LightGBM-based Ransomware Detection using API Call Sequences

Duc Thang Nguyen, Soojin Lee*
Department of Computer Science and Engineering
Korea National Defense University, Nonsan, Republic of Korea

***Abstract*—Along with the development of technology as well as the explosion in digital data in the era of fourth industrial revolution, cyberattacks using ransomware are emerging as a serious threat to many agencies and organizations. The harm of ransomware is not limited to the areas of information technology and finance but also affects areas related to people's lives, such as the medical field. Therefore, research to identify and detect these types of malicious code is urgent. this paper present a novel approach of identifying and classifying ransomware based on dynamic analysis techniques combined with the use of machine learning algorithms. First, this research focused on the Application programming interface (API) call functions that are extracted during a dynamic analysis of executable samples using the Cuckoo sandbox. Second, research used LightGBM, a gradient boosting decision tree algorithm, for training and then detecting and classifying normal software and eight different types of ransomware. Experimental results showed that the proposed approach achieves an overall accuracy rate of 98.7% when performing multiclass classification. In particular, the detection rates of ransomware and normalware were both 99.9%. At the same time, the accuracy in identifying two specific types of ransomware, WannaCry and Win32:FileCoder, reached 100%.**

Keywords—Ransomware; machine learning; API call; dynamic analysis technique; gradient boosting decision tree; GBDT; lightGBM

I. Introduction

Ransomware is a form of malicious code which, upon infecting a victim's device, encrypts and then steals the victim’s data and prevents legitimate access to it until the victim pays a ransom. In early versions of ransomware, the code mainly performed the trick of locking the victim's system, aimed at non-computer savvy users. However, today's ransomware mainly uses crypto-viral extortion techniques. These methods take advantage of the most modern encryption techniques to encrypt almost all of the victim's personal data (e.g., photos, documents, texts). Even the most knowledgeable users or experts also face considerable difficulties. It is almost impossible to recover the data until receiving the decryption key. In a properly executed crypto-viral ransomware attack, recovering data without a decryption key is a problem that is difficult to solve. The attack also requires digital currencies that are difficult to track, such as Bitcoin, for ransom, making it even more difficult to investigate and track down the culprit.

The use of ransomware is accelerated and becoming increasingly dangerous compared to levels seen in the past. Ransomware is now a national security issue for all countries around the world, and it will only become worse. In particular, during the recent Covid-19 pandemic, attacks on many hospitals and medical facilities indicate a new risk of ransomware, considering that its influence caused the death of a patient (Fireeye’s 2021 report). Ransomware at present is real threat to humans' lives. Threat actors will increasingly target the most critical assets, such as sensitive data and architectures, held by organizations, leading to much higher ransom amounts. Ransoms have already reached the tens of millions of dollars and are expect to grow. While many organizations pay ransoms and do regain access to their data. And they often forget that the attackers still have their data and can allow anyone to buy the data right from their websites (SophosLab’s Threat Report 2021). Data theft creates a secondary extortion market.

The continued success of ransomware poses a serious cyber security threat. According to the statistics of reputable security firms, ransomware can spread maliciously in many types of ways, via sophisticated techniques used to avoid detection by antivirus software. Therefore, the need to analyze these types of malware is urgent given the explosion of data in the fourth industrial revolution with millions of Internet of Things devices connected to the Internet every day.

Attackers are increasingly turning to ransomware as a service (RaaS) with more customization capabilities that rapidly increase the number of ransomware variants and types. Therefore, traditional signature-based detection techniques are not effective. Current techniques often build on complex models that combine many features extracted through static analysis or dynamic analysis, along with various transformations to distinguish between ransomware and normal software. They are based on certain features extracted from a dynamic analysis or static analysis, such as API sequences, opcode strings of files, file entropy levels, and/or change in system files. The use of the API function call sequence to detect and classify ransomware types had been applied in many practical studies, such as in [1], [2], [3], [4], showing promising results. However, the fundamental problem of detection methods based on static analysis is weak detection when attackers use code obfuscation methods or zero-day attacks. Furthermore, the malicious code classification methods based on API functions extracted from static analysis lead to one drawback. These methods are easily evaded when an attacker inserts normal API calls or declares unused API functions during a ransomware execution.

Among various features, this research focuses on the Windows API call frequency, which is extracted via a dynamic analysis technique. Proposal method use it as the primary

*Corresponding Author

factor in conjunction with certain transformations that improve the speed and accuracy in identifying instances of malicious code extortion. Proposal method also applies LightGBM, a gradient boosting decision tree (GBDT) algorithm, to increase the accuracy and speed of detecting and classifying ransomware.

The next sections of the paper are organized as follows: Section 2 presents ransomware detection techniques that are currently being studied. Section 3 describes the proposed approach and algorithms used, and Section 4 explains the collected dataset and the experimental method. Section 5 analyzes the experimental results and Section 6 concludes the paper.

II. Related Research

There are two main approaches when analyzing and detecting ransomware: static analysis and dynamic analysis [5]. While the static analysis technique mainly focuses on analyzing and checking the file structure and executable file formats without running the file, a dynamic analysis allows malware to run to observe its behavior in the system ultimately to eliminate the infection.

This Section reviews several ransomware analysis methods which are on the basis of the above two techniques in terms of the extracted information. Then, current multiclass classification methods will be reviewed.

In terms of API sequences, to achieve malicious purposes especially when implementing ransomware, attackers must use and execute a specific API sequence. So there are big differences between malicious codes and normal software in API call sequences. There have been several studies focusing on an analysis of API call sequences to detect general malware as well as ransomware. For example, in [1] Sgandurra et al. presented a ransomware detection method based on dynamic analysis and applied a type of machine learning known as EldeRan. They collected several features extracted from their dynamical analysis, such as API calls, registry and file system change logs, dropped files, and crypto-function patterns in binary files. EldeRan studied a dataset with 1524 samples consisting of 582 ransomware and 942 benign samples. By using a regularized logistic regression method for classification, they could achieve 96.3% detection rate in binary classification.

In [3] Hwang et al. combined a Markov model with random forest model to build two-stage mixed ransomware detection model. The Markov model is used to capture the characteristics of ransomware with the Windows API call sequence pattern that obtained by a dynamic analysis. During the second stage, the random forest machine learning model's mission is to control misclassified samples in the remaining data. The accuracy of this two-stage mixed detection method is 97.3%. In binary classification, False positive (FP) and False negative (FN) rate are relatively high, 4.8% and 1.5% respectively. In [6], Bae et al. used the Intel PIN tool to extract Windows API call sequences and then generated n-gram sets from these API sequences. These n-gram sets were used to classify ransomware, malware and benign files. The authors concluded that their method could detect ransomware with a detection accuracy up to 98.65%.

Several file-based techniques can identify the presence or existence of ransomware based on the transformation in files of system or in files of a particular format. In [7], after studying a dataset including 1359 ransomware samples from 2006 to 2014, Kharraz et al. concluded that it is not complicated to design an advanced technique to block several types of ransomware by monitoring file system anomaly activities. This method can be effective even against those using sophisticated cryptographic malware or some types of zero-day ransomware attacks. In [8], Lee et al. measured the entropy of six different file formats and then used machine learning to detect infected files to protect the original file in a backup system while synchronizing the time. By identifying files infected with ransomware, this method allows the recovery of those files from system storage when the user's system is infected. Khammas [4] proposed a method that detects ransomware based on a static analysis. The method used frequent pattern mining and the gain ratio technique to extract 1000 features directly from raw binary files. A random forest technique is applied to the classification process. The dataset consists of 1680 executable files made up of 840 ransomware and 840 normal files. The accuracy rate was 97.7%.

In network-based studies, Cabaj et al. [9] proposed a solution to identify ransomware based on HTTP traffic communication when the ransomware connects to the attacker's C&C server. The experimental results obtained detection rates of 97–98%. However, the authors only monitored and observed the network traffic communication of two types of ransomware, CryptoWall and Locky. The author in [10] presented an advanced ransomware identification method based on an analysis of network traffic activities. The study observed TCP, HTTP, DNS, and NBNS traffic and extracted 18 different features. They prototyped a multi-classifier network-based ransomware detection method that combines of two different levels: the packet level and the flow level. The highest detection accuracy rates for the two corresponding levels were 97.92% and 97.08%. However, this research only focused and analyzed on the Locky ransomware's network activities and is thus not suitable for other types of ransomware.

Other researchers also used a hybrid method that integrates dynamic and static analysis techniques to distinguish between ransomware and normalware. For instance, Shaukat et al. [11] presented a method called RansomwareWall. The set of features collected by static analysis and dynamic analysis is fed to the machine learning engine for binary classification of samples as ransomware or benign. Using a dataset of 574 samples from 12 ransomware families, the experimental result presented detection rates ranging from 85.7% (using logistic regression) to 98.25% (using a gradient tree boosting algorithm).

For multiclass classification, some researchers are not only working to distinguish between ransomware files and normal files but also looking for ways to distinguish between different types of ransomware. For example, Zhang et al. proposed an approach for multiple classifications of seven ransomware families based on a static analysis [12]. They built N-gram sequences by extracting opcode sequences from Portable executable (PE) file samples and then calculated the term frequency - inverse document frequency (TF-IDF) to identify the feature N-gram vectors. The feature vectors were then subjected to five machine-learning methods to classify ransomware. The dataset included 1,787 ransomware samples of seven ransomware families crawled from VirusTotal that broke out from 2012 to 2017. In the experiments, the best accuracy achieved was 91.43% for the multiclass classification method when using a random forest algorithm and 99.3% for the binary classification of ransomware and 'goodware'.

In [13], Baldwin et al. presented a WEKA toolset for ransomware multiclass classification based on a static analysis. They extracted 443 opcodes from binary files and used them to calculate the percentage of each opcode occurrence relative to the overall opcode. The support vector machine (SMV) learning technique was used for binary classification between benign and ransomware, while the PUK kernel was used for multiclass classification. The best accuracy gained from the results was approximately 96.5% when differentiating a dataset consisting of 443 samples of six classes (one benign and five ransomware families). Vinayakumar et al. studied a dataset of 974 samples (219 benign files and 755 ransomware files from seven ransomware families) and focused on the API call sequence for ransomware detection [2]. Their method gathered 131 API sequences with a dynamic analysis technique and used a multi-layer perceptron (MLP) model for classification. The experimental results showed that the best accuracy rate was 98% for multiclass classification; however, the true positive rates (TPR) of crypto-locker and cryptowall ransomware were only 88.9% and 83.3%.

Current studies mainly focus on binary classification between ransomware and normalware. However, with the rapid growth of blackmail attacks as well as the variety of types of ransomware, the detection and detailed classification of each ransomware family type are necessary at present. There have been a few studies related to multiclass classification, but those studies mainly focused on classifying categories together. Moreover, the accuracy when identifying each type of ransomware is not very high, leading to the ineffective prevention of malicious code, placing user data in danger. In order to overcome the drawbacks of previous multiclass classification techniques, this paper present a novel approach based on a dynamic analysis and the LightGBM algorithm to detect multiple types of ransomware and to distinguish between ransomware and benign files.

III. Proposed Method and Algorithm

A. LightGBM Algorithm

Decision trees "learn" by breaking down observations based on feature values. In the decision tree learning process, finding the best split is the most time-consuming stage. Two algorithms which use different gradient boosting decision tree (GBDT) implementations to find the best splits are as follows:

- Pre-sort: Object values are pre-sorted and all split points can be evaluated.
- Histogram-based: Continuous features are divided into separate bins used to create histograms for features.

Histogram-based algorithms are more efficient in terms of memory consumption and training speeds. However, for every feature, all data instances must be scanned to find all possible split points. So that both pre-sorted and histogram-based methods become slower as the number of instances or features increases. The LightGBM algorithm aims to address the training speed and memory consumption issues associated with typical implementations of GBDT when working with large datasets.

First, LightGBM grows the tree in a leaf-wise manner using a vertical growth strategy. This is different from the horizontal growth strategy (level-wise growth) tactics of the other decision tree algorithms. When growing leaf-wise, the gradient-based method can help the errors minimize and effectively reduce the loss. The balance of the tree is maintained via a level-wise growth strategy, whereas the leaf-wise strategy helps to reduce the loss the most. With the same number of leaves, a leaf-wise-based tree will be deeper than other trees. In particular, when necessary leaf-wise growth can be used to grow a tree into a more balanced tree. Compared to horizontal growth, vertical planting can provide converge much more rapidly [14].

Secondly, LightGBM developed two techniques to reduce memory consumption and speed up the training time [15]. These are gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB). With GOSS, LightGBM reduces the number of instances by keeping all large instance gradients and random sampling instances with small gradient instances. The complexity of constructing the histogram for all features is O(#data *#features) and the complexity of subsequently finding the optimal split points is proportional to O(#bins * #features). Generally, #bins << #data. Therefore, this approach is computationally much more efficient than earlier approaches.

Algorithm 1. Gradient-based One-Side Sampling (GOSS) Technique

```
Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: loss: loss function, L: weak learner
models ← {}, fact ← (1-b)/a
topN ← a × len(I), randN ← b × len(I)
for i = 1 to d do
    preds ← models.predict(I)
    g ← loss(I, preds), w ← {1,1,...}
    sorted ← GetSortedIndices(abs(g))
    topSet ← sorted[1:topN]
    randSet ← RandomPick(sorted[topN:len(I)],randN)
    usedSet ← topSet + randSet
    w[randSet] × = fact : Assign weight fact to the small gradient
      data.
    newModel ← L(I[usedSet], - g[usedSet],w[usedSet])
    models.append(newModel)
```

Step 1: Based on the list sorted according to the data instance gradient values, GOSS selects the top a× 100% largest gradient instances.

Step 2: Perform random sampling b× 100% on the remaining instances with small gradients.

Step 3: Recalculate the information gained by amplifying the sampled data of small gradients with (1-a)/b.

In this way, LightGBM can focus more on larger gradient (under-trained) instances without altering the original data distribution much.

EFB is a technique that uses a greedy algorithm to combine (or bundle) these mutually exclusive features into a single object (bundle of exclusive objects) and thus reduce the size. The complexity of feature histogram building is now proportional to the number of bundles O(#data * #bundle) rather than the number of features O(#data * # feature). With EFB, LightGBM can reduce the GBDT training time without having a great impact on the accuracy.

Algorithm 2. Exclusive Feature Bundling (EFB) Technique

```
Input: numData: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f in F do
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin ← new Bin(numData)
for i = 1 to numData do
    newBin[i] ← 0
    for j = 1 to len(F) do
    if F[j].bin[i] ≠ 0 then
    newBin[i] ← F[j].bin[i] + binRanges[j]
Output: newBin, binRanges
```

By experimenting on several public datasets, the results demonstrated that using the LightGBM algorithm increased the training speed by more than 20 times while maintaining the same level of accuracy.

In conclusion, LightGBM offers many advantages when used to address current practical issues:

- Higher efficiency as well as faster training speeds.
- Lower memory consumption.
- Better accuracy.
- Can handle large-scale data well.
- Supports GPU and parallel learning.

B. *Proposed Ransomware Detection Method*

This approach fully utilizes the advantages of LightGBM algorithm described above and presented in some previous studies [16] and [17]. Because the API functions that used in the each sample (ransomware and benign) are very different, so that the dataset based on this features is spare. To fill the gap in current ransomware multiclass classification and to overcome the disadvantages of previous methods, this research present an approach based on a dynamic analysis and apply the LightGBM algorithm to process highly sparse data. By only using the API call sequence as the primary factor, this approach is more simple than others.

To recognize a portable executable (PE) file as good software or ransomware and further to categorize ransomware into their respective categories, we utilize a machine learning architecture, as shown in Fig. 1.

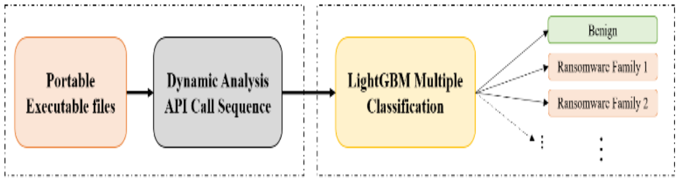

Portable Executable files
Dynamic Analysis API Call Sequence
LightGBM Multiple Classification
Benign
Ransomware Family 1
Ransomware Family 2

Fig. 1. Ransomware Multiple-Classification Proposed Method.

In the first step, "tagged" PE files are analyzed by means of a dynamic analysis. After which information extraction of the API call sequence functions of each individual file were performed and consider them as key features in this proposal. For the second step, samples with corresponding features and assigned labels are passed as input to the LightGBM multiclass classifier to generate learning trees that help to distinguish between good software and ransomware.

IV. Dataset and Experimental Method

A. *Database Gathering and Analysis*

Currently, no complete dataset of ransomware on Windows platforms has been made public in cyberspace. Many authors have collected ransomware samples from multiple sources and built datasets for their own research. On the other hand, samples of ransomware often exist only sporadically in some test datasets. These dataset all malware marked ransomware, regardless of ransomware types. Previous researchers have also actively grouped ransomware types, but in experimental studies, they still mainly stop at distinguishing ransomware from normalware. Currently, there are very few researchers delving into the simultaneous identification and discernment of benign software and ransomware of various categories.

Towards the above goal, this research attempt to build a ransomware dataset for research that not only helps to distinguish between ransomware and benign software but also improves the accuracy when classifying each type of ransomware. The dataset was constructed based on a number of scientific guidelines and best practices suggested by Rossow [18]. Ransomware samples were collected from two of the most popular data-sharing sources, VirusTotal[1] and Virusshare[2], under academic license and with the administrator's consent. Research focused on collecting both recent and earlier ransomware samples (from 2014 to early 2021) and worked to gather as much of each type of ransomware as possible. Because malicious samples were collected from two different sources, this research used a distinctive SHA hash to avoid duplications in the sample dataset. To be cautious when choosing the ransomware, we confirmed that an instance of

[1] https://www.virustotal.com/
[2] https://virusshare.com/

malware is ransomware if at least five antivirus engines marked it belonging to this category. Research rely on the naming policy of the Avast engine to determine the family of each sample before starting the process of analyzing and collecting data about the behavior of ransomware.

For benign samples, the executable files in the Windows system directory (…\Windows\System32) with a variety of functions and features of the files were selected. This could help to assess and classify malware in a more objective manner.

While Table I shows information about the benign files and ransomware samples that were taken from the two main sources, Table II details the types of ransomware along with the number of samples collected in each case.

After a dynamic analysis of 5,811 samples, we can realize that the number of API functions used by a sample ranges from 01 to 172 with approximately 286 different API functions. Ransomware and benign files both use the same API functions, but for separate purposes. Moreover, the number of API function calls in each executable file differs significantly. There are functions that are only called and used one to two times in one PE file but are used many times in other executables. During the dynamic analysis, it was noted that there are API functions invoked and used by a file during its execution up to hundreds of thousands of times. At the same time, the number of APIs used by the each sample also differs across ransomware and benign files. While 'goodware' files mostly use 10-20 different API functions, the number of API functions used by ransomwares typically exceed 100. Therefore, the dataset is extremely sparse. This is highly suitable when applying the LightGBM algorithm given its many advantages when experimenting on this sparse dataset.

B. Experiment

The experimental process is depicted in Fig. 2 and is divided into five main steps, as follows:

Step 1: Dynamic Analysis

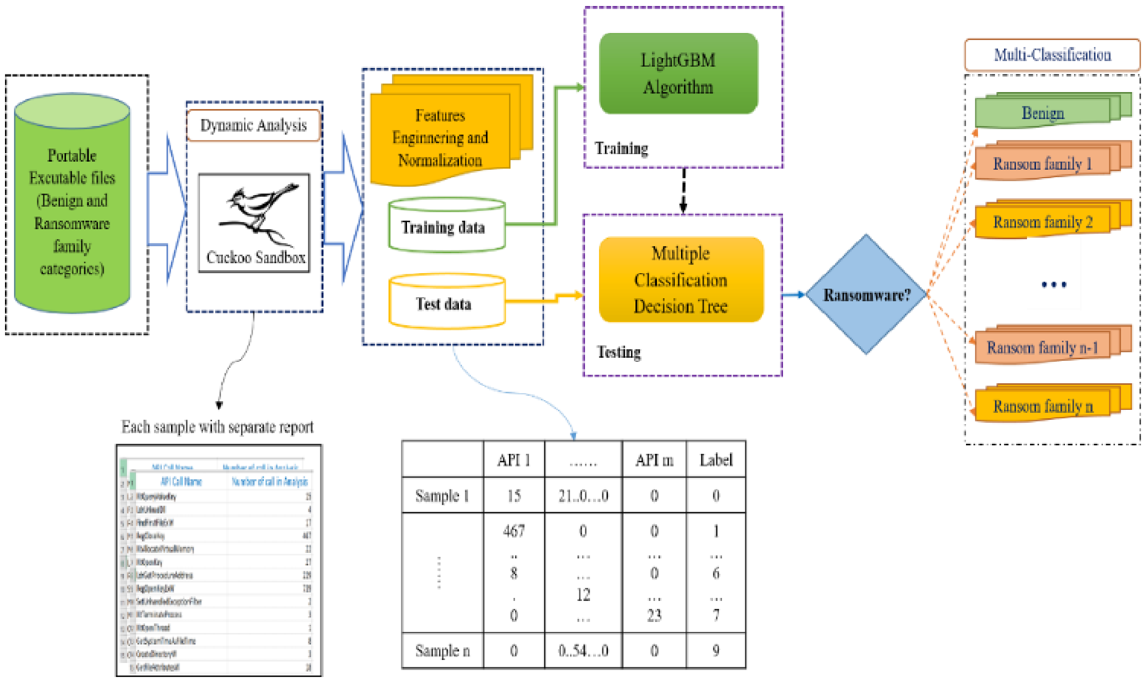
After being collected, the executable files were divided into categories, in this case benign files and ransomware files of different types (eight types of malicious codes).

TABLE I. THE NUMBER OF FILE COLLECTED FROM EACH SOURCE

Sample	**Source**	**Number of sample**
Benign	Windows system files	4,008
Ransomware	Virusshare.com	1,373
	Virustotal.com	430
Total		5,811

TABLE II. RANSOMWARE FAMILIES

No.	**Ransomware Family**	**Number of sample**
1	Reveton	522
2	TeslaCrypt	167
3	Win32:Ransom	204
4	Win32:Cryptor	123
5	Win32:Crypt	146
6	LockScreen	123
7	WannaCry	491
8	Win32:FileCoder	27
Total (Ransomware)		1,803



	API 1	……	API m	Label
Sample 1	15	21..0…0	0	0
⋮	467 .. 8 . 0	0 … … 12 …	0 … 0 … 23	1 … 6 … 7
Sample n	0	0..54…0	0	9

Fig. 2. Experimental Process.

The PE files are then put into the Cuckoo sandbox environment for a dynamic analysis. This is necessary and ensures safety and convenience during the dynamic analysis. The ransomware is executed in a simulated environment, during which all behaviors of every sample are collected in sandbox logs. Fig. 3 explains the Cuckoo architecture.

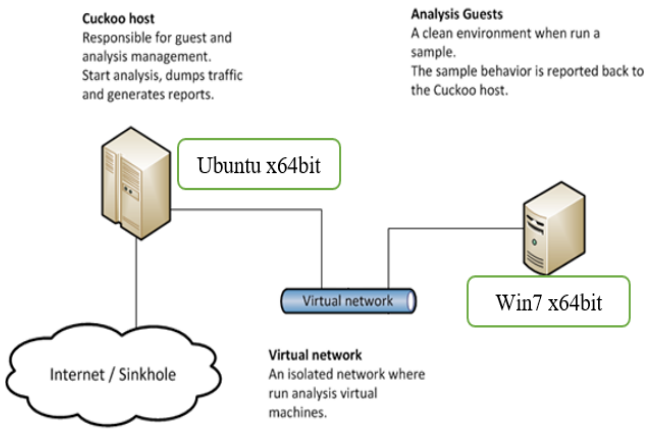

Cuckoo host
Responsible for guest and analysis management. Start analysis, dumps traffic and generates reports.
Analysis Guests
A clean environment when run a sample. The sample behavior is reported back to the Cuckoo host.
Ubuntu x64bit
Virtual network
Win7 x64bit
Internet / Sinkhole
Virtual network
An isolated network where run analysis virtual machines.

Fig. 3. The Cuckoo Sandbox's Architecture.

Step 2: Extract API call sequence

Then information regarding the API calls that were executed during the dynamic analysis is extracted, including information about the API functions list in the order of execution and the number of times each function was executed for each individual PE file, as displayed in Fig. 4.

Step 3: Feature Engineering and Normalization

The extracted data is then compiled into a summary of information for the entire sample. The information in the resulting table includes a list of all sample PE files along with information regarding the API calls that each file used and the number of times each function was used. With regard to API functions that are not called, the corresponding value for that function for the sample file is set to 0. At the same time, each executable file pattern is labeled corresponding to the type of ransomware or benign file, as in Table III. Because LightGBM works effectively with a sparse dataset, all features will be used in the training and testing phases.

TABLE III. LIST OF CLASS LABELS

No.	Class name	Label
1.	Benign	0
2.	Reveton	1
3.	TeslaCrypt	2
4.	Win32:Ransom	3
5.	Win32:Cryptor	4
6.	Win32:Crypt	5
7.	LockScreen	6
8.	WannaCry	7
9.	Win32:FileCoder	8

Data Normalization: Research used the MinMaxScaler of scikit-learn for data normalization. Given that the scope of the raw data is very wide, for some machine learning algorithms their objective functions will not work properly and may produce bias when the data is not normalized. MinMaxScaler normalization scales the range of all features to the range of [0, 1].

The transformation is given by Alg.3:

Algorithm 3. The MinMaxScaler nomarlization

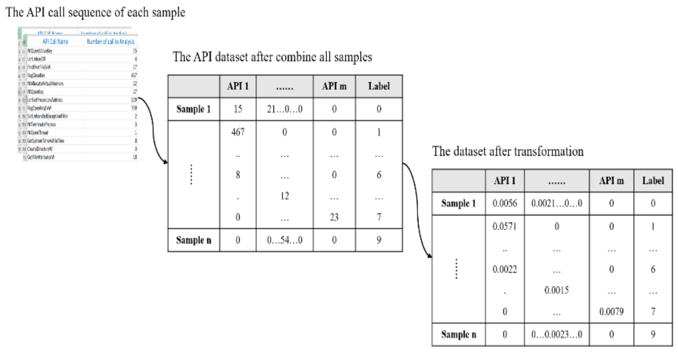
$$X_{std} = \frac{X - X_{\min(axis=0)}}{X_{\max(axis=0)} - X_{\min(axis=0)}}$$

$$X_{Scaled} = X_{std} * (max - min) + min$$

Where: Xmin, Xmax: min, max of one feature

min, max: min, max of overall data.

The following Fig. 4 shows the data normalization process from after extracting sandbox log file to before feeding them to LightGBM algorithm.


The API call sequence of each sample
API Call Name
Number of call in Analysis
The API dataset after combine all samples
The dataset after transformation

	API 1	......	API m	Label
Sample 1	15	21...0...0	0	0
⋮	467	0	0	1
	..	...	...	...
	8	...	0	6
	.	12	...	...
	0	...	23	7
Sample n	0	0...54...0	0	9

	API 1	......	API m	Label
Sample 1	0.0056	0.0021...0...0	0	0
⋮	0.0571	0	0	1
	..	...	...	...
	0.0022	...	0	6
	.	0.0015	...	...
	0	...	0.0079	7
Sample n	0	0...0.0023...0	0	9

Fig. 4. The Transformation of the Data.

All data is divided as follows: training set: 80%, test set: 20%. Each dataset is then used for the subsequent training and test phases.

Step 4: Training phase

In [17], Dongzi et al. illustrated the LightGBM training process with the model consists M trees in Algorithm 4.

Algorithm 4. The training of LightGBM

```
Require: Input: Training set $\{(x_i, y_i)\}_{i=1}^{N}$
Ensure: Output: LightGBM model $\hat{y}_i^{(t)}$
1. Initialize the first tree as a constant: $\hat{y}_i^{(0)} = f_0 = 0$
2. Train the next tree by minimizing the loss function:
    $f_t(x_i) = \underset{f_t}{\text{argmin}}\, L_{(t)} = \underset{f_t}{\text{argmin}}\, L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i))$
3. Get the next model in an additive manner:
    $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
4. Repeat the Step 2 and Step 3 until the model reaches the stop condition.
5. Obtain and return the final model:
$\hat{y}_i^{(t)} = \sum_{t=0}^{M-1} f_t(x_i)$
```

Here: $f_t(x_i)$ and $\hat{y}_i^{(t)}$: are correspondingly the learned function and predictive value of sample i at iteration t.

$L_{(t)}$: The loss function represents the error between the prediction y^and the true value y.

The stop condition of the training process occurs when the process reaches the M-th iteration or when the loss value of the model is lower than the predefined loss value.

Step 5: Testing phase

The test dataset is classified based on GBDT trees that have been created during training phase.

V. Experimental Result

A. Evaluation Criteria

To evaluate the detection performance of the proposed method, this paper employed the following metrics: the accuracy of the entire model, precision, recall, the F1 measure of each class, and a confusion matrix.

When evaluating each class, the class being evaluated is the positive class and the remaining eight classes are the negative class. True positive (TP) refers to the number of positive class samples that are correctly classified. False positive (FP) refers to the number of negative class samples that are misclassified into the class under evaluation. True negative (TN) represents samples in the negative class are correctly classified into the negative class. False negative (FN) represents samples in the consideration class that are misclassified.

Precision is defined as the ratio of true positive scores among those classified as positive.

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

Recall is the fraction of the relevant samples that are successfully classified.

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

The F1 score is used to evaluate the quality of the model.

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

Accuracy is determined simply by calculating the ratio between the number of correctly classified samples and the total in the test dataset.

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

For multiclass classification, the overall accuracy is the ratio of the sum of the true positives of all families divided by the total sample. It is determined according to the following formula.

$$Overall - accuracy = \frac{correctly\ classified\ instances}{total\ number\ of\ instances}$$

The confusion matrix (CM), M = $[m_{x,y}]_{n \times n}$, is used to evaluate the quality of the classifier's output on the dataset. The values of the diagonal elements represent the number of samples and the percentage of correct predictions, while the other elements represent the samples that have been classified incorrectly. A confusion matrix with higher diagonal values represents a higher percentage of correct predictions.

B. Experimental Results

As shown in Table IV, the classification accuracy is very high, with overall accuracy of about 98.7%. However, the correct identification rate for all ransomware is close to 96%, while the lowest rate of identification for TeslaCrypt is 89.5%.

TABLE IV. Classification Evaluation Results

No.	Classes	Size	Precision	Recall	F1-Score
1	Reveton	102	0.961	0.961	0.961
2	TeslaCrypt	38	0.971	0.895	0.932
3	Win32:Ransom	38	1.000	0.921	0.959
4	Win32:Cryptor	25	0.885	0.920	0.902
5	Win32:Crypt	21	0.800	0.952	0.870
6	LockScreen	22	1.000	1.000	1.000
7	WannaCry	100	0.980	1.000	0.990
8	Win32:FileCoder	3	1.000	1.000	1.000
9	Benign	814	1.000	0.999	0.999
Total		1163	Overall accuracy = 0.987		

Fig. 5 presents the CM of the eight ransomware classes, in this case Reveton, TeslaCrypt, Win32:Ransom, Win32:Cryptor, Win32:Crypt, LockScreen, WannaCry, and Win32:FileCoder, along with the benign files in the experiments. The CM shows that the proposed method provides the best classification for three ransomware families, LockScreen, WannaCry and Win32:FileCoder, in which not a single sample is misclassified. This is followed by Reveton, with accuracy of approximately 96.1%.

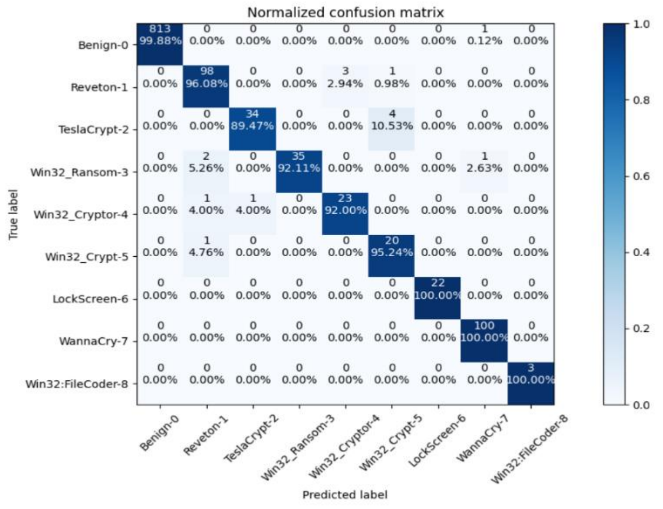


Fig. 5. Confusion Matrix.

Fig. 6 shows feature importance based on the number of times that a feature is used as split points in all learned trees. The most important API call function for detection and classification of the ransomware in this case is “NtOpenKey,” used more than 1000 times as a split point in a learned tree. This is followed by the two other API call functions of “NtAllocateVirtualMemory” and “NtTerminateProcess,” called 995 and 866 times, respectively. The three API call functions above are also among the API call functions most commonly used by samples (ranks: “NtTerminateProcess:” 1st, “NtOpenKey:” 4th, and “NtAllocateVirtualMemory:” 21st). However, while the "NtAllocateVirtualMemory" and "NtTerminateProcess" functions were used many times by all PE files (19,564,674 and 5,182,284 times) and are correspondingly ranked 3rd and 12th, the “NtOpenKey” function was only called 86,487 times and is ranked 111th of 286 (as showed in Table V). This shows that the importance of features in determining split points does not depend much on the number of times they are called or the number of file that used them.

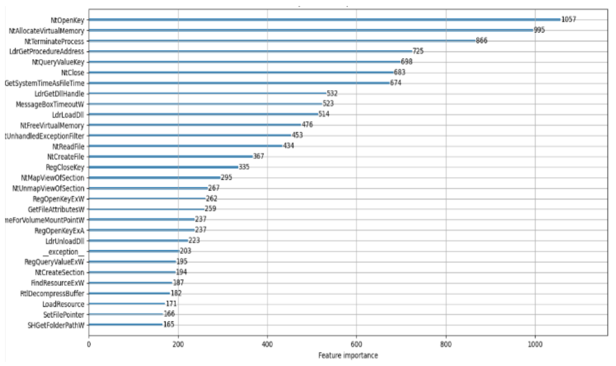


Fig. 6. The Top 30 Important Features in the Experiment.

VI. Analysis and Discussion

The experimental results have shown that the method proposed here to identify and classify malicious codes based on API call functions obtained from dynamic analysis results combined with the GBDT LightGBM algorithm can achieve high performance.

TABLE V. The use of API Call by PE Files

No.	API name	Times used
1	RegSetValueExA	26,547,071
2	ShellExecuteExW	26,135,049
3	**NtAllocateVirtualMemory**	**19,564,674**
4	NtOpenFile	14,268,859
5	OpenServiceW	9,484,238
…	…	…
12	**NtTerminateProcess**	**5,182,284**
…	…	…
111	**NtOpenKey**	**86,487**
…	…	…
286	NtShutdownSystem	1

The proposed method identified ransomware samples with very high accuracy, reaching 100%. When evaluating the effectiveness of distinguishing between malicious code and normal software, the experimental results show that the system did not miss or miscategorized any ransomware sample as normal software. This promises to bring about a positive effect with regard to protecting system and user data. At the same time, the rate of misidentification of benign software was low, as less than 0.01% of benign samples were misidentified as malicious samples. Therefore, the proposed method will not affect system availability and allows the user experience to be retained.

When evaluating the effectiveness of classifying ransomware types, out of eight types of malware conducted experimentally, the proposed method has the ability to identify sensitively three types of extortion malware. In this case LockScreen, WannaCry, and Win32:FileCoder, with absolute precision of 100%. In addition, there were a few small mistakes between different types of malware, such as TeslaCrypt, Win32:Ransom, and Win32:Crypt.

The test results here also demonstrated the advantages of the dynamic analysis in support of ransomware detection. The method based on dynamic analysis greatly reduced the number of features in the sample database. According to this study of the collected dataset, the number of features (API call functions) that must be analyzed and processed during the static analysis method is very large at approximately 6,684 different features with ransomware and nearly 28,500 different features in the benign case. Meanwhile, using the dynamic method, the number of API call functions to be processed for all ransomware and benign samples was only 282. This enhances the efficiency of the data analysis, classification and processing steps while also minimizing the time required for the training and detection phases, which are very time-consuming steps given numerous features. The proposed method also minimizes interference from an attacker to bypass static-analysis-based methods, such as by adding normal API call patterns or by attempting to use an obfuscation technique.

VII. Conclusion

This chapter discusses the results of the proposed method, the advantages of the dynamic analysis technique in supporting ransomware detection and classification.

In fact, proposal method achieves a 98.7% classification accuracy rate, with excellent ransomware recognition and a low error rate during benign software classification. Compared to previous studies, the experimental results not only dominate in terms of detection between ransomware samples and goodware (99.9% accuracy versus 98.65%, 97.74%, and 97.3% as in [6], [4] and [3] but is also more efficient when classifying ransom types of malware (between proposal method at 96% and corresponding rates of 88.9%, 94.2%, and 91.4% as in [2], [13], [12]. This helps to increase the efficiency of the identification process of malicious code, thereby accelerating the response and implementing countermeasures to protect the system when necessary. In particular, using LightGBM algorithm significantly shortens the time compared to other machine learning or GBDT algorithms.

The study also demonstrated the role of each API function in identifying and classifying ransomware by assessing the importance of each API function by determining the split points during the construction of the learning trees used here. This makes it possible for us to engage in more research and evaluations to reducing the number of attributes further when the number of file samples increases in the future. This helps to reduce the computational pressure while maintaining the accuracy of the method, thus enhancing the efficiency of the system.

The proposed plan has shown very positive results. Experiments also highlighted the importance of each API function in the detection and classification process. Therefore, work to reduce the number of API functions when the numbers of ransomware samples and types increase in order to reduce the computational burden while ensuring high accuracy also represents a promising research direction for future studies.

References

[1] Sgandurra D., Muñoz-González L., Mohsen R., Lupu E.C., 2016. Automated dynamic analysis of ransomware: benefits, limitations and use for detection. Cryptography and security, arXiv:1609.03020.

[2] Vinayakumar R., Soman K.P., Senthil Velan K.K., Ganorkar S., 2017, Evaluating shallow and deep networks for ransomware detection and classification. International conference on advances in computing, communications and informatics, pp259-265, doi:10.1109/ICACCI.2017.8125850.

[3] Hwang J., Kim Y., Lee S., Kim K., 2020. Two-stage ransomware detection using dynamic analysis and machine learning techniques. Wireless personal communications volume 112, 2597–2609. doi:10.1007/s11277-020-07166-9.

[4] Khammas B.M.., 2020, Ransomware detection using random forest technique, ICT Express,Volume 6, Issue 4, pp325-331. doi:10.1016/j.icte.2020.11.001.

[5] Sikorski M. and Honig A., 2012. Practical malware analysis: The hands-on guide to dissecting malicious software, William Pollock Publisher, 38 Ringold Street, San Francisco, CA, 802pp.

[6] Bae S.I., Lee G.B., Im E.G., 2020. Ransomware detection using machine learning algorithms. Concurrency computat pract exper. doi:10.1002/cpe.5422.

[7] Kharraz A., William R., Davide B., Leyla B., Engin K., 2015. Cutting the gordian knot: A look under the hood of ransomware attacks. 12th International conference: Detection of intrusions and malware, and vulnerability assessment, Milan, Italy, pp3-24. doi:10.1007/978-3-319-20550-2 1.

[8] Lee K., Lee S.Y., Yim K.B., 2019. Machine learning based file entropy analysis for ransomware detection in backup systems. IEEE Access, vol. 7, pp110205-110215. doi:10.1109/ACCESS.2019.2931136.

[9] Cabaj K., Gregorczyk M., Mazurczyk W., 2018. Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics. Computers & electrical engineering, vol66, pp353-368. doi:10.1016/j.compeleceng.2017.10.012.

[10] Almashhadani A.O., Kaiiali M., Sezer S., O'Kane P.,2019. A multi-classifier network-based crypto ransomware detection system: a case study of locky ransomware. IEEE Access, vol. 7, pp47053-47067, doi:10.1109/ACCESS.2019.2907485.

[11] Shaukat S.K., Ribeiro V.J., 2018. RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning. 10th International conference on communication systems & networks (COMSNETS), pp. 356-363. doi:10.1109/COMSNETS.2018.8328219.

[12] Zhang H., Xiao X., Mercaldo F., Ni S., Martinelli F., Sangaiah A.K., 2019. Classification of ransomware families with machine learning based onN-gram of opcodes. Future generation computer systems, vol90, pp211-221. doi: 10.1016/j.future.2018.07.052.

[13] Baldwin J., Dehghantanha A., 2018. Leveraging support vector machine for opcode density based detection of crypto-ransomware. In: Dehghantanha A., Conti M., Dargahi T. Cyber threat intelligence. advances in information security, vol70. Springer, Cham. doi:10.1007/978-3-319-73951-9_6.

[14] Haijian Shi, 2007. Best-first decision tree learning. PhD thesis, The University of Waikato, pp120.

[15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, 2017. LightGBM: A highly efficient gradient boosting decision tree. Microsoft, https://www.microsoft.com/en-us/research/publication/lightgbm-a-highly-efficient-gradient-boosting-decision-tree/ [accessed 15 January, 2021].

[16] Mohammad A.A., Ahmed M.A., Mouhammd A., 2020. Robust intelligent malware detection using light gbm algorithm. International journal of innovative technology and exploring engineering (IJITEE), doi:10.35940/ijitee.F4043.049620.

[17] Dongzi J., Yiqin L., Jiancheng Q., Zhe C., Zhongshu M., 2020. SwiftIDS: Real-time intrusion detection system based on LightGBM and parallel intrusion detection mechanism. Computers & security. doi:10.1016/j.cose.2020.101984.

[18] C. Rossow , 2012. Prudent practices for designing malware experiments: status quo and outlook. IEEE symposium on security and privacy, pp65-79, doi:10.1109/SP.2012.14.