Applied Data Science and Analysis
Vol.2023, **pp**. 126-142
DOI: https://doi.org/10.58496/ADSA/2023/012 ; ISSN: 3005-317X
https://mc04.manuscriptcentral.com/adsa

Research Article

# Application of Sequential Analysis on Runtime Behavior for Ransomware Classification

Chee Keong NG [1, *], Tahsien Al-Quraishi [1], Tony De Souza-Daw [1]

[1] *Victorian Institute of Technology, Department of Information Technology, Melbourne, Victoria, Australia*

**ARTICLE INFO**

Article History
Received 02 Sep 2023
Accepted 27 Oct 2023
Published 23 Nov 2023

Keywords
Ransomware
Machine Learning
Dynamic Analysis

**ABSTRACT**

The unprecedented development and massive proliferation of Internet technology, computing /storage capability and emerging business model, like cloud and IoT, brings not only incredible changes to human lifestyle but also numerous, complex and continuing cyber security threats, one noticeable example among them is malware. Static analysis has been popular and widely used in many anti-virus engine. However, static analysis can be avoided using techniques such as packing, polymorphism, and metamorphism. In this paper, I propose a novel method focuses on the feature extraction, which exploits the inherent encryption behaviour of ransomwares. Specifically, runtime malicious sequential analysis is adopted to establish the desired feature set, which further facilitate the identification of the inherent encryption function. With the proposed method, an accuracy level of 96% was achieved

## 1. INTRODUCTION

Despite the fact, the unprecedented and massive proliferation of the Internet, applications and computing has enhanced human lifestyle, it comes along with numerous complex threats[1]. Threats including malicious software, popularly referred to as malware. It is designed to disrupt the operations of a computer, collect sensitive information, or gain access to private systems[2]. The emergence of malware can be traced back to the late seventies[3], after which the world has continued to witness advanced, evolved and mutated malware types with the capacity to inflict considerable system damage. The most popular malware categories include ransomware, Worms, Viruses, and Trojans[4]. And the focus of this paper is on ransomware and malicious sequential analysis of its runtime behaviour. Ransomware is a special form of malware that encrypts files in victims computer and creates demands message for payment to decrypt the infected files.

In 2016 saw organized criminal gang extensively used malware. In fact, the mass media labelled 2016 “the year of the ransomware”[5]. Growing ransomware’s popularity is an excellence get-rich and win-win marketing strategy through extortion when compare to other malware forms. Ransomware effects are irreversible when appropriate encryptions are used [6]. Given this irreversibility and the extortionist approaches used by ransomware writers, victims often pay amounts between USD 10,000 and USD 17,000 for corporates and between USD 300 and USD 700 for individuals[7]. Despite recommendations against paying the ransom, it is often the only way for most corporates and individuals to re-access their locked or encrypted files[8].

While researchers have been working hard to detect ransomware, malware writers have been focusing on developing workaround to avoid detection. The most common detection method used are based on signatures. This method entails the comparison of signature string extracts from a sample recorded in its database. Once the string matches a record, the sample will be deemed malicious. Clearly, this method is only efficient to existing known malware. Moreover, it can be avoided easily by using obfuscation technique[9].

Another simple detection method is static analysis. As a passive approach, static analysis studies the opcode and the properties of the malware with the hope of extracting the structural properties from the binary strings and source codes that inimitably denote the malware[10-11]. Nevertheless, malware writers have adopted the technique of packing and other obfuscation technique to prevent the revealing of the actual payload. Furthermore, if the writers choose to use packers to further compress and encrypt their payloads, static analysis also fails to perform the detection[11].

*Corresponding author. Email: ckng@kpkt.gov.my

These limitations associated with static analysis triggered the birth of dynamic analysis. The purpose of dynamic analysis is to understand the malware through its behaviour. Dynamic analysis is performed by executing the malware in a safe virtual environment to monitor and log the behaviour and changes to the system. Researchers or analysts perform the behaviour analysis in a simulated sandbox or debugger platform to understand the activities of the targeted malware[12]. The dynamic analysis technique shows more effectiveness in the identification of the actual intention of the malware under observation[13]. Following the approach of dynamic analysis and according to the final state of the virtual environment after execution, we classify the ransomware as three different classes: no-execution, partially executed and fully executed.

The no-execution class implies that the sandbox has not been infected. This is common with ransomware that is equipped with anti-virtualization techniques. The partially executed outcome shows that ransomware is in the midst of the process. Since some ransomware may take longer time to run than others, so it is common that they are in the partially executed state because researchers had not allocated enough time for them. In this case, the ransomware is still running and the ransomware note is not displayed in either the background or the foreground. The uniqueness of most ransomware are that they will self-terminate once they display the ransomware message. With the fully executed outcome, the ransomware with the ransom note display.

This is one of the great distinctions between ransomware and other malware. The life of ransomware is short. It runs, encrypts, and displays a ransom message before exit. The whole execution takes an average of 3 minutes depending on the amount of content in the victim's computer. The full-execution outcome is the most desired because it enables the analyst/researcher to capture the full picture of the behaviour of ransomware by allowing an appropriate amount of time to run.

The features extracted from behavioural analysis fall under two categories. They are API calls and file system changes. The API calls extract information such as DLL used, n-gram feature, sequence of API and API graph, and they all can be mathematically translated into features. This extraction of the feature can be affected by adding extra and non-sensical API calls. However, feature extraction from file system changes is not affected. This is simply because it ignores the extra code or function added to malware.

Our work focuses on extracting features from a higher abstraction level which is capable of providing high, semantically meaningful information. The proposed framework converts the closely related behaviour pattern from ransomware into distinctive feature. The use of high-level abstraction in behavioural analysis is vital when analysing ransomware dynamically. Unlike other malware, the damage caused by ransomware is observable visually after the system is compromised. Ransomware often leaves numerous traces of its present actions, which can provide useful information about its behaviour. This approach provides better ransomware classification.

In contrast, extant research presents with three significant limitations that are worth noting. Firstly, most of the researchers' proposed methods or models deal with malware in a very general sense or even do not include ransomware for the detection. Secondly, the features selected for the detection are massive, for example, more than 2000. Finally, most of the features (n-gram) used are general in nature and not tailored to a specified malware class. In an attempt to overcome these significant limitations, we initiated two measures. First, we extract features that are specific to ransomware using high abstraction level information. Second, we introduce a small feature group for maximum performance.

The proposed method contributes to the literature and practical understanding of ransomware by introducing a small, but specialized feature to increase the detectability of different ransomware class by using Pattern Sequence Extraction for feature engineering.

The rest of the paper is organized as follows. Section II will describe the related work. The third section provides a detailed description of the method used to extract the high-level features and the machine learning model. In section IV, the results are compared and discussed. Section V is the discussion where thought, decision and limitation of the proposed method brought forward. The last section concludes the finding of the proposed method.

## 2. RELATED WORK

As mentioned earlier, there are three primary techniques for malware detection: signature-based, static-based, and behavioural-based methods. The signature-based method has been widely used in many anti-malware applications[14] due to its simplicity and ease of feature extraction. This method involves comparing the extracted signature with the one stored in the database. If there is no match, the test sample is classified as benign.

In static-based detection, features are extracted by analysing the opcode, binary string, and source code. Unique properties identified during this analysis are extracted as features. However, both the signature-based and static-based methods have drawbacks and often fail to detect variants of known or unknown malware. Malware developers employ obfuscation techniques such as polymorphism and metamorphism[15], which are highly effective and still prevalent in modern malware samples.

On the other hand, behavioural-based detection monitors the behaviour of suspicious samples for unusual activities. If the activity attempted by the sample is considered abnormal or unusual, it is labelled as malicious[14]. The features for behavioural-based detection are extracted dynamically, leading to the term "dynamic analysis." This method allows the sample to be executed in a controlled environment called a sandbox[12], with most sandboxes implemented in a virtual environment using virtualization software like VMWare or Virtualbox[16].

One of the most commonly used features for behavioural detection is API call analysis[17]. This approach involves monitoring the requests made by Windows executables in the operating system. API calls can be extracted either statically or dynamically, with dynamic extraction being preferable due to its ability to evade evasion techniques like encryption and packing[16].

Ronghua[18] proposed a method that relies solely on API calls as features to differentiate between benign and malicious samples. This straight-forward approach is effective, achieving an accuracy of up to 97%. However, it fails to reveal the sequence of malware behaviour, and malware developers can bypass detection by inserting dummy API calls[19].

The work in[20] represents one of the earliest attempts to detect malicious files by examining the sequence of API calls. Hofmeyr20 argued that short sequences of API calls are more likely to belong to legitimate files. Dolly[21] also utilized API call sequences for their model, capturing these calls and employing feature selection to filter out distinct API sequences.

The sequence of API selection is based on the odds ratio selections algorithm on four API grams. The experimental results were better than those reported in[18]. Notably, Dolly's proposal also recorded the number of occurrences of the API call sequence from the executable as a feature. However, this method fails to detect polymorphic and unknown malware. Additionally, as reported in[19], malware developers can manipulate the sequence of API calls by adding insignificant APIs in between the main API calls to disrupt the sequence.

To address this problem, Y. Ki et al.[19] Introduced a sequencing method to group DLLs or API calls and use them to identify encryption activity. However, this method still suffers from the issue of intentionally adding insignificant API calls. To mitigate this problem, the results are filtered from the file activities to produce a cleaner sequence.

Ki[19] also proposed another API call sequence extraction method that utilizes DNA sequence alignment algorithms like Multiple Sequence Alignment (MSA) and Longest Common Subsequence (LCS) to extract API call sequence patterns. This sequence algorithm is classified into global alignment and local alignment. Global alignment is useful for finding sequences of the most similar length and strings, while local alignment identifies highly similar subsequence in a given sequence. This method is capable of detecting unknown malware[19].

According to [21], API mining is one of the more popular methods for selecting API calls. Ye[21] introduced an intelligent malware detection system that employs an Objective-Oriented Association (OOA) mining algorithm to generate rules from extracted API calls. Ahmadi[22] used the iterative pattern mining method to extract frequent iterative API call patterns and included the Fisher score as part of the feature selection. Fan 2 proposed the Malicious Sequential Pattern Extraction (MSPE) mining algorithm to identify malicious API call sequences. MSPE adopts the concept of OOA to discover sequential patterns with malicious characteristics.

Another approach, known as API dependency, has been introduced to detect highly obfuscated malware. Based on the idea that new malware is a recreation of a previous sample, Ammar[23] proposed the use of API call dependency to build an API call graph for malware detection. This graph tracks and presents the flow of values between procedures in graphical form to be understood by human and visualise. The system extracts API calls and their dependencies from the sample, with each API call representing a node linked by dependencies. The proposed system adopts the Longest Common Subsequence (LCS) algorithm for graph matching. Importantly, API calls represent a low-level abstraction, and it is generally recognized that behaviour analysis at a higher abstraction level can be used to extract the fingerprint of malware[24]. This fingerprint comprises system state changes such as file changes and process creation. Rather than assessing malware solely based on low-level system events, Bailey et al.[24] Focus on understanding what the malware does to the system. This is crucial for assessing the extent of damage caused. Aziz[25] proposed a behavior-based automated malware analysis system that profiles the malware sample under consideration using file system, network activity logging, and registry monitoring. Researchers in [26] also proposed a model capable of distinguishing benign from malicious activity by capturing file interactions with the system. This interaction focuses on file system and registry activities such as reading, writing, and executing files.

Recently, works such as[27] have attempted to extract features from both high and low-level system events. Sandboxes, such as Cuckoo for API call extraction and Virmon, a Windows kernel-level notification routine for behaviour analysis, have been implemented. These methods have achieved accuracy levels of up to 92% using the online machine learning service Jubatus. The lower accuracy compared to [18] and [21] can be attributed to the dataset used, which contains more recent and sophisticated malware. Another contributing factor is the massive number of features (N-grams) used, which can reach into the thousands and include a large proportion of unimportant features[23].

## 3. METHODOLOGY

In principle, the operation of ransomware occurs in three stages: the first stage searches for the target file, the second stage encrypts the target file, and the third stage displays the ransom message. These stages illustrate the behaviour of the ransomware, and features are extracted based on them.

### 3.1 Feature of Encryption Process

It is worth mentioning that while some older versions of ransomware may use the search function to locate target files within a folder for efficiency reasons, most of them have abandoned this approach. On the other hand, the newest ransomware adopts the concept that "all files will be encrypted anyway", and therefore, it encrypts any files it encounters on the hard drive. As a result, recent ransomware only requires two steps, as shown in the Figure 1.

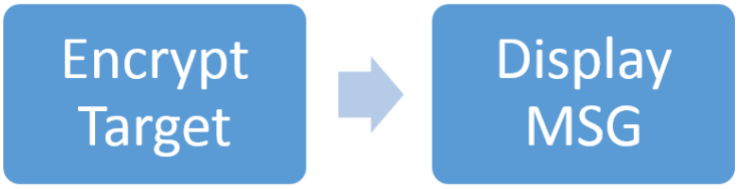


Fig. 1. The basic operation for ransomware

Different ransomware variants employ various encryption methods to achieve their objectives. There are three types of encryption methods. The first method involves modifying the targeted file to render it inaccessible, even though there are no observable visual differences. This method can be found in the Spora ransomware family, although it is not widely used. The second method reads, encrypts, and then deletes the target file. In this approach, the target file is not permanently removed from the disk, and there is a high chance of file recovery. The last method is the most advantageous from the attacker's perspective because the chance of retrieving the file is practically zero with a valid key/ password. This method reads, encrypts, and overwrites the target file with the encrypted version, permanently removing the original file from the disk. This method is popular and is adopted by many ransomware families, such as Cerber and TeslaCrypt.

The proposed model consists of two sections. The first section is used to monitor how the sample interacts with the sandbox, and the second section involves the feature extraction process. In the first section, bait files, which are also the targeted files for the analysed ransomware, are used to entice the ransomware into compromising them. These files are often in popular formats. For this research, we selected MS Word (.docx), ZIP (.zip), and image (.jpg) file formats to monitor the behaviour of the ransomware. The sample is executed within a safe and secure virtual sandbox environment using the Noriben sandbox[29]. After the monitoring process, the malware activities are logged and extracted for the second section.

The feature extraction process begins by identifying the sequence of file activities related to the encryption process in the log file. This is done by locating the bait files used. Each sequence may vary in length due to the introduction of noise or the sample's multi-encryption capabilities. The excess activities are filtered to determine the precise encryption steps for feature formulation. The method for extracting features related to the encryption function is defined below.

Figure 2 shows the first step of the feature extraction process. A sample contains a series of file activities for all the bait files. Let sample be Sample $= \{b1, \ldots, bt\}$, where b1 is the first file activity and $bt$ is the last activity. The encryption activities need to be identified and extracted. The activities are organised into sets of activities for the encryption process for each bait file. The encryption activities in each bait file is formulated as $s = \{bi, bi+1, \ldots, bi+n\}$, where s is referred to a bait file is the j-th bait file, i is the start point of the encryption activity associated with j, and n is the last position of the encryption process for file j. In this way, an encryption sequence for multiple bait files can be denoted as $S = \{s1, s2, \ldots, sm\}$, where length(sj) and length(sj+1) are not necessarily the same (note: the i refers to any location). In order to filter and retain the actual activity of the encryption function, bi is associated with bm, if path within bi = bm = bait file name of s. This step is illustrated in Figure 3.

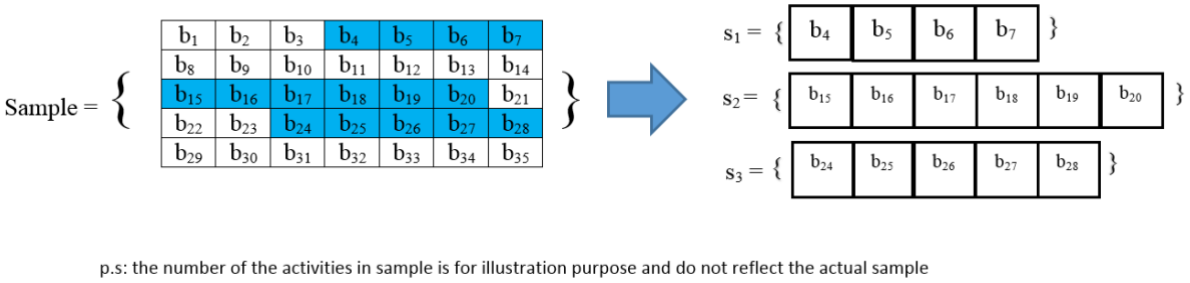


Fig. 2. Step 1: Extract the encryption sequence from the bait files. The sequences for encryption activities are identified. This is demonstrated by the blue coloured boxes. Each sequence is extracted and they are vary in length

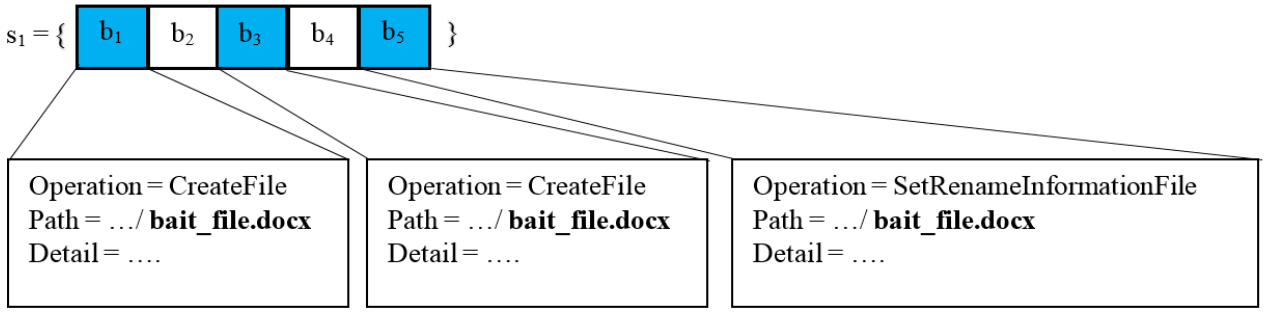


Fig. 3. Identification and Organization of Encryption Activities Across Multiple Bait Files

The third step is to formulate the feature using the information extracted from the first two steps. Table I shows the formula for each feature.

TABLE I. FEATURE EXTRACTED FROM THE ENCRYPTION PROCESS

| S/No | Feature | Description | Formula |
|---|---|---|---|
| 1 | Enc1 ... Enc6 | Each Enc represent a process activity | bi, where path of bait file in bi = path of bait file in bi+1 |
| 2 | EncryptionStep | Number of activities for the encryption process | length(s) |
| 3 | EncryptionMin | Minimum number of activities | Min(S) |
| 4 | EncryptionMax | Maximum number of activities for the encryption process with noise | Max(S) |
| 5 | EncryptionMean | The average number of activities for the encryption process | $\frac{\sum_{i=1}^{n}(s_i \dots s_n)}{numofS}$ |

## 3.2 Ransomware Message Feature

Unlike other malware where secrecy is vital, ransomware ensures that its victim receives the message of its infiltration after all target files are encrypted. In other words, ransomware will reveal itself via its message in the final stage of its operation. The message is displayed in the following manner:

- Execute notepad to display the ransomware message
- Execute the default browser to display the ransomware message;
- Execute the default photo viewer to display the ransomware message;
- Change the background of the desktop to display the ransomware message.

Most ransomware will select one or more of these methods to ensure its victims know of its presence. The form of message can be in text (txt),picture (jpg or png) or hypertext (HTML). After analyzing the log record of several ransomware samples, we are able to identify a few messaging behaviors such as the creation of the message file, number of file created and type of file created. We also encountered few ransomware, which create the message file before the encryption process.

Most ransomware create and store their messages in the target directories after they have encrypted the file in those directories. The ransomware note is a useful feature for distinguishing ransomware families. The features selected to represent the ransom message are as listed below:

- The number of message formatted in;
- Position of the message;
- The message file.

### 3.3 Proposed Model

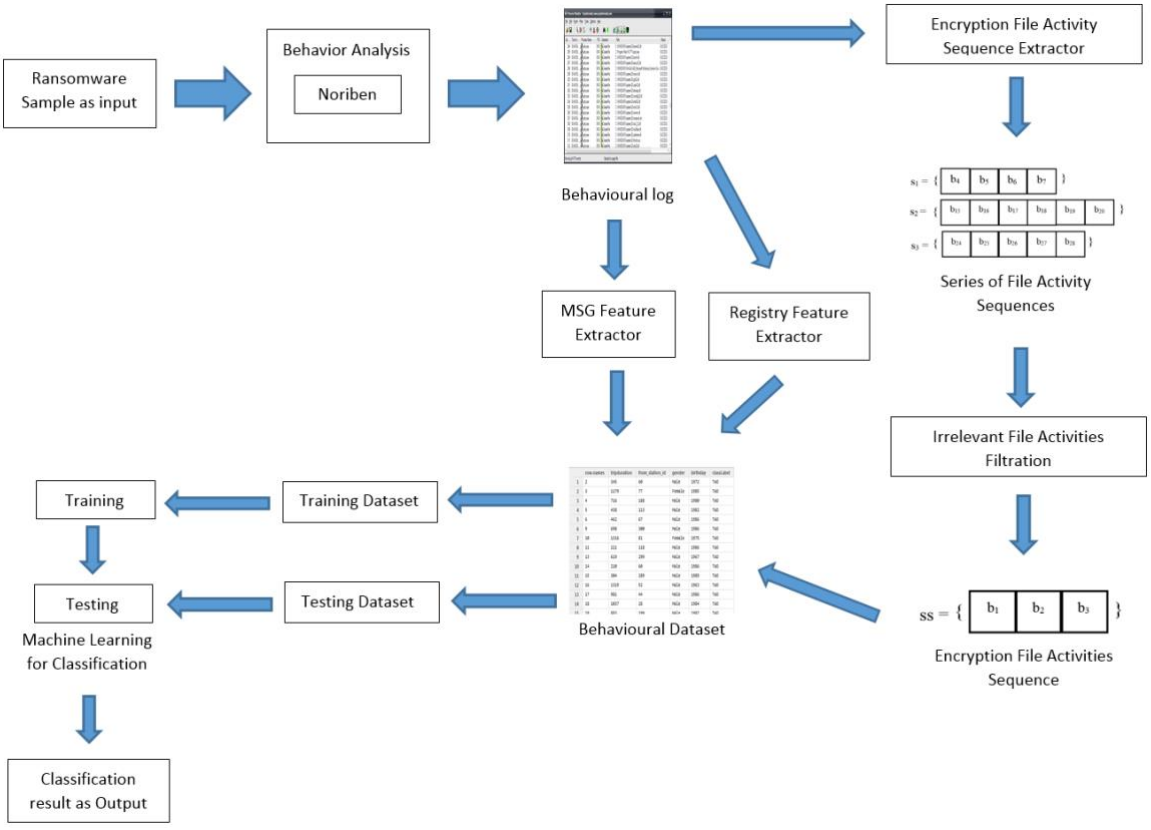


Fig. 4. Proposed Model

The proposed model consists of 3 main stages. The first stage is behavioural extraction. During this stage, the behaviour of the ransomware is captured by executing the sample in a sandbox. The report log is extracted from the sandbox for the next stage. The next stage is the feature extraction stage. The feature extraction is described in section 3.1 in detail. Virustotal API[30] is used to determine the label of each sample. The final stage of the proposed system requires the prepared dataset to be fed into the machine learning algorithms for the result. The dataset is divided into training set and test set. While the training set is used to obtain the classification model, the testing set is used for evaluation purposes. Figure 4 illustrates the overview of the proposed method.

# 4. EXPERIMENT AND RESULT

## 4.1Experiment Setup

The experiment is performed on a higher-performance laptop. The laptop is equipped with Windows 7, 2.60GHz i7-3720QM CPU, 12 GB of RAM and 512 GB of SSD hard drive. An automated Sandbox, Noriben, is also implemented to collect the runtime behaviour of the ransomware sample. It was deployed to extract windows kernel-level notification routines.

Noriben is a simple sandbox written using python. The analysis in the sandbox adopts procmon. The procmon extracts information at the high abstraction level. This information includes the state changes of the operating system resources such as file registry, process/ thread and network activities. These states are saved into a log file.

## 4.2 Ransomware Dataset

The sample for the dataset is obtained from virusShare Malware Platform31 and Malware-Traffic-Analysis[29]. VirusShare is one of the largest repositories that provides virus sample for research purpose. VirusShare has grouped the samples according to the malware type and members are able to download the samples in form of zip file format. Despite this, testing and analysis are necessary since most of the samples do not exhibit the ransomware behaviours. Most of the ransomware samples presented in the zip file are prior to 2016 and the oldest sample goes as far as 2013.

The ransomware samples in the Malware-Traffic-Analysis repository are categorized in accordance with the class type of ransomware. The samples selected are between 2016 and 2017. The sample downloaded from the Malware-Traffic-Analysis repository provides a more accurate label for the dataset. However, the repository has a fairly limited amount of ransomware samples. Table 2 illustrate the proposed feature set.

TABLE II. PROPOSED FEATURE SET

| Feature Description | Feature Description |
|---|---|
| RegSetValue | Number of registry set value activities |
| RegDeleteKey | Number of registry delete key activities |
| RegDeleteValue | Number of registry delete value activities |
| SetSecurityFile | Number of set security file activities |
| TotalReg | Total number of RegSetValue, RegDeleteKey and RegDeleteValue |
| ProcessCreate | Number of process created |
| Enc1 | First file activity for the encryption process |
| Enc2 | Second file activity for the encryption process, if any |
| Enc3 | Third file activity for the encryption process, if any |
| Enc4 | Forth file activity for the encryption process, if any |
| Enc5 | Fifth file activity for the encryption process, if any |
| Enc6 | Sixth file activity for the encryption process, if any |
| EncryptionStep | Number of activities for the encryption process |
| EncryptionMin | Minimum number of activities |
| EncryptionMax | Maximun number of activities for the encryption process with noise |
| EncryptionMean | The average number of activities for the encryption process |
| NumOfMsg | Number of activities for the encryption process |
| MsgFile | Number of activities for the encryption process |
| MsgPos | Number of activities for the encryption process |
| TCPNetwork | Number of activities for the encryption process |

The samples downloaded contain PE and DLL from virusShare. Since the sandbox used recognized executable sample file only, PE files are selected. The sample is allowed to be executed in a virtual Windows 7 environment with .NET framework 4.5.1 installed for 3 minutes until the ransom message is shown. Commonly used files such as Microsoft Word documents,

pictures, and zip files are kept in the sandbox to allow ransomware to unveil its full potential. The ransomware samples that run successfully are then included into the dataset.

Finally, our dataset is comprised of 1008 ransomware samples, 19 distinct features and 13 different classes of ransomware. Table II describes the 13 features that are extracted by analysing the behaviour pattern of ransomware.

### 4.3 Evaluation Metric

To assess the performance of the proposed feature, the following evaluation measures were used in the result:

- Accuracy (overall and balanced)
- Precision (overall and balanced)
- Recall (overall and balanced)
- F2 Measure (overall and balanced)

These indicators are standard means of measuring that are used to determine the effectiveness of the proposed model. Unlike the evaluation metric for binary classification, the calculation used in the evaluation metrics for the multiple classification is more complex. Before we proceed to the formulation, let's consider the following:

- q is the number of families.
- $Y_i$ is the ground truth class assignment of the $i^{th}$ sample.
- $x_i$ is the $i^{th}$ sample.
- $h(x_i)$ is the predicted class for the $i^{th}$ sample.
- |.| is the cardinality of a set.

Let $S = \{(x_i, Y_i)|1 \leq i \leq q)\}$ be the test set. For the j-th class $y_j$, four basic quantities characterizing the classification performance on this class can be defined based on h(.)[32]:

$$TPj = |\{xi|yj \in Yi \wedge yj \in h(xi), 1 \leq i \leq q\}|; \quad (1)$$
$$FPj = |\{xi|yj /\in Yi \wedge yj \in h(xi), 1 \leq i \leq q\}|; \quad (2)$$
$$TNj = |\{xi|yj /\in Yi \wedge yj /\in h(xi), 1 \leq i \leq q\}|; \quad (3)$$
$$FNj = |\{xi|yj \in Yi \wedge yj /\in h(xi), 1 \leq i \leq q\}|. \quad (4)$$

$TPj, FPj, TNj \; and \; FNj$ represent the true positive, false positive, true negative, and false negative with respect to $y_j$.

The accuracy test is used to determine the number of samples being classified correctly. This is calculated after comparing the predicted results with the ground truth. The accuracy test for class j is defined by the number of samples being correctly classified in class j divide by total samples in class j :

$$ACC_j = \frac{|Y_j \cap h(x_j)}{Y_j \cup h(x_j)}, \quad (5)$$

Precision is the ratio of how much of the predicted sample is correct. The numerator finds how many classifications in the predicted vector has common with the ground truth, and the ratio computes, how many of the predicted true classifications are actually in the ground truth. The precision is used to measure the relevance of the result.

Recall is the ratio of how many of the actual classifications are predicted, The numerator finds how many classifications in the predicted vector has in common with the ground truth, it then finds the ratio to the number of actual labels and getting the fraction of the actual labels are predicted.

Recall is a measure of how many truly relevant results are returned. The precision and recall procedures for class j are achieved based on formulas shown below respectively:

$$P_j = \frac{TP_j}{TP_j + FP_j}, \quad (6)$$

$$R_j = \frac{TP_j}{TP_j + FN_j}, \quad (7)$$

In this study, we focus on ransomware classification, no clean sample is involved in any of the experiment. The true positive means the classification of the target ransomware to the targeted ransomware family and false positive refers to the classification of the wrong sample to the targeted ransomware. False negative is interpreted as the misclassification of the target ransomware to the wrong family.

F1 score is the harmonic average of the precision and recall. F1 is used to reveal the balance between the precision and recall procedures. The formula for achieving this is shown below:

$$F1 = 2 * \frac{P*R}{P+R}, \tag{8}$$

Each test class has a different sample size and needs to be calculated according to the proportion over the entire sample size. It can be formulated as:

$$prop_j = \frac{num\_of\_samples}{entire\_samples}, \tag{9}$$

The formulas to calculate the overall result for precision, recall, F1 and accuracy are as follows:

$$P_o = \sum_{i=1}^{n}(prop_i * P_i), where\ (1 \leq i \leq q) \tag{10}$$

$$R_o = \sum_{i=1}^{n}(prop_i * R_i), (1 \leq i \leq q) \tag{11}$$

$$F1_o = \sum_{i=1}^{n}(prop_i * F1_i), \text{where}(1 \leq i \leq q) \tag{12}$$

$$ACC_o = \sum_{i=1}^{n}(prop_i * ACC_i), \text{where}(1 \leq i \leq q) \tag{13}$$

where o represents the consolidated end result for its representative.

The result for performance measurement is computed using SKlearn which is one of the machine learning package from Python. SKlearn is a useful tool that provides high consistency in the computed result.

### 4.4 Machine Learning Implementation

In order to demonstrate that the proposed feature can produce high-performance results, 8 machine learning models are adopted. The machine learning models are part of the Python SKlearn package. The selected machine learning models have a good reputation and are widely used in many research papers27. They are:

- Cat Boosting[33] is a machine-learning model that addresses the challenges posed by categorical features in data. It is designed to handle categorical variables natively, without the need for preprocessing or one-hot encoding, which sets it apart from many other machine learning algorithms.
- KNN[34] is short for k-nearest neighbour. KNN is a non-parametric and lazy learning algorithm. It uses the data point from training samples to predict the new sample point by matching it to the nearest k samples. k is the number of nearest training sets to the testing sample.
- Decision Tree[35] is a classification-decision tree for the given dataset by recursive partitioning the data. The decision is grown using depth-first strategy. The algorithm considers all the possible tests that can split the data set and selects a test that gives the best information gain.
- Histogram Boosting[36]: It uses a histogram-based approach for gradient boosting. It efficiently handles both numerical and categorical features, scales well with large datasets, and provides competitive performance for classification tasks while offering options for regularization to prevent overfitting.
- Extra Tree Classifier[37] stands for extremely randomized trees. This model splits nodes by choosing cut-points at random. The whole learning sample is used to grow the trees.
- Random Forest[38] is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.
- GX Boosting[39] stands for "Extreme Gradient Boosting". It is a powerful and widely used gradient-boosting machine-learning algorithm. It is known for its efficiency, scalability, and excellent predictive performance.
- Gradient Boosting[40] builds a model in a stage-wise fashion and generalizes them by allowing optimization of an arbitrary differentiable loss function.

Various settings were tested on the machine model to accomplish the best performance. After extensive experimentation, the performance achieved for machine learning, such as the Histogram Boosting, of each setting did not differ significantly from the default setting. The setting for the other machine learning did reveal better performance from the default setting.

For decision tree algorithm, the default setting of 5 max depths recommended in the SKlearn python package was employed, as the optimized setting. The optimized setting for the random forest learning model is 50 trees (n_estimators), 50 stages (n_estimators) for gradient boosting, 3 neighbours for KNN, C is 1 for SVM, 50 trees for ExtraTreeClassifier, and C for Logistic Regression is 0.001.

### 4.5 Accuracy and Classification Result

After the feature extraction process, the dataset was normalized and converted to more orthogonal directional using the Principal Component Analysis algorithm. The dataset is then split into two training and testing sets. The training and testing set are fed into the machine learning model and the results are displayed in Table 3.

TABLE III. OVERALL RESULT FOR EACH MACHINE LEARNING MODEL

| | ACC (Training) | ACC (Testing) | Precision | Recall | F1 |
|---|---|---|---|---|---|
| Histogram Boosting | 0.9834 | 0.9876 | 0.9877 | 0.9876 | 0.9876 |
| Cat Boosting | 0.9768 | 0.9802 | 0.9799 | 0.9802 | 0.9796 |
| GX Boosting | 0.9818 | 0.9851 | 0.9854 | 0.9851 | 0.9851 |
| Extra Tree | 0.9967 | 0.9604 | 0.9460 | 0.9595 | 0.95 |
| Classifier Random Forest | 1 | 0.9581 | 0.94076 | 0.9577 | 0.949 |
| Gradient Boosting | 1 | 0.9431 | 0.9273 | 0.9429 | 0.9354 |
| KNN | 0.9603 | 0.9480 | 0.9324 | 0.9480 | 0.9393 |
| Decision Tree | 0.9288 | 0.9346 | 0.9186 | 0.9337 | 0.9252 |

The proposed model is able to achieve good training and testing performance, as shown in Table 4. The overall accuracy for training and testing shown in the table reveals that high accuracy results ranging from 92% to 99%. The high accuracy rate reveals high true positives for classifying the ransomware in their respective classes. Most of the machine learning algorithm models can achieve precision, recall, and F1 scores of 90% and above, which also reveal a low false negative and false positive. Among the machine learning models, Histogram Boosting Classifier and GX Boosting Classifier recorded the best performance. We further probe the details of them in our dataset and the results are reported in Tables IV and V . Both models are able to achieve an accuracy 98% and has a margin of 0.7 difference. The next runner-up is GX Boosting and Extra Tree Classifier which are able to achieve above 95% accuracy. In the following discuss, we focus on the result obtained from the 3 best performing machine learning models.

TABLE V. PERFORMANCE FOR HISTOGRAM BOOST CLASSIFIER

| Family | Num of Sample | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|---|
| Cerber | 107 | 1 | 1 | 1 | 1 |
| Locky | 22 | 1 | 1 | 1 | 1 |
| Bandarchor | 7 | 1 | 1 | 1 | 1 |
| GlobeImposter | 13 | 1 | 1 | 1 | 1 |
| Jaff | 15 | 1 | 1 | 1 | 1 |
| CryptoShield | 13 | 1 | 1 | 1 | 1 |
| TeslaCrypt | 286 | 0.98 | 0.99 | 0.96 | 1 |
| Spora | 14 | 0.67 | 1 | 0.8 | 1 |
| Wannacry | 10 | 1 | 1 | 1 | 1 |
| Gpcoder | 18 | 0.67 | 0.57 | 0.62 | 0.75 |
| Eldorado | 235 | 1 | 0.98 | 0.99 | 0.98 |
| Xorist | 259 | 0.97 | 0.98 | 0.98 | 0.99 |
| Filecoder | 9 | 1 | 1 | 1 | 1 |

Table 5 shows the performance result for Histogram Boost Classifier. The result reveals several perfect score for recall, precision, F1 and accuracy. The Histogram Boost algorithm is able to achieve 7 out of 13 classes in perfect result and at least 4 near perfect score. The model is able to detect most of the ransomware to achieve 98%, this is due to the bad performance score for ransomware families like Gpcoder (57%). 43% of the testing sample has been classified as Xorist. It is also noted that there is a small number of Xorist ramsomware classified as Gpcoder but the number is insignificant.

TABLE VI. PERFORMANCE FOR GX BOOSTING CLASSIFIER

| Family | Num of Sample | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|---|
| Cerber | 107 | 1 | 1 | 1 | 1 |
| Locky | 22 | 1 | 1 | 1 | 1 |
| Bandarchor | 7 | 0.333 | 1 | 0.5 | 1 |
| GlobeImposter | 13 | 1 | 1 | 1 | 1 |
| Jaff | 15 | 1 | 1 | 1 | 1 |
| CryptoShield | 13 | 1 | 1 | 1 | 1 |
| TeslaCrypt | 286 | 1 | 0.99 | 0.99 | 0.994 |
| Spora | 14 | 0.67 | 1 | 0.8 | 1 |
| Wannacry | 10 | 1 | 1 | 1 | 1 |
| Gpcoder | 18 | 0.71 | 0.71 | 0.71 | 0.625 |
| Eldorado | 235 | 1 | 1 | 1 | 0.99 |
| Xorist | 259 | 0.98 | 0.96 | 0.97 | 0.96 |
| Filecoder | 9 | 1 | 1 | 1 | 1 |

Perfect score for recall, precision, F1 and accuracy achieved by the GX Boosting Classifier model can be seen in Table 4. There are at least 6 out 13 perfect in each performance measure. However, there is a ransomware family which is not performed to expectation. It is Gpcoder. The accuracy rate is 62%. But this, however, does not affect the overall performance as Gpcoder ransomware made up of 2% of the dataset.

TABLE VII. PERFORMANCE FOR CAT BOOSTING CLASSIFIER

| Family | Num of Sample | Precision | Recall | F1 | Accuracy |
|---|---|---|---|---|---|
| Cerber | 107 | 1 | 1 | 1 | 1 |
| Locky | 22 | 1 | 1 | 1 | 1 |
| Bandarchor | 7 | 1 | 1 | 1 | 1 |
| GlobeImposter | 13 | 1 | 1 | 1 | 1 |
| Jaff | 15 | 1 | 1 | 1 | 1 |
| CryptoShield | 13 | 1 | 1 | 1 | 1 |
| TeslaCrypt | 286 | 0.981 | 0.991 | 0.986 | 0.991 |
| Spora | 14 | 0.667 | 1 | 0.8 | 1 |
| Wannacry | 10 | 1 | 1 | 1 | 1 |
| Gpcoder | 18 | 0.667 | 0.571 | 0.615 | 0.571 |
| Eldorado | 235 | 1 | 0.981 | 0.990 | 0.981 |
| Xorist | 259 | 0.972 | 0.981 | 0.977 | 0.981 |
| Filecoder | 9 | 1 | 1 | 1 | 1 |

For Table 6 , the Cat Boosting Classifier model has achieved perfect accuracy score for 8 ransomware families and 2 ransomware families with near perfect score. The inability to score perfect overall result is due to the near perfect score for some families and also contribution by 1 imperfect ransomware family. These imperfect score is contributed by Gpcoder class ransomware.

Histogram Boosting Classifier, Cat Boosting Classifier and GX Boosting provide some consistency as shown in Tables 4, 5 and 6 . Specifically, they perform poorly for Gpcoder family. An initial observation is that the poor performance result may be incurred by the insufficient sample for the training process. It is clear from Table 4 , 5 and 6 that all poorly performed families are relatively small in number of samples. However, this assumption is quickly overruled after a closer examination. It is apparent that 61% of the ransomware families used in the experiment have under 20 samples and 92% of these families are able to achieve 80% and above in accuracy testing.

In order to reveal the rationale behind the poor performance, the performance results from other machine learning models are analysed. After examining the results, it is reasonable to conclude that the poor performance result is caused by inefficient labelling. There are inconsistencies while retrieving the result for the label from various Anti-Virus engines in VirusTotal. The decision for the label for each sample is made mostly based on the most frequent keyword appearing in the Anti-Virus engines. In the subsequent discussion, we try to investigate this phenomenon further in a quantitative manner. For this aim, the confusion matrix of the success rate of the Histogram Boosting Classifier model in recognising the samples from different families is depicted in Figure V.

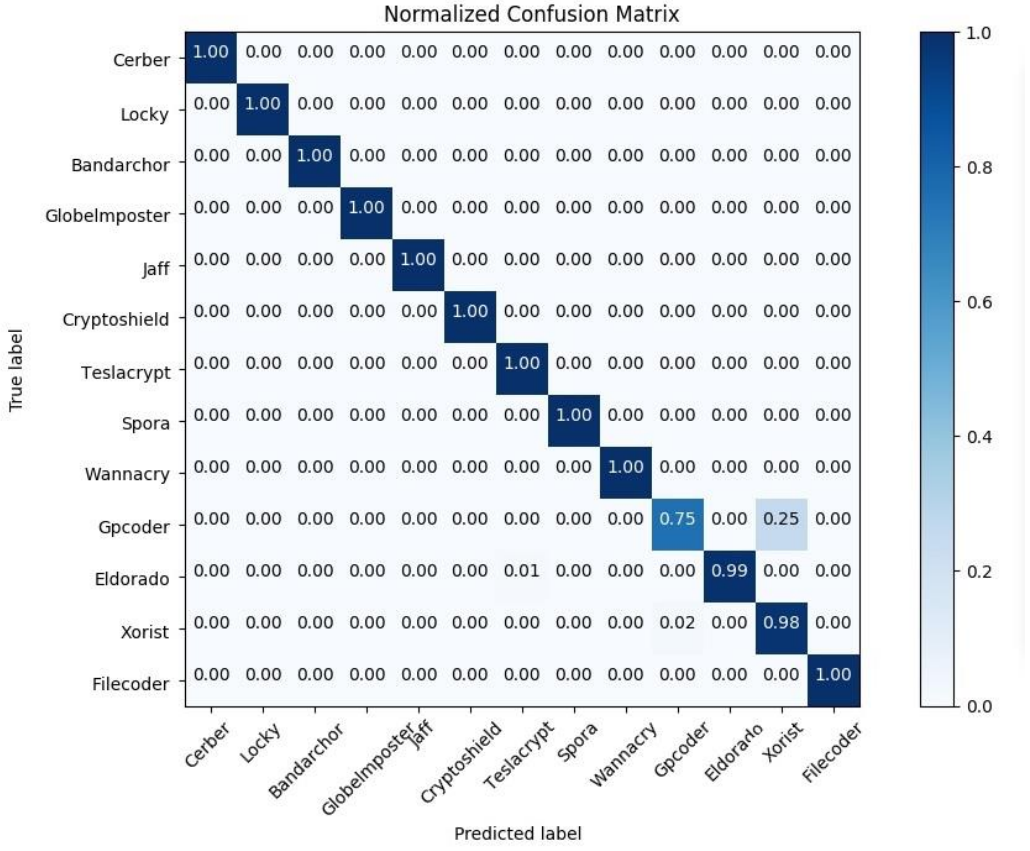


Fig. 5. The Confusion Matrix for Histogram Boosting Classifier

It displays the correct and incorrect prediction with respect to the actual ground truth. The confusion matrix is made up of m x m entries where m is the total number of ransomware families. The rows of the table represent the actual families and the columns represent the predicted families. The diagonal elements in the matrix show the number of correct predictions, they can also be projected as the true positive to the actual families. The off-diagonal shows the number of incorrect predictions which will be referred to as the false positive (top right side) and false negative (bottom left side).

As mentioned previously, the inaccuracy levels associated with Gpcoder is due to the inconsistencies among the Anti-Virus engines. As shown in Figure V , the classification of Gpcoder family has evenly spread across a few families such as Xorist. However, it is still manage to retain 50% of its own uniqueness. Note that however, designing an efficient labelling mechanism is important but it is out of the scope of this work. And quantitatively, the amount of samples from Gpcoder is small (i.e, 2%), it will not impact the classification performance significantly.

## 5. PROPOSED METHOD AND API CALLS BASED MODEL

TABLE VIII. COMPARISON OF PROPOSED METHOD WITH OTHER METHOD

| Reference | Year | Method | Feature | Type | Num of samples | Accuracy |
|---|---|---|---|---|---|---|
| Proposed Method | 2023 | Histogram Boosting | 19 | Classification(13) | 1000 | 98% |
| [24] | 2015 | SVM | 26 | Classification(12) | 115157 | 98% |
| [41] | 2016 | Similarity Threshold | NA | Binary | 13600 | 96% |
| [42] | 2017 | RNN | 239 | Binary | 15700 | 96% |
| [28] | 2018 | Random Forest | 28 | Binary | 100 MB | 87% |

The purpose of the proposed feature extraction method is to illustrate the theory of using less but vital feature can be effective in classification. This section will consist of 2 parts. In the first part, we will perform an observational comparison for the proposed method and other proposed models. Table 7 summarizes the number of feature, classification type, number of samples and accuracy rate of the proposed method and other existing works. Ransomware Detection and classification is not a new subject and the proposed method has scored very promising accuracy result. The number of classes and dataset do affect the accuracy score. 28 scores an accuracy rate of 87% for detection, this is the lowest in the table. One of the reason is that the size of the dataset and the feature set are relatively small and it is insufficient to achieve an above 90% accuracy result. Works in [41] and [42] are able to produce high accuracy result since they use binary classification. It is common to have high accuracy result for such classification if sufficient dataset is provided. This is evidentially proven in 41, there are 13600 samples for 2 classes. It can be observed from the table that work in [43] scores 98% for the classification test by using massive dataset. It is clear that there are several factors that needs to be fulfilled in order to achieve good performance result. They are:

- Small classification size,
- Large feature size and
- Large dataset.

On the contrary, the proposed method does not follow any of the above recommendations. Table VII shows the proposed method has the smallest feature set, among the smallest dataset and has the largest classification size. And yet it is able to achieve an accuracy of 96%. The feature extraction technique has also contributed to and directly impacted the result.

In the second part, we will evaluate API calls-based model with the proposed model. In this evaluation, the number of features and the performance based on accuracy will be compared against the proposed model. The ransomware samples used in the proposed model are converted into the API calls dataset for the API-based model.

As mentioned, there are 2 types of feature extraction from dynamic analysis. They are feature based on API calls and feature based on file activities. API calls-based feature extraction has been used widely and is being adopted in many proposed models.

The aim of the proposed dynamic analysis is as following:

- use few features and maintain classification accuracy,
- take advantage of feature extracted from the high abstraction level and
- use a sequential extraction for the encryption process.

I compare the proposed feature and the API calls-based dynamic analysis proposed by [18]. First, we compare both models conceptually. The difference in the feature is discussed. The experiment is also conducted and the result will be revealed in detail later in this section.

During the experiment conducted, there were 127 features in the feature set after extracting the API calls from all the different families. For simplicity purposes, we will use the API calls from the three more popular ransomware families. Table VIII lists the example of a dataset for each of the 3 different ransomware families using API Calls. There are 33 unique API calls extracted in total.

TABLE IX. API CALLS EXTRACTED FROM CERBER, LOCKY AND TESLACRYPT18

| | | |
|---|---|---|
| LdrLoadDll | LdrGetProcedureAddress | OpenSCManagerW |
| OpenServiceW | NtDuplicateObject | NtCreateThreadEx |
| NtResumeThread | SendNotifyMessageA | WNetGetProviderNameW |
| NtClose | CreateProcessInternalW | NtAllocateVirtualMemory |
| NtFreeVirtualMemory | LdrGetDllHandle | GetFileType |
| DeleteFileW | GetSystemDirectoryW | GetSystemWindowsDirectoryW |
| RegOpenKeyExA | RegOpenKeyExW | LoadStringW |
| RegQueryValueExW | RegCloseKey | NtOpenKey |
| NtQueryValueKey | NtCreateMutant | GetNativeSystemInfo |
| NtQueryAttributesFile | NtOpenSection | NtMapViewOfSection |
| NtCreateSection | OleInitialize | 0NtCreateFile |

According to [18], if all these API calls are required to form a feature set for each sample in vector, the vector will contain 33 rows for the 3 ransomware families. This is illustrated in Table 9. Each API call is considered as a feature. The amount of API call will increase when the number of ransomware families increases. It is vital to ensure that all extracted API calls are included into the feature set. Hence, the feature set for the API calls-based dynamic analysis will always have more than a hundred features in its dataset. Another issue is the level of difficulty to include new ransomware family into the dataset. The new ransomware family may add new API calls to the existing list which will require the dataset to be recreated again.

TABLE X. API CALLS FOR CERBER, LOCKY AND TESLACRYPT18

| Sample | Feature Representation |
|---|---|
| TeslaCrypt | {1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} |
| Cerber | {0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} |
| Locky | {1,1,0,0,0,0,0,0,0,1,0,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} |

The proposed feature does not have such an issue as the design of the feature extraction method is based on the behaviour of the ransomware. The feature set is tailored for most ransomware families and will not change if an unknown ransomware or a new ransomware family is required to be added to the dataset. The encryption process activity is represented using 6 feature spaces and this can be seen in Table 2. The proposed features have provided enough room for new ransomware family if its encryption process requires more dimensional space.

TABLE XI. THE ACCURACY PERFORMANCE FOR API CALLS DYNAMIC ANALYSIS AND THE PROPOSED MODEL18

| | Dynamic process[18] using API Calls | Proposed Model |
|---|---|---|
| Random Forest | 0.8529 | 0.9604 |
| Gradient Boosting | 0.8529 | 0.9273 |
| KNN | 0.8824 | 0.9480 |
| Decision Tree | 0.7353 | 0.9346 |

Table 11shows the accuracy comparison for the proposed model and the dynamic analysis using API calls. For the API call-based dynamic analysis, the experimental ransomware samples are analysed using the cuckoo sandbox to extract the API calls. The API calls are factorised and translated using various popular machine-learning models. The result shows the accuracy performances for the machine learning models like Random Forest Decision Tree and KNN. In Table 10, it shows that the API call-based model only achieves 88% whereas the proposed model is able to derive 96%.

TABLE XII. THE DURATION TAKEN FOR API CALLS BASED DYNAMIC ANALYSIS AND THE PROPOSED MODEL TO CONVERT FEATURE[18]

| | Dynamic process[18] using API Calls | Proposed Model |
|---|---|---|
| Random Forest | 0.8529 | 0.9604 |
| Gradient Boosting | 0.8529 | 0.9273 |
| KNN | 0.8824 | 0.9480 |
| Decision Tree | 0.7353 | 0.9346 |

In the next experiment, we record the amount of time taken for the classification process. Table 12 shows the duration for each process. The proposed method and API-based dynamic analysis have different processes. Despite the difference, the tasks in some processes such as running the sandbox, extracting information from the sandbox and converting features are similar. In this experiment, the sandbox for both models is executed for 3 minutes. It is clearly shown in the table that the API calls-based dynamic analysis takes more process than the proposed model to create the feature. But it takes a shorter duration to create the dataset < 1s. This is due to the simplicity of the extraction method. The difference, however, is not significant and does not impact the performance of the proposed model.

## 6. DISCUSSION

### 6.1 Limitation

While the proposed feature has yielded good results, there are several limitations encountered during the experiment's implementation. First and foremost, the proposed feature is limited to executable files and has not been tested on other file formats. This limitation is partly due to the choice of sandboxes used in the experiment. There are more advanced sandboxes, such as Cuckoo44, which offer greater functionality and the ability to analyze different file formats. Despite the recommendations for such advanced sandboxes, we opted for the less popular Noriben sandbox. We made this decision because Noriben offered advantages such as ease of setup and customization for the specific purposes of this experiment. Its simplicity aligned with the objectives of our experiment, leading us to overlook the sandbox's limitations.

Additionally, some of the created features depend on the content within the virtual machine. The placement of bait files may influence measurements for features like maximum encryption length and encryption mean value. The content's role is crucial, especially when considering the behaviour of ransomware. To capture high-level behavioural activities, bait files in various formats are necessary. The goal is to simulate the real working environment as closely as possible with "real" files. To minimize inconsistencies, we can extract more encryption sequence samples for feature formulation.

The sandbox used in our experiment does not employ any counter-antivirtualization techniques. Samples with antivirtualization measures remain inactive in the sandbox environment, making it impossible for the feature extractor to gather information for the log file. In this paper, the implementation of counter-antivirtualization techniques is not emphasized and falls outside the scope of our focus. We have primarily concentrated on the functioning samples to align with the objectives outlined in this paper.

### 6.2 Partially Executed Sample

Partially executed samples occur when monitoring stops while the ransomware is still active. In most cases, this situation is undesirable. The reason is that the full potential or sometimes the true objective of the malware remains undisclosed due to insufficient time allocated to the sample. During data collection, some ransomware takes longer to compromise the honeypot than others. However, this is not a significant issue and has no impact on data collection. Three minutes are typically sufficient for most ransomware to complete their objectives and partially executed ransomware samples still provide adequate information about the encryption process, ransom text file, and changes in the registry. This information is available in the log as long as the ransomware is active.

## 7. CONCLUSION

Malware behaviour can be used to distinguish between different malware families. This paper described the key behaviours of ransomware: searching, encrypting, and displaying. It also discussed some dynamic analysis frameworks and their

limitations. Feature extraction is a challenging task, as it is an ever-evolving concept with new approaches constantly aiming to outperform their predecessors. Our proposed method is simple, relevant, and unique. It utilizes features that best describe ransomware behaviour to improve classification performance. These features can be extracted from runtime behavioural analysis. One of the most important components used in building the feature set is the encryption function. Sequential analysis is performed to determine and extract these features. The results show over 98% accuracy during training and testing. Many of the behavioural tendencies used in this research can be applied as representative features for defining ransomware behaviour. In the future, we plan to extend this research to other malware families and experiment with polynomial feature sets. This will help us determine the extent to which our approach can be applied to understand the behaviours of other malware classes and families. Additionally, we encourage other researchers across different platforms to test our proposed methodology to either confirm our findings or challenge them. This will help us identify the strengths and weaknesses of our feature extraction approach and inform our efforts to improve it.

## Financial Support

This research was independently conducted without any financial assistance from institutions or sponsors.

## Conflict of Interest

The authors affirm the absence of any conflicting interests associated with this study.

## Acknowledgment

The authors express sincere gratitude to the institution for its unwavering moral support and encouragement during the research process.

## References

[1] O. A. Aminat, M. O. Adaobi, C. A. Chukwuka, et al., "Design and Implementation of a Malware Detection System On Smartphones," International Journal of Information, Business and Management, vol. 14, no. 1, pp. 171–181, 2022.
[2] M. Saqib, P. Fournier-Viger, M. Zohaib, G. Chen, and Y. Wu, "MalSPM: Metamorphic malware behavior analysis and classification using sequential pattern mining," Computers & Security, vol. 118, p. 102741, 2022.
[3] N. Milošević, "History of malware," arXiv preprint arXiv:1302.5392, 2013.
[4] Ö. Aslan, M. Ozkan-Okay, and D. Gupta, "Intelligent behavior-based malware detection system on cloud computing environment," IEEE Access, vol. 9, pp. 83252–83271, 2021.
[5] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," Computers & Electrical Engineering, vol. 66, pp. 353–368, 2018.
[6] S. Mohurle and M. Patil, "A brief study of wannacry threat: Ransomware attack 2017," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, 2017.
[7] S. Mujeye, "Ransomware: To Pay or Not to Pay? The results of what IT professionals recommend," in Proceedings, pp. 76–81, 2022.
[8] C. Moore, "Detecting ransomware with honeypot techniques," in Proceedings, pp. 77–81, IEEE, 2016.
[9] S. P. Choudhary and M. D. Vidyarthi, "A simple method for detection of metamorphic malware using dynamic analysis and text mining," Procedia Computer Science, vol. 54, pp. 265–270, 2015.
[10] G. Kakisim, S. Gulmez, and I. Sogukpinar, "Sequential opcode embedding-based malware detection method," Computers & Electrical Engineering, vol. 98, p. 107703, 2022.
[11] A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: a survey and research directions," Computers & Security, vol. 74, pp. 144–166, 2018.
[12] R. Kaur and M. Singh, "A Survey on Zero-Day Polymorphic Worm Detection Techniques," IEEE communications surveys and tutorials, vol. 16, no. 3, pp. 1520–1549, 2014.
[13] U. Urooj, B. A. S. Al-rimy, A. Zainal, F. A. Ghaleb, and M. A. Rassam, "Ransomware detection using dynamic analysis and machine learning: A survey and research directions," Applied Sciences, vol. 12, no. 1, p. 172, 2021.
[14] Y. Ding, X. Xia, S. Chen, and Y. Li, "A malware detection method based on family behavior graph," Computers & Security, vol. 73, pp. 73–86, 2018.
[15] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," Wireless Personal Communications, vol. 112, pp. 2597–2609, 2020.
[16] M. Jain and P. Bajaj, "Techniques in detection and analyzing malware executables: A review," International Journal of Computer Science and Mobile Computing, vol. 3, no. 5, pp. 930–935, 2014.
[17] C. Kruegel, "Full system emulation: Achieving successful automated dynamic analysis of evasive malware," in Proceedings, pp. 1–7, 2014.

[18] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," in Proceedings, pp. 23–30, IEEE, 2010.
[19] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," International Journal of Distributed Sensor Networks, vol. 11, no. 6, p. 659101, 2015.
[20] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," Journal of computer security, vol. 6, no. 3, pp. 151–180, 1998.
[21] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," Journal in computer virology, vol. 4, no. 4, pp. 323–334, 2008.
[22] M. A. Ammar, "LOB and CPM integrated method for scheduling repetitive projects," Journal of construction engineering and management, vol. 139, no. 1, pp. 44–50, 2012.
[23] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in Proceedings, pp. 178–197, Springer, 2007.
[24] A. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," Computers & Security, vol. 52, pp. 251–266, 2015.
[25] A. Fattori, A. Lanzi, D. Balzarotti, and E. Kirda, "Hypervisor-based malware protection with accessminer," Computers & Security, vol. 52, pp. 33–50, 2015.
[26] A. Pektaş and T. Acarman, "Malware classification based on API calls and behaviour analysis," IET Information Security, vol. 12, no. 2, pp. 107–117, 2017.
[27] J. Wainer, "Comparison of 14 different families of classification algorithms on 115 binary datasets," arXiv preprint arXiv:1606.00930, 2016.
[28] G. Cusack, O. Michel, and E. Keller, "Machine Learning-Based Detection of Ransomware Using SDN," in Proceedings, pp. 1–6, 2018.
[29] Malware-Traffic-Analysis.net, "Malware-Traffic-Analysis.net," 2016.
[30] VirusTotal, "VirusTotal - Free Online Virus, Malware and URL Scanner," 2018.
[31] Virusshare.com, "Virusshare.com," 2016.
[32] M.-L. Zhang and Z.-H. Zhou, "A Review on Multi-Label Learning Algorithms," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 8, pp. 1819–1837, 2014.
[33] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," Advances in neural information processing systems, vol. 31, 2018.
[34] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," Pattern recognition, vol. 40, no. 7, pp. 2038–2048, 2007.
[35] R. Kohavi, "Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid," in Proceedings, pp. 202–207, 1996.
[36] A. Guryanov, "Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees," in Proceedings of the X Conference, Springer, 2019, pp. 39–50.
[37] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," Machine Learning, vol. 63, no. 1, pp. 3–42, 2006.
[38] A. Liaw and M. Wiener, "Classification and regression by randomForest," R News, vol. 2, no. 3, pp. 18–22, 2002.
[39] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in Proceedings of the X Conference, 2016, pp. 785–794.
[40] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Annals of Statistics, vol. 29, no. 5, pp. 1189–1232, 2001.
[41] A. Kharraz, S. Arshad, C. Mulliner, W. K. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in Proceedings of the X Conference, 2016, pp. 757–772.
[42] S. Maniath, A. Aravind, P. Prabaharan, V. G. Sujadevi, A. U. P. Sankar, and S. Jan, "Deep learning LSTM based ransomware detection," in Proceedings of the Y Conference, 2017.
[43] A. K. Jain and B. B. Gupta, "A machine learning based approach for phishing detection using hyperlinks information," Journal of Ambient Intelligence and Humanized Computing, pp. 1–14, 2018.
[44] D. Octaviano and M. Iqbal, Cuckoo Malware Analysis. Packt Publishing Ltd, 2013.