LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background

In today’s digital age, the increasing dependence on interconnected systems and online platforms has led to an exponential rise in cyber threats. From ransomware attacks and phishing scams to data breaches and advanced persistent threats, cyberattacks are becoming more frequent and sophisticated. Organizations, governments, and even individuals are now prime targets. As networks become more complex, monitoring them for potential intrusions becomes essential. However, traditional tools such as firewalls and antivirus software, while effective in some cases, are largely reactive and may not detect advanced or subtle forms of unauthorized access.

To address this challenge, Intrusion Detection Systems (IDS) have emerged as a key component in the cybersecurity landscape. IDS solutions are designed to monitor network traffic continuously, identify suspicious activities, and generate alerts when potentially malicious behavior is detected. Despite their effectiveness, many IDS solutions are either expensive or difficult to set up and manage, making them inaccessible to small organizations, educational institutions, or cybersecurity learners.

CyberPulse IDS was developed as a response to this gap in accessibility. It is a lightweight, real-time IDS designed for educational and small-scale deployment. Unlike traditional IDS tools that operate as black-box systems with minimal visualization, CyberPulse emphasizes clarity, usability, and real-time awareness. It integrates powerful Python libraries such as Scapy for packet sniffing, Nmap for port scanning, and Plotly Dash for dynamic visualizations. By combining packet analysis, threat detection, and intuitive dashboards, CyberPulse bridges the gap between raw technical output and practical security insights.

The project also addresses another critical need: **cybersecurity education and awareness**. In many cases, users are unaware of the ongoing threats on their network until significant damage has occurred. CyberPulse IDS brings those threats to the forefront through live traffic monitoring, alerts, and auto-generated reports, allowing users to react before vulnerabilities are exploited.

Furthermore, with the rise of BYOD (Bring Your Own Device), IoT devices, and remote work, the average network is now more exposed than ever. Tools like CyberPulse IDS are vital for identifying abnormal behavior, such as suspicious IP addresses, unauthorized port usage, or network flooding, and taking preemptive action.

This project not only fulfills a functional need but also serves as a stepping stone for further research and development in real-time security monitoring, AI-driven anomaly detection, and threat intelligence sharing. Ultimately, CyberPulse IDS aims to simplify the complexity of cybersecurity monitoring while equipping users with actionable information and a better understanding of network behavior.

1.2 Problem Statement

The rapid digitization of infrastructure has created vast, interconnected networks that support everything from communication and banking to education and healthcare. While these systems provide immense benefits, they also create a larger attack surface for cybercriminals. Despite investments in firewalls, antivirus programs, and access controls, many organizations still fall victim to intrusions that bypass conventional security measures.

A core reason for these breaches is the absence of real-time visibility into network traffic. Without proper monitoring, organizations are unable to identify and respond to threats before damage is done. Intrusion Detection Systems (IDS) offer a solution by actively monitoring and analyzing network traffic for signs of malicious activity. However, commercial IDS tools such as Snort, Suricata, or proprietary solutions from Cisco or Palo Alto Networks come with challenges:

- High cost of implementation and licensing.
- Complex configuration and maintenance.
- Steep learning curves, requiring skilled personnel.
- Lack of intuitive, user-friendly interfaces.

These challenges make IDS tools inaccessible to small businesses, students, educators, and cybersecurity enthusiasts. Even open-source options, while free, often lack the features to visualize traffic or generate digestible reports that can be shared across teams or used for auditing purposes.

Furthermore, most existing tools do not combine port scanning, packet sniffing, visualization, and reporting into a single streamlined application. Security analysts often rely on multiple tools—Wireshark for sniffing, Nmap for scanning, Grafana for visualizations, and others for reporting. Managing these tools together can be cumbersome and inefficient.

CyberPulse IDS was conceived to solve these problems. It is designed to be a lightweight, modular, and accessible IDS that can be deployed in local environments to detect network intrusions and visualize them in real-time. It allows users to scan ports, sniff live traffic, identify suspicious activity, and auto-generate security reports without relying on complex configurations or multiple external tools.

The core problem addressed by CyberPulse IDS is this:

“How can we create an affordable, easy-to-use, and integrated Intrusion Detection System that offers real-time monitoring, visualization, and reporting for small networks or educational purposes?”

By focusing on ease of use, automation, and clarity, the project fills a significant gap in current IDS solutions, especially for cybersecurity learners and small-scale network environments.

1.3 Objective

The primary objective of the CyberPulse IDS project is to design and implement a **real-time Intrusion Detection System** that provides users with live visibility into their network activities while also enabling effective threat detection and response. The project aims to make network monitoring and threat detection accessible, educational, and efficient for a wide range of users.

The key objectives of the project are outlined below:

1. Develop a Real-Time IDS Tool

Create a functional and efficient IDS capable of sniffing network packets in real-time, analyzing them for signs of intrusion, and presenting that information clearly to the user.

2. Integrate Port Scanning

Incorporate port scanning capabilities using Scapy and Nmap to allow the system to assess open, closed, or filtered ports on hosts within a network—an essential step in identifying vulnerabilities.

3. Build an Interactive Dashboard

Design a visually appealing, real-time dashboard using **Dash and Plotly**, which graphically displays network traffic, packet types, IP addresses, threat alerts, and scanning results.

4. Enable Automated Reporting

Include the ability to generate detailed reports in **PDF and PowerPoint formats**. These reports should summarize detected threats, scanning results, and traffic trends for documentation or audits.

5. Ensure System Performance with Multithreading

Implement **multithreading** to allow packet sniffing, scanning, data visualization, and report generation to run concurrently without causing performance degradation.

6. Promote Cybersecurity Awareness

Make the tool educational and intuitive, enabling students and beginners to explore how real-time network traffic analysis works and how intrusions can be detected.

7. Lay Foundation for Future Enhancements

Ensure that the architecture allows future improvements such as AI-based anomaly detection, cloud integration, and multi-user access features.

By achieving these objectives, CyberPulse IDS not only functions as a practical tool for real-time security monitoring but also contributes to the growing need for accessible and educational cybersecurity solutions.

1.4 Scope

The scope of CyberPulse IDS is carefully defined to balance functionality, performance, and usability. It focuses on delivering a fully working IDS prototype that is modular, scalable, and educational—suitable for small-scale networks, personal use, or academic research.

Features include:

1. Real-Time Packet Sniffing:

The system captures and analyzes live network packets using Scapy, helping users observe real-time traffic patterns and detect anomalies.

2. Port Scanning:

The integration of Nmap allows CyberPulse IDS to perform active scans on the network to identify open, closed, or filtered ports, which is crucial for vulnerability assessment.

3. Intrusion Detection & Alerts:

The tool uses basic logic to identify suspicious patterns (e.g., multiple SYN packets, port scanning behavior) and provides real-time alerts in the dashboard.

4. Interactive Dashboard:

The system employs Dash and Plotly to create a web-based user interface that displays traffic data, scan results, and system logs using dynamic graphs and tables.

5. Automated Report Generation:

Using the FPDF and python-pptx libraries, the system creates downloadable PDF and PPT reports containing graphs, summaries, and analysis for each session.

6. Multithreaded Performance:

The tool is optimized for simultaneous operation of sniffing, scanning, and visualization processes, ensuring smooth performance on average machines.

7. Offline and Local Use:

The tool is designed to run on a local machine without cloud or internet dependency, which adds a layer of privacy and usability in lab settings.

1.5 Methodology

The development of CyberPulse IDS followed a modular, agile-based software development life cycle, combining cybersecurity principles with software engineering best practices. The methodology ensured scalability, maintainability, and ease of integration across different components.

Step 1: Requirements Analysis

- The first step involved understanding user needs, primarily focusing on network security awareness, intrusion detection, real-time visualization, and simplified usability. Inputs from cybersecurity students and IT professionals helped shape the feature set.

Step 2: Tool & Library Selection

- To maximize efficiency and accessibility:

- **Python** was chosen for its rich library ecosystem and rapid development capabilities.
- **Scapy** was used for sniffing and low-level network analysis.
- **Nmap** was integrated via subprocess calls for advanced port scanning.
- **Dash & Plotly** were selected for building the interactive dashboard.
- **FPDF & python-pptx** handled report generation.
- **Multithreading** ensured all features could operate simultaneously.

Step 3: Modular Architecture Design

- The system was divided into six core modules:

1. **Sniffer Module**: Captures network packets in real-time.
2. **Analyzer Module:** Inspects packets for suspicious behavior.
3. **Port Scanner**: Scans for open/closed ports using Nmap.
4. **Dashboard Module**: Presents traffic stats and threats in graphs.
5. **Report Generator**: Creates session summaries in PDF/PPT.
6. **User Interface (UI)**: Web-based dashboard for user control.

Step 4: Development & Integration

- Each module was built independently, tested for stability, and then integrated. Multithreading allowed the sniffer to run concurrently with the dashboard without freezing or lag.

Step 5: Testing & Simulation

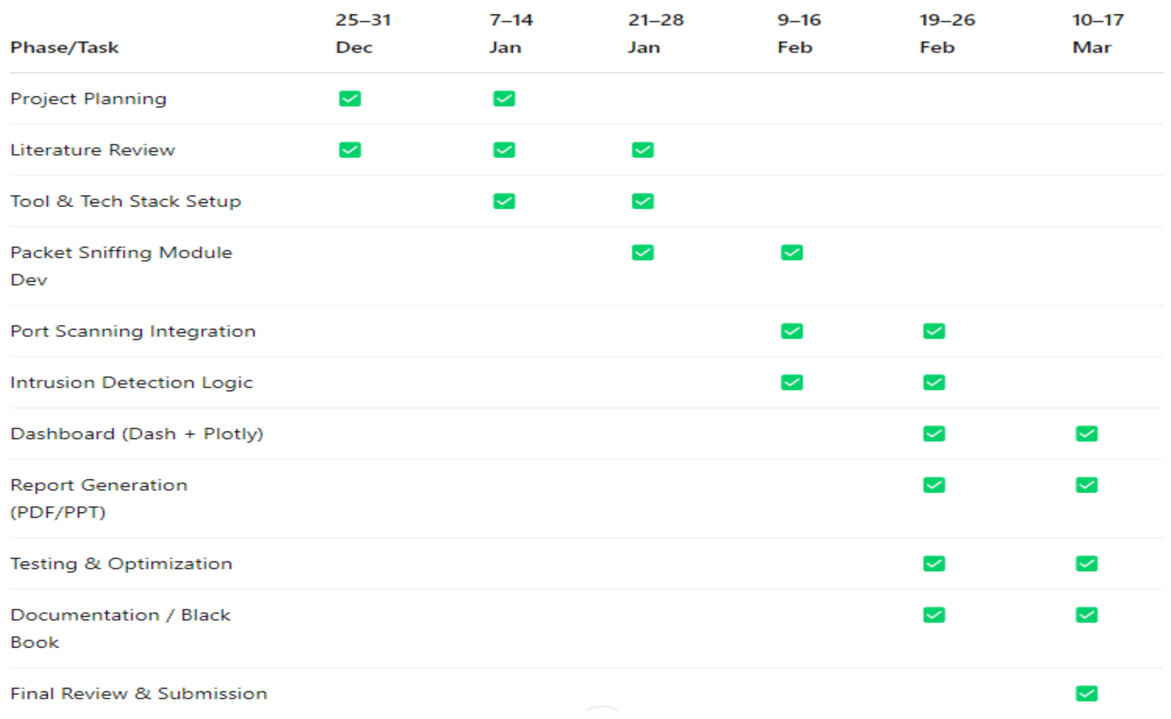
- The IDS was tested under various conditions:

- Normal traffic from web browsing and emails.
- Simulated attacks (port scans, SYN floods) from virtual machines.
- Performance benchmarking with large data volumes.

Step 6: Documentation & Deployment

- Comprehensive documentation was prepared, including a user manual, code comments, and a configuration guide. The system was packaged to run on any Python-supported platform.

This modular and iterative approach ensured robust performance, scalability for future improvements, and real-world usability.



Phase/Task	25–31 Dec	7–14 Jan	21–28 Jan	9–16 Feb	19–26 Feb	10–17 Mar
Project Planning	✓	✓				
Literature Review	✓	✓	✓			
Tool & Tech Stack Setup		✓	✓			
Packet Sniffing Module Dev			✓	✓		
Port Scanning Integration				✓	✓	
Intrusion Detection Logic				✓	✓	
Dashboard (Dash + Plotly)					✓	✓
Report Generation (PDF/PPT)					✓	✓
Testing & Optimization					✓	✓
Documentation / Black Book					✓	✓
Final Review & Submission						✓

Figure 1: Gantt Chart of the Project

CHAPTER 2: SURVEY OF TECHNOLOGIES

2.1 Overview of IDS Techniques

Intrusion Detection Systems (IDS) are critical components in any cybersecurity framework, designed to detect unauthorized access or anomalies within a network or system. IDS techniques are broadly classified based on detection methods and deployment strategies. Understanding these categories is essential to grasp the strengths and limitations of various IDS approaches.

Types of IDS Based on Detection Technique

1. Signature-Based Detection

➢ This technique uses predefined patterns or "signatures" of known attacks. It’s highly effective against known threats but cannot detect novel or zero-day attacks. For example, detecting a specific malware strain that has a unique packet structure or IP behavior.

- **Advantage:** High accuracy for known threats
- **Limitation:** Ineffective against unknown attacks

2. Anomaly-Based Detection

➢ Anomaly-based systems define a baseline of normal network behavior and alert when deviations occur. These systems use statistical models or machine learning algorithms.

- **Advantage:** Can detect new and unknown threats
- **Limitation:** High false positive rate due to dynamic traffic patterns

3. Hybrid Detection

➢ Combines signature and anomaly detection for better accuracy and coverage. It offers a balance between performance and threat coverage but may require more computational resources.

Types of IDS Based on Deployment

1. Host-Based IDS (HIDS)

➢ Deployed on individual systems to monitor file integrity, system logs, and processes. It’s useful for internal threat detection and system-specific anomalies.

Example: OSSEC, Tripwire

2. Network-Based IDS (NIDS)

- Monitors network traffic in real-time, looking for suspicious packets or traffic patterns. Usually placed at critical points in the network like gateways or switches.

Example: Snort, Suricata

3. Hybrid IDS

- Combines HIDS and NIDS to provide a more comprehensive view. These systems correlate data from multiple sources to detect distributed or coordinated attacks.

Emerging IDS Trends

- **AI/ML-based IDS:** Machine learning helps adapt to evolving attack vectors and reduce false positives.
- **Cloud-native IDS:** Monitors traffic in cloud infrastructures like AWS or Azure.
- **Behavioral Analytics:** Uses user and entity behavior analysis (UEBA) to detect insider threats.

CyberPulse IDS primarily operates as a Network-Based IDS with a rule-based anomaly detection approach, while also integrating active scanning using tools like Nmap. It adopts a modular structure, allowing future expansion to include ML-based detection and cloud integration.

2.2 Previous Work and Research

Over the years, researchers and cybersecurity professionals have developed a wide array of IDS solutions tailored to different environments. These projects vary in terms of detection methodology, data sources, visualization tools, and scope. Here’s a detailed look at some notable past works in this domain.

Snort (2001 – Present)

- ➢ Snort is one of the most popular open-source NIDS tools, developed by Sourcefire and later acquired by Cisco. It uses signature-based detection and supports protocol analysis, content searching, and traffic logging.

- **Strengths:** High-speed packet logging, wide community support, rule-based configuration
- **Limitations:** Complex setup, limited real-time visualization

Suricata (2009 – Present)

- ➢ An advanced IDS/IPS tool developed by the Open Information Security Foundation (OISF), Suricata offers multi-threaded processing and supports both signature and anomaly-based detection.

- **Strengths:** High performance, supports deep packet inspection
- **Limitations:** Requires tuning, lacks a built-in GUI

Bro/Zeek (1995 – Present)

- ➢ Zeek is a powerful network analysis tool that logs traffic in-depth and enables scripting for custom detections. It’s not a traditional IDS but is widely used for traffic behavior analytics.

- **Strengths:** Extremely customizable
- **Limitations:** Steep learning curve, limited real-time alerting

AI/ML-Based IDS Research

Recent studies in academic journals have explored the use of supervised and unsupervised learning for anomaly detection. Datasets like **KDD Cup 99**, **NSL-KDD**, and **CICIDS 2017** have been used to train models for detecting DoS, port scans, botnets, and more.

- **ML Algorithms used**: Decision Trees, Random Forest, SVM, Autoencoders
- **Outcome**: Promising accuracy but models are sensitive to concept drift and data imbalance

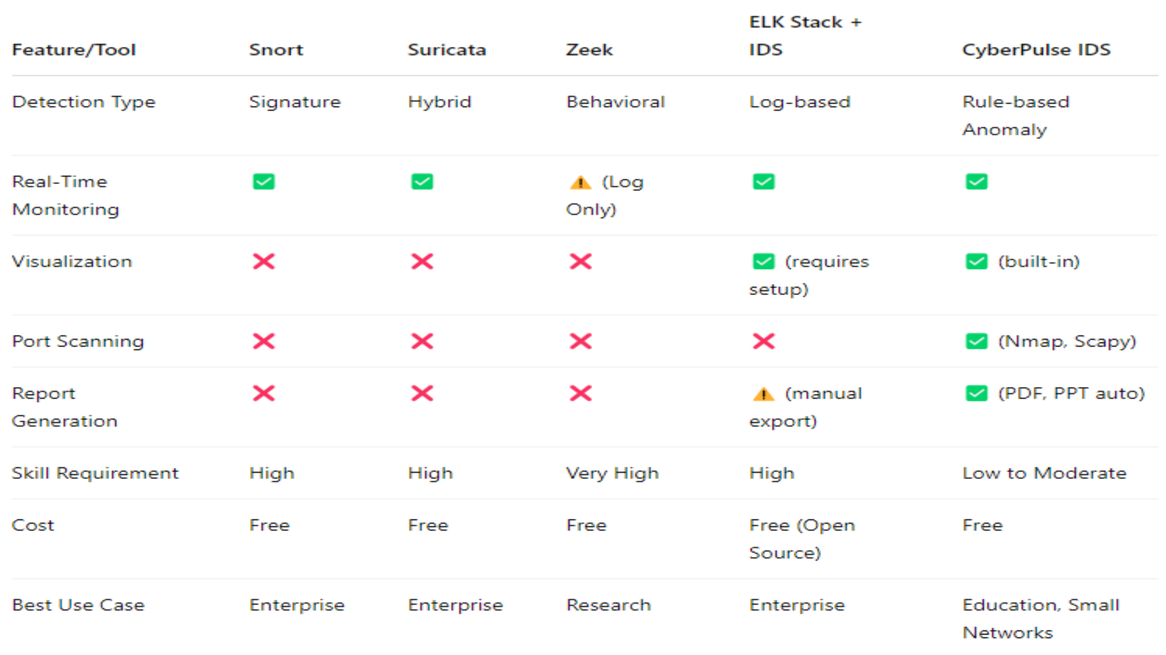
Dashboards and Visualization

Visualization tools like **ELK Stack**, **Grafana**, and **Splunk** are used with IDS logs to build real-time dashboards. However, these tools require integration and configuration. Few existing IDS solutions offer a built-in visualization component, which is a gap CyberPulse IDS aims to address directly.

CyberPulse IDS builds upon this body of work by combining **packet capture, active scanning, real-time dashboard visualization**, and **automated reporting**—all in one tool, designed for educational and lightweight deployment scenarios.

2.3 Comparative Analysis

To better understand the strengths and innovations of CyberPulse IDS, it's useful to compare it with some popular IDS tools across key dimensions such as functionality, complexity, cost, and visualization. This comparison also highlights the niche that CyberPulse IDS is designed to fill.



Feature/Tool	Snort	Suricata	Zeek	ELK Stack + IDS	CyberPulse IDS
Detection Type	Signature	Hybrid	Behavioral	Log-based	Rule-based Anomaly
Real-Time Monitoring	✅	✅	⚠️ (Log Only)	✅	✅
Visualization	❌	❌	❌	✅ (requires setup)	✅ (built-in)
Port Scanning	❌	❌	❌	❌	✅ (Nmap, Scapy)
Report Generation	❌	❌	❌	⚠️ (manual export)	✅ (PDF, PPT auto)
Skill Requirement	High	High	Very High	High	Low to Moderate
Cost	Free	Free	Free	Free (Open Source)	Free
Best Use Case	Enterprise	Enterprise	Research	Enterprise	Education, Small Networks

Figure 2: Comparative analysis of Tools and CyberPulse IDS

Unique Advantages of CyberPulse IDS:

- **All-in-one design:** Combines scanning, sniffing, detection, and reporting.
- **User-friendly dashboard:** No need for third-party integration.
- **Educational Focus:** Designed to aid learning and understanding of network security principles.
- **Lightweight Deployment:** Ideal for personal systems, VMs, or classroom labs.

Limitations Compared to Industry Tools:

- Basic detection logic (no DPI or ML yet)
- Not optimized for large, high-speed networks
- Lacks deep integration with SIEM platforms or cloud services

Overall, CyberPulse IDS sits at the intersection of functionality and simplicity, offering features that are often distributed across multiple tools in a single, unified package—without the steep learning curve or infrastructure overhead.

CHAPTER 3:
REQUIREMENTS AND ANALYSIS

3.1 Proposed System

The proposed system, **CyberPulse IDS**, is an integrated intrusion detection solution designed to provide real-time monitoring, threat detection, and actionable insights into network activities.

It combines several key modules to form a cohesive and efficient system:

1. **Packet Sniffer Module:** This core module continuously captures live network packets using the Scapy library. It dissects each packet to extract relevant information such as IP addresses, port numbers, protocols, and payload data. This information is crucial for detecting unauthorized or malicious traffic patterns.

2. **Port Scanner:** This module uses Nmap to perform active port scans on devices within the network. It identifies open, closed, and filtered ports and highlights potential vulnerabilities or misconfigured services.

3. **Intrusion Detection Engine:** Based on packet data and traffic behavior, this engine analyzes unusual activity patterns—such as repeated access attempts, malformed packets, or traffic surges—using rule-based or anomaly-based techniques to detect intrusions.

4. **Visualization Dashboard:** The data captured and processed by the system is visualized through an interactive web dashboard using Dash and Plotly. The dashboard features:

   - Pie charts for protocol distribution.
   - Bar graphs for port scan results and packet frequency.
   - Real-time counters and alerts for detected threats.

5. **Automated Reporting:** For record-keeping and analysis, the system generates downloadable reports in PDF and PPT formats. These reports summarize packet data, port scan results, and detected threats.

6. **Performance Optimization:** The entire system runs on multithreaded logic, ensuring efficient handling of real-time packet capture, port scanning, visualization, and report generation without lag.

By integrating these components, CyberPulse IDS acts as a powerful tool for network administrators to monitor and secure networks proactively.

3.2 Requirements Specifications

To successfully implement and run the **CyberPulse IDS**, it is essential to define the minimum and recommended hardware and software requirements. These specifications ensure optimal performance, real-time traffic monitoring, data visualization, and smooth multitasking during scanning and report generation.

3.2.1 Hardware Requirements

➢ To run CyberPulse IDS efficiently, the following hardware specifications are recommended:

- **Processor:** Intel i5 or higher / AMD Ryzen 5 or higher
- **RAM:** Minimum 8 GB (16 GB recommended for real-time processing)
- **Storage:** At least 500 GB HDD or 256 GB SSD (SSD preferred for faster performance)
- **Network Interface Card (NIC):** Required for packet sniffing and port scanning
- **Display:** 1080p monitor or higher resolution for dashboard clarity
- **Peripheral Devices:** Keyboard, mouse, and optionally, speakers for alert sounds

3.2.2 Software Requirements

➢ The system relies on the following software tools and technologies:

- **Operating System:** Windows 10/11, Linux (Ubuntu 20.04 or higher)
- **Programming Language:** Python 3.8 or higher
- **Libraries/Frameworks:**
  - *Scapy* – for packet sniffing and analysis
  - *Nmap* – for port scanning and vulnerability detection
  - *Dash & Plotly* – for interactive dashboard visualization
  - *FPDF / python-pptx* – for automated report generation
  - *Threading module* – for multithreaded performance
- **Python Environment:** Anaconda / virtualenv (optional but recommended)
- **Browser:** Chrome, Firefox, or Edge – to access the web-based dashboard
- **Additional Tools:** PowerPoint (for viewing .ppt reports), PDF Reader

CHAPTER 4: SYSTEM DESIGN

4.1 Overview

CyberPulse IDS is a real-time intrusion detection system developed to monitor, analyze, and report malicious network activities. The system is built with modular design principles and comprises multiple interconnected components to ensure scalability, efficiency, and real-time performance.

The design integrates **packet sniffing**, **port scanning**, **rule-based intrusion detection**, **interactive data visualization**, and **automated reporting** into a single dashboard. This allows users (administrators/security analysts) to interact with the system, observe live traffic patterns, detect intrusions, and generate reports seamlessly.

The system employs **multithreading** to allow simultaneous execution of modules (e.g., sniffing, scanning, and UI updates) and ensures that performance remains optimal during high-volume traffic monitoring.

4.2 Components and Modules

1. Packet Sniffing Module (Scapy)

- Captures live packets from the network.
- Extracts protocol information, IPs, flags, and packet sizes.
- Stores the data for analysis and visualization.

2. Intrusion Detection Engine

- Applies rule-based logic to identify anomalies.
- Flags suspicious patterns (e.g., SYN floods, port scans).
- Logs alerts in real-time.

3. Port Scanning Module (Nmap)

- Periodically scans internal/external IP addresses.
- Identifies open ports, OS types, and known vulnerabilities.
- Helps detect unauthorized services or exposed hosts.

4. Preprocessing Module

- Cleans and transforms captured traffic data.
- Extracts features for the detection engine and visualization.
- Tags or filters unusual activity.

5. Dashboard Visualization (Dash & Plotly)

- Displays real-time graphs, pie charts, and traffic maps.
- Shows top source IPs, protocol usage, and alert logs.
- Interactive controls allow zooming, filtering, and time-based views.

6. Report Generator (FPDF & PowerPoint API)

- Converts analysis and traffic logs into downloadable reports.
- Supports PDF and PowerPoint formats.
- Automatically includes charts and timestamps for auditing.

7. Web Interface (Dash UI)

- Allows users to view alerts, download reports, and monitor data.
- Hosted locally or over a secure internal network.

4.3 Workflow of Proposed System

1. System Initialization

- The process starts when the system is launched.
- This includes loading all libraries, setting up network interfaces, initializing threads, and starting backend processes like Scapy, Nmap, Dash, etc.

2. Packet Sniffing Module (Scapy)

- This module begins real-time packet capture from the selected network interface (e.g., Wi-Fi, LAN).
- Captures vital information such as source/destination IP, ports, protocol type, flags (SYN, ACK, FIN), and packet size.
- Works continuously in the background, pushing live data to the analysis pipeline.

3. Port Scanning Module (Nmap)

- Periodically scans local or defined IP ranges for open ports and active services.
- Helps identify vulnerabilities or services exposed unintentionally (e.g., open Telnet port).
- The scan results are sent to the preprocessing module to be merged with sniffed data.

4. Preprocessing Module

- Cleans, filters, and standardizes the data from sniffing and scanning.
- Removes irrelevant packets, extracts only useful features (like packet size, protocol, TTL, etc.).
- This cleaned dataset is now ready for intrusion detection and visualization.

5. Feature Extraction

- Key features required for detecting attacks are extracted:
    - E.g., number of connections per IP, flag patterns (e.g., too many SYNs), high traffic volume in a short span.
- The feature vectors are either used directly in rule-based detection or prepared for a future ML-based model.

6. Intrusion Detection Engine

- This is the core module where actual detection happens.
- Based on defined rules (e.g., if IP sends > 100 SYN packets in 5 secs → raise alert).
- Logs all detected intrusions and passes that information to the dashboard and logging module.

7. Alert Logging System

- Stores logs of detected anomalies in a structured format (CSV/JSON).
- Includes time of attack, source/destination, detected attack type, and threat level.
- These logs are also used in generating reports.

8. Visualization Dashboard (Dash + Plotly)

- Presents live network activity and intrusion alerts in an interactive dashboard.
- Graphs include:
    - Traffic over time
    - Top IPs by activity
    - Alerts by severity
    - Protocol usage breakdown
- Users can zoom, filter, and explore the data in real-time.

9. Report Generator (PDF & PPT)

- At user request or at timed intervals, the system compiles data into:
    - PDF reports with summary tables and graphs.
    - PowerPoint reports suitable for presentations or audits.
- These are stored locally for review or sent to admins.

10. User Interface (Dash UI)

- Central place for all interaction:
    - Start/stop monitoring
    - View graphs and tables
    - Download reports
    - Change scan intervals or monitored interfaces

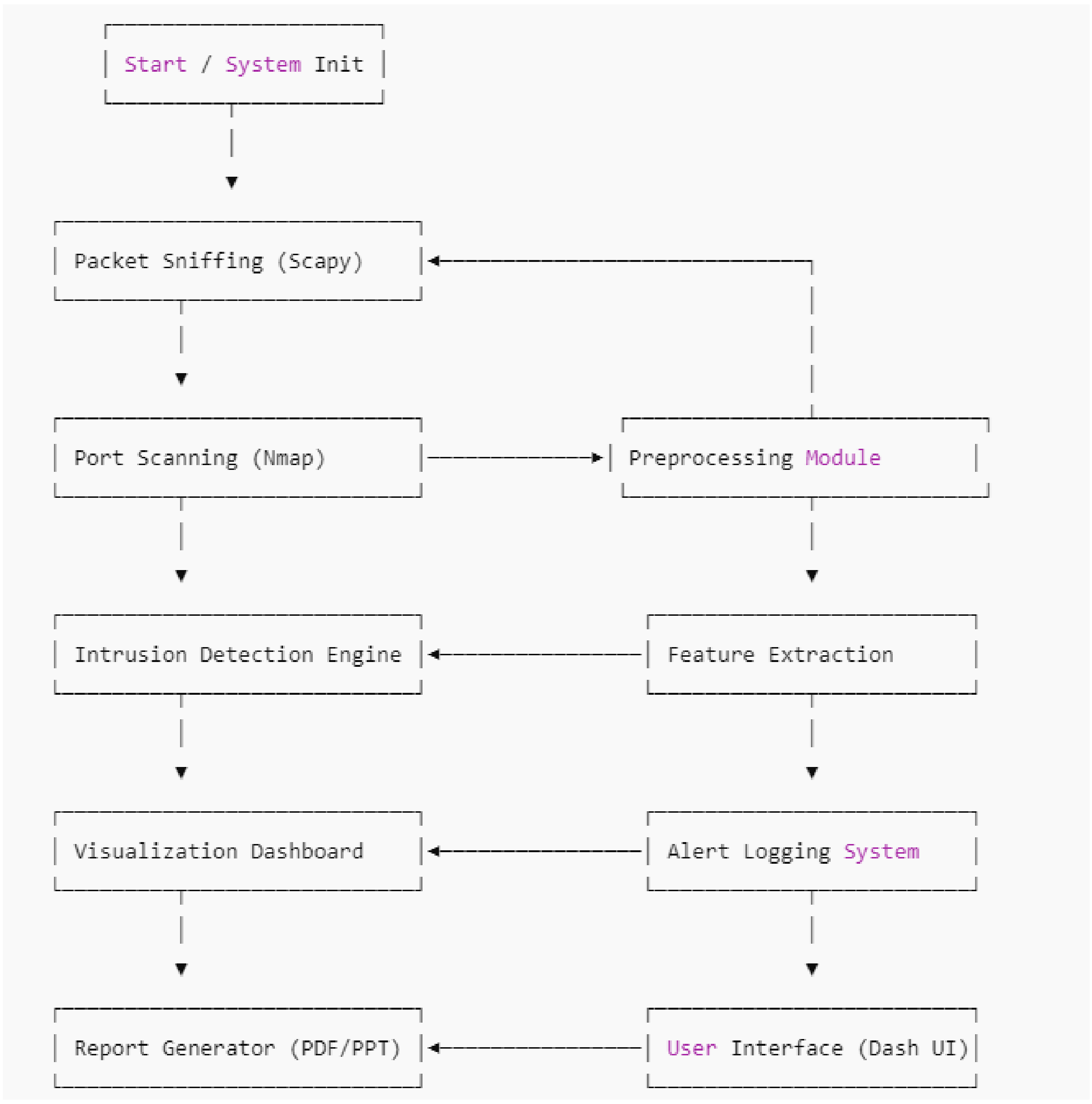


Figure 3: Workflow Chart of the Proposed System

CHAPTER 5:
IMPLEMENTATION AND TESTING

5.1 Data Collection and Preprocessing

The accuracy and reliability of any Intrusion Detection System (IDS) are highly dependent on the quality and structure of the input data. This section outlines the data collection process followed by CyberPulse IDS, along with the preprocessing techniques used to convert raw data into meaningful information for real-time threat detection.

5.1.1 Data Collection Process:

The data used in CyberPulse IDS is collected directly from network interfaces and system scans using the following tools:

- **Scapy (Packet Sniffer):**
  - ➢ Captures live packets from the network interface. Each packet contains details such as source and destination IP, port numbers, protocol type, packet length, flags, TTL, and timestamps.
- **Nmap (Port Scanner):**
  - ➢ Periodically scans for open ports and active services on connected hosts. It helps detect unusual port activity or exposure to vulnerable services.
- **Traffic Loggers:**
  - ➢ All captured packets and scan results are logged and stored in structured formats (CSV/JSON), timestamped and labeled for tracking.
- **Data Sources:**
  - ➢ Local network interface (Wi-Fi/Ethernet)
  - ➢ Live network traffic from virtual or physical environments
  - ➢ Synthetic attack simulations (e.g., port scans, SYN floods)

This collected data forms the base of all intrusion analysis and is pipelined to the preprocessing module.

5.1.2 Data Preprocessing Techniques:

Once the data is collected, it undergoes multiple preprocessing steps to ensure it is clean, consistent, and ready for threat detection or further analysis.

- **Data Cleaning**
  - ➢ Removal of malformed, incomplete, or duplicated packets.
  - ➢ Filtering out non-IP traffic (e.g., ARP, multicast, broadcast if not needed).
- **Feature Extraction**
  - ➢ Selecting only relevant features such as:
    - ➢ Source/Destination IP
    - ➢ Protocol Type (TCP, UDP, ICMP)
    - ➢ Packet Size
    - ➢ TCP Flags (SYN, ACK, FIN)

- Time-to-live (TTL)
- Port numbers

- **Encoding**
  - Categorical data (e.g., protocol type, flag status) is encoded into numerical format if used for machine learning later.
- **Timestamp Parsing**
  - All packet arrival times are converted into standard datetime formats for easier aggregation and trend analysis.
- **Anomaly Labeling**
  - Known attack behaviors (e.g., multiple SYNs without ACK) are labeled for detection or model training purposes.
- **Normalization (Future ML Scope)**
  - Numerical features like packet size or port range may be normalized for compatibility with machine learning models.

5.2 Model Development

To enhance CyberPulse IDS beyond rule-based detection, we explore a machine learning-based detection model that learns from network behavior patterns. This section explains the model's architecture, training process, and evaluation methodology.

5.2.1 Model Architecture

The ML model is designed as a binary classifier to distinguish between normal and malicious network activity using a supervised learning approach. Below is the proposed architecture:

- **Model Type:**

Feedforward Neural Network / Decision Tree Classifier / Random Forest (selectable based on accuracy-performance trade-off)

- **Input Layer:**

Extracted features from preprocessed network traffic, such as:

- Source/Destination IP (numerically encoded)
- Protocol (TCP/UDP/ICMP)
- Packet Size
- TCP Flags (SYN, ACK, FIN, etc.)
- Time-to-live (TTL)
- Connection duration
- Source and destination ports

❖ **Hidden Layers:**

- 2–3 Dense layers
- Activation: ReLU
- Dropout layers (optional) to prevent overfitting

❖ **Output Layer:**

Single neuron with Sigmoid activation (output: 0 = Normal, 1 = Intrusion)

5.2.2 Model Training

The model is trained on a labeled dataset of network traffic—either real-time captured data or publicly available intrusion detection datasets such as:

- CICIDS 2017
- NSL-KDD
- UNSW-NB15

❖ **Training Steps:**

1. **Data Preprocessing:**

- Feature extraction, normalization, encoding categorical values
- Train/test split (e.g., 80/20)

2. **Model Selection:**

- Use baseline models (Decision Trees, SVM) for comparison
- Build and tune neural network if more accuracy is required

3. **Loss Function & Optimizer:**

- **Loss Function:** Binary Cross Entropy
- **Optimizer:** Adam or SGD
- **Metrics:** Accuracy, Precision, Recall

4. **Epochs & Batch Size:**

- Train over 10–50 epochs with mini-batches (e.g., batch size = 64)

5. **Validation:**

- Apply cross-validation to reduce overfitting and evaluate generalization

5.2.3 Model Evaluation

After training, the model is evaluated using various metrics to test its reliability in detecting intrusions.

❖ **Evaluation Metrics:**

- **Accuracy:** % of correct predictions
- **Precision:** % of correct positive predictions (alerts)
- **Recall (Sensitivity):** % of intrusions correctly detected
- **F1-Score:** Harmonic mean of precision and recall
- **Confusion Matrix:** Breakdown of TP, FP, TN, FN

❖ **Testing Scenarios:**

- Simulated attacks (e.g., Port Scans, SYN Floods, Brute Force)
- Normal browsing and network activity
- Mixed traffic (real + synthetic)

The trained model can then be integrated into CyberPulse IDS to replace or supplement the rule-based engine, enabling smarter and more adaptive intrusion detection.

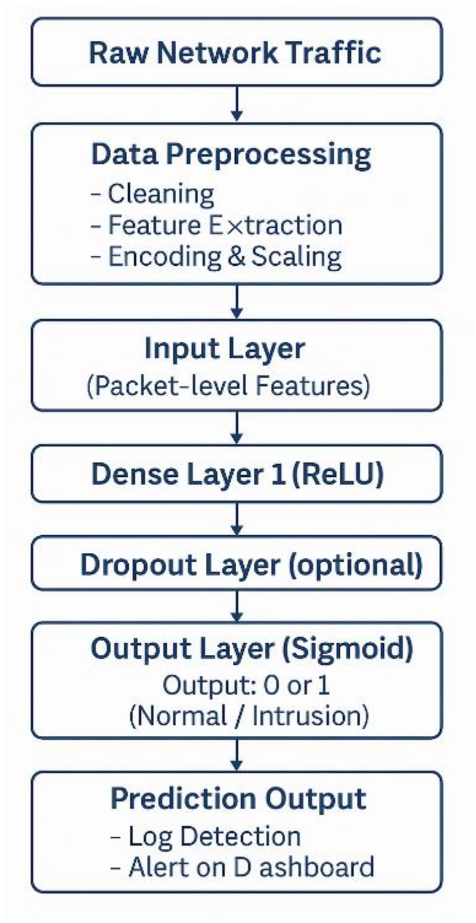


Figure 4: Model architecture Flow chart

5.3 Code Snippets

```
import dash
from dash import dcc, html, dash_table
import dash_bootstrap_components as dbc
import plotly.graph_objects as go
import plotly.express as px
from scapy.all import sniff, IP, sr1, TCP
import threading
import pandas as pd
import datetime
import ipaddress
import nmap  # python-nmap library
import os
from fpdf import FPDF
from pptx import Presentation
from pptx.util import Inches
from io import BytesIO
```

5.3.1 Initialize the Dash app

```
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.CYBORG])
server = app.server  # For deployment
```

5.3.2 Data storage

```
packet_data = []
external_scan_results = []
scan_progress = ""
```

5.3.3 Initialize nmap scanner

```
nm = nmap.PortScanner()

# IP Classification function
def is_internal(ip):
    internal_ranges = [
        '10.0.0.0/8',
        '172.16.0.0/12',
        '192.168.0.0/16',
        '127.0.0.0/8'
    ]
    try:
        ip_obj = ipaddress.IPv4Address(ip)
        for network in internal_ranges:
            if ip_obj in ipaddress.IPv4Network(network):
                return True
        return False
    except:
        return False
```

5.3.4 Packet capture callback

```
def packet_callback(packet):
    global packet_data
    if packet.haslayer(IP):
        src = packet[IP].src
        dst = packet[IP].dst
        new_entry = {
            "Timestamp": datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            "Source IP": src,
            "Destination IP": dst,
            "Protocol": packet[IP].proto,
            "Length": len(packet),
            "Source External": not is_internal(src),
            "Destination External": not is_internal(dst)
        }
        packet_data.append(new_entry)
        if len(packet_data) > 1000:
            packet_data.pop(0)
```

5.3.5 Start packet sniffing in a separate thread

```
def start_sniffing():
    sniff(prn=packet_callback, store=False)

threading.Thread(target=start_sniffing, daemon=True).start()
```

5.3.6 Lightweight port scanner using scapy

```
def scapy_port_scan(ip, ports=[80]):
    global external_scan_results, scan_progress
    results = []
    for port in ports:
        scan_progress = f"Scanning {ip}:{port}"
        packet = IP(dst=ip)/TCP(dport=port, flags="S")
        response = sr1(packet, timeout=1, verbose=0)
        if response and response.haslayer(TCP):
            if response[TCP].flags == 0x12:  # SYN-ACK
                results.append({"IP": ip, "Port": port, "Status": "Open"})
                external_scan_results.append({"IP": ip, "Port": port})
            elif response[TCP].flags == 0x14:  # RST-ACK
                results.append({"IP": ip, "Port": port, "Status": "Closed"})
        else:
            results.append({"IP": ip, "Port": port, "Status": "Filtered"})
    scan_progress = "Scan completed"
    return results
```

5.3.7 Nmap-based scanning

```
def nmap_scan(targets="127.0.0.1", ports="80"):
    global external_scan_results, scan_progress
    try:
        scan_progress = "Starting scan..."
        nm.scan(hosts=targets, ports=ports, arguments='-T4')
        scan_progress = "Scan completed"

        for host in nm.all_hosts():
            for proto in nm[host].all_protocols():
                ports = nm[host][proto].keys()
                for port in ports:
                    if nm[host][proto][port]['state'] == 'open':
                        external_scan_results.append({
                            "IP": host,
                            "Port": port,
                            "Service": nm[host][proto][port]['name']
                        })
        return True
    except Exception as e:
        scan_progress = f"Scan error: {str(e)}"
        return False
```

5.3.8Generate PDF report

```
def generate_pdf_report(df):
    pdf = FPDF()
    pdf.add_page()
    pdf.set_auto_page_break(auto=True, margin=15)
    pdf.set_font("Arial", size=12)
    pdf.cell(200, 10, txt="Intrusion Detection Report", ln=True, align="C")
    pdf.ln(10)
    for index, row in df.iterrows():
        pdf.cell(200, 10, txt=f"Timestamp: {row['Timestamp']}", ln=True)
        pdf.cell(200, 10, txt=f"Source IP: {row['Source IP']}", ln=True)
        pdf.cell(200, 10, txt=f"Destination IP: {row['Destination IP']}", ln=True)
        pdf.cell(200, 10, txt=f"Protocol: {row['Protocol']}", ln=True)
        pdf.cell(200, 10, txt=f"Packet Length: {row['Length']}", ln=True)
        pdf.cell(200, 10, txt=f"Source External: {row['Source External']}", ln=True)
        pdf.cell(200, 10, txt=f"Destination External: {row['Destination External']}", ln=True)
        pdf.ln(10)
    pdf_filename = "intrusion_report.pdf"
    pdf.output(pdf_filename)
    return pdf_filename
```

5.3.9 Generate PowerPoint report

```
def generate_ppt_report(df):
    prs = Presentation()

    # Title Slide
    title_slide_layout = prs.slide_layouts[0]
    slide = prs.slides.add_slide(title_slide_layout)
    title = slide.shapes.title
    subtitle = slide.placeholders[1]
    title.text = "Intrusion Detection Report"
    subtitle.text = "Generated on: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

  # Summary Slide
    summary_slide_layout = prs.slide_layouts[5]  # Blank slide
    slide = prs.slides.add_slide(summary_slide_layout)
    title = slide.shapes.title
    title.text = "Summary"
    left = top = Inches(1)
    width = height = Inches(6)
    txBox = slide.shapes.add_textbox(left, top, width, height)
    tf = txBox.text_frame
    tf.text = (
        f"Total Packets Captured: {len(df)}\n"
        f"Unique Source IPs: {df['Source IP'].nunique()}\n"
        f"Unique Destination IPs: {df['Destination IP'].nunique()}\n"
        f"Protocols Detected: {', '.join(map(str, df['Protocol'].unique()))}"
    )

    # Protocol Distribution Chart
    protocol_counts = df['Protocol'].value_counts().to_dict()
    fig_protocol = go.Figure(go.Pie(
        labels=list(protocol_counts.keys()),
        values=list(protocol_counts.values()),
        hole=0.3,
        marker=dict(colors=px.colors.sequential.RdBu)
    ))
    fig_protocol.update_layout(
        title="Protocol Distribution",
        plot_bgcolor='#222',
        paper_bgcolor='#222',
        font=dict(color='white')
    )
    protocol_image = BytesIO(fig_protocol.to_image(format="png"))
    slide = prs.slides.add_slide(summary_slide_layout)
    slide.shapes.title.text = "Protocol Distribution"
    slide.shapes.add_picture(protocol_image, Inches(1), Inches(2), width=Inches(6))

    # Save the PPT
    ppt_filename = "intrusion_report.pptx"
    prs.save(ppt_filename)
    return ppt_filename
```

5.3.10 Dashboard layout

```
app.layout = dbc.Container([
    dbc.NavbarSimple(
        brand="Intrusion Detection Dashboard",
        color="primary",
        dark=True,
        className="mb-3"
    ),
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Live Packet Monitoring"),
                dbc.CardBody(dcc.Graph(id='live-packet-chart'))
            ]),
        ], width=6),
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Protocol Breakdown"),
                dbc.CardBody(dcc.Graph(id='protocol-chart'))
            ]),
        ], width=6)
    ]),
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("External Traffic Analysis"),
                dbc.CardBody(dcc.Graph(id='external-traffic-chart'))
            ]),
        ], width=12)
    ]),
    dbc.Row([
        dbc.Col([
            dbc.Button("Generate Report", id="report-btn", color="success", className="mb-3"),
            dcc.Download(id="download-csv"),
            dcc.Download(id="download-pdf"),
            dcc.Download(id="download-ppt"),
            dbc.Button("Scan External IPs", id="scan-btn", color="warning", className="ml-3 mb-3"),
            dbc.Input(id="scan-target", placeholder="Enter IP or range (e.g., 192.168.1.1 or 192.168.1.0/24)",
                type="text", className="mb-2"),
            dbc.Input(id="scan-ports", placeholder="Enter ports (e.g., 80,443 or 1-100)",
                type="text", value="80", className="mb-2"),
            dbc.RadioItems(
                id="scan-method",
                options=[
                    {"label": "Quick Scan (Scapy)", "value": "scapy"},
                    {"label": "Detailed Scan (Nmap)", "value": "nmap"}
                ],
                value="scapy",
                inline=True,
                className="mb-2"
            ),
            html.Div(id="scan-progress", className="text-info mb-2"),
            html.Div(id="scan-results", className="mt-3")
        ], width=12)
    ]),
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Packet Details Table"),
                dbc.CardBody(dash_table.DataTable(
                    id='packet-table',
                    style_table={'overflowX': 'auto'},
                    style_cell={
                        'textAlign': 'center',
                        'padding': '5px',
                        'backgroundColor': '#222',
                        'color': 'white',
                        'maxWidth': '150px'
                    },
                    style_header={
                        'backgroundColor': '#444',
                        'color': 'white',
                        'fontWeight': 'bold'
                    }
                ))
            ])
        ], width=12)
    ]),
    dcc.Interval(id='interval-update', interval=2000, n_intervals=0)
])
```

5.3.11 Dashboard update callback

```
@app.callback(
    [
        dash.Output('live-packet-chart', 'figure'),
        dash.Output('protocol-chart', 'figure'),
        dash.Output('external-traffic-chart', 'figure'),
        dash.Output('packet-table', 'data'),
        dash.Output('packet-table', 'columns')
    ],
    [dash.Input('interval-update', 'n_intervals')]
)
def update_dashboard(n):
    global packet_data
    if not packet_data:
        return go.Figure(), go.Figure(), go.Figure(), [], []
```

```
df = pd.DataFrame(packet_data)

# Line Chart for Packet Monitoring
fig_packet = go.Figure()
fig_packet.add_trace(go.Scatter(
    x=df["Timestamp"],
    y=df["Length"],
    mode='lines+markers',
    name='Packet Length',
    marker=dict(color='#EF553B')
))
fig_packet.update_layout(
    title="Live Packet Monitoring",
    xaxis_title="Time",
    yaxis_title="Packet Size",
    template="plotly_dark",
    plot_bgcolor='#222',
    paper_bgcolor='#222',
    font=dict(color='white')
)

# Protocol Pie Chart
protocol_counts = df['Protocol'].value_counts().to_dict()
fig_protocol = go.Figure(go.Pie(
    labels=list(protocol_counts.keys()),
    values=list(protocol_counts.values()),
    hole=0.3,
    marker=dict(colors=px.colors.sequential.RdBu)
))
fig_protocol.update_layout(
    title="Protocol Distribution",
    plot_bgcolor='#222',
    paper_bgcolor='#222',
    font=dict(color='white')
)

# External Traffic Bar Chart
external_traffic = df.groupby(['Source External', 'Destination External']).size().reset_index(name='Count')
external_traffic['Category'] = external_traffic.apply(
    lambda row: f"{'External' if row['Source External'] else 'Internal'} → {'External' if row['Destination
External'] else 'Internal'}",
    axis=1
)
fig_external = px.bar(
    external_traffic,
    x='Category',
    y='Count',
    color='Count',
    color_continuous_scale='Teal',
    title='Internal/External Traffic Patterns'
)
fig_external.update_layout(
    plot_bgcolor='#222',
    paper_bgcolor='#222',
    font=dict(color='white'),
    xaxis=dict(tickangle=45)
)
```

```
    # Prepare table data and columns
    columns = [
        {"name": "Timestamp", "id": "Timestamp"},
        {"name": "Source IP", "id": "Source IP"},
        {"name": "Destination IP", "id": "Destination IP"},
        {"name": "Protocol", "id": "Protocol"},
        {"name": "Length", "id": "Length"},
        {"name": "Src External", "id": "Source External"},
        {"name": "Dst External", "id": "Destination External"}
    ]
    table_data = df.to_dict('records')
    return fig_packet, fig_protocol, fig_external, table_data, columns

# Report generation callback
@app.callback(
    [
        dash.Output("download-csv", "data"),
        dash.Output("download-pdf", "data"),
        dash.Output("download-ppt", "data")
    ],
    [dash.Input("report-btn", "n_clicks")],
    prevent_initial_call=True
)
def generate_report(n_clicks):
    df = pd.DataFrame(packet_data)
    if df.empty:
        return dcc.send_bytes(b"Empty report", "report.txt"), None, None

    # Generate CSV report
    csv_filename = "intrusion_report.csv"
    df.to_csv(csv_filename, index=False)

    # Generate PDF report
    pdf_filename = generate_pdf_report(df)

    # Generate PPT report
    ppt_filename = generate_ppt_report(df)

    # Return all files as separate downloads
    csv_data = dcc.send_file(csv_filename)
    pdf_data = dcc.send_file(pdf_filename)
    ppt_data = dcc.send_file(ppt_filename)

    # Clean up files after download
    os.remove(csv_filename)
    os.remove(pdf_filename)
    os.remove(ppt_filename)

    return csv_data, pdf_data, ppt_data
```

5.3.12 External IP scan callback

```
@app.callback(
    [
        dash.Output("scan-results", "children"),
        dash.Output("scan-progress", "children")
    ],
    [dash.Input("scan-btn", "n_clicks")],
    [
        dash.State("scan-target", "value"),
        dash.State("scan-ports", "value"),
        dash.State("scan-method", "value")
    ],
    prevent_initial_call=True
)
def scan_external_ips(n_clicks, target, ports, method):
    global external_scan_results, scan_progress
    external_scan_results.clear()

    if not target:
        return "Please enter a target IP or range", "Error: No target specified"

    ports = ports if ports else "80"

    try:
        if method == "scapy":
            scan_progress = "Starting quick scan with Scapy..."
            # For scapy, we'll just scan the first IP if a range is given
            target_ip = target.split('/')[0] if '/' in target else target
            port_list = [int(p) for p in ports.split(',')] if ',' in ports else [int(ports)]
            results = scapy_port_scan(target_ip, port_list)

            if not results:
                return "No open ports found", "Scan completed - no open ports"

            scan_df = pd.DataFrame(results)
            return [
                dash_table.DataTable(
                    data=scan_df.to_dict('records'),
                    columns=[{"name": i, "id": i} for i in scan_df.columns],
                    style_table={'overflowX': 'auto'},
                    style_cell={
                        'textAlign': 'center',
                        'padding': '5px',
                        'backgroundColor': '#222',
                        'color': 'white',
                        'maxWidth': '150px'
                    },
                    style_header={
                        'backgroundColor': '#444',
                        'color': 'white',
                        'fontWeight': 'bold'
                    }
                )
            ], "Scan completed"

        elif method == "nmap":
            scan_progress = "Starting detailed scan with Nmap..."
            success = nmap_scan(target, ports)

            if not external_scan_results:
                return "No open ports found", "Scan completed - no open ports"

            scan_df = pd.DataFrame(external_scan_results)
            return [
                dash_table.DataTable(
                    data=scan_df.to_dict('records'),
                    columns=[{"name": i, "id": i} for i in scan_df.columns],
                    style_table={'overflowX': 'auto'},
                    style_cell={
                        'textAlign': 'center',
                        'padding': '5px',
                        'backgroundColor': '#222',
                        'color': 'white',
                        'maxWidth': '150px'
                    },
                    style_header={
                        'backgroundColor': '#444',
                        'color': 'white',
                        'fontWeight': 'bold'
                    }
                )
            ], "Scan completed"

    except Exception as e:
        return f"Scan error: {str(e)}", f"Error: {str(e)}"

if __name__ == '__main__':
    app.run_server(debug=True, host='127.0.0.1', port=8050)
```

CHAPTER 6: RESULTS

6.1 Testing and Results

The CyberPulse IDS system underwent functional testing in a simulated network environment to evaluate its performance, accuracy, and responsiveness. During testing, the system was able to:

- Capture and analyze live network traffic using Scapy.
- Detect anomalies and potential intrusions, such as repeated access attempts and unusual protocol usage.
- Perform accurate port scanning with Nmap, identifying open and vulnerable ports.
- Display real-time data on the dashboard using interactive charts and graphs.
- Generate automated reports in PDF and PPT formats summarizing threats and traffic insights.
- Operate smoothly using multithreading, ensuring real-time performance with minimal lag.

These results confirm the system's ability to effectively monitor networks, detect threats, and present the findings through a user-friendly interface.

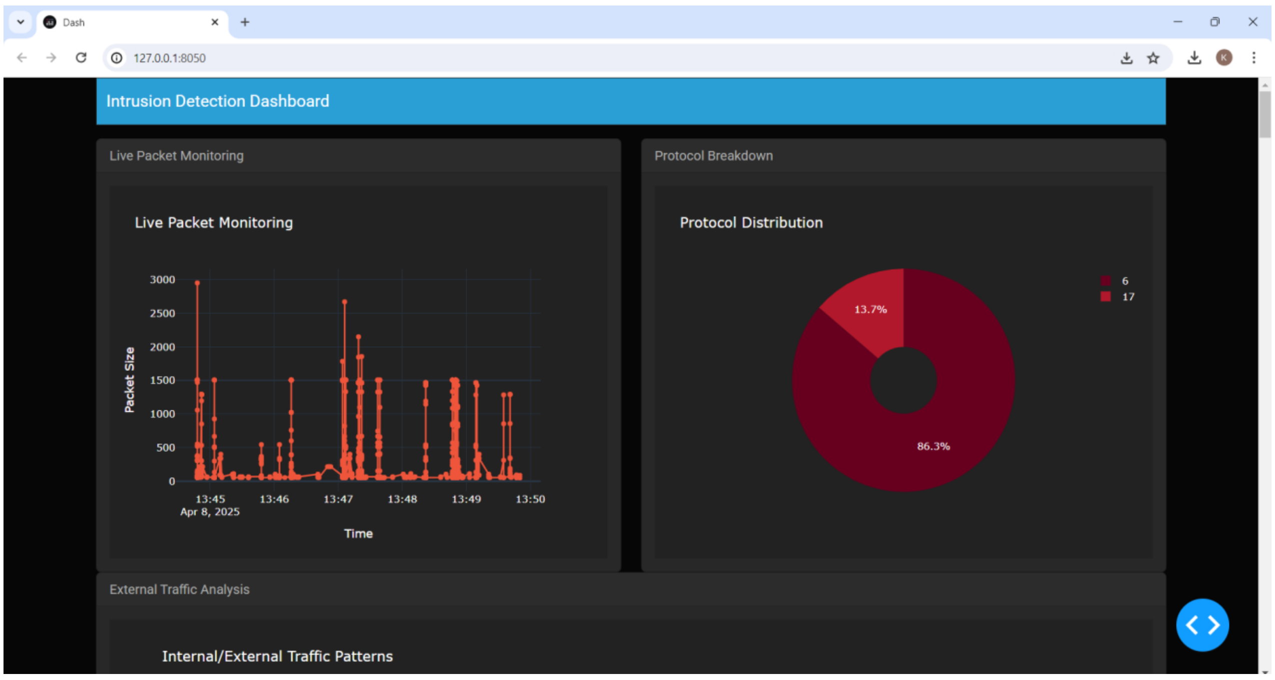


Figure 5: CyberPulse IDS dashboard

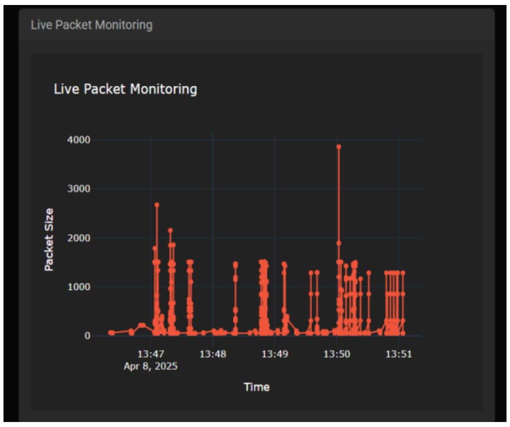


Figure 6:Live Packet Monitoring Graph

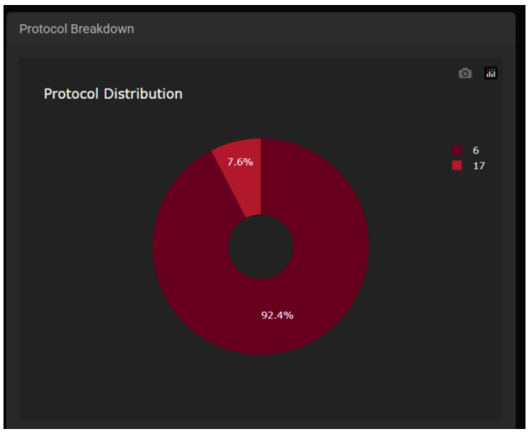


Figure 7: Protocol Breakdown Piechart

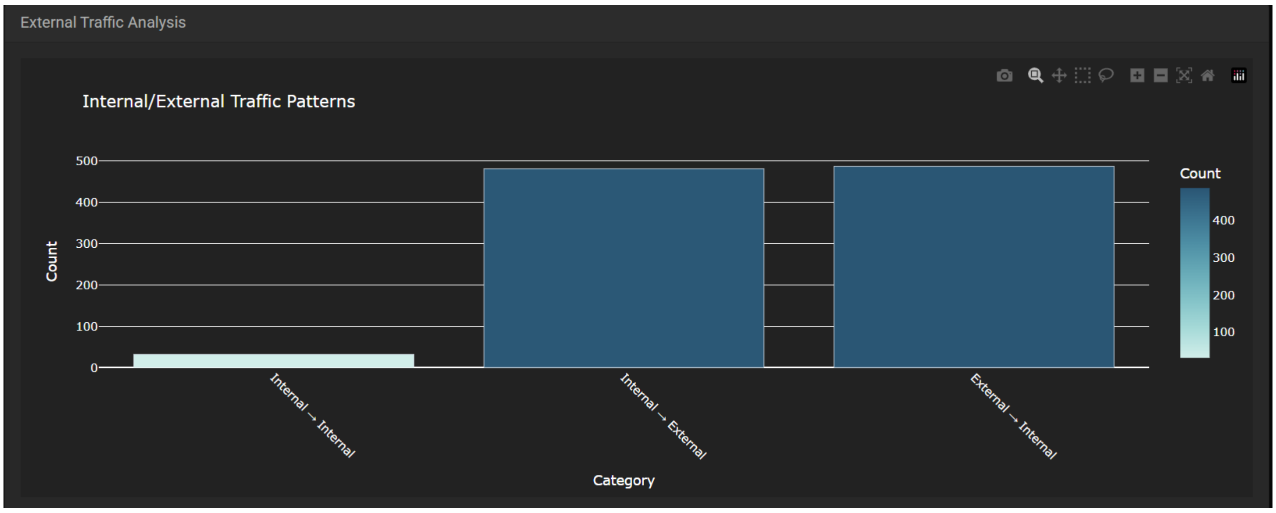


Figure 8:External Traffic Analysis BarGraph

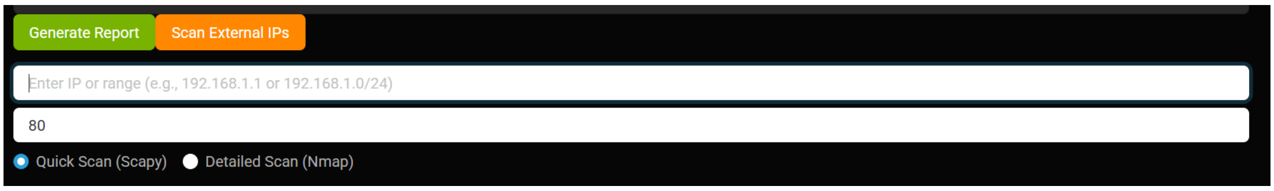
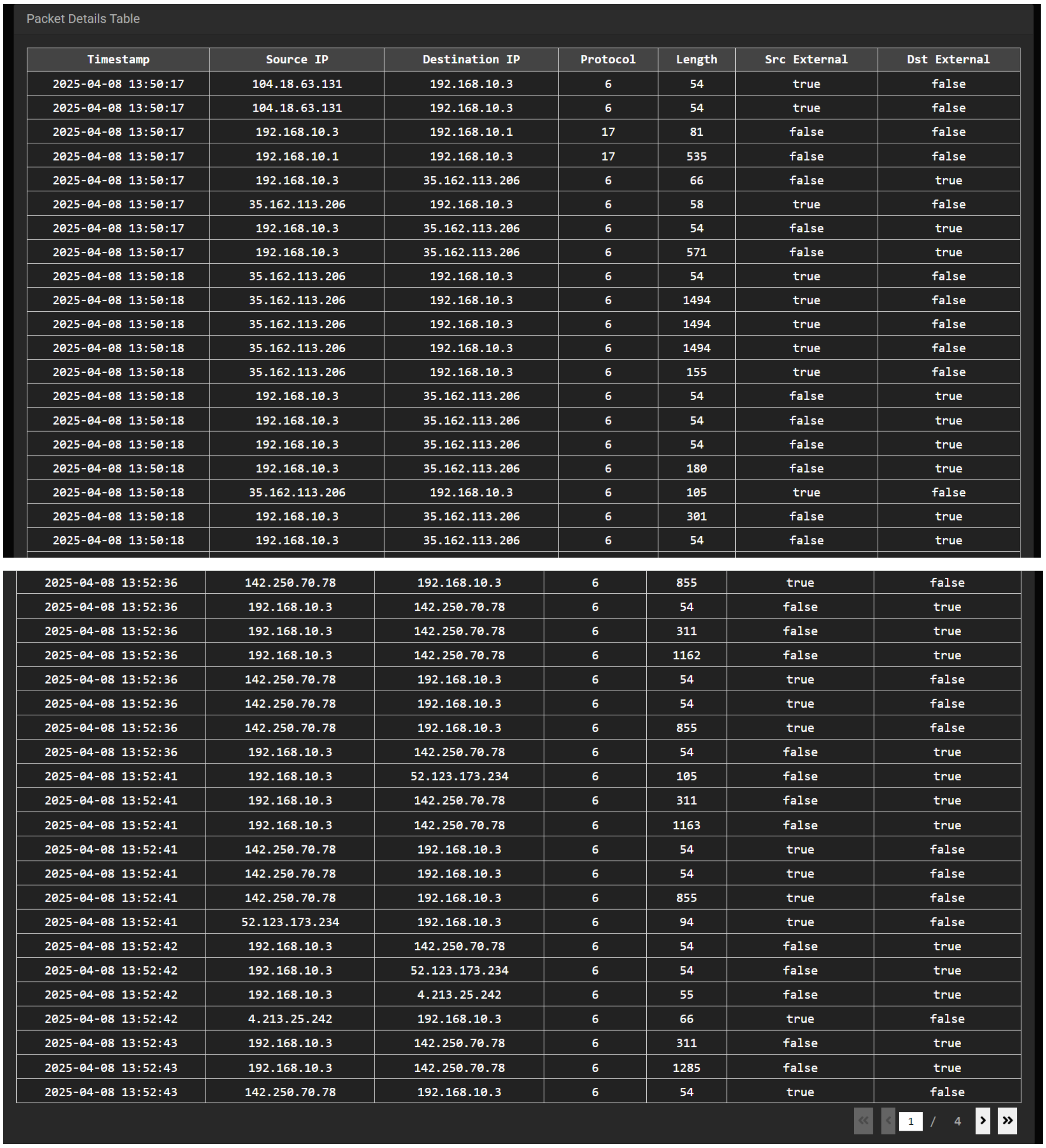


Figure 9:Generate Report and Scan IPs option



Packet Details Table

Timestamp	Source IP	Destination IP	Protocol	Length	Src External	Dst External
2025-04-08 13:50:17	104.18.63.131	192.168.10.3	6	54	true	false
2025-04-08 13:50:17	104.18.63.131	192.168.10.3	6	54	true	false
2025-04-08 13:50:17	192.168.10.3	192.168.10.1	17	81	false	false
2025-04-08 13:50:17	192.168.10.1	192.168.10.3	17	535	false	false
2025-04-08 13:50:17	192.168.10.3	35.162.113.206	6	66	false	true
2025-04-08 13:50:17	35.162.113.206	192.168.10.3	6	58	true	false
2025-04-08 13:50:17	192.168.10.3	35.162.113.206	6	54	false	true
2025-04-08 13:50:17	192.168.10.3	35.162.113.206	6	571	false	true
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	54	true	false
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	1494	true	false
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	1494	true	false
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	1494	true	false
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	155	true	false
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	54	false	true
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	54	false	true
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	54	false	true
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	180	false	true
2025-04-08 13:50:18	35.162.113.206	192.168.10.3	6	105	true	false
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	301	false	true
2025-04-08 13:50:18	192.168.10.3	35.162.113.206	6	54	false	true
2025-04-08 13:52:36	142.250.70.78	192.168.10.3	6	855	true	false
2025-04-08 13:52:36	192.168.10.3	142.250.70.78	6	54	false	true
2025-04-08 13:52:36	192.168.10.3	142.250.70.78	6	311	false	true
2025-04-08 13:52:36	192.168.10.3	142.250.70.78	6	1162	false	true
2025-04-08 13:52:36	142.250.70.78	192.168.10.3	6	54	true	false
2025-04-08 13:52:36	142.250.70.78	192.168.10.3	6	54	true	false
2025-04-08 13:52:36	142.250.70.78	192.168.10.3	6	855	true	false
2025-04-08 13:52:36	192.168.10.3	142.250.70.78	6	54	false	true
2025-04-08 13:52:41	192.168.10.3	52.123.173.234	6	105	false	true
2025-04-08 13:52:41	192.168.10.3	142.250.70.78	6	311	false	true
2025-04-08 13:52:41	192.168.10.3	142.250.70.78	6	1163	false	true
2025-04-08 13:52:41	142.250.70.78	192.168.10.3	6	54	true	false
2025-04-08 13:52:41	142.250.70.78	192.168.10.3	6	54	true	false
2025-04-08 13:52:41	142.250.70.78	192.168.10.3	6	855	true	false
2025-04-08 13:52:41	52.123.173.234	192.168.10.3	6	94	true	false
2025-04-08 13:52:42	192.168.10.3	142.250.70.78	6	54	false	true
2025-04-08 13:52:42	192.168.10.3	52.123.173.234	6	54	false	true
2025-04-08 13:52:42	192.168.10.3	4.213.25.242	6	55	false	true
2025-04-08 13:52:42	4.213.25.242	192.168.10.3	6	66	true	false
2025-04-08 13:52:43	192.168.10.3	142.250.70.78	6	311	false	true
2025-04-08 13:52:43	192.168.10.3	142.250.70.78	6	1285	false	true
2025-04-08 13:52:43	142.250.70.78	192.168.10.3	6	54	true	false

Figure 10: Packets Detail Table

Report Generation

CyberPulse IDS includes a robust report generation feature that enables users to save and analyze captured network data in three different formats: PDF, PowerPoint (PPT), and CSV. This functionality supports better documentation, review, and sharing of intrusion detection results.

1. PDF Report

The system automatically generates a well-structured PDF report that includes:

- Summary of captured packets
- Port scan results
- Detected threats
- Timestamps and statistics

The PDF format is ideal for documentation, audits, and official reporting.

2. PowerPoint (PPT) Report

CyberPulse IDS creates a presentation-ready PPT file containing:

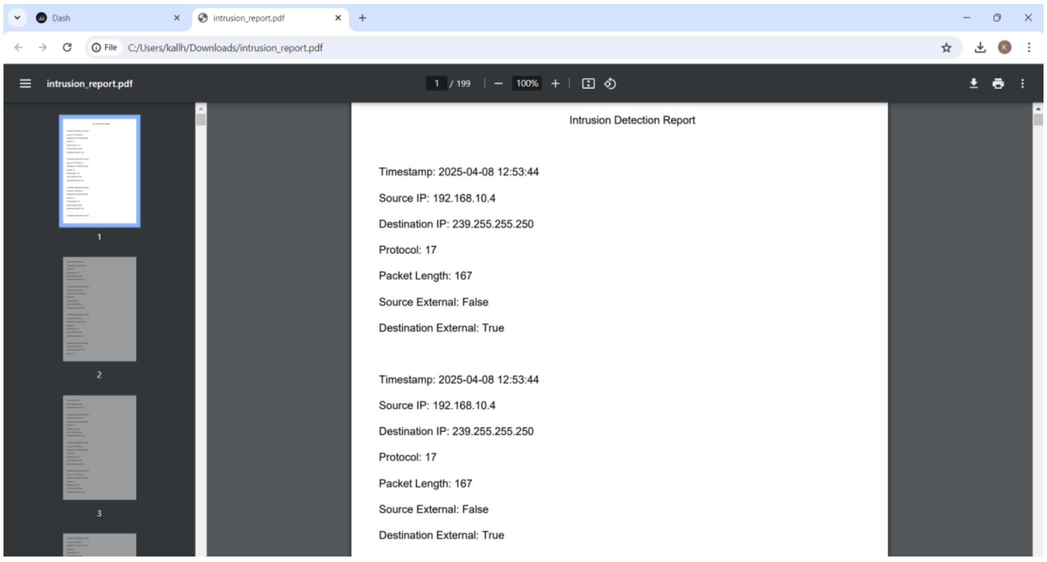
- Graphical summaries
- Key metrics
- Charts and scan results

This format is suitable for meetings, seminars, and quick visualization of findings.

3. CSV Export

A CSV file is also generated, which includes raw packet data and scan results in a tabular format. This allows for:

- Easy import into Excel or other data analysis tools
- Custom filtering and deeper analysis



Intrusion Detection Report

Timestamp: 2025-04-08 12:53:44

Source IP: 192.168.10.4

Destination IP: 239.255.255.250

Protocol: 17

Packet Length: 167

Source External: False

Destination External: True

Timestamp: 2025-04-08 12:53:44

Source IP: 192.168.10.4

Destination IP: 239.255.255.250

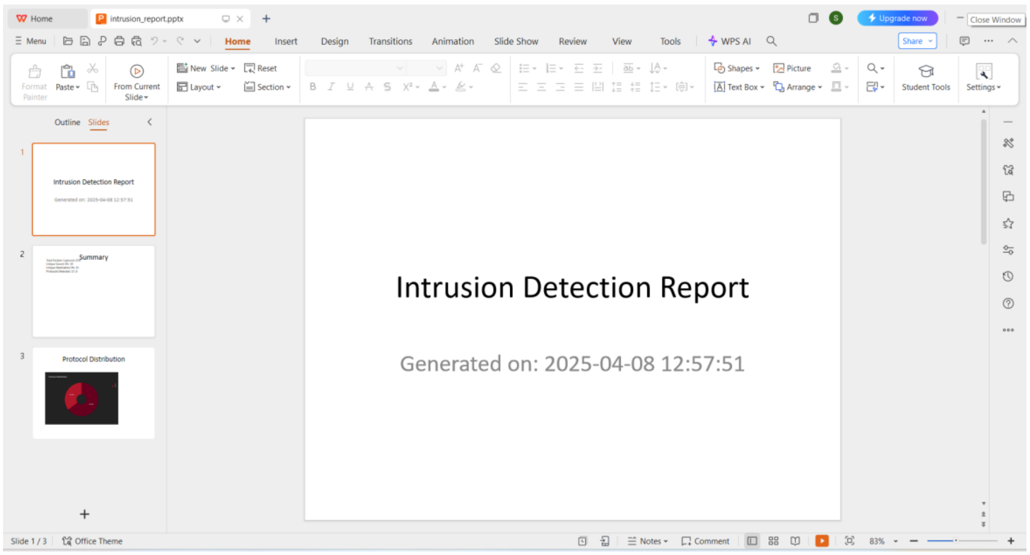
Protocol: 17

Packet Length: 167

Source External: False

Destination External: True

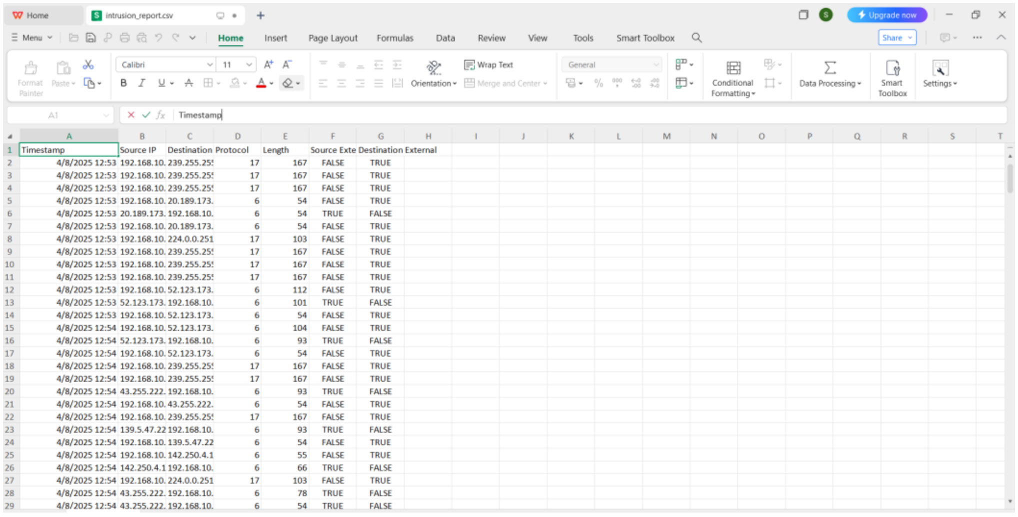
Figure 11(a): Report in pdf format



Intrusion Detection Report

Generated on: 2025-04-08 12:57:51

Figure 11(b): Report in ppt format



Timestamp	Source IP	Destination	Protocol	Length	Source Exte	Destination External
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	20.189.173.	6	54	FALSE	TRUE
4/8/2025 12:53	20.189.173.	192.168.10.	6	54	TRUE	FALSE
4/8/2025 12:53	192.168.10.	20.189.173.	6	54	FALSE	TRUE
4/8/2025 12:53	192.168.10.	224.0.0.251	17	103	FALSE	TRUE
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:53	192.168.10.	52.123.173.	6	112	FALSE	TRUE
4/8/2025 12:53	52.123.173.	192.168.10.	6	101	TRUE	FALSE
4/8/2025 12:53	192.168.10.	52.123.173.	6	54	FALSE	TRUE
4/8/2025 12:54	192.168.10.	52.123.173.	6	104	FALSE	TRUE
4/8/2025 12:54	52.123.173.	192.168.10.	6	93	TRUE	FALSE
4/8/2025 12:54	192.168.10.	52.123.173.	6	54	FALSE	TRUE
4/8/2025 12:54	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:54	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:54	43.255.222.	192.168.10.	6	93	TRUE	FALSE
4/8/2025 12:54	192.168.10.	43.255.222.	6	54	FALSE	TRUE
4/8/2025 12:54	192.168.10.	239.255.25	17	167	FALSE	TRUE
4/8/2025 12:54	139.5.47.22	192.168.10.	6	93	TRUE	FALSE
4/8/2025 12:54	192.168.10.	139.5.47.22	6	54	FALSE	TRUE
4/8/2025 12:54	192.168.10.	142.250.4.1	6	55	FALSE	TRUE
4/8/2025 12:54	142.250.4.1	192.168.10.	6	66	TRUE	FALSE
4/8/2025 12:54	192.168.10.	224.0.0.251	17	103	FALSE	TRUE
4/8/2025 12:54	43.255.222.	192.168.10.	6	78	TRUE	FALSE
4/8/2025 12:54	43.255.222.	192.168.10.	6	54	TRUE	FALSE

Figure 11(c): Report in csv format

CHAPTER 7:
CONCLUSION AND FUTURE WORK

7.1 Conclusion

The **CyberPulse IDS** project successfully demonstrates the development of a real-time **Intrusion Detection System** using open-source technologies. By integrating **packet sniffing**, **port scanning**, and **anomaly-based intrusion detection**, the system provides deep insights into network traffic behavior. The inclusion of a user-friendly **dashboard**, along with automated **PDF and PPT reporting**, enhances the accessibility and usability of the tool for both technical and non-technical users.

The use of **Python, Scapy, Nmap, Dash, and Plotly**, along with multithreading, ensures that the system operates efficiently and delivers accurate, timely threat detection. This project not only strengthens network defense mechanisms but also raises cybersecurity awareness by offering live monitoring and reporting features.

7.2 Future Work

While CyberPulse IDS is a functional and effective solution, several improvements and enhancements can be considered for future development:

- **AI-Powered Intrusion Detection:** Integrate machine learning models to detect unknown or zero-day attacks more accurately.
- **Cloud-Based Integration:** Enable the IDS to operate on cloud platforms for scalable and distributed network monitoring.
- **Real-Time Alert System:** Implement SMS/email alerts or push notifications for immediate response to detected threats.
- **User Management System:** Add role-based access control for multi-user environments.
- **Advanced Logging and Storage:** Introduce a secure database (e.g., MongoDB, Elasticsearch) for long-term data storage and log analysis.
- **Mobile Dashboard Access:** Optimize the dashboard for mobile devices to enable monitoring on the go.

These future enhancements would make CyberPulse IDS more scalable, intelligent, and adaptable to modern network environments.

CHAPTER 8: REFERENCES

8.1 References

https://www.amazon.com/dp/013452733X

https://www.springer.com/gp/book/9780387873726

https://www.pearson.com/store/p/computer-security-principles-and-practice/P100000880392

https://nostarch.com/networksecuritymonitoring

https://www.elsevier.com/books/violent-python/oconnor/978-1-59749-957-6

https://www.unb.ca/cic/datasets/nsl.html

https://www.unb.ca/cic/datasets/ids-2017.html

https://scapy.readthedocs.io/en/latest/

https://nmap.org/book/

https://dash.plotly.com/introduction

https://www.sciencedirect.com/science/article/abs/pii/S0167404820302923

https://ieeexplore.ieee.org/document/8688737

https://ieeexplore.ieee.org/document/9010704

https://arxiv.org/abs/2001.11710

https://www.packtpub.com/product/python-for-cybersecurity/9781800206012

https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection

https://data.mendeley.com/datasets/k9v9c9dkc8/1

https://ieeexplore.ieee.org/document/10008945

https://www.sciencedirect.com/science/article/pii/S2352914821000673

https://arxiv.org/abs/2207.09761