A Midterm Progress Report on

MALICIOUS WEBSITE DETECTION AND BLOCKING SYSTEM USING eBPF

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

RAVINDER KAUR
2104168

RANBIR SINGH
2104166

AMRINDER SINGH
2104066

JANUARY 2025

UNDER THE GUIDANCE OF
ER. KAMALJEET KAUR

GURU NANAK DEV ENGINEERING COLLEGE, GILL ROAD, GILL PARK, LUDHIANA

INDEX

CHAPTER: 1 INTRODUCTION

This project aims to design and develop a comprehensive system leveraging Deep Packet Inspection (DPI) to analyze network traffic, extract critical metadata, and enhance network management with secure and efficient tools. The proposed system will focus on extracting Server Name Indication (SNI) headers from ClientHello packets, enabling the identification of target servers in encrypted connections, a vital step in network traffic analysis. To ensure efficient data handling, we plan to integrate the system with a MySQL database, automating dependency installations to streamline operations. Compatibility with ARM-based Linux systems will be a priority, with tailored setup and build scripts designed to optimize performance on resource-constrained devices. Multithreading will be implemented to enhance the system's efficiency by enabling concurrent processing, and DPI plugins will be developed to provide deep insights into packet-level data, supporting advanced network monitoring, traffic categorization, and anomaly detection.

Automation and optimization are key aspects of the project, with planned tools such as Makefiles and shell scripts to simplify building, deployment, and configuration processes. The system will also include functionality for generating detailed data analysis reports to uncover valuable insights into network traffic patterns and trends, aiding decision-making and performance optimization. Domain whitelisting will be incorporated to address specific use cases, such as securing mail servers while maintaining high-security standards. Robust logging mechanisms and debugging tools will ensure the system operates smoothly, with quick identification and resolution of potential runtime issues.The deliverables for this project will include a well-structured and documented C++ codebase for packet analysis and database integration.

1.1 OVERVIEW

1.1.1 Project Objective To develop a system that utilizes Deep Packet Inspection (DPI) to analyze network traffic and extract meaningful data. Enable secure and efficient network management through advanced data handling and automation.

1.1.2 Key Features SNI Header Extraction: Extract server names from ClientHello packets. Database Integration: Store and manage network data using MySQL. ARM Compatibility: Ensure the system functions seamlessly on ARM-based Linux platforms. Multithreading: Enhance performance by implementing concurrent processing. Deep Packet Inspection (DPI): Analyze packet-level data for deeper insights.

1.2 SCOPE OF PROJECT

1.2.1 Network Traffic Analysis Inspect TCP packets and extract critical metadata, such as server names. Manage and analyze data flows effectively.

1.2.2 Database Management Transition from PostgreSQL to MySQL for better compatibility and performance. Automate dependency installations for smooth integration.

1.2.3 Deployment on ARM Systems Set up ARM virtual machines for development and testing. Create build scripts optimized for ARM architecture.

1.2.4 Deep Packet Inspection Introduce and develop DPI plugins for detailed packet analysis. Perform data analysis and generate comprehensive reports.

1.2.5 Automation and Optimization Create Makefiles for efficient building and deployment. Develop scripts for dependency automation and installation.

1.3 DELIVERABLES

1.3.1 Codebase Well-structured and documented C++ code for packet analysis and database management.

1.3.2 Deployment Tools Shell scripts for autoinstallation and automated setup. ARM build and deployment scripts.

1.3.3 Reports and Documentation Comprehensive data analysis reports. README and user guides for setup and usage.

1.3.4 Final Product A robust system capable of analyzing network traffic using DPI, integrated with database management, and optimized for ARM compatibility.

1.4 KEY FEATURES

- SNI Header Extraction This feature involves extracting Server Name Indication (SNI) headers from ClientHello packets in network traffic. It helps identify the target server for encrypted connections, a critical step in network traffic analysis.
- Database Integration The system stores and manages network-related data using a MySQL database. This integration ensures efficient data handling, with automated dependency installations for smooth operation.
- ARM Compatibility Designed to run seamlessly on ARM-based Linux systems, the project includes setup and build scripts specifically tailored for ARM architectures. This enables deployment on resource-constrained devices.
- Multithreading Multithreading functionality ensures efficient processing of network data by enabling concurrent operations, reducing delays, and improving overall system performance.
- Deep Packet Inspection (DPI) DPI plugins analyze packet-level data, providing deep insights into network activity. This enables advanced network monitoring, traffic categorization, and anomaly detection.
- Automation Tools The project includes Makefiles, shell scripts, and automated dependency installers to simplify the building, deployment, and configuration processes.
- Data Analysis and Reporting The system generates detailed data analysis reports, offering insights into network traffic trends and patterns. This supports decision-making and network optimization.
- Whitelisting Support Domains can be whitelisted for specific use cases, such as securing mail servers, ensuring smooth functionality while maintaining security protocols.
- Logging and Debugging Robust logging mechanisms and debugging tools ensure smooth operation, aiding in the identification and resolution of potential issues during runtime.

- Configuration Management The system reads and applies configurations from JSON files, allowing flexibility and scalability in managing network settings.

1.5 OBJECTIVES

Upon completion, the project aims to achieve the following objectives:

1. To detect websites based on Deep Packet Inspection.
2. To log the website traffic to a database.
3. To block websites using eBPF.

CHAPTER: 2 SYSTEM REQUIREMENTS

The system requires a Linux-based environment with support for ARM architecture, as the project is designed for deployment on both standard and resource-constrained devices. It needs a C++ compiler (e.g., GCC) and essential build tools like Make for compiling the code. MySQL is required for database management, along with dependencies for JSON parsing and socket communication. Additionally, a virtual machine setup (e.g., QEMU) is necessary for testing on ARM platforms.

2.1 Hardware Requirements

2.1.1 Development Environment:

- Nebero proprietary firewall based on Banana PI M7
- Laptop with i5 8GB RAM
- Network devices for testing.
- Cisco L1 switch.

2.1.2 Deployment Environment:

- **Processor**: Multi-core (4+ cores recommended).
- **Memory**: Minimum 8 GB RAM
- **Storage**: Minimum 100 GB SSD
- **Network**: High-speed internet connection with low latency for optimal cloud service performance.

2.2 SoftwareRequirements

2.2.1 Development Tools:

- **IDE**: Visual Studio Code (open-source).
- **Version Control**: Git with GitHub or GitLab (open-source platforms).Network devices for testing.

2.2.2 Database:

Mysql database with mysqlcppconn-dev library

2.3 Open Source Technologies

The following technologies are free to use and contribute to the cost-effectiveness of the project:

1 Deep Packet Inspection: NDPI, Netifyd

2 Filtering: eBPF with BTF support enabled in kernel

3 Database caching: Tommyds library for efficient caching.

4 RapidJson by tencent for JSON parsing

CHAPTER: 3 SOFTWARE REQUIREMENT ANALISYS

3.1 Problem Definition

3.1.1 Background

Modern networks face challenges in monitoring, managing, and analyzing encrypted traffic due to increasing security concerns and growing data volumes. Extracting meaningful insights from such traffic is essential for secure and efficient operations, especially for identifying anomalies and managing data flow in real-time.

3.1.2 Key Challenges

1. **Encrypted Traffic Analysis:** Difficulty in extracting critical metadata (e.g., server names) from encrypted connections.
2. **Cross-Platform Compatibility:** Ensuring the system works on both standard Linux and ARM-based environments.
3. **Performance Optimization:** Processing large volumes of network data efficiently using multithreading and optimized algorithms.
4. **Automation and Scalability:** Simplifying deployment and configuration processes for varied environments.

3.1.3 Motive

The objective is to create a robust system capable of extracting, analyzing, and managing network traffic data using Deep Packet Inspection (DPI) while maintaining compatibility with ARM-based systems and supporting automation for seamless deployment.

3.2 Modules and Functionalities

3.2.1 Packet Analysis Module

Functionality:

- Extract Server Name Indication (SNI) headers from ClientHello packets.
- Perform Deep Packet Inspection (DPI) to analyze network traffic.
- Handle socket communication and data parsing for real-time operations.

Purpose:

This module focuses on analyzing packet-level data, enabling the identification of encrypted traffic destinations and providing detailed insights for network management.

3.2.2 Database Management Module

Functionality:

- Store extracted metadata in a MySQL database.
- Support automated dependency installation for seamless integration.
- Provide a reliable interface for querying and managing stored data.

Purpose:

This module ensures that all extracted data is efficiently stored and retrievable, enabling smooth data analysis and reporting functionalities.

3.2.3 ARM Compatibility Module

Functionality:

- Compile and optimize the system for ARM-based Linux environments.
- Create and test ARM build scripts for deployment.
- Verify compatibility and performance on resource-constrained devices.

Purpose:

This module extends the system's usability to ARM-based platforms, ensuring adaptability for IoT and other lightweight devices.

3.2.4 Automation Module

Functionality:

- Generate Makefiles and shell scripts for automated builds.
- Automate dependency installation and configuration reading via JSON files.
- Provide scripts for deploying and managing the system across environments.

Purpose:

Automation reduces manual efforts and ensures consistent deployments across different platforms, saving time and avoiding errors.

3.2.5 Multithreading Module

Functionality:

1. Implement multithreading to process network data concurrently.
2. Optimize performance to handle high volumes of traffic efficiently.

Purpose:

This module enhances system speed and scalability, ensuring that data processing remains efficient even under heavy traffic loads.

3.2.6 Logging and Debugging Module

Functionality:

1. Provide comprehensive logging for system operations and DPI analysis.
2. Identify and resolve runtime issues with advanced debugging tools.

Purpose:

This module ensures system reliability and assists developers in maintaining and improving the software by identifying potential issues.

3.3 Summary

The software is designed to address the challenges of analyzing encrypted network traffic and managing data efficiently in various environments. Through its modular design, the system offers functionalities such as packet analysis, database management, ARM compatibility, automation, multithreading, and robust logging. Each module is crafted to ensure scalability, performance, and ease of deployment, making the project suitable for diverse use cases and platforms.

CHAPTER: 4 SYSTEM DESIGN

4.1 Project Design

4.1.1 Overview

The system is designed to analyze network traffic by extracting metadata, performing Deep Packet Inspection (DPI), and managing the processed data effectively. It is modular, allowing seamless integration of various functionalities such as packet analysis, database management, automation, and multithreading. The design prioritizes scalability, compatibility, and performance, ensuring the system operates efficiently on both standard Linux and ARM-based platforms.

4.1.2 Key Components

1. Packet Analysis Layer: Handles the extraction of SNI headers and performs DPI for detailed traffic inspection.
2. Database Management Layer: Manages the storage and retrieval of extracted metadata using MySQL.
3. Automation Layer: Includes scripts and Makefiles for automated builds and deployment.
4. Compatibility Layer: Ensures the system functions on ARM platforms with specialized build scripts.
5. Multithreading Engine: Enables concurrent processing of traffic data for improved performance.

4.2 Solutions

1. SNI Header Extraction: A dedicated module for parsing ClientHello packets to extract SNI headers, facilitating encrypted traffic analysis.

2. Efficient Data Storage: A MySQL database integrated to store network traffic data, ensuring quick retrieval and secure management.

3. Deep Packet Inspection (DPI): Plugins for detailed traffic analysis and anomaly detection, providing actionable insights.

4. ARM Compatibility: Specialized scripts and testing on QEMU to ensure seamless functionality on ARM-based devices.

5. Automation: Deployment scripts and Makefiles to simplify setup, reduce manual intervention, and ensure consistency.

6. Performance Optimization: Multithreading for concurrent processing, ensuring minimal latency and efficient traffic handling.

4.3 Design Diagrams

4.3.1 Block Diagram

This diagram will illustrate the overall structure of the system, including the flow of data between modules such as packet analysis, database management, and DPI plugins. Key components like the ARM compatibility layer and multithreading engine will also be highlighted.

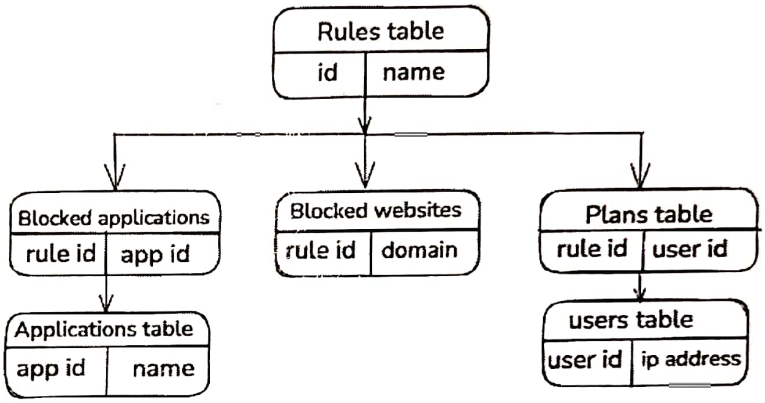

Fig.1 (Block Diagram for data segregation)

4.3.2 Data Flow Diagram (DFD)

The DFD will depict how data flows through the system, starting from the network packet capture stage to metadata extraction, storage in the database, and final analysis through DPI plugins.

DFD LEVEL0:

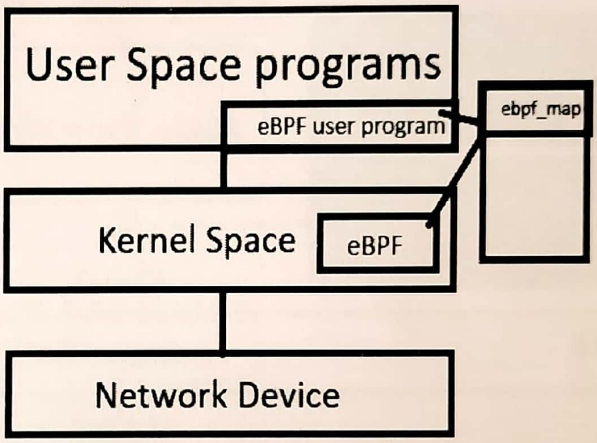

Fig. 2 (Data Flow Diagram)

4.3.3 Communication Diagram:

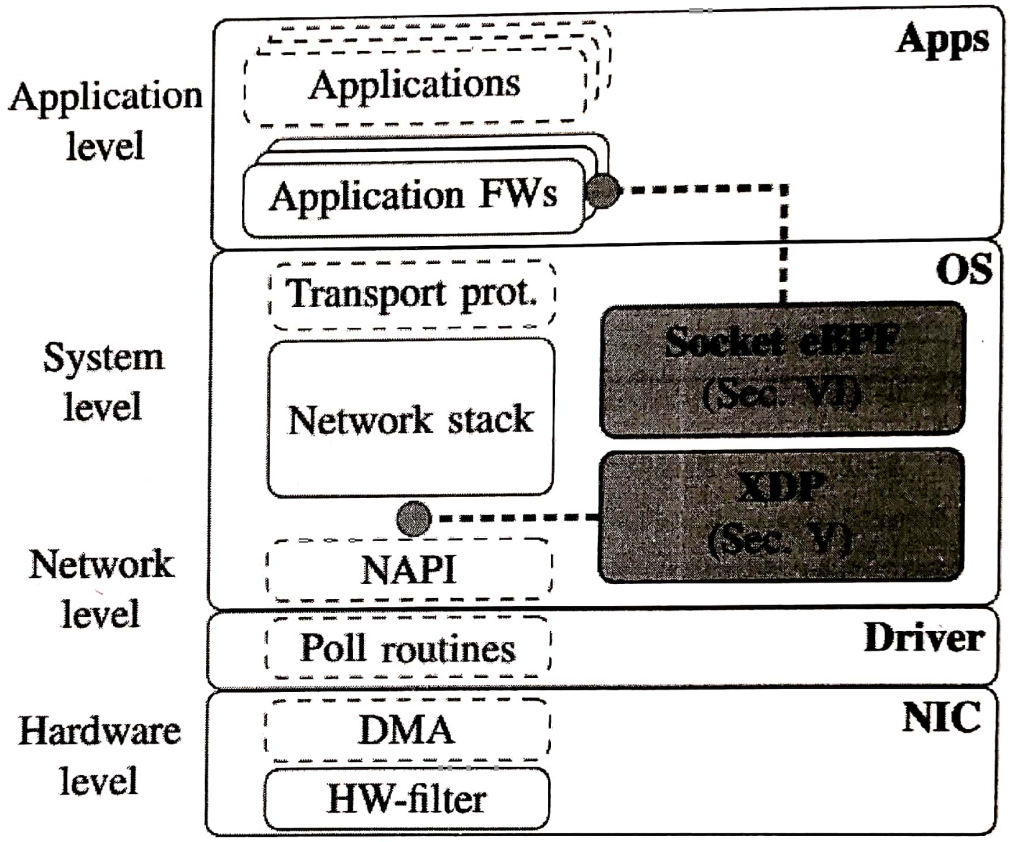

Fig. 3 (Communication Diagram)

CHAPTER: 5 TESTING MODULE

5.1 Overview of Testing

Testing was conducted to ensure the system met the functional and non-functional requirements of the project. Each module was tested independently before integrating them into the complete system. The process included simulating real-world network traffic, verifying database operations, and evaluating performance on ARM-based platforms. Special attention was given to ensuring compatibility, performance, and robustness of the Deep Packet Inspection (DPI) functionalities under various conditions.

5.2 Testing Phases

5.2.1 Packet Analysis Testing

To verify the accuracy of SNI header extraction from ClientHello packets:

1. **Method:** Simulated various TCP handshake scenarios using tools like Wireshark and custom packet generators.
2. **Test Case:** Injected packets with valid and malformed SNI headers to ensure the module could handle both correctly.
3. **Outcome:** The system accurately extracted SNI headers from all valid packets and ignored malformed ones without crashing.

5.2.2 Database Functionality Testing

To validate the MySQL database integration:

1. **Method:** Ran SQL queries to insert, update, and retrieve data during high-volume packet analysis simulations.
2. **Test Case:** Tested concurrent database transactions from multiple threads to check for deadlocks or data corruption.
3. **Outcome:** The database successfully handled concurrent operations without data loss or integrity issues.

5.2.3 ARM Compatibility Testing

To ensure the system's functionality on ARM-based devices:

1. **Method:** Deployed the system on a QEMU ARM virtual machine and a Raspberry Pi.
2. **Test Case:** Tested all features, including packet analysis and database operations, on the ARM platform.
3. **Outcome:** The system performed reliably on ARM, with no noticeable difference in functionality compared to standard Linux environments.

5.2.4 Multithreading Testing

To validate the efficiency and correctness of multithreaded operations:

1. **Method:** Simulated high network traffic loads with multiple concurrent threads processing data.
2. **Test Case:** Monitored CPU and memory usage to ensure efficient resource utilization. Checked for race conditions and thread safety.
3. **Outcome:** The multithreading implementation was stable, with no data corruption or significant performance degradation under heavy loads.

5.2.5 DPI Functionality Testing

To ensure accurate packet inspection and data analysis:

1. **Method:** Used a mix of real-world network traffic and synthetic data to test DPI plugins.
2. **Test Case:** Checked the system's ability to identify packet anomalies, categorize traffic, and generate detailed reports.
3. **Outcome:** DPI plugins provided accurate and actionable insights, with detailed logs for further analysis.

5.2.6 Automation Testing

To validate the reliability of scripts for building, deployment, and dependency installation:

1. **Method:** Ran the automation scripts on fresh Linux environments to simulate real-world deployment scenarios.
2. **Test Case:** Tested failure scenarios like missing dependencies or configuration errors.
3. **Outcome:** The scripts handled errors gracefully and provided clear feedback to the user, ensuring a smooth setup process.

5.3 Summary of Testing Results

All modules and functionalities of the system were rigorously tested to ensure they met the project's objectives. The testing process revealed a few minor bugs, such as race conditions in multithreaded operations and database connection timeouts under heavy loads, which were resolved promptly. Overall, the system proved to be stable, efficient, and reliable under diverse conditions, making it ready for deployment in real-world environments.

CHAPTER: 6.PERFORMANCE OF THE PROJECT DEVELOPED

6.1 Overview

The Malicious Website Detection and Blocking System using eBPF has been successfully developed and deployed in a real-time environment. The project aims to enhance network security by identifying and blocking access to malicious websites dynamically while maintaining minimal system overhead. The system currently focuses on monitoring and blocking HTTPS traffic by inspecting the Server Name Indication (SNI) header and comparing it against a list of known malicious or unwanted websites.

6.2 Functional Capabilities Developed

6.2.1 Network Packet Capture Using eBPF

The system utilizes **eBPF (Extended Berkeley Packet Filter)** to efficiently capture and analyze network packets in real time.

1. **Targeted Traffic:** The system is specifically designed to monitor **HTTPS traffic** for malicious content.
2. **Packet Interception:** The **XDP (eXpress Data Path)** program is used to intercept incoming and outgoing packets at the network interface level with minimal performance overhead.

6.2.2 Malicious Website Detection Mechanism

1. **Data Sources for Malicious Sites:** The system relies on **online threat intelligence sources** for real-time updates on known malicious websites. Additionally, users are given the option to manually **insert custom blacklists** of websites they wish to block.

2. **Detection Strategy:**
    - The system extracts the **SNI (Server Name Indication) header** from the TLS handshake and matches it against the malicious website database.
    - If a match is found, the system **blocks** access to the website before the connection is established.

6.2.3 Blocking Mechanism

The **XDP-based filtering approach** ensures that malicious sites are blocked at the packet-processing stage, preventing the connection before it reaches the user's system.

Advantages of XDP in eBPF:

- **Ultra-low latency filtering** (operates at the kernel level, reducing processing time).
- **Efficient network security enforcement** without significant CPU overhead.

6.2.4 Logging and Monitoring

The system currently logs network activity and tracks websites accessed by users. However, real-time alerts for users are **not yet fully implemented**.

1. **What is logged?**
    - Website visited (hostname extracted from SNI header).
    - IP address of the user accessing the site.
    - Time of access.
    - Data usage statistics.
2. **Future Enhancement:**
    - Implementing **real-time alerts** to notify users about blocked websites instead of manual log checking.

6.2.5 Web Interface Development

A **partially completed web interface** allows administrators to monitor network activity effectively.

1. **Current Features:**
    - Displays a list of visited websites per user.
    - Shows **IP addresses** of users accessing different websites.
    - Provides **pie charts** for:
        - Data usage per **hostname** (which sites consume the most data).
        - Data usage per **user** (which user is consuming the most bandwidth).
2. **Planned Improvements:**
    - A more interactive dashboard with **search and filtering** capabilities.
    - Real-time alerting system integrated into the interface.

6.3 Performance Evaluation

6.3.1 Resource Usage

The system has been tested for resource consumption and shows excellent efficiency.

1. **Memory Usage:** The program runs within **16 MB of RAM** under normal conditions, with a maximum usage of **50 MB** under heavy traffic.
2. **CPU Consumption:** The use of **XDP and eBPF ensures minimal CPU overhead**, making the system ideal for real-time network monitoring and blocking.

6.3.2 Deployment in a Real-Time Environment

The project has been **successfully deployed in a live network environment** and tested for functionality and performance.

Real-world Testing Observations:

1. Successfully captures network packets and logs visited sites.
2. Effectively blocks malicious websites based on the defined threat database.
3. Ensures **minimal impact on system performance** while filtering network traffic.

6.3.3 Strengths and Achievements

1. **High-speed packet filtering** using eBPF with **minimal latency**.
2. **Efficient HTTPS traffic analysis** using the **SNI header** for domain-based filtering.
3. **Flexible threat detection** mechanism with both **automatic updates** and **manual blacklist configuration**.
4. **Partially developed web interface** that provides **insightful data visualization** through logs and charts.
5. **Lightweight and scalable** solution, suitable for enterprise and personal network security.

6.4 Areas for Improvement and Future Work

While the project has achieved significant milestones, further improvements are planned:

1. **Real-time User Alerts:** Implementing **pop-up notifications** or **browser warnings** when a malicious site is detected.
2. **Enhanced Web Dashboard:** Adding search, filtering, and detailed analysis features.
3. **Machine Learning Integration:** Exploring **ML-based threat detection** for improved accuracy.
4. **Dynamic Threat Intelligence Updates:** Ensuring real-time automatic updates from multiple security sources.

6.5 Conclusion

The Malicious Website Detection and Blocking System using eBPF has demonstrated **high efficiency, low resource consumption, and effective blocking capabilities** in a real-time environment. With ongoing development, including enhanced user alerts and an advanced web interface, the project aims to provide a **comprehensive and intelligent** network security solution.

CHAPTER: 7 OUTPUT SCREEN

1. To detect websites based on Deep Packet Inspection.

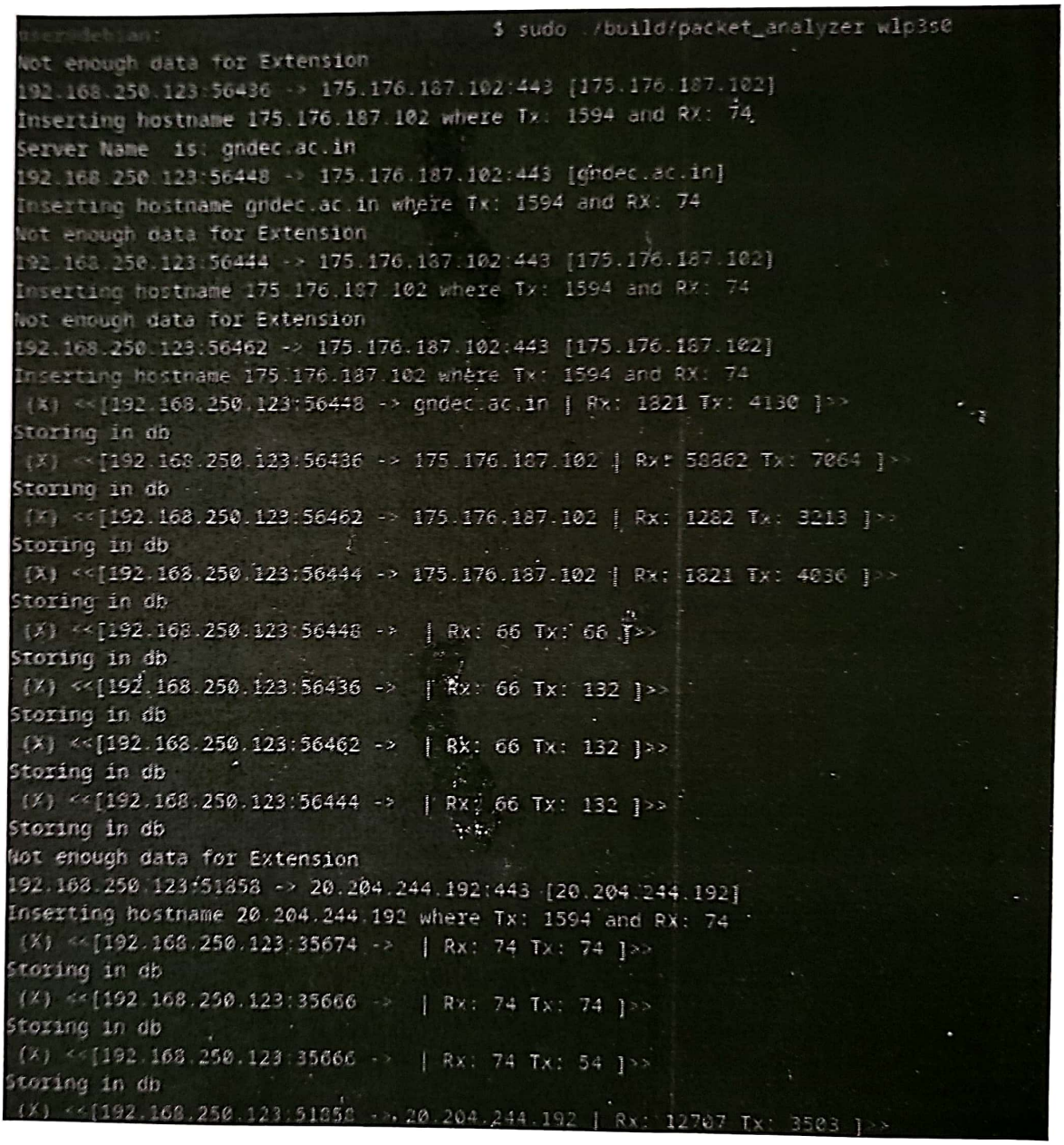

*Fig. 4 (*detection of websites based on Deep Packet Inspection)

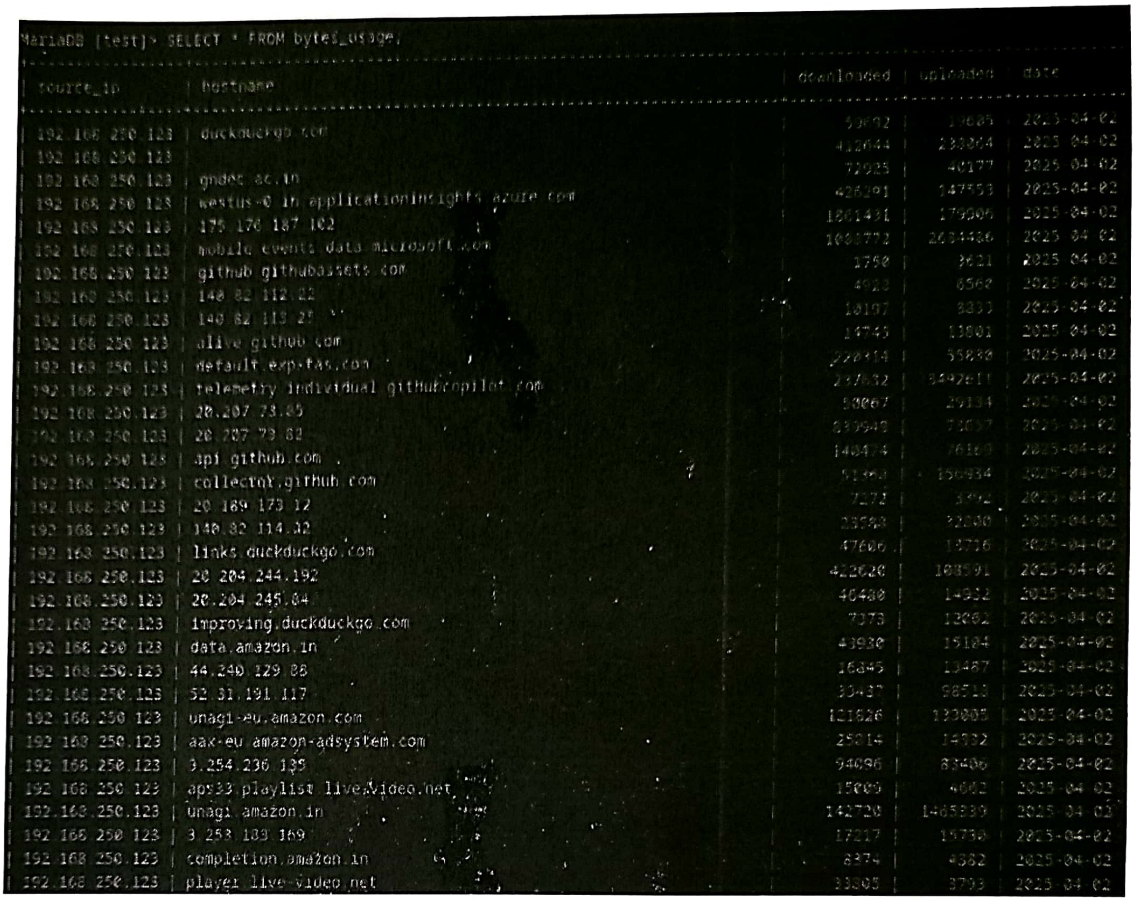

*Fig. 5 (*logging the website traffic to a database.)

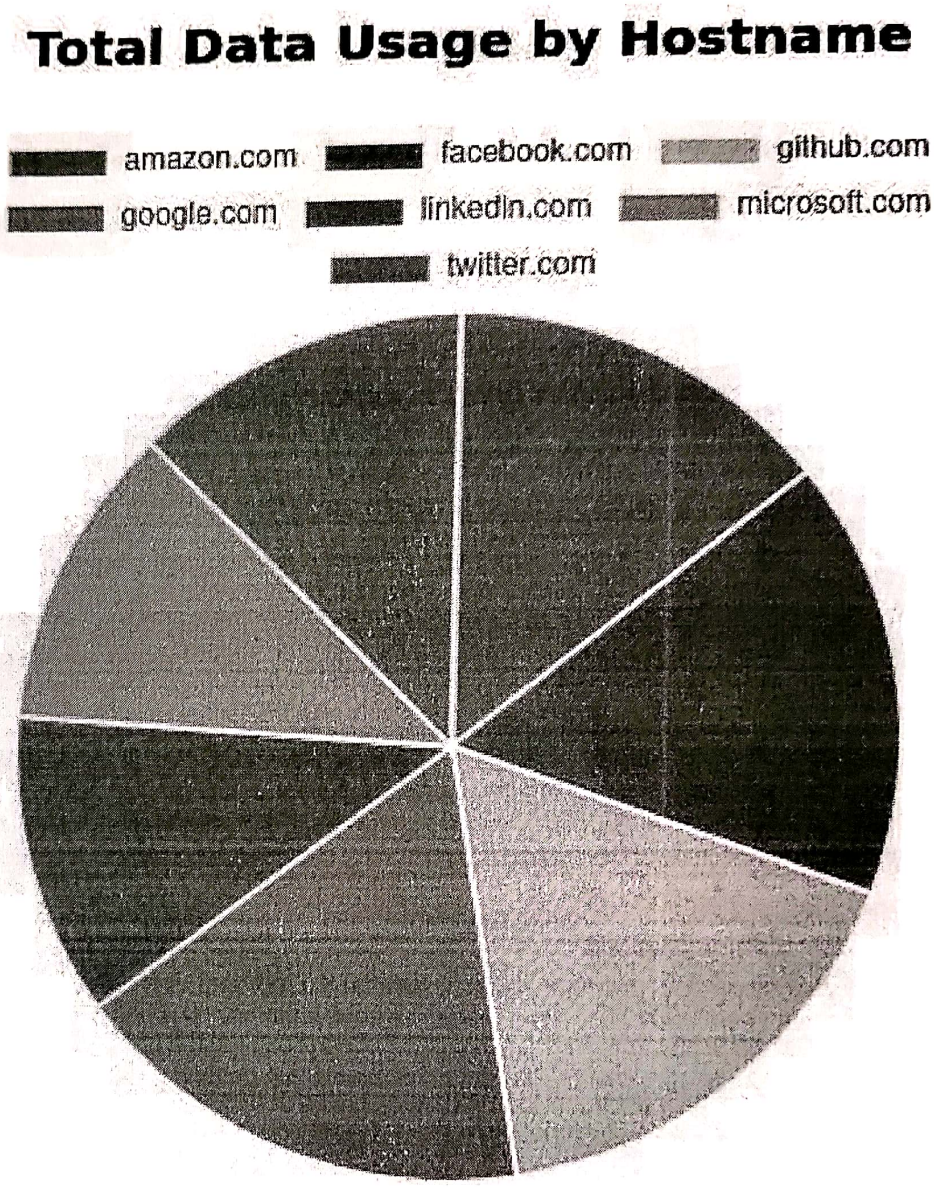

Fig. 6 (Total data used by hostname)

Network Data Usage

Source IP	Downloaded (Bytes)	Uploaded (Bytes)
192.168.1.10	2493937	191345
192.168.1.11	2703714	230863
192.168.1.12	2061713	297529
192.168.1.13	1827036	295061
192.168.1.14	2827280	275297

Fig. 6 (Network Data Usage)

Network Data Usage by Hostname

Hostname	Downloaded (Bytes)	Uploaded (Bytes)
amazon.com	1703670	176542
facebook.com	1923702	134556
github.com	2014813	143209
google.com	2074068	187653
linkedin.com	1357802	164197
microsoft.com	1359257	262962
twitter.com	1480368	220976

Fig. 6 (Network data used by hostname)

CHAPTER: 8 REFERENCES

1. D. Shamsimukhametov, A. Kurapov, M. Liubogoshchev and E. Khorov, "Early Traffic Classification With Encrypted ClientHello: A Multi-Country Study," in *IEEE Access*, vol. 12, pp. 142979-142993, 2024, doi: 10.1109/ACCESS.2024.3469730.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10697158&isnumber=10380310

2. R. Mohite and B. Thangaraju, "Enhancing Container Security with Per-Process Per-Container Egress Packet Filtering Using eBPF," *2024 International Conference on Electrical, Computer and Energy Technologies (ICECET*, Sydney, Australia, 2024, pp. 1-8, doi: 10.1109/ICECET61485.2024.10698563.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10698563&isnumber=10697986

3. D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak and G. Carle, "Performance Implications of Packet Filtering with Linux eBPF," *2018 30th International Teletraffic Congress (ITC 30)*, Vienna, Austria, 2018, pp. 209-217, doi: 10.1109/ITC30.2018.00039.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8493077&isnumber=8493038