A Robust Machine Learning Approach for Malware Detection in Linux

By

Rabbiya Tariq

(Registration No: 00000399879)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

Master of Science in
Information Security

Supervisor: Dr. Waseem Iqbal

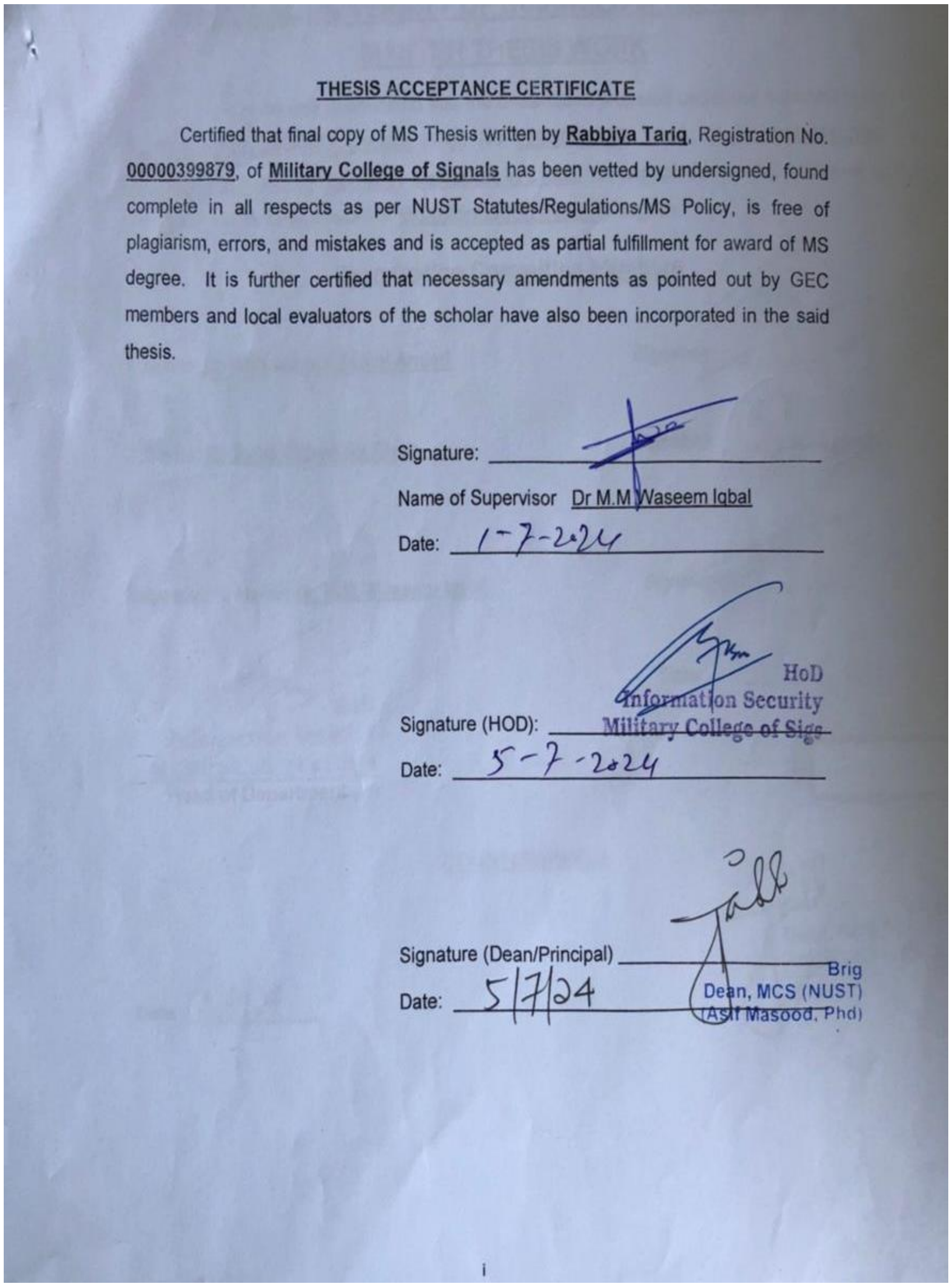
Military College of Signals (MCS)

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2023)

THESIS ACCEPTANCE CERTIFICATE



THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by **Rabbiya Tariq**, Registration No. **00000399879**, of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations/MS Policy, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members and local evaluators of the scholar have also been incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor Dr M.M Waseem Iqbal

Date: 1-7-2024

Signature (HOD): [illegible]

HoD
Information Security
Military College of Sigs

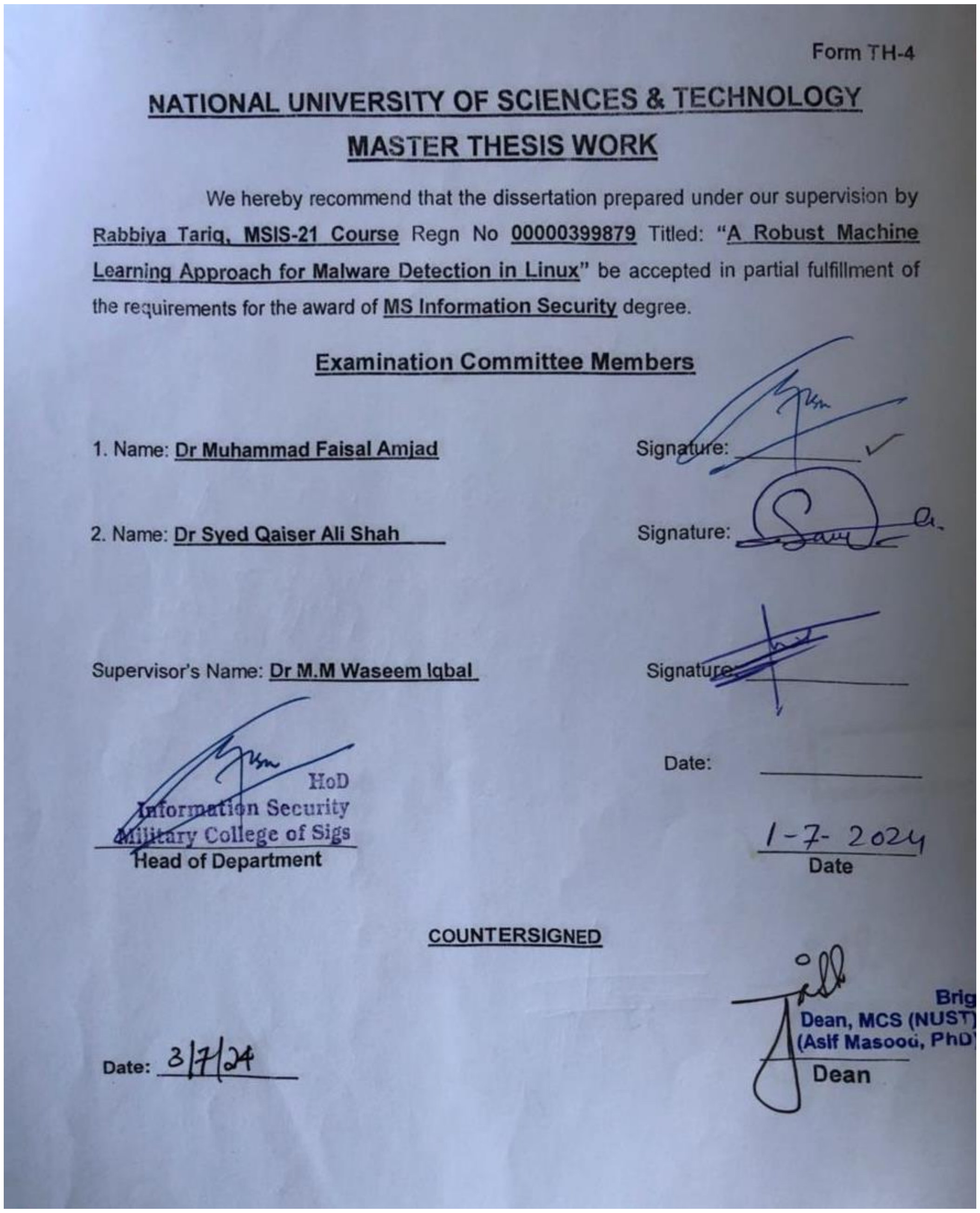
Date: 5-7-2024

Signature (Dean/Principal) [illegible]

Brig
Dean, MCS (NUST)
(Asif Masood, Phd)

Date: 5/7/24

i



Form TH-4

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

MASTER THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by Rabbiya Tariq, MSIS-21 Course Regn No 00000399879 Titled: "A Robust Machine Learning Approach for Malware Detection in Linux" be accepted in partial fulfillment of the requirements for the award of MS Information Security degree.

Examination Committee Members

1. Name: Dr Muhammad Faisal Amjad	Signature: [illegible]
2. Name: Dr Syed Qaiser Ali Shah	Signature: [illegible]
Supervisor's Name: Dr M.M Waseem Iqbal	Signature: [illegible]
	Date:

[illegible]
HoD
Information Security
Military College of Sigs
Head of Department

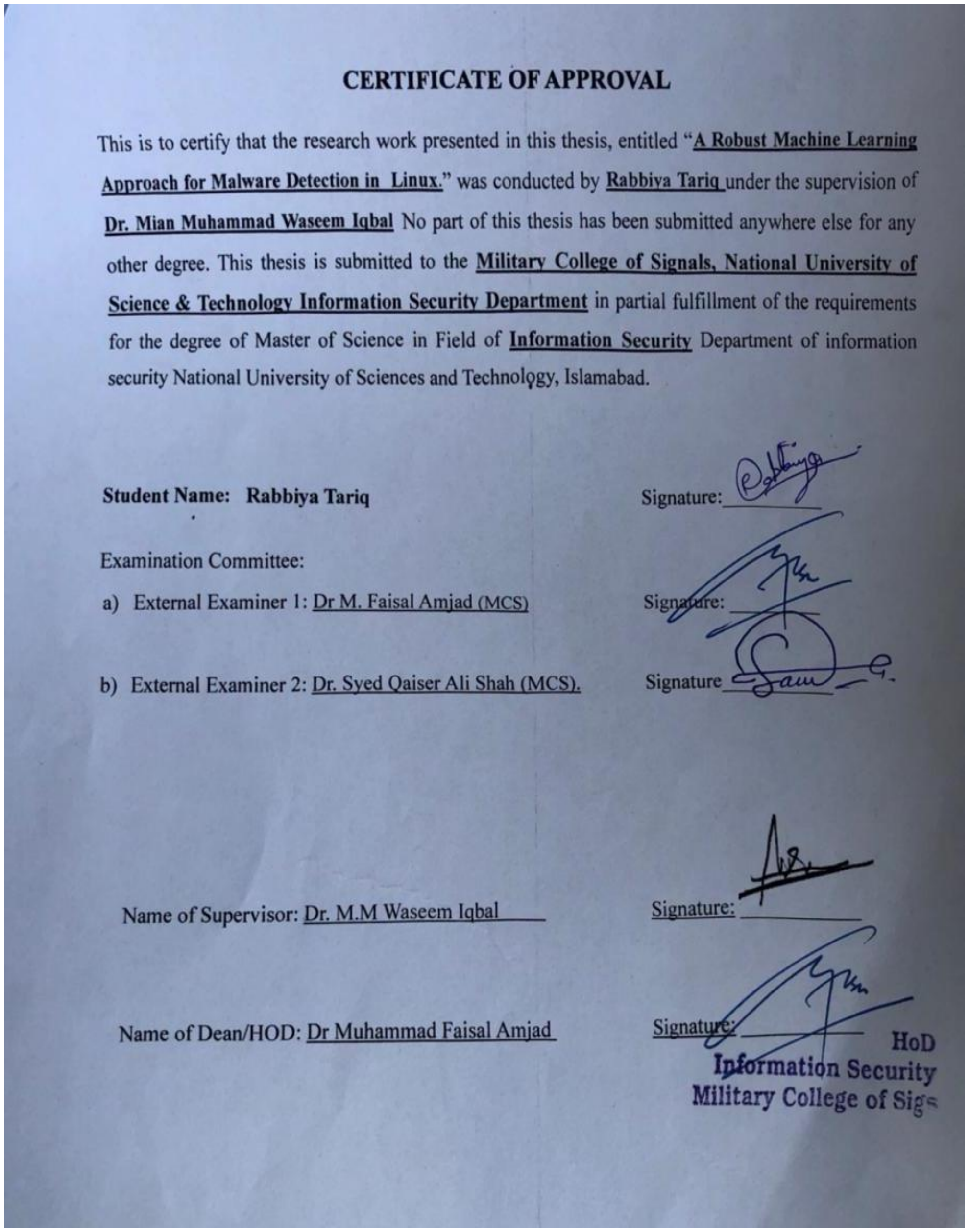
1-7- 2024
Date

COUNTERSIGNED

Date: 3/7/24

[illegible]
Brig
Dean, MCS (NUST)
(Asif Masood, PhD)
Dean

CERTIFICATE OF APPROVAL



CERTIFICATE OF APPROVAL

This is to certify that the research work presented in this thesis, entitled "**A Robust Machine Learning Approach for Malware Detection in Linux.**" was conducted by **Rabbiya Tariq** under the supervision of **Dr. Mian Muhammad Waseem Iqbal** No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Military College of Signals, National University of Science & Technology Information Security Department** in partial fulfillment of the requirements for the degree of Master of Science in Field of **Information Security** Department of information security National University of Sciences and Technology, Islamabad.

Student Name: Rabbiya Tariq Signature: [illegible]

Examination Committee:

a) External Examiner 1: Dr M. Faisal Amjad (MCS) Signature: [illegible]

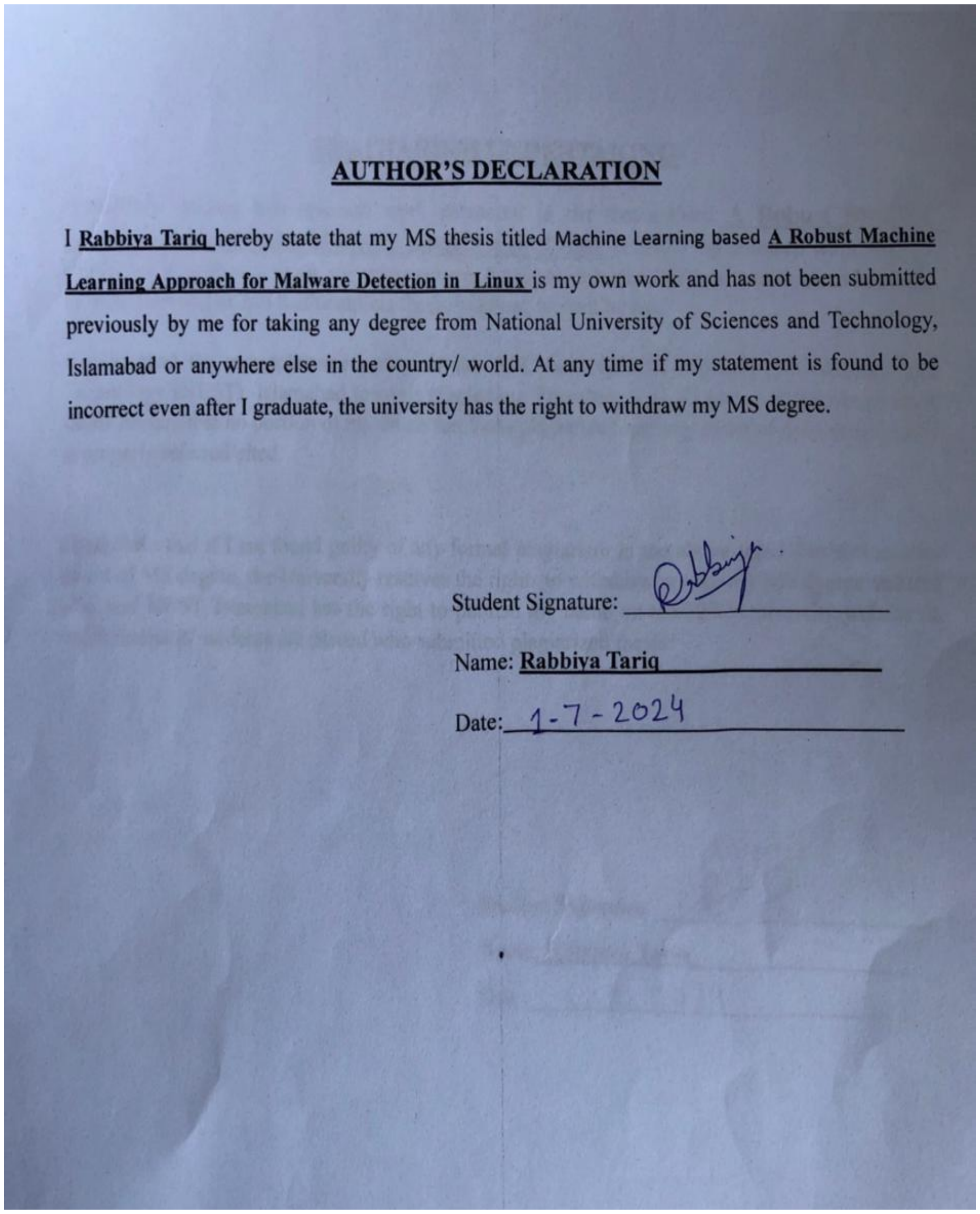
b) External Examiner 2: Dr. Syed Qaiser Ali Shah (MCS). Signature [illegible]

Name of Supervisor: Dr. M.M Waseem Iqbal Signature: [illegible]

Name of Dean/HOD: Dr Muhammad Faisal Amjad Signature: [illegible]

HoD
Information Security
Military College of Sig

AUTHOR'S DECLARATION



AUTHOR'S DECLARATION

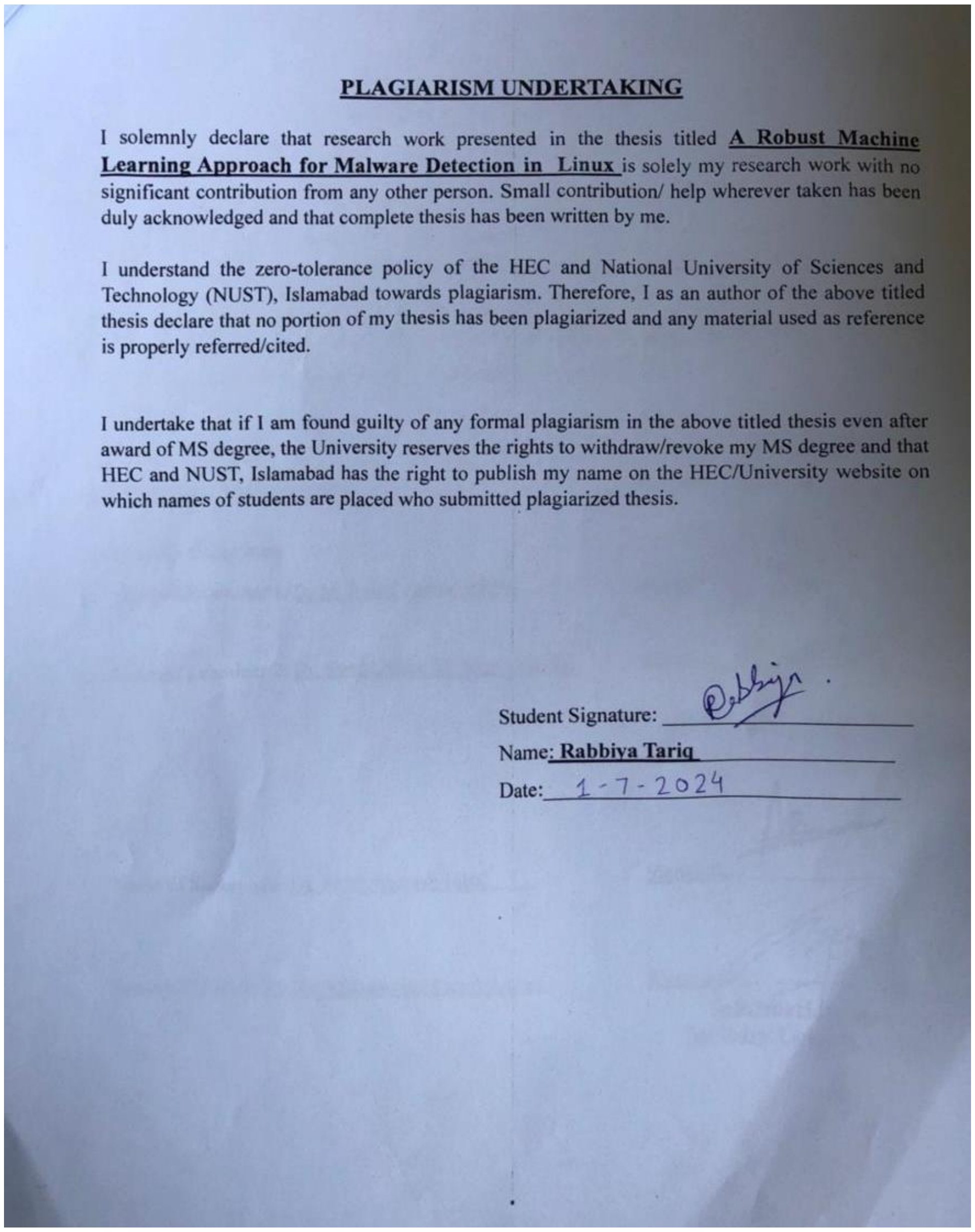
I **Rabbiya Tariq** hereby state that my MS thesis titled Machine Learning based **A Robust Machine Learning Approach for Malware Detection in Linux** is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world. At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Student Signature: [illegible]

Name: **Rabbiya Tariq**

Date: 1-7-2024

PLAGIARISM UNDERTAKING



PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled **A Robust Machine Learning Approach for Malware Detection in Linux** is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

Student Signature: [illegible]

Name: **Rabbiya Tariq**

Date: 1-7-2024

DEDICATION

To my parents, whose love, encouragement, and sacrifices have paved the way for all my achievements. Your unwavering belief in me has been a constant source of strength and inspiration. This thesis is dedicated to you.

To my brothers, who have been my guiding lights and steadfast supporters throughout my academic journey. Your support has been invaluable.

To my mentor and advisor, Dr. Mian Muhammad Waseem Iqbal, for his profound wisdom, guidance, and faith in my abilities. Your mentorship has been instrumental in shaping my path as a researcher.

To my friends, who have been the pillars of strength and a source of joy throughout this academic endeavor. Your friendship has made this journey not only bearable but also profoundly enjoyable.

ACKNOWLEDGEMENTS

Primarily, I am profoundly grateful to Almighty Allah, the most gracious and the most merciful, who bestowed upon me health, wisdom, knowledge, and the power of communication. I owe a great deal of gratitude to my supervisor, whose valuable guidance, encouragement, and supervision made it possible for me to undertake this project and complete my training in this area. I am deeply thankful for the support of my colleagues, especially Syed Sohaib Karim, Major Amara Riaz and Madiha Hassan whose cooperation was instrumental in the completion of my thesis. I am also immensely grateful to my parents, brother and sisters for their motivation, sacrifice, cooperation, love, and affection, which have helped me overcome my difficulties and enabled me to complete this research work.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS and acronyms

ML	Machine Learning
RF	Random Forest
LR	Logistic Regression
FNN	Feedforward Neuro Network
OS	Operating System

Abstract

This thesis has presented the application of machine learning techniques to the detection of a class of ransomware that specifically targets Linux-based systems. Due to the increasing prevalence and complexity of such ransomware, traditional detection methods, which are primarily based on signature-based matching, are becoming obsolescent. This research endeavored to provide a fresh, novel, and effective way of combating malware that affects Linux operating systems in the form of a hybrid analysis novel methodology that combined static and dynamic analysis methods to extract maximum features from malware samples. The obtained features were then used to train different machine learning models, such as Random Forest, Decision Trees, etc. The model's effectiveness was verified using accuracy, precision, recall, F1-score, etc. The experimental results demonstrated the effectiveness of the applying machine learning techniques to improve malware detection and develop a robust means of enhancing cybersecurity in a world where threats are ever-growing

Keywords: Machine Learning, Ransomware, Malware Analysis, Linux

Chapter 1: Introduction

1.1 Overview:

Over the last few years, the threat landscape concerning Linux systems has significantly transformed, with a conspicuous hike in both quantity and complexity of malware targeting these platforms. Ransomware has emerged as one of the most common types of malwares among these threats, causing extensive damage on users. This section provides an overview of Linux malware, concentrating mainly on one type of Linux malware i.e., ransomware, and describes the significance of creating efficient detection systems based on machine learning techniques.

Linux is ranked more secure as an operating system alongside others. But recently cyber criminals have turned to malware attacks on Linux systems in personal as well as enterprise levels just because these systems have become popular among the computer users for both personal as well as enterprise use. There are different types of malwares in Linux, Among the malwares used are the ransomwares which can be defined as a form of malicious software that blocks access to data files by simply encrypting them and later demands for payment from the owner to grant him or her access to his or her own files. In the past few years, ransomware attacks have increased to a huge level.

Cybersecurity professionals face an important challenge in relation to the growth of ransomware that is designed specifically for Linux. In many cases, the standard signature-based detection methods do not perform as fast as the variations in malware multiply rapidly. Therefore, there is a need for more sophisticated ways for identifying when Linux machines are being attacked since the existing ones are not reliable. The primary objective of this study is to offer solutions that will help detect Linux-based malware using machine learning methodologies while focusing on Linux based ransomware and its different variants.

1.2 Motivation:

The reason for doing this research is that there is a growing menace of malware that mainly targets Linux based systems, particularly ransomware. Linux has been traditionally seen as more secure than any other operating system. However, recent developments show an increase in Linux-specific malware which poses great danger for individual users as well as corporations. For one to detect and forestall these threats there is a need for new ways and means where machine learning can be highly effective as per practical solutions.

Most of the traditional antivirus solutions' ability to detect emerging Linux malware by means other than signature-based means is not effective. This is because they are not able to identify and handle malicious programs that are not available in their databases. Machine learning, however, offers a more effective and adaptable approach to malware detection. In fact, it is the use of machine learning algorithms that has enabled antivirus programs to detect malicious programs with an extremely high probability and without the need for regular updates. In other words, machine learning algorithms have enabled antivirus programs to detect common characteristics and patterns in the known viruses, hence the ability to detect unknown malware.

1.3 Problem Statement

Cyber-security professionals have a big task when it comes to handling the growing number of Linux malware, specifically ransomware. It is not easy for conventional signature-based detection techniques to be effective in the face of malware whose types change fast. Linux systems also remain very prone to attacks due to inadequate strong and efficient detection systems. These issues will be resolved by this research endeavor using machine-learning methods to detect Linux based ransomware. The purpose of this research is to address this problem by creating machine learning techniques that can identify ransomware on Linux with particularity.

Ransomware attacks on Linux systems can have catastrophic effects on enterprises or people. Encrypting files while asking for money in exchange for decryption keys is one of the things ransomwares does; this can result in terrible consequences. Spreading to other computers within the same network through lateral movement, one ransomware attack can cause severe damage by infecting many systems apart from the intended target Computer-assisted detection is about finding and controlling such dangers, but old-fashioned antivirus programs cannot cope with them effectively. For detecting malware in particular Code written by computers that can learn from experience or find patterns within data on its own.

1.4 Research Objectives

The fundamental tenets for this research thesis are summarized in the following broad range of objectives:

- **RO1:** To study and analyze the Malware behavior specifically ransomware in Linux Operating system.
- **RO2:** To propose an enhanced Machine Learning based mechanism to detect Malware in Linux
- **RO3**: Comprehensive analysis of proposed methodology with existing solution available.

1.5 Relevance to National Needs

- Cybersecurity is a fundamental aspect of a country's safety and survival in today's digital age. Major government systems could be paralyzed, and strategic data lost to attacks by malicious software aimed at critical areas such as government operations or important facilities. Contributing machine learning algorithms for the detection of Linux malware can make a significant contribution towards global security mechanisms.
- As Linux is used widely in servers, IoT devices and other important systems, hackers are focusing on this platform. For our country's digital infrastructure to be secured we must come up with state-of-the-art ways to figure out and resist these cyber threats.
- Disturbing the confidentiality of individuals may occur after a malware strikes against Linux systems due to data leaking. It is worth noting that this study aimed at protecting national intelligence and the personal life of citizens from unauthorized data access and leaking.
- To address this issue, there needs to be a collaboration of universities, government institutions, and private sector organizations dealing with cybersecurity issues. For example, such collaboration can produce hacking tools for computer systems that are up to date.

1.6 Area of Application

The usage of machine learning to detect Linux malware has multiple uses in different industries:

- The government and defense - shielding sensitive information from cyber assaults.
- Health care - securing electronic health information and devices against malware threats, while
- Finance - at the same time guaranteeing that all financial transactions go on securely without being hindered by any extortion maneuver directed at the banking system.

- Critical Infrastructure -To secure fundamental services like energy, transport, and telecommunications, from potential cyber threats.

The cost of undesired and destructive cyber-attacks can be prevented, and such critical systems protected through the application of AI-based malware detection mechanisms for example. This works by closely scrutinizing vast repositories of pre-existing malware strains enabling machine-learning algorithms to spot shared attributes as well as recurrent trends; thus, enabling them to recognize novel malware types that were not present before in known antivirus software systems.

1.7 Advantages

- The aim of malware detection research is to enhance cybersecurity practices and hearten computer systems and networks from cyber threats with greater effectiveness in developing detection technologies.
- Enhanced cybersecurity practices come because of using better detection techniques by this, it is possible for experts to contrive more efficient detection techniques which would enable organizations protect themselves more effectively against viruses thus reducing possible damage after a hacking event.
- The field of malware detection research helps in developing a good understanding of emerging cyber risks as well as methods used by cyber criminals in their activities hence more focus in this area is essential. Knowing the behavior and features that define several types of malwares will enable scholars to come up with mechanisms that are capable of accurately detecting these programs thereby ensuring computer security.

- It is here that research projects in malware detection present themselves as excellent opportunities for students interested in pursuing careers in cybersecurity as well as for researchers. It is through solving actual problems that require practical solutions that students can get direct training thus acquiring skills necessary for conducting their job duties in this field.
- Detection research on malware therefore has worldwide significance. By creating better means of detecting and sharing these findings with wider cyber security networks, scholars contribute towards the global efforts to eradicate e-crime and make internet less dangerous.

1.8 Thesis Organization

The research work has been organized and distributed in the following chapters listed below. Also Fig. 1.1, represents the layout of this thesis which is described in detail in this section.

- **Chapter 1**: A brief introduction is given in this chapter. Problem statement followed by research objectives, relevance to national need, area of application and its advantages are elaborated.
- **Chapter 2**: This chapter describes related works carried out by different researchers on Linux in the malware field. Then we will see an overview Linux Malwares, and specially ransomware and its 10 types that we will use to perform our research on. At the end there is an attack chain of Linux based ransomwares.
- **Chapter 3**: This chapter explains the proposed model in detail. It is covering the major research objectives of this research by explaining all phases of proposed model in detail and presenting how malware detection will be performed using proposed methodology

- **Chapter 4**: This Chapters presents all the practical implementation of the modules discuss in chapter three and the results of our analysis
- **Chapter 5:** This Chapter sums up the research with conclusions drawn and discusses the future aspects of the research.

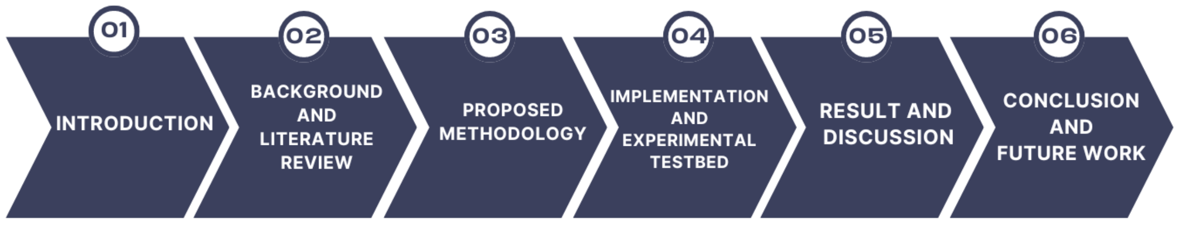

01
INTRODUCTION
02
BACKGROUND AND LITERATURE REVIEW
03
PROPOSED METHODOLOGY
04
IMPLEMENTATION AND EXPERIMENTAL TESTBED
05
RESULT AND DISCUSSION
06
CONCLUSION AND FUTURE WORK

Figure 1 Thesis Organization

CHAPTER 2: Background and Literature Review

2.1 Introduction:

This chapter provides a comprehensive summary of previous research on Linux malware detection and how different research has detected malware within Linux environment. It also presents background knowledge about Linux based malware, how these are increasing at a rapid rate, specifically Linux Ransomware and its variants.

2.2 Literature Review:

For the literature we have studied different research work but the existing literature on Linux malware detection is very less as compared to windows malware detection. In one work [1] authors present a new system for identifying Linux malware from a non-signature-based angle focusing on zero-day malware detection. They developed a system from the ground up, named ELF-Miner, that forensically analyzes the ELF header files. This generates and selects 383 specific uniquely curated features from the ELF header files that distinguish malware from benign files. The study demonstrates the features classification's potential with a filtering algorithm known as information gain and removes many random and irrelevant features utilizing a pre-processing filter. It then utilizes 383 features in several classical and bio-inspired machine learning classifiers. From repeated testing on a pre-defined test set with 709 Linux malware samples, the ELF-Miner shows over 99% detection accuracy while maintaining a very low false alarm rate under 0.1%. The main breakthrough of the paper is this feature, as it provides a solid foundation for malware detection on Linux without relying on any known malware signatures, enabling it to identify unknown threats in real-time. Given the rise of Linux systems in various fields, they are a prime target for cyber-attacks, making such an approach relevant.

In another work, Author [2] proposes a method to accurately identify the presence of malware in Linux based IoT devices using static feature analysis methodology. The novelty of this work lies in the method’s speed and accuracy in resource-constraint IoT devices and a proposed solution to the same. The features are extracted from ELF executables, and then, using the chi-square method, the features are reduced drastically and retrained on a limited feature set to still achieve a high detection. This method allows this framework to run on such hardware. After feature reduction, the author trains the model on machine-learning algorithms such as J48, JRIP, PART, Random Forest, Naïve Bayes, Logistic, and RIDOR. All six classifiers were able to achieve above 99% accuracy and precision, with extremely low FPR and FNR ratios considered helpful for a software-based system searched for malicious software in an IoT device known for minimal computational power. This case is particularly relevant as Linux-based IoT devices grow in popularity in various fields and become a target for malware attacks.

In this paper the author [3] suggests a new way to detect library functions in Linux malware, specifically in malware that is statically linked and deprived of information on function names and addresses. This method is interesting primarily because the modification mentioned above impedes traditional methods of function-level analysis. A more holistic approach was taken to the issue, and, as a result, pattern matching was used to detect library functions among 2,256 malware samples, the majority of which were designed with the help of Intel 80386 architecture. The authors claim that their strategy was highly effective: Library functions have been found in over 90% in 97.7% of cases. The available data is the result of combining static analysis tools with pattern matching, and they can be seen as a new tool for similar studies on threats in Linux space. Therefore, this study is valuable both in terms of developing understanding of how Linux malware is built and increasing the quality of forensic studies.

Authors [4] discuss a novel approach to machine learning enabled in detecting Linux malwares from the dynamical extraction of system calls. The new mechanism is a breakthrough because it enhances malware detection on the Linux platform, which is often less studied compared to malware entry into Windows environments. The Key Components include Dynamic Extraction of System Calls. The authors dynamically extract system calls using strace, a system call tracer. Such extraction is critical because it captures the semantics of the executables' behavior which is essential in detecting malice. The authors used benign and malwares executables to determine the optimal set of features that would make the given model a classifier or not. The paper assesses the performance of classifiers such as SVM and NB in classifying a file as malware or benign based on the extracted feature. The proposed models will have exploited the dynamical nature of system calls to effectively make decisions regarding the classification. The approach led to a high classification accuracy of 97%, meaning the model is flexed and suitable for differentiating malware in Linux environments. - The authors used real malware in their experimentation, making the findings more application warrantable and robust.

The work presented by Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti, is a crucial addition to the field of cybersecurity research for several reasons. First and foremost, despite the increasing importance of embedded devices and the Internet of Things, investigations into malware have historically been scarce and have focused primarily on Windows-based malware due to its market dominance. Thus, the decision to limit one's study to Linux malware represents a significantly understudied topic, which is more than beneficial. The key insights and contributions of this work include: the diversity of targets; b the development of the malware analysis pipeline; c the overview of Linux-specific techniques; and d the empirical discovery described in the Conclusions. Furthermore, this work may serve as a valuable resource for your future thesis. By providing open-source tools and results of the conducted research, the authors suggest an effective way future scholars may address research questions. Therefore, it is reasonable to include this work into the dissertation structure since it provides a methodological framework based on real research experience.

In another approach [5] authors used for identifying malware on Linus systems employing system tracers such as ftrace and strace. This paper is of high importance as Linux powers a vast number of diverse platforms, namely servers and connected IoT devices, and it is targeted increasingly often by malicious software due to these platforms' connectivity and importance. Prominent points. Malware Detection Techniques. The primary approach mentions the drawbacks of signature-based detection and the necessity of dynamic analysis to find zero-day malware. This approach is feasible using a system tracer that is employed for monitoring system execution during the process. System Tracers and Sandboxing. The system tracer allows for collecting run-time data and is essential for understanding the malware's relationship with the running system. Sandbox is discussed to safely test-run malware for dynamic analysis without risk to the host system. Machine Learning Models. The paper introduces machine learning models for categorizing the collected data into malicious and benign. This approach was considerably successful in increasing the detection rates of signature systems facing a new type of sophisticated malware. Challenges and Recommendations. Various difficulties are tackled, such as sandboxing not being as bulletproof as advertised, the increase in computational costs due to the tracing, and the need for more high-quality datasets to train the models effectively. Future Research Directions. The paper outlines future research such as further refinement of tracing and sandboxing techniques, creating more adaptable models that require fewer false positives, and various ML models that can adapt to new forms of malware. In conclusion, the study's authors demonstrate the urgency of utilizing modern technologies and methodologies against modern malware that interests Linus systems. It provides valuable data for increasing the resilience of the anti-malware systems and preventing most of the infections.

Another [6] methodology combines machine learning methodology with tree-based classifiers to detect malware. In addition, the authors discuss the use of two analysis types: static and dynamic. This method helps to increase the precision of detection for unknown malware. Furthermore, it employs Gain Ratio and Symmetric Uncertainty for feature selection. As classifiers Random Forest, J48, and REPTree are used. This method has a high accuracy of 99.82% and a minimal false-positive rate of 0.002%. The researchers analyzed a large amount of data, 24,000 files, malware, and not facilitating research validation. As a result, it can be said that the use of a hybrid method in analyzing showed on Linux experimental that it is effective. It is recommended for future research to reduce the number of features and to expand the analysis to other file types.

Authors in research [7] aims to tackle ransomware detection through an approach that uses dynamic features and deep learning methods. In this paper, the scholars examine the various dynamic features such as API call sequences, DLL usage, enumerated directories, mutual exclusions, and registry key operations and how they affect ransomware detection accuracy. The methods used in dynamic analysis are as follows: API call sequences and the pattern that the executable program makes are investigated since the dynamic API call sequence often reveals a pattern of malicious use. DLLs and patterns of uses are also studied since some DLLs are frequently used by malware. Enumerated directories in which ransomware is repeated and tries to locate and encrypt the user’s files are simulated, as are the mutuality in which ransomware uses files to alternately encrypt and encrypt ransom files. Furthermore, mutual exclusions are determined by the analysis of mutual exclusions that ransomware may use to prevent files from being accessed simultaneously or maintain control of the system. SQL and how it interacts with system registry keys are analyzed, and the architecture of the design reveals intent with respect to persistence or configuration changes. As for the models, CNN and LSTMs were used, as well as simple MLP. In the experiment conducted to obtain the results, the scholars used three datasets that infected the system and one dataset that had not infected it. The results produced all the accuracies and were computed and analyzed using TPR and FPR, as showed in the results. Hence, the experiment concludes that API call sequences produce an excellent true-positive rate but at the cost of high false-positive rates. On the other hand, enumeration directories produce lower false positives and accurate results.DLL. This dynamic feature results in a high true positive rate, but it also causes a high false positive rate since it fails to classify benign behavior. However, mutual exclusions produce noble results, as do registry operations, which remain to be adjusted but is the most revealing feature of ransomware. It is predetermined that the symbiosis of the dynamic features and deep learning models increase ransomware detection. Thus, further work is in the benefit to the highly adaptable framework with this integration.

In paper [8] on which the authors list various components of their analysis, as well as the machine learning models, they used to detect ransomware. These are the detailed aspects based on the techniques and models they provided: **Analysis Techniques:** File and Network Monitoring: The system files are examined for susceptibility to attack after being introduced in the computer subunit. All packets are observed for traversing the network system to identify any packet that may be ransomware. Behavioral Analysis: The behavioral overview of files and packets into the computer system is recorded, and any piece of software or packet behaving strangely will trigger a red flag. Packet Assessment: All packets in the subunit are inspected heavily. The act is done at a micro-level that scrutinizes each network bit, and any anomaly of the ransomware piece is identified quickly. Random Forest: This was specified during the data training stage. Random Forest is an ensemble learning technique that involves multiple decision trees in making predictions. This technique balances an added advantage of accuracy and being immune to overfitting.

Deep Learning Models: Deep learning models were not specified to include CNNs since the architectural framework was not elaborated in my initial brief. However, the deep learning models are used to sweep through high-level data aspects to quote any pattern that is likely to be challenging for other ML techniques. Feature Extraction with Python's pefile Module: The decompile tool is used to extract features from executables for further training of the model. This includes metadata from the header files and attributes of the binary. Feature Selection and Extraction: Features are selected by opportunistic teachers who review the data to select the most likely suspects to pick from the features that the researchers used to train the features selected. The magnitude of file rigging is identified in a lucrative technique. For example, a hashed value can be rough in a certain smart size square or a debug size, or an accurate major image version in the system files. The models are programmed in a labeled training dataset with both benign and malicious programs. The training is an iterative phase by which the developers inform the models of the normal and customary file behavior concerning the file characteristics given. The performance of the models analyzed against novel and incognito word ransomware to prove the system's daily efficiency. The prototype of the software should maintain high detection of ransomware viruses. The integration of both machine learning models and the analyzing methodology propose the detection of ransomware viruses.

2.3 Summary of Related Work:

Table 1 Summary of Literature

Sr. No	Research Paper	Algorithm Used	Technique	Limitations
1	ELF-miner: using structural knowledge and data mining methods to detect new (Linux) malicious executable	RIPPER, PART, J48	Performed static analysis on ELF executable. Extracted features from elf header and applied machine learning for malware detection	Very small dataset used. Only static analysis is performed
2	Malware detection through mining symbol table of ELF	No machine learning used	Symbol table is extracted from the elf executables and used for malware detection	Static analysis is only done.
3	A machine learning approach for Linux malware detection	IBK-5, Random Forest and Ada boost	Performed dynamic analysis and extract system calls and applied machine	Less Files are used. Only dynamic analysis performed
4	Integrated Static and Dynamic analysis for malware detection	SVM, RBF and J48	PSI is extracted from binary files as static and Api calls are use as dynamic feature to detect malware	Small dataset used. Very less features
5	In-execution dynamic malware analysis and detection by mining information in process control blocks of linux OS.	J-RIP ad J48	Dynamic analysis is performed on PCB and extracted features are used to train ML model.	Small and old dataset. Only dynamic analysis
6	Detection of Advance Linux Malware using ML technique	LMT, Random Forest and NBT	Hybrid analysis is done on ELF files for extracting features and then using features for machine learning	Small dataset, which can affect accuracy of machine learning.

7	Detection of Malicious Executable in Linux Environment Using Tree-Based Classifier	J48 and Random Forest	Both static and dynamic analysis is done to extract features. System calls and elf header is used as features	Very Less features are used.
8	Linux malware detection using extended-symmetric	Ada Boost and Random Forest	System calls are used to extract feature and information gain method	Small dataset. Only focus on system calls

2.4. Overview of Linux Malware ¬ Ransomwares:

Linux malware is malicious software created to target Linux-based operating systems. Although Linux is known for its exceptional security, it can still be vulnerable to malware. Linux malware may infect various systems, including servers, desktop computers, IoT equipment, embedded systems, and network appliances. Since Linux has gained great popularity for personal and industrial purposes, the frequency and acuteness of Linux malware assaults have risen significantly. According to a report published in [9] 55% attacks are observed within Linux environment beating out windows for the first time. In figure-1 we can see the attack ratio on different operating systems.

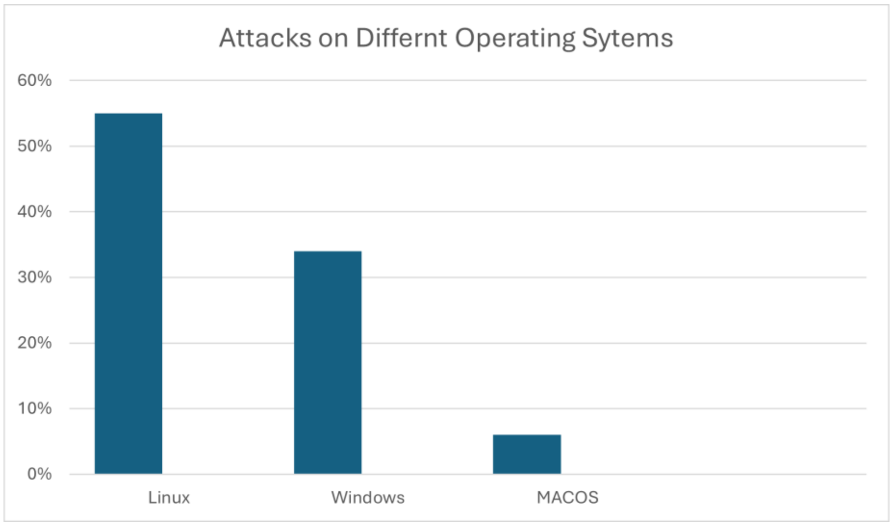


Figure 2 Attack Ratio on Different OS

As we started our research, we found a significant number of increases are reported in the attacks on Linux systems and maximum attacks are Ransomware attacks on Linux systems. Before getting into details, we must know what Ransomware is.

2.4.1. Ransomware:

Malware known as "Ransomware" is capable of infecting computers running on Windows, Mac, Linux distributions, including Debian and Ubuntu. An attack of this kind would encrypt documents after infiltrating a network or device and finding the important ones. Frequently, a message requesting payment for the return of the encrypted files is the first indication that an attack has occurred. This may seem scary to an individual, but it might seriously harm a company's operations and reputation.

2.4.2. Linux Ransomware:

Ransomware attacks against Linux systems rose by 75% in 2022 compared to the previous year. [10] An increasing number of Linux variants have been released by ransomware gangs. In the figure below we can see the rising number of ransomware attacks in Linux environment:

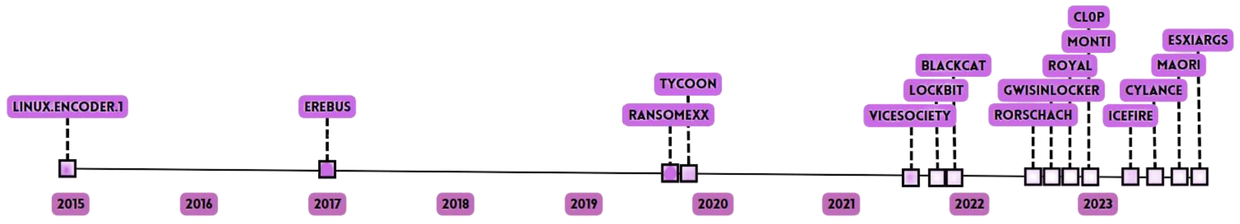


Figure 3 Timeline of Ransomware Attack

2.4.3. Variants of Linux Ransomware:

There are many different variants of Linux Ransomware, however in our research we have include these 10 variants to understand the workflow of Linux ransomwares:

2.4.3.1. AvosLocker:

The sophisticated ransomware known as AvosLocker was first identified in June 2021 and became well-known for its twice extortion method. AvosLocker is intended to both steal and encrypt files from a victim's PC. After that, the attackers demand payment of a ransom before releasing the material to the public.

2.4.3.2. Blackcat:

The ransomware family ALPHV, commonly referred to as BlackCat, is used in ransomware as a service (RaaS) activity. Available on Linux-based operating systems (Debian, Ubuntu, Ready

NAS, Synology), Windows, and VMware ESXi, ALPHV is built in the Rust programming language.

2.4.3.3. Darkside:

The DarkSide ransomware functioned as "ransomware-as-a-service". DarkSide demanded Bitcoin and other cryptocurrency ransoms after encrypting and stealing confidential data from large organizations. Like a legitimate company, they also set up a functional platform and provided real-time chat help.

2.4.3.4. Royal:

Targeting VMware ESXi virtual machines, Royal Ransomware is the most recent ransomware operation that facilitates Linux device encryption. The Equinix Threat Analysis Center found it for the first time. Linux Royal Ransomware uses the command line to execute properly.

2.4.3.5. Cylance:

Cylance is also a Linux based ransomware which operates using command line argument. Its main target is VMware ESXi virtual machines. After getting access to the system, it encrypts all the files specified in the command line argument. And leaves a ransom note at the end.

2.4.3.6. Cl0p:

The deadly Clop ransomware, which is a member of the well-known Cryptomix ransomware family. It encrypts files by planting clop extension on infected systems and actively evading protection measures.

2.4.3.7. Revil:

With the first appearance of REvil in May 2019, the ransomware-as-a-service (RaaS) operation behind it has grown to become one of the most active and successful threat groups. Windows systems have been the main target of REvil. New samples, aimed at Linux systems, have been discovered, though.

2.4.3.8. RansomExx:

Ransomware like LockBit demands money to be paid to unlock the encrypted file. Rather than focusing primarily on consumers, it targets corporations and government bodies. The institutions who would be inconvenienced and have the resources to provide a sizable payment are its possible targets.

2.4.3.9. IceFire:

The ransomware IceFire, also referred to as iFire, encrypts files, appends the ". iFire" extension to filenames and generates a ransom letter called "iFire-readme. txt". Ice Fire's mission is to lock down files until a ransom is paid.

2.4.3.10. LockBit:

Adopting the Ransomware-as-a-Service (RaaS) architecture, LockBit is an extremely persistent and intelligent ransomware program. The ransomware variant LockBit was most often used globally in 2022 and is still widely used in 2023.

Figure 4 illustrates all the ransomware variants used in this research.

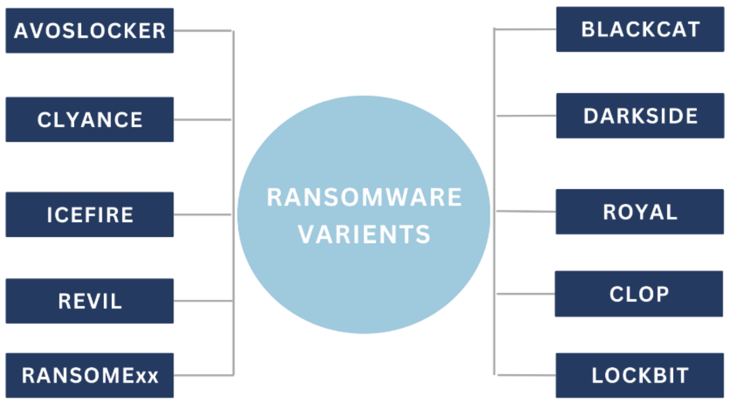


Figure 4 Variants of Linux Ransomware

2.5. Attack Chain of Linux Ransomware:

Attack chain is a process in which the attacker attacks a system. In other words, we can say the steps an attacker follows to attack the system. In figure-5 we have described the attack chain of Linux ransomware. It follows the steps given below:

Step 1: Attack finds vulnerability in the system.

Step 2: After finding vulnerability, he tries to gain access to the system.

Step 3: Once he is successful in gaining access, he starts installing ransomware

Step 4: After successful installation of ransomware, he tests if the encryption is working.

Step 5: Then he checks CMD line argument which is used to give path of files to encrypted.

Step 6: Then he starts searching whether files are already encrypted or not.

Step 7: If not, he drops the ransomware, and it encrypts all the files mentioned in the given path.

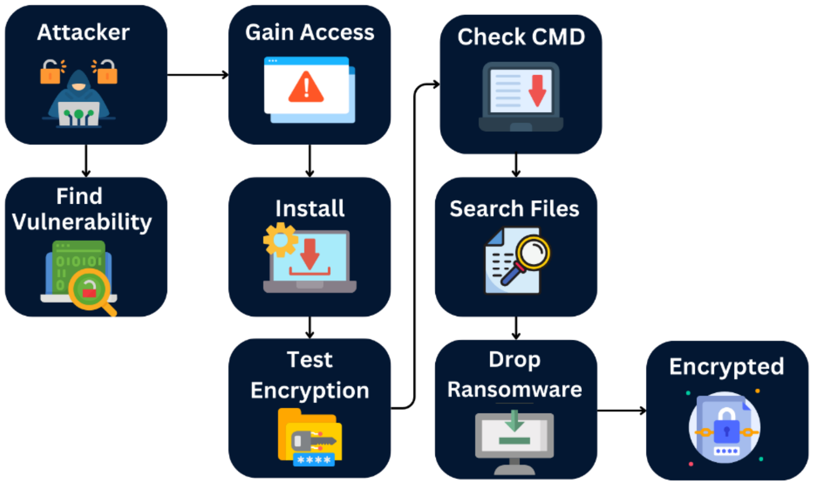


Figure 5 Ransomware Attack Chain

2.6. Chapter Summary:

In this chapter we discussed the existing literature on Linux malware and its detection systems. After that we presented the entire literature along with its limitations in tabular form. We have also discussed the increasing rate of attacks within Linux environment, Linux based ransomware and its different variants that we have covered in this research work. At the end we discussed the attack chain steps followed by Linux ransomware.

Chapter 3 Proposed Methodology

3.1. Introduction:

This chapter highlights the proposed methodology for detecting Linux based ransomware using machine learning. It describes the flow diagram of our framework, data collection, different analysis methods, feature extraction, steps for data preprocessing, importance of feature selection, different machine learning algorithms and evaluation metrics based on which we will decide our results.

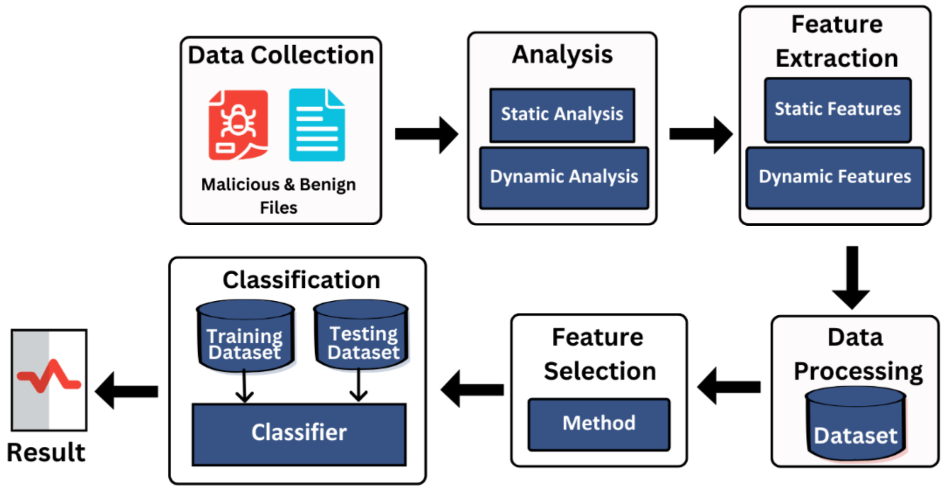


Figure 6 Proposed Methodology

3.2. Data Collection:

To create a dataset, we need to collect data samples of both malicious files as well as benign files. On these samples we will perform different methods to get useful information from these files and later it will help us in training the machine learning model. As we are doing our research on Linux malware, we need to collect ELF (Executable and Linkable Format) files. ELF [11] is a common standard file format for executable files, object code, shared libraries, and core dumps. The structure of ELF is shown in Figure 7.

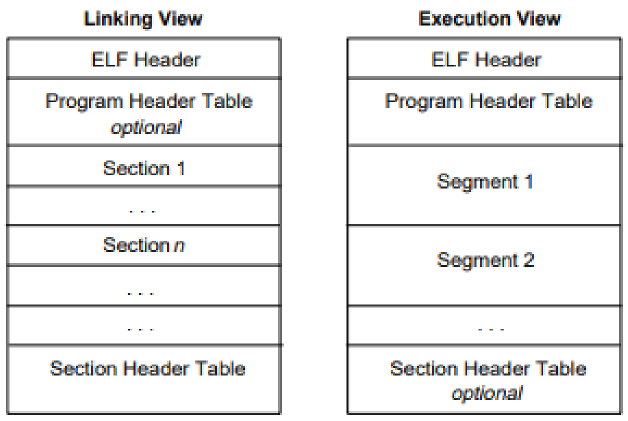


Figure 7 ELF Structure

3.3. Malware Analysis Methods:

To analyze the elf files, we need to perform different malware analysis techniques on our file samples to find the actual working of the file and whether it’s malicious or benign. There are three main types of malware analysis [12]:

- **Static Analysis**: static analysis is the examination of the code and structure of the malware without its execution. Static analysis includes the examination of the file’s metadata, headers, strings, and embedded resources. Static analysis helps to identify the signatures of known malware, understand the structural peculiarities of the malware, and reveal potential indicators of compromise without executing the code. A drawback of static analysis is that it is unable to detect behavior-based malware and threats, as well as polymorphic or obfuscated code.

- **Dynamic Analysis:** Dynamic analysis is the process of running malware in a confined environment and tracking what the malware does to learn more about its behavior. Behavioral analysis, code emulations, and memory analysis are all involved in dynamic analysis. The behavior of malware in activities includes file system changes, registry alterations, network activity, and process behavior may all be detected in real-time. However, not all malware behavior is detectable, and it may go undetected if the system does not have enough telemetry or if the malware is developed to avoid detection.

- **Hybrid Analysis:** Hybrid analysis combines both static and dynamic analysis to offer experts an in-depth perspective of malware. In this case, the approach integrates static analysis which helps to define and understand malware's structure and code, as well as dynamic analysis to research its behavior upon execution. From this perspective, hybrid analysis helps experts achieve better understanding of malware's attributes, behaviors, and impacts on a system. As such, it improves malware detection, analysis, and mitigation.

  In our work we will use Hybrid analysis, to get both static as well as dynamic features of the samples because now a days malwares are strong in functionality it gets hard to understand the actual working of a malware because of sophisticated techniques used by malware. Therefore, only by using static or only dynamic analysis is not sufficient to build the detection system.

3.4. Feature Extraction:

The extraction of features [13] is a critical step of developing effective machine learning models to detect malware. By converting raw data into a subset of relevant features, one will lower the dimension of the data while retaining the essential information for detection. Not only can this approach help improve the performance of machine learning models by directing models to only relevant and discriminatory features, but it can also improve the ability to generalize from the training on new malware samples never previously seen by the model. In addition, feature extraction improves the interpretability of machine learning models by enabling one to understand which information may be essential for detection. In simple words we can say feature extraction steps can result in more accurate, efficient, and interpretable detection models which can lead to improved cybersecurity outcomes. There are several types of features which you can extract from the analysis report. In our work we will use static as well as dynamic features.

3.5. Data Preprocessing:

Once we have extracted the features our next step is to clean the data. Machine learning models cannot understand data without processing it. We must clean our dataset using different techniques, so it gets easy for the model to learn it. Data processing [14] can be done in following sequence:

- **Data Cleaning:** In this step, we will check all the missing values in our dataset so we can perform the next steps.
- **Fill missing values:** First we will find all the NaN values then we will perform fill method to fill all those values.
- **Remove duplicated rows**: After filling missing values we will check if there are any duplicated rows. If there are duplicated rows, we will remove them.
- **Checking datatype:** Then we will check datatype of all the columns so that we can decide which encoding method we need to use
- **Performing encoding**: Once we find all datatype, we need to make them all float integer so we will use different encoding methods. Such as, one-hot encoding, label encoding and frequency encoding.
- **Normalize data:** by applying scaling methods we will then normalize the dataset.

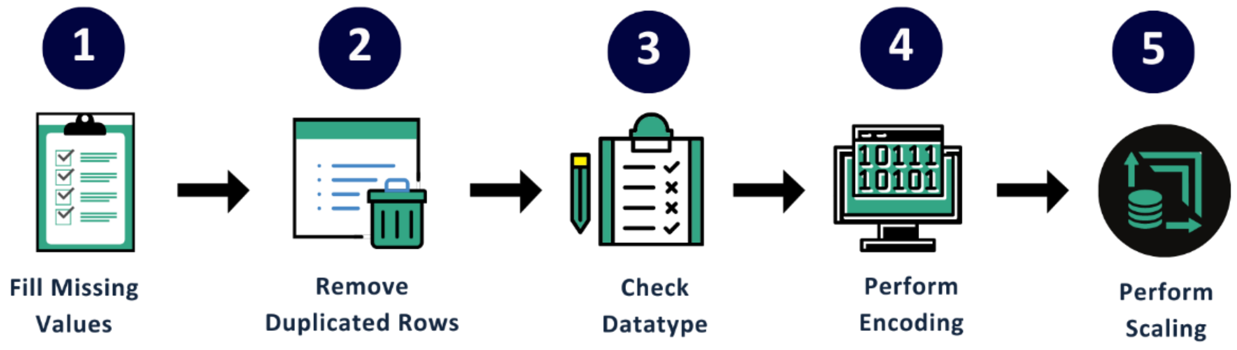


Figure 8 Data Preprocessing

3.6. Feature Selection:

The selection of relevant features is integral in the development of good machine learning models for malware detection. By choosing the most informative features, the coders can decrease the dimensionality of the data, which helps to develop better and more efficient code. Indeed, by selecting the features that are the most significant in developing the models, the coders contribute to better performance. It may reduce overfitting and assist in enhancing the ability to generalize. Furthermore, it may lead to a significant decline in the time necessary for the computation of predictive model development and use. It is especially vital for real-time malware defenses. Also, such a selection results in a greater interpretation of the model's results. The logical predictions are to be based only on the key features that the security workers can easily grasp. It may benefit in quicker identifying of the malware features and characteristics. Thus, choosing relevant features is critical in the development of successful and interpretable malware detection models.

In our work we are using we will use Information Gain. It is used for malware detection where we must identify discriminative features, the kind of features that help in classifying a malware file and any other benign file. Calculations of I.G help in understanding which feature is highly discriminative between the two. A higher information gain implies a better distinction. It is helpful in feature selection where many discriminative features will help in generating a more useful classifier model in malware detection.

3.7. Classification:

After performing all test steps, we need to decide which classifier we need to apply machine learning [15] on our dataset. Selecting classifier is a particularly crucial step, we must check our dataset depending on the data in the dataset and what type of work we want to get done by the classifier we select our classifier. Following are some classifiers that we have selected to test our research work:

- **Random Forest:** It is an ensemble learning method that builds a few decision trees at training time and reports the class which is the mode of the classes in the case of a classifier.
- **Decision Tree:** It is an easy to understand and interpretable model that splits the dataset recursively into subsets based on the highest-importance feature, resulting in a tree constructed structure.
- **Logistic Regression:** It is a linear model for a binary classification problem that models the probability of a default class using a logistic function.
- **FNN:** It is a neuro network model which is used to form networks.

3.8. Evaluation Metrices:

The metrics will help figure out whether the machine learning-based malware detection achieves optimal levels in detecting the malware and, at the same time, minimizing the false positives and false negatives. The evaluation metrics include:

- **Accuracy:** Referring to the proportion of samples correctly classified against the total samples.
- **Precision:** The proportion of true positives within the samples classified as positive.
- **Recall:** The proportion of true positive within the samples considered positive out of all positive samples.
- **F1-score:** It is the harmonic mean of precision and recall, thus being a balanced measure between precision and recall.

3.9.Chapter Summary:

This chapter presented a detailed proposed methodology which is used for the detection of Linux malware. Starting from collection of samples to malware analysis, data processing, feature extraction and selection. Training ML classifiers for detection the Linux based ransomware. All these modules are discussed in this chapter.

Chapter 4: Implementation and Experimental Results

4.1 Introduction:

In this chapter we will practically perform the things we have described in chapter three. Our proposed architecture includes data collection analysis module, feature extraction module, data preprocessing, feature selection module and classification module.

4.2 Experimental setup:

For performing malware analysis, we have use virtual machine on local system. Whereas for the creation of dataset we collected malicious samples from malware bazar. Tasks related to feature extraction, feature selection and training ML classifiers we have used Google Colab platform. The programming language that is used for performing all the tasks is python.

- **Hardware Configuration:**

Lenovo laptop was used for the implementation of this research with following aspects:

Table 2 Hardware configuration

Processor	Intel(R) Core (TM) i5-4210U
Memory	8.00 GB
Operating system	Ubuntu 20.04 LTS

- **Programming Language:**

- **Python:**

Guido van Rossum first developed Python programming language in 1991 and released it to the public in the same year. Consequently, writing code in Python is both easy and understandable due to white spaces. Python is generally an interpreted programming language that was designed for general purposes.

- **Google Colab:**

A free cloud service provided by Google called Google Colaboratory offers users a simple means for writing and running Python code through Jupyter notebooks. It consists of a web accessible Jupyter Notebook that allows for running of Python code without any local installations or maintenance. All the work done from feature extraction, feature selection, data processing, train of model all the work is done in google colab.

4.3 Collection of Dataset:

To create our dataset, we have collected malicious files and benign files. We have used malware bazaar [16] for malicious files and for benign files we have used files from different directories of Linux which includes **/user/ bin, /bin, /sbin.** We have collected samples of 10 types of Linux based ransomware. However, we were only able to find 70 samples of those types and 70 benign files. So, our dataset has a total of 140 samples.

Table 3 Dataset samples

Sr. No	File Type	Category	No. of samples
1	ELF	Malicious	70
2	ELF	Benign	70

4.4 Perform Malware Analysis:

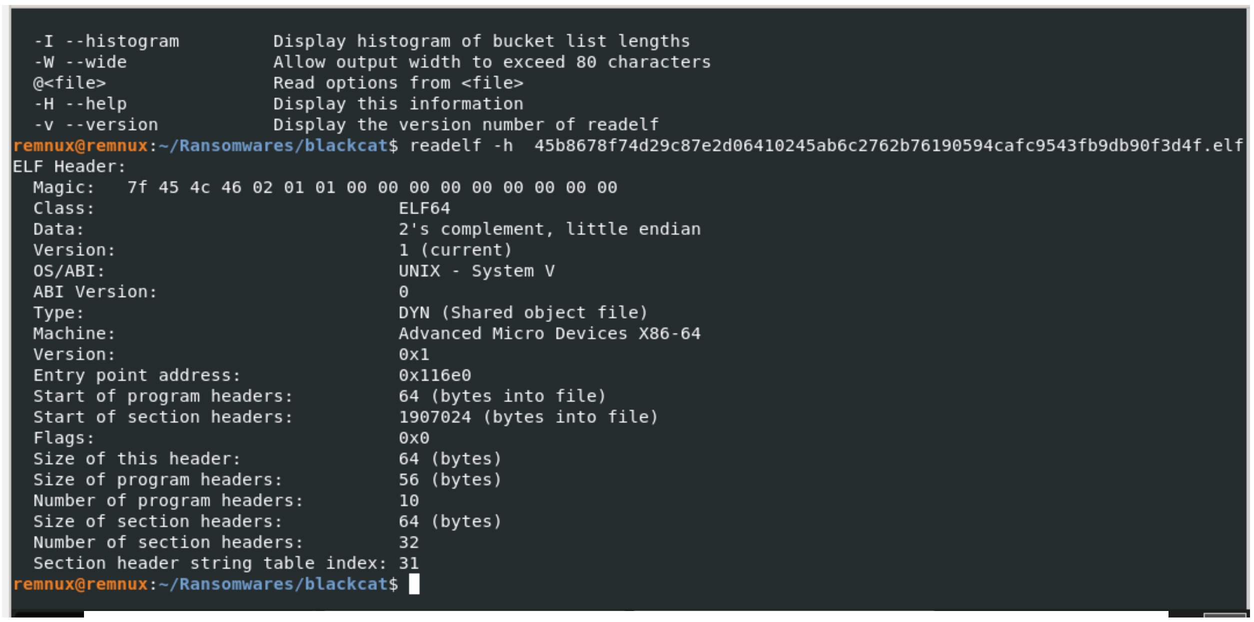
In our method we have used hybrid analysis so that we can use both static and dynamic useful features.

4.4.1 Static Analysis:

To perform static analysis, we have used REMnux Operating System. It is specially designed for malware analysis and reverse engineering.

REMnux: It is a Linux tool for figuring out and breaking down malignant programming[17]. Experts can use it to explore malware without finding, introducing, and arranging the devices. It has various apparatuses that can be used for both static and dynamic examination. On our dataset, we have utilized Readelf, Capa, and Peframe to extract useful static features from the files in our work.

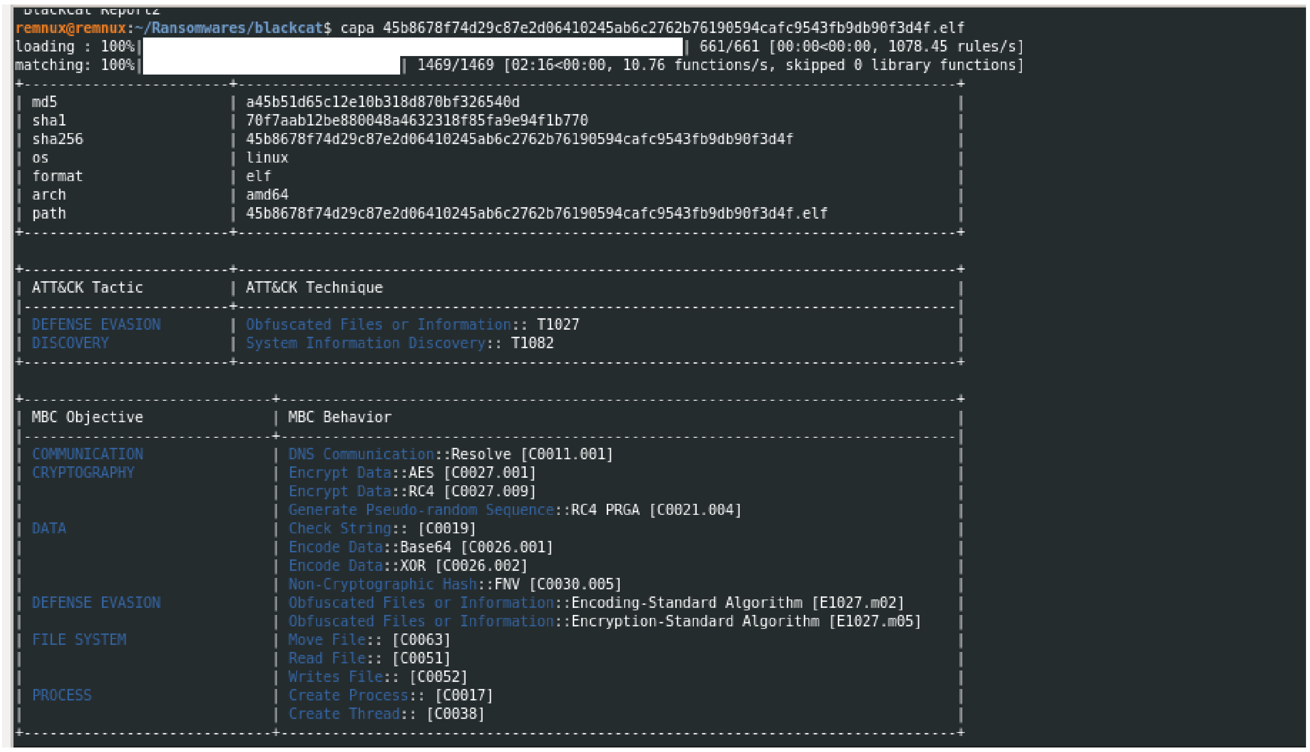
- **Readelf:** With Unix-like systems, readelf is a tool that shows different details about object files, much like obj dump. The GNU binutils include it.



```
  -I --histogram         Display histogram of bucket list lengths
  -W --wide              Allow output width to exceed 80 characters
  @<file>                Read options from <file>
  -H --help              Display this information
  -v --version           Display the version number of readelf
remnux@remnux:~/Ransomwares/blackcat$ readelf -h  45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f.elf
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x116e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          1907024 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         10
  Size of section headers:           64 (bytes)
  Number of section headers:         32
  Section header string table index: 31
remnux@remnux:~/Ransomwares/blackcat$ 
```

Figure 9 Readelf Output

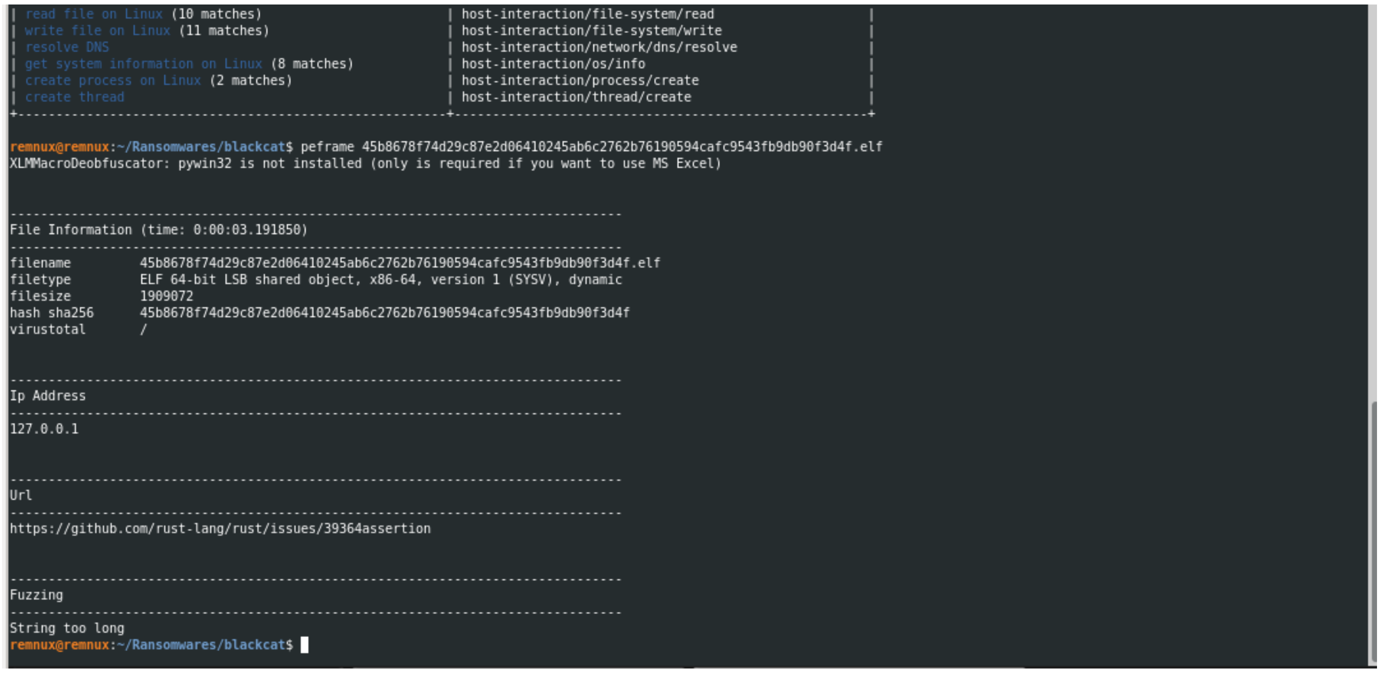
- **Capa:** It is an open-source program called Capa is used to analyze malicious programs. Capa offers a platform that the community can use to exchange, identify, and codify behaviors that we've observed in malware. Capa analyses executable files to find their capabilities.



```
remnux@remnux:~/Ransomwares/blackcat$ capa 45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f.elf
loading : 100%|                                                              | 661/661 [00:00<00:00, 1078.45 rules/s]
matching: 100%|                              | 1469/1469 [02:16<00:00, 10.76 functions/s, skipped 0 library functions]
+-------------------------+-----------------------------------------------------------------------------------+
| md5                     | a45b51d65c12e10b318d870bf326540d                                                  |
| sha1                    | 70f7aab12be880048a4632318f85fa9e94f1b770                                          |
| sha256                  | 45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f                  |
| os                      | linux                                                                             |
| format                  | elf                                                                               |
| arch                    | amd64                                                                             |
| path                    | 45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f.elf              |
+-------------------------+-----------------------------------------------------------------------------------+

+-------------------------+-----------------------------------------------------------------------------------+
| ATT&CK Tactic           | ATT&CK Technique                                                                  |
|-------------------------+-----------------------------------------------------------------------------------|
| DEFENSE EVASION         | Obfuscated Files or Information:: T1027                                           |
| DISCOVERY               | System Information Discovery:: T1082                                              |
+-------------------------+-----------------------------------------------------------------------------------+

+------------------------------+------------------------------------------------------------------------------+
| MBC Objective                | MBC Behavior                                                                 |
|------------------------------+------------------------------------------------------------------------------|
| COMMUNICATION                | DNS Communication::Resolve [C0011.001]                                       |
| CRYPTOGRAPHY                 | Encrypt Data::AES [C0027.001]                                                |
|                              | Encrypt Data::RC4 [C0027.009]                                                |
|                              | Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]                        |
| DATA                         | Check String:: [C0019]                                                       |
|                              | Encode Data::Base64 [C0026.001]                                              |
|                              | Encode Data::XOR [C0026.002]                                                 |
|                              | Non-Cryptographic Hash::FNV [C0030.005]                                      |
| DEFENSE EVASION              | Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]     |
|                              | Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]   |
| FILE SYSTEM                  | Move File:: [C0063]                                                          |
|                              | Read File:: [C0051]                                                          |
|                              | Writes File:: [C0052]                                                        |
| PROCESS                      | Create Process:: [C0017]                                                     |
|                              | Create Thread:: [C0038]                                                      |
+------------------------------+------------------------------------------------------------------------------+
```

Figure 10 Capa Output

- **Peframe:** It is an open-source utility designed to carry out static analysis on suspicious files and portable executable malware. Malware researchers may find information on suspicious files, suspicious sections and routines, packer, xor, digital signature, mutex, anti-debug, anti-virtual machine, and much more with its assistance.



```
| read file on Linux (10 matches)                        | host-interaction/file-system/read                      |
| write file on Linux (11 matches)                       | host-interaction/file-system/write                     |
| resolve DNS                                            | host-interaction/network/dns/resolve                   |
| get system information on Linux (8 matches)            | host-interaction/os/info                               |
| create process on Linux (2 matches)                    | host-interaction/process/create                        |
| create thread                                          | host-interaction/thread/create                         |
+--------------------------------------------------------+--------------------------------------------------------+

remnux@remnux:~/Ransomwares/blackcat$ peframe 45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f.elf
XLMMacroDeobfuscator: pywin32 is not installed (only is required if you want to use MS Excel)


--------------------------------------------------------------------------------
File Information (time: 0:00:03.191850)
--------------------------------------------------------------------------------
filename         45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f.elf
filetype         ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamic
filesize         1909072
hash sha256      45b8678f74d29c87e2d06410245ab6c2762b76190594cafc9543fb9db90f3d4f
virustotal       /


--------------------------------------------------------------------------------
Ip Address
--------------------------------------------------------------------------------
127.0.0.1


--------------------------------------------------------------------------------
Url
--------------------------------------------------------------------------------
https://github.com/rust-lang/rust/issues/39364assertion


--------------------------------------------------------------------------------
Fuzzing
--------------------------------------------------------------------------------
String too long
remnux@remnux:~/Ransomwares/blackcat$ 
```

Figure 11 Peframe Output

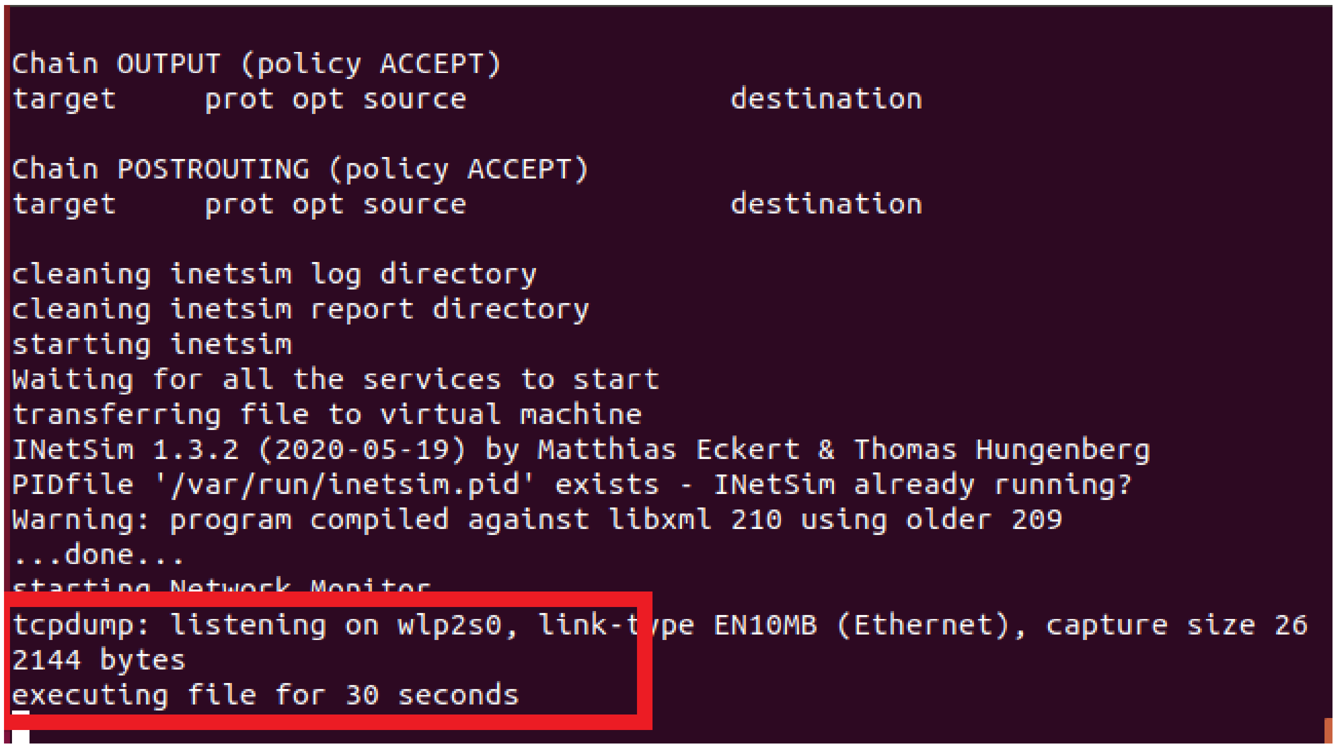
4.5 Dynamic Analysis:

To perform dynamic analysis, we have set up a sandbox to perform automated dynamic analysis. We have used Limon Sandbox.

4.5.1 Limon Sandbox:

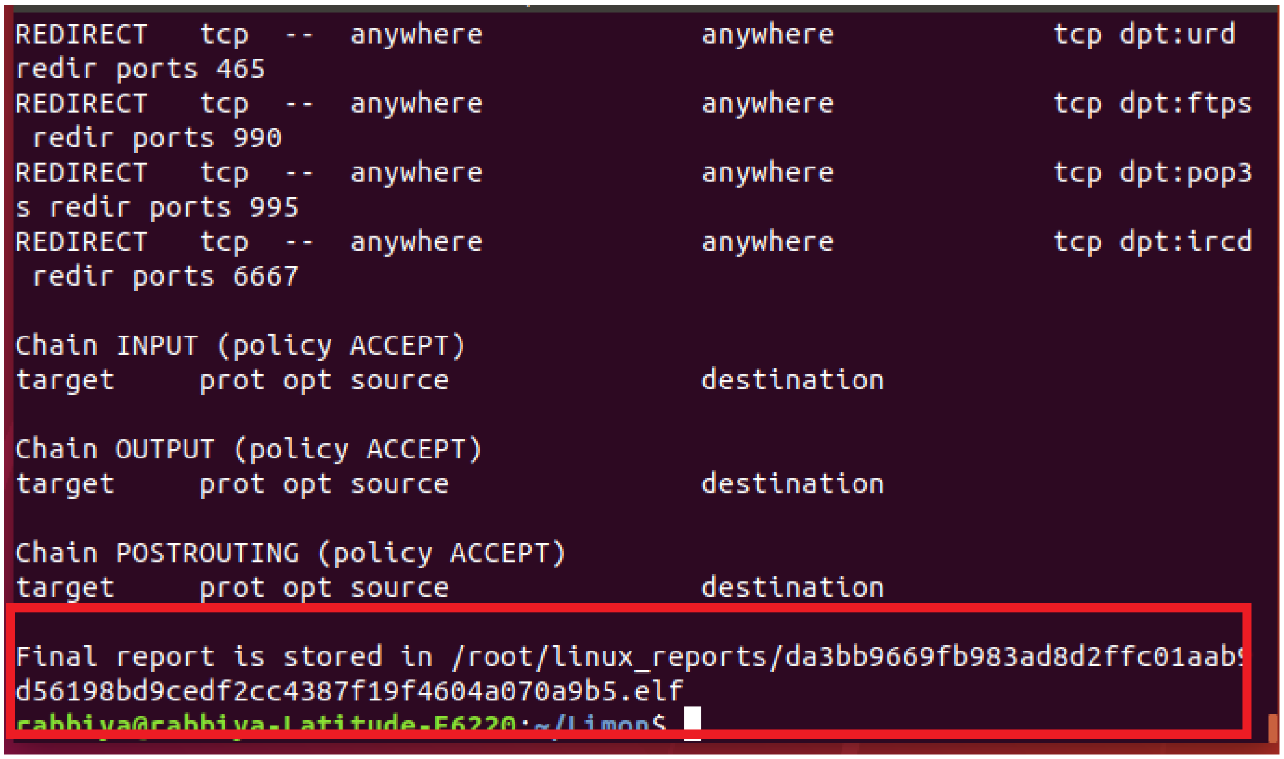
With the help of Limon[18], we may execute an executable in a controlled, sandboxed environment and receive a report on its runtime behavior.

of an executable. A host computer that controls the guest computer is part of the configuration for the limon sandbox. Ubuntu 18.04 was utilized in this study as the host and guest operating systems, respectively. The file runs in the guest machine's full privileged mode to provide a better understanding of a file. The path to the file in the Limon Sandbox is determined by doing a command line analysis of every file in a brand-new virtual computer. A current snapshot is taken during virtual machine setup so that limon can reverse after the le is executed. The entire trail of the network, system calls, and functions is saved in a text file called final report by the limon sandbox to the analysis report folder after execution is finished.



```
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

cleaning inetsim log directory
cleaning inetsim report directory
starting inetsim
Waiting for all the services to start
transferring file to virtual machine
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
PIDfile '/var/run/inetsim.pid' exists - INetSim already running?
Warning: program compiled against libxml 210 using older 209
...done...
starting Network Monitor
tcpdump: listening on wlp2s0, link-type EN10MB (Ethernet), capture size 26
2144 bytes
executing file for 30 seconds
```

Figure 12 Limon File Execution



```
REDIRECT   tcp  --  anywhere             anywhere             tcp dpt:urd
redir ports 465
REDIRECT   tcp  --  anywhere             anywhere             tcp dpt:ftps
 redir ports 990
REDIRECT   tcp  --  anywhere             anywhere             tcp dpt:pop3
s redir ports 995
REDIRECT   tcp  --  anywhere             anywhere             tcp dpt:ircd
 redir ports 6667

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination

Final report is stored in /root/linux_reports/da3bb9669fb983ad8d2ffc01aab
d56198bd9cedf2cc4387f19f4604a070a9b5.elf
rabbiya@rabbiya-Latitude-E6220:~/Limon$
```

Figure 13 Saving Dynamic Analysis Report

4.6 Extracting Features:

In this phase we extracted features from the data generated by different tools during the analysis. We have used Features from both static and dynamic analyses in this work which were good for the classifier explained in this section.

4.6.1 Static Features:

We have extracted static features from various fields of an ELF file. Most of the information is extracted from the ELF file header.

- ELF header
- File information

Following table contains static features:

Table 4:List of static features

Sr. No	Name	Datatype	Description
1	file_size	float	Size of file
2	Identification	obj	Special no to identify file
3	Class	String	Elf class
4	Version	int	Elf version
5	Os/ABI	String	Operating system
6	Machine	String	Machine name
7	ent_add	int	Entry point address
8	start_prog_header	int	Start of program headers
9	start_sec_header	int	Start of section headers
10	number_flag	int	No of Flag
11	size_header	int	Size of this header
12	size_prog_header	int	Size of program headers
13	num_ prog_header	int	Number of program headers
14	num_ sec_header	int	Number of section headers
15	sec_head_st_ind	int	Section header string table index
16	file_ent	float	Entropy of whole file content

4.6.2 Dynamic Features:

From dynamic analysis we have extracted:

4.6.2.1 System calls frequency:

After executing the files, we extracted all the system calls and then applied a python script to find the frequency of every system call.

Table 5:List of some System calls

Sr. No	Name	Description
1	access	check whether the **calling** program has **access** to a specified file
2	Bind	associates an address with the socket descriptor
3	Clone	creates a new process
4	Exce	to execute file
5	connect	**system call** connects the socket referred to by the file descriptor
6	chmod	modifies the access rights of the file

4.6.2.2 Encryption Method:

Another feature that we have used is encryption method, as we have analyzed Linux based ransomwares, encryption is main feature of a ransomware. Since ransomware usually encrypts files on the victim's machine, keeping an eye out for unusual encryption activity might be helpful in spotting possible ransomware attacks

Table 6: List of some Encryption features

Sr. No	Name	Datatype	Description
1	Encryption Algorithm	string	Type of encryption algorithm used
2	Key size	int	Size of encryption key

3	extension	string	Extension used by ransomware after encryption

4.6.2.3 Network Artifacts:

In ransomware attacks there is a high chance of network activity because the binary must connect to its C2C server after getting access of the system. So, we have extracted some features from network activity

Sr. No	**Name**	**Datatype**	**Description**
1	Total Packets	int	No. of packets sent and receive
2	DNS Queries	binary	Did query, yes or no
3	HTTP Requests	binary	Did Request, yes or no
4	HTTPS Traffic	int	Traffic captured
5	Unique Source IPs	int	Unique source Ip address
6	Unique Desti IPs	int	Unique destination Ip address
7	Protocol	string	Type of protocol used

4.7 Feature Selection:

For feature selection we have used the selection method: Information Gain [19]. It gives score to every feature based on the information that feature contains. Standard score ranges from 0 to 1. Features below 0 are considered useless. Features are calculated as follows:

$$\mathbf{IG(X) = H(Y) - H(Y|X)}$$

Where:

- **IG(X)** is the Information Gain of feature X.
- **H(Y)** is the entropy of the target variable Y before the split
- **H(Y|X)** is the conditional entropy of Y given feature X.

In the below graphs we can see the top 10 static and dynamic features selected by Information Gain.

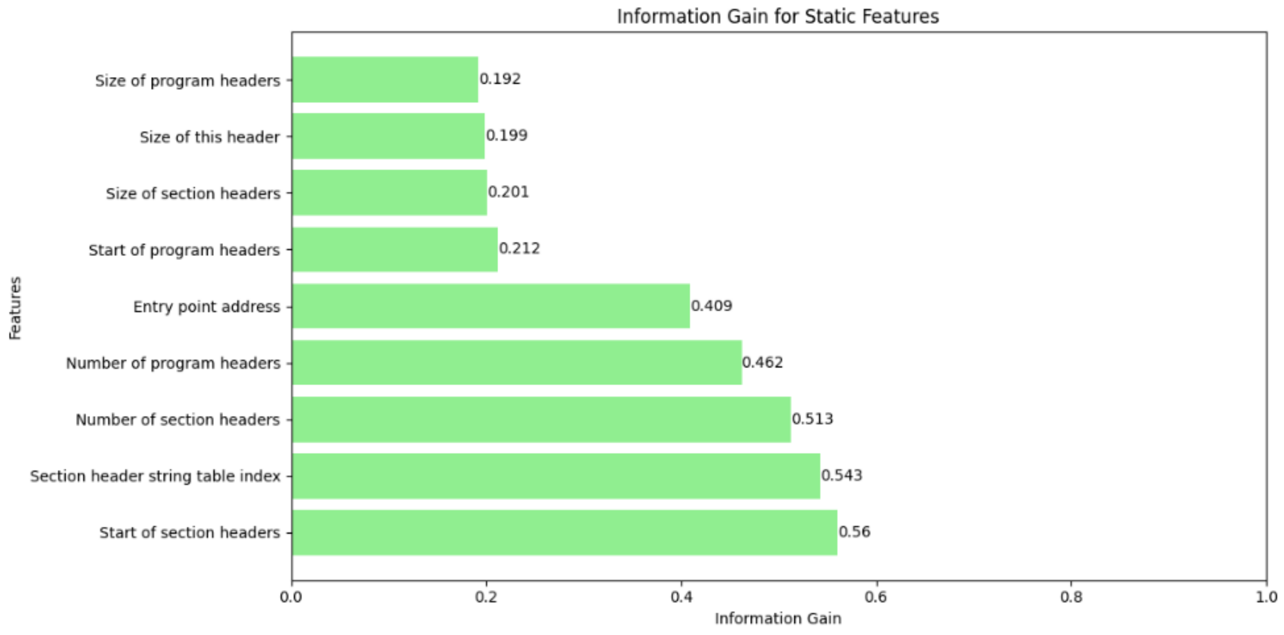


Figure 14 Top 10 static features

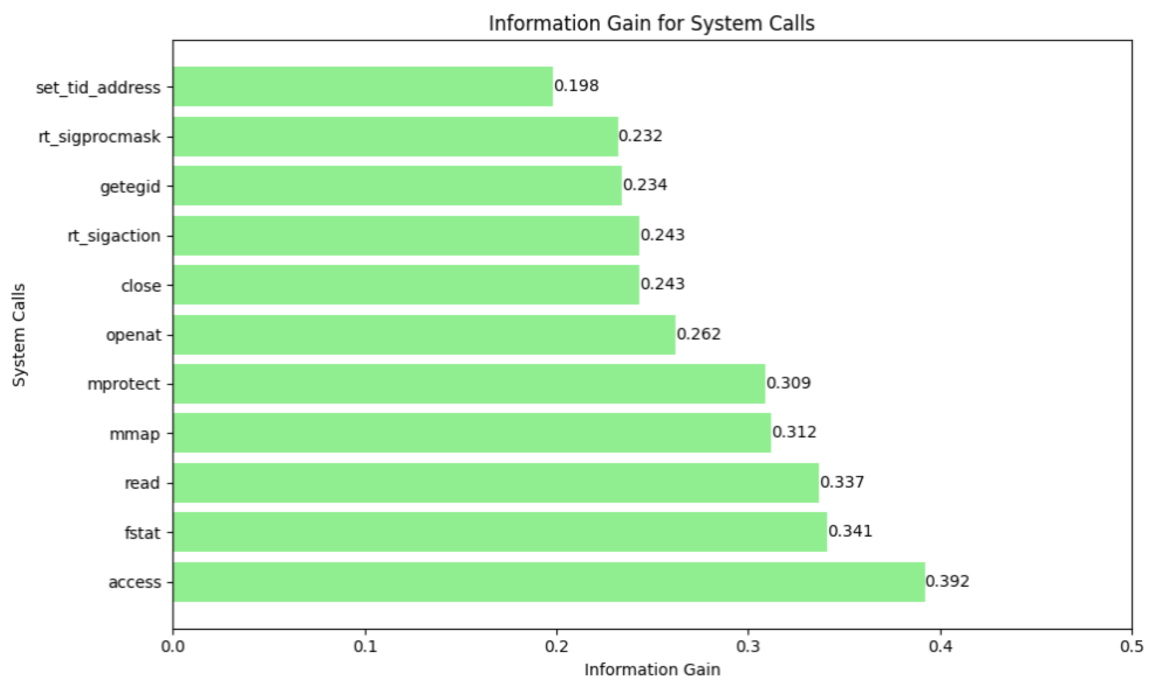


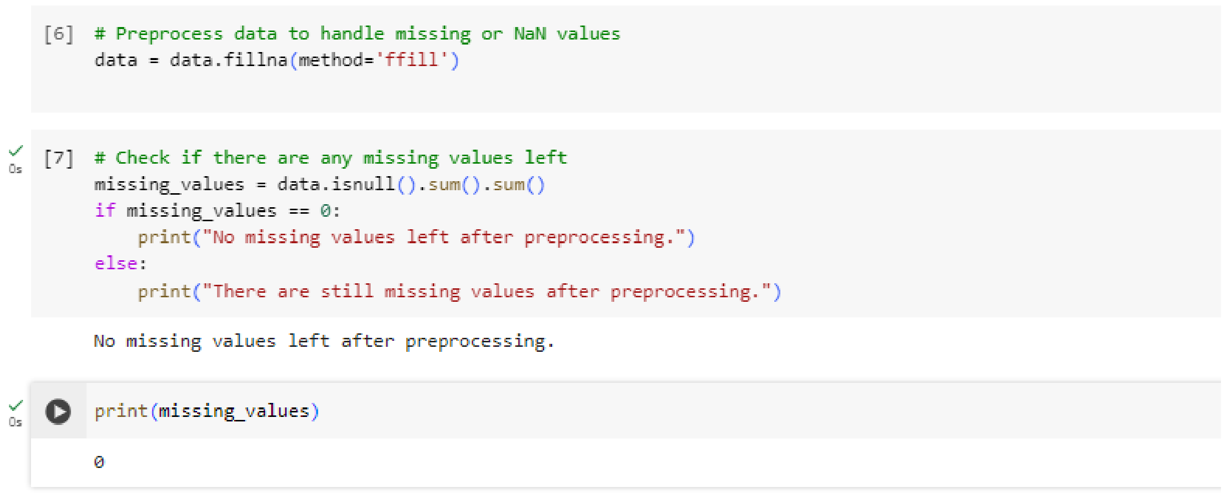
Figure 15 Top 10 Dynamic Features

4.8 Perform Data Processing:

. Data processing can be done in following sequence:

Step 1: Fill missing values:

First, we will find all the NaN values then we will perform fill method to fill all those values.

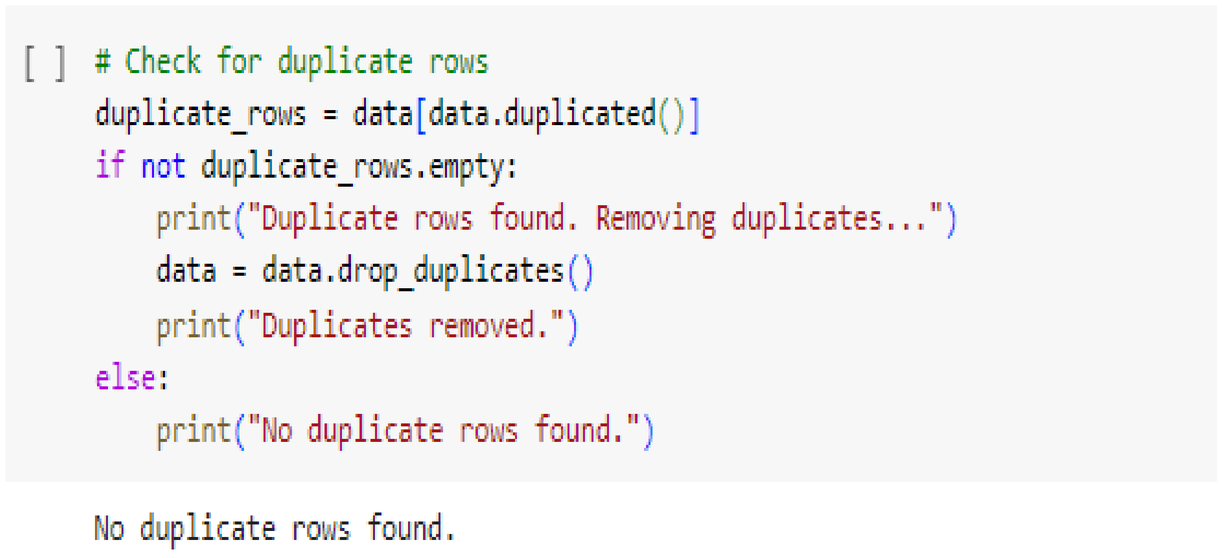


```
[6] # Preprocess data to handle missing or NaN values
    data = data.fillna(method='ffill')

[7] # Check if there are any missing values left
    missing_values = data.isnull().sum().sum()
    if missing_values == 0:
        print("No missing values left after preprocessing.")
    else:
        print("There are still missing values after preprocessing.")

    No missing values left after preprocessing.

    print(missing_values)

    0
```

Figure 16 Handle Missing Values

Step 2: Remove duplicated rows:

After filling missing values we will check if there any duplicated rows. If there are duplicated rows, we will remove them.



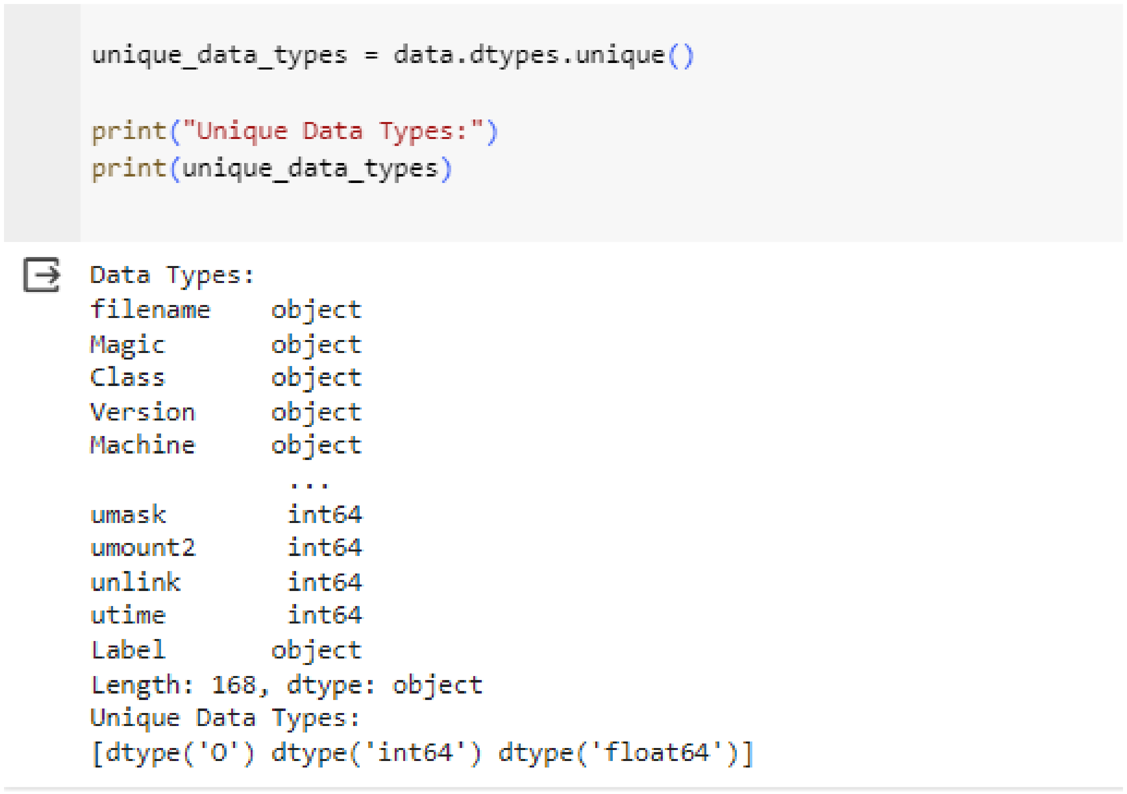
```
[ ] # Check for duplicate rows
    duplicate_rows = data[data.duplicated()]
    if not duplicate_rows.empty:
        print("Duplicate rows found. Removing duplicates...")
        data = data.drop_duplicates()
        print("Duplicates removed.")
    else:
        print("No duplicate rows found.")

    No duplicate rows found.
```

Figure 17 Check Duplicated Rows

Step 3: Checking datatype:

Next step is to check all the unique data types do we can perform encoding to make them all float datatype. In the figure below we can see we have 3 different data types:

- Object
- Float
- Integer



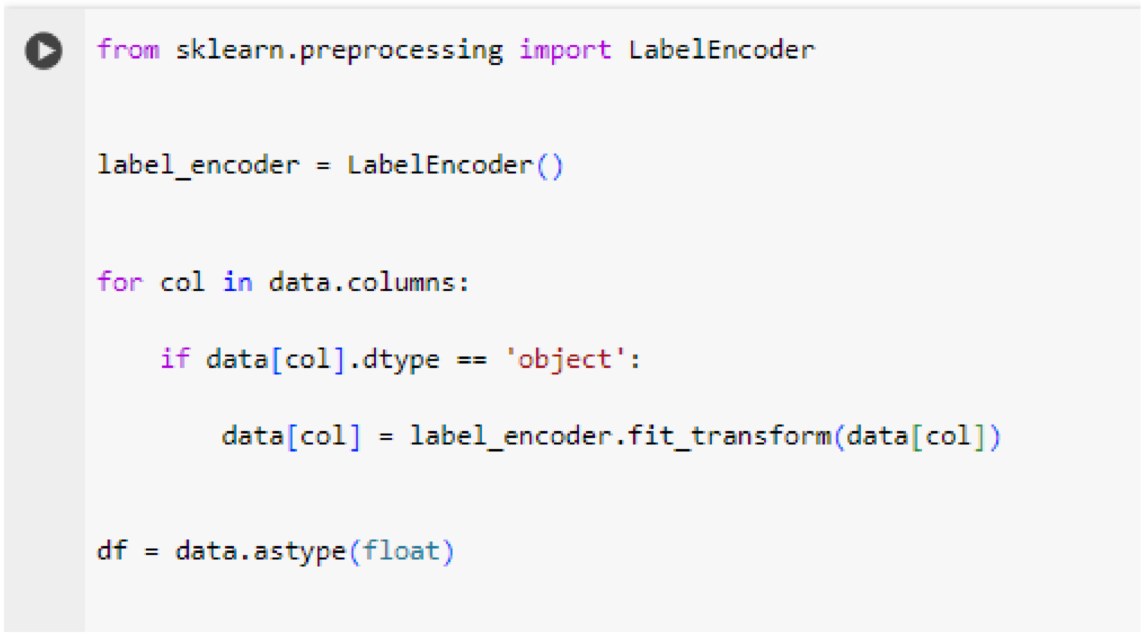
```
unique_data_types = data.dtypes.unique()

print("Unique Data Types:")
print(unique_data_types)
```

```
Data Types:
filename     object
Magic        object
Class        object
Version      object
Machine      object
              ...
umask         int64
umount2       int64
unlink        int64
utime         int64
Label        object
Length: 168, dtype: object
Unique Data Types:
[dtype('O') dtype('int64') dtype('float64')]
```

Figure 18 Check Datatype

Step 4: Performing encoding:

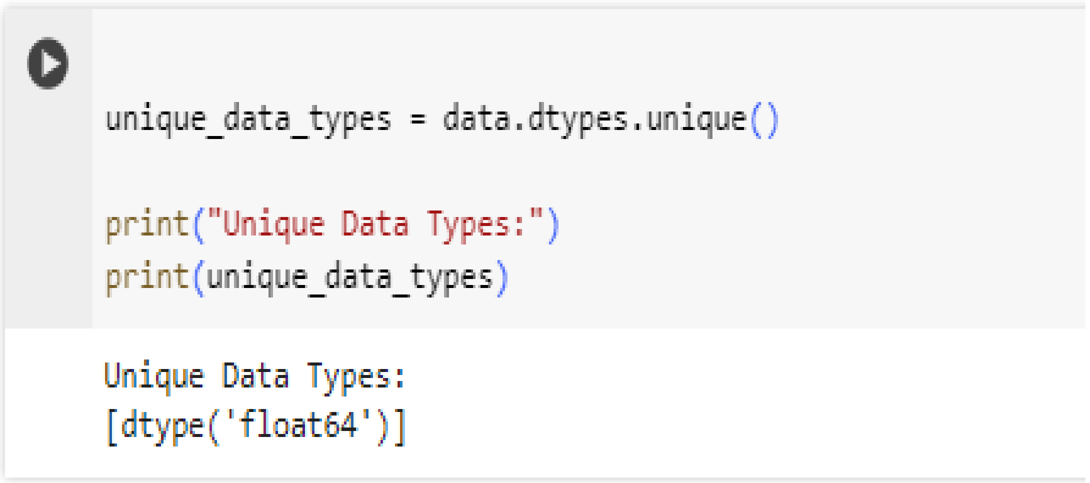
Once we find all datatype, we need to make them all float integer so we will use label encoding methods.



```python
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

for col in data.columns:
    if data[col].dtype == 'object':
        data[col] = label_encoder.fit_transform(data[col])

df = data.astype(float)
```

Figure 19 Before Label Encoding

After performing label encoding, we can see we have only one data type left which is float.



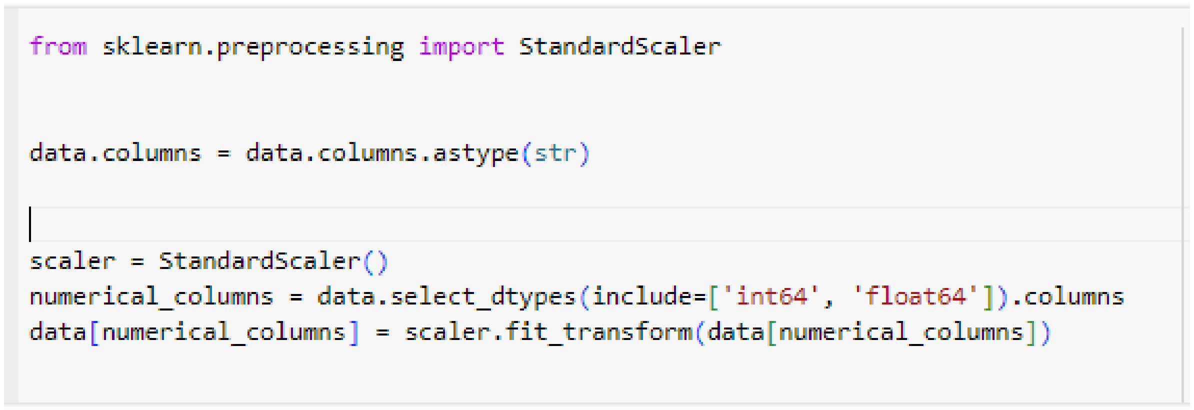
```python
unique_data_types = data.dtypes.unique()

print("Unique Data Types:")
print(unique_data_types)
```

```
Unique Data Types:
[dtype('float64')]
```

Figure 20 After Label Encoding

Step 5: Normalize data:

By applying the standard scaling method, we normalize the dataset.



```
from sklearn.preprocessing import StandardScaler


data.columns = data.columns.astype(str)


scaler = StandardScaler()
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

Figure 21 Standard Scaling

4.9 Model Selection:

4.9.1 J48:

J48, more popularly recognized as C4.5, is an ADT technique that is broadly employed in categorization systems. The following are the principal parameters of J48 algorithm:

- **Confidence Factor (CF):** Specify the confidence threshold to control tree pruning.
- **Minimum Number of Instances per Leaf (M):** Indicate the smallest number of examples required to divide a node.
- **Minimum Number of Instances per Split (L):** Specifies the minimum number of instances required to split a node.
- **Binary Splits (B):** Used to ascertain if binary divisions must be imposed divide a node
- **Subtree Raising (S):** It can be used to elevate useless sub trees and so improving tree performances

4.9.2 Random forest:

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It improves the performance and reduces overfitting compared to a single decision tree classifier. It has some parameters:

- ***Number of Trees (n_estimators):*** *The number of decision trees in the forest.*
- ***Maximum Depth of Trees (max_depth):*** *The maximum depth of each decision tree.*
- ***Minimum Number of Samples to Split a Node (min_samples_split):*** *The minimum number of samples required to split an internal node.*
- ***Min Number of Samples in Each Leaf Node (min_samples_leaf):*** *The minimum number of samples required to be at a leaf node.*

4.9.3 Logistic Regression:

The afore-mentioned logistic regression model works on several parameters. The following are some of the parameters:

- ***max_iter*** refers to the maximum number of iterations taken for the optimization algorithm.
- ***C*** refers to the inverse of regularization strength whereby smaller values are specified with higher regularization strengths.
- ***class_weight='balanced'*** helps to pass class weights as they are assumed to be inverse class frequencies. This parameter is used to solve the problem related to the weight imbalance.

4.9.4 FNN

The last classifier that we used is a neural network, specifically a feedforward neural network (also known as a multi-layer perceptron, MLP) which is implemented using TensorFlow's Kera's API. It is designed for binary classification tasks. The key parameters of this neural network classifier are:

- **Epochs:** Number of epochs for training.
- **Validation split:** 20% of the training data is used for validation.
- **Verbose:** Verbosity mode for training logs.

4.10 Chapter Summary:

This chapter gives a detailed approach that we have used to implement our proposed methodology. How we have created our own dataset. To perform malware analysis we have used both static and dynamic analysis. For static we have used REMnux and for dynamic analysis we have used Limon sandbox. After performing feature extraction, we have selected a list of static and dynamic features. Then we applied an Information gain feature selection method to find which features are useful. At the end we selected 4 ML classifiers and their parameters to train our dataset.

Chapter 5: Result and Discussion

5.1. Overview:

In this chapter we will describe evaluation metrices based on which we will measure the performance of all the classifiers. The performance is measured by using accuracy, F1-score, Recall and precision. At the end we will also perform a comparative analysis between the performance of all the 4 classifiers.

5.2. Evaluation Metrices:

The metrics [20] will help figure out whether the machine learning-based malware detection achieves optimal levels in detecting the malware and, at the same time, minimizing the false positive and false negatives.

Following are the metrics that we have used:

• **Accuracy:** The ratio of correctly predicted instances to the total instances in the dataset.

$$\text{Accuracy} = \frac{(\mathbf{TP + TN})}{(\mathbf{TP + TN - FP + FN})}$$

• **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations.

$$\text{Precision} = \frac{TP}{(\mathbf{TP + FP})}$$

• **Recall:** The ratio of correctly predicted positive observations to all actual positives.

$$\text{Recall} = \frac{TP}{(\mathbf{TN + FN})}$$

• **F1 Score:** The weighted average of Precision and Recall.

$$\text{F1 Score} = \mathbf{2}\,\frac{\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

Where:

TP = True Positives

TN = True Negatives

FP = False Positives

FN = False Negatives

5.3. Performance:

5.3.1. Performance of J48:

Precision:

- Precision for class 0 (94%) means that when the model predicts an instance as class 0, it is correct 94% of the time.
- Precision for class 1 (93%) indicates that the model is correct 93% of the time when it predicts an instance as class 1**.**

Recall:

- Recall for class 0 (94%) means that the model successfully identifies 94% of all actual class 0 instances.
- Recall for class 1 (93%) means that 93% of actual class 1 instances were correctly identified by the model.

F1-Score:

- F1-score for class 0 (94%) suggests a good balance between the precision and recall for class 0.

- F1-score for class 1 (93%) similarly indicates a good balance for class 1, though slightly lower than class 0.

Accuracy:

Overall accuracy of 94% indicates that the model correctly predicts the class (whether 0 or 1) for 94% of all the test instances.

Table 7 Accuracy Result of J48

Label	Precision	Recall	F1 score
0	0.94	0.94	0.94
1	0.93	0.93	0.93
Accuracy	0.95		

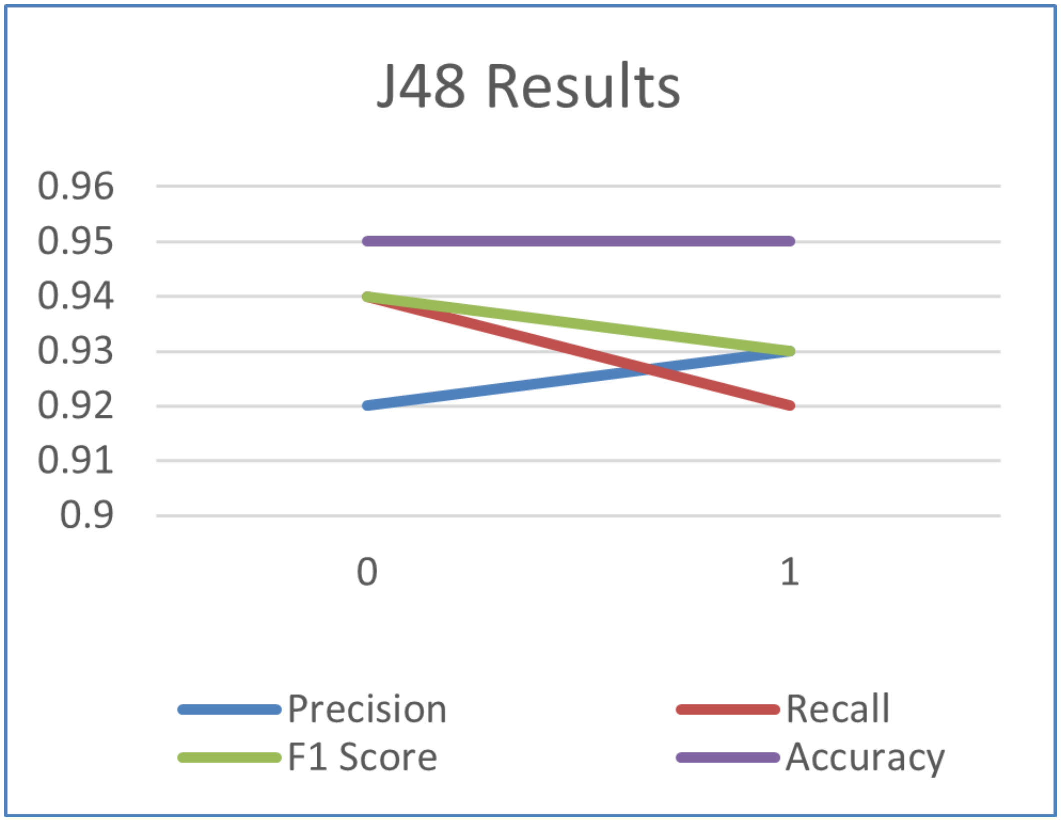


Figure 22 Performance Result of J48

5.3.2. Performance of Random Forest:

Precision:

- For class 1, a precision of 1.00 means that every instance predicted as positive was positive, which is excellent.
- For class 0, the precision is slightly lower at 0.89, but this is still quite good.

Recall:

- For class 0, a recall of 1.00 indicates perfect recall.
- For class 1, the recall is 0.87, which suggests that the model missed about 13% of actual positives for this class.

F1-Score:

- F1-score close to 1 is excellent, and here, both classes have high F1-scores (0.94 for class 0 and 0.93 for class 1), which suggests a good balance between precision and recall.

Accuracy:

- The overall accuracy of 0.9375 indicates that the model correctly predicted about 93.75% of the total instances. This is generally considered a high accuracy rate.

Table 8 Accuracy Result of RF

Label	Precision	Recall	F1 score
0	0.89	1.00	0.94
1	1.00	0.87	0.93
Accuracy	0.94		

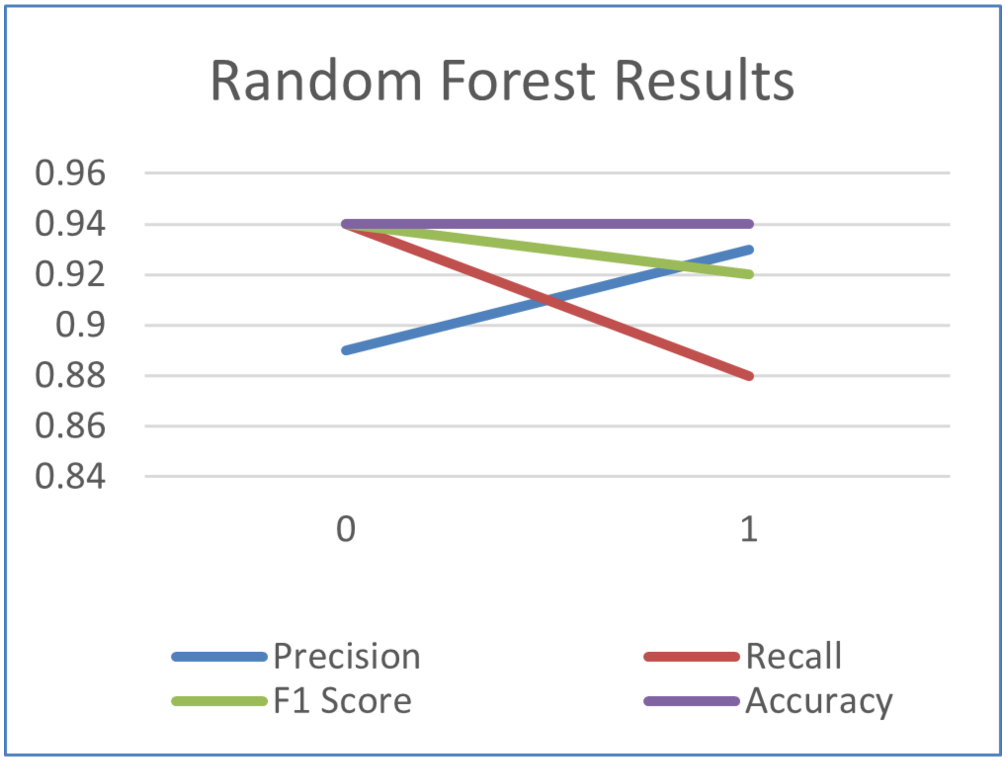


Figure 23 Performance Result of RF

5.3.3. Performance of Logistic Regression:

Precision:

- For class 0, the precision is 0.85. This means 85% of instances predicted as class 0 are indeed class 0, which is quite good though slightly lower than ideal.
- For class 1, the precision is 1.00, indicating perfect precision; all instances predicted as class 1 are truly class 1.

Recall:

- For class 0, the recall is 1.00, indicating perfect recall; all actual class 0 instances were predicted correctly.
- For class 1, the recall is 0.80. This means the model missed 20% of actual class 1 instances, which suggests room for improvement in identifying this class.

F1-Score:

- The F1-scores are 0.92 for class 0 and 0.89 for class 1. Both scores are high, suggesting a good balance between precision and recall, especially for class 0.

Accuracy:

- The overall accuracy of 0.90625 (or 90.625%) is quite strong. This indicates that the model correctly predicted the class for about 90.625% of the cases in the dataset.

Table 9 Accuracy Result of LR

Label	Precision	Recall	F1 score
0	0.85	1.00	0.92
1	1.00	0.80	0.89
Accuracy	0.91		

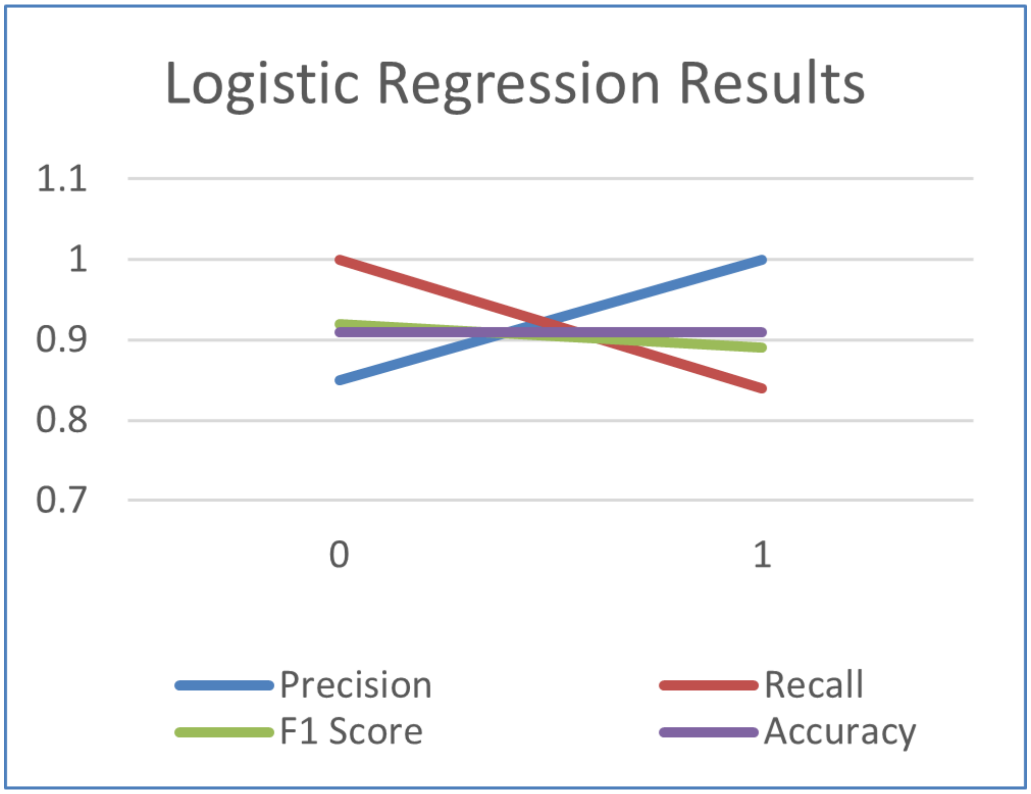


Figure 24 Performance Result of LR

5.3.4. Performance of FNN:

Precision:

- For class 0, the precision is 0.85. This means 85% of instances predicted as class 0 are indeed class 0, which is quite good though slightly lower than ideal.
- For class 1, the precision is 1.00, indicating perfect precision; all instances predicted as class 1 are truly class 1.

Recall:

- For class 0, the recall is 1.00, indicating perfect recall; all actual class 0 instances were predicted correctly.
- For class 1, the recall is 0.80. This means the model missed 20% of actual class 1 instances, which suggests room for improvement in identifying this class.

F1-Score:

- The F1-scores are 0.92 for class 0 and 0.89 for class 1. Both scores are high, suggesting a good balance between precision and recall, especially for class 0.

Accuracy:

- The overall accuracy of 0.97 (or 97%) is quite strong. This indicates that the model correctly predicted the class for about 90.625% of the cases in the dataset.

Table 10 Accuracy Result of FNN

Label	Precision	Recall	F1 score
0	0.80	1.00	0.92
1	1.00	0.80	0.89
Accuracy	0.97		

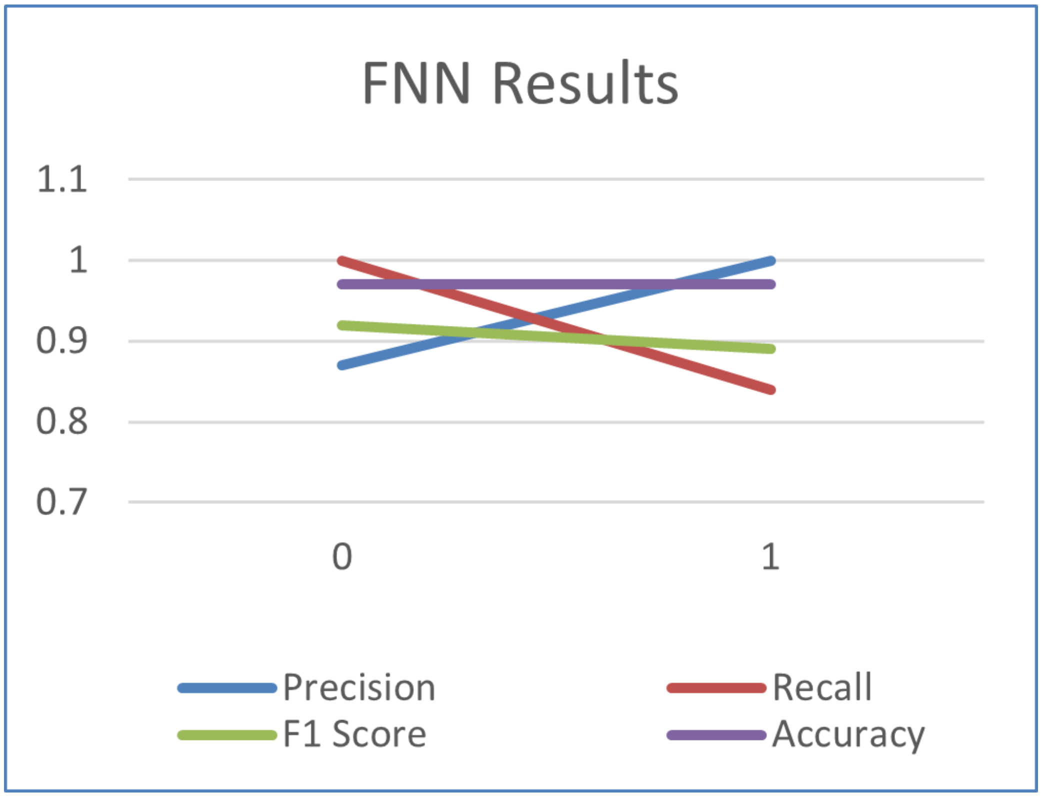
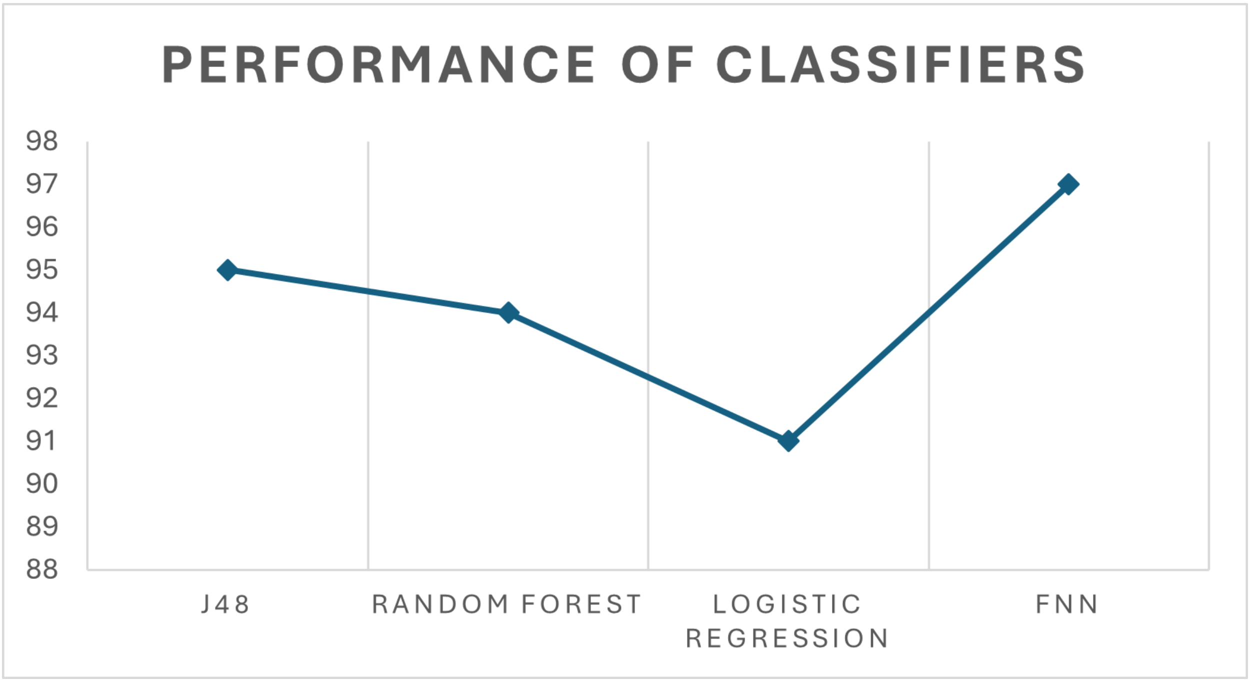


Figure 25 Performance Result of FNN

5.4. Performance comparison between classifiers:

Among all the 4 classifiers the best performance is FNN with 97% accuracy, after FNN classifier j48 has the best performance with 95% accuracy. Whereas RF has the 3rd highest accuracy, that is 94%. Logistic regression has 91% accuracy which is the lowest accuracy among all the classifiers. However, all the classifiers have accuracy above 90% which indicates that this research work presents a good detection system for Linux based malware.



5.5. Chapter Summary:

We have used 4 different parameters to measure the performance of classifiers. According to accuracy FNN has the highest score 97% followed by j48 which has 2nd highest score 95%. Random forest has the 3rd highest score 94%. Logistic regression has the lowest score among all the other classifiers, which is 91%.

Chapter 6: Conclusion and Future Recommendations

6.1. Overview

In this last chapter we recommended some different future work aspects that can be helpful for future researchers. Other than that, we sum up our research work and provided a conclusion of the contribution this research has made in the world of cyber security, all this is covered in this chapter.

6.2. Advanced feature selection and extraction techniques.

Further improvements in feature selection and extraction will be necessary to increase the efficiency of machine learning models in malware detection. The use of state-of-the-art algorithms such as deep learning, particularly CNNs and RNNs, may help to uncover the complex nonlinear patterns in the data that traditional models gloss over. Additionally, utilizing autoencoders for dimensionality reduction could optimize the feature set towards the most discriminative attributes, which would enable more accurate prediction and quicker processing.

6.3. Expansion to comprehensive malware detection frameworks.

The current study deals with ransomware only, a small fraction of the malware landscape that targets Linux systems. It is critical to create a broader malware detection framework that can recognize a variety of malware threats, including zero-day exploits, spyware, and rootkits. This will necessitate the development of a scalable and adaptable architecture capable of accommodating new malware signatures and behaviors, which can be accomplished via adaptive learning methods that update the models continuously without human intervention.

6.4. Real-time detection and response systems.

Implementing real-time detection and response systems is critical for reducing the consequences of malware on the infected systems. Future work might include integrating the developed machine learning models into existing IDS and IPS models for Linux, which would enable the automation of response measures such as quarantining the system, installing patches, and performing backups to avoid data loss.

6.5. Joint research and development:

The difficulty of contemporary cyber threats requires a collaborative effort in which diverse industries and disciplines work together. Future work will be essential, with collaboration with academic institutions, cybersecurity companies, and technology industry players providing access to a vast array of data and knowledge that can improve the research's robustness and practicality. This collaboration would also facilitate the development of uniform Linux malware data models that are critical in the training and testing machine learning custom tools.

6.6. Conclusion:

The thesis has successfully constructed and tested a machine learning framework for detecting Linux systems ransomware, and carefully evaluated its efficacy in comparison to prior detection methods. The merits of using a blended analytical approach incorporating static and dynamic analyses are proven to be effective, with high ransom accuracy rate and high recall rate and precision. This achievement is a significant advancement in cybersecurity, specifically on Linux systems which are less commonly researched in malware detection studies. In Conclusion, the thesis, regarding the research problem, contributes several factors:

6.7. Methodological Contributions:

This thesis has brought a distinctive method for malware detection that marries both static and dynamic analyses to strengthen the identification of various ransomware through the generation of an improved feature set that afforded to efficiently identify many ransomware types.

Machine Learning Contributions: The paper shows a positive comparison between different ML classifiers which include, Logistic Regression, Random Forest and Decision Tree's potential ability to address ransomware cybersecurity difficulties.

6.7.1. Practical Contributions: I also lay down the basis for generating operational tools or systems to protect and enforce systems in the Linux ecosystem.

6.7.2. Contributions to Cybersecurity: The method shortens the ransomware detection time and among the other benefits also retains the ransomware attack window at its worst, while the cost of ransomware hacking is increasing. Furthermore, the method is vulnerable to changes in techniques and tactics demonstrated by new ransomware, defending the method from change and needing another review.

6.8. Chapter Summary:

This chapter presents future recommendations for this research and the existing literature which includes advance feature selection and extraction methods, using big dataset for better accuracy, and creation of real time detection system. Other than this it gives conclusion of this research work which explains the summary of this research.

References

[1] F. F. Shahzad, "ELF-miner: using structural knowledge and data mining methods to detect new," *IEEE,* 2011.

[2] F. S. M. F. M. Shahzad, "In-execution dynamic malware analysis and detection by mining information in process control blocks of linux OS," 2015.

[3] R. K. C. S. K. S. Sanjay Sharma, "Detection of Advanced Malware by Machine Learning Techniques," *Springer,* 2020.

[4] N. K. a. A. Handa, "Detection of Advanced Linux Malware," Springer, India, 2020.

[5] K. V. P. Asmitha, "Linux malware detection using extended–symmetric uncertainty," *IEEE,* 2014.

[6] V. P. Asmitha K A, "A machine learning approach for linux malware detection," *International Congress on Information and Communication Technology,* 2014.

[7] C. R. K. S. S. Vaishali, "Detection of Malicious Executable in Linux Environment Using Tree-Based Classifer," *Springer,* 2021.

[8] J. Miller, "Linux Ransomware Poses Significant Threat to Critical Infrastructure," Dark Reading , 18 7 2023. [Online]. Available: https://www.darkreading.com/vulnerabilities-threats/linux-ransomware-poses-significant-threat-to-critical-infrastructure.

[9] "REMnux: A Linux Toolkit for Malware Analysis," [Online]. Available: https://remnux.org/.

[10] "Limon sandbox," [Online]. Available: https://github.com/monnappa22/Limon.

[11] "Dark reading," [Online]. Available: https://www.darkreading.com/vulnerabilities-threats/linux-ransomware-poses-significant-threat-to-critical-infrastructure.

[12] "T. I. Standard. Executable and Linking Format (ELF) Specification Version 1.1," 1993. [Online].

[13] M. a. P. R. C. a. o. Salvador and Galar, "Data level preprocessing methods," *Learning from Imbalanced Data Sets,* 2018.

[14] B. a. T. A. S. a. o. Azhagusundari, "Feature selection based on information gain," *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 2, pp. 18--21, 2013.

[15] D. a. A. L. a. B. R. Ucci, "Survey of machine learning techniques for malware analysis," *Elsiver Computers & Security,* 2019.

[16] V. a. others, "Classification model evaluation metrics," *International Journal of Advanced Computer Science and Applications,* vol. 12, pp. 599--606, 2021.

[17] "Malware Bazar," [Online]. Available: https://bazaar.abuse.ch/. [Accessed 12 September 2023].

[18] D.-S. a. M. S.-D. Huang, "Linear and nonlinear feedforward neural network classifiers: a comprehensive understanding," *Journal of Intelligent Systems,* vol. 9, pp. 1-38, 1995.

[19] I. Riadi, "Implementation of malware analysis using static and dynamic analysis method," *International Journal of Computer Applications,* 2015.

[20] C.-T. a. W. N.-J. a. X. H. a. E. C. Lin, "Feature selection and extraction for malware classification.," *J. Inf. Sci. Eng.,* vol. 21, 2015.