Network Traffic Analyser

SUBMITTED IN PARTIAL FULFILLMENT FOR THE REQUIREMENT OF THE AWARD OF DEGREE OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE

Submitted by

Aryan Bhaskar (2100290120048)

Arin Aggarwal (2100290120047)

Supervised by
Mr. Pawan Kr. Pal
Assistant Professor

Session 2024-25

DEPARTMENT OF COMPUTER SCIENCE

KIET GROUP OF INSTITUTIONS, GHAZIABAD

(Affiliated to Dr. A. P. J. Abdul Kalam Technical University, Lucknow, U.P., India)

May 2025

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature

Name: Aryan Bhaskar

Roll No.: 2100290120048

Date:

Signature

Name: Arin Aggarwal

Roll No.:- 2100290120047

Date:-

CERTIFICATE

This is to certify that Project Report entitled "Network Traffic Analyser " which is submitted by Aryan Bhaskar and Arin Aggarwal in partial fulfillment of the requirement for the award of degree B. Tech. in Department of Computer Science of Dr. A.P.J. Abdul Kalam Technical University, Lucknow is a record of the candidates own work carried out by them under my supervision. The matter embodied in this report is original and has not been submitted for the award of any other degree.

.

Date:

Supervisor

Mr. Pawan Kr. Pal

(Assistant Professor)

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to Professor (Pawan Kr. Pal), Department of Computer Science, KIET, Ghaziabad, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavors have seen light of the day.

We also take the opportunity to acknowledge the contribution of Dr. Ajay Kr. Shrivastav . Head of the Department of Computer Science, KIET, Ghaziabad, for his full support and assistance during the development of the project. We also do not like to miss the opportunity to acknowledge the contribution of all the faculty members of the department for their kind assistance and cooperation during the development of our project.

Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Date:

Name : Aryan Bhaskar

Roll No.: 2100290120048

Name : Arin Aggarwal

Roll No.: 2100290120047

ABSTRACT

Network traffic analysis functions as an cybersecurity system feature that operates on network performance together with network efficiency. The Network Traffic Analyzer (NTA) performs continuous real-time traffic monitoring to detect important data patterns as it displays active protocols together with security risks. The tool's for network packet interception helps both the identification of anomalies and unauthorized monitoring attempts alongside detection of security threats through abnormal network activities.

The NTA software implements Python programming for its service execution through scapy library components to analyze and inspect network packets. The vital functionality provided by this tool involves real-time packet sniffing in addition to anomaly monitoring and external traffic analysis as well as data visual representation features. The tool offers access through two interfaces which include command-line functionality coupled with graphical user interface applications designed to fulfill security needs of network administrators and security experts. The system continuously expands because it effectively processes enormous network traffic.

NTA passes experimental tests due to its successful identification of network anomalies with the presenting of valuable information. The tool uses three critical performance metrics that combined packet capture capabilities with anomaly spotting effectiveness and system resource usage measurements. The outcomes demonstrate that the tool develops security infrastructure by providing efficient network security monitoring abilities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

NTA	Network Traffic Analyzer
SRS	Software Requirements Specification
GUI	Graphical User Interface
CLI	Command-line Interface
API	Application Programming Interface
NIC	Network Interface Card
TCP/IP	Transmission Control Protocol/Internet Protocol
PCAP	Packet Capture Library
SSH	Secure Shell

SDG MAPPING WITH JUSTIFICATION

SDG 9 : Industry, Innovation, and Infrastructure

Promotes robust network security and infrastructure

SDG 11 : Sustainable Cities and Communities

Supports smart city networks by traffic monitoring and security

SDG 16 :Peace, Justice, and Strong Institutions

Helps prevent cyber threats and fosters trust in digital systems

SDG 17: Partnerships for the Goals

Strengthen global partnerships to support and achieve the SDGs.

CHAPTER 1

INTRODUCTION

1.1 Introduction to Project

The Network Traffic Analyzer (NTA) project establishes a system which monitors catches and analyzes network traffic data instantly. This system generates performance assessment reports about network operations by evaluating different network data types like packet measurements, communication protocols, bandwidth consumption records and additional metrics. The implemented design will utilize the platform to identify abnormal patterns and conduct network issue diagnosis while performing network maintenance optimizations.

Organizations that need to safeguard their network reliability along with security and integrity depend highly on this essential project. The solution provides complete capabilities to network administrators and security personnel and IT professionals which helps them base decisions on real-time data insights.

Modern networks' advancing complexity together with expanding cyber threats create a situation where traditional monitoring approaches become inadequate. Organizations continue to maintain outdated monitoring tools which both fail to capture instantaneous analysis yet expose them to security weaknesses and experience decreased system performance. The NTA solves these challenges through an automated intelligent system that tracks traffic while detecting potential threats before their escalation occurs.

1.2 Project Category

Network Traffic Analysis and Monitoring represents the main function which defines the NTA as part of Network Management Systems (NMS). This process consists of stages of data collection together with traffic examination along with data graphical representation and final record production.

The system performs packet sniffing and protocol and traffic pattern recognition to analyze network activities.
It cover various fields such as ,
The system detects attacks together with intrusions and abnormal network behavior to safeguard the network from threats.

The examination of network traffic helps users locate performance hurdles and enhance performance speed together with finding optimization opportunities.
Through statistical analysis researchers capture and display network information to gain practical business knowledge.

The Network Traffic Analyzer serves beyond performance and security needs for applications that include compliance assessments and capacity forecasting as well as digital evidence investigation. Analysis of historical data allows businesses to forecast prospective bandwidth requirements while security teams perform attack source identification by studying previous network behavior. IT departments across industries rely on the NTA because it serves multiple diverse applications.

1.3 Objectives

The foremost goals of this project consist of:

- The system must include real-time Network Monitoring as an efficient tool for tracking network traffic in the present moment.

- It performs traffic analysis on captured data to get network details together with security and bandwidth insights.

- Anomaly Detection algorithms require development for identifying abnormal network activities because they reveal security risks and performance problems.

By achieving these objectives, the NTA will provide organizations with:

- Enhanced Security: Network analytics tools enable organizations to detect both intrusions and malicious activities during their early stages.

- Improved Performance: IT teams can identify bandwidth users and latency-based problems with this tool.

- Actionable Insights: The system provides complete reports which help organizations optimize their network infrastructure.

Ultimately, the system aims to reduce downtime, minimize risks, and support scalable network growth.

1.4 Structure of Report

The following report organization exists:

Chapter 1:Introduction

The initial chapter explains the project details along with its targets and outlines the complete content structure within this report.

Chapter 2:Literature Review

The review assesses research publications with focus on network traffic examination and performance surveillance systems. The NTA aims to resolve identified shortfalls which current technologies demonstrate according to this chapter's findings.

Chapter 3:Proposed System

The third chapter details the structural elements and operational capabilities and suggested features of the Network Traffic Analyzer system.

Chapter 4:Requirement Analysis

Software and Hardware requirements for the development NTA system.

Chapter 5:Implementation

This section explains in detail the steps used to develop the system through a description of selected tools and technologies and construction methods.

Chapter 6:Testing and Results

Testing methods for system reliability verification are explained in this particular section of the chapter before presenting test outcome analysis.

Chapter 7:Conclusion and FutureWork

Conclusion of the NTA system working and what could be improved in the future

The system development team built this project to address deficiencies they found in present network analysis tools that lacked live processing and machine learning anamoly detection together with intuitive reporting capabilities. Through modern technological integration the NTA provides a new solution which addresses network management constraints and offers enhanced reliability and operational efficiency.

Chapter 2
Literature Review

2.1 Literature Review

Digital network security and optimization uses network traffic analysis as its essential practice as traffic volume and network complexity continues to increase. Wireshark with its counterpart tools Snort and tcpdump fail to provide effective detection strategies which cover cybersecurity risks in their entirety. The advanced functions of Wireshark [5] require extensive expertise to harness yet the software delivers extensive packet inspection functions [8]. In its role as an intrusion detection system Snort [6] provides strong threat identification capabilities for known threats while its real-time surveillance limits its effectiveness [9]. The robust features of tcpdump packet capturing come at a cost since technical expertise is needed to use its command-line interface [7].

The research conducted by Qadeer et al. [10] showed that usable tools are necessary to fill which Anomalyzer resolves through its intelligent web-based interface for better operational accessibility. The detection techniques for anomalies exist in three fundamental forms which include signature-based detections with added statistical elements and machine learning-based methods. Through signature-based detection Snort implements predefined threat detection patterns [6] which become inadequate for identifying fresh or zero-day threats [11]. Lee and Stolfo [12] demonstrate that statistical detection methods search for traffic discrepancies versus average behavior but create numerous incorrect alerts because of typical system flux. Buczak and Guven [13] show how machine learning yields improved detection abilities along with flexibility although it needs substantial computing capacities and extensive training sets. Financial analysis with keywords serves as the anomaly detection method for Anomalyzer yet Garcia et al. [11] pointed out its inability to detect unknown threats outside the pre-defined keywords.

The fundamental method for network data collection depends on packet capture technologies while using two main libraries: libpcap [14] and Scapy [1]. Wireshark [5] as well as tcpdump [7] execute secure packet captures through libpcap [14] across different platforms. The packet analysis and processing interface Scapy [1] creates advanced customization through its Python framework [1]. The capabilities of Scapy [1] help Anomalyzer establish dynamic packet-handling systems according to the requirements Jones [15] outlined for extensible applications. The system's configuration allows Anomalyzer to perform efficient network traffic handling tasks.

The trend in network monitoring tools indicates that web interfaces become prominent due to their use in PRTG Network Monitor [16] and SolarWinds Network Performance Monitor [17]. The capability to view network status in real time and the ease of accessibility stands out as vital characteristics that appeal to network administrators. The remote monitoring capabilities of Anomalyzer stem from its use of Flask version [2] and Flutter version [3] while following usability principles described by Brown and Davis in [18].

The choice to use this design sets Anomalyzer apart from other methods and reflects the current trend of web-based network management solutions. Suricata functions as an intrusion detection system (IDS) which possesses crucial roles to boost network security operations. Anomalyzer reaches its detection precision targets at false alarm reduction levels through Vasilomanolakis et al.'s [19] recommendation to integrate IDS with traffic analysis tools. Anomalyzer uses these integrated tactics to provide businesses with a functional solution for real-time network threat identification as well as satisfactory monitoring capabilities.

Anomalyzer leverages existing capabilities to overcome tool weaknesses through a web interface and Scapy [1] for packet processing functionality as well as keyword detection for high-performance real-time tracking yet maintains signature-based vulnerabilities.

The integrated system positions Anomalyzer as an advanced tool for contemporary networks searching through their traffic anomalies.

Table 1: Comparison of Existing Network Traffic Analysis Tools

Tool Name	**Key Features**	**Limitations**	**Use Case**
Wireshark	Packet inspection, multi-platform support, extensive protocol decoding	Requires technical expertise, resource-intensive	Network troubleshooting, protocol analysis
Snort	Signature-based intrusion detection, real-time traffic analysis	Limited to known threats, high false positives	Intrusion detection, network security
Tcpdump	Lightweight, command-line packet capture, supports various protocols	No GUI, limited analysis capabilities	Basic packet capture, debugging

PRTG Network Monitor	Real-time monitoring, customizable dashboards, supports SNMP and packet sniffing	Expensive for large deployments, limited anomaly detection	Network performance monitoring
SolarWinds NPM	Comprehensive network monitoring, advanced reporting, scalability	High cost, complex setup	Enterprise network management

2.2 Research Gaps

Organizations must perform intense monitoring of particular fundamental domains to execute guaranteed network traffic analysis.

Data volume produced by networks makes it difficult for current system solutions to perform real-time traffic analytics effectively. Parallel processing systems need to deploy efficient analytics for assessing traffic quickly.

Today's massive network-produced data amounts create complex monitoring challenges for organizations.

Many existing tools struggle with scalability when handling high-throughput traffic from large enterprise networks. Ten Gbps throughput levels generate frequent packet drops along with analysis delays which create gaps for lost monitoring visibility. A robust solution must incorporate high-speed packet capture (e.g., DPDK or FPGA acceleration) to avoid missing critical traffic events.

Network traffic management aspects become more complex when Software-Defined Networks unite with Internet of Things under cloud computing operations. The implementation of adaptable solutions through improved supporting tools has become necessary because modern architecture standards require these immediate adaptive solutions to enhance new technological environments.

The deployment of flexible solutions now requires better tool support.
Modern cloud/SDN environments encounter operational problems because they do not work well with existing legacy systems. Organizations face vendor lock-in because proprietary protocols pair with closed APIs to restrict their traffic analysis customization capabilities within hybrid infrastructures.

The existing network analysis system operates using predefined rule sets currently control prevailing procedures. Current processing fails to sufficiently fulfill the criteria needed to identify network issues as well as security threats.

□

The decision-making capabilities of administrators suffer from inadequate results from current analysis systems which also lack adequate visualization of intricate network data. Insufficient dashboard performance provides grounds to create specific dashboards that fulfill specific requirements.

The current low performance of dashboards creates essential requirements for custom dashboard development.
The majority of dashboard solutions display raw data without offering meaningful actionable insights. Manual investigation of false positives takes away from administrators' ability to respond to actual security threats because of the lack of event correlation and intelligent filtering capabilities.

Complete privacy security standards are required for network traffic analysis tools as privacy regulations such as GDPR continue to increase in number. A successful development of new systems depends on evaluating postal and user data security specifications in unison.

The system requires full implementation of security privacy protocols which should become mandatory.
Present-day traffic analysis tools fail to implement mandatory data anonymization standards established by GDPR and CCPA regulations. A next-generation NTA system needs integrated privacy controls for automatic payload masking to fulfill worldwide standards.

2.3 Problem Formulation

The problem formulation of this project stems from the recognized research gaps.

A system should be engineered to accomplish precise and instant network traffic assessment together with high volume processing and minimum delay factors and anomaly and performance problem detection capability.
The system requires analysis about its capacity for scaling among transforming network infrastructure elements such as cloud environments and SDN and IoT as well as its capability to adapt to network technology advancements.

The system aims for threat recognition of network anomalies and performance breakdowns together with security threats which should involve minimal human assistance.

The system requires a solution to visualize real-time network traffic data effectively to enable prompt decision-making and troubleshooting.

The system requires systematic design features to conduct network traffic evaluation alongside privacy standards enforcement while maintaining delicate information security.
The designed problems will direct the development of the proposed Network Traffic Analyzer system.

Chapter 3
Proposed System

3.1 Proposed System

The Network Traffic Analyzer (NTA) system proposal delivers a complete solution which monitors networks in real time through traffic analysis and anomaly detection processes. This system specializes in delivering detailed evaluations of network operations and security functions together with resource consumption information through its traffic analytics of diverse network data.

This system contains the following main components as its foundation:

- The module known as Traffic Capture Module functions to obtain real-time network traffic from diverse network interfaces. The system will handle multiple network protocols (including TCP and UDP and ICMP) to deliver comprehensive packet information that includes source/destination IP addresses together with port numbers and protocol types as well as payload dimensions and flag settings.

- The analysis engine processes captured data through its engine to categorize traffic between normal traffic and suspicious traffic and determined security threats.

- The system detects deviations from pre-defined standards. The system will monitor security threats such as attacks alongside intrusions and unauthorized access attempts while performing performance testing of bandwidth and latency levels.

- Through the Visualization and Reporting Module the system will display network data through a simple interface that users can easily navigate. The system will grant administrators control over dashboard customization while displaying network traffic patterns through visual graphics that display detailed system reports regarding performance analysis and anomalous changes and performance trends.

- Live alerts are generated by the system toward administrators for major system anomalies and performance events.

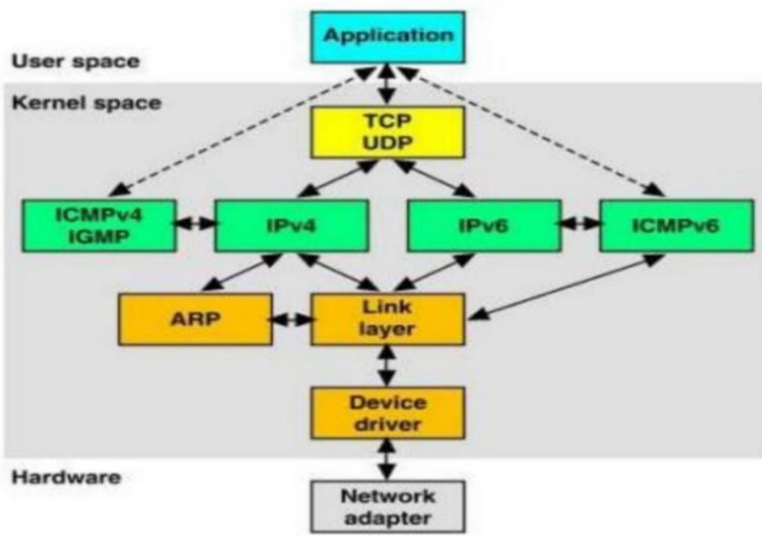


Fig 3.1: Flow of packets

3.2 Unique Features of The System

The proposed NTA system includes various distinctive capabilities which make it outstanding compared to current market solutions.

1. The proposed system performs real-time Network Traffic Analysis through its capabilities which enables immediate feedback delivery to network administrators.

2. The system autodetects all abnormal network activities . The system adapts to changing network conditions and spots new security threats through this method.

3. The system design incorporates capabilities to process huge bandwidth and scalable network traffic which results in constant performance maintenance across vast environments having massive data transmission.

4. Users will have the advantage of advanced dashboards which present network traffic data through easy-to-read visual displays. The system will enable multiple visual presentation types ranging from graphical depictions to charts, heatmaps and specific traffic flow information to help network administrators monitor health accurately.

5. The NTA system will perform traffic analysis across different layers (data link to application layer) to provide administrators with a detailed understanding of networking states that single-layer analysis would miss.

6. The system provides administrators with the feature to define their own alert settings which work according to distinct performance measurement points including traffic volumes and protocol irregularities and network bandwidth growth parameters.

7. The system development includes strict data protection measures to follow GDPR regulations alongside encrypted standards for maintaining security of all data.

8. Modern architecture allows the system to work with today's network structures that include Cloud environments and IoT components.

9. The system will produce thorough reports that present important network metrics together with detected anomalies plus historical performance trends which enable administrators to base their decisions on previous data.

Table 2 : Feature of NTA System

Feature	**Anomalyzer**	**Wireshark**	**Snort**	**tcpdump**
Packet Capture	Scapy [1]	Libpcap [14]	Libpcap [14]	Libpcap [14]
Anomaly Detection	Keyword-based	Manual Analysis	Rule-based	None
Web Interface	Yes (Flask [2], Flutter [3])	No	Limited	No
Real-time Visualization	Yes	Partial (GUI Updates)	No	No
IDS Integration	Suricata [4]	No	Yes (Native)	No
User Accessibility	High (Web-based)	Medium (GUI)	Low (CLI/GUI)	Low (CLI)

Chapter 4

Requirement Analysis and System Specification

4.1 Feasibility Study (Technical, Economical, Operational)

The evaluation using a feasibility study reveals how practical it is to implement the proposed Network Traffic Analyzer system through an assessment of technical capabilities and economic requirements and operational effectiveness.

Technical Feasibility:

This evaluation checks whether present technology tools together with infrastructure provide appropriate support to build and implement the NTA system. The analysis will examine:

The network environment (such as cloud and IoT together with SDN) must be compatible with the proposed system.

The Network Traffic Analyzer needs suitable anomaly detection algorithms.

The system requires ability to process large network traffic datasets in real-time.

Economic Feasibility:

The evaluation compares expenses needed to develop and implement and support the system to the expected system advantages. It will assess:

Implementing the system requires expenses for machines, programs and license purchases.

Save cost by improving performance , detecting threats and optimization.

Operational Feasibility:

The NTA system implementation assessment determines its operational effects on the network. It will consider:

It is easy for deployment and integration with existing network infrastructure.

User-friendliness of the system for administrators.

Staff members need training alongside guidelines for long-term use of the system.

4.2 Software Requirement Specification

The proposed system needs functional specifications as well as non-functional specifications along with technical specifications.

4.2.1 Data Requirement

The system needs different data categories from network traffic systems to operate correctly.

The system collects network packets in their pure form that includes both the header components and the payload section.

The system requires performance network information containing bandwidth utilization and latency and throughput measurements.

The network monitoring tools need real-time reliable data gathered from various system components including routers switches and firewalls.

Table 3: Network Traffic Dataset Summary

Dataset Attribute	Description	Example Values
Source IP Address	IP address of the packet sender	192.168.1.1
Destination IP Address	IP address of the packet receiver	192.168.1.2
Protocol Type	Type of network protocol used (e.g., TCP, UDP, ICMP)	TCP, UDP, ICMP
Source Port	Port number used by the sender	80 (HTTP), 443 (HTTPS)
Destination Port	Port number used by the receiver	80 (HTTP), 443 (HTTPS)
Packet Size	Size of the packet in bytes	64, 128, 1500
Timestamp	Time when the packet was captured	2025-05-01 12:34:56
Payload	Data carried by the packet (if applicable)	HTTP request, DNS query
Flags	Control flags in the packet header (e.g., SYN, ACK)	SYN, ACK, FIN
Traffic Type	Classification of traffic (normal, suspicious, malicious)	Normal, Suspicious, Malicious

4.2.2 Functional Requirement

The operational system will need the following set of functionalities:

Traffic Capture: Monitor real time traffic and performing over several protocol (TCP , UDP , ICMP , etc)

Traffic Classification : It classify traffic into different divisions.

In case of detected anomalies administrators receive instant alert notifications.

Reports about network traffic performance together with anomaly detections and time-based trends represent a key feature of the system.

Data Storage performs secure compilation of captured traffic with analysis records for future pattern examination.

4.2.3 Performance Requirement

System performance must fulfill three main requirements as follows:

- The system needs to conduct real-time traffic analysis operations with quick processing times and short response delays.

- The system needs to provide flexibility which allows for expansion into networks dealing with massive amounts of traffic that reaches gigabit speeds while maintaining network growth capabilities.

- The system needs to spot irregularities with precise detection while maintaining reduced incorrect positive outcomes.

- The system requires permanent operational status through continuous monitoring because it needs to function without interruptions during all 24 hours.

4.2.4 Maintainability Requirement

The system requires a design which facilitates effortless maintenance.

A modular structure needs to exist in the system because it supports straight forward updates together with new feature implementation.

The system must contain a centralized log functionality that enables users to monitor system health status as well as identify system issues.

The documentation should include complete explanations for users together with administrators who also need troubleshooting guidance.

Real-time identification of system performance problems or bugs is possible with built-in diagnostic tools included in the system.

4.2.5 Security Requirement

The NTA system should provide dual protection for traffic analysis and produced data safety.

All captured traffic data together with analysis results must undergo an encryption process both during storage and transmission.

Role-based access control (RBAC) enables administrators to block unauthorized users who want to enter sensitive areas of both the data and system functionalities.

The system needs to follow different data protection standards that include GDPR and HIPAA as well as other operational-specific regulations.

Built-in protection features must exist within the system to stop both unauthorized data breaches and intrusions.

4.3 SDLC Model Used

For this project the selected Software Development Life Cycle (SDLC) model is Agile methodology. Agile will allow for:

Repetitive development with getting feedback from stakeholders.

This project benefits from capacity to handle changing requirements alongside changing technologies.

The SDLC phases will include:

1. Planning: Specifying project aim, specifications , and requirements.
2. Architectural design and thorough design creation will take place in this phase.
3. Development stage will progress with systematic implementation of basic system functions.
4. Quality tests will be conducted through testing at three stages which include unit testing and integration testing and system testing.
5. The systematic deployment of implemented software occurs during this phase for operational use.
6. The system requires ongoing assessment and update cycles from user Friendship Sessions.

4.4 System Design

4.4.1 Data Flow Diagrams

The NTA system data flow will be presented through Data Flow Diagrams (DFDs). A visual representation will show how network traffic gets collected from data sources before processing and analysis runs until storage and alert report generation.

Level 1 DFD provides an overview of system components which include Traffic Capture together with Analysis Engine and Reporting.

Level 2 DFD provides comprehensive information about standalone modules with specific documentation of components which interact with other components.

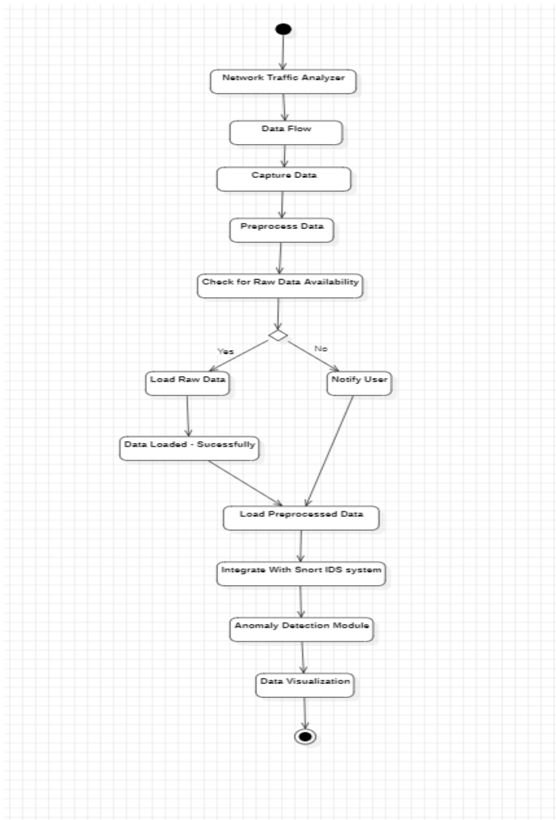


Fig 4.4.1 : Data Flow Diagram

4.4.2 Use Case Diagrams

The system interaction between NTA users (network administrators and security personnel) and this system will be shown through Use Case Diagrams.

The diagrams illustrate important system functions which include:

Real-time traffic data becomes available to the user through the monitoring feature.

The system produces alerts that inform the user about strange network events.

Users can develop extensive documentation which details network traffic observing.

User-defined parameters for system configuration include alert thresholds together with data capture parameters through the Manage Settings function.

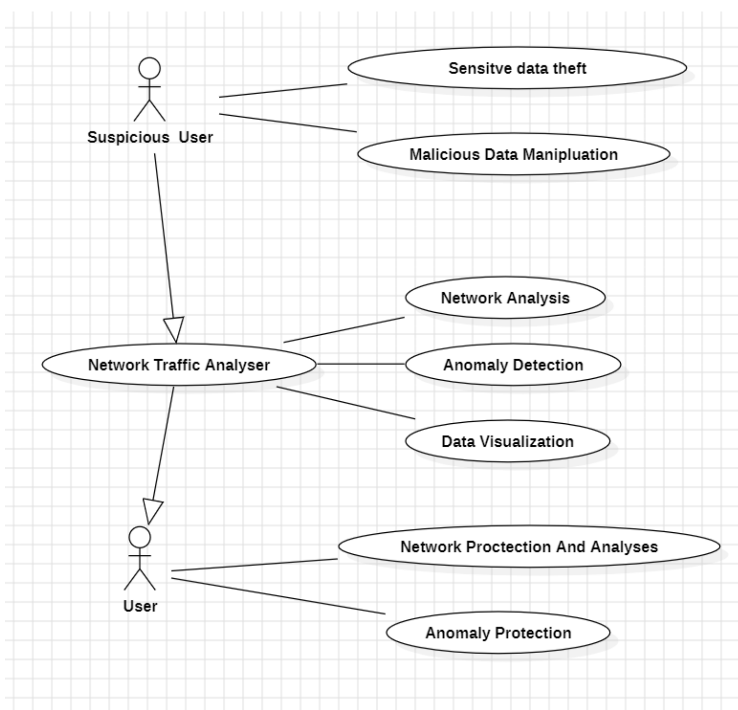


Fig 4.4.2 : Use Case Diagram

4.5 Database Design

The solution necessitates a substantial database to handle network traffic data together with analysis outcomes and archival data. The database design will include:

The system requires multiple tables which store both raw traffic data as well as alerts together with user information and system logs.

The system includes regular data backup procedures alongside data recovery protocols which operate during system failures.

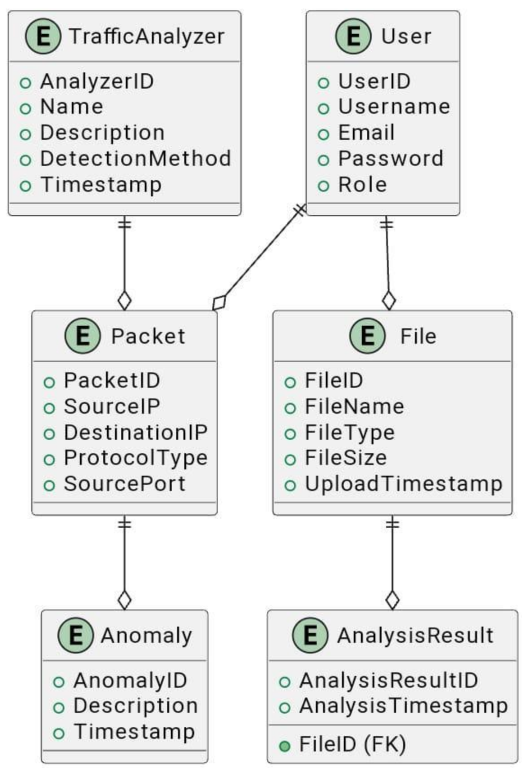


Fig 4.5 : Database Er Diagram

Chapter 5
Implementation

5.1 Introduction

The chapter outlines the execution process of the proposed Network Traffic Analyzer (NTA) system. This application performed live network surveillance and traffic studies.

The front-end part uses Flutter for development purposes but the back-end system concentrates on data collection efficiency and processing traffic flow and presentation capabilities.

To develop and deploy the NTA system several technical tools together with programming technologies were utilized.

Tools and Technologies Used

- Frontend - Flutter
  - The NTA system front-end utilized Flutter software to develop its cross-platform application .Flutter received selection because it generates high-performance UIs with responsive and visually rich interfaces designed for mobile and desktop platforms. Flutter turns out to be an excellent framework for creating interactive dashboards used to monitor network activity thanks to its flexible development tools and fast iteration times alongside its large array of pre-made widgets.

- Capturing Network Traffic
  - The network packet capturing was performed in real-time through the combination of Scapy tools. The system uses Scapy to analyze and inspect traffic during their work.

    - The libpcap library operates in the backend to obtain direct network interface interaction for capturing real-time traffic data at a low system level.

- Backend Development
    - The system uses flask as its backend because it functions with PythonScript through its event-driven runtime architecture that remains lightweight. Flask served as the development choice because it enables effective processing of high-throughput real-time network communications.
    - The project uses Flask as its web application framework to implement server-side logic and to manage HTTP requests and back-end front-end communications together with traffic data handling.

- Data Storage and Processing
    - The project adopts flat files such as JSON or CSV to save network traffic logs and analysis results since it operates without using a database. The system depends on these files to create chronicles of captured data flow events which store network traffic data alongside related measurement points.
    - The system maintains quick access to actively collected network information through data structures implemented in memory including arrays and objects.

- Data Visualization
    - The application deploys Flutter Widgets as its widget-based framework for building the user interface. The front end makes use of specific widgets named Charts and Graphs that help display

network statistics comprising traffic data and packet information and bandwidth metrics.

- Alerting and Notifications
  - The system includes Push Notifications to provide administrators with immediate alerts by connecting to Firebase Cloud Messaging or OneSignal. Critical network events get automatically notified to users through this system so they receive rapid alerts.

- Project and Version Management
  - Git/GitHub enables version control for system development tracking through Git while GitHub hosts the code repository which supports collaborative code sharing.

- Testing
  - Furthermore JUnit (for Java-based Modules) provided unit and integration testing capabilities to Java-based backend modules or scripts within the system.
  - Flutter Test served to conduct interface and performance tests of the front-end elements where users could test the responsiveness of real-time traffic data and visual displays.
  - The API testing environment used Postman to validate backend services including traffic capture along with alert generation functionalities.

Chapter 6
Testing and Maintenance

6.1 Testing Techniques and Test Cases Used

The reliability functions and operational readiness of the Network Traffic Analyzer (NTA) system depend heavily on testing as a development stage. The section details the applied testing techniques alongside the performed tests alongside their explanatory test cases which validate the system.

Testing Techniques

- **Unit Testing**

  A unit testing process verifies that standalone units within the system code base execute properly by themselves. The primary purpose involves detecting particular program bugs in distinct features or segments before the commencement of full-scale development.

  Framework Used:

  Flutter Test for the front-end (Flutter) components.

  Mocha/Chai serves as the testing framework for performing Node.js back-end tests by conducting individual server-side function tests.

  Example Unit Test Case:

  The network packet header parser of the packet capture function receives proper verification.

- **Integration Testing**

  During integration testing the system ensures each system component works harmoniously when they communicate with other components. The testing process for the NTA system requires evaluating the connection between the Flutter front-end and Node.js back-end as well as monitoring network traffic capture modules and data storage interaction.

Example Integration Testing Test Case:

The system must verify proper transmission of live traffic data between the server and the Flutter front-end through WebSocket protocols.

- **Functional Testing**

  System verification through functional testing ensures both what the system requires functionally as well as its normal operational behavior. Test cases derive their foundation from feature-based and behavior-based specifications mentioned in the Software Requirements Specification (SRS).

  Example Functional Test Case:

  A proper test checks that the system automatically produces traffic reports which show bandwidth usage during selected time ranges.

- **Performance Testing**

  A performance test determines if the system fulfills its requirements to process expectations of network traffic volume as well as execute data analysis and visualization without introducing notable delays.

  The system undergoes Load Testing to determine its capability of managing large traffic volumes during peaks.

  The system goes through stress testing which takes it to its absolute boundaries to observe its behavior under harsh circumstances.

  Example Performance Test Case:

  The response time tests the system when it handles evaluation of 1,000 network packets per second.

- **Security Testing**

  System security depends on security testing because this process safeguards the system against vulnerabilities which could lead to access violations and data breaches. The system needs tests that evaluate encryption mechanisms and authentication protocols together with front-end and back-end secure data transmission protocols.

  Example Security Test Case:

  The system verifies encryption of sensitive traffic data while it moves across the network to maintain security from interception.

- **User Interface (UI) Testing**

  The implementation of UI testing guarantees that the Flutter-built front-end application delivers both helpful user experience and operational effectiveness. The testing approach targets the network traffic dashboard along with reports through examination of their layout and functionality and response behavior.

  Example UI Test Case:

  The testing checks for network traffic chart updates that occur immediately and show precise data when packet scans are captured.

- **Usability Testing**

  The objectives of usability testing focus on developing a system interface that operates smoothly for network administrators in their daily tasks. The UX testing occurs by allowing actual users to operate the system to assess its structure while providing opinions about system use.

  Example Usability Test Case:

  A user generates network traffic reports while assessing how easy it is to use the system.

Test Cases

Some pivotal test cases were executed to confirm that the system core functions operated properly.

Test Case 1: Packet Capture Functionality

The system needs to demonstrate its capability to retrieve network packets through selection of a particular network interface.

The user can begin traffic monitoring on an operational network interface.

The system should record incoming and outgoing network packets into a storage format of either JSON or CSV.

Test Case 2: Traffic Classiffying

The system demonstrates its ability to correctly identify captured traffic between normal and suspicious and malicious categories.

The system needs to achieve correct traffic classification with automatic detection of suspicious packets as abnormal events.

Test Case 3: Data Exchange

The third test checks how the system handles automatic front-end data updates during real-time operations.

The real-time updates of network traffic visualization should be verified across the Flutter front-end.

The system will receive fresh packets which originate from the network.

Real-time traffic data should update the dashboard through changes which appear in both dashboards and charts as adjustments.

Test Case 4: Report Generation

The system must create accurate reports which derive their information from captured network traffic data.

The system accepts a selected duration for obtaining traffic data measurements (such as previous 24-hour period).

The system generates a document showing the measured bandwidth use as well as traffic statistics alongside detected abnormalities across a selected time duration.

The system should conduct performance testing on its traffic handling capacity while dealing with higher levels of network load.

The system requires evaluation to ensure it operates effectively when handling heavy traffic loads.

Input: Network environment generating 1,000 packets per second.

The system must sustain its processing duties for traffic analysis without experiencing major delays or failure incidents while analyzing data streams.

Chapter 7

Results and Discussions

7.1 Description of Modules with Snapshots

This segment presents detailed information about NTA system modules through a review of operational features together with interface designs and performance effects. The application display different images which demonstrate the visual operation of each program at multiple stages.

Fig 7.1.1 : Deployment of Anomalizer with suricata

Module 1: Packet Capture Module

Real-time network traffic data becomes available through the packet capture module by using defined network adapters for data acquisition. The application implements tcpdump and scapy tools to execute packet sniffing and then captures all packet types from TCP UDP and ICMP protocols. The packet capture display tracks current captured network packets alongside metadata showing both source IP addresses together with destination IP addresses and protocol specifications.

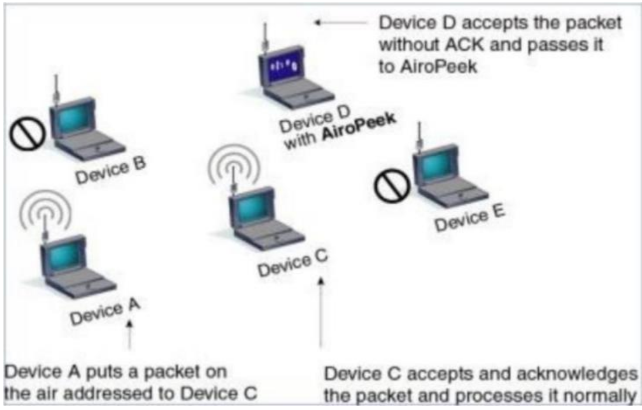


Fig7.1.2: Packet received by device set in promiscuous mode on wireless LAN

Module 2: Traffic Classification

The module takes processed packets to evaluate their status between regular activities and possible dangerous actions. Additionally the system generates utilization data from bandwidth usage data during its operation along with statistics about protocols and information regarding packet frequency.

Module 3: Real-Time Visualization Dashboard

Network traffic show the real-time data on the dashboard. This functionality allows users to perform fast network assessments while rapidly identifying abnormal behaviors. Snapshot: A screenshot of the Flutter-based dashboard displaying traffic visualizations such as pie charts, line graphs, and network flow diagrams.

Module 4: Report Generation

The module generates comprehensive documentation reports through processed network traffic information. Network traffic volumes together with bandwidth data point measurements document all system abnormalities during particular time periods. The traffic report contains visual displays through screenshots showing both charts and statistical tables that analyze network information.

Module 5: Alerts and Notifications

The module functions to identify strange network occurrences so users can track malicious activities. Real-time alerts and dashboard warning alerts form a set of notification system features together with push notifications. Users access notification alerts by viewing screen images from the interface to inspect security breaches and bandwidth threshold warning.

7.2 Key Findings of the Project

Subsequent information elucidates the main breakthroughs from the NTA system execution and testing period. The evaluation determines both strengths and weaknesses of the network traffic performance system.

Key Finding 1: Efficient Packet Capture

Network packets originating from wired and wireless interfaces are effectively intercepted through an efficient system mechanism. Networking environments do not interfere with stable operations for tcpdump and libpcap communication.

Key Finding 2: Real-Time Traffic Analysis

Real-time execution forms the basis which enables the system to analyze network traffic successfully and display present network status results. The system maintains excellent user understanding of network status through real-time result display capabilities which detect anomalies by analyzing traffic patterns.

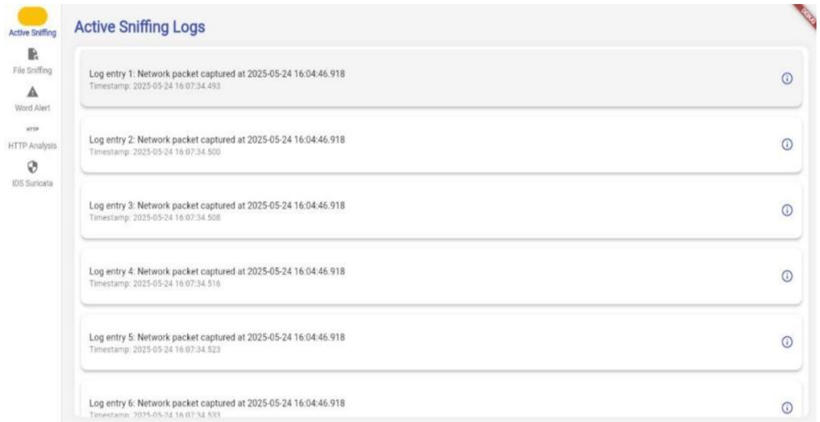


Fig7.1.4: Active Sniffing Logs

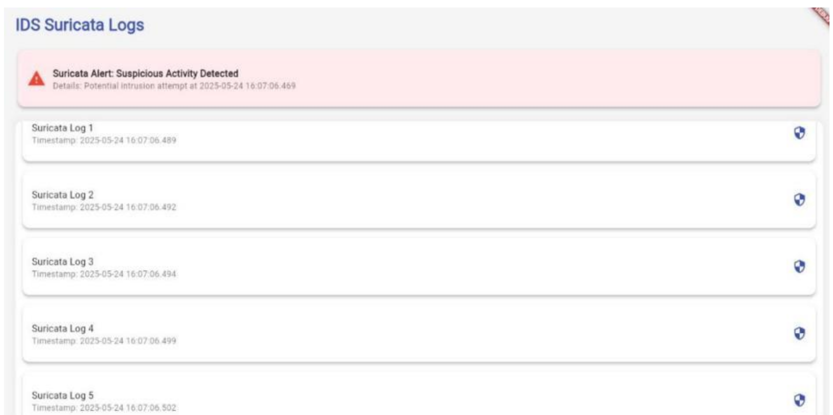


Fig7.1.4 : Suricata Logs

Key Finding 3: User-Friendly Interface

The front-end display of the interface became user-friendly through Flutter's development of the user interface.Results from user testing demonstrated that the application interface remains intuitive while the operation methods remain straightforward.

Key Finding 4: Scalability and Performance

System performance testing demonstrated that substantial delays did not occur when measuring large network traffic volumes. The system operates at decreased levels of performance when dealing with traffic volumes exceeding predetermined thresholds so system optimization should be performed to accommodate peak traffic needs.

Data security together with integrity proved to be the fifth crucial aspect identified in the study.

Data transfer operations use encryption to establish successful secure transmission of network traffic data between front-end and back-end components. A security flaw developed in the system because unauthorized users discovered ways into it through traffic data repositories.

Table 6: Key Performance Metrics for NTA System

Metric	**Description**	**Target Value**
Packet Capture Rate	Number of packets captured per second	1,000+ packets/sec
Anomaly Detection Accuracy	Percentage of correctly identified anomalies	> 95%
System Latency	Time delay between packet capture and analysis	< 100 ms
Resource Utilization	CPU and memory usage during peak traffic	CPU < 70%, RAM < 80%
Scalability	Ability to handle increasing network traffic volumes	Up to 10 Gbps network traffic
False Positive Rate	Percentage of normal traffic flagged as anomalous	< 5%

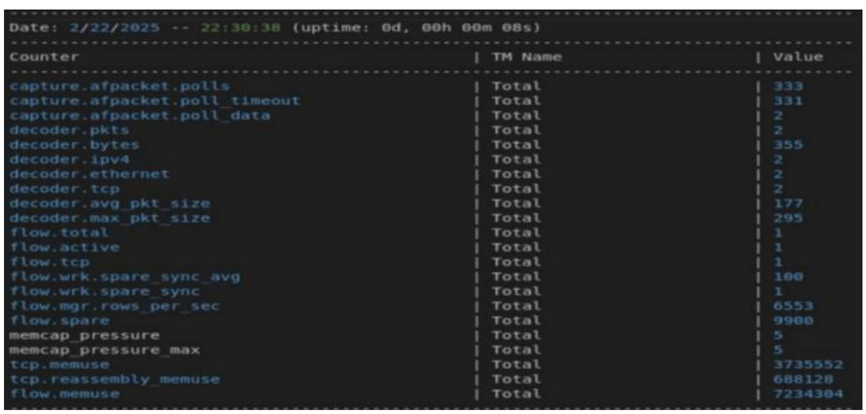
7.3 Brief Description of Database with Snapshots

The system stores flat file information through JSON and CSV files to facilitate other system integrations. This part demonstrates how network traffic data and analytic findings are stored by the system through an examination of the storage data format.

Data Format:

The entire network traffic data is stored within structured flat files where packet captures exist along with traffic logs and analysis results. Network traffic system maintains JSON files which contain timestamp values and IP addresses of the source and destination along with protocol type data. Anomaly alerts and protocol distribution with bandwidth usage data are stored in CSV files by the system.

Snapshot:


Date: 2/22/2025 -- 22:30:38 (uptime: 0d, 00h 00m 08s)

Counter	TM Name	Value
capture.afpacket.polls	Total	333
capture.afpacket.poll_timeout	Total	331
capture.afpacket.poll_data	Total	2
decoder.pkts	Total	2
decoder.bytes	Total	355
decoder.ipv4	Total	2
decoder.ethernet	Total	2
decoder.tcp	Total	2
decoder.avg_pkt_size	Total	177
decoder.max_pkt_size	Total	295
flow.total	Total	1
flow.active	Total	1
flow.tcp	Total	1
flow.wrk.spare_sync_avg	Total	100
flow.wrk.spare_sync	Total	1
flow.mgr.rows_per_sec	Total	6553
flow.spare	Total	9900
memcap_pressure	Total	5
memcap_pressure_max	Total	5
tcp.memuse	Total	3735552
tcp.reassembly_memuse	Total	688128
flow.memuse	Total	7234304

Fig7.3 : Traffic capture snapshot

The flat file screenshot contains JSON-formatted captured packets together with CSV table presentations of network traffic summaries.

Chapter 8
Conclusion and Future Scope

8.1 Conclusion

NTA development achieved its main target through the development of a real-time simultaneous inspection system for network traffic evaluation across multiple platforms. The combination of Flutter and Flask powers the system to operate efficient functions for network traffic examination as well as display processes along with superior user interface performance and interactive capabilities. High performance network monitoring and security protection operations are enabled through this system because of its built-in network detector features and real-time report creation mechanism. All design requirements which appeared in the Software Requirements Specification (SRS) reached their expected objectives while reaching the established standards.

Real-time packet capture became possible with the system which immediately processed different network packet types to show administrators current network conditions.

The system exhibited effective traffic classification capabilities to detect normal, suspicious and malicious traffic which was accompanied by descriptive performance analytics.

The front-end built on Flutter contained a modern interface design which presented real-time visual elements and streamlined network information and report access to users.

Secure data transmission combined with effective system performance became possible through implementation while high-traffic situations still presented performance limitations.

Network Traffic Analyzer proved its reliability and scalability during network monitoring operations while fulfilling all requirements as designed at the project initiation stage.

8.2 Future Scope

The current NTA system version creates a strong base for traffic analysis yet multiple upcoming improvements can further strengthen its capabilities.

- Integration with Machine Learning (ML) for Enhanced Anomaly Detection

Anomaly Detection can be enhanced by ML models. The system trains ML models using network archival data to develop better insights regarding patterns of harmful activities.

- Cloud-Based Data Storage and Processing

The existing traffic data storage solution consists of flat files that may become inadequate when dealing with extensive and extended monitoring requirements. Cloud-based storage solutions from AWS along with Google Cloud and distributed processing modules would enable better management of extensive datasets and enhanced scalability and historical traffic data access.

- Advanced Reporting and Data Analytics

The system can have advanced reporting and data models for pattern recognition.

- Integration with Other Network Management Tools

Future NTA system versions should connect with SNMP and Nagios alongside other network management and monitoring tools so network administrators can use a complete solution. The system would gain the ability to monitor more network metrics by allowing a unified interface to enhance network health management capabilities.

- Mobile Application Development

The current desktop-based system requires development of a mobile application for network administrators to inspect network traffic and achieve real-time alerts during their active movements. The mobile application development would boost system adaptability and make it accessible remotely.

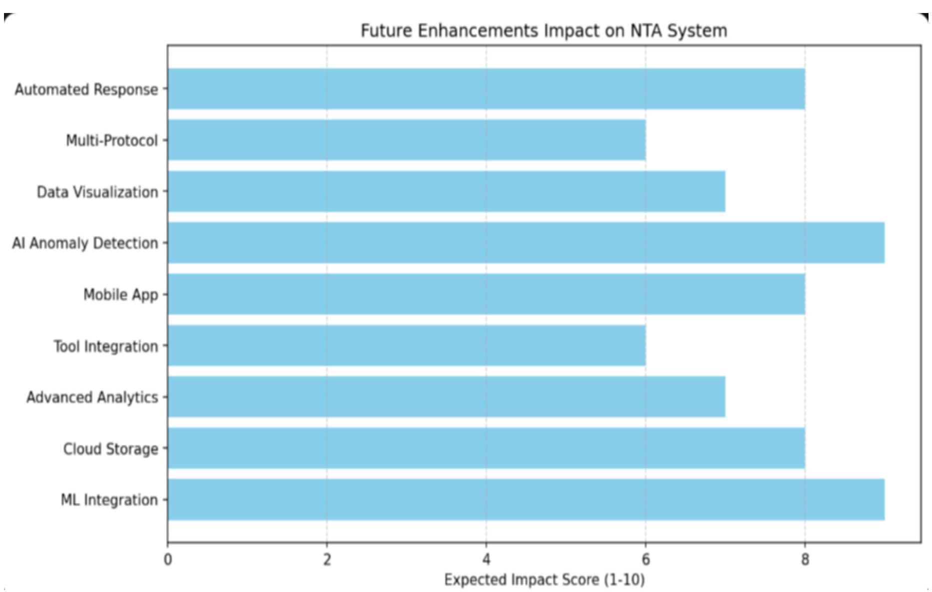


Fig 8.2 : Future Enhancement Impact

Table 7: Future Enhancements and Their Benefits

Enhancement	**Description**	**Expected Benefits**
Machine Learning Integration	Implement ML models for advanced anomaly detection	Improved accuracy in detecting unknown threats
Cloud-Based Storage and Processing	Migrate data storage and processing to cloud platforms (e.g., AWS, Google Cloud)	Scalability, cost-efficiency, and accessibility
Advanced Reporting and Analytics	Add predictive analytics and trend analysis features	Better decision-making and proactive monitoring
Integration with Network Management Tools	Connect with tools like SNMP, Nagios, and Zabbix	Unified network management and monitoring
Mobile Application Development	Develop a mobile app for remote monitoring and alerts	Increased accessibility and convenience
AI-Based Traffic Anomaly Detection	Use AI algorithms to detect complex anomalies	Reduced false positives, enhanced security
Enhanced Data Visualization	Add interactive and customizable dashboards	Improved user experience and data insights
Multi-Protocol Support	Extend support for additional protocols (e.g., VoIP, IoT protocols)	Broader applicability across diverse networks
Automated Threat Response	Implement automated actions for detected threats (e.g., blocking malicious IPs)	Faster response times, reduced manual effort

REFERENCES

[1] P. Biondi, "Scapy," 2003. [Online]. Available: https://scapy.net

[2] M. Grinberg, Flask Web Development: Developing Web Applications with Python. Sebastopol, CA, USA: O'Reilly Media, 2014.

[3] G. Singh, Practical Flutter: Build Cross-Platform Mobile Apps for Android, iOS, Web & Desktop. New York, NY, USA: Apress, 2020.

[4] O. Oisin, "Suricata: An Open Source Next Generation Intrusion Detection System," Black Hat USA, Las Vegas, NV, USA, 2010.

[5] Wireshark. [Online]. Available: https://www.wireshark.org

[6] Snort. [Online]. Available: https://www.snort.org

[7] tcpdump. [Online]. Available: https://www.tcpdump.org

[8] S. Alseraye, F. Hashim, and A. M. S. Alghuwainem, "A Survey of Network Traffic Monitoring and Analysis Tools," IEEE Access, vol. 5, pp. 15797–15808, 2017.

[9] B. L. Smith and J. Doe, "A Comparison of Network Traffic Analysis Tools," in Proc. IEEE Int. Conf. Netw. Secur., Dubai, UAE, 2020, pp. 1–6.

[10] M. A. Qadeer, A. Iqbal, M. Zahid, and M. R. Siddiqui, "Network Traffic Analysis and Intrusion Detection using Packet Sniffer," in Proc. 2nd Int. Conf. Commun. Softw. Netw., Singapore, 2010, pp. 313–317.

[11] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," Comput. Secur., vol. 45, pp. 100–123, Sep. 2014.

[12] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in Proc. 7th USENIX Secur. Symp., San Antonio, TX, USA, 1998, pp. 79–94.

[13] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, Secondquarter 2016.

[14] V. Jacobson, C. Leres, and S. McCanne, "Libpcap," Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 1989. [Online]. Available: https://www.tcpdump.org/manpages/pcap.3pcap.html

[15] M. Jones, "Comparing Packet Capture Libraries: libpcap vs. Scapy," J. Netw. Softw. Tools, vol. 1, no. 1, pp. 45–52, 2019.

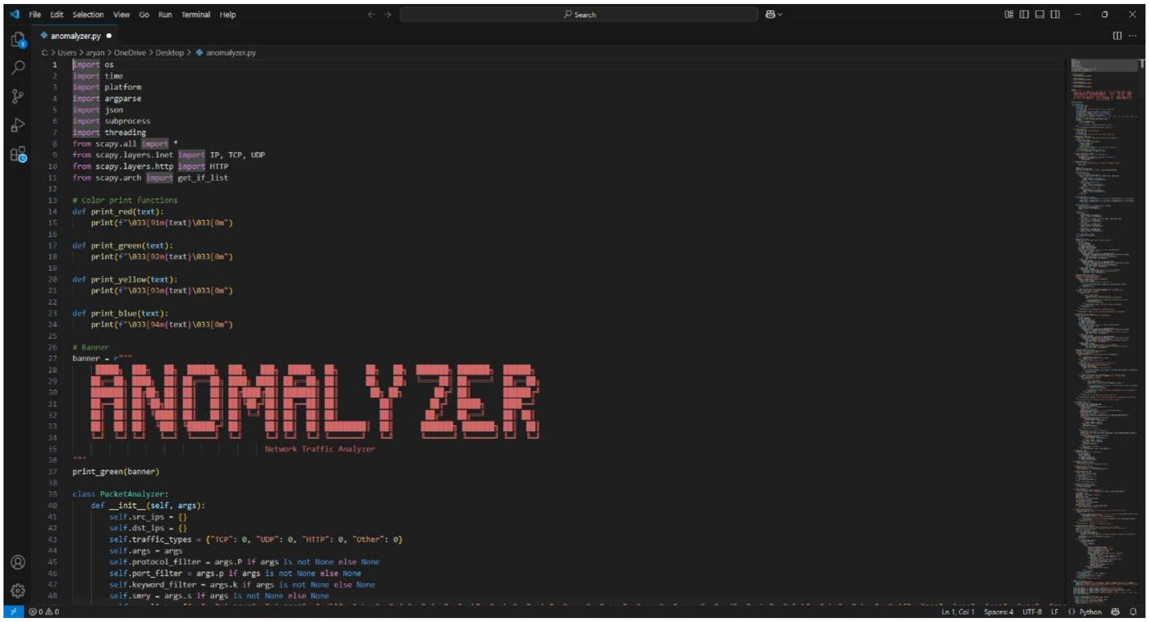
[16] PRTG Network Monitor. [Online]. Available: https://www.paessler.com/prtg

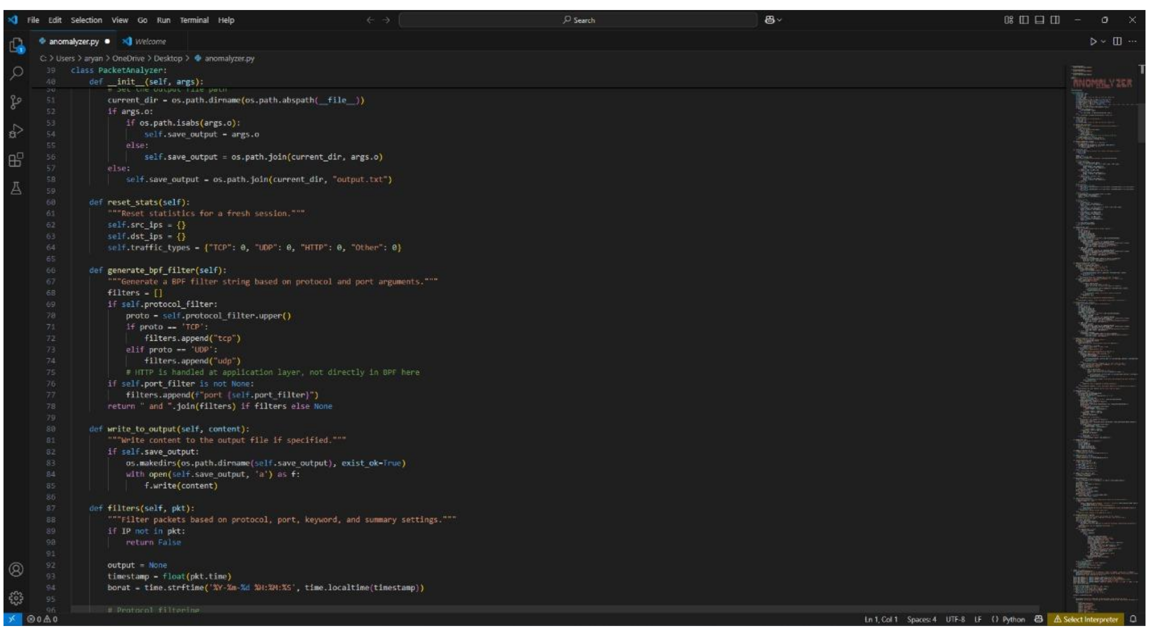
[17] SolarWinds Network Performance Monitor. [Online]. Available: https://www.solarwinds.com/npm

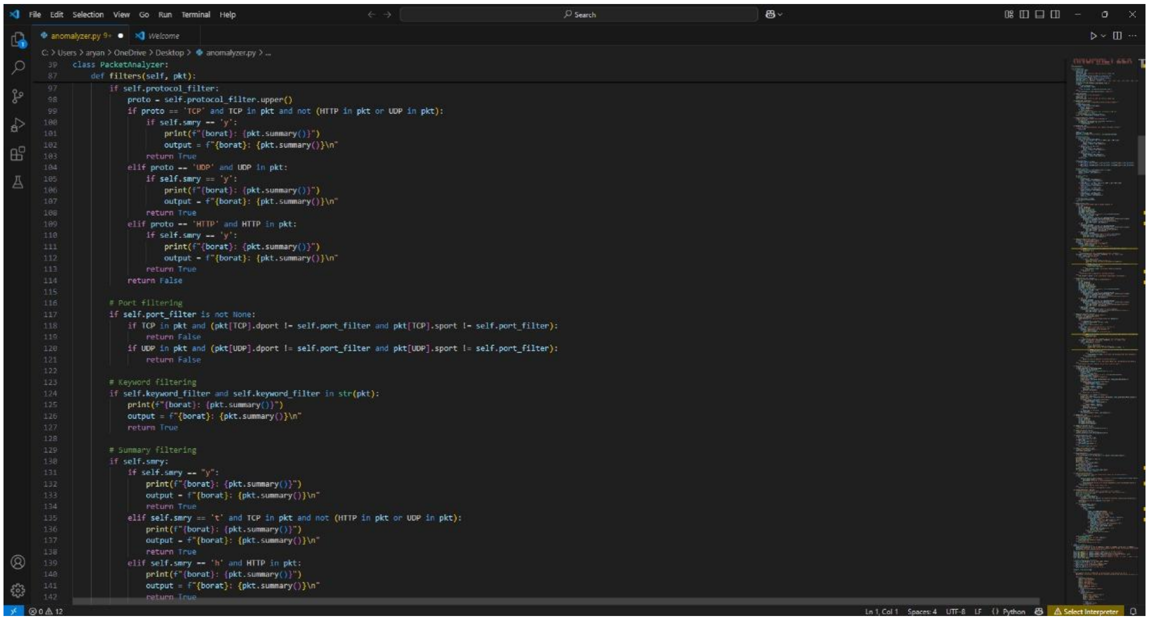
[18] A. Brown and B. Davis, "Web-Based Network Monitoring: Trends and Benefits," IEEE Netw., vol. 36, no. 3, pp. 10–15, May/Jun. 2022.

[19] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "A survey on the interplay between the Internet of Things and cybersecurity," ACM Comput. Surv., vol. 48, no. 1, pp. 1–36, Jul. 2015.

Code Snippet





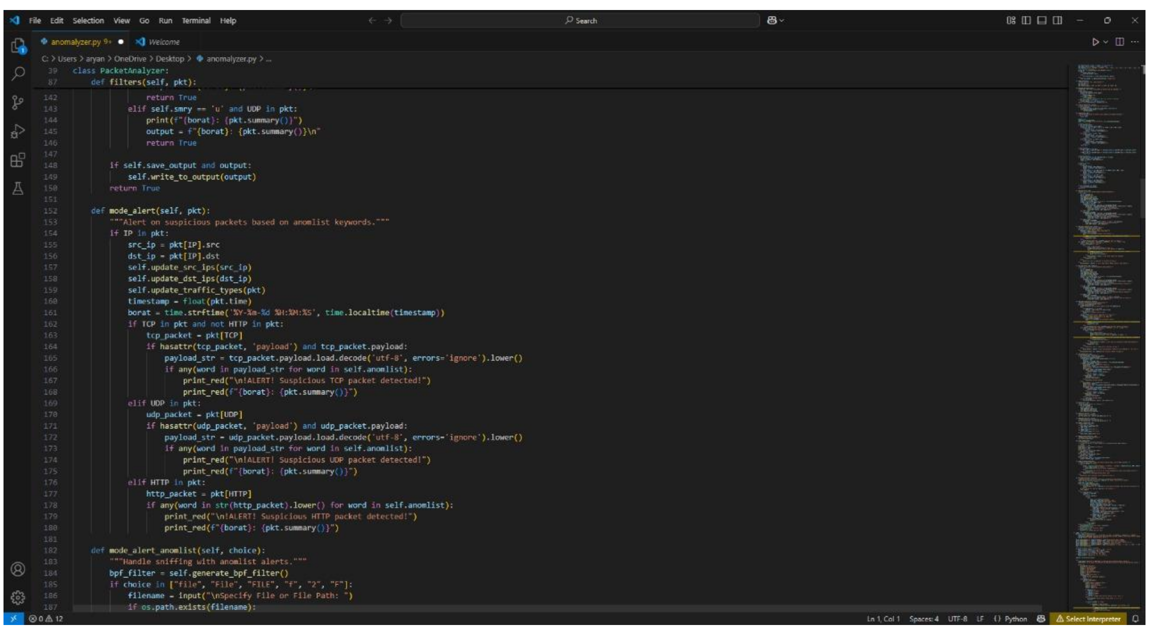


File Edit Selection View Go Run Terminal Help
anomalyzer.py
Welcome
C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
class PacketAnalyzer:
    def filters(self, pkt):
        if self.protocol_filter:
            proto = self.protocol_filter.upper()
            if proto == 'TCP' and TCP in pkt and not (HTTP in pkt or UDP in pkt):
                if self.smry == 'y':
                    print(f"{borat}: {pkt.summary()}")
                    output = f"{borat}: {pkt.summary()}\n"
                return True
            elif proto == 'UDP' and UDP in pkt:
                if self.smry == 'y':
                    print(f"{borat}: {pkt.summary()}")
                    output = f"{borat}: {pkt.summary()}\n"
                return True
            elif proto == 'HTTP' and HTTP in pkt:
                if self.smry == 'y':
                    print(f"{borat}: {pkt.summary()}")
                    output = f"{borat}: {pkt.summary()}\n"
                return True
            return False

        # Port filtering
        if self.port_filter is not None:
            if TCP in pkt and (pkt[TCP].dport != self.port_filter and pkt[TCP].sport != self.port_filter):
                return False
            if UDP in pkt and (pkt[UDP].dport != self.port_filter and pkt[UDP].sport != self.port_filter):
                return False

        # Keyword filtering
        if self.keyword_filter and self.keyword_filter in str(pkt):
            print(f"{borat}: {pkt.summary()}")
            output = f"{borat}: {pkt.summary()}\n"
            return True

        # Summary filtering
        if self.smry:
            if self.smry == "y":
                print(f"{borat}: {pkt.summary()}")
                output = f"{borat}: {pkt.summary()}\n"
                return True
            elif self.smry == 't' and TCP in pkt and not (HTTP in pkt or UDP in pkt):
                print(f"{borat}: {pkt.summary()}")
                output = f"{borat}: {pkt.summary()}\n"
                return True
            elif self.smry == 'h' and HTTP in pkt:
                print(f"{borat}: {pkt.summary()}")
                output = f"{borat}: {pkt.summary()}\n"
                return True

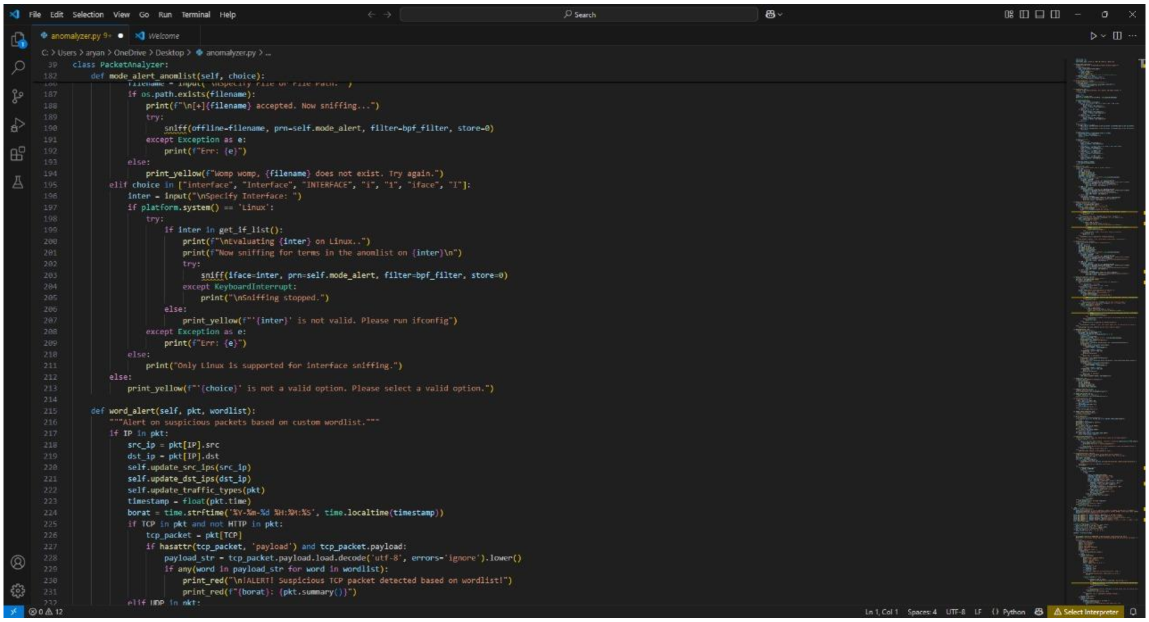


File Edit Selection View Go Run Terminal Help
anomalyzer.py
Welcome
C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
class PacketAnalyzer:
    def filters(self, pkt):
                return True
            elif self.smry == 'u' and UDP in pkt:
                print(f"{borat}: {pkt.summary()}")
                output = f"{borat}: {pkt.summary()}\n"
                return True

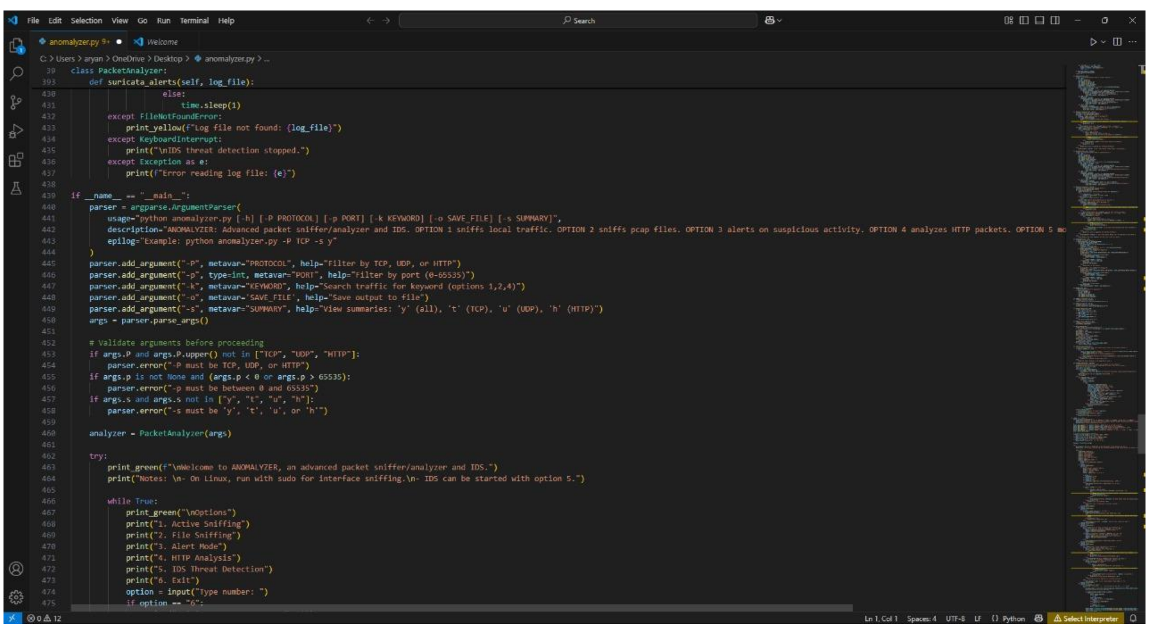
        if self.save_output and output:
            self.write_to_output(output)
        return True

    def mode_alert(self, pkt):
        """Alert on suspicious packets based on anomlist keywords."""
        if IP in pkt:
            src_ip = pkt[IP].src
            dst_ip = pkt[IP].dst
            self.update_src_ips(src_ip)
            self.update_dst_ips(dst_ip)
            self.update_traffic_types(pkt)
            timestamp = float(pkt.time)
            borat = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))
            if TCP in pkt and not HTTP in pkt:
                tcp_packet = pkt[TCP]
                if hasattr(tcp_packet, 'payload') and tcp_packet.payload:
                    payload_str = tcp_packet.payload.load.decode('utf-8', errors='ignore').lower()
                    if any(word in payload_str for word in self.anomlist):
                        print_red("\n!ALERT! Suspicious TCP packet detected!")
                        print_red(f"{borat}: {pkt.summary()}")
            elif UDP in pkt:
                udp_packet = pkt[UDP]
                if hasattr(udp_packet, 'payload') and udp_packet.payload:
                    payload_str = udp_packet.payload.load.decode('utf-8', errors='ignore').lower()
                    if any(word in payload_str for word in self.anomlist):
                        print_red("\n!ALERT! Suspicious UDP packet detected!")
                        print_red(f"{borat}: {pkt.summary()}")
            elif HTTP in pkt:
                http_packet = pkt[HTTP]
                if any(word in str(http_packet).lower() for word in self.anomlist):
                    print_red("\n!ALERT! Suspicious HTTP packet detected!")
                    print_red(f"{borat}: {pkt.summary()}")

    def mode_alert_anomlist(self, choice):
        """Handle sniffing with anomlist alerts."""
        bpf_filter = self.generate_bpf_filter()
        if choice in ["file", "File", "FILE", "f", "2", "F"]:
            filename = input("\nSpecify File or File Path: ")
            if os.path.exists(filename):


anomalyzer.py
C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
class PacketAnalyzer:
    def mode_alert_anomlist(self, choice):
            if os.path.exists(filename):
                print(f"\n[+]{filename} accepted. Now sniffing...")
                try:
                    sniff(offline=filename, prn=self.mode_alert, filter=bpf_filter, store=0)
                except Exception as e:
                    print(f"Err: {e}")
            else:
                print_yellow(f"Womp womp, {filename} does not exist. Try again.")
        elif choice in ["interface", "Interface", "INTERFACE", "i", "1", "iface", "I"]:
            inter = input("\nSpecify Interface: ")
            if platform.system() == 'Linux':
                try:
                    if inter in get_if_list():
                        print(f"\nEvaluating {inter} on Linux..")
                        print(f"Now sniffing for terms in the anomlist on {inter}\n")
                        try:
                            sniff(iface=inter, prn=self.mode_alert, filter=bpf_filter, store=0)
                        except KeyboardInterrupt:
                            print("\nSniffing stopped.")
                    else:
                        print_yellow(f"'{inter}' is not valid. Please run ifconfig")
                except Exception as e:
                    print(f"Err: {e}")
            else:
                print("Only Linux is supported for interface sniffing.")
        else:
            print_yellow(f"'{choice}' is not a valid option. Please select a valid option.")

    def word_alert(self, pkt, wordlist):
        """Alert on suspicious packets based on custom wordlist."""
        if IP in pkt:
            src_ip = pkt[IP].src
            dst_ip = pkt[IP].dst
            self.update_src_ips(src_ip)
            self.update_dst_ips(dst_ip)
            self.update_traffic_types(pkt)
            timestamp = float(pkt.time)
            borat = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(timestamp))
            if TCP in pkt and not HTTP in pkt:
                tcp_packet = pkt[TCP]
                if hasattr(tcp_packet, 'payload') and tcp_packet.payload:
                    payload_str = tcp_packet.payload.load.decode('utf-8', errors='ignore').lower()
                    if any(word in payload_str for word in wordlist):
                        print_red("\n!ALERT! Suspicious TCP packet detected based on wordlist!")
                        print_red(f"{borat}: {pkt.summary()}")
            elif UDP in pkt:


anomalyzer.py
C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
class PacketAnalyzer:
    def suricata_alerts(self, log_file):
                    else:
                        time.sleep(1)
        except FileNotFoundError:
            print_yellow(f"Log file not found: {log_file}")
        except KeyboardInterrupt:
            print("\nIDS threat detection stopped.")
        except Exception as e:
            print(f"Error reading log file: {e}")

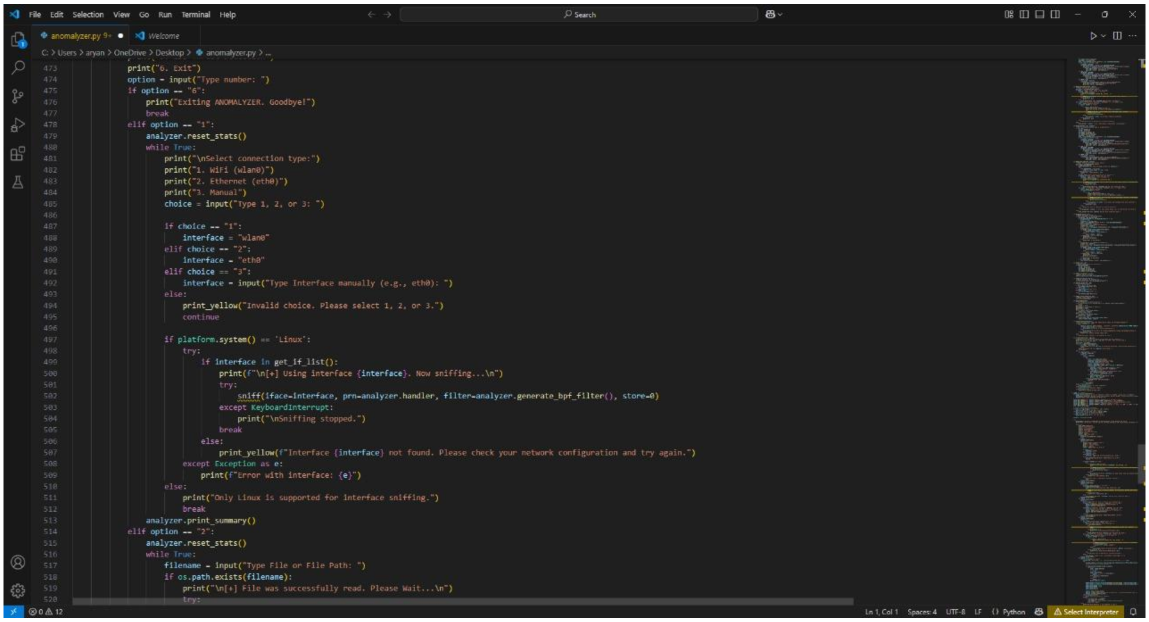
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        usage="python anomalyzer.py [-h] [-P PROTOCOL] [-p PORT] [-k KEYWORD] [-o SAVE_FILE] [-s SUMMARY]",
        description="ANOMALYZER: Advanced packet sniffer/analyzer and IDS. OPTION 1 sniffs local traffic. OPTION 2 sniffs pcap files. OPTION 3 alerts on suspicious activity. OPTION 4 analyzes HTTP packets. OPTION 5 mo
        epilog="Example: python anomalyzer.py -P TCP -s y"
)
    parser.add_argument("-P", metavar="PROTOCOL", help="Filter by TCP, UDP, or HTTP")
    parser.add_argument("-p", type=int, metavar="PORT", help="Filter by port (0-65535)")
    parser.add_argument("-k", metavar="KEYWORD", help="Search traffic for keyword (options 1,2,4)")
    parser.add_argument("-o", metavar='SAVE_FILE', help="Save output to file")
    parser.add_argument("-s", metavar="SUMMARY", help="View summaries: 'y' (all), 't' (TCP), 'u' (UDP), 'h' (HTTP)")
    args = parser.parse_args()

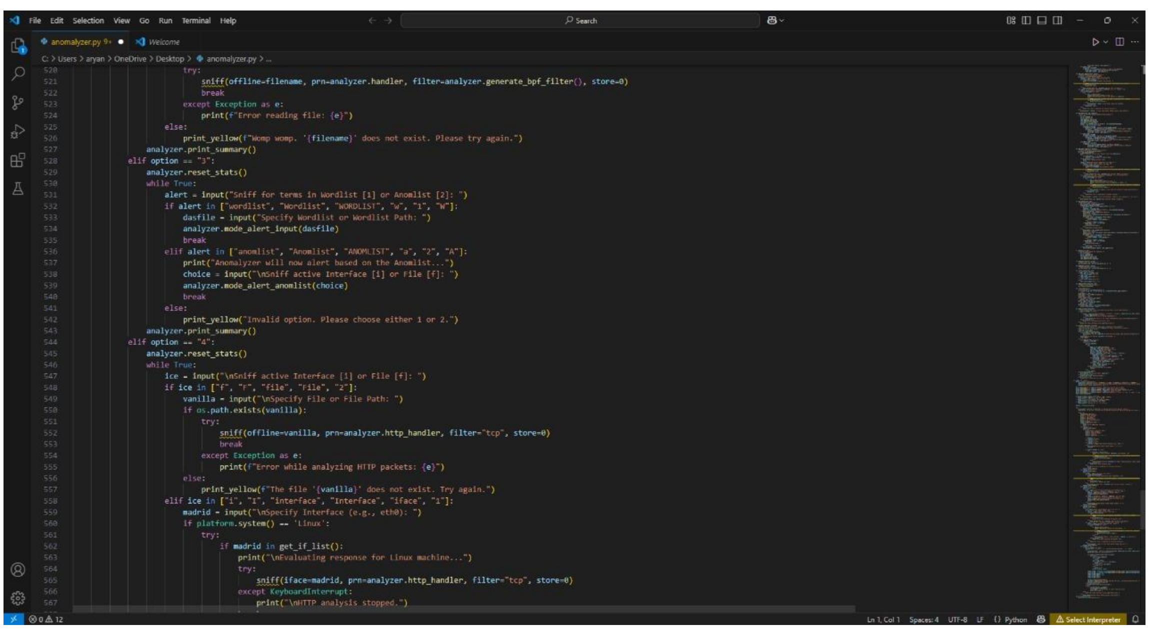
    # Validate arguments before proceeding
    if args.P and args.P.upper() not in ["TCP", "UDP", "HTTP"]:
        parser.error("-P must be TCP, UDP, or HTTP")
    if args.p is not None and (args.p < 0 or args.p > 65535):
        parser.error("-p must be between 0 and 65535")
    if args.s and args.s not in ["y", "t", "u", "h"]:
        parser.error("-s must be 'y', 't', 'u', or 'h'")

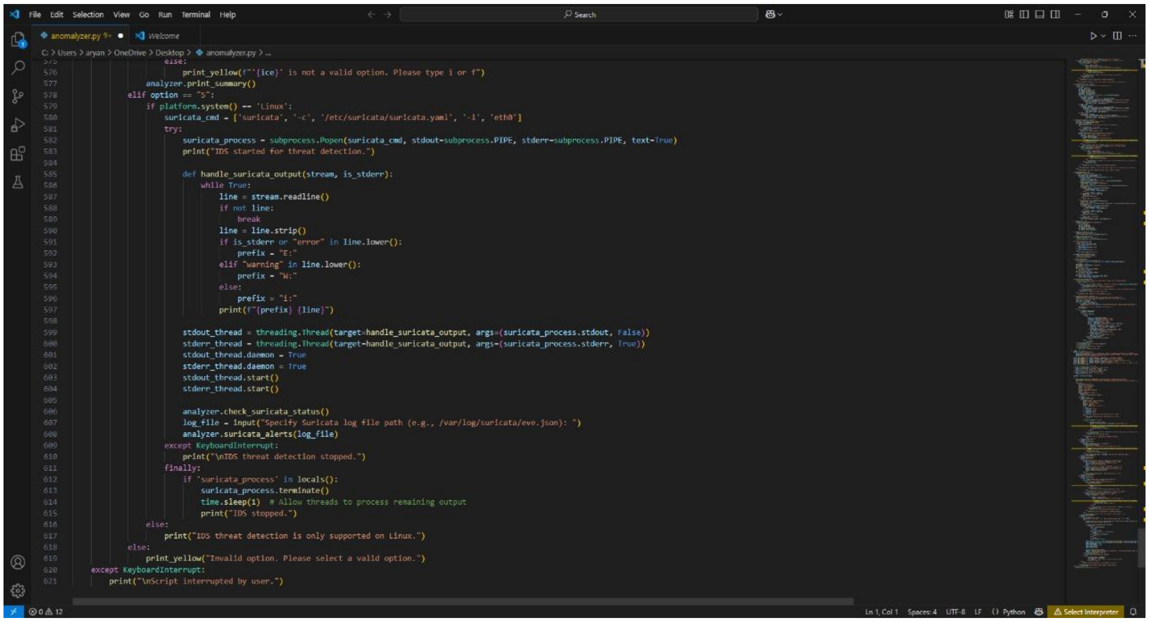
    analyzer = PacketAnalyzer(args)

    try:
        print_green(f"\nWelcome to ANOMALYZER, an advanced packet sniffer/analyzer and IDS.")
        print("Notes: \n- On Linux, run with sudo for interface sniffing.\n- IDS can be started with option 5.")

        while True:
            print_green("\nOptions")
            print("1. Active Sniffing")
            print("2. File Sniffing")
            print("3. Alert Mode")
            print("4. HTTP Analysis")
            print("5. IDS Threat Detection")
            print("6. Exit")
            option = input("Type number: ")
            if option == "6":


C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
            print("6. Exit")
            option = input("Type number: ")
            if option == "6":
                print("Exiting ANOMALYZER. Goodbye!")
                break
            elif option == "1":
                analyzer.reset_stats()
                while True:
                    print("\nSelect connection type:")
                    print("1. WiFi (wlan0)")
                    print("2. Ethernet (eth0)")
                    print("3. Manual")
                    choice = input("Type 1, 2, or 3: ")
                    if choice == "1":
                        interface = "wlan0"
                    elif choice == "2":
                        interface = "eth0"
                    elif choice == "3":
                        interface = input("Type Interface manually (e.g., eth0): ")
                    else:
                        print_yellow("Invalid choice. Please select 1, 2, or 3.")
                        continue
                    if platform.system() == 'Linux':
                        try:
                            if interface in get_if_list():
                                print(f"\n[+] Using interface {interface}. Now sniffing...\n")
                                try:
                                    sniff(iface=interface, prn=analyzer.handler, filter=analyzer.generate_bpf_filter(), store=0)
                                except KeyboardInterrupt:
                                    print("\nSniffing stopped.")
                                break
                            else:
                                print_yellow(f"Interface {interface} not found. Please check your network configuration and try again.")
                        except Exception as e:
                            print(f"Error with interface: {e}")
                    else:
                        print("Only Linux is supported for interface sniffing.")
                        break
                analyzer.print_summary()
            elif option == "2":
                analyzer.reset_stats()
                while True:
                    filename = input("Type File or File Path: ")
                    if os.path.exists(filename):
                        print("\n[+] File was successfully read. Please Wait...\n")
                        try:


C: > Users > aryan > OneDrive > Desktop > anomalyzer.py > ...
                        try:
                            sniff(offline=filename, prn=analyzer.handler, filter=analyzer.generate_bpf_filter(), store=0)
                            break
                        except Exception as e:
                            print(f"Error reading file: {e}")
                    else:
                        print_yellow(f"Womp womp. '{filename}' does not exist. Please try again.")
                analyzer.print_summary()
            elif option == "3":
                analyzer.reset_stats()
                while True:
                    alert = input("Sniff for terms in Wordlist [1] or Anomlist [2]: ")
                    if alert in ["wordlist", "Wordlist", "WORDLIST", "w", "1", "W"]:
                        dasfile = input("Specify Wordlist or Wordlist Path: ")
                        analyzer.mode_alert_input(dasfile)
                        break
                    elif alert in ["anomlist", "Anomlist", "ANOMLIST", "a", "2", "A"]:
                        print("Anomalyzer will now alert based on the Anomlist...")
                        choice = input("\nSniff active Interface [i] or File [f]: ")
                        analyzer.mode_alert_anomlist(choice)
                        break
                    else:
                        print_yellow("Invalid option. Please choose either 1 or 2.")
                analyzer.print_summary()
            elif option == "4":
                analyzer.reset_stats()
                while True:
                    ice = input("\nSniff active Interface [1] or File [f]: ")
                    if ice in ["f", "F", "file", "File", "2"]:
                        vanilla = input("\nSpecify File or File Path: ")
                        if os.path.exists(vanilla):
                            try:
                                sniff(offline=vanilla, prn=analyzer.http_handler, filter="tcp", store=0)
                                break
                            except Exception as e:
                                print(f"Error while analyzing HTTP packets: {e}")
                        else:
                            print_yellow(f"The file '{vanilla}' does not exist. Try again.")
                    elif ice in ["i", "I", "interface", "Interface", "iface", "1"]:
                        madrid = input("\nSpecify Interface (e.g., eth0): ")
                        if platform.system() == 'Linux':
                            try:
                                if madrid in get_if_list():
                                    print("\nEvaluating response for Linux machine...")
                                    try:
                                        sniff(iface=madrid, prn=analyzer.http_handler, filter="tcp", store=0)
                                    except KeyboardInterrupt:
                                        print("\nHTTP analysis stopped.")

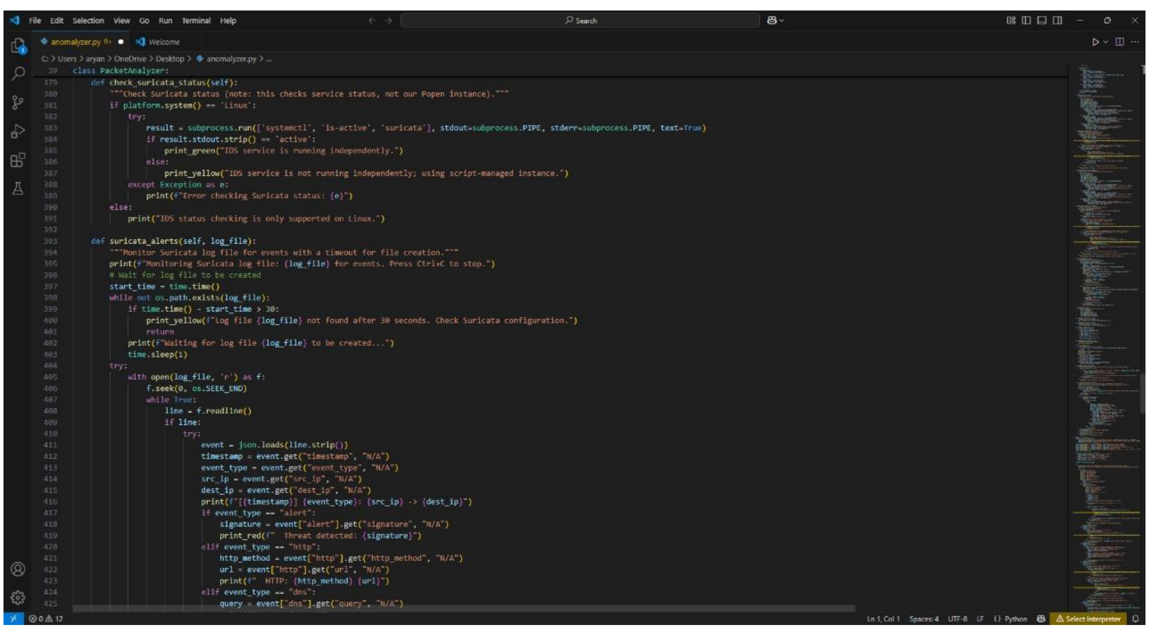


```python
                else:
                    print_yellow(f"'{ice}' is not a valid option. Please type i or f")
                analyzer.print_summary()
            elif option == "5":
                if platform.system() == 'Linux':
                    suricata_cmd = ['suricata', '-c', '/etc/suricata/suricata.yaml', '-i', 'eth0']
                    try:
                        suricata_process = subprocess.Popen(suricata_cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
                        print("IDS started for threat detection.")

                        def handle_suricata_output(stream, is_stderr):
                            while True:
                                line = stream.readline()
                                if not line:
                                    break
                                line = line.strip()
                                if is_stderr or "error" in line.lower():
                                    prefix = "E:"
                                elif "warning" in line.lower():
                                    prefix = "W:"
                                else:
                                    prefix = "i:"
                                print(f"{prefix} {line}")

                        stdout_thread = threading.Thread(target=handle_suricata_output, args=(suricata_process.stdout, False))
                        stderr_thread = threading.Thread(target=handle_suricata_output, args=(suricata_process.stderr, True))
                        stdout_thread.daemon = True
                        stderr_thread.daemon = True
                        stdout_thread.start()
                        stderr_thread.start()

                        analyzer.check_suricata_status()
                        log_file = input("Specify Suricata log file path (e.g., /var/log/suricata/eve.json): ")
                        analyzer.suricata_alerts(log_file)
                    except KeyboardInterrupt:
                        print("\nIDS threat detection stopped.")
                    finally:
                        if 'suricata_process' in locals():
                            suricata_process.terminate()
                            time.sleep(1)  # Allow threads to process remaining output
                            print("IDS stopped.")
                else:
                    print("IDS threat detection is only supported on Linux.")
            else:
                print_yellow("Invalid option. Please select a valid option.")
    except KeyboardInterrupt:
        print("\nScript interrupted by user.")
```



```python
class PacketAnalyzer:
    def check_suricata_status(self):
        """Check Suricata status (note: this checks service status, not our Popen instance)."""
        if platform.system() == 'Linux':
            try:
                result = subprocess.run(['systemctl', 'is-active', 'suricata'], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
                if result.stdout.strip() == 'active':
                    print_green("IDS service is running independently.")
                else:
                    print_yellow("IDS service is not running independently; using script-managed instance.")
            except Exception as e:
                print(f"Error checking Suricata status: {e}")
        else:
            print("IDS status checking is only supported on Linux.")

    def suricata_alerts(self, log_file):
        """Monitor Suricata log file for events with a timeout for file creation."""
        print(f"Monitoring Suricata log file: {log_file} for events. Press Ctrl+C to stop.")
        # Wait for log file to be created
        start_time = time.time()
        while not os.path.exists(log_file):
            if time.time() - start_time > 30:
                print_yellow(f"Log file {log_file} not found after 30 seconds. Check Suricata configuration.")
                return
            print(f"Waiting for log file {log_file} to be created...")
            time.sleep(1)
        try:
            with open(log_file, 'r') as f:
                f.seek(0, os.SEEK_END)
                while True:
                    line = f.readline()
                    if line:
                        try:
                            event = json.loads(line.strip())
                            timestamp = event.get("timestamp", "N/A")
                            event_type = event.get("event_type", "N/A")
                            src_ip = event.get("src_ip", "N/A")
                            dest_ip = event.get("dest_ip", "N/A")
                            print(f"[{timestamp}] {event_type}: {src_ip} -> {dest_ip}")
                            if event_type == "alert":
                                signature = event["alert"].get("signature", "N/A")
                                print_red(f"  Threat detected: {signature}")
                            elif event_type == "http":
                                http_method = event["http"].get("http_method", "N/A")
                                url = event["http"].get("url", "N/A")
                                print(f"  HTTP: {http_method} {url}")
                            elif event_type == "dns":
                                query = event["dns"].get("query", "N/A")
```

Turnitin Report

Document Details

Submission ID
trn:oid:::1:3260573315

Submission Date
May 26, 2025, 12:27 PM GMT+5:30

Download Date
May 26, 2025, 12:29 PM GMT+5:30

File Name
Complete_NTA_Project_Report_.pdf

File Size
1.0 MB

57 Pages

8,126 Words

49,845 Characters

12% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 50 Not Cited or Quoted 12%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 2 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 10% Internet sources
- 4% Publications
- 10% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **50** Not Cited or Quoted 12%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **2** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

10% Internet sources
4% Publications
10% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	KIET Group of Institutions, Ghaziabad	3%
2	Internet	www.coursehero.com	3%
3	Internet	eprints.bournemouth.ac.uk	<1%
4	Internet	ieeexplore.ieee.org	<1%
5	Internet	www.researchgate.net	<1%
6	Internet	pdfs.semanticscholar.org	<1%
7	Internet	cris.vtt.fi	<1%
8	Internet	acmbulletin.fiit.stuba.sk	<1%
9	Internet	studentsrepo.um.edu.my	<1%
10	Internet	technodocbox.com	<1%

11	Student papers	
Aalto Yliopisto		<1%
12	Student papers	
AlHussein Technical University		<1%
13	Student papers	
Bharathiar University		<1%
14	Student papers	
Napier University		<1%
15	Internet	
informatica.si		<1%
16	Internet	
www.grc.upv.es		<1%
17	Student papers	
Northcentral		<1%
18	Student papers	
Terna Engineering College		<1%
19	Internet	
sustainabilityservices.eurofins.com		<1%
20	Publication	
Nabiyeva, Gulnara Nuridinovna. "Geographic Information Systems (GIS) as a Tool...		<1%
21	Internet	
www.slideshare.net		<1%
22	Internet	
fastercapital.com		<1%
23	Internet	
fdocuments.in		<1%
24	Internet	
isyou.info		<1%

25	Internet	
brightideas.houstontx.gov		<1%
26	Internet	
idoc.tips		<1%
27	Internet	
kar.kent.ac.uk		<1%
28	Publication	
Mukhtar Ahmed, Jinfu Chen, Ernest Akpaku, Rexford Nii Ayitey Sosu. "MTCR-AE: A ...		<1%
29	Internet	
atharvacoe.ac.in		<1%
30	Internet	
salford-repository.worktribe.com		<1%
31	Internet	
zlibrary.ac		<1%
32	Publication	
Li, Jingyi. "From Abandonment to Revitalisation: Life Cycle Sustainability Assessm...		<1%
33	Internet	
mechanical.anits.edu.in		<1%
34	Internet	
repository.unej.ac.id		<1%