

Ransomware on the Run: A Detection Approach using Effective API and Machine Learning Models



By

Asad Iqbal

2020-NUST-MS-IS-330557

Supervisor

Dr. Mehdi Hussain

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(July 2023)

Approval

It is certified that the contents and form of the thesis entitled "Ransomware on the Run: A Detection Approach using Effective API and Machine Learning Models" submitted by Asad Iqbal have been found satisfactory for the requirement of the degree

Advisor : Dr. Mehdi Hussain

Signature:  _____

Date: 12-Jul-2023

Committee Member 1: Dr. Hasan Tahir


Signature:  _____

12-Jul-2023

Committee Member 2: Dr. Qaiser Riaz

Signature:  _____

Date: 12-Jul-2023

Signature:  _____

Date: 12-Jul-2023


THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Ransomware on the Run: A Detection Approach using Effective API and Machine Learning Models" written by Asad Iqbal, (Registration No 00000330557), of SEECS has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____ 

Name of Advisor: Dr. Mehdi Hussain

Date: 12-Jul-2023

HoD/Associate Dean: Dr. Mehdi Hussain 

Date: 25-Aug-2023

Signature (Dean/Principal): _____

Date: _____


Dedication

Dedicated to my parents for their unconditional love, prayers and support throughout my life; my siblings, especially my brother whose support and help in everything makes life easier.

Certificate of Originality

I hereby declare that this submission titled "Ransomware on the Run: A Detection Approach using Effective API and Machine Learning Models" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name: Asad Iqbal

Student Signature: 

Acknowledgement

First of all, I would like to thank Allah, the Almighty for giving me the ability and strength to carry out this research. My deepest gratitude to my supervisor Dr. Mehdi Hussain for his continuous support and guidance during my thesis. I could not have imagined having a better supervisor and mentor for my master's degree. I am also thankful to my teachers for providing me with an academic base, which enabled me to complete this thesis.

I am thankful to all my fellows and friends for their support and motivation. Last but not the least, I would like to thank my parents for their endless prayers and support throughout.

Table of contents

Approval	Error! Bookmark not defined.
THESIS ACCEPTANCE CERTIFICATE	Error! Bookmark not defined.
Dedication.....	iii
Certificate of Originality	Error! Bookmark not defined.
Acknowledgement.....	v
Table of contents	ix
List of Abbreviations.....	xii
List of tables	xii
List of figures	xiii
Abstract.....	xiv
1. Introduction.....	1
1.1 Overview	1
1.2 Attack Vector	2
1.3 Thesis Motivation.....	5
1.4 Research Objectives.....	5
1.5 Research Questions.....	6
1.6 Problem Statement.....	8
1.7 Solution Description.....	8
1.8 Thesis Contribution	9
1.9 Thesis Organization.....	9
1.10 Summary.....	10
2. Literature Review.....	11
2.1 Overview	11
2.2 Machine Learning.....	12
2.2.1 Machine Learning Algorithms.....	12
2.3 Related Work	14
2.4 Summary.....	19

3. Research Methodology	20
3.1 Introduction/Overview:	20
3.2 Research Methodology:	20
3.3 Detection at the Pre-Encryption Stage.....	22
3.3.1 Discretization.....	25
3.3.2 Ransomware Signature.....	25
3.3.3 Malware Detection by Employing Machine Learning Models	26
3.4 Research Methodology	26
3.4.1 Ransomware Samples	27
3.4.2 Cuckoo Sandbox.....	29
3.4.3 Extraction of Data.....	32
3.4.4 Machine Learning Algorithm	35
3.4.5 Signature Database	37
3.5 Pre-Encryption Model Verification.....	38
3.6 Summary.....	39
4. Experimental Setup.....	40
4.1 Overview	40
4.2 Setting up the Environment	41
4.3 Model Implementation	43
4.4 Installing Pre-requisite Software.....	45
4.5 Summary.....	46
5. Experimental Results	47
5.1 Overview	47
5.2 Evaluation Measures	47
5.3 Performance Evaluation of Proposed Model.....	48
5.4 Summary.....	53
6. Discussion and Analysis	54
6.1 Overview	54
6.2 Comparison with Reference Approach.....	64
6.3 Applicability of the Approach.....	68
6.4 Summary.....	69
7. Conclusion & Future Work	70
7.1 Conclusion	70
7.2 Limitation & Future Work	71
Bibliography	73

Appendices a
Appendix-A.....a

List of Abbreviations

API – Application Programming Interface

RF – Random Forest

SVM – Support Vector Machine

DT – Decision Tree

KNN – K-Nearest Neighbour

NB – Naïve Bayes

VC – Voting Classifier

ROC – Receiver Operating Characteristics

TP – True Positive

FP – False Positive

List of tables

Table 1 Crypto-Ransomware Techniques and Detection Methods.	16
Table 2 APIs with “CRYPT” Keywords.	30
Table 3 Machine Configuration.	31
Table 4 Cuckoo Sandbox Setting.....	32
Table 5 System Specifications.	41
Table 6 Performance of Proposed Scheme using Random Forest Classifier with 10-Fold Cross-Validation.	48
Table 7 Performance of Proposed Scheme using SVM Classifier with 10-Fold Cross- Validation.....	49
Table 8 Performance of Proposed Scheme using Decision Tree Classifier with 10-Fold Cross- Validation.....	50
Table 9 Performance of Proposed Scheme using KNN & Naive Bayes e Classifier with 10- Fold Cross-Validation.....	51
Table 10 Kok. at el. Achieved Accuracy [27].	55
Table 11 Read Functions [53] of Ranking Features 46.	55
Table 12 Write Functions [53] Of Ranking Features 46.....	58
Table 13 Accuracies of Machine Learning Models.....	63
Table 14 Performance Comparison of Proposed Scheme and Kok S. H. [27] on 10-Fold Validations.	65
Table 15 Results of Detecting Dangerous APIs Using Cuckoo-Based API Calls with Random Forest, SVM, Decision Tree, KNN, And Naive Bayes Classifiers, Utilizing 10-Fold Cross- Validation.....	67

List of figures

Figure 1 Types of Ransomwares.....	3
Figure 2 Ransomware Attack Vectors [3]	4
Figure 3 Machine Learning Algorithm Model Analysis.....	22
Figure 4 Proposed Model Flow Diagram.....	27
Figure 5 Machine Learning Algorithm Code.....	28
Figure 6 Cuckoo Sandbox Interface.	29
Figure 7 Extraction of API Calls.	33
Figure 8 Confusion Matrix of Malicious and Benign Samples.	36
Figure 9 Data Structure of Signature Database Repository.	38
Figure 10 Flow Diagram of Ransomware Detection.	39
Figure 11 Proposed Model Execution in Operating System.....	43
Figure 12 The Receiver Operating Characteristic (ROC) curve obtained from the proposed method using the Random Forest classifier.	49
Figure 13 ROC Curve Obtained by Proposed Method for Decision Tree Classifier.....	50
Figure 14 ROC Obtained by the proposed method on the KNN classifier.....	52
Figure 15 ROC Analysis of the Proposed Method on Naïve Bayes Classifier.....	52
Figure 16 Features Importance.	55
Figure 17 ROC Curves of ML Models	64
Figure 18 The Detection Accuracy of The Suggested and Chart of S. H. Kok [27] Methodologies In Diverse Machine Learning Classifiers.	66
Figure 19 Comparison of Number of Features and Detection Accuracy between Proposed and Existing Approaches.	68

Abstract

The research introduces an innovative approach for the early detection of crypto-ransomware, a form of malware that encrypts a victim's data and demands ransom for decryption. Various detection techniques, including behavior-based analysis, API calls, system calls, network communication patterns, static and dynamic analysis, are commonly employed to detect ransomware. However, these techniques consist of various challenges such as adversarial attacks, classification errors, difficulties in detecting zero-day attacks, performance and scalability limitations, and limited efficiency of machine learning models for detection. The major challenges consist of larger number of (read/write) APIs calls that are employed for detection of ransomware. Further, it indirectly increases the complexity of the detection system. In this study, we developed an efficient ransomware detection method that utilizes a lower number of attributes. The proposed scheme adopts a two-level detection approach, combining a signature-based technique and sandbox analysis using machine learning (ML) algorithms and an application program interface (API) generated by Cuckoo Sandbox. The signature-based technique compares ransomware signatures with a database of known ransomware, utilizing hashing techniques such as SHA. The sandbox analysis, complemented by ML algorithms and the API, aims to identify ransomware prior to the encryption process. The scheme is evaluated using various ML classifiers, including Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbour (KNN), with an 80:20 training and testing ratio. In addition, the proposed scheme was assessed through 10-fold cross-verification. Experimental results demonstrate the proposed approach accurately identify 26 contributing read/write ransomware attributes with 98% accuracy. It also surpassing the existing detection techniques while employing a minimal number of attributes. Early detection of ransomware is vital in preventing data encryption, potentially saving victims from paying ransoms.

Keywords: APIs, Crypto-ransomware, Machine Learning, Malware, Cuckoo Sandbox.

1. Introduction

The first chapter of the thesis introduces the research, outlining its objectives, methods, and significance. It discusses the need for effective ransomware detection mechanisms and provides an overview of the subsequent chapters.

1.1 Overview

Ransomware locks digital data and modifies system logins to lock victims' resources. The attacker claims ransom from the target for acquiring access to its resources. In 2017, the WannaCry cyberattack infected over 200,000 systems across 150 countries [1]. Ransomware is playing a vital role in malware categories. Ransomware is still recognized as one of the top malwares that cyber security experts have placed on high alert. Despite decreased infection rates, the cost-effectiveness of ransomware has increased, as hackers target specific internal communication, demanding larger ransoms. Cybercriminals are drawn to ransomware for its cost-effectiveness.

Internet Usage: The widespread use of the internet has enabled global connections [2], making communication more convenient. However, this also increases the risk of international cyber-attacks. For instance, a cybercriminal can launch an offensive operation against a specific corporation from any location. Since there are geographical distances and varying regulations, authorities will be hindered in responding to such attacks that are made on a global scale.

Crypto-Currencies Popularity: The second reason is the widespread adoption of cryptocurrencies. The anonymity of the owner of this digital currency makes it hard for the

authorities to identify them, thus providing cybercriminals with an untraceable and safe route to extort ransom money [29].

Digitization: The third reason for the trend toward digital data storage is the cost-effectiveness of this method. As time progresses, the costs of digital storage are decreasing, and it has several advantages over physical data, including being searchable, occupying less space, and the ability to create backups or replicas.

Encryption Algorithms: Encryption, a critical security measure, can be employed to meet the security requirements of an information system, such as confidentiality, integrity, availability, authenticity, and accountability. Apart from availability, encryption helps achieve all of these objectives; however, ransomware can exploit encryption to take control of user data and demand a ransom.

Easy Accessibility of Ransomware: The accessibility of ransomware is a benefit for those engaged in cybercrime, as ransomware development kits are available on the Dark Web and can be downloaded without cost or bought for a reasonable price [5]. Moreover, a ransomware-as-a-service profit-sharing system exists between hackers, where one party creates the source code for the ransomware and disseminates it to potential targets. These aspects contribute to the growth and continued development of ransomware by cyber criminals.

1.2 Attack Vector

Most significant source of a ransomware infection is email [3]. Cybercriminals sends phishing emails containing ransomware and attachments or links to fake documents or legitimate-looking websites [2]. On clicking the attachment, the ransomware will launch on the victim's machine and automatically start to spread throughout the system. Advanced ransomware will

then perform reconnaissance on the victim's computer before contacting its developer, often known as the Command and Control (CC) Centre in cyber security. Once the ransomware has established a connection with the CC, it will share the gathered information regarding the victim and request the cryptographic keys depicted in Figure 1.

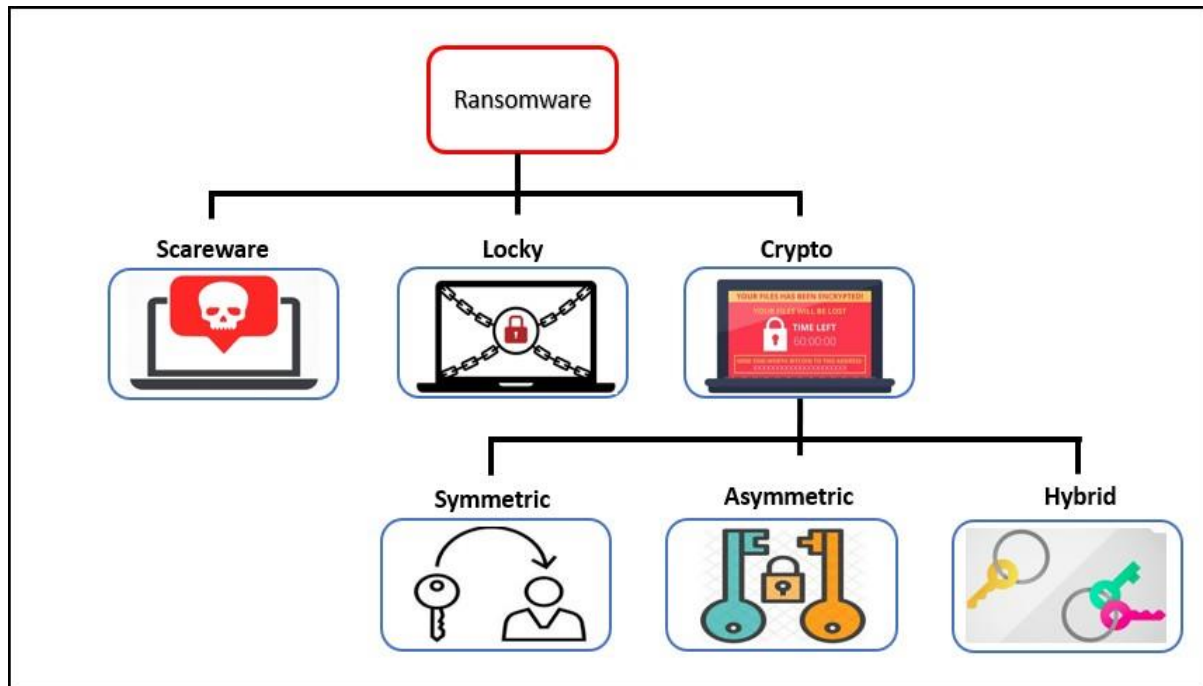


Figure 1 Types of Ransomwares.

Using these keys, the ransomware will encrypt the victim's crucial files, such as Microsoft Documents, images, media files, metadata, and more, only exposing itself to the victim once all vital files/ data have been encrypted [3]. According to Humayun et al. [3], 63.3% of ransomware infections are attributable to emails, with 35.7% relating to opening an attachment and 28.6% to clicking on a link, as illustrated in Figure 2. Moreover, 21.4% of victims report being unaware of how they became infected with ransomware [3]. Additionally, ransomware typically provides its victims with a limited timeframe to pay the ransom before the amount

increases.

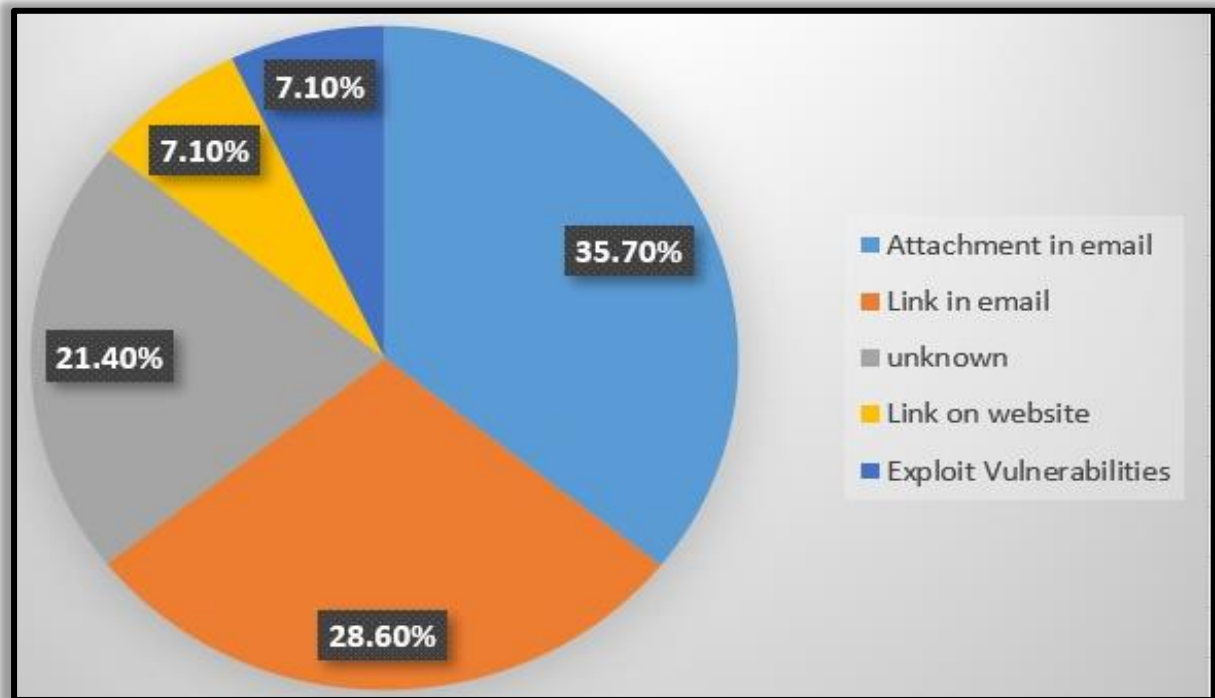


Figure 2 Ransomware Attack Vectors [3]

Another common vector of a ransomware attack is when the victim visits a malicious or infected website, either intentionally or inadvertently. This attack is known as malvertising, where a redirection strategy is employed when a person unknowingly uses an exploit kit to navigate from a secure to a dangerous website. Attackers insert an exploit kit into an inline frame, also referred to as an iframe, on a legitimate website, leading to this situation [4]. Windows OS is the most widely used operating system and has been the primary target of ransomware attacks. In terms of ransomware detection, the primary point to consider is the Application Program Interface (API) of Windows. Generally, APIs are the main interface with the operating system; every program utilizes API for execution. Thus, the analysis of API becomes an effective tool for ransomware detection. This study focuses on crypto-ransomware, which uses an encryption technique to encrypt the data and files of its victims. The chosen form of ransomware is crypto-ransomware, as the damage it causes is usually severe and irreversible

[5]. This is particularly true if the user's data is protected by a strong encryption technique, such as hybrid encryption. In general, the pre-encryption stage of a crypto-ransomware outbreak is crucial since recovery is challenging after encryption. Consequently, an early detection model for ransomware is necessary to protect against a ransomware attempt or reduce the harm of a ransomware attack.

1.3 Thesis Motivation

The focus of this research is on the detection of ransomware through the use of APIs, with a particular emphasis on pre-encryption detection in Windows platforms. There have been various approaches to ransomware detection using API calls and existing machine learning models, each with their own advantages and disadvantages. However, the identification of effective API calls and ensuring security can present challenges. Drawing inspiration from a previous study by Kok S. H. et al. [27], which also employed API-based ransomware detection at the pre-encryption stage, our research seeks to build upon this work and address major limitations and challenges associated with API-based ransomware detection.

This research is driven by the desire to enhance the performance of ransomware detection and minimize the number of features required for classification. Kok S. H. et al. [27] benchmark study, which utilized Random Forest classifiers, relied on 232 unique features. This study proposes two key objectives. Firstly, to detect ransomware at the pre-encryption stage through the usage of API. Secondly, to store the signature of newly identified ransomware in a database to facilitate early detection in the future.

1.4 Research Objectives

This study has the following objectives to detecting ransomware attacks.

1. Analyze the API behavior associated with crypto-ransomware attacks to identify key patterns and behaviors indicative of such attacks.
2. Develop a pre-encryption technique capable of detecting crypto-ransomware at an early stage, before it progresses to the encryption phase.
3. Enhance existing datasets for machine learning-based research, specifically focusing on ransomware detection. This dataset will be available online and can be utilized in future studies to improve detection models and algorithms.
4. Improve the signature-matching process by creating an enhanced repository of SHA-256 ransomware signatures. This repository will aid in the identification and categorization of ransomware strains, facilitating more accurate and efficient detection methods.

By achieving these objectives, the study aims to advance the field of ransomware detection by providing insights into API behavior, developing novel detection techniques, improving datasets for research purposes, and enhancing the signature-matching process for more effective identification of ransomware attacks.

1.5 Research Questions

This section describes the following research questions which are devised to perform this study:

- **Why is this research required?**

Windows ecosystem is continuously being threatened by malware which poses many security risks to the user's data. Since this data is usually of great value to the users, therefore, the users want some kind of protection in this regard.

CHAPTER 1: INTRODUCTION

There already exists a large set of detection methodologies, each providing its own benefits to the community. There is a need to see if those mechanisms can be modified to provide lighter detection approaches in terms of computational costs.

- **What is the significance of the study? And what steps are involved in the research?**

This study is about analyzing the importance of static malware analysis. The purpose of the study is to classify the malicious ransomware files/applications using the least number of static features. It will help malware analysts and the research community to quickly identify malicious ransomware applications. The study performs qualitative as well as quantitative static analysis of ransomware. To perform the study, we have broadly divided our approach into the following four steps:

- Samples collection and environmental setup
- Feature Extraction
- Classification results generation
- Report Writing

- **What are the aims of this study?**

The study mainly focuses on the following aspects:

- a) Discovering the set of minimum static features that can correctly classify a malicious ransomware application.
- b) Identifying the best classifying model among those considered for this study i.e., Random Forest SVM, Decision Tree, KNN and Naïve Bayes.

1.6 Problem Statement

Windows is a widely used operating system in homes and organizations worldwide, making it a common target for malicious actors seeking to exploit its vulnerabilities. With a vast number of applications available online, it can be challenging for users to distinguish between legitimate and malicious software. Malware authors often disguise their harmful software as benign applications or offer them for download alongside popular software, making it easy for users to unintentionally install them. Once installed, these malicious programs can steal sensitive information, damage data, or take control of the affected system.

Therefore, the problem statement of our study is as follows; “To determine significant static features set for windows-based applications to build a light-weight and efficient ransomware detection mechanism using supervised ML algorithms”.

1.7 Solution Description

The research provides a low-cost malware detection approach using the API-based static analysis of Windows applications and files. In this research, for the APIs extraction process, we have first analyzed ransomware files in Cuckoo Sandbox [10] to extract the APIs which are used for encryption functionality, from the dataset collected from VirusShare, VirusTotal, and TheZoo repository [11-13]. Following the extraction of APIs, they are filtered using the feature importance technique. Once the feature set is generated, a variety of machine-learning models are utilized to detect malicious applications.

1.8 Thesis Contribution

Our research strategy, titled "Ransomware on the Run: A Detection Approach using Effective API and Machine Learning Models," has successfully investigated and enhanced detection accuracies by incorporating established classifiers such as Random Forest and SVM. The proposed scheme offers the following contributions:

- The proposed approach achieves a high detection accuracy of approximately 98% while reducing the API feature set by approximately 88%.
- This significant improvement in accuracy demonstrates the effectiveness of the proposed approach in accurately identifying ransomware attacks while simultaneously minimizing the computational burden associated with analyzing a large number of API features.
- Introducing a lightweight malicious ransomware detection framework that offers an efficient and streamlined approach to detecting ransomware. This framework is designed to be lightweight in terms of computational resources required, making it suitable for deployment in resource-constrained environments while maintaining a high level of accuracy.
- The proposed approach is equally applicable with other existing machine learning classifiers, enabling its usage in various contexts. This flexibility and adaptability make it possible to integrate the framework into existing systems and workflows, ensuring its usability and effectiveness across different scenarios and environments.

1.9 Thesis Organization

The organization of the thesis is presented as follows. Chapter 2 throws light on previous work done related to static analysis and detection of ransomware applications. The Pre- Encryption

detection method concept for detecting crypto-ransomware is detailed in the chapter 3. The experimental setup is discussed in Chapter 4. Chapter 5 showcases the result of the experiment. This section also discusses the activities/events performed during the results collection. Chapter 6 is dedicated to the discussion and analysis of the experiment results. Lastly, chapter 7 sheds light on the conclusion with possible directions for the future.

1.10 Summary

In this chapter, basic concepts are discussed regarding ransomware analysis such as static and dynamic analysis implied in the detection of malicious applications. It provides an overview with the aim and scope of the thesis. Further, it presents the main objectives of the research work with the overall thesis organization. In the next chapter, we will look at the literature review that has been conducted for this thesis.

2. Literature Review

The second chapter of the thesis explores the topic of related research and terminology. This includes studies and investigations conducted by scholars in the field that are relevant to the present study and have contributed to the development of a novel solution.

2.1 Overview

In literature various studies and technical reports have highlighted the threat of malware and its impact on different organizations. With the advancement of technology, such as IoT and cloud computing, which enable global connectivity, there has been an increase in malware attacks [6]. This can be a potential problem if end-users are unaware of the threats posed by these cutting-edge technologies [7]. Ransomware is the most prevalent malware family targeting organizations and end-users on a large scale [8]. This increase is due to a direct connection between the victim and attacker, with the latter demanding a ransom from the victim. Data reveals that 37% of victimized organizations did not get their data back even after paying the ransom [9]. Once ransomware is executed in a system, it encrypts all data using encrypted algorithms and keeps the decryption key secret, thus making the data inaccessible to the victim [14]. Encryption can be used for both beneficial and malicious purposes; it can be used to preserve confidentiality and protect important data, but attackers can also use it as crypto-ransomware to prevent access to data [15].

The current anti-malware solution like anti-virus, IDS, etc., which works on signature-based detection, are not able to detect any zero-day attacks because the new malware signature is not being updated to the signature database [16]. Cybercriminals show a keen interest in ransomware, a new variant of ransomware being created that bypasses the anti-virus software

[17]. Ransomware as a service (RaaS) allows reaching out to non-technical users with malicious intentions [18]. A severe countermeasure is needed to address this dangerous level of threat to the organization and end user. Extensive work is being done to detect ransomware at an initial level; some techniques are based on static and dynamic analysis [19].

2.2 Machine Learning

Machine learning is defined as: “a technology that focuses on developing computer algorithms that are capable of emulating human intelligence by incorporating ideas from neuroscience, psychology, computer science, control theory, probability and statistics, information theory, and philosophy” [38].

Machine learning has successful applications in many fields such as robotics, computer vision, entertainment, and medicine. The aim of this technology is to humanize computers by educating themselves about the surrounding environment and previous experiences, with or without any supervised learning [38].

2.2.1 Machine Learning Algorithms

This section presents various machine learning algorithms including Support Vector Machines (SVM), Random Forest (RF), Decision Tree (DT), K-Nearest Neighbour (KNN), Naïve Bayes (NB), and Voting Classifiers (VC).

a) Support Vector Machine (SVM)

Support Vector Machines (SVM) is a machine learning algorithm that aims to classify data points into two or more categories. It does so by finding a boundary plane that separates these categories based on the variables included in the model [38].

b) Random Forest (RF)

Random Forest (RF) is an algorithm that uses a combination of decision trees to construct models that represent complex and nonlinear functions. It builds a collection of decision trees, each one using a random subset of features and data to create an ensemble with low bias and low variance [39].

c) Decision Tree (DT)

The Decision Tree (DT) algorithm is a flowchart-like structure used for making predictions based on feature values. It consists of decision nodes and leaf nodes, where each branch represents a possible decision or outcome. DT is useful for classification and regression tasks, but it may lead to overfitting if not pruned correctly.

d) K-Nearest Neighbour (KNN)

K-Nearest Neighbour (KNN) is an effective algorithm used for classification and regression. It classifies new cases based on similarity to existing data, without explicitly building a model. It is suitable for tasks with many features and limited data. However, it can be computationally expensive.

e) Naïve Bayes (NB)

Naïve Bayes (NB) is a simplified derivative Bayesian network that uses an assumption of feature variable independence for classification. This method is useful in high-dimensional spaces, but it can result in inaccurate probability estimates due to the unrealistic independence assumption [38].

f) Voting Classifier (VC)

Voting Classifier (VC) is an ensemble method that combines multiple models to make predictions. It can be used for classification and regression tasks and is believed to be more accurate and robust than individual models. However, it may be computationally expensive.

2.3 Related Work

The static analysis includes source code analysis in which ransomware will not be activated. Several techniques are available, and analysis is performed by obtaining the opcode from the source code. The opcode will be used to generate an n-gram code. These n-gram codes will undergo machine learning training to create a prediction model for ransomware detection [20]. Recently, another approach that based on the search for ransom messages; indirectly leaving the victim with one ransomware execution [21]. Similarly, another static analysis approach examined the executable file headers for ransomware detection. Moreover, researchers also proposed a ransomware detection solution by analyzing the source code and deriving rules [22]. The fundamental issue with static analysis is that the examined source code may be flawed due to obfuscation techniques.

The ransomware dynamic analysis requires the execution of the malware in a protected environment such as a sandbox. This analysis is very effective due to direct interaction. Further, intelligent malware can also predict the execution environments whether it is real-time or in a sandbox, and tuned to be inactive for the sandbox. Recently, in [23] authors proposed the detection methodology by analyzing the encryption process. As encryption process involves repetitive activities/patterns that can further be analyzed at file system levels to predict the ransomware patterns. Tracking the encryption frequency is also a technique to distinguish between attack and legitimate encryption. Another approach based on dynamic analysis is using system API calls to track ransomware interactions with the operating system [24]. A multi-

CHAPTER 2: LITERATURE REVIEW

layer detection technique called ransomWall combines two types of analysis [25]. Recently, other detection solutions included authentication-based access to essential files [26], tap-using honey files, and sustainable awareness.

In [27] author studies crypto ransomware behaviors through function calls or APIs (Application Program Interface). They provided a two-level ransomware detection model; Level one detects ransomware through signature-based and the second level through machine learning algorithms using APIs calls. They used the RISS [41] dataset as a reference which has 30,967 apps, including both benign and malicious applications. After analyzing this dataset, the author extracted only ransomware features and created a new dataset with 232 APIs features and 1800 entries. The RISS dataset comprised multiple features, while the new dataset focused on the pre-encryption function. The author uses a machine learning model to detect the ransomware using APIs features and achieved 100% detection accuracy on the Random Forest classifier. Further, after the detection of ransomware, its signature is stored in a database for future detection. For experiments, Cuckoo Sandbox, Windows 10 Pro, and Ubuntu 18.04 platforms are used. However, a number of features can further be optimized by reducing the insignificant features while maintaining similar detection accuracy. Recently, another approach [28] presented a similar model [27] on APIs-based detection while reducing the number of features from 232 to 206. However, the detection accuracy was reduced by 1% as compared to [27] on the K-NN classifier. Meanwhile, the proposed model is only applicable to detect known ransomware. In contrast, our proposed model employs only 26 features that can detect both known and unknown ransomware.

Zakaria et al. [29], produced a similar framework for the detection of crypto-ransomware called RENTAKA. The author created a crypto-ransomware dataset using the RISS dataset as a reference. They successfully reduced the feature to 80 and dropped the 3% (97.03%) detection accuracy as compared to [27] using Support Vector Machine (SVM). They also employed

CHAPTER 2: LITERATURE REVIEW

various classification models: SVM, Random Forest, Naive Bayes, KNN, and J48. However, this study has not provided any detail about the proposed model and even did not discuss what type of encryption APIs were actually targeted.

Another study on the APIs-based ransomware detection method proposed by Anand et al. [30] uses the feature importance technique. The author performs API call analysis on ransomware behavior to look for a specific pattern. This study collected 653 executable files, including both benign and malicious applications. After analysis of APIs, they perform feature ranking and create a dataset. The collected dataset was given to the machine learning model by applying four classifiers: Random Forest, C 5.0, AdaBoost, and SVM. The highest obtained detection accuracy was reported at around 95.38% using the AdaBoost classifier. However, the contribution consists only of employing feature ranking. The CRED Alqahtani et al. [31] also used the same detection approach using machine and deep learning. Furthermore, RMOC [32], MELCOR [33], RANSID [34], ROPsight [35], RaFS [36], and RaRE [37], are the recent techniques to decrease ransomware attacks. Table 1 shows the general techniques that are proposed by the researchers to detect ransomware.

Table 1 Crypto-Ransomware Techniques and Detection Methods.

Framework	Pre-Encryption Methods	Identification Features	Dataset
Elderan [47], 2016	Dynamic analysis of Ransomware by executing an application for only 20 seconds	30,967 functions named (system operations, system calls, APIs, performance)	RISS

CHAPTER 2: LITERATURE REVIEW

PEDA [27], 2020	Identification of CRYPTO APIs at the start	232 APIs	RISS
CRED [31], 2020	APIs and IRP-based correlation and detection	IRP and APIs features	Centric- Process Centric-Data
RMOC [32], 2021	Machine learning-based classification and clustering of ransomware attacks	Malware samples and ransomware attack data	Not specified
MELCOR [33], 2021	Machine learning-based network traffic analysis	Specific ransomware families (e.g., Ryuk, Maze)	Not specified
RANSID [34], 2021	Behavioral analysis and machine learning-based detection of ransomware attacks	Network and endpoint behavior	Not specified
ROPsight [35], 2021	Detection of ransomware attacks using Return-Oriented Programming (ROP)	ROP-based techniques	Not specified
RaFS [36], 2022	File system monitoring and analysis using machine learning and rule-based techniques	Suspicious file access patterns	Not specified
RaRE [37], 2022	Machine learning-based detection of ransomware attacks	System call sequences	Not specified

CHAPTER 2: LITERATURE REVIEW

RENTAKA [29], 2022	Dynamic analysis of crypto- ransomware	80 APIs	RISS
-----------------------	---	---------	------

The table presents different ransomware detection approaches and their merits and demerits. Elderan relies on dynamic analysis but missed sophisticated attacks. PEDDA identifies CRYPTO APIs but overlook non-API-based ransomware. CRED combines APIs and IRP correlation but limited to specific features. RMOC uses ML for classification and clustering but relies on training dataset quality. MELCOR analyzes network traffic but didn't discussed about offline ransomware. RANSID applies behavioral analysis and ML but faces false positives and emerging pattern challenges. Similarly, ROPsight detects ROP-based attacks but is limited to techniques. RaFS monitors file system access but may struggle with distinguishing legitimate and malicious patterns. RaRE uses ML on system call sequences but requires significant computational resources. RENTAKA analyzes APIs but missed complex ransomware behavior. Consider these trade-offs when choosing a detection approach.

While the existing techniques for detecting and identifying ransomware attacks have made significant advancements, there are still some limitations and research gaps that need to be addressed. Many of the current approaches focus on specific aspects of ransomware detection, such as dynamic analysis, machine learning, or behavior analysis. However, they lacks of comprehensive coverage or fail to consider evolving ransomware techniques. Additionally, the lack of standardized datasets and evaluation methodologies makes it challenging to compare and validate the effectiveness of different techniques. Furthermore, some approaches may suffer from false positives or false negatives, impacting the accuracy and reliability of detection. To improve the effectiveness of ransomware detection, future research should strive to develop integrated frameworks that consider multiple detection dimensions and leverage diverse datasets. Additionally, efforts should be made to enhance the evaluation process by

establishing standardized benchmarks and metrics for assessing the performance of different detection techniques. Addressing these limitations and research gaps will contribute to the development of more robust and reliable ransomware detection methods.

Generally, the optimal approach to detect ransomware before it is executed and thus, minimize the damage to the organization. In this study, a dynamic solution is proposed to ensure early detection, by providing signature matching with a maintained database, and the use of API calls and machine learning. Once files are encrypted, the data cannot be recovered, even upon the removal of ransomware from the system.

2.4 Summary

The chapter covers the background and related work of the thesis. It includes a critical analysis of existing research and schemes used in the literature, which aids in formulating a solution to the identified problem. The next chapter will discuss the research methodology followed during the thesis.

3. Research Methodology

Chapter 3 outlines the research methodology used in this thesis, which involves collecting malware samples, constructing a feature set, and using machine learning classifiers for malware detection. The methodology follows a structured approach and utilizes methods such as data collection, preprocessing, feature extraction and selection, and machine learning model selection and evaluation.

3.1 Introduction

Research refers to a systematic and organized approach aimed at investigating a particular issue or field of interest to generate new knowledge and insights [40]. It involves defining and redefining a known problem, formulating a hypothesis, collecting and analyzing data, making assumptions, deriving conclusions, and testing the conclusion to verify the hypothesis [42]. In essence, research is conducted to answer a specific problem and identify appropriate solutions [41]. Its ultimate objective is to contribute to the advancement of knowledge in a particular field or discipline, whether through the development of new theories, testing of existing theories, or discovery of new facts and information. Overall, research is a rigorous and structured process that requires a disciplined and methodical approach to obtain reliable and valid results.

3.2 Research Methodology

This section presents the research methodology. The proposed approach revolves around a pre-encryption ransomware detection system that focuses on extracting encryption APIs usage from

ransomware applications. The emphasis lies in analyzing a specific set of features that have demonstrated effectiveness in distinguishing and improving the detection rate of ransomware. To enable classification, the system utilizes various models, including Random Forest (RF), Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbour, Naïve Bayes, and Voting classifiers. In this approach, features are manually selected based on their significant influence on the effectiveness of ransomware detection. The study encompasses the following key components:

- A. The first component involves the collection of both malicious and benign samples, which is a crucial step in developing an effective ransomware detection system.
- B. The second component involves identifying and constructing a feature set that can distinguish ransomware from benign applications.
- C. The third component involves filtering, finalizing, and extracting the most effective features from the dataset.
- D. The fourth component involves utilizing supervised machine learning methods to classify potential Windows malware by training the models on the filtered features dataset.
- E. After performing the aforementioned stages, a report of the malware analysis summary is generated which can be used to carry out further evaluations.

3.3 Detection at the Pre-Encryption Stage

In this section, we will discuss the pre-encryption stage of ransomware detection which is comprised of two levels. The first level is the analysis of system call or Application Program Interface (API) to identify ransomware behavior with the assistance of a trained machine learning algorithm. This level allows for the detection of previously unknown ransomware. The second detection level is the identification of known ransomware with the help of a signature database. The data flow of the pre-encryption framework for machine learning is depicted in Figure 3. Subsequent sections will provide a detailed explanation of how the pre-encryption model functions in real-world scenarios. It has been observed that the primary vector of a ransomware infection is an email attachment. The pre-encryption model enables users to inspect a file before opening it to detect whether it contains ransomware.

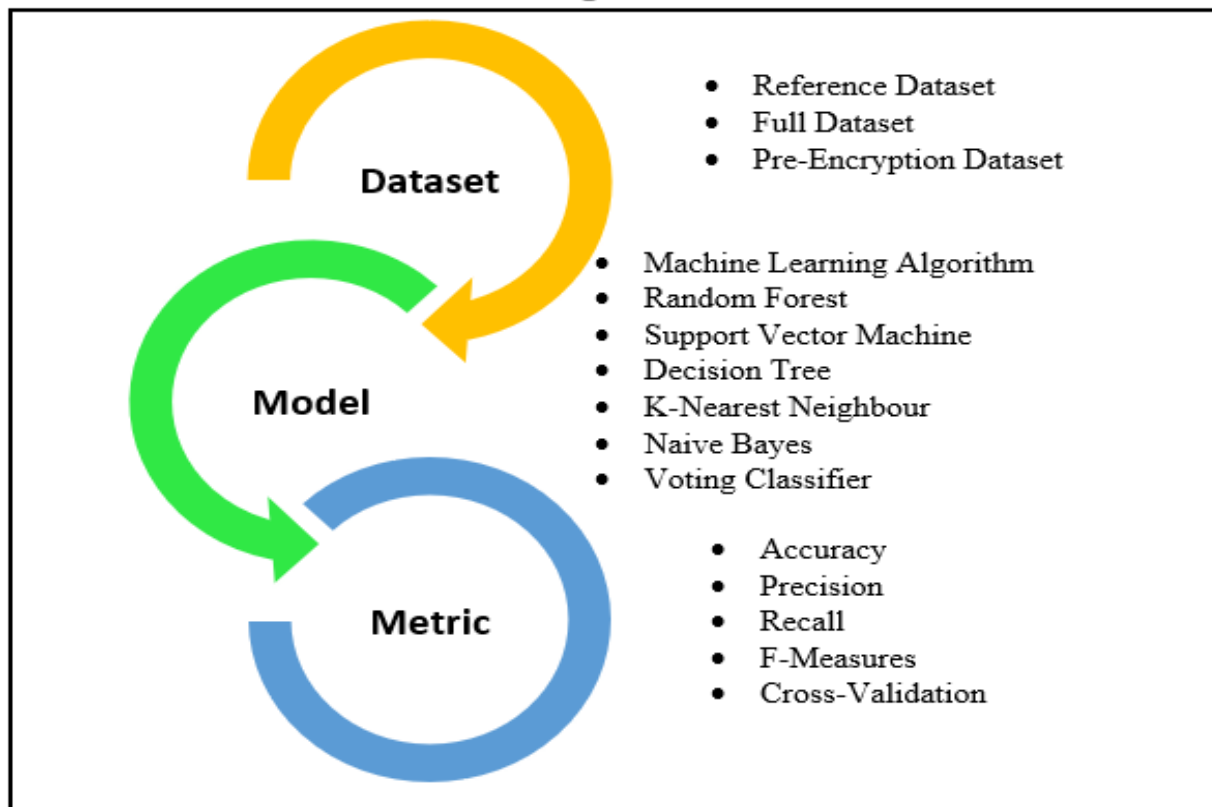


Figure 3 Machine Learning Algorithm Model Analysis.

CHAPTER 3: RESEARCH METHODOLOGY

Before sending the file to the pre-encryption model for analysis, the user must first select the file. Following file submission, this model will use SHA-256 to create the file's hash signature. Any file that can be hashed into a single character with a set length and unique to its contents. SHA-256 specifies that 256 or 64 characters of signature data will be generated (4 bits of space fulfills 1 character). The hashes of known ransomware that are stored in the Signature Archive will be compared to generated signature for matching. Comparing the entire file's content is a slower process than verifying the hash fingerprint.

Furthermore, a hash value requires significantly less storage space. If a match is found, the pre-encryption approach would alert the user of the discovery and store the file separately. However, this technique is limited and can only detect known ransomware. In the context of an unmatched file, it will be sent to the Cuckoo Sandbox for unidentified malware. The file will then be read by Cuckoo Sandbox in a secure virtual setting to monitor its behavior. Upon completion of processing, Cuckoo will generate an analysis report of the file. In this study, we will discuss APIs that are employed to open a file. The pre-encryption model identifies APIs before the encryption function call. The model locates these APIs by searching for the word "crypt." If the keyword is present, the API is stored in a file and the process will proceed to the next step; otherwise, the process will continue until the end. After being converted into data format, the retrieved APIs are stored in a comma-separated value (CSV) format. The ML model will employ the extracted API features for classification purposes. The pseudocode can be found in Algorithm 1.

The study employed the Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbour (KNN), and Naive Bayes (NB) models to detect the presence of ransomware in files after discretizing the data. In case ransomware is detected, the file's signature is stored in the Signature Repository. On the other hand, if the pre-encryption model determines that the file is not ransomware, the user will receive a notification indicating that

the file can be considered safe. The Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbour (KNN), and Naive Bayes (NB) models were utilized, following the discretization of the data, to detect the presence of ransomware in files. If ransomware is identified, the pre-store the file's signature in the Signature Repository. If the pre-encryption model determines that the outcome is not ransomware, the user will be notified that the file is safe to consider.

Algorithm 1: *Pseudocode of Applying of Machine Learning Classifiers.*

```

-----
// Adding Trained Dataset to Machine Learning models
i.    Upload trained dataset model in CSV format.
      Function: upload_trained_dataset_model(file_path)
      =====
      dataset = read_csv(file_path)
      Return dataset
ii.   Debug for missing conditions.
      Function: check_missing_conditions()
      =====
      If trained_dataset is None Then
          display_error("Trained dataset model not found.")
          Return False
      Else
          Return True
iii.  If the model is not found, display the error.
      Function: display_error(message)
      =====
      Print message
iv.   Search signature in the database.
      Function: search_signature_in_database()
      =====

```

signature = search_database()

Return signature

- v. Else use a trained model for the prediction of new data.

Function: predict_using_trained_model(trained_dataset, new_data)

=====

model = train_model(trained_dataset)

prediction = model.predict(new_data)

Return prediction

- vi. Print the outcome and analysis.

Function: print_outcome_and_analysis(prediction)

=====

Print "Prediction:", prediction

analysis = perform_analysis(prediction)

Print "Analysis:", analysis

3.3.1 Discretization

Discretization is an essential pre-processing phase aimed at converting continuous information into discrete values to enhance the predictive accuracy of machine learning algorithms [43]. This process involves the utilization of rules-based and tree-based algorithms. While there are numerous techniques available for discretizing continuous variables, the ideal approach should result in a meaningful division of the variable, aligning with the desired allocation of classes. The goal is to find an optimal discretization strategy that appropriately represents the underlying data distribution and facilitates effective classification.

3.3.2 Ransomware Signature

Hashing is a form of encryption with a one-way function [46]. It can only produce digest codes and cannot reverse the process to acquire plaintext. Even a tiny change in the content could result in an entirely different digest code. The comparing of generated digest code is usual

practice to verify the message's integrity. Therefore, in this study, we employed hashing to create distinctive ransomware signatures. Calculating the message length is necessary for SHA-256 to work properly. Extra bits are appended once the message size increases or decreases from 64 bits to keep the message a multiple of 512. In this case, the first bit will be zero of the appended bits. The modulo of the initial message with 232 will be used to fill these final 64 bits. The compression algorithm is applied to the resulting 512 bits. On the generation of the last digest code, the hashing operation is repeated 64 times to generate the hash.

3.3.3 Malware Detection by Employing Machine Learning Models

Random Forest contains Decision Trees that are combined in the bagging process [44]. The DT's prediction has an architecture similar to a tree growing from the root, moving towards branches, and ending up in the leaves. Features are randomly selected to form multiple DTs, and the final prediction is made by taking an average of all DT results. [45]. The research discovered that the RF method produced good results in identifying malware [45]. In addition, this study also used the Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Naive Bayes (NB), and Voting Classifier. Additionally, we discovered that these classifiers could perform well, particularly for discrete-type datasets. Due to this reason, we suggested the discretization of data in our pre-processing stage.

3.4 Research Methodology

This section presents the methodology for ransomware detection proposed in this study, which is illustrated in Figure 4. The proposed approach is divided into five groups: Sample Collections, Cuckoo Sandbox, Data, Machine Learning, and Database. Each group is discussed in further detail below

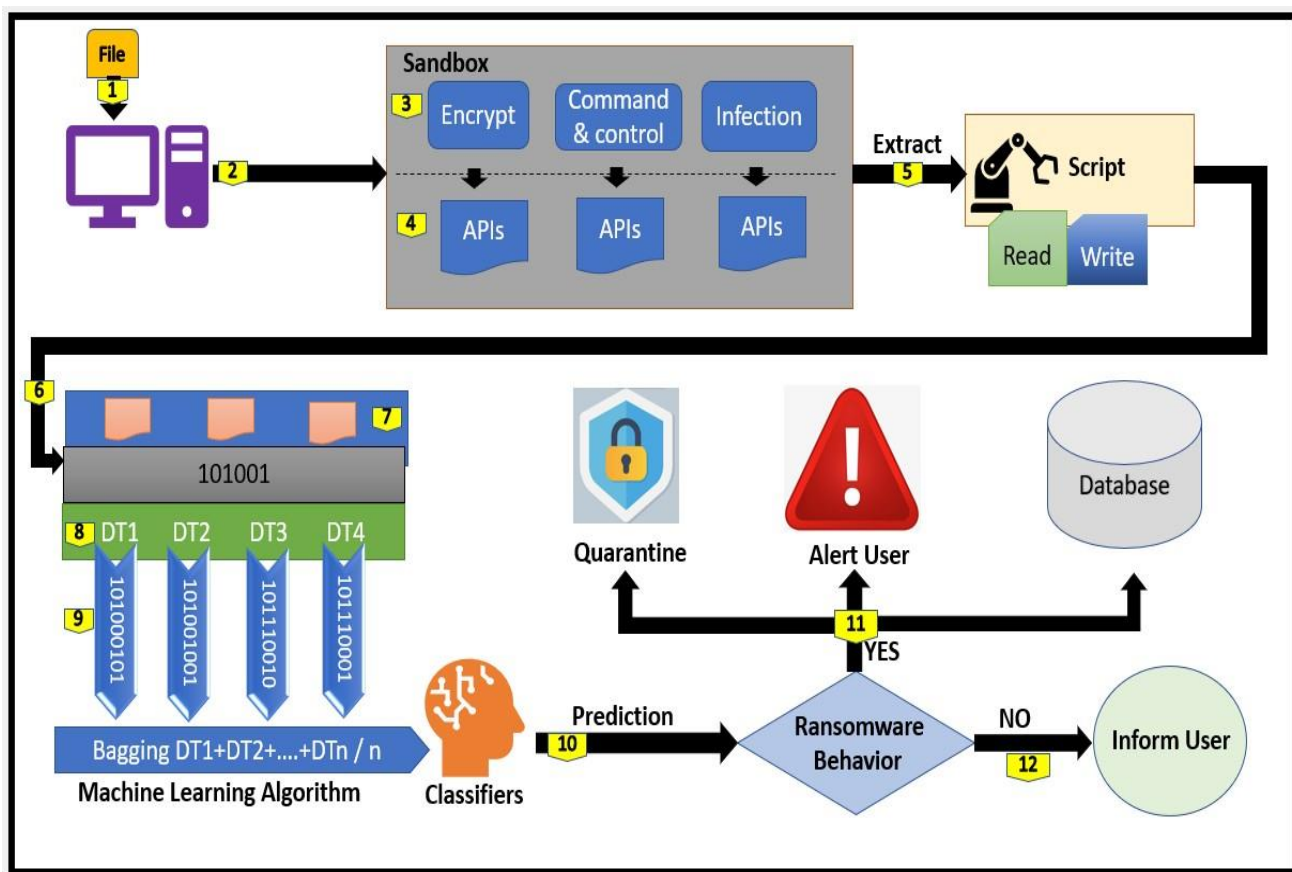


Figure 4 Proposed Model Flow Diagram.

3.4.1 Ransomware Samples

The section will discuss the collection of ransomware data. It was initially started with the Sgandurra et al. [47] dataset that consisted of 582 ransomware from eleven different categories, along with 942 benign applications (or Goodware). This dataset covers Locky- ransomware, and crypto-ransomware, which have different file operations, strings, directory activities, deleted file extensions, registry key actions, and API metrics that are reported. There are approximately 30,967 elements in the dataset that can be classified into seven main categories. Analyzing each application for 30 seconds in cuckoo sandbox allows its features to be recorded. An important component is an API, which shows how an application interacts with the operating system.

CHAPTER 3: RESEARCH METHODOLOGY

In literature, various approaches have been used with the above dataset for analysis, where the API-based attributes have had a significant impact on detection rates [5]. Therefore, API-based features were also utilized. Dataset samples from Sgandurra et al. (2016) [47] were downloaded from VirusShare (VS). To further enhance the dataset with novel ransomware samples, updated samples were acquired from both VirusShare and theZoo (TZ) repository. VS was available to security experts, incident investigators, forensic investigators, and curious individuals, and contained 34,235,166 samples, which required registration and verification by the website administrator. Additionally, 357 fresh crypto-ransomware samples were obtained from different websites, with an additional 56 crypto-ransomware samples collected from TZ. As a result, 995 software samples were executed on Cuckoo Sandbox, with only 904 of them providing relevant reports. We employed the Cuckoo Sandbox to execute the collected samples to choose the pre-encryption samples, these general steps are mentioned in Figure 5. As a result, the Sandbox report lists the order in which the sample called each API in different iterations.

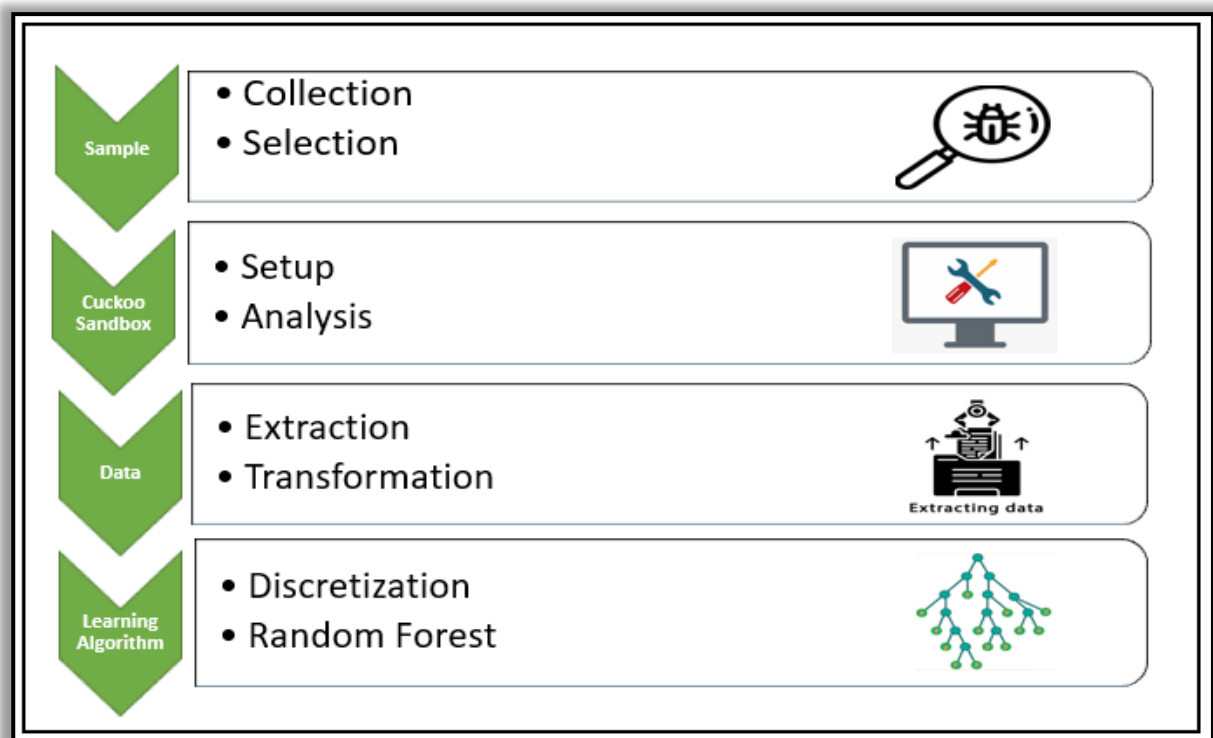


Figure 5 Machine Learning Algorithm Code.

For example, an API sequence with the keyword "crypt" was chosen. It was discovered that APIs with the keyword "crypt" execute encryption functions, as indicated in Table 2. There were 205 samples for encryption.

3.4.2 Cuckoo Sandbox

The malware analysis Cuckoo sandbox can monitor a program's execution activities. Each API call made by the sample once it has been entered into the sandbox is recorded by Cuckoo Sandbox its interface is shown in figure 6. Following that, Cuckoo will produce a report with a list of all API calls. The Cuckoo was installed on the Haier (Intel(R) Core (TM) m3- 7Y30 CPU @ 1.00GHz 1.61GHz, 8GB RAM, Ubuntu 20.04, and Windows 10 Professional), however configuring Cuckoo was quite complex. It requires several prerequisite programs for the complete installation process. It was necessary to have Python 2.7, Python 3, MongoDB, Postgresql, XenAPI, TCPdump, and VirtualBox. After that, Ubuntu 20.04 was able to successfully install Cuckoo Sandbox 2.0.7.

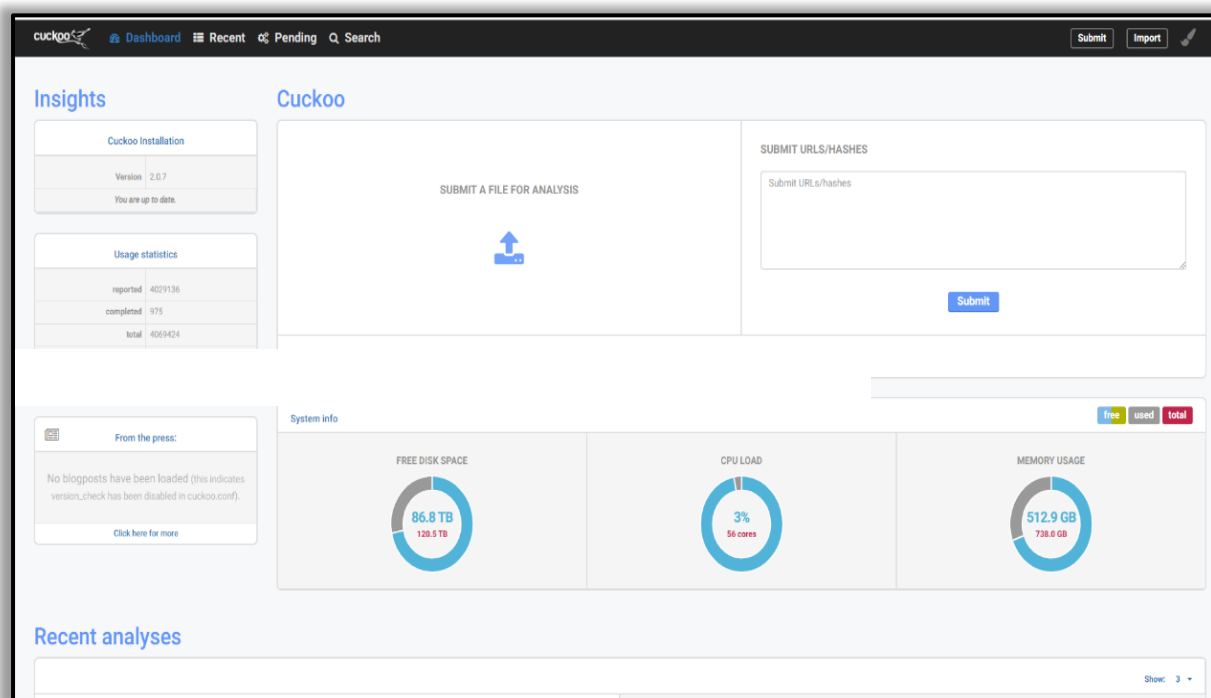


Figure 6 Cuckoo Sandbox Interface.

CHAPTER 3: RESEARCH METHODOLOGY

To ensure that the analysis could be done securely, we had to build up the virtual system using Virtualization before using Cuckoo Sandbox. Windows 7 was installed on the guest machine set up in VirtualBox (Win7).

Windows 7 was the most widely used operating system before Windows 10 Pro (Jan 2020-Jan 2021) [48], and Windows is the target of the majority of targeted ransomware.

Table 2 APIs with "CRYPT" Keywords.

No.	API	Description
1.	CryptAcquireContextA	A specific cryptographic service provides (CSP) was requested by CryptoAPI to supply a key container.
2.	CryptAcquireContextW	
3.	CryptExportKey	To securely export one symmetric encryption key or two asymmetric cryptographic keys out of a specific CSP
4.	CryptDecodeObjectEx	To decode using the lpzStructType parameter's definition of the structure type
5.	CryptEncrypt	Using the hkey option to conduct the encryption function with the encryption key supplied by CSP.
6.	CryptCreateHash	The information flow enabled secure session communication to be hashed to start.
7.	CryptGenKey	Create a public or private key pair for asymmetrical cryptography or a random encryption key for symmetric encryption.

8.	CryptHashData	To include details in a hash item that supported long or continuous transmissions.
----	---------------	--

We set up a Windows 7 guest machine in VirtualBox and followed the network configuration requirements in Table 3 to connect the guest machine to the Cuckoo Sandbox host machine. We then took a screenshot of the guest's computer to ensure that each time we analyzed a sample, we could restart the guest's PC without any issues. The default settings for Cuckoo Sandbox were altered as detailed in Table 4. The guest machine had to be initialized in VirtualBox before a sample analysis could be conducted in Cuckoo Sandbox, after which the Cuckoo Web service could be initiated. On average, it took 3 minutes to process each sample.

Table 3 Machine Configuration.

Steps	Description
i.	Network Address Translation (NAT) configuration.
ii.	Enable DHCP.
iii.	Disable adaptor (Host-Only).
iv.	Win7 machine firewall also is disabled.
iv.	Create User Account Control (UAC).
v.	Enabled IPv4 on the guest machine.
vi.	Install Python on both machines.
vii.	Install other libraries i.e., Pillow.
ix.	Copy Cuckoo Sandbox agent.py to Win7.
x.	Runs OS agent.py in desired OS.

By executing the program, we identify all API calls made by the sample which has been inputted into the Cuckoo Sandbox and recorded. We then obtain a complete report from Cuckoo listing all API calls, which can be done in two ways; either by downloading it directly from the browser or by copying the report file from the address “/.cuckoo/STORAGE/ANALYSIS/2/REPORTS/report.json.”

For this investigation, the latter method was chosen as it creates a single “rar” file which is compressed and contains multiple files, with the sample calling all APIs listed in a single “JSON” file.

Table 4 Cuckoo Sandbox Setting.

Steps	Description
i.	Start and enable the DB database in DB report.conf file.
ii.	The guest machine will be the same name as “Win7” in storage.conf file.
iii.	Configure the IP statically in both VirtualBox.conf and WIN/ Snapshot.

3.4.3 Extraction of Data

It is necessary to retrieve only pertinent data from the Cuckoo Sandbox report, as the information provided was extensive and unrelated to our investigation. All pre-encryption Application Program Interfaces (APIs) were monitored for their behavior prior to calling any encryption functions. To ensure that our script was able to extract the required data successfully, we developed a script to extract each API and compared the result against the “apistats” found in the Cuckoo report as shown in figure 7. After confirming that our code was generating the same outcome, “crypt” was established as the extraction stop point. The pseudocode for extracting the “crypt” call is shown in Algorithm 2.

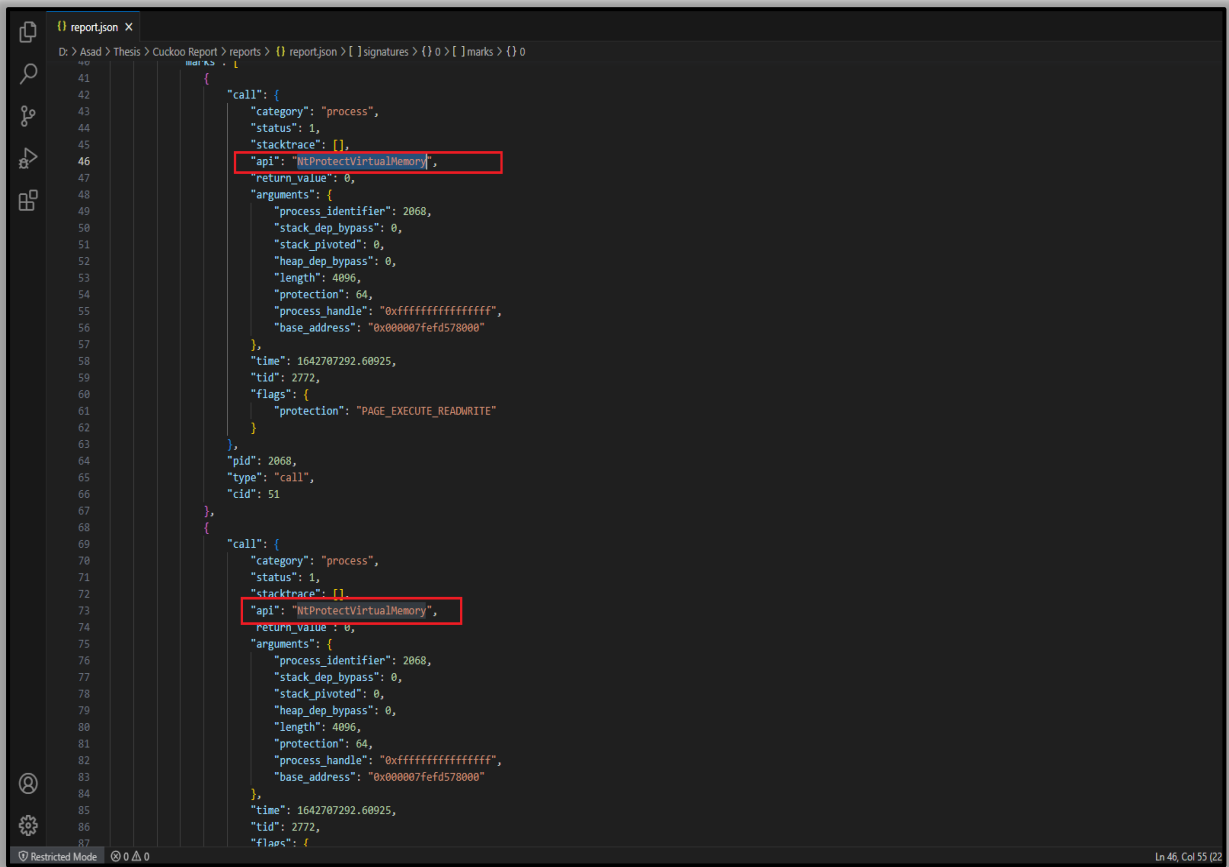


Figure 7 Extraction of API Calls.

Algorithm 2: Extraction of Data from Cuckoo Report.

// Data Extraction

- i. Open and read the desired file.

Procedure data_extraction(file_path)

keywords = ["crypt"] // Add more keywords as needed

extracted_words = []

file = open(file_path, 'r')

lines = file.readlines()

- ii. Skip the first line and Move to the subsequent line.

for line in lines[1:]:

line = line.strip()

=====

iii. Identify and look for “API” functions.

if "API" is found in line then

=====

iv. When API is found, go to the subsequent line.

next_line = lines[lines.index(line) + 1].strip()

=====

v. API has the keyword “crypt,” string or function then halt.

if any keyword in keywords is found in next_line then

exit loop

else

=====

vi. Else words will be stored.

Extracted_words.append(line.split())

=====

vii. Else go to the subsequent section.

Continue

=====

viii. Stop loop.

Break

=====

ix. Exit.

return extracted_words

=====

```
// Example usage:  
  
file_path = "path/to/your/file.txt"  
  
extracted_data = data_extraction(file_path)  
  
print(extracted_data)
```

The data processing outlined above resulted in a "txt" format and converted into a "CSV" file format to enable the ML algorithms to utilize it. This "CSV" had 235 entries, with the sample name appearing in the first column, the second column indicating whether the sample was benign (zero) or ransomware (one), and the third column containing a source indication for the sample. After the APIs from all the samples had been extracted, 232 APIs were visible in the fourth column and beyond. Consequently, three datasets were created based on the retrieved data. The first dataset included the APIs for all samples; the second dataset, containing only the malware with encryption functionality, was only used to extract the pre-encryption APIs; and the third dataset was a subset of Sgandura et al. (2016), with the addition of some new benign and ransomware applications, which was used to extract the necessary API dataset focusing only on encryption methods.

3.4.4 Machine Learning Algorithm

The ML algorithm is an essential element that may distinguish between known and unidentified ransomware assaults. The method was broken down into two steps: first, the data was discretized; next, Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbour (KNN), Naive Bayes (NB), and Voting Ensemble were used to train the prediction models. We used all three datasets to evaluate the ML model's efficacy in two different training-to-testing ratios. The data for both training and testing were divided by the ratio's original setting of 80:20. The second ratio, 70:30, allocated 30% 70% training, and 30% testing. Additionally, 10-fold verification tests were also run.

Accuracy, True Positive Rate (TP Rate), False Positive Rate (FP Rate), Precision, Recall, F-measure, and Cross-Validation were the evaluation measures employed. The TP Rate measures how effectively the machine learning algorithm model can estimate the positive accurately, whereas the FP Rate measures how well it predicts the positive wrong. Accuracy measures how effectively our model identified ransomware. The Accuracy of the affirmative prediction is measured by precision. Recall measures how accurately a favorable forecast was made. The balance of precision and recall is determined by the F-measure. The area under the curve for TP rate vs. FP rate is known as ROC, while the area under the curve for precision versus recall is known as PRC. To visually demonstrate the performance of our SVM model, we present

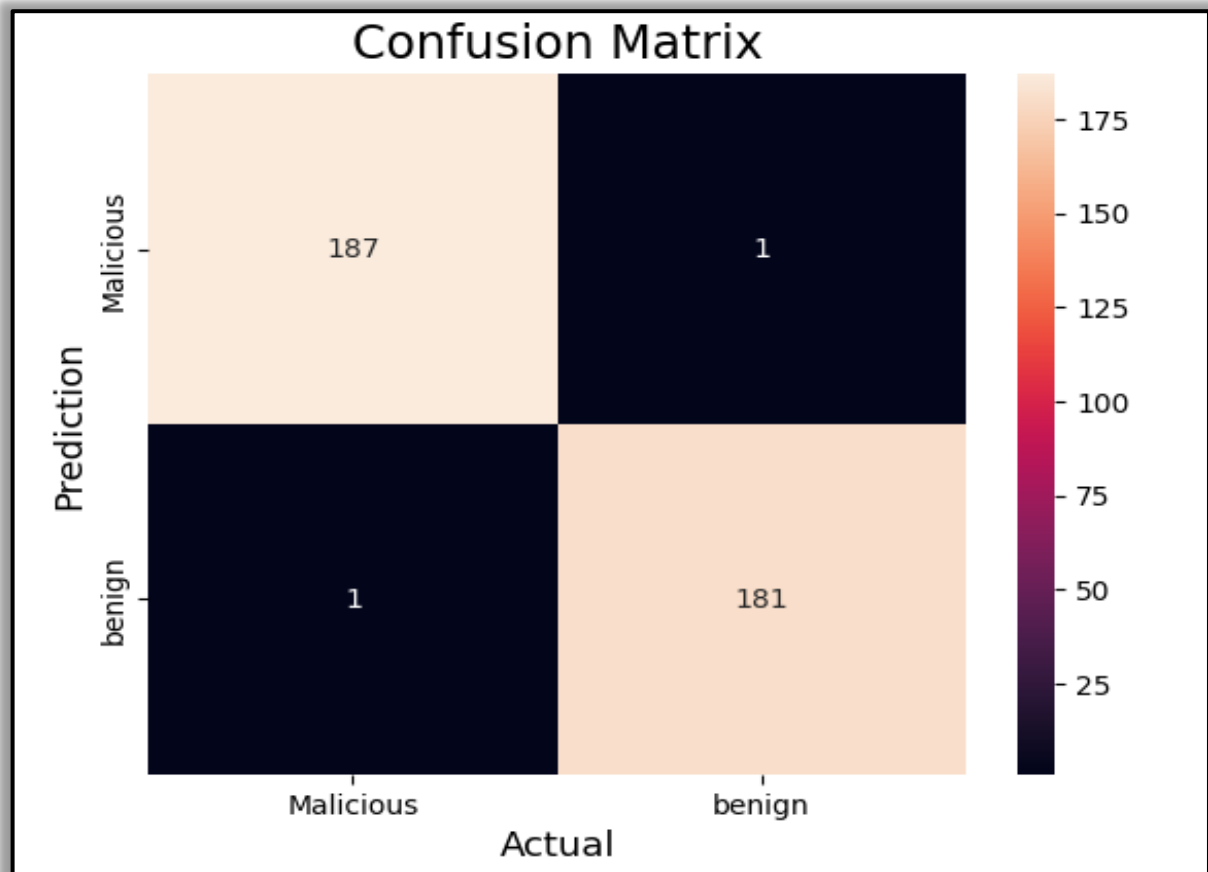


Figure 8 Confusion Matrix of Malicious and Benign Samples.

figure 8, which showcases the confusion matrix. This matrix allows us to identify the True Positives (TP) and False Positives (FP) in terms of distinguishing between malicious and benign applications.

3.4.5 Signature Database

Once a sample has been identified as ransomware by the ML model described earlier, the file's hash in the form of a signature, and other essential details, will be maintained in a DB server (MySQL). The signature hash was generated via SHA- 256, hashing the malware file containing 64 characters. The file's content might be quickly and easily matched with this technique; ransomware can be found immediately and without Cuckoo Sandbox analysis. On average, Cuckoo Sandbox takes 2-4 minutes to inspect the file.

This method was substantially more efficient, quicker, and more accurate. Nevertheless, a signature repository had to be utilized for it to function initially; an ML technique could be used to create it. This process is very meticulous and tailored to the particular malicious software sample; even the slightest changes will make it useless. The design of the Signature Repository's MySQL database is presented in Figure 6. When the ML model identifies a file as ransomware, it isolates it and places crucial information into the two database tables, Sample and Repeats. If ML identified the ransomware, it implied that this was the user's primary attack from the ransomware.

The Sample table consists of eight fields, the first of which is Hash-Value, which contains the file's hash code. The FileName field stores the initial file name. The Attack field holds the number of times the user has been targeted by the ransomware. The Date and Time attribute records the date and time of the first ransomware attack, while the Size field stores the file's size in megabytes and the created field keeps track of the file's creator. The Location column records the file's quarantined location after its extension is removed to deactivate it. The signature matching method will detect the same ransomware if it strikes again. In such a situation, the information of the second attack will be inserted into the table as Repeat, and the

file will be deleted with one value added to the Attack field in the Sample table as shown in Figure 9.

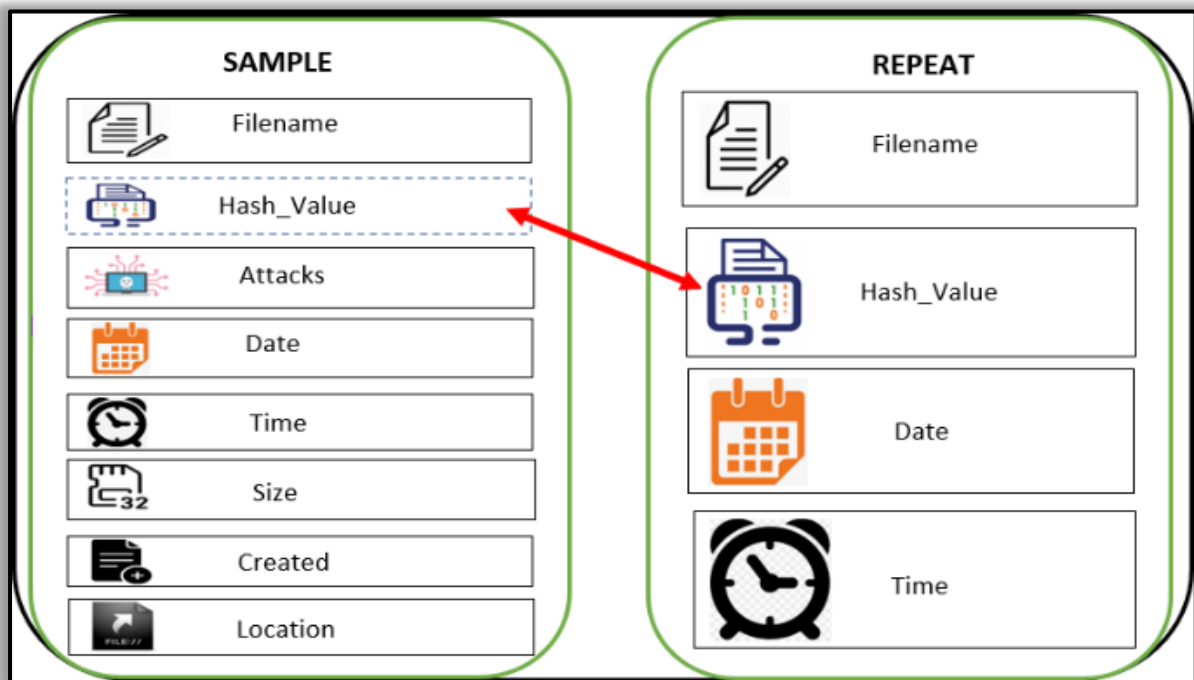


Figure 9 Data Structure of Signature Database Repository.

3.5 Pre-Encryption Model Verification

Similarly, the equation that determines how to calculate API pattern identification using the trained ML algorithm of the pre-encryption model is shown in Figure 7. When the test file has been tagged as “Un-match,” the pre-encryption APIs will be retrieved for pattern recognition [49,50]. This procedure will reveal whether the file was “Goodware” or “Ransomware.”

The mathematical expression for the process signatures updates to the model’s Signature Repository is also shown in Figure 10. When a file is flagged as “Ransomware,” the signature repository is checked to see if it already contains the signature; if not, the signature repository is updated with the new signature [51,52].

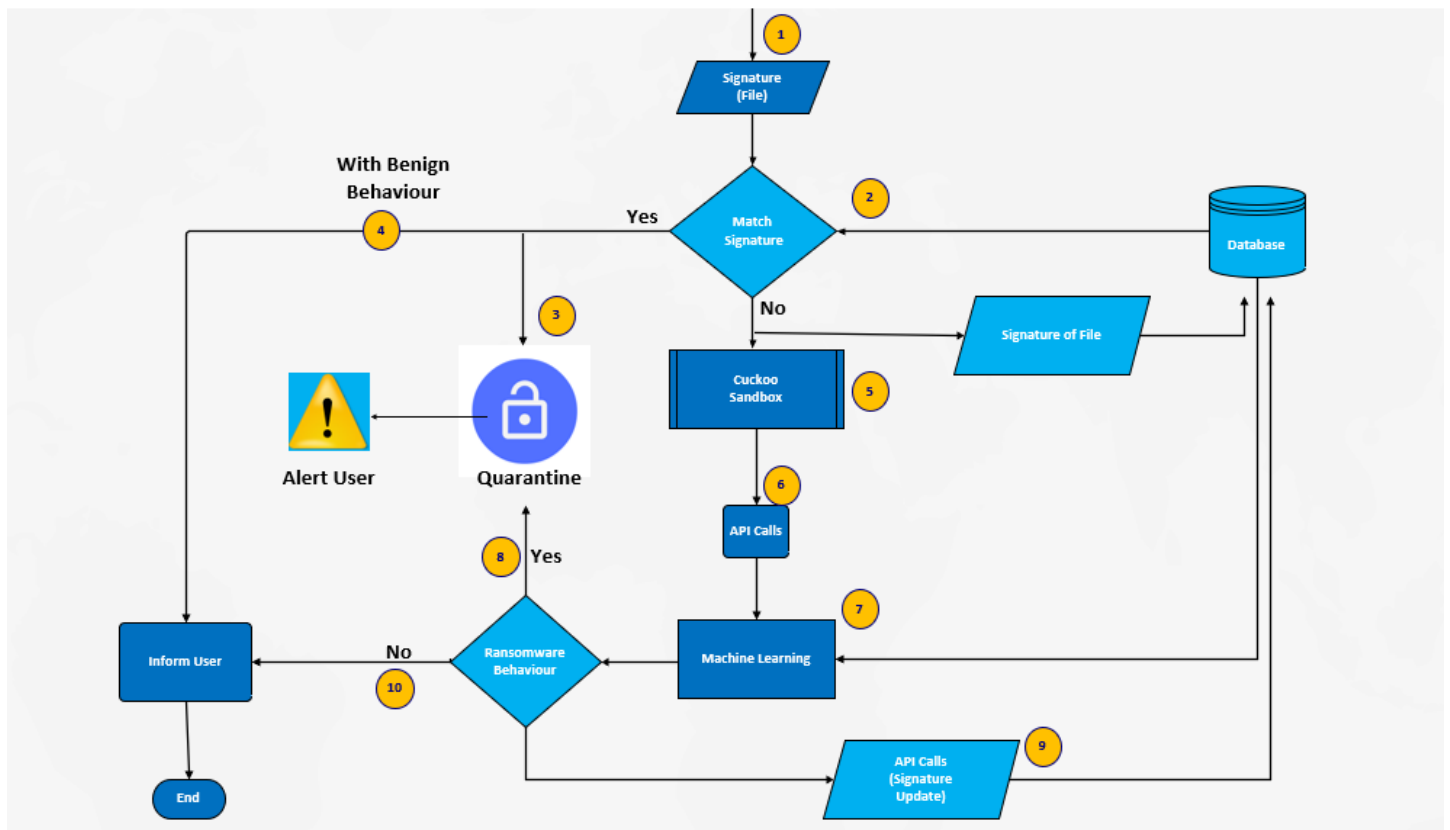


Figure 10 Flow Diagram of Ransomware Detection.

3.6 Summary

In this chapter, we highlight the methodologies used in this research to obtain the results and offers guidance to achieve similar outcomes. The approach involves collecting datasets from reliable sources, identifying the feature set by filtering the data, and utilizing Machine Learning techniques for classification. The following chapter will delve into the experimental setup designed for this specific analysis.

4. Experimental Setup

Chapter 4 presents a detailed description of the experimental setup that has been carefully designed to support the research objectives. The chapter justifies the methods and techniques used and provides comprehensive information about the system configurations that were implemented during the research.

4.1 Overview

The experimental setup comprises several crucial components that facilitate the execution of the research process with a focus on Windows Pre-encryption scenarios. The first component is the Windows Pre-encryption Dataset. This dataset serves as the primary input for conducting the analysis and evaluation of pre-encryption scenarios. It encompasses a comprehensive collection of both malicious and benign applications/files. Careful curation ensures that the dataset represents a diverse range of samples encountered in real-world scenarios, capturing potential threats and legitimate files accurately. Another vital component is the dedicated PC allocated for running the Python-based code. This PC serves as the computational platform, equipped with the necessary hardware and software resources to handle the experimental workload efficiently. The specifications of the PC, including its processing power, memory capacity, and storage capabilities, are carefully selected to meet the requirements of the experimentation process. This ensures smooth execution and accurate results.

The Python-based code forms the core of the experimental setup. It is meticulously designed to generate a comprehensive feature set and conduct the evaluation process effectively. Leveraging the Windows Pre-encryption dataset, the code extracts relevant features, performs advanced analysis techniques, and executes classification or evaluation algorithms. The

codebase adheres to best practices in software development, ensuring readability, modularity, and reproducibility. Integrating these components into the experimental setup enables researchers to systematically analyze and evaluate the behavior of malicious and benign applications/files in Windows Pre-encryption scenarios. The dataset provides a reliable and representative sample pool, enabling robust experimentation. The dedicated PC, optimized for computational efficiency, ensures that the Python-based code operates seamlessly and delivers accurate results. The well-crafted code encapsulates the required algorithms and techniques, facilitating feature extraction, classification, and evaluation processes.

Overall, this professional and systematic experimental setup allows researchers to gain valuable insights into the characteristics and behavior of applications/files in the context of Windows Pre-encryption. The findings derived from this experimentation contribute to advancing knowledge in the field of cybersecurity and aid in the development of effective countermeasures against potential threats.

4.2 Setting up the Environment

For carrying out experimentation, a windows-based machine has been used. The specifications of the system have been shown in below Table 5.

Table 5 System Specifications.

Property	Description
<i>Manufacturer</i>	HAIER
<i>Model</i>	Y11C
<i>Architecture</i>	x64 based
<i>Operating System</i>	Windows 10 Pro
<i>Processor</i>	Intel Core m3-7Y30 CPU@ 1.00GHz
<i>RAM</i>	8 GB

<i>Storage</i>	1 TB
----------------	------

Windows-based machine manufactured by Haier, model Y11C. It operates on a 64-bit architecture, allowing it to efficiently utilize the capabilities of modern computing systems. The chosen operating system is Windows 10 Pro, which provides a robust and secure environment for conducting various tasks and experiments. At the core of the system lies an Intel Core m3-7Y30 CPU. This processor belongs to the Intel Core series, known for its performance and power efficiency. The specific model, m3-7Y30, operates at a base clock speed of 1.00GHz. While the base clock speed may seem relatively low, it is important to note that modern processors often utilize dynamic clocking mechanisms to adjust their speed based on workload demands. This enables efficient power management and optimal performance for various computational tasks.

To support the processing capabilities, the system is equipped with 8 GB of RAM. Random Access Memory (RAM) plays a crucial role in determining the system's ability to handle and process data effectively. With 8 GB of RAM, the system can accommodate a significant amount of data and ensure smooth multitasking during experimentation. For data storage purposes, the system offers a spacious 1 TB storage capacity. This storage capacity allows for the efficient retention and retrieval of experimental data, software tools, and other necessary files. The sizable storage ensures that researchers have ample space to store and analyze experimental results without concerns of running out of storage capacity. By considering these specifications, the system demonstrates a suitable configuration for carrying out experiments and conducting various computational tasks. The combination of a 64-bit architecture, Windows 10 Pro operating system, Intel Core m3-7Y30 CPU, 8 GB of RAM, and 1 TB of storage capacity

provides a balanced and capable platform for experimental research, ensuring smooth performance and efficient data handling throughout the experimentation process.

4.3 Model Implementation

The implementation of pre-encryption detection for Windows platforms utilizing the Python programming language posed a challenge due to the primary design of Cuckoo Sandbox for Linux platforms, which operate on different operating systems. To address this requirement, an extensive research effort was undertaken, leading to the discovery that Linux can be installed as a subsystem on the Windows platform. This realization proved to be a viable solution, enabling the seamless integration of Linux functionalities within the Windows environment. As a result, the pre-encryption detection methodology can be effectively applied, leveraging the Linux subsystem to enhance the detection capabilities on Windows-based systems.

Cuckoo Sandbox, a powerful open-source tool for dynamic malware analysis, primarily designed for Linux, was identified as a key component for pre-encryption detection. To overcome the platform compatibility limitations, the Windows Subsystem for Linux (WSL) feature was identified as a valuable resource. By installing a Linux-based operating system, specifically Ubuntu 20, as a subsystem, Cuckoo Sandbox was successfully deployed within this Linux environment on the Windows platform. This integration allowed the utilization of Cuckoo Sandbox's advanced analysis capabilities for pre-encryption detection. In conjunction with Cuckoo Sandbox, the MySQL relational database system

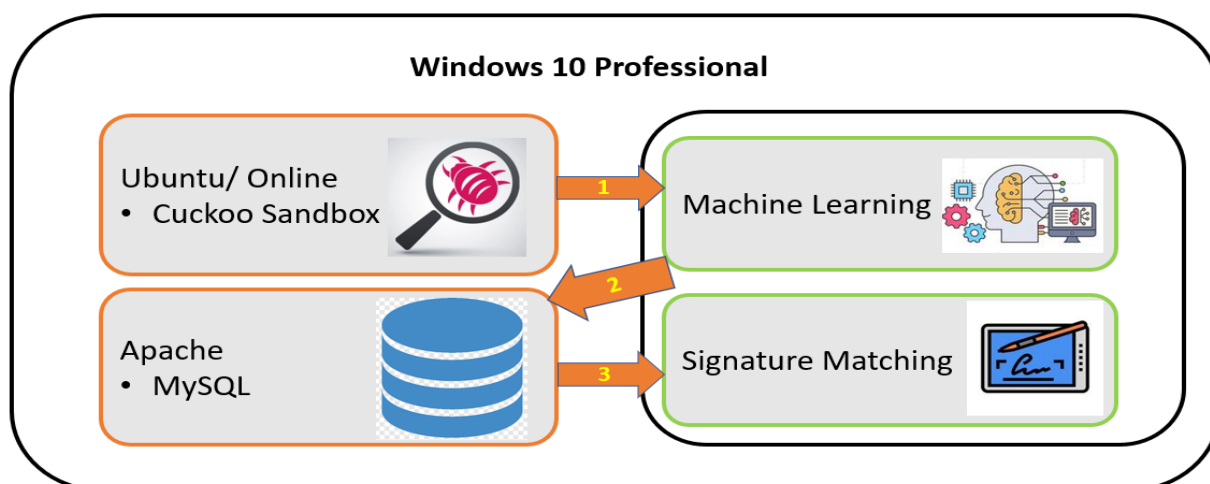


Figure 11 Proposed Model Execution in Operating System.

CHAPTER 4: EXPERIMENTAL SETUP

played a crucial role in supporting the pre-encryption detection process. Once the ML algorithm within Cuckoo Sandbox identifies ransomware, the signature of the malicious file is recorded in the MySQL repository. This repository, managed by the Apache server, serves as a centralized storage for all recognized ransomware signatures. Through the comparison of file data hashes, the system can accurately identify ransomware and initiate appropriate actions to mitigate the threat.

The Windows 10 Professional system, known for its robust security features, implements the pre-encryption principle within its framework as seen in Figure 11. This provides a solid foundation for the integration of Cuckoo Sandbox and the utilization of the Linux subsystem for pre-encryption detection. With the ML algorithm's pre-encryption API developed within Cuckoo Sandbox, the system gains the ability to detect and analyze potential threats before they undergo encryption, significantly improving the effectiveness of the detection process. The pre-encryption detection system's operation involves a well-defined workflow. Once ransomware is identified by the ML algorithm within Cuckoo Sandbox, its signature is stored in the MySQL repository, providing a comprehensive reference for future detection instances. The Apache server, which runs MySQL, ensures efficient storage and retrieval of the recognized ransomware signatures. The system's front-end module, typically implemented as a graphical user interface (GUI), facilitates seamless interaction with the user and promptly notifies them about the presence of ransomware, enabling swift response and mitigation measures. The successful integration of Cuckoo Sandbox with the Linux subsystem on the Windows platform exemplifies the adaptability and versatility of the Windows ecosystem. This innovative approach expands the capabilities of pre-encryption detection, empowering researchers and security practitioners to effectively combat the evolving landscape of cybersecurity threats. By leveraging the combined power of Cuckoo Sandbox, the MySQL relational database system, and the Linux subsystem, the pre-encryption detection system provides an advanced and proactive defense mechanism against ransomware attacks on Windows-based systems.

In conclusion, the integration of Cuckoo Sandbox and the Linux subsystem within the Windows environment represents a significant breakthrough in pre-encryption detection. This professional and meticulously engineered solution showcases the interdisciplinary nature of cybersecurity research,

where innovative approaches are devised to address complex challenges in protecting critical systems and data from malicious threats.

4.4 Installing Pre-requisite Software

Following Software needs to be installed before the experimentation process can be followed:

1. Archiving Tool (WinRAR): An archiving tool such as WinRAR is required for extracting application samples. You can download WinRAR from the following link:
<https://www.win-rar.com/download.html?&L=0>
2. Visual Studio Code: Visual Studio Code is a code editor that can be used to test and run code locally. You can download Visual Studio Code from the following link:
<https://code.visualstudio.com/download#>
3. Python Interpreter: A Python interpreter, preferably version 3.8 or higher, is needed for extracting applications, generating feature sets, and performing classifications. You can download the Python interpreter from the official Python website:
<https://www.python.org/downloads/release/python-380/>
4. Machine Learning Libraries: Install the necessary machine learning libraries to run machine learning code. You can refer to the documentation or installation instructions specific to the libraries you require. One resource for installing Python packages is ApmMonitor.
<https://apmonitor.com/pds/index.php/Main/InstallPythonPackages>
5. MySQL Database: Install the MySQL Database, which will be used for storing application metadata such as name, signatures, size, package name, etc. This will speed up the process and help keep track of already de-compiled applications. You can download MySQL from the official website: <https://dev.mysql.com/downloads/mysql/>
6. Database Viewer for MySQL: Install a database viewer for MySQL, such as MySQL Workbench. This tool will allow you to import and view datasets available in the form of a database. You can download MySQL Workbench from the following link:

<https://dev.mysql.com/downloads/workbench/>

7. XAMPP: Install XAMPP, which includes the Apache server, to locally set up a MySQL server for storing signatures. XAMPP provides an easy way to set up a development environment.

You can download XAMPP from the official website:

<https://www.apachefriends.org/download.html>

4.5 Summary

In this chapter, we explain the experimental setup proposed for the analysis. The chapter covers the process of collecting the necessary dataset, setting up the required environment, and the related codebase. Additionally, it provides information on the installation of the necessary software for the analysis process and their source

5. Experimental Results

Chapter 5 presents the results obtained from the research in the form of classification results and provides an analysis of these results. The chapter compares the results with the original benchmark approach [27] and discusses the achievements of the proposed approach.

5.1 Overview

Windows applications use API and system calls to provide functionality to the users, which are exploited by malware developers for conducting cybercrimes. In this study, extensive analysis has been carried out on a windows dataset representing benign and malicious applications. We will discuss different evaluation metrics employed while performing the analysis to measure the effectiveness of the approach.

5.2 Evaluation Measures

For evaluation, the following metrics are employed: Sensitivity, Precision, Accuracy, Area Under Curve (AUC), and the Receiver Operating Characteristic (ROC). Correspondingly, the following formulas represent their definitions:

$$1) \textit{ Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

$$2) \textit{ Sensitivity} = \frac{TP}{TP + FN}$$

$$3) \textit{ Precision} = \frac{TP}{TP + FP}$$

$$4) \textit{ F1 - Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

True positive (TP), false positive (FP), true negative (TN), and false negative (FN) are crucial metrics utilized in assessing the effectiveness of a classification model. TP represents the number of positive samples correctly classified as positive by the model. FP refers to the number of negative samples erroneously classified as positive. TN denotes the number of negative samples correctly classified as negative. FN signifies the number of positive samples incorrectly classified as negative. These metrics play a vital role in evaluating the accuracy and performance of a classification model.

5.3 Performance Evaluation of Proposed Model

To assess the effectiveness of our research, we utilized various classification models to demonstrate the generalizability of our approach. To prevent overfitting and ensure stability, we employed cross-validation techniques during the experiments. Specifically, we used 10-fold cross-validation where the samples were randomly divided into ten separate sets, with one set used for testing and the remaining nine sets used for training the model. The results of the 10-fold classification are presented in Tables 6-9, with corresponding ROC curves shown in Figures 12-15.

Table 6 Performance of Proposed Scheme using Random Forest Classifier with 10-Fold Cross-Validation.

Features	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Cross Validation (%)
26	RF	99.45	99.46	99.46	99.46	99.76

The results presented in Table 6 highlight the exceptional performance of the proposed methodology when utilizing the Random Forest classifier. The achieved accuracy of 99.45%, precision of 99.46%, and f1-score of 99.46% indicate the high effectiveness of the classifier in

CHAPTER 5: EXPERIMENTAL RESULTS

accurately identifying pre-encryption threats. The corresponding Receiver Operating Characteristic (ROC) curve, shown in Figure 12, further demonstrates the classifier's ability to strike a balance between sensitivity and specificity, with the curve being closer to the top-left corner, indicating superior performance.

Table 7 Performance of Proposed Scheme using SVM Classifier with 10-Fold Cross-Validation.

Features	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Validation (%)
26	SVM	99.27	99.28	99.29	99.28	99.53

Additionally, Table 7 showcases the performance of the proposed feature set when combined with the Support Vector Machine (SVM) classifier. The results indicate an accuracy of

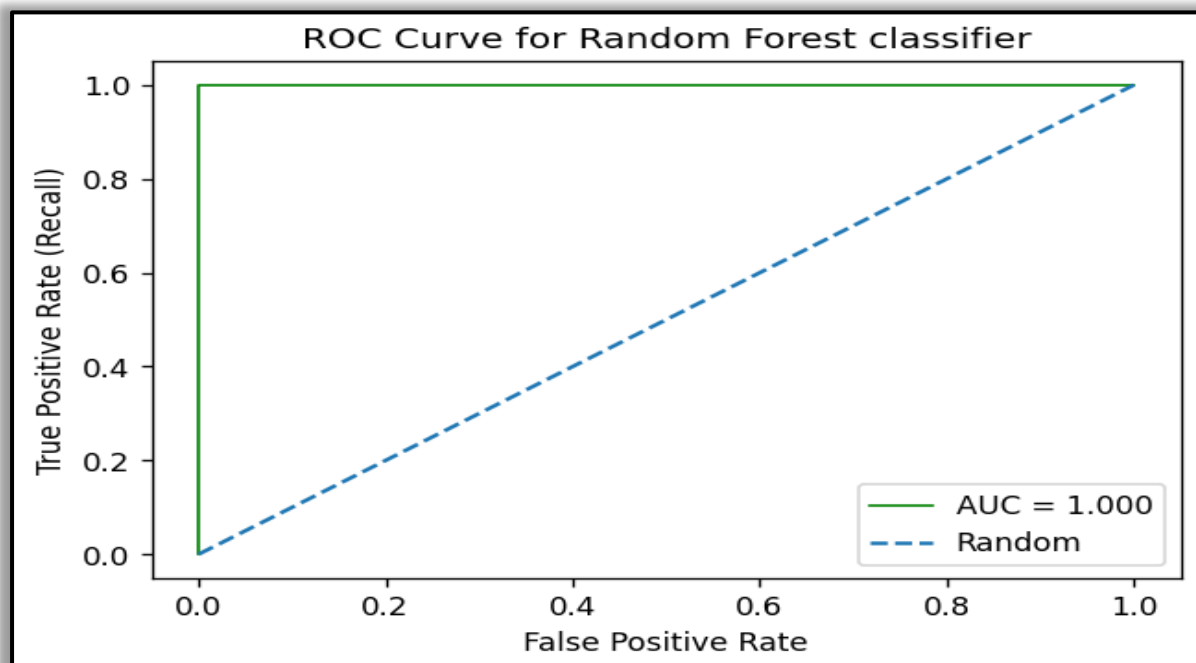


Figure 12 The Receiver Operating Characteristic (ROC) curve obtained from the proposed method using the Random Forest classifier.

99.27%, precision of 99.28%, and an f1-score of 99.28%. These metrics further confirm the efficacy of the SVM classifier in accurately classifying pre-encryption threats.

CHAPTER 5: EXPERIMENTAL RESULTS

The proposed methodology for pre-encryption threat detection achieves remarkable results with an accuracy of 99.45%, precision of 99.46%, and an f1-score of 99.46% when utilizing the Decision Tree classifier, as demonstrated in Table 8.

Table 8 Performance of Proposed Scheme using Decision Tree Classifier with 10-Fold Cross-Validation.

Features	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Validation (%)
26	DT	99.45	99.46	99.46	99.46	99.53

Additionally, Figure 13 displays the corresponding Receiver Operating Characteristic (ROC) curve, which showcases the methodology's ability to balance sensitivity and specificity effectively. These exceptional performance metrics and the visualization support the efficacy

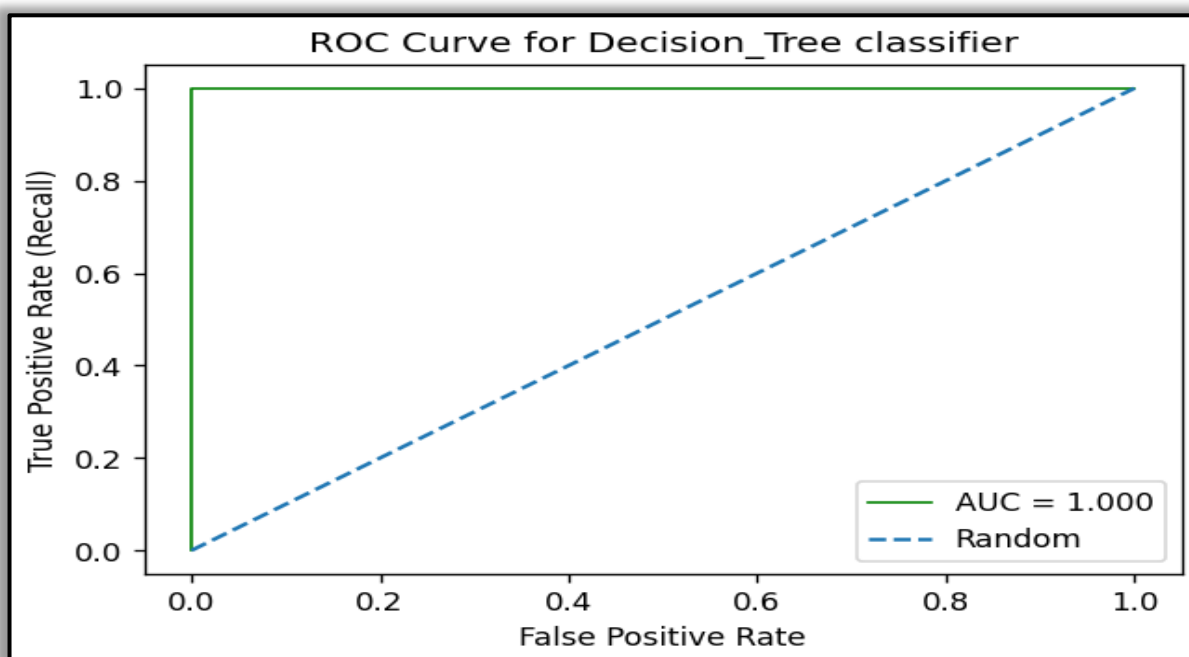


Figure 13 ROC Curve Obtained by Proposed Method for Decision Tree Classifier.

and reliability of the proposed methodology, highlighting its potential as a robust tool for detecting pre-encryption threats and strengthening cybersecurity measures.

CHAPTER 5: EXPERIMENTAL RESULTS

Table 9 provides an overview of the accuracy achieved by the KNN (K-Nearest Neighbour) and Naive Bayes classifiers, showcasing their performance in pre-encryption threat detection. The KNN classifier achieves an accuracy rate of 99.27%, accompanied by a precision and f1-score of 99.28%. The Naive Bayes classifier demonstrates a higher accuracy rate of 99.47%, with a precision of 97.57% and an f1-score of 99.47%. These results highlight the effectiveness of both classifiers in accurately classifying pre-encryption threats.

Table 9 Performance of Proposed Scheme using KNN & Naive Bayes e Classifier with 10-Fold Cross-Validation.

No. of Features	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Validation (%)
26	KNN	99.27	99.28	99.29	99.28	99.38
	NB	97.47	97.57	97.50	97.47	97.91

To further assess their performance, ROC graphs are presented for the KNN and Naive Bayes classifiers in Figures 14 and 15, respectively. The ROC curves visually depict the trade-off between true positive rate (sensitivity) and false positive rate (1 - specificity). The proximity of the curves to the top-left corner indicates a higher performance in correctly identifying positive instances while minimizing false positives. The ROC graphs further validate the classification capabilities of both the KNN and Naive Bayes classifiers in pre-encryption threat detection.

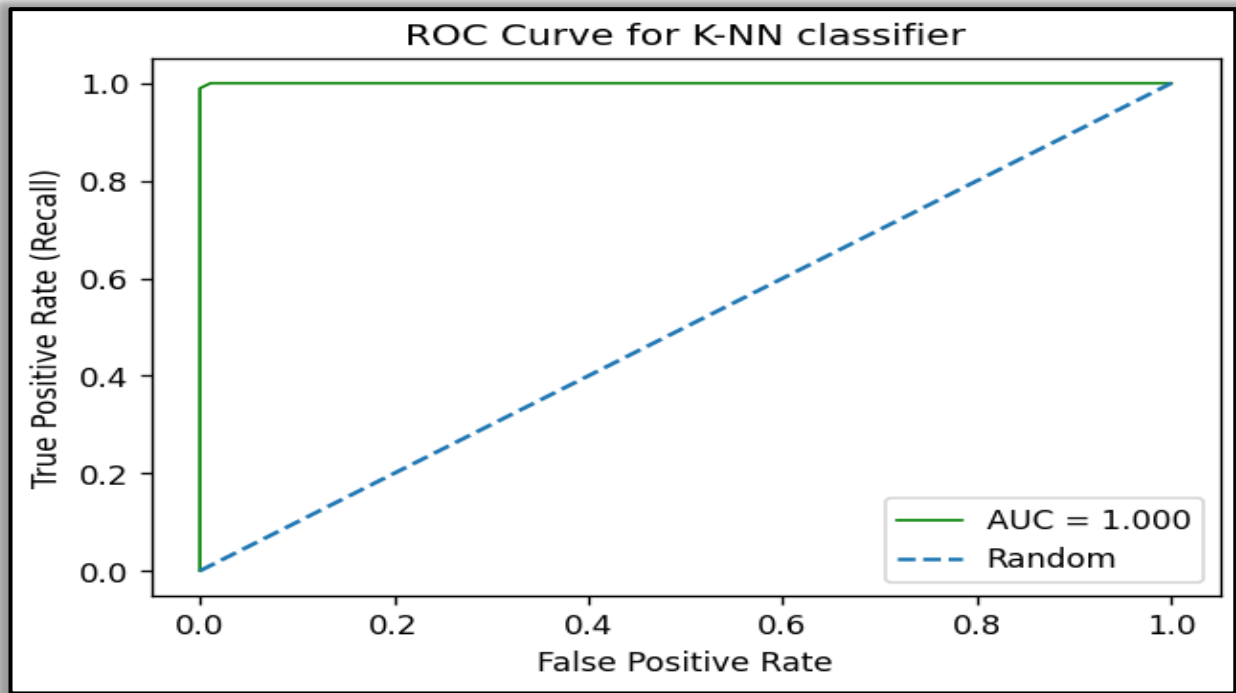


Figure 14 ROC Obtained by the proposed method on the KNN classifier.

Overall, the results from Table 9 and the ROC graphs in Figures 14 and 15 demonstrate the efficacy of the KNN and Naive Bayes classifiers in accurately detecting pre-encryption threats. These findings contribute to the comprehensive evaluation of the proposed methodology and

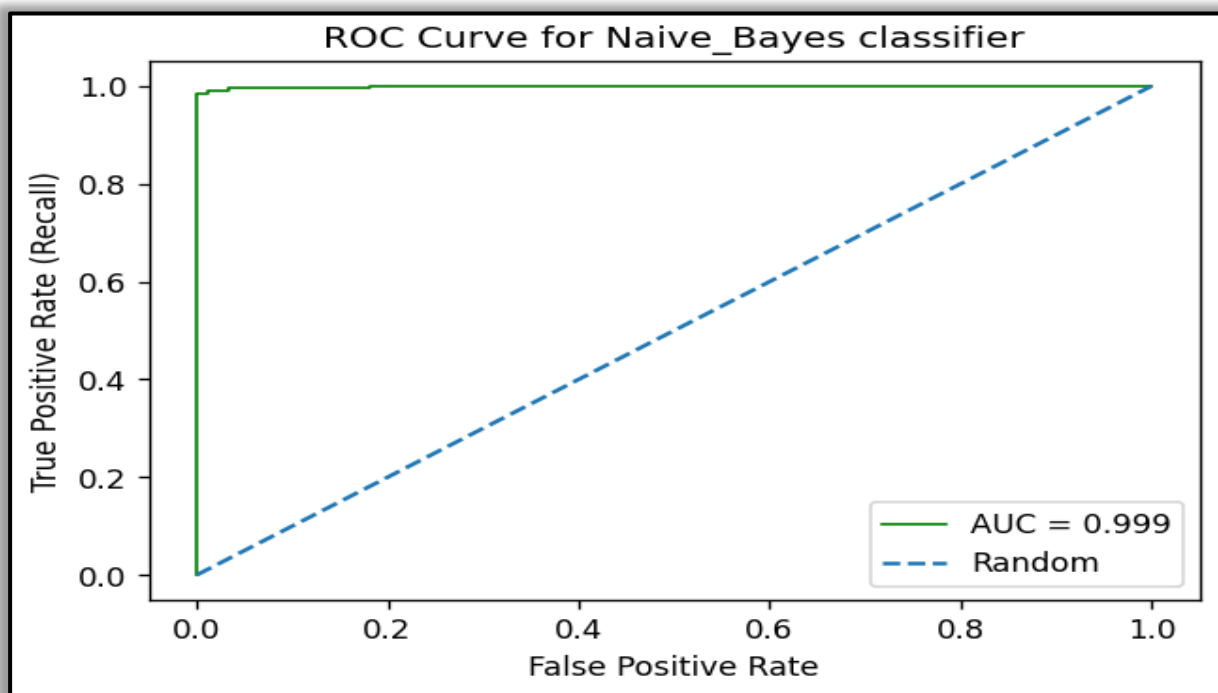


Figure 15 ROC Analysis of the Proposed Method on Naive Bayes Classifier.

provide valuable insights for selecting the most suitable classifier for pre-encryption threat detection applications.

5.4 Summary

In this chapter we present an in-depth analysis of the results obtained from the research. The chapter discusses the outcomes of the proposed approach and evaluates its effectiveness. In the subsequent chapter, the validation and verification of the results achieved will be provided

6. Discussion and Analysis

In this chapter, we will focus on the validation and verification of the analysis and results obtained during the experimentation, specifically in relation to the proposed feature set. The chapter compares the results obtained from the classifiers discussed in the previous chapter (Random Forest, SVM, Decision Tree, KNN, and Naive Bayes) with a benchmark approach. Additionally, the results obtained are evaluated against a benchmark for verification purposes.

6.1 Overview

In this section, we will evaluate and compare the performance of proposed and existing techniques. For evaluation, we have employed metrics such as Accuracy, precision, recall, TRP, FPR, etc.

From Table 10, it is shown that [27] employed the 232 APIs features while achieving 100% accuracy on the random forest model. However, from experiments, we realized that only 46 APIs calls are sufficient instead of 232 APIs to detect the ransomware. For this, we employed the random forest feature importance technique and successfully identified the most relevant 46 features while achieving 99% detection accuracy.

In Figure 16, the features utilized by Kok S.H et al. for the detection of ransoms are categorized. It is evident from Figure 16 that certain features demonstrate a higher degree of influence. Consequently, by employing a feature ranking approach, we have selected only those features that exhibit significant impact in the detection of crypto ransoms, as depicted in the figure. Furthermore, we also categorized the 46 APIs in terms of reading and writing functions/APIs to identify the significant contribution of APIs. Tables 11 and 12 show that the proposed model consists of 20 read functions and 26 write functions. The identified read-and-

CHAPTER 6: DISCUSSION AND ANALYSIS

write API can be seen in Table 11 and Table 12. After analyzing read-write functions, we employ these features separately in our machine-learning models to evaluate the accuracies.

Table 10 Kok. et al. Achieved Accuracy [27].

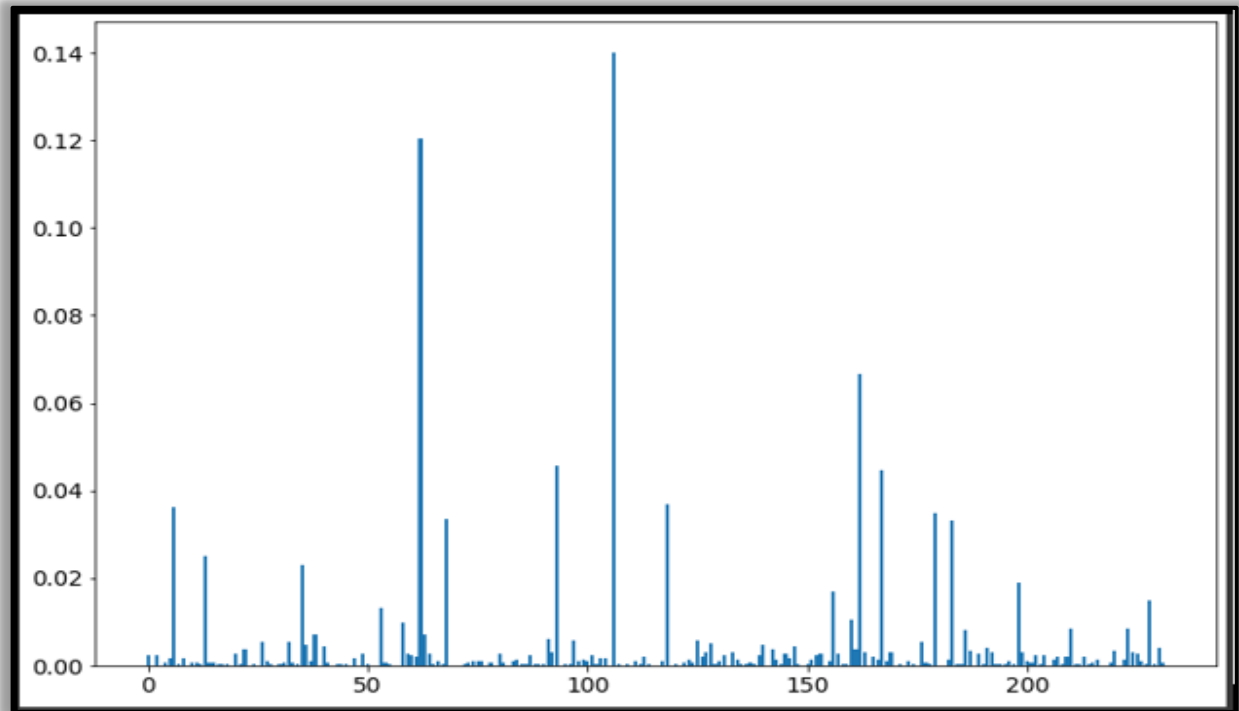


Figure 16 Features Importance.

Features (APIs)	Model	Accuracy	Precision	Recall	F1-Score	Cross Validation
232	Random Forest	100%	100%	100%	100%	99.93%

Table 11 Read Functions [53] of Ranking Features 46.

S. No	Function	Purpose	Description
1.	NtReadVirtualMemory	Read	Read the memory and analyze its structure.

CHAPTER 6: DISCUSSION AND ANALYSIS

2.	LookupPrivilegeValueW	Read	Using a specified system's locally unique identity (LUID), the LookupPrivilegeValue function locates the privilege name's local representation.
3.	GetDiskFreeSpaceExW	Read	The amount of space available on a disk volume is fetched is composed of the total amount of space, the total amount of free space, and the total amount of free space that is available to the user that is associated with the calling thread.
4.	OpenServiceA	Read	Opens an existing service.
5.	NtOpenKey	Read	
6.	GetSystemTimeAsFileTime	Read	Obtains the time and date for the current system. Coordinated Universal Time (UTC) has been used to format the data.
7.	NtDelayExecution	Read	A pointer to the delay interval value is provided as the NtDelayExecution function's second argument.
8.	GetVolumePathNameW	Read	Discovers the volume mount point where the mounted path is located.
9.	NtQueryValueKey	Read	For a registry key, the ZwQueryValueKey procedure returns a value entry.

CHAPTER 6: DISCUSSION AND ANALYSIS

10.	FindResourceExA	Read	Identifies the location of a resource in the provided module with the specified type and name.
11.	GetFileAttributesW	Read	For a specific file or directory, retrieves file system attributes.
12.	FindResourceExW	Read	Identifies the location of the resource in the supplied module with the specified type, name, and language.
13.	NtGetContextThread	Read	Obtains the context of the chosen thread.
14.	GetCursorPos	Read	The location of the mouse cursor in screen coordinates can be retrieved.
15.	NtOpenDirectoryObject	Read	Reveals a current directory object.
16.	Getaddrinfo	Read	The getaddrinfo function converts an ANSI hostname to an address without regard to protocol.
17.	NtReadFile	Read	Reads data from an open file.
18.	SearchPathW	Read	Looks for a particular file in a given path.
19.	GetSystemMetrics	Read	The requested system metric or system configuration setting is retrieved.

CHAPTER 6: DISCUSSION AND ANALYSIS

20.	GetSystemDirectory	Read	Reveals the system directory's path. System files like drivers and dynamic-link libraries can be found in the system directory.
-----	--------------------	------	---

Table 12 Write Functions [53] Of Ranking Features 46.

S.NO	Function	Purpose	Description
1.	VirtualProtectEx	Write	Changes the security on a subset of committed pages in the address space of a certain process.
2.	HttpSendRequestW	Write	HttpSendRequest allows the client to specify additional headers to send along with the request while sending the given request to the HTTP server.
3.	NtFreeVirtualMemory	Write	The NtFreeVirtualMemory operation releases, decommits, or both a group of pages in a specific process' virtual address space.
4.	CreateThread	Write	A new thread is created for a process by the CreateThread function. The starting address of the code that the new thread will execute must be specified by the generating thread.

CHAPTER 6: DISCUSSION AND ANALYSIS

5.	LdrLoadDll	Write	Using the low-level function LdrLoadDll, a DLL can be loaded into a process.
6.	SetEndOfFile	Write	To shrink or enlarge a file, the SetEndOfFile function can be used. If the file is expanded, the contents of the region between both the old end as well as the new ends are not defined.
7.	UuidCreate	Write	The UUID created by the UuidCreate function cannot be linked to the computer's ethernet address where it was created. Additionally, it cannot be linked to other UUIDs produced on the same computer.
8.	SetFilePointer	Write	This method stores the file pointer in two LONG values. To deal with file pointers that are larger than a single LONG value, use the SetFilePointerEx function.
9.	NtProtectVirtualMemory	Write	The EAF-generated guard pages are removed by the NtProtectVirtualMemory function. Windows shellcode is compatible with Windows version 7 and later with the proof-of-concept code.

CHAPTER 6: DISCUSSION AND ANALYSIS

10.	CryptAcquireContextW	Write	Use the CryptAcquireContext function to get a handle on a certain key container within a particular cryptographic service provider (CSP). The resulting handle is used in calls to CryptoAPI methods using the selected CSP.
11.	MessageBoxTimeoutW	Write	The MsgBoxWithTimeout function creates a new thread to allow the message box to be closed.
12.	NtWriteVirtualMemory	Write	This function copies the address range of the currently running process into the address space of the specified process.
13.	CreateDirectoryW	Write	It creates a brand-new directory. If the underlying file system permits security on files and directories, the function attaches a specified security descriptor to the new directory.
14.	NtCreateKey	Write	The ZwCreateKey routine opens an existing registry key or generates a new one.
15.	WriteProcessMemory	Write	Writes information to a specific process's memory location. If any portion of the

			location to be written to is not accessible, the operation will fail.
16.	LdrUnloadDll	Write	Using the low-level function LdrLoadDll, a DLL can be loaded into a process.
17.	ReadProcessMemory	Write	ReadProcessMemory copies information in the given address range from the target process's address space into the target buffer in the current process. Any process with a handle that has PROCESS VM READ access can call the function.
18.	NtSetInformationFile	Write	NtSetInformationFile modifies a file's information. Any FILE XXX INFORMATION structure member that is not supported by a specific device or file system is ignored.
19.	NtTerminateProcesses	Write	Puts an end to the specified process and all of its threads.
20.	NtWriteFile	Write	Data is written to an open file by the ZwWriteFile function.
21.	VirtualFreeEx	Write	The specified process releases, decommits, or releases and decommits a the portion of memory located within its virtual address space.

22.	NtSetContextThread	Write	
23.	CoInitializeSecurity	Write	Sets the process's default security values and registers security.
24.	NtResumeThread	Write	Increases a thread's suspensions per minute. As soon as the suspend count drops to zero, the thread restarts its operation.
25.	InternetCloseHandle	Write	The function discards any unfinished data and ends any pending activities on the handle.
26.	NtClose	Write	To close an object handle, use the NtClose procedure.

Table 13 shows the accuracies of proposed machine learning models with precision, recall, F1 score, and their cross-validation check. We got 100% accuracy using the Random Forest model, 99.81% using SVM, 99.63% using the Decision Tree, 98.09 using K-NN, 97.29% using Naive Bayes, and 99.18% of the Voting Classifier for 46 features. Further, digging into reading functions we get 96.20% using RF, SVM 96.38% DT 95.84% K-NN 96.38% NB 92.23%, and VC 96.48% accuracy.

We focused our efforts on the write function, as it contributes the most to ransomware compared to reading functions. After all, ransomware encrypts files at the first stage. We get

CHAPTER 6: DISCUSSION AND ANALYSIS

99.45% accuracy using Random Forest, SVM 99.27% DT 99.45% K- NN 99.27% NB 97.47%, and the Voting classifier, which takes an average of all classifiers have an accuracy of 99.72%.

This means that we can achieve the similar accuracy as presented in [27], using only 26 features of write function APIs. Our efficient dataset was verified using four other machine learning models, i.e., support vector machine, decision tree, k-nearest neighbour, naive bayes, and voting classifier with random forest. It presents a better result with fewer API functions.

Table 13 Accuracies of Machine Learning Models.

S. No	Features (APIs)	Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Cross Validation (%)
1.	Read & Write Combine (46)	RF	100	100	100	100	99.76
		SVM	99.81	99.82	99.82	99.82	99.61
		DT	99.63	99.64	99.64	99.64	99.45
		KNN	99.09	99.10	99.10	99.10	99.37
		NB	97.29	97.40	97.29	97.29	97.75
		VC	99.18	99.19	99.19	99.19	99.93
2.	Read (20)	RF	96.20	96.22	96.20	96.21	97.44
		SVM	96.38	96.40	96.39	96.39	96.51
		DT	95.84	95.86	95.84	95.85	97.21
		KNN	96.38	96.47	96.37	96.39	96.67
		NB	92.23	92.87	92.31	92.22	91.71
		VC	96.48	96.48	96.49	96.48	97.69
3.	Write (26)	RF	99.45	99.46	99.46	99.46	99.76
		SVM	99.27	99.28	99.29	99.28	99.53
		DT	99.45	99.46	99.46	99.46	99.53
		KNN	99.27	99.28	99.29	99.28	99.38
		NB	97.4	97.57	97.50	97.47	97.91
		VC	99.72	99.72	99.74	99.73	99.45

Similarly, for evaluating the matrices of our model, Accuracy, Precision, Recall, F-Measure, Cross-Validation, and ROC curves (Receiver Operating Characteristics Curve) were used to measure the probability of classification models at different levels. The curve has a true positive rate along the x-axis and a false positive rate along the y-axis. Figure 17-A, 17-B, 17-C, and 17-D represent ROC curves of different Machine learning models.

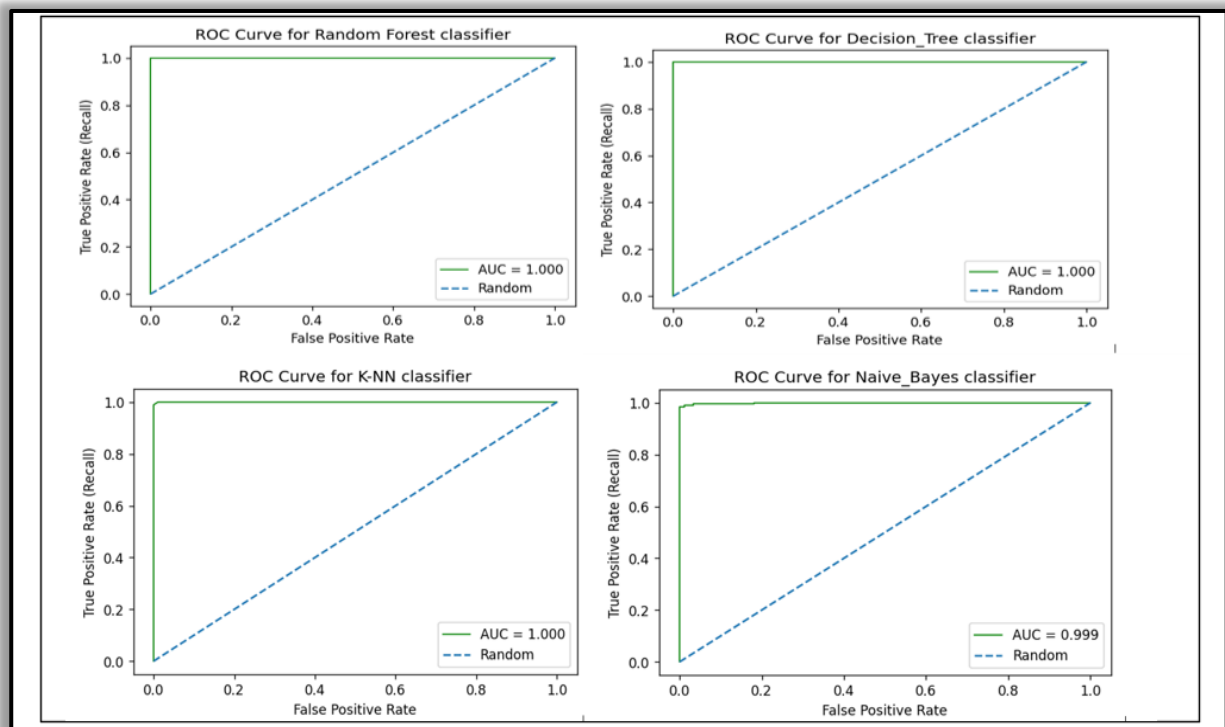


Figure 17 ROC Curves of ML Models

6.2 Comparison with Reference Approach

Table 14 provides a comprehensive performance comparison between the proposed approach and Kok S. H. et al.'s method [27]. The corresponding visual representation can be observed in Figure 18. The findings indicate that the Random Forest classifier exhibits the best detection performance. It's worth noting that the proposed approach achieves similar detection rates on SVM, Decision Tree, KNN, and Naive Bayes models, using a smaller feature set, compared to Kok S. H. et al.'s method [27]. Consequently, the Random Forest-based ensemble classifier,

CHAPTER 6: DISCUSSION AND ANALYSIS

using the proposed feature set, achieves an accuracy rating of $99 \pm 0.5\%$. Moreover, Table 14 reveals that the proposed approach reduces the feature set by 88.79% while still maintaining comparable detection rates. Overall, the proposed approach outperforms Kok S. H. et al.'s method [27] in terms of accuracy, and the Random Forest classifier proves to be the best-performing model.

Table 14 Performance Comparison of Proposed Scheme and Kok S. H. [27] on 10-Fold Validations.

Classifier	Kok et al [23] Approach				Proposed Approach			
	# of features	Precision (%)	Accuracy (%)	Cross-Validation (%)	# of features	Precision (%)	Accuracy (%)	Cross-Validation (%)
Random Forest	232	100	100	99.93	26	99.46	99.45	99.76
SVM		100	100	99.66		99.28	99.27	99.53
Decision Tree		99.74	99.72	99.32		99.46	99.45	99.53
KNN		98.39	98.37	99.39		99.28	99.27	99.38
Naïve Bayes		98.11	98.10	97.36		97.57	97.4	97.91
Average		99.24	99.23	99.13		99.01	98.96	99.22
Reduce % of features	0%				88.79%			

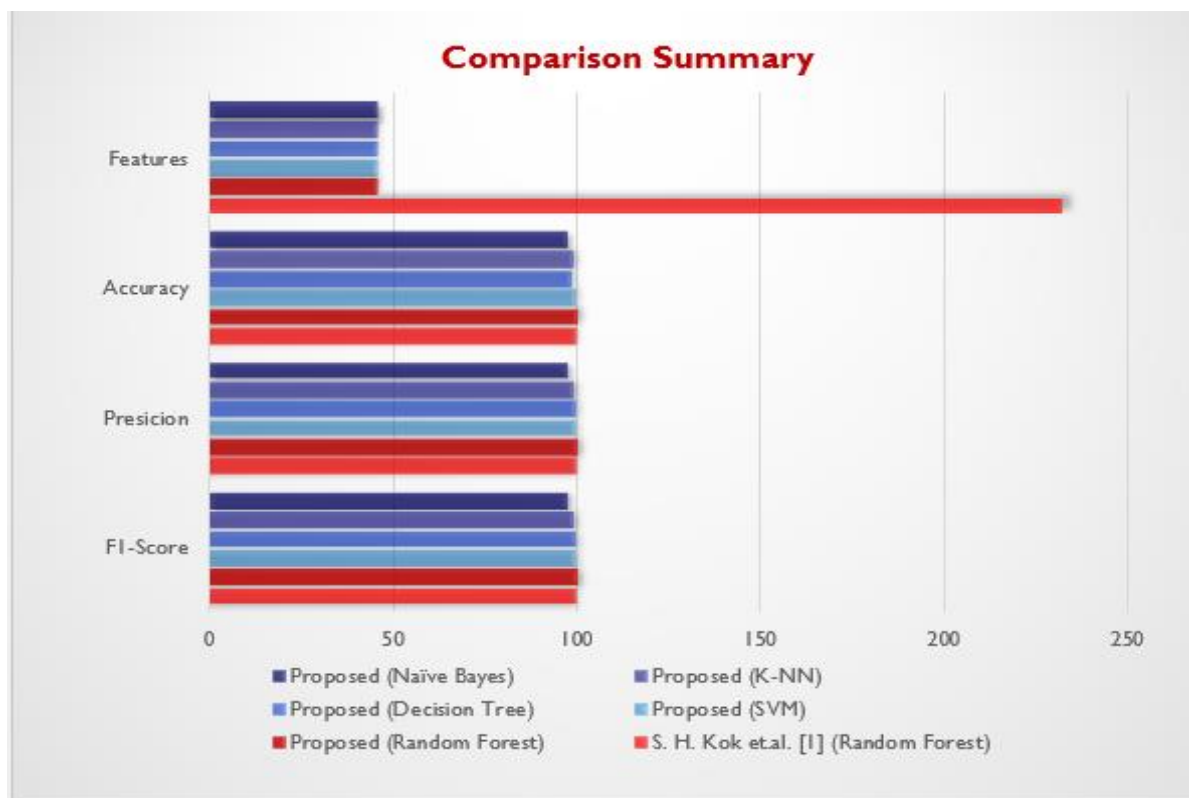


Figure 18 The Detection Accuracy of The Suggested and Chart of S. H. Kok [27] Methodologies In Diverse Machine Learning Classifiers.

During the experiments, Table 15 presents the initial results of the proposed Cuckoo-based dangerous APIs set on various classifiers, which include Random Forest, SVM, Decision Tree, KNN, and Naïve Bayes. The proposed feature set demonstrated similar detection accuracies while utilizing a significantly smaller number of features, 26 times smaller, compared to the feature set of the existing method, which was 232 times larger. This reduction in the number of features amounts to an 88.79% decrease. Figure 19 provides a comprehensive and insightful performance comparison between the proposed method and several existing approaches, shedding light on the number of features employed and the corresponding detection accuracies. Notably, the study by S.H. Kok [27] achieved an impressive accuracy of 100% by leveraging a rich feature set of 232, encompassing a complete set of read and write APIs. Similarly, M. Anwar [28] attained an accuracy of 87% utilizing 206 API features, demonstrating the efficacy of their approach.

CHAPTER 6: DISCUSSION AND ANALYSIS

Furthermore, Wira Z.A [29] demonstrated a noteworthy accuracy of 97.07% by effectively utilizing a reduced feature set of 80. P. Mohan Anand [30] achieved a commendable accuracy of 95.38% with the aid of 135 carefully selected features. Anshika Sharma [54], focusing specifically on ransomware detection in IoT environments, achieved an accuracy of 96.62% by leveraging a concise set of 32 features that primarily encompassed network attributes such as IP address, SSL State, and ports. It is important to emphasize that Anshika Sharma's model is designed exclusively for IoT networks and may not be directly applicable to other environments such as Windows, Linux, Android, and iOS.

In contrast to the aforementioned approaches, the proposed method in this study stands out by achieving an exceptional accuracy of approximately 98% while employing an impressively minimal feature set of just 26. This surpasses the performance of the existing methods, underscoring the effectiveness and efficiency of the proposed approach. By focusing primarily on write function APIs, the model showcases a more streamlined and targeted approach, resulting in enhanced accuracy and efficiency compared to other existing methods.

The findings from Figure 19 highlight the significant advancements in detecting ransomware and the increasing effectiveness of different feature sets and methodologies. The proposed method, with its superior accuracy and efficiency, holds promise for improving ransomware detection in various environments, making it a noteworthy contribution to the field.

Table 15 Results of Detecting Dangerous APIs Using Cuckoo-Based API Calls with Random Forest, SVM, Decision Tree, KNN, And Naïve Bayes Classifiers, Utilizing 10-Fold Cross-Validation.

Approach	Precision (%)	Recall (%)	F1 - Score (%)	Accuracy (%)	Cross Validation
Random Forest	99.46	99.46	99.46	99.45	99.76
SVM	99.28	99.29	99.28	99.27	99.53

Decision Tree	99.46	99.46	99.46	99.45	99.53
KNN	99.28	99.29	99.28	99.27	99.38
Naïve Bayes	97.57	97.50	97.47	97.47	97.91
Voting Classifier	99.72	99.74	99.73	99.72	99.45

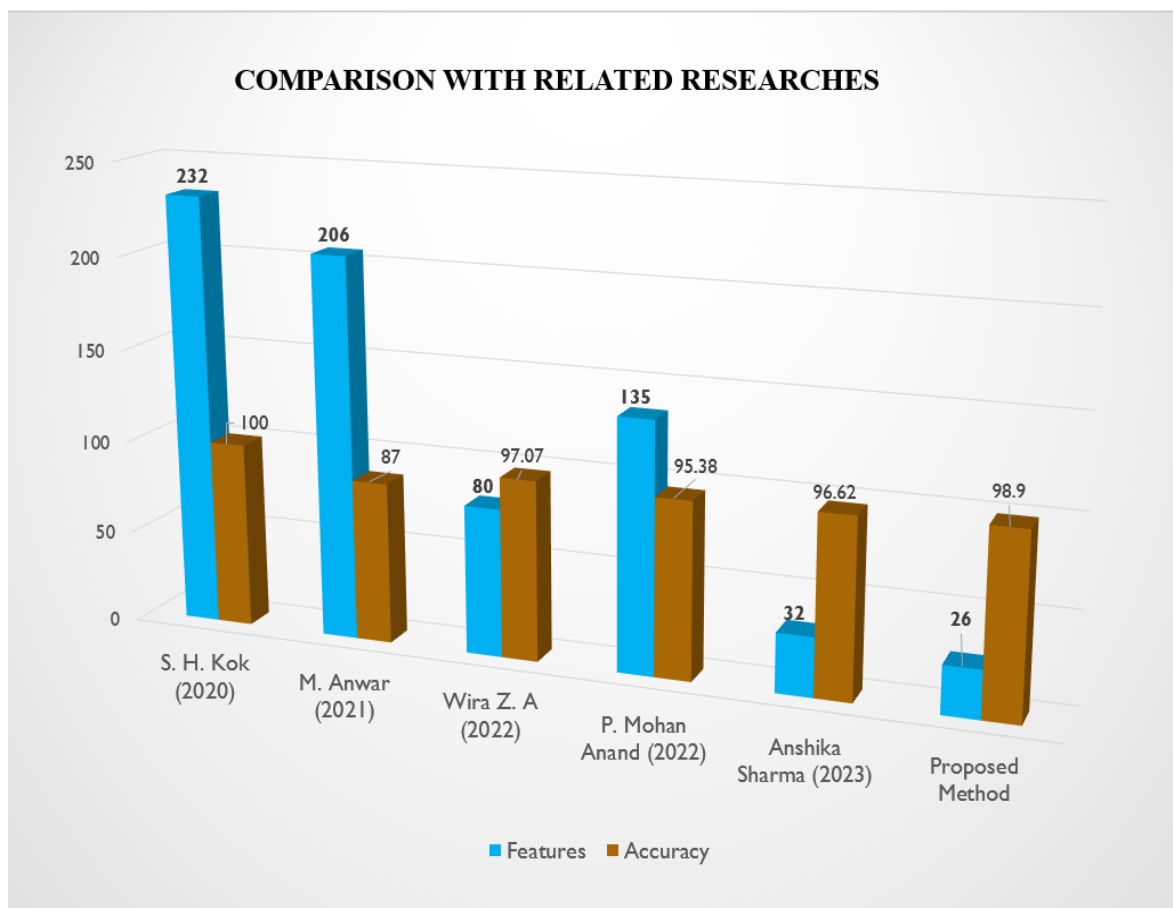


Figure 19 Comparison of Number of Features and Detection Accuracy between Proposed and Existing Approaches.

6.3 Applicability of the Approach

The below use cases are presented to better understand the application of the proposed approach. To support the ransomware detection analysis early detection patterns could help in minimizing and in the ideal case preventing the damages. Thus, based on the above analysis we can utilize the effectiveness of the research in many cases, such as those in the following:

- A. The proposed approach can distinguish between malicious and benign applications based on the requested API calls during the execution process.
- B. The approach can be integrated as a lightweight anti-ransomware module on end devices. Upon installation of new apps, the module extracts the API features and sends them to a trained classifier for classification. Based on the results, the module allows the app to run or Figure reports a detection to the end user.
- C. The approach can be deployed on Windows application stores to categorize apps before they are available to the general public.
- D. Host-based and market-based implementations of the approach can be utilized to provide additional security measures. This implementation can help verify apps that are not available on official app stores.

6.4 Summary

Chapter 7 provides a summary of the important results obtained from the research and discusses the validations that were performed to verify these findings. The chapter also compares the results obtained from the benchmark approach to those achieved through the proposed approach. Additionally, some potential applications of the proposed approach are discussed. In the subsequent chapter, the conclusion and future work will be presented.

7. Conclusion & Future Work

In this chapter, we will summarize the main findings and suggest future research directions, highlighting unresolved research problems that require attention from the research community.

7.1 Conclusion

The use of strong encryption by crypto-ransomware, a dangerous type of malware, can render a victim's digital files worthless and may even be irrecoverable, even after the ransom is paid. The creation of a pre-encryption model for crypto-ransomware offering two degrees of pre-encryption detection is the strongest aspect of this research. The signature of the file is compared to known ransomware signatures using SHA-256 hashing, permitting a quick and accurate identification without launching the file and allowing for early detection. In the second detection level, an ML model that has been trained is utilized to identify both known and unknown ransomware by examining the API created during the pre-encryption stage. Once the ML algorithm identifies new or unique malware, the ransomware signature will be stored in the signature repository for future use in the first detection level.

This research has yielded a dataset of newly available ransomware for supervised machine learning and the signature repository, both of which could be advantageous for future research on ransomware. However, it is important to note that the use of pre-encryption detection models has some limitations. For example, ransomware that relies on the Windows API for encryption is not detectable by pre-encryption models. As such, pre-encryption models should be used as an additional measure of

protection, rather than as the sole source of ransomware detection. Additionally, it is important to acknowledge that while malware appears in various forms worldwide, the pre-encryption model is only capable of detecting one variant of ransomware.

7.2 Limitation & Future Work

The results obtained from the pre-encryption model show promise, but further improvements are necessary before integrating it into the product. One of the challenges faced in this process is the need for a complete installation and configuration of auxiliary applications like Cuckoo Sandbox and MySQL. To address this, it is worth exploring the development of a stand-alone pre-encryption model that does not require different configurations of supporting applications. Creating a stand-alone tool would simplify the implementation process and make it more convenient for users. By packaging all the necessary components and dependencies into a single executable or containerized application, users can easily run the tool without having to deal with complex installations or additional software requirements.

In addition to making the pre-encryption model stand-alone, employing deep learning techniques can enhance its effectiveness in detecting unknown ransomware. Deep learning algorithms, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated success in various domains, including image recognition and natural language processing. By training a deep learning model on a diverse dataset containing known ransomware samples as well as benign software, the model can learn to recognize patterns and features that indicate ransomware attacks. Furthermore, techniques like transfer learning can be explored, where a pre-trained model on a large dataset, such as ImageNet, is fine-tuned for the specific task of ransomware detection. This approach can save computational resources and training time while still achieving high performance.

CHAPTER 7: CONCLUSION & FUTURE WORK

To improve the model's accuracy, it is crucial to ensure a comprehensive and up-to-date training dataset consisting of diverse ransomware samples. Regular updates and inclusion of emerging threats will help the model stay robust against evolving ransomware attack techniques.

In summary, the pre-encryption model's promising results call for further improvement before integration into the product. Developing it as a stand-alone tool and leveraging deep learning techniques can enhance its usability and effectiveness in detecting unknown ransomware. Continuous updates to the training dataset and exploration of transfer learning techniques will contribute to the model's accuracy and adaptability.

Bibliography

- [1] Ransomware attack report. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.
- [2] Diro, A., Reda, H., Chilamkurti, N., Mahmood, A., Zaman, N., Nam, Y., 2020. Lightweight authenticated encryption scheme for the internet of things based on publish-subscribe communication. *IEEE Access* 8, 60539–60551.
- [3] Humayun, M., Niazi, M., Jhanjhi, N., Alshayeb, M., Mahmood, S., 2020. Cyber security threats and vulnerabilities: a systematic mapping study. *Arab. J. Sci. Eng.* no, 0123456789.
- [4] Hull, G., John, H., Arief, B., 2019. Ransomware deployment methods and analysis: views from a predictive model and human responses. *Crime Sci.* 8 (1), 2. Kok, S.H., Abdullah, A., Jhanjhi, N.Z., Supramaniam, M., 2019. Ransomware, threat and detection techniques: a review. *Int. J. Comput. Sci. Netw. Secur.* 19 (2), 136–146.
- [5] Kok, S.H., Abdullah, A., Jhanjhi, N.Z., Supramaniam, M., 2019. Prevention of cryptoransomware using a pre-encryption detection algorithm. *Computers* 8 (4), 1–15.
- [6] Homayoun, S. et al., 2019. DRTHIS: Deep ransomware threat hunting and intelligence system at the fog layer. *Future Gener. Comput. Syst.* 90, 94–104.
- [7] Mathur, A., Idika, N., 2007. A Survey of Malware Detection Techniques. *Dep. Comput. Sci. Purdue Univ.*, no. March 2007.
- [8] Cabaj, K., Gregorczyk, M., Mazurczyk, W., 2018. Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics. *Comput. Electr. Eng.* 66, 353–368.
- [9] Lee, S., Kim, H.K., Kim, K., 2019. Ransomware protection using the moving target defense perspective. *Comput. Electr. Eng.* 78, 288–299.
- [10] Cuckoo Sandbox for malware analysis. [Online]. Available: <https://cuckoosandbox.org/download>.
- [11] Virus Share for ransomware samples. [Online]. Available: <https://virusshare.com/>.
- [12] VirusTotal for ransomware samples. [Online]. Available: <https://www.virustotal.com/>.
- [13] VirusTotal for ransomware samples. [Online]. Available: <https://github.com/ytisf/theZoo>.
- [14] Al-rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M., 2019. Crypto-ransomware early

BIBLIOGRAPHY

- detection model using novel incremental bagging with enhanced semirandom subspace selection. *Future Gener. Comput. Syst.* 101, 476–491.
- [15] Yu, Z., Gao, C.Z., Jing, Z., Gupta, B.B., Cai, Q., 2018. A practical publickey encryption scheme based on learning parity with noise. *IEEE Access* 6, 31918–31923.
- [16] Celiktas, B., Karacuha, E., 2018. The Ransomware Detection and Prevention Tool Design by Using Signature and Anomaly Based Detection Methods. Istanbul Technical University.
- [17] Ren, A.L.Y., Liang, C.T., Hyug, I.J., Brohi, S.N., Jhanjhi, N.Z., 2020. A three-level ransomware detection and prevention mechanism. *EAI Endorsed Trans. Energy Web* 7 (26), 1–7.
- [18] Alhawi, O.M.K., Baldwin, J., Dehghantaha, A., 2018. Leveraging machine learning techniques for windows ransomware network traffic detection. *Adv. Inf. Secur.* 70, 1–11.
- [19] E. P. Torres P. and S. G. Yoo, 2017. Detecting and neutralizing encrypting Ransomware attacks by using machine-learning techniques: a literature review. *Int. J. Appl. Eng. Res.* 12 (18), 7902–7911.
- [20] Hussain, K., Hussain, S.J., Jhanjhi, N.Z., Humayun, M., 2019. SYN flood attack detection based on bayes estimator (SFADBE) for MANET. In: 2019 Int. Conf. Comput. Inf. Sci. ICCIS 2019, pp. 1–4.
- [21] Alzahrani, A., et al., 2018. Randroid: Structural Similarity Approach for Detecting Ransomware Applications in Android Platform, pp. 892–897.
- [22] Cimitile, A., Mercaldo, F., Nardone, V., Santone, A., Visaggio, C.A., 2018. Talos: no more ransomware victims with formal methods. *Int. J. Inf. Secur.* 17 (6), 719–738.
- [23] Kardile, A.B., 2017. Crypto Ransomware Analysis and Detection Using Process Monitor.
- [24] Techniques, D., Analysis, M., 2019. A comparative assessment of obfuscated ransomware detection methods 23(2), 45–63.
- [25] Shaukat, S.K., Ribeiro, V.J., 2018. RansomWall: a layered defense system against cryptographic ransomware attacks using machine learning. *IEEE*.
- [26] Ami, O., Elovici, Y., Hendler, D., 2-18. Ransomware Prevention using Application Authentication-Based File Access Control.
- [27] Kok, S. H., Azween Abdullah, and N. Z. Jhanjhi. "Early detection of crypto-ransomware using pre-encryption detection algorithm." *Journal of King Saud University-Computer*

BIBLIOGRAPHY

- and Information Sciences (2020).
- [28] Almousa, May, Sai Basavaraju, and Mohd Anwar. "API-Based Ransomware Detection Using Machine Learning-Based Threat Detection Models." 2021 18th International Conference on Privacy, Security and Trust (PST). IEEE, 2021.
- [29] Zakaria, Wira ZA, et al. "RENTAKA: A Novel Machine Learning Framework for Crypto-Ransomware Pre-encryption Detection." International Journal of Advanced Computer Science and Applications 13.5 (2022).
- [30] Anand, P. Mohan, PV Sai Charan, and Sandeep K. Shukla. "A Comprehensive API Call Analysis for Detecting Windows-Based Ransomware." 2022 IEEE International Conference on Cyber Security and Resilience (CSR). IEEE, 2022.
- [31] Alqahtani, Abdullah, Mazen Gazzan, and Frederick T. Sheldon. "A proposed Crypto-Ransomware Early Detection (CRED) Model using an Integrated Deep Learning and Vector Space Model Approach." 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2020.
- [32] Kim, Y., Kim, S., & Shin, S. Y. (2021). RMOC: A Ransomware Classification System Using a Restricted Boltzmann Machine and an Online Clustering Algorithm. IEEE Access, 9, 35846-35860.
- [33] Park, J., Jung, I., Kim, K., & Kim, J. (2021). MELCOR: Machine Learning Based Ransomware Classification for Smart Grid. IEEE Transactions on Smart Grid, 12(3), 2193-2203.
- [34] Rashid, T. A., Azab, M., & Ali, M. H. (2021). RANSID: Ransomware Detection Framework Using Behavioral Analysis and Machine Learning. IEEE Access, 9, 109963-109980.
- [35] Nguyen, T. D., Nguyen, D. T., Nguyen, D. D., Nguyen, D. D., & Le, N. T. (2021). ROPsight: Detection of Ransomware Attacks Using Return-Oriented Programming. IEEE Access, 9, 107169-107183.
- [36] Jung, I., Lee, S., Kim, K., & Kim, J. (2022). RaFS: A Rule- and Machine Learning-Based Ransomware File System Detection Framework. IEEE Access, 10, 55571-55582.
- [37] Jang, J., & Kim, J. (2022). RaRE: Ransomware Attack Recognition Engine Based on Sequence Modeling. IEEE Access, 10, 5645-5657.
- [38] El, I., Li, N. R., & Murphy, M. J. (n.d.). Theory and Applications Machine Learning in Radiation Oncology.
- [39] Ellis, K., Kerr, J., Godbole, S., Lanckriet, G., Wing, D., & Marshall, S. (2014). A random forest classifier for the prediction of energy expenditure and type of physical activity

BIBLIOGRAPHY

- from wrist and hip accelerometers. *Physiological Measurement*, 35(11).
<https://doi.org/10.1088/0967-3334/35/11/219>.
- [40] S. Rajasekar, P. Philominathan, and V. Chinnathambi, “Research Methodology”, 2013. Available: <http://arxiv.org/pdf/physics/0601009.pdf>.
- [41] W. C. Booth, G. G. Colomb, and J. M. Williams, “The Craft of Research.”[Online]. Available: http://sir.spbu.ru/en/programs/master/master_program_in_international_relations/digital_library/Book_Research_seminar_by_Booth.pdf
- [42] C. Woody, “Chapter 3: Research Methodology,” 2001. Available: https://shodhganga.inflibnet.ac.in/bitstream/10603/2026/16/16_chapter_3.pdf
- [43] Ahmad-Azani, N.I., Yusoff, N., Ku-Mahamud, K.R., 2018. Fuzzy discretization technique for bayesian flood disaster model. *J. Inf. Commun. Technol.* 17 (2), 167–189.
- [44] Fawagreh, K., Gaber, M.M., Elyan, E., 2014. Random forests: From early developments to recent advancements. *Syst. Sci. Control Eng.* 2 (1), 602–609.
- [45] Kok, S., Abdullah, A., Supramaniam, M., Pillai, T.R., Hashem, I.A.T., 2019. A comparison of various machine learning algorithms in a distributed denial of service intrusion. *Int. J. Eng. Res. Technol.* 12 (1), 1–7.
- [46] Gowthaman, A., Sumathi, M., 2015. Performance study of enhanced SHA-256 algorithm. *Int. J. Appl. Eng. Res.* 10 (4), 10921–10932.
- [47] Sgandurra, D., Munoz-gonzalez, L., Mohsen, R., Lupu, E.C., 3026.
Automated dynamic analysis of ransomware: benefits, limitations and use for detection.
- [48] Desktop Windows Version Market Share World- wide (Mar 2019–Mar 2020). StatCounter, 2020. [Online]. Available: <https://gs.statcounter.com/osversionmarketshare/windows/desktop/worldwide>.
- [49] Rehman, A., Latif, S., Zafar, N.A., 2019a. Automata based railway gate control system at level crossing. In: 2019 Int. Conf. Commun. Technol. ComTech 2019, no. ComTech, pp. 30–35.
- [50] Rehman, A., Latif, S., Zafar, N.A., 2019b. Formal modeling of smart office using activity diagram and non deterministic finite automata. In: 2019 Int. Conf. Inf. Sci. Commun. Technol. ICISCT 2019, pp. 1–5.
- [51] VDMTools User Manual. Kyushu University, 2016.
- [52] Larsen, P.G., Lausdahl, K., Coleman, J., Wolff, S., 2015. Overture VDM-10 Tool Support: User Guide, no. September. Aarhus University.

BIBLIOGRAPHY

- [53] Read and Write function APIs. [Online]. Available: <https://docs.microsoft.com/>.
- [54] Sharma, Anshika, Himanshi Babbar, and Amit Kumar Vats. "Ransomware Attack Detection in the Internet of Things using Machine Learning Approaches." 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC). IEEE, 2023.

Appendices

Appendix-A

Code for Machine Learning

```

# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import VotingClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier

#CSV File Upload from local machine
from google.colab import files
uploaded = files.upload()
cell_df = pd.read_csv('Goodware3-12.csv')

#Classes Distribution
benign_df = cell_df[cell_df['Label (1 Ransomware / 0 Goodware)']==0][0:943]
malignant_df = cell_df[cell_df['Label (1 Ransomware / 0 Goodware)']==1][944:1847]

#Removing Unwated Columns (bcz these cant be executed)
y = np.asarray(cell_df['Label (1 Ransomware / 0 Goodware)'])
cell_df.pop('id')
cell_df.pop('Label (1 Ransomware / 0 Goodware)')
cell_df.pop('Ransomware Family')
feature_df = cell_df
X = np.asarray(feature_df)

#Training and Testing the dataset using Split Method
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=10)

#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier
#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)
print("RANDOM FOREST MODEL")
#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
y_predict1=clf.predict(X_test)
#print (y_predict)

# Model Accuracy, how often is the classifier correct?
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print(" RF_Accuracy:", metrics.accuracy_score(y_test, y_predict1))
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict1, digits=4))
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = clf, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation ")
print(accuracies.mean())
print(accuracies.std())

```

```

# get the probability distribution
r_probs=[0 for i in range(len(y_test))]
probas = clf.predict_proba(X_test)
fpr4,tpr4,_d= metrics.roc_curve(y_test, r_probs)
# get false and true positive rates
fpr, tpr, thresholds = roc_curve(y_test, probas[:,0], pos_label=0)
# get area under the curve
roc_auc = auc(fpr, tpr)
# PLOT ROC curve
plt.figure(dpi=120)
plt.plot(fpr, tpr, lw=1, color='green', label=f'AUC = {roc_auc:.3f}')
plt.plot(fpr4,tpr4,'--',label="Random")
plt.title('ROC Curve for Random Forest classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.legend()
plt.show()

from sklearn.metrics import confusion_matrix
# compute the confusion matrix
cm = confusion_matrix(y_test,y_predict1)
import seaborn as sns
#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Malicious', 'benign'],
            yticklabels=['Malicious', 'benign'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()

#SVM Model
print("SVM MODEL")
from sklearn import svm
classifier = svm.SVC(kernel='linear', gamma='auto', C=2)
classifier.fit(X_train, y_train)
y_predict2 = classifier.predict(X_test)

#Evaluation
from sklearn.metrics import classification_report
print("SVM Accuracy:",metrics.accuracy_score(y_test, y_predict2))
print(classification_report(y_test, y_predict2, digits=4))
print("Applying k-Fold Cross Validation ")
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print(accuracies.mean())
print(accuracies.std())

# compute the confusion matrix
cm = confusion_matrix(y_test,y_predict2)
import seaborn as sns
#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Malicious', 'benign'],
            yticklabels=['Malicious', 'benign'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()

# Create Decision Tree classifier object
clf1 = DecisionTreeClassifier()
# Train Decision Tree Classifier
clf1.fit(X_train,y_train)
#Predict the response for test dataset
y_pred3 = clf1.predict(X_test)

print("Decision Tree Accuracy:",metrics.accuracy_score(y_test, y_pred3))
print(classification_report(y_test, y_pred3, digits=4))
print("Applying k-Fold Cross Validation ")
accuracies = cross_val_score(estimator = clf1, X = X_train, y = y_train, cv = 10)
print(accuracies.mean())
print(accuracies.std())

```