Detection of rootkits using memory analysis

By

Basirah Noor

Fall-2019-MS-IS 00000318515 SEECS

Supervisor

Dr Sana Qadir

Department of Computing

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

In

School of Electrical Engineering & Computer Science (SEECS) ,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(June 2023)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Detection of rootkits using memory analysis" written by BASIRAH NOOR, (Registration No 00000318515), of SEECS has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: 

Name of Advisor: Dr. Sana Qadir

Date: 15-Jun-2023

HoD/Associate Dean: _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Approval

It is certified that the contents and form of the thesis entitled "Detection of rootkits using memory analysis" submitted by BASIRAH NOOR have been found satisfactory for the requirement of the degree

Advisor : Dr. Sana Qadir

Signature: 

Date: 15-Jun-2023

Committee Member 1:Dr. Hasan Tahir

Signature: 

15-Jun-2023

Committee Member 2:Dr. Razi Arshad

Signature: 

Date: 16-Jun-2023

Signature: _____

Date: _____

Dedication

With the name of Allah the most beneficent, I dedicate this research to my parents and family, who have made so many sacrifices for me and motivate me during my whole journey. I am also grateful to my supervisor Dr Sana Qadir who guided me through out the process. I dedicate this research to her also.

Certificate of Originality

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

Student Name: BASIRAH NOOR

Student Signature: _____

Acknowledgement

First of all, I am very grateful to Allah S.W.T for granting me the ability to conclude this research work. After that, I want to thank my supervisor, Dr. Sana Qadir, who guided me throughout the process and shown her confidence in me and constant support till the end. There is nothing which can payback for her efforts throughout my research period to complete it successfully. I am also grateful to Dr. Hasan Tahir and Dr. Razi Arshad for their valuable remarks and observations as my Guidance and Evaluation Committee. Finally, I also want to express my gratitude to my parents and family who encouraged me throughout the journey. and supported me during my research.

Contents

List of Figures

List of Tables

Abstract

Rootkits are malicious software designed to hide their presence and activities on compromised systems. Traditional detection methods often struggle to identify rootkits due to their ability to mimic normal files and evade detection. However, volatile memory analysis has emerged as a powerful technique for monitoring system activities. In this thesis, we propose the use of memory analysis combined with machine learning and deep learning to develop an effective rootkit detection model. By analyzing the contents of the system's volatile memory, our model aims to identify suspicious patterns and behaviors that indicate the presence of rootkits. We employ machine learning and deep learning approach to train the model on a comprehensive dataset of known rootkit samples, enabling it to learn and recognize the distinct characteristics associated with these stealthy malwares. Through extensive experiments and evaluations, we assess the performance and accuracy of our model in detecting various types of rootkits. The results demonstrate the effectiveness of memory analysis combined with machine learning/deep learning in rootkit detection, offering a promising solution to combat the ever-evolving threat of these elusive malware. This research contributes to the development of advanced defense mechanisms and enhances the security posture of systems against rootkit attacks.

Chapter 1

Introduction

Rootkit is a malicious program that exist with extremely insidious nature. It is designed in such a way to hide its existence in the system and provide an unauthorized root access to the attacker. The attacker would have a complete control over the system. The heinous nature of these programs to evade detection, make it difficult for the detection tools to know when the rootkits have in filtered the system.

Once it gets installed on the system, alien behaviors can be observed that shows that some remote attacker has got an access to the system. These types of malwares are extremely dangerous as the existence of the attack cannot be judged by traditional detection tools and techniques, as a result can cause significant data loss and damage to the system. As these malicious programs require the root level or administrative level privileges, people under red hats typically exploit the vulnerabilities of the targeted system to gain access.

1.1 What are rootkits?

The term rootkit is derived from two words "root" and kit". In UNIX and LINUX environments, "root" refers to administrator who has the highest level of access to the system. While the "kit" refers to some collection of tools and techniques. Therefore, a rootkit can be more precisely defined as a set of tools or techniques that allow any hacker to attain administrator level access and maintain it throughout the attack while still remained undetected by the authorized users and administrators [3].

As discussed previously the main aim of the rootkits is to provide a full control of the

system to a remote attacker over some server or host. The first step is to install a rootkit on a system to obtain a root level access and in some cases only a user-level access. This can be achieved by exploiting the vulnerabilities of the system.

The next thing is to stay undetected by the authorized users and administrators by hiding themselves under the legitimate programs. Rootkits may consist of several utilities including the ability conceal the running processes, hide directories and change the system calls.

1.1.1 Rootkit Technologies

Various technologies can be used by the rootkits to achieve its goal to remain un-detected and stealthy. Some of them are discussed below.

- **User-mode hooks:** Rootkits use user-mode hooks to redirect or modify the system calls to hide its presence in the system.
- **Kernel-level hooks:** Rootkits use kernel-mode hooks to conceal processes and files and monitors keenly the system activity while remain hidden hence can alter the behavior of the Operating System.
- **Firmware Rootkits:** These rootkits infect the firmware of the device, For example BIOS etc.
- **Direct Kernel Object Manipulation (DKOM):** This is a method used by rootkits to directly manipulate with the kernel level objects [5].
- **Hypervisor rootkits:** These rootkits target the virtual environments running on a system and hence hide themselves from the host operating system and covertly perform malicious activities.
- **Bootkits:** These types of rootkits gain control of a target system by infecting its master boot record (MBR). In this way these can execute the malicious activities even before the operating system loads [4].

1.1.2 Rootkit Infection

Hosts or servers can be compromised by the rootkits using various methods like backdoor mechanism, exploiting vulnerabilities or social engineering tricks. Once the system gets

compromised the rootkits can be used by the hackers to conceal their presence.

Following way can be studied as how can Rootkits get injected and perform malicious activities on targeted server/host.

- ***Stage 1:*** **Investigating victim's system for vulnerability:** The first stage of the rootkit attack is to gather information about the targeted host or server. This aims to identify any existing vulnerabilities on the target system or network that can be exploited.
- ***Stage 2:*** **Attack and compromise the server:** Once the information has been gathered about the vulnerabilities, the next step is to gain access and compromise the targeted system. Once access is obtained, the attacker will escalate privileges.
- ***Stage 3:*** **Rootkit installation:** The next step after gaining access is to install the rootkit on the targeted system. As the rootkit hides its identity, it covers the tracks followed by the hacker and creates a backdoor to maintain access to the compromised system in the future.
- ***Stage 3:*** **Controlling the victim's system:** After gaining access and installing the rootkit, the attacker would have full control over the targeted system. In addition, all their activities will be hidden. The hacker can use the compromised system to attack other systems connected to the targeted one, creating a DDOS effect. The DDOS effect is one grave consequence of rootkit attacks. There are many others too. The attacker can use them to gain access to valuable assets.

1.1.3 Rootkit Detection

The researchers have devised many detection techniques but have associated limitations with them. Some traditional detection techniques are as under [6]:

- **Signature-based rootkit detection:** It is the most traditional method in which the known signatures or patterns are checked to detect the rootkits. As malwares are becoming more stronger, this technique is not so effective in these days and hence do not provide sufficient true positives.

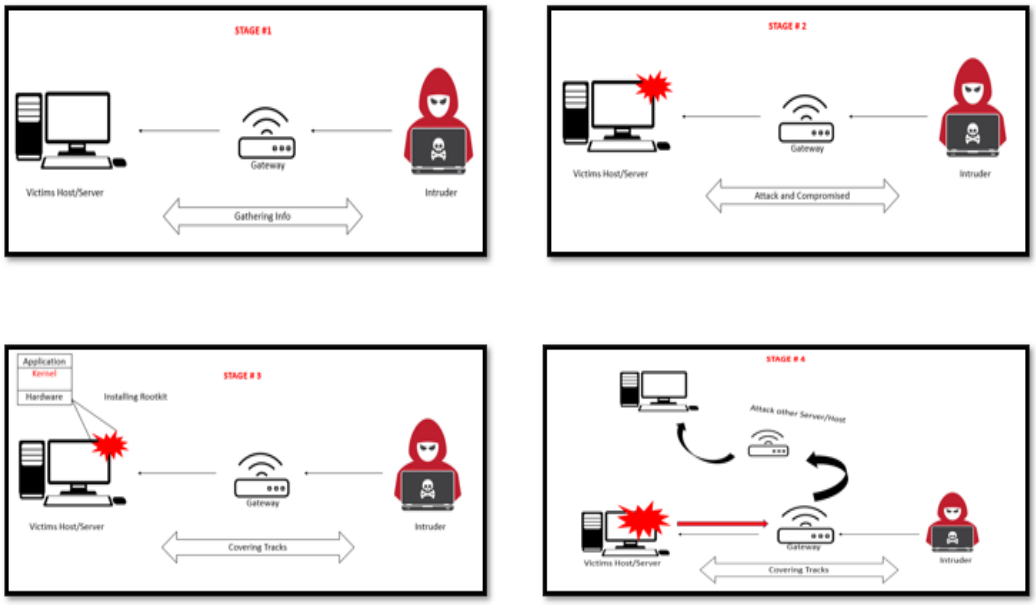


Figure 1.1: Rootkit Infection Process

- **Behavior based rootkit detection:** This technique analyses the behavior of the system to detect any rootkit. The behavior is compared with the clean system to report the results. Again, the technique is not much useful because rootkits have a property to hide themselves, and their behavior sometimes look legitimate.
- **Integrity checking:** This technique involves a comparison of windows registry or key files of a clean system with the targeted system.
- **Difference based detection:** This technique involves a comparison of the binaries with their RAM counterparts to identify if they are identical or not. If not, then it can be a sign of rootkit existence.
- **Memory analysis:** This process involves the analysis off the captured memory dumps. As the rootkits reside inside the memory, so this can be most efficient method.

The ongoing research revolves around the memory analysis technique because every other technique struggles to detect the rootkit because of its hiding property.

1.2 Problem Statement

As discussed above, rootkits hide themselves to be detected by the administrator and users. They operate covertly on a compromised system and evade detection as a result they can cause serious security threats. Traditional detection methods sometimes are insufficient to detect these deadly malwares. Therefore, there is a need to device a standard mechanism that can detect these malwares efficiently. This thesis aims to explore the efficiency of memory analysis technique to detect the rootkits. Also, we are intended to develop an automated tool by using machine learning algorithms and deep learning to easily and accurately identify the presence of rootkits in a system.

1.3 Research Objectives

In order to address the limitations of the existing work, the research aims to meet following objectives.

- To propose an efficient technique based on the memory analysis to detect the hidden rootkits.
- To develop algorithms and tools to automate the process of analyzing the memory dumps and detect the rootkits
- To study the new set of features extracted from memory images, such as DLLs, handles, privileges, network connections, modules, injections and services
- To propose a method that also address the limitations involve during the feature extraction for rootkits detection.

1.4 Thesis Motivation

Since late 1980s, rootkits have become a great concern for the security engineers. At that time these were used to conceal log files and application binaries. The first generation of these malicious softwares only targeted the user-level programs. These infections were relatively easy to detect as simplest checksum method could be used to achieve the purpose. However, now a day, threats posed by recent rootkits are even greater as

operating system is being used by many devices like smartphones, IoT nodes, computers and other embedded devices [1] [2]. As a result, many researchers are working to design efficient method to detect the modern rootkits that can overcome the traditional detection methods.

The rootkit attacks are also increasing day by day. According to reports by Avast (a leading cybersecurity company), in 2020, the number of users that got affected by rootkits was about 10,000 which has been increased to 100,000 in 2021. Hence, the purpose of this research is to device a reliable and efficient detection technique for the rootkits that are difficult to detect by addressing the major limitations in the existing literature.

1.5 Thesis Organization

The thesis is organized in the following way.

Chapter 2 of the thesis will present the literature review on the techniques utilized for the detection of rootkits, the algorithms and limitations that hinders the efficient detection of rootkits.

In chapter 3, the research methodology will be discussed. This includes the discussion on tools and techniques, installation, process such as data preprocessing, feature selection/extraction and model training. The detail about the data acquisition, preprocessing and feature extraction phases are discussed in chapter 4.

Chapter 5, whereas, discuss the model evaluation. This chapter will also provide the validation process. Results will be presented and discussed in the same chapter.

Lastly, Chapter 6 provides the complete summary of the research and suggests some future work that can be carried out. In a nutshell, this thesis aims to provide a mechanism to detect the rootkits based on the memory analysis, that can also address the issues in existing work.

Chapter 2

Literature Review

This chapter illustrates the existing research on the detection of Rootkits, thoroughly review them and will indicate the potential research gaps. For the purpose, we have gathered a wide range of conference proceedings, books, peer-reviewed articles and journal papers. The result summary of this chapter will help to serve as a foundation for the ongoing thesis, and proposed research method.

2.1 Rootkit and its Detection Techniques

The privacy and security features of any network experience a significant threat of Rootkits [7]. They provide a backdoor route to the hacker to penetrate in the operating system, hence allow an unauthorized access. The malware writers putting their continuous efforts to evolve the rootkits technology, making it even more difficult to detect or remove them from the infected system. Therefore, it is difficult for the users to protect their systems from such attacks.

Despite the growing threats as posed by the rootkits, the literature shows a little effective detection methods and tools. The reason probably due to the hiding technology of rootkits and complexity of developing the effective detection procedures.

2.1.1 Rootkit Distribution Methods:

The primary objective of the people who spread rootkits is to acquire data from the targeted system and cyberespionage. About 77 of rootkit attacks involve only data acquisition [8]. Sometimes the motive of the attackers are others, for example, to achieve

financial gains and exploit the asset of victims.

Distribution of rootkits is mostly done by social engineering methods, like phishing, Apps that looks similar to legitimate sites, fake websites etc [9],[10],[11]. Whereas the motivation remained the financial gains and data acquisition, precisely cookies and credentials.

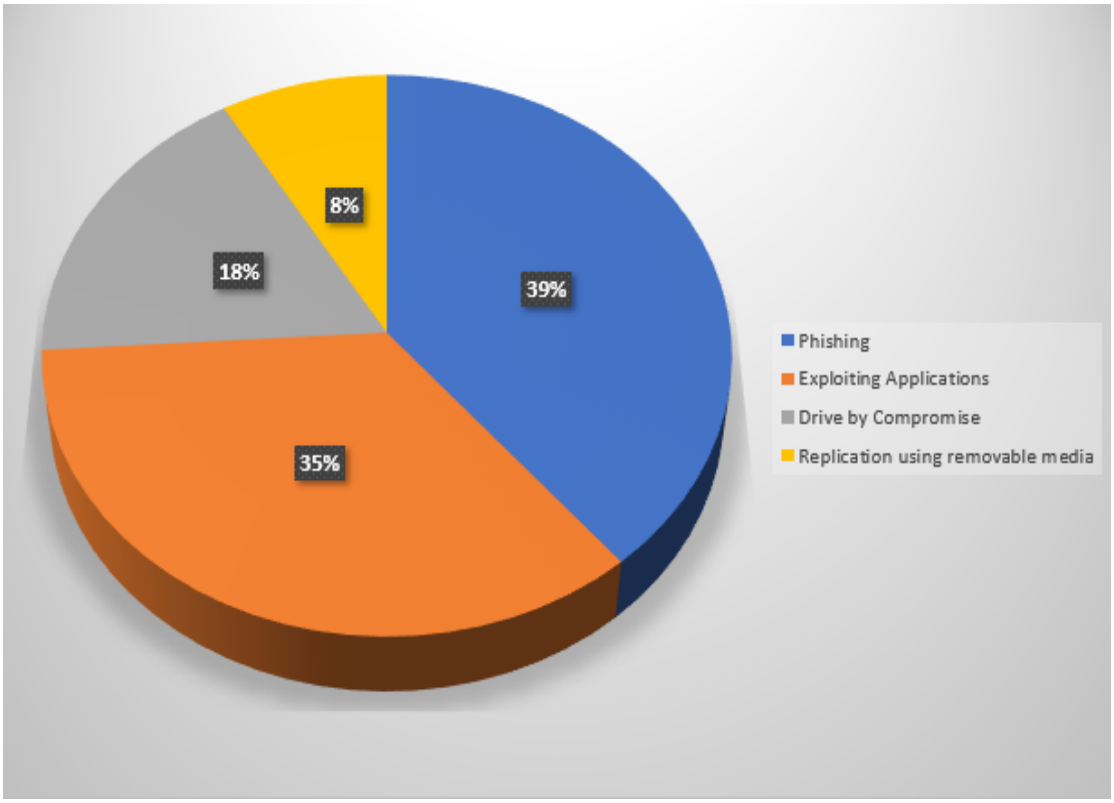


Figure 2.1: Rootkit Distribution Methods

2.1.2 Comparison of Rootkit Detection Techniques

Since the creation of the first rootkit (NT Rootkit) researchers are trying to device better detection techniques for them. Virtualization bases detection techniques have been proposed by many researchers but they show many limitations at end user level, like detecting emulation environment too as a rootkit, special hardware requirements etc [13], [14]. Signature based are the most common ones. Other approaches are behavior based, memory analysis and learning based detection. Table 2.1 gives us a comparison of rootkit detection approaches.

Table 2.1: Comparison of Rootkit Detection Approaches

Detection Approach	**Strengths**	**Weaknesses**
Signature-Based	Easiest way to detect; Cost-effective solution; Doesn't require to load the dump file to analyze	Easy to remain hidden by these softwares; Difficult to detect DKOM attacks; Performance overhead
Behavior-Based	Doesn't require any prior knowledge about the rootkits; Reduced performance overhead; Behavior of malicious body can be emulated	It will not work effectively when the rootkit has compromised many devices in a row; Difficult to detect DKOM attacks; Requires the source code of OS kernel to identify the attack; Doesn't work well with the hidden behavior of rootkits
Memory Analysis	Can detect any anomaly or unauthorized access to the targeted system; Can overcome the hidden behavior of the rootkits	Time inefficient; Lacks automation; Less accuracy
Learning-Based	Rootkits leave fingerprints on program memory accesses; Detection is performed separately	Significant performance overhead; There can be a delay

2.2 Machine Learning

Machine learning is evolved within two fields of knowledge, computer science and artificial intelligence (AI), according to IBM it revolves around the utilization of algorithms and data to simulate the learning process of humans without their intervention to improve the accuracy over time [15].

Machine learning basically helps to solve two main problems, regression and classification. Regression usually involves continuous real value problems, such as integers or floating numbers that represent quantities such as sizes or amounts, whereas classification deals with the problems where the predicted variable consists of only discrete categories or labels. Its model is assigned to test given input with specific category or label.

As the rootkits detection uses a binary classification method, where it needs to classify the data into either malicious/rootkit or benign. Whereas if we classify rootkits into sub categories, then we need to tackle this problem using a multi-class classification by assigning rootkit families to distinct categories.

Therefore, the machine learning plays a vital role in the detection of rootkits. Now a days many researchers are proposing various solutions to detect rootkits and other malwares using the properties of machine learning. The field machine learning is mainly categorized into three sub-categories: supervised learning, unsupervised learning and reinforced learning [16].

The supervised learning deals with the training of the data that are labeled. This means that the result is the probability between the known categories. Whereas, the unsupervised learning deals with training of the data that is not labeled. In comparison to supervised learning, the unsupervised requires relatively larger dataset to suggest efficient results. The process of machine learning involves many steps which include data acquisition, data preprocessing/ feature engineering, feature selection, training model and evaluation. The detailed process will be discussed in the upcoming chapters for detecting the rootkits.

In the proposed solution, we have utilized the supervised learning and deep learning for the detection of rootkits with following listed algorithms.

2.2.1 Random Forest

The first algorithm that had been used is random forest (RF). It is widely been used as a supervised learning algorithm. It helps in various classification and regression problems by building ensemble of decision trees thru selecting the training data. For classification problems, the result is generated by the majority decision while for regression, the result is generated by using average mechanism.

We selected Random Forest because of many advantages it poses. The foremost one is that it addresses the over fitting issue by combining values from different trees. Hence reduce the impact of biases created by individual tree. As we increase the number of trees, the accuracy of the algorithm also gets better [17]. Also, Random Forest can easily handle both the continuous data and categorical data, hence making the algorithm adaptable for different types of data sets [18].

Random Forest is also a powerful tool for rootkit detection. It can efficiently detect the potential rootkit activities by analyzing various features and characteristics of the system dump. As said above, Random Forest is a useful tool that can handle both types of datasets, categorical and continuous, making the algorithm very feasible to detect as well as classify the rootkits.

There are many articles in the existing literature that used the random forest for the malware detection (rootkits are type of a malware). Li et al. used Random Forest for the rootkit detection by extracting different features from its kernel [19]. Similarly, C. Felan et al. for detecting unknown malwares based on the system calls and they achieved high detection rate [20].

Therefore, Random Forest is a very reliable machine learning algorithm that we can use for the rootkit detection. Its property to deal with different types of datasets, and solving problems by creating multiple trees make it well suited for the job.

2.2.2 Support Vector Machine

Support vector machine (SVM) is another algorithm based on the supervised machine learning category. It can be used for classification purposes and also for the rootkit detection. However, rootkits pose significant challenges for SVM to perform classification with greater accuracy [21].

Many researchers demonstrated how to improve the performance of the algorithm in existing literature. Many studies also presented various techniques and methods to increase its accuracy to efficiently detect rootkits [22]. Even the algorithm has been used for the rootkit detection, the primary objective remains same, which is to identify the optimal decision boundary, it facilitates the classification process on the given data [23]. In this way the rootkit positive entries can easily be separated from the benign files.

Specialized SVM techniques are another option that can help to address the challenges that SVM face in case of the rootkit detection. For example, two specialized SVM techniques are introduced in the literature [24], multiple instance learning and ensemble based SVM model to improve the detection rate and robustness. Also, by integrating the deep learning algorithms with SVM, more better results can be obtained.

In a nut shell, the Support vector machine is a good choice to detect the rootkits because of its classification properties. The existing literature focus on the ways to increase the performance of the SVM, hence the methods can be used in the ongoing research to facilitate rootkit detection.

2.2.3 Decision Tree

Another supervised machine learning technique that is quite popular among the researchers to detect the malwares is Decision tree. The algorithm can perform both the classification and regression. It forms tree to predict the results and is suitable for many applications. It can also help us to detect the rootkits.

Decision tree is a considerate option to tackle rootkit detection by leveraging the rules represented as the branches of the decision tree and corresponding results as the leaf nodes. Many researchers have used this algorithm for malware detection and also for the rootkits. For example, in [22], the authors have also used decision tree to detect the kernel rootkits.

In a structural form, it is a flow chart type structure in which internal nodes are test values, branches are the results of the tests and leaf node would be a final result. The root node contains the entire dataset. At each internal node, a rule for specific feature is been evaluated, using which branches are created leading to child nodes in different paths. The child nodes after many recursive steps gives a final decision typically when

certain depth is reached.

The algorithm has many advantages, for example these can be easily visualized and interpreted. Also, the algorithm can be performed on both types of datasets categorical as well as numerical. However, the analyst usually faces overfitting issues while using the decision tree algorithm.

2.2.4 K-Nearest Neighbor

K-Nearest Neighbors also knows as K-NN is a supervised machine learning algorithm used for both the classification problems and regression. In the testing procedure, it gets votes from its k nearest neighbors to classify the values/instances. K-NN compares the new instance during the testing phase with the data stored and based on the similarities and differences with the existing data, it classifies the new instance [26]. The algorithm is also known as a lazy learner because of its property that it does not immediately learn the pattern but compare the new instances with the stored data based on similarities.

For Rootkit detection, K-NN can also be explored. K-NN can be trained on the dataset that has both the rootkit instances and benign instances. Relevant features as modules, handles, APIs, system calls, network behaviors etc are extracted from the given dataset during the training phase and get labeled as benign or malign. Whereas during the testing phase, the K-NN calculates the similarity of the new instance with the already tested instances. K-NN poses many benefits to the Rootkit detection from hyper parameter tuning to feature selection and feature engineering, thereby improve the accuracy rate.

2.3 Deep Learning

Deep learning is known as a sub field of machine learning that focuses on the Artificial intelligence, which enables models to learn hierarchical data representations. Instead of depending on manually created features, it seeks to automatically learn hierarchical features or representations from raw input data. Deep learning has received a lot of attention and has had great success across several industries, including speech recognition, natural language processing, and computer vision.

2.3.1 Long and Short Term Memory

Long Short-Term Memory (LSTM) is a significant deep learning technique. It is a type of a Recurrent Neural Network (RNN) that is used to capture the sequential nature of the system data make it useful for the rootkit detection. LSTM can learn the anomalies and patterns by leveraging the sequential information that may help to detect the rootkits.

Just like classification algorithms, dataset for LSTM also requires to train a part of the dataset. The training data can consist of many features that are been considered or recorded. Each instance in the training data is labeled as either normal or infected. During this phase of the training, the LSTM learns the sequential information and patterns in the training data while in the testing phase, it processes the sequential data associated with each instance. Also, it compares the temporal dependencies and patterns with the trained dataset. Based on the trained dataset, LSTM can tell either the instance is malign or benign.

The efficiency of the LSTM depends on the quality of the training data when the concern is detecting the rootkits or other malwares. The techniques like the cross validation and hyper parameter tuning can be applied to get more optimized and efficient results.

2.4 Related Work

Many researchers have studied to detect the malwares and rootkits utilizing the memory analysis. Some studies are discussed below.

2.4.1 Studies based on Memory Analysis for Malware Detection

This section will throw a light on the top articles, where the researchers used memory analysis process to detect the malwares.

The memory forensics to detect and classify the malwares presents a promising approach to overcome the limitations from the basic static and behavioral analysis. Sihwail et al. studied the effectiveness of extracting the memory-based images to reveal information that can help to detect the malwares [27]. The analysis was done using the volatility v2.6. The accuracy rate they achieved is 98.5. they also created a memory-based dataset in binary form and made it available to the researchers on GitHub [28]. They

have also used classification algorithms like SVM, Random Forest, K-NN, Naïve Bayes and Decision Tree.

Bozkir et al. tried to detect the malware by analyzing the memory dumps as RDB images [29]. Their study combined the computer vision and machine learning to detect and classify the malwares. The detection of unknown malwares is improved by the process they used up till 20.78 % across the multiple machine learning algorithms. They used Random Forest, SVM and XGBoost algorithms for the purpose. The study highlights the practicality of the computer vision-based scheme for protection against the malicious applications.

The research carried out by [30] also studies the importance of memory analysis to capture the footprints of malware and extract the hidden code from the obfuscated malwares. The authors have written a python-based plugin for the volatility v2.6 and named it as VolMemLyzer, that can direct extract 36 features and convert the results into csv format. The plugin provides high accuracy to classify the malwares too. A dataset has been created using the too of about 1900 instances.

The detection of obfuscated and hidden malware creates many challenges in the field of memory analysis. Carrier et al. researched to transform the VolMemLyzer framework already studied in [30] so that it can extract 26 new memory features making it more efficient [31]. The plugin can be used to detect the malwares (Ransomware, Trojan Horse and Spyware). They also utilized the classification algorithms and demonstrates high efficiency in detecting obfuscated and hidden malware with an accuracy of 99% and F1-Score as 99.02. An extended dataset has also been created that is a very positive contribution for the researchers in this field. Joseph et al. studied the importance of memory analysis to address the issues created by the ransomware [32]. They proposed a memory analysis-based approach to detect the presence of ransomwares. The approach integrated the user-defined tools with search tools like YARA to increase the accuracy.

2.4.2 Studies based on Rootkit Detection

This section will throw a light on the top articles where the researchers tried to detect Rootkits using various methods including the memory analysis.

The first research will detect the kernel rootkits using their defined approach named as TKRD (trusted kernel rootkit detection) [33]. TKRD combines the memory foren-

sic analysis with bio-inspired machine learning technology. The training features are extracted from volatile memory dumps using the Volatility framework. The extracted features include modules, threads, drivers, IRP and SSDT hooks, callbacks, and timers. The experimental results show that the process gives a high accuracy in detecting the windows kernel rootkits.

The researchers in [24] used dynamic analysis to detect the rootkits. A hardware-assisted virtualization-based kernel-level rootkit detection (VKRD) system that is trained by machine learning techniques was introduced by them that will isolate the memory region and hardware registers access for a kernel module hence the operations of the kernel module are intercepted. The isolation creates a performance overhead.

Nagy et al. in their research addresses the challenge of detecting rootkits in embedded IoT devices by using dynamic analysis [34]. they discussed the trusted execution environments (TEE) available in popular IoT platforms. The TEE provides an isolated environment for our rootkit detection algorithms, preventing interference from rootkits even with root privileges. They also proposed algorithms for detecting rootkit components in persistent storage.

2.4.3 Discussion

All the above existing researches have used various techniques including the dynamic analysis and memory analysis to detect the rootkits. Whereas the researches that used memory analysis for malware detection have also been discussed. All listed techniques have some limitations too. As memory analysis seems to be promising technique that had been recently used for malware detection can be a good technique for rootkit detection too. As rootkits hide themselves in memory and have root access so the approach can provide better solutions. Also, if the dataset with instances recorded over years is used then it will give more efficient results.

The rootkits have evolved themselves over the time. They are a true example of concept drift (Machine learning terminology used to define the change in relationship of input and targeted variable over the period of time). To overcome this issue, the dataset must be extensive and must be prepared over a long period of time.

In this research we have proposed a rootkit detection technique that uses memory analysis, machine learning and deep learning to address all the issues that rootkit detection

faces.

2.5 Summary

In this chapter we have discussed the rootkits detection techniques, its distribution methods and infection process. More over we have also discusses the machine learning and deep learning algorithms that can be used for rootkit detection. Other than this we highlighted the existing research on the use of memory analysis for the detection of various malwares as well as the existing trend of techniques used to detect the rootkits. In the upcoming chapter, the research methodology will be discussed that is used during this project.

Chapter 3

Research Methodology

This chapter will represent the methodology used through this study. All the steps and phases will be defined later in the chapter, while the details will be discussed in the subsequent chapters. As the project involves memory analysis and machine learning, the steps involved include the creation of memory dumps (data acquisition), memory analysis, data cleaning and labeling, exploratory data analysis, normalization, feature engineering and model evaluation. Figure 3.1 shows the complete process that had been followed during the research. Following the description of the process, next section will highlight the tools and technologies that had been used in the research.

3.1 Research Methodology

3.1.1 Data Acquisition

For any Machine Learning and Deep learning algorithm, there are two ways of data acquisition, one is to use to the existing datasets and transform them according to the ongoing research and other way is to collect the samples and generate your own dataset for the research. In our research we have created our own dataset by collecting the samples and analyzing them to create the dataset that can be used for the experiment.

Creating memory dumps:

The first step will be capturing the memory image. The process is crucial as we are intended to use the memory analysis technique so this can help us to do the job. This

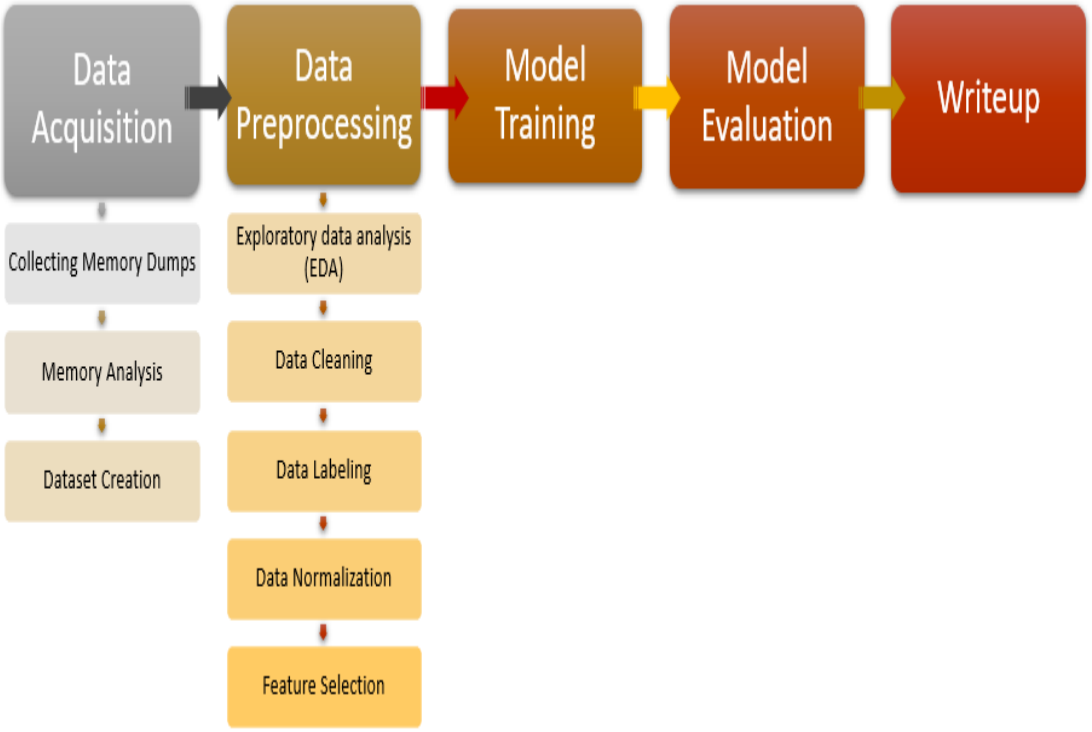


Figure 3.1: Research Methodology (Process model)]

step captures all the memory information including the RAM. The memory image can provide memory contents and system states, which can be used to diagnose any problem (in our case it can help us to detect the presence of rootkits). There are many ways and tools that can be used to create the dump depending upon the requirement.

Memory Analysis:

The second step of data acquisition is memory analysis. In this step, the memory dumps are fed to any tool that can perform memory forensics or memory analysis. Analysis is done based on different features under consideration. The result from memory analysis is in raw form and need to be further statistically analyzed which is done in the next step.

Dataset Creation:

In this step the raw data from the previous step is fed to excel file and perform basic statistical analysis, like mean, mode etc. After which the statistical results for every sample are combined in one sheet to create the dataset for the ongoing project.

3.1.2 Data Preprocessing

This step is also known as feature engineering, and can be completed in the following steps.

Exploratory Data Analysis (EDA):

EDA is considered to be a crucial stage for machine learning as well as deep learning as in this step the raw dataset is analyzed to summarize the main features [35]. Many statistical methods and graphical methods are applied to visualize the data. Hence EDA is used to analyze and understand the dataset in detail to further use it for data preprocessing/feature engineering.

Data Cleaning Labeling:

Data cleaning is a process where the instances are entries that may lead to inaccurate results are removed. This involves various techniques and methods, for example remov-

ing the null, empty or duplicate entries. Whereas Data labeling is a process in which labels are assigned to the data entries. In our case we labeled the data as B and M.

Data Normalization:

The data normalization is an important part of the data preprocessing/ feature engineering. In this process the entries throughout the dataset are checked and converted to a defined range to facilitate the data preprocessing and model training. The process is considered to be important for two basic reasons, one is it increases the cohesiveness of the data, second is that it changes the data consistently throughout the dataset. This also increases the accuracy and detection rates.

Feature Selection:

Feature selection is an important step where the most important features are selected that have greater impact on the results. This step is typically performed in combination with the EDA and Data normalization as they create a loop to yield a dataset that can provide best results for the given algorithm. The process is beneficial as it can help to improve the accuracy also it helps to reduce the computational time hence reducing the cost.

By following the above-mentioned steps of feature engineering, raw dataset can be cleaned, labeled and normalized that can further help to train the machine learning and deep learning model to get better results.

3.1.3 Model Training

After data preprocessing, now we can use the dataset for model training. During this phase, a concerned machine learning or deep learning algorithm/model is trained using the part of dataset. The process is done by examining the data for its features, patterns to generate the predictions. This must be kept in mind that the appropriate algorithm must be used as different algorithms shows different performance with same feature set. The nature of the problem, the dataset, complexity, computational efficiency should be taken into account. Therefore, by use of well processed dataset with right algorithm, the accuracy of results can be improved.

3.1.4 Model Evaluation

This would be the last step for our research. This step involves comparing of the predictions with the expected outcomes. It is important to ensure that the testing data must be different from the training data otherwise false high level of accuracy is obtained. If the evaluation does not meet the required criteria than the model is fed to tuning, where the parameters of the algorithm are adjusted to optimize its performance. Hit and trial rule can be used between parameter configurations for model training to increase the accuracy and performance of the model. Hence better performance will help us to detect the rootkits with high accuracy.

3.2 Tools, technologies and environments

During the research process, we have used different tools, technologies, plugins and environment. We collected the rootkit samples and run on the real devices to capture the memory image which is then being analyzed using the memory analysis technique. The table 3.1 lists all the tools and technologies we used during the process of detecting the rootkits from the memory images.

Table 3.1: Tools/Technologies/Environment used

Tools/Technologies/Environment	**Description of Usage**
VMware Workstation Pro v16.2 [36]	The standard desktop hypervisor used to run the controlled environment for Rootkit testing and analysis
Microsoft Windows 11x64 for Workstations [37]	The Windows 11 environment was used to test the rootkit samples
Kali Linux 2020 [38]	The Linux environment is suitable for penetration testing and security analysis. Memory analysis was performed in this environment
Dumpit.exe [39]	The tool was used for memory acquisition inside Windows

Continued on next page

Tools/Technologies/Environment	**Description of Usage**
Volatility v3.2 [40]	This tool was installed inside the Linux environment for memory analysis purposes
Python [41]	Python programming language was used for writing different scripts of data preprocessing and model training and evaluation
WEKA v3.9 [51]	WEKA is a python-based tool for machine learning algorithms. Different models were tested using this tool
VirusTotal [43]	VirusTotal is an online repository for different malwares. It was used to download the rootkit samples
VirusShare [44]	VirusShare is also an online repository for different malwares. It was used to download the rootkit samples
Numpy [45]	It is a Python library used for performing all mathematical operations
Pandas [46]	It is another Python library used for data processing
Matplotlib [47]	This is a Python library used for graphical representation like confusion matrix for the data
Category-encoders [48]	It is a Python library used to encode categorical variables into numeric using different techniques
Tensor Flow [49]	It is a Python library used for high-performance numerical computation
Scikit-learn [50]	This Python library was used for implementing machine learning algorithms

Continued on next page

Tools/Technologies/Environment	**Description of Usage**
Jupyter Notebook v6.4.12 [51]	It is a web-based interactive computing notebook environment used for machine learning and deep learning
OverLeaf [52]	It is an online typesetting system which was used for the write-up
Microsoft Excel [53]	It is a Microsoft tool used for basic statistical analysis and creating the dataset

3.3 Summary

In this chapter we discussed the research methodology to be followed throughout the ongoing research. We have also highlighted all the tools, technologies and environments that have been used during the research with the description of why they have been used.

Chapter 4

Data Acquisition and Preprocessing

In this chapter we will go through in detail with the data acquisition and data preprocessing phase. The first part of the chapter will define the data acquisition process while the second half will define the data preprocessing phase.

4.1 Data Acquisition

Data acquisition is an important step of forensics as well as machine learning and deep learning. In this step the data is collected with care to be processes further. In the ongoing research we have collected the rootkits sample from the online repositories of VirusTotal [43] and VirusShare [44]. The rootkits samples are then tested in the Windows 11 environment. Following are the phases in which data acquisition was done our research.

4.1.1 Creating Memory Dump

In the first phase of the data acquisition the memory image is taken with the help of the Dumpit tool. The whole RAM is being captured. As we had used a hypervisor, windows 11 was installed on a virtual machine. For a malware to be run in a secure environment, certain steps must be taken.

- Windows must be activated

- Use a different network segment or a different network subnet apart from the main network
- System resources must be realistic (Memory can be from 1GB to 8/16GB). The virtual machine that we had created has a RAM of 1 GB
- Windows Defender must be disabled

Rootkit samples were run to see their behaviors. For the purpose two types of memory images are obtained. One is when no rootkit is present there (benign image) and the other one is where rootkit is running in the same environment (malign). The memory images are 1 GB each as shown in the figure 4.1.

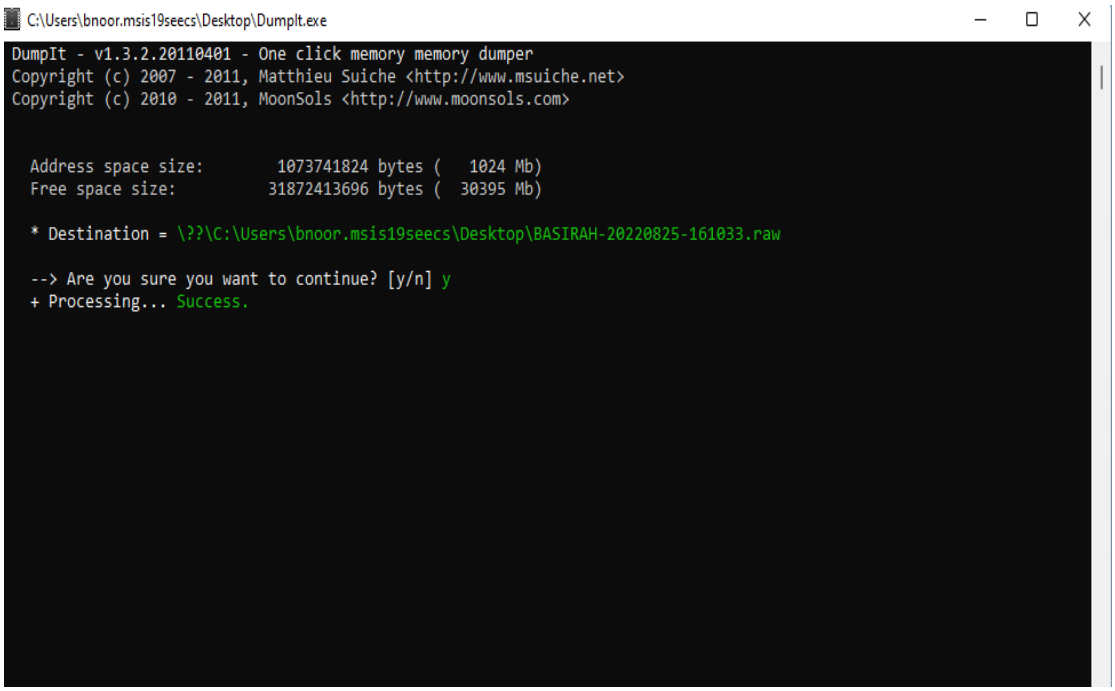


Figure 4.1: Memory image creation using dumpit tool

4.1.2 Memory Analysis

The second phase of the data acquisition is memory analysis. Volatility is a strong tool that has been used for the purpose. It was installed in a kali Linux environment. The memory images that we had collected were transferred to this virtual machine and analyzed one by one. Figure 4.2 shows the general info of one of the memory dumps collected and tested by the volatility.

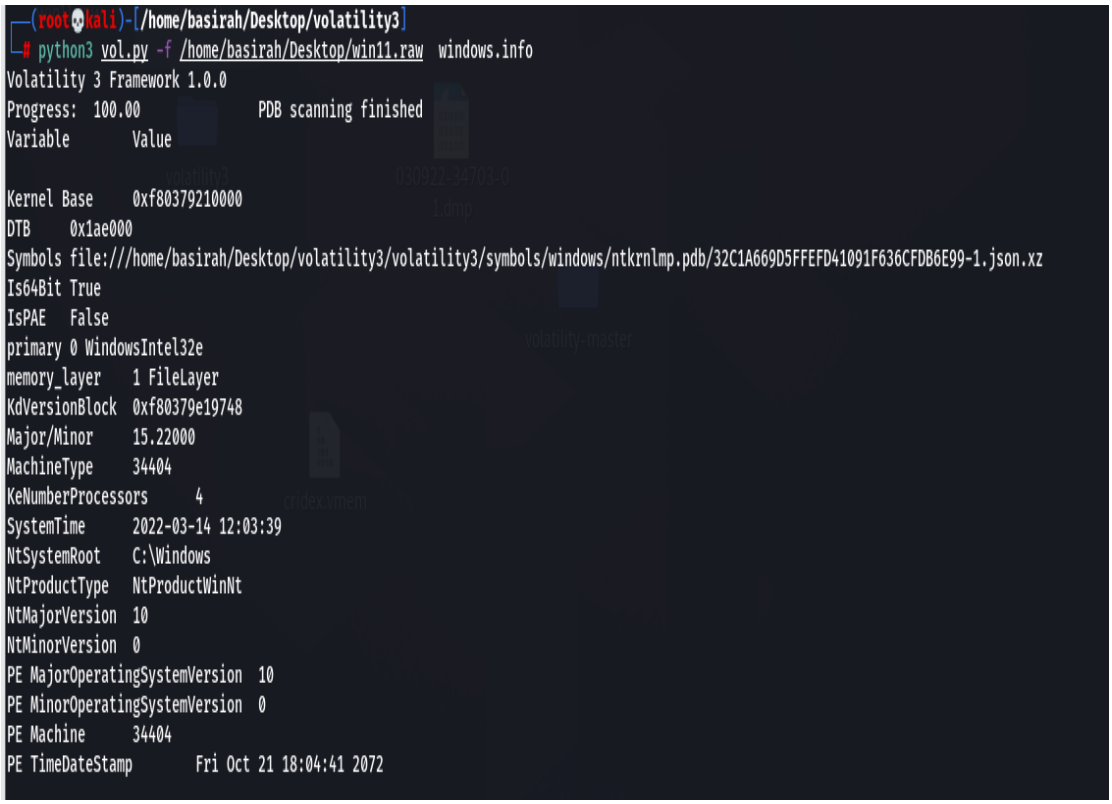


Figure 4.2: General info of the Memory Dump by volatility

The memory features in which we are interested in are: Processes, DLL, handles, modules, injected codes, networks, services, callbacks and privileges. Different commands are used to capture the above-mentioned features. Table 4.1 is created to list the features, and corresponding volatility commands that were used. Whereas table 4.2 lists all 53 sub-features on which our analysis will base.

Table 4.1: Memory Features and Volatility commands for Rootkit Analysis

Features	**Volatility Commands**	**Description**
Processes	`windows.pslist`	The rootkits directly affect the processes by showing themselves as legitimate programs.
DLL	`windows.dlllist`	The rootkits use DLLs to insert malicious code using root access to a legitimate program.
Handles	`windows.handles`	Handles give information about the rootkit's behavior, such as reading or writing a file with root access or accessing registry keys.
Modules	`windows.modules`	Rootkits can modify modules, allowing them to inject their code into the operating system's memory space.
Injected codes	`windows.malfind`	This command can determine if rootkits have injected any malicious code into a legitimate process.
Network	`windows.netscan`	This command provides information about how the rootkit communicates with the attacker to send and receive commands.
Services	`windows.svcscan`	Rootkits can affect background processes known as services.

Continued on next page

Features	Volatility commands	Description
Callbacks	`windows.callbacks`	Rootkits can tamper with callbacks that are executed when any event occurs.
Privileges	`windows.privileges`	Privileges can indicate if rootkits have given any commands to legitimate processes to perform specific tasks, such as changing the time.

Table 4.2: List of Features and Sub-features

Sr No.	Features	Sub-features
1	Processes	pslist.nprocc
2	Processes	pslist.nppidd
3	Processes	pslist.avgthreadss
4	Processes	pslist.nprocs64bitt
5	Processes	pslist.avghandlerss
6	DLL	dlllist.ndllss
7	DLL	dlllist.avgdllsperprocc
8	Handles	handles.nhandless
9	Handles	handles.avghandlesperprocc
10	Handles	handles.nportt
11	Handles	handles.nfilee
12	Handles	handles.neventt
13	Handles	handles.ndesktopp
14	Handles	handles.nkeyy
15	Handles	handles.nthreadd
16	Handles	handles.ndirectoryy
17	Handles	handles.nsemaphoree
18	Handles	handles.ntimerr
19	Handles	handles.nsectionn
20	Handles	handles.nmutantt

Continued on next page

Sr No.	Features	sub-features
21	Handles	handles.nsymboliclinkk
22	Handles	handles.nkeyedeventt
23	Handles	handles.nprocesss
24	Handles	handles.ntokenn
25	Handles	handles.nwindowstationn
26	Handles	handles.niocompletionn
27	Handles	handles.nwmiguidd
28	Handles	handles.nwaitableportt
29	Handles	handles.njobb
30	Injected Codes	malfind.ninjectionss
31	Injected Codes	malfind.commitChargee
32	Modules	modules.nmoduless
33	Services	svcscan.nservicess
34	Services	svcscan.kerneldriverss
35	Services	svcscan.fsdriverss
36	Services	svcscan.processservicess
37	Services	svcscan.sharedprocessservicess
38	Services	svcscan.interactiveprocessservicess
39	Services	svcscan.nactivee
40	Callbacks	callbacks.ncallbackss
41	Network	Netscan.nudpp
42	Network	Netscan.ntcpp
43	Privileges	Priv.nprocesss
44	Privileges	Priv.avgprivperprocc
45	Privileges	priv.SeSystemEnvironmentPrivilegee
46	Privileges	priv.SeSystemProfilePrivilegee
47	Privileges	priv.SeSystemtimePrivilegee
48	Privileges	priv.SeTakeOwnershipPrivilegee
49	Privileges	priv.SeTcbPrivilegee
50	Privileges	priv.SeTimeZonePrivilegee

Continued on next page

Sr No.	Features	sub-features
51	Privileges	priv.SeUndockPrivilege
52	Privileges	Priv.SeSecurityPrivilege

4.1.3 Creation of dataset

For each sample the information about the features mentioned in table 4.1 are collected and transferred to the csv file. Some statistical analysis has been performed in the Microsoft excel to obtain the sub features listed in table 4.2. As a result, we have obtained features for 400 benign samples and 400 rootkit samples. Figure 4.3 shows a glimpse of the excel file when the data from the volatility is transferred to excel file before performing the basic statistical analysis to obtain values for the features listed in table 4.2.

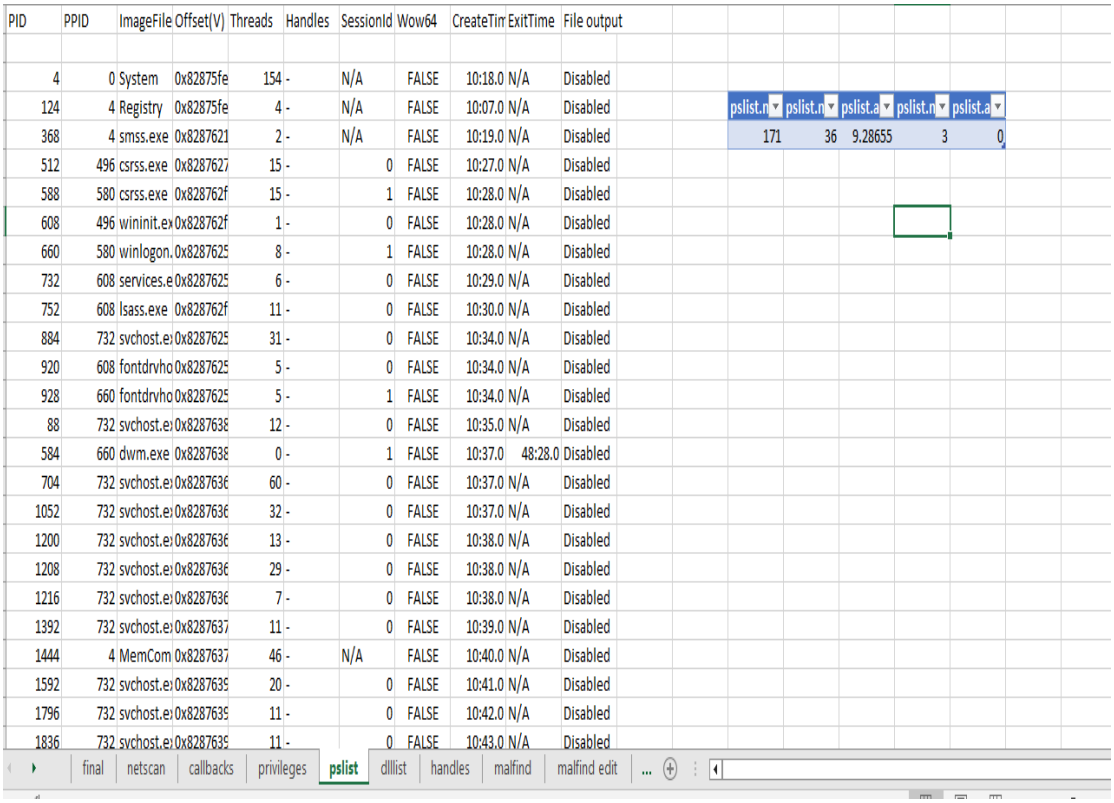


Figure 4.3: Transferring data from volatility to Microsoft Excel

4.2 Data Preprocessing

in this section we shall deiscuss the process of feature engineering/ data preprocessing in detail.

4.2.1 Exploratory Data Analysis

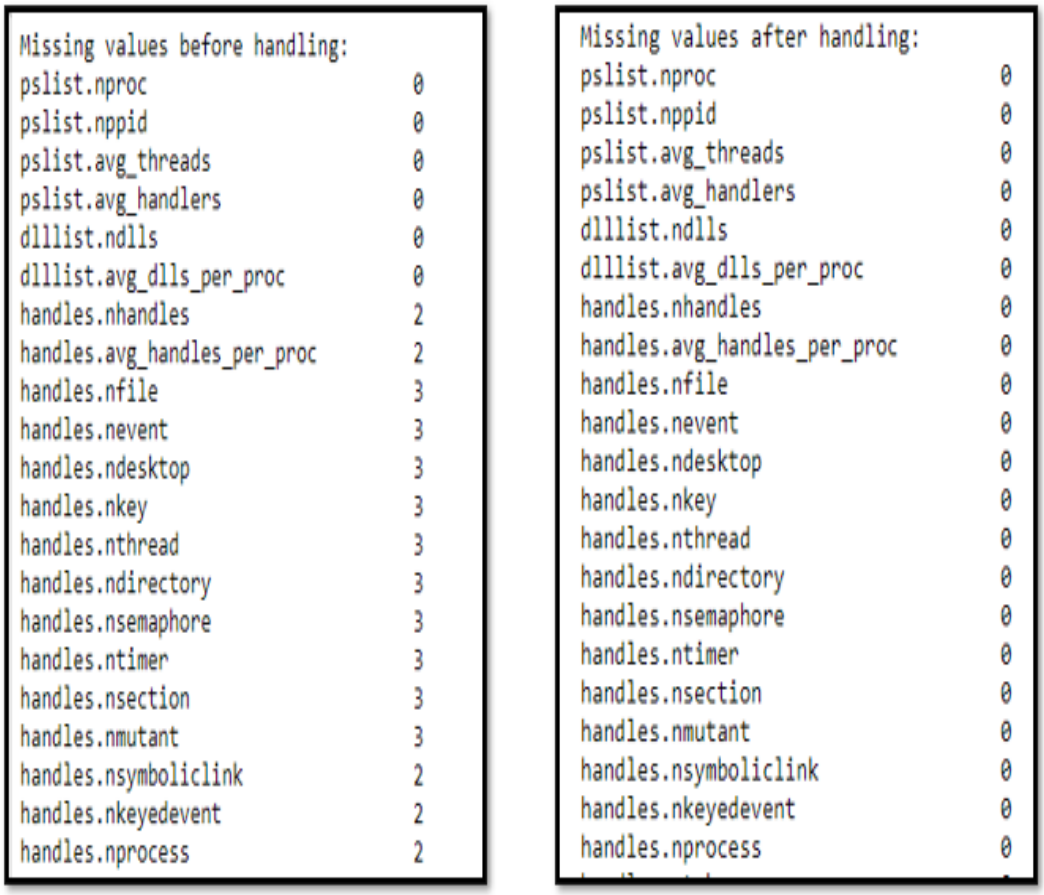
It is an important step of the data preprocessing. Data is analyzed in detail to highlight the important attributes. We have used the python and some of its libraries like pandas [45] and matplotlib [46] to perform the exploratory data analysis. In the first step we checked the data using the python. It helped us to calculate the weight of each attribute by using the mean, minimum, maximum, standard deviation, quantiles etc. next we checked for any missing value and outliers according to the calculated quantiles. Figure 4.4 shows the results of EDA that we have performed on the section of dataset.

	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.avg_handlers
count	800.000000	800.000000	800.000000	800.000000
mean	0.398932	0.449667	0.310457	0.483222
std	0.088429	0.153284	0.209596	0.297635
min	0.000000	0.000000	0.000000	0.000000
25%	0.354167	0.266667	0.117793	0.200767
50%	0.395833	0.466667	0.254447	0.330260
75%	0.416667	0.600000	0.521837	0.792135
max	1.000000	1.000000	1.000000	1.000000

Figure 4.4: Exploratory Data Analayis

4.2.2 Data Cleaning

This step will clean the dataset by removing the missing entries. Missing entries can create many issues for the next steps to detect the rootkits. Python library Pandas has been used for the purpose. We used the function fillna() to replace the missing entries with 0. The figure 4.5 shows the missing entries inside the section dataset. And how our code handles to remove the missing entries.



Missing values before handling:

pslist.nproc	0
pslist.nppid	0
pslist.avg_threads	0
pslist.avg_handlers	0
dlllist.ndlls	0
dlllist.avg_dlls_per_proc	0
handles.nhandles	2
handles.avg_handles_per_proc	2
handles.nfile	3
handles.nevent	3
handles.ndesktop	3
handles.nkey	3
handles.nthread	3
handles.ndirectory	3
handles.nsemaphore	3
handles.ntimer	3
handles.nsection	3
handles.nmutant	3
handles.nsymboliclink	2
handles.nkeyedevent	2
handles.nprocess	2

Missing values after handling:

pslist.nproc	0
pslist.nppid	0
pslist.avg_threads	0
pslist.avg_handlers	0
dlllist.ndlls	0
dlllist.avg_dlls_per_proc	0
handles.nhandles	0
handles.avg_handles_per_proc	0
handles.nfile	0
handles.nevent	0
handles.ndesktop	0
handles.nkey	0
handles.nthread	0
handles.ndirectory	0
handles.nsemaphore	0
handles.ntimer	0
handles.nsection	0
handles.nmutant	0
handles.nsymboliclink	0
handles.nkeyedevent	0
handles.nprocess	0

Figure 4.5: Missing values before and after Data Cleaning

4.2.3 Data Normalization

Data normalization is also an important step for our dataset as it will make all the values in the range of 0 and 1. The step is important because it will increase the accuracy of the model. For the purpose we have used the min max method [54]. We have performed

the data normalization using the python. The formula for the min max method is as below.

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{4.2.1}$$

The figure 4.6 shows the data normalized for the section of our data before and after the data normalization.

	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc
0	42	12	13.044319	0	298.189057	2160	51.428571
1	42	16	11.321509	0	272.325027	2087	48.574388
2	44	12	12.559837	0	286.733131	2153	48.931818
3	41	12	13.291813	0	297.472689	2076	50.087996
4	41	14	12.887641	0	288.356706	2081	50.094247

	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc
0	0.416667	0.266667	0.546119	NaN	0.860177	0.539939	0.898432
1	0.416667	0.533333	0.303725	NaN	0.669007	0.508075	0.726662
2	0.458333	0.266667	0.477954	NaN	0.775502	0.536883	0.748173
3	0.395833	0.266667	0.580940	NaN	0.854882	0.503274	0.817754
4	0.395833	0.400000	0.524075	NaN	0.787503	0.505456	0.818130

Figure 4.6: Data section before and after data normalization

4.2.4 Feature Selection

Feature selection is also an important step. For example in figure 4.6 we, have seen that for the column pslist.nproc64bit we have got NaN values. Such values will hinder the results by the model. So we need to select the best features for the model. This process

is called feature selection. There are many ways to perform the feature selection but we have used the recursive elimination method [55] to remove the least important features for the dataset. As a result, we removed following features and left only with 47.

- Pslist.nproc64bit
- Handles.nport
- Svcscan.fsdrivers
- Scsscan.interactiveprocessservices
- Priv.SeTcbPrivilege
- Priv.seSecurityPrivelege

4.3 Summary

In this chapter we had went in detail with the data acquisition process and basic data pre processing steps that we followed during the course of this project.

Chapter 5

Model Training and Evaluation

In this chapter we will see in detail how we trained our models and the results we have received from them. The results will be evaluation using the measures described in section 5.2. In the end our results will be compared with the existing literature.

5.1 Model Training

For a machine learning and deep learning mechanism it is a very import step. In this step we chose machine learning and deep learning algorithms and provide them with some of the data from the dataset to train the model.

For the rootkit detection we have chosen four machine learning algorithms (Support vector machine, Random Forest, K-Nearest Neighbor and Decision Tree), and one deep learning algorithm (Long- and Short-term Memory Model). We divided the dataset into two parts. 75% of it was used to train the model and 25% of the dataset was used to test the model.

5.2 Model Evaluation Measures

The evaluation process is the critical step as it will allow us to see that how good are our solutions. There can be many evaluation measures that can be used to measure the effectiveness of the proposed method, but we have chosen accuracy, precision, recall, F1-score and time execution as evaluation parameters. Except time execution, other chosen parameters are defined by a confusion matrix [56].

Confusion Matrix:

A confusion matrix is a 2x2 matrix with different actual and predicted values as shown in figure 5.1. It is used to calculate the accuracy, precision, F1-score, Recall, Specificity etc. The values in the matrix are knows as

- True Positive (TP)
- False Positive (FP)
- True Negative (TN)
- False Negative (FN)

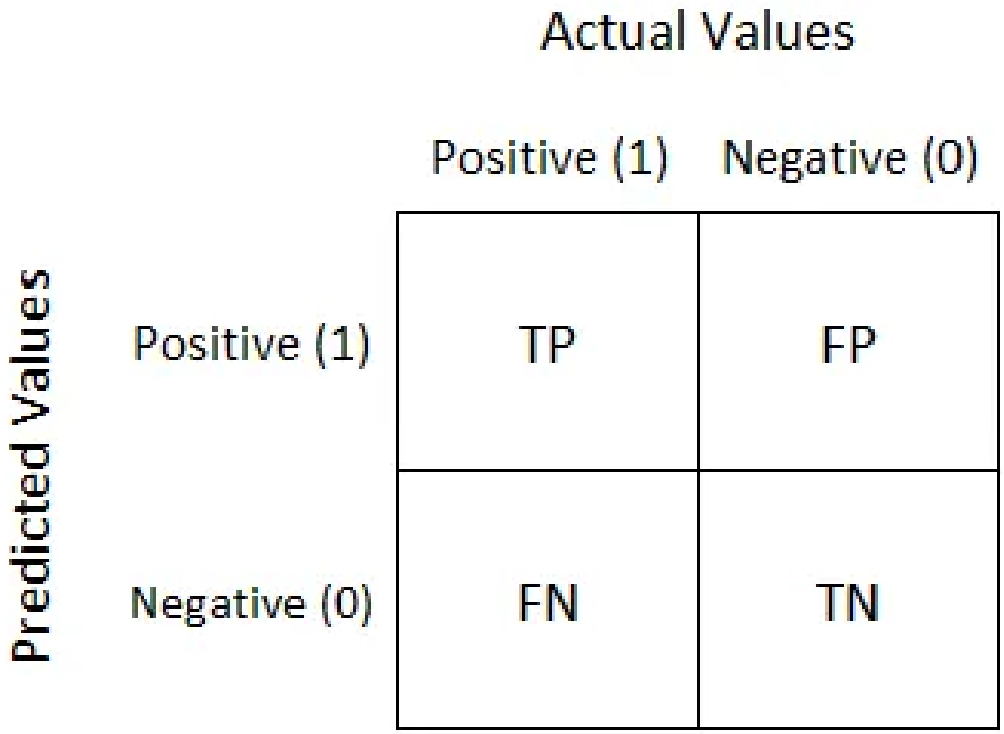


Figure 5.1: Understanding of Confusion Matrix

In the confusion matrix, Positive and Negative are predicted values and True and False are actual values. In our case where rootkits are going to be detected using the machine learning and deep learning techniques, positive (1) is considered as rootkit and negative (0) is considered as benign.

True Positive (TP): True positive means that the model predicted the result as positive and it is right. In our case it means the model predicted the memory dump sample has rootkit in it and the prediction is true.

True negative (TN): True negative means that the model predicted the result as negative and it is right. In our case it means the model predicted the memory dump sample is benign and the prediction is true.

False positive (FP): False positive means that the model predicted the result as positive and it is wrong. In our case it means the model predicted the memory dump sample has rootkit in it and the prediction is wrong. This is also known as type I error.

False negative (FN): False negative means that the model predicted the result as negative and it is wrong. In our case it means the model predicted the memory dump sample is benign and the prediction is wrong. This is also known as type II error.

5.2.1 Accuracy

It is defined as how much data is true, correct or free from errors and mistakes or we can say that it is the quality of measurement. Its formula is given below.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2.1}$$

5.2.2 Precision

It is defined by percentage of the positive predictions that are correct. Its formula is given below.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2.2}$$

5.2.3 F1-score

It is defined as a harmonic mean of recall and precision. It is usually used to compare two models. formula for it is given below.

$$\text{F1\_score} = \frac{2 \times (Precision \times Recall)}{Precision + Recall} \tag{5.2.3}$$

5.2.4 Recall

It is defined by percentage of the actual positive predictions that are correctly classified. Its formula is given below.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2.4}$$

5.2.5 Execution time

It is defined as the time a model takes to calculate the result for the given dataset. In our case we have calculated the execution time using the time library of python.

5.3 Results and Evaluation

In this section we will through light on the results we have obtained and will evaluate them according to the metrics mentioned in section 5.2.

As mentioned in section 5.1, that we divided the data into 75/25 and trained the dataset with decision tree, random forest, support vector machine, K-nearest neighbor (KNN) and long- and short-term Memory (LSTM) method. 25% data was used for the evaluation purpose for the above-mentioned data models. Initially we chose 53 features, after the feature selection our features were left to be 47.

The confusion matrix is used to calculate different evaluation metrics. The confusion matrix of different models are shown in figure 5.2.

Final results show that support vector machine shows the best result with accuracy equal to 96.2%. after that Random Forest gives accuracy of 95.5%. if we compare the two, the execution time of SVM is less than the RF. Overall K-nearest neighbor takes lowest time to generate the results. LSTM shows a result 85.8% which is not so good but as a deep learning model we can consider it a good outcome. The table 5.1 shows the final results for different models. we testes a deep learning model too.

5.3.1 Validation

Validation is an import phase of this process. The phase helps us to validate our results. For the purpose we have used k-fold cross validation to check our results.

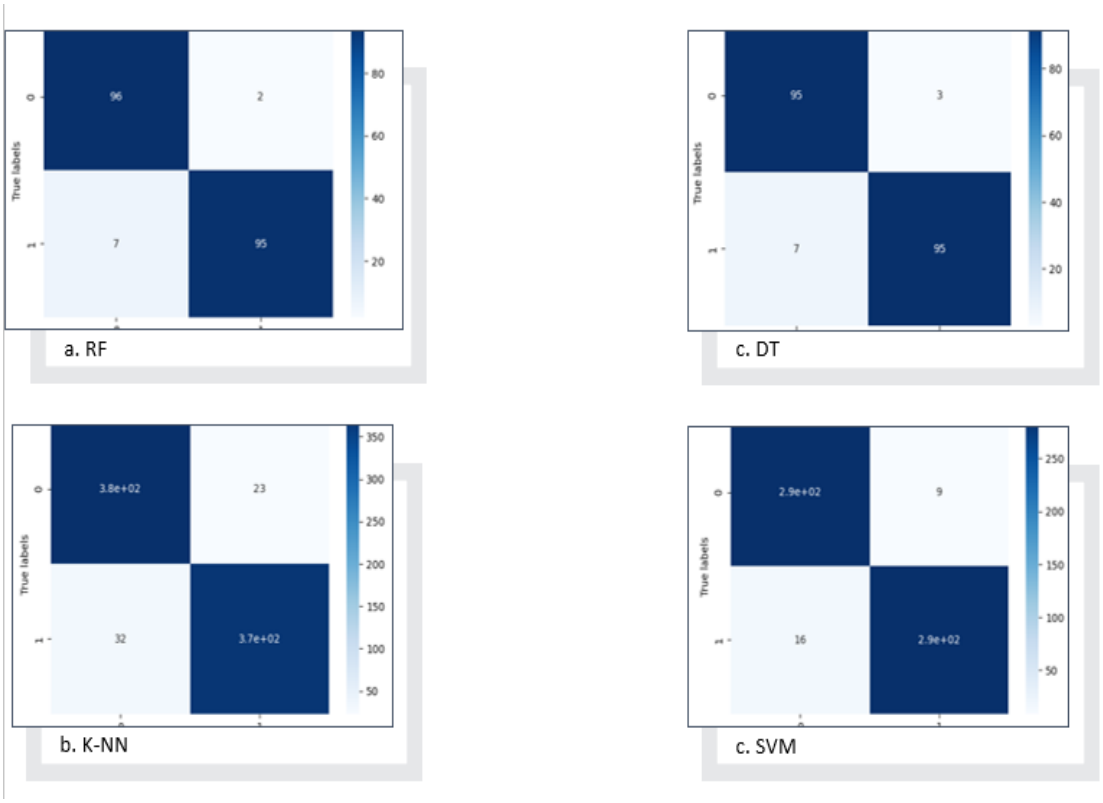


Figure 5.2: Confusion matrix for different models

Table 5.1: Final results for Different Algorithms

Algorithms	**Accuracy**	**Recall**	**Precision**	**F1-Score**	**Execution Time**
Random Forest	95.5%	93%	98%	95%	0.25s
K-Nearest Neighbor	92.8%	91.5%	93.8%	92.7%	0.03s
Decision Tree	95%	93%	97%	95%	0.04s
Support Vector Machine	96.2%	94%	99%	96%	0.1s
Long- and Short-Term Memory	85.8%	83%	87%	85%	124.02s

K-fold cross-validation is a method used for machine learning evaluation. It evaluates the performance of a model on a limited dataset. It helps to assess how well the model generalizes to new, unseen data. The basic idea behind k-fold cross-validation is to split the available data into k subsets or folds. The model is trained and evaluated k times, each time using a different fold as the test set and the remaining folds as the training set. The performance metrics obtained from each fold are then averaged to get an overall estimate of the model's performance.

We have used WEKA tool for the purpose to check k-fold cross validations for RF, K-NN, DT and SVM for k=2,5,10. Table 5.2 shows the k- fold cross validation.

Table 5.2: K-fold cross-validation for Different K values

Algorithms	**K=2**	**K=5**	**K=10**
Random Forest	95.8%	95.5%	95.1%
K-Nearest Neighbor	93.5%	93.1%	92.8%
Decision Tree	95.3%	95%	95%
Support Vector Machine	96%	96.2%	96.2%

5.4 Comparison with existing literature

In the field of research, significant advancements have been made in the area of rootkit detection. Various researchers have proposed different approaches utilizing machine learning, deep learning, and other techniques for malware detection. In Table 5.3, we present a compilation of research works closely related to our proposed solution, along with their respective results.

There are many aspects in which our research differs from the existing rootkit detection solution. For detection of rootkits, memory analysis got a very little importance so far, but we found this method to be a promising method for finding the rootkits. In respect to memory analysis, we had used volatility 3.2 which is a new version that encounters symbol list instead of profiles, so dealing with this also remained a new thing for us. Also, before machine learning algorithms were only common, we tried to test a deep learning algorithm too. Deep learning models are considered to be better than machine learning algorithms because they exhibit strong generalization capabilities, making them

suitable for tasks with diverse and complex patterns. However, deep learning requires substantial computational resources and labeled training data. Another good thing about our research is that we have also evaluated our results based on the execution time, but existing literature on rootkit detection have never used that.

Our research work aims to contribute to the field of malware detection, particularly in the context of rootkits. We strive to fill the gaps identified in existing solutions and provide an effective and robust approach for identifying and mitigating the threats posed by rootkit malware.

5.5 Summary

In this chapter we have gone in detail with the model training process and model testing process. We evaluated the results and compared them with the existing literature.

Table 5.3: Comparison of Detection Approaches and Classifier Algorithms with Accuracy

Ref	**Detection Type and Algorithm**	**Dataset Source**	**Benign**	**Rootkit**	**Classifier Algorithm with Accuracy**
[57]	Dynamic/Machine learning	VirusTotal	24000	4500	SVM (99.9) OC-SVM (75.1) Naïve Bayes (75.35) DT (83.71)
[33]	Memory Forensics/Machine Learning	MalShare VirusShare	2600	10500	J84 (96.5) BayesNet (89.4) Naïve Bayes (86.4) SVM (94.09)
[24]	Dynamic/Machine Learning	Rootkit.com VirusShare VX Heaven	473	418	DT (95.11) RF (96.74) KNN (91.85)
[58]	Dynamic/Deep Learning	x	1300	700	FNN (67.7)
Our Approach	Memory Analysis/Both Machine learning and Deep Learning	VirusTotal VirusShare VX Heaven	400	400	DT (95) RF (95.5) KNN (92.8) SVM (96.2) LSTM (85.8)

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Rootkit is a malicious program that exist with extremely insidious nature. The heinous nature of these programs to evade detection, make it difficult for the detection tools to know when the rootkits have infiltered the system. Once it gets installed on the system, alien behaviors can be observed that shows that some remote attacker has got an access to the system. These types of malwares are extremely dangerous as the existence of the attack cannot be judged by traditional detection tools and techniques, as a result can cause significant data loss and damage to the system. Although there are many efforts available to detect the malwares particularly rootkits, but still there always remained loop holes. Therefore, there is a need to device a standard mechanism that can detect these rootkits efficiently. Therefore, we explored the efficiency of memory analysis technique to detect the rootkits. Also, we tried to develop an automated tool by using machine learning algorithms and deep learning to easily and accurately identify the presence of rootkits in a system.

We used memory analysis combined with machine learning and deep learning algorithms. For the purpose, we captured the memory dumps (data acquisition) at different time, performed memory analysis, created csv file, performed data preprocessing and after that model training and then model evaluation. We have evaluated based on accuracy, execution time, precision, recall and f1-Score. Lastly, we have performed K-fold cross validation to validate the results of our model.

We have provided rootkit detection solution using Random Forest, Support vector ma-

chine, K- Nearest neighbor, decision tree and long and short-term memory method. Our final results show that support vector machine gives the best result with accuracy equal to 96.2%. after that Random Forest that gives accuracy of 95.5%. if we compare the two, the execution time of SVM is less than the RF. Overall K-nearest neighbor takes lowest time to generate the results. LSTM shows a result 85.8% which is not so good but as a deep learning model we can consider it a good outcome. Also, the execution time of LSTM is far more than classification algorithms. Deep learning execution time is generally greater than that of traditional machine learning due to computationally intensive operations like matrix multiplications and convolutions contribute to longer execution times.

Hence this research work provides a machine learning and deep learning-based rootkit detection model which can be used in future. The thesis has contributed in following way.

6.2 Thesis Contribution

This research work makes following novel contribution to the literature.

- Volatility has never been used for rootkits detection before.
- We used volatility 3.2, which has symbol tables instead of profiles. Before this volatility 2.6 had been used for malware detection that uses profiles for memory analysis.
- We have created our own dataset by capturing memory images from the windows 11.
- Proposed an efficient technique based on the memory analysis to detect the hidden rootkits.
- Reduced the computational overhead by choosing best minimal number of features (features=47).

6.3 Future Work

There are still room for improvements which could be made in this research work to detect the rootkits.

6.3.1 Multi-classification of Rootkits sub categories

There are several notable rootkit families that have been identified and studied by security researchers. The research can be extended in a way to tackle them too. The process needs to adopt multi class categorization. Here are a few examples of those sub categories:

- Alureon/TDL
- Zeus/Zbot
- Rustock
- Hacking Team's RCS

This thing needs to be kept in mind that rootkit families are dynamic and new variants continue to emerge. So, there would be always a room of improvement at this point to detect these categories of rootkits.

6.3.2 Reducing time for Memory Analysis

The most time taking step throughout the project is memory analysis. Performing memory analysis on each and every sample takes a lot of effort. And then collecting information about the DLLs, Handles, Processes, Privileges, services, networks, modules and injection codes require execution of separate commands. After which the information is converted to the csv file and final values for the sub features are collected.

The process can be minimized by writing a plugin for volatility in python language that can execute all these commands at a time and outputs in a csv file with final values.

Bibliography

[1] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy, and L. Iftode, "Rootkits on smart phones," Proceedings of the Eleventh Workshop on Mobile Computing Systems Applications - HotMobile '10, 2010, doi: https://doi.org/10.1145/1734583.1734596.

[2] A. Bunten, "UNIX and Linux based Rootkits Techniques and Countermeasures," www.semanticscholar.org, 2004. https://www.semanticscholar.org/paper/UNIX-and-Linux-based-Rootkits-Techniques-and-Bunten/ (accessed Mar. 31, 2023).

[3] S. Manap, "Rootkit: Attacker undercover tools." Accessed: Apr. 06, 2020. [Online]. Available: http://forum.ouah.org/salirootkit.pdf

[4] A. Matrosov, E. Rodionov, and S. Bratus, Rootkits and Bootkits | No Starch Press. 2019. Accessed: Jan. 25, 2020. [Online]. Available: https://nostarch.com/rootkits

[5] "Detecting DKOM - Designing BSD Rootkits [Book]," www.oreilly.com. https://www.oreilly.com/library/view/designing-bsd-rootkits/9781593271428/ch07s02.html (accessed Mar. 11, 2023).

[6] G. Gross, "Rootkit Detection: Techniques and Best Practices," cybersecurity.att.com, Mar. 28, 2016. https://cybersecurity.att.com/blogs/security-essentials/rootkit-detection-techniques-and-best-practices (accessed Mar. 11, 2023).

[7] L. Liu, Z. Yin, Y. Shen, H. Lin, and H. Wang, "Research and Design of Rootkit Detection Method," Physics Procedia, vol. 33, pp. 852–857, 2012, doi: https://doi.org/10.1016/j.phpro.2012.05.145.

[8] "Rootkits: evolution and detection methods," ptsecurity.com, Nov. 03, 2021. https://www.ptsecurity.com/ww-en/analytics/rootkits-evolution-and-detection-methods/

[9] "Kronos Reborn | Proofpoint US," Proofpoint, Jul. 24, 2018. https://www.proofpoint.com/us/threat-insight/post/kronos-reborn (accessed Feb. 25, 2023).

[10] C. Cimpanu, "Scranos rootkit expands operations from China to the rest of the world," ZDNET, Apr. 16, 2019. https://www.zdnet.com/article/scranos-rootkit-expands-operations-from-china-to-the-rest-of-the-world/ (accessed Mar. 25, 2023).

[11] "More Bad News for Android Users," threatstop. https://www.threatstop.com/blog/2016/07/13/more-bad-news-for-android-users (accessed Mar. 25, 2023).

[12] "Matrix - Enterprise | MITRE ATTCKTM," attack.mitre.org, Apr. 01, 2022. https://attack.mitre.org/matrices/enterprise/

[13] "Root Out Rootkits, an Inside Look at McAfee Deep," McAfee, 2013. http://www.intel.ph/content/www/xa/en/%20enterprise-security/mcafee-deep-defender-deepsafe-rootkit-protectionpaper.html

[14] S. Grobman, "Method and apparatus to detect kernel mode rootkit events through virtualization traps," U.S. Patent 7 845 009, Nov. 2010.

[15] IBM, "What is Machine Learning?," www.ibm.com, 2023. https://www.ibm.com/topics/machine-learning

[16] S. Das and M. J. Nene, "A survey on types of machine learning techniques in intrusion prevention systems," IEEE Xplore, Mar. 01, 2017. https://ieeexplore.ieee.org/document/8300169 (accessed Mar. 26, 2020).

[17] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001, doi: https://doi.org/10.1023/a:1010933404324.

[18] D. R. Cutler et al., "RANDOM FORESTS FOR CLASSIFICATION IN ECOLOGY," Ecology, vol. 88, no. 11, pp. 2783–2792, Nov. 2007, doi: https://doi.org/10.1890/07-0539.1.

[19] A. Kumar, K. Abhishek, S. K. Shandilya, and M. R. Ghalib, "Malware Analysis Through Random Forest Approach," Journal of Web Engineering, Dec. 2020, doi: https://doi.org/10.13052/jwe1540-9589.195610.

[20] C. C. Felan, Garcia, and Felix P Muga Ii, "Random Forest for Malware Classification," 2016.

[21] S. Wehunt, "A survey on rootkit detection and prevention techniques.," Proceedings of Student Research and Creative Inquiry Day, vol. 1, May 2017.

[22] M. Nadim, W. Lee, and D. Akopian, "Kernel-level Rootkit Detection, Prevention and Behavior Profiling: A Taxonomy and Survey," 2023.

[23] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: https://doi.org/10.1007/bf00994018.

[24] D. Tian, R. Ma, X. Jia, and C. Hu, "A Kernel Rootkit Detection Approach Based on Virtualization and Machine Learning," IEEE Access, Jul. 2019, doi: https://doi.org/10.1109/ACCESS.2019.2928060.

[25] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android Malware Detection Based on Network Traffic Using Decision Tree Algorithm," Advances in Intelligent Systems and Computing, pp. 485–494, 2018, doi: https://doi.org/10.1007/978-3-319-72550-5_46.

[26] Analytics Vidhya, "A Practical Introduction to K-Nearest Neighbor for Regression," Analytics Vidhya, May 07, 2019. https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/ (accessed Mar. 27, 2023).

[27] R. Sihwail, K. Omar, and K. Akram Zainol Ariffin, "An Effective Memory Analysis for Malware Detection and Classification," Computers, Materials Continua, vol. 67, no. 2, pp. 2301–2320, 2021, doi: https://doi.org/10.32604/cmc.2021.014510.

[28] sihwail, "sihwail/malware-memory-dataset," GitHub, Feb. 05, 2021. https://github.com/sihwail/malware-memory-dataset (accessed Apr. 14, 2023).

[29] A. S. Bozkir, E. Tahillioglu, M. Aydos, and I. Kara, "Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision," Computers Security, vol. 103, p. 102166, Apr. 2021, doi: https://doi.org/10.1016/j.cose.2020.102166.

[30] A. H. Lashkari, B. Li, T. L. Carrier, and G. Kaur, "VolMemLyzer: Volatile Memory Analyzer for Malware Classification using Feature Engineering," IEEE Xplore, May 01, 2021. https://ieeexplore.ieee.org/document/9452028

[31] T. Carrier, P. Victor, A. Tekeoglu, and A. Lashkari, "Detecting Obfuscated Malware using Memory Feature Engineering," Proceedings of the 8th International Conference on Information Systems Security and Privacy, 2022, doi: https://doi.org/10.5220/0010908200003120.

[32] P. Joseph and J. Norman, "Systematic Memory Forensic Analysis of Ransomware using Digital Forensic Tools," International Journal of Natural Computing Research, vol. 9, no. 2, pp. 61–81, Apr. 2020, doi: https://doi.org/10.4018/ijncr.2020040105.

[33] X. Wang, J. Zhang, A. Zhang, and J. Ren, "TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis," Mathematical Biosciences and Engineering, vol. 16, no. 4, pp. 2650–2667, 2019, doi: https://doi.org/10.3934/mbe.2019132.

[34] R. Nagy, K. Németh, D. Papp, and L. Buttyán, "Rootkit Detection on Embedded IoT Devices," Acta Cybernetica, Aug. 2021, doi: https://doi.org/10.14232/actacyb.288834.

[35] "What is Exploratory Data Analysis? | IBM," www.ibm.com. https://www.ibm.com/topics/exploratory-data-analysis#:~:text=Exploratory%20data%20analysis%20(EDA)%20is

[36] "Download VMware Workstation Pro," VMware, Dec. 18, 2019. https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html

[37] Microsoft, "Windows 11 Pro for Workstations | Microsoft," Microsoft Windows for Business. https://www.microsoft.com/en-us/windows/business/windows-11-pro-workstations

[38] Kali, "Our Most Advanced Penetration Testing Distribution, Ever.," Kali.org, 2020. https://www.kali.org/

[39] "DumpIt Free Download for Windows 11, 10, 7, 8/8.1 | Down10.Software," down10.software. https://down10.software/apps/dumpit-windows (accessed Apr. 04, 2023).

[40] "Release Downloads | Volatility Foundation," volatilityfoundation. https://www.volatilityfoundation.org/releases-vol3

[41] Python, "Welcome to Python.org," Python.org, May 29, 2019. https://www.python.org/

[42] VirusTotal, "VirusTotal," Virustotal.com, 2019. https://www.virustotal.com/gui/home/upload

[43] "VirusShare.com," Virusshare.com, 2020. https://virusshare.com/

[44] Numpy, "NumPy," Numpy.org, 2009. https://numpy.org/

[45] Pandas, "Python Data Analysis Library — pandas: Python Data Analysis Library," Pydata.org, 2018. https://pandas.pydata.org/

[46] J. D. H. Droettboom Michael, "matplotlib: Python plotting package," PyPI. https://pypi.org/project/matplotlib/3.5.3/ (accessed Jun. 01, 2023).

[47] W. McGinnis, "category-encoders 2.5.1.post0 : A collection of sklearn transformers to encode categorical variables as numeric," PyPI. https://pypi.org/project/category-encoders/

[48] "tensorflow," PyPI, Feb. 26, 2019. https://pypi.org/project/tensorflow/

[49] "scikit-learn," PyPI, Mar. 2019. https://pypi.org/project/scikit-learn/

[50] Jupyter, "Project Jupyter," Jupyter.org, 2019. https://jupyter.org/

[51] "Weka 3 - Data Mining with Open Source Machine Learning Software in Java," Waikato.ac.nz, 2019. https://www.cs.waikato.ac.nz/ml/weka/

[52] Overleaf, the Online LaTeX Editor, "Overleaf, Online LaTeX Editor," Overleaf.com, 2019. https://www.overleaf.com/

[53] Microsoft, "Microsoft Excel, Spreadsheet Software," www.microsoft.com, 2022. https://www.microsoft.com/en-us/microsoft-365/excel

[54] "sklearn.preprocessing.minmax_scale," scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.html (accessed Apr. 04, 2023).

[55] "sklearn.feature_selection.RFE — scikit-learn 0.23.1 documentation," scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection. RFE.html

[56] S. Narkhede, "Understanding Confusion Matrix," Medium, May 09, 2018. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[57] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, "On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters," Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Apr. 2017, doi: https://doi.org/10.1145/3052973.3052999.

[58] P. Luckett, J. T. McDonald, and J. Dawson, "Neural Network Analysis of System Call Timing for Rootkit Detection," IEEE Xplore, Apr. 01, 2016. https://ieeexplore.ieee.org/abstract/document/7942417 (accessed Apr. 23, 2022).

[59] SciKit-Learn, "Cross-validation: evaluating estimator performance — scikit-learn 0.21.3 documentation," Scikit-learn.org, 2009. https://scikit-learn.org/stable /modules/cross_validation.html

[60] "time — Time access and conversions — Python 3.7.2 documentation," Python.org, 2000. https://docs.python.org/3/library/time.html