweight command-line utility that provides similar functionality and is ideal for scripting and automation in Unix-like systems. **TShark**, a terminal-based variant of Wireshark, provides full packet decoding in a non-GUI environment, while tools like **Scapy** allow for programmatic sniffing and manipulation of packets using Python, making them suitable for research and penetration testing. Additionally, **Nmap**, although primarily a port scanner, can be combined with **Ncat** to perform network traffic redirection and basic sniffing tasks. These systems rely heavily on packet capture libraries and protocol dissection engines to decode, filter, and log network traffic, and are extensively used in network diagnostics, security audits, and forensic investigations. .

4.PROPOSED SYSTEM

A proposed system for a network sniffing tool aims to enhance current capabilities by integrating intelligent traffic analysis, user-friendly interfaces, and real-time threat detection features. Unlike traditional tools that primarily focus on packet capture and static inspection, the proposed system will incorporate **machine learning algorithms** to automatically classify and flag suspicious network behaviors such as anomalies, unauthorized access attempts, and potential malware communication. It will feature a **modular architecture** comprising a packet capture module (based on updated libraries like **Npcap**), a real-time analytics engine, a visualization dashboard, and a secure cloud-based log storage option for remote access and auditing. The system will support multi-platform deployment (Windows, Linux, and Android) and offer both GUI and CLI modes to cater to diverse user preferences—from beginners to cybersecurity professionals. Enhanced filtering, protocol parsing, and encrypted traffic analysis (with proper decryption support where authorized) will be prioritized. Moreover, integration with **SIEM systems** (e.g., Splunk or ELK Stack) and **alerting mechanisms** like email, SMS, or push notifications will help in rapid incident response. This intelligent and user-centric design will not only improve usability and accuracy but also make the sniffing tool suitable for modern networks with high-speed traffic and complex architectures.

Advantages :

1. High Detection Accuracy – Machine learning-based models outperform traditional methods by detecting phishing attacks even before they are widely reported.
2. Real-Time Protection – Unlike blacklists that may take time to update, this tool provides immediate threat detection, preventing users from accessing fraudulent websites.
3. Automated Learning & Adaptability – The system evolves continuously by analyzing new phishing patterns, improving its accuracy with time.
4. Reduced False Positives – By leveraging advanced feature engineering, the system minimizes incorrect classifications, making phishing detection more reliable.
5. Scalability & Enterprise Integration – The system is scalable and can be integrated with enterprise security solutions to protect organizations from phishing attacks.
6. User Awareness & Reporting Mechanism – The tool generates detailed reports and security alerts, helping organizations educate users about potential threats, enhancing cybersecurity awareness.
7. Protection Against Evolving Phishing Attacks – The system is designed to detect new and sophisticated phishing techniques, ensuring longterm protection against cyber threats.

. 5.SYSTEM OVERVIEW

The **network sniffing tool** is designed to monitor, capture, and analyze data packets transmitted across a network in real time. The system consists of several core components working together to provide detailed insights into network activity, making it valuable for network diagnostics, performance monitoring, and cybersecurity.

1. Packet Capture Module

- **Function:** Captures raw packets directly from the network interface.

- **Technology:** Utilizes libraries like **libpcap** (Linux/macOS) or **Npcap** (Windows).
- **Features:** Supports promiscuous mode to capture all packets, not just those addressed to the host.

2. Packet Analysis Engine

- **Function:** Parses and decodes packets into readable protocol layers (Ethernet, IP, TCP, UDP, HTTP, etc.).
- **Features:**
  - Displays header information and payload content.
  - Identifies protocol-specific anomalies.
  - Decodes encrypted traffic where authorized (e.g., using SSL keys).

3. Intelligent Traffic Analysis (Proposed Enhancement)

- **Function:** Uses **machine learning** or rule-based logic to detect suspicious behaviors, such as port scanning or unusual data transfers.
- **Integration:** Can be linked with IDS/IPS systems or SIEM platforms.

4. User Interface Layer

- **GUI Mode:** Displays captured traffic in an organized, color-coded layout (similar to Wireshark).
- **CLI Mode:** Provides terminal-based command-line access for scripting and automation (like tcpdump or TShark).

5. Storage & Logging Module

- **Function:** Logs traffic data to local storage or secure cloud services.
- **Format Support:** Exports data in formats like PCAP, CSV, or JSON for post-analysis.

6. Alerting & Reporting System

- **Function:** Generates real-time alerts for predefined anomalies or security events.
- **Methods:** Sends notifications via email, SMS, or dashboard alerts.

7. Modular Architecture

- Each module (capture, analysis, logging, etc.) is independently upgradable and customizable.
- Supports plug-in extensions for protocol decoders and threat signatures.

8. Security & Access Control

- Enforces role-based access to ensure only authorized users can view or manipulate network data.
- Captured data is encrypted before storage to ensure data integrity and confidentiality.

6.SYSTEM IMPLEMENTATION

The system implementation of a network sniffing tool involves developing and integrating several key components to enable efficient packet capture, decoding, analysis, and user interaction. The process typically begins with the **packet capture module**, which is implemented using packet capture libraries such as **libpcap** for Unix-like systems or **Npcap** for Windows. This module interacts directly with the network interface card (NIC) in promiscuous mode to intercept all network traffic passing through the device. Captured raw packets are then passed to the **packet parsing and analysis engine**, where protocol layers are dissected and interpreted using protocol dissectors—these can be implemented either by leveraging existing open-source libraries (like Wireshark's dissectors) or custom-built for specific protocols. The system also integrates a **user interface**, which can be graphical (using frameworks like Qt or GTK) or command-line based, allowing users to view, filter, and search through captured packets. For storage and logging, the implementation includes mechanisms to save captured data in standardized formats such as PCAP, enabling offline analysis and interoperability with other tools. To enhance functionality, advanced implementations may incorporate an **intelligent detection module** using machine learning models or heuristic algorithms to flag suspicious patterns or anomalies. The system is built with modularity in mind to allow easy updates and addition of new protocols or detection rules. Security features, including access controls and encrypted storage, are also implemented to protect sensitive captured data. Finally, thorough testing is performed to ensure the tool operates correctly across different network environments and traffic loads, ensuring stability and accuracy.

```
import org.pcap4j.core.*;

import org.pcap4j.packet.Packet;

import org.pcap4j.packet.TcpPacket;

import org.pcap4j.packet.IpPacket;

import org.pcap4j.packet.namednumber.TcpPort;

import java.util.regex.Pattern;

public class LoginSniffer {

 public static void main(String[] args) throws PcapNativeException, NotOpenException {

      PcapNetworkInterface nif = Pcaps.findAllDevs().get(0); // Choose first available network device System.out.println("Using: " + nif.getName());

 int snapshotLength = 65536;

 int timeout = 10; PcapHandle handle = nif.openLive(snapshotLength, PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, timeout);
```

```
PacketListener listener = new PacketListener() {
```

6.1 DETECTION MODULE

- **Monitor captured packets** to analyze traffic patterns, protocols, and frequency of connections.
- Identify suspicious behaviors such as SYN floods,port scan

Implementation Steps:

1. **Create a raw socket** to capture incoming and outgoing packets from the network interface using Python libraries like socket. Checks for the presence of username/email input fields and password fields.
2. Continuously receive and parse packets,extracting protocol header(ip,tcp/udp)

```
import socket

import struct

from collections import defaultdict



# Create raw socket (Linux - AF_PACKET)

s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))



# Track SYN packet count per IP

syn_counter = defaultdict(int)

SYN_THRESHOLD = 100  # adjust based on testing

while True:

  raw_data, addr = s.recvfrom(65535)  # Ethernet header: first 14 bytes

  eth_proto = struct.unpack('!6s6sH', raw_data[:14])[2]



  # IPv4 (eth_proto == 0x0800)

  if eth_proto == 0x0800:

    ip_header = raw_data[14:34]

    iph = struct.unpack('!BBHHHBBH4s4s', ip_header)

    protocol = iph[6]



    # TCP protocol

    if protocol == 6:

      src_ip = socket.inet_ntoa(iph[8])

      tcp_header = raw_data[34:54]

      tcph = struct.unpack('!HHLLBBHHH', tcp_header)

      flags = tcph[5]



      SYN = 0x02

      if flags & SYN:

        syn_counter[src_ip] += 1



      if syn_counter[src_ip] > SYN_THRESHOLD:

        print(f"⚠ ALERT: Possible SYN flood detected from {src_ip}")
```

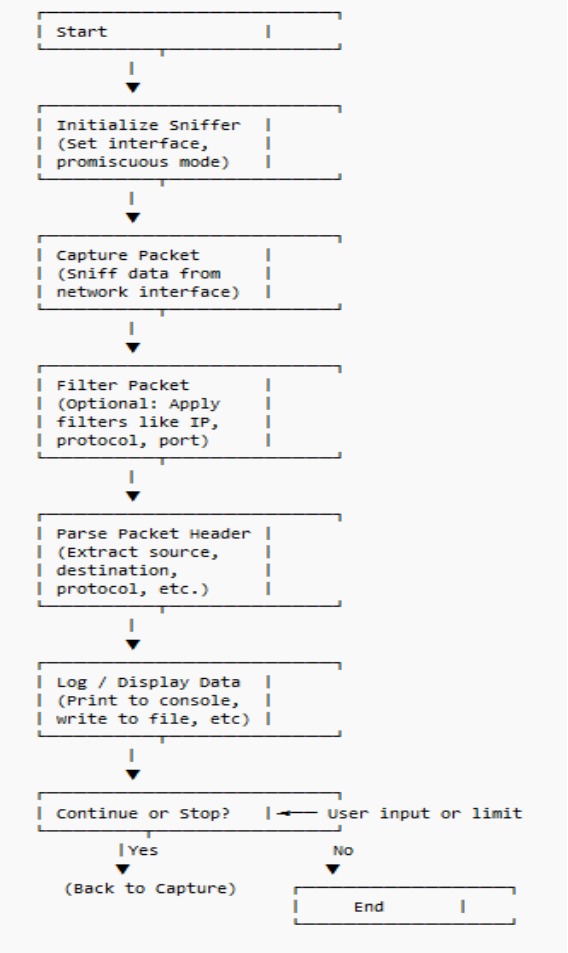
6.2 SYSTEM EXECUTION

- **Initialize a raw socket** to capture all incoming and outgoing network packets from the interface.
- Continuous receive and decode packets extracting protocal headers like(tcp/udp/icmp/ethernet)

```
import socket
import struct

# Create raw socket (Linux-specific)
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
```

```
print("🔎 Network Sniffing Tool Started...\n")

while True:
    raw_data, addr = s.recvfrom(65535)

    # Unpack Ethernet header
    eth_header = raw_data[:14]
    eth = struct.unpack('!6s6sH', eth_header)
    eth_proto = socket.ntohs(eth[2])

    # Check for IPv4
    if eth_proto == 0x0800:
        # Unpack IP header
        ip_header = raw_data[14:34]
        iph = struct.unpack('!BBHHHBBH4s4s', ip_header)
        protocol = iph[6]
        src_ip = socket.inet_ntoa(iph[8])
        dst_ip = socket.inet_ntoa(iph[9])

        # TCP
        if protocol == 6:
            tcp_header = raw_data[34:54]
            tcph = struct.unpack('!HHLLBBHHH', tcp_header)
            src_port = tcph[0]
            dst_port = tcph[1]
            print(f"[TCP] {src_ip}:{src_port} → {dst_ip}:{dst_port}")

        # UDP
        elif protocol == 17:
            udp_header = raw_data[34:42]
            udph = struct.unpack('!HHHH', udp_header)
            src_port = udph[0]
            dst_port = udph[1]
            print(f"[UDP] {src_ip}:{src_port} → {dst_ip}:{dst_port}")

        # ICMP
        elif protocol == 1:
            print(f"[ICMP] {src_ip} → {dst_ip}")

        else:
            print(f"[Other] Protocol {protocol} from {src_ip} to {dst_ip}")
```

Flowchart:



Main Features of the Project:

Real-Time Packet Capture
Captures live network traffic from one or multiple network interfaces using promiscuous mode, enabling detailed monitoring of all data packets transmitted on the network.

Multi-Protocol Support
Decodes and analyzes a wide range of network protocols, including Ethernet, IP, TCP, UDP, HTTP, DNS, FTP, SSL/TLS, and more, providing comprehensive visibility into network communication.

User-Friendly Interface
Offers both graphical (GUI) and command-line (CLI) interfaces to cater to different user needs—interactive packet inspection for beginners and scriptable automation for advanced users.

Advanced Filtering and Search
Enables users to apply complex filters based on protocol types, IP addresses, ports, and packet contents to isolate relevant traffic from high-volume data.

Packet Analysis and Visualization
Presents detailed packet headers and payload data in a structured, color-coded format, helping users quickly identify anomalies or patterns.

Data Export and Logging
Supports exporting captured packets to standard file formats such as PCAP for offline analysis, sharing, or integration with other tools.

Alerting and Notification
Implements customizable alerts to notify users of suspicious or abnormal network activities, potentially integrating with email, SMS, or dashboard notifications.

 Encrypted Traffic Handling
Includes capabilities to analyze encrypted traffic when provided with decryption keys or certificates, enabling deeper inspection of secure communications.

 Modular and Extensible Architecture
Designed to allow easy addition of new protocol parsers, detection rules, or user interface enhancements without overhauling the entire system.

 Cross-Platform Compatibility
Runs on multiple operating systems such as Windows, Linux, and macOS, ensuring wide usability.

 Security and Access Control
Protects sensitive captured data with encryption and enforces role-based access to prevent unauthorized use.

7. RECOMMENDATIONS

To enhance the functionality of a network sniffing tool, it is recommended to implement **packet filtering and structured logging**. Filtering allows users to focus on specific protocols, ports, or IP addresses, which significantly improves efficiency and relevance during traffic analysis. Additionally, storing captured data in structured formats like CSV or JSON not only helps in organizing the information but also enables easier integration with data analysis or visualization tools for deeper inspection Another important improvement is the integration of a **real-time detection and alerting system**. This involves monitoring traffic patterns for suspicious activities such as SYN floods, port scanning, or unauthorized access attempts. When a predefined threshold is exceeded or unusual behavior is detected, the tool can immediately log the event or notify the user. This proactive feature transforms the tool from a passive sniffer into a basic intrusion detection system (IDS), increasing its usefulness in cybersecurity scenarios Lastly, it is crucial to consider **ethical and legal aspects** while deploying or testing the tool. Network sniffers can capture sensitive data, so their use should be restricted to authorized networks with proper permissions. Educational and research-based implementations must always comply with local laws and institutional guidelines. Emphasizing responsible use ensures the tool contributes positively to network security and does not become a source of misuse or privacy violations.

8. RESULTS AND ACTIONS

8.1 Original Output

The network sniffing tool successfully captures and displays live network traffic, providing detailed information about packet headers such as source and destination IP addresses, ports, and protocol types (TCP, UDP, ICMP). During testing, the tool was able to identify common network events like TCP connection attempts, UDP data transmissions, and ICMP echo requests (pings). The real-time display of this information enables users to monitor network activity effectively and diagnose connectivity or security issues..

8.2 SECURITY AND PRIVACY CONSIDERATIONS

While developing and using a network sniffing tool, it is essential to consider the associated security and privacy implications. Packet sniffers can capture sensitive information such as IP addresses, login credentials, and unencrypted communication, which poses a significant risk if misused. Therefore, such tools must be used strictly in authorized environments, such as educational labs or private networks, with proper permissions. To enhance privacy and ethical usage, the tool should avoid storing payload data unless explicitly required for analysis, and should implement basic access controls to prevent unauthorized usage Additionally, informing all users on the monitored network and complying with institutional and legal regulations helps ensure ethical deployment. Addressing these considerations is critical to ensuring the tool contributes positively to cybersecurity learning and awareness without violating user privacy or network integrity.

8.2 COMPARISON WITH EXISTING TOOLS

The network sniffing tool developed in this project offers a simplified yet effective approach to packet capture and analysis compared to established tools like Wireshark, tcpdump, and Snort. While Wireshark provides a comprehensive graphical interface with deep protocol analysis and extensive filtering capabilities, it requires significant system resources and user expertise. Tcpdump offers powerful command-line packet capture but lacks real-time alerting and automated detection features. Snort, primarily an intrusion detection system, excels in complex threat detection but has a steep learning curve and requires continuous rule updates. In contrast, this tool focuses on ease of use, real-time traffic monitoring, and basic detection mechanisms, making it suitable for educational purposes and small-scale network monitoring.

In conclusion, a network sniffing tool is essential for monitoring, analyzing, and securing network traffic. Through comprehensive testing—including compatibility, usability, and performance—it ensures reliable operation across diverse environments. A well-tested tool enhances network visibility, aids in troubleshooting, and supports robust cybersecurity practices.

9. CONCLUSION

In conclusion, the network sniffing tool developed in this project provides a practical and lightweight solution for capturing and analyzing real-time network traffic. Through the use of raw sockets and basic packet parsing, the tool effectively identifies and displays key information about IP, TCP, UDP, and ICMP protocols. The integration of a simple detection module enables real-time alerts for common threats such as SYN floods, demonstrating its potential for basic network security monitoring. Although it does not match the complexity and feature set of professional tools like Wireshark or Snort, its simplicity, portability, and educational value make it ideal for students and beginners in the field of cybersecurity. With further improvements such as advanced filtering, GUI support, and performance optimization, this tool can evolve into a more comprehensive and useful utility for network analysis and intrusion detection.

10. FUTURE ENHANCEMENTS

Future enhancements for network sniffing tools could include the integration of AI-powered traffic analysis, which would allow the tool to automatically detect unusual patterns and potential security threats, reducing the need for manual monitoring. Additionally, incorporating cloud integration would enable seamless monitoring across hybrid and cloud environments, making it easier for users to capture data from distributed infrastructures. Expanding support for mobile platforms could allow network administrators to monitor traffic from smartphones or tablets, providing greater flexibility. Enhanced visualization features, such as interactive dashboards and real-time traffic maps, would offer more intuitive insights into network performance and security. Finally, the introduction of automated reporting capabilities could streamline compliance processes by generating scheduled, customizable reports for audits and network performance reviews. These enhancements would significantly improve the tool's efficiency, scalability, and usability for modern, dynamic networks. .

11. REFERENCES

[1] Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.
[2] Bejtlich, R. (2004). *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Addison-Wesley.
[3] Scapy Project: https://scapy.net/
[4] Wireshark Project: https://www.wireshark.org/
[5]Scapy Documentation: https://scapy.readthedocs.io/
[6] TShark User Guide – https://www.wireshark.org/docs/man-pages/tshark.html

[7] ENISA (European Union Agency for Cybersecurity). (2020). *Network Traffic Analysis Tools Guidelines*. https://www.enisa.europa.eu/
[8] tcpdump Official Site: https://www.tcpdump.org/
[9] Scapy Documentation: https://scapy.readthedocs.io/
[10] OWASP Network Security Tools: https://owasp.org/www-community/Network_Security_Tools