*electronics*

MDPI

*Article*

# MALGRA: Machine Learning and N-Gram Malware Feature Extraction and Detection System

**Muhammad Ali [1], Stavros Shiaeles [2,*], Gueltoum Bendiab [2] and Bogdan Ghita [1]**

[1] School of Engineering, Computing, and Mathematics, University of Plymouth, Plymouth PL4 8AA, UK; muhammad.ali@plymouth.ac.uk (M.A.), bogdan.ghita@plymouth.ac.uk (B.G.)
[2] School of Computing, University of Portsmouth, Portsmouth PO1 2UP, UK; gueltoum.bendiab@port.ac.uk
* Correspondence: stavros.shiaeles@port.ac.uk

Received: 29 June 2020; Accepted: 17 October 2020 ; Published: 26 October 2020


**Abstract:** Detection and mitigation of modern malware are critical for the normal operation of an organisation. Traditional defence mechanisms are becoming increasingly ineffective due to the techniques used by attackers such as code obfuscation, metamorphism, and polymorphism, which strengthen the resilience of malware. In this context, the development of adaptive, more effective malware detection methods has been identified as an urgent requirement for protecting the IT infrastructure against such threats, and for ensuring security. In this paper, we investigate an alternative method for malware detection that is based on N-grams and machine learning. We use a dynamic analysis technique to extract an Indicator of Compromise (IOC) for malicious files, which are represented using N-grams. The paper also proposes TF-IDF as a novel alternative used to identify the most significant N-grams features for training a machine learning algorithm. Finally, the paper evaluates the proposed technique using various supervised machine-learning algorithms. The results show that Logistic Regression, with a score of 98.4%, provides the best classification accuracy when compared to the other classifiers used.

**Keywords:** malware; dynamic analysis; sandbox; SNDBOX; N-grams; API call; machine learning; Logistic Regression; Naive Bayes; Random Forests; Decision Tree

## 1. Introduction

Malware is a broad term that refers to any piece of software designed intentionally to damage the normal functionality of a computer or a network [1]. Malicious behaviour may involve illegal activities such as stealing sensitive information (such as login credentials, credit cards, or other privacy-related information), gaining unauthorized access to private systems, or espionage. Current malware target widely and indiscriminately from individuals and residential customers to IT systems within large organisations or critical country-wide infrastructures (including nuclear plants and water supply systems), which traditionally have been considered highly secure [2]. Within this spectrum, according to recent reports [3,4], there is a significant increase in the production of malware variants that are targeting critical infrastructures. In addition, existing malware variants are continuously evolving, as malware writers improve their detection avoidance mechanisms. The most recent SonicWall Cyber Threat Report [5] indicates that the SonicWall service discovered nearly 440,000 malware variants in 2019, which averages to over 1200 malicious software being released every day. In the same context, a recent security report by Panda Lab identify the existence of over 2 million new malware binaries in 2019 [6].

Given the above statistics and observations, it can be concluded that current security mechanisms face an uphill struggle dealing with the levels and complexity of newly released malware variants [6,7]. A multitude of techniques and mechanisms [8–13] have been proposed by researchers for malware

analysis and detection, focused on following and replicating the behaviour of the malware and dynamically adapting to it.

The aim of this paper is to propose an effective feature extraction and representation algorithm that improves the classification accuracy of existing malware detection systems. The proposed detection system is based on N-grams and machine learning and, due to its capabilities inherited from the domain of machine learning, provides a cheaper, more adaptive solution to replace the traditional expensive malware analysis.

### *1.1. Malware Detection*

A malware detector is a program that is used to scan information systems to detect, identify and prevent IT infrastructure from malicious software; therefore, three main goals of detection systems are scanning of the system, detecting the malicious software and removing the malware. Presently, the malware detection system uses signatures of existing malware with limited heuristics to detect sophisticated malware such as the polymorphic and metamorphic strain of malware. A malware detection program *MD* is defined as a computational function whose job is to examine any software that might be malicious of clean, therefore, MD: S->malicious , clean. The latest and even traditional antivirus software examines the software *S* to discover whether it is malicious or benign by comparing the signature of the given software S with the database containing the signatures of known malware *msig*. If the signature of software S is matched then it is flagged as a malware else cleanware and the above definitions can be represented as:

$$\mathbf{MD(S)} = \begin{cases} \text{Malicious} & \text{if msig} \in S \\ \text{Clean ,} & \text{otherwise} \end{cases}$$

Two primary malware detection methodologies are widely utilized by security experts: static analysis also known as code analysis and dynamic analysis. These two approaches help researchers quickly and thoroughly identify the damage a malware can produce as well as provide appropriate countermeasures to be utilised by antivirus or Intrusion Detection Systems such as signatures.

### *1.2. Malware Analysis Techniques*

Malware analysis is the study of investigating malware to understand its behaviour; it also articulates how to study the different components of malicious software. From the interaction point of view, there are two types of malware analysis, namely static and dynamic.

#### 1.2.1. Static Analysis Technique

This type of analysis is performed by determining the signature of the malicious binary file. This signature is a unique identifier for each binary file, calculated based on the hash of the file [3]. Numerous approaches have been proposed by researchers [14–18] to perform static analysis. Examples of static analysis include extracting the byte code sequence from the binary by disassembling the binary file to extract the opcode sequences and mining control flow graph from assembly file and sometimes mining API calls from the binary file. All these extraction methods are based solely on the characteristics of a binary file. Each of the techniques mentioned above constitutes feature sets, which are later used for detecting the malware. There are several advantages of static analysis, such as being quite fast and not requiring any control environment to execute malicious software. However, malware writers also employ specific coding methods, such as metamorphism and polymorphism, which dynamically modify the content of the binary code without significantly altering its functionality but rendering static analysis unusable. In order to eliminate these issues, analysis must focus on the resulting behaviour and functionality of the code rather than its content.

#### 1.2.2. Dynamic Analysis Technique

To overcome the deficiencies of the static analysis method, dynamic analysis techniques [19–22] execute malicious software and trace its behaviour by analysing the actual program instructions and monitor the malicious code behaviour while executed in a sandbox environment. Sandbox technology is a safe environment for malware analysis, as it allows the malware to execute in an isolated environment in a form of a "black hole" containing the untrusted programs and, should the malicious software attempt to access remote hosts, it can block or redirect traffic to prevent it from accessing the live network. Dynamic analysis has been considered as an effective technique for understanding and classifying metamorphic and polymorphic malware in particular, as it observes the interaction of malware with the operating system in a quarantined environment to collect the behavioural characteristics that would ultimately help in creating a effective defence mechanism.

Researchers have proposed numerous techniques [7,23,24] which reuse the concepts from a wide range of computational approaches, including graph theory, machine learning, information visualization and so on. In addition to the aforementioned benefits, machine learning-based algorithms are considered to be the most effective technique because of the self-learning capabilities they possess and the popularity they have gained among the research community. This approach analyses the available malware file information by using different features derived from static and dynamic analysis of the malware [23]. Then, extracted features are used to train the classification model to discriminate between malware and legitimate software. Finally, the trained model is used to provide predictions about unknown software. Although much work has been done in this area using different dynamic and static malware features, there is still a need for improvement in identification and mitigation of malware. More specifically, there is a significant need for an efficient features extraction approach that can accurately describe the malicious behaviour of a malware, and implicitly increase the accuracy of the malware detection mechanism. In this paper, we present such an effective feature extraction and representation algorithm that can improve classification accuracy for malware detection systems. Furthermore, the paper provides both theoretical foundations and experimental results to validate the designs of the proposed approach.

The novelty of this paper is based on the designing of a classification system that is based on N-grams and machine learning algorithms to detected malware using new features by utilizing the proposed feature extraction and representation algorithm. At first, we used the dynamic analysis method for which we utilized an AI-based sandbox to extract an Indicator of Compromise (IOC) from malicious files. In the next step, we applied our proposed algorithm to create N-grams features. In scenario one, we have taken the Application Programming Interface (API) calls along with the memory location of their arguments to construct valid N-grams whereas in scenario 2 the N-grams were constructed by taking the function calls along with the address of its argument. The purpose of taking two different settings is to explore those features, which result in optimal accuracy based on our results for other methods, that could vary. The paper also proposes Term Frequency–Inverse Document Frequency (TF-IDF) as a viable novel statistical alternative used to identify the most significant N-grams features for training a machine learning algorithm. Finally, the paper evaluates the proposed technique using various supervised machine learning algorithms. The results show that Logistic Regression with a score of 98.4 % provides the best classification accuracy compared to the other classifiers used if we take scenario 1 setting whereas Logistic Regression gives 84.5% accuracy if we apply scenario 2 settings. Furthermore, this research also aims to take two different features set setting and to compare which one is the best in terms of accuracy. We are confident that this study will help security researchers in building effective malware detection systems.

The rest of this paper is structured as follows: Section 2 gives an overview of related work. Section 3 presents the proposed methodology. Section 4 discusses experimental methodology and steps. Section 5 includes the conclusion and future work.

## 2. Related Work

Malware analysis and detection are crucial tasks to counter malicious attacks and prevent them from conducting their harmful acts. However, it is not always an easy task, especially when dealing with new and unknown malware that has never been seen. Conventional security mechanisms are rely on a specific set of signatures and employ static analysis techniques such as model checking and theorem proving to perform detection [1,3]. The malware functionality is explored by examining its static properties that imply maliciousness of the analysed file [3], then a signature (or a pattern) that identifies its unique characteristics can be crafted, so that specific malware can be identified in the future, including similar variants [1,25]. In this context, various malware detection techniques based on signatures and static analysis have been proposed by the research community [1,23,25]. Many of these works used the opcode sequence (or operational code) as a feature in malware detection by calculating the similarity between opcode sequences, or frequency of appearance of opcode sequences [26–28]. For instance, work in [26] proposed a new method to detect variants of known malware families based on the frequency of appearance of opcode sequences. However, this technique can only deal with known malware variants. Work in [29] presented a method to detect obfuscated calls relating to "push", "pop" and "ret" opcodes. They have proposed a state machine technique to cope with obfuscated calls. The proposed approach contains many deficiencies, such as authors being unable to cope with the scenario when push and pop instructions are decomposed into multiple instructions. Researchers in [30] extracted the opcode distribution from PE files, which can be used to identify obfuscated malware. However, this research was not effective in detecting malware, as some of the prevalent opcodes were not able to correctly identify malware.

Several other works have addressed malware detection using a Control Flow Graph (CFG) to extract the malicious program structure [31,32]. Most of these detection methods were based on comparing the CFG shapes associated to the original malware with that of variants [1]. For instance, the study in [31] compared basic block instructions of an original malware with those of its variants by using the Longest Common Sub-sequence (LCS). In [33], the authors extracted the system call dependency graphs from a corpus of malware containing 2393 executables. The resulting analysis method led to an accuracy of 86.77%. The main drawback of such malware detection techniques, which are based on static analysis of the malware program, is their vulnerability to evasion techniques like packing and obfuscation [1,23], which modify the malware payload by compressing or encrypting the data and severely limit the attempts to statically analyse the malware. When employing such obfuscation methods (packing, polymorphism, oligomorphism and metamorphism), attackers may successfully recycle existing malware by converting the malware binaries to packed and compressed files which reveal no information and therefore bypass the signature-based detection system [34].

To overcome the limitations of static analysis, many dynamic (or behaviour) analysis techniques have been developed. These techniques execute the malicious software in a controlled, confined and simulated environment in order to model the behaviour of the malware [1]. This kind of detection methods can detect malicious files based on normal and abnormal activities perceived in the isolated environment, with normal activities referring to the processes produced by benign applications and abnormal activities including the specific characteristic behaviours of malware [35]. In addition, dynamic analysis methods capture the interaction between the execution of the malicious sample and the operating system, thereby collecting the artefacts that allow security analysts to develop a technical defence. To hinder such efforts, advanced and sophisticated malware samples have the ability to check for the presence of a virtual machine or a simulated operating system environment. When detecting that it is being analysed by the sandbox agent, some malware modifies its behaviour, causing the analysis to yield incorrect results. The latest research work suggests [36,37] that traditional sandboxes are not evasive resistance because they hook data by dropping their agent in a controlled environment that can be easily detected by advanced strains of malware. As a result, they either stop their executing or execute with limited functionality.

*N-Grams*

An N-gram is a substring of a given sample of text or speech string with a length $n$. This string can include several types depending upon the application. For example, it can include letters, words, phonetics, syllables, etc. N-grams are created by splitting a text string into substrings of fixed length. For example, world MALWARE 3-grams will look like this "MAL", "ALW", "LWA", "WAR", "ARE". As a result of the string-based nature of analysis files, this technique has been widely adopted by the security researchers to represent the features of malware. The IBM research group [30] is considered to be a pioneer in using N-grams for malware analysis, having started work in the area since the 1990s. More recently, researchers also introduced the concept of using N-grams to create malware signatures. However, one of the main drawbacks of this line of research was that the early studies lacked an experimental methodology to prove the claim. Santos et al. [38] demonstrated that unknown variants of malware could be detected effectively using the N-grams technique by extracting code and text fragments from a corpus of malware that was executed in the control environment. Furthermore, signatures of these executables were created to train/test the classifier. In [33], a similar study was proposed, where N-grams were used to represent the features e.g., API calls, arguments, etc. and to reduce the features space to a manageable size, a feature reduction technique was applied which results in only those N-grams that significantly influence the accuracy. Similarly, Ref. [39] also demonstrated a classification method using N-grams. In this method, 2312 malware samples were executed in a controlled environment to obtain Indicators of Compromise (IOC) using dynamic analysis. The primary focus of the study was using the API log data to construct feature vectors. The N-gram technique was used to represent these features and, in a later stage, TF-IDF was employed to calculate the frequency of occurrence of these N-grams. Finally, the N-grams with a higher frequency of occurrence were used for model training and testing. The experimental results of the study depict the average precision and recall as 55% and 90%, respectively.

In [40], N-grams profiles were used as a malware detection mechanism and to design an effective, robust system. The feature vectors were created based on the frequency of N-grams, which were extracted from 25 malware and 40 clean files. Finally, the study claimed to achieve 94 % classification accuracy using the K nearest neighbour algorithm. Kolter and Maloof in [41] introduced an N-grams-based malware detection system. This method uses 4-grams as features and uses the information gain method to find the top 500 N-grams as the most significant features for classifying malware. The research utilizes several learning algorithms to train/test model, such as Naive Bayes, Support Vector Machines, Boosted Tree, etc. However, the experimental results and ROC curves depict that the Boosted Decision Tree produces good classification accuracy as compared to other algorithms. In [42], Shabtai et al. used static analysis for malware detection with different N-gram sizes N = $(\overline{1;6})$. In the study, several classifiers were implemented to check the efficiency and efficacy of the system.Finally, it was observed from experimental the results that system is performing better at N = 2 as compared to other values of N-grams.

In [43], a similar study was done by Moskovitch et al. using N-gram opcode analysis to investigate the detection of malware. Although this study had implemented different classification methods as compared to the previous study, even then the experiential results show that N = 2 is best in terms of malware detection. Different machine learning algorithms have been widely used for malware detection, including classification, clustering, time series, etc. Decision Tree, Random Forest, Logistic Regression, Support Vector Machine, etc. are the most common classifying algorithms.

Classification is a controlled process that is usually separated into 2 phases: the first step includes the preparation of classification using a classification algorithm centred on the specimen characterizing and based on a specific area. There are a variety of classification algorithms used in the literature to classify malware. Such classification algorithms are discussed in-depth as below:

Decision Tree algorithms create a model of decision-making dependent on real data component values. Any node without leaves in the trees is a measure of an anti-category feature in the study set of each anti-leaf branch division, and a leaf network is a class or class allocated for that node.

A direction from either the root to a leaf vertex defines a ranking law [44]. The Application Program Interface, creatively named as the testing item by PE, uses a shifting window mechanism to remove this functionality and adopts the Decision Tree Algorithm for identifying unknown malware, which has been suggested by [45] to detect uncertain malware. The consequence reveals that the Decision Tree is beyond the Naive Bayesian algorithm, and the consistency is more reliable than the other two algorithms.

The Random Forest is an aggregate learning method that creates a plurality of decision-making bodies and produces a forecast, which is the fashion of the different trees groups. For individual trees to expand, a collection from the training database (local set) is chosen with the remaining samples to approximate fitness. Trees are generated by separating the regional game at any nodes from a sampled subset of variables as per the importance of a random variable. The study in [46] proposed a methodology focused on string knowledge dependent on vulnerability classification utilizing numerous well-recognized detection algorithms, such as IB1, AdaBoost and Random Forest. The studies have shown that the grouping strategies of IB1 and Random Forest are the most successful for this area. Since the spread of polymorphic and metamorphic malware, dynamic analysis has been established as an effective method to model the behaviour of these malware samples in a controlled environment [25,47].

In this study, we aim to improve the current state of the art in malware analysis by presenting the design and experimental evaluation of a malware detection system, with the following contributions:

(a) Malware behavioural modelling using advance sandbox: In contrast to other studies and research work where the traditional sandboxes such as Cuckoo, Norman, Joe, etc. were used to model the behaviour of malware as from our previous research work [36], we found that they are not so effective in capturing the behaviour of advanced and sophisticated malware; therefore, we have utilized AI-based sandbox in this work to perform dynamic analysis and to model the behaviour of the malware.

(b) Feature extraction and representation algorithm: we presented an effective feature extraction and representation algorithm that helped in building the malware detection system with optimal accuracy based on our results for other methods it could vary. We select a set of observable features from analysis files generated during dynamic analysis and whose values can be used to infer whether a given sample is malware or not. We evaluate the most significant features in terms of its usefulness for malware detection.

(c) Optimise Classification: We present the design of a classification system that uses Naive Bayes, Decision Tree, Random Forest to detected malware using new features.

(d) Experimental Evaluation: We evaluate the accuracy of the classification system on a corpus of more than 60 malicious and 60 clean samples. To evaluate our methodology, K-fold cross validation is used and experimental results show that our proposed detection system achieved 98.43% detection accuracy with a very minimum false positive rate.

## 3. Proposed Methodology

In this paper, we proposed an innovative approach to extract the most significant malware features that can result in optimal accuracy for the methods tested based on our results, however this may vary for other methods. The behavioural modelling of the malicious samples was captured in the form of log files, generated using the dynamic analysis technique. In the next phase, the N-grams technique was utilized to create N-grams features set and in order to reduce the feature space, the TF-IDF method is applied. Lastly, feature sets were converted into binary vectors that will be used by the machine learning algorithms for training and testing purposes. In this context, four learning algorithms have been used to evaluate the performance of the proposed approach including the Logistic Regression (LR), Random Forest (RF), Decision Tree (DT) and Naive Bayes (NB). The whole proposed methodology is shown in Figure 1.

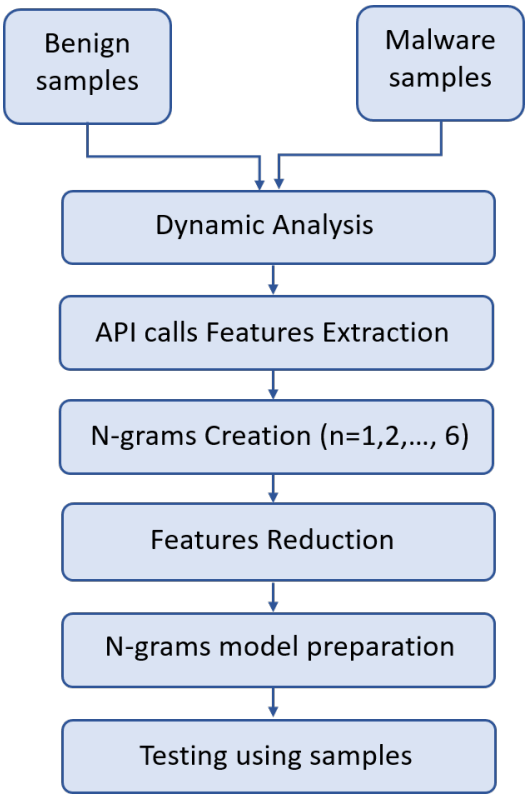


**Figure 1.** Architecture of proposed scheme.

### *3.1. Outline of the Proposed Work*

In this section, we have discussed our methodology for classifying malware and benign samples as shown in the Figure 1. Following are the steps adopted in our approach.

1. Collection of the malicious and clean sample in PE file formats.
2. Extracting the features from executables by performing dynamic analysis.
3. Generating N-grams for n = $(\overline{1,6})$ for both malware and benign samples.
4. Reducing feature space by applying the feature reduction technique.
5. Generating different N-grams models using the classifier e.g., Naive Bayes, Decision Tree, etc.
6. Test samples are validated using each N-gram model. The standard evaluation metrics (True Positive Ratio—TPR, False Negative Ratio—FNR, True Negative Ratio—TNR, and False Positive Ratio—FPR ) were used to find the sensitivity and accuracy.

### *3.2. Stages of Proposed Methodology*

This section discusses the stages of the proposed methodology. It comprises of three stages as delineated in Figure 1. Stage 1 is a monitoring stage in which behaviour modelling of samples were done using an AI-based sandbox, Stage 2 is feature engineering in which N-grams features were created using strings information as extracted from the text files. Stage 3 describes the use of machine learning algorithms (classification algorithms) to determine whether an input sample is malicious or benign. The details of each stage is explained in the following sections.

- Monitoring stage

In this stage, the data corpus was collected from the virus share [48], a website containing a large repository of malicious as well as clean samples. The 60 selected malicious samples belong to a different class of malware, including trojan, backdoor, worm, etc. and 60 legitimate samples were collected from the trust entities websites. Furthermore, these samples were executed in SNDBOX, an AI-based

sandbox (https://app.sndbox.com/login) to model the behaviour of malicious samples. The reason for using this AI-based sandbox is that it has an invisible agent that deceives malware by executing its full range of intended functionality, revealing its true malicious nature, intent and capabilities, which is one of the fundamental requirements to model the behaviour of most advanced as well as sophisticated malware and such detection was not possible with traditional agent-based sandboxes as revealed by the latest research [37].

- Feature Engineering Stage

In general, features engineering (which include feature extraction, selection and representation) is a crucial step in machine learning tasks and has a significant influence on the performance that the classification model can achieve. It refers to the process of transforming the raw, vague and broad collection of inputs into different sets of features is referred to as a feature extraction process. The main objective of this process is to select significant features that can help in building effective malware detection system. Like in other domains, feature extraction is also considered as the most crucial stage of malware detection because it helps determine the most effective representation of malicious samples.

Malware researchers have proposed numerous methods for features engineering such as, binary features extraction, frequency feature extraction, frequency weight feature extraction, hidden Markov model, N-grams, etc. Furthermore, the feature vectors of fixed length created from the above process were used by a machine-learning algorithm to create a learning model. Therefore, when it comes to developing an efficient model, feature engineering is the most vital step. Innumerable methods have been proposed by the research community to represent features that are in the form of opcodes, API calls and sequences of code of bytes to fixed-size feature vectors using several techniques. One of the most significant techniques to feature representation is the use of N-grams.

To evaluate the proposed method, a routine was written to python to extract the string information from the analysis files. This study mainly focuses on two scenarios and in the first scenario, we have taken the API calls along with the memory location of their arguments (the function along their counts are shown in Table 1) to construct valid N-grams, whereas in scenario 2 the N-grams were constructed by taking the function calls along with the address of its argument as shown in the Figure 2. The purpose of taking two different settings is to explore those features that can produce good classification accuracy. Therefore, all the other features were discarded and API-N-grams for n = $(\overline{1,6})$ were generated for both scenarios, and later on used to create a feature vector. For each N-grams set, we sorted it according to the frequency of occurrence and eliminated grams below a threshold to reduce the feature space as shown in the Table 1.

**Table 1.** List of strings extracted from analysis files.

| API Calls | Count |
|---|---|
| ZwOpenFile | 42 |
| ZwCreateSection | 33 |
| ZwFlushInstructionCache | 22 |
| ZwWriteFile | 12 |
| ZwOpenKeyEx | 59 |
| ZwCreateUserProcess | 2 |
| ZwTerminateProcess | 1 |

Furthermore, the effective feature set was calculated using the TF-IDF algorithm, a statistical method used to evaluate how relevant a word is to a document in a document corpus. It is calculated by multiplying two metrics, to find the occurrence of a word in a given document. In any given collection of documents, the occurrence of certain words is more as compared to others such as “of”, “the”, “a”, etc. Therefore, the same idea was applied to selected features, that certain calls pertain to all operations of any program that might or might not be malicious, so we have utilized the TF-IDF value

to determine the feature weighting. The TF-IDF weight of a term is computed using below mentioned formulas:

$$\textbf{TF(w, d) = fd(w): frequency of w in document d}$$

$$\textbf{IDF(w, D) = log 1 + |D| 1 + df(d,w)}$$

where **'TF'**, which stands for term frequency, is an occurrence of a specific word **'w'** in document **'d'**, and **'IDF'** stands for inverse term frequency is the number of times the word occurs in a document whereas **df(d, w)** is the number of documents the word **'w'** appears in. In this research work, we measured IDF of a sequence based on whether that sequence is unique or exist in all samples of malware. It is a logarithmically scaled fraction of a value calculated by dividing the total number of malware by the number of malware containing that API calls. Thus, the proposed work utilized the TF-IDF method to determine the feature weighting and to find which feature set is giving the best accuracy.

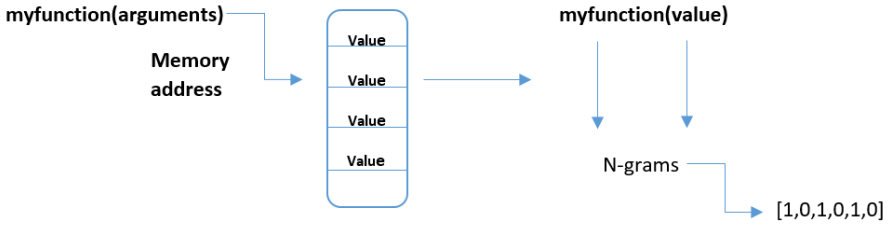


**Figure 2.** Creation of N-grams for scenario 1.

- Learning and Verification Stage

In the last stage, N-grams features were converted into binary vectors to train machine learning algorithms. Supervised learning algorithms were also utilized for training/testing the purpose. For example, Logistic Regression, Naive Bayes, Decision Tree and Random Forest were implemented. The experimental results show the performance of the proposed scheme, where the LR produced the best accuracy results in comparison with other learning algorithms, with an overall accuracy rate of 98.43% in case of scenario 1 and 84.5% in case of scenario 2.

### *3.3. Proposed Algorithm*

To extract the features from text files generated by dynamic analysis, we propose an algorithm formally described in Algorithm 1. Let $D$ be the data corpus containing samples, $S_i$ and $S$ be a set containing both malicious and clean samples, we can write:

$$MVC \leftarrow S$$

$$\text{Whereas S= } \{S_1, S_2, \ldots S_n\}$$
$$\text{i= 1,2,3,...n}$$

and finally data corpus $D$ containing samples can be written as:

$$D = S_i \tag{1}$$

The log files were generated for all the samples included in the data corpus $D$ using dynamic analysis, the significant features were extracted from the logs files as they are of significant importance because they allow the application program to access low-level hardware using these calls and lots of studies suggested that [49–51] cybercriminals use the same set of API calls to perform malicious

activities in the system. In the next step, the N-gram method was used to generate API-N-grams with a sorted table categorized according to the frequency of occurrences. Each N-gram set was sorted and grams below a specific threshold (less than 500 was discarded) were eliminated to reduce feature space. In the next step, we made a table by eliminating the N-grams with a lower frequency while keeping those with a higher frequency. Finally, these selected N-grams constitute the feature sets.

**Algorithm 1:** Methodology.

```
Result: Feature Extractions, representation and conversation to feature vectors
Data: Dataset 'D': contains both malicious and clean samples
MVC ← S_i
/* M are malicious, C are clean and S are samples                    */
begin
foreach S_i ϵ D do
    Generate behaviour analysis files from the dynamic analysis in AI-based sandbox;
    Extract the API calls along with its arguments from behaviour analysis files and discard other
     features e.g registry values, DNS calls etc.;
    Creating 1, 2, 3.... Till 6 n-grams for API calls along with its arguments;
    Make a sorted table of API-n-grams according to the frequency of occurrence;
foreach S_i ϵ D do
    foreach API-n-gram do
        Calculate the frequency of API n-gram using
        TF-IDF
        if frequency of API n-gram > than defined threshold (Taken 500 as a threshold) then
            Add that API-n-gram to Unique list & Sort it
            with the frequency of occurrence
    foreach API-n-gram in sorted Unique list do
        Add corresponding API-n-gram to
        feature vector
    Creation of binary feature vectors for n-grams
```

## 4. Experimental Methodology and Steps

The whole methodology can be presented in the following four steps:

### *4.1. Dataset Collection*

Data are the most important part of any prediction. The quality of the data used is instrumental in testing hypotheses and reaching accurate conclusions. Therefore, the most vital part of any type of research should be the collection of a trusted and accurate dataset. We urge that explicit care should be taken with the source, method and quality of collecting data. These days, many websites contain vast repositories of both malicious and benign data samples, one example used in this research is Virushare [48]. After quality-sourced data, we conducted an experimental investigation on 60 malicious and 60 benign samples as shown in the following tables. The benign dataset included various application software while the malicious dataset included both polymorphic and metamorphic malware belonging to different families such as trojans, viruses, rootkits, worms, etc.

### *4.2. Dataset Preparation*

In any research, data constitute the input/output variables required to make a prediction that comes either in unstructured which implies that data are undefined and not properly labelled, or come

in structured forms, which implies that data are properly labelled. In our proposed work, we have taken the structured form of data which was labelled and categorized as malicious and benign using virus total and VTI reputation scoring engines. VirusTotal, a website owned by Google, used to inspect any submitted samples against more than 70 antivirus scanners database along with websites blacklisting services, where as SNDBOX uses VTI score to label the data and both of these engines were utilized in this research work to label the data.

In this study, we aimed to get those features which have a significant impact on the accuracy of the system and for this reason the focus was to experiment on a small dataset consisting of 60 clean and 60 malicious samples. It is worth noting that even if there are works with more samples there is not any other work utilising the sandbox we used which provides far better information than Cuckoo and other sandbox available [37]. The amount of data we had collected from the analysis of the 120 sample utilising SNDBOX was really huge. The importance of this work rely on the feature extraction methodology and we plan to use more samples on later state after we develop robust classifier using these features vectors. Furthermore, once we were able to develop our system, we experimented with the impact of a slightly larger dataset in regards to the accuracy by increasing the number to 90 benign and malware (180 in total) samples and observed that classifier accuracy was not affected by the sample size. Hence, it was found that if the feature engineering stage is completed properly with effective and efficient tools (like in this study using AI-based sandbox and proper feature engineering technique) then the balanced dataset is of secondary importance. However, in the case of an imbalanced dataset, the results would be different and it will affect the accuracy as values will be missing which are important for the feature engineering.

### *4.3. Cloud-Based Virtual Lab*

We created a cloud-based virtual lab to run the samples (Figure 3). The analysis testbed included a cloud-based sandbox (SNDBOX) used to export all of the information from the samples collected from the virus share. With SNDBOX as the main malicious software analyser, we analysed each sample in various programs such as Windows 7 Ultimate SP1 environments for 60 s with Adobe Acrobat Reader DC 2019, Adobe Flash Player 3,1Google Chrome 70.0.3., Java 8 Update 19, Microsoft.NET Framework 4.7, Microsoft Office Standard 2010, Python 2.7.15 and WinRAR 5.61. The results of each analysis request were saved as a subfolder containing all the raw logs, pcap files, images, JSON files and any other information obtained during the analysis

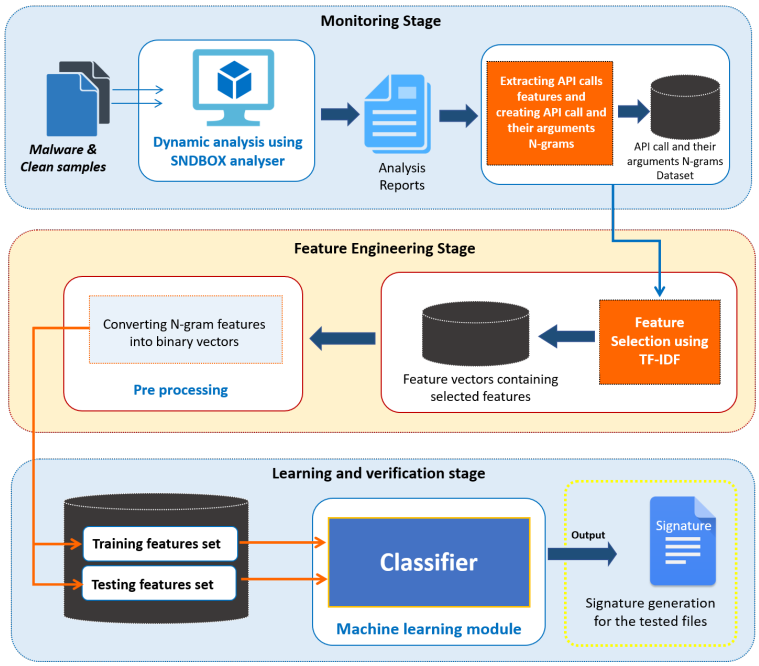


**Figure 3.** Flow diagram of proposed scheme.

### *4.4. Pre-Processing and Feature Generation*

In this stage, the samples were pre-processed and cleaned to remove noise and irrelevant entries from the log files generated using dynamic behavioural analysis. In this research, the focus was on extracting the features which have significant impact on accuracy; therefore, only those artifacts were taken and the rest of the artifacts were discarded. In scenario one we have taken API calls along with the memory location of their arguments to construct valid N-grams whereas in scenario 2 the N-grams were constructed by taking the function calls along with the address of its argument. The purpose of taking two different settings is to explore those features that can produce good classification accuracy. In the next stage, the N-grams with n = $(\overline{1,6})$ were created (as shown in the Figures 4–6) for these selected indicators of compromise and stored in a table based on frequency of occurrence. The reason for such values (n = 1 through 6) is that lots of research work

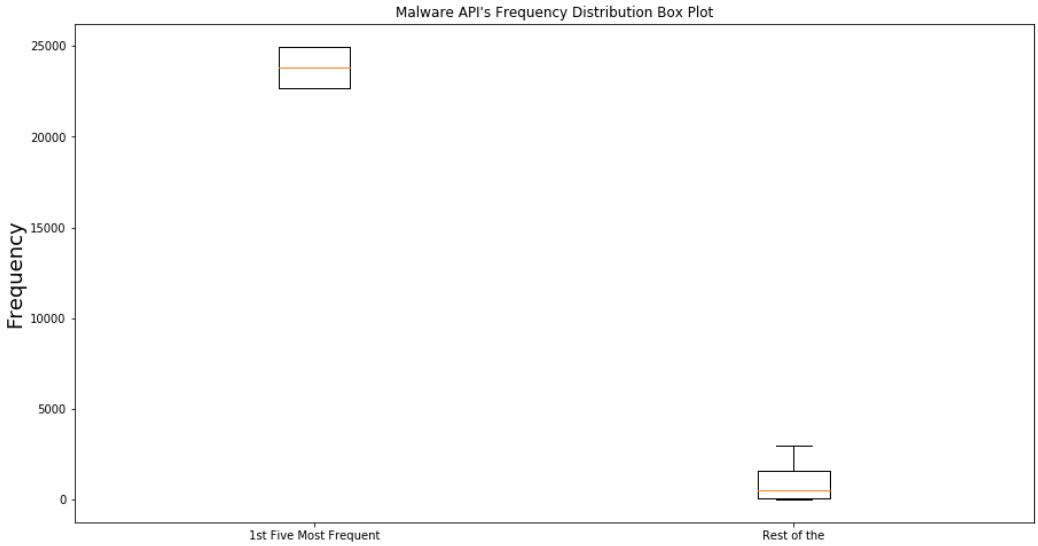


**Figure 4.** Box plot for malware API frequency distribution.

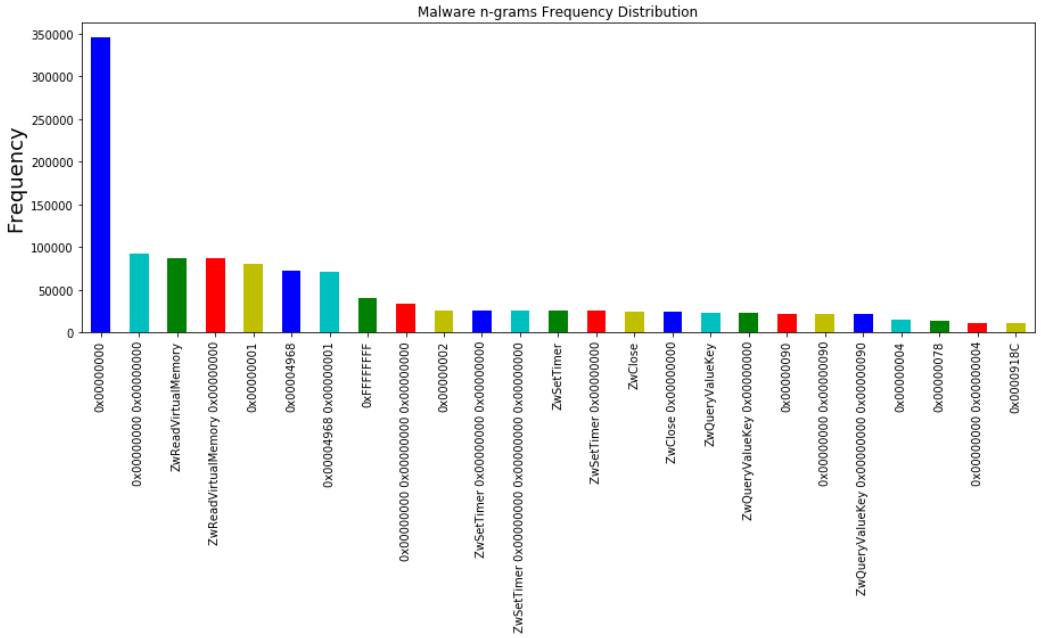


**Figure 5.** Malware API N-gram frequency distribution.

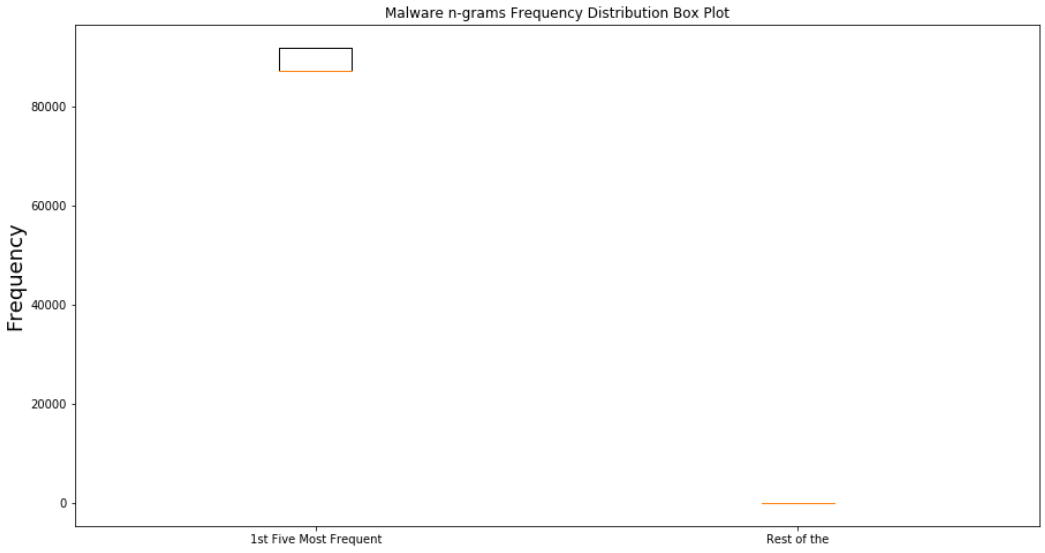


**Figure 6.** Malware API N-gram frequency distribution.

Several studies, such as [52,53], have mentioned that N-gram performs well between this range with a lower error rate. Moreover, the feature vector was created as follows. First, we generated the set of one till six API-call-grams (for both scenarios 1 and 2) for each file generated through dynamic analysis, then we sorted each of these N-grams set and a unique sorted list was constructed by applying TF-IDF to reduce the feature space, later on, a table was generated containing N-grams for the calls corresponding to each sample file in data corpus. Finally, these sorted function grams constitute the features. In the Table 2, we have presented an example of a feature vector generated using our proposed algorithm.

**Table 2.** A sample dynamic feature vector.

| Class | 1-Gram | 2-Gram | 3-Gram | ... | 6-Gram |
|---|---|---|---|---|---|
| Malware | 1 | 0 | 1 | ... | 1 |
| clean | 0 | 1 | 0 | ... | 0 |

### *4.5. Classification Algorithm and Evaluation Metrics*

In the following section, we described its accuracy, F-measure and precision. Accuracy is defined as the ratio of the number of right predictions out of the total number of predictions and is represented as:

$$\text{Accuracy} = \frac{TP + TN}{Totalsample}$$

A false positive rate is when a clean file is wrongly classified as a malicious file by the system and is represented as:

$$\text{False Positive Rate} = \frac{FP}{TN + FP}$$

True positive rate is when a malicious or clean file is rightly classified and written as

$$\text{True Positive Rate} = \frac{TP}{FN + TP}$$

where $TP$ = True Positive, $TN$ = True Negative, $FP$ = False Positive and $FN$ = False Negative, respectively.

In this step, we applied the supervised machine learning algorithms, such as Logistic Regression, Random Forest, Decision Tree and Naive Bayes, because they possess stronger reliability compared to other, more unsupervised approaches. Additionally, we randomized the data corpus and split it 80/20 into training and testing datasets using the IPython Jupyter notebook (v 5.7.2) to keep different proportions. The result of these experiments revealed that the best classification accuracy was produced by the Logistic Regression as compared to other learning algorithms, with an accuracy of 98.43% for scenario 1 and 84.5% in case of scenario 2. To validate the proposed methodology, 10-fold cross-validation was applied; the data corpus was randomly divided into ten disjoint sets known as *folds* and the purpose is that both the training and testing phase should be executed ten times. In each iteration step, one fold is used for the testing set and the remaining nine folds for training set, so as a result, each sample of data corpus was used 10 times for training and once for testing. Finally, the performance of each classifier was evaluated in the form confusion matrix as shown in the Figure 7 (scenario 2), a specific table layout that displays the performance level of a classification system.

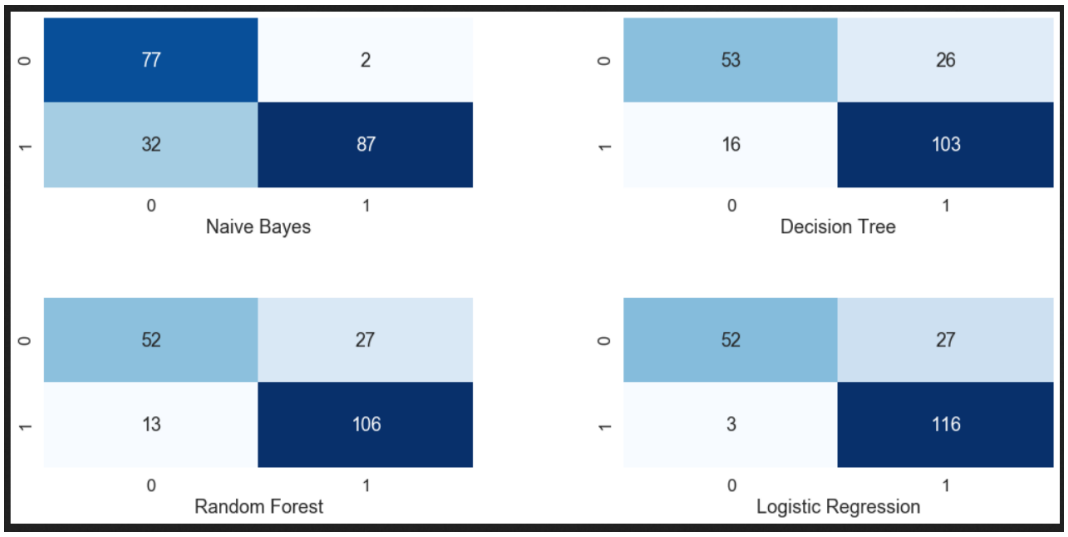


**Figure 7.** Confusion matrix.

As it can be observed, the proposed combination of features and algorithm performs consistently better than their counterparts from previous studies, both in the case of investigating API calls in Table 3 and function calls in Table 4. As highlighted, logistic regression is the most successful segregation algorithm in both scenarios. For completeness, we also list in the malware samples in Table 5 and the clean samples in Table 6 below.

**Table 3.** Result of scenario 1 and comparison with other studies.

| Classifiers | Model Accuracy | Study [54] | Study [55–57] | Study [58] | Study [49] |
|---|---|---|---|---|---|
| Naive Bayes | 92.91% | 91.6% | | 91% | 89% |
| Decision Tree | 97.64% | | 84% | | |
| Random Forest | 97.64% | 96.65% | 94.9 % | 96.2 | 91% |
| Logistic Regression | 98.43% | | 97.7% | | |

**Table 4.** Scenario 2 results of experiment.

| Classifiers | Model Accuracy | Precision | Recall | f1 Score |
|---|---|---|---|---|
| Naive Bayes | 82.83% | 0.98 | 0.73 | 0.84 |
| Decision Tree | 78.79% | 0.80 | 0.87 | 0.83 |
| Random Forest | 79.8% | 0.80 | 0.89 | 0.84 |
| Logistic Regression | 84.5% | 0.81 | 0.97 | 0.89 |

**Table 5.** Malware samples that were used.

| | Category | Virus Name | Type: Spy-Steal Data | C and C | Backdoor | Stealth |
|---|---|---|---|---|---|---|
| 1 | Trojan | Trojan.Generic.pafish | X | | | |
| 2 | Trojan | Trojan.Win32win32.duqu | | | | |
| 3 | Trojan | Trojan.Generic.Cerber.exe | | | | X |
| 4 | Trojan | Trojan. Win32Mole.exe | | | | X |
| 5 | Trojan | Trojan. Win32.Spora.exe | | | | X |
| 6 | Trojan | Trojan.Win32GrandCrab-01.exe | | | | X |
| 7 | Trojan | Trojan. Win32.Delf.xo | | | X | |
| 8 | Trojan | Trojan. Win32.DarkTequila.exe | | | | |
| 9 | Trojan | Trojan. Win32.psiphon.exe | X | | | |
| 10 | Trojan | Trojan.Generic.yigzwl | | | | X |
| 11 | Trojan | Trojan.Generic.Vcffipzmnipbxzdl | | | | X |
| 12 | Worm | Win32.Gamarue | X | X | | X |
| 13 | Worm | W32.Cridex.A.worm | X | X | | X |
| 14 | Worm | Worm.VBS.Agent | | X | | |
| 15 | Worm | Worm.Win32.3DStars | | | X | X |
| 16 | Worm | Worm.Generic3.PEM | | | X | |
| 17 | Worm | Worm.Win32.Mira.A | X | | | |
| 18 | Worm | Worm.Generic2.CMVO | X | | X | |
| 19 | Worm | Worm.Win32.Cake | | | | X |
| 20 | Worm | Worm.Win32.Fever | X | | | X |
| 21 | Worm | Worm.Win32.Monkey.exe | X | | | |
| 22 | Worm | Worm.Win32.Mydoom.a.exe | | | X | X |
| 23 | Worm | Worm.Win32.Pikachu.exe | X | | | |
| 24 | Worm | Worm.Win32.Postman.exe | | | | X |
| 25 | Worm | Worm.Win32.Sharpei.a.exe | | | | X |
| 26 | Worm | Worm.Win32.Silver.exe | | | | X |
| 27 | Worm | Worm.Win32.Sobig.exe | X | X | | |
| 28 | Worm | Worm.KOOBFCE.SMC | X | | X | |
| 29 | Worm | W32/Wabot | X | X | | |
| 30 | Worm | Worm.vid.exe | X | | | |
| 31 | Worm | Email-Worm.Win32.Mydoom.l | | | X | X |
| 32 | Worm | Email-Worm.Win32.Naked | X | | | |
| 33 | Worm | Worm.Christmas-wishes.doc | X | | | |
| 34 | Worm | Win32.WannaCry.EXE | X | | X | X |
| 35 | Worm | Win32.F7F105F9.exe | | | | |
| 36 | Worm | Win32.2tetup.exe | X | | | |
| 37 | Worm | Win32.GrandCrab-01.exe | X | | | |
| 38 | Worm | Win32.GlobeImposter.exe | X | | | |
| 39 | Botnet | Win32.Lolbot.aoi | | | X | |
| 40 | Botnet | WORM/IrcBot.tlq | X | X | X | |
| 41 | Botnet | W32.Jorik$_{L}$*olbot.O*!*tr* | X | | X | |
| 42 | Botnet | Win32.SdBot.aamk | X | X | X | |
| 43 | Botnet | W32.ZBot.42352 | X | | X | X |
| 44 | Botnet | Win32.Jorik.SdBot.e | | | X | |
| 45 | Botnet | MSIL.NanoBot.ibh | | | X | |
| 46 | Botnet | Win32.Zbot.vtii | X | | X | X |
| 47 | Botnet | Win32.Ngrbot.anak | | | | X |
| 48 | Botnet | Win32.Alinaos.G | X | X | | |
| 49 | Botnet | GenericKD.2143403 | | | X | |
| 50 | Botnet | Win32/ChkBot.A | | | X | |
| 51 | Botnet | MSIL/Lizarbot.A | X | X | X | |
| 52 | Botnet | Win32.Jorik.Lolbot.f | X | X | X | |
| 53 | Botnet | Win32.Zbot.sbdj | X | | X | X |
| 54 | Botnet | MSIL.NanoBot.bi | X | | X | |
| 55 | Botnet | Win32.Ngrbot.uyk | | | | X |
| 56 | Botnet | Win32.Boht.qo | | X | X | |
| 57 | Botnet | W32/Zbot.AJJU!tr | X | | X | X |
| 58 | Botnet | Win32.VBInject | | | | X |
| 59 | Botnet | Trickbot | | X | | |
| 60 | Botnet | obfuscated.js | | | | X |

**Table 6.** Clean samples that were used.

| | Category | Sample Name | Type |
|---|---|---|---|
| 1 | Normal | grammarlyaddinsetup.pe32 | Executable |
| 2 | Normal | Poweriso6-x64. | Executable |
| 3 | Normal | Vlc-2-2-1-win32 | Executable |
| 4 | Normal | Wireshark-win64-2.6.5 | Executable |
| 5 | Normal | ProtonVPN.exe | Executable |
| 6 | Normal | Notepad.exe | Executable |
| 7 | Normal | McAfeeWebAdvisor.exe | Executable |
| 8 | Normal | Putty2.exe | Executable |
| 9 | Normal | FTPDesktopClient.exe | Executable |
| 10 | Normal | SQLiteStudio-3.2.1.exe | Executable |
| 11 | Normal | KeePass-2.40-Setup | Executable |
| 12 | Normal | LinuxLiveUSB Creator 2.9.4.exe | Executable |
| 13 | Normal | flashplayer32-install.exe | Executable |
| 14 | Normal | Firefox Setup 14.0.1 | Executable |
| 15 | Normal | 7za.EXE | Executable |
| 16 | Normal | GoogleUpdateSetup.exe | Executable |
| 17 | Normal | Epson512523eu.exe | Executable |
| 18 | Normal | Microsoft-Toolkit.exe | Executable |
| 19 | Normal | Googlewebdesigner-win.exe | Executable |
| 20 | Normal | PDFSAM*Installer.exe* | Executable |
| 21 | Normal | FoxitReader-Setup.exe | Executable |
| 22 | Normal | TeamViewer-Setup.exe | Executable |
| 23 | Normal | Internet.Download.Manager.exe | Executable |
| 24 | Normal | TrueCrypt.exe | Executable |
| 25 | Normal | SkypeSetup.exe | Executable |
| 26 | Normal | HottNotes4.1Setup.exe | Executable |
| 27 | Normal | Normal TorchSetup | Executable |
| 28 | Normal | GitHubDesktopSetup | Executable |
| 29 | Normal | Nektar Bolt v1.0 CE.exe | Executable |
| 30 | Normal | ForkInstaller.exe | Executable |
| 31 | Normal | hashcat32.exe | Executable |
| 32 | Normal | AdobePatchInstaller.exe | Executable |
| 33 | Normal | TWUploader.exe | Executable |
| 34 | Normal | vmnat.exe | Executable |
| 35 | Normal | SenseDriver.exe | Executable |
| 36 | Normal | ISSetup.dll | DLL |
| 37 | Normal | SrvCtl.dll | DLL |
| 38 | Normal | panfinder.exe | Executable |
| 39 | Normal | strings.exe | Executable |
| 40 | Normal | procexp.exe | Executable |
| 41 | Normal | games.exe | Executable |
| 42 | Normal | acc.exe | Executable |
| 43 | Normal | KutoolsforExcelSetup.exe | Executable |
| 44 | Normal | DTools.exe | Executable |
| 45 | Normal | winsdk$_{w}$*eb.exe* | Executable |
| 46 | Normal | ClipboardHistory.exe | Executable |
| 47 | Normal | MEGAsync.exe | Executable |
| 48 | Normal | AnyDesk.exe | Executable |
| 49 | Normal | npp.7.6.Installer.exe | Executable |
| 50 | Normal | CVHP.exe | Executable |
| 51 | Normal | WinSCP-5.13.6-Setup.exe | Executable |
| 52 | Normal | coreftplite64.exe | Executable |
| 53 | Normal | eagleget-setup.exe | Executable |
| 54 | Normal | NetAssemblyInfo.exe | Executable |
| 55 | Normal | angrybird.exe | Executable |
| 56 | Normal | fdminst-lite.exe | Executable |
| 57 | Normal | sigcheck.exe | Executable |
| 58 | Normal | RBInternetEncodings500.dll | DLL |
| 59 | Normal | cryptolibcps-5.0.43.exe | Executable |
| 60 | Normal | Truecrypt.exe | Executable |

## 5. Conclusions and Future Work

In conclusion, this study proposes a malware detection system approach for malware based on N-grams and machine learning using the dataset collected from Virushare. We analysed the corpus of data using AI-based sandbox (SNDBOX) to generate behaviour reports that contained artifacts of malicious files. The next step proposed a representation algorithm that was utilized to extract features into a multi-dimensional vector space, including API calls and its arguments. In a later stage, we developed the features set using the N-grams method. Finally, they were transformed into binary vectors for training/testing machine learning classifiers such as Decision Tree, Random Forest, Logistic Regression and Naive Bayes. We measured the efficiency and efficacy of the classifiers using a confusion matrix. The experimental results indicate that Logistic Regression produces the best possible classification accuracy, compared to others.

In the future, we are planning to do further research on the capability of N-grams analysis in malware detection. Furthermore, we are also planning to take large datasets belonging to different categories of malware, such as trojan, botnet, worm, ransomware, spyware, etc. In future studies, we are also planning to increase the number of features from API calls to registry values, DNS requests, HTTPS requests, system changes, etc. to train/test it with deep learning algorithms.

**Author Contributions:** Conceptualisation, M.A. and S.S.; methodology, M.A.; software, M.A. and S.S.; validation, M.A., S.S. and B.G.; formal analysis, M.A.; investigation, M.A.; resources, S.S. and B.G.; writing—original draft preparation, M.A., S.S. and B.G.; writing—review and editing, S.S., B.G., G.B.; visualization, G.B., S.S. and B.G.; supervision, S.S. and B.G.; project administration, S.S., B.G. and G.B.; funding acquisition, S.S. All authors have read and agreed to the published version of the manuscript.

**Funding:** This project has received funding from the European Union's Horizon 2020 research and innovation pro-gramme under grant agreement No. 786698. This work reflects authors' view and Agency is not responsible for any use that may be made of the information it contains.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| TP | True Positive |
| TN | True Negative |
| FP | False Positive |
| FN | False Negative |

## References

1. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–48. [CrossRef]
2. Barría, C.; Cordero, D.; Cubillos, C.; Palma, M. Proposed classification of malware, based on obfuscation. In Proceedings of the 2016 6th International Conference on Computers Communications and Control (ICCCC), Oradea, Romania, 10–14 May 2016; pp. 37–44.
3. Bencsáth, B.; Pék, G.; Buttyán, L.; Félegyházi, M. Duqu: Analysis, detection, and lessons learned. In Proceedings of the ACM European Workshop on System Security (EuroSec), Bern, Switzerland, 10 April 2012.
4. Stone, R. A Call to Cyber Arms. 2013. Available online: https://science.sciencemag.org/content/339/6123/1026 (accessed on 27 June 2020).
5. Sonicwall. Sonicwall Cyber Threat Report: Threat Actors Pivot Toward More Targeted Attacks, Evasive Exploits. Available online: https://www.sonicwall.com/resources/2020-cyber-threat-report-pdf/ (accessed on 27 June 2020).
6. Pandalabs. Panda Security Launches Its Threat Insights Report 2020. Available online: https://www.pandasecurity.com/emailhtml/2004-report-threath-20/Threat-Insights-Report-en.pdf (accessed on 27 June 2020).

7. Kwon, D.; Kim, H.; Kim, J.; Suh, S.C.; Kim, I.; Kim, K.J. *A Survey of Deep Learning-Based Network Anomaly Detection*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 1–13.
8. Mohaisen, A.; Alrawi, O.; Mohaisen, M. Amal: High-fidelity, behavior-based automated malware analysis and classification. *Comput. Secur.* **2015**, *52*, 251–266. [CrossRef]
9. Alam, S.; Horspool, R.N.; Traore, I.; Sogukpinar, I. A framework for metamorphic malware analysis and real-time detection. *Comput. Secur.* **2015**, *48*, 212–233. [CrossRef]
10. Nauman, M.; Azam, N.; Yao, J. A three-way decision making approach to malware analysis using probabilistic rough sets. *Inf. Sci.* **2016**, *374*, 193–209. [CrossRef]
11. Yakura, H.; Shinozaki, S.; Nishimura, R.; Oyama, Y.; Sakuma, J. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In *Eighth ACM Conference on Data and Application Security and Privacy*; ACM: New York, NY, USA, 2018; pp. 127–134.
12. Ming, J.; Xin, Z.; Lan, P.; Wu, D.; Liu, P.; Mao, B. Impeding behavior-based malware analysis via replacement attacks to malware specifications. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 193–209. [CrossRef]
13. Vasiliadis, G.; Polychronakis.M.; Ioannidis, S. GPU-assisted malware. *Int. J. Inf. Secur.* **2015**, *14*, 289–297. [CrossRef]
14. Kim, K.S.; Shin, H.J.; Kim, H.S. A Bit Vector Based Binary Code Comparison Method for Static Malware Analysis. *JCP* **2018**, *13*, 545–554. [CrossRef]
15. Hassen, M.; Carvalho, M.M.; Chan, P.K. Malware classification using static analysis based features. In Proceedings of the IEEE Symposium Series on Computational Intelligence, Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–7.
16. Mithal, T.; Shah, K.; Singh, D.K. Case studies on intelligent approaches for static malware analysis. In *Emerging Research in Computing, Information, Communication and Applications*; Springer: Singapore, 2016; pp. 555–567.
17. Nagano, Y.; Uda, R. Static analysis with paragraph vector for malware detection. In Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM, Beppu, Japan, 5–7 January 2017; pp. 1–7.
18. Seideman, J.D.; Khan, B.; Vargas, A.C. Malware biodiversity using static analysis. In *International Conference on Future Network Systems and Security*; Springer: Cham, Switzerland, 2015; pp. 139–155.
19. Carlin, D.; O'Kane, P.; Sezer, S. Dynamic analysis of malware using run-time opcodes. In *Data Analytics and Decision Support for Cybersecurity*; Springer: Cham, Switzerland, 2017; pp. 99–125.
20. Kakisim, A.G.; Nar, M.; Carkaci, N.; Sogukpinar, I. Analysis and evaluation of dynamic feature-based malware detection methods. In *International Conference on Security for Information Technology and Communications*; Springer: Cham, Switzerland, 2018; pp. 247–258.
21. Vemparala, S.; Di Troia, F.; Corrado, V.A.; Austin, T.H.; Stamo, M. Malware detection using dynamic birthmarks. In *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*; ACM: New York, NY, USA, 2016; pp. 41–46.
22. Fang, Y.; Yu, B.; Tang, Y.; Liu, L.; Lu, Z.; Wang, Y.; Yang, Q. A new malware classification approach based on malware dynamic analysis. In *Australasian Conference on Information Security and Privacy*; Springer: Cham, Switzerland, 2017; pp. 173–189.
23. Liu, L.; Wang, B.S.; Yu, B.; Zhong, Q.X. Automatic malware classification and new malware detection using machine learning. *Front. Inf. Technol. Electron. Eng.* **2017**, *18*, 1336–1347. [CrossRef]
24. Shijo, P.; Salim, A. Integrated static and dynamic analysis for malware detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]
25. Afianian, A.; Niksefat, S.; Sadeghiyan, B.; Baptiste, D. Malware dynamic analysis evasion techniques: A survey. *arXiv* **2018**, arXiv:1811.01190
26. Santos, I.; Brezo, F.; Nieves, J.; Penya, Y.K.; Sanz, B.; Laorden, C.; Bringas, P.G. Idea: Opcode-sequence-based malware detection. In *International Symposium on Engineering Secure Software and Systems*; Springer:Berlin/Heidelberg, Germany, 2010; pp. 35–43.
27. Santos, I.; Devesa, J.; Brezo, F.; Nieves, J.; Bringas, P.G. Opem: A static-dynamic approach for machine-learning-based malware detection. In *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 271–280.

28. Zhang, X.; Sun, M.; Wang, J.; Wang, J. Malware Detection Based on Opcode Sequence and ResNet. In *International Conference on Security with Intelligent Computing and Big-Data Services*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 489–502.
29. Lakhotia, A.; Kumar, E.U.; Venable, M. A method for detecting obfuscated calls in malicious binaries. *IEEE Trans. Softw. Eng.* **2005**, *31*, 955–968. [CrossRef]
30. Brooks, R.A.; Maes, P. *Artificial life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*; MIT Press: Cambridge, MA, USA, 1994;
31. Anju, S.; Harmya, P.; Jagadeesh, N.; Darsana, R. Malware detection using assembly code and control flow graph optimization. In *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*; ACM: New York, NY, USA 2010; pp. 1–4.
32. Vinod, P.; Laxmi, V.; Gaur, M.S.; Kumar, G.; Chundawat, Y.S. Static CFG analyzer for metamorphic Malware code. In *Proceedings of the 2nd International Conference on Security of Information and Networks*; ACM: New York, NY, USA, 2009; pp. 225–228.
33. Altaher, A.; Ramadass, S.; Ali, A. Computer virus detection using features ranking and machine learning. *Aust. J. Basic Appl. Sci.* **2011**, *5*, 1482–1486.
34. Moser, A.; Kruegel, C.; Kirda, E. Limits of static analysis for malware detection. In Proceedings of the Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430.
35. Ab Razak, M.F.; Anuar, N.B.; Othman, F.; Firdaus, A.; Afifi, F.; Salleh, R. Bio-inspired for features optimization and malware detection. *Arab. J. Sci. Eng.* **2018**, *43*, 6963–6979. [CrossRef]
36. Ali, M.; Shiaeles, S.; Papadaki, M.; Ghita, B.V. Agent-based vs agent-less sandbox for dynamic behavioral analysis. In Proceedings of the 2018 Global Information Infrastructure and Networking Symposium (GIIS), Thessaloniki, Greece, 23–25 October 2018; pp. 1–5.
37. Ali, M.; Shiaeles, S.; Clarke, N.; Kontogeorgis, D. A proactive malicious software identification approach for digital forensic examiners. *J. Inf. Secur. Appl.* **2019**, *47*, 139–155. [CrossRef]
38. Santos, I.; Penya, Y.K.; Devesa, J.; Bringas, P.G. N-grams-based File Signatures for Malware Detection. *ICEIS (2)* **2009**, *9*, 317–320.
39. Nakazato, J.; Song, J.; Eto, M.; Inoue, D.; Nakao, K. A novel malware clustering method using frequency of function call traces in parallel threads. *IEICE Trans. Inf. Syst.* **2011**, *94*, 2150–2158. [CrossRef]
40. Liangboonprakong, C.; Sornil, O. Classification of malware families based on n-grams sequential pattern features. In Proceedings of the 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA), Melbourne, Australia, 19–21 June 2013; pp. 777–782.
41. Kolter, J.Z.; Maloof, M.A. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.* **2006**, *7*, 2721–2744.
42. Shabtai, A.; Moskovitch, R.; Feher, C.; Dolev, S.; Elovici, Y. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Secur. Inf.* **2012**, *1*, 1. [CrossRef]
43. Moskovitch, R.; Feher, C.; Tzachar, N.; Berger, E.; Gitelman, M.; Dolev, S.; Elovici, Y. Unknown malcode detection using opcode representation. In *European Conference on Intelligence and Security Informatics*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 204–215.
44. Ben-Haim, Y.; Tom-Tov, E. A Streaming Parallel Decision Tree Algorithm. *J. Mach. Learn. Res.* **2010**, *11*, 849–872.
45. Zhu, L.; XU, Y. Application of C4.5 algorithm in unknown malicious code identification. *J. Shenyang Univ. Chem. Technol* **2013**, *27*, 78–82.
46. Tian, R.; Islam, R.; Batten, L.; Versteeg, S. Differentiating malware from cleanware using behavioural analysis. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software, Nancy, Lorraine, France, 19–20 October 2010; pp. 23–30.
47. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv. (CSUR)* **2008**, *44*, 1–42. [CrossRef]
48. VirusShare. Malware Repository Platform. Available online: https://virusshare.com (accessed on 6 December 2019).
49. Salehi, Z; Ghiasi, M.; Sami, A. A Miner for Malware Detection Based on API Function Calls and Their Arguments. In Proceedings of The 16th CSI International Symposium on Artificial Intelligence and Signal Processing, Shiraz, Fars, Iran, 2–3 May 2012; pp. 563–568.

50. Okamoto, K.; Tamada, H.; Nakamura, M.; Monden, A.; Matsumoto, K.I. Dynamic software birthmarks based on API calls. *IEICE Trans. Inf. Syst.* **2006**, *89*, 1751–1763.
51. Sami, A.; Yadegari, B.; Rahimi, H.; Peiravian, N.; Hashemi, S.; Hamze, A. Malware detection based on mining API calls. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; ACM: New York, NY, USA, 2010; pp. 1020–1025.
52. Pektaş, A.; Eriş, M.; Acarman, T. Proposal of n-gram based algorithm for malware classification. In Proceedings of the Fifth International Conference on Emerging Security Information, Systems and Technologies, Nice/Saint Laurent du Var, France, 21–27 August 2011; pp. 7–13.
53. Raff, E.; Zak, R.; Cox, R.; Sylvester, J.; Yacci, P.; Ward, R.; Tracy, A.; McLean, M.; Nicholas, C. An investigation of byte n-gram features for malware classification. *J. Comput. Virol. Hacking Tech.* **2018**, *14*, 1–20. [CrossRef]
54. Sayfullina, L.; Eirola, E.; Komashinsky, D.; Palumbo, P.; Miche, Y.; Lendasse, A.; Karhunen, J. Efficient detection of zero-day android malware using normalized bernoulli naive bayes. In Proceedings of the IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 198–205.
55. Garg, V.; Yadav, R.K. Malware Detection based on API Calls Frequency. In Proceedings of the 4th International Conference on Information Systems and Computer Networks, Mathura, India, 21–22 November 2019; pp. 400–404.
56. Salehi, Z.; Sami, A.; Ghiasi, M. Using feature generation from API calls for malware detection. *Comput. Fraud Secur.* **2014**, *9*, 9–18. [CrossRef]
57. Kumar, B.J.; Naveen, H.; Kumar, B.P.; Sharma, S.S; Villegas, J. Logistic regression for polymorphic malware detection using ANOVA F-test. In Proceedings of the International Conference on Innovations in Information, Embedded and Communication Systems, Coimbatore, India, 17–18 March 2017; pp. 1–5.
58. Devesa, J.; Santos, I.; Cantero, X.; Penya, Y.K.; Bringas, P.G. Automatic Behaviour-based Analysis and Classification System for Malware Detection. *ICEIS* **2010**, *2*, 395–399.