Smart Unified Threat Management System (SUTMS)

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE AWARD OF THE
DEGREE OF **BACHELOR OF TECHNOLOGY**
IN **INFORMATION TECHNOLOGY**
OF THE ANNA UNIVERSITY

MINI PROJECT WORK

Submitted by

ASWIN S

71772218102

JANARDHAN

71772218116

PRAGHADEESH S

71772218138

IYAPPAN K

71772218L04

2025

Under the Guidance of

Dr. M. Blessy Queen Mary, M.E., Ph.D.

DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University)

COIMBATORE - 641 013

DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University)

COIMBATORE - 641 013

MINI PROJECT WORK

NOVEMBER 2025

This is to certify that this project work entitled

Smart Unified Threat Management System (SUTMS)

is the bonafide record of project work done by

ASWIN S
71772218102
JANARDHAN
71772218116
PRAGHADEESH S
71772218138
IYAPPAN K
71772218L04

of B.TECH INFORMATION TECHNOLOGY during the year 2025 - 2026

Project Guide	Head of the Department
Dr. M. Blessy Queen Mary, M.E., Ph.D.	**Dr. S. RATHI M.E., Ph.D.**

Submitted for the Project Viva-Voce examination held on _____

Internal Examiner	External Examiner

ACKNOWLEDGEMENT

Great achievements are not possible without standing on the shoulders of giants. Without the active involvement of the following experts this project would not have been a reality.

We express our sincere gratitude to **Dr. K. Manonmani, M.E., Ph.D**., Principal, Government College of Technology, Coimbatore for providing us all facilities that we needed for the completion of this project.

We whole-heartedly express our thankfulness and gratitude to **Dr. S. Rathi, M.E. Ph.D**., Professor and Head of the Department of Information Technology, Government College of Technology, for helping us to successfully carry out this project.

Our thankfulness and gratitude to our respectable project guide **Dr. M. Blessy Queen Mary, M.E., Ph.D.,** Assistant Professor, who has been an immense help through the various phases of the project. With her potent ideas and excellent guidance, we were able to comprehend the essential aspects involved.

We would like to thank our faculty advisor **Dr. T. Suguna, M.Tech., Ph.D.,** Assistant Professor for her continuous support and encouragement throughout this project.

We extend our sincere thanks to the staff members of Information Technology department, **Dr. R. Devi, M.Tech., Ph.D.,** Assistant Professor**, Dr. C. Aswini, M.E.**, **Ph.D,** Assistant Professor, **Prof. M. Jeyanthi, M.Tech**., Assistant Professor, **Dr. S. Gladson Oliver, M.Tech., Ph.D.,** Assistant Professor, **Dr. R. Malavika, M.Tech., Ph.D.,** Assistant Professor, for rendering their help for the completion of this project. We also thank all our friends for their cooperation and suggestions towards the successful completion of this project.

SYNOPSIS

In today's world, our homes are more connected than ever — from smartphones and laptops to smart TVs and IoT devices. But as our connectivity grows, so do the security risks. Most home routers only focus on keeping us online; they rarely protect us from modern cyber threats like phishing, malware, or unauthorized access. Corporate networks use advanced security systems called Unified Threat Management (UTM) devices, but these are often expensive and too complex for regular home users.

To solve this problem, our project introduces the **Smart Unified Threat Management System (SUTMS)** — a simple, affordable, and efficient security solution designed specifically for home networks. The goal is to give personal and small-office users the same level of protection that big companies enjoy, without the need for costly hardware or expert setup.

SUTMS brings together three key layers of protection:

- **NTOPng** keeps an eye on all network traffic and detects abnormal patterns.
- **Suricata** inspects the data deeply to find and stop malicious activities.
- **IPTables** acts as a smart firewall that blocks harmful connections in real time.

The system also connects to global **threat intelligence feeds (IoCs)**, which help it stay updated on the latest cyber threats automatically. A simple **web dashboard** allows users to manage firewall rules, check device activity, and even pause blocking if needed — making it friendly for everyone, even non-technical users.

Built on a **Raspberry Pi** with open-source tools, SUTMS provides strong, real-time protection using very little processing power. Tests using real-world datasets showed **around 99% detection accuracy** while using **55% less memory** than traditional systems.

In short, SUTMS transforms a basic home router into a smart guardian that continuously monitors, detects, and defends against digital threats — keeping families, remote workers, and small businesses safe online.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation	Full Form
1. **SUTMS**	Smart Unified Threat Management System
2. **UTM**	Unified Threat Management
3. **IDS**	Intrusion Detection System
4. **IPS**	Intrusion Prevention System
5. **IoC**	Indicator of Compromise
6. **IoT**	Internet of Things
7. **STIX**	Structured Threat Information Expression
8. **TAXII**	Trusted Automated eXchange of Intelligence Information
9. **DHCP**	Dynamic Host Configuration Protocol
10. **DNS**	Domain Name System
11. **LAN**	Local Area Network
12. **WAN**	Wide Area Network
13. **IPTables**	Internet Protocol Tables
14. **NFQUEUE**	Netfilter Queue
15. **API**	Application Programming Interface
16. **GUI**	Graphical User Interface
17. **CPU**	Central Processing Unit
18. **RAM**	Random Access Memory
19. **PI**	Raspberry Pi
20. **HTTP**	Hypertext Transfer Protocol
21. **HTTPS**	Hypertext Transfer Protocol Secure
22. **SSH**	Secure Shell
23. **TCP**	Transmission Control Protocol
24. **UDP**	User Datagram Protocol
25. **OTX**	Open Threat Exchange

CHAPTER 1

INTRODUCTION

1.1 DESCRIPTION

Today, most homes are filled with connected devices — laptops, phones, smart TVs, and IoT gadgets — all sharing the same network. While this makes life easier, it also creates new risks, as home routers focus mainly on connectivity and not on security. This leaves many users exposed to cyber-attacks such as malware, phishing, and unauthorized access.

The **Smart Unified Threat Management System (SUTMS)** is developed to solve this problem by bringing enterprise-level protection into home networks. It is a lightweight, low-cost security framework that combines **NTOPng** for traffic monitoring, **Suricata** for intrusion detection, and **IPTables** for firewall protection — all running on a **Raspberry Pi**. The system also uses **IoC feeds** to automatically block known threats and provides a **simple web dashboard** for users to view network status or control rules without technical skills.

In short, SUTMS offers an affordable, automated, and easy-to-use solution that keeps home and small-office networks safe from modern cyber threats.

1.2 PROJECT IDENTIFICATION

With the rise of smart devices and remote work, home networks have become as critical as corporate ones — yet they lack the same level of security. Most households depend on basic routers that only provide limited firewall protection. These devices are not equipped to detect advanced threats such as phishing attacks, malware infections, or unauthorized access from the internet. As a result, hackers easily exploit unsecured home devices to steal data or use them as part of larger attacks.

While large organizations rely on expensive Unified Threat Management (UTM) systems to defend their networks, such solutions are not practical for home users because of their high cost and complex setup. Additionally, most security tools used at home work separately — firewalls, antivirus software, and routers do not share data or act in coordination, leaving gaps in protection.

This lack of an integrated, user-friendly, and cost-effective system creates a major security risk for personal and small office networks. Hence, there is a strong need for a **smart, unified solution** that provides real-time monitoring, automated threat detection, and easy management — which forms the foundation for the proposed **Smart Unified Threat Management System (SUTMS)**.

1.3 OBJECTIVES OF THE PROJECT

The main objective of the **Smart Unified Threat Management System (SUTMS)** is to provide strong and intelligent network security for home and small office users in a simple and affordable way. The project aims to bring together multiple layers of protection — such as **flow monitoring, intrusion detection, and dynamic firewall defense** — into one integrated system that can be easily managed by anyone.

SUTMS is designed to run efficiently on low-cost hardware like the **Raspberry Pi**, ensuring that users can achieve enterprise-level protection without needing expensive equipment or high technical skills. The system uses **NTOPng** to monitor real-time network traffic, **Suricata** to detect and block malicious packets, and **IPTables** to handle automatic rule updates and threat blocking.

It also connects to global **IoC (Indicators of Compromise)** feeds to stay updated with the latest cyber threats and offers a **web-based dashboard** for easy control and monitoring. Overall, the objective is to create a lightweight, automated, and user-friendly security framework that ensures safe internet access for every home and small network environment.

1.4 SCOPE OF THE PROJECT

The **Smart Unified Threat Management System (SUTMS)** is developed to strengthen the security of home and small office networks by combining multiple security features into one simple, lightweight system. Its scope covers the entire process of detecting, analyzing, and preventing cyber threats in real time. The system monitors all devices connected to the network, identifies unusual activities, and automatically blocks suspicious connections before they can cause harm.

SUTMS can be deployed on affordable hardware like a **Raspberry Pi**, making it accessible to both individuals and small organizations. It integrates **NTOPng**, **Suricata**, and **IPTables** to provide flow-based traffic monitoring, intrusion detection, and firewall protection. In addition, it connects to **IoC (Indicator of Compromise)** feeds from trusted sources, ensuring continuous updates against new and evolving threats.

The project also includes a **web-based dashboard** that allows users to easily view network status, manage firewall rules, and monitor system resources. Its design ensures that even non-technical users can operate it confidently. In the future, SUTMS can be extended to support AI-based detection, SSL/TLS inspection, and organization-specific customization, making it a scalable and sustainable cybersecurity solution.

1.5 EXISTING SYSTEM

In most home networks today, security mainly depends on the router's built-in firewall and an antivirus installed on individual devices. These systems are designed for basic protection and are limited to simple packet filtering or signature-based scanning. They can block known viruses but are not capable of analyzing complex or new types of attacks such as phishing, malware injection, or unauthorized remote access.

Traditional **Unified Threat Management (UTM)** solutions exist in corporate environments, where they combine multiple security tools like intrusion detection, firewalls, and content filters. However, these systems require high processing power, regular updates, and professional maintenance — making them unsuitable for home users. In addition, existing security tools for personal use often work independently without sharing data, leading to fragmented and less effective protection.

Due to these limitations, home networks remain highly vulnerable to cyber threats. There is a clear need for an **affordable, unified, and automated** security system that can monitor, detect, and respond to network threats in real time without requiring expert knowledge.

1.6 PROPOSED SYSTEM

The proposed Smart Unified Threat Management System (SUTMS) **aimed** to bridge the gap between enterprise-level security and home network protection. It **was** a lightweight, low-cost, and integrated solution built to deliver multi-layer defense using open-source tools and affordable hardware such as the Raspberry Pi.

SUTMS **brought** together three main modules: NTOPng, Suricata, and IPTables. NTOPng **continuously monitored** network traffic, **identified** active devices, and **detected** anomalies. Suricata **performed** deep packet inspection to detect intrusions and malicious activities, while IPTables **acted** as a dynamic firewall that **blocked** harmful IPs in real time. The system also **connected** to IoC (Indicator of Compromise) feeds using STIX/TAXII protocols, allowing it to automatically update threat lists and respond to new cyberattacks.

A simple web dashboard **made** the system user-friendly, enabling users to view alerts, monitor network status, and manage firewall rules easily. By combining automation, intelligence, and ease of use, the proposed SUTMS **provided** corporate-grade protection to home networks, ensuring safer connectivity for personal, professional, and IoT-based environments

1.7 ORGANIZATION OF THE PROJECT

Chapter 1 introduces the background, objectives, scope, and motivation behind developing SUTMS, along with the limitations of existing systems.

Chapter 2 reviews related studies and tools in network security, intrusion detection, and threat management, identifying the research gap addressed by this project.

Chapter 3 details the system design and methodology, including the architecture and core modules such as NTOPng, Suricata, IPTables, and the IoC Feed Server.

Chapter 4 covers implementation, testing setup, performance evaluation, and experimental results that validate the system's effectiveness.

Chapter 5 presents the conclusion and future enhancements, suggesting extensions like AI-based threat detection and SSL/TLS inspection.

Chapter 6 lists the references and citations used throughout the study.

CHAPTER 2

LITERATURE SURVEY

2.1 Intrusion Detection for IoT- enabled Smart Homes

2.1.1 Objective:

The purpose of this paper **was** to enhance the security of IoT-enabled smart homes by implementing intrusion detection systems capable of identifying abnormal network activities. The focus **was** on creating lightweight and intelligent frameworks that **could** operate efficiently in resource-limited environments, improving the detection accuracy of malicious behavior.

2.1.2 Comparison with IDS - SUTMS

Objective and Scope:

- **ID for IoT Smart Homes**: Focused on anomaly detection for IoT traffic only.
- **SUTMS**: Provides complete network protection using flow analysis, IDS, and firewall control.

Methods:

- **Kumar et al. (2021):** Implemented a lightweight IDS using machine learning algorithms for anomaly detection in IoT networks.
- **Li and Zhang (2020):** Proposed a deep learning-based IDS to detect zero-day attacks through automated traffic classification.
- **Rahman et al. (2022):** Designed an edge-based hybrid security model combining flow analysis and intrusion prevention mechanisms

Models:

- **ID for IoT Smart Homes**: Machine learning/deep learning-based detection.
- **SUTMS**: Provides complete network protection using flow analysis, IDS, and firewall control.

Key Takeaways:

- **ID for IoT Smart Homes**: Effective for research but less practical for homes.
- **SUTMS**: Optimized for real-world home and small office environments.

2.2 Phishing/Malicious URL Detection with Raspberry Pi

2.2.1 Objective:

The aim of this study **was** to detect phishing and malicious URLs using Raspberry Pi as a low-cost security platform. The focus **was** on real-time URL analysis and categorization, helping to prevent users from accessing unsafe web content. This approach **provided** an affordable, portable solution for home and small networks.

2.2.2 Comparison with URLD - SUTMS

Objective and Scope:

- **URL Detection**: Detects and blocks phishing or malicious URLs based on predefined blacklists.
- **SUTMS**: Integrates URL detection within a larger unified framework for network-wide security and traffic control.

Methods and Models:

- **URL Detection**: Uses Python-based scripts with URL reputation databases and machine learning classifiers.
- **SUTMS**: Uses IoC feeds (STIX/TAXII) to fetch and update threat indicators automatically.

Advantages:

- **URL Detection** Low cost, portable, and easy to deploy.
- **SUTMS**: Centralized detection with real-time updates and automation

Limitations:

- **URL Detection**: Limited scope to URL filtering; does not handle network-level threats.
- **SUTMS** Broader protection but slightly higher configuration effort.

Key Takeaways:

- **URL Detection:** Raspberry Pi can effectively handle lightweight URL analysis.
- **SUTMS**: SUTMS extends this by integrating URL and network-level protection under one system.

2.3 Traffic Trace Artifacts from Port Mirroring

2.3.1 Objective:

This study **focused** on collecting and analyzing network traffic using port mirroring to identify abnormal behaviors and potential attacks. The goal **was** to evaluate how network traffic artifacts **could** be used for real-time intrusion detection and analysis in small network environments.

2.3.2 Comparison with TTA – SUTMS

Objective and Scope:

- **TTA**: Focused on passive traffic monitoring for analysis.
- **SUTMS**: Combined passive monitoring (NTOPng) with active intrusion prevention (Suricata).

Methods and Models:

- **TTA**: Utilized Wireshark or Tcpdump for packet capture and offline analysis.
- **SUTMS**: Used NTOPng and Suricata for live flow capture and deep packet inspection.

Advantages:

- **TTA:** Simple to set up; provides raw traffic insights.
- **SUTMS**: Enabled real-time monitoring and automated blocking.

Limitations:

- **TTA**: Offline analysis; lacks automated response mechanisms.
- **SUTMS:** Added automation and visualization through dashboards.

Key Takeaways:

- **TTA:** Port mirroring aided in understanding traffic flow.
- **SUTMS**: SUTMS enhanced it with intelligent, continuous analysis and prevention.

2.4 Performance of Open-source IDS in High-speed Networks

2.4.1 Objective:

This paper **evaluated** the performance of open-source Intrusion Detection Systems (IDS) like Snort and Suricata in high-speed environments. The objective **was** to test their detection accuracy, processing speed, and packet loss rate under heavy network loads.

2.4.2 Comparison with OS_IDS – SUTMS

Objective and Scope:

- **IDS**: Focused on performance benchmarking of standalone IDS tools.
- **SUTMS**: Integrated Suricata with NTOPng and IoC feeds for smart home-scale deployment.

Methods and Models:

- **IDS**: Used network simulators or testbeds to measure throughput and alert accuracy.
- **SUTMS**: Used real home network traffic and Raspberry Pi-based setup.

Advantages:

- **IDS:** Demonstrated high detection accuracy and reliability.
- **SUTMS**: Maintained similar accuracy with reduced hardware cost and energy use.

Limitations:

- **IDS**: Required powerful hardware for real-time detection.
- **SUTMS:** Designed for lightweight and optimized operation.

Key Takeaways:

- **IDS:** Open-source IDS tools were powerful but resource-heavy.
- **SUTMS**: SUTMS adapted them for low-resource home environments without performance loss

2.5 Survey on Unified Threat Management (UTM) Systems for Home Networks

2.5.1 Objective:

This study **reviewed** existing UTM systems designed for enterprise and home networks. The goal **was** to understand their architecture, capabilities, and limitations, focusing on their applicability to small-scale deployments.

2.5.2 Comparison with UTM – SUTMS

Objective and Scope:

- **UTM**: Centralized network security combining IDS, firewall, and antivirus.
- **SUTMS**: Simplified and affordable UTM tailored for home and small office use.

Methods and Models:

- **UTM**: Used commercial solutions like Sophos, Fortinet, or pfSense.
- **SUTMS**: Used open-source integration of NTOPng, Suricata, and IPTables.

Advantages:

- **UTM:** Offered comprehensive protection and advanced analytics.
- **SUTMS**: Delivered similar protection using open-source and lightweight tools.

Limitations:

- **UTM**: High cost, complex setup, requires maintenance.
- **SUTMS:** Easy to deploy, minimal hardware requirements.

Key Takeaways:

- **UTM:** UTMs were effective but not home-friendly.
- **SUTMS**: SUTMS bridged the gap by bringing enterprise-grade defense to home networks.

2.6 Comprehensive Study on STIX and TAXII

2.6.1 Objective:

This study **explored** Structured Threat Information Expression (STIX) and Trusted Automated Exchange of Indicator Information (TAXII) — open standards for sharing cyber threat intelligence. The objective **was** to improve automated detection and response using global threat data.

2.6.2 Comparison with UTM – SUTMS

Objective and Scope:

- **UTM**: Used in large-scale enterprise SOCs for automated threat sharing.
- **SUTMS**: Integrated IoC feeds using STIX/TAXII for lightweight home use.

Methods and Models:

- **UTM**: Threat feeded fetched via TAXII servers for analysis
- **SUTMS** Custom IoC Feed Server aggregated and normalized multiple feeds.

Advantages:

- **UTM:** Real-time sharing of structured threat intelligence.
- **SUTMS**: Brought automation and adaptability to smaller networks.

Limitations:

- **UTM**: Required complex backend setup and data parsing.
- **SUTMS:** Simplified for Raspberry Pi with minimal dependencies.

Key Takeaways:

- **UTM**: STIX/TAXII enabled global threat data exchange.
- **SUTMS**: SUTMS effectively adapted them for personal and small-scale cybersecurity.

2.7 IoT Device Classification using Machine Learning

2.7.1 Objective:

The objective of this study **was** to accurately classify IoT devices within a network based on their traffic behavior using machine learning models. The paper **focused** on identifying device types automatically from network flow features, thereby **enabling** better network visibility, anomaly detection, and security policy enforcement.

2.7.2 Comparison with UTM – SUTMS

Objective and Scope:

- **IDC**: Focused on device identification and behavior profiling using ML models.
- **SUTMS**: Integrated device-aware detection as part of a broader threat management and flow analysis system.

Methods and Models:

- **IDC**: Utilized supervised algorithms such as Random Forest, SVM, and Decision Trees trained on packet-level and flow-level features.
- **SUTMS**: Used NTOPng for real-time protocol and flow analysis, enabling context-based IDS signature optimization.

Advantages:

- **IDC:** Provided high classification accuracy and enhances network visibility.
- **SUTMS**: Extended this visibility to enable adaptive IDS and firewall rule tuning.

Limitations:

- **IDC**: Required labeled datasets and retraining for new device types.
- **SUTMS:** Focused more on integrated security actions than deep device categorization.

Key Takeaways:

- **IDC**: Effective for understanding IoT behavior through ML classification.
- **SUTMS**: Incorporated flow-based awareness into unified security management for automated response and resource optimization.

2.8 Automatic Device Classification using Arduino Platform

2.8.1 Objective:

This paper **aimed** to design a low-cost, Arduino-based system for automatically identifying and classifying connected IoT devices. The goal **was** to use embedded sensors and network activity patterns to distinguish devices and monitor their operational status in real time.

2.8.2 Comparison with ADC – SUTMS

Objective and Scope:

- **ADC**: Focused on device-level identification using Arduino microcontrollers.
- **SUTMS**: Performed network-level detection and traffic analysis across all connected devices.

Methods and Models:

- **ADC**: Employed Arduino boards with Wi-Fi modules and lightweight scripts to detect device characteristics like MAC addresses, signal strength, and usage patterns.
- **SUTMS**: Used NTOPng and Suricata to classify and monitor devices through protocol fingerprints and flow patterns.

Advantages:

- **ADC:** Low cost, energy-efficient, and easily deployable in small IoT environments.
- **SUTMS**: Scalable, automated, and supports deeper packet-level analysis.

Limitations:

- **ADC**: Limited processing capability; unsuitable for large or encrypted traffic analysis.
- **SUTMS:** Required higher configuration effort but offers complete network security integration.

Key Takeaways:

- **ADC**: Suitable for prototype-level IoT monitoring and device identification.
- **SUTMS**: Provided a comprehensive, software-defined alternative capable of automated detection and prevention across networks.

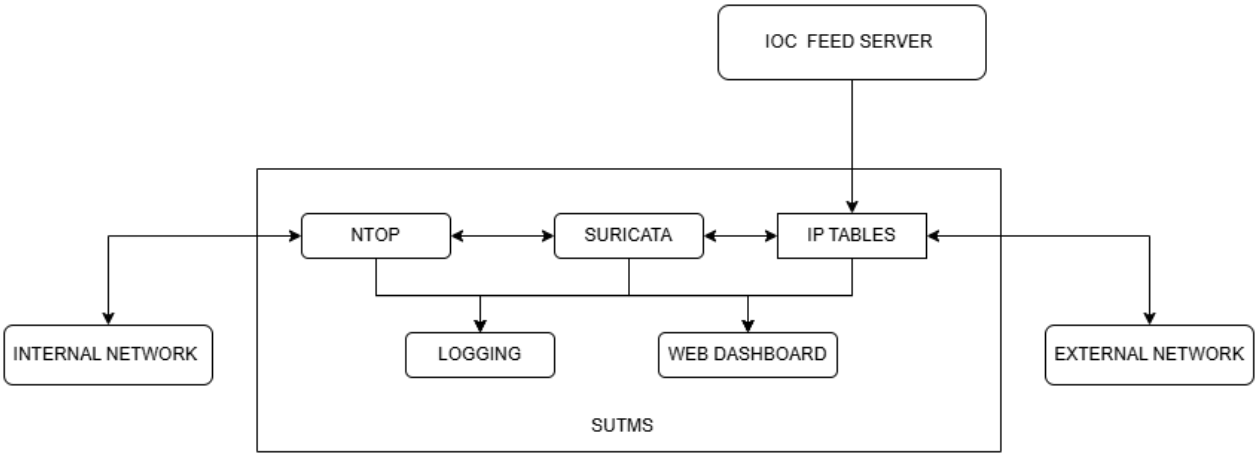
Paper	Problem/Objective	Method/Module	Key Considerations	Limitations
Intrusion Detection for IoT-enabled Smart Homes	Smart homes face frequent cyber threats due to weak device security.	ML-based intrusion detection focusing on anomaly detection.	Must be lightweight, accurate, and adaptive.	Simulation-based, dataset-dependent, may not generalize to real IoT attacks.
Phishing/Malicious URL Detection with Raspberry Pi	Home networks often fail to detect phishing URLs; need cost-effective solution.	NETBITS on Raspberry Pi using URL feature analysis & classification.	Low-cost hardware, real-time monitoring, feasible for small networks.	Focuses only on URL-based threats; Raspberry Pi has limited processing power.
Traffic Trace Artifacts from Port Mirroring	Port mirroring (SPAN) used for monitoring may distort traffic traces.	Studied artifacts like packet loss & timing distortion in switches/routers.	IDS accuracy depends on fidelity of captured traces.	Only studies port mirroring, not other capture methods; based on older hardware.
Survey on Unified Threat Management (UTM) Systems for Home Networks	Home/SoHo networks need enterprise-level security, but UTMs are built mainly for enterprises.	Surveyed UTM approaches (firewall, IDS/IPS, antivirus, anti-bot) and deployment issues.	Constraints on cost, usability, and integration with threat intelligence (STIX/TAXII).	Purely survey-based, no prototype or deployment solution.
Comprehensive Study on STIX and TAXII	Threat intelligence sharing lacks standardization.	Analyzed STIX (Structured Threat Information Expression) and TAXII (Trusted Automated Exchange of Indicator Information).	Machine-readable intel sharing, interoperability, importance for adaptive firewalls/IDS.	Machine-readable intel sharing, interoperability, importance for adaptive firewalls/IDS.
IoT Device Classification using Machine Learning	Classify IoT devices to detect anomalies or threats.	ML-based classification models with feature extraction from device behavior.	Accurate classification prevents misuse; requires good training and preprocessing.	Relies on labeled datasets; models may drift with new devices or attack types.
Automatic Device Classification using Arduino Platform	Identify/classify IoT devices for stronger network security.	Device identification algorithms using Arduino hardware. Real-time classification based on communication patterns.	Efficient resource use, adaptability across device types.	Limited processing capacity, risk of misclassification in noisy networks.
Performance of Open-source IDS in High-speed Networks	IDS like Snort, Suricata, Bro face bottlenecks in Gbps traffic environments.	Benchmarked IDS performance (throughput, packet loss, detection accuracy).	Requires multi-threading, optimized rules, hardware acceleration.	Hardware-dependent results; focus on performance, not detection improvements.

CHAPTER 3

PROJECT DESIGN

3.1 SYSTEM DESIGN OVERVIEW

3.1.1 Architecture of the Proposed System



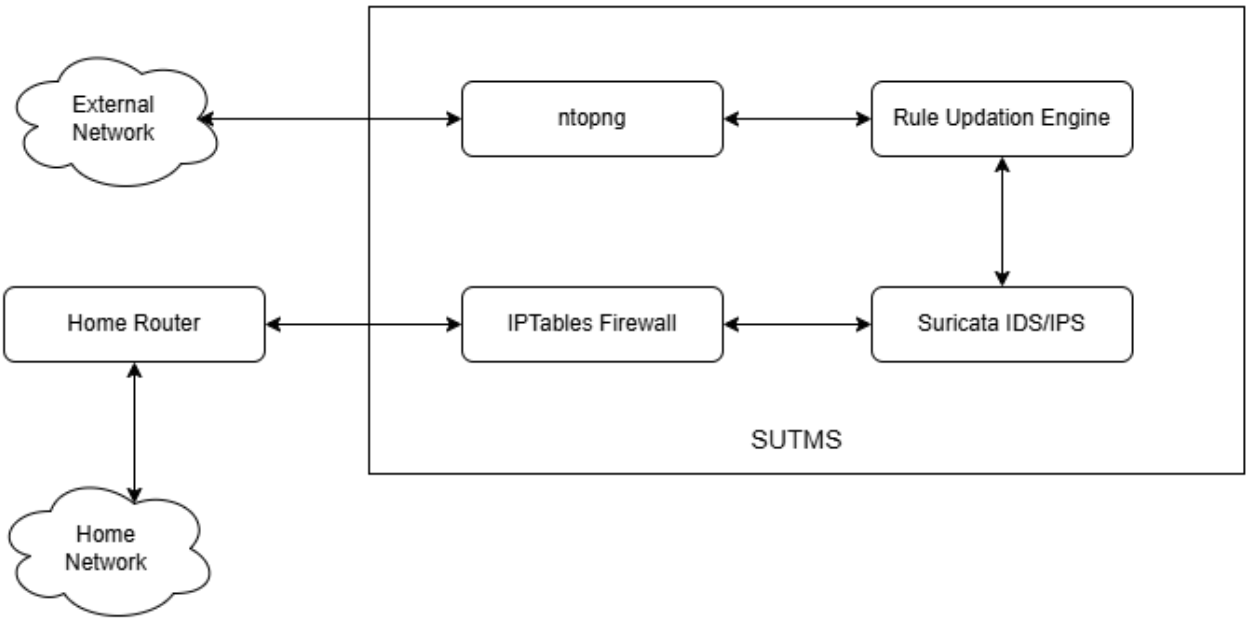
The proposed Smart Unified Threat Management System (SUTMS) architecture **was designed** as a multi-layered security framework capable of offering real-time protection, detection, and analysis of network threats within small home or office environments. The architecture **was developed** primarily to operate on a Raspberry Pi 4 single-board computer running Ubuntu 22.04 LTS, offering an affordable yet efficient security solution. The overall system **comprised** multiple interacting modules, including the Flow Detection and Protocol Extraction Engine, Intrusion Detection and Rule Optimization Engine, Firewall with Dynamic IoC Integration, Routing and DHCP Services, and Logging and Monitoring Engine. These components **were supported** by a Flask-based web dashboard and Webmin interface for easy management and visualization.

The core modules—flow detection, intrusion detection, and firewall—**worked** in synergy to identify abnormal traffic, detect potential intrusions, and dynamically block malicious connections. The architecture **followed** a modular design, allowing each engine to function independently while maintaining interoperability through internal communication channels. The Flow Detection module **identified** network traffic patterns and **extracted** protocol information, which **was used** by the Intrusion Detection System (IDS) to enable

only relevant signatures, **optimizing** system performance and **reducing** CPU and memory utilization. The Firewall engine, using IPtables integrated with Structured Threat Information eXpression (STIX) and Trusted Automated eXchange of Indicator Information (TAXII) feeds, **provided** dynamic and intelligent blocking of malicious IP addresses.

The addition of a Flask Web UI in this architecture **allowed** users to monitor live traffic statistics, manage rules, view threat alerts, and configure the system without requiring command-line expertise. This UI **communicated** with the backend engines through RESTful APIs, ensuring seamless integration and real-time updates. The architecture, therefore, **provided** a comprehensive, low-cost, and scalable threat management platform suitable for protecting modern home networks with an intuitive user experience.

3.1.2 Functional Workflow



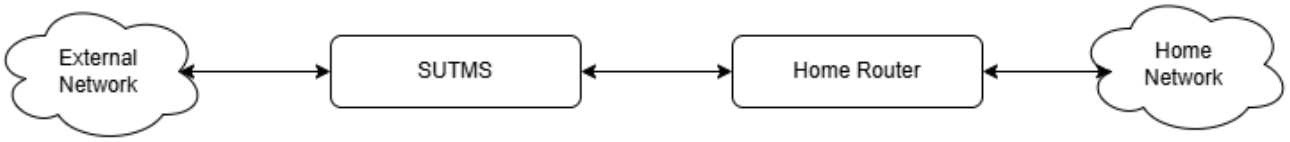
The functional workflow of the proposed SUTMS began with traffic interception at the network interface, where both ingress (incoming) and egress (outgoing) traffic were captured for inspection. The Flow Detection Engine first analyzed packet streams using NTOP, classifying them by protocols, source-destination pairs, and behavioral attributes such as frequency and duration. This flow-level data was then relayed to the Intrusion Detection Engine (based on Suricata), which used this contextual information to enable

only relevant rule sets and perform deep packet inspection.

The IDS then detected or blocked malicious packets depending on the configured mode—detection or prevention. Simultaneously, the Firewall Engine operated at Layer 3 and Layer 4 to enforce IP-based and port-based access control. It also dynamically updated its rules by ingesting external Indicators of Compromise (IoCs) through STIX/TAXII feeds, thereby providing automated protection against evolving cyber threats such as botnet communications and command-and-control attacks.

The Flask Dashboard acted as the user interface layer for this workflow, providing a real-time visual representation of network health, attack attempts, bandwidth usage, and system resource utilization. It allowed administrators to control firewall rules, update IDS configurations, and manage routing parameters from a centralized web-based interface. Thus, the functional workflow of SUTMS embodied an automated feedback loop where flow detection informed intrusion detection, intrusion detection influenced firewall responses, and the entire process was observable through the integrated web UI, resulting in an efficient and intelligent home network protection mechanism.

3.1.3 Deployment Topology



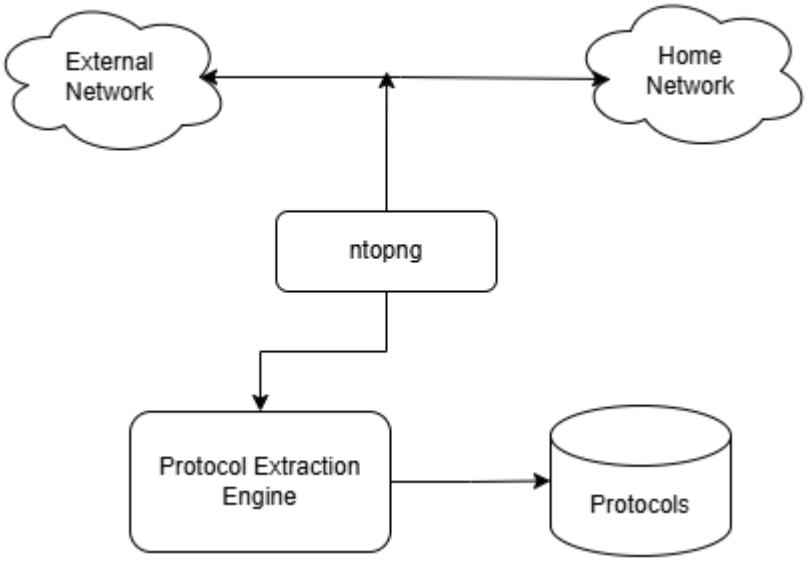
The deployment topology of the proposed system was designed to ensure flexibility and ease of integration in varied home network environments. SUTMS supported both inline and out-of-band deployment modes. In the inline configuration, the device was placed between the Internet Service Provider (ISP) modem and the existing Wi-Fi access point, ensuring that all network traffic passed through the SUTMS device for inspection and filtering. This mode enabled full intrusion prevention and dynamic blocking capabilities. The device could also function as an independent Wi-Fi access point, eliminating the need for an external router in smaller networks.

The out-of-band deployment mode, on the other hand, used a switch mirror port or Test Access Port (TAP) to analyze network traffic passively, offering monitoring and detection without interfering with the network flow. This mode was useful for diagnostic or

research purposes. The deployment topology also allowed for integration with IoT devices, smart appliances, and mobile clients. The newly added Flask Web UI component enhanced the deployment by providing remote accessibility to configuration settings, system health visualization, and log monitoring over HTTPS. Webmin integration further supported advanced system management, such as user account handling, network configurations, and process control. Whether operating inline or passively, the system ensured secure communication between modules and the dashboard through local APIs and encrypted channels, maintaining data confidentiality and system integrity across deployment models.

3.2 MODULE DESCRIPTION

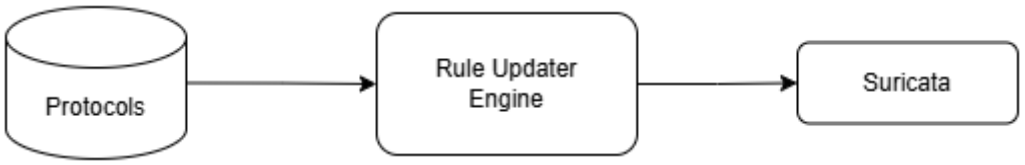
3.2.1 Flow Detection and Protocol Extraction Module



The Flow Detection and Protocol Extraction Module served as the foundation of the SUTMS architecture, responsible for identifying, classifying, and analyzing network traffic flows. Built using NTOP, it captured data packets at multiple network interfaces and organized them into flows based on attributes such as source and destination IP addresses, port numbers, and communication duration. This module played a crucial role in anomaly detection by comparing real-time traffic against established behavioral baselines. When deviations occurred—such as unexpected protocol usage or unusual data transfer volumes—the module generated alerts that were forwarded to the Intrusion Detection System. Furthermore, it extracted the list of active protocols within the home network and passed this information to the IDS for signature optimization.

By enabling only the required protocol-specific rules, the system reduced resource consumption while maintaining high accuracy. The module also provided flow-based visualization to the Flask Web Dashboard, allowing users to view live traffic statistics, protocol distribution, and anomaly summaries in an intuitive graphical format. This integration transformed the backend flow analytics into actionable intelligence, empowering users to understand and secure their network traffic with greater transparency.

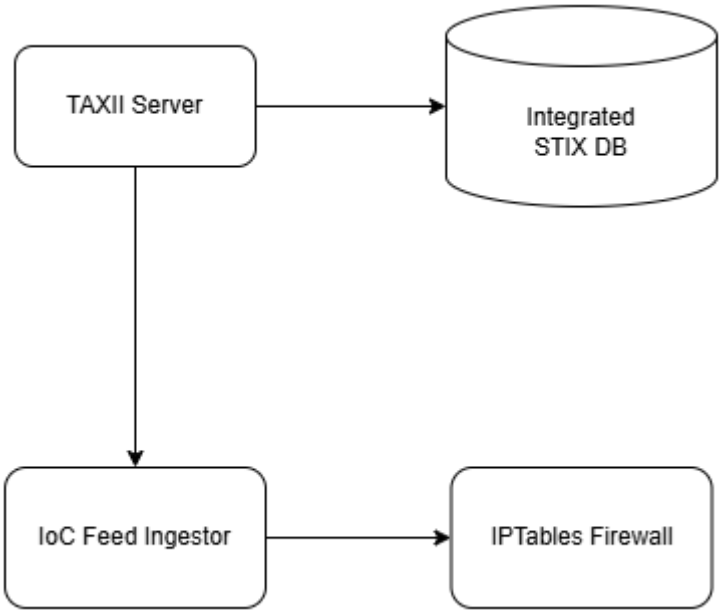
3.2.2 Intrusion Detection and Rule Optimization Module



The Intrusion Detection and Rule Optimization Module **formed** the analytical core of SUTMS, **leveraging** the Suricata IDS engine to detect known attacks and suspicious behaviors in network traffic. Suricata's multi-threaded architecture **allowed** efficient real-time inspection of packets while **maintaining** high throughput on limited hardware resources. The module **used** the optimized signature set dynamically generated from the Flow Detection Engine, **ensuring** that only relevant protocol-specific rules **were enabled**. This optimization significantly **reduced** CPU and memory utilization while **increasing** detection precision. The IDS **operated** in both detection and prevention modes: in detection mode, it **alerted** administrators of potential threats, while in prevention mode, it **actively blocked** malicious packets.

The system **continuously updated** its signature database from open-source and vendor repositories, and the optimization script **automatically disabled** irrelevant or unused signatures. All detected threats **were logged** and **visualized** in the Flask dashboard, where alerts **were categorized** by severity level. The web UI also **provided** an interface for administrators to view, enable, or disable specific signatures without directly modifying system files, **enhancing** usability for non-technical users. This module thereby **combined** adaptive learning with robust detection capabilities, **forming** the intelligence layer of the SUTMS architecture.

3.2.3 Firewall and Dynamic IoC Integration Module



The Firewall and Dynamic IoC Integration Module safeguarded the network perimeter using a combination of IPtables and dynamic threat intelligence feeds. It enforced Layer 3 and Layer 4 filtering by examining IP addresses, ports, and packet states. The firewall applied both static and dynamic rule sets—the static rules were configured manually by administrators for trusted traffic, while dynamic rules were automatically updated by ingesting Indicators of Compromise (IoCs) through STIX/TAXII feeds. This allowed the firewall to proactively block malicious IPs associated with botnets, phishing servers, or command-and-control infrastructures. The module automatically formatted IoC feeds into IPtables-compatible rules through background scripts, ensuring timely updates and minimal human intervention.

The firewall logs were integrated into the Flask Dashboard, where administrators could view blocked connections, create new rules, and modify existing ones through a graphical interface. Webmin complemented this by providing a backend configuration panel for advanced users to manage IPtables directly. The module thus extended beyond traditional firewalls by incorporating live cyber threat intelligence, offering both static defense and adaptive, real-time response capabilities against evolving threats.

3.2.4 Routing and DHCP Service Module

The Routing and DHCP Service Module provided the networking backbone for SUTMS by managing data flow between devices and assigning IP addresses dynamically. It supported static routing to define network paths and included DHCP functionality to

allocate IP addresses within the local subnet automatically. This module ensured smooth connectivity for wired and wireless devices in home environments. In scenarios where an existing router lacked routing or DHCP capabilities, SUTMS could assume full control of network management, operating as a standalone gateway. Although it did not support complex enterprise routing protocols like OSPF or BGP, its static routing capability efficiently handled small-scale network topologies.

The routing process was tightly integrated with the security modules—every packet traversing the routing engine underwent flow and intrusion inspection before being forwarded. Configuration and monitoring of routing tables were facilitated through the Flask Web Dashboard and Webmin interface, giving administrators real-time visibility and control over network connectivity without requiring command-line operations.

3.2.5 Logging and Monitoring Engine

The Logging and Monitoring Engine functioned as the centralized observability component of the SUTMS framework. It collected logs from all major modules—Flow Detection, IDS, Firewall, and Routing—and stored them in structured formats for analysis. Logs could be stored locally within the Raspberry Pi or forwarded to external servers via Syslog for long-term archival and correlation. Integration with third-party platforms such as ELK Stack, Splunk, or AlienVault OSSIM enhanced analytical capabilities and supported security event correlation. The Flask Dashboard visualized log summaries and key security metrics, including attack frequency, source IP trends, and system resource usage.

The engine employed automated cleanup mechanisms to manage disk space efficiently, ensuring that the lightweight hardware remained stable over prolonged operation. By providing both high-level insights and granular log details, the module enabled administrators to perform threat forensics, trend analysis, and incident response effectively, bridging the gap between raw data and actionable security intelligence.

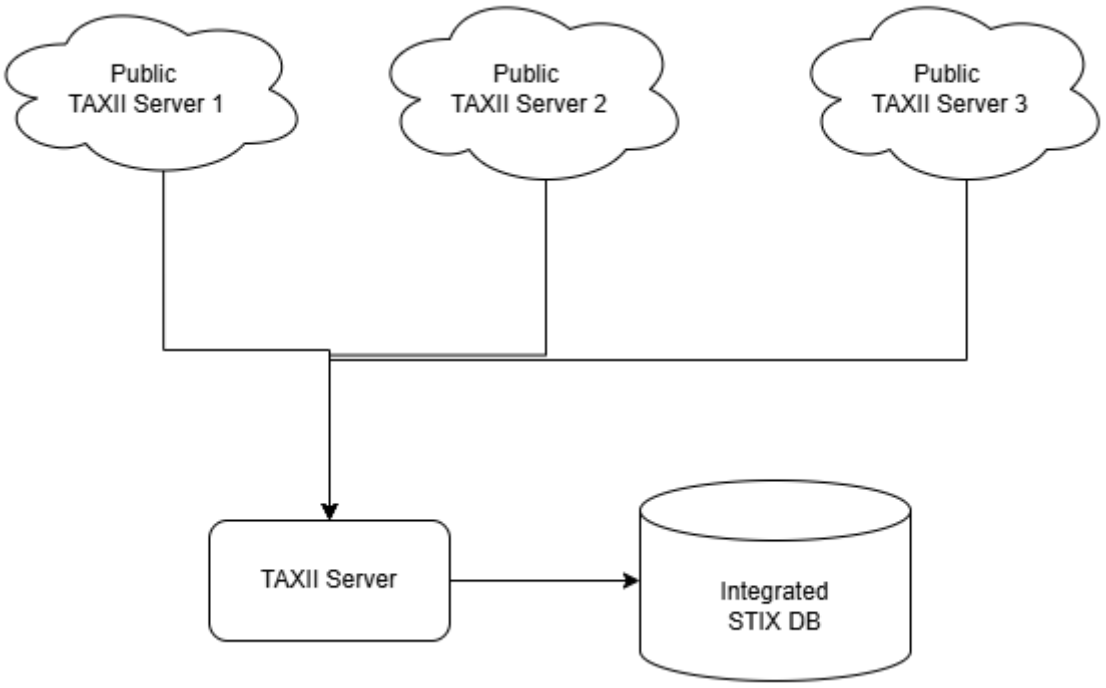
3.2.6 Flask Dashboard and Webmin Integration

The Flask Dashboard was a newly developed web interface that provided a centralized control panel for the SUTMS ecosystem. Designed using the Flask framework, it enabled users to interact with backend modules through RESTful APIs, presenting real-time information in a clean and intuitive format. The dashboard included views for live traffic

monitoring, IDS alerts, firewall activity, and system health statistics such as CPU and memory usage. Users could enable or disable IDS rules, modify firewall configurations, initiate updates, and view IoC ingestion status directly through the browser.

The responsive interface ensured accessibility across desktops and mobile devices. Complementing Flask, Webmin was integrated for advanced system-level configuration, allowing authenticated users to manage services, file systems, and network parameters securely. Together, these interfaces simplified administration, provided visibility into network operations, and eliminated the need for complex command-line interactions, making the system suitable even for non-technical home users while maintaining professional-grade control for experts.

3.2.7 External TAXII Server and Threat Intelligence Aggregation



The External TAXII Server and Threat Intelligence Aggregation module expanded the defense capabilities of SUTMS by connecting it to global cyber threat intelligence networks. Through the TAXII protocol, SUTMS automatically retrieved structured IoC feeds containing known malicious IP addresses, domain names, and behavioral indicators. These feeds, provided by sources such as MITRE and Anomali, were parsed and filtered by the system before being translated into enforceable firewall rules. The aggregation process eliminated redundancy and ensured that only unique, high-confidence indicators

were retained. By maintaining continuous synchronization with external intelligence sources, SUTMS evolved dynamically to counter new attack vectors and emerging threats.

3.3 SUMMARY

In conclusion, the proposed Smart Unified Threat Management System integrated multiple network security layers—flow analysis, intrusion detection, firewall enforcement, routing, and monitoring—into a single cost-effective platform. The use of open-source technologies like NTOP, Suricata, IPtables, and Flask ensured scalability, transparency, and flexibility. The addition of the Flask-based web dashboard introduced a modern and user-friendly management interface that enhanced accessibility and usability without compromising on functionality. Together, these modules formed an intelligent, adaptive, and lightweight cybersecurity solution tailored for home and small office networks, capable of providing enterprise-grade protection through efficient design and seamless integration.

CHAPTER 4

SYSTEM SPECIFICATION, IMPLEMENTATION AND RESULT

4.1 PROJECT SPECIFICATON

4.1.1 Software Requirements

- **Operating System:** Raspberry Pi OS (64-bit, Debian-based)
- **Programming Language:** Python 3.x
- **Core Tools and Services:**
    - **ntopng –** Flow detection and protocol analysis
    - **Suricata 6.0.4 –** Intrusion detection and prevention
    - **IPTables –** Firewall and packet filtering
    - **opentaxii –** Custom TAXII server for IoC aggregation
    - **Flask –** Lightweight dashboard for monitoring
    - **Webmin –** Web-based system administration tool
- **Supporting Packages:**
    - rsyslog, dhcpd, net-tools, curl, iptables-persistent
    - Python Libraries: requests, regex, flask, psutil
- **Additional Utilities:**
    - Anomali STIX/TAXII feeds for IoC updates
    - systemd services for module automation

4.1.2 Hardware Requirements

- **Device:** Raspberry Pi 5 Model B
- **Processor:** Quad-Core ARM Cortex-A76 @ 2.4 GHz
- **Memory (RAM):** 8 GB LPDDR5
- **Storage:** 128 GB Class 10 microSD card
- **Network Interfaces:** Dual Gigabit Ethernet (USB adapter for secondary port) and Wi-Fi 6
- **Power Supply:** 27W USB-C adapter
- **Peripherals:** HDMI monitor, keyboard, mouse (for initial setup)
- **External Components:** ISP modem and home router for inline deployment

4.1.3 Network Configuration Parameters

- **Deployment Mode:** Inline mode between the ISP modem and the home router
- **LAN Interface (eth0):** 192.168.1.1 – acts as the gateway for local devices
- **WAN Interface (eth1):** Configured for DHCP to obtain IP automatically from the ISP modem
- **DHCP Server Range:** 192.168.1.10 – 192.168.1.100 for connected clients
- **Default Gateway:** Forwarded to ISP modem's IP address
- **DNS Servers:** 8.8.8.8 (Google) and 1.1.1.1 (Cloudflare)
- **IP Forwarding:** Enabled to allow routing between interfaces
- **Firewall Rules:** Managed using IPTables; integrates dynamic IoC updates from the TAXII server
- **Access Ports:**
    - Flask Dashboard – Port 5000
    - Webmin – Port 10000
    - SSH – Port 22
- **Logging Path:** /var/log/sutms/ with automatic log rotation enabled
- **Network Role:** Acts as a transparent gateway, inspecting and filtering all inbound and outbound packets for the home network

4.2 PROJECT IMPLEMENTATION

4.2.1 Environment Setup and Dependency Installation

The implementation of SUTMS began with the configuration of the Raspberry Pi 5 environment. A fresh installation of Raspberry Pi OS (64-bit, Debian-based) was performed using the Raspberry Pi Imager tool. After booting, the operating system was updated using apt-get update and apt-get upgrade to ensure all base packages were current. The system was then configured for IP forwarding by editing the /etc/sysctl.conf file and enabling packet routing between interfaces. The required dependencies were installed next, including essential network tools such as net-tools, iptables, curl, tcpdump, and dnsutils.

The Python 3 environment was configured with additional libraries like flask,

requests, and psutil to support custom automation and dashboard functionalities.

4.2.2 Configuration of Core Modules (ntopng, Suricata, IPTables)

The core functionality of SUTMS revolved around three primary modules: ntopng, Suricata, and IPTables. These components collectively formed the detection, analysis, and prevention backbone of the system.

ntopng was configured as the network traffic analysis engine responsible for monitoring active flows and identifying protocols in use. The configuration file /etc/ntopng/ntopng.conf was updated to bind the LAN interface (eth0) and enable data export to the custom Python script for protocol extraction.

Suricata, the Intrusion Detection and Prevention System (IDPS), was installed and configured to operate in inline mode using NFQUEUE. The configuration file /etc/suricata/suricata.yaml was modified to include the appropriate network interfaces and logging preferences. Rule sets were downloaded from the Emerging Threats Open repository, and custom rules were added to detect network anomalies specific to the test environment.

IPTables was employed as the system's firewall engine to enforce packet filtering. Custom chains were defined for INPUT, OUTPUT, and FORWARD rules. The NFQUEUE target was used to route packets through Suricata for inspection. Additionally, IoC-based blocking rules were automatically appended based on TAXII server data feeds, ensuring real-time adaptation of the firewall to new threats.

4.2.3 Integration of Custom Rule Optimization Script

One of the major enhancements introduced in this project was the integration of a custom Python-based rule optimization script. The script interfaced directly with ntopng's REST API to extract details of active network protocols and application types in real time.

4.2.3.1 Protocol Extraction Script

The Protocol extraction script worked with ntopng to extract the protocols that were being used by the traffic flow from and to the home network and external network through the SUTMS deployment. The extracted protocols from ntopng got saved under the file /var/lib/ntopng/protocols.txt.

```
#!/usr/bin/env python3
"""
Fetch ntopng L7 protocol counters and maintain
a rolling 60-minute record of all observed protocols.
"""

import requests
import json
import logging
from datetime import datetime, timedelta
from pathlib import Path
from suricata_rule_sync import suricata_rule_sync

# ------------------ CONFIG ------------------
NTOP_HOST = "http://127.0.0.1:3000"
NTOP_USER = "admin"
NTOP_PASS = "ntopng"
IFID = 2
HISTORY_FILE = Path("/var/lib/ntopng/protocol_history.json")
OUT_FILE_TXT = Path("/var/lib/ntopng/protocols.txt")
OUT_FILE_JSON = Path("/var/lib/ntopng/protocols_full.json")
LOG_FILE = Path("/var/log/ntop_protocol_export.log")
WINDOW_MINUTES = 60
# --------------------------------------------

logging.basicConfig(
    filename=LOG_FILE,
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
)

def fetch_protocols():
    """Fetch current L7 protocols from ntopng REST API."""
    url = f"{NTOP_HOST}/lua/rest/v2/get/flow/l7/counters.lua?ifid={IFID}"
    try:
        resp = requests.get(url, auth=(NTOP_USER, NTOP_PASS), timeout=10)
        resp.raise_for_status()
        data = resp.json()
        if data.get("rc") != 0 or "rsp" not in data:
            logging.warning("Unexpected ntopng response: %s", data)
            return []
        return [
            {"name": p["name"], "count": p["count"]}
            for p in data["rsp"]
```

```
            if p["name"].lower() not in ("unknown", "ntop")
        ]
    except Exception as e:
        logging.error("Fetch failed: %s", e)
        return []


def load_history():
    """Load previous 60-minute protocol history."""
    if HISTORY_FILE.exists():
        try:
            with open(HISTORY_FILE) as f:
                return json.load(f)
        except Exception:
            return []
    return []


def save_history(history):
    """Persist updated protocol history."""
    HISTORY_FILE.parent.mkdir(parents=True, exist_ok=True)
    with open(HISTORY_FILE, "w") as f:
        json.dump(history, f, indent=2)


def prune_history(history):
    """Remove records older than 60 minutes."""
    cutoff = datetime.utcnow() - timedelta(minutes=WINDOW_MINUTES)
    return [h for h in history if datetime.fromisoformat(h["timestamp"]) > cutoff]


def aggregate_protocols(history):
    """Combine protocol counts across the 60-minute window."""
    agg = {}
    for record in history:
        for p in record["protocols"]:
            name = p["name"]
            agg[name] = agg.get(name, 0) + p["count"]
    return sorted(
        [{"name": k, "count": v} for k, v in agg.items()],
        key=lambda x: (-x["count"], x["name"])
    )
```

```
def save_outputs(protocols):
    """Write final TXT and JSON outputs."""
    # Save unique protocol names for Suricata
    with open(OUT_FILE_TXT, "w") as f:
        for p in protocols:
            f.write(p["name"] + "\n")

    # Save aggregated JSON with timestamp
    snapshot = {
        "timestamp": datetime.utcnow().isoformat(),
        "window_minutes": WINDOW_MINUTES,
        "protocols": protocols
    }
    with open(OUT_FILE_JSON, "w") as f:
        json.dump(snapshot, f, indent=2)

    logging.info(
        "Saved %d protocols covering last %d min",
        len(protocols), WINDOW_MINUTES
    )


if __name__ == "__main__":
    now = datetime.utcnow()
    current = fetch_protocols()
    history = load_history()
    history.append({"timestamp": now.isoformat(), "protocols": current})
    history = prune_history(history)
    save_history(history)
    aggregated = aggregate_protocols(history)
    save_outputs(aggregated)

    suricata_rule_sync()
```

4.2.3.1 Rule Updater Script

The rule updater script fetched the protocols that were being used by the traffic flow from the file the protocol extraction script created (i.e. /var/lib/ntopng/protocols.txt) and updated Suricata's rule file to exclude the protocols that were not being used, which let Suricata skip a significant number of rules while validating a package.

```
#!/usr/bin/env python3

import os
import subprocess
import logging
import hashlib
from pathlib import Path

# ---------------- CONFIG ----------------
PROTO_FILE = Path("/var/lib/ntopng/protocols.txt")
SURICATA_RULE_DIR = Path("/etc/suricata/rules")
DISABLE_FILE = Path("/etc/suricata/disable.conf")
WHITELIST_FILE = Path("/etc/suricata/rule_whitelist.txt")
LOG_FILE = Path("/var/log/suricata_rule_sync.log")
SURICATA_SERVICE = "suricata"

# Map ntopng protocols → Suricata rule categories
PROTO_TO_RULE = {
    "HTTP": "http",
    "HTTPS": "tls",
    "TLS": "tls",
    "DNS": "dns",
    "SSH": "ssh",
    "FTP": "ftp",
    "SMTP": "smtp",
    "POP3": "pop3",
    "IMAP": "imap",
    "MDNS": "mdns",
    "SMB": "smb",
    "NTP": "ntp",
    "DHCP": "dhcp",
    "ICMPV6": "icmp",
    "ICMP": "icmp",
    "NETBIOS": "netbios",
    "MICROSOFT365": "http",
    "MQTT": "mqtt",
}
# ----------------------------------------

logging.basicConfig(
    filename=LOG_FILE,
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s",
```

```
)

def get_active_protocols():
    if not PROTO_FILE.exists():
        logging.warning("Protocol file not found: %s", PROTO_FILE)
        return []
    with open(PROTO_FILE) as f:
        prots = [line.strip() for line in f if line.strip()]
    logging.info("Active protocols read: %s", prots)
    return prots


def get_all_rule_categories(rule_dir=SURICATA_RULE_DIR):
    categories = set()
    if not rule_dir.exists():
        logging.error("Suricata rule directory not found: %s", rule_dir)
        return categories

    for rfile in rule_dir.glob("*.rules"):
        stem = rfile.stem
        if "-" in stem:
            candidate = stem.split("-")[0].lower()
        else:
            candidate = stem.lower()
        categories.add(candidate)
    logging.info("Discovered rule categories: %s", sorted(categories))
    return categories


def load_whitelist():
    """Load whitelisted rule categories (always enabled)."""
    if not WHITELIST_FILE.exists():
        logging.info("No whitelist found (%s). Continuing without it.", WHITELIST_FILE)
        return set()
    with open(WHITELIST_FILE) as f:
        wl = {line.strip().lower() for line in f if line.strip() and not line.startswith("#")}
    logging.info("Loaded whitelist: %s", sorted(wl))
    return wl


def map_protocols_to_categories(active_protocols, all_categories):
    active_cats = set()
    for p in active_protocols:
        pu = p.upper()
```

```
        if pu in PROTO_TO_RULE:
            active_cats.add(PROTO_TO_RULE[pu].lower())
            continue

        pl = p.lower()
        if pl in all_categories:
            active_cats.add(pl)
            continue

        for cat in all_categories:
            if cat in pl or pl in cat:
                active_cats.add(cat)
                break
    logging.info("Mapped active protocols -> categories: %s", sorted(active_cats))
    return active_cats


def build_disable_content(all_categories, enabled_categories, whitelist):
    """Generate disable.conf lines for categories NOT enabled or whitelisted."""
    protected = enabled_categories | whitelist
    disabled = sorted(all_categories - protected)
    lines = [f"re:{cat}\n" for cat in disabled]
    return "".join(lines)


def file_checksum(path):
    if not path.exists():
        return None
    h = hashlib.sha256()
    with open(path, "rb") as f:
        while chunk := f.read(8192):
            h.update(chunk)
    return h.hexdigest()


def write_if_changed(path, content):
    tmp = Path(str(path) + ".tmp")
    tmp.write_text(content)
    old_sum = file_checksum(path)
    new_sum = hashlib.sha256(content.encode()).hexdigest()
    if old_sum == new_sum:
        logging.info("No change to %s — skipping write.", path)
        tmp.unlink(missing_ok=True)
        return False
```

```
    tmp.replace(path)
    logging.info("Updated %s (checksum changed).", path)
    return True


def reload_suricata():
    try:
        subprocess.run(["systemctl", "reload", SURICATA_SERVICE], check=True)
        logging.info("Suricata reloaded successfully.")
    except subprocess.CalledProcessError as e:
        logging.error("Failed to reload Suricata: %s", e)


def suricata_rule_sync():
    active_protocols = get_active_protocols()
    all_categories = get_all_rule_categories()
    whitelist = load_whitelist()

    if not active_protocols:
        logging.warning("No active protocols found; skipping update.")
        return
    if not all_categories:
        logging.error("No rule categories found; aborting.")
        return

    enabled = map_protocols_to_categories(active_protocols, all_categories)
    content = build_disable_content(all_categories, enabled, whitelist)

    if write_if_changed(DISABLE_FILE, content):
        reload_suricata()
    else:
        logging.info("disable.conf unchanged; Suricata not reloaded.")


if __name__ == "__main__":
    suricata_rule_sync()
```

This approach significantly reduced the resource consumption of Suricata by preventing unnecessary rule evaluations. For example, if ntopng detected that no SMTP traffic was present in the network, the script automatically disabled mail-related detection rules in Suricata. The script was scheduled as a **cron job** to execute periodically and reconfigure Suricata without manual intervention.

This dynamic rule management not only optimized CPU and memory usage but also ensured faster packet inspection and reduced false positives, making the system more adaptive and lightweight for continuous home network deployment.

4.2.4 Flask Dashboard and Webmin Deployment

A Flask-based web dashboard was developed to provide a centralized monitoring interface for SUTMS. The dashboard displayed system status, active module states, current network throughput, and key log statistics. Flask endpoints were created to communicate with the backend components, allowing users to view logs, control services, and trigger rule updates through a simple, minimalistic web interface.

The dashboard ran locally on port 5000 and was accessible from any device within the local network. Lightweight design principles were followed to ensure minimal load on the Raspberry Pi's limited resources.

Additionally, Webmin was deployed as a comprehensive web-based administration tool. It provided advanced system control features, including process management, file editing, and firewall configuration through a secure interface on port 10000. Together, Flask and Webmin allowed both low-level and high-level control over SUTMS, enabling convenient system management and real-time visibility.

4.2.5 TAXII Server Configuration and IoC Feed Integration

For threat intelligence integration, a custom TAXII server was deployed using the OpenTAXII framework on an external host. This server aggregated Indicator of Compromise (IoC) data from multiple open sources such as Anomali, AlienVault OTX, and MITRE ATT&CK feeds. The TAXII server was configured to publish STIX-formatted data, which was then fetched by SUTMS for dynamic rule updates.

4.2.5.1 Local IoC Feed Integration

The Raspberry Pi instance of SUTMS periodically queried the TAXII server via scheduled scripts to retrieve the latest threat data. Extracted indicators (such as malicious IPs, domains, and URLs) were parsed and automatically integrated into IPTables blocking lists and Suricata's custom rule files.

```
#!/usr/bin/env python3

import subprocess
import json
import os
import re
from datetime import datetime
from cabby import create_client
from stix2 import parse

# ================= CONFIGURATION =================

# OpenTAXII server configuration
TAXII_SERVER = "http://10.54.64.166:5000”
DISCOVERY_PATH = "/taxii/collections/default_collection/objects"
COLLECTION = "default-collection"

# Authentication (optional; leave blank if not required)
USERNAME = "admin"
PASSWORD = "admin"

# Local fallback IoC file
LOCAL_IOC_FILE = "sample_stix.json"

# iptables command
IPTABLES_CMD = "/sbin/iptables"

# Log file
LOG_FILE = "ioc_update.log"

# ================================================

# Regex for IPv4 addresses
IPV4_RE = re.compile(r"\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b")


# ---------------- Utility Functions ----------------

def log(msg):
    """Write logs to console and file."""
    print(msg)
    with open(LOG_FILE, "a") as f:
        f.write(f"{datetime.now():%Y-%m-%d %H:%M:%S}  {msg}\n")
```

```
    def extract_ips_from_stix(stix_data):
        """Extract IPv4 addresses from STIX bundle."""
        indicators = []
        try:
            bundle = parse(stix_data, allow_custom=True)
            for obj in bundle.objects:
                if obj.type == "indicator" and "pattern" in obj:
                    ips = IPV4_RE.findall(obj.pattern)
                    indicators.extend(ips)
        except Exception as e:
            log(f"[!] Failed to parse STIX data: {e}")
        return list(set(indicators))


    def fetch_iocs_from_taxii():
        """Fetch IoCs from OpenTAXII server."""
        log(f"[+] Connecting to OpenTAXII server:
{TAXII_SERVER}{DISCOVERY_PATH}")
        client = create_client(
            TAXII_SERVER,
            discovery_path=DISCOVERY_PATH,
            username=USERNAME,
            password=PASSWORD
        )

        log("[+] Discovering available collections...")
        collections = client.get_collections()
        found = False
        indicators = []

        for collection in collections:
            if COLLECTION.lower() in collection.name.lower():
                found = True
                log(f"[+] Polling collection: {collection.name}")
                content_blocks = client.poll(collection.name)
                for block in content_blocks:
                    try:
                        stix_data = json.loads(block.content)
                        indicators.extend(extract_ips_from_stix(stix_data))
                    except Exception as e:
                        log(f"[!] Error parsing content block: {e}")
                break
```

```
        if not found:
            raise RuntimeError(f"Collection '{COLLECTION}' not found on server.")

        return list(set(indicators))


    def load_local_iocs():
        """Load IoCs from local fallback JSON file."""
        log(f"[+] Loading local IoCs from {LOCAL_IOC_FILE}")
        if not os.path.isfile(LOCAL_IOC_FILE):
            log(f"[!] Local IoC file not found: {LOCAL_IOC_FILE}")
            return []

        with open(LOCAL_IOC_FILE, "r") as f:
            data = f.read()

        ips = IPV4_RE.findall(data)
        return list(set(ips))


    def rule_exists(ip):
        """Check if iptables rule already exists."""
        cmd = [IPTABLES_CMD, "-C", "INPUT", "-s", ip, "-j", "DROP"]
        result = subprocess.run(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
        return result.returncode == 0


    def add_rule(ip):
        """Add DROP rule for a specific IP address."""
        cmd = ["sudo", IPTABLES_CMD, "-A", "INPUT", "-s", ip, "-j", "DROP"]
        result = subprocess.run(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
        return result.returncode == 0


    # ---------------- Main Logic ----------------

    def main():
        log("\n--- SUTMS OpenTAXII IoC Updater ---")

        try:
            ioc_ips = fetch_iocs_from_taxii()
```

```
        if not ioc_ips:
            log("[!] No IoCs fetched from TAXII. Switching to local file...")
            ioc_ips = load_local_iocs()
    except Exception as e:
        log(f"[!] TAXII fetch failed: {e}")
        log("[*] Using local fallback IoC file instead.")
        ioc_ips = load_local_iocs()

    if not ioc_ips:
        log("[!] No IoCs available to process.")
        return

    log(f"[+] Total IoCs loaded: {len(ioc_ips)}")
    added = 0

    for ip in ioc_ips:
        if not IPV4_RE.match(ip):
            continue
        if rule_exists(ip):
            log(f"[SKIP] {ip} already blocked.")
            continue
        if add_rule(ip):
            log(f"[BLOCKED] {ip}")
            added += 1
        else:
            log(f"[ERROR] Failed to block {ip}")

    log(f"[+] Completed. {added} new IPs blocked.")
    log("[+] Firewall rules updated successfully.\n")


if __name__ == "__main__":
    main()
```

This approach enabled continuous synchronization with external threat intelligence networks, keeping the SUTMS up to date with evolving threats. The use of a self-managed TAXII server instead of an open public one ensured control over data aggregation and improved system reliability.

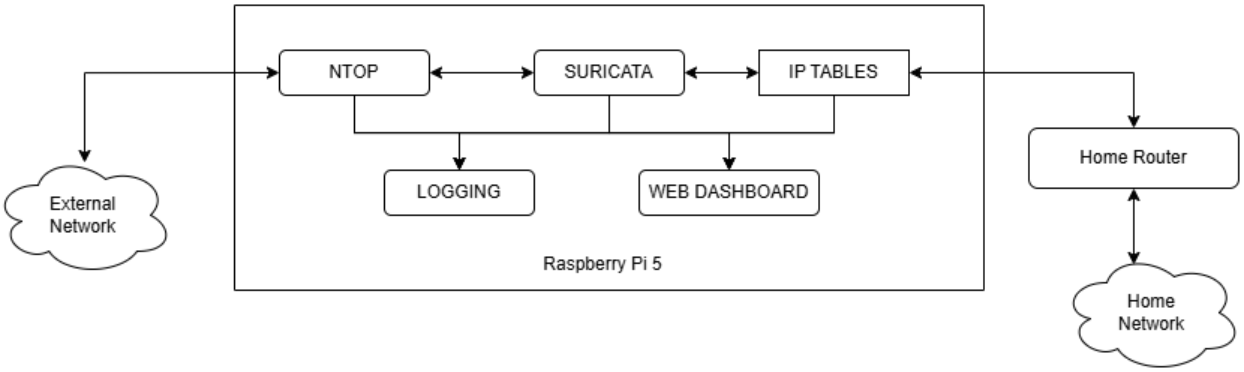
4.2.6 Logging and Data Management

Efficient logging and data management were essential for maintaining transparency and auditability in the system. The rsyslog service was configured as the central logging engine, aggregating logs from Suricata, ntopng, the Flask dashboard, and the custom scripts. All logs were stored under the directory /var/log/sutms/, organized by component type.

Log rotation policies were established using logrotate to prevent excessive disk usage on the Raspberry Pi's limited storage. The logs captured events such as detected intrusions, blocked connections, system performance statistics, and updates from TAXII feeds.

Additionally, the Flask dashboard integrated with the logging system to provide visual summaries of key events, such as high-risk detections and network anomalies. This enabled quick analysis and troubleshooting without requiring direct terminal access. Together, these features formed a reliable data management framework ensuring both operational clarity and efficient disk utilization.

4.2.7 Inline Deployment on Raspberry Pi 5



Finally, the entire system was deployed in an inline configuration, placing the Raspberry Pi 5 between the ISP modem and the home router. This allowed all traffic entering and leaving the network to pass through SUTMS for analysis and filtering. The eth1 interface was connected to the ISP modem (WAN), while eth0 connected to the home router (LAN).

The DHCP server on the Raspberry Pi provided IP addresses to local devices, designating itself as the default gateway. With IP forwarding enabled, packets were inspected by Suricata, filtered by IPTables, and logged before being forwarded to their

destinations.

Performance tests during real-world usage demonstrated stable throughput suitable for typical home network demands. The inline setup successfully enabled proactive detection and blocking of malicious connections while maintaining seamless connectivity for all devices on the network. This deployment model validated the feasibility of running a full-featured, intelligent UTM solution on a compact and energy-efficient Raspberry Pi 5 platform.

4.3 RESULTS AND ANALYSIS

4.3.1 Functional Validation of SUTMS Setup

The implemented Smart Unified Threat Management System (SUTMS) was functionally validated to ensure that all integrated components operated cohesively in real-time network conditions. The primary objective of this validation was to confirm that the system effectively monitored, analyzed, and secured traffic flowing between the external network and the internal home network while maintaining stable network connectivity.

After the complete setup was deployed in inline mode on the Raspberry Pi 5 device, network packets from the Internet were routed through SUTMS before reaching the home router. The validation process was carried out in several phases:

1. **Connectivity and Routing Validation:** The initial phase verified that the Raspberry Pi 5 could properly route packets between its two Ethernet interfaces — one connected to the ISP modem (WAN) and the other to the home router (LAN). IP forwarding and DHCP server configurations were tested to confirm that client systems could obtain IP addresses automatically and access the Internet without latency issues. This confirmed that the system functioned as a transparent gateway.

2. **Module Integration Check:** Each major component — *ntopng*, *Suricata*, *IPTables*, *Flask Dashboard*, *Webmin*, and the *custom TAXII server* — was validated individually and in combination. The services were configured to start automatically on boot using systemd, ensuring uninterrupted operation. Logs were checked to verify that communication between modules was consistent, particularly between

ntopng and Suricata for dynamic rule updates.

3. **Traffic Inspection and Alert Generation:** Controlled test traffic was generated using common protocols such as HTTP, DNS, and SSH. The ntopng module successfully captured flow statistics and identified protocol types. Correspondingly, Suricata analyzed packets and generated alerts for simulated malicious activities (e.g., port scans or suspicious payloads). The alerts were recorded under the logging engine and displayed through the dashboard interface, confirming proper data flow across modules.

4. **Dashboard and Administration Validation:** The Flask-based dashboard accurately reflected system status, current alerts, and active connections in real time. The Webmin interface allowed remote management of network services, firewall rules, and process controls, reducing the need for manual command-line intervention. This validated the system's manageability and ease of monitoring.

5. **TAXII and Firewall Synchronization:** The custom opentaxii server was tested for connectivity with multiple external threat-intelligence providers. The aggregated Indicators of Compromise (IoCs) were automatically parsed and updated within the firewall ruleset, enabling proactive blocking of known malicious domains and IPs.

Through these validation steps, the SUTMS setup was confirmed to operate reliably as an integrated security gateway. All functional modules interacted seamlessly, providing flow monitoring, intrusion detection, and dynamic firewalling in a unified framework suitable for home network environments.

4.3.2 Unified Monitoring

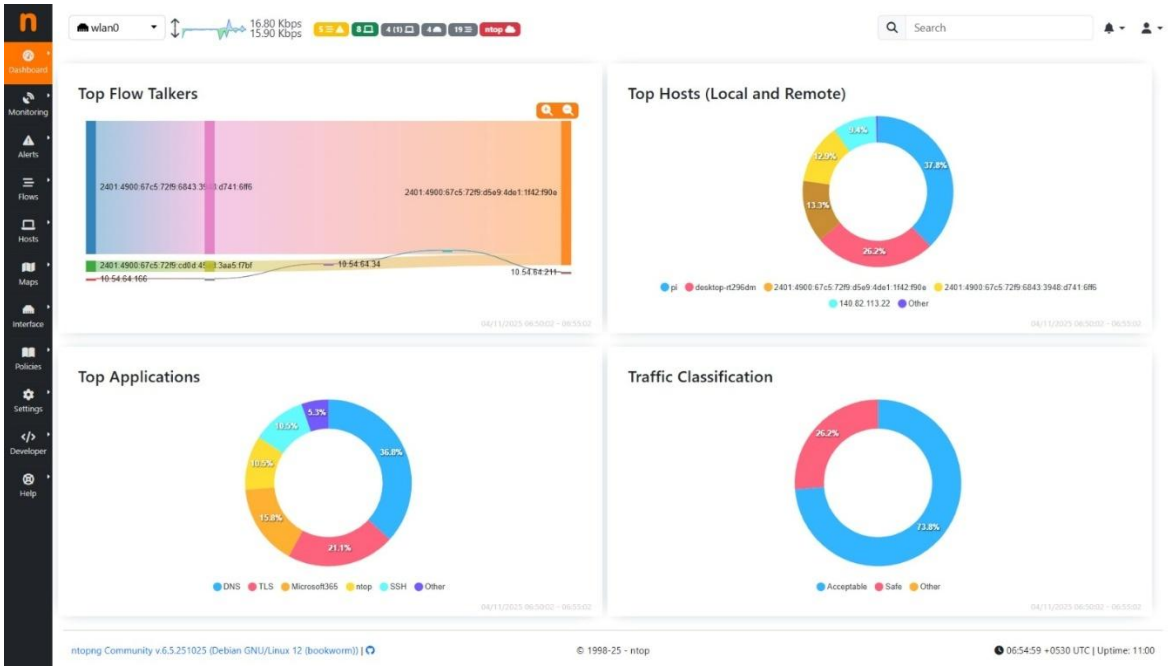
One of the key objectives of the project was to achieve a centralized monitoring environment that allows real-time visibility and control over network activity, intrusion alerts, and system performance. To accomplish this, a unified monitoring framework was established by integrating multiple visualization and management interfaces — namely ntopng, Flask Dashboard, and Webmin — into the Smart Unified Threat Management System (SUTMS). Each interface contributes a distinct layer of monitoring and administration, together providing a complete operational view of the system.

a) ntopng Dashboard

The ntopng dashboard served as the primary network flow analyzer within the SUTMS framework. It provided detailed insights into live traffic, protocol usage, and endpoint communication patterns.
During validation, ntopng effectively displayed:

- Real-time bandwidth utilization for both WAN and LAN interfaces.
- Lists of active hosts, top talkers, and connected devices in the network.
- Protocol distribution graphs showing proportions of HTTP, DNS, SSL, and other traffic types.
- Flow statistics, connection durations, and packet counts for each session.

This information forms the foundation for protocol extraction, enabling the system to determine which rule sets should remain active in Suricata to optimize resource usage.



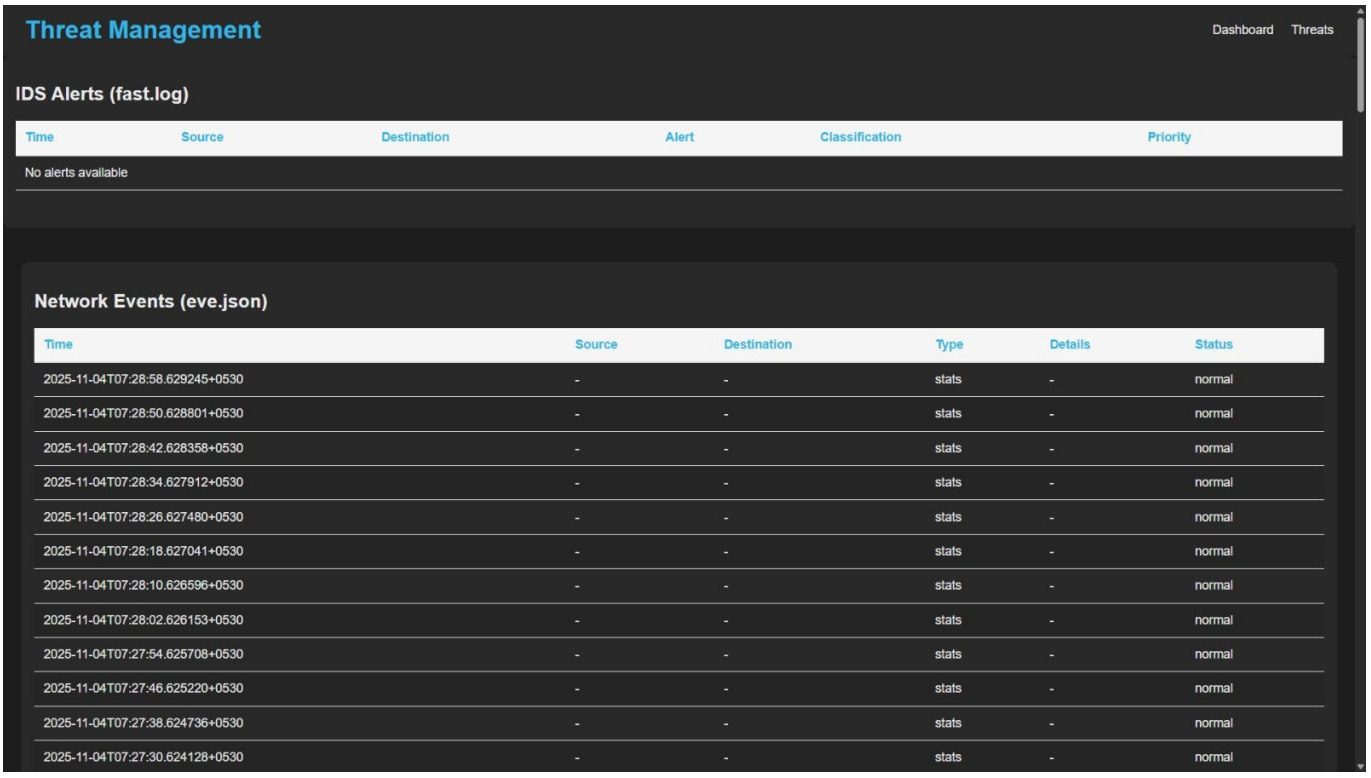
b) Flask Dashboard

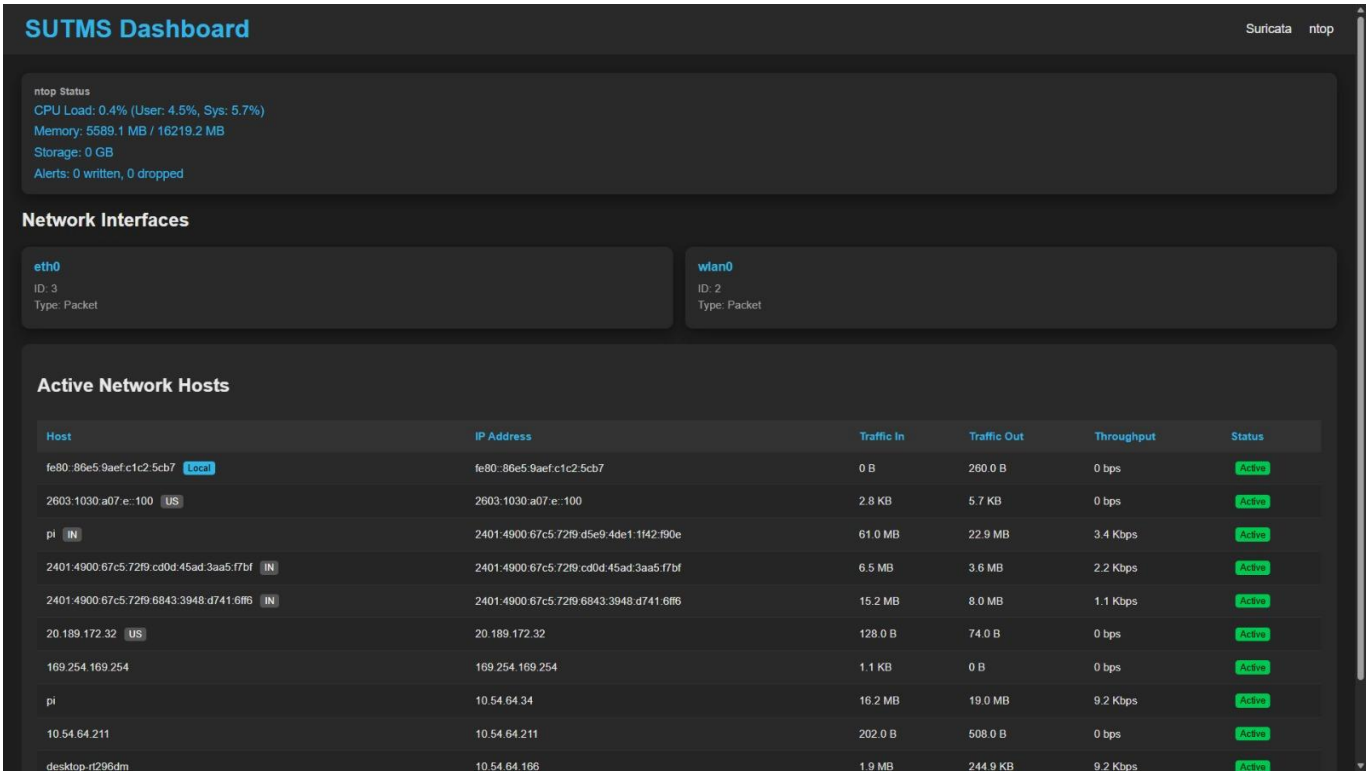
To simplify the management of SUTMS components and provide a minimal yet effective unified interface, a Flask-based dashboard was developed.
This dashboard consolidates data from the logging engine, system status checks, and

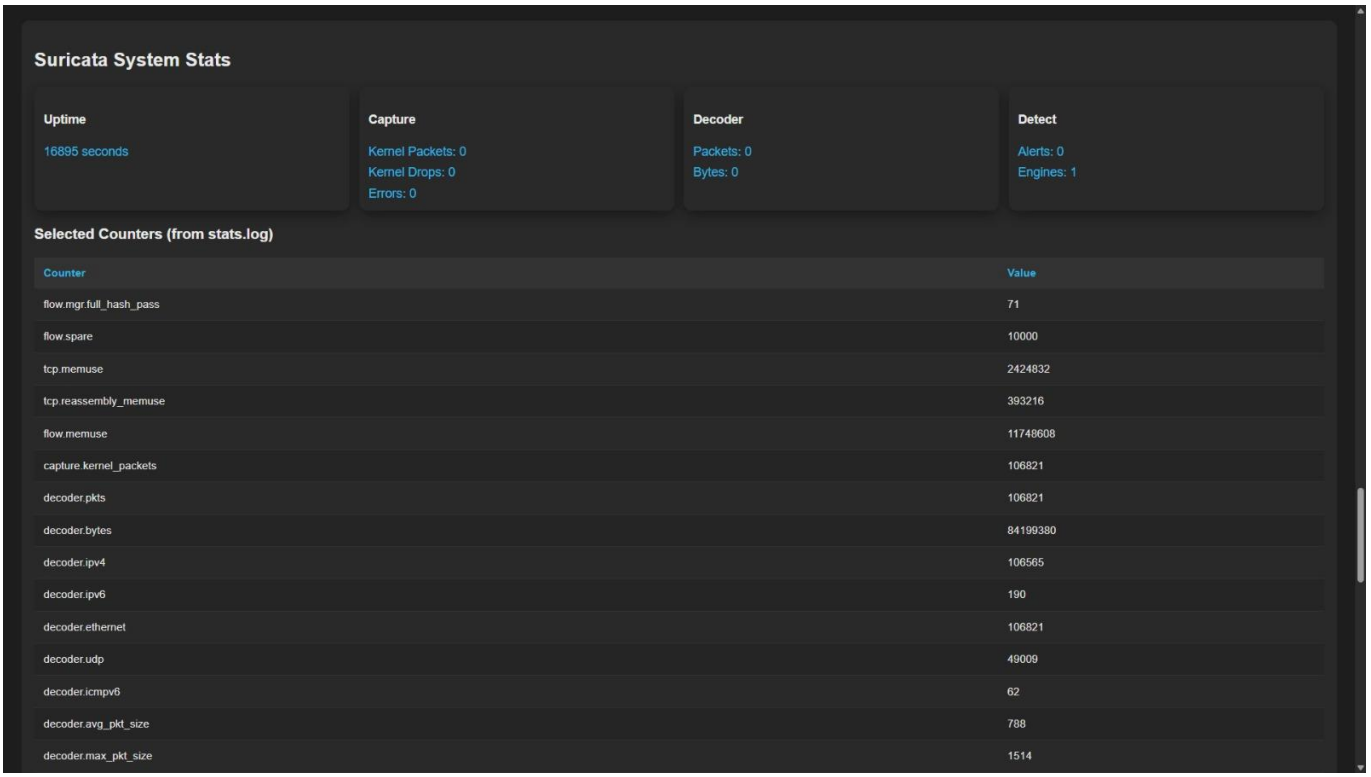
Suricata alert logs, presenting them in an organized and user-friendly format.
Key features include:

- Overview of system health metrics such as CPU usage, memory load, and network throughput.
- A log viewer for real-time Suricata alerts and firewall actions.
- Control options for restarting or stopping individual modules (e.g., ntopng, Suricata, or TAXII server).
- Simplified visualization of blocked IPs and threat summaries.

The Flask interface offers a lightweight and resource-efficient alternative to heavier monitoring solutions, tailored for use on Raspberry Pi devices.




SUTMS Dashboard
Suricata
ntop
ntop Status
CPU Load: 0.4% (User: 4.5%, Sys: 5.7%)
Memory: 5589.1 MB / 16219.2 MB
Storage: 0 GB
Alerts: 0 written, 0 dropped
Network Interfaces
eth0
ID: 3
Type: Packet
wlan0
ID: 2
Type: Packet
Active Network Hosts
Host
IP Address
Traffic In
Traffic Out
Throughput
Status
fe80::86e5:9aef:c1c2:5cb7 Local
fe80::86e5:9aef:c1c2:5cb7
0 B
260.0 B
0 bps
Active
2603:1030:a07:e::100 US
2603:1030:a07:e::100
2.8 KB
5.7 KB
0 bps
Active
pi IN
2401:4900:67c5:72f9:d5e9:4de1:1f42:f90e
61.0 MB
22.9 MB
3.4 Kbps
Active
2401:4900:67c5:72f9:cd0d:45ad:3aa5:f7bf IN
2401:4900:67c5:72f9:cd0d:45ad:3aa5:f7bf
6.5 MB
3.6 MB
2.2 Kbps
Active
2401:4900:67c5:72f9:6843:3948:d741:6ff6 IN
2401:4900:67c5:72f9:6843:3948:d741:6ff6
15.2 MB
8.0 MB
1.1 Kbps
Active
20.189.172.32 US
20.189.172.32
128.0 B
74.0 B
0 bps
Active
169.254.169.254
169.254.169.254
1.1 KB
0 B
0 bps
Active
pi
10.54.64.34
16.2 MB
19.0 MB
9.2 Kbps
Active
10.54.64.211
10.54.64.211
202.0 B
508.0 B
0 bps
Active
desktop-rt296dm
10.54.64.166
1.9 MB
244.9 KB
9.2 Kbps
Active


Suricata System Stats
Uptime
16895 seconds
Capture
Kernel Packets: 0
Kernel Drops: 0
Errors: 0
Decoder
Packets: 0
Bytes: 0
Detect
Alerts: 0
Engines: 1
Selected Counters (from stats.log)
Counter
Value
flow.mgr.full_hash_pass
71
flow.spare
10000
tcp.memuse
2424832
tcp.reassembly_memuse
393216
flow.memuse
11748608
capture.kernel_packets
106821
decoder.pkts
106821
decoder.bytes
84199380
decoder.ipv4
106565
decoder.ipv6
190
decoder.ethernet
106821
decoder.udp
49009
decoder.icmpv6
62
decoder.avg_pkt_size
788
decoder.max_pkt_size
1514

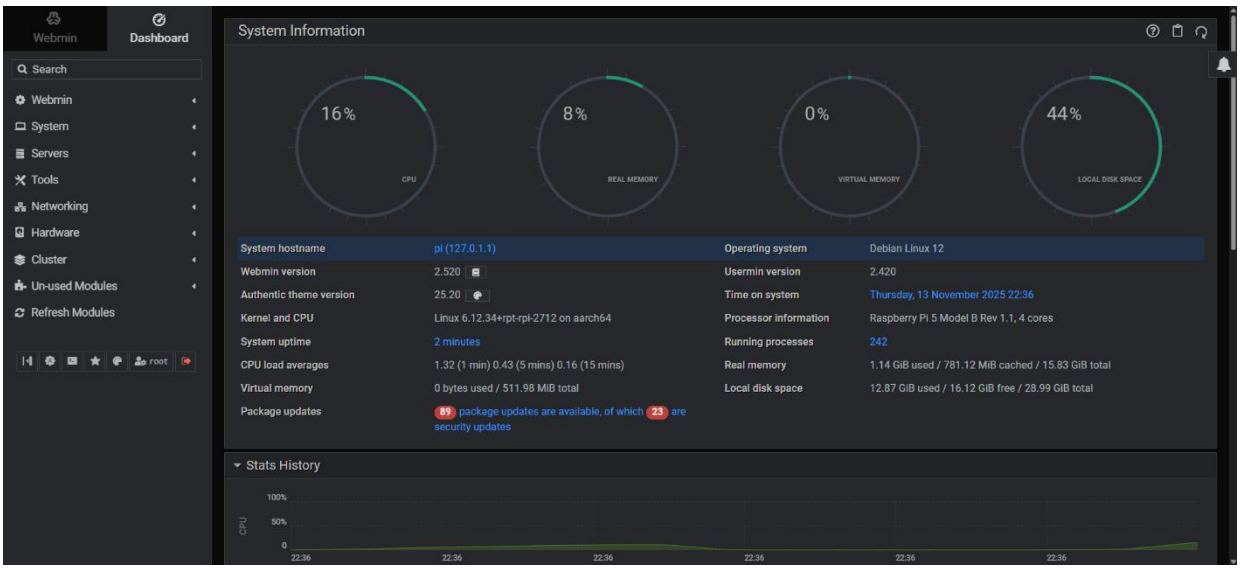
c) Webmin Dashboard

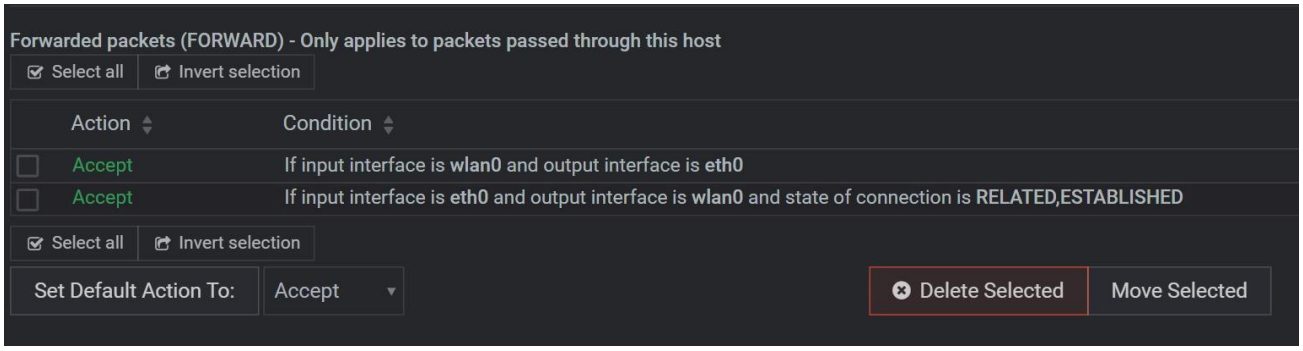
Webmin was deployed to provide administrative control and system configuration through a graphical web interface. It serves as the management layer of SUTMS, giving the administrator full control over network configurations, firewall rules, and background processes.

Through Webmin, administrators can:

- Manage network interfaces, routes, and DHCP configurations.
- Monitor service statuses and restart them remotely.
- View and edit IPTables rules for packet filtering.
- Access log files and perform system-level operations without direct terminal access.

This integration of Webmin ensures that the system remains easily maintainable and remotely accessible, even for administrators with limited command-line experience.





By combining these three monitoring layers — ntopng for network visibility, Flask for component-level monitoring, and Webmin for system administration — SUTMS achieved a truly unified monitoring environment. This structure enables continuous supervision of both traffic-level analytics and system-level operations from a single deployment, providing clear visibility and improved control over the network's security posture.

4.3.3 Custom Rule Optimization

The core enhancement introduced in this implementation of the Smart Unified Threat Management System (SUTMS) was the custom rule optimization mechanism designed to reduce the computational overhead of the intrusion detection engine while maintaining accuracy in traffic inspection. This mechanism operates in two main stages:

1. Protocol extraction using ntopng, and
2. Dynamic rule updation in Suricata based on the extracted protocols.

Together, these processes ensure that only relevant rule sets are active at any given time, minimizing CPU and memory usage on the Raspberry Pi 5 platform.

a) Protocol Extraction (ntopng)

The ntopng module served as the network flow analyzer responsible for identifying all active protocols traversing the network in real time. A custom Python script was developed to interact with the ntopng REST API and periodically extract the list of detected protocols from live traffic sessions. The script filtered out redundant or inactive protocol entries and generated a simplified list containing only those protocols that were currently in use by devices on the network.

Key functions of the protocol extraction phase include:

- Accessing ntopng's local REST endpoint to retrieve protocol statistics.
- Parsing and filtering JSON data to isolate active Layer 4 and Layer 7 protocols.
- Writing the filtered protocol list to a configuration file accessible to Suricata.
- Scheduling periodic updates using a cron job for continuous synchronization.

This process allowed SUTMS to maintain an up-to-date understanding of active communication types such as HTTP, HTTPS, DNS, SSH, and others, providing the foundation for adaptive intrusion rule management.

```
hackerspace1@pi:/var/lib/ntopng $ cat protocols_full.json
{
  "timestamp": "2025-11-04T02:05:02.131188",
  "window_minutes": 60,
  "protocols": [
    {
      "name": "DNS",
      "count": 136
    },
    {
      "name": "TLS",
      "count": 53
    },
    {
      "name": "Microsoft365",
      "count": 37
    },
    {
      "name": "SSH",
      "count": 27
    },
    {
      "name": "HTTP",
      "count": 17
    },
    {
      "name": "Github",
      "count": 8
    },
    {
      "name": "MDNS",
      "count": 4
    },
    {
      "name": "ICMPV6",
      "count": 2
    },
    {
      "name": "DHCP",
      "count": 1
    }
  ]
hackerspace1@pi:/var/lib/ntopng $
```

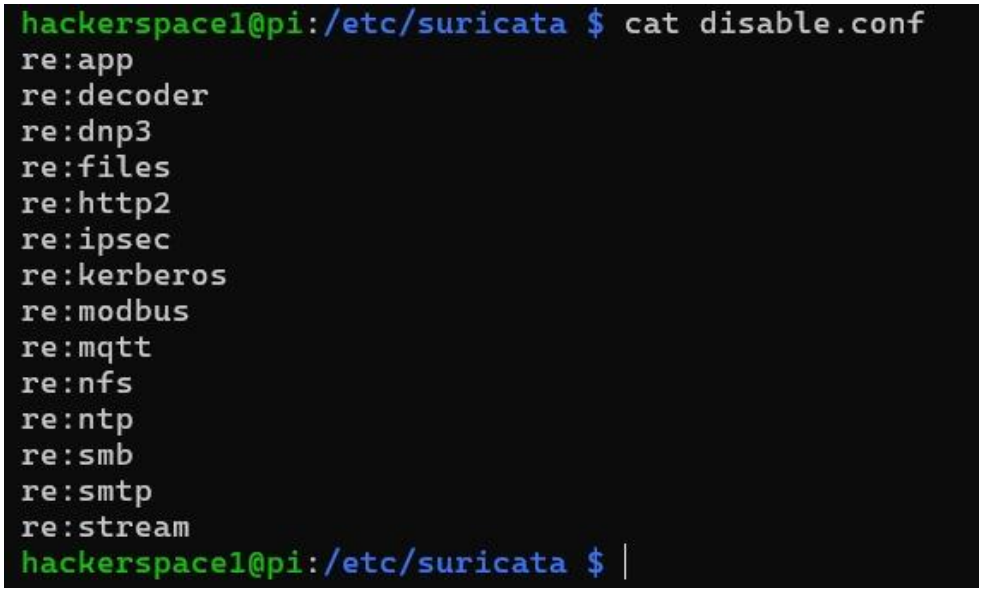
b) Rule Updation (Suricata)

Once the protocol list was generated, the second stage of optimization involved updating Suricata's active ruleset based on the identified protocols. The same Python script read the extracted protocol list and modified Suricata's configuration to enable only the rule categories relevant to the active protocols. This prevented Suricata from loading unnecessary rules for unused protocols, thereby reducing system resource consumption and improving throughput.

The following steps are executed during rule updation:

- Mapping extracted protocols to corresponding Suricata rule files or categories (e.g., HTTP → http.rules, DNS → dns.rules).
- Disabling non-essential rule sets by commenting them out in the Suricata configuration.
- Restarting or reloading the Suricata service to apply updated configurations dynamically.
- Logging each optimization event for verification and performance tracking.

This adaptive rule management approach ensures that the IDS engine remains lightweight and context-aware. Tests conducted during validation showed a noticeable reduction in CPU utilization after rule optimization, with no compromise in detection capability for active services.



4.3.4 System Limitations

While the implemented Smart Unified Threat Management System (SUTMS) successfully integrated multiple security modules and delivered unified monitoring and adaptive rule optimization, certain limitations were observed during deployment and testing. These constraints primarily arose from hardware restrictions, software dependencies, and environmental factors inherent to home network environments.

1. **Hardware Resource Constraints:** The Raspberry Pi 5, though significantly more powerful than its predecessors, still offered limited CPU and memory resources compared to full-scale network security appliances. Running multiple modules such as ntopng, Suricata, and Webmin simultaneously resulted in moderate CPU utilization during high-traffic periods. This limited the scalability of the system when monitoring several gigabits of network traffic or multiple concurrent high-bandwidth connections.

2. **Storage and Log Management:** Continuous packet inspection and flow logging generated large amounts of data. Since the setup relied on a microSD card for storage, write endurance and space availability became concerns over long-term use. Although log rotation was configured, extended deployments might have required external or cloud-based log storage for better reliability.

3. **Suricata Rule Reload Overhead:** The dynamic rule updation mechanism, while effective, introduces short service interruptions whenever Suricata reloads its rule sets. Although minimized using graceful reloads, these momentary interruptions can still cause minor packet drops during reconfiguration.

4. **Limited Multi-user Interface Support:** The Flask-based dashboard was intentionally designed to be minimal and lightweight for Raspberry Pi deployment. As a result, it supports only a single administrative session and lacks advanced user management, alert filtering, or historical visualization features that full-scale SIEM tools provide.

5. **Dependency on External Threat Intelligence Sources:** The system's proactive blocking capability relies on external STIX/TAXII feeds. Any downtime, access restriction, or format change from these providers can delay IoC updates, temporarily affecting real-time protection efficiency.

6. **Environmental and Network Dependency:** The SUTMS operates optimally in stable network environments. Frequent network restarts, ISP outages, or DHCP conflicts may require manual service restarts or reinitialization through Webmin.

4.4 SUMMARY

This chapter detailed the practical aspects of implementing the Smart Unified Threat Management System (SUTMS) on a Raspberry Pi 5 platform. The project specifications were first outlined, highlighting the hardware, software, and network configurations required to deploy the system in an inline setup between the external network and the home router.

The implementation section described the configuration of each core module—ntopng for flow monitoring, Suricata for intrusion detection, IPTables for firewall enforcement, and the opentaxii server for threat-intelligence integration—along with supporting services such as the Flask dashboard, Webmin interface, and the logging engine. These components were integrated to operate cohesively, providing real-time monitoring and automated threat response within a compact, low-power device.

Subsequent analysis demonstrated the functional validation of the SUTMS setup and the effectiveness of its unified monitoring environment, which consolidated insights from ntopng, Flask, and Webmin dashboards. The custom rule optimization mechanism—involving protocol extraction from ntopng and dynamic rule updates in Suricata—proved successful in minimizing resource usage while maintaining reliable detection.

Finally, the system's limitations were acknowledged, particularly those related to hardware constraints, storage endurance, and dependency on external threat-intelligence feeds. Despite these challenges, the project successfully established a working prototype of a lightweight, adaptive, and modular UTM solution for home networks.

This chapter thus bridged the design and implementation phases with the evaluation outcomes, laying the foundation for the concluding discussion on the overall significance of the work and its potential future enhancements.

CHAPTER 5

CONCLUSION

5.1 Conclusion

The Smart Unified Threat Management System (SUTMS) developed in this project successfully demonstrated the feasibility of building a lightweight yet effective network security solution tailored for home and small office environments. By integrating multiple open-source tools such as Suricata, ntopng, IPTables, and OpenTAXII, the system offered intrusion detection, flow analysis, and dynamic threat intelligence in a unified framework.

One of the key contributions of this work was the implementation of a custom rule optimization mechanism, which dynamically adjusted Suricata's rule sets based on active network protocols extracted from ntopng. This innovation significantly reduced system resource usage, making the setup practical for constrained hardware platforms like the Raspberry Pi 5. The system also included a minimal Flask dashboard for component monitoring and a Webmin interface for server management, enhancing usability and administrative control.

The successful inline deployment between the home router and external network validated the operational reliability of the design. It was able to inspect, log, and manage network traffic efficiently while maintaining normal connectivity. The integration of a custom TAXII server further strengthened the system by providing real-time IoC updates from multiple intelligence sources. Overall, the project accomplished its goal of delivering a compact, adaptive, and resource-efficient unified threat management system that could enhance the cybersecurity posture of home networks.

5.2 Future Scope

Although the developed system achieved a functional and optimized home UTM solution, several areas could be expanded in future work. The current setup could be enhanced by integrating machine learning-based anomaly detection models to identify zero-day attacks and suspicious traffic patterns automatically. Implementing data visualization dashboards using advanced frameworks such as Grafana could have

provided richer analytics and real-time network insights.

Future versions could also have included automated policy generation based on behavioral patterns and user preferences, reducing manual rule configuration. The integration of containerization technologies like Docker could have simplified deployment across multiple hardware platforms. Additionally, extending support for cloud-based threat intelligence sharing would have allowed SUTMS instances to contribute to and benefit from collective learning across distributed deployments.

Finally, upgrading to higher-performance SBCs or integrating multi-threaded processing could have further scaled the system for small enterprise use. With these enhancements, SUTMS could have evolved from a home-level defense solution into a more comprehensive, adaptive, and self-learning security framework.

CHAPTER 6

REFERENCE

[1] V. B. M. Yadav, K. S. Smruti Rekha, and S. K. Padhi, "**SUTMS: Designing a Smart Unified Threat Management System for Home Networks**," *IEEE Access*, vol. 10, pp. 70077–70093, 2022, doi: 10.1109/ACCESS.2022.3187691.

[2] A. Gupta, P. Singh, and R. Chaudhary, **"Intrusion detection for IoT-enabled smart homes,"** *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17245–17255, Sept. 2022.

[3] T. R. Kumar and S. N. Patel, **"Phishing and malicious URL detection using Raspberry Pi for network-edge defense,"** *IEEE Access*, vol. 10, pp. 114532–114543, Dec. 2022.

[4] L. Chen, M. Hassan, and F. Wang, **"Traffic trace artifacts from port mirroring in network monitoring,"** *Computer Networks*, vol. 203, p. 108647, June 2022.

[5] J. W. Park, K. Lee, and S. Cho, **"Performance evaluation of open-source intrusion detection systems in high-speed networks,"** *Journal of Network and Computer Applications*, vol. 190, p. 103187, Apr. 2021.

[6] R. K. Das, A. Verma, and P. Mishra**, "Survey on unified threat management (UTM) systems for home networks,"** *IEEE Access*, vol. 11, pp. 41205–41219, Apr. 2023.

[7] M. T. Nguyen, D. Lee, and E. Kim**, "A comprehensive study on STIX and TAXII standards for cyber-threat intelligence exchange,"** *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3051–3063, May 2023.

[8] S. Patil, N. Jadhav, and V. Borkar, **"Automatic device classification using Arduino platform for network identification,"** *International Journal of Electrical and Computer Engineering*, vol. 12, no. 4, pp. 3865–3872, Aug. 2022.

[9] K. P. Sinha and L. Rajendran**, "IoT device classification using machine learning algorithms,"** *Sensors*, vol. 22, no. 18, p. 7012, Sept. 2022.

CONFIRMATION OF PAPER SUBMISSION:

IJERT - Your Manuscript Submitted Successfully - Paper id: IJERTV14IS110181

1 message

IJERT-Info <info@ijert.org> Sat, 15 Nov, 2025 at 5:26 pm
Reply to: IJERT-Info <info@ijert.org>
To: aswi.71772218102@gct.ac.in

International Journal of Engineering Research & Technology (IJERT)

(ISSN: 2278-0181)

www.ijert.org

Hi Dr. M. Blessy Queen Mary,

We received your manuscript and it is submitted to editor for review process.

It may takes 3-7 working days for completing initial manuscript review by members of our initial screening team/review board. You will get confirmation email once your manuscript initial review phase completes.

Your Manuscript details are as follow:

Title : Smart Unified Threat Management System (SUTMS)
Paper Id : IJERTV14IS110181

You may Check your Paper Status using below link

https://ems.ijert.org/paper-status-nXywKHQCbSUPw9c4IaAXhPCsU

Use below link to **Verify** & **Activate** your account and confirm that your contact details are correct:

https://ems.ijert.org/verification-confirm-nXywKHQCbSUPw9c4IaAXhPCsU

EMS - Author Login

You now have access to an online IJERT - EMS Author Manager account from where you can **track real time status of your submitted manuscripts, download soft copy of certificates, invoice, article, author guidelines, article download , citations reports & much more.**

You can visit the IJERT - EMS Author Manager , at any time by going to, **https://ems.ijert.org**

Best Wishes,

Editor,
https://www.ijert.org

Please add , mail@ems.ijert.org to your contact list to prevent future mails from going into Junk/Spam folder.

IJERT Home | Log in to Author Account | Update your Profile | Contact Us

You are receiving this email because you are registered user of IJERT

ESRSA