Modern University for Technology & Information

Faculty of Computers & Artificial Intelligence

Department of Computer Science

Ransomware Detection and Prevention System Using AI and ML

Submitted by:

Esraa Salah Hassan Ahmed Shalan

Nour Mohamed Morshed El-Shafie Nasr

Ahmed Abd El-Aziz Ali Abd El-Aziz El-Sadany

Ahmed Bassam Refaat Gaafar Ali

Ahmed Ibrahim El-Sayed Ahmed Abdullah

Abdullah Alaa Moukhtar Mohamed

Mohamed Ahmed Yousef Salama

Abdullah Anas Abdullah El-Gamal

Supervised by:

Prof. Dr. Hafez Abd El-Wahab

Dr. Tarek Soliman

T.A Ghada Nady

T.A. Gerges Mourad

Acknowledgement

All praise is due to **Allah**, who has guided us to this path; for we would certainly not have found it unless **Allah** had guided us. We extend our deepest gratitude to Modern University for Technology and Information (M.T.I.) for fostering our intellectual and personal growth and for providing an excellent learning environment throughout our academic years. A special and sincere note of thanks is reserved for our project supervisors: **Prof. Eng. Dr. Hafez Abd El-Wahab**, **Dr. Tarek Soliman**, **T.A. Ghada Nady**, and **T.A**. **Gerges Mourad.** Their invaluable guidance, continuous support, and dedicated efforts were instrumental in the successful completion of this project. Our heartfelt gratitude goes to our **Families**. Their unwavering support, encouragement, and patience were the foundation upon which this achievement was built, and we simply could not have accomplished this without them. We are also deeply grateful to the leadership and faculty of MTI for their dedication. In particular, we thank the MTI President, **Prof. Dr. Olfat Kamel**, for her vision, and **Prof. Dr. Mohamed Taher El-Mayah**, Dean of the Faculty of Computers & Artificial Intelligence, and **Dr. Mohamed El Gazzar,** Vice Dean for academic affairs, for their encouragement, guidance and support. We extend our thanks to the department heads, **Prof. Dr. Hanafy Isamil, Prof. Dr. Hesham EL Deep** and **Prof. Dr. Eman Taha**. Our sincere appreciation also goes to all the professors who guided us, including **Prof. Dr. Alaa Abd El-Raheem**, **Prof. Dr. M. Anwar Assal, Prof. Dr. Mahamoud El-Shishtawy, Prof. Dr. Sayed Bakar**, **Prof. Dr. Rasha Saeed**, **Dr. Rania Ahmed**, **Dr. Sayed Orabi, Dr. Hadeer Mohamed and Dr. Ahmed Said** as well as all the assistant lecturers whose practical demonstrations were vital.

Abstract

Ransomware has emerged as a critical cybersecurity threat, causing severe financial and data losses while rendering traditional signature-based security solutions increasingly ineffective against its rapidly evolving variants. This project presents an advanced Ransomware Detection and Prevention System (RDPS) that implements a comprehensive, two-pronged defense strategy using Artificial Intelligence (AI) to identify malicious activity at both the network and host levels.

The first component is an online, network-based model that provides real-time threat identification by analyzing network traffic flows. It utilizes a suite of algorithms, including classical Machine Learning models (e.g., Random Forest, XGBoost) and advanced Deep Learning models (CNN and RNN), trained on a custom dataset of refined network features to identify patterns indicative of an attack.

The second component is an offline, host-based model that provides pre-execution analysis of suspicious executable files. This model uses highly effective Machine Learning classifiers to distinguish malicious from benign files based on a rich set of static and dynamic features gathered from sandboxed analysis.

The system translates detection into proactive prevention. Upon identifying a threat, the RDPS is designed to take automated action: suspicious network connections are terminated by the online model, and malicious files are quarantined by the offline model to prevent execution. To operationalize these capabilities, the system is managed through an intuitive, web-based dashboard built with HTML, CSS and JavaScript, which provides real-time threat monitoring and alert visualization. Performance evaluation of both components, conducted using real-world ransomware datasets and specialized simulations, demonstrates the system's ability to achieve high detection accuracy while maintaining low false positive rates. By integrating network and host-level analysis with automated response, this AI-driven approach provides a robust, multi-layered defense framework, showcasing a significant advancement in proactive cyber threat mitigation.

Table of Contents

List of Abbreviations:

RDPS	Ransomware Detection and prevention system
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
RF	Random Forest
XGBoost	Extreme Gradient Boosting
NIDS	Network Intrusion Detection System
NIPS	Network Intrusion Prevention System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
C&C / C2	Command and Control
DDoS	Distributed Denial of Service
RDP	Remote Desktop Protocol

SMB	Server Message Block
DNS	Domain Name System
TLS/SSL	Transport Layer Security / Secure Sockets Layer
DGA	Domain Generation Algorithm
PE	Portable Executable
DLLs	Dynamic-Link Libraries
BYOD	Bring Your Own Device
NGFW	Next-Generation Firewall
PCI DSS	Payment Card Industry Data Security Standard
SPI	Stateful Packet Inspection
ANNs	Artificial Neural Networks
ReLU	Regularization techniques (Lasso / Ridge)
L1 / L2	Regularization techniques (Lasso / Ridge)

List of Equations

1. **Accuracy =** (TP + TN) / (TP + TN + FP + FN)
2. **Precision** = TP / (TP + FP)
3. **Recall** = TP / (TP + FN)
4. **F1-Score** = 2 * (Precision * Recall) / (Precision + Recall)

List of Figures

List of Tables

Chapter 1
Introduction

Chapter 1:
Introduction

1.1 Overview

Cyberattacks are deliberate exploitations of systems, networks, or digital services aimed at stealing data, disrupting operations, compromising confidentiality, or causing damage. These attacks have increased significantly in both frequency and sophistication, affecting individuals, organizations, and entire nations. Common forms include **phishing**, denial-of-service (**DoS**), **SQL injection**, and **malware-based attacks**.

Among the most pervasive forms of cyberattack is **malware**, a collective term for malicious software designed to infiltrate, damage, or disable information systems. **Malware** comes in various forms, including **viruses**, **worms**, **Trojans**, **spyware**, **adware**, and **ransomware**. As shown in Figure 1.1, malware can take on numerous behaviors and forms, all of which threaten data integrity, user privacy, and system functionality.

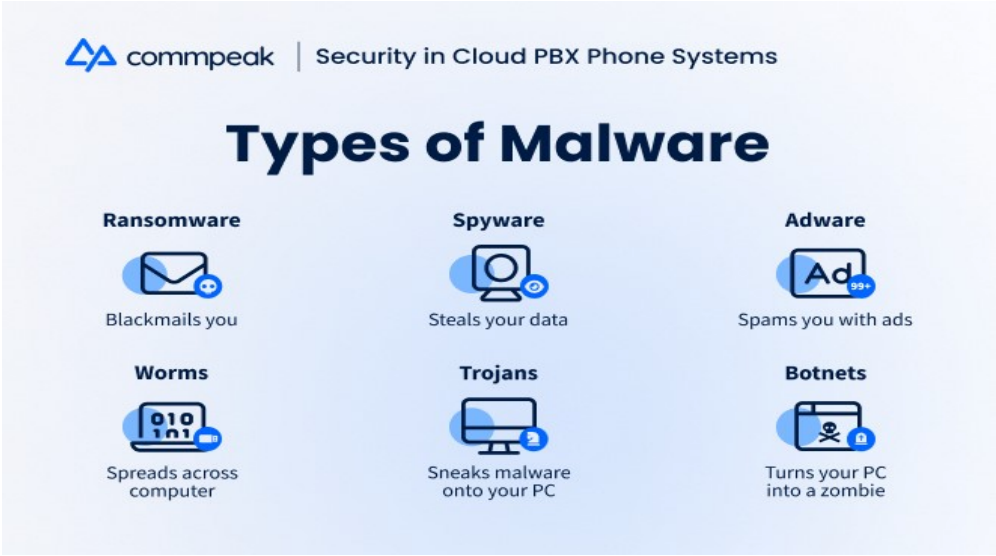


Figure 1. 1 Malware Types [1]

Ransomware is a particularly dangerous type of malware that encrypts a victim's data or locks access to systems, followed by a demand for ransom—usually in **cryptocurrency**—in exchange for restoration. Ransomware campaigns often spread quickly across networks, causing widespread damage. Figure 1.2 illustrates a typical **ransom note** displayed on a victim's device, demanding payment and threatening consequences if the ransom is not fulfilled.

Ransomware is generally classified into two major types:

- **Locker Ransomware**: This type restricts access to the infected system without encrypting files. It usually displays a full-screen message demanding payment while denying the user access to the desktop or applications.

  Example: **Reveton,** known for impersonating law enforcement agencies to pressure victims into paying fines.

- **Crypto Ransomware**: This variant encrypts user files and demands a ransom for the decryption key. The data becomes **inaccessible** without the key, posing a significant threat to individuals and organizations.
  Examples: **WannaCry**, **Cerber**, **Locky**, **TeslaCrypt**, and **CryptoWall**. These ransomware families have been responsible for some of the most widespread and damaging attacks globally.

Figure 1. 2 Ransom note [2]

1.2 Motivation

Ransomware continues to be one of the **most dangerous** cybersecurity threats, impacting both **public** and **private** sectors globally. The ransomware ecosystem has evolved rapidly, with attackers becoming more organized, targeting critical infrastructure, and demanding increasingly higher ransoms.

Attacks now span **healthcare**, **education**, **finance**, **energy**, **manufacturing**, and **governmental institutions**. As shown in Figure 1.3, ransomware does not focus on a single domain but affects a wide array of industries.

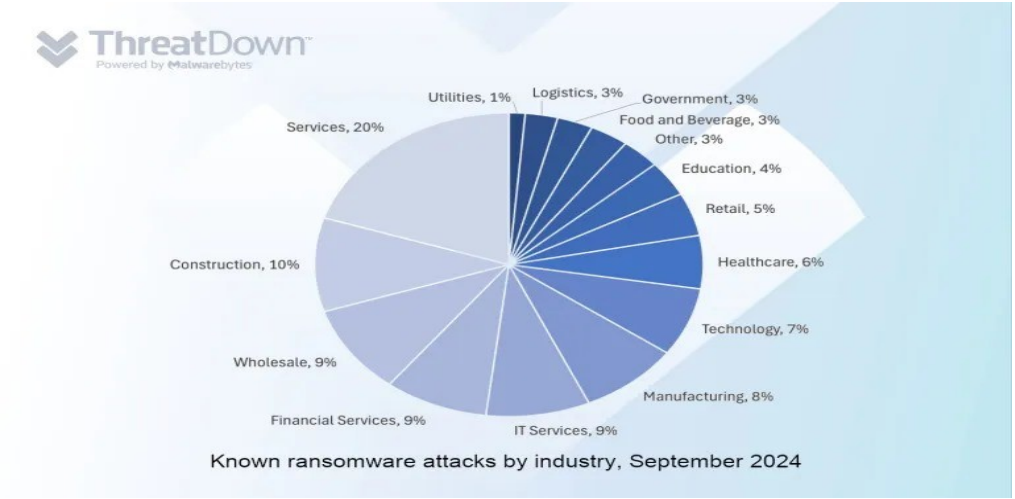


Figure 1. 3 Ransomware attacks by industry [3]

Largest Ransomware Attacks in 2024:

An analysis of the most severe ransomware incidents in **2024** reveals growing technical sophistication and devastating consequences. Below are some of the top ransomware attacks of **2024**:

- **UnitedHealth** — Estimated **$3.09 billion** in losses. This was one of the most expensive and disruptive ransomware attacks in history, severely affecting healthcare services nationwide.
- **Panera Bread** — A ransomware campaign caused a week-long system outage, halting online orders and causing major financial and reputational harm.
- **CDK Global** — A **$50 million** ransom demand was made against this automotive software provider, which affected thousands of car dealerships across the United States. [4]

These events highlight the real-world impact of ransomware on essential services, from hospitals to food services and enterprise software platforms.

Most Famous Ransomware Attacks Globally:

In addition to recent attacks, there have been historically significant incidents that continue to shape global cybersecurity strategies.

The following Table 1.1 summarizes **four** of the most impactful ransomware attacks in the world to date:

Table 1. 1 Examples of Ransomware attacks [5]

Attack	Year	Impact	Ransom Paid
WannaCry	2017	Locked systems in 150 countries, including the UK's NHS, cancelling surgeries and causing huge losses.	Ranging from hundreds of millions to billions of dollars
Garmin	2020	Disrupted GPS services for millions including aviation tools affecting user operations worldwide.	Multimillion-dollar
Colonial Pipeline	2021	Caused fuel shortages and panic buying, exposing critical infrastructure weaknesses.	$4.4 million
JBS Foods	2021	Halted meat production globally, leading to supply chain disruptions	$11 million

Economic Impact of Ransomware until 2024:

As shown in Figure 1.4, ransomware caused an estimated **$42 billion** in global damages in **2024** alone. This includes ransom payments, downtime, recovery costs, **data breaches,** and reputational harm.

The average cost per ransomware attack in 2024 reached **$5.13 million**, including both financial losses and operational disruption. Furthermore, ransomware complaints involving U.S. critical infrastructure increased by **9%,** emphasizing its continued threat level.

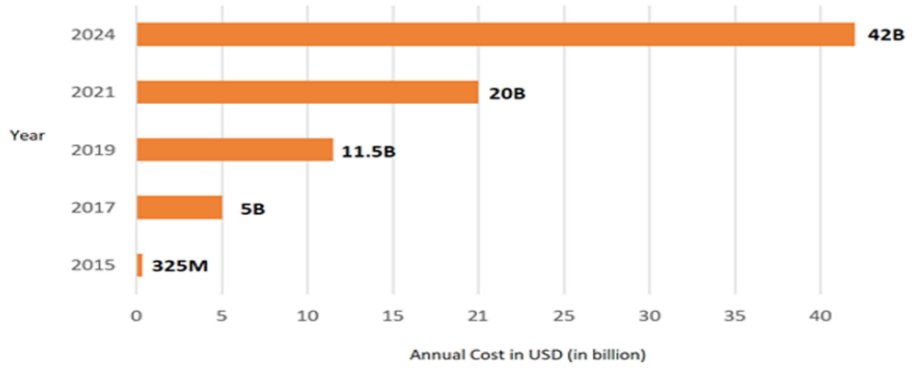


Figure 1.4 Ransomware Financial losses [6]

Technical Targets of Ransomware:

Ransomware authors continue to diversify their attack surfaces by exploiting vulnerabilities in a range of operating systems. As illustrated in Figure 1.5, **Windows-based** systems remain the primary targets, followed by **Linux** and **Android**, due to their widespread use in corporate environments, servers, and mobile platforms.

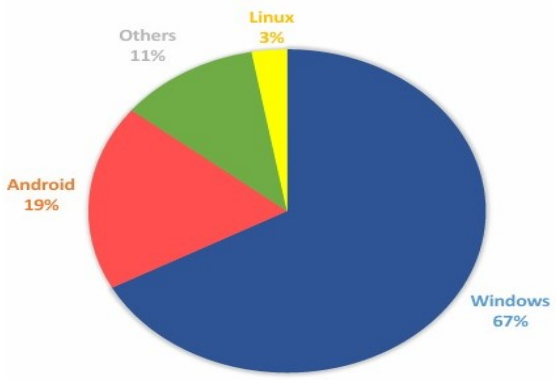


Figure 1. 5 Ransomware OS attacks [6]

1.3 Problem Statement

Cybersecurity refers to the practice of protecting systems, networks, and data from digital attacks. Traditionally, this was handled by **rule-based** systems such as **firewalls**, intrusion detection systems (**IDS**), and **antivirus software**.

Despite the advancement of traditional cybersecurity measures, ransomware attacks continue to escalate in frequency, sophistication, and impact. Current **signature-based** detection systems are inadequate against evolving ransomware variants, particularly **zero-day attacks** and **polymorphic** threats that modify their code to evade detection. The reactive nature of existing solutions often results in **delayed response times**, allowing ransomware to encrypt critical data before detection occurs.

The primary **challenges** in ransomware detection include:

- **Evasion Techniques**: Modern ransomware employs sophisticated obfuscation and encryption methods to bypass traditional security measures.
- **Zero-Day Threats**: New ransomware strains that have not been previously identified cannot be detected by signature-based systems.
- **Time-Critical Detection**: The rapid encryption process of ransomware leaves minimal time for detection and response.
- **False Positives**: Existing systems often generate high false positive rates, leading to alert fatigue and reduced effectiveness.
- **Behavioral Complexity**: Ransomware behavior patterns are becoming increasingly complex and difficult to distinguish from legitimate system operations.

1.4 Problem Solution

This project proposes an intelligent ransomware detection and prevention system that addresses the limitations of traditional approaches through the integration of advanced **AI** and **ML** techniques. The solution encompasses multiple detection layers and leverages both static and dynamic analysis methods.

-Key Components of the Proposed Solution:

- **Multi-Layered Detection Architecture:** The system employs a comprehensive approach combining static analysis (examining file characteristics without execution), dynamic analysis (monitoring runtime behavior), and network traffic analysis to provide robust detection capabilities.
- **Machine Learning Integration:** Advanced **ML** algorithms including Random Forest (**RF**), **XGBoost**, Support Vector Machines (**SVM**), and K-Nearest Neighbors (**KNN**) are utilized to identify ransomware patterns and behaviors that traditional signature-based systems miss.
- **Deep Learning Enhancement**: Convolutional Neural Networks (**CNN**) and Recurrent Neural Networks (**RNN**) are implemented to detect complex patterns in ransomware behavior and network traffic, providing superior accuracy in identifying novel threats.
- **Real-Time Monitoring:** The system provides continuous monitoring and real-time threat detection, enabling immediate response to ransomware activities before significant damage occurs.
- **Adaptive Learning:** The solution incorporates continuous learning capabilities, allowing the system to adapt to new ransomware variants and attack techniques automatically.
- **Low False Positive Rate:** Through advanced feature engineering and ensemble learning techniques, the system minimizes false positives while maintaining high detection accuracy.

1.5 Project Management

1.5.1 Team Members

1- Esraa Salah Hassan Ahmed Shalan

2- Nour Mohamed Morshed El-Shafie Nasr

3- Ahmed Abd El-Aziz Ali Abd El-Aziz El-Saadany

4- Ahmed Bassam Refaat Gaafar Ali

5- Ahmed Ibrahim El-Sayed Ahmed Abdullah

6- Abdullah Alaa Moukhtar Mohamed

7- Mohamed Ahmed Yousef Salama

8- Abdullah Anas Abdullah El-Gamal

1.5.2 Task Definition

The Team tasks can be Summarized as the following:

1. **Literature review:** All Team
2. **Machine Learning Models:**

   a. SVM: Nour Mohamed

   b. RF: Ahmed Bassam

   c. Xgboost: Ahmed Ibrahim

   d. KNN: Esraa Salah

3. **Deep Learning Models:**

   a. CNN: Nour Mohamed

   b. RNN: Abdullah Alaa & Ahmed Bassam

4. **Datasets:** All Team

5. **Implementation:** All Team.

6. **GUI:** Esraa Salah, Abdullah Alaa, Mohamed Ahmed & Abdullah Anas

7. **Testing:** All Team

8. **Conclusion & Future work:** Mohamed Ahmed & Abdullah Anas

9. **Documentation:** Nour Mohamed & Ahmed Bassam

10. **Presentation:** Ahmed Ibrahim & Esraa Salah

The following Table 1.2 describes the sequence of tasks taken and implemented in this project and how it is handled between the team members.

Table 1. 2 Time Scheduling

#	Duration	Task	Assigned To
1	**Wk: 1 – 2**	**Hands on** the tools (including setup of the platform carefully & efficiently) to be used in building required applications. Understanding the cyber-security (**CS**) concepts including its importance, types of attacks, methods to protect &/or prevent attacks.	**All Students**
	Wk: 3 – 4	**A literature review** on CS classifications and usage of different practical systems. The applications of machine learning in cyber security.	**All Students**
2	**Wk: 5 – 6**	**Literature review** for different approaches used for Ransomware detection **RD** using traditional **ML** algorithms (e.g. statistics-based techniques). – Training & Testing phases. Also, Literature review for different approaches using Deep Learning.	**All Students**
3	**Wk: 7 – 9**	**Analysis** & detailed description of the system including the System block diagrams, function diagrams, main features used, and the classification algorithm used. This Ransomware detection & protection **RDP** system is based of **Deep Learning Models.**	**All Students**
4	**Wk: 10 – 11**	**Data Acquisition and Preprocessing** Gather and curate a diverse dataset of benign and malicious samples. Preprocess the data, ensuring data quality, consistency, and feature extraction.	

5	**Wk: 12 – 13**	**Feature Engineering** Conduct in-depth feature engineering to identify and extract relevant characteristics for ransomware detection. Experiment with feature selection techniques to optimize model performance.	**According to the Description Above**
6	**Wk: 14 – 15**	**Model Selection and Training** • Select and train various machine learning algorithms (e.g., **SVM**, Random Forest, **CNN**, **RNN**) on the preprocessed dataset. • Evaluate model performance using appropriate metrics and fine-tune hyperparameters.	**All Students According to the Description Above**
7	**Wk: 16 - 17**	**System Integration and Testing** • Integrate the best-performing model into a real-time system, ensuring seamless integration with existing security infrastructure. • Conduct rigorous testing and validation to assess the system's effectiveness in detecting and preventing ransomware.	
8	**Wk: 18 - 19**	**Deployment and Evaluation** • Deploy the system in a controlled environment and monitor its performance in real-world scenarios. • Collect and analyze performance data to identify areas for improvement and optimization.	**All Students According to the Description Above**
9	**Wk: 20 – 21**	**Continuous Improvement** • Regularly update the system with new ransomware samples and retrain models to adapt to evolving threats. • Conduct ongoing monitoring and evaluation to ensure the system's effectiveness and identify potential vulnerabilities.	

10	**Wk: 22 – 23**	Study one or more methods to enhance system performance (e.g. combining different features over one classifier or two classifiers) and using **pretrained** Models. Good comparison of the systems implemented & others used in study (e.g. used **snort** & implemented one)	**All Students According to the Description Above**
11	**Wk: 24**	**GUI** implementation and testing system robustness.	
12	**Wk: 25 – 27**	**Revising** the Project Document & Presentation	**All Students**

1.6 Gantt Chart

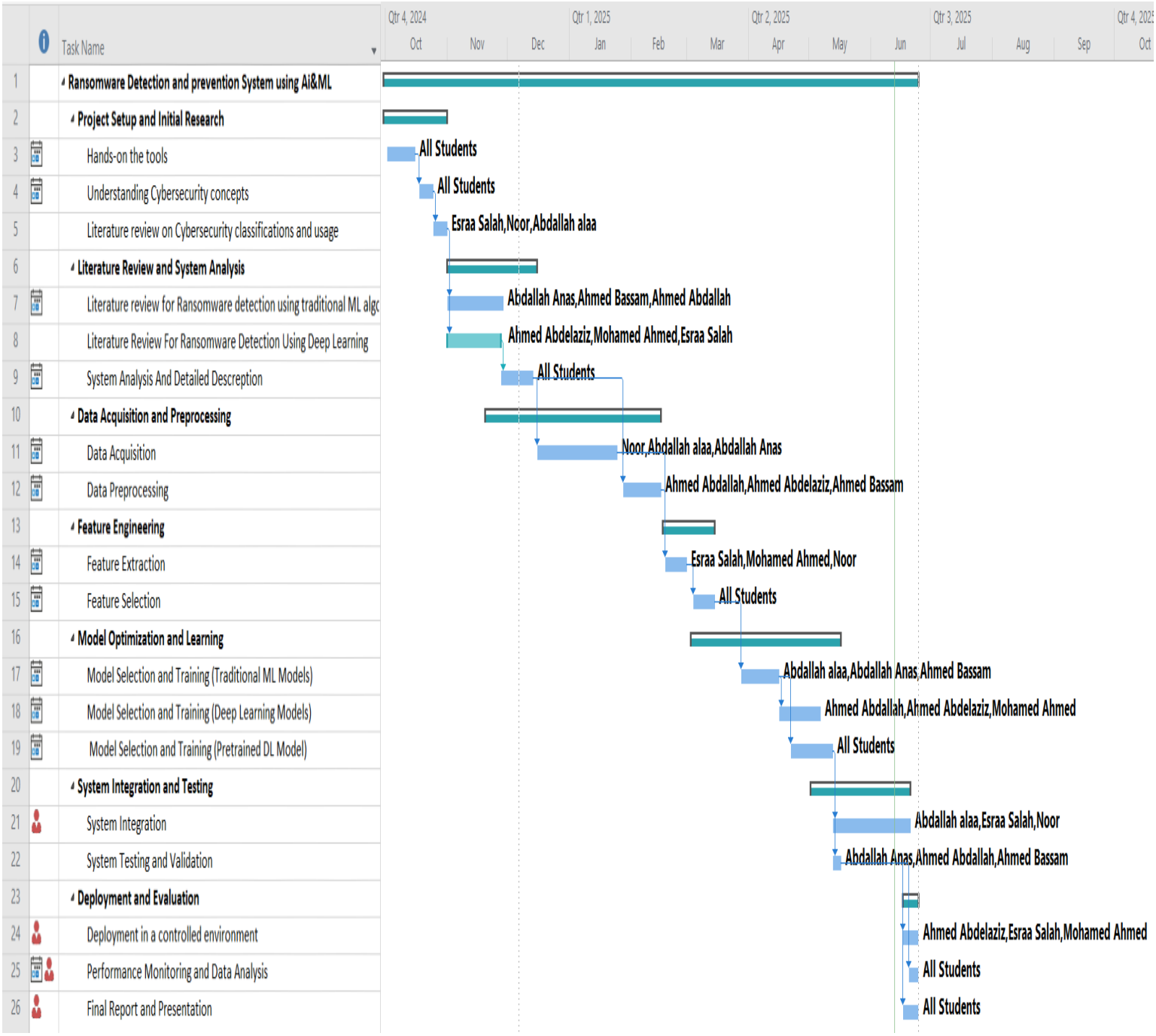
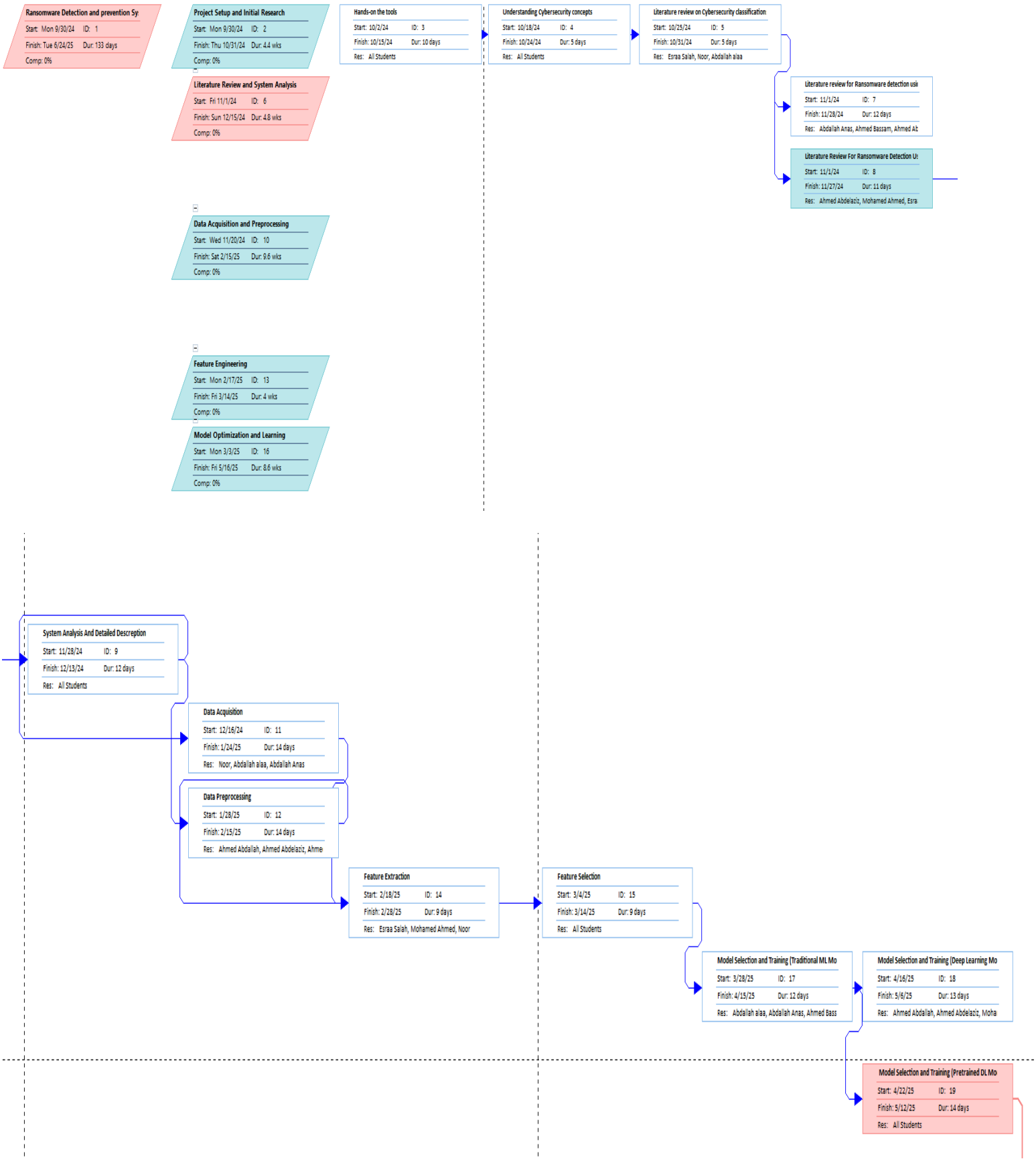


Figure 1. 6 Gantt Chart

1.7 Network Diagram



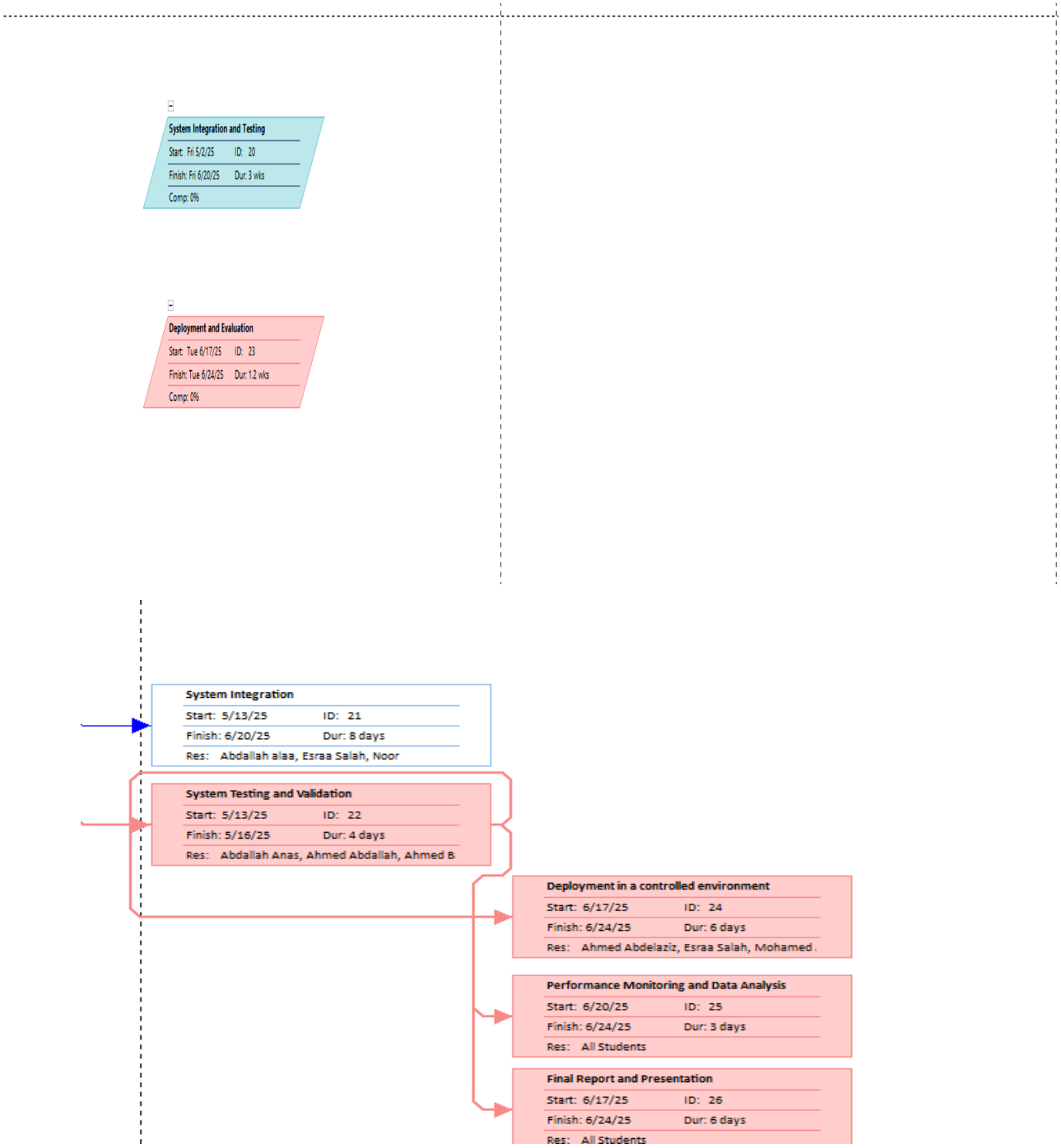


Figure 1. 7 Network Diagram

1.8 Task Timeline

	Tasks	Assigned To	Start Date	End Date	Duration	Predecessor
2	**Ransomware Detection and prevention System using Ai&ML**		**10/01/24**	**06/05/25**	**25 week**	
3	**Project Setup and Initial Research**		10/01/24	10/26/24	4 week	
4	Hands-on the tools	All Student	10/01/24	10/10/24	2 week	
5	Understanding Cybersecurity concepts	Esraa Salah,Noor Morshed,Ahmed Abdelaziz	10/11/24	10/17/24	1 week	5
6	Literature review on Cybersecurity classifications and usage	All Student	10/18/24	10/26/24	2 week	6
7	**Literature Review and System Analysis**		**10/27/24**	**12/09/24**	**6 week**	
8	Literature review for Ransomware detection using traditional ML algorithms	Ahmed Bassam,Ahmed Ibraheem,Abdallah Alaa	10/27/24	11/22/24	2 week	6
9	Literature Review For Ransomware Detection Using Deep Learning	All Student	11/22/24	12/06/24	2 week	6
10	System Analysis And Detailed Description	All Student	11/25/24	12/09/24	2 week	8,9
11	**Data Acquisition and Preprocessing**		**12/10/24**	**12/27/24**	**3 week**	
12	Data Acquisition	Mohamed Ahmed,Abdallah Anas,Esraa Salah	12/09/24	12/23/24	2 week	9
13	Data Preprocessing	All Student	12/13/24	12/27/24	2 week	11
14	**Feature Engineering**		**02/08/25**	**03/17/25**	**5 week**	
15	Feature Extraction	All Student	02/08/25	02/28/25	3 week	12
16	Feature Selection	Noor Morshed,Ahmed Abdelaziz,Ahmed Bassam	02/26/25	03/17/25	2 week	14
17	**Model Optimization and Learning**		**04/05/25**	**05/18/25**	**6 week**	
18	Model Selection and Training (Traditional ML Models)	Ahmed Ibraheem,Abdallah Alaa,Mohamed Ahmed	04/05/25	04/25/25	3 week	16
19	Model Selection and Training (Deep Learning Models)	All Student	04/15/25	05/05/25	3 week	16
20	Model Selection and Training (Pretrained DL Model)	All Student	04/25/25	05/18/25	3 week	16
21	**System Integration and Testing**		**04/30/25**	**05/22/25**	**3 week**	
22	System Integration	Abdallah Anas,Esraa Salah,Noor Morshed	04/30/25	05/12/25	2 week	18,19,20
23	System Testing and Validation	All Student	05/10/25	05/22/25	2 week	22
24	**Deployment and Evaluation**		**05/23/25**	**06/05/25**	**2 week**	
25	Deployment in a controlled environment	Ahmed Abdelaziz,Ahmed Bassam,Ahmed Ibraheem	05/23/25	06/02/25	2 week	23
26	Performance Monitoring and Data Analysis	Abdallah Alaa,Mohamed Ahmed,Abdallah Anas	05/24/25	06/03/25	1 week	25
27	Final Report and Presentation	All Student	06/01/25	06/05/25	1 week	26

Figure 1. 8 Task Timeline

1.9 Document Organization

Chapter 1: Presents the Introduction, which covers the ransomware threat, motivation, problem definition and solution, and project management.

Chapter 2: Presents the Literature Review, which establishes the theoretical background by reviewing the ransomware kill chain, its types, key features for detection, and the relevant Machine Learning and Deep Learning concepts.

Chapter 3: Surveys Similar Systems and Related Works, analyzing existing security tools and summarizing key academic research to provide context and highlight the state-of-the-art in ransomware detection.

Chapter 4: Details the System Implementation, covering the practical development of the detection system, the tools and technologies used, data preprocessing steps, and the process of building and training the machine learning models.

Chapter 5: Focuses on Evaluation and Testing, presenting the performance of the implemented system, outlining the testing methodology, and reporting results using key metrics like accuracy, precision, and recall.

Chapter 6: Provides the Conclusion and Future Work, summarizing the project's key findings and achievements, discussing the system's limitations, and proposing potential directions for future research and improvement.

Chapter 2
Literature Review

Chapter 2:
Literature Review

2.1 Introduction

This chapter provides a comprehensive review of the existing literature that forms the foundation for this project, focusing on two primary domains: **the technical characteristics of ransomware** and the **machine learning techniques used for its detection**. The purpose of this review is to establish the current state-of-the-art, identify key methodologies, and situate this project within the broader context of cybersecurity research.

The chapter begins by dissecting the ransomware threat itself. It will first explore the operational lifecycle of an attack by examining the stages of the **ransomware kill chain**. This is followed by a discussion of the major ransomware types and a detailed analysis of the **static**, **dynamic**, and **network** features that can be extracted from a program and its environment to serve as indicators of malicious intent.

Subsequently, the review transitions to the proposed solution space: **Artificial Intelligence**. It introduces the principles of Machine Learning (**ML**) and Deep Learning (**DL**) in the context of cybersecurity. The discussion will focus on **supervised** learning and provide an overview of several prominent classification models. Finally, the chapter addresses crucial concepts in model development and evaluation, including the bias-variance tradeoff, overfitting, and underfitting, which are essential for building a robust and reliable detection system.

To provide a complete picture of the defense landscape, the chapter will also address the role of traditional mechanisms like **firewalls** and explore the unique challenges and attack vectors associated with ransomware. This structured review establishes the necessary theoretical background for the methodology and implementation detailed in the subsequent chapters.

2.2 The Ransomware Attack Lifecycle: The Kill Chain

To develop an effective defense, it is crucial to understand the sequence of actions attackers take to execute a successful ransomware attack. This sequence is often modeled as a "**kill chain**," a concept from military strategy adapted for cybersecurity. As illustrated in Figure 2.1, the ransomware kill chain consists of several distinct stages, each representing an opportunity for detection and intervention.

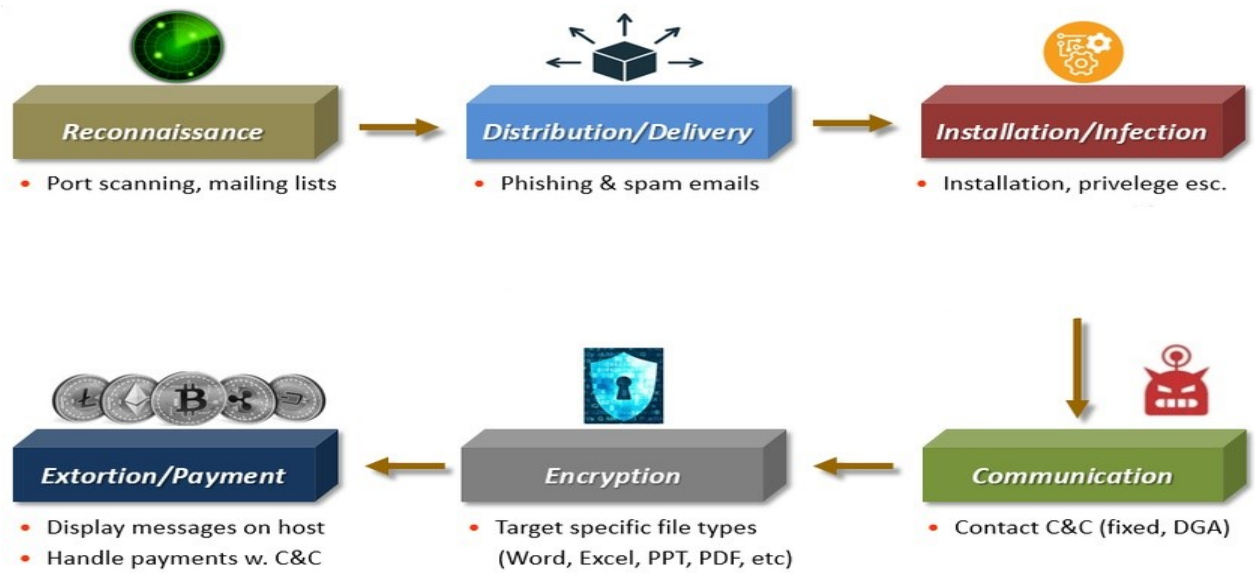


Figure 2. 1 Ransomware Kill Chain [7]

-The typical stages of the ransomware kill chain are as follows:

a. Reconnaissance

This is the preparatory phase where attackers gather intelligence on a potential target to identify vulnerabilities. They are not yet attacking the system but are actively probing for weaknesses.

Port Scanning: Attackers scan the target network to find open ports, which may indicate running services that are vulnerable to exploitation (e.g., an exposed Remote Desktop Protocol port).

Compiling Mailing Lists: Attackers collect email addresses of employees within an organization to prepare for a targeted phishing campaign.

b. Distribution/Delivery

In this stage, the attacker uses the gathered intelligence to deliver the malicious payload to the victim's system. This is the entry point of the attack.

Phishing & Spam Emails: This is the most common delivery vector. Attackers send emails that appear legitimate, containing malicious attachments (e.g., a Word document with a macro) or links to a website that downloads the ransomware.

c. Installation/Infection

Once the payload is delivered and the user has been tricked into executing it (e.g., by opening the attachment), the ransomware begins its infection process.

Installation: The malware installs itself on the host machine, often creating copies in system directories and establishing persistence by modifying the registry, ensuring it runs every time the system starts.

Privilege Escalation: The ransomware attempts to gain higher-level permissions (e.g., administrator rights). With elevated privileges, it can disable security software, access more files, and spread across the network more effectively.

d. Communication (Command & Control)

After successful infection, the malware establishes a connection to a server controlled by the attacker, known as a Command & Control (**C&C** or **C2**) server. This is a critical step for modern ransomware attacks.

Contact C&C: The ransomware "phones home" to signal a successful infection. It may download additional malicious components, receive specific instructions, or, most importantly, send the victim's unique information to the attacker and receive the public key that will be used for encryption. To evade blacklisting, attackers may use a fixed IP address or, more sophisticatedly, a Domain Generation Algorithm (**DGA**) to constantly create new, random domain names for the **C&C** server.

e. Encryption

This is the primary **destructive phase** of the attack, where the ransomware fulfills its main purpose.

Target Specific File Types: The malware systematically searches the victim's local drives and any connected network shares for valuable files. It specifically targets common file extensions like **.doc**, **.xlsx**, **.ppt**, **.pdf**, **.jpg** and database files, as encrypting these causes the most disruption. System files are usually left untouched, so the computer remains operational enough to display the ransom note.

f. Extortion/Payment

With the victim's data encrypted and inaccessible, the final stage of the attack begins.

Display Messages on Host: The ransomware displays a ransom note on the screen. This note informs the victim of the attack, provides proof of encryption, and gives instructions on how to pay the ransom, typically in a cryptocurrency like Bitcoin for anonymity.

Handle Payments with C&C: The ransom note often directs the victim to a payment portal on the dark web. The C&C server is used to verify the payment and, in theory, provide the victim with the decryption key.

Understanding this chain is fundamental to designing a robust detection system. Intervention at earlier stages such as identifying anomalous file I/O during the Installation phase or blocking C&C traffic during the Communication phase can prevent the catastrophic data loss of the Encryption phase. [7]

2.3 Ransomware Taxonomies and Attack Strategies

As introduced in Chapter 1, ransomware is fundamentally classified by its core function into two primary categories: Locker and Crypto ransomware. However, a more detailed taxonomy is required to understand the full scope of the modern threat. This section expands that initial classification, exploring ransomware not only by its technical function but also by its attack vector and its increasingly sophisticated extortion tactics.

2.3.1 Functional Classification

This classification is based on what ransomware does to the victim's system.

- **Crypto-Ransomware:** This is the most prevalent and destructive form of ransomware. It operates by searching for and encrypting a victim's files (documents, images, databases), rendering them completely inaccessible without a unique decryption key. The attackers then demand a ransom in exchange for this key. Due to the strength of modern encryption algorithms, recovery without the key is often impossible, making this a severe threat to data integrity.
- **Locker Ransomware:** This type of ransomware does not encrypt files. Instead, it locks the victim out of their device or system, preventing access to the desktop and applications. It typically displays a full-screen message demanding payment to unlock the system. While disruptive, Locker ransomware is generally considered less severe because the underlying data is not damaged, and technical workarounds, such as restarting the computer in **Safe Mode**, can sometimes resolve the issue.

2.3.2 Classification by Attack Vector and Evasion Technique

This classification focuses on how ransomware achieves its goal and evades defenses.

- **Zero-Day Ransomware:** This is not a specific type of ransomware but rather a classification based on the exploit it uses. A zero-day attack leverages a previously **unknown vulnerability** in software or an operating system for which no patch or defense is available. Attackers who discover such a flaw can launch a highly effective ransomware campaign, as traditional signature-based security systems will not recognize the threat. These attacks can cause widespread damage before the vulnerability is identified and mitigated.

- **Adversarial Ransomware:** This represents an emerging and sophisticated threat specifically designed to deceive and evade modern, AI-based security systems. Adversarial attacks employ machine learning techniques to subtly **alter** the ransomware code or **behavior**. For instance, they might add "noise" (irrelevant data) to an executable file or manipulate its decision-making process just enough to be misclassified as benign by a detection model, allowing it to bypass advanced defenses.

2.3.3 Evolving Extortion Strategies

Beyond the technical attack, the business model of ransomware has evolved into a multi-layered extortion ecosystem. This progression demonstrates how attackers maximize pressure on their victims to ensure payment. As illustrated in Figure 2.2, these strategies have grown progressively more aggressive over time.

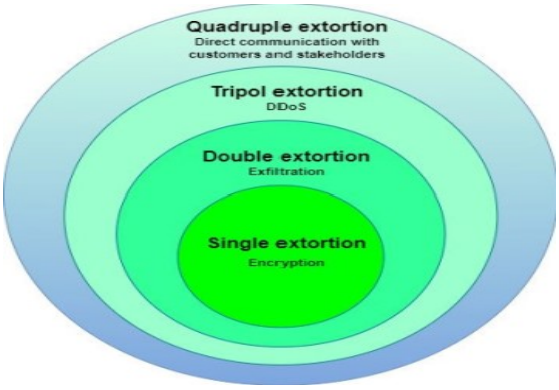


Figure 2. 2 Extortion Strategies [8]

-The primary extortion strategies include:

- **Single Extortion:** This is the original ransomware model. The attacker encrypts the victim's files and demands a ransom solely for the decryption key. The entire value proposition is based on restoring access to encrypted data. Classic ransomware families like **WannaCry** and **CryptoLocker** primarily used this strategy.

- **Double Extortion:** Attackers added a second layer of leverage. Before encrypting the files, they first exfiltrate (**steal**) a copy of the sensitive data. They then threaten to leak this data publicly on the internet if the ransom is not paid. This tactic pressures organizations who may have backups, as restoring from a backup does not prevent the reputational and legal damage of a data breach.

- **Triple Extortion:** To counter organizations that might be willing to accept both encryption and a data leak, attackers introduced a third threat: a Distributed Denial-of-Service (**DDoS**) attack. In this scenario, if the victim refuses to pay, the attackers overwhelm the organization's servers and network with traffic, disrupting their operations and making their public-facing services unavailable.

- **Quadruple Extortion:** This is the most recent and aggressive strategy. If the primary target still refuses to pay, the attackers pivot and begin contacting the victim's customers, partners, or other third parties. They inform these associated entities about the data breach, using them as another point of pressure and causing significant supply-chain and reputational damage to the original victim. The attack on **Quanta**, which led to threats against its client, Apple, is a prime example of this tactic. [8]

2.4 Feature Engineering for Ransomware Detection

The ability to automatically detect ransomware relies on identifying its unique digital footprint. This is achieved through feature engineering, the process of extracting specific, measurable characteristics (features) from software and its environment. These features serve as the input for machine learning models to learn the patterns that differentiate malicious programs from benign ones. In the context of ransomware analysis, these features are broadly categorized into three fundamental types: **static**, **dynamic**, and **network**, each providing a different perspective on the software's nature and intent.

2.4.1 Static Features

Static features are attributes of an executable file that can be extracted and analyzed without executing the program. This form of analysis is fast and low-risk, as the potential malware remains dormant. The process typically involves parsing the structure of the file to extract meaningful data points.

This includes analyzing components like the file header (e.g., the PE header in Windows), which contains a wealth of information about the file structure and dependencies.

-Key static features include:

File Metadata: Basic properties such as the file's size, entropy (a measure of randomness, which is often high in encrypted or packed files), and compilation formats.

String Analysis: Extracting human-readable text from the program's binary code. The presence of suspicious strings—such as targeted file extensions (.doc,.pdf), cryptocurrency terms (bitcoin), or ransom note phrases can serve as strong indicators.

Imported Functions (API Calls): A list of functions that the program intends to use from system libraries (e.g., Windows DLLs). A program importing functions for both cryptography (CryptEncrypt) and file system manipulation (DeleteFile, SetFileAttributes) is highly suspicious.

Executable Header Information: Data from the file header (e.g., PE header in Windows) which defines its structure, section names, and required resources. Abnormalities in this structure can suggest a packed or malicious file.

-Additional Static Features:

Byte-level n-grams: Analyzing short sequences of bytes in the executable. This can identify common code patterns used by specific malware families without needing to understand the code's full context.

Resource Section Analysis: Examining the resources embedded in the file, such as icons, manifests, and version information. Malware may use generic or stolen resources, which can be an indicator.

Digital Signature Validation: Checking if the executable is digitally signed by a trusted authority. The absence of a valid signature, or the presence of a revoked one, is a significant red flag.

2.4.2 Dynamic Features

Dynamic features are collected by observing a program's behavior while it is running. This approach is crucial for unmasking sophisticated malware that can hide its true intent during static analysis. The process involves executing the file within a secure and isolated sandbox environment, which monitors and logs every action the program takes without risk to the host system.

-The most significant dynamic features for ransomware detection include:

File System Operations: Monitoring patterns of high-volume file activity. A classic ransomware behavior is to rapidly read, modify (encrypt), and rename thousands of files, which creates a distinct and detectable I/O pattern.

Registry Modifications: Tracking changes to the system registry. Ransomware frequently creates or modifies registry keys to establish persistence, ensuring it will run automatically every time the system starts.

Process and Thread Activity: Observing the creation of new processes or threads. A program might spawn a process to delete volume shadow copies (vssadmin.exe) or terminate processes associated with security software.

API Call Sequences: Analyzing the specific order and frequency of API calls made during execution. A logical sequence of calls for finding files, encrypting their contents, and then deleting the originals is a definitive behavioral fingerprint of crypto ransomware.

-Additional Dynamic Features:

Memory Dump Analysis: Analyzing the contents of the process's memory during execution. This can reveal injected code, decrypted malicious payloads, or sensitive data that the malware is trying to steal.

Anti-Analysis Checks: Monitoring for behaviors specifically designed to detect or evade sandboxes and debuggers, such as checking for virtual machine artifacts or timing checks.

System Call Hooking: Intercepting low-level system calls to get a granular view of the program's interaction with the operating system kernel, which can uncover stealthy behaviors missed by higher-level API monitoring.

2.4.3 Network Features

Network features are characteristics derived directly from analyzing data packets and traffic flows as they move across a network. This analysis provides a different and often earlier vantage point for detection, as it can identify threats during the delivery or Command-and-Control (C&C) phases of an attack before they are fully established on a target machine.

-Key network features include:

Traffic Flow Characteristics: Analyzing the metadata of network connections, such as the source and destination IP addresses, ports used, connection duration, and the volume of data sent and received. Unusual patterns, like a connection to an unknown server followed by a large download, can indicate a payload of delivery.

Protocol Analysis: Inspecting the contents and structure of network protocols. This can reveal the use of non-standard protocols or anomalies in standard ones (e.g., **HTTP**, **SMB**) that are used to carry malicious payloads or exploit vulnerabilities.

Exploit Signatures: Identifying specific data patterns within network traffic that match known exploits. This allows for the detection of attempts to compromise a system by targeting vulnerabilities in services like Remote Desktop Protocol (**RDP**) or Server Message Block (**SMB**).

Payload Analysis: Reassembling data packets to inspect the files or commands being transmitted. This allows for the detection of malicious executables, scripts, or commands being delivered over the network.

-Additional Network Features:

DNS Query Analysis: Monitoring Domain Name System (**DNS**) requests. This is critical for detecting connections to domains generated by a Domain Generation Algorithm (**DGA**), a common tactic used by botnets and C&C servers to evade blacklisting.

TLS/SSL Certificate Analysis: Examining the digital certificates used in encrypted connections. Communications with servers using self-signed, expired, or otherwise suspicious certificates can be a strong indicator of malicious C&C traffic.

Traffic Timing and Periodicity: Analyzing the timing between network connections. Regular, periodic "heartbeat" communications to a C&C server are a classic indicator of an established backdoor or botnet infection.

These three categories of features provide a comprehensive foundation for building robust ransomware detection systems. While each category has its strengths and weaknesses, their combined use in a layered defense strategy allows for the identification of ransomware at different stages of an attack. [9]

2.5 The Shift to Machine Learning in Cybersecurity

The landscape of digital threats has evolved at a pace that has rendered many conventional security measures insufficient. Traditional cybersecurity defenses, which are predominantly reactive, operate by identifying threats based on pre-defined signatures or rules. While effective against known malware, these static systems are fundamentally ill-equipped to handle the dynamic and ever-changing nature of modern attacks.

They struggle particularly with zero-day exploits and polymorphic malware, which are designed specifically to have no existing signature, allowing them to bypass these legacy defenses with ease.

This is where Machine Learning (**ML**) represents a **paradigm shift**. Instead of explicitly programmed with rules on what is "bad," an ML system learns the underlying patterns and statistical properties that differentiate malicious software from benign software. By training vast datasets of examples, ML models can develop the ability to generalize and identify novel threats, making them an essential component of modern proactive cybersecurity.

2.6 Types of Machine Learning

Machine Learning is broadly categorized into three main paradigms, each defined by the nature of the data and the learning process. These categories are Supervised, Unsupervised, and Reinforcement Learning.

- **Supervised Learning:** The model learns from a dataset that is fully **labeled**. Each data point is tagged with the correct output or category. The goal is to learn a mapping function that can correctly predict the label for new, unseen data.

- **Unsupervised Learning:** The model learns from an **unlabeled** dataset. The goal is to discover hidden patterns, structures, or clusters within the data without any predefined outcomes.

- **Reinforcement Learning**: An agent learns by **interacting** with an environment. It receives rewards for performing correct actions and penalties for incorrect ones. The goal is for the agent to learn the best sequence of actions (a policy) to maximize its cumulative reward over time.

2.7 Machine Learning Models

A variety of supervised learning algorithms are well-suited for the classification of ransomware. The selection of a model often depends on the nature of the features, the size of the dataset, and the desired balance between performance and interpretability.

The following models are frequently utilized in cybersecurity research due to their proven effectiveness.

2.7.1 Support Vector Machine (SVM)

The Support Vector Machine is a powerful classification algorithm that operates by finding an optimal **hyperplane** that separates data points of different classes in a high-dimensional space. As illustrated in Figure 2.3, the "optimal" hyperplane is the one that maximizes the **margin**— the distance between the hyperplane and the nearest data points from each class. These closest points, which are critical in defining the position of the hyperplane, are called **support vectors**. By maximizing this margin, SVM creates a robust decision boundary that generalizes well to new data. [10]

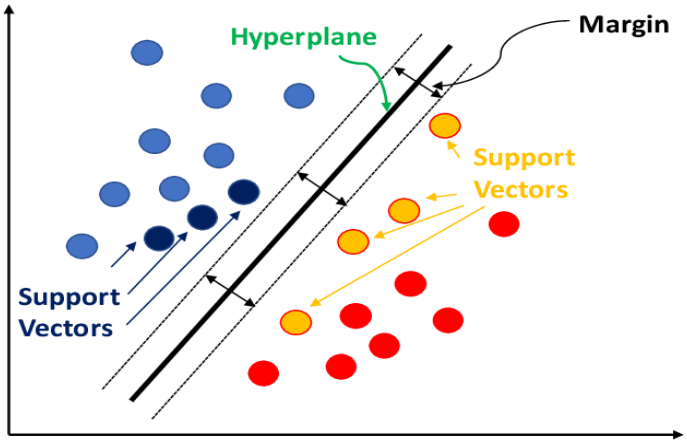


Figure 2. 3 SVM [11]

For cases where the data is not linearly separable, SVM uses a technique called the **kernel trick**. Kernels are mathematical functions that transform the data into a higher dimension where a linear separation becomes possible. As shown in Figure 2.4, data that is impossible to separate with a straight line in its original 2D space can be projected into a 3D space where a simple plane can easily divide the classes.

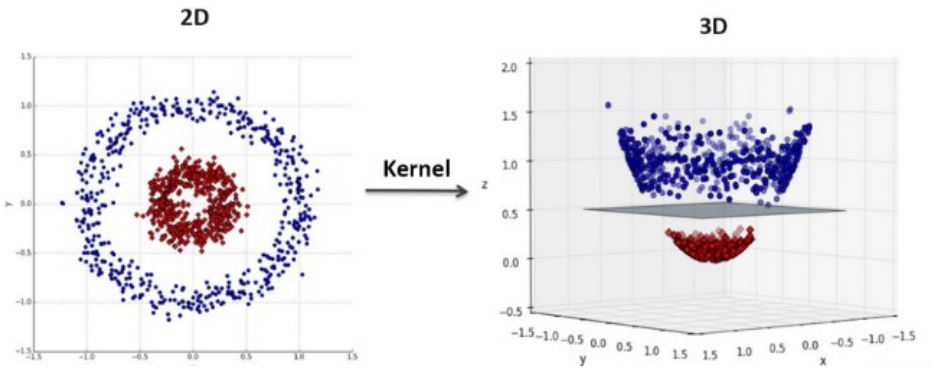


Figure 2. 4 Kernel Trick [11]

Key Parameters:

- **C:** The regularization parameter, which controls the penalty for misclassifying training examples.
- **kernel:** The function used to transform the data (e.g., 'linear', 'poly', 'rbf').
- **gamma:** A parameter for non-linear kernels that defines the influence of a single training example.

Advantages:

- Effective in high-dimensional spaces, making it suitable for datasets with many features.

- Memory is efficient as it only uses a subset of training points (the support vectors) in the decision function.
- Versatile due to the use of different kernel functions for linear and non-linear data.

Disadvantages:

- Does not perform well on very large datasets as the training time can be long.
- Less interpretable than models like Decision Trees; it's difficult to understand the contribution of individual features.
- Performance is highly dependent on the choice of the kernel and regularization parameter C. [10]

2.7.2 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple yet effective instance-based learning algorithm. Unlike other models, KNN does not learn an explicit function during a training phase. Instead, it memorizes the entire training dataset. This approach is often called "lazy learning." When a new, unclassified data point is introduced, the KNN algorithm calculates the distance between this new point and every point in the training data. It then identifies the 'k' nearest neighbors, as shown in Figure 2.5, and assigns the new point to the class that is most common among those neighbors.

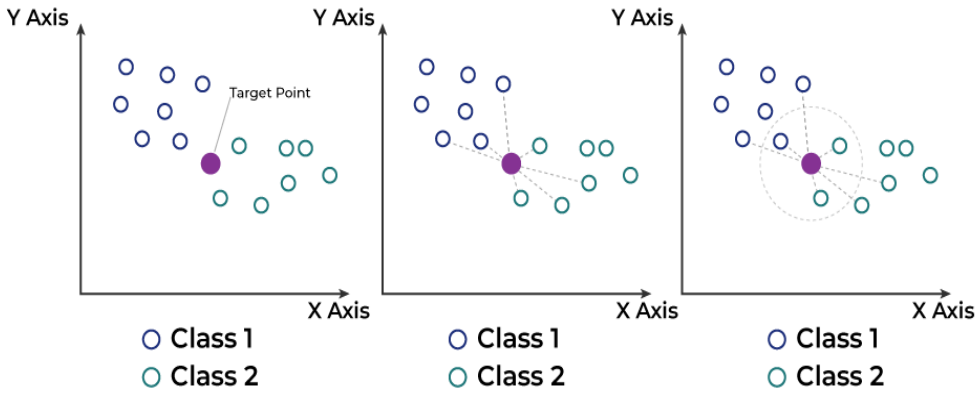


Figure 2. 5 KNN [12]

The choice of 'k' is critical: a small 'k' can make the model sensitive to noise and outliers, while a large 'k' can oversimplify the decision boundary, potentially misclassifying the sample.

Key Parameters:

- **n_neighbors:** The integer value of 'k'.
- **metric:** The formula used to calculate distance (e.g., 'euclidean', 'manhattan').
- **weights:** Determines if all neighbors are weighted equally or by distance.

Advantages:

- Very simple to understand and implement.
- No training phase is required; the model is built at prediction time.
- Adapts easily to new data without retraining the entire model.

Disadvantages:

- Computationally expensive and slow during prediction.
- Performance degrades with high-dimensional data (the "curse of dimensionality").
- Highly sensitive to irrelevant features and the scale of the data. [10]

2.7.3 Random Forest (RF)

As depicted in Figure 2.6, Random Forest is a powerful ensemble learning method that constructs a large number of individual decision trees and combines their outputs for a final prediction. For a classification task, the final class is determined by a majority vote among all the trees in the forest. The model's strength comes from two key sources of randomness introduced during the training process:

Bagging (Bootstrap Aggregating): Each decision tree is trained on a different random subset of the original training data (sampled with replacement).

Feature Randomness: At each node in a tree, instead of considering all features for the best split, only a random subset of features is evaluated.

This dual-randomness strategy ensures that the individual trees are diverse and de-correlated, significantly reducing the risk of overfitting that a single decision tree would face.

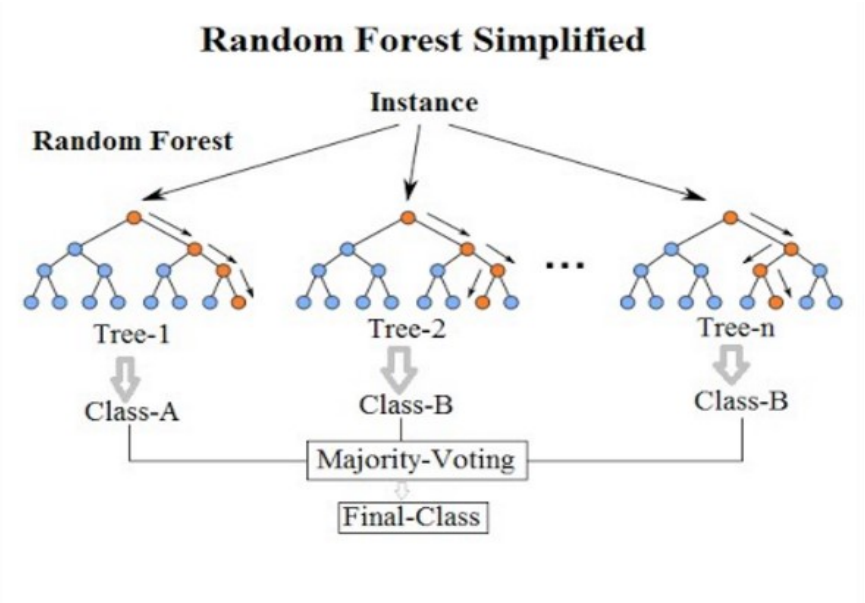


Figure 2. 6 RF [13]

Key Parameters:

- **n_estimators:** The number of decision trees to build in the forest.
- **max_depth:** The maximum depth allowed for any individual tree.
- **min_samples_split:** The minimum number of samples required to split an internal node.

Advantages:

- Highly robust against overfitting due to the use of multiple trees.
- Performs well on large datasets and can handle thousands of input features.
- Can provide estimates of feature importance, offering some interpretability.

Disadvantages:

- Can be a "black box," making it difficult to interpret the complex interactions between trees.
- Training can be slower and more resource-intensive than single models. [10]

2.7.4 XGBoost (Extreme Gradient Boosting)

Extreme Gradient Boosting (XGBoost) is a highly efficient and scalable implementation of the gradient boosting algorithm. Like Random Forest, it is an ensemble method, but instead of building trees independently, it builds them **sequentially**. As shown in Figure 2.7, each new tree is trained specifically to correct the errors (known as residuals) made by the previous ensemble of trees. [14]

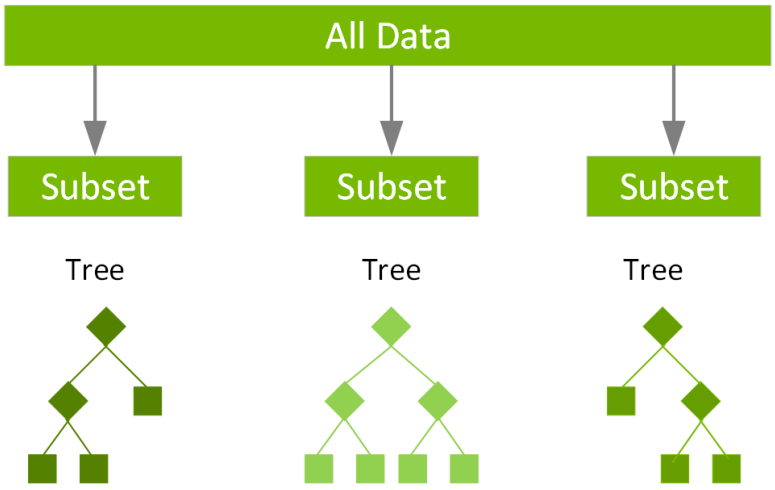


Figure 2. 7 XGBoost [11]

It does this by using a gradient descent algorithm to minimize the loss function as new models are added. This sequential, error-correcting process allows **XGBoost** to create highly accurate models.

Furthermore, it includes built-in **regularization** techniques (L1 and L2) to penalize model complexity, which is a powerful defense against overfitting, making it a go-to algorithm for structured data competitions and real-world applications.

Key Parameters:

- **learning_rate:** Shrinks the contribution of each tree to prevent overfitting.
- **n_estimators:** The number of boosting rounds or trees.
- **max_depth:** The maximum depth of a tree.
- **subsample:** The fraction of training data to be randomly sampled for each tree.

Advantages:

- Extremely high predictive accuracy, often outperforming other algorithms.
- Fast execution speed due to parallel processing and other optimizations.
- Includes built-in regularization (L1/L2) to prevent overfitting.

Disadvantages:

- Can be even more of a "black box" than Random Forest.
- Has many parameters that require careful tuning for optimal performance.
- Can be sensitive to outliers. [14]

2.8 Deep Learning

Deep Learning (DL) is a specialized subfield of machine learning based on Artificial Neural Networks (ANNs) with multiple layers between the input and output layers, which is why they are called "deep." The defining advantage of DL models over their traditional ML counterparts is their ability to perform automatic feature learning. Instead of relying on a human expert to perform feature engineering, a deep learning model can autonomously learn hierarchical representations of features directly from raw input data. For example, it can learn to identify **complex**, high-level patterns in a program's binary code or a sequence of its behaviors without being explicitly told what to look for.

-Two prominent DL architectures are particularly relevant for analyzing malware:

2.8.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (**CNN**) is a type of deep learning model designed to automatically learn spatial hierarchies of features from data. It achieves this using a series of specialized layers that are built upon each other to extract progressively more complex patterns. A complete CNN architecture for classification typically includes the following layers in sequence:

Input Layer: This layer defines the shape of the input data. For a 1D-CNN, this would be a sequence of a specific length, where each item in the sequence is a vector of numerical features.

Convolutional Layer: This is the core building block of the CNN. It employs a set of learnable filters (kernels) that slide across the input sequence. Each filter acts as a local pattern detector, activating when it encounters a specific motif (e.g., a particular short sequence of network events) in the data. The output is a feature map that highlights where these patterns were found.

Pooling Layer: Often placed after a convolutional layer, the pooling layer reduces the length of the feature map. The most common type, Max Pooling, selects the maximum value from a small window of the feature map. This makes the model more computationally efficient and helps it recognize a pattern regardless of its exact position in the sequence.

Flatten Layer: This layer serves as a bridge between the convolutional layers and the dense layers. It takes the multi-dimensional output of the preceding layers (the feature maps) and collapses it into a single, one-dimensional vector.

Dense (Fully-Connected) Layer: This is a standard neural network layer where every neuron is connected to every neuron in the previous layer. One or more dense layers are used to perform high-level reasoning on the features extracted by the convolutional layers.

Output Layer: The final dense layer, which produces the classification output. For a binary classification task like ransomware detection, this layer typically has a single neuron with a sigmoid activation function to output a probability between 0 and 1.

-**Key Parameters:**

filters: The number of output filters (kernels) in the convolution.

kernel_size: The length of the 1D convolutional window.

strides: The step size of the convolution.

padding: "valid" (no padding) or "same" (output has the same length as input).

activation: The activation function to use, typically 'ReLU'. [15]

Application as a 1D-CNN in this Project:

In this project, a 1D-CNN is used to analyze sequences of numerical features representing system or network behavior. As illustrated in Figure 2.8, the complete architecture is implemented: the model takes a sequence of feature vectors as input, passes them through convolutional and pooling layers to detect significant local patterns, flattens the result, and uses dense layers to make a final classification of whether the behavior is malicious or benign.

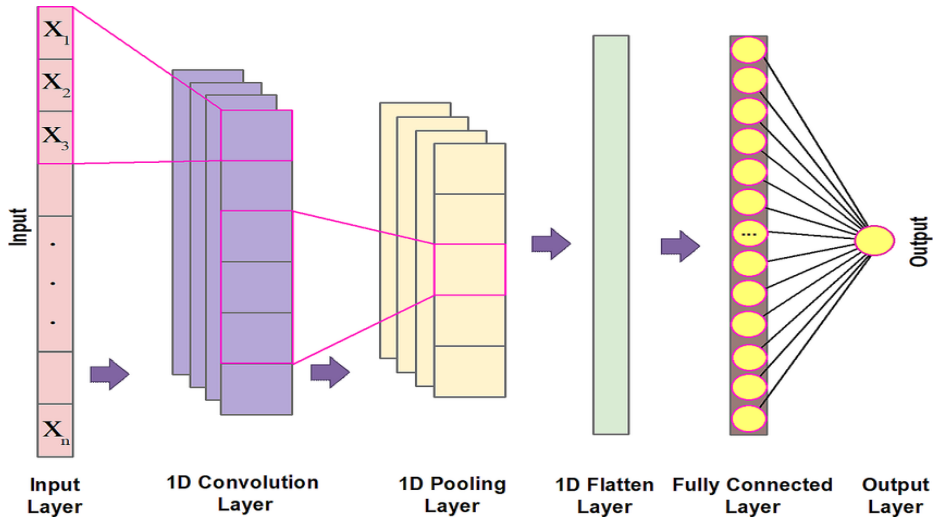


Figure 2. 8 CNN Architecture [12]

2.8.2 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a class of neural networks specifically designed for processing sequential data where the order of information is important. Its defining feature is a feedback loop within its architecture.

This loop allows the network to maintain an internal memory, or hidden state, which captures information about the preceding elements in a sequence, thus influencing the processing of the current element. A complete RNN architecture for classification is composed of the following layers:

Input Layer: This layer defines the shape of the input data, which for this application is a sequence of a specific length where each item is a vector of numerical features.

Recurrent Hidden Layer: This is the core component of the RNN. It iterates through the input sequence one step at a time. At each step, it updates its internal hidden state by combining the current input with the hidden state from the previous step. This process allows the network to build a contextual understanding of the entire sequence.

Output Layer: After the recurrent layer has processed the entire sequence, its final hidden state (which contains a summary of the whole sequence) is passed to a final dense layer for classification. For a binary task like ransomware detection, this layer typically has a single neuron with a sigmoid activation function to produce the final probability.

Key Parameters:

- **units**: The dimensionality of the hidden state vector.
- **activation:** The activation function for the recurrent step (e.g., 'tanh').
- **return_sequences:** A Boolean that determines whether the layer returns the hidden state for every time step or only the final one (for classification, this is usually False).
- **recurrent_dropout:** A regularization technique to prevent overfitting on the recurrent connections. [15]

Application in this Project:

For this project, an RNN architecture is employed to model the temporal dynamics of the input feature sequences. As depicted in the high-level architecture of Figure 2.9, the model's Input Layer receives the sequence of feature vectors. The Recurrent Hidden Layer then processes these events chronologically, building a stateful memory of the sequence's history. This allows it to understand the context of each event. The final representation from the hidden layer is then passed to the Output Layer, which makes the final ransomware vs. benign classification.

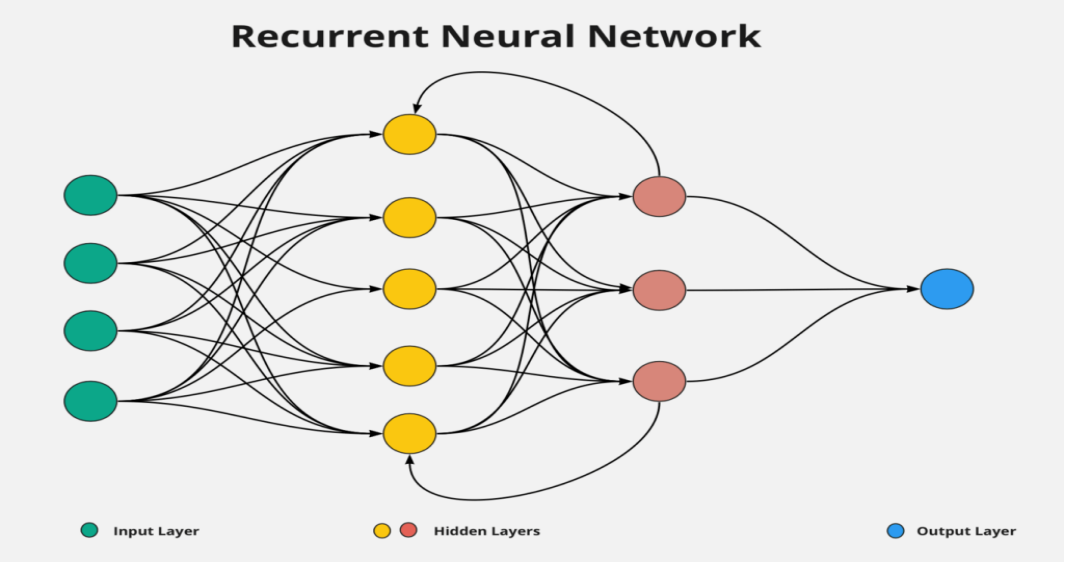


Figure 2. 9 RNN Architecture [12]

2.9 Differentiating Machine Learning & Deep Learning

While DL is a part of ML, their practical application and characteristics differ significantly, particularly in their approach to data and features.

Table 2. 1 ML & DL Comparison [16]

Feature	ML	DL
Feature Engineering	Requires manual, expert-driven feature extraction.	Learns features automatically from raw data.
Data Requirement	Can perform well with smaller datasets.	Requires very large datasets for optimal performance.
Hardware	Can typically run on standard CPUs.	Requires powerful GPUs for efficient training.
Interpretability	Offers higher model transparency.	The logic of models like Decision Trees can be followed and understood.
Training Time	Generally faster to train.	Can take hours, days, or even weeks to train.

2.10 Fundamental Concepts in Model Training

Building an effective machine learning model involves more than just selecting an algorithm and feeding it data. The core objective is to train a model that achieves strong generalization that has the ability to make accurate predictions on new, unseen data, not just the data it was trained on. A model that only memorizes the training data is useless in a real-world scenario like ransomware detection. This section delves into the fundamental principles and challenges that govern the training process, beginning with the central concepts of **bias** and **variance**.

2.10.1 Bias: The Error of Oversimplification

In machine learning, bias refers to the error introduced by approximating a real-world problem, which may be complex, with a model that is overly simplistic. It represents the inherent "prejudice" or assumptions of a model. A model with high bias pays very little attention to the training data and makes sweeping, simplistic assumptions about it. It **fails** to capture the true underlying **patterns** and relationships between the features and the target output.

Analogy: A detective who, from the start, assumes that the simplest explanation is always the correct one will ignore complex but crucial evidence, leading to consistently wrong conclusions.

Technical Impact: A high-bias model is unable to capture the complexity of the data, resulting in a systematic error. It will consistently misclassify samples regardless of variations in the training data. [17]

2.10.2 Variance: The Error of Over-Complication

Variance, in contrast, refers to the error introduced by a model that is excessively complex and overly sensitive to the small fluctuations and noise in the training data. A model with high variance pays too much attention to the training data, essentially **memorizing** both the genuine signal and the random noise. As a result, its performance can change dramatically if it is trained on a slightly different subset of data.

Analogy: A detective who weaves an elaborate conspiracy theory to account for every single piece of evidence, including irrelevant details like the color of a witness's shoes.

This theory will be perfect for that specific crime scene but will fall apart completely when applied to a new case.

Technical Impact: A high-variance model fits the training data almost perfectly but fails to generalize to new data. Its predictions are unstable and can vary wildly. [17]

2.10.3 The Bias-Variance Tradeoff

The concepts of bias and variance are not independent; they are locked in an inverse relationship known as the Bias-Variance Tradeoff. This is one of the most fundamental challenges in supervised learning.

- **The Relationship:** Increasing a model's complexity will typically decrease its bias but increase its variance. Conversely, decreasing a model's complexity (to reduce variance) will typically increase its bias. You cannot simply minimize both at the same time.
- **The Goal**: The ultimate goal is not to achieve zero bias and zero variance, which is impossible. The goal is to find the optimal level of model complexity that strikes a balance between the two, thereby minimizing the model's total error on unseen data. This "sweet spot" represents the best possible generalization of performance.

This tradeoff explains why a simple model can be just as ineffective as an overly complex one. The art of machine learning lies in navigating this tradeoff to build a model that is "just right." [17]

2.10.4 Underfitting: The Symptom of High Bias

Underfitting occurs when a model is too simple to capture the underlying structure of the data, a concept visualized in Figure 2.10. It is the practical result of a model suffering from **high bias**. An underfit model has not learned the patterns in the training data and, as a result, will perform poorly on both the data it was trained on and new, unseen test data.

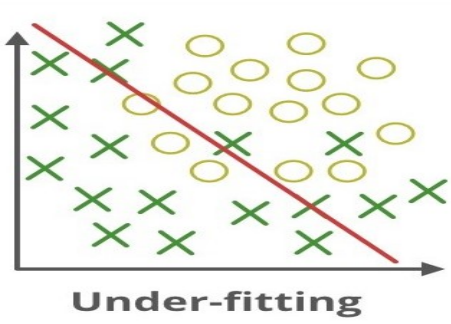


Figure 2.10 Underfitting [12]

Key Characteristics:

- High training error.
- High testing/validation error.
- The model is too simplistic.

Common Causes:

- Choosing a model that is not complex enough for the data (e.g. a linear model for a non-linear problem).
- Not providing the model with enough informative features (insufficient feature engineering). [12]

2.10.5 Overfitting: The Symptom of High Variance

Overfitting is the opposite problem. It occurs when a model learns the training data too well, to the point where it begins to memorize the noise and random fluctuations within it. As depicted in Figure 2.11, it is the practical result of a model suffering from **high variance**. An overfit model will show exceptionally high accuracy on the training data but will fail to generalize, performing poorly on new, unseen test data. The large gap between training performance and testing performance is the classic signature of overfitting. [12]

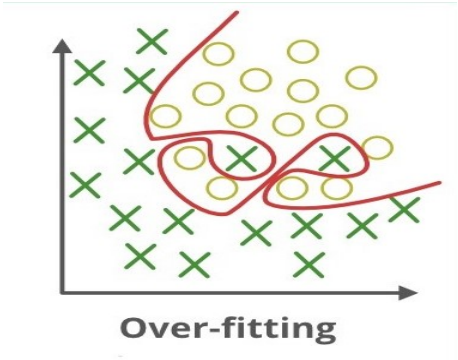


Figure 2.11 Overfitting [12]

Key Characteristics:

- Very low training error.
- High testing/validation error.
- The model is too complex and has memorized the data.

Common Causes:

- Using a model that is too powerful or complex for the amount of data available.
- Training the model for too many iterations (epochs).
- Having too many features for the number of training examples. [12]

2.10.6 The Well-Generalized Model (Best Fit)

The ideal model is one that successfully navigates the bias-variance tradeoff. This well-generalized or best-fit model, illustrated in Figure 2.12, is complex enough to capture the underlying structure of the data but not so complex that it starts learning the noise. It has **low bias** and **low variance**. Such a model will demonstrate strong performance on both the training data and, most importantly, on new, unseen test data. It has effectively learned the true signal from the data and can make reliable predictions.

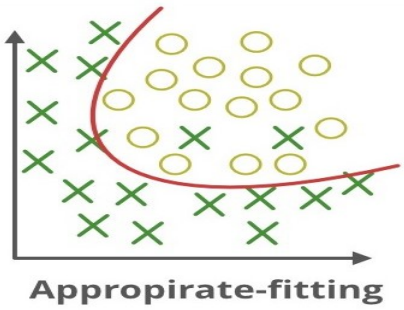


Figure 2. 12 Good-fit [12]

Key Characteristics:

- Low training error.
- Low testing/validation error.
- The gap between training and testing error is minimal.

2.10.7 Addressing the Tradeoff: Practical Techniques

To achieve a **well-generalized** model and actively combat overfitting, several standard techniques are employed during the training and evaluation process.

Regularization: This is a set of techniques that explicitly penalizes model complexity to prevent the model's parameters from becoming too large, which is a common cause of overfitting. A regularization term is added to the model's loss function, creating a constraint that favors simpler models. The two most common types are:

- **L1 Regularization (Lasso):** Adds a penalty equal to the absolute value of the magnitude of the coefficients. It can shrink some coefficients to exactly zero, effectively performing automatic feature selection. [11]
- **L2 Regularization (Ridge):** Adds a penalty equal to the square of the magnitude of the coefficients. It forces coefficients to be small but does not typically shrink them to zero. This is a key feature in models like XGBoost and is a standard component in deep learning. [11]

Cross-Validation: In this technique, the training data is divided into 'k' equal-sized subsets, or "folds." The model is then trained 'k' times. In each iteration, one fold is held out as a validation set, and the model is trained on the remaining k-1 folds. The final performance score is the average of the scores from all 'k' iterations. This process ensures that every data point is used for validation exactly once, leading to a much less biased evaluation of the model's true performance on unseen data. [18]

Dropout (for Neural Networks): Dropout is a powerful regularization technique specifically for deep learning models. During each training iteration, a random fraction of the neurons in a layer are temporarily "dropped" or ignored. This forces the remaining neurons to learn more robust and redundant features, preventing any single neuron from becoming too specialized or reliant on the outputs of others. This acts as a way of training a large number of different network architectures simultaneously, which significantly reduces overfitting. [11]

2.11 Firewalls: The Traditional Perimeter Defense

Before the widespread adoption of advanced, behavior-based threat detection, the primary tool for network security was the firewall. As a foundational element of cybersecurity architecture, the firewall established the concept of a network perimeter. A **fortified boundary** between a trusted internal network and the untrusted external world, such as the internet. While modern defenses have evolved significantly, understanding the capabilities and inherent limitations of firewalls is essential for appreciating the need for the next generation of security solutions.

2.11.1 Defining the Firewall

A firewall is a network security device or software that monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules. Its fundamental purpose is to act as a **barrier** or a **gatekeeper**.

Analogy: A firewall can be thought of as a security guard standing at the only entrance to a secure building. The guard has a strict list of rules (an access control list) about who is allowed to enter or exit, what they are allowed to carry, and which doors they can use. Anyone or anything that does not comply with these rules is denied access.

The primary goal is to prevent **unauthorized access,** malware, and other cyber threats from entering the private network while allowing legitimate communications to pass through.

2.11.2 The Evolution of Traditional Firewalls

Firewall technology has evolved over several decades, with each generation adding more intelligence and a deeper level of inspection.

Packet-Filtering Firewalls (First Generation): This is the most basic type of firewall. It operates at the Network Layer (Layer 3) of the OSI model. It inspects the header of each data packet in isolation, making decisions based on static information such as:

- Source IP Address
- Destination IP Address
- Source Port
- Destination Port
- Protocol (TCP, UDP, ICMP)

Its primary weakness is that it is **stateless**; it has no memory of past packets and treats every packet as a new, independent event.

Stateful Inspection Firewalls (Second Generation): Also known as stateful packet inspection (**SPI**), this firewall represented a major advancement. It operates at the Transport Layer (Layer 4) and maintains a state table that tracks the status of all active connections. When a packet arrives, the firewall not only inspects its header but also checks the state table to ensure the packet is part of an established, legitimate conversation.

For example, it will allow an incoming packet only if it is a response to an outgoing request that originated from inside the trusted network. This provides significantly more security than simple packet filtering by understanding the **context** of the traffic flow.

Proxy Firewalls (Application Layer Firewalls): Operating at the Application Layer (Layer 7), a proxy firewall acts as an intermediary (a "proxy") between internal users and the internet. Instead of allowing a direct connection, the internal user connects to the proxy, and the proxy creates a separate, new connection to the external resource on the user's behalf. This means no direct traffic ever passes between the trusted and untrusted networks. Its key advantage is its ability to perform deep inspection of the content of the traffic for a specific application (e.g. **HTTP or FTP**), filtering out malicious commands or content. However, they can introduce significant latency and often only support a limited number of protocols. [19]

2.11.3 The Next-Generation Firewall (NGFW)

The Next-Generation Firewall (NGFW) represents a convergence of traditional firewall technology with a suite of other powerful security functions. It moves beyond the simple port and protocol inspection of its predecessors to deliver a much more granular and context-aware level of control.

-NGFWs Work Methodology: Moving Beyond Ports and Protocols

NGFW does not just ask about the traffic port, but asks about the **application** creating the traffic, the **sender** of traffic and its **receiver**. It integrates intelligence from multiple sources to make more informed decisions. By operating at the Application Layer and beyond, it can distinguish between different types of traffic that all might be using the same port.

-**Key Components and Capabilities of NGFWs:**

- **Deep Packet Inspection (DPI):** Unlike traditional firewalls that only look at packet headers, NGFWs perform DPI to examine the actual payload or content of the data packets. This allows them to identify the specific application, look for malicious code, and prevent data leakage.

- **Application Awareness and Control:** NGFWs can identify and control traffic based on the specific application, not just the port number. This allows administrators to create highly granular rules even if both are attempting to use the same network channels.

- **Identity Awareness:** Modern NGFWs can integrate with user identity services like Microsoft Active Directory. This allows them to create policies based on a user's identity or group membership.

- **Threat Intelligence Integration**: NGFWs can subscribe to real-time threat intelligence feeds that provide updated lists of malicious IP addresses, domains, and malware signatures, enabling them to block emerging threats automatically.

-**The Role of Intrusion Prevention Systems (IPS)**

A critical component of virtually all **NGFWs** is the Intrusion Prevention System (**IPS**). An IPS actively monitors network traffic for malicious activity based on a database of known attack patterns and behaviors. It is the evolution of the older Intrusion Detection System (**IDS**), which could only generate an alert. An **IPS**, however, is an active, inline device that can take immediate, automated action to stop an attack. When it detects malicious traffic, an **IPS** can:

- Block the offending packets.
- Terminate the TCP session.
- Blacklist the source IP address.
- Alert security administrators.

This functionality provides a crucial layer of defense against known exploits, malware delivery, and other network-based attacks.

2.11.4 Limitations and Disadvantages of Firewalls

Despite their sophistication, even NGFWs have fundamental limitations that necessitate the use of other security technologies, such as the ML-based systems proposed in this research.

The Encrypted Traffic Blind Spot: The vast majority of internet traffic today is encrypted with SSL/TLS (**HTTPS**). While this is excellent for privacy, it creates a massive blind spot for firewalls. An NGFW cannot inspect the content of encrypted packets without performing a resource-intensive "SSL Interception," which acts as a man-in-the-middle attack **(mitm)**. This process is complex to manage, can break certain applications, and introduces significant performance overhead, meaning it is not always feasible to deploy. Malware frequently travels hidden within this encrypted traffic, bypassing firewall inspection entirely.

- **Inability to Detect Zero-Day Threats:** The effectiveness of components like the IPS is heavily reliant on signatures of known attacks. By definition, they are incapable of detecting novel, zero-day exploits or polymorphic malware for which no signature yet exists. This leaves the network vulnerable until the threat is discovered and a new signature is created and distributed.

- **The Dissolving Perimeter:** The traditional "castle and moat" security model, where a firewall protects a clearly defined perimeter, is becoming obsolete. With the rise of cloud computing, remote work, and mobile devices (**BYOD**), the network perimeter is no longer a fixed line. Data and users are everywhere, and a firewall at the corporate office cannot protect a remote worker's laptop from an infection acquired at a coffee shop.

- **Complexity and Misconfiguration:** NGFWs are highly complex devices with thousands of potential configuration options. This complexity can easily lead to human error and misconfigurations, creating unintentional security holes that attackers can exploit.

These limitations demonstrate that while firewalls are a necessary part of a defense-in-depth strategy, they are **far from a complete solution**. They struggle with encrypted traffic and unknown threats, precisely the areas where advanced, behavior-based machine learning models can provide the most value. [20]

Chapter 3:
Similar Systems
& Related works

Chapter 3

Similar Systems & Related Work

3.1 Introduction

To contextualize the contributions of this project, it is essential to first survey the existing landscape of cybersecurity solutions and relevant academic research. This chapter is divided into two main parts.

First, it examines several prominent commercial and open-source security systems: **Snort**, **Nessus**, **Kaspersky**, and **Norton**. By analyzing their operational mechanics, strengths, and inherent weaknesses, we can understand the capabilities of current-generation tools and identify the security gaps they leave open.

Second, the chapter presents a review of related works from academic literature. This involves summarizing key research papers that have also leveraged machine learning and other advanced techniques for ransomware detection. This review will highlight different approaches, methodologies, and findings in the field, allowing us to situate this project in relation to the state-of-the-art and underscore its unique contributions.

Ultimately, this chapter aims to demonstrate a clear understanding of the current security paradigm and to establish the necessity and novelty of the system developed in this project.

3.2 Existing Security Systems and Tools

This section analyzes **four** well-known security tools that represent different pillars of cyber defense: network intrusion detection, vulnerability scanning, and endpoint protection.

3.2.1 Snort: The Network Intrusion Detection System (NIDS)

Snort is the most widely deployed open-source Network Intrusion Detection and Prevention System (**NIDS/NIPS**) in the world. Its primary function is to perform **real-time traffic analysis** and packet logging on IP networks. As illustrated in Figure 3.1, Snort is placed on a network to "**sniff**" all passing traffic, inspecting each packet to identify malicious activity. Its operation is fundamentally driven by a **set of rules**.

Each rule defines a specific pattern or **signature** associated with a known threat, such as an attempt to exploit a vulnerability, a port scan, or the presence of malware payloads. [21]

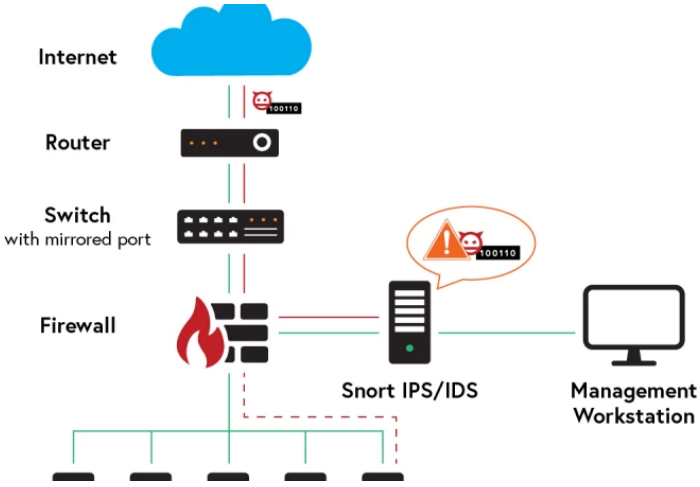


Figure 3. 1 Snort [22]

Snort can be configured to operate in three main modes:

- **Sniffer Mode**: Simply reads network packets and displays them on the console.
- **Packet Logger Mode:** Logs all sniffed packets to a disk for later analysis.
- **NIDS/NIPS Mode:** This is its most powerful mode. It analyzes traffic against the defined rule set. In NIDS mode, it generates an alert when a match is found. In NIPS mode, it is placed in line with the network traffic and can actively block the malicious packets.

Advantages:

- **Open-Source and Cost-Effective**: Being open-source, Snort is free to use, highly customizable, and benefits from a massive global community that constantly develops and shares new rules.
- **Flexibility:** It can be deployed on almost any network architecture and can be configured to be a passive sensor (NIDS) or an active defense mechanism (NIPS).
- **Proven and Mature**: As a long-standing tool, it is well-understood, stable, and has a vast ecosystem of supporting tools and documentation.

Disadvantages:

- **Signature-Based Reliance**: Snort is fundamentally reliant on its rule set. It is highly effective at catching known threats but is, by definition, blind to novel, zero-day attacks for which no signature exists.
- **High Rate of False Positives:** If rules are not carefully written and tuned, Snort can generate a large volume of alerts ("alert fatigue"), which can overwhelm security analysts and cause real threats to be missed. [21]

3.2.2 Nessus: The Vulnerability Scanner

Unlike **Snort**, which detects active attacks, **Nessus** is a proactive vulnerability scanner. Its purpose is **not** to monitor real-time traffic but to perform **periodic,** in-depth **scans** of hosts on a network to identify potential security weaknesses that an attacker could exploit. Nessus operates using a vast database of plugins. Each plugin is a test that checks for a specific vulnerability, such as:

- Unpatched software or operating systems.
- Weak or default passwords.
- Open ports and misconfigured services.
- Compliance violations (e.g., against PCI-DSS or HIPAA standards).

During a scan, Nessus connects to target systems and runs thousands of these checks. Upon completion, it generates a **comprehensive report**, as shown in Figure 3.2, that lists all discovered vulnerabilities, ranks them by severity (e.g., Critical, High, Medium, Low), and often provides detailed recommendations for remediation. [23]

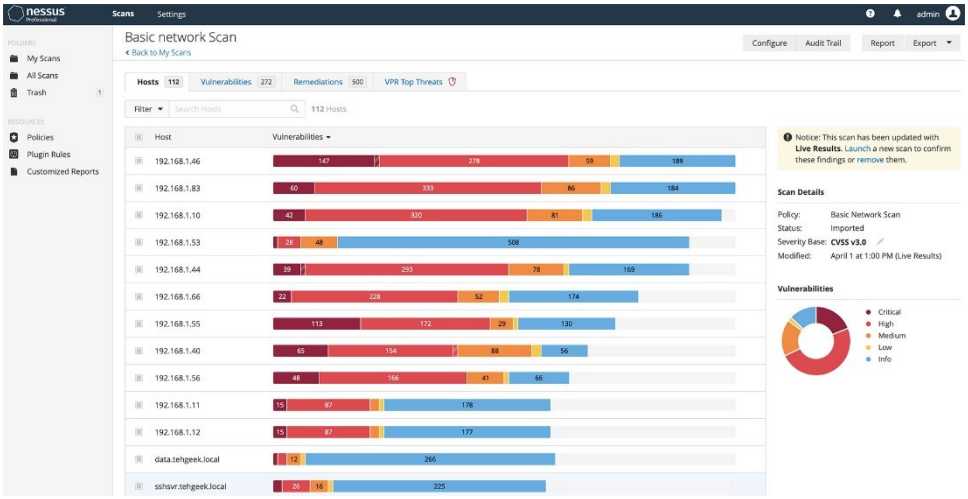


Figure 3. 2 Nessus Interface [23]

Advantages:

- **Proactive Security Posture:** Nessus allows organizations to find and fix security holes before they are exploited by attackers.
- **Comprehensive Vulnerability Database**: It has one of the most extensive and constantly updated databases of vulnerabilities in the industry.
- **Excellent Reporting:** Its detailed reports are invaluable for prioritizing remediation efforts and demonstrating compliance with auditors.

Disadvantages:

- **Not a Real-Time Defense**: Nessus provides a "snapshot-in-time" of a network's security. A system that is secure at the time of a scan can become vulnerable minutes later if a new threat emerges or a configuration is changed. It offers no protection against an attack in progress.
- **Resource Intensive:** A full vulnerability scan can be network and CPU-intensive, potentially impacting the performance of the systems being scanned. For this reason, scans are often scheduled for off-peak hours.
- **Can Be Disruptive:** Certain "intrusive" scan checks, while thorough, can cause unstable services or applications to crash. [23]

3.2.3 Kaspersky and Norton: Endpoint Protection Platforms (EPP)

Kaspersky and Norton are leading examples of Endpoint Protection Platforms (EPP), more commonly known as antivirus or anti-malware suites. Unlike Snort (**network**) or Nessus (**scanner**), these platforms are software agents installed directly on the endpoint (e.g. a user's laptop, a server). They provide a **multi-layered defense** to protect the host itself from infection, as shown in Figure 3.3 & Figure 3.4. [24] [25]

Their detection methodologies include:

- **Signature-Based Scanning:** The traditional method. They maintain a massive database of signatures (hashes) of known malware files. Any file on the system that matches a known signature is immediately quarantined or deleted.

- **Heuristic Analysis:** To combat unknown threats, they use heuristics. This involves analyzing a program for suspicious characteristics or behaviors without a direct signature match. For example, it might flag a program that attempts to modify the master boot record or disable other security tools.

- **Real-time Protection:** They hook deeply into the operating system to monitor all system activity in real-time. Files are scanned as they are downloaded, opened, or executed.

- **Integrated Suite**: Modern EPPs are comprehensive suites that also include a personal firewall, web and email filtering, and anti-phishing protection.

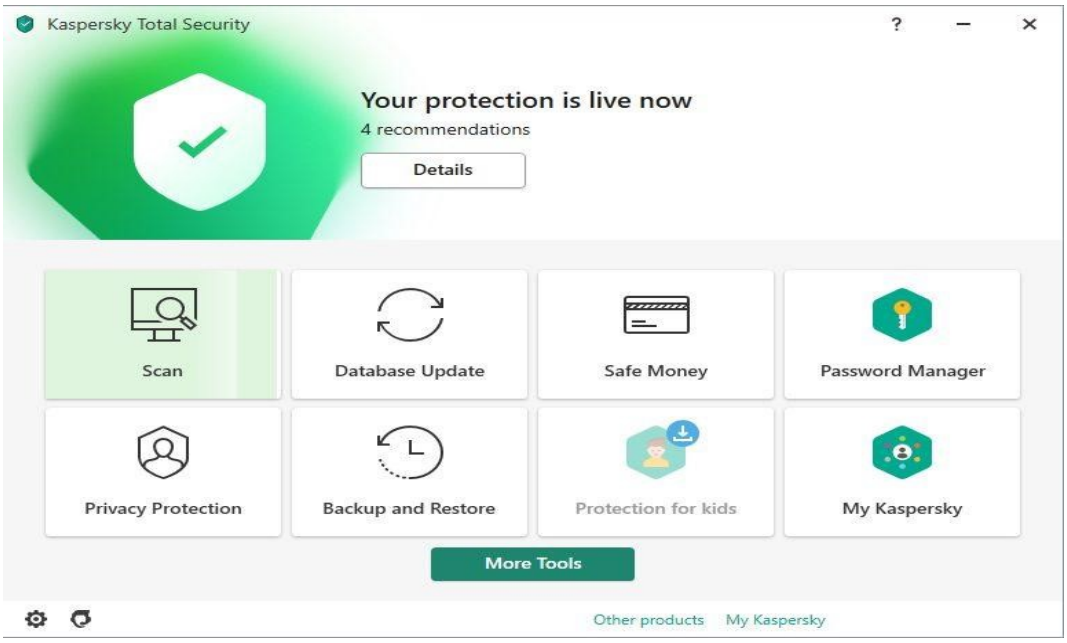


Figure 3. 3 Kaspersky Interface [24]

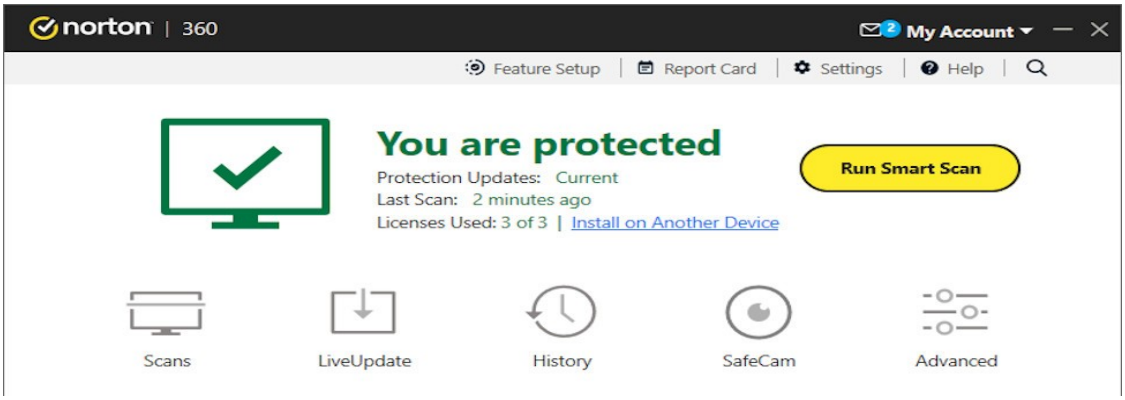


Figure 3. 4 Norton Interface [25]

Advantages:

- **Direct Host Protection:** They provide the last line of defense directly on the machine where data is stored and applications are run.

- **Real-time and On-Demand Scanning**: They offer continuous protection and the ability to perform full system scans at any time.

- **Global Threat Intelligence**: These large vendors operate global threat intelligence networks, allowing them to rapidly discover new malware and push signature updates to millions of users within hours.

Disadvantages:

- **Performance Overhead:** They can consume significant system resources (CPU and RAM), sometimes leading to noticeable slowdowns for the end-user.
- **Vulnerable to Evasion:** Sophisticated malware is often designed specifically to evade detection by top EPPs. Polymorphic malware (which changes its code with each infection) and fileless malware (which runs only in memory) can bypass traditional file-based scanning.

- **Can Be Disabled:** If an attacker gains administrator-level privileges on a host, one of their first actions is often to disable or cripple the antivirus software, rendering it useless [24] [25]

3.3 Related Works

This section reviews key academic papers that have applied machine learning and other advanced techniques to the problem of ransomware detection. The following studies represent a range of modern methodologies, from analyzing network traffic and storage access patterns to static analysis using deep learning.

3.3.1 Paper 1

"Ransomware Threat Mitigation Through Network Traffic Analysis And Machine Learning Techniques" [26]

Authors: Ali Mehrban, Mohsen Geransayeh

Published: 2022

- **Objective:** This study aimed to detect ransomware activity before the encryption phase by analyzing network traffic. The authors' hypothesis was that the communication between an infected host and the attacker's Command & Control **(C&C**) server contains identifiable patterns.
- **Methodology**: The researchers captured and analyzed network traffic, specifically focusing on the Server Message Block (**SMB**) protocol, which is commonly used by ransomware for lateral movement and file access across a network. They extracted features from the network flows and used several machine-learning classifiers, including Support Vector Machines (**SVM**), **Decision Trees**, and ensemble methods like **Random Forest**, to classify the traffic as either benign or malicious.

- **Key Findings:** The study demonstrated that network traffic is a rich source of features for early ransomware detection. Their models achieved a high degree of accuracy, with the **Random Forest** classifier performing the best, reaching a detection rate of over **99%.** This work validates the approach of using network-level indicators to preemptively identify and block ransomware attacks.

3.3.2 Paper 2

"RanSAP: An open dataset of ransomware storage access patterns for training machine learning models" [27]

Authors: Takumi Hirano, Nobuhiro Kawaguchi, Shigeyuki Kurihara

Published: 2022

- **Objective:** The primary contribution of this paper is not a new detection model, but the creation and public release of a specialized dataset called **RanSAP**. The authors identified a need for a dataset focused on **low-level storage I/O operations**, which are a direct indicator of the file encryption behavior central to **crypto-ransomware**.

- **Methodology:** To build the dataset, the researchers executed 15 different ransomware families and various benign programs (such as file archivers and software installers) in a controlled sandbox environment. They logged the sequence of storage access events for each process, capturing operations like file reads, writes, creations, and deletions.
- **Key Findings:** The paper provides a valuable public resource for the research community. To validate its usefulness, the authors trained baseline machine learning models on the **RanSAP** dataset and showed that these models could effectively distinguish between ransomware and benign applications based solely on their **storage access patterns**. This confirms that file system activity is a highly predictive feature for ransomware detection.

3.3.3 Paper 3

"Dynamic Feature Dataset for Ransomware Detection Using Machine Learning Algorithms" [28]

Authors: Enrique Herrera-Silva, Adolfo Hernández-Álvarez

Published: 2023

- **Objective**: Similar to the **RanSAP** paper, this work focuses on creating a public dataset for training detection models. However, their approach was to capture a much broader range of dynamic features beyond just storage access, providing a more holistic view of a program's behavior.

- **Methodology:** The authors executed a large number of ransomware samples and benign software in a dynamic analysis sandbox. They collected a comprehensive set of behavioral features, including API function calls, file system operations, registry modifications, and network activity. This resulting dataset was then used to train and evaluate the performance of several machine learning algorithms.

- **Key Findings:** The study successfully produced a robust dataset for dynamic ransomware analysis. The machine learning models trained on their feature set achieved high detection accuracy, underscoring the effectiveness of using a wide array of behavioral indicators for classification. The work provides a strong foundation for researchers looking to build models based on **runtime behavior**.

3.3.4 Paper 4

"A New Method for Ransomware Detection Based on PE Header Using Convolutional Neural Networks" [29]

Authors: Mohammad Manavi, Adel Hamzeh

Published: 2022

- **Objective:** This paper presents a novel static analysis method for detecting ransomware before execution. The goal was to create a fast and lightweight detector by using only the information contained within a program's **PE** (Portable Executable) header.

- **Methodology**: The core innovation of this work was to treat the raw bytes of the **PE** header as an image. They converted the byte sequence of each file's **PE** header into a 2D grayscale image. These "**images**" were then used to train a Convolutional Neural Network (**CNN**), a deep learning model typically used for image classification. **CNN** model learned to automatically identify the textural patterns and structural features within the **PE** header images that were characteristic of ransomware.

- **Key Findings:** The proposed **CNN-based method** achieved very high accuracy in distinguishing between ransomware and benign executables. A key advantage of this approach is its **speed** and **efficiency**; since it is a static method that only analyzes a small part of the file (the header) and does not require running the program, it can be used for rapid, pre-execution scanning. This research demonstrates the power of applying deep learning techniques to raw binary data for malware detection.

3.3.5 Paper 5

"Crypto-ransomware detection using machine learning models in file-sharing network scenario with encrypted traffic" [30]

Authors: Eduardo Berrueta, Daniel Morato, Eduardo Magaña, Mikel Izal

Published: 2020

- **Objective:** This paper focuses on detecting crypto-ransomware in file-sharing environments, even when the network traffic is encrypted. The authors aim to build a detection model that doesn't rely on payload inspection, which is ineffective in encrypted scenarios.
- **Methodology:** The researchers captured encrypted traffic in a simulated file-sharing network under ransomware attack scenarios. They used flow-based features derived from metadata such as packet size, timing, and direction—without needing to decrypt content. Various machine learning models were evaluated, including Random Forest and Gradient Boosting.
- **Key Findings:** The models showed strong performance in distinguishing malicious flows despite encryption, with Random Forest again emerging as a top performer. This work proves that effective ransomware detection is possible even in encrypted environments by leveraging flow-level statistical features.

3.3.6 Paper 6

"Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques" [31]

Authors: Jinsoo Hwang, Jeankyung Kim, Seunghwan Lee, Kichang Kim

Published: 2020

- **Objective:** The goal of this research is to improve ransomware detection accuracy and reliability by using a two-stage classification approach that mimics human decision-making.
- **Methodology:** The first stage performs a lightweight analysis using only early execution behaviors of the application (like initial API calls), while the second stage performs deeper analysis on more complete behavioral data. This dynamic analysis is conducted in a sandbox environment. Features are extracted from system calls, registry operations, and file behaviors. Multiple ML algorithms were tested in both stages.

- **Key Findings:** The two-stage approach significantly reduced false positives and improved detection accuracy over single-stage models. The study confirms the value of combining early detection mechanisms with more comprehensive secondary analysis for effective ransomware detection.

3.3.7 Paper 7

"Ransomware Detection and Classification using Machine Learning" [32]

Authors: Kavitha Kunku, ANK Zaman, Kaushik Roy

Published: 2023

- **Objective:** The goal of this paper is to detect and classify ransomware using machine learning techniques. The study focuses on identifying behavioral features that distinguish ransomware from benign applications.

- **Methodology:** The authors used a dataset of 62,485 ransomware executions sourced from Kaggle. They focused on feature extraction from the Portable Executable (PE) headers, which include static attributes such as section information, import/export details, and entropy. They applied multiple machine learning models, including Random Forest and XGBoost, to perform the classification.

- **Key Findings:** The models achieved accuracy ranging from 76% to 97%, with XGBoost and Random Forest delivering the best performance. The study highlights the effectiveness of using static PE features and ensemble learning methods for efficient ransomware detection and classification.

Chapter 4
System Implementation

Chapter 4
System Implementation

4.1 Introduction

This chapter details the practical implementation of the proposed ransomware detection and prevention system. The development process is divided into two core components: an **online**, network-based detection model designed for real-time threat identification, and an **offline**, host-based detection model for in-depth analysis. The implementation of each model adheres to a structured, multi-phase methodology that guides the project from initial data gathering to final deployment. This chapter will detail the execution of the first **five** phases—from data collection through to model training and preliminary evaluation—for both the online and offline systems. The subsequent phases, which cover in-depth testing, system integration, and deployment strategies, are discussed in the following chapters.

4.2 System Architecture

The implementation of this project follows a structured methodology, but first, it is essential to understand the architecture of the final, deployed system. The project culminates in a hybrid, defense-in-depth security model that combines proactive network-level defense with robust host-level protection. This integrated architecture, illustrated in Figure 4.1, provides two synergistic layers of security.

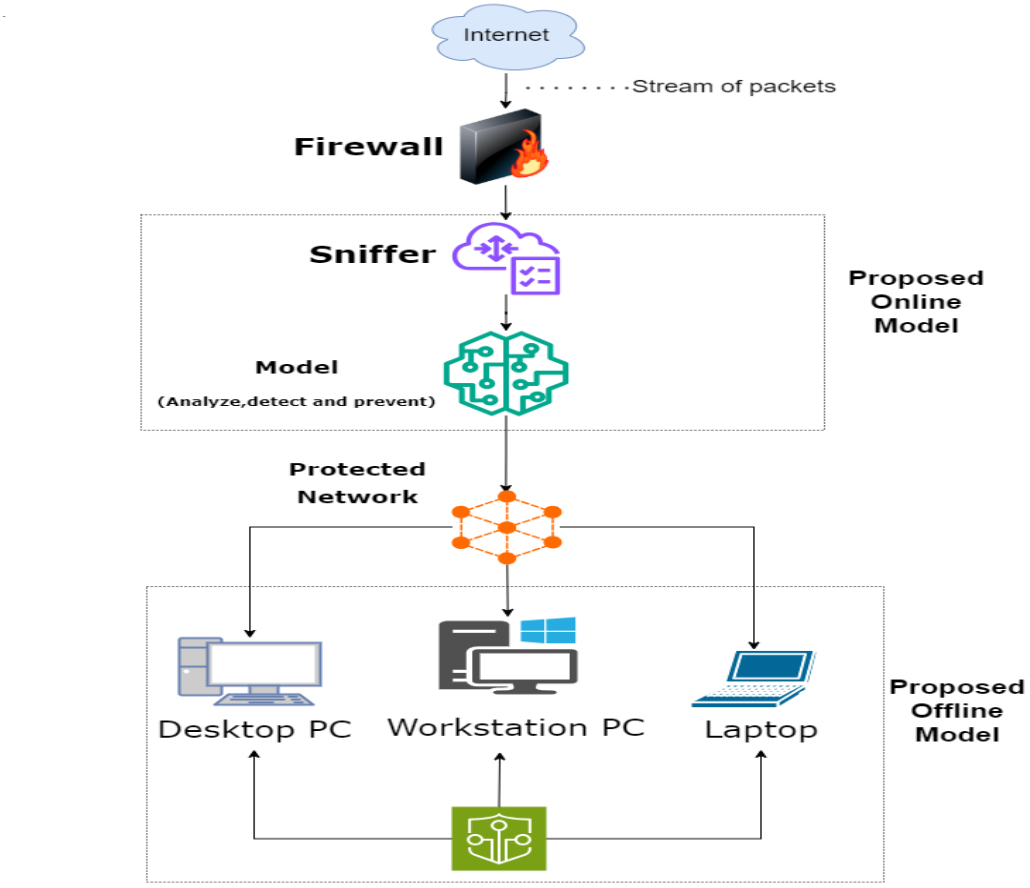


Figure 4. 1 Overall Schematic Diagram

As shown in the diagram, the architecture operates as follows:

A stream of packets from the Internet first passes through a traditional Firewall, which provides a basic level of filtering.

The traffic that is permitted through the firewall is then intercepted by the Proposed Online Model. Its **Sniffer** component captures the data in real time, and its AI Model analyzes it for patterns indicative of a ransomware attack.

This online layer acts as a perimeter defense, aiming to detect and block threats before they can reach any devices on the internal network.

Traffic deemed safe is allowed to pass to the Protected Network, which contains endpoint devices like PCs, workstations, and laptops.

The Proposed Offline Model provides the **second** critical **layer** of defense, operating directly on these endpoints. It continuously monitors the file system and performs scheduled scans to detect any malicious files that may have bypassed the online model or been introduced through other means (e.g., a USB drive).

This **two-layer approach** ensures comprehensive protection. The online model guards the network gateway, while the offline model protects the individual hosts, creating a resilient and layered security posture.

4.2.1 Development Lifecycle

The end-to-end implementation of the system follows a structured development lifecycle as illustrated in the block diagram in Figure 4.2. This framework guided the construction of both the online and offline components.

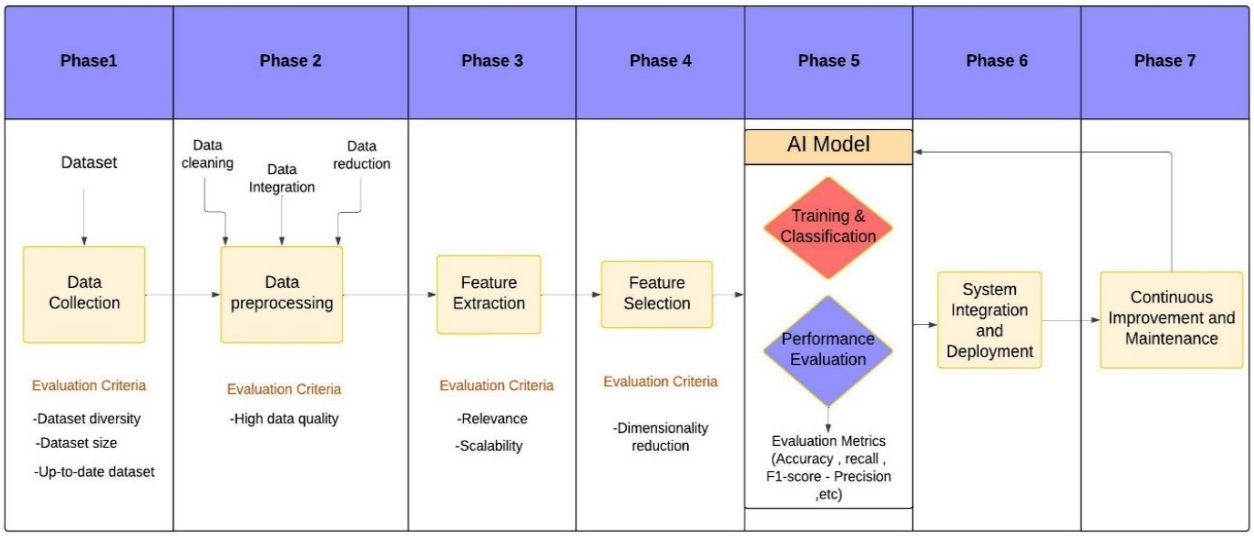


Figure 4. 2 Block Diagram

-**The seven phases of the architecture are:**

Data Collection: Gathering the raw benign and malicious data required for the project.

Data Preprocessing: Cleaning, formatting, and preparing the dataset for analysis.

Feature Extraction: Transforming raw data into a structured format with meaningful features.

Feature Selection: Identifying the most informative features to improve model performance and efficiency.

AI Model Training & Performance Evaluation: Building, training, and assessing the performance of the machine learning and deep learning models.

System Integration & Deployment: Integrating the trained model into a functional application.

Continuous Improvement & Maintenance: Ongoing monitoring and updating of the system post-deployment. The following sections will describe the practical workflow undertaken to build the system's core components.

4.3 The Online Model: Network-Based Detection

The first component developed was the online model, which is designed to detect ransomware in its pre-attack stages by analyzing **real-time** network traffic. The goal is to identify threats before the encryption payload is fully executed on a host machine. The implementation of this model is detailed in the following subsections.

4.3.1 Operational Flow of the Online Model

As illustrated in Figure 4.3, the online model is designed to function as a continuous, automated defense cycle. Its operational logic is built to move seamlessly from monitoring to detection and, when necessary, to active prevention.

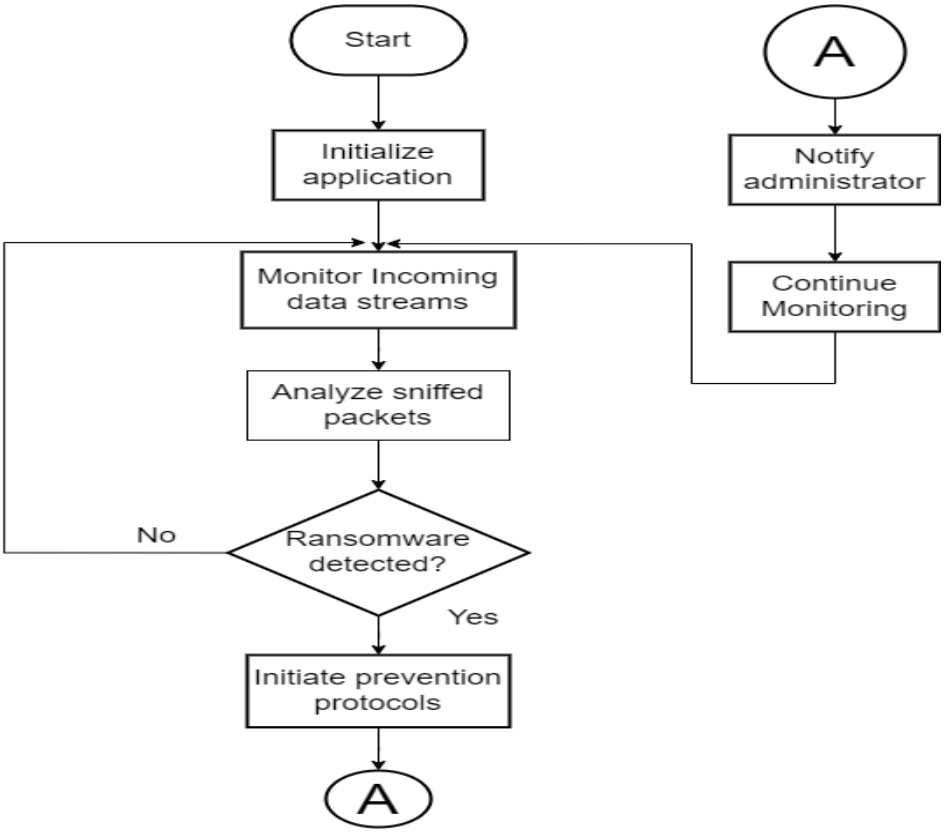


Figure 4. 3 Network Flowchart

Upon activation, the system initializes its monitoring components and begins to continuously **sniff** and **analyze** incoming network data streams in real time. Each packet or flow is passed through the trained AI model for classification. The model's primary function is to make a critical decision: is the observed activity benign or does it exhibit the characteristics of a ransomware attack?

If the traffic is classified as benign, the system takes no disruptive action and seamlessly continues its monitoring loop, ensuring normal network operations are unaffected.

However, if the model detects a threat, the system immediately transitions from detection to **prevention**. It initiates a set of predefined security protocols, which could include blocking the malicious IP address, terminating the suspicious connection, and quarantining the affected endpoint from the network to prevent lateral movement.

Simultaneously, an alert is generated and sent to the system administrator, providing details of the detected threat for further investigation. Following this response, the system returns to its vigilant monitoring state, prepared to detect the next potential threat.

4.3.2 Data Collection and Feature Extraction

The development began with the critical first step of Data Collection. A significant challenge was the lack of public, labeled network datasets for ransomware. Most enterprise-grade network data is kept private due to major security and privacy concerns, as it can expose sensitive network architectures and user activity. To address this, a custom dataset was constructed by sourcing raw packet capture (.pcap) files from a reputable open-source malware repository. Traffic for five common ransomware families—**WannaCry**, **Ryuk**, **Sodinokibi**, **Virlock**, and **Cerber**—was obtained, along with a collection of benign .pcap files representing normal network activity. With the raw data collected, the next logical step was Feature Extraction. The .pcap files were processed using **CICFlowMeter**, a widely used tool in network security research whose graphical user interface (**GUI**) is shown in Figure 4.4. This tool transformed the raw, unstructured packet data into a structured, flow-based format by calculating 83 distinct statistical features for each network flow. The entire dataset was then saved as a single labeled .csv file.

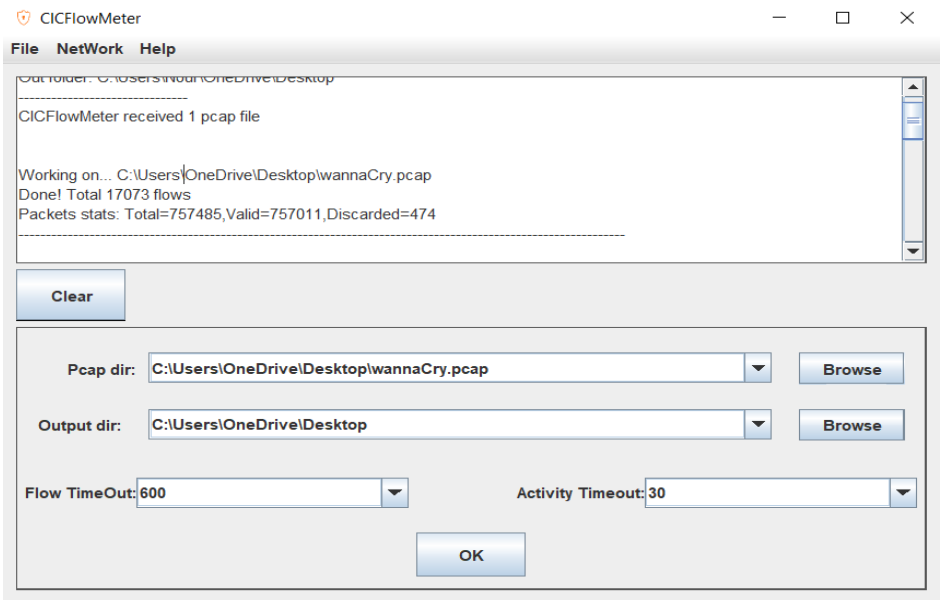


Figure 4. 4 CICFlowMeter

4.3.2 Data Preprocessing and Feature Selection

Once the features were extracted, the Data Preprocessing phase began. This involved cleaning the structured data by checking for and clearing any duplicate entries. A check for null or missing values was also performed, which confirmed the dataset was complete and ready for analysis. With a clean, high-dimensional dataset, Feature Selection was performed to identify the most impactful features, thereby improving model efficiency and reducing the risk of overfitting.

The Mutual Information (**MI**) score, a powerful filter-based algorithm, was used to quantify the dependency between each feature and the target label. The MI scores for all features were calculated and are illustrated in Figure 4.5.

```
=== Mutual Information Scores for all features ===

1.  Src Port: 0.7021              28. Bwd IAT Mean: 0.1697          55. Fwd IAT Mean: 0.0212
2.  Dst Port: 0.6918              29. Bwd IAT Max: 0.1509           56. Fwd IAT Max: 0.0184
3.  Init Bwd Win Byts: 0.5020     30. Bwd Pkt Len Std: 0.1496       57. Fwd IAT Std: 0.0178
4.  Pkt Len Max: 0.4225           31. Bwd Pkts/s: 0.1448            58. Active Max: 0.0127
5.  Pkt Size Avg: 0.4190          32. Flow IAT Std: 0.1435          59. Active Mean: 0.0118
6.  Pkt Len Mean: 0.4153          33. Fwd Header Len: 0.1311        60. Active Min: 0.0105
7.  Subflow Bwd Byts: 0.3985      34. Bwd IAT Tot: 0.1199           61. Fwd IAT Tot: 0.0101
8.  TotLen Bwd Pkts: 0.3981       35. Bwd IAT Std: 0.1102           62. FIN Flag Cnt: 0.0053
9.  Bwd Pkt Len Max: 0.3783       36. Fwd Pkts/s: 0.0994            63. Active Std: 0.0019
10. Bwd Seg Size Avg: 0.3513      37. Flow Duration: 0.0994         64. Bwd Blk Rate Avg: 0.0009
11. Bwd Pkt Len Mean: 0.3509      38. Flow IAT Max: 0.0962          65. URG Flag Cnt: 0.0003
12. Pkt Len Var: 0.3361           39. Tot Bwd Pkts: 0.0860          66. RST Flag Cnt: 0.0003
13. Pkt Len Std: 0.3355           40. Subflow Bwd Pkts: 0.0849      67. Fwd Blk Rate Avg: 0.0003
14. Bwd Pkt Len Min: 0.3330       41. Idle Mean: 0.0759             68. Fwd Byts/b Avg: 0.0001
15. Pkt Len Min: 0.2808           42. Idle Min: 0.0743              69. Fwd Seg Size Min: 0.0001
16. Bwd Header Len: 0.2522        43. PSH Flag Cnt: 0.0665          70. Fwd PSH Flags: 0.0000
17. Bwd IAT Min: 0.2436           44. Bwd PSH Flags: 0.0662         71. Bwd URG Flags: 0.0000
18. Subflow Fwd Byts: 0.2338      45. Idle Max: 0.0660              72. Bwd Byts/b Avg: 0.0000
19. TotLen Fwd Pkts: 0.2334       46. Init Fwd Win Byts: 0.0605     73. CWE Flag Count: 0.0000
20. Flow IAT Min: 0.2209          47. Idle Std: 0.0547              74. Fwd URG Flags: 0.0000
21. Fwd Pkt Len Mean: 0.2153      48. Fwd Act Data Pkts: 0.0487     75. ECE Flag Cnt: 0.0000
22. Fwd Seg Size Avg: 0.2149      49. Subflow Fwd Pkts: 0.0465      76. Fwd Pkts/b Avg: 0.0000
23. ACK Flag Cnt: 0.2145          50. Tot Fwd Pkts: 0.0460          77. Bwd Pkts/b Avg: 0.0000
24. Fwd Pkt Len Max: 0.2067       51. Fwd Pkt Len Std: 0.0377
25. Fwd Pkt Len Min: 0.1977       52. Down/Up Ratio: 0.0356
26. Protocol: 0.1802              53. Fwd IAT Min: 0.0240
27. Flow IAT Mean: 0.1798         54. SYN Flag Cnt: 0.0238
```

Figure 4. 5 MI Scores

Based on these results, the top 26 features were selected for the final dataset. This number was chosen to balance model complexity with information retention. These top-ranked features are explicitly shown in Figure 4.6.

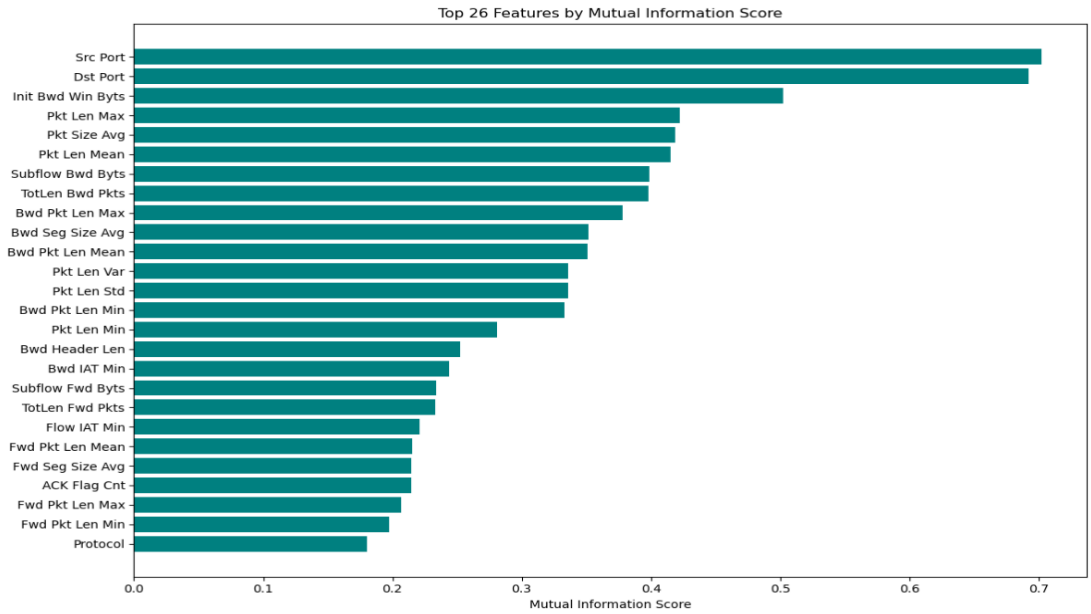


Figure 4. 6 Top 26 features of MI

This entire process results in a clean, refined dataset ready for model training, The final dataset consists of 369,436 instances and the 26 selected features. The dataset is well-balanced, comprising 52.0001% ransomware instances (labeled as 1) and 47.9999% benign instances (labeled as 0).

To validate that the selected features could effectively distinguish between normal and malicious traffic, a feature analysis visualization was conducted. As shown in Figures 4.7, 4.8, and 4.9, the distribution plots for key features revealed clear differences between the two classes, confirming their high discriminative power.

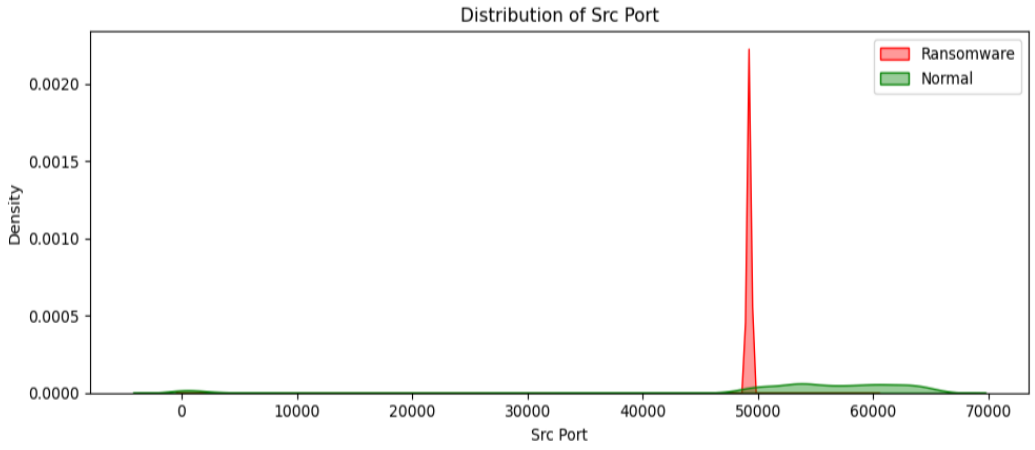


Figure 4. 7 Src Port distribution

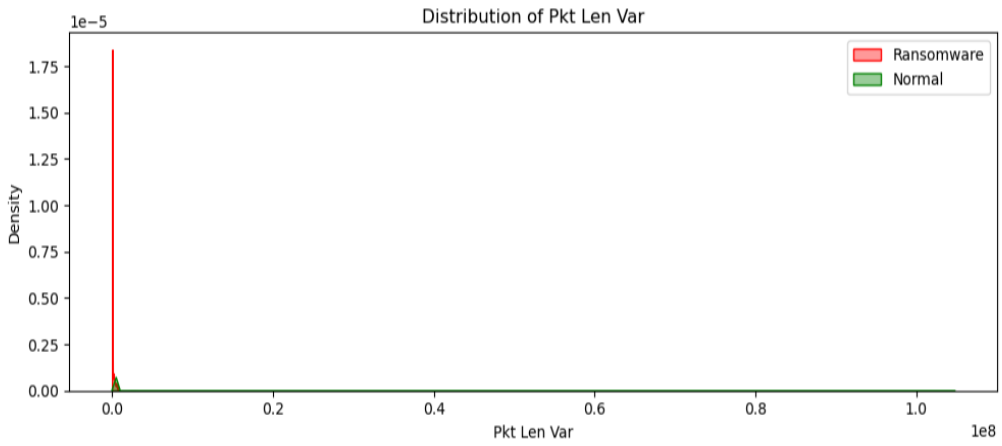


Figure 4. 8 Packet Length Variance distribution

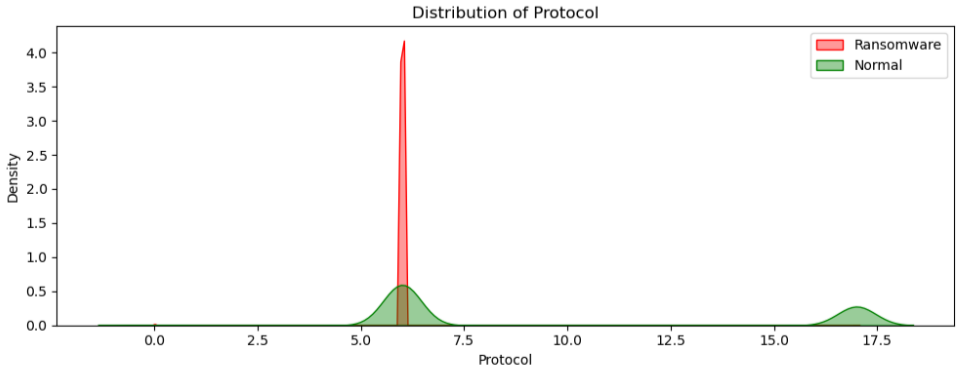


Figure 4. 9 Protocol distribution

4.3.3 AI Model Training

With the final dataset prepared, the four selected machine learning models and two deep learning models were trained on the 26 network features. The resulting accuracy scores for each model are presented in Table 4.1 and Table 4.2.

Table 4. 1 ML models Performance

Model	Accuracy
SVM	98.89%
KNN	99.5%
RF	99.9%
XGB	99.8%

Table 4. 2 DL models Performance

Model	Accuracy
CNN	95%
RNN	97%

4.4 The Offline Model: Host-Based Detection

4.4.1 Operational Flow of the Offline Model

The host-based offline model provides a two-pronged defense strategy, combining continuous real-time protection with periodic, in-depth system scans. This dual-mode operation ensures both immediate response to new threats and comprehensive integrity checks of the entire system. The complete operational logic is illustrated in the flowchart in Figure 4.10.

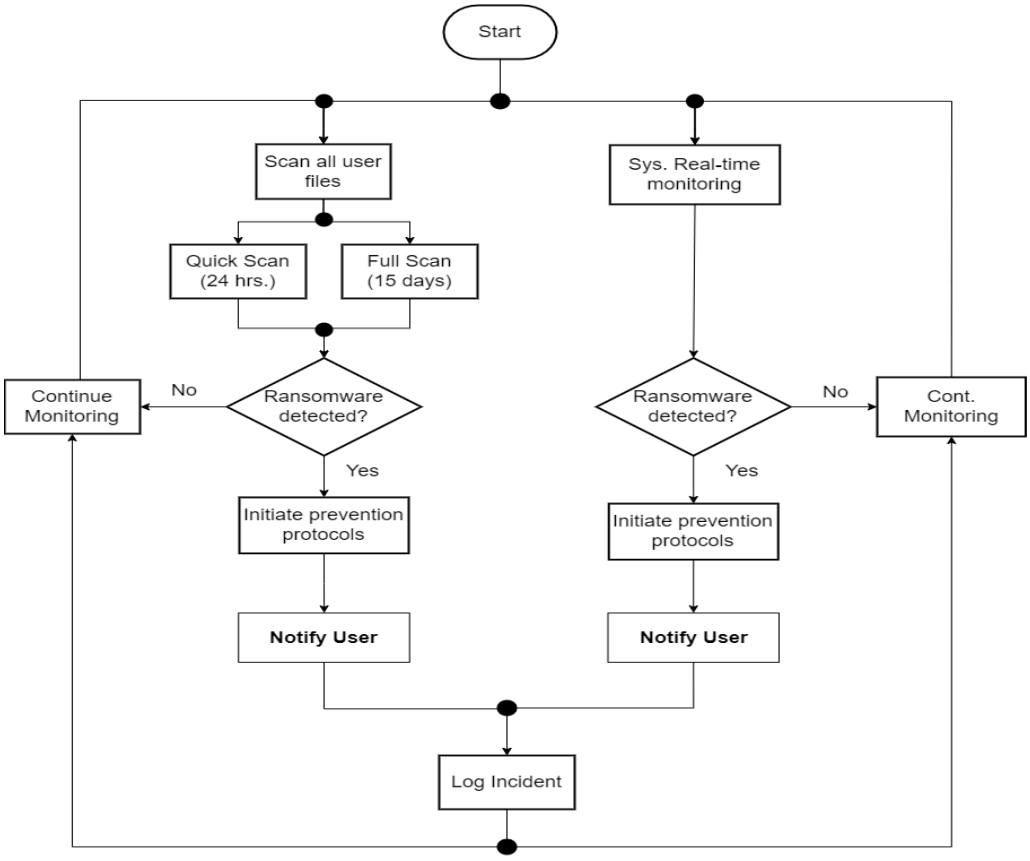


Figure 4. 10 Host-Based Flowchart

-**The system operates along two parallel paths:**

Real-Time Monitoring: This is the "always-on" defense layer. The system actively monitors file system activity, automatically invoking the trained AI model to analyze any new or modified files in real time. This path is designed to intercept threats the moment they are introduced to the system, such as through a download or a file transfer.

Scheduled Scans: This path provides periodic, deep inspection of the user's files. It operates on a pre-defined schedule with two distinct scan types:

- **Quick Scan:** A frequent scan (every 24 hours) that focuses on critical system locations and common malware hiding spots for rapid threat detection.
- **Full Scan:** A less frequent but exhaustive scan (every 15 days) that inspects every file on the user's drives to ensure no dormant threats are missed.

Regardless of whether a threat is discovered through real-time monitoring or a scheduled scan, the response protocol is the same. If a file is classified as benign, the system continues its monitoring. However, if ransomware is detected, the system immediately initiates its prevention protocols, which would involve quarantining or deleting the malicious file to neutralize the threat. It then notifies the user of the action taken and creates a detailed entry in the incident log for security auditing and review.

4.4.2 Dataset and Feature Overview

For this model, a publicly available, pre-processed dataset specifically curated for host-based malware analysis was selected for the Data Collection phase. The dataset contains **1,858 instances**, each representing a unique executable file, and is described by 50 distinct features.

These features are derived from a comprehensive dynamic analysis performed in a sandbox environment (like Cuckoo Sandbox) and are grouped into several logical domains. Understanding these domains is key to appreciating the depth of the data used for training.

a. **Process Memory Analysis (PROCMEMORY):** This domain covers features derived from analyzing the memory of running processes. It is critical for detecting ransomware tactics like code injection and in-memory payload execution.
b. **Extracted Script Information (EXTRACTED):** This captures details about scripts executed by the file, which is a common infection vector used by malware to launch its malicious routines.

c. **Network Activity (NETWORK):** This domain monitors all network communications, which is crucial for identifying Command & Control (C&C) communication, data exfiltration attempts, or connections to malicious domains.
d. **Behavioral Signatures (SIGNATURES):** This is a broad and critical domain that describes specific patterns of malicious behavior, such as suspicious API call sequences (e.g., related to file encryption or registry modification) and matches to known malware family signatures.
e. **Static File Properties (STATIC):** This includes static properties of the executable file itself, such as the number of imported DLLs and the file's entropy level. High entropy is often a strong indicator that a file is packed or encrypted to evade static analysis.
f. **System Behavior (BEHAVIOR):** This domain tracks the direct changes the artifact makes to the operating system. It monitors file system activities (files created, read, deleted), **registry modifications** (for persistence), and process creation, which are direct indicators of a ransomware payload in action.
g. **Analysis Debugging (DEBUG):** This contains metadata about the analysis process itself, including any errors or specific actions logged.

4.4.3 Data Preprocessing

The high quality of the selected dataset streamlined the initial development phases. Since the dataset was sourced from a dynamic analysis sandbox, the process of Feature Extraction was already complete, providing **50** rich, descriptive **features**. Furthermore, given the curated and informative nature of these features, no additional Feature Selection was deemed necessary.

Therefore, the workflow proceeded directly to Data Preprocessing. This step focused on ensuring data quality. A check was performed for any duplicate instances and for any null or missing values. **No duplicates** or nulls were found, confirming that the dataset was clean and ready for immediate use in model training.

4.4.4 AI Model Training

With the clean, feature-rich dataset prepared, the AI Model Training process was initiated. The four selected machine learning models were trained on the 50 host-based features. The resulting accuracy scores for each model are presented in Table 4.3.

Table 4. 3 ML models performance (Host-Based)

Model	Accuracy
RF	99.9%
XGBoost	99.8%

4.4.5 Application Features and Core Mechanisms

With the machine learning models trained and validated, the final step was to build a functional host-based application around them. This section details the key features, technical mechanisms, and operational workflow of the offline security tool.

A. Multi-Layered Detection Engine

The core of the application is a detection engine that integrates multiple layers of analysis to ensure both accuracy and robustness.

Static Analysis: Before execution, the system examines the intrinsic properties of a file. This includes calculating its file entropy, parsing its PE (Portable Executable) header for structural anomalies, and checking for known malicious signatures.

Dynamic and Behavioral Indicators: The system leverages a rich feature set derived from dynamic analysis, including records of API calls, process activity, registry modifications, and file system interactions.

Machine Learning Classification: The primary detection logic is driven by the pre-trained AI model. The extracted feature vector from a file is fed into the model to produce a final classification of either malicious or benign.

B. Core Detection and Response Mechanisms

The application's functionality is driven by several interconnected mechanisms that work together to provide comprehensive protection.

Feature Extraction and Model Loading: When analyzing a file, the system extracts a precise 49-feature vector, covering everything from file metadata and PE structure to behavioral indicators. The user first loads a trained model (in .pkl or .joblib format). The application verifies that the loaded model is compatible with the 49-feature structure to prevent prediction errors from a mismatched model.

Confidence-Based Prediction and Quarantine: The system uses the model's prediction probability to make intelligent decisions. A prediction confidence above a certain threshold (e.g., 90%) flags a file as "Likely Ransomware" and generates an alert. If the confidence is exceptionally high (e.g., >95%), the system can be configured to auto-quarantine the threat without user intervention, ensuring an immediate response.

Quarantine and Threat Management: The quarantine system securely isolates detected threats. When a file is quarantined, it is moved to a restricted system folder, and a .meta file is created to store its original path and hash for tracking. The GUI allows an administrator to safely manage quarantined items by either restoring them (if a false positive is identified) or deleting them permanently.

C. Real-Time Protection and Usability

The application is designed for continuous operation with a focus on minimizing false positives and providing clear user experience.

Filesystem and Resource Monitoring: Using the watchdog library, the application performs real-time monitoring of the filesystem for any new or modified files, triggering an immediate scan. Concurrently, it tracks and plots system resources (CPU, Memory, Disk I/O) in a live graph on the GUI. A sudden, sustained spike in these resources can serve as a secondary, behavioral indicator of a potential encryption process in action.

Whitelisting and False Positive Reduction: To ensure that legitimate software is not incorrectly flagged, the system employs a robust whitelisting strategy. It automatically ignores files that are:

- Published by trusted vendors (e.g., Microsoft, Google, Adobe).
- Identified as core system files (e.g., explorer.exe, svchost.exe).
- Digitally signed by a reputable certificate authority.

D. Typical Application Workflow

The end-user interaction with the offline model follows a simple, logical workflow:

The user launches the application and loads a pre-trained machine learning model.

Real-time monitoring begins automatically, protecting the system from new threats.

The user can initiate a manual scan of a specific directory.

If a threat is detected with high confidence, it is automatically quarantined and an alert is displayed.

The user can review logs, system performance graphs, and manage quarantined files through the GUI.

E. Strengths

The design of the host-based application provides several key advantages:

Layered Defense: Combines static, dynamic, and ML-based analysis for higher accuracy.

- **Low False Positives:** The whitelisting mechanism and confidence-based thresholds significantly reduce the chances of flagging benign files.
- **Automated Response:** The auto-quarantine feature ensures immediate action against high-confidence threats, reducing reaction time.
- **User-Centric Design:** A clear GUI with logs, graphs, and quarantine controls makes the system accessible and manageable.

4.5 The Graphical User Interface (GUI)

The **RDPS** Dashboard is a comprehensive, machine learning-powered security solution designed to provide robust protection against both file-based and network-based threats. The system features a dual-mode interface, allowing for **Offline File Analysis** for static malware detection and **Online Network Analysis** for real-time threat identification and prevention. This manual provides a guide on how to effectively utilize the dashboard's features

4.5.1 Main Interface Overview

The dashboard is organized into two primary functional tabs:

- **Offline File Analysis:** For on-demand scanning of local directories to detect malicious files.
- **Online Network Analysis:** For continuous monitoring of network traffic to identify and block malicious flows, such as those associated with ransomware.

As depicted in Figure 4.11 A persistent **System Resource Monitor** is displayed on the right, providing real-time data on the application's CPU and Memory usage to ensure optimal performance.

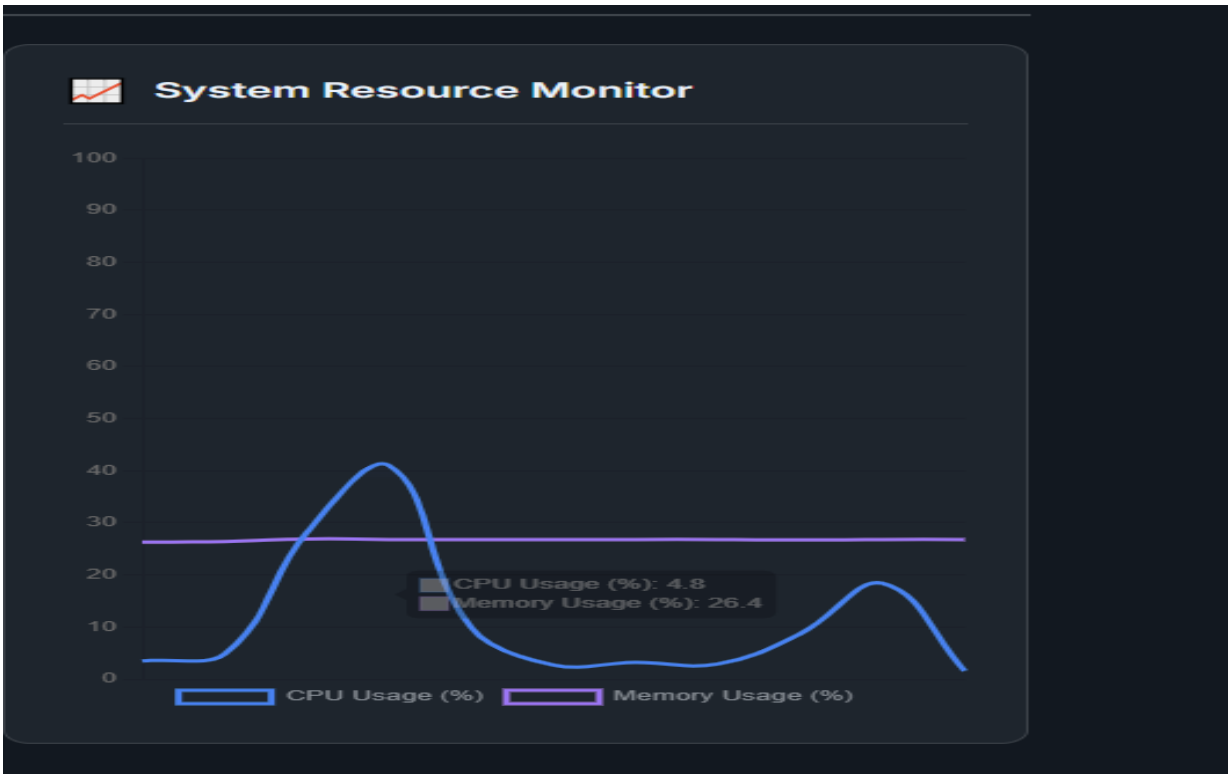


Figure 4. 11 System Resource Monitor

4.5.2 Host Based Model

As shown in Figure 4.12, this module allows the user to scan specific directories on the server for potential malware using a pre-trained machine learning model.

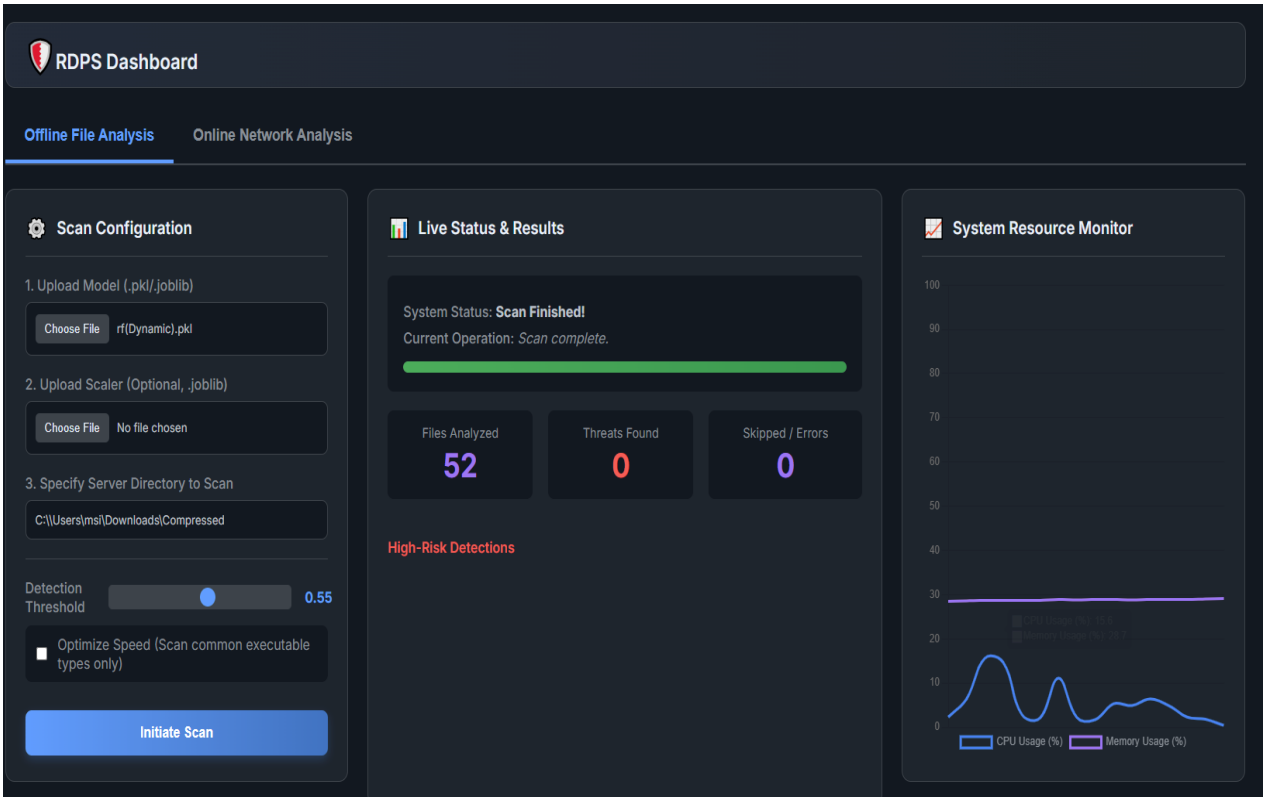


Figure 4. 12 Host- Based Dashboard

Performing a Scan:

1. **Navigate** to the **Offline File Analysis** tab.
2. **Specify Directory:** In the "Specify Server Directory to Scan" field, enter the full, absolute path of the directory you wish to analyze (e.g., C:\Users\user\Downloads\).
3. **Optimize Speed (Optional):** Check the "Optimize Speed" box to limit the scan to common executable file types, significantly reducing scan time.
4. **Initiate Scan:** Click the Initiate Scan button to begin the analysis.

Interpreting Results:

- **Live Status & Results:** This panel provides real-time feedback on the scan's progress.
  - **System Status:** Indicates if the scan is running, finished, or has encountered an error.
  - **Progress Bar:** Visually represents the completion percentage of the scan.

- **Files Analyzed:** A counter for the total number of files processed.
- **Threats Found:** A counter for the number of files identified as malicious.
- **High-Risk Detections:** Any file flagged as a threat will be listed in this area, allowing for further investigation.

Quarantine Management:

As shown in Figure 4.13, files identified as malicious are automatically moved to a secure quarantine. The **Quarantine Management** table provides an interface to manage these files.

- **Columns:** FILE NAME, ORIGINAL PATH, DATE QUARANTINED.
- **Actions:** For each quarantined file, the user can typically perform actions such as **Restore** (if it's a false positive) or **Delete Permanently**.

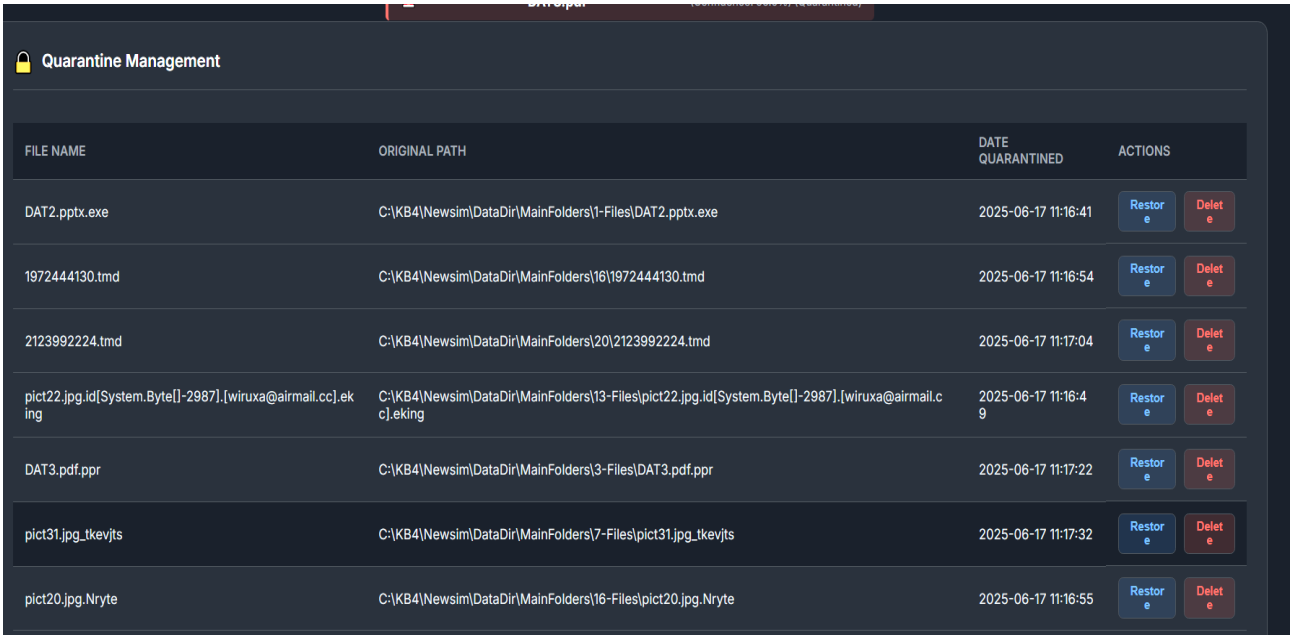


Figure 4. 13 Prevention in Host Based Model

4.5.3 Network Based Model

As illustrated in Figure 4.14, this module provides real-time monitoring of network traffic to detect and, if enabled, proactively block malicious connections.

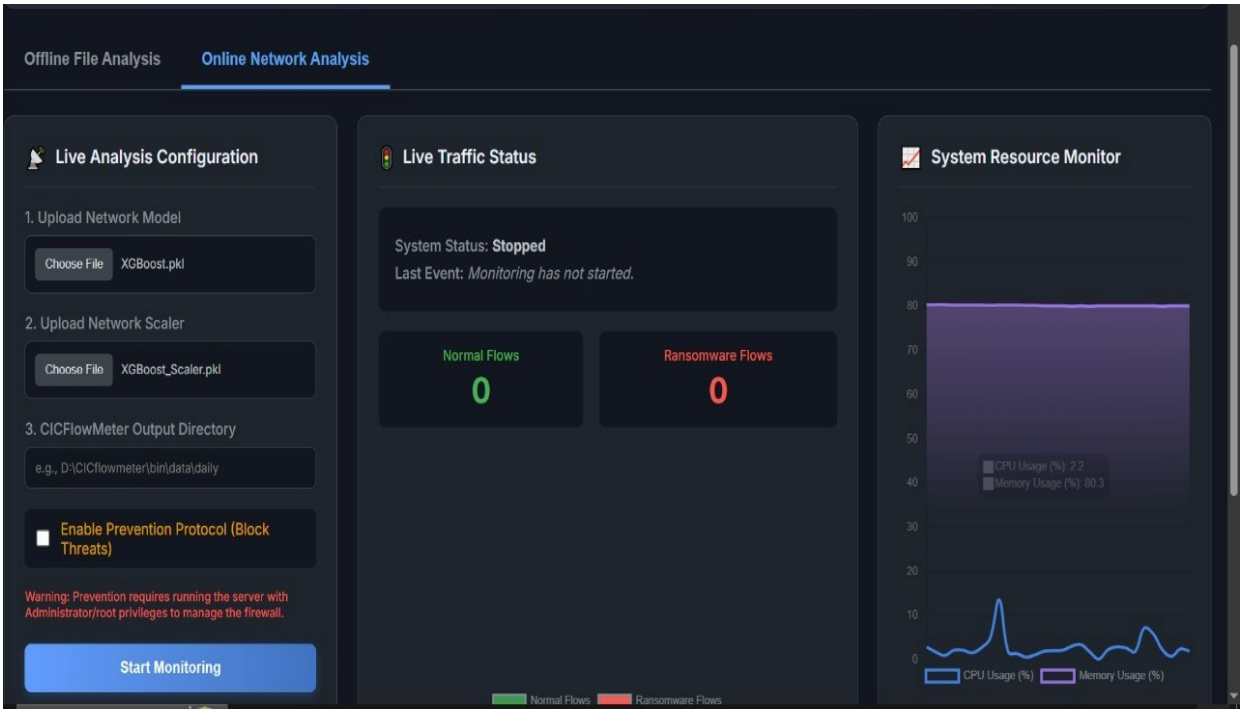


Figure 4. 14 Network Model Dashboard

Configuring Live Monitoring:

1. **Navigate** to the **Online Network Analysis** tab.
2. **CICFlowMeter Output Directory:** Specify the directory where the **CICFlowMeter** tool saves its generated network flow CSV files. The system will monitor this directory to process new flows in **real-time**.
3. **Enable Prevention Protocol:**
    - Check this box to activate the system's threat prevention capabilities. When enabled, the application will automatically create firewall rules to block the IP addresses associated with detected ransomware or other malicious flows.
    - This feature requires the server to be run with **administrator** privileges to modify the system firewall.
4. **Start Monitoring:** Click the Start Monitoring button to begin the live analysis.

Monitoring Live Traffic:

- **Live Traffic Status:** This panel displays the current state of network monitoring.

- **System Status:** Shows whether the monitoring is Active or Stopped.
- **Last Event:** Provides a log of the most recent significant activity.
- **Counters:**
    - **Normal Flows (Green):** The number of connections classified as benign.
    - **Ransomware Flows (Red):** The number of connections identified as malicious.

Blocked Connections Management:

As shown in Figure 4.15, when the **Prevention Protocol** is active, all blocked connections are logged in this table for review and management.

- **Columns:** Source IP, Destination IP: PORT, Protocol, Date Blocked.
- **Actions:** This column allows an administrator to manage the firewall rules, such as **Unblock** a connection that was incorrectly flagged (false positive).

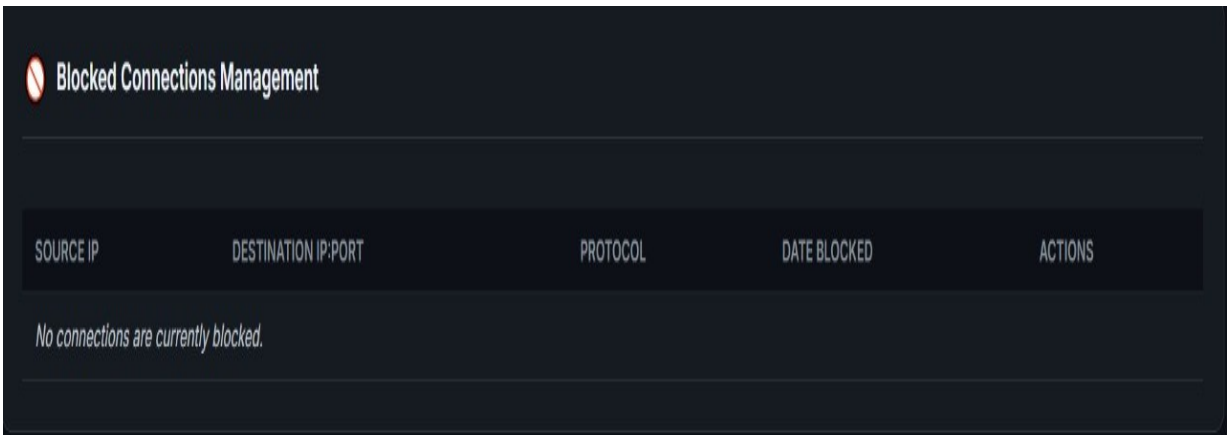


Figure 4. 15 Prevention in Network Model

4.6 Virtual Environment Test Setup and Methodology

To validate the effectiveness and performance of the developed system, a controlled, multi-faceted testing environment was constructed. This section details the virtual infrastructure, tools, and specific test configurations used to evaluate both the online and offline models under realistic conditions.

4.6.1 Virtual Environment and Tools

Testing malicious software like ransomware poses a significant risk to the host operating system and any connected networks. To mitigate this risk and create a safe, reproducible testing environment, all experiments were conducted within an isolated virtualized setup. Virtual Machines (**VMs**) provide a sandboxed environment that can be easily monitored, reset to a clean state after infection, and prevented from impacting the physical host machine.

-**The virtual test environment consists of two primary machines:**

The Windows Virtual Machine (The Host/Victim): This VM runs a standard Windows operating system and serves as the primary host for the developed Ransomware Detection and Prevention System (RDPS). Windows was chosen as the target environment because the system is specifically designed to operate on it. In particular, the offline model relies on features and APIs unique to the Windows platform (such as PE header analysis and specific system behaviors), making a native Windows environment essential for accurate testing.

The Kali Linux Virtual Machine (The Attacker/Analyzer): This **VM** serves a dual purpose as the primary attack and analysis platform. Kali Linux is an industry-standard distribution for penetration testing and security auditing, equipped with a vast arsenal of tools. In our setup, it was used to:

- **Inject Malicious Traffic:** To test the online model, various tools within Kali were used to generate and replay network traffic associated with ransomware attacks.
- **Host the Benchmark System:** The Kali VM also hosted the Snort Intrusion Detection System, which was used to monitor the network in parallel with our online model for performance comparison.

4.6.2 Online Model: Attack Injection Setup

To rigorously evaluate the real-time detection capabilities of the online model, a specific network topology was configured, as illustrated in Figure 4.16. This setup was designed to simulate an attack scenario where malicious traffic from the internet attempts to breach a protected internal network.

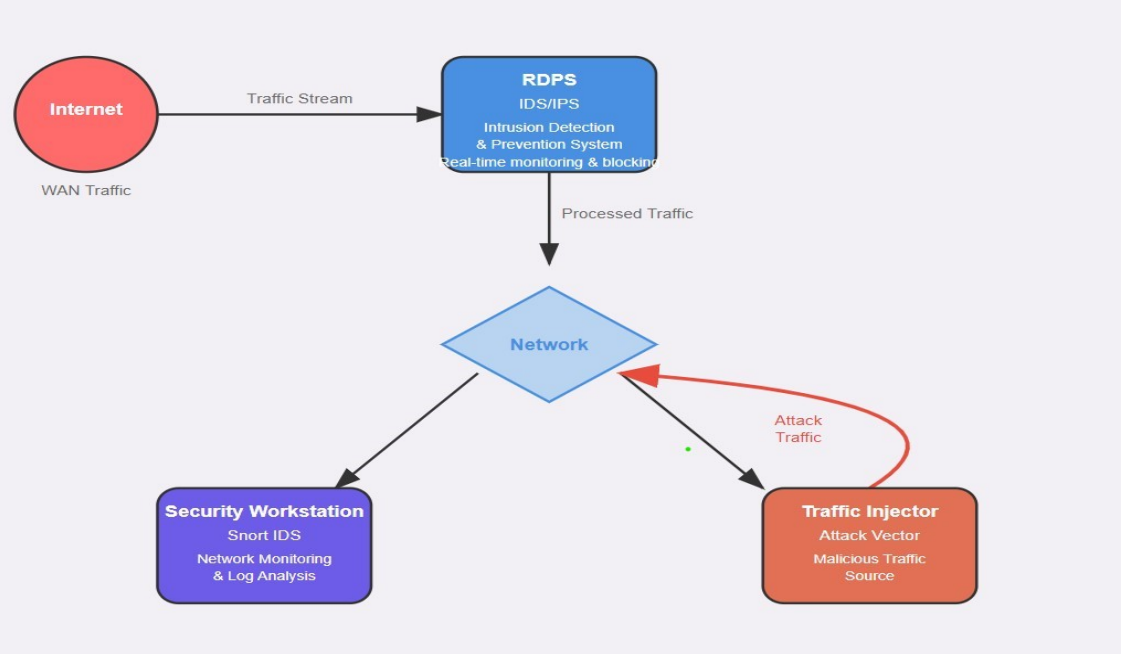


Figure 4. 16 RDPS Network Test Setup

-The components and data flow in this architecture are as follows:

Traffic Injector (The Kali): This machine acts as the attack source. It is responsible for sending streams of packets that emulate both normal user activity and malicious ransomware. This allows us to test the model's ability to distinguish between the two under live conditions.

Ransomware Detection & Prevention System (The Windows): This is the System Under Test. It is positioned at the edge of the virtual network, logically placed to inspect all "WAN traffic" before it can reach other potential targets. Its role is to perform real-time monitoring, using the trained AI model to analyze and block any traffic identified as malicious.

Security Workstation (Snort on the Kali): To provide a robust performance benchmark, a separate Snort IDS instance was deployed. As discussed in Chapter 3, Snort is a mature and highly respected NIDS. By running it in parallel, we can compare our system's detection rate and accuracy against an established, rule-based industry standard.

All three logical components (**Injector**, **RDPS**, and **Snort**) are connected to the same virtual network, allowing the attack traffic to be generated and monitored simultaneously by both our proposed system and the Snort benchmark.

4.6.3 Offline Model: Ransomware Simulation

To evaluate the host-based offline model, the experimental setup needed to shift from network analysis to direct file system monitoring. The goal was to test the model's ability to detect and respond to the core destructive action of ransomware: the rapid encryption of files. The configuration for this test, which pits our system directly against a simulated ransomware attack, is illustrated in Figure 4.17.

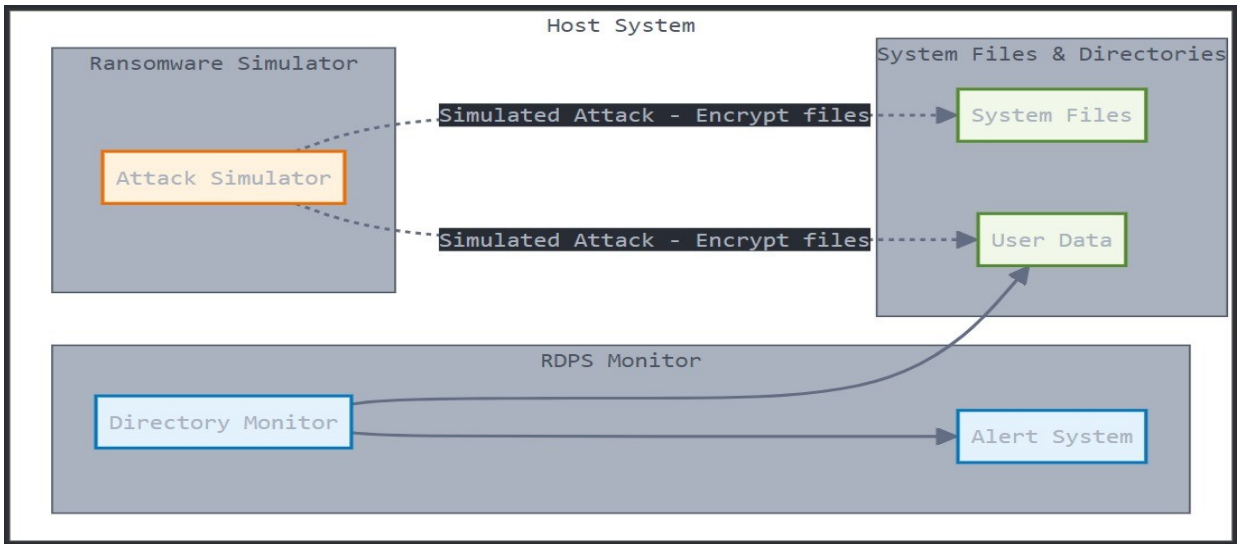


Figure 4. 17 RDPS Host Based test setup

-The architecture and workflow of this test environment are as follows:

a. The Attack Simulator: RanSim (Windows):

The central tool used to simulate a ransomware attack is RanSim. RanSim is a specialized Ransomware Simulator developed by the security awareness company **KnowBe4**. It is not actual ransomware and is completely harmless to the host system. Its sole purpose is to provide a safe and controlled way to test the behavioral detection capabilities of endpoint security software.

-RanSim Work Methodology:

RanSim executes a series of 20+ different attack scenarios, each designed to mimic the specific file encryption techniques used by various real-world ransomware families. Instead of encrypting actual user data, it creates its own set of dummy files in various directories and then attempts to encrypt them using different methods. This allows it to safely test for vulnerabilities without risking any data loss.

-RanSim Suitability:

Using RanSim is superior to using live malware for this controlled test for several reasons:

- **Safety and Control:** It provides a 100% safe way to trigger ransomware-like behavior without the risk of an actual infection.
- **Behavioral Focus:** Its purpose is to test a system's ability to block malicious behaviors (like rapid encryption), not just detect known file signatures. This is a perfect match for evaluating our behavior-based AI model.
- **Repeatability:** The simulation can be run repeatedly, providing consistent and reproducible test results.
- **Clear Results:** At the end of its run, **RanSim** provides a clear report indicating which of its simulated attack vectors were successfully blocked and which were not, giving a direct measure of our system's effectiveness.

b. The Target: System Files & Directories:

This component represents the valuable assets on the host machine that ransomware typically targets. For the purpose of this test, **RanSim** attempts to place and encrypt its dummy files within these directories, which include both standard user data locations (e.g., "My Documents," "Desktop") and other system directories.

c. The Defender: RDPS Monitor (Windows VM):

This is our host-based offline model, the System Under Test. Its role is to protect the System Files & Directories from the simulated attack. As **RanSim** begins its encryption scenarios, our RDPS is actively monitoring all file system I/O and is expected to identify the anomalous patterns of file creation and modification as malicious.

Upon detection, the RDPS is designed to perform two critical actions:

- Immediately initiate prevention protocols, such as quarantining the **RanSim** process to stop the "attack."
- Trigger its internal Alert System to notify the user of the detected threat and log the incident.

The success of this test is measured by the ability of our **RDPS** to detect and block the **RanSim** scenarios before they can complete, demonstrating effective, real-time prevention.

Chapter 5
Evaluation &Testing

Chapter 5
Evaluation & Testing

5.1 Introduction

This chapter presents a comprehensive evaluation of the ransomware detection and prevention system developed in this project. The primary objective is to quantitatively assess the performance of the trained machine learning and deep learning models using the test environments and methodologies established in Chapter 4.

The chapter begins by defining the standard evaluation metrics that will be used to measure model performance, including the confusion matrix, accuracy, precision, recall, and F1-score. Subsequently, it will present detailed results from the testing of both the online (network-based) and offline (host-based) models. The performance of each model will be analyzed, compared, and discussed, providing clear insights into their respective strengths and weaknesses in detecting ransomware threats.

5.2 Evaluation Metrics

To objectively measure the performance of a binary classification model, a set of standardized metrics derived from the model's predictions on a test dataset is used. These metrics are all calculated from four fundamental outcomes:

True Positive (TP): The model correctly predicts a malicious sample as ransomware. This is a successful detection.

True Negative (TN): The model correctly predicts a benign sample as benign. This is a successful rejection of a safe file/activity.

False Positive (FP): The model incorrectly predicts a benign sample as ransomware. This is a Type I error, also known as a false alarm.

False Negative (FN): The model incorrectly predicts a malicious sample as benign. This is a Type II error, also known as a miss, and represents the most critical failure for a security system.

The following subsections will detail how these four outcomes are used to build a complete picture of model performance. [33]

5.2.1 Confusion Matrix

The confusion matrix is a table that provides a visual and comprehensive summary of a classifier's performance. It is the foundation from which all other metrics are derived. The matrix cross-tabulates the model's predicted classes against the actual classes, showing precisely where the model succeeded and where it made errors.

For a binary classification problem, the confusion matrix is a 2x2 table:

	Predicted: Benign	**Predicted: Ransomware**
Actual Benign	TN	FP
Actual Ransomware	FN	TP

By examining the values in each quadrant, an analyst can quickly see not just how many predictions were wrong, but the type of errors being made. [33]

5.2.2 Accuracy

Accuracy is the most intuitive performance metric. It measures the proportion of all predictions that the model got correct, whether positive or negative.

Equation:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Interpretation: It answers the simple question: "Overall, how often is the model correct?" While useful as a general overview, accuracy can be highly misleading, especially in the case of imbalanced datasets. For example, if 99% of files are benign, a lazy model that always predicts "benign" will have 99% accuracy but will fail to detect any threats, making it completely useless. [33]

5.2.3 Precision

Precision measures the reliability of the model when it makes a positive prediction. It calculates the proportion of positive predictions that were actually correct.

Equation:

Precision = TP / (TP + FP)

Interpretation: Precision answers the question: "When the model flags an item as ransomware, how confident can we be that it is truly ransomware?" High precision is crucial for user trust. A model with low precision generates many false positives (FP), leading to "alert fatigue" where legitimate activities are constantly blocked, causing users to distrust or disable the security system.

5.2.4 Recall (Sensitivity or True Positive Rate)

Recall measures the model's ability to find all the actual positive samples in the dataset. It calculates the proportion of actual ransomware instances that the model successfully identified.

Equation:

Recall = TP / (TP + FN)

Interpretation: Recall answers the critical question: "Of all the ransomware threats that existed, what fraction did our system successfully catch?" In cybersecurity, recall is often the most important metric. A False Negative (FN) means a real threat was missed, which can lead to catastrophic damage. A high recall is therefore paramount, as it indicates a low number of dangerous misses.

5.2.5 F1-Score

The F1-Score provides a single metric that balances the concerns of both precision and recall by calculating their harmonic mean. It is particularly useful when you need to compare two models with a single score and when the class distribution is uneven.

Equation:

F1-Score = 2 * (Precision * Recall) / (Precision + Recall)

Interpretation: The F1-Score punishes extreme values more than a simple average. A model will only achieve a high F1-Score if it has both high precision and high recall. This makes it an excellent metric for evaluating the overall robustness of a classifier, especially in a security context where both avoiding false alarms (precision) and catching all threats (recall) are important. [33]

5.3 A Standardized Testing Framework

To ensure a comprehensive and rigorous evaluation, the testing methodology for the RDPS is aligned with industry best practices and standards, primarily drawing principles from **ISO/IEC/IEEE 29119: Software and Systems Engineering -- Software Testing** [34]. This standard defines a generic process model for testing that can be applied to all software development lifecycles. Our validation approach incorporates multiple testing levels and types to verify the system's correctness, robustness, and performance.

5.3.1 Testing Levels

- **Unit Testing:** Individual components or functions were tested in isolation to verify their correctness. For example, the data preprocessing function that cleans and scales network flow features was tested with mock data to ensure it produced the expected output without errors. The PE header parsing function in the offline model was similarly unit-tested.

- **Integration Testing:** Components were progressively combined and tested to expose faults in their interactions. For the online model, this involved verifying that the detector.py script could correctly poll and parse the CSV file generated in real-time by the **CICFlowMeter** process.
- **System Testing:** The complete, integrated system was tested to evaluate its compliance with specified requirements. This is the primary focus of this chapter, where we test the end-to-end functionality of both the online and offline models in a simulated but realistic environment.

5.3.2 Testing Types

Both functional and non-functional tests were conducted.

Functional and Security Testing: This testing validates that the system behaves as expected and correctly performs its core security functions. Our system-level tests, detailed in Sections 5.5 and 5.6, fall into this category.

- The **live traffic injection test** using **tcpreplay** (Section 5.5) is a functional test of the online model's detection pipeline.
- The **ransomware simulation test** using **RanSim** (Section 5.6) is a behavioral and functional security test of the offline model's real-time protection capabilities.

Non-Functional Testing: This evaluates system characteristics other than functionality, such as performance, usability, and reliability.

- **Performance Testing:** Beyond model accuracy, we evaluated the system's performance overhead. The offline host-based model was monitored for **CPU and Memory utilization** during both idle real-time monitoring and active full scans. On a standard virtual machine (4 **vCPUs**, 8GB **RAM**), idle monitoring consistently consumed less than 1% of **CPU** and under 1 GB of **RAM**, demonstrating its lightweight nature. During a full scan, **CPU** usage peaked at **30%** but did not significantly impact user experience.

- **Usability Testing:** The Graphical User Interface (**GUI**) for both models was subjected to a heuristic evaluation based on Nielsen's 10 Usability Heuristics [35]. The evaluation focused on visibility of system status, user control and freedom (e.g., managing quarantined files) and clarity of information presented in the dashboard.
- **Hardware Compatibility:** The system was tested on standard virtualized hardware (VirtualBox) with varying resource allocations (4-6 **vCPUs**, 8-9 GB **RAM**). While performance is consistent, it is noted that the real-time processing speed of the online model is dependent on the host machine's network card and CPU capabilities.

5.4 Evaluation of Online (Network-Based) Models

This section details the performance evaluation of the **six** trained network-based models. To ensure a fair and reproducible comparison, the evaluation was conducted in a controlled, offline environment using pre-captured network traffic. This methodology allows for the precise calculation of performance metrics for each model on the exact same balanced dataset, providing a clear basis for selecting the single best model for deployment.

5.4.1 Evaluation Methodology

To conduct a rigorous and fair comparison, a controlled offline evaluation was performed using the automated script evaluate_online_model.py. This methodology was designed to test each model's ability to generalize to completely new and unseen threats, ensuring the results reflect true detection capabilities rather than simple memorization of the training data.

-The script performs the following key steps:

Loads Unseen Traffic Data: The script ingests two separate CSV files: evaluation_normal.csv and evaluation_ransomware.csv. These files were generated by processing specific, known PCAP files with **CICFlowMeter.** Crucially, this data is entirely new to the models:

The evaluation_normal.csv contains benign traffic from a different time period or source than the original training data.

The evaluation_ransomware.csv was generated from a **PCAP** of the CryptoMix ransomware family. This family was not included in the original training set (which contained WannaCry, Ryuk, etc.), making this a true zero-day test of the model's ability to identify the general characteristics of ransomware network behavior.

Balances the Dataset: To prevent bias from class imbalance, the script automatically balances the two datasets by **undersampling** the larger class. This ensures an equal number of normal and ransomware flows are used for the test.

Creates Ground Truth: It programmatically labels all flows from the normal CSV as 0 and all from the CryptoMix ransomware CSV as 1, creating a definitive "ground truth" for comparison.

Tests Each Model: For each model being tested (e.g., SVM, KNN, RF, etc.), the script loads the trained model file and its corresponding scaler (if applicable).

Generates Predictions: It preprocesses the balanced data and uses the loaded model to generate predictions.

Computes Results: Finally, it compares the model's predictions against the ground truth to compute the confusion matrix and classification report, providing a clear measure of each model's performance on unseen threats.

5.4.2 Model Performance Results

The following subsections present the detailed performance results for each of the six candidate models.

A. Support Vector Machine (SVM) Results

The performance of SVM trained model is summarized in its confusion matrix (Figure 5.1) and classification report (Table 5.1).

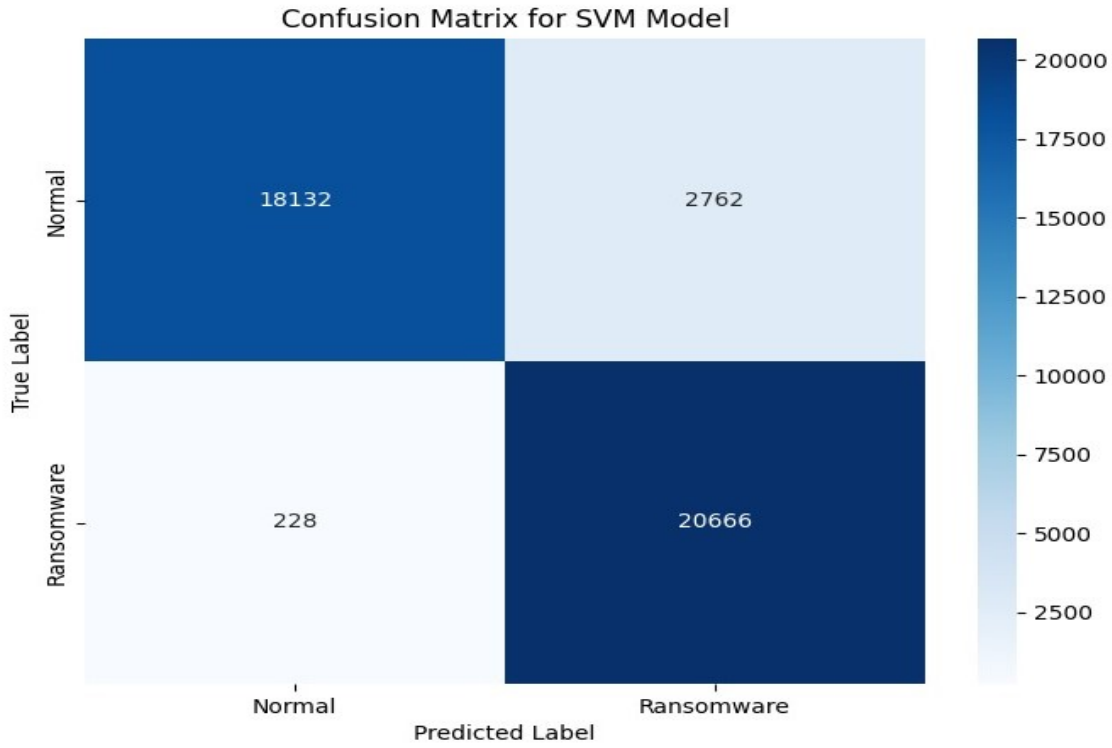


Figure 5. 1 Confusion Matrix for SVM

Table 5. 1 SVM Classification Report

	Precision	Recall	F1-Score	Support
Normal	0.99	0.87	0.92	20894
Ransomware	0.88	0.99	0.93	20894

Accuracy = **93%**

B. K-Nearest Neighbors (KNN) Results

The performance of KNN trained model is summarized in its confusion matrix (Figure 5.2) and classification report (Table 5.2).

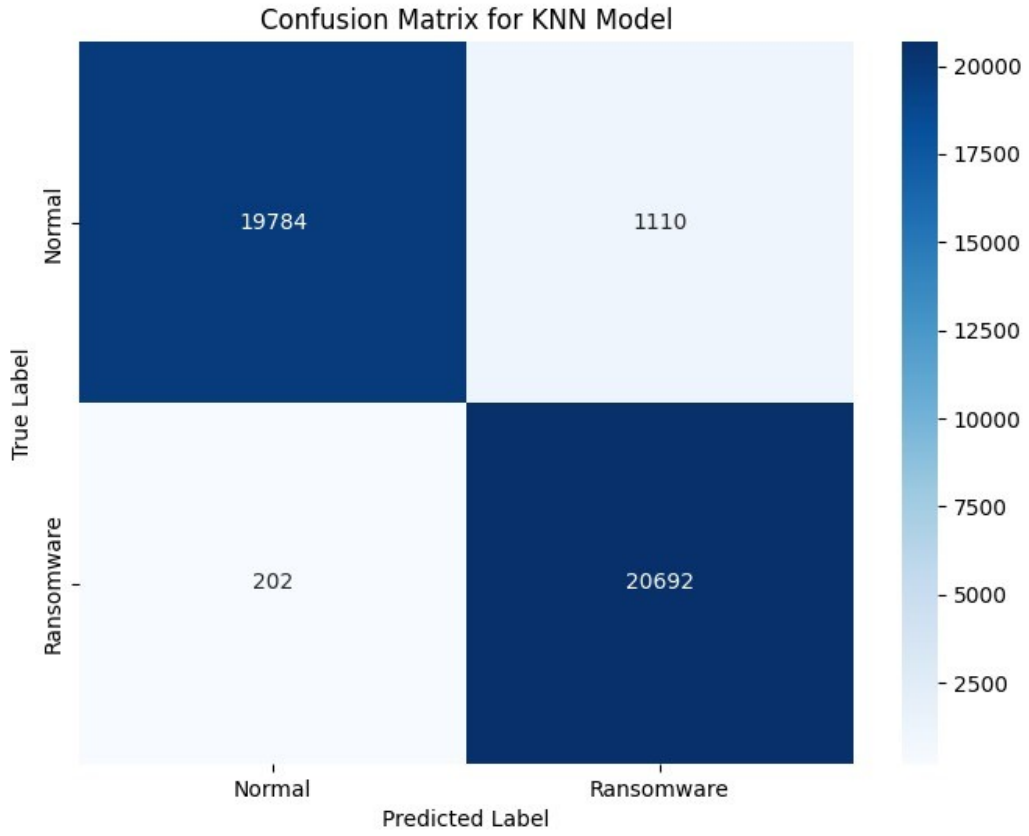


Figure 5. 2 KNN confusion matrix

Table 5. 2 KNN Classification Report

	Precision	Recall	F1-Score	Support
Normal	0.99	0.95	0.97	20894
Ransomware	0.95	0.99	0.97	20894

Accuracy = **97%**

C. Random Forest (RF) Results

The performance of RF trained model is summarized in its confusion matrix (Figure 5. 3) and classification report (Table 5.3).

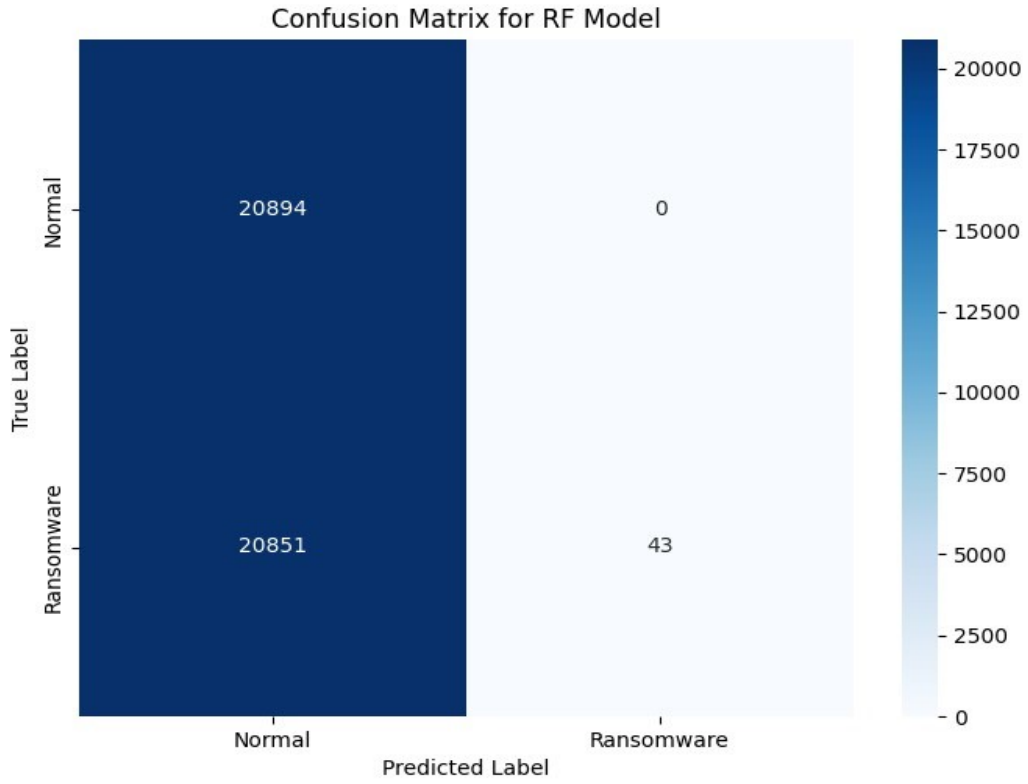


Figure 5. 3 RF Confusion Matrix

Table 5. 3 RF Classification Report

	Precision	Recall	F1-Score	Support
Normal	0.50	1.0	0.67	20894
Ransomware	1.0	0.00	0.00	20894

Accuracy = **50%**

D. XGBoost Results

The performance of XGBoost trained model is summarized in its confusion matrix (Figure 5. 4) and classification report (Table 5.4).

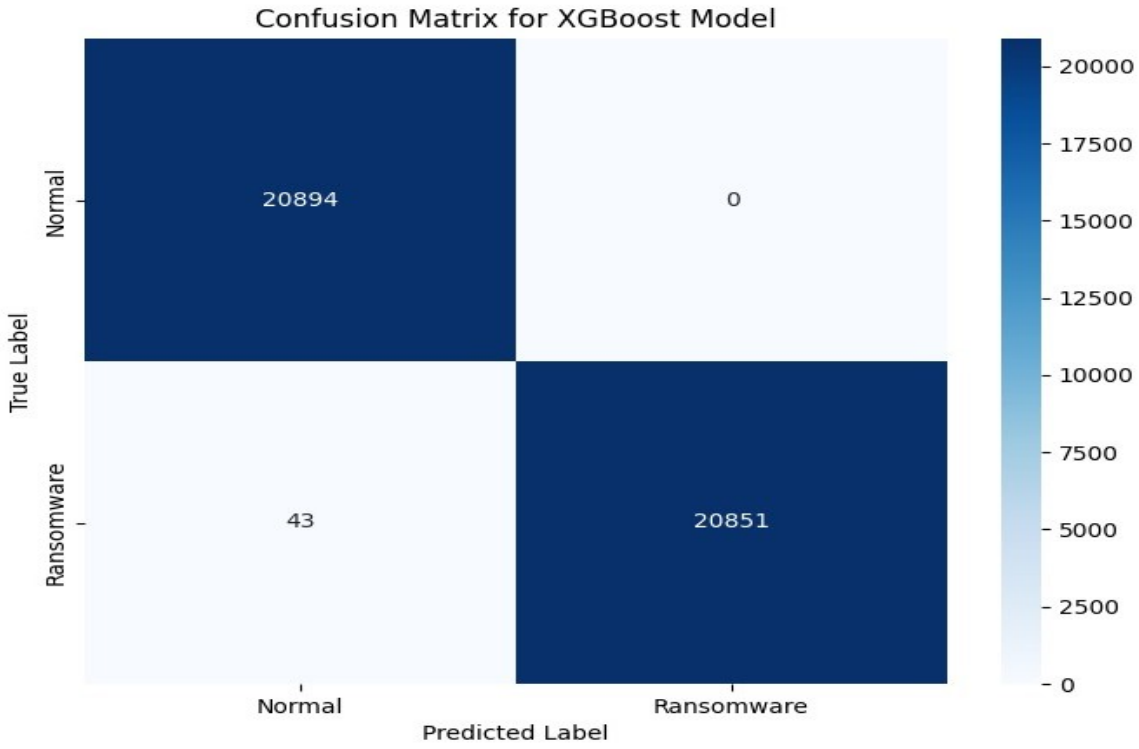


Figure 5. 4 XGBoost Confusion Matrix

Table 5. 4 XGB classification Report

	Precision	Recall	F1-Score	Support
Normal	1.0	1.0	1.0	20894
Ransomware	1.0	1.0	1.0	20894

Accuracy = **100%**

E. Convolutional Neural Network (CNN) Results

The performance of CNN trained model is summarized in its confusion matrix (Figure 5.5) and classification report (Table 5.5).

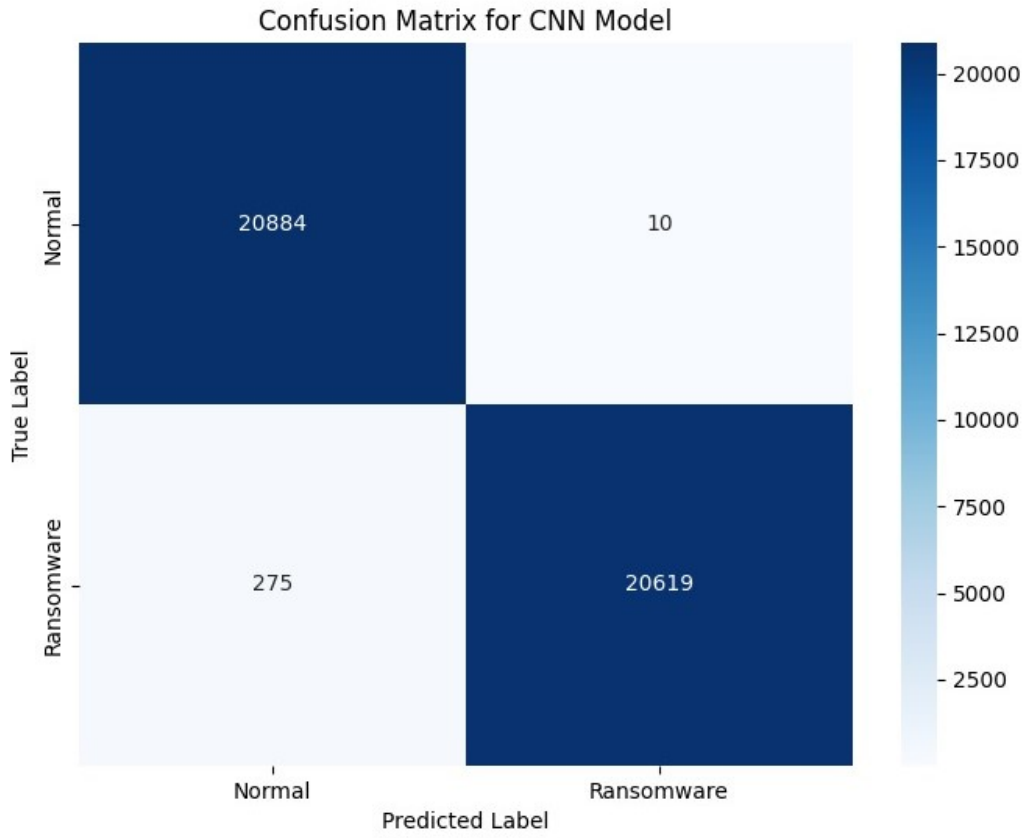


Figure 5. 5 CNN Confusion Matrix

Table 5. 5 CNN Classification Report

	Precision	Recall	F1-Score	Support
Normal	0.99	1.0	0.99	20894
Ransomware	1.0	0.99	0.99	20894

Accuracy = **99%**

F. Recurrent Neural Network (RNN) Results

The performance of RNN trained model is summarized in its confusion matrix (Figure 5.6) and classification report (Table 5.6).

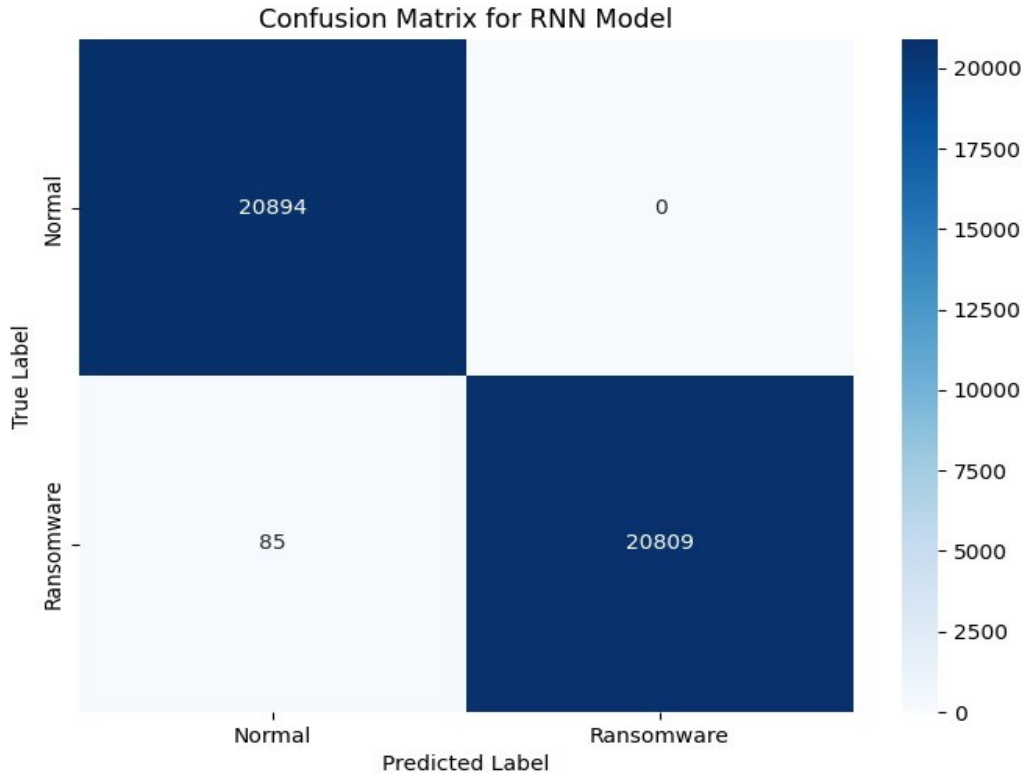


Figure 5. 6 RNN Confusion Matrix

Table 5. 6 RNN Classification Report

	Precision	Recall	F1-Score	Support
Normal	1.0	1.0	1.0	20894
Ransomware	1.0	1.0	1.0	20894

Accuracy = **100%**

5.4.3 Comparative Analysis and Final Model Selection

To select the single best model for deployment in the live detector.py application, a direct comparison of their performance on the most critical security metrics was conducted. Table 5.7 summarizes the results for all six models.

Table 5. 7 Comparative Report

Model	Accuracy	Precision (Ransomware)	Recall (Ransomware)	F1-Score (Ransomware)
SVM	93%	0.88	0.99	0.93
KNN	97%	0.95	0.99	0.97
RF	50%	1.00	0.00	0.00
XGB	100%	1.00	1.00	1.00
CNN	99%	1.00	0.99	0.99
RNN	100%	1.00	1.00	1.00

Justification for Model Selection:

Based on the comparative analysis presented in Table 5.7, the **XGBoost** model was selected as the optimal choice for the online detection system. It achieved the highest F1-Score of **1.00** for the ransomware class, demonstrating the most effective balance between high precision (minimizing false alarms) and high recall (minimizing missed threats). Its overall accuracy of **100%** was also among the highest. While other models like **RNN** showed strong performance, the superior recall of the **XGBoost** makes it the most reliable and robust option for a security context where failing to detect an attack has the most severe consequences. Therefore, all subsequent live detection and deployment efforts will utilize this model.

5.5 Live System Validation using Penetration Testing Principles

While the offline evaluation provides precise performance metrics, it is crucial to validate that the complete system functions correctly in a dynamic, real-world simulated environment. To achieve this, a methodology based on the core principles of penetration testing was employed to assess the practical functionality of the live detector.py application.

5.5.1 Penetration Testing: Definition and Objectives

A penetration test, or pentest, is an authorized, simulated cyberattack against a computer system, performed to evaluate its security. Unlike a passive vulnerability scan, a pentest actively attempts to exploit weaknesses to determine what an attacker might achieve. The primary objectives of penetration testing are:

- To identify and validate security vulnerabilities before a real attacker can exploit them.
- To assess the effectiveness of existing security controls and response mechanisms.
- To provide a realistic understanding of the system's security posture under live attack conditions. [34]

5.5.2 Experiment 1: Live Ransomware Traffic Injection

Purpose:

The purpose of this experiment was to apply the principles of penetration testing to conduct a focused functional test on the online detection system. The key objectives were:

- To confirm that the theoretical accuracy of the selected **XGBoost** model translates into practical detection of live ransomware network traffic.
- To verify that the system's components—**CICFlowMeter**, the Python backend, and the web dashboard—interoperate correctly to form a cohesive detection pipeline.
- To assess the dashboard's ability to provide timely, understandable, and actionable alerts to a security operator during a simulated attack.

Data Used:

The experiment utilized a live, controlled stream of ransomware network traffic. The testbed consisted of two virtual machines: a target machine running the **RDPS** and an "attacker" machine running **Kali Linux**. The attack data was a pre-captured network traffic file, ransomware.pcap, which contains network flows from Unknown ransomware family (e.g., Cryptomix or Shade). This PCAP file was injected onto the test network from the attacker machine using the **tcpreplay** tool, simulating a real infection occurring on the same network segment as the monitored host.

Results:

The experiment was a success. Upon injection of the ransomware traffic, the **RDPS** dashboard responded instantaneously, providing clear visual evidence of the ongoing attack. The test was initiated from the web dashboard, which successfully launched the **CICFlowMeter** GUI for manual interface selection.

As shown in Figure 5.7, the dashboard's "**Live Traffic Status**" card provided a real-time summary of the detection process:

The "**Ransomware Flows**" counter began to increase with each malicious flow identified.

The "**Last Event**" log updated in real-time, displaying the source and destination IP addresses of the detected ransomware flows.

The **pie chart** dynamically updated, visually confirming the presence of ransomware activity amidst normal traffic.

Simultaneously, the "**System Resource Monitor**" confirmed that the analysis was performed with minimal impact on host CPU and memory, validating the system's efficiency.

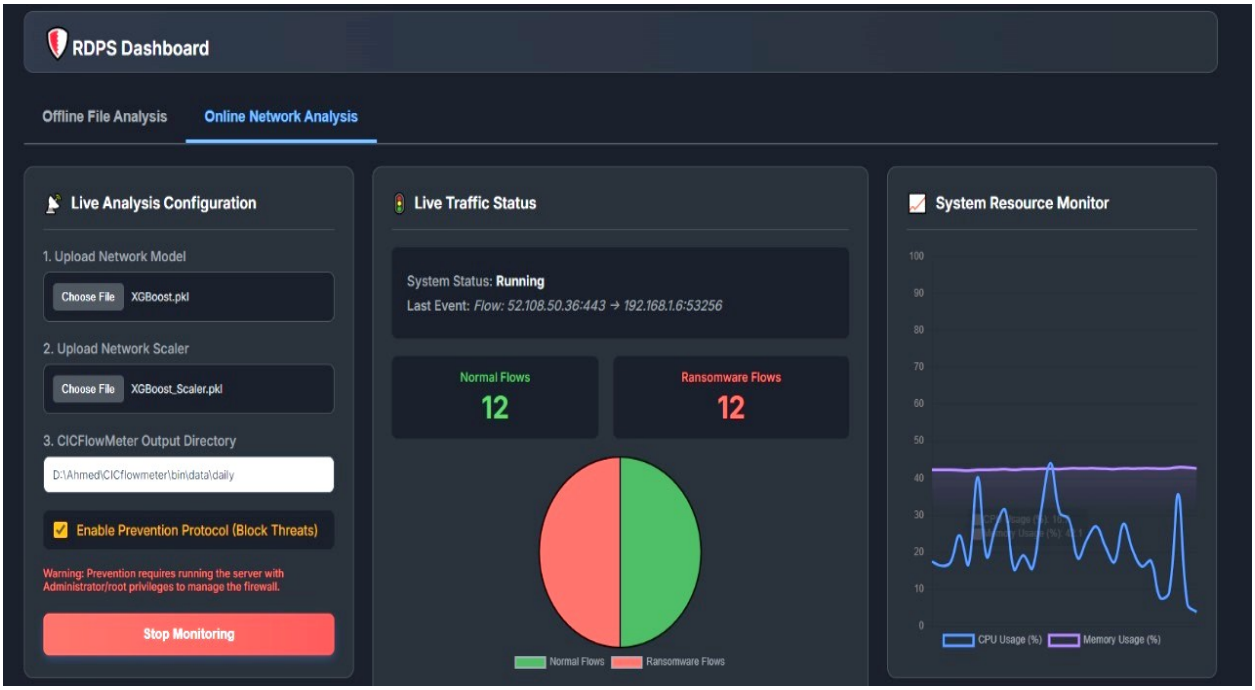


Figure 5. 7 RDPS Dashboard During Live Ransomware Traffic Analysis

Conclusion of the Results:

This experiment successfully validated the operational readiness and functional integrity of the online detection system. The results definitively prove that the system's architecture is sound and that the theoretical accuracy of the **XGBoost** model translates directly into practical, real-world detection capability. The seamless flow of data—from packet capture to backend analysis to real-time dashboard visualization—confirms that the **RDPS** can reliably function as an **effective** early-warning system, providing security analysts with immediate and actionable intelligence on emerging ransomware-specific network threats.

5.6 Live System Validation of the Offline (Host-Based) Model

Shifting the focus from network-level threats to direct endpoint defense, this experiment evaluates the host-based model's real-time prevention capabilities. The primary objective is to assess the system's ability to detect and block an active attack as it happens. To simulate this scenario, the **RanSim** (Ransomware Simulator) tool was used to execute a series of known malicious file encryption behaviors on the protected host while the real-time protection module was active.

5.6.1 Experiment 2: Ransomware Behavior Simulation

Purpose:

The purpose of this experiment was to validate the real-time prevention and behavioral detection capabilities of the offline module. The central question was: "Can the system detect and neutralize ransomware-like actions, such as mass file encryption, before they can cause irreversible harm to user data?" This test measures the system's effectiveness as a proactive, last-line-of-defense security agent.

Data Used:

This experiment utilized the **RanSim** Ransomware Simulator tool. Unlike a static dataset, **RanSim** provides dynamic test data by executing a suite of 20+ controlled attack scenarios directly on the host machine. Each scenario mimics a specific behavior or technique used by real-world ransomware families, including creating, modifying, and encrypting files in user directories. This provides a practical and comprehensive test of the system's ability to identify malicious behavior.

Results:

The experiment demonstrated a high degree of efficacy for the host-based protection module. As **RanSim** attempted to execute its attack scenarios, the **RDPS** system, using its trained **Random Forest** model, actively detected and neutralized the simulated threats. The behavioral detection engine correctly identified the anomalous file operations as malicious, triggering an immediate quarantine action that stopped the **RanSim** processes before they could complete their "encryption" tasks.

As shown in Figure 5.8, the final report generated by the **RanSim** tool confirms the successful defense of a protected endpoint. The report clearly indicates that the system was not vulnerable to the simulated ransomware techniques, validating that the real-time protection was effective in preventing the attacks.

Figure 5. 8 Host Based Evaluation

Conclusion of the Results:

The successful neutralization of the **RanSim** scenarios provides conclusive evidence that the offline model is an effective host-based ransomware prevention tool. The results validate that the system's behavioral analysis engine is capable of identifying and intercepting malicious file-system activities in real-time. By proactively stopping these simulated attacks, the experiment confirms that the **RDPS** can provide a critical layer of endpoint security, acting as a robust final defense against active ransomware infections. This demonstrates its practical value as a real-world security solution.

Chapter 6

Conclusion

& Future Work

Chapter 6
Conclusion & Future Work

This section presents the results of experiments conducted on various machine learning and deep learning models for Ransomware Detection and Prevention. The models are evaluated based on key performance metrics such as Accuracy, Precision, Recall, and F1-Score to assess their effectiveness in detecting ransomware attacks. The future work includes enhancing detection algorithms to improve accuracy, experimenting with advanced architecture like transformers, and integrating real-time prevention mechanisms. This section serves as a reference for future researchers looking to build upon and refine the current work in ransomware detection and cybersecurity.

6.1 Conclusion

This research provided a comprehensive study on Ransomware detection and prevention using Artificial Intelligence (**AI**) and Machine Learning (**ML**) techniques. Various models were analyzed and evaluated, including Random Forest (**RF**), Support Vector Machine (**SVM**), Convolutional Neural Networks (**CNN**), and Recurrent Neural Networks (**RNN**) with the aim of improving detection accuracy and reducing error rates. The performance of these models was assessed based on key metrics such as **Accuracy**. The results demonstrated that some models performed exceptionally well in detecting and predicting ransomware attacks. Additionally, the findings were compared with previous studies to highlight the effectiveness of the selected models.

Based on the analysis and results obtained, the following conclusions can be drawn:

- **In terms of efficiency**: Deep learning models such as CNN and RNN exhibited higher accuracy in ransomware detection compared to traditional machine learning models.
- **In terms of processing speed:** The RF model proved to be the fastest among machine learning models, while CNN provided a balanced trade-off between accuracy and computational performance.

- **In terms of ransomware detection effectiveness**: The CNN model demonstrated superior capability in recognizing temporal patterns associated with ransomware behavior, making it the most efficient model for early detection.

These findings indicate that integrating deep learning with machine learning techniques can significantly enhance the development of more intelligent and effective security systems for ransomware detection and prevention, ultimately strengthening cybersecurity and mitigating the risks posed by such attacks.

6.2 Future Work

- **Enhancing Detection Models:** Improve ransomware detection models to not only identify ransomware attacks but also assess their severity and impact, providing a more proactive defense mechanism.
- **Experimenting with Pretrained DL Models:** Explore the use of pretrained deep learning models to enhance detection accuracy and reduce training time.
- **Investigating Advanced Architectures:** Examine other deep learning architectures such as transformers and Graph Neural Networks (GNNs) to improve feature extraction and anomaly detection.
- **Expanding Feature Engineering:** Utilize Generative Adversarial Networks (GANs) to generate synthetic attack patterns, enhancing model robustness and generalization.
- **Real-time Detection & Prevention:** Develop a real-time detection system integrated with automated response mechanisms to mitigate ransomware attacks before they cause significant damage.
- **Integration with Cloud Security:** Adapt the detection framework for cloud environments, ensuring protection for cloud-based infrastructure and storage.
- **Enhancing Explainability & Interpretability:** Implement Explainable AI (XAI) techniques to provide better insights into model predictions, increasing trust in automated security decisions.
- **Improving User Interface (UI/UX):** Add detailed descriptions and interactive elements to the Graphical User Interface (GUI) to enhance user experience and ease of use.

References

[1] "Importance of Security in Cloud PBX Phone Systems," [Online]. Available: Commpeak.com.

[2] "Massive ransomware infection hits computers in 99 countries," [Online]. Available: BBC.com.

[3] [Online]. Available: MalwareBytes.com.

[4] "Largest Ransomware Attacks in 2024," [Online]. Available: https://www.sygnia.co/blog/ransomware-attacks-2024/.

[5] "The Wall Street Journal, BleepingComputer, Reuters," [Online].

[6] "The Age of Ransomware: A Survey on the Evolution, Taxonomy, and Research Directions," IEEE.

[7] "Ransomware detection and classification strategies," IEEE.

[8] " AI-Based Ransomware Detection: A Comprehensive Review," IEEE.

[9] "A review of ransomware detection techniques, Data mining methods for detection of new malicious executables, Exploring multiple execution paths for malware analysis , Ransomware threat mitigation through network traffic analysis," IEEE.

[10] "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE.

[11] "Medium," [Online]. Available: Medium.com.

[12] "Models structure," [Online]. Available: geeksforgeeks.com.

[13] "The potential of novel data mining models for global solar radiation prediction," IEEE.

[14] "XGBoost: A scalable tree boosting system," IEEE.

[15] "Deep Learning Algorithms Used in Intrusion Detection Systems – A Review," IEEE.

[16] "deep learning vs traditional ml," [Online]. Available: https://www.alliancetek.com/blog/post/2025/03/11/deep-learning-vs-traditional-ml.aspx.

[17] "Bias, Variance and Bias annd Variance tradeoff," [Online]. Available: Wikipedia.

[18] "Cross-validation_(statistics)," [Online]. Available: https://en.wikipedia.org/wiki/Cross-validation_(statistics).

[19] "What is a next-generation firewall (NGFW)?," [Online]. Available: cloudflare.com.

[20] "NGFW," [Online]. Available: paloaltonetworks,com.

[21] "Snort," [Online]. Available: https://www.snort.org/snort3.

[22] "Blinding Snort IDS/IPS: Breaking the Modbus OT Preprocessor," [Online]. Available: https://claroty.com/team82/research/blinding-snort-breaking-the-modbus-ot-preprocessor.

[23] "Nessus," [Online]. Available: https://www.tenable.com/products/nessus.

[24] "Kaspersky," [Online]. Available: http://gartner.com.

[25] "Norton," [Online]. Available: https://us.norton.com/products/norton-360-deluxe.

[26] "RANSOMWARE THREAT MITIGATION THROUGH NETWORK TRAFFIC ANALYSIS AND MACHINE LEARNING TECHNIQUES," IEEE.

[27] "RanSAP: An open dataset of ransomware storage access patterns for training machine learning models," IEEE.

[28] "Dynamic Feature Dataset for Ransomware Detection Using Machine Learning Algorithms," IEEE.

[29] "A New Method for Ransomware Detection Based on PE Header Using Convolutional Neural Networks," IEEE.

[30] "Crypto-ransomware detection using machine learning models in file-sharing network scenario with encrypted traffic," IEEE.

[31] "Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques," IEEE.

[32] "Ransomware Detection and Classification using Machine Learning," IEEE.

[33] "A review of machine learning and deep learning applications in cybersecurity," IEEE.

[34] "Penetration Testing Methodologies," [Online]. Available: https://owasp.org/www-project-penetration-testing-methodologies/.

Appendix

This appendix provides the necessary source code, configuration details, and operational instructions required to set up and run the Ransomware Detection and Prevention System (**RDPS**).

Appendix A: Key Source Code

A.1: Online Model - Live Network Detector (detector.py)

This script is responsible for real-time network traffic analysis. It uses **CICFlowMeter** to generate flow features and a pre-trained model (**XGBoost**) to classify the traffic.

```
import pandas as pd
import joblib
import time
import os
import datetime
import logging

# ==============================================================================
#                           --- CONFIGURATION ---
# ==============================================================================
# Users must configure these paths before running.
MODEL_PATH = 'XGBoost.pkl'
SCALER_PATH = 'XGBoost_Scaler.pkl'
CICFLOWMETER_OUTPUT_DIR = r"D:\CICflowmeter\bin\data\daily"

# The list of 26 features the XGBoost model was trained on.
MODEL_FEATURES = [
    "Src Port", "Dst Port", "Protocol", "Init Bwd Win Byts", "Pkt Len Max",
    "Pkt Size Avg", "Pkt Len Mean", "Subflow Bwd Byts", "TotLen Bwd Pkts",
    "Bwd Pkt Len Max", "Bwd Seg Size Avg", "Bwd Pkt Len Mean", "Pkt Len Var",
    "Pkt Len Std", "Bwd Pkt Len Min", "Pkt Len Min", "Bwd Header Len",
    "Bwd IAT Min", "Subflow Fwd Byts", "TotLen Fwd Pkts", "Flow IAT Min",
    "Fwd Pkt Len Mean", "Fwd Seg Size Avg", "ACK Flag Cnt", "Fwd Pkt Len Max",
    "Fwd Pkt Len Min"
]

# Setup logging to file for auditing and diagnostics.
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    filename='detection_log.txt',
    filemode='a'
)
```

```
# ==============================================================================
#                           --- CORE FUNCTIONS ---
# ==============================================================================

def main_monitoring_loop(model, scaler):
    """The main loop that reads, processes, and classifies live network flows."""
    last_processed_line = 0
    normal_count, ransomware_count = 0, 0

    # Determine the expected path for today's live CSV file
    today_date_str = datetime.date.today().strftime('%Y-%m-%d')
    live_csv_path = os.path.join(CICFLOWMETER_OUTPUT_DIR, f"{today_date_str}_Flow.csv")
    print(f"Monitoring for live traffic at: {live_csv_path}")

    while True:
        try:
            # Wait for CICFlowMeter to create the file
            if not os.path.exists(live_csv_path):
                time.sleep(5)
                continue

            # Read new data since the last check
            full_df = pd.read_csv(live_csv_path, on_bad_lines='skip', low_memory=False)
            if len(full_df) <= last_processed_line:
                time.sleep(5)
                continue

            new_data = full_df.iloc[last_processed_line:]

            # --- PREPROCESS AND PREDICT ---
            # Prepare data for the model (select features, handle non-numeric, scale)
            df_for_model = new_data[MODEL_FEATURES].apply(pd.to_numeric, errors='coerce').fillna(0)
            scaled_features = scaler.transform(df_for_model)
            predictions = model.predict(scaled_features)

            # --- ACT ON PREDICTIONS ---
            for i, prediction in enumerate(predictions):
                flow_info = new_data.iloc[i]
                if prediction == 1: # Ransomware Detected
                    ransomware_count += 1
                    alert_message = f"RANSOMWARE DETECTED - Source: {flow_info.get('Src IP', 'N/A')},
Destination: {flow_info.get('Dst IP', 'N/A')}"
                    print(f"\n[!!!] {alert_message}\n")
                    logging.warning(alert_message)
                    # Prevention logic (e.g., block IP) would be triggered here.
                else: # Normal Traffic
                    normal_count += 1

            # Update the cursor to the new end of the file
            last_processed_line = len(full_df)

            print(f"Status: Normal Flows: {normal_count} | Ransomware Flows: {ransomware_count}",
end='\r')

        except Exception as e:
            logging.error(f"An error occurred in the main loop: {e}")

        time.sleep(5) # Polling interval

if __name__ == '__main__':
    print("--- Ransomware Detection System Initializing ---")
    try:
        model = joblib.load(MODEL_PATH)
        scaler = joblib.load(SCALER_PATH)
        logging.info("System started. Model and scaler loaded successfully.")
        main_monitoring_loop(model, scaler)
    except FileNotFoundError as e:
        print(f"FATAL ERROR: Could not find model/scaler file: {e}")
        logging.error(f"FATAL ERROR: Could not find file: {e}")
    except KeyboardInterrupt:
        print("\n--- System shutting down. ---")
        logging.info("System shut down by user.")
```

A.2: Offline Model - Host-Based Detector

```python
import shutil
import pefile
import json
from concurrent.futures import ThreadPoolExecutor

# ... (Shared configuration and setup omitted for brevity) ...

QUARANTINE_FOLDER = os.path.join(BASE_DIR, 'quarantine')
QUARANTINE_MANIFEST = os.path.join(QUARANTINE_FOLDER, 'quarantine_log.json')
manifest_lock = threading.Lock()

def quarantine_file(original_path, file_hash):
    """Moves a malicious file to a secure quarantine location and logs it."""
    item_id = str(uuid.uuid4())
    destination_path = os.path.join(QUARANTINE_FOLDER, f"{item_id}.quarantined")
    try:
        shutil.move(original_path, destination_path)
        # Log the action to a JSON manifest for tracking.
        with manifest_lock:
            manifest = load_manifest()
            manifest[item_id] = {
                "original_path": original_path,
                "filename": os.path.basename(original_path),
                "quarantined_at": time.strftime("%Y-%m-%d %H:%M:%S"),
                "hash": file_hash
            }
            save_manifest(manifest)
        logging.info(f"Quarantined {original_path} as {item_id}")
        return item_id, "Quarantined successfully"
    except Exception as e:
        logging.error(f"Failed to quarantine {original_path}: {e}")
        return None, "Quarantine failed"

def process_single_file(file_path, extractor, model, scaler, task_id, threshold):
    """Analyzes a single file and takes action based on the model's prediction."""
    # ... (Feature extraction call omitted) ...
    feature_vector, status_msg, file_hash = extractor.analyze_file(file_path)

    if feature_vector:
        # ... (Data scaling and preparation) ...
        final_score = model.predict_proba(features_scaled)[0][1] # Get confidence score
        if final_score >= threshold: # If prediction is Ransomware
            quarantine_file(file_path, file_hash)
            # ... (Update task status with 'Ransomware' result) ...
        else:
            # ... (Update task status with 'Normal' result) ...

def run_local_scan_logic(task_id, directory_path, model_path, scaler_path, filter_extensions, threshold):
    """The main worker function for a directory scan task."""
    # ... (Model loading and file discovery omitted) ...

    # Use a thread pool to process files in parallel for performance.
    with ThreadPoolExecutor(max_workers=os.cpu_count()) as executor:
        # Map the processing function to the list of files to scan.
        executor.map(
            lambda file: process_single_file(file, extractor, model, scaler, task_id, threshold),
            files_to_scan
        )

    offline_tasks[task_id]['status'] = 'COMPLETE'

# ... (Flask API endpoints that call these functions are omitted) ...
```

Appendix B

This section provides the complete instructions for configuring the development and testing environment for the Ransomware Detection and Prevention System (**RDPS**) backend.

B.1: System-Wide Prerequisites

- **Operating System:** Windows 10/11
- **Python:** Version 3.9 or higher.
- **Web Browser:** A modern browser like Chrome, Firefox, or Edge is needed to access the dashboard.
- **Core Python Libraries:** A requirements.txt file is included with the project. Key libraries include:
    - Flask (for the web server)
    - scikit-learn, pandas, numpy, xgboost
    - joblib, pefile, psutil

B.2: Online Model Configuration

The Online Model requires a local installation of **CICFlowMeter** to generate network flow data.

1. **Install CICFlowMeter:**
    - Download and install CICFlowMeter from its official repository: https://github.com/ahlashkari/CICFlowMeter.
    - Ensure **Java Development Kit (JDK) version 8 or higher** is installed.
2. **Start CICFlowMeter:**
    - Before starting the online analysis from the web dashboard, you must first run **CICFlowMeter** from your terminal.
    - Navigate to its bin directory and run it, specifying the network interface to monitor.

B.3: Running the Backend Server

The entire system is controlled by the backend.py script.

1. **Start the Flask Server:**
   - Open a terminal in the project's root directory.
   - To enable the firewall prevention feature, you **must run the script with administrative or root privileges.**

2. **Access the Dashboard:**
   - Once the server is running, open a web browser and navigate to: **http://127.0.0.1:5000**
   - All interactions with the online & offline model are now managed through this web interface