Classification of ransomware families with machine learning based on N-gram of opcodes

Hanqi Zhang, Xi Xiao, Francesco Mercaldo, Shiguang Ni, Fabio Martinelli, Arun Kumar Sangaiah

PII: S0167-739X(18)30732-5
DOI: https://doi.org/10.1016/j.future.2018.07.052
Reference: FUTURE 4370

To appear in: *Future Generation Computer Systems*

Received date : 31 March 2018
Revised date : 9 July 2018
Accepted date : 25 July 2018

Classification of Ransomware Families with Machine Learning Based on *N*-gram of Opcodes

Hanqi Zhang[a, b], Xi Xiao[b], Francesco Mercaldo[c], Shiguang Ni[b*], Fabio Martinelli[c], Arun Kumar Sangaiah[d]

[a]*College of Physical Science and Technology, Central China Normal University, Wuhan, China*
[b]*Graduate School at Shenzhen, Tsinghua University, Shenzhen, China*
[c]*Institute for Informatics and Telematics, National Research Council of Italy, Pisa, Italy*
[d]*School of Computing Science and Engineering, VIT University, Vellore, India*

Abstract: Ransomware is a special type of malware that can lock victims' screen and/or encrypt their files to obtain ransoms, resulting in great damage to users. Mapping ransomware into families is useful for identifying the variants of a known ransomware sample and for reducing analysts' workload. However, ransomware that can fingerprint the environment can evade the precious work of dynamic analysis. To the best of our knowledge, to overcome this shortcoming, we are the first to propose an approach based on static analysis to classifying ransomware. First, opcode sequences from ransomware samples are transformed into *N*-gram sequences. Then, Term frequency-Inverse document frequency (*TF-IDF*) is calculated for each *N*-gram to select feature *N*-grams so that these *N*-grams exhibit better discrimination between families. Finally, we treat the vectors composed of the *TF* values of the feature *N*-grams as the feature vectors and subsequently feed them to five machine-learning methods to perform ransomware classification. Six evaluation criteria are employed to validate the model. Thorough experiments performed using real datasets demonstrate that our approach can achieve the best *Accuracy* of 91.43%. Furthermore, the average *F1-measure* of the "wannacry" ransomware family is up to 99%, and the *Accuracy* of binary classification is up to 99.3%. The proposed method can detect and classify ransomware that can fingerprint the environment. In addition, we discover that different feature dimensions are required for achieving similar classifier performance with feature *N*-grams of diverse lengths.

Keywords: ransomware classification, static analysis, opcode, machine learning, *N*-gram

1. Introduction

According to Internet Security Threat Report [1], more than 357 million new malware samples were disclosed in 2016. Malicious software can erode or steal data and destroy computer systems [2-4]. Ransomware is one of the most harmful types of malware, and it forces victims to pay ransoms in exchange for encrypted data [5]. Unlike traditional malware, it is difficult to kill ransomware even when it is discovered, and the loss is irreversible even its removal [6]. The earliest known ransomware "AidsInfo" was discovered in 1989. Thereafter, ransomware developed rapidly, and in recent years, many new families of ransomware have emerged. Last year, "wannacry" ransomware infected more than 200,000 computers in 150 countries in less than a day [7], posing enormous danger to users.

Ransomware classification can help identify the variants of a known ransomware sample and effectively reduce the workload of analysts. However, most studies on ransomware pertain to its

* Corresponding author at: Graduate School at Shenzhen, Tsinghua University, Shenzhen, China
Tel: +86 18038153875. E-mail address: ni.shiguang@sz.tsinghua.edu.cn

detection, and to the best of our knowledge, there has been only one work on ransomware classification [8]. In the said work, researchers collected application programming interface (API) sequences from samples of seven ransomware families by dynamically simulating the execution of ransomware in a sandbox environment; then, they applied machine learning algorithms to perform the classification. They found that the classifier with multi-layer perception (MLP) achieved the best performance. However, ransomware with the capability to fingerprint dynamic environments can evade dynamic analysis [9]. Therefore, methods based on dynamic analysis fail in detecting this type of ransomware. To overcome this drawback of dynamic analysis and fill the gaps in the literature on ransomware family classification, we propose an approach based on static analysis, which does not require running software dynamically. With this approach, we can detect and classify ransomware instances even when the said instances employ fingerprinting techniques.

In the present work, we collect a total of 1787 ransomware samples from eight families, i.e., “cryptolocker”, “cryptowall”, “cryrar”, “locky”, “petya”, “reveton”, “teslacrypt”, and “wannacry”, and 100 trusted software samples. We first transform the opcode sequences to *N*-gram sequences and arrange *N*-grams by *TF-IDF* in descending order to select feature *N*-grams. Then, *TF* is calculated for each feature *N*-gram, which constitutes the feature vector. Based on these vectors, five machine-learning algorithms, namely, Decision Tree, Random Forest, K-Nearest Neighbor, Naive Bayes, and Gradient Boosting Decision Tree, are applied to build classification models. We perform extensive experiments with various *N*-grams (N = 2, 3, and 4) and different feature dimensions (29-228). The experimental results show that the proposed method can achieve good classification performance when using the Random Forest algorithm. The best *Accuracy* of the proposed approach is 91.43% and the *F1-measure* for one of the eight ransomware families, i.e., wannacry, consistently remains as high as 99%. Moreover, the *Accuracy* of binary classification between ransomware and trusted software is up to 99.3%. Further, we find that to achieve similar performance, *N*-grams with different lengths require diverse numbers of feature *N*-grams.

The three main contributions of the present paper are as follows.

1) To the best of our knowledge, we are the first to leverage a static method for ransomware classification, which overcomes the primary shortcoming of dynamic analysis. The proposed method can detect and classify ransomware that can fingerprint the environment.

2) We employ *N*-grams as features and apply *TF-IDF* to select feature *N*-grams so that these *N*-grams can discriminate between families. Thus, the proposed method achieves high *Accuracy* both in multi-classification and binary classification.

3) We conduct comprehensive experiments on real-world datasets. Different machine-learning algorithms are used to investigate the effects of *N*-grams of various lengths and feature vectors of diverse dimensions. Therefore, the present work is done in many aspects of a real environment.

The remainder of this paper is organized as follows. In Section 2, we introduce related works on malware, as well as ransomware detection and classification. In Section 3, we describe our methodology, and in Section 4, we explain the experimental datasets and library, time of proposed approach, significant *N*-grams of each family, evaluation metrics, and experimental results. In Section 5, we present a few concluding remarks and an outline for future work.

2. Related Works

In this section, a few works on malware and ransomware are mentioned because ransomware is a specific type of malware. We first introduce recent studies on multi-classification of malware families based on API sequences and opcodes, which is similar to our multi-classification work. Then, we present several investigations on ransomware analysis, detection, and multi-classification since ransomware is a special type of malware.

In terms of API sequences, specific API sequences must be executed by attackers to achieve malicious purposes, and an analysis of API sequences can help us better understand these malicious purposes [10]. Therefore, a few analysts have proposed methods based on API sequences as follows. Kwon *et al*. [11] extracted API sequences as features through dynamic analysis in a sandbox environment and then employed recurrent neural network (RNN) to identify malware. The average *Accuracy* was 71% because the API sequences of much malware are similar. Mohaisen *et al*. [12] built a system named CHATTER to classify malware, in which it was assumed that two malware samples from the same family have similar calling orders of system events. CHATTER can analyze high-level system events and map individual events onto an alphabet. Then, it concatenated those letters to capture execution traces and utilized the *N*-gram to extract useful sequences. In the experiment of Mohaisen *et al*., *Accuracy* was approximately 80%. Hansen *et al*. [13] regarded API sequences as features and applied information gain ratio. The weighted average *F1-measure* was 86.4%. Similarly, Pekta *et al*. [10] extracted *N*-grams from API sequences as features and calculated *TF-IDF* as feature value. The best *Accuracy* was 92.5%, indicating that *N*-grams have better ability in terms of extracting meaningful sequences. In Refs. [10-13], API sequences were obtained by running the software. These methods belong to dynamic analysis. However, malware may be able to fingerprint the environment of dynamic analysis [9] and implement some measures, such as stalling code [14], to prevent execution by an analyst, which makes the research more difficult. Therefore, malware that can fingerprint the environment cannot be detected by dynamic analysis, which is the limitation of the dynamic analysis of malware.

Considering the above-mentioned limitation of dynamic analysis, to compensate for insufficiency of dynamic analysis, a few scholars adopted opcodes as features, and such methods fall under static analysis, which we introduce later in the paper. Opcodes exist in the assembly code of software [15] and can be obtained through disassembly. Therefore, researchers can analyze malware without simulated execution. Liangboonprakong *et al*. [16] applied a static method based on text analysis. *N*-grams were extracted from opcodes, and *TF-IDF* was treated as feature value. Sequential floating forward selection was then utilized to reduce the feature dimension, and the Decision Tree, Artificial Neural Network, and Support Vector Machine algorithms were applied. However, in [10, 16], *TF-IDF* was regarded as feature value. Yet, an *N*-gram may have similar *TF-IDF* values in different families because *IDF* remains the same and the *TF* of different families may be similar, resulting in low discrimination of this *N*-gram, and the significant *N*-grams cannot be extracted for classification. Hassen *et al*. [17] presented a method with *N*-gram and stop words. They regarded the JMP, LOOP, and CALL opcodes as stop words denoting delimiters in text analysis to divide different functional parts and used only one *N*-gram from each functional block. In their approach, the factor of human subjectivity was unavoidable, which led to lack of objectivity and authenticity. This is one of the gaps of multi-classification based on *N*-gram. In addition, various other methods [18-22] have been proposed.

Ransomware is a special category of malware, in which the attackers' target is ransom. In this part, a few investigations related to the analysis and detection of ransomware are presented first, and then, we introduce the research on multi-classification. Kumar *et al*. [23] introduced the principles of encryption and ways of attack of different types of ransomware. Gaze [5] analyzed four families of ransomware and discovered that symmetric, asymmetric, and hybrid encryption methods are used by ransomware. Kharraz *et al*. [24] investigated the core part of 1359 ransomware samples of 15 families and indicated that several different ransomware exhibited similar characteristics when attacking a system. Based on deeper analyses of ransomware families, therefore, several analysts performed ransomware detection based on ransomware characteristics. In [7, 25], the authors detected ransomware by using honeypot techniques, which can stop ransomware from attacking a system when it infects trap files. Kharraz *et al*. [9] established a system named UNVEIL that can simulate users' interfaces to detect unusual behavior. Most works in the literature [7, 9, 25-30] have focused on ransomware detection. Many studies pertaining to the analysis and detection of ransomware have been conducted but few pertaining to classification. Researchers can focus on the analyses of unknown ransomware families based on the classification of known families. To our knowledge, Vinayakumar *et al*. [8] is the only study related to ransomware classification. In their method, samples from seven ransomware families and a benign software family were used to perform classification. The method not only predicted whether an instance was benign or malicious but also classified a sample in one family. The authors considered 131 API sequences collected from a sandbox during dynamic analysis and employed MLP to train the model. The experimental results showed that the performance of the classifier with MLP was better than that of the classifiers with other machine-learning algorithms. In MLP, the *Accuracy* was 98%; in the case of ransomware families, i.e., cryptolocker and cryptowall, the best TPR is only 88.9% and 83.3%. Owing to the drawback of dynamic analysis described above, researchers cannot extract significant API sequences if the environment is fingerprinted by the ransomware.

Table 1 shows a comparison between several of the aforementioned studies on family classification. Given that we could find only one work on ransomware classification, i.e., Vinayakumar *et al*. [8], so we add a few studies related to malware classification for comparison. In Refs. [11, 13], API sequences were extracted based on dynamic analysis. However, in dynamic analysis, if the malware/ransomware can fingerprint the environment, the analysis fails. In [10, 16], *TF-IDF* was employed as the value of feature *N*-grams. However, the *TF-IDF* of the same *N*-gram may be similar in different families, which hinders the extraction of significant *N*-grams. In [17], human intervention is required for setting stop words. Vinayakumar *et al*. [8] presented an approach based on dynamic analysis, which, too, is affected by the limitation of dynamic analysis.

Table 1
Comparison of extant classification literature

Authors	Techniques	Weaknesses
Kwon *et al*. [11] Hansen *et al*. [13]	API sequence	Cannot classify malware instances that can fingerprint the environment
Pekta *et al*. [10] Liangboonprakong *et al*. [16]	*N*-gram, *TF-IDF*	Cannot extract significant *N*-grams
Hassen *et al*. [17]	*N*-gram	Requires human intervention
Vinayakumar *et al*. [8]	MLP	Cannot classify ransomware instances that can fingerprint the environment

In order to fill the gaps in current methods of multi-classification of ransomware and overcome the drawback of dynamic analysis, in the present paper, we propose an approach based on static analysis that does not need to dynamically simulate ransomware execution and can perform classification even when fingerprinting techniques are employed by ransomware. To the best of our knowledge, we are the first to aim to discriminate between ransomware families by exploiting static analysis, which overcomes the primary limitation of dynamic analysis.

3. Methodology

Fig.1 shows the workflow of the proposed method. First, we extract the *N*-gram opcode sequences from the original datasets. Second, meaningful feature *N*-grams are selected. Third, we calculate the *TF* values of each *N*-gram sequence to constitute a feature vector. Finally, the feature vectors are employed to train the classification models.

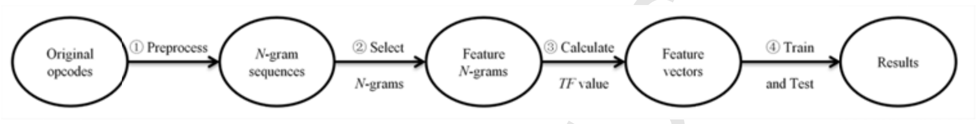

Fig.1. Workflow of proposed method

3.1 Preprocessing

The original datasets are composed of opcodes obtained from ransomware samples of eight widespread families by using the IDAPRO disassembler. The proposed method exploits static analysis, that is, it does not require the samples to run on physical machines. *N*-grams are extracted from the opcodes of each ransomware sample. For example, Fig.2 demonstrates the procedure of 3-gram extraction, in which one rectangle stands for one ransomware sample, and (sub, mov, mov) represents a 3-gram. The sequence consisting of the *N*-grams of one ransomware sample is defined as an *N*-gram sequence in our method. Algorithm 1 demonstrates the procedure of preprocessing. We employ the *N*-grams of different lengths to execute Algorithm 1, so do Algorithms 2 and Algorithm 3.

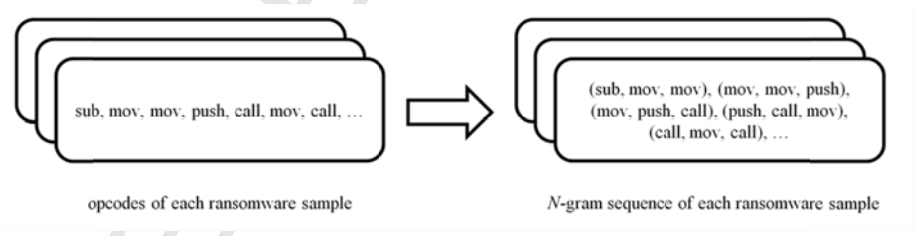
sub, mov, mov, push, call, mov, call, …
(sub, mov, mov), (mov, mov, push), (mov, push, call), (push, call, mov), (call, mov, call), …
opcodes of each ransomware sample
N-gram sequence of each ransomware sample

Fig.2. 3-gram extraction

Algorithm 1 Preprocessing

```
Input:
    opcodeSampleSet = set of all ransomware samples contain opcodes
Output:
    ngramSequenceSet = set of all N-gram sequences
1: function Preprocessing(opcodeSampleSet)
2:     for sample in opcodeSampleSet do
3:         transform opcodes in sample to ngrams
4:         connect ngrams to ngramSequence
5:         insert ngramSequence to ngramSequenceSet
6:     end for
7: end function
```

We select *N*-grams for the following reasons. A single opcode in malware is not harmful to the system independently, but an opcode sequence composed of more than one opcode in a specific order can be harmful. Further, an *N*-gram is a short sequence of *N* opcodes. Therefore, a few meaningful opcode sequences can be captured by the *N*-gram. The *N*-grams have the ability to predict the occurrence of phenomena [31]. We extract *N*-grams of length 2, 3, and 4, separately, from each family in the raw datasets.

3.2 Selection of N-grams

A large number of *N*-grams are present in the eight families. It is impossible to use all *N*-grams to generate the feature vectors. *TF-IDF* is a typical approach to identify important words, and it is based on the theory of language modeling [32]. *TF-IDF* contains heuristic intuition that a term that appears in many documents is not a good discriminator and should be given less weight than one that occurs in few documents [32]. We calculate *TF-IDF* for each *N*-gram by using Eqs. (1)-(3):

$$TF(i,j) = \frac{n(i,j)}{\sum_k n(k,j)} \tag{1}$$

where *TF*(*i*,*j*) denotes the frequency of *N*-gram *i* in *N*-gram sequence *j*, *n*(*i*,*j*) represents the number of times *N*-gram *i* occurs in *N*-gram sequence *j*, and $\sum_k n(k,j)$ stands for the number of *N*-grams in *N*-gram sequence *j*. Then,

$$IDF(i) = log_2 \frac{|D|}{|\{j: i \in j | j \in D\}|} \tag{2}$$

where *IDF*(*i*) describes whether *N*-gram *i* is rare in all *N*-gram sequences, and *D* is the set of all *N*-gram sequences. $|\{j: i \in j | j \in D\}|$ refers to the number of *N*-gram sequences that contain *N*-gram *i*. Finally,

$$TF\text{-}IDF(i,J_f) = TF(i,J_f) \times IDF(i) \tag{3}$$

In Eq. (3), we combine all *N*-gram sequences from one ransomware family *f* to form a long *N*-gram sequence J_f. $TF\text{-}IDF(i,J_f)$ represents the *TF-IDF* value of *N*-gram *i* in the long *N*-gram sequence J_f.

TF-IDF evaluates the importance of an *N*-gram in a ransomware family, which increases proportionally with the times of occurrence of the *N*-gram in a family. Moreover, it is inversely proportional to the number of occurrences of the same *N*-gram in all ransomware families [33]. *TF-IDF* can distinguish each family from other families to the extent possible because the *N*-gram of large *TF-IDF* implies that the occurrence frequency in one family is high and the occurrence frequency in other families is low.

We sort the *N*-grams in each family by their *TF-IDF* values in descending order and select the top *t* *N*-grams in each family, which are called feature *N*-grams. We then combine 8 × *t* feature *N*-grams from the eight families to form the feature *N*-grams of one classifier. Since several same *N*-grams exist in the feature *N*-grams extracted from different families, we eliminate the redundant feature *N*-grams. Algorithm 2 depicts the program executed to realize this step.

3.3 Calculation of TF Value

We calculate the *TF* value of each feature *N*-gram in one *N*-gram sequence by using Eq. (1). All values of the feature *N*-grams in an *N*-gram sequence constitute one vector, which we call the feature vector of the corresponding ransomware. Fig.3 illustrates one example of the generation of a feature vector. The number of *N*-grams in the *N*-gram sequence is 10, and the number of times an *N*-gram (sub, mov, mov) occurs in the *N*-gram sequence is 2; then, we get the *TF* value of (sub, mov, mov) as 0.2 by using Eq. (1). We obtain the *TF* values of all feature *N*-grams in the same way to form one vector. By repeating the above process for each *N*-gram sequence to form one vector, we obtain all feature vectors. Algorithm 3 describes the process of calculation.

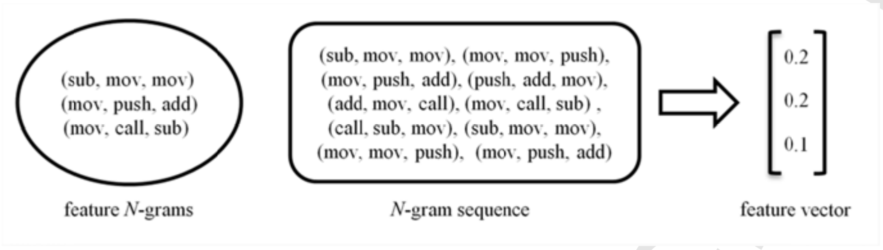
(sub, mov, mov)
(mov, push, add)
(mov, call, sub)
(sub, mov, mov), (mov, mov, push),
(mov, push, add), (push, add, mov),
(add, mov, call), (mov, call, sub) ,
(call, sub, mov), (sub, mov, mov),
(mov, mov, push), (mov, push, add)
0.2
0.2
0.1
feature N-grams
N-gram sequence
feature vector

Fig.3. Generation of one feature vector

Algorithm 2 SelectionOfNgrams

```
Input:
    $ransomwareFamilySet$ = the set of ransomware family
    $t$ = the number of N-grams we select in each family
Output:
    $ngramSet$ = set of $8 \times t$ N-grams in $ransomwareFamilySet$
 1: function $SelectionOfNgrams(ransomwareFamilySet, t)$
 2:     for $family$ in $ransomwareFamilySet$ do
 3:         for $ngramSequence$ in $ngramSequenceSet$ do
 4:             if $ngramSequence$ is from $family$ then
 5:                 insert $ngramSequence$ to $longNgramSequence$
 6:             end if
 7:         end for
 8:         $D$ = the number of $ngramSequence$ in $ngramSequenceSet$
 9:         for $ngram$ in $longNgramSequence$ do
10:             $TF[ngram]$ = the frequency of $ngram$ in $longNgramSequence$
11:             $F = 0$
12:             for $ngramSequence$ in $ngramSequenceSet$ do
13:                 if $ngram$ in $ngramSequence$ then
14:                     $F = F + 1$
15:                 end if
16:             end for
17:             $IDF[ngram] = \log(D/F)$
18:             $TF\text{-}IDF[ngram] = TF[ngram] \times IDF[ngram]$
19:         end for
20:         sort $TF\text{-}IDF$ by value
21:         insert top $t$ $ngram$ in $TF\text{-}IDF$ to $ngramSet$
22:     end for
23: end function
```

The dimension of the feature vector is the number of different feature *N*-grams. Thus, we define the feature dimensions as the number of different feature *N*-grams.

3.4 Training and Testing

After obtaining all feature vectors, we use the feature vectors with known labels to train the classifiers, that is, we resort to supervised classification. Finally, the trained classifiers can predict the labels of new instances in the form of feature vectors. Then, the performance of the method can be validated. In our work, we apply several traditional machine-learning algorithms, namely, Decision Tree, Random Forest, K-Nearest Neighbor, Naive Bayes, and Gradient Boosting Decision Tree, instead of deep learning. We do not employ deep learning because it requires millions of samples and many trial-and-error procedures to adjust the parameters to train the correct model [34]. By contrast, the classifier based on the proposed approach with the machine learning algorithm can perform classification, which does not require a large of samples.

Algorithm 3 CalculationOfTFValue

```
Input:
    $ngramSequenceSet$ = set of all N-gram sequences
    $ngramSet$ = set of $S \times t$ N-grams in $ransomwareFamilySet$
Output:
    $featureVectorSet$ = set of $featureVector$
 1: function $CalculationOfTFValue(ngramSequenceSet, ngramSet)$
 2:     for $ngramSequence$ in $ngramSequenceSet$ do
 3:         $featureVector$ = list()
 4:         for $ngram$ in $ngramSet$ do
 5:             $tf$ = the frequency of $ngram$ in $ngramSequence$
 6:             insert $tf$ to $featureVector$
 7:         end for
 8:         insert $featureVector$ to $featureVectorSet$
 9:     end for
10: end function
```

Decision Tree is an inductive method of machine learning [35]. In this method, a tree is constructed by selecting the attribute that best splits the training examples into their proper classes [35]. The root is the beginning of the classification, and the leaf node denotes the class label [35]. During classification, the sample downs the tree through each node that corresponds to each attribute [35]. In Decision Tree, the effect can be improved mainly by adjusting the metric that evaluates the quality of a feature and by adjusting the maximum depth of the tree.

Random Forest is an ensemble learning method, and it is similar to Decision Tree [17]. In general, Decision Tree applies all feature sequences to generate the model. However, in Random Forest, only a few random features are selected to produce a Decision Tree as a base learner [17], and several base-learners' results are combined via a voting mechanism. In Random Forest, we can modify the number of base learners, number of features to consider for each node, and maximum depth of the base learners to ameliorate the classifier.

K-Nearest Neighbor is a supervised and instance-based learning algorithm in machine learning [36]. The classifier in the K-Nearest Neighbor algorithm identifies *K* nearest neighbors of a few training instances relative to the new sample and then marks a class label according to the greatest number of neighbors relative to the new sample [36]. The group to which the test sample belongs is distinguished based on its distance from the *K* points.

Naive Bayes is a machine learning algorithm based on probability. It calculates the probability that a sample belongs to a given family. Naive Bayes is based on the Bayes theorem, which assumes that all features are independent of each other [37]. It only requires a small number of training data to complete training, and it is useful for text categorization [37].

Gradient boosting is a gradient-based approach that gradually increases the effectiveness of the classifier [38]. Gradient Boosting Decision Tree (GBDT) is one of the several gradient boosting algorithms [38]. It is widely used in machine learning, and it utilizes Decision Tree as a weak classifier [38]. In GBDT, the classifier is improved by adjusting the number of iterations, contribution of each iteration, and proportion of samples used in each iteration.

These five algorithms are simple, common, and widely used. Furthermore, the tree-based algorithm is predominant, and it not only provides data that can be understood easily but is also computationally cheap and great at handling irrelevant features [39].

4. Experiments and Results

We evaluate the effectiveness of the proposed method on real-world ransomware samples from the most widespread families. Here, we first introduce our datasets and library of machine learning. Then, we describe discriminate *N*-grams for each family and list the times of model training and prediction. Thereafter, we introduce six evaluation criteria. Finally, we describe the experimental results in detail.

4.1 Experimental Data and Library

In this section, we introduce our experimental datasets first and then present the machine learning library we employ. We obtain ransomware samples by crawling the VirusTotal API. All downloaded applications are checked using VirusTotal, which confirms that the malicious samples contained ransomware with specific family payloads. We utilize the IDAPRO disassembler to obtain the original opcodes. The interactive disassembler (IDA) is applied to computer software to generate assembly language source code from machine-executable code. IDA can perform automatic code analysis by using cross-references between code sections, knowledge of parameters of API calls, and other information [40].

We use eight families of ransomware that broke out from 2012 to 2017 in our experiments: cryptolocker, cryptowall, cryrar, locky, petya, reveton, teslacrypt, and wannacry. Among these, both cryptolocker and cryptowall encrypt users' files by using the RSA algorithm. Cryrar places the victim's files in encrypted and legal RAR-sfx archives. Locky uses the AES algorithm to encrypt a large number of files when macros in malicious files are enabled. Petya and reveton blackmail users by preventing them from accessing the operating system. Early teslacrypt was designed to encrypt game-related files in the user's system, while newer ones can encrypt other types of files as well. Wannacry attacks through the Windows vulnerability EternalBlue and can spread itself automatically to attack more systems.

In our datasets, the total number of ransomware samples is 1787. The maximum sample number of one family, i.e., wannacry, is 431; minimum sample number of one family, i.e., cryptolocker, is 93; and average sample number of each family is 224. The maximum number of opcodes in one sample, minimum number of opcodes in one sample, average number of opcodes in one family, and total number of opcodes in all samples of one family are listed in Table 2. From Table 2, the family with the largest average number of opcodes, i.e., 338180, is cryrar, and the family with the smallest average number of opcodes, that is, 33047, is cryptolocker.

Table 2
Number of opcodes of all samples in each ransomware family

Families	Maximum	Minimum	Average	Total
cryptolocker	308910	2317	33047	3040409
cryptowall	327162	905	35651	8449358
cryrar	2685130	1063	338180	38552605
locky	268503	59	41597	7820365
petya	2901221	201	110612	17144958
reveton	406920	721	51160	14376123
teslacrypt	254299	1137	49877	14264996
wannacry	1462318	2	35831	15443496

In our work, we use Scikit-learn as the machine-learning library. Scikit-learn is a Python module that integrates a variety of state-of-the-art machine-learning algorithms for supervised and unsupervised problems [41].

By using Scikit-learn, we can adjust the parameters of the algorithms. After several tests with different parameters in different algorithms, we finally selected the following parameters. In Decision Tree, the metric used to evaluate the quality of a feature is set to "gini", and the maximum depth of the Decision Tree is 60. In Random Forest, we set the number of base learners to 130, maximum depth of the base learners to 30, and number of features to consider for each node to "log2", which refers to the logarithm of the number of all features. In the K-Nearest Neighbor algorithm, *K* is set to 4. In GBDT, we set the number of iterations to 150. The contribution of each iteration is 0.2, and the proportion of samples used in each iteration is set to 0.6.

4.2 N-gram with Discrimination and Execution Time

After we obtain feature *N*-grams that can distinguish each family from other families to the extent possible, we select the highest two *N*-grams from each family based on their *TF-IDF* values in descending order and called them *N*-grams with discrimination. We list 2-grams, 3-grams, and 4-grams with discrimination of the cryptolocker, cryrar, petya, and wannacry families in Table 3. Clearly, in cryptolocker and wannacry, most of the *N*-grams with discrimination are composed of the same opcode, such as "inc, inc", "sub, sub, sub", and "movb, movb, movb, movb". However, they consist of different opcodes in the cryrar and petya families, such as "mov, callq", "or, lock, jne", and "lock, jne, mov, push".

Table 3
2-grams, 3-grams, and 4-grams with discrimination of cryptolocker, cryrar, petya, and wannacry families

cryptolocker	cryrar	petya	wannacry
inc, inc	mov, callq	int3, int3	movb, movb
sub, sub	callq, mov	lock, jne	nop, nop
inc, inc, inc	retq, int3, int3	or, lock, jne	movb, movb, movb
sub, sub, sub	mov, callq, mov	lock, jne, mov	stos, jmp, adc
inc, inc, inc, inc	retq, int3, int3, int3	daa, push, inc, daa	movb, movb, movb, movb
sub, sub, sub, sub	pop, retq, int3, int3	lock, jne, mov, push	in, stos, jmp, adc

In the present work, the proposed approach is evaluated based on two time components, namely, time required for classifier training and time required for prediction. The time required for classifier training consists of preprocessing time, selecting *N*-grams time, calculating *TF* value time, and training time, as shown in Fig.1. In the preprocessing time, we create *N*-gram sequences of all *N*-grams in each family. The time required for selecting *N*-grams is the time spent selecting significant *N*-grams. In the time for calculating *TF* value, we transform all *N*-gram sequences into the feature vectors, and these are used as training data in our model. The training time is the time spent training the model. Because the training time with different algorithms is not the same, we calculate the average time of all algorithms. Table 4 shows time required for classifier building, that is, the time for preprocessing, selecting *N*-grams, calculating *TF* value, training, and the total time for different *N*-gram lengths. It is clear that the times required for preprocessing and selecting

N-grams for different *N*-gram lengths are similar. However, the time required to calculate the *TF* value increases fast with increasing of *N*-gram length, while the training time is negligible.

Table 4
Time required for classifier building (unit: seconds)

	2-gram	3-gram	4-gram
Preprocessing	1425.8	1503.6	1458.3
Selecting *N*-grams	2088.8	2157.5	2304.2
Calculating *TF* value	1897.7	6117.5	11546.7
Training	3.2	2.4	1.9
Total	5415.5	9781.0	15311.1

The time of prediction comprises of vector generation time and classification time. In vector generation time, we convert opcodes to *N*-grams, and then, the *N*-grams are used to generate a feature vector. The classification time suggests the time to classify a ransomware sample. With respect to one *N*-gram, we compute the average time of five algorithms for the same ransomware sample, and then calculate the average time on 1787 samples. In the vector generation time, the times for extracting 2-gram, 3-gram, and 4-gram are 0.8 s, 0.84 s, and 0.82 s, respectively. However, classification time is very short, and it can be ignored. Table 5 presents the prediction time for different *N*-gram lengths. It only takes 1.86 s to predict a ransomware sample in 2-gram. In 3-gram, the prediction time is 4.26 s, while it is 7.27 s in 4-gram. Owing to the complexity of data processing, long training time is inevitable, but in our approach, the time required to predict one ransomware instance is small.

Table 5
Prediction time (unit: seconds)

2-gram	3-gram	4-gram
1.86	4.26	7.27

4.3 Evaluation Criteria

To evaluate our method rigorously, we apply tenfold cross-validation in the experiments. Cross-validation, sometimes called rotation estimation, is a model validation technique, and it is mainly applied to prediction models [42, 43]. Tenfold cross-validation has been recommended in the literature [42]. We partition all samples into 10 equal-sized subsamples, and one of the subsamples is treated as a testing set while the other nine subsamples constitutes the training set [43]. This process is repeated 10 times. To rigorously evaluate our model, the evaluation criteria cover not only a single ransomware family, that is, *Precision*, *Recall*, *F1-measure*, and *Accuracy*, but also the entire model, i.e., *Accuracy*, and *Confusion Matrix*.

First, we introduce the evaluation metrics of binary classification, i.e., *Precision*, *Recall*, *F1-measure, and Accuracy*. In each evaluation, the evaluated family, *A*, is regarded as the positive class, and the remaining seven families as the negative class. *TP* (True Positive) is the number of samples that belong to family *A* and are predicted as family *A*. *TN* (True Negative) is the number of samples that do not pertain to family *A* and are not identified as family *A*. *FP* (False Positive) is the number of samples that do not belong to family *A* but are predicted as family *A*. *FN* (False Negative) is the number of samples that pertain to family *A* but are not recognized as family *A* [22].

Precision measures the performance of the classifier in predicting a positive family [44]. We calculate the *Precision* of each family separately, as follows:

$$Precision = \frac{TP}{TP+FP} \tag{4}$$

Recall represents the percentage of positive instances classified correctly [44]. We calculate the *Recall* of each family separately by using the following formula:

$$Recall = \frac{TP}{TP+FN} \tag{5}$$

F1-measure considers *Precision* and *Recall* synthetically, which evaluates the quality of a classifier. The closer its value is to 1, the better is the model quality. We calculate the *F1-measure* of each family separately, as follows:

$$F1\text{-}measuer = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

Accuracy stands for the correct rate of classification in both positive family and negative family [22]. The *Accuracy* of each family is calculated separately by using the following formula:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{7}$$

In particular, for multi-classification, *Accuracy* is the number of correct results in all families divided by the total number of all samples [10]. It can be calculated by using Eq. (8).

$$Accuracy = \frac{correctly\ classified\ instances}{total\ number\ of\ instances} \tag{8}$$

Confusion Matrix $M = [m_{x,y}]_{n\times n}$ contains *n* rows and *n* columns, which indicates the amount of deviation between the predicted results and the actual families. It is applied to multi-classification as well. In the matrix, each row represents actual families, and each column shows the type of predicted results. $m_{x,y}$ in the matrix refers to the number of samples that belong to family *x* and are recognized as family *y*.

4.4 Experimental Results for Multi-classification

In this section, the extensive experimental results obtained in this study are depicted. The *Accuracy* of different algorithms with various *N*-gram lengths and different feature dimensions is described. In addition, we show the *F1-measure* of different ransomware families with respect to various *N*-gram lengths and different feature dimensions in Random Forest. The *Confusion Matrix*, *Precision*, *Recall*, *F1-measure*, and *Accuracy* of different families in the case of the highest *Accuracy* of 91.43% in 3-gram Random Forest with feature dimensions of 123 are illustrated as well.

4.4.1 Accuracy

Fig.4-8 demonstrate the changes in *Accuracy* with increasing feature dimensions in 2-gram, 3-gram, and 4-gram in different algorithms. We increase the feature dimensions by expanding the number of top *t N*-grams, as mentioned in Section 3.2.

In Fig.4, when the feature dimensions are approximately 40, the classifier with 2-gram demonstrates an advantage over the other classifiers. However, the performance of the classifiers with 2-gram and 3-gram is similar when the number of feature dimensions is between 80 and 100. In general, the classifier with 3-gram performs better than the other classifiers. The *Accuracy* values of classifiers with 2-gram, 3-gram, and 4-gram in the Decision Tree algorithm are all in the range of 83.42% to 86.57%.

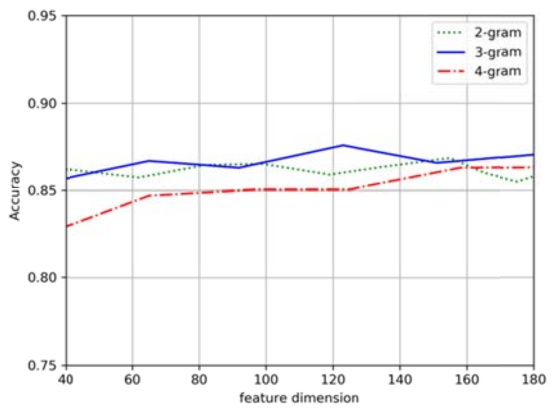

Fig.4. *Accuracy* of different feature dimensions in Decision Tree

Fig.5 illustrates similar growth trends for the *Accuracy* of the classifiers with 2-gram, 3-gram, and 4-gram with increasing feature dimensions in the K-Nearest Neighbor algorithm. From the figure, the *Accuracy* of the classifier with 2-gram is better than that of the other classifiers when the number of feature dimensions is lower than 130, but the *Accuracy* of the classifier with 3-gram is the best when the number of feature dimensions is higher than 130. The *Accuracy* values of classifiers with 2-gram, 3-gram, and 4-gram in the K-Nearest Neighbor algorithm range from 82.52% to 88.19%.

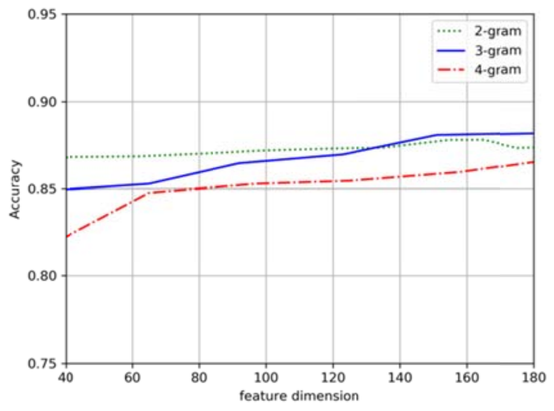

Fig.5. *Accuracy* of different feature dimensions in K-Nearest Neighbor

From Fig.6, the *Accuracy* of the classifiers with 2-gram and 3-gram is very similar when the number of feature dimensions is between 40 and 180. However, the classifier with 3-gram performs the best among all classifiers, regardless of the number of feature dimensions between 60 and 160, and the classifier with 4-gram performs the worst when the number of feature dimensions is less than 140. The *Accuracy* values of the classifiers with 2-gram, 3-gram, and 4-gram in the Random Forest algorithm range from 85.15% to 91.43%.

In Fig.7, clearly, the *Accuracy* of all classifiers in the Naive Bayes algorithm is not good, regardless of the number of feature dimensions. The worst *Accuracy* of the classifier with 2-gram is close to 56%, while that of the classifier with 4-gram is 45.89%. The best *Accuracy* of all classifiers is no more than 71%. The *Accuracy* values of classifiers with 2-gram, 3-gram, and 4-gram in the Naive Bayes algorithm ranges from 45.89% to 70.34%.

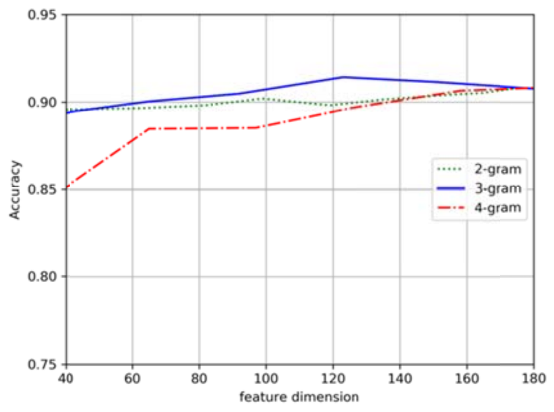

Fig.6. *Accuracy* of different feature dimensions in Random Forest

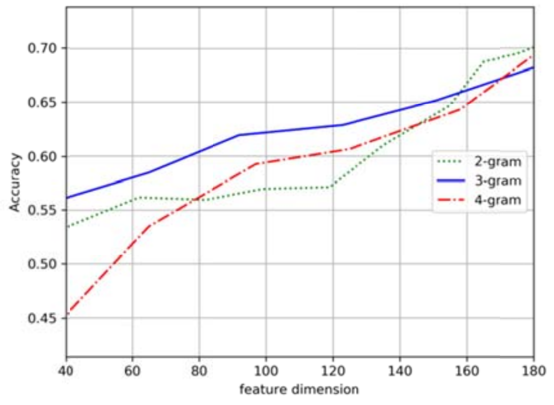

Fig.7. *Accuracy* of different feature dimensions in Naive Bayes

Fig.8 describes the *Accuracy* of the classifiers with increasing number of feature dimensions in GBDT. From Fig.8, the classifier with 2-gram performs the best when the number of feature dimensions is lower than 120, and the classifier with 4-gram exhibits the worst performance when the number of feature dimensions is no more than 80. However, the performance of all classifiers is close when the number of feature dimensions is higher than 140. The *Accuracy* values of classifiers with 2-gram, 3-gram, and 4-gram in the GBDT algorithm are in the range of 84.03% to 89.98%.

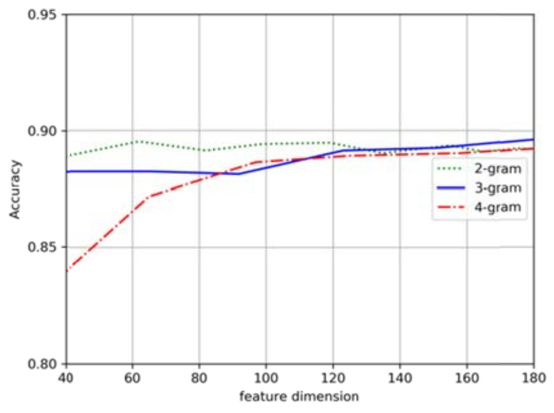

Fig.8. *Accuracy* of different feature dimensions in GBDT

Accuracy is one of the comprehensive evaluation indicators of the classifiers. In our work, we employ three *N*-grams, i.e., 2-gram, 3-gram, and 4-gram, for comparison. With respect to *Accuracy*, to reach a similarly high level of performance, an *N*-gram with diverse lengths requires different numbers of feature *N*-grams. For example, in GBDT, 2-gram can achieve better *Accuracy* with fewer than 120 feature dimensions, while 3-gram can achieve better *Accuracy* when the number of feature dimensions is close to 180, and all classifiers in GBDT achieve similar *Accuracy* when the number of feature dimensions is 140. In the K-Nearest Neighbor algorithm, the performance of 3-gram is better than that of 2-gram when the number of feature dimensions is greater than 140, and when the number of feature dimensions is lower than 140, the classifier with 2-gram performs the best.

From Figs. 4-8, the performance of the Naive Bayes algorithm is the worst. Owing to the assumption that all features are independent of each other [37], and seems false in real-world tasks, including our work [37], therefore, Naive Bayes cannot be applied in our experiments. In terms of *Accuracy*, it can be said that Random Forest is obviously better than Decision Tree, K-Nearest Neighbor, and Naive Bayes, and it is slightly better than GBDT in 3-gram. Thus, the ensemble learning algorithms are better than the simple algorithms. Thus, in the following parts, we use the results of Random Forest for illustration.

4.4.2 F1-measure

F1-measure is a synthesized evaluation indicator of *Precision* and *Recall* that can be used to evaluate model quality. Fig.9 depicts the *F1-measure* of different families with feature dimensions of 31 in 2-gram, 3-gram, and 4-gram Random Forest. In Fig.9, the performance of 2-gram is better than that of the other classifiers in six families, i.e., cryptolocker, cryptowall, cryrar, reveton, teslacrypt, and wannacry.

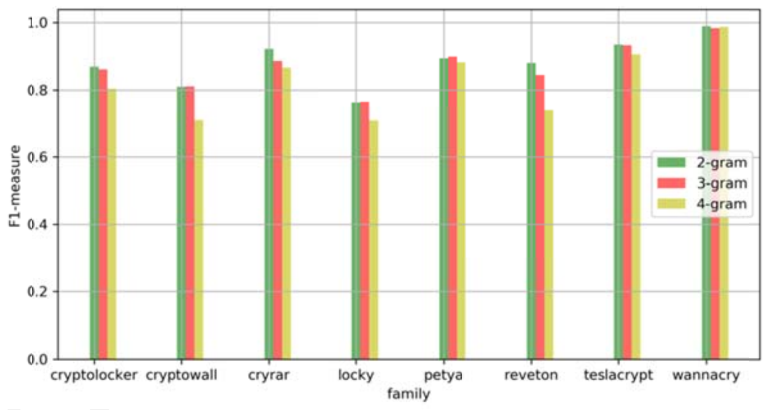

Fig.9. *F1-measure* with feature dimensions of 31 in Random Forest

Fig.10 is similar to Fig.9, but with different feature dimensions of 115. The performance of classifiers with 2-gram and 3-gram are better than that of the classifier with 4-gram. From Fig.10, the classifier with 3-gram performs better than the other classifiers on five families, i.e., cryptolocker, cryptowall, cryrar, locky, and wannacry.

Fig.11 shows *F1-measure* with feature dimensions of 186 in the case of Random Forest with 2-gram, 3-gram, and 4-gram. Different from Fig.9 and 10, in terms of *F1-measure*, the capability of 4-gram is better than that of the other classifiers on five families, i.e., cryptolocker, cryptowall, petya, teslacrypt, and wannacry.

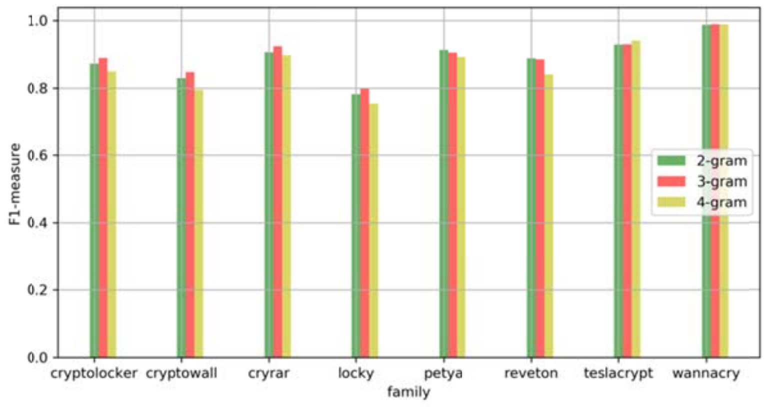

Fig.10. *F1-measure* with feature dimensions of 115 in Random Forest

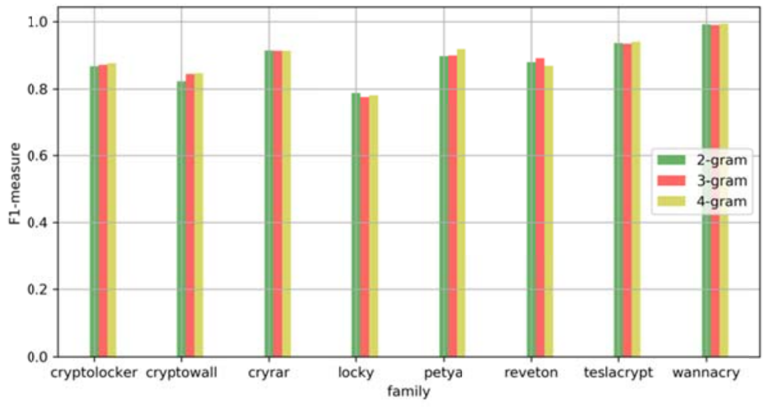

Fig.11. *F1-measure* with feature dimensions of 186 in Random Forest

A comparison of Figs .9-11 shows that classifiers with different feature dimensions or various *N*-gram lengths in Random Forest have good capability on wannacry with approximately 99% *F1-measure*, but they exhibit poor capability on locky, in which case the *F1-measure* is at around 80%. In addition, Figs. 9-11 indicate that the classifiers with *N*-grams of various lengths can perform well with different feature dimensions.

4.4.3 Confusion Matrix

Fig.4-8 demonstrate that Random Forest performs better than Decision Tree, K-Nearest Neighbor, Naive Bayes, and GBDT. Furthermore, in our experiments, the classifier with 3-gram and feature dimensions of 123 in Random Forest performs better and more stably, and *Accuracy* can reach up to 91.43%. Therefore, we present the *Confusion Matrix*, *Precision*, *Recall*, *F1-measure*, and *Accuracy* of different families in the case of 3-gram Random Forest with feature dimensions of 123.

Fig.12 shows the *Confusion Matrix* of the eight families in the experiments, namely, cryptolocker (CL), cryptowall (CW), cryrar (CR), locky (L), petya (P), reveton (R), teslacrypt (T), and wannacry (W). It is evident from Fig.12 that the classifier has the best classification capability for wannacry, in which family, only one sample is misclassified. Two of these families have very high similarity, i.e., cryptowall and reveton, as manifested in the many misclassified samples between them.

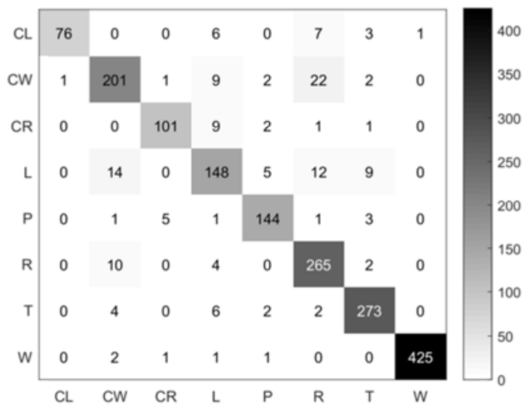

Fig.12. *Confusion Matrix* with feature dimensions of 123 in 3-gram Random Forest

Table 6 illustrates the *Precision*, *Recall*, *F1-measure*, and *Accuracy* for one family used in the classification model. Clearly, the performance in the case of wannacry is the best not only in terms of *Precision* but also in terms of *Recall*. The *F1-measure* for cryptolocker, cryrar, petya, reveton, teslacrypt, and wannacry are all above 89%, but that for locky is only 79.5%. The results demonstrate that *Accuracy* is good, and it is under 98% only for cryptowall, locky, and reveton.

Table 6
Precision, *Recall*, *F1-measure*, and *Accuracy* with feature dimensions of 123 in 3-gram Random Forest

	CL	CW	CR	L	P	R	T	W
Precision	0.987	0.866	0.935	0.804	0.923	0.854	0.931	0.998
Recall	0.817	0.844	0.886	0.787	0.929	0.943	0.951	0.988
F1-measure	0.894	0.855	0.909	0.795	0.926	0.896	0.941	0.993
Accuracy	0.989	0.962	0.989	0.957	0.987	0.966	0.981	0.997

4.5 Experimental for Binary Classification

In this section, the process and results of binary classification between trusted software and ransomware are introduced. We gather eight ransomware families as one large ransomware family. The process of this work is identical to that of our method, that is, we preprocess the trusted family and the ransomware family and then extract significant *N*-grams from both families. Thereafter, we calculate the *TF-IDF* values of each feature *N*-gram, and we set the dimensions of the feature vector to 180. In this experiment, we utilize tenfold cross-validation as well.

Binary classification of ransomware can also be termed as ransomware detection. It is important to evaluate the ability of the model to detect ransomware rather than detecting trusted software. Therefore, *Recall* is more considerable than *Precision* and *F1-measure*, which implies whether the real ransomware samples are classified to the ransomware family or the trusted family. Thus, we introduce only *Recall* and the best *Accuracy* of our work. The results of the detection experiment are excellent. Table 7 presents the *Recall* of the ransomware family with 2-gram, 3-gram, and 4-gram in different algorithms, namely, Decision Tree (DT), Random Forest (RF), K-Nearest Neighbor (KNN), Naive Bayes (NB), and Gradient Boosting Decision Tree (GBDT). Explicitly, the classifier with NB performs worst, and the performances of the classifiers with RF and KNN are slightly better than those of other classifiers. The best *Accuracy* of all classifiers in binary classification is 99.3% in the case of the classifier with 3-gram RF. Therefore, our approach is useful for ransomware detection. In classification with 3-gram RF with feature dimensions of 180, *Recall* is up to 99.8%, i.e., only 4 of 1787 ransomware samples were misclassified. Thus, the classifier based on the proposed method can correctly detect ransomware instances, which are injected obfuscation techniques.

Table 7
Recall of ransomware family with different *N*-gram lengths in different algorithms

	DT	RF	KNN	NB	GBDT
2-grams	0.992	0.997	0.998	0.976	0.994
3-grams	0.996	0.998	0.997	0.884	0.997
4-grams	0.993	0.998	0.997	0.704	0.994

5. Conclusions and Future Work

Approaches based on dynamic analysis cannot deal with ransomware that can fingerprint the environment. To overcome this drawback and fill the gaps in the extant literature on multi-classification of ransomware, herein, we propose a static approach based on text analysis to map ransomware into families. We apply five machine-learning algorithms (DT, RF, KNN, NB, and GBDT) with various opcode *N*-grams (2-gram, 3-gram, and 4-gram) and diverse feature dimensions to build classifiers. *TF-IDF* is employed to select feature *N*-grams, leading to high *Accuracy*. Thorough experiments on a real datasets manifest that Random Forest is superior to the other algorithms. The best *Accuracy* obtained is 91.43%, and the highest *F1-measure* of nearly 99% is achieved on wannacry. In addition, the results indicate that the classifiers with *N*-grams of various lengths can perform well with different feature dimensions. Moreover, our method can discriminate between ransomware and trusted software, in which case, the best *Accuracy* is 99.3%, and the best *Recall* is 99.8%.

According to the multi-classification results, the proposed approach based on *N*-grams can identify a known ransomware sample to its family, which means it can reduce the workload of analysts effectively, and it can be employed for ransomware classification. Moreover, the proposed method is superior in ransomware detection based on the results of binary classification, and it can be applied to detect ransomware. Three families, namely, cryptowall, locky, and reveton, cannot be distinguished well according to *Accuracy* for binary classification, which is a limitation of our method. In the future, we plan to further analyze the distinctions among opcodes across different ransomware families, especially, cryptowall, locky, and reveton.

Acknowledgements

This work is supported by the NSFC projects (61773229, 61771273, 61202358), the National High-tech R&D Program of China (2015AA016102), Guangdong Natural Science Foundation (2018A030313422), the RD Program of Shenzhen (JCYJ20160531174259309, JCYJ20170307153032483, JCYJ20160331184440545, JCYJ20170307153157440) and the Interdisciplinary Research Project of Graduate School at Shenzhen of Tsinghua University (JC2017005).

References

[1] Internet Security Threat Report https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en .pdf

[2] H. HaddadPajouh, A. Dehghantanha, R. Khayami, K.K.R. Choo, Intelligent OS X malware threat detection with code inspection, Journal of Computer Virology and Hacking Techniques [In press] https://doi.org/10.1007/s11416-017-0307-5

[3] H. HaddadPajouh, A. Dehghantanha, R. Khayami, K.K.R. Choo, A deep recurrent neural network based approach for Internet of things malware threat hunting, Future Generation Computer Systems [In press] https://doi.org/10.1016/j.future.2018.03.007

[4] N. Milosevic, A. Dehghantanha, K.K.R. Choo, Machine learning aided Android malware classification, Computers & Electrical Engineering 61 (2017) 266-274.

[5] A. Gazet, Comparative analysis of various ransomware virii, Journal in Computer Virology 6 (1) (2010) 77-90.

[6] B.A.S. Al-rimy, M.A. Maarof, S.Z.M. Shaid, Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions, computers & security 74 (2018) 144-166.

[7] J.A. Gómez-Hernández, L. Álvarez-González, P. García-Teodoro, R-Locker: Thwarting ransomware action through a honeyfile-based approach, computers & security 73 (2018) 389-398.

[8] R. Vinayakumar, KP. Soman, K.K.S. Velany, S. Ganorkar, Evaluating shallow and deep networks for ransomware detection and classification, in: Proceedings of 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 2017, pp. 259-265.

[9] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, UNVEIL: A large-scale, automated approach to detecting ransomware, in: Proceedings of the 25th USENIX Conference on Security Symposium (USENIX Security), 2016, pp. 757-772.

[10] A. Pekta, T. Acarman, Classification of malware families based on runtime behaviors, Journal of Information Security and Applications 37 (2017) 91-100.

[11] I. Kwon, E.G. Im, Extracting the representative API call patterns of malware families using recurrent neural network, in: Proceedings of the International Conference on Research in Adaptive and Convergent Systems, ACM, 2017, pp. 202-207.

[12] A. Mohaisen, A.G. West, A. Mankin, O. Alrawi, Chatter: Classifying malware families using system event ordering, in: Proceedings of 2014 IEEE Conference on Communications and Network Security (CNS), IEEE, 2014, pp. 283-291.

[13] S.S. Hansen, T.M.T. Larsen, M. Stevanovic, J.M. Pedersen, An approach for detection and family classification of malware based on behavioral analysis, in: Proceedings of 2016 International Conference on Computing, Networking and Communications (ICNC), IEEE, 2016, pp. 1-5.

[14] C. Kolbitsch, E. Kirda, C. Kruegel, The power of procrastination: detection and mitigation of execution-stalling malicious code, in: Proceedings of the 18th ACM conference on Computer and communications security, ACM, 2011, pp. 285-296.

[15] Z. Ghezelbigloo, M. VafaeiJahan, Role-opcode vs. opcode: The new method in computer malware detection, in: Proceedings of 2014 International Congress on Technology, Communication and Knowledge (ICTCK), IEEE, 2014, pp. 1-6.

[16] C. Liangboonprakong, O. Sornil, Classification of malware families based on n-grams sequential pattern features, in: Proceedings of 2013 8th IEEE Conference on Industrial Electronics and Applications (ICIEA), IEEE, 2013, pp. 777-782.

[17] M. Hassen, M.M. Carvalho, and P.K. Chan, Malware classification using static analysis based features, in: Proceedings of 2017 IEEE Symposium on Computational Intelligence (SSCI), IEEE, 2017, pp. 1-7.

[18] N.E. Lakhdari, A. Boukhtouta, M. Debbabi, Inferring malware family through application protocol sequences signature, in: Proceedings of 2014 6th International Conference on New Technologies, Mobility and Security (NTMS), IEEE, 2014, pp. 1-5.

[19] Y. Zhang, C. Rong, Q. Huang, Y. Wu, Z. Yang, J. Jiang, Based on Multi-Features and Clustering Ensemble Method for Automatic Malware Categorization, in: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, 2017, pp. 73-82.

[20] G. Pitolli, L. Aniello, G. Laurenza, L. Querzoni, R. Baldoni, Malware family identification with BIRCH clustering, in: Proceedings of 2017 International Carnahan Conference on Security Technology (ICCST), IEEE, 2017, pp. 1-6.

[21] J.S. Luo, D.C. Lo, Binary malware image classification using machine learning with local binary pattern, in: Proceedings of 2017 IEEE International Conference on Big Data (Big Data), IEEE, 2017, pp. 4664-4667.

[22] J. Fu, J. Xue, Y. Wang, Z. Liu, C. Shan, Malware Visualization for Fine-Grained Classification, IEEE Access (99) 2018 1-13.

[23] S.M. Kumar, M.R. Kumar, Cryptoviral Extortion: A virus based approach, International Journal of Computer Trends and Technology (IJCTT) 4 (5) (2013) 1149-1153.

[24] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, E. Kirda, Cutting the gordian knot: A look under the hood of ransomware attacks, in: Proceedings of International Conference on Detection of Intrusions and Malware & Vulnerability Assessment, 2015, pp. 3-24.

[25] C. Moore, Detecting ransomware with honeypot techniques, in: Proceedings of 2016 Cybersecurity and Cyberforensics Conference (CCC), IEEE, 2016, pp. 77-81.

[26] K. Cabaj, P. Gawkowski, K. Grochowski, A. Kosik, Developing malware evaluation infrastructure, in: Proceedings of 2016 Federated Conference on Computer Science and Information Systems, IEEE, 2016, pp. 981-989.

[27] J. K. Lee, S.Y. Moon, J.H. Park, CloudRPS: a cloud analysis based enhanced ransomware prevention system, The Journal of Supercomputing 73 (7) (2017) 3065-3084.

[28] MM. Ahmadian, HR. Shahriari, SM. Ghaffarian, Connection-monitor & connection-breaker: a novel approach for prevention and detection of high survivable Ransomware, in: Proceedings of 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), IEEE, 2015, pp. 79-84.

[29] M.M. Ahmadian, H.R. Shahriari, 2entFOX: A Framework for High Survivable Ransomwares Detection, in: Proceedings of 2016 13th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), IEEE, 2016, pp. 79-84.

[30] A. Azmoodeh, A. Dehghantanha, M. Conti, K. R. Choo, Detecting crypto-ransomware in IoT networks based on energy consumption footprint, Journal of Ambient Intelligence and Humanized Computing (2017) 1-12.

[31] S. Banerjee, T. Pedersen, The design, implementation, and use of the ngram statistics package, in: Proceedings of International Conference on Intelligent Text Processing and Computational Linguistics, 2003, pp. 370-381.

[32] Z. Wen, Y. Taketoshi, T. Xijin, A comparative study of TF*IDF, LSI and multi-words for text classification, Expert Systems with Applications 38 (3) (2011) 2758-2765.

[33] E. Ugo, S. Sabrina, M. Fernando, C. Giuseppe, Approximate TF–IDF based on topic extraction from massive message stream using the GPU, Information Sciences 292 (2015) 143-161.

[34] M. Philippe, N. Vincent, M. Christophe, L. Alex, Survey on deep learning for radiotherapy, Computers in Biology and Medicine 98 (2018) 126-146.

[35] J.Z. Kolter, M.A. Maloof, Learning to detect malicious executables in the wild, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 470-478.

[36] Y.X. Ding, W. Dai, Y.B. Zhang, Malicious code detection using opcode running tree representation, in: Proceedings of 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), IEEE, 2014, pp. 616-621.

[37] M. Andrew, N. Kamal, A Comparison of Event Models for Naive Bayes Text Classification, in: Proceedings of Fifteenth National Conference on Artificial Intelligence (AAAI), 1998.

[38] S. Jeany, J. Ilchae, P. Kayoung, H. Bohyung, Tracking-by-segmentation with online gradient boosting decision tree, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 3056-3064.

[39] P. Harrington, Machine Learning in Action, Manning Publications Co. (2012).

[40] Hex-Rays, IDA: About https://www.hex-rays.com/products/ida/

[41] P. Fabian, V. Gaël, G. Alexandre, M. Vincent, T. Bertrand, G. Olivier, B. Mathieu, P. Peter, W. Ron, D. Vincent, V. Jake, P. Alexandre, C. David, B. Matthieu, P. Matthieu, D. Édouard, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12 (2011) 2825-2830.

[42] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI), 1995, pp. 1137-1143.

[43] Wikipedia, Cross-validation (statistics) https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation

[44] H. Divandari, B. Pechaz, M.V. Jahan, Malware detection using Markov Blanket based on opcode sequences, in: Proceedings of 2015 International Congress on Technology, Communication and Knowledge (ICTCK), IEEE, 2015, pp. 564-569.

Hanqi Zhang

Hanqi Zhang is a student in Central China Normal University, Wuhan, China. His research interests focus on communication systems and information security.

Xi Xiao

Xi Xiao is an associate professor in Graduate School At Shenzhen, Tsinghua University. He got his Ph.D. degree in 2011 in State Key Laboratory of Information Security, Graduate University of Chinese Academy of Sciences. His research interests focus on information security and the computer network.

Francesco Mercaldo

Francesco Mercaldo obtained his PhD in 2015 with a dissertation on malware analysis using machine learning techniques. The core of his research is finding methods and methodologies to detect new threats applying the empirical methods of software engineering as well as studying the current mechanisms to ensure security and private data in order to highlight the vulnerability. Currently he works as post-doctoral researcher at the Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy.

Shiguang Ni

Shiguang Ni received the M.S. degree in applied psychology from Beijing Normal University, Beijing, China, in 2008 and the Ph.D. degree from Department of Psychology, Tsinghua University, Beijing, China, in 2013, respectively. He is currently an Associate Professor in the Division of Social Science and Management, Graduate School at Shenzhen, Tsinghua University, Shenzhen, China. His interdisciplinary research interests include behavior big data, data mining and data preprocessing on describing and predicting personality and public health.

Fabio Martinelli

Fabio Martinelli is a senior researcher at the Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy, where he is the scientific coordinator of the security group. His main research interests involve security and privacy in distributed and mobile systems and foundations of security and trust. He serves as PC-chair/organizer in several international conferences/workshops. He is the co-initiator of the International Workshop series on Formal Aspects in Security and Trust (FAST). He is serving as scientific co-director of the international research school on Foundations of Security Analysis and Design (FOSAD) since 2004 edition. He chairs the WG on security and trust management (STM) of the European Research Consortium in Informatics and Mathematics (ERCIM). He usually manages R&D projects on information and communication security and he is involved in several FP6/7 EU projects.

Arun Kumar Sangaiah

Arun Kumar Sangaiah had received his Doctor of Philosophy (PhD) degree in Computer Science and Engineering from the VIT University, Vellore, India. He is presently working as an Associate Professor in School of Computer Science and Engineering, VIT University, India. His area of interest includes software engineering, computational intelligence, wireless networks, bio-informatics, and embedded systems.

Hanqi Zhang

Xi Xiao

Francesco Mercaldo

Shiguang Ni

Fabio Martinelli

Arun Kumar Sangaiah

Highlights

As far as we know, we are the first to classify ransomware using a static method.

TF-IDF can select feature *N*-grams which distinguish each family as much as possible.

We conduct comprehensive experiments on real-world datasets.

The effect of *N*-grams of various lengths with diverse dimensions is investigated.