Minor Project Report From chapter 1

by Yash Rana

Submission date: 25-Apr-2025 06:40PM (UTC+0530)
Submission ID: 2656571257
File name: Minor_Project_Report_from_chapter_1.docx (1.52M)
Word count: 2583
Character count: 15307

INTRODUCTION

1.1 Introduction and Problem Summary

In the modern digital age, with ever-changing cyber threats and malicious network activity, network traffic monitoring in real-time is crucial in maintaining cybersecurity and network performance. [1] Applications such as Wireshark provide powerful packet analysis capabilities but can be overly complex and have a high learning curve for novices and experts alike who are looking for ease of use and light-weight solutions. [2]

There is a strong requirement for a user-friendly, GUI-oriented network sniffer product that supports simple and advanced network traffic inspection with minimal setup. [3] Traditional packet sniffers usually work in command-line interfaces or need thorough understanding of network protocols, so they become less friendly to newbies, junior security analysts, or incident response teams looking for quick insights. [4] [5]

This deficiency is filled by this project by presenting ScapyUI, a GUI tool developed using Python that leverages the strength of the Scapy library for packet sniffing but presents a neat and interactive user interface developed using Tkinter. It enables users to sniff live network traffic, dissect protocol-specific data (such as HTTP, DNS, TCP/UDP), read .pcap files, and save logs efficiently for documentation or forensic analysis. [6] [7]

1.2 Aim and Objectives of the Project

Aim:

To develop an interactive and user-friendly GUI-based packet sniffer using Scapy and Tkinter that enables effective network traffic monitoring and protocol-level packet analysis. [8]

Objectives:

- To implement real-time packet sniffing with support for TCP, UDP, ICMP, DNS, and HTTP protocols. [4] [6]
- To create a GUI using Tkinter for displaying live and previously captured packet data. [7]
- To enable users to open and analyse .pcap files in a new GUI window. [9]
- To integrate the option to save logs and extracted data as PDF for reporting or auditing. [10]

- To enhance accessibility for learners and professionals without requiring command-line expertise.

1.3 Scope of the Project

This project focuses on creating a lightweight desktop application capable of performing fundamental and advanced packet analysis with an easy-to-use GUI. It provides:

- Real-time network traffic capture and classification.
- Parsing and display of HTTP, DNS, TCP/UDP, and ICMP/ICMPv6 traffic. [8]
- Compatibility with IPv4 and IPv6 protocols. [11]
- PCAP file parsing and viewing capabilities. [10]
- PDF log export feature for sharing and documentation.
- Open-source accessibility for further academic or commercial enhancement.

This project is ideal for educational institutions, network security learners, forensic investigators, and SOC analysts needing quick traffic insights without complex configurations or setups. [1] [12] [13]

LITERATURE SURVEY

2.1 Current/Existing System

2.1.1 Study of Current System

There are a number of network monitoring and packet analysis utilities in the world of cybersecurity, with the most commonly used being Wireshark, tcpdump, and Ettercap.

- Wireshark is a GUI-based network protocol analyser with rich features that is utilized by experts for packet inspection. [2]
- tcpdump is a robust command-line network packet analysis and capture tool of choice among system administrators and developers due to its lightweight nature. [14]
- Ettercap is well known for MITM (Man-in-the-Middle) attacks and LAN sniffing. [15]

These tools offer high-end features, such as advanced network packet dissection and troubleshooting capabilities, but most of them demand high domain expertise from users, and some of them are high-resource tools for straightforward or specific use scenarios. [3]

2.1.2 Problem & Weakness of Current System

Although current tools such as Wireshark are extensive, they bring the following difficulties:

- Complex UI and information overload, thus less newbie-friendly.
- Heavyweight programs that might be inappropriate for systems with low resources.
- High learning curve for students or professionals who are new to packet analysis.
- Limited personalization for inclusion of particular protocol filters or light-weight features.
- No easy logging or PDF report export for simple audit or incident response usage.

2.2 Requirements of New System

The suggested system, ScapyUI, plans to overcome the shortcomings of existing systems by providing:

- A light-weight GUI interface based on Tkinter, deployable easily on any OS that supports Python. [16]
- Real-time sniffing and filtering of significant protocols such as TCP, UDP, HTTP, DNS, and ICMP. [17]
- In-depth packet view in readable form without flooding the user.
- A .pcap file reader to display previously sniffed packets in a user-friendly format. [10]
- Exporting traffic logs as PDF for documentation or forensic purposes.
- Both IPv4 and IPv6 support, providing greater compatibility with contemporary networks. [11]

2.3 Feasibility Study

2.3.1 Technical Feasibility

The system is computationally possible as:

- It is written in Python, which is a platform-independent open-source language. [18]
- Scapy is utilized by the project, an extremely powerful packet manipulation Python library. [6]
- The GUI is coded in Tkinter, a commonly distributed Python module having negligible dependencies. [19]
- It doesn't need any licensed or commercial software.
- It may be operated using systems having comparatively modest hardware features.

2.3.2 Operational Feasibility

From an operational perspective:

- The tool is user-friendly, making it easy to adopt by students and security professionals.
- It accommodates various use cases such as live sniffing, pcap analysis, and report generation.
- Reduces reliance on sophisticated tools and enables quick learning and analysis.
- The open-source nature allows users to customize or expand features as needed.

2.4 Tools/Technology Required

Tool/Technology	Purpose
Npcap	Library for capturing data [20]
Python 3.x	Base programming language [18]
Scapy	Packet sniffing and manipulation [6]
Tkinter	GUI development [21]
PIL (Pillow)	For image handling (if logos or images are used) [22]
reportlab	PDF generation
OS / Platform	Windows/Linux (cross-platform compatibility)
rdpcap (Scapy)	To read .pcap files [9]
threading	For handling live sniffing without freezing the UI

DESIGN: ANALYSIS, DESIGN METHODOLOGY AND IMPLEMENTATION STRATEGY

3.1 Function of System

The ScapyUI Network Sniffer utility is meant to offer an easy-to-use interface for network packet capture, analysis, and visualization. The main features are:

- Real-time packet sniffing
- Packet information viewing in a GUI-friendly manner
- Parsing and reading .pcap files
- Log exporting to PDF
- Viewing protocols such as TCP, UDP, DNS, HTTP, and ICMP

3.1.1 Use Case Diagram

The Use Case Diagram demonstrates the overall interaction of the user and the system. It determines the primary functionalities presented to the user like initiating packet sniffing, terminating it, reading a .pcap file, and exporting the contents to a PDF. This diagram assists in the visualization of the system's functionality from the point of view of the end-user and aids in requirement gathering and verification. [23]

Actors:

- User

Use Cases:

- Start Packet Capture
- View Packet Details
- Read PCAP File
- Export Logs to PDF
- View About Info
- Stop Packet Capture

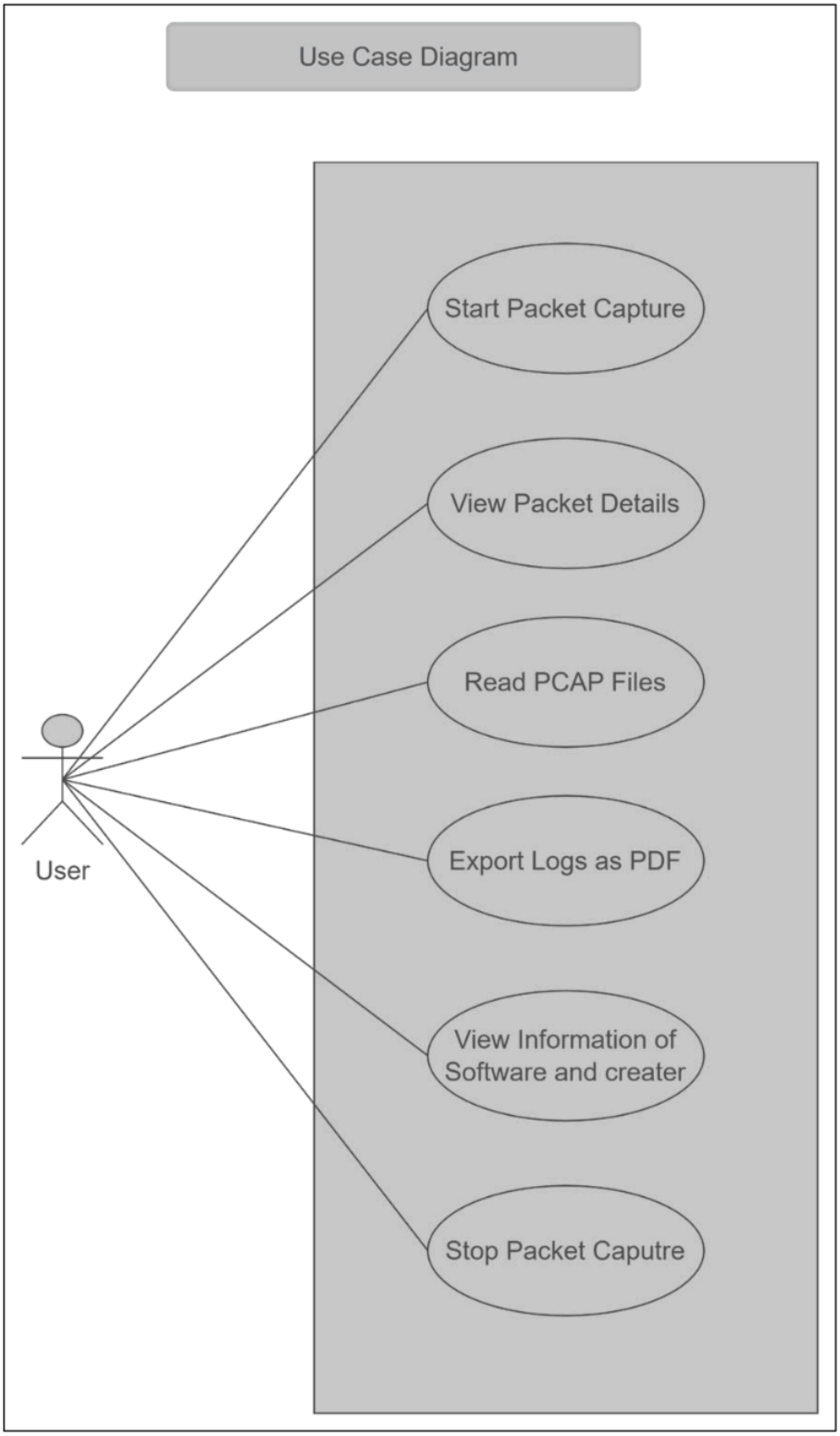

Fig 1 Use Case Diagram

3.1.2 Activity Diagram

The Activity Diagram illustrates the flow of control in the system. It highlights the sequence of operations from the application launch, selecting an action (for instance, sniffing or opening a file), and viewing or exporting packet details. This diagram helps in understanding the workflow and logic followed by the system on the basis of user actions. [24]

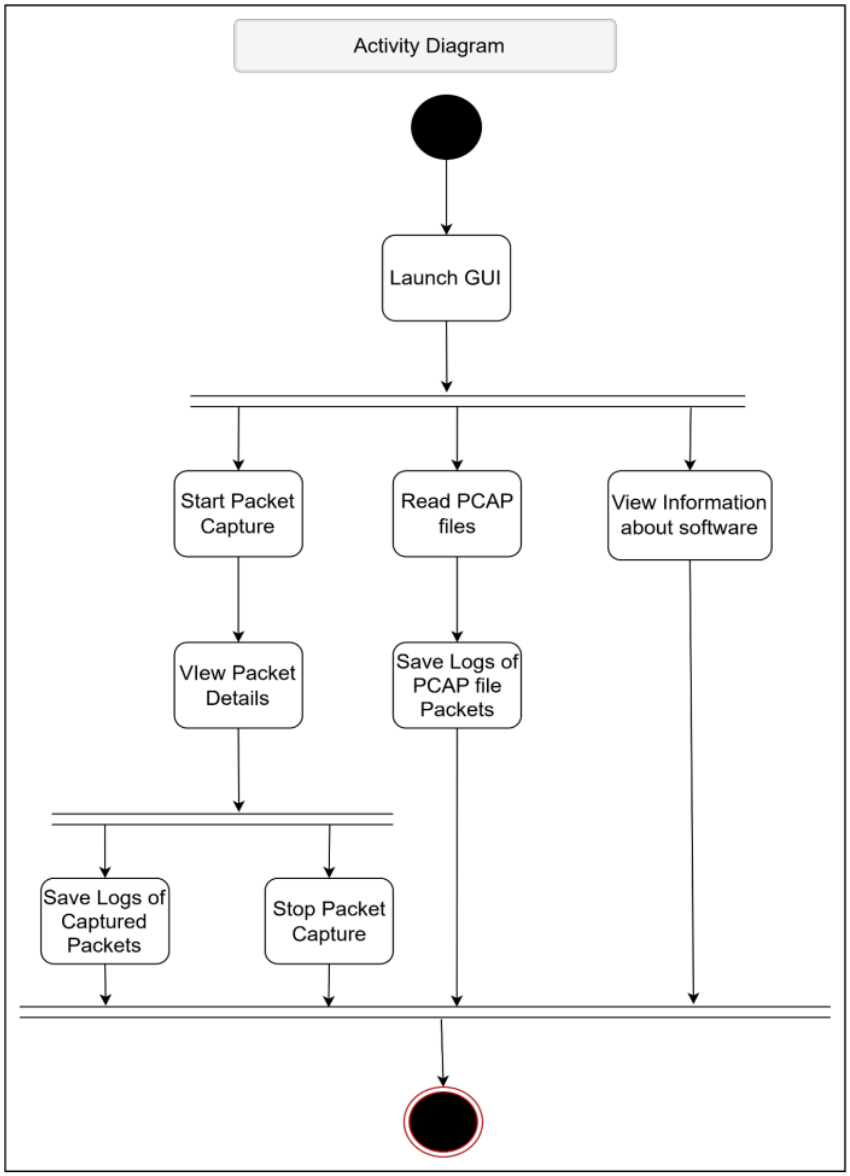

Fig 2 Activity Diagram

3.1.3 Sequence Diagram

The Sequence Diagram illustrates how various parts of the system are interacted with over time. It illustrates the interaction among the user interface, packet sniffer, .pcap reader, and PDF exporter classes. The diagram stresses the sequence of message exchanges, thus it makes it easier to comprehend object collaboration at runtime. [25]

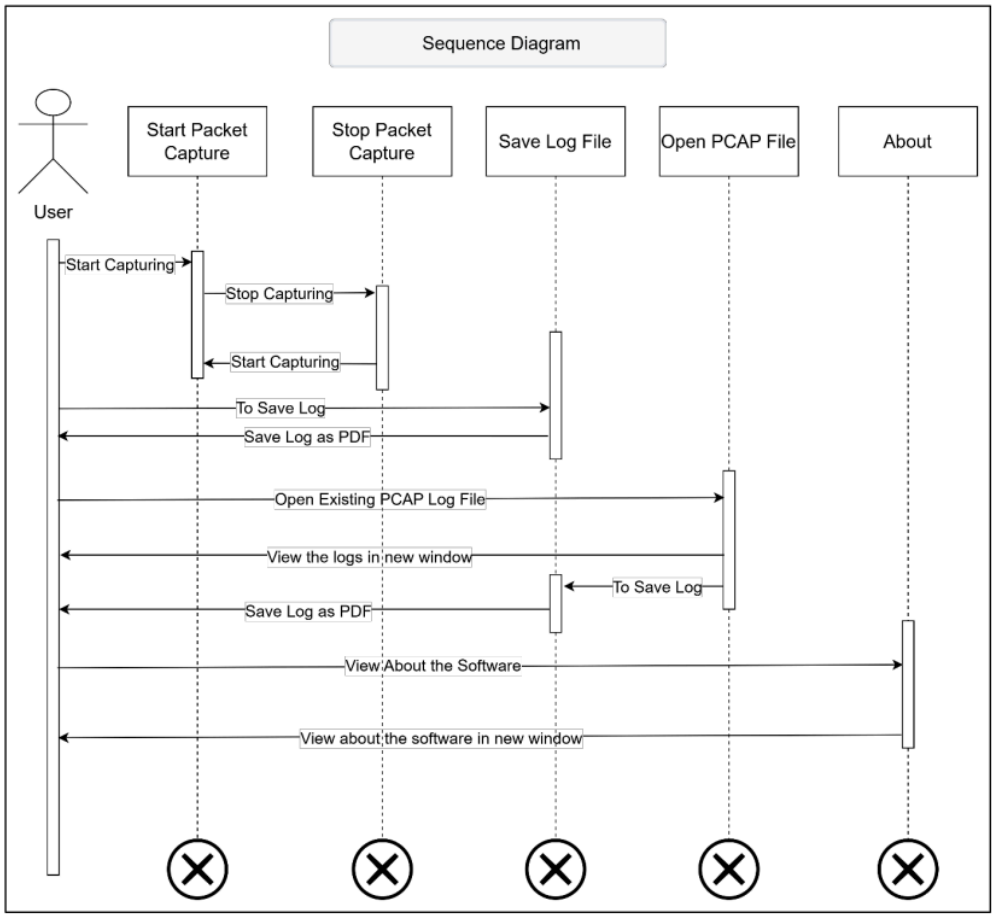

Fig 3 Sequence Diagram

3.2 Data Modelling

The tool works with structured packet data. While it does not connect to a database, internal object modelling helps with capturing, storing, and displaying packets.

3.2.1 Entity-Relationship Diagram

Although the system does not utilize a traditional database, the ER Diagram logically represents the relationship among key entities such as the User Action,

Packet, Packet Details, and Export File. It helps to visualize how user interactions generate packets, which in turn contain details and can be exported.

Since there's no database, showing logical entities like:

- **User Action**
- **Packet**
- **Packet Details**
- **Export File**

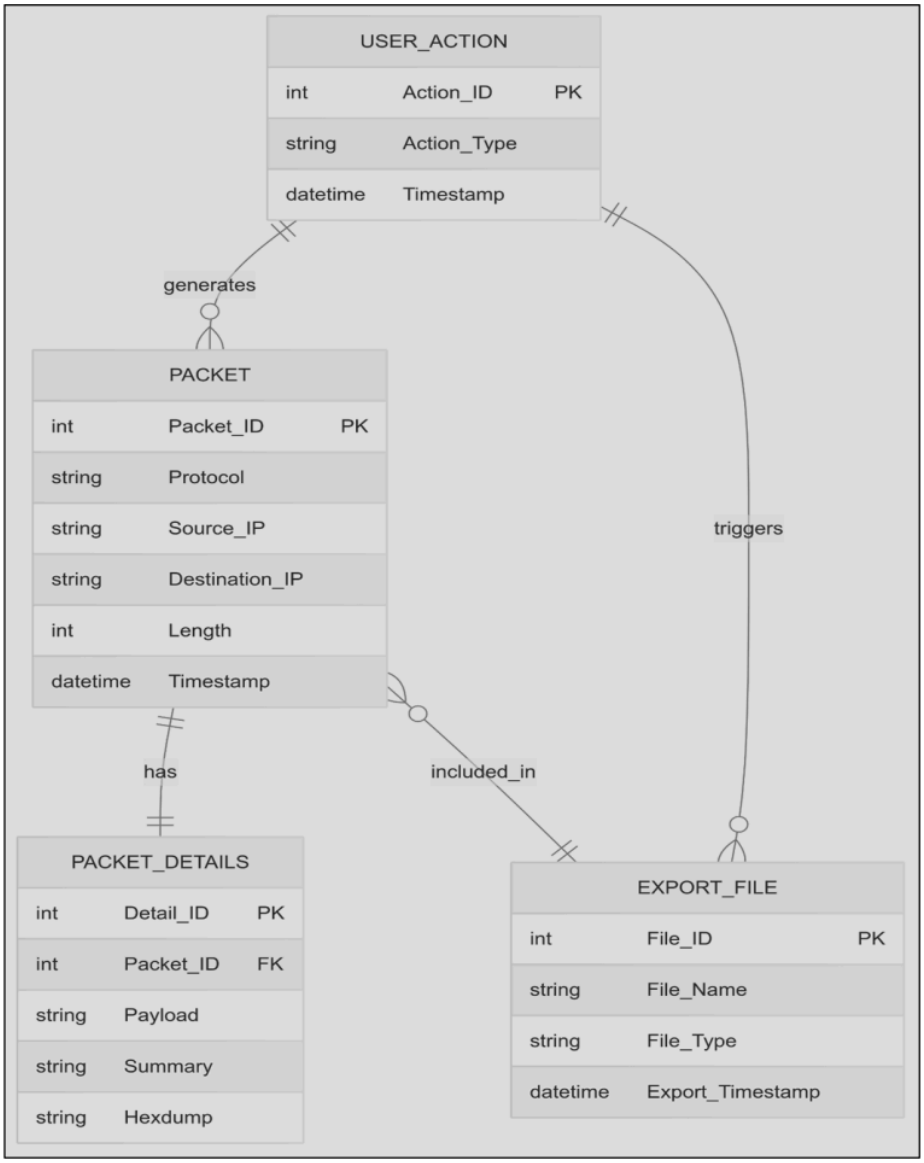

Fig 4 Entity Relationship Diagram

3.2.2 Class Diagram

The Class Diagram illustrates the system's structural design in terms of classes and their relationships. Key classes are PacketSniffer, GUIHandler, PCAPReader, and PDFExporter, each responsible for certain tasks such as capturing, displaying, reading, and exporting packet information. This diagram gives insight into object-oriented architecture and code maintainability.

Main Classes:

- PacketSniffer
- GUIHandler
- PCAPReader
- PDFExporter

These classes encapsulate functionality for sniffing, GUI management, pcap reading, and export.

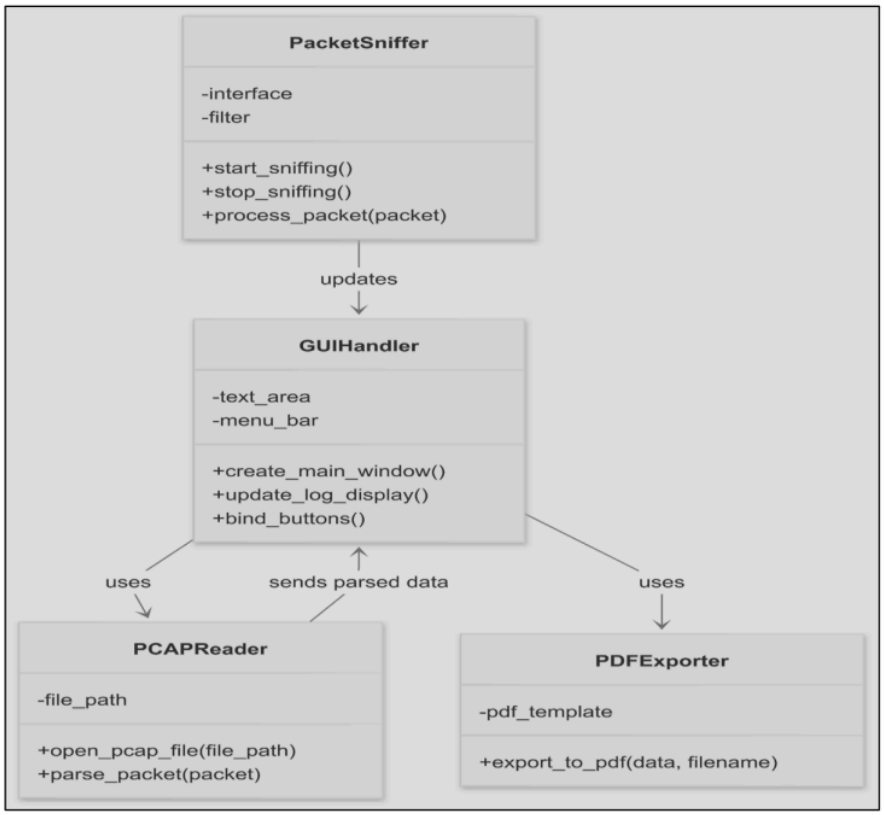

Fig 5 Class Diagram

3.3 Functional & Behavioural Modelling

3.3.1 Data Flow Diagram

Level 0 DFD:

The Level 0 DFD, also known as the context diagram, provides a top-level overview of the entire system. It identifies the system as a single process and shows the main external entities (like the user) and the data flowing between the system and those entities. [26]

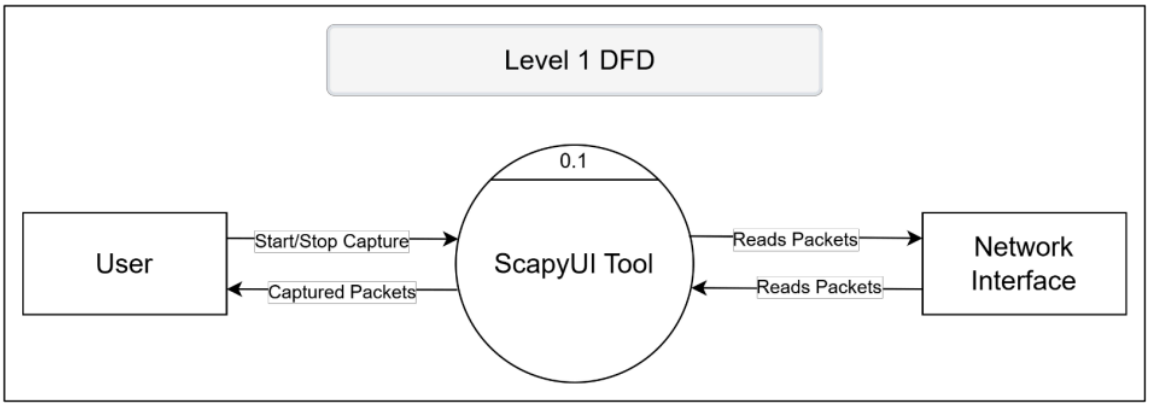

Fig 6 Level 0 Data Flow Diagram

- ScapyUI has multiple modules: Live Capture, PCAP Reader, Packet Detail Viewer, Export Handler

Level 1 DFD:

This diagram expands on the Level 0 DFD by breaking the main process into sub-processes such as packet capturing, packet display, and file exporting. It shows how data moves through different parts of the system and highlights intermediate data stores and transformations. [26]

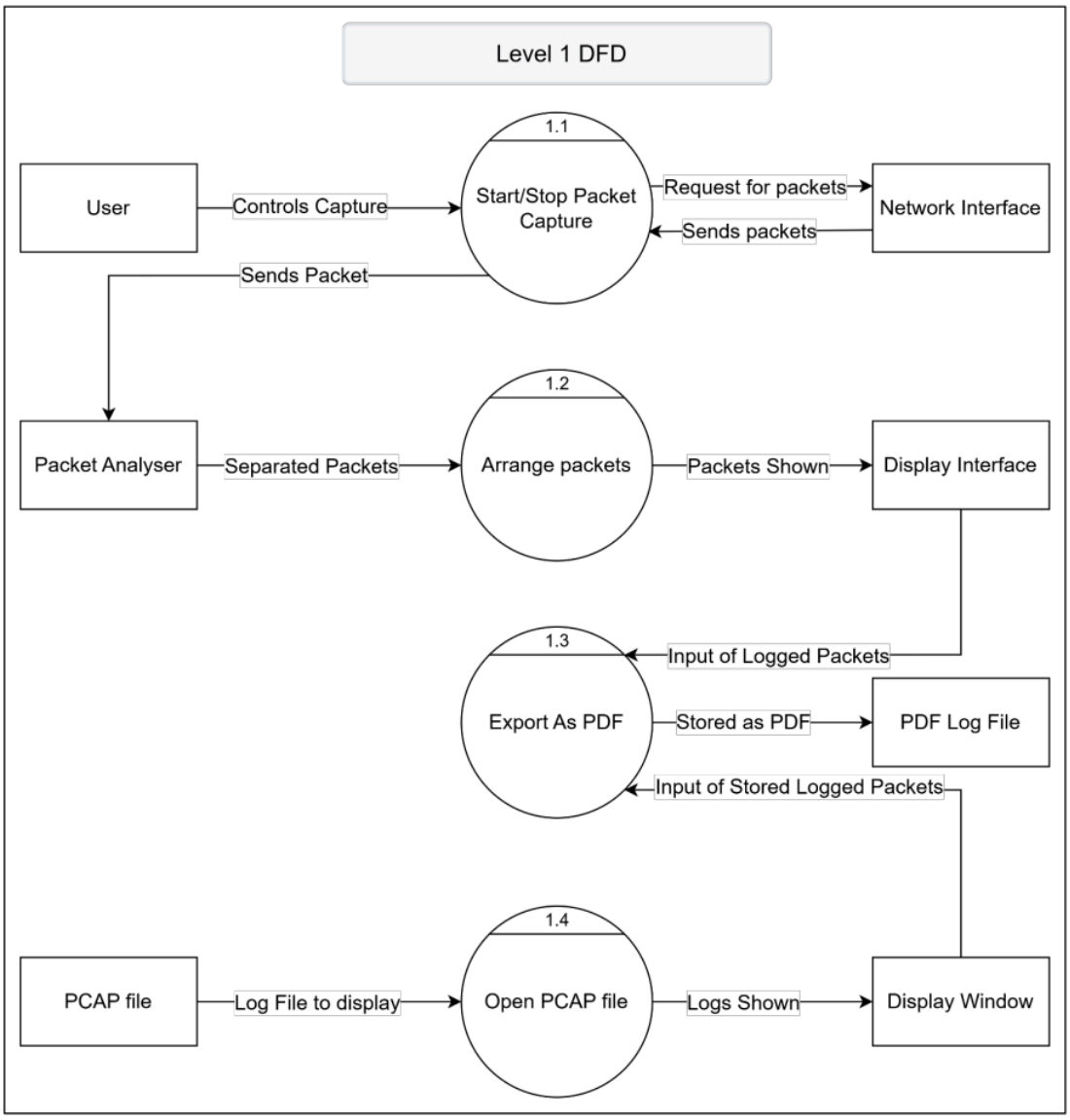

Fig 7 Level 1 Data Flow Diagram

Level 2 DFD:

The Level 2 DFD provides a more detailed decomposition of the sub-processes shown in Level 1. It delves into specific data operations such as filtering packets, extracting HTTP/DNS data, and formatting export files. This level of detail is particularly helpful for developers to understand internal data flows and processing logic. [26]

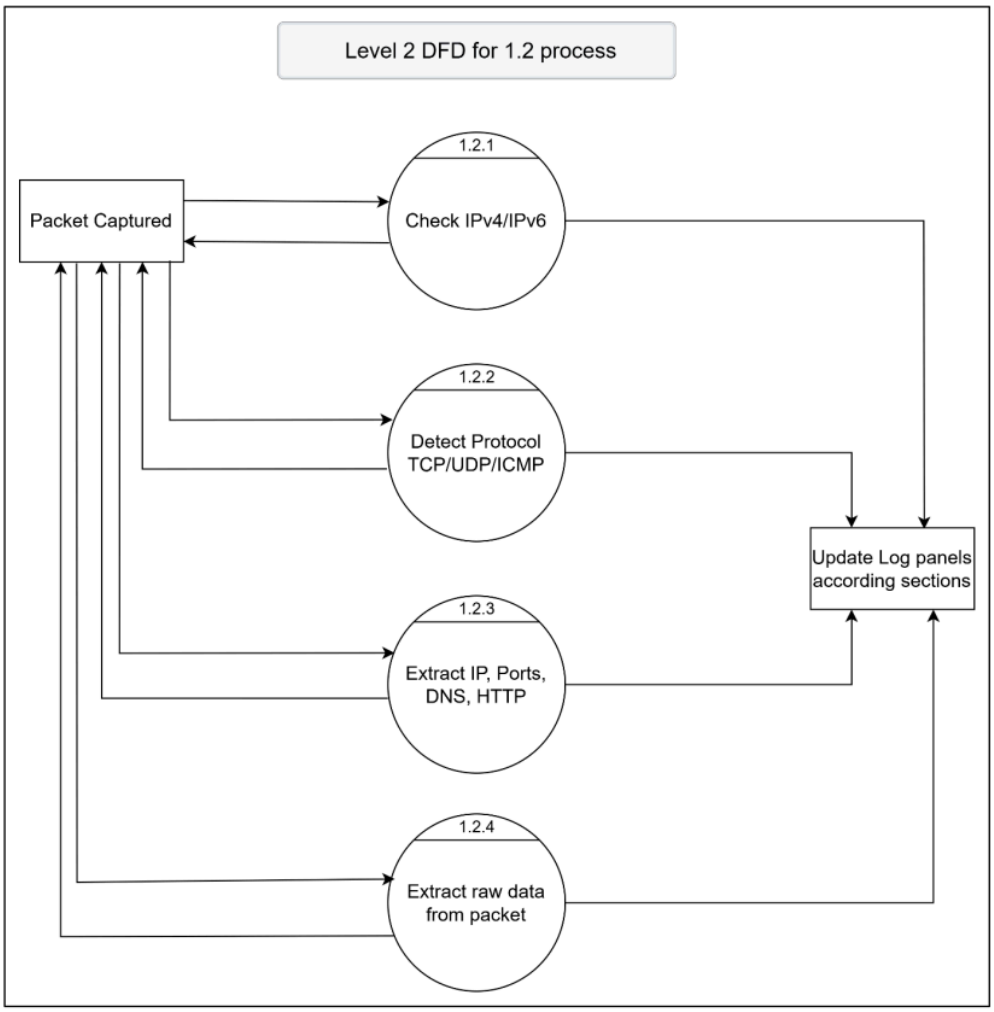

Fig 8 Level 2 Data Flow Diagram

3.3.2 Data Dictionary

The Data Dictionary provides detailed information about the various data elements used in the system. Each data item is defined with its name, type, description, and related process or module. This helps ensure consistency in naming and data usage across the system design and implementation. [27]

Element	Description	Data Type
packet_data	Captured packet info	Dictionary/List
protocol_type	Type of protocol (TCP/UDP/etc.)	String
timestamp	Time of packet capture	String/Datetime
source_ip	Sender's IP address	String
dest_ip	Receiver's IP address	String
pcap_file_path	Filepath of loaded PCAP	String
log_content	Text to be exported as PDF	String

IMPLEMENTATION

4.1 Implementation Environment

The implementation of the ScapyUI – Network Sniffer was carried out using the Python programming language and the Tkinter library for the GUI. The tool was developed and tested on a Windows 11 operating system with administrative privileges to allow packet sniffing. [17]

4.1.1 Model Used in Developing

The **Iterative Model** was used during development. [28] This approach allowed the application to be built step-by-step with continuous feedback and testing at each phase. Each iteration added functionality such as:

- Live packet capture
- PCAP file reader
- GUI enhancements
- Export to PDF

This model enabled refining of requirements and allowed incremental upgrades, making it suitable for a student project.

4.1.2 Software Prototyping Types

Evolutionary Prototyping was followed. Initial prototypes focused on the basic GUI and packet capture features, which were gradually improved based on testing and feedback. [29] This helped in enhancing usability, adding new features like:

- Real-time display
- Viewing packet structure
- PCAP reader in a new window
- PDF export with line formatting

4.2 Coding Standard

To ensure readability and maintainability, the following coding standards were followed:

- **Naming Convention**: CamelCase for class names, snake_case for variables and functions. [30]

- **Commenting**: Inline and block comments to describe code functionality. [30]
- **Modular Design**: Code was broken into functions and classes to follow the DRY (Don't Repeat Yourself) principle. [30]
- **PEP8 Guidelines**: The Python code adheres to PEP8 standards for spacing, naming, and formatting. [31] [30]

4.3 Laboratory Setup

The project was implemented and tested in the following lab setup:

Component	Description
Operating System	Windows 11 64-bit
RAM	8 GB
Python Version	3.10+
Dependencies	scapy, tkinter, fpdf, rdpcap, reportlab, datetime
Permissions	Administrator access for packet sniffing

4.4 Tools and Technology Used

Tool/Technology	Purpose
Python	Core programming
Tkinter	GUI development
Scapy	Packet sniffing and decoding [32]
FPDF	PDF export
Wireshark (optional)	PCAP verification [14] [33]
VS Code / PyCharm	Code writing and testing

4.5 Screenshots / Snapshots

1. Main GUI Interface

Fig 9 Main GUI Interface

2. Live Packet Capture

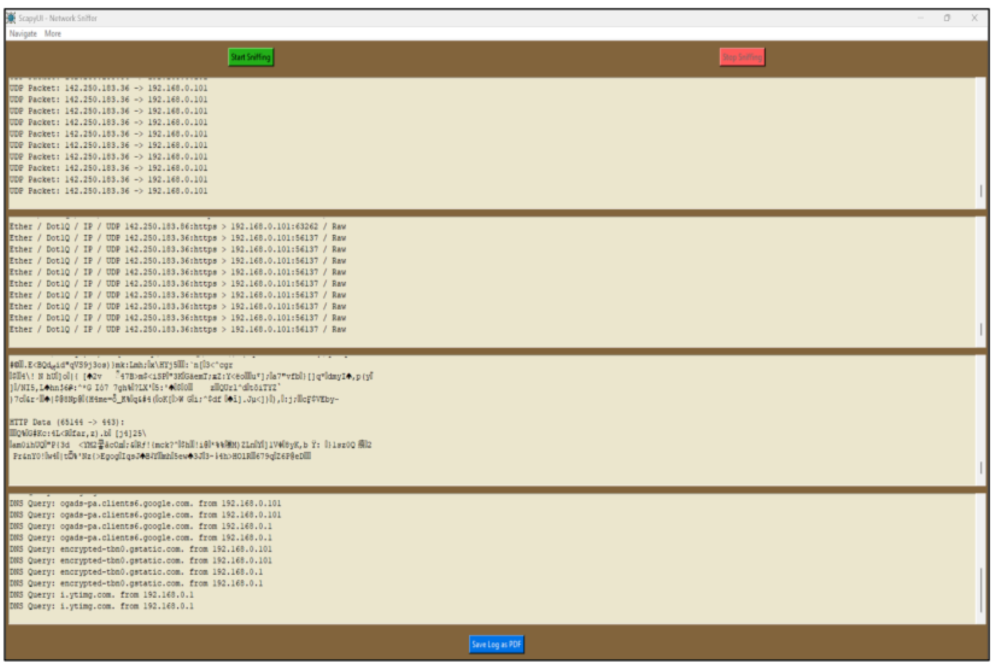

Fig 10 Live Packet Capture

3. PCAP File Reader Window

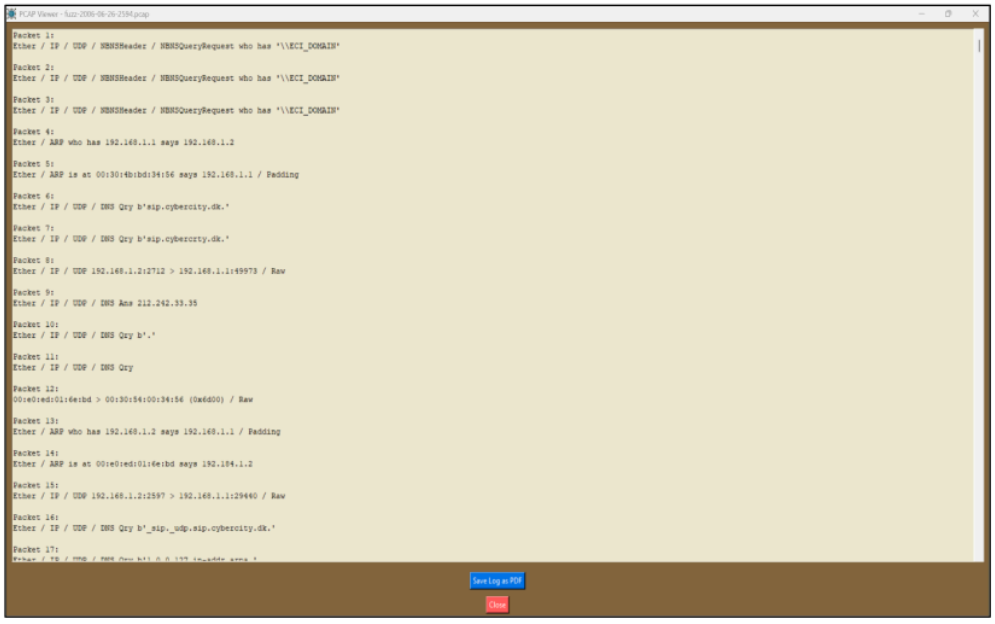

Fig 11 PCAP File Reader Window

4. PDF Export Output

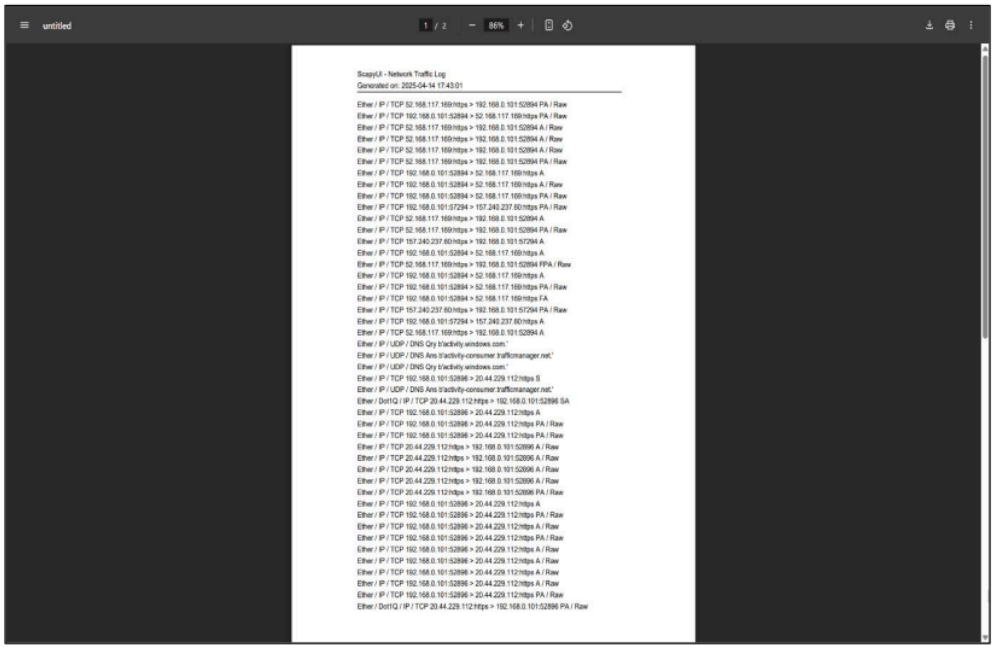

Fig 12 PDF Export Output

SUMMARY OF RESULTS AND FUTURE SCOPE

5.1 Advantages / Unique Features

The ScapyUI – Network Sniffer offers a user-friendly graphical interface to facilitate live packet capturing and analysis using Python. Unlike traditional command-line tools, this application provides accessibility and usability to students, educators, and entry-level professionals. Key advantages and features include:

- **GUI-Based Packet Sniffer**: Simplifies the packet analysis process with a clear and responsive interface.
- **Real-Time Capture Display**: Packets are captured and displayed in real-time with details on protocol, source, destination, and length.
- **PCAP File Reader**: Allows users to open and analyse .pcap files in a separate window.
- **PDF Export Feature**: Enables saving of log data as a PDF for reporting or documentation.
- **Modular and Scalable Design**: The architecture allows easy integration of advanced features such as filtering, protocol decoding, or statistics.
- **Cross-Platform Capability**: Can run on multiple operating systems with Python support.

5.2 Results and Discussions

The application was tested across various scenarios including:

- Live packet sniffing on local network traffic.
- Opening previously saved .pcap files for offline analysis.
- Exporting logs to a structured and formatted PDF file.

Performance Observations:

- The tool was able to capture and display packets efficiently with minimal delay.
- .pcap files were parsed accurately using Scapy, and the contents were correctly displayed.
- The log formatting for PDF export required handling of line breaks and dynamic content sizing, which was successfully implemented.

- No crashes or memory issues were observed during continuous captures under moderate traffic.

This demonstrates that the tool serves as a functional lightweight packet analyser suitable for academic and research use.

5.3 Future Scope of Work

While the current implementation meets the basic requirements of a GUI-based sniffer, the following enhancements can be considered for future versions:

- **Packet Filtering**: Add real-time protocol-based filtering (e.g., TCP, UDP, ICMP).
- **Search Functionality**: Enable keyword or IP-based search in captured data.
- **Protocol Decoding**: Enhance support for deep packet inspection and protocol decoding (HTTP, DNS, etc.).
- **Hex Dump View**: Display the raw hexadecimal view of packet data for forensic purposes.
- **Cloud Integration**: Store and analyse logs on cloud-based platforms for collaborative research or remote access.
- **Security Features**: Add alert-based mechanisms to detect suspicious or malicious packets in real-time.

These upgrades would significantly improve the tool's capabilities, making it a more complete solution for educational and operational use in network security environments.

CONCLUSION

The creation of ScapyUI – Network Sniffer has been effectively able to overcome the objective of having a GUI-based, light-weight network packet analyser utilizing Python and Scapy. The project had endeavoured to reduce the difference between command-line utility packet analysis applications and the usability of graphical programs, particularly targeting students, instructors, and starting-level cybersecurity operators.

Throughout the course of the project, I conceptualized and executed an interactive, modular, and effective tool capable of monitoring the network in real-time, .pcap file analysis, and exporting the logs into PDF. The integration of Tkinter and Scapy facilitated a unified framework for capturing and analysing packets, whereas executing PDF and GUI functionality enabled enhanced usability.

The tool satisfies the minimal needs of packet analysis and has a solid platform for extensions in the future, including real-time filtering, protocol decoding, and threat detection. It illustrates the effective application of Python in the creation of operational cybersecurity tools and adds value to the academic learning process in digital forensics and network security.

Through this project, I experienced firsthand software design, GUI development, network packet analysis, and tool prototyping. The project not only enhances knowledge of network protocols and packet structures but also culminates in hands-on expertise in Python programming and GUI application development.

Minor Project Report From chapter 1

ORIGINALITY REPORT

1%	1%	0%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

1%

★ www.coursehero.com

Internet Source

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		