International Journal on Engineering Technologies and Informatics

Detection Of Metamorphic Malware Through Opcode and Api Call System Using Machine Learning

Research Article

Volume 6 Issue 1- 2025

Author Details

Asia Othman Aljahdali, Dalya Bokhari, Wejdan Alghamdi*

Cybersecurity Department, College of Computer Sciences and Engineering, University of Jeddah, Saudi Arabia

*Corresponding author

Dr. Asia Othman Aljahdali, University of Jeddah, Cybersecurity Department, College of Computer Sciences and Engineering, Saudi Arabia

Article History

Received: April 11, 2025 Accepted: April 17, 2025 Published: April 21, 2025

Abstract

Malware is continually evolving, forcing security analysts and researchers to keep pace by improving their cyber defenses. The sophistication and diversity of malicious software presents significant challenges to protecting computer systems and networks against intrusion. The increase of malicious software has been amplified because of the adoption of obfuscation techniques, which are employed to elude detection and conceal its underlying purpose. Antivirus software uses different techniques that are insufficient for detecting metamorphic malware because they can change the internal structure of the code while keeping the same functionality. Most organizations completely rely on commercial antivirus software that uses signature-based detection techniques to find any vulnerabilities. Code obfuscation techniques can successfully evade the previous technique. To overcome the previous limitations of traditional antivirus engines and keep up with new cyberattacks and variants of malware, researchers began adopting machine learning to enhance their approaches because machine learning is well suited for processing huge amounts of data. This paper presents a novel approach to detect metamorphic malware efficiently, we propose an efficient model for metamorphic malware detection by using seven different machine learning methods such as support vector machines, decision trees, random forests, naive bays, k-nearest neighbours, logistic regression, and stochastic gradient descent for the classification. We also propose a feature selection method to enhance the classification and detection of metamorphic malware, based on opcode and API call features, and analyse the detection accuracy result to understand the feature impact on metamorphic malware detection. Our model achieved a high accuracy rate for all datasets using decision trees and random forest classifiers. We also discovered that dominant features play a significant role in improving detection rates.

Keywords: Metamorphic Malware; Malware Detection; Machine Learning; Dynamic Analysis

Introduction

The constant growth of digital transformation in all aspects of life has led to more potential security threats appearing daily, which can easily invade the privacy and integrity of data as well as affect or interrupt system functionality and performance. Malware is one of the most significant and grave hazards to cybersecurity, governance, organizations, and individuals. It is composed of malicious software. Viruses, worms, Trojans, spyware, adware, ransomware, rootkits, keyloggers, and other similar classifications are included [1]. since malware enters a system without the consent of the user and carries out undesirable acts that affect the security principles of confidentiality, availability, and integrity (CIA) and is designed to cause corruption to computer systems and compromise user security, such as stealing data from the host system and corrupting other executable files' code, infecting system files, and writing large amounts of data to memory. Malware targets a variety of platforms, including servers, PCs, mobile phones, and smart devices. However, modern malware is also created for financial gain, to destroy a nation's defenses systems, or to exert political influence. Malware can infect your computer in a variety of ways, including through the download of free or legitimate software from the internet that includes hidden malware, downloading email attachments containing malware, visiting an infected website, clicking off a fake error message that launches malware, etc. [2] The statistical analysis from AV-Test shows a huge growth in malware every day. In 2022, 96,516.839 new malware pieces were created.

Malware detection and classification have become crucial research

areas in cyber security because new malware generations have used obfuscation and evasion techniques to avoid detection and thus produce more sophisticated dynamic malware, such as metamorphic malware. [2] Metamorphic malware is capable of mutating itself with each infection into a new form to hide the malicious code without changing its functionality by using obfuscation techniques. Code obfuscation can be accomplished by dead-code insertion, register exchanging, instruction permutation and reordering, and instruction replacement. [3] We will discuss this obfuscation technique in Section III. A variety of methodologies exist to detect malware, encompassing signature-based, behaviour-based, heuristic-based, normalization-based, and machine learning-based approaches. Signature-based detection is a prevalent method utilized by commercial products for malware detection. However, it lacks the capability to identify novel, unidentified, and sophisticated malware, including metamorphic malware, which leads to zero-day attacks. Most organizations completely rely on commercial antivirus software that uses signature-based detection technology to find any vulnerabilities or intrusions. This weakens organizations' defenses against metamorphic malware, which is a strong motivation for study and development into their detection [4].

Due to the increase in sophisticated malware and the legacy of signature-based detection, it is required to use an automated approach to reduce human involvement and enhance the accuracy of the detection method. We suggest machine learning (ML), which is commonly used to effectively classify and detect malware as well as spam mail [5]. Furthermore, it is very successful because it can deal with significant volumes of data, such as API calls, assembly code (Opcode), and byte code, which humans find unacceptable. Based on a variety of types of data that affect the overall detection and classification performance, ML approaches include support vector machines (SVM), naive bayes (NB), and decision trees (DT).

In this research, we aim to enhance and increase the accuracy of detecting metamorphic malware and future mutations. The primary concept is to utilize a machine learning model that uses a large dataset to ensure the accuracy rate of the selected methodology. The subsequent sections of the paper are structured in the following: Section II discusses the background concepts, and structure of metamorphic malware. Section III presents a metamorphic malware analysis. Section IV discusses metamorphic malware detection techniques. Section V studies the related works. Section VI presents the research methodology (the machine learning model). Section VII presents the research experiments and evaluation. Section VIII covers the conclusion and recommendations for future work.

Background

Malware Overview

Malware is malicious software. It is like any other software application but has malicious intentions. Malware comes in a variety of simple and complicated forms, including polymorphic and metamorphic malware. Malware can execute itself locally or be remotely managed through the Internet. Malware has a wide range of actions it can conduct to achieve its objectives. These actions could be described as the following: corrupt, access, or delete file activity, registry activity, run or modify undesirable service activity, mutex activity, process activity, runtime DLL activity, and network activity [4].

Malware can be classified into two distinct categories: first-generation and second-generation. In the first generation, the malware's structure is unchanged, but in the second generation, each malware variation has a different internal structure while still performing the same behaviours. Second-generation malware is further divided into encrypted (packing), oligomorphic, polymorphic, and metamorphic malware based on how variations in malware are generated [1].

Metamorphic malware is the most challenging threat in the cyber security world; it is highly sophisticated and reduces the importance of signature-based detection. Metamorphic malware exhibits body-polymorphism, wherein it generates a new form while maintaining its original functionality, instead of generating a new Decryptor. Just like polymorphic malware, obfuscation techniques can be employed to generate new iterations. We discuss the details of metamorphic malware, obfuscation techniques, and how to detect this type of malware [1].

Metamorphic Malware Structure and Behaviour

Metamorphic malware acts as though it automatically mutates itself each time it propagates or is distributed to a new host. It modifies its syntax or structure during each propagation to avoid signature-based detection while preserving its malicious functionality. Although its functionality is semantically identical, it modifies its code using semantics-preserving transformations to make one malware variant look very different from another. To prevent the same detection from being effective on all mutated variants, Metamorphic malware possesses an 80% morphing engine, which functions by taking the 20% malicious code as input and dynamically altering it during runtime to produce a syntactically distinct yet semantically comparable version [1].

Two copies of the same malware can be significantly different from each other by repeatedly obfuscating, and most detection techniques are unable to fully recognize metamorphic malware. Obfuscation techniques such as dead code insertion, register substitution, instruction replacement, instruction permutation, and code reordering/transposition. In 2000, the Win32/Ghost virus was created with 362,880 different variants [1]. One of the strongest metamorphic malwares, W32/NGVCK, was created in 2001 with the help of the Next Generation Virus Creation Kit (NGVCK) [2].

The components of a metamorphic engine are a disassembler, a code analyser, a code transformer, and an assembler. When the virus finds the location of its code, it requires converting the code into assembly instructions, which is handled by an internal disassembler. The code analyser is responsible for providing essential information for the code transformer module. The code transformer needs some essential information, such as the structure and flow diagram of the program, subroutines, duration of variables and registers, and so on. This information helps the code transformer work appropriately. The code transformer, or obfuscator, is the brain of the mutation engine. It is responsible for obfuscating the code and modifying the binary sequence of the virus. The other modules are designed to provide the requirements of the obfuscation module. It may use several obfuscation techniques and finally convert the newly produced mutated assembly code of the virus into machine binary code by the last module, Assembler [2]. Metamorphic malware cannot be detected with signature-based detection because professional metamorphic malware can create an infinite number of variants that behave similarly and do not have a single pattern used to detect them. Therefore, to detect powerful metamorphic infections, antivirus scanning engines must adopt highly developed heuristics and behaviour-based detection methods [3] (Figure 1).

Obfuscation Techniques

The obfuscation technique makes malware difficult to detect. It changes a malicious code to a new different version while preserving the same functionality. The primary purpose of this technique was to protect the intellectual property of software developers. Later, malware developers exploited this technique to hamper detection and analysis, Polymorphic and metamorphic malware use this technique. Metamorphic malware uses several different techniques to evade detection and make it more difficult to be analysed and understand [22], it changes its codes into new generations but keeps the same malicious functionality [4].

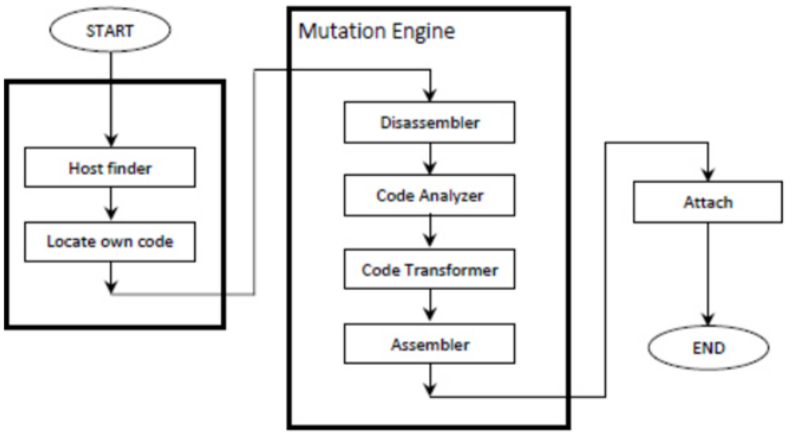

Figure 1: Assembly of replicator and mutation engine in Metamorphic virus [3].

Obfuscation adds unnecessary instructions or garbage/dead to an existing code to change the structure while keeping the same behaviour [7]. The binary sequence of a malicious code is changed without affecting the original functionality. Obfuscation techniques will be discussed below in detail [8]:

Dead Code Insertion/ Garbage code insertion

The insertion of dead code or garbage code is the simplest way to modify the binary sequence of the virus program without affecting the functionality or behaviour of the code but confusing and overloading the emulator during code analysis [9], such as instruction NOP does not have any functionality, the instructions do not alter the content of CPU registers or memory There are several types of dead codes.

Register Swapping/ Register Exchanging

The virus instruction operands are saved in various registers in this method for each new infection, it replaces another unused register for the use of a register in an instruction. It implies switching registers or memory variables of various malware variants and changing the binary sequence of the code, this does not affect a program's behaviour, but it evades signature-based detection [7].

Instruction Replacement/Substitution

In this obfuscating technique, instruction is replaced with equivalent instruction by keeping the same functionality but changing the code with a library of equivalent instructions. The branch conditions can be reversed, register moves can be replaced by push/pop sequences, alternate opcode encoding, and xor/sub and or/test interchanging are examples of instructions that can be substituted.

Instruction Permutation/Reordering

In this method reorder the sequence of instructions, without changing the result. Through this rearranging process, binary sequences of the code look different in various generations [4].

Code reordering

This method restructures the sequences of binary code. It accomplishes this by employing sophisticated transfer of control or jump instructions as the foundation for obfuscation, while preserving functionality and restoring the original program execution flow via unconditional or conditional branches. The reordering may be executed at the level of individual instructions or entire blocks of code. [3] [7].

Metamorphic Malware Analysis

The impact of malware analysis and the nature of the data involved have become increasingly influential in the process of detection. This impact extends to the classification of files during investigations, thus affecting the overall accuracy of detection models. Malware analysis is an important process of malware detection to understand the malware structure, its main characteristics, and functionality, Different essential elements are discovered during analysis and reveal information about malware functionality. Before developing effective detection systems, and must be performed analysis malware first [2]. There are three primary methods of analysis: static, dynamic, and hybrid. Various forms of data have been retrieved through the utilization of static, dynamic, and hybrid analysis techniques. These include Byte code, Opcode, API calls, file data, registry data, and other relevant data sources. The objective of this analysis is to gain a comprehensive understanding of the files under examination and afterwards classify them as either malware or benign files based on their primary function [10].

Static Analysis

Static analysis is the process that malicious files can be identified without actually executing on the device. Because malware is not run, static analysis is safer than dynamic analysis. It is divided into two categories, including basic static analysis and advanced static analysis. Basic static analysis reveals the malicious program's version, file format, and any suspicious imports, in addition to other basic information. Basic static analysis is quick and easy, but ineffective since it can lose important information. Advanced static analysis deals with structure analysis, which requires an understanding of operating system principles, assembly language, and compiler code. Examining the malware's internal code allows it to analyse the functionality of the malware. Through this analysis, information can be obtained about the identification of malware, passwords, libraries, URLs, and programming languages. Mutants and function routines can be found. The code may be disassembled and decompiled using advanced static analysis. Static analysis, however, is unable to deal with packing and obfuscation. Although it indicates packing, the binary must be unpacked for static analysis to be successful [8].

When conducting static analysis on binary executables or source code, a wide range of static data can be gathered. This includes data from the portable executable header (PE-header) as well as derived data such as string-based entropy and compression ratio. Furthermore, researchers have the option to utilize other tools, like the IDA Pro disassembler and Python-based modules, for the purpose of gathering static opcode and API call data. Despite the ability of static analysis to trace all potential execution pathways, it is susceptible to the influence of packing and encryption methodologies [4]. There are two main advantages of static analysis. First, since malware doesn't need to be executed during analysis, it is safe. Second, it gives more detailed information about malware's execution paths.

Dynamic Analysis

Dynamic analysis is s carried out in a virtual environment to prevent the malware from actually infecting computer systems. Dynamic an-

alysis involves running a malicious program and observing its runtime characteristics before analysing it. Malicious behaviours are observed and recorded. The malware unpacks itself during this time, and modifications it makes to the system are also observable. the basic dynamic analysis is observing the basic behaviours that malware executes, such as the creation of new processes, file activities, or registry activities [2].

On the other hand, advanced dynamic analysis thoroughly examines the internal state of a malicious program that is now executing. It performs a thorough internal inspection to get more detailed information on the malicious behaviours and employs advanced debugging techniques to single-step through the infected code. Any hidden code obtained by packing is disclosed during runtime code analysis. A malware's identity is dynamically determined. Function calls, parameter analysis, and information flow are all depicted. The activities of files, processes and dynamic link libraries (DLL) are disclosed. The dynamic analysis aids in the detection of packing and obfuscation properties as well as metamorphic mutations. It is possible to perform memory analysis. Examined is how malware interacts with the file system, processes, and network [4]. A variety of data can be gathered through the implementation of a dynamic analysis methodology. Malicious activities can be depicted through the analysis of executable file behaviour and the preservation of memory images during runtime. The identification of executable file behaviours involves the collection of executed API calls, machine actions, file-related data, as well as registry and network data. Opcode-based memory pictures have the potential to serve as dynamic representations of harmful operations. When dynamically conducted, the analysis of obfuscated malware reveals its activities, although dynamic analysis fails to meet all harmful circumstances necessary for detecting all execution routes [10].

There is a list of tools for dynamically analysing malware and performing advanced and specific monitoring of some functionalities including, Process Monitor, Process Explorer, Regshot, NetCat, Wireshark, OllyDbg, etc [9]. Advantages of dynamic analysis include handling packed files, analysing massive malware corpora automatically, and investigating real-time malware behaviour. The disadvantages of dynamic analysis include, it costs a lot to compute and uses a lot of system resources, the potential for missing out on some execution paths when malware under monitoring goes dormant, the risk of a network-capable malware infecting the host system from the virtual environment, and the difficulty of monitoring malware that can refuse to execute when run in a virtual environment [1].

Hyper Analysis

When malware has more complex code, basic static analysis is inefficient, and complex malware can occasionally evade detection by sandbox technology. The best of both methods can be achieved by combining the two different malware analysis methodologies. In addition to extracting many more features from statically generated and previously unobserved code, hybrid analysis can find hidden malicious code. It can identify unknown threats, including those coming from the most advanced malware.Data extracted using static and dynamic analysis were combined to minimize the limitations of each analysis method and increase detection rates. Different tools, including Cuckoo Sandbox, IDA Pro Disassembler, and Olley'd, are used to gather dynamic and static data, and then hybrid feature sets are built based on several types of data, such as strings, opcodes, API calls, and others [10].

Metamorphic Malware Detection Techniques

Advanced malware developers deploy a variety of obfuscation strategies and techniques to avoid detection. As previously mentioned in this paper, code packing, polymorphism, and metamorphism are the most commonly used obfuscation techniques. The process of detecting malware is the mechanism that needs to be implemented to uncover and identify the malicious activities of the files under investigation. As a result, many malware detection methods are improving year after year without a unique approach, which does not provide 100% success with all types of malware and families in all situations. Malware, particularly metamorphic malware, is notoriously difficult to detect. By constantly mutating its internal structure while remaining inside an infected system, it poses an interesting threat. As a consequence, the malware was discovered utilizing four malware detection techniques based on signatures, behaviours, heuristics, and newly developed machine learning, which were based on the two primary characteristics of malware: signatures and behaviours. Therefore, developing a reliable method for detecting metamorphic malware is essential. The methods for malware detection are discussed in the sections that follow [1].

Signature-based detection

Signature detection is the most common technique in virus protection software. Signature detection is based on unique signature pattern matching, in which a sequence of bits against malware is stored in a malware database, signatures pattern stored in the database has to be previously extracted to compare the given testing files' signatures, to an updated database of signatures and make a final decision based on the matching state. When an anti-virus scans the system, it looks for programs with signatures in the malware database. If a file is matched with the files in the malware database, it is marked as malware, too, and thus used later for malware signature matching. This approach cannot detect malicious files for which no signatures have been recorded yet. It is not capable of detecting zero-day attacks and sophisticated malware like metamorphic and is readily overcome by straightforward code obfuscation techniques. Because of their low false positive rate and low processing cost, the majority of antivirus programs employ this strategy [8]. The advantages of using signature-based detection are speed and accuracy. Along with this, the disadvantage of using signature detection is Any new malware would never be detected by these static detectors. Thus, the malware database needs to be constantly updated. Even a small obfuscation will cause this detection technique to break. Thus, dynamic analysis is required for the virus's dynamic behaviours [1].

Anomaly-based detection

This detection approach is based on monitoring and analysing behaviours during the runtime of malware in a controlled environment (a virtual machine or sandbox), then determining if the given file is following normal behaviour or not [10]. It overcomes the limits of signature-based detection by using heuristic approaches to detect normal behaviour. After monitoring the executable files in an isolated environment and collecting the observed behaviours, features extraction techniques were developed to extract the sensitive features that will allow the developed model to classify the known malicious behaviours as well as any behaviour that appears to be similar to them in terms of false positive behaviours. The ability to identify novel malware behaviours in addition to known ones using run-time behaviour collection has made this approach more valuable than signature-based approaches. As a result, the majority of the studies in the review focused on using behavioural-based approaches to increase malware detection ratios in the form of continuous, sequential, and common behaviours. Any file that does not classify as normal is then classified as malicious software. The definition of anomalous and normal is defined by the user, so the classification is not very accurate. A malware detector has been developed for classifying input files based on file structure. A test file requiring classification is provided for detector input. The file is classified as abnormal if it appears to be, otherwise, it is marked as benign. After classification, it is examined to determine whether it was a malicious malware file or simply a false positive. Detecting an anomaly can have a large number of false positives or false negatives. Therefore, anomaly detection is used with signature detection to provide greater accuracy [1].

Heuristic-based detection

This method has been used in many studies to support the model-

ling of identifying and detecting malicious malware through the establishment of generic rules that use extracted data that is provided by dynamic or static analysis as rule input. There is no single type of data that is commonly used with this approach, but the researchers have mostly used all types of data at the same rate, including API calls, network data, registry data, imported DLL, and others. In addition, the creation of generic rules plays a major role in the final discussion of malware detection and classification [11]. A heuristic-based approach has two approaches to detecting malicious software. Firstly, in the static approach, a suspicious program is disassembled to find a match with the known malware pattern, if any. If the result of the scan exceeds the preset threshold, then the program is labelled as infected. Secondly, in the dynamic approach, code emulation techniques are used by simulating the processor and operating system to detect suspicious operations (an attempt to open other executable files to modify its content, changing the Master Boot Record, concealing themselves from the operating system, etc.) on a virtual machine. The resulting rules of the two approaches can be created manually according to the expertise and experience of professional analysts or by automatically using machine learning techniques, the YARA tool, and other technologies based on expert knowledge of analysis. Several studies have been done to develop malware detection models by which decisions have been taken based on the automated behavioural rules that are created using machine learning techniques and the YARA tool. On the other hand, based on statically extracted string data [10].

Machine learning detection

Malware detection using machine learning techniques has grown in popularity in recent years and is commonly used for the detection and classification of malware as in [12]. This method is very successful because it can deal with massive amounts of data, such as Application Programming Interface (API) calls, assembly code (Opcode), and byte code, which is unacceptable for humans. ML techniques provide a high degree of generality, have become an active domain in the field of cybersecurity, and have an impact on high detection and classification accuracy. Naive Bayes, decision trees, data mining, neural networks, vector machines, and hidden Markov models are popular machine learning techniques among researchers for detecting 2nd generation malware. This technique is not intended to replace standard detection methods but rather to supplement them [10] Robert Moskov itch et al. proposed malware detection based on computer behaviour monitoring (features). His evaluation results show that using a classification algorithm on only 20 features resulted in a mean detection accuracy of more than 90% [11]. The benefit of machine learning techniques is that they will not only detect known malware but will also serve as knowledge for the detection of new malware. Machine learning techniques are generally more computationally demanding than standard anti-malware, so they may not be suitable for end users. It can, however, be implemented at the enterprise gateway level as a central anti-malware engine to supplement anti-malware. Although infrastructure is expensive, it can help protect valuable enterprise data from security threats and prevent massive financial losses [10].

Normalization

Malware created by advanced toolkits like UPX and Mitsfall is difficult to detect. For the detection of such malware, normalization techniques can be used to improve the detection rate of an existing anti-malware program. In this technique, the normalizer accepts the obfuscated version of the malware, removes the obfuscation from the program, and generates the normalized executable. Following normalization, the signature of the malware is extracted and compared to the signature of the canonical form [11]. The malware normalizer algorithm operates as shown in Figure 2. First, it decompresses the malware PE binary code, then disassembles the compressed PE code, and finally, the normalizer checks for and eliminates obfuscation performed on the file, producing normalized code. After that, the malware detector will extract the created normalized code's signature and compare it to the signature stored in the signature database. The new signature of malware in the canonical form is stored to avoid future compromise [13]. Recently, a general malware normalizer with a detection rate of up to 81% that can store a large number of obfuscation methods in the form of automata structures was proposed [10].

Related Work

In recent years, the number of research projects on machine learning detection algorithms for detecting evasion-type malware has increased rapidly. Vivekanand et al. [14] propose a model for detecting polymorphic and metamorphic malware through a deeper examination of API calls, significant features, and their parameters that permit polymorphism in malware to handle the detection and classification challenges. Their work focuses on behavioural (dynamic) feature analysis and APIs, and it also proposes a feature engineering approach for the improved classification of malware families. For classification, they used eight different malware family types. There were two modules created. The first module provides detection information about submitted files to identify whether they are malicious or benign, while the second module is used to classify identified malware according to its family. A file is first submitted for dynamic analysis in the cuckoo sandbox. It is a virtual environment for analysing malware. to extract every call and parameter made using the accessible API (Application Program Interface). According to how they work, APIs are divided into seven categories: register, file, network, services, synchronization, system, and process. Every sample of malware and benign file has a unique collection of API traces that are saved in a dataset as an API trace. After applying feature engineering to the dataset, two datasets were created: one for malware classification and the other for malware detection. Dataset 1 includes API calls as a feature and their associated parameters for both malicious and benign samples. 9995 malicious and 9995 benign samples totalled 594 features employed to detect malware. Dataset 2 includes eight different types of API calls from malware. The malware families that were rated from 0 to 7 were worms, trojans, advanced persistence threats (ATP), crypto-malware, Zeus, downloaders, backdoors, and viruses. It has 1859 columns and 6396 rows. To establish which family of malware the given hash value of the malware belongs to, they used the Virus Total service to produce this dataset. The top 30 features were employed in this model for training and validation after feature engineering was applied to the dataset, giving each feature a value to indicate its significance. These features were noted by the Extra Trees Classifier. Eight families of malware were employed in dataset 2. They used SMOTE (Synthetic Minority Oversampling Technique) to balance the malware class. These malware families were labelled from 0 to 7. Because this dataset is unbalanced, it will have an impact on the model's accuracy and predictability. To comprehend, explain, and justify the connections between features and the anticipated class, they used Exploratory Data Analysis (EDA) for quantitative analysis. They observed based on this link to improve understanding of the issue and develop new solutions. Every dataset contains features that are unrelated to one another, which can affect the model's accuracy. EDA is used in this model to remove this kind of feature. The researchers focused on malware that runs on Windows and utilized machine learning to examine its behaviour. From the zoo, malware samples are downloaded. Malware samples contain metamorphic and polymorphic. The virtual environment's API and parameters were all dynamically extracted. After applying feature engineering to dataset 1, the model was trained on 80% of the data with eight types of machine learning algorithms. Random Forest had a higher accuracy of 98.74 percent, detecting malware in 2005 out of 2032 samples. Dataset 2 was used to classify malware that is metamorphic and polymorphic using a second model. It includes a few families of malware that use evasion techniques for classification. 1859 features in total are present in dataset 2. Following feature engineering and data balance with SMOTE, this model was trained and tested with SVM, KNN, and RF. Random Forest provided the best performance, with an accuracy of 96%. utilizing feature engineering and dynamic feature analysis.

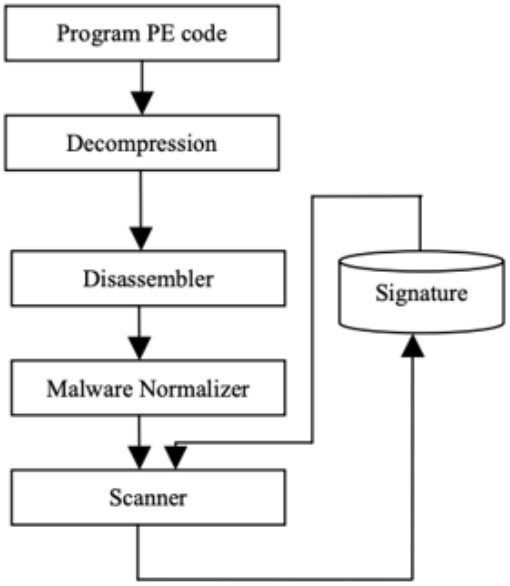

Figure 2: Malware Normalization and signature comparison [11].

The main drawback of their work is that only dynamic analysis was employed to detect malware for Windows 7 and that there were not many samples of polymorphic and metamorphic malware used for classification. If a PE file does not contact an API and instead accesses an OS resource directly, then the proposed method is invalid. Future advancements will include the use of a hybrid method for malware detection. Researchers have found that API calls may effectively identify malware behaviour and can be used in conjunction with machine learning algorithms to efficiently detect malware.

Namita et al [22] proposed a model to detect malware on the Windows platform based on machine learning and API calls. Virus Share data repository was used to retrieve malware samples, which totaled 2500 malicious samples and 2500 benign samples for Windows 10. The cuckoo sandbox environment was used to execute these samples and extract their behavioural reports. These reports comprise information from performed samples in the form of API calls, such as network communication, process, registry, and file system actions. API calls provide a great representation of how a program behaves when it is being executed. The data acquired from API calls are used to extract three different feature sets: API call usage, API call frequency, and API call sequences. Usage, frequency, and sequences of API calls are essential aspects in determining a sample's behaviour. All of these aspects were combined to create an API-integrated feature set, which is a more useful feature set. TF-IDF assesses the relevance of each API call feature included in the integrated API call feature set. A good classification model is built on robust and representative features. These features offer critical information for building a powerful and effective machine-learning model that can accurately identify malicious and benign programs. It is important to eliminate features from the feature set that is unnecessary or irrelevant. to only display informational features that will improve the detection method's accuracy and extract important features while reducing the feature dimension space. An API-integrated feature set is utilized in conjunction with feature selection techniques. Several feature selection techniques are categorized into three groups: filter, wrapper, and hybrid methods. The suggested malware detection model is developed using machine learning algorithms including Decision Tree, SVM, Logistic Regression, and k-Nearest Neighbour. Detection using Machine Learning Algorithms and API Calls Usage Feature Set with DT, SVM, LR, and kNN algorithms, it was found that the DT and KNN models performed well, but the SVM and LR models performed poorly, with high false positives and false negatives. When using the API frequency feature set rather than the API usage feature set, all machine learning algorithms performed better. With this feature set, the SVM algorithm had a maximum accuracy of 98.8% with few false positives and negatives, which had an impact on the method's accuracy. Also, detection with API call sequences showed that all algorithms achieved 96.5% accuracy or higher with very low false positive and false negative rates.

The API call sequence feature set outperformed the API usage and API frequency feature sets in terms of performance. Finally, the proposed method detection performance with an integrated API feature set is evaluated. The results show how well machine learning algorithms with an integrated feature set from APIs perform classification. It is clear from the data that, when using this feature set, all algorithms achieved 99.6% accuracy. Furthermore, this feature set significantly reduced the occurrence of false positives and negatives. Moreover, this feature set beats other feature sets for API calls in terms of detection performance. Due to its advantages over other feature sets, the API-integrated feature set has since been used for subsequent trials and results. This method's ability to find malware can be improved by adding more families of samples and their different versions.

Mohammad Ali et al [15] designed a classification system approach to identify malware based on N-grams and machine learning and developed an efficient feature extraction and representation algorithm that may be used to enhance the approach. In this study, they used a small, structured, and labelled dataset from Virus Hare that had 60 harmless and 60 harmful samples of both polymorphic and metamorphic malware. The proposed approach contains three stages, monitoring, feature engineering, and learning and verification. In the first stage, a data corpus was evaluated using an artificial intelligence-based sandbox (SANDBOX) to create behaviour reports containing malicious file artifacts. The second stage which is the most crucial step in designing an efficient malware detection approach is feature engineering. The researchers proposed two scenarios for N-gram feature creation and extraction: In scenario 1, the researchers utilized API calls and the memory location of their arguments to construct valid N-grams; in scenario 2, they used function calls and the address of their arguments to construct N-grams.

The objective of using two different approaches setting to extract N-gram features is to find out which way helps with accurate classification. For both scenarios, API-N-grams with n = (1, 6) were produced and then utilized to construct a feature vector. To minimize the feature space, they sorted each set of N-grams based on their frequency of occurrence and eliminated the lowest-frequency grams. The Term Frequency–Inverse Document Frequency (TF-IDF) algorithm was

used to produce the effective feature set. It was used to calculate the feature weighting and figure out which feature set was the most accurate. In the last stage, N-gram feature sets were transformed into binary vectors that would be utilized for training and testing by machine learning algorithms. In this context, four learning algorithms, including logistic regression (LR), random forest (RF), decision tree (DT), and Naive Bayes, were utilized to evaluate the performance of the proposed system approach: logistic regression (LR), random forest (RF), and decision tree (DT) (NB). In comparison to other learning methods, logistic regression gave the highest classification accuracy, 98.43% for scenario 1 and 84.5% for scenario 2. The researchers intended to do a future study on the effectiveness of N-gram analysis in detecting malware. In addition, they intend to acquire enormous databases pertaining to various types of malwares. In future research, they intend to extend the number of features, including API calls, registry values, DNS queries, HTTPS requests, and system modifications, to train and evaluate them with deep learning algorithms.

Saima et al [16] proposed an effective malware detection technique that uses features from the executable file's PE header and Section table to classify malware families. The PE file format is the primary executable file format for Windows operating systems. The executable files are disassembled using the IDA-Pro disassembler for debugging and disassembling reasons. It is Microsoft-centric and includes DLLs. Binary instructions are transformed into higher-level structures and code mnemonics, from which desirable and significant features are extracted. The proposed model was created using a static analysis approach. It detects malware before the executable file is executed. By examining the extracted features of the executable file, malware is analysed. Some research relies on prior information from the PE file header for feature extraction. They took the PE file header as a whole to analyse the characteristics of the executable files. Techniques like n-datagram, grayscale, and many others are utilized for feature extraction. The classifier that assists in determining if the file is malicious or not is built using these features. They extracted 1340 benign executable (.exe and.dll) files from the System32 folders of Windows 7 and Windows 8.1, as well as 1530 malicious executable files from the Virus Share and VxHaven websites. The virus total online program was used to scan executable samples for the presence of worms, viruses, Trojan horses, and other malware. The IDA-Pro disassembler is used to disassemble both malicious and benign samples to examine the function of the portable executable files that have been disassembled. and gathered a significant amount of data from the disassembled executable file, storing different characteristics to comprehend the executable file. The different classification methods of supervised machine learning are utilized to look for patterns and their implications. on the data that has already been assigned a class. classification techniques, including k-Nearest Neighbours, Decision Trees, Random Forests, Naive Bays, and Support Vector machines, are used. Based on the information gathered, after comparing the combined feature set of a file header, optional header, and section header to the feature set that only had an optional header, they were able to get higher detection rates. Performance of classification with optional header features Random Forest and Decision Tree classifiers provide the highest accuracy results when compared to other classifiers.

The Decision Tree classifier's accuracy rate was 97.12%, whereas the Random Forest classifier's accuracy rate was 97.24%, which is slightly higher. We obtained 97.5% precision, 97.9% recall, 97.50% true positive rate, and 3.07% false positive rate with the Random Forest classifier. They merged features from all three headers to increase the accuracy of the results for more accurate malware detection. Classification performance using combination features the best accuracy rate of all the classifiers, 98.63%, was achieved by the Random Forest classifier. We obtained a 98.68% true positive rate and a 1.42% false positive rate with the Random Forest classifier. The scope of the proposed research is the identification of malware in Windows executable files. Future studies should strive to develop some strategies to extract those features and extend the approach to include feature selection in more hybrid scenarios. By addressing the problems with both methods of analysis and creating a model based on hybrid analysis, which combines static and dynamic analysis.

Aakash Wadhwani [17] conducted two kinds of experiments, the first experiment was designed to reduce the accuracy of malware detection by causing code metamorphosis. The second experiment focuses on improving the detection accuracy of the morphing algorithm, which is the main objective of the experiment. In the first experiment, Aakash Wadhwani introduced the theory of a metamorphic engine that transforms JavaScript-based code into a different code form at the Abstract Syntax Tree (ATS). The Abstract Syntax Tree is a hierarchical structure of programming source code. It simplifies code and gives important information about the instructions needed to change the code. The techniques implemented in the morphing contain steps for each source code; the first phase is to transfer code to AST, followed by dead code insertion, instruction reordering, function reordering, and instruction substitution, which change the signature of the original code without changing its functionality. After morphing ATS back to its original code, the result of this would be a morphed code. Then he used the following ML models for the classification, K Nearest Neighbours, Random Forest, Support Vector Machine, and Naive Bayes. The result of the experiment Then, he classified the dataset by using the following ML models: K nearest neighbours, random forest, support vector machine, and naive bayes. The outcome of the experiment is a 21.95% decrease from 95.25 to 73.3 percentage. Maximum accuracy decreases by 37.3% while utilizing K-Nearest Neighbour. In the second experiment, he constructed a morphing code detection by using N-gram and HMM features with several machine learning models including K Nearest Neighbour, SVM, Naive Bayes, and Random Forest. With N-grams, SVM provides the highest detection accuracy of 97%. In the experiment using HMM feature vectors, SVM performs best with 96.8% accuracy, followed by KNN with 96.38% accuracy. His future work is to experiment with Malware called GAN and try detecting Transcriptase malware.

Sanjay et al [18] Introduced an approach based on the opcodes occurrence to enhance the detection capability and accuracy of undiscovered sophisticated malware. The method includes the preparation of a dataset, choosing promising features, training a classifier, and detect advanced malware. first step, Kaggle Microsoft malware dataset have been used in this study by downloaded and collected two type of data benign and malware programs (7212 files) for the windows platform. The datasets used in the method were filtered and cleaned to remove noise, and then the data was prepared by computing the weight of benign and malicious files based on the acceptable assembly code weight, which should be equal to or less than 147.0 MB.

The obstacles posed by the growth dataset used in this study were overcome through two methods: first, selection of instance, and second, selection of feature. In their method, the number of instances (rows) in the dataset was reduced by selecting the most appropriate rows and the most relevant attributes. Feature selection, is used to choose the most relevant properties (features) in a dataset. These two methods are particularly effective at reducing data because they filtered and cleaned up data noisy. This means they take up less space, take less time, and make classifiers work better. The next step is feature selection, which is the most critical and important step in ML for maintaining accuracy. In this study, Fisher Score (FS) was used for feature selection and later studied the following features: information gain (IG), gain ratio (GR), chi-square (CS), and uncertainty symmetry (US).

Based on these feature selection, the top 20 feature have been picked. After the feature selection, the next step is to determine the accurate classifier for the detection of sophisticated malware. They train 9 classifiers available in WEKA GUI on each feature selection (FS, IG, GR, CS, and US) using the top 20 features to determine the most effective

classifier for detecting malware. RF, LMT, NBT, RT and J48 GRAFT had the highest accuracy in detection. These five classifiers were chosen for in-depth analysis. They have randomly selected 3005 malicious and 2286 benign programs, representing 50 percent of the whole dataset. Finally, they provided a technique based on the occurrence of opcodes to enhance the detection rate of suspected sophisticated malware. The proposed method applies the Fisher Score method for feature selection and uses five classifiers to recognize unknown malware. Using the proposed method, LMT, RF, J48 Graft, and NBT can all spot malware with 100% accuracy.

Based on the comparison in Table 1, the proposed solution will detect metamorphic malware, which is considered the most sophisticated malware and the most difficult to detect. Malware can spread rapidly and mutate. It is affecting information integrity, confidentiality, and availability. The biggest issue with metamorphic malware is that it uses several different techniques to evade detection and make it more difficult to analyse and understand. The extracted feature in previous related work and studies was based on static or dynamic analysis. Also, it has limitations in the study of malware behaviour, and some of them use small samples or old ones. In this work, we propose a method for detecting metamorphic malware using an opcode and API call features and analyse the detection accuracy result to understand the feature impact on metamorphic malware detection. Then we apply different machine learning algorithms and compare the performance and accuracy of the models to get an effective solution for detecting metamorphic malware. This paper aims to provide research answers to the following questions:

a.What are the most effective machine learning methods for detecting metamorphic malware, and which have the highest accuracy?

b.Does the feature affect and enhance the detection of metamorphic malware?

c.What are the most significant features useful for detecting metamorphic malware?

Research Methodology

In this section, we describe our proposed method. The main goal of this method is to find the most accurate model for detecting metamorphic malware from a set of files and to figure out what the most important features are for detecting malware.

In order to accomplish that, we use two different machine learning technologies to evaluate the accuracy of detection. All machine learning models were built based on a supervised approach that uses a classification algorithm based on binary classification. The first technology is Weka machine learning software, and the second is Google Collab to build machine learning models. The proposed method was performed on two different metamorphic datasets, one based on an opcode and the other on an API call. Operational Code (Opcode) A compilation of machine-language instructions makes up an executable program. These instructions consist of two parts: the operational code and a list of operands. The part of the code known as Opcode defines the operations, but the operatives, or the data to be processed, may also contain additional information about the executable files.

The frequency of opcode occurrences is regarded as a remarkable indicator for differentiating malicious files from benign ones. The first dataset includes opcode as a feature and its parameters with respect to malware and benign samples. The morphs created by a metamorphic engine have certain characteristics in common. Application Programming Interface (API) calls are one of the most reliable methods for detecting evasive malware. It indicates the actual function of executable files during runtime. Sequential analysis of these API calls also shows how they relate to each other in terms of their context.

We used API call sequences as a feature in the second dataset. The most relevant information about API call sequences is also provided by the frequent patterns that are derived from them.

The experiment runs using two datasets, where the research runs in four main steps:

i.sample collection,

ii.feature selection,

iii.splitting the dataset, and

iv.executable file classification, as shown in Figure 3.

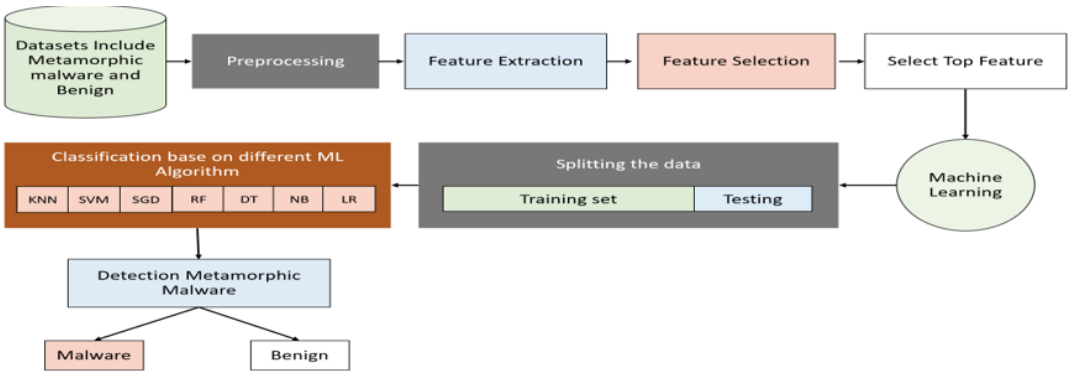

Figure 3: Proposed methodology.

Sample Collection

Dataset 1 consists of five different classes of executable files taken from GitHub [18]. First-class files are benign files with 4783 rows and 1808 columns that have been gathered from various versions of the Microsoft Windows directory and programs in program files. The second class consists of 100 virus samples produced using the "Next Generation Virus Creation Kit." One of the best-known tools for producing metamorphic malware is NGVCK. The third class includes 100 virus samples generated by the "Phalcon-Skims Mass-Produced Code Generator (PS-MPC)." The fourth class has 500 samples created by the "Mass Code Generator (MPCGEN)," while the last class contains 63 examples of viruses created by the "Second Generation Virus Generator (G2)." All of the previous technologies give the code a toolbox that enables it to alter its morphing with each execution by applying obfuscation methods. It should be mentioned that during obfuscation, their functions do not diminish. As a result, the new morphs perform the same functions as previous generations but have a different signature. After that, the benign file was cleaned and normalized. Normalization is a scaling method used to change the values in the numerical columns of a dataset to a standard scale before machine learning. It

is only needed when the ranges of the features in machine learning models are different. that led the data to be organized in a way that was consistent across all records and fields. Moreover, it makes entry types more cohesive, resulting in data cleaning, lead generation, segmentation, and an improvement in data quality. After data cleaning and processing, the final dataset contains 1208 samples and 200 features.

The dataset 2 [21] file is taken from GitHub. These were collected by monitoring the behaviours of the malware and retaining a record of the system calls that each malware executes. The feature consists of the frequency of a potential system call and the time-ordered list of system calls collected from the log by performing the main activity of the malware using specific tools, 251 system calls were taken into account.

Our primary concern in this experiment is metamorphic malware. It is difficult to gather enough samples and their mutations. Therefore, we use a small dataset that includes mutants from Android malware. This dataset consists of two files, each containing benign and malicious samples that were collected from different sources. The first set of data has 100 samples of malware and benign code from the web. These samples come from Droid Kungfu, Doug Alek, and other malware families. The second malware sample is Evolved Malware (EM), generated from the Malgenome Dump and Contagion Minidump variations. This set of samples, which were created using quality-diversity MAP-Elites and classical Evolutionary Algorithm (EA) algorithms, includes three families: Doug Alek, from GGtracker, and Droid Kungfu, on the other hand, are The benign samples are collected from the Google Play Store.

Feature Selection

Feature selection techniques are used to choose a subset of characteristics so that an effective machine learning-based classification model can be made that works well. The foundation of a successful classification model is based on robust and representative features. These features provide critical data for developing a machine-learning model capable of distinguishing between malware and benign files. Unfortunately, some features often add little or nothing to the detection process because they are redundant or have nothing to do with the collection of features. Because of this, it is essential to eliminate them and keep just informative features that will raise the detection method's accuracy. We gather massive amounts of data to train a model so that machine learning can get better. Most of the time, a lot of the data we get is just random noise, and some of the columns in our dataset may not have a big effect on how well our model works. In addition, when there is a large amount of data available, training a model may take longer. Moreover, the model might become erroneous as a result of this useless input. Feature selection methods were generally classified as filter methods, wrapper methods, intrinsic methods, and embedded methods, as shown in Figure 4.

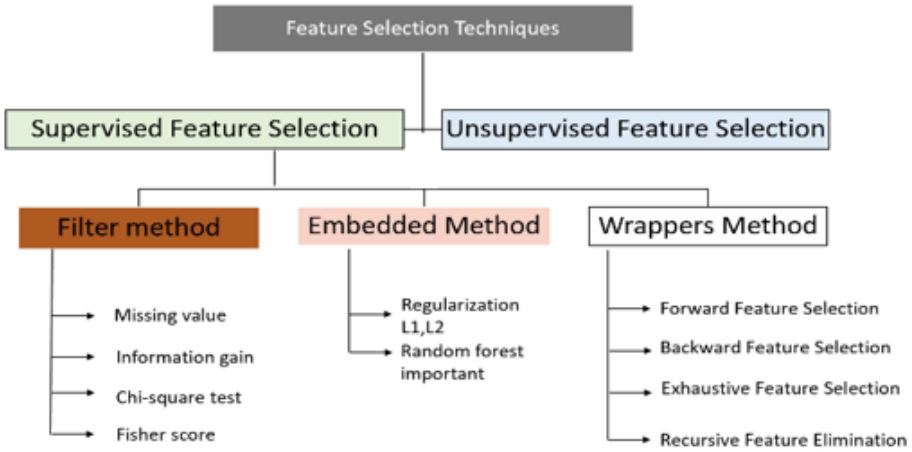

Figure 4: Feature Selection Techniques.

In our proposed method, we use Weka software's filter method and information gain attribute to rank the most important features and Google Collab to find the most important ones. After that, we reduce the number of features and compare all the results to determine the most significant features useful for metamorphic malware detection.

Splitting the Dataset

After the feature selection phase, we used the dataset that included important features to divide it into two sets. For analysing the dataset, we applied the cross-validation method of machine learning and split it into a training set and a test set. Also, we applied for 80% training and 20% testing. The training set makes up the greater portion of the dataset (80%), while the test set makes up the smaller portion (20%). The model is built on the training set and tested on the test set. Now that it has been learned, the model can determine if the executable file being used is malicious or not.

Executable File Classification

In the classification of executable files using data that already has class labels, the various methods of supervised machine learning are utilized to identify patterns and inferences. Using the gathered important features, classification techniques including Decision Tree, Random Forest, k-Nearest Neighbours, Naïve Bays, Logistic Regression, Stochastic Gradient Descent, and Support Vector Machine are used.

In this case, classification techniques are used to sort the data into groups by training a model and feeding new data to the trained model to make predictions. With the help of classifiers as a teacher or way to learn, the model was able to teach itself how to find patterns and inferences for prediction. After the model has been trained, it is checked against testing data to find out how well the learning method that was used to train the data worked. These classifiers are used to build various models. The random forest classifier is used to build the most effective model. Because it consists of different decision trees that offer the best classification results overall, random forest is regarded as the best classifier. Decision trees, a single unit of the random forest, execute dataset splitting in a tree-like structure by running a feature test at each node that optimizes a certain condition.

Analysis Dataset

The relationships between the features and the predicted data were comprehended, defined, and explained through the examination of

datasets. Using this connection, we were able to draw conclusions and learn more about the issue. Every dataset has features that have nothing to do with each other, which can affect how accurate the model is. After selecting the most important characteristic, we applied the model for training and testing.

We split up the first dataset according to the number and importance of features; thus, we used 12 sets with 200 features and reduced them to 5 features. We do the same with the second dataset, but it consists of two files: the first file is divided into three sets, starting with 250 features and reducing to 20, and the second file is divided into three sets, starting with 250 features and reducing to 31, to analyze and compare the accuracy of detection based on the most important feature.

Experimentation And Evaluation

Experimentation and evaluation

In this section, we examine the results of the proposed approach, the experimental setup, various metrics, and the accuracy of the detection and classification of metamorphic malware.

a.The experiments have a dual focus:

b.The first section focuses on identifying the most effective and accurate machine learning methods.

c.The second part concentrates on determining which features are the most significant and useful for metamorphic malware detection.

Experiment Setup

In this research, we focused on two different platforms Windows-based and Android-based malware, and its detection using machine learning. Malware samples are downloaded from GitHub. The malware sample contains metamorphic malware, and we use Windows 10. We used machine learning toolkits, including the Weka GUI software version v3.9.6, which is based on the Java programming language, and Google Colab, a cloud development environment based on Python. A Google Colab is used for training metamorphic malware datasets and finding the important feature of the effective detection result. Weka machine learning software is used for training metamorphic malware datasets and finding the ranking feature that affects the detection result.

Evaluation

In our experiment, evaluation metrics are critical for both measuring classification performance and directing classifier modelling. During the evaluation step, we employ several metrics. They are as follows:

Accuracy: It measures the proportion of accurate predictions to all input samples.

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

Precision: Is measured by dividing the number of correct positive results by the total number of positive results that the classifier predicted.

$$Precision = \frac{TP}{TP + FP}$$

Recall: It is determined by the total of all true positives divided by the total of all actual positives.

$$Recall = \frac{TP}{TP + FN}$$

F-score: It is a measure of precision and recall. If the F1-Score is 1, it means the model is working perfectly.

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

True Positive Rate (TPR) is the rate of samples that contained malware and were correctly identified. The TPR is defined by dividing the total number of malicious executable files.

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (FPR) is the rate of samples that were incorrectly identified as containing malware and did contain malware. The FPR is defined by dividing the total number of benign executable files.

$$FPR = \frac{FP}{FP + TN}$$

True Negative Rate (TNR). The rate of samples that did not contain malware and were not identified.

False Negative Rate (FNR). The rate of samples that contained malware that was not identified.

Result of 1st Experiment

The first experiment on the proposed method for Dataset 1 focuses on identifying the most effective and accurate machine-learning methods that can determine whether an executable is malicious, metamorphic malware, or benign. We found Random Forest, J48 (Decision Tree), and Naive Bayes Achieve the highest accurate machine learning methods for both experiments in Weka and Google Colab with all features, as shown in Figure 5.

In the second experiment, we determined which features are the most significant and useful for malware detection, along with the impact of accurate machine learning methods, and determined the highest one.

After using feature selection on dataset 1, training a model can take longer when there is a lot of data available. Also, this unrelated feature could make the model faulty. Therefore, we reduce the number of features using the filter method to rank features in Weka and select important features that are related to each other in Google Collab to get the best result.

Therefore, we used 12 datasets ranging from 200 features to 5 features. Also, the model is trained in two different ways: first, we train the model with 80% of the training data, and we use cross-validation 10 with 7 different machine learning algorithms such as Decision Tree (DT, J48), Random Forest (RF), k-Nearest Neighbours (KNN), Naive Bays (NB), Stochastic Gradient Descent (SGD), Support Vector Machine (SVM), and Logistic Regression (LR). Random Forest gave a higher accuracy of 100% for all datasets with different numbers of features using different machine learning tools (Weka and Google Collab), as we see in the table.

We achieved those detection rates with 200 features until 60 features gave the same result, and from 40 features to 20 features, the detection rate decreased a little, but with five features, it decreased even more. We recommend using approximately 20–30 features related to each other to get a good result while reducing time and resources.

The experiment on Google Collab using important feature selection shows that the dataset contains 200 features, which were reduced to 60 features while maintaining the same detection accuracy. We train 80% and test the other 20%, as shown below: Random Forest, Logis-

tic Regression, Decision Tree Classifier (J48), Naive Bayes, and Stochastic Gradient Descent to achieve 100% of malware detection, In addition, we measured precision, recall, f-measure, TPR, FPR, TNR, and FNR as shown in Tables 2. We observe that while reducing the feature based on important feature selection on Google Collab, the accuracy of the machine learning method is affected directly, and once it starts reducing the accuracy number as illustrated by the evaluation matrix, the FP Rate starts increasing, which is the number of benign executables misclassified as malware as shown in Tables 3 and 4 In the case of 10- and 5-feature selection, the evaluation matrix shows how the features play critical roles in reducing the accuracy of the machine learning method, as shown in Tables 5 and 6.

Below are the experimental results for all 12 features of the dataset by using ranking feature selection and all other methods using Weka software. The first experiment used 80% training and 20% testing as shown below in Table 7, and the second experiment used all 200 features with 10% cross-validation as shown in (Figure 6-8) (Table 8).

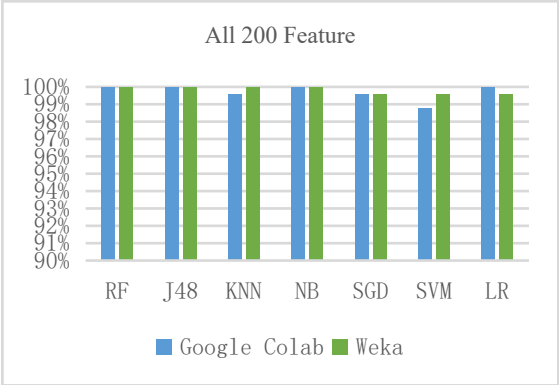

Figure 5: All Feature.

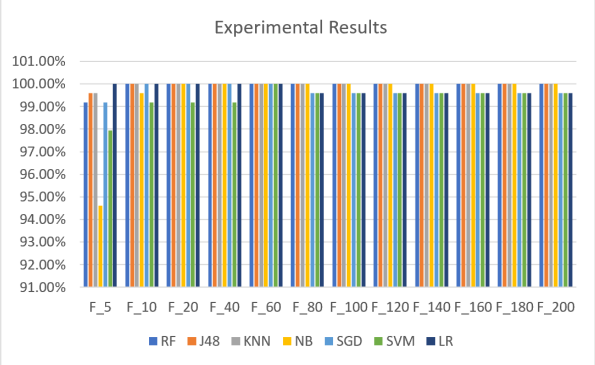

Figure 6: Experimental Result using 80% training.

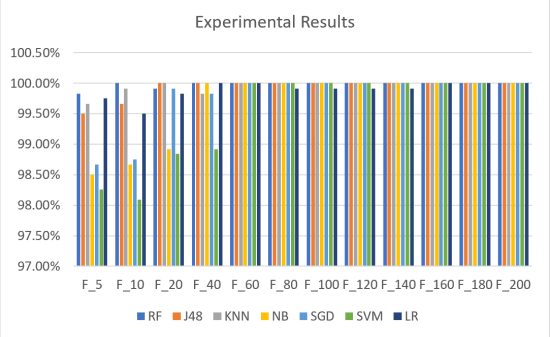

Figure 7: Experimental Result using cross-validation 10 training.

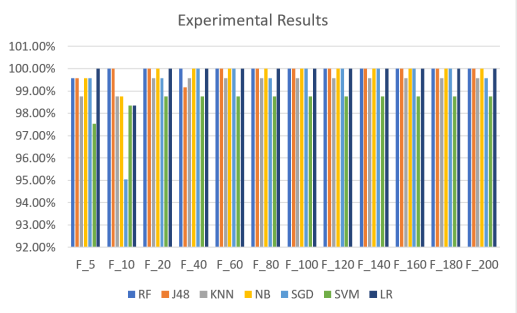

Figure 8: Experimental Results from Google Colab.

Result of 2nd Experiment

The first experiment on the proposed method for Dataset 2 based on Android malware, which consists of two files, the first one collected from the malware web and the second one generated evolved malware from different algorithms, focuses on identifying the most effective and accurate machine-learning methods that can determine whether an executable is malicious metamorphic malware or benign.

After performing our proposed method on the first set of "Malware Web", we observed that the training model through cross-validation decreased RF detection accuracy by -3% while LR increased accuracy by +6%, as shown below in Tables 10 and 11. Detection accuracy for all features using Cross validation 10 training in Weka software.

In a Google Collab experiment on malware web sets, we observed detection accuracy with 250 features remaining the same as 36 features after employing the Important Feature Method. We discovered that Dataset 2 contains approximately 214 features that are useless for detection and training models while consuming time and resources. Random forest, decision tree, and naive bayes are the ones that have gotten the best results in terms of accuracy. As shown below in Tables 12 and 13 (figure 9).

In the second experiment on the proposed method for dataset 2, which is evolved mutants, we observed that Weka ML software gives us high accuracy in most ML model detection of metamorphic malware from the malware web. In addition, cross-validation reduces the detection accuracy in both data set files in dataset 2, as shown below in Tables 14 and 15.

In a Google Collab experiment on evolved mutant sets, important feature selection shows that the dataset contains 250 features, which were reduced to 31 features while maintaining a similar detection accuracy rate, except for NB, which was reduced to 95%. As shown below in (Tables 16-18) (Figure 10)

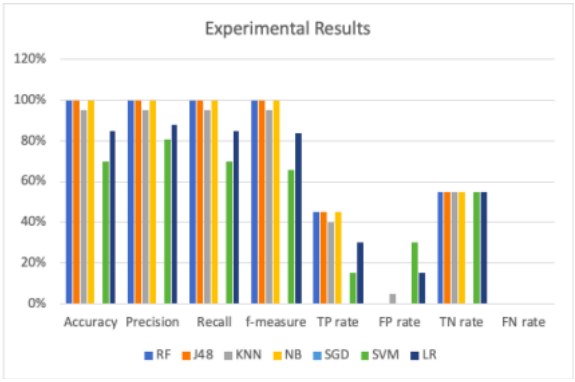

Figure 9: Experimental Results for dataset2-file 1 from Google Colab.

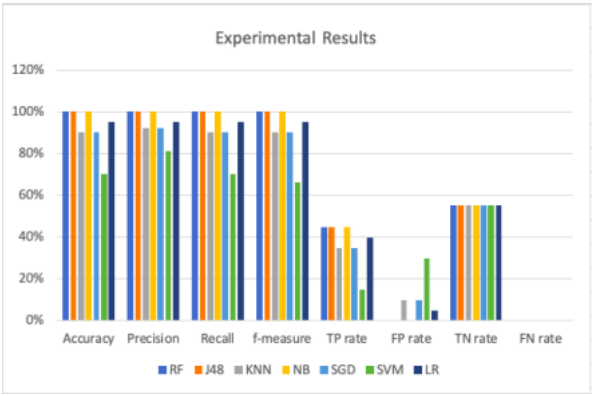

Figure 10: Experimental Results for dataset2-file 2 from Google Colab.

Discussion

This subsection compares the detection rate accuracy of the proposed metamorphic malware detection method with other research presented in related work [14-18, 22]. Vivekanand et al. [14] presented a model that achieved a detection accuracy rate of 98.74 percent with Random Forest, and the classification model gave an accuracy of 96% using the Random Forest classifier through API call features. Namita et al. [22] presented a model that achieved a detection accuracy rate of 99.9% with an API-integrated feature set using hybrid feature selection. Mohammad Ali et al. [15] proposed method achieved that NB and LR gave the highest classification accuracy, 98.43% for scenario 1 and 84.5% for scenario 2, based on N-gram features. Saima et al. [16] introduced a model based on static analysis that extracted the features from the PE header of the executable file; when using only the optional header, they achieved the highest accuracy of 97.24% to classify malware with a random forest classifier. Additionally, the combined extracted features from the file, optional, and section headers were used to detect malware with a random forest classifier, which achieved the greatest accuracy of 98.63%. Aakash Wadhwani [17] also produces models based on static analysis using two features: N-grams and HMM. The model achieves that, using N-gram features, the SVM method gives the best accuracy, which is 97%; when using HMM feature vectors, SVM performs best with 96.8% accuracy, while KNN ranks second with 96.38% accuracy. Sanjay et al. [18] introduced an approach based on the frequency of opcode occurrence and used five feature selection methods (FS, IG, GR, CS, and US) to select the top 20 features; therefore, they achieved that the Fisher score method is the best among all and gets accuracy of 100 % in the cases of Random Forest, LMT, NBT, and J48 Graft (Figure 11).

In our model, we focus on feature selection using the filter method with the information gain attribute in Weka software and select important features in Google

Study	Feature	Algorithm	Accuracy		Feature	No. of the important feature	Algorithm	Accuracy	When reducing features to the top around 20 features	When reducing features more
Vivekanand et al.2022	API call	RF	98.74 %	In our Model	Opcode	All 200 feature	RF	100%	100%	99.58%
Namita et al. 2022	API-integrated feature	DT, SVM, LR, and KNN	99.9%				J48			99.58%
							NB			99.58%
							LR			100%
Mohammed ali et al.2020	N-grams	NB	98.43%				SGD		99.58%	99.58%
		LR	84.5%				KNN	99.58%	99.58%	98.76%
Saima et al.m2019	PE header-optional header	RF	97.24%				SVM	98.76%	98.76%	97,93%
	combined extracted features	RF	98.63%		API call	All 250 features	RF	100%	100%	100%
Aakash Wadhwani.2019	N-gram	SVM	97%				J48		90%	90%
	HMM	SVM	96.8%				NB		100%	95%
		KNN	96.83%				LR	85%	85%	85%
Sanjay Sharmaet al 2018	Fisher score method	RF, LMT, NBT, and J48	100%				SGD	90%	85%	85%
							KNN	95%	95%	90%
							SVM	70%	70%	70%

Figure 11: Comparing all accuracy with previous related work.

Conclusion And Future Work

The objective of this study was to identify the most effective and accurate machine learning methods and determine which features have an impact on metamorphic malware detection. We first introduced metamorphic malware structure and behaviour, then demonstrated obfuscation techniques. As we discussed in this work, there are different approaches of malware analysis and methods to detect it.

The main part of this project was an experiment on detecting variant metamorphic malware datasets on Windows and Android platforms. We attempt to detect metamorphic malware by using machine learning techniques to develop models that are proficient in identifying metamorphic malware and its future variations using seven different methods of ML, compare the results, and identify which feature is important to detect and classify metamorphic malware.

We have shown experimentally that in the first dataset based on the opcode feature, the decision trees and random Collab with two different datasets. We also reduce the number of features and compare all the results; therefore, we achieve that the detection accuracy rate in all selection features achieves 100% with RF and DT, but when reducing the number of important features, the accuracy decreases to 97.17% with SGD when using Weka software, but when using Google Collab, RF gives the highest accuracy with all the features and SVM gives the lowest accuracy with 97.52% when reducing to 5 important features in the first dataset based on the frequency of Opcode.

In the second dataset based on an API call as a feature, we have two files, the first one collected from malware web and the second one generated using two different algorithms. We achieve that the RF, DT, and NB give the highest accuracy rate of 100%, while the SVM gives the lowest accuracy rate of 70%, as shown in Table. forest classifiers achieve a high accuracy rate with 100% detection for all datasets with a diverse amount of features.

In addition, in the 2nd dataset that includes a time-ordered sequence of the system calls as a feature, the result shows that random forest gives the best detection rate. Finally, Our results show that the amount of features plays a minor role in improving detection rates. While detection depends mainly on the dominant features. Moreover, the large number of features might consist of irrelevant features that affect detection accuracy and consume a lot of resources.

In the future, it will be an interesting field of work to detect metamorphic malware by using a hybrid analysis approach to improve recognize the behaviours of metamorphic malware that uses obfuscated techniques to evade detection and extracts hyper-features to improve detection malware and its future mutation. We also propose to experiment with mutation malware detection using a deep learning algorithm.

References

1. Sk Sasidharan, C Thomas (2017) Short Review on Metamorphic Malware Detection in Hidden Markov Models. International Journal of Advanced Research in Computer Science and Software Engineering 7(2).
2. P Ször, P Ferrie (2001) Hunting for metamorphic. Virus Bulletin Conference.
3. S Ibrahim, M Masrom, BB Rad Camouflage (2012) In Malware: From Encryption to Metamorphism. IJCSNS International Journal of Computer Science and Network Security 12(8).
4. E Masabo, KS Kaawaase, JS Otim3, J Ngubiri, D Hanyurwimfura (2018) A State of The Art Survey on Polymorphic Malware Analysis and Detection Techniques.
5. MZ Gashti (2017) Detection of Spam Email by Combining Harmony Search Algorithm and Decision Tree Eng Technol Appl Sci Res 7(3): 1713-1718.
6. A Al-Marghilani (2021) Comprehensive Analysis of IoT Malware Evasion Techniques. EngTechnol Appl Sci Res11(4): 7495-7500.
7. I You, K Yim (2017) Malware Obfuscation Techniques: A Brief Survey International Conference on Broadband Wireless Computing, Communication and Applications.
8. J M Borello, L Mé viruses (2008) Code obfuscation techniques for metamorphic. Journal in Computer Virology.
9. X Li, P Loh Viruses, Mechanisms of Polymorphic and Metamorphic (2011) European Intelligence and Security Informatics Conference.
10. FA Aboaoja, A Zainal, FA Ghaleb, BA Alrimy, TA Eisa, et.al. (2022) Malware Detection Issues, Challenges and Future Directions:A Survey. 12(17).
11. A Sharma, S K Sahay (2014) Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. International Journal of Computer Applications 90(2).
12. K Aldriwish (2021) A Deep Learning Approach for Malware and Software Piracy Threat Detection. EngTechnol Appl Sci Res 11(6):7757–7762.
13. Vinod P, V Laxmi, MS Gaur (2009) Survey on Malware Detection Methods.
14. V Kuriyal, D Bordoloi, DP Singh, V Tripathi (2022) Metamorphic and polymorphic malware detection and classification using dynamic analysis of API calls. AIP Conference Proceedings 2481 (1).
15. M Ali, S Shiaeles, G Bendiab, B Ghita (2020) MALGRA: Machine Learning and N-Gram Malware Feature Extraction and Detection System. 26(10).
16. S Naz, D K Singh (2019) Review of Machine Learning Methods for Windows International Conference on Computing, Communication and Networking Technologies (ICCCNT).
17. A Wadhwani (2019) JavaScript Metamorphic Malware Detection Using Machine Learning Techniques. Master's Projects 688.
18. AVatlas 2023.
19. S Sharma, R Krishna, S K Sanjay K Sahay (2019) Detection of Advanced

Malware by Machine Learning Techniques.

20. H Lakhotiya, A Sharma Metamorphic Malware Classification.
21. K Babagba.
22. N Dabas, P Ahlawat, P Sharma (2022) An Effective Malware Detection Method Using Hybrid Feature Selection and Machine Learning Algorithms. Research Article-Computer Engineering and Computer Science 17(2).
23. Feature Selection Techniques in Machine Learning.