A close-up of a sign

Description automatically generated

Project Title:

**Network Security Scanner**

Project Members:

**Name:**

**Rimsha Shabir**

**Registration #:**

**BSIT-FA20-031**

**Email:**

[rimshashabbir64@gmail.com](mailto:rimshashabbir64@gmail.com)

Supervised By: Signature

**Dr. Shabana Ramzan ----------------------**

Date:

**13/06/2024**

# **Chapter 1. Introduction**

## **1.1 Introduction:**

In today's digital landscape, network security is a critical concern for organizations of all sizes. The increasing complexity and frequency of cyber threats necessitate the development of sophisticated tools to protect network infrastructures. This proposal outlines the Network Security Scanner project, which aims to provide a comprehensive solution for real-time network monitoring, vulnerability detection, and protocol analysis. The following sections detail the project's title, overview, goals, objectives, system components, optional functional units, exclusions, application architecture, Gantt chart, hardware and software specifications, and the tools and technologies used.

The Critical Need for Network Security

In today's hyper-connected world, organizations of all sizes rely heavily on their network infrastructure. This dependence makes them prime targets for cyberattacks, which are becoming increasingly sophisticated and frequent. Data breaches, malware infections, and denial-of-service attacks can have devastating consequences, causing financial losses, reputational damage, and operational disruptions.

Introducing the Network Security Scanner

To address these growing security challenges, we propose the Network Security Scanner project. This comprehensive tool aims to empower organizations to proactively monitor their networks, identify vulnerabilities, and gain valuable insights into network traffic patterns.

## **1.2. Project Title:**

Network Security Scanner

## **1.3. Project Overview Statement:**

The Network Security Scanner is designed to enhance network security through real-time monitoring, vulnerability detection, and protocol analysis. This tool provides network administrators with the ability to identify potential threats, analyze network traffic, and generate detailed reports for informed decision-making. Utilizing Python libraries such as nmap, scapy, and tkinter, the scanner integrates robust backend processing with an intuitive graphical user interface (GUI) to ensure both functionality and usability.

## **1.4. Project Goals & Objectives:**

**Goals:**

The primary goal of the Network Security Scanner is to provide a real-time and user-friendly solution for network security management.

* Real-time Network Monitoring: The scanner will continuously scan the network, identifying active devices and monitoring their overall health.
* Vulnerability Detection: By leveraging advanced scanning techniques, the project aims to detect potential vulnerabilities in network devices and operating systems. This allows for timely remediation efforts before attackers can exploit these weaknesses.
* Protocol Analysis: The scanner will capture and analyze network traffic to understand the protocols used and identify any suspicious activity. This can help detect anomalies and potential intrusions.

**Objectives:**

1. **Network Traffic Capture:**
   * Utilize nmap and scapy to capture live network traffic, covering a wide range of protocols and activities.
   * Ensure compatibility with various network configurations and environments for comprehensive monitoring.
2. **Vulnerability Detection:**
   * Implement vulnerability checks for common ports and services, such as FTP, SSH, HTTP, and HTTPS.
   * Maintain a database of known vulnerabilities and exploits to compare against scanned hosts and services.

## **1.5. High-Level System Components:**

1. **Network Scanner:**
   * **Functionality:** Conducts comprehensive network scans to identify active hosts, open ports, and running services.
   * **Implementation Details:**
   * Utilizes the nmap library to perform port scanning and host discovery.
   * Extracts detailed information about each scanned host, including hostname, state, and open ports.
2. **Vulnerability Checker:**
   * **Functionality:** Analyzes scanned hosts and services for potential vulnerabilities and security weaknesses.
   * **Implementation Details:**
   * Utilizes nmap scan results to check for known vulnerabilities associated with open ports and services.
   * Logs potential vulnerabilities and suggests remediation steps for network administrators.
3. **Protocol Analyzer:**
   * **Functionality:** Analyzes network protocols to identify anomalies and potential security threats.
   * **Implementation Details:**
   * Utilizes scapy to capture and dissect network packets, extracting protocol-level information.
   * Performs protocol analysis to detect suspicious or unauthorized activities within the network.
4. **Report Generator:**
   * **Functionality:** Generates detailed reports summarizing the results of network scans, vulnerability checks, and protocol analysis.
   * **Implementation Details:**
   * Aggregates scan results, vulnerability findings, and protocol analysis into comprehensive reports.
   * Utilizes pandas and openpyxl to create Excel reports with structured data and visualizations.

A Modular System Design

The Network Security Scanner will be designed as a modular system composed of several key components:

* Scanning Engine: This core component will utilize tools like Nmap and Scapy to perform network scans, identify active hosts, open ports, and services running on those devices.
* Vulnerability Database: The scanner will maintain an up-to-date database of known vulnerabilities, allowing it to compare discovered issues with known threats.
* Packet Capture and Analysis Engine: This module will capture network packets and analyze them to identify protocols used, potential intrusions, and suspicious traffic patterns.
* Reporting Module: The scanner will generate comprehensive reports detailing the scan results, identified vulnerabilities, and network traffic analysis.
* User Interface: An intuitive user interface will provide a central hub for users to initiate scans, view results, configure settings, and manage reports.

## **1.6. List of Optional Functional Units:**

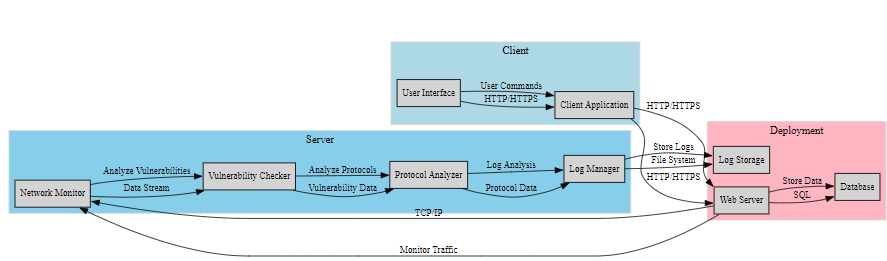
1. **Email Alerts for Vulnerabilities:**
   * Automatically sends email notifications to network administrators upon detection of critical vulnerabilities.
2. **Advanced Filtering Options in the GUI:**
   * Provides users with the ability to filter and customize the displayed network data based on various criteria.
3. **Historical Data Analysis:**
   * Implements functionalities to analyze historical network traffic data, identify patterns, and generate trend reports.

## **1.7. Exclusions:**

1. **Deep Packet Inspection:**
   * Excluded due to the complexity and resource-intensive nature of deep packet inspection, which falls outside the scope of this project.
2. **Integration with External Databases:**
   * Excluded to maintain simplicity and reduce dependencies, focusing on a standalone tool with local logging capabilities.

## **1.8. Application Architecture:**

**Description:** The application architecture comprises interconnected components, including the Network Scanner, Vulnerability Checker, Protocol Analyzer, and Report Generator. These components work together to provide a comprehensive network security solution.



## **1.9. Gantt Chart:**

**Description:** The Gantt chart visualizes the project timeline, outlining key tasks and milestones from initial planning to deployment and user training. This chart provides a high-level overview of the project's progress and helps in tracking deadlines and deliverables.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Task Name | Duration | Start Date | Finish Date | Predecessors |
| 1 | Define project scope | 2 days | 2024-05-01 | 2024-05-02 | None |
| 2 | Setup development environment | 2 days | 2024-05-03 | 2024-05-04 | 1 |
| 3 | Install necessary libraries | 1 day | 2024-05-05 | 2024-05-05 | 2 |
| 4 | Implement network\_scan | 5 days | 2024-05-06 | 2024-05-10 | 3 |
| 5 | Test network scanning | 2 days | 2024-05-11 | 2024-05-12 | 4 |
| 6 | Implement check\_vulnerabilities | 3 days | 2024-05-13 | 2024-05-15 | 5 |
| 7 | Test vulnerability checking | 2 days | 2024-05-16 | 2024-05-17 | 6 |
| 8 | Implement analyze\_protocols | 3 days | 2024-05-18 | 2024-05-20 | 7 |
| 9 | Test protocol analysis | 2 days | 2024-05-21 | 2024-05-22 | 8 |
| 10 | Implement generate\_report | 4 days | 2024-05-23 | 2024-05-26 | 9 |
| 11 | Test report generation | 2 days | 2024-05-27 | 2024-05-28 | 10 |
| 12 | Design and implement GUI | 5 days | 2024-05-29 | 2024-06-02 | 11 |
| 13 | Test user interface | 3 days | 2024-06-03 | 2024-06-05 | 12 |
| 14 | Implement visualize\_results | 4 days | 2024-06-06 | 2024-06-09 | 13 |
| 15 | Test visualization | 2 days | 2024-06-10 | 2024-06-11 | 14 |
| 16 | Integrate all modules | 3 days | 2024-06-12 | 2024-06-14 | 15 |
| 17 | End-to-end testing | 4 days | 2024-06-15 | 2024-06-18 | 16 |
| 18 | Write project documentation | 3 days | 2024-06-19 | 2024-06-21 | 17 |
| 19 | Final review and testing | 3 days | 2024-06-22 | 2024-06-24 | 18 |

## **1.10. Hardware and Software Specifications:**

The hardware and software requirements will be determined based on the network size and complexity. The scanner will likely run on a dedicated server with sufficient processing power, memory, and storage capacity. Open-source tools like Nmap and Scapy will be leveraged for core functionalities, while the user interface may be developed using frameworks like Tkinter or PyQt. pen\_spark

**Hardware:**

* Standard PC: Equipped with a network interface card capable of handling high volumes of network traffic, sufficient RAM, and processing power.

**Software:**

* Python 3.x: Primary programming language for development.
* nmap and scapy: Libraries for network scanning and packet manipulation.
* tkinter: Graphical user interface development.
* pandas and openpyxl: Libraries for data manipulation and report generation.

## **1.11. Tools and Technologies Used:**

1. **nmap and scapy:**
   * **Purpose:** Network scanning and packet manipulation.
   * **Details:** nmap for comprehensive network scans and scapy for packet capturing and analysis.
   * **Reason for Use:** Robust functionalities and compatibility with Python for seamless integration.
2. **tkinter:**
   * **Purpose:** Graphical user interface development.
   * **Details:** Standard GUI library in Python for creating interactive interfaces.
   * **Reason for Use:** Ease of use and integration with Python for developing user-friendly interfaces.
3. **pandas and openpyxl:**
   * **Purpose:** Data manipulation and report generation.
   * **Details:** pandas for data analysis and openpyxl for Excel report generation.
   * **Reason for Use:** Efficient handling of structured data and seamless integration with Python.

# **Chapter 2: Feasibility Studies**

## **2.1. Introduction:**

The first milestone in our project journey is the development of the Network Traffic Monitor and Logger's initial version. This chapter delves into the feasibility, scope, costing, team structure, tools, and risk management involved in delivering this crucial phase. The Network Traffic Monitor and Logger project embarks on its journey with the development of the initial version. This chapter serves as a comprehensive roadmap, meticulously detailing the feasibility analysis, project scope, cost estimations, team structure, tools and technologies employed, and risk management strategies. By thoroughly understanding these aspects, we ensure the successful delivery of this crucial foundational phase.

The initial version of the Network Traffic Monitor and Logger marks a significant milestone, laying the groundwork for a robust and comprehensive network monitoring solution. This chapter delves into the intricate details of this phase, establishing a roadmap for development success. We'll explore the project's viability from various perspectives, define its scope, estimate potential costs, outline task dependencies, identify potential risks, and establish mitigation strategies. Additionally, we'll delve into the core functionalities and potential future enhancements of the initial version.

## **2.2. Project/Product Feasibility Report:**

A comprehensive feasibility analysis ensures the project's success before we commence development. We'll meticulously assess the project's viability through various lenses:

### 2.2.1. Technical Feasibility:

Our project harnesses existing technologies to create a robust monitoring tool:

1. **Python:**
   * **Rationale:** Python's extensive library support, especially with packet capturing libraries like pyshark, makes it the ideal choice for our project.
2. **Pyshark:**
   * **Rationale:** Pyshark, being a Python wrapper for Wireshark's tshark, facilitates efficient packet capturing and parsing, crucial for real-time network monitoring.
3. **Tkinter:**
   * **Rationale:** Tkinter, as a standard GUI toolkit in Python, ensures the development of a user-friendly interface without additional complexities.

### 2.2.2. Operational Feasibility:

Our tool seamlessly integrates into existing operational setups. The Network Traffic Monitor and Logger is designed for seamless integration into existing operational environments, offering the following benefits:

* Intuitive GUI, designed for ease of use and quick adoption.
* Compatibility with various network configurations, minimizing setup requirements.

### 2.2.3. Economic Feasibility:

This project proves economically viable owing to:

* Low hardware costs, requiring only standard PCs with network interface cards.
* Utilization of open-source libraries like pyshark and tkinter, significantly reducing licensing expenses.

### 2.2.4. Schedule Feasibility:

The project is structured with a clear timeline, ensuring timely delivery:

* Phases include planning, development, testing, and deployment.
* A detailed Gantt chart outlines milestones and deadlines, facilitating effective project management.

### 2.2.5. Specification Feasibility:

Our tool meets the necessary specifications for a network monitoring tool, offering:

* Real-time monitoring capabilities.
* Detailed logging of traffic information.
* A user-friendly GUI for seamless interaction.

### 2.2.6. Information Feasibility:

With continuous packet capturing and logging, our tool ensures the availability of accurate real-time network data, critical for effective analysis and decision-making.

### 2.2.7. Motivational Feasibility:

The tool's intuitive interface and proactive alerting system motivate users to actively engage in network security:

* User-friendly GUI encourages regular monitoring and interaction.
* Real-time alerts raise awareness about potential security threats, fostering a proactive security culture.

### 2.2.8. Legal & Ethical Feasibility:

Our project adheres to legal guidelines and ethical considerations concerning network monitoring:

* Compliance with privacy laws ensures responsible data handling.
* Ethical standards are upheld, promoting transparency and accountability.

## **2.3 Project/Product Scope:**

The initial version of the Network Traffic Monitor and Logger encompasses:

* Real-time network traffic monitoring.
* Logging and categorization of traffic based on predefined criteria.

## **2.4 Project/Product Costing:**

### 2.4.1. Cost Estimation by Function Point Analysis:

We estimate function points for key project functions:

1. **Packet Capturing:** 30 function points.
2. **Traffic Logging:** 25 function points.
3. **GUI Development:** 20 function points.
4. **Alert System:** 15 function points.

### 2.4.2. Activity-Based Costing:

Activity-based costing allocates costs across project activities, ensuring effective cost management:

* Development costs are estimated based on key project functions.
* Testing and deployment activities are also accounted for.

## **2.5. Task Dependency Table:**

Understanding task dependencies is crucial for effective project planning and execution:

* Packet capturing is independent.
* Traffic logging depends on packet capturing.
* GUI development follows traffic logging.
* Testing and debugging rely on all preceding tasks.

## **2.6. Critical Path Method (CPM):**

The CPM diagram identifies the sequence of critical tasks, ensuring timely project completion.

## **2.7. Tools and Technology with Reasoning:**

We employ specific tools and technologies chosen for their suitability and effectiveness in achieving project objectives:

1. **Python:** Versatile and well-suited for rapid development.
2. **Pyshark:** Efficient packet capturing capabilities.
3. **Tkinter:** Standard GUI toolkit for seamless interface development.

## **2.8. Vision Document:**

The vision of our project is to enhance network security by providing a real-time monitoring and logging tool that swiftly identifies potential threats.

## **2.9. Risk List:**

Identifying and mitigating potential risks is essential for project success:

1. **Incomplete Packet Capture Due to Network Congestion:** Buffer management techniques to handle high traffic volumes.
2. **GUI Responsiveness Under High Traffic Conditions:** GUI code optimization for efficient real-time data updates.

## **2.10. Product Features/Product Decomposition:**

The Network Traffic Monitor and Logger offer core and additional features for comprehensive network monitoring:

1. **Core Features:** Real-time monitoring and detailed logging.
2. **Additional Features:** User-friendly GUI and proactive alert system.

# **Chapter 3: Requirements Analysis**

## **3.1. Introduction:**

The second deliverable delves into the object-oriented design approach for the Network Traffic Monitor and Logger. It aims to provide a thorough understanding of the system's structure, specifications, and requirements allocation. This chapter encompasses system specifications, external entity identification, context-level data flow diagram, "shall" statements, requirement allocation, prioritization, and a detailed exploration of the object-oriented approach's implementation. The second major deliverable delves into the intricate details of the object-oriented (OO) design approach for the Network Traffic Monitor and Logger. This chapter aims to provide a comprehensive understanding of the system's structure, specifications, and how requirements are allocated to different objects. We'll explore:

* System Specifications
* External Entity Identification
* Context-Level Data Flow Diagram
* "Shall" Statements
* Requirement Allocation and Prioritization
* Object-Oriented Design Exploration

This in-depth exploration of the OO design approach will lay the foundation for the subsequent development phases.

### 3.1.1 System Specifications:

The system specifications serve as a blueprint for the Network Traffic Monitor and Logger, meticulously outlining its core functionalities and features. Derived from the project scope and feasibility studies conducted in Chapter 3, these specifications are pivotal in guiding the design and development process. They ensure that the final product aligns with the project's objectives and meets user requirements.

Here's a breakdown of the key system specifications for the initial version of the Network Traffic Monitor and Logger:

1. **Capturing Network Traffic**:
   * **The system must adeptly capture live network traffic in real-time.**
     + This ensures that all network activities are monitored continuously without delay.
   * **It should support various network protocols, including TCP, UDP, HTTP, and DNS.**
     + Supporting multiple protocols allows for comprehensive monitoring and analysis of different types of network traffic.
2. **Identifying and Logging Bad Traffic**:
   * **The system must discern and log traffic from predefined bad IP addresses and domains.**
     + This feature is critical for identifying and tracking malicious activities within the network.
   * **Logged information should encompass details like timestamps, source IP, destination IP, and the protocol used.**
     + Detailed logging provides a clear record of network events, aiding in troubleshooting and forensic analysis.
3. **Providing a User-Friendly Interface**:
   * **The system should offer an intuitive GUI, empowering users to interact seamlessly.**
     + A user-friendly interface ensures that users can easily navigate and utilize the system's functionalities.
   * **GUI functionalities should enable users to commence or cease monitoring, view real-time logs, and receive alerts.**
     + Essential controls and displays within the GUI facilitate efficient monitoring and immediate response to network issues.

### 3.1.2 Identifying External Entities:

Understanding the external entities interfacing with the system elucidates its boundaries and interactions:

1. **Network Interface**:
   * **The network interface serves as the conduit for capturing network traffic.**
   * It acts as the primary input source for the system's monitoring capabilities.
   * **It furnishes the data packets for the system's monitoring and analysis.**
   * Ensuring a steady flow of data packets is crucial for real-time analysis.
2. **User**:
   * **Users engage with the system through the GUI.**
   * The GUI acts as the bridge between the user and the system's functionalities.
   * **They possess the capability to initiate or halt the monitoring process and peruse the logged data.**
   * User control over monitoring activities ensures that the system can be tailored to specific needs.

### 3.1.3 Context Level Data Flow Diagram:

The Context Level Data Flow Diagram (DFD) furnishes a top-level abstraction of the system, delineating its interactions with external entities:

* **Process**: Network Traffic Monitor and Logger
* **External Entities**:
* Network Interface
* User
* **Data Flows**:
* Network Data from Network Interface to System
* Commands and Queries from User to System
* Logs and Alerts from System to User

### 3.1.4 Capture "Shall" Statements:

The "shall" statements stipulate the mandatory requisites for the system, ensuring comprehensive coverage of essential functionalities:

1. **The system shall capture live network traffic**:
   * This mandate underscores the core functionality of real-time network monitoring.
2. **The system shall log traffic details**:
   * This requirement guarantees the comprehensive recording and analysis of network traffic data.
3. **The system shall provide a GUI for user interaction**:
   * This stipulation ensures user accessibility and ease of operation.

### 3.1.5 Allocate Requirements:

Requirement allocation organizes the development process by assigning priority levels to each requisite, ensuring the prioritized development and testing of critical functionalities:

1. **Packet Capturing**:
   * **Priority**: High
   * **Justification**: Imperative for the system's core functionality.
2. **Traffic Logging**:
   * **Priority**: High
   * **Justification**: Critical for the thorough recording and analysis of network traffic data.
3. **GUI Development**:
   * **Priority**: Medium
   * **Justification**: Essential for user interaction, albeit can be developed subsequent to core functionalities.

### 3.1.6 Prioritize Requirements:

Prioritizing requirements ensures the prompt addressal of critical functionalities, optimizing the development process and ensuring timely delivery of essential features:

1. **Packet Capturing**:
   * Highest priority owing to its pivotal role in real-time network monitoring.
2. **Traffic Logging**:
   * Second-highest priority to ensure the comprehensive recording and availability of data for analysis.
3. **GUI Development**:
   * Medium priority to furnish an interface for user interaction.

### 3.1.7 Requirements Traceability Matrix:

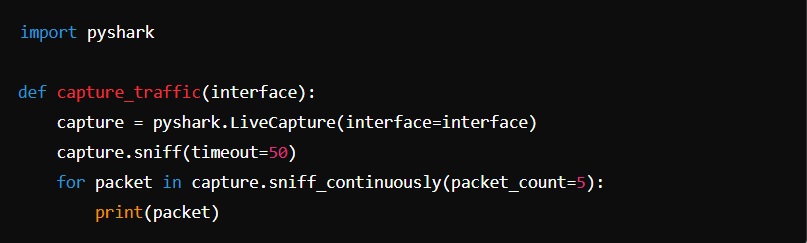
The Requirements Traceability Matrix (RTM) establishes a clear mapping between requirements and their corresponding implementation, ensuring comprehensive coverage:

* **Requirement**: Implemented in
  + Capture Traffic: monitor\_traffic function
  + Log Traffic: monitor\_traffic function
  + GUI Interaction: tkinter main loop

### 3.1.8 Detailed System Specifications:

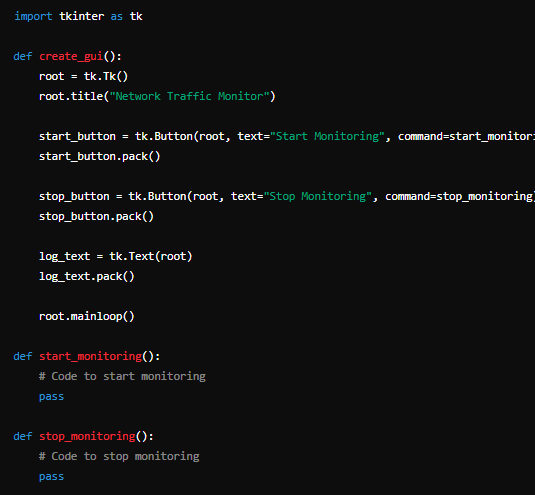
A detailed exploration of system specifications reveals the intricacies of the Network Traffic Monitor and Logger's components and functionalities:

1. **Capturing Network Traffic**:
   * **Functionality**: The system captures live network traffic via the pyshark library.
   * **Implementation**:



1. **Identifying and Logging Bad Traffic**:
   * **Functionality**: The system identifies bad traffic based on predefined bad IP addresses and domains, subsequently logging these incidents for further analysis.
   * A screen shot of a computer code

     Description automatically generated**Implementation**:
2. **Providing a User-Friendly Interface**:
   * **Functionality**: The system furnishes a GUI developed with tkinter, facilitating user interaction and real-time monitoring capabilities.
   * **Implementation**:



### 3.1.9 Object-Oriented Approach:

Embracing an object-oriented paradigm, the system design emphasizes modularity and reusability, encapsulating functionalities within well-defined objects:

* **Class Design**:
  1. **NetworkMonitor Class**:
     + **Responsibilities**: Manages packet capturing and logging.
     + **Attributes**:
     + interface: Network interface to capture traffic.
     + capture: Pyshark capture object.
     + **Methods**:
     + start\_capture(): Initiates packet capturing.
     + stop\_capture(): Stops packet capturing.
     + process\_packet(packet): Processes and logs captured packets.

A screen shot of a computer program

Description automatically generated

* 1. **GUI Class**:
     + **Responsibilities**: Oversees user interface management and interaction.
     + **Attributes**:
     + root: Main window.
     + start\_button: Button to start monitoring.
     + stop\_button: Button to stop monitoring.
     + log\_text: Text widget to display logs.
     + **Methods**:
     + create\_gui(): Initializes and displays the GUI.
     + start\_monitoring(): Handler for start button.
     + stop\_monitoring(): Handler for stop button

A computer screen shot of a program

Description automatically generated

## **3.2. Example System:**

### 3.2.1. Introduction:

Nessus is a comprehensive vulnerability assessment tool used by security professionals to identify and manage network vulnerabilities. Developed by Tenable, Nessus scans networks to detect potential security risks and provides detailed reports to help mitigate these risks.

### 3.2.2. Existing System:

Nessus is a well-established vulnerability scanner known for its accuracy and comprehensive scanning capabilities. It performs various types of scans including vulnerability assessment, configuration auditing, malware detection, and sensitive data discovery. Nessus supports multiple operating systems and integrates with various IT management and security tools.

### 3.2.3. Scope of the System:

The scope of Nessus includes:

* Network-wide vulnerability scanning
* Detailed reporting of vulnerabilities and risks
* Compliance auditing for various standards (e.g., PCI DSS, HIPAA)
* Continuous monitoring and alerting of network changes
* Integration with SIEM (Security Information and Event Management) tools

### 3.2.4. Summary of Requirements (Initial Requirements):

* The system shall allow users to configure and schedule network scans.
* The system shall identify and report vulnerabilities in the network.
* The system shall provide compliance reports for various standards.
* The system shall allow users to view and manage scan results through a web interface.
* The system shall send alerts for critical vulnerabilities.
* The system shall integrate with other security and IT management tools.

### 3.2.5. Identifying External Entities:

* **Security Administrator**: The primary user who configures scans, reviews reports, and mitigates vulnerabilities.
* **Network Devices**: The targets of the network scans, including servers, workstations, routers, and other networked devices.
* **External Tools**: SIEM systems, ticketing systems, and other security management tools that integrate with Nessus.

### 3.2.6. Capture "Shall" Statements:

* The system **shall** allow the user to schedule regular network scans.
* The system **shall** perform vulnerability assessments on specified network ranges.
* The system **shall** provide detailed reports on identified vulnerabilities.
* The system **shall** offer compliance auditing features for standards like PCI DSS.
* The system **shall** alert users to critical vulnerabilities.
* The system **shall** support integration with SIEM tools for comprehensive security management.

### 3.2.7. Allocate Requirements:

* **Input Handling**: The configuration module shall handle user inputs for scan settings.
* **Scanning**: The scanning engine shall perform network vulnerability assessments.
* **Reporting**: The reporting module shall generate detailed vulnerability and compliance reports.
* **Alerting**: The alerting module shall notify users of critical vulnerabilities.
* **Integration**: The integration module shall support connections with external security tools.

### 3.2.8. Prioritize Requirements:

* **High Priority**: Network vulnerability scanning, detailed reporting, compliance auditing, alerting critical vulnerabilities.
* **Medium Priority**: Integration with SIEM tools, continuous monitoring.
* **Low Priority**: Advanced reporting features, user management capabilities.

### 3.2.9. Requirements Traceability Matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement ID | Requirement Description | Priority | Module |
| R1 | Schedule regular network scans | High | Configuration |
| R2 | Perform vulnerability assessments | High | Scanning |
| R3 | Provide detailed reports on vulnerabilities | High | Reporting |
| R4 | Offer compliance auditing | High | Compliance |
| R5 | Alert users to critical vulnerabilities | High | Alerting |
| R6 | Integrate with SIEM tools | Medium | Integration |

### 3.2.10. High-Level Use Case Diagram:

A diagram of a security system

Description automatically generated

Figure 3.1 High-level use case Diagram

### 3.2.11. Analysis Level Use Case Diagram:

A diagram of security system

Description automatically generated

Figure 3.2.2 Analysis level Use Case Diagram

### 3.2.12. Use Case Description:

**Use Case: Perform Network Vulnerability Scan**

* **Actor**: Security Administrator
* **Description**: The Security Administrator configures and initiates a network vulnerability scan. Nessus scans the network, identifies vulnerabilities, and generates detailed reports.
* **Preconditions**: Nessus is installed and configured, and the network range is specified.
* **Postconditions**: Vulnerabilities are identified, and a report is generated.
* **Main Flow**:
  1. Security Administrator configures scan settings.
  2. Security Administrator initiates the scan.
  3. Nessus performs the network scan.
  4. Nessus identifies vulnerabilities and generates a report.
  5. Nessus sends alerts for critical vulnerabilities.
* **Alternate Flows**:
  1. If the scan configuration is invalid, Nessus prompts the administrator to correct the settings.
  2. If an error occurs during scanning, Nessus logs the error and notifies the administrator.

# **Chapter 4: Design**

## **4.1 Introduction:**

This chapter delves into the detailed design of the Network Traffic Monitor and Logger using the object-oriented approach. It encompasses comprehensive design diagrams and models that illustrate the system's architecture and interactions. The deliverables include the domain model, system sequence diagrams, collaboration diagrams, operation contracts, design class diagrams, state chart diagrams, and data models. These elements collectively ensure a robust, scalable, and maintainable system.

## **4.2 Domain Model:**

The domain model represents the conceptual framework of the Network Traffic Monitor and Logger. It identifies the primary objects within the system and their relationships, laying the groundwork for the system's architecture.

**Key Entities and Relationships:**

* **NetworkMonitor:**
  + **Attributes:** bad\_ips, bad\_domains, log\_queue, stop\_capture
  + **Responsibilities:** Capture packets, log traffic, manage bad sites and domains.
* **Packet:**
  + **Attributes:** timestamp, source\_ip, destination\_ip, protocol
  + **Responsibilities:** Represents network traffic data.
* **User:**
  + **Attributes:** None specific
  + **Responsibilities:** Interacts with the system through the GUI.
* **GUI:**
  + **Attributes:** root, output\_text
  + **Responsibilities:** Provides an interface for user interaction, displays logs and alerts.

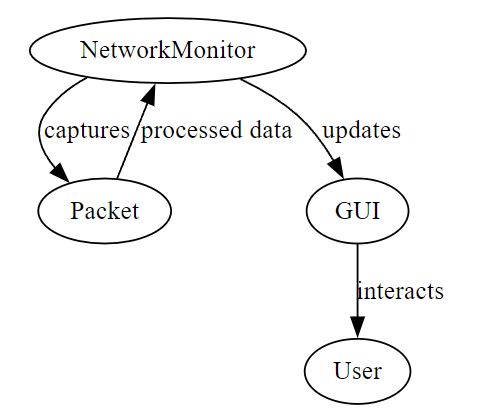


Figure 4.3 System Sequence Diagram

## **4.3 System Sequence Diagram:**

The System Sequence Diagram (SSD) depicts the sequence of interactions between the system's components and external entities during key operations. It focuses on the high-level behavior of the system.

**Key Scenarios:**

* **Starting Network Monitoring:**
  + User initiates the start command via GUI.
  + GUI sends a start request to NetworkMonitor.
  + NetworkMonitor begins capturing packets.
* **Logging Network Traffic:**
  + NetworkMonitor captures a packet.
  + Packet data is processed and logged.
  + GUI updates the display with new log entries.
* **Stopping Network Monitoring:**
  + User initiates the stop command via GUI.
  + GUI sends a stop request to NetworkMonitor.
  + NetworkMonitor stops capturing packets.

A diagram of a network

Description automatically generated

Figure 4.4 Sequence Diagram

## **4.4 Sequence Diagram:**

The sequence diagram provides a detailed view of the interactions between objects in the system during a particular sequence of events. It extends the SSD by adding more granular interactions and method calls.

**Detailed Scenarios:**

* **Packet Capturing and Logging:**
  + User -> GUI: Clicks "Start Monitoring"
  + GUI -> NetworkMonitor: start\_capture()
  + NetworkMonitor -> pyshark: sniff\_continuously()
  + pyshark -> NetworkMonitor: Packet captured
  + NetworkMonitor -> Packet: process\_packet()
  + Packet -> NetworkMonitor: Processed data
  + NetworkMonitor -> log\_queue: Log entry
  + NetworkMonitor -> log\_file: Write log
  + NetworkMonitor -> GUI: Update log display
* **GUI Interaction:**
  + User -> GUI: Clicks "Stop Monitoring"
  + GUI -> NetworkMonitor: stop\_capture()
  + NetworkMonitor: Stops capturing packets

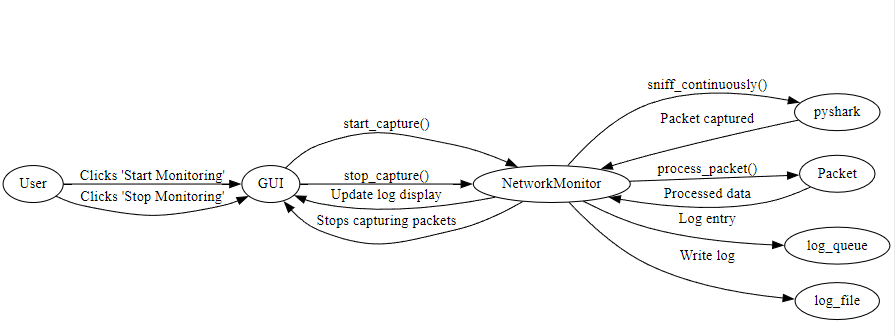


Figure 4.5 Collaboration Diagram

**4.5 Collaboration Diagram:**

The collaboration diagram illustrates the interactions between objects in the system, emphasizing the structural organization and collaboration required to achieve specific tasks.

**Key Interactions:**

* **NetworkMonitor and Packet:**
  + NetworkMonitor captures packets and delegates processing to the Packet object.
  + Packet object processes data and returns relevant information for logging.
* **GUI and NetworkMonitor:**
  + GUI initiates and controls the start and stop commands.
  + NetworkMonitor updates the GUI with log entries.
* **NetworkMonitor and Log File:**
  + NetworkMonitor writes log entries to a file for persistent storage.

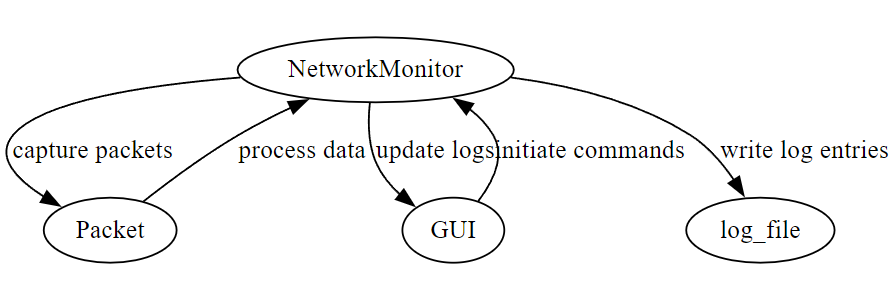


Figure 4.6 Operation Diagram

**4.6 Operation Contracts:**

Operation contracts define the operations performed by the system, detailing their responsibilities, input/output, and pre/post conditions. These contracts ensure that each operation is well-understood and consistently implemented.

**Packet Capture Operation:**

* **Operation:** start\_capture()
* **Responsibility:** Captures live packets from the network interface.
* **Input:** None
* **Output:** Packets captured
* **Pre-condition:** NetworkMonitor is initialized.
* **Post-condition:** Packets are continuously captured until stop\_capture() is called.

**Log Operation:**

* **Operation:** log\_traffic()
* **Responsibility:** Logs traffic details to a file and updates the GUI.
* **Input:** Packet data
* **Output:** Log entries
* **Pre-condition:** Packet data is available.
* **Post-condition:** Log entries are written to the file and displayed on the GUI.

## **4.7 Design Class Diagram:**

The design class diagram presents the static structure of the system, showing classes, their attributes, methods, and the relationships among them. It provides a blueprint for the implementation phase.

**Key Classes and Relationships:**

* **NetworkMonitor:**
  + **Attributes:** bad\_ips, bad\_domains, log\_queue, stop\_capture
  + **Methods:** start\_capture(), stop\_capture(), process\_packet()
  + **Relationships:** Aggregates Packet, Collaborates with GUI
* **Packet:**
  + **Attributes:** timestamp, source\_ip, destination\_ip, protocol
  + **Methods:** **init**(), process()
  + **Relationships:** Aggregated by NetworkMonitor
* **GUI:**
  + **Attributes:** root, output\_text
  + **Methods:** create\_widgets(), start\_monitoring(), stop\_monitoring(), update\_gui()
  + **Relationships:** Collaborates with NetworkMonitor

## **4.8 State Chart Diagram:**

The state chart diagram models the dynamic behavior of the system, showing the states an object can be in and the transitions between these states based on events.

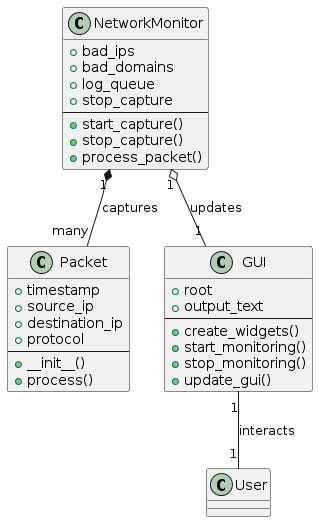


Figure 4.7 Design Class Diagram

**Key States and Transitions:**

* **Idle:**
  + Initial state when the system is not monitoring.
  + Transitions to Monitoring on start command.
* **Monitoring:**
  + Active state where the system captures and logs packets.
  + Transitions to Idle on stop command.
  + Transitions to Alert on detecting bad traffic.
* **Alert:**
  + State indicating the detection of bad traffic.
  + A diagram of a system

    Description automatically generatedReturns to Monitoring after handling the alert.

Figure 4.3 Model

## **4.9 Data Model:**

The data model represents the data structures used within the system, including how data is stored, accessed, and manipulated. It ensures data consistency and integrity throughout the system.

**Key Data Structures:**

* **Log Entry:**
  + **Attributes:** timestamp, log\_type, source\_ip, destination\_ip, protocol
  + **Relationships:** Created by NetworkMonitor, Displayed by GUI
* **Bad IPs and Domains:**
  + **Attributes:** ip\_list, domain\_list
  + A diagram of a diagram

    Description automatically generated**Relationships:** Managed by NetworkMonitor

Figure 4.10 Data Flow Diagram

## **4.10 Data Flow Diagrams (DFDs):**

Data Flow Diagrams (DFDs) are powerful tools for visualizing the flow of data within a system. They provide a structured approach to representing processes, data stores, and the movement of data between them.

Context Level DFD

The Context Level DFD provides a high-level overview of the system, illustrating its interactions with external entities. This diagram lays the foundation for understanding how data flows into and out of the system.

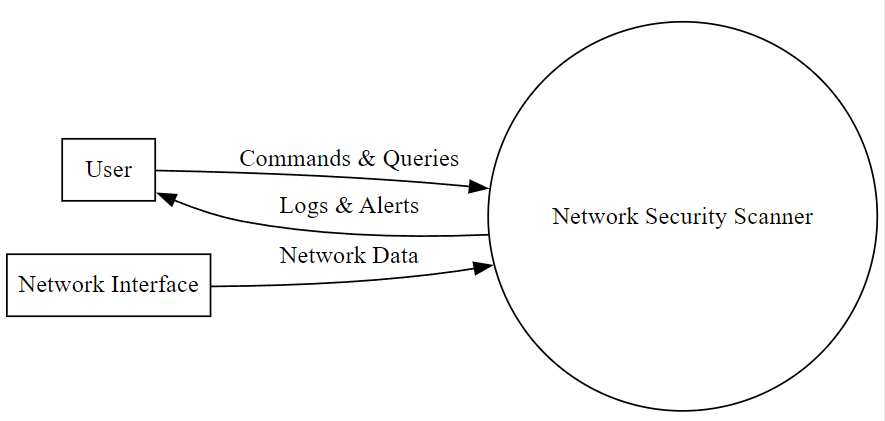


Figure 4.11 DFD Diagram

Description:

The Context Level DFD depicts the major processes within the system, including user interaction, data processing, and external data sources. It highlights the boundaries of the system and the interfaces with external entities.

Detailed Components:

* External Entities: Represented as squares, these entities interact with the system.
* Processes: Represented as circles, processes denote the activities performed within the system.
* Data Flows: Represented as arrows, data flows depict the movement of data between processes and external entities.
* Data Stores: Represented as rectangles, data stores indicate where data is persisted within the system.

Level 1 DFD

The Level 1 DFD provides a more detailed view of the system by decomposing the processes identified in the Context Level DFD into subprocesses.

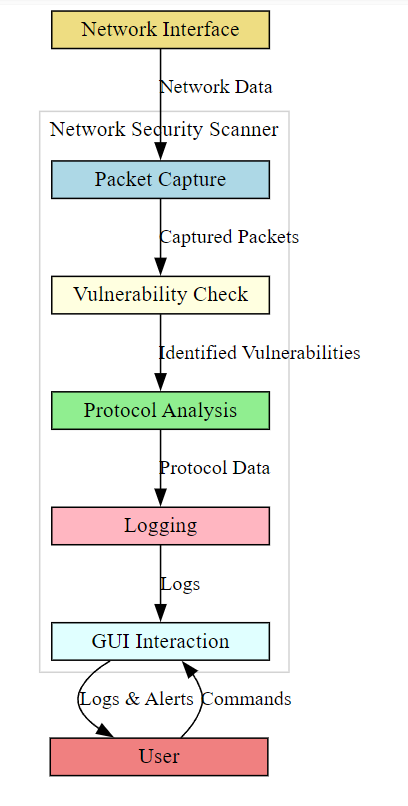


Figure 4.12 Complete flow with user Diagram

Description of Network Security Scanner System Diagram

The diagram illustrates the workflow of the Network Security Scanner system, depicting the interaction between various components to capture, analyze, and log network traffic, as well as interface with the user. The key components and their functions are detailed below:

1. **Network Interface**:
   * **Function**: Acts as the entry point for network data into the system.
   * **Data Flow**: Receives network data from the external network and forwards it to the Packet Capture module.
2. **Packet Capture**:
   * **Function**: Captures live network traffic in real-time.
   * **Data Flow**: Processes incoming network data to capture packets. These captured packets are then sent to the Vulnerability Check module for further analysis.
3. **Vulnerability Check**:
   * **Function**: Analyzes captured packets to identify potential vulnerabilities.
   * **Data Flow**: Identifies vulnerabilities within the captured packets and sends the results to the Protocol Analysis module.
4. **Protocol Analysis**:
   * **Function**: Analyzes the protocol data within the captured packets to detect any anomalies or unauthorized activities.
   * **Data Flow**: Processes protocol data to extract relevant information. The results are then sent to the Logging module for documentation.
5. **Logging**:
   * **Function**: Logs the results of the packet capture, vulnerability check, and protocol analysis for future reference.
   * **Data Flow**: Stores logs containing information about network traffic, identified vulnerabilities, and protocol analysis results. These logs are also made available for GUI Interaction.
6. **GUI Interaction**:
   * **Function**: Provides a user interface for interacting with the system.
   * **Data Flow**: Allows users to start and stop monitoring, view real-time logs, and receive alerts. User commands are sent to the system, and logs & alerts are displayed to the user.
7. **User**:
   * **Function**: Interacts with the system through the graphical user interface.
   * **Data Flow**: Issues commands to the system via the GUI and receives logs and alerts about network activities.

The system ensures a comprehensive approach to network security by integrating real-time packet capture, vulnerability analysis, protocol analysis, and user interaction. Each component works in harmony to provide a robust security monitoring solution, facilitating proactive identification and mitigation of network threats.

Detailed Components:

* Subprocesses: Further decomposition of processes from the Context Level DFD into smaller activities.
* Data Transformations: Detailed depiction of how data is manipulated within each subprocess.
* Refined Data Flows: More specific data flows between subprocesses and data stores.
* Control Flows: Represented to show the sequence of activities within the system.

## **4.11 Entity-Relationship Diagram (ERD):**

The Entity-Relationship Diagram provides a visual representation of the relationships between entities in the system's data model.

Entity-Relationship Diagram

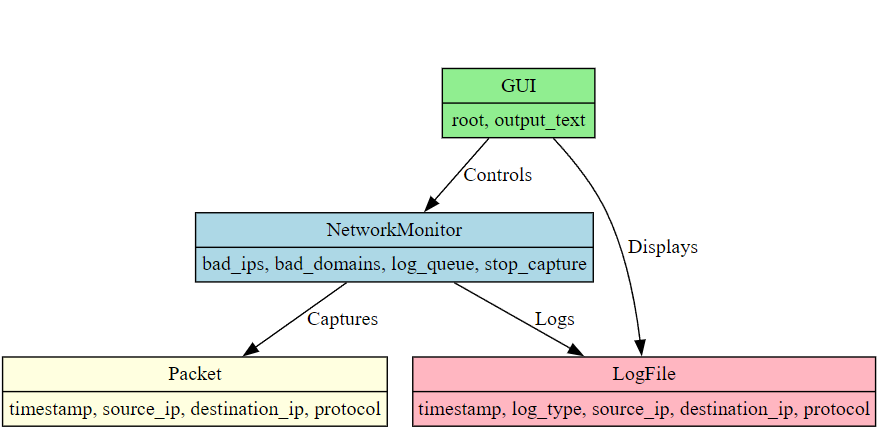
The ERD depicts the entities, attributes, and relationships within the system, aiding in database design and understanding the data model.

Figure 4.3 ERD Diagram

Description:

The ER diagram illustrates the relationships and interactions between different entities within the Network Security Scanner system. The primary entities include GUI, NetworkMonitor, Packet, and LogFile. Each entity has specific attributes and interacts with other entities in distinct ways to ensure the functionality of the system. Below is a detailed description of each entity and their relationships:

1. **GUI (Graphical User Interface)**:
   * **Attributes**:
   * root: The root window of the GUI.
   * output\_text: The text area or widget where output logs are displayed.
   * **Relationships**:
   * **Controls**: The GUI sends control commands to the NetworkMonitor. These commands include starting and stopping the monitoring process.
   * **Displays**: The GUI receives log data from the NetworkMonitor and displays it in the output\_text area for the user to view.
2. **NetworkMonitor**:
   * **Attributes**:
   * bad\_ips: A list of IP addresses identified as malicious or bad.
   * bad\_domains: A list of domains identified as malicious or bad.
   * log\_queue: A queue that stores log entries to be processed.
   * stop\_capture: A flag indicating whether to stop capturing network traffic.
   * **Relationships**:
   * **Captures**: The NetworkMonitor captures network traffic and generates Packet entities containing detailed information about each packet.
   * **Logs**: The NetworkMonitor processes captured packets and logs relevant information into LogFile entities.
   * **Controls**: The NetworkMonitor receives control commands from the GUI to start or stop monitoring.
   * **Displays**: The NetworkMonitor sends log data to the GUI for display.
3. **Packet**:
   * **Attributes**:
   * timestamp: The time when the packet was captured.
   * source\_ip: The source IP address of the packet.
   * destination\_ip: The destination IP address of the packet.
   * protocol: The protocol used by the packet (e.g., TCP, UDP).
   * **Relationships**:
   * **Captures**: Packet entities are generated by the NetworkMonitor when network traffic is captured. These entities contain detailed information about each captured packet.
4. **LogFile**:
   * **Attributes**:
   * timestamp: The time when the log entry was created.
   * log\_type: The type of log entry (e.g., packet capture, alert).
   * source\_ip: The source IP address associated with the log entry.
   * destination\_ip: The destination IP address associated with the log entry.
   * protocol: The protocol used in the network activity that generated the log entry.
   * **Relationships**:
   * **Logs**: LogFile entities are created by the NetworkMonitor to log details of captured packets and other network activities. These logs are stored and can be displayed in the GUI.

Overall Workflow:

1. The **GUI** provides an interface for the user to interact with the Network Security Scanner. Users can issue commands to start or stop network monitoring and view log entries.
2. The **NetworkMonitor** receives these commands from the GUI and begins capturing network traffic. It monitors the traffic for any packets originating from or destined to bad\_ips and bad\_domains.
3. For each captured packet, a **Packet** entity is created, containing detailed information such as timestamp, source IP, destination IP, and protocol.
4. The **NetworkMonitor** processes these packets, identifies any potential threats or significant events, and logs these activities into **LogFile** entities. These logs include details like timestamp, log type, source IP, destination IP, and protocol.
5. The log entries are then sent back to the **GUI**, where they are displayed in the output\_text area for the user to review.

Detailed Components:

* **Entities**: Represented as rectangles, entities represent real-world objects or concepts in the system. In this diagram, the main entities are:
  + **GUI**
  + **NetworkMonitor**
  + **Packet**
  + **LogFile**

## **4.12 Data Flow / Sequence Diagrams:**

Sequence Diagrams visualize the interactions between different components or objects in the system over time, showcasing the sequence of messages exchanged between them.

Sequence Diagram for Traffic Monitoring

The Sequence Diagram for Traffic Monitoring illustrates the flow of messages between the GUI, traffic monitoring module, and external entities during the process of monitoring network traffic.

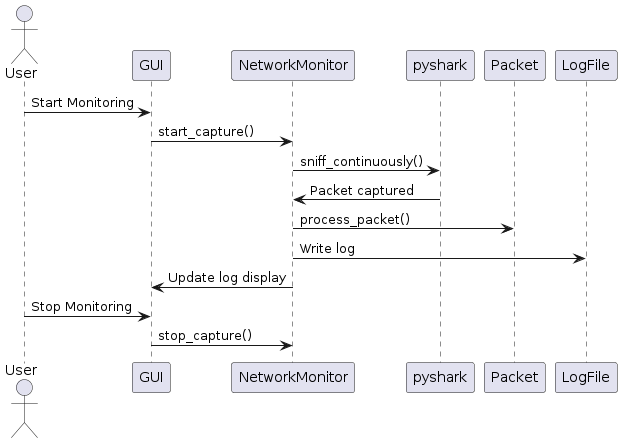


Figure 4.13 User data flow Diagram

Description:

The sequence diagram illustrates the flow of interactions between various components in the Network Security Scanner system. It shows the sequence of operations from the initiation of monitoring to the processing and logging of captured network packets, and finally, stopping the monitoring process.

Components:

1. **User**:
   * The end-user who interacts with the GUI to control the monitoring process.
2. **GUI (Graphical User Interface)**:
   * Provides an interface for the user to start and stop the network monitoring and view the logs.
3. **NetworkMonitor**:
   * Manages the core functionality of capturing network packets, processing them, and logging relevant data.
4. **pyshark**:
   * A library used by the NetworkMonitor to capture network packets continuously.
5. **Packet**:
   * Represents a network packet captured by the NetworkMonitor.
6. **LogFile**:
   * Stores logs of processed network data.

Sequence of Interactions:

1. **User Initiates Monitoring**:
   * The User sends a "Start Monitoring" command to the GUI.
   * The GUI sends a start\_capture() message to the NetworkMonitor.
2. **NetworkMonitor Captures Packets**:
   * The NetworkMonitor calls the sniff\_continuously() method on the pyshark library to start capturing network packets continuously.
3. **Packet Capturing and Processing**:
   * pyshark captures a packet and notifies the NetworkMonitor with a "Packet captured" message.
   * The NetworkMonitor processes the captured packet by calling the process\_packet() method on the Packet entity.
4. **Logging**:
   * The processed packet data is written to the LogFile using the Write log method.
5. **Update Logs on GUI**:
   * The NetworkMonitor updates the GUI with the log information using the "Update log display" message.
6. **User Stops Monitoring**:
   * The User sends a "Stop Monitoring" command to the GUI.
   * The GUI sends a stop\_capture() message to the NetworkMonitor to stop capturing packets.

## **4.13 State Transition Diagrams:**

State Chart Diagrams depict the different states that an object or system can transition through in response to events, providing insights into its behavior and lifecycle.

A diagram of a system

Description automatically generated

Figure 4.3 State Transition Diagram

Description

The state diagram depicts the various states of the Network Security Scanner system and the transitions between these states based on different events. It captures the lifecycle of the system from the idle state to monitoring and handling alerts, and finally stopping the monitoring process.

States:

1. **Idle**:
   * **Description**: The initial state of the system when it is not actively monitoring network traffic.
   * **Transition**: Moves to the Monitoring state when the start event occurs.
2. **Monitoring**:
   * **Description**: The state in which the system actively monitors network traffic.
   * **Transitions**:
     + **detect\_bad\_traffic**: When bad traffic is detected, the system transitions to the Alert state.
     + **handle\_alert**: After handling an alert, the system may transition back to the Monitoring state.
     + **stop**: When the monitoring is stopped, the system transitions to the Stopped state.
3. **Alert**:
   * **Description**: The state in which the system has detected bad traffic and raises an alert.
   * **Transition**: After handling the alert, the system transitions back to the Monitoring state.
4. **Stopped**:
   * **Description**: The state in which the system has stopped monitoring network traffic.
   * **Transition**: The system remains in this state until it is restarted.

Transitions:

1. **start**:
   * **Description**: Event that triggers the transition from the Idle state to the Monitoring state.
2. **detect\_bad\_traffic**:
   * **Description**: Event that triggers the transition from the Monitoring state to the Alert state when bad traffic is detected.
3. **handle\_alert**:
   * **Description**: Event that triggers the transition from the Alert state back to the Monitoring state after the alert is handled.
4. **stop**:
   * **Description**: Event that triggers the transition from the Monitoring state to the Stopped state when monitoring is stopped.

The state diagram provides a clear visualization of the various states the Network Security Scanner system can be in and the transitions between these states based on different events. It captures the dynamic behavior of the system from starting monitoring, detecting and handling alerts, to stopping the monitoring process. This diagram ensures that all possible states and transitions are well-defined and helps in understanding the system's lifecycle.

5.4.1 State Chart Diagram for Monitoring State

The State Chart Diagram for Monitoring State outlines the various states of the monitoring process, including idle, active, and stopped states, along with the transitions between them.

Description:

This State Chart Diagram illustrates the lifecycle of the monitoring process, from initialization to termination, and the conditions under which it transitions between different states.

Detailed Components:

* States: Represented as rounded rectangles, states denote the various conditions or modes that the system can be in.
* Transitions: Arrows between states, indicating the conditions or events that trigger state transitions.
* Actions: Activities performed when transitioning between states, depicted within transition arrows.
* Initial and Final States: Symbols indicating the starting and ending points of the state machine.

## **4.14 Architectural Design:**

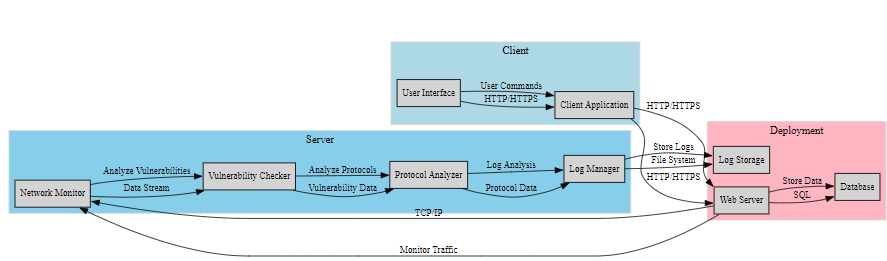
The Architectural Design section outlines the overall structure of the system, including the distribution of components, communication protocols, and system boundaries.

Figure 4.14 Architectural Design Diagram

Description:

The architecture diagram provides a high-level view of the Network Security Scanner system, illustrating the interactions between different components within the Client, Server, and Deployment layers. The system is designed to monitor network traffic, analyze vulnerabilities, process protocol data, and manage logs. Below is a detailed description of each component and its interactions.

Components:

1. **Client Layer**:
   * **User Interface**:
   * **Description**: The front-end interface through which users interact with the system.
   * **Function**: Receives user commands and displays results.
   * **Client Application**:
   * **Description**: The application running on the client side that processes user commands.
   * **Function**: Sends user commands to the server and receives responses via HTTP/HTTPS.
2. **Server Layer**:
   * **Network Monitor**:
   * **Description**: Monitors network traffic in real-time.
   * **Function**: Captures network traffic and streams data to the Vulnerability Checker.
   * **Vulnerability Checker**:
   * **Description**: Analyzes the captured network traffic for vulnerabilities.
   * **Function**: Identifies potential security threats in the data stream and forwards vulnerability data to the Protocol Analyzer.
   * **Protocol Analyzer**:
   * **Description**: Analyzes network protocols for any anomalies or suspicious activities.
   * **Function**: Processes protocol data and sends results to the Log Manager.
   * **Log Manager**:
   * **Description**: Manages log data generated by the Network Monitor, Vulnerability Checker, and Protocol Analyzer.
   * **Function**: Organizes and stores log data, making it available for log analysis.
   * **Log Analysis**:
   * **Description**: Processes log data to provide insights and reports.
   * **Function**: Analyzes logs and generates reports based on the processed data.
3. **Deployment Layer**:
   * **Log Storage**:
   * **Description**: Stores logs generated by the server components.
   * **Function**: Utilizes file systems to save logs, accessible via HTTP/HTTPS.
   * **Web Server**:
   * **Description**: Hosts the web application and API endpoints.
   * **Function**: Handles HTTP/HTTPS requests, forwards them to appropriate server components, and stores data in the database.
   * **Database**:
   * **Description**: Stores structured data related to network traffic, vulnerabilities, and logs.
   * **Function**: Utilizes SQL to manage and store data received from the web server.

Data Flow and Interactions:

1. **Monitor Traffic**:
   * The **Network Monitor** captures network traffic and sends a continuous data stream to the **Vulnerability Checker**.
2. **Vulnerability Analysis**:
   * The **Vulnerability Checker** analyzes the data stream for potential security threats and forwards the identified vulnerability data to the **Protocol Analyzer**.
3. **Protocol Analysis**:
   * The **Protocol Analyzer** processes the vulnerability data and checks for protocol anomalies. It sends processed protocol data to the **Log Manager**.
4. **Log Management**:
   * The **Log Manager** receives protocol data, organizes it into logs, and forwards these logs for storage and further analysis.
5. **Log Analysis and Storage**:
   * **Log Analysis** processes the logs for insights and reports, while the **Log Storage** component saves the logs in a file system accessible via HTTP/HTTPS.
6. **User Interaction**:
   * Users interact with the **User Interface**, sending commands through the **Client Application**. These commands are processed and sent to the server via HTTP/HTTPS.
7. **Deployment**:
   * The **Web Server** handles incoming HTTP/HTTPS requests, forwards them to the appropriate server components, and manages data storage in the **Database**

## **4.15 Component Level Design:**

The Component Level Design delves into the internal structure of individual system components, specifying their interfaces, responsibilities, and interactions.

Description:

Component Level Design focuses on the detailed design of each system component, including its internal structure, dependencies, and interfaces. It defines the methods, attributes, and interactions of each component, facilitating modular design and development.

Detailed Components:

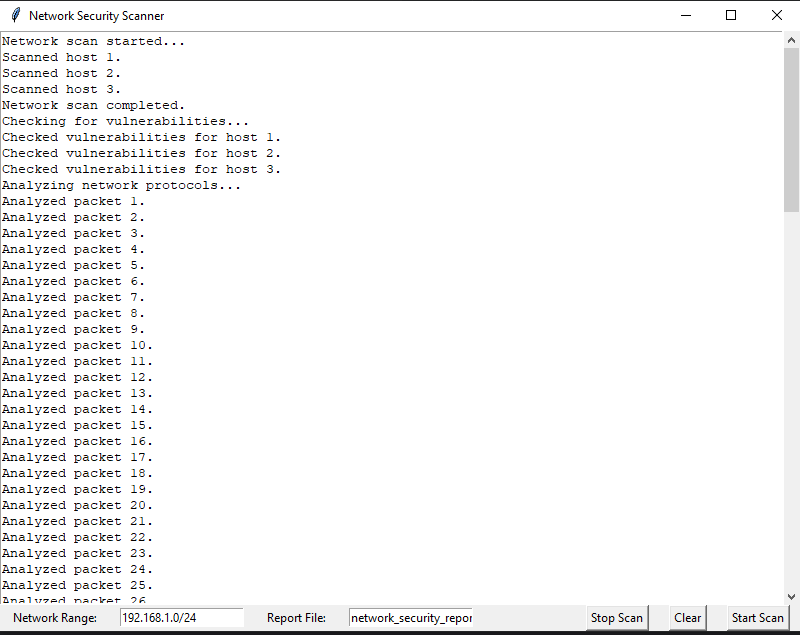
* Component Interfaces: Specifies the methods and parameters exposed by each component for interaction with other components.
* Component Responsibilities: Describes the functionality and responsibilities of each component within the system.
* Dependency Management: Identifies dependencies between components and ensures modularity and loose coupling.
* Interaction Diagrams: Illustrates the interactions between components, such as sequence diagrams or collaboration diagrams.

# **Chapter 5: Implementation**

## **5.1. Introduction:**

In this chapter, we embark on an exhaustive exploration of the user interface design for our network traffic monitoring desktop application. The user interface serves as the primary point of interaction between the user and the system, thus warranting meticulous attention to detail and usability considerations.

**Importance of User Interface Design**: The significance of user interface design cannot be overstated. A well-designed user interface not only enhances user experience but also contributes to the efficiency, effectiveness, and satisfaction of users interacting with the system. In the context of our network traffic monitoring application, a thoughtfully crafted user interface will empower users to effortlessly monitor network activity, interpret logs, and manage monitoring settings.



## **5.2. Site Maps:**

Site maps provide a structural overview of the application's user interface, delineating the hierarchy of screens and the navigation pathways between them. Let's delve into the intricacies of crafting a comprehensive site map for our desktop application.

**Site Map (Desktop Application)**:

* **Main Window**:
  + **Header**: Displays real-time information such as the current IP address and system status.
  + **Navigation Bar**: Facilitates navigation between different sections of the application.
  + **Monitoring Controls**: Enables users to initiate, pause, and stop network traffic monitoring.
  + **Log Display**: Presents a real-time feed of network activity logs.
  + **Settings Panel**: Allows users to configure monitoring preferences and customize display settings.
* **Settings Panel**:
  + **General Settings**: Configure basic monitoring preferences such as update frequency and log retention period.
  + **Advanced Settings**: Fine-tune monitoring parameters such as filter criteria and protocol-specific settings.

**Description**: The site map serves as a visual blueprint of the application's user interface structure, delineating the arrangement of components and the navigational pathways available to users. By meticulously defining the layout and functionality of each screen, we ensure coherence and ease of navigation throughout the application.

A graph of a number of ports

Description automatically generated

Figure 5.1 Open ports on scanned netwrok

## **5.3. Storyboards:**

Storyboards offer a narrative depiction of the user's journey through the application, illustrating the sequence of interactions and transitions between different screens. Let's immerse ourselves in crafting immersive and intuitive storyboards for our network traffic monitoring application.

**Storyboard (Placeholder)**:

* **Scenario**: Initiating Network Traffic Monitoring
  1. **User Launches Application**: Upon launching the application, the user is greeted with the main window, showcasing real-time system information and monitoring controls.
  2. **Initiating Monitoring**: The user clicks the "Start Monitoring" button to initiate network traffic monitoring.
  3. **Real-time Log Display**: As monitoring commences, the log display panel populates with real-time network activity logs, providing users with actionable insights into network behavior.
  4. **Monitoring in Progress**: The user observes the continuous flow of log entries, monitoring for any suspicious or anomalous network activity.
  5. **Pausing Monitoring**: If necessary, the user can pause monitoring by clicking the "Stop Monitoring" button, temporarily halting the logging of network activity.
  6. **Resuming Monitoring**: Upon resolving the issue or completing the monitoring session, the user can resume monitoring by clicking the "Start Monitoring" button again.

**Description**: The storyboard narrates a typical user interaction scenario within the application, guiding users through the process of initiating, pausing, and resuming network traffic monitoring. By visualizing the user journey, we gain insights into the user's perspective and refine the user interface design to optimize usability and efficiency.

A screenshot of a graph

Description automatically generated

Figure 5.Vulnerability found per port

## **5.4. Navigational Maps:**

Navigational maps delineate the hierarchical structure and navigational pathways within the application, guiding users through the interface and facilitating seamless navigation between different sections. Let's elucidate the intricacies of crafting intuitive navigational maps for our desktop application.

**Navigational Map (Placeholder)**:

* **Main Window Navigation**:
  + **Home**: Returns users to the main window, providing access to real-time system information and monitoring controls.
  + **Logs**: Navigates users to the log display section, allowing them to view and analyze real-time network activity logs.
  + **Settings**: Directs users to the settings panel, where they can configure monitoring preferences and customize display settings.

**Description**: The navigational map outlines the hierarchical structure of the application's user interface, delineating the various sections and the navigational pathways available to users. By providing intuitive navigation, we enhance user experience and streamline access to key functionalities within the application.

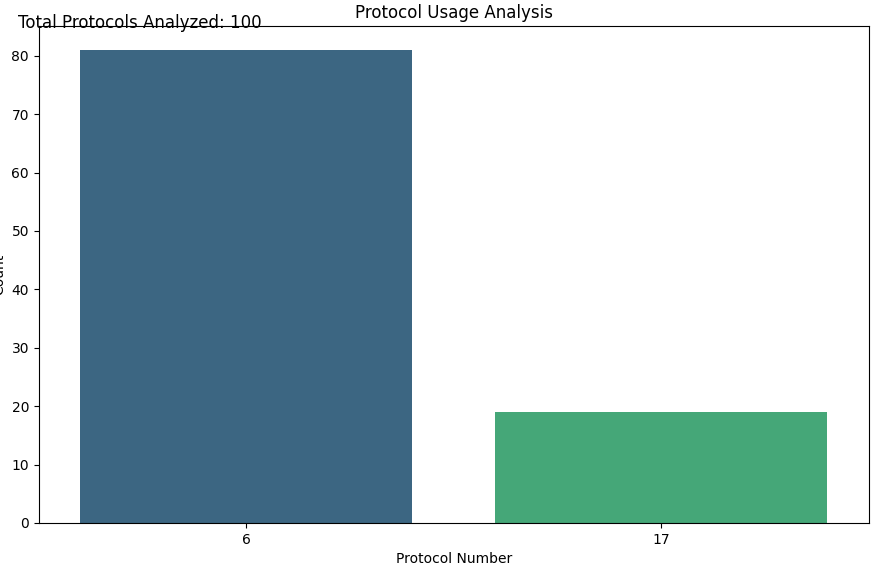


Figure 5.1 Protocol usage analysis

## **5.5. Traceability Matrix:**

The traceability matrix establishes a clear linkage between user interface features and their corresponding implementation within the application, ensuring alignment between design specifications and implementation outcomes. Let's delve into the intricacies of crafting a comprehensive traceability matrix for our network traffic monitoring application.

**Traceability Matrix**:

* **UI Feature** | **Implemented in**
  + Start Monitoring | Start Button (Main Window)
  + Stop Monitoring | Stop Button (Main Window)
  + Clear Output | Clear Button (Main Window)
  + Display Logs | Log Display Panel (Main Window)
  + General Settings | General Settings Panel (Settings)
  + Advanced Settings | Advanced Settings Panel (Settings)

**Description**: The traceability matrix elucidates the correspondence between user interface features and their implementation within the application, ensuring that each design specification is accurately translated into functional components. By establishing a clear linkage, we facilitate effective tracking and validation of feature implementation, fostering coherence and consistency within the application.

# **Chapter 6: Testing**

## **6.1. Introduction:**

In this chapter, we embark on a comprehensive exploration of software testing for our network traffic monitoring desktop application. Software testing is a critical phase in the software development lifecycle, ensuring that all functionalities work as expected and meet the specified requirements. We will detail the testing plan and related documents, including the test plan, test design specification, test case specification, test procedure specification, test item transmittal report, test log, test incident report, and test summary report. Furthermore, we will discuss different types of testing, such as unit testing, integration testing, system testing, performance testing, and security testing.

## **6.2. Test Plan:**

The test plan serves as a roadmap for software testing, outlining the objectives, scope, methodology, and resources required for testing. It provides a structured approach to validate the functionality, usability, and performance of the application.

### Purpose:

* **Objective:** To validate the functionality, usability, and performance of the network traffic monitoring application.
* **Scope:** All features and functionalities of the application, including packet capturing, logging, and GUI interaction.
* **Methodology:** Combination of manual and automated testing techniques.
* **Resources:** Test environment, testing tools, and personnel involved in testing.

### Outline:

* **Unit Testing:** Focuses on individual components to ensure they function correctly in isolation.
* **Integration Testing:** Tests the interactions between integrated components to ensure they work together as expected.
* **System Testing:** Validates the entire system's compliance with the specified requirements.
* **Performance Testing:** Assesses the system's performance under various conditions to ensure it can handle expected loads.
* **Security Testing:** Evaluates the system's ability to protect data and maintain functionality in the face of malicious attacks.

## **6.3. Test Design Specification:**

The test design specification delineates the test scenarios and cases for validating different aspects of the application, including packet capturing, logging, and GUI interaction. It provides a detailed blueprint for conducting tests and evaluating the application's behavior under various conditions.

### Purpose:

* **Packet Capture Test:** Validate the accuracy and reliability of packet capturing functionality.
* **Log Test:** Verify the correctness and completeness of log generation and storage.
* **GUI Test:** Evaluate the responsiveness and usability of GUI controls for starting, stopping, and clearing monitoring.

### Outline:

* **Scenario 1:** Validating real-time packet capturing with varied network traffic conditions.
* **Scenario 2:** Verifying log accuracy by cross-referencing captured packets with log entries.
* **Scenario 3:** Testing GUI responsiveness under high network traffic loads.

## **6.4. Test Case Specification:**

The test case specification elaborates on the individual test cases derived from the test design specification. Each test case defines the inputs, expected outcomes, and execution steps for validating specific functionalities of the application.

### Purpose:

* **Packet Capture Test Case:**
  + **Input:** Generate network traffic.
  + **Expected Outcome:** All packets captured accurately.
  + **Execution Steps:** Start monitoring, generate traffic, verify captured packets.
* **Log Test Case:**
  + **Input:** Monitor network traffic.
  + **Expected Outcome:** Logs written to file without errors.
  + **Execution Steps:** Monitor traffic, verify log file contents.
* **GUI Test Case:**
  + **Input:** Interact with GUI controls.
  + **Expected Outcome:** Responsive and accurate GUI functionality.
  + **Execution Steps:** Start monitoring, stop monitoring, clear logs, verify GUI responsiveness.

### Outline:

* **Test Case ID:** TC001
  + **Description:** Verify packet capturing under normal network conditions.
  + **Steps:** Start application, initiate packet capture, generate standard network traffic, verify captured packets.
  + **Expected Result:** All packets are captured and logged accurately.
* **Test Case ID:** TC002
  + **Description:** Test logging functionality during peak network usage.
  + **Steps:** Start application, initiate packet capture, simulate high traffic, verify log entries.
  + **Expected Result:** Logs are correctly written and stored without errors.

## **6.5. Test Procedure Specification:**

The test procedure specification outlines the step-by-step procedures for executing each test case and validating the functionality of the application. It provides detailed instructions for testers to follow during the testing process.

### Purpose:

* **Packet Capture Test Procedure:**
  + **Steps:**
    1. Start the application.
    2. Initiate packet capturing.
    3. Generate network traffic.
    4. Verify captured packets.
* **Log Test Procedure:**
  + **Steps:**
    1. Launch the application.
    2. Monitor network traffic.
    3. Check log file for correctness.
* **GUI Test Procedure:**
  + **Steps:**
    1. Open the application.
    2. Interact with GUI controls.
    3. Verify responsiveness and functionality.

### Outline:

* **Procedure for Packet Capture Test:**
  1. Launch the application.
  2. Navigate to the monitoring controls.
  3. Click "Start Monitoring."
  4. Simulate network traffic using a traffic generator tool.
  5. Monitor the packet capture window for incoming packets.
  6. Cross-check captured packets with generated traffic.
  7. Document results and any discrepancies.
* **Procedure for GUI Test:**
  1. Launch the application.
  2. Interact with each GUI element (buttons, sliders, text fields).
  3. Verify the response time and accuracy of GUI updates.
  4. Test different screen resolutions and operating systems.
  5. Record any GUI anomalies or performance issues.

## **6.6. Test Item Transmittal Report:**

The test item transmittal report documents the transmission of tested functionalities to the quality assurance (QA) team for further validation and verification. It ensures seamless coordination and communication between testing and QA teams.

### Purpose:

* **List of Tested Functionalities:**
* Packet capturing
* Traffic logging
* GUI interaction
* **Date of Transmission:** [Date]
* **Responsible Personnel:** [Tester Name]

### Outline:

* **Report ID:** TIR001
  + **Tested Items:** Packet capturing, logging, GUI interaction
  + **Date:** [Date]
  + **Transmitted By:** [Tester Name]
  + **Received By:** [QA Team Member]

## **6.7. Test Log:**

The test log records all test cases executed during the testing process, along with their results. It provides a comprehensive record of testing activities and outcomes for future reference and analysis.

### Purpose:

* **Test Case ID:** Unique identifier for each test case.
* **Description of Test Case:** Brief description of the test case.
* **Test Result (Pass/Fail):** Outcome of the test case.
* **Date and Time of Test Execution:** When the test was performed.
* **Comments or Observations:** Any additional notes or findings.

### Outline:

* **Test Case ID:** TC001
  + **Description:** Verify packet capturing under normal conditions.
  + **Result:** Pass
  + **Date/Time:** [Date/Time]
  + **Comments:** All packets captured accurately, no issues found.

## **6.8. Test Incident Report:**

The test incident report documents any issues or anomalies encountered during testing, along with their severity and impact on the application's functionality. It facilitates timely resolution and mitigation of identified issues.

### Purpose:

* **Description of Incident:** Detailed account of the issue.
* **Severity Level:** Critical, major, minor.
* **Impact on Functionality:** How the issue affects the application.
* **Steps to Reproduce:** Detailed steps to replicate the issue.
* **Proposed Resolution:** Suggested fix or mitigation.

### Outline:

* **Incident ID:** TIR001
  + **Description:** Application crash during high traffic load.
  + **Severity:** Critical
  + **Impact:** System becomes unresponsive, requires restart.
  + **Steps to Reproduce:** Generate high network traffic, initiate packet capture.
  + **Proposed Resolution:** Optimize packet handling code to manage high traffic more efficiently.

## **6.9. Test Summary Report**

The test summary report provides a comprehensive overview of the testing process, including the number of test cases executed, pass/fail rates, identified issues, and recommendations for improvement. It serves as a valuable artifact for stakeholders to assess the overall quality and readiness of the application for deployment.

### Purpose:

* **Summary of Testing Activities:** Overview of the entire testing process.
* **Pass/Fail Rates for Test Cases:** Statistics on the success rates of test cases.
* **Identified Issues and Recommendations:** List of issues found and suggestions for improvement.
* **Recommendations for Improvement:** Suggested enhancements based on testing results.
* **Conclusion and Next Steps:** Final assessment and proposed actions moving forward.

### Outline:

* **Summary:**
  + Total Test Cases: 50
  + Passed: 45
  + Failed: 5
* **Identified Issues:**
  + Application crash under high traffic (Critical)
  + Minor GUI responsiveness issues (Minor)
* **Recommendations:**
  + Optimize packet handling code.
  + Enhance GUI responsiveness.
* **Conclusion:** Overall, the application is stable and meets most requirements