Ensemble Model Ransomware Classification: A Static Analysis-based Approach

Shanoop Johnson, R. Gowtham, and Anand R. Nair

Abstract The growth of malware attacks has been phenomenal in the recent past. The COVID-19 pandemic has contributed to an increase in the dependence of a larger than usual workforce on digital technology. This has forced the anti-malware communities to build better software to mitigate malware attacks by detecting it before they wreak havoc. The key part of protecting a system from a malware attack is to identify whether a given file/software is malicious or not. Ransomware attacks are time-sensitive as they must be stopped before the attack manifests as the damage will be irreversible once the attack reaches a certain stage. Dynamic analysis employs a great many methods to decipher the way ransomware files behave when given a free rein. But, there still exists a risk of exposing the system to malicious code while doing that. Ransomware that can sense the analysis environment will most certainly elude the methods used in dynamic analysis. We propose a static analysis method along with machine learning for classifying the ransomware using opcodes extracted by disassemblers. By selecting the most appropriate feature vectors through the tf-idf feature selection method and tuning the parameters that better represent each class, we can increase the efficiency of the ransomware classification model. The ensemble learning-based model implemented on top of N-gram sequence of static opcode data was found to improve the performance significantly in comparison to RF, SVN, LR, and GBDT models when tested against a dataset consisting of live encrypting ransomware samples that had evasive technique to dodge dynamic malware analysis.

Keywords Tf-idf · N-gram · Random forest · Opcode · SVM · Ransomware · Voting classifier

S. Johnson (✉) · R. Gowtham · A. R. Nair
Department of Computer Science and Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore, India
e-mail: cb.en.p2cse19021@cb.students.amrita.edu

R. Gowtham
e-mail: r_gowtham@cb.amrita.edu

A. R. Nair
e-mail: r_anand@cb.amrita.edu

S. Smys et al. (eds.), *Inventive Computation and Information Technologies*, Lecture Notes in Networks and Systems 336, https://doi.org/10.1007/978-981-16-6723-7_12

1 Introduction

Around three hundred and fifty-seven million new malware samples were revealed by early 2017, according to the Internet Security Threat Report conducted by Symantec Broadcom [1]. Malicious software (malware) is capable of eroding or stealing data and compromising computer systems. The earliest form of malware was observed to have appeared in the beginning of the 1970s. The Creeper Worm in the 70s was a program that replicates itself and attaches itself in the remote system and locks it down [2]. The tremendous leap in technological advancements has contributed to massive growth in malware risks with respect to the aspects such as magnitude, numbers, type of operation. The wide-scale development in interconnected devices like smartphones, systems, and IoT devices allows smart and sophisticated malware to be developed. In 2013, more than 250,000 computers were infected with the CryptoLocker ransomware, which then dramatically increased to 500,000 devices during 2014 [3]. In 2017, with an average attack growth rate of 350%, global organizations financial losses due to this malware attack surpassed five billion dollars [4]. By the year 2021, it is expected to reach 20 billion dollars [5]. Global ransomware security agencies noted that the existing solutions are not successful in protecting against attacks by ransomware and suggested the need for active solutions to be developed.

Ransomware analysis attempts to give information on the functionality, intent, and actions of a specific type of software. Two kinds of malware analysis are available: static analysis and dynamic analysis. Static analysis implies analyzing the source code and related artifacts of an executable or malignant file without execution. Dynamic analysis, on the contrary, requires analyzing the behavioral aspects of an executable in action by running it. Both approaches have their own advantages and pitfalls which we need to identify. Static analysis is quicker, but the method could prove to be ineffective if ransomware is effectively hidden using code obfuscation techniques. Machine learning approaches to detect such variants are still evolving. However, conventional identification and analysis of ransomware are hardly able to keep pace with the latest variants of evolved malware and their attacks.

Compared to all other malware attacks, ransomware attacks are time-sensitive. We need to identify and establish the files behavior within seconds in order to prevent havoc to the system. Unlike other attacks, if we allow the ransomware attack to manifest and then monitor and remove the ransomware, the effect of the attack will be irreversible as is the intent of the attack. There are several dynamic analysis models available at present, and Windows Defender service identifies a whole part of it within its databases extensively. However, fresh attacks are seemingly hard to detect as it might not correspond to any existing signature. For this, we need to establish the files intent based on the opcode it executes in the system. We believe if we can train the model with a satisfactorily large enough number of samples, the effect of code obfuscation techniques and random opcode injection into the executable files employed by the malware in order to confuse the model can sufficiently be overcome.

We have identified that static analysis of opcodes extracted from ransomware samples using disassemblers can perform well in classification. Compared to the tedious dynamic analysis setup, our method is easily operable. We use open-source disassemblers to get the disassembled opcode sequences corresponding to the ransomware executable. This study proposes a better ransomware detection model established on static analysis of opcode sequences, which offers more efficient countermeasures against ransomware attacks by detecting them early. The paper proposes a binary classification of benign and malignant samples.

2 Related Work

Since ransomware is a particular genre of malware, to give more insight to the scope of our research, several pieces of research on malware are also cited in this section. First, we present the most recent research initiatives that focus on time-sorted application programming interface (API) call sequences and opcodes on the classification of malware families, which are comparable to the classification methodology that we propose.

In the early stages of malware detection and mitigation, signature-dependent detection methods were suggested. At that point in time, it was assumed that automatic generation of signatures of malware as much as possible was necessary, and that it would increase the pattern-matching speed and performance [6, 7].

With respect to dependence on time-sorted API call sequences, in order to accomplish the malicious purpose intended by the malware, a particular sequence of API calls, in that very order must be executed by attackers, and a study of time-sorted API call sequences will help us better understand the intent of malicious codes [6]. A. Kharrazand and S. Arshad suggested methods based on sequences of API such as dynamic analysis in a sandbox setting, and Kwon et al. [8] extracted time-sorted API call sequences as characteristics and then using RNN to classify and so on.

Hansen et al. [9] considered time-sorted API call sequences as features. The average weighted F1-measure came out around eighty percent. Pekta [8] took N-grams as features from time-sorted API call sequences and measured term frequency-inverse document frequency (tf-idf). The highest accuracy obtained was 92.5%, suggesting that in terms of selecting meaningful sequences, N-grams have better ability to pinpoint on malicious behavior. Some malware could, however, mostly fingerprint the dynamic analysis environment [8] and enforce steps, such as code stalling [10] to avoid detection by an AV engine or human, making research more strenuous. So, dynamic analysis does not detect malware/ransomware that can identify the system or the sandboxing environment in which its being tested, which is the drawback of dynamic malware analysis. Gowtham Ramesh and Anjali Menen proposed a finite-state machine model [11] to dynamically detect ransomware samples by employing a set of listeners and decision-making modules which identify changes in the system within the specific set of use cases defined in the underlying system.

Samples of ransomwares belonging to seven families and samples of applications (apps) associated to categories within the benign class were used in their technique to perform basic classification [12]. Not only did the system predicts whether the sample belonged to the safe class or the malignant class, but also grouped the sample into the corresponding family. During dynamic analysis, the authors considered almost one hundred and thirty-one time-sorted API call sequences obtained by executing the samples in a sandbox and leveraged deep learning to build a model and tested it. The experiment, thus, conducted outperformed almost all the existing models based on machine learning algorithm classifiers. The precision was almost 98% in multi-layer perceptron (MLP); the strongest true positivity rate (TPR) is just 88.9; and 83.3% is observed for several families of ransomware like CryptoWall. Because of the limitation of dynamic analysis mentioned above, if ransomware is able to identify the sandbox used, researchers will not be able to extract meaningful time-sorted API sequences.

Classifying malwares based on machine learning requires extracting features from malware samples first. For that, we have static and dynamic analysis methods available. Static analysis does not require the sample to be executed. It basically extracts opcodes, scans the format of PE headers, and disassembles the sample. Disassemblers like PEiD [6], IDA Pro [7] have their own databases containing file signatures to identify packers and headers. VirusTotal [8] can detect known malware samples using 43 antivirus engines. In this paper, we disassemble the samples using IDAPro for the purpose of static analysis. In dynamic analysis, the sample which we execute has complete access to the system resources. But, the environment will be a controlled one, probably a sandbox. In this, the software can modify registry keys also. At the termination of execution, the sandbox reverts to its original state, and the environment logs the behavior of the software.

In 2017, Chumachenko [13] proposed malware classification based on dynamic analysis using a scoring system in Cuckoo sandbox. They identified features such as registry keys, mutexes, processes, IP addresses, and API calls and formed the feature vector to perform machine learning algorithms. But, this method was time-consuming and had a limited dataset. In 2015, Berlin and Saxe [14] used a histogram of entropy values of opcode bytes as feature vector for a neural network-based classification. Vu Thanh Nguyen [15] used an artificial immune network for classification. But, both these methods leveraged a highly imbalanced dataset. Both comprised of over 80% malicious samples. Even with synthetic oversampling, the model will be highly biased to one class.

3 System Design

Gathering a balanced dataset is the crux of any machine learning-based application. Cleaning and pruning the data to be precise for the intended application can aid in creating a very efficient model. We have to create a balanced dataset containing the assembly language code (.asm files) corresponding to each sample belonging to the benign and malignant classes. We use IDA Pro to disassemble the executable (.exe) files and convert them into .asm files. We have a command line feature available with IDA Pro to perform this activity conveniently in batches so that we can obtain a dataset corresponding to a large set of samples.

3.1 Dataset

For the purpose of this study, we collected ransomware samples from virusshare.com repository which had a collection of over thirty-five thousand samples along with their hash values. Around eight thousand samples were selected from this based on size of the files. This is a relatively good number of samples used for study when compared against other studies made in machine learning classification area which is going to be discussed in Section IV. These samples were from various cryptographic ransomware families, including TeslaCrypt, WannaCry, Petya, CryptoWall, and Cerber. Table 1 gives the complete information about these families and samples such as encryption technique, when the malware was released, and the sample count from each family that is presenting the dataset.

To make a balanced dataset for binary classification, almost eight thousand benign .exe files were taken from the local systems available at the various computer laboratories and staff departments at the University. We included several types of benign applications like basic file manipulation executables, DLLs, common tools, video players, browsers, drivers Both benign and malignant files were disassembled using IDA Pro into .asm files using command line interface.

Table 1 Ransomware families used

Family	Year	Techniques	Target	#Samples
CryptoLocker	2013	RSA	User files	741
CryptoWall	2014	RSA 2048 bit	User files	706
Cryrar	2012	RAR-sfx	User files	583
Locky	2016	RSA 2048 bit	User files	567
Petya	2016	AES-128MBR	User files	593
Reventon	2012	N/A	User files	617
TeslaCrypt	2015	ECC	Games and multimedia files	398
WannaCry	2017	RSA 2048 bit	User files	436
Cerber	2016	RSA 2048 bit	User files	263

3.2 Implementation

Figure 1 shows the complete system design. After creating a balanced dataset from benign and malignant files, we generate N-grams of opcode sequences from disassembled files. Then, N-gram sequences obtained are ranked, and top feature vectors are chosen. We can also set a threshold to limit the number of feature vectors to be chosen. In the next step, we compute the tf-idf for the chosen feature vectors. That is, each N-gram sequence is given a probability factor which represents how well they represent their classes.

3.3 Preprocessing

The collected .exe files need to be disassembled using any popular disassemblers. We used IDA Pro and a snippet of IDA pro as shown in Fig. 2. In order to batch process, we created a script file which runs sequentially and obtained disassembled .asm files. The method proposed here takes advantage of static analysis; that is, the samples are not needed to run on physical machines.

From the .asm files, we need to extract the opcode sequences in time-sorted order so that we can confirm which sequences will cause harm. We then record these time-sorted opcode sequences of all the benign and malignant files into a single text file along with the corresponding class label.

From each ransomware sample .asm file, various continuous sequences of opcodes of varying length (N grams of opcodes) are extracted from the original opcode sequence. In malware, any one opcode execution may not harm the device, but it may be detrimental to a machine if a sequence consisting of more than one opcode in a particular order is executed. In addition, a short, contiguous series of N-opcodes is called an N-gram. Hence, the N-gram can grab a few meaningful opcode sequences.

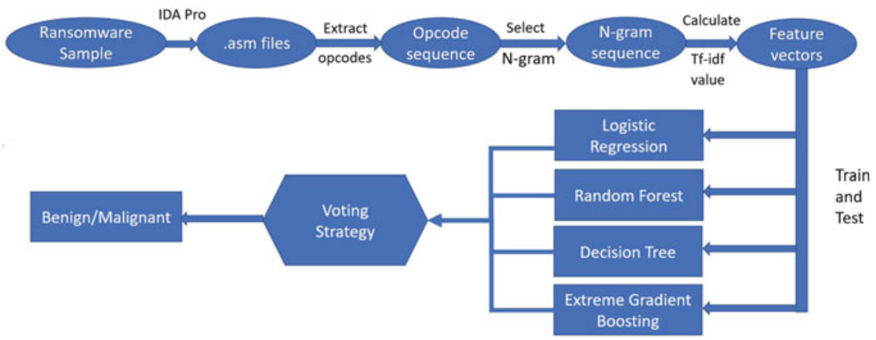
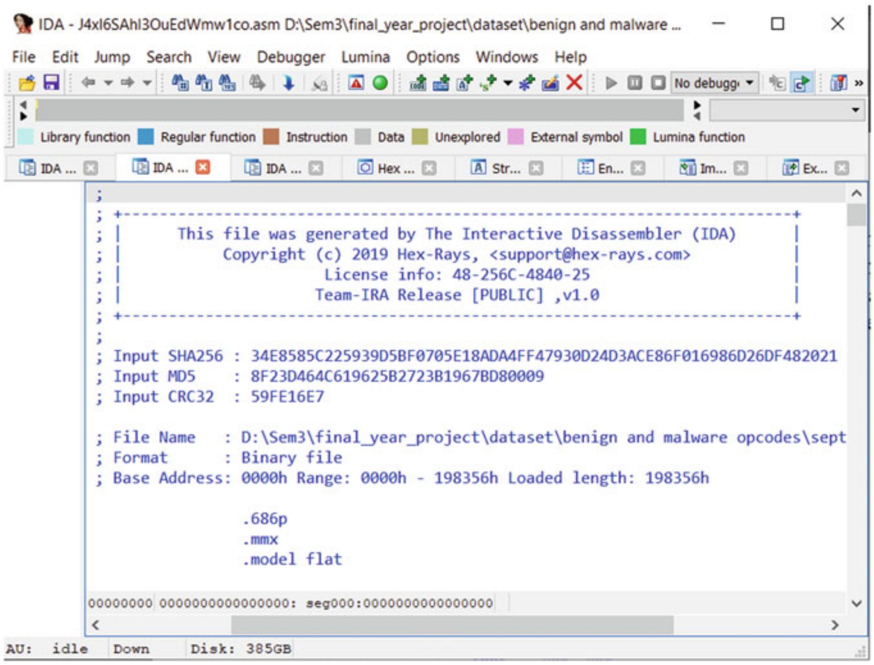


Fig. 1 Model diagram


IDA - J4xl6SAhl3OuEdWmw1co.asm D:\Sem3\final_year_project\dataset\benign and malware ...
File Edit Jump Search View Debugger Lumina Options Windows Help
No debugg
Library function Regular function Instruction Data Unexplored External symbol Lumina function
IDA ... IDA ... IDA ... Hex ... Str... En... Im... Ex...

```
;
; +-------------------------------------------------------------------------+
; |      This file was generated by The Interactive Disassembler (IDA)      |
; |           Copyright (c) 2019 Hex-Rays, <support@hex-rays.com>           |
; |                      License info: 48-256C-4840-25                      |
; |                     Team-IRA Release [PUBLIC] ,v1.0                     |
; +-------------------------------------------------------------------------+
;
; Input SHA256 : 34E8585C225939D5BF0705E18ADA4FF47930D24D3ACE86F016986D26DF482021
; Input MD5    : 8F23D464C619625B2723B1967BD80009
; Input CRC32  : 59FE16E7

; File Name   : D:\Sem3\final_year_project\dataset\benign and malware opcodes\sept
; Format      : Binary file
; Base Address: 0000h Range: 0000h - 198356h Loaded length: 198356h

                .686p
                .mmx
                .model flat

00000000 0000000000000000: seg000:0000000000000000
AU:  idle   Down    Disk: 385GB
```

Fig. 2 Disassembled source code of a ransomware sample

The N-grams have the power to predict certain phenomena occurring if we obtain enough samples to train. It is also prudent to have an equal mix of benign and malignant samples to reduce bias toward any one class.

3.4 Computing Tf-idf Value

Out of all the ransomware families used in the study and the benign files, we obtained a significant number of N-gram sequences. Now, we need an efficient method to calculate the importance of each feature vector and rank them. Tf-idf uses a common approach that utilizes principle of language modeling so as to classify essential words. The heuristic intuition of tf-idf is that a word that is appearing in several corpuses is probably not a good indicator of a specific corpus and may not be allotted more points than the sequences that are found to turn up in fewer corpuses. By using Eqs. (1), (2), and (3), we determine the tf-idf for every N-gram.

$$\mathrm{TF}_{t,d} = f(t,d)k_f(t,d) \tag{1}$$

Here, $\mathrm{TF}_{(t,d)}$ represents the total number of times the N-gram t occurs in dth N-gram sequence. $f_{(t,d)}$ shows the frequency of N-gram t occurring in N-gram sequence d, and $\sum kf_{(t,d)}$ denotes the total of N-grams in d. Then,

$$\mathrm{IDF}(t) = \log 2|N||\{d\colon t \in d|d \in N\} \tag{2}$$

Here, $\mathrm{IDF}_{(t)}$ specifies if N-gram t is that uncommon in all sequences of N-gram, whereas N is the collection of all sequences of N-grams. $|d\colon t \in d|d \in N|$ represents the count of sequences of N-grams that actually has N-gram t.

$$\text{TF-IDF}_{(t,Df)} = \mathrm{TF}_{(t,Df)} \times \mathrm{IDF}_{(i)} \tag{3}$$

In above Eq. (3), we use every sequence of N-gram within malignant files f so as to form a lengthy N-gram sequence Df . $\mathrm{TF} - \mathrm{IDF}_{(i,\mathrm{Df})}$ indicates the value of tf-idf in N-gram t of the lengthy N-gram sequence Df.

In a ransomware family, tf-idf measures the value of an N-gram, which increases as the count of a N-gram increases in a class. In addition, it is negatively proportional to how many times the exact N-gram is found to crop up in other ransomware families. To a certain degree possible, tf-idf will differentiate each class from other classes since the N-gram with a larger tf-idf score means that the number times it exists in one class is more and the frequency of occurrences across the latter classes is very low.

In each class, we arrange the N-grams as N-grams function in the descending order of their respective tf-idf score and pick only the most potent N-grams from both classes. To get the feature N-grams for one classifier, we combine $b * t$ feature vectors, i.e., t N-grams from each of the b classes. Since there are many similar N-grams features derived from different classes, the repeating N-grams features are eliminated.

By using Eq. 1, compute the term frequency (TF) score of each N-gram feature vector. In an N-gram, the values of the N-gram features are represented as one single vector, which is the corresponding ransomware feature vector for the next step. In the N-gram sequence, if the number of features present in the N-gram is 10, and the number of occurrences of a particular N-gram, say (del, mov, sub) is 2; then, using Eq. 1, we compute the TF score of (del, mov, sub). In order to form one vector, we calculate the TF score of each feature vector like this. We acquire all feature vectors by performing the same procedure for all N-grams in order to obtain the feature vector. The feature dimension is the count of all N-grams chosen. Table 2 shows the sample feature vector:

3.5 *Training, Validation, and Testing*

In the previous steps, for each class, we obtained the most indicative features. These have the class label information as well that needs to be fed into the ML algo-

Table 2 Sample 3-g sequence

Sample ID	1	2	3	4
Class label	1	1	0	1
add lock add	0.003764	0	0.00244	0.00276
add cmp mov	0.00084	0.01534	0.00140	0.00739
add mov mov	0.237791	0	0.03269	0.02501

rithm. During the training phase, we have trained the model using four machine learning models, namely SVM, random forest (RF), logistic regression (LR), and gradient boosting decision tree (GBDT) algorithms. We were able to conclude that the random forest algorithm gave much better classification accuracy when trained and tested with various feature lengths and N-gram sizes. We divided the dataset into 80 to 20 ratios for training and validation. We employed a randomized search on hyperparameters of all classification algorithms employed to find the optimum hyperparameter values and then use those for final model.

We obtained validation accuracy as high as 99% with the random forest classifier. Then, we use an ensemble learning model called voting classifier which predicts the class based on their highest probability of chosen class. It simply aggregates the findings of each classifier passed into voting classifier and predicts the output class based on the highest majority of voting. At last, the trained classifier model can, with a high degree of accuracy, predict the labels of unknown ransomware samples. By measuring the classification accuracy, false positive rate (FPR), and false negative rate (FNR), the performance of the model can be tested and compared. We tested the model using various other active ransomware samples provided by SERB and benign files, and it performed above par, yielding a testing accuracy of 94%.

4 Results and Discussions

Ransomware classification methods that rely upon dynamic analysis cannot fathom the samples that can detect the sandboxing environment in which they are analyzed. To cope with this limitation and to produce a model with a better binary classification accuracy, here, we made use of a static analysis-based approach based on opcode execution to classify ransomwares. We employ ensemble learning using voting classifier on top of four major machine learning algorithms (RF, SVM, LR, and GBDT) to model classifiers. We also experimented with varying lengths of N-gram sequences (2, 3, and 4-g) and different threshold values to limit the number of features from each class. Tf-idf is utilized to pick the most relevant features, which led to a very good performance by the classification model. The model was used on diverse combinations of the dataset, and it proves that random forest (RF) has a better performance compared to other algorithms followed by support vector machine (SVM). We also

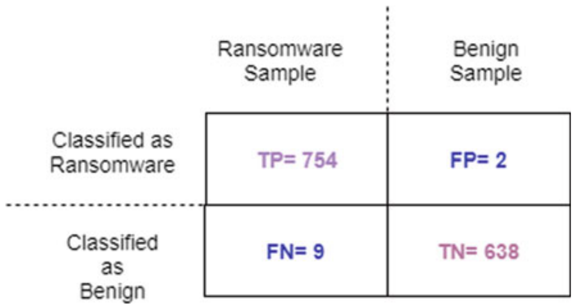


Fig. 3 Confusion matrix voting classifier

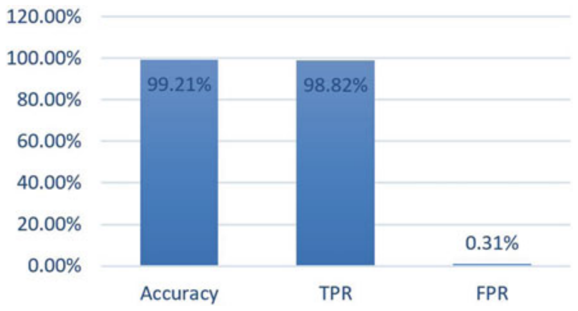


Fig. 4 Assessment of the experimental results

used randomized search cross-validation method to find the optimum hyperparameter values for each algorithm used.

The highest validation accuracy obtained using ensemble model voting classifier was noted to be 99.21% as shown in Fig. 3. The experimental results clearly show that this model can detect ransomwares with a false positive rate of just 0.31% as shown in Fig. 4.

We performed an extensive testing using different *N*-gram and feature lengths on all the chosen classification algorithms. Owing to the excellent feature selection technique, we got splendid results with an average of 98% classification accuracy in all algorithms combined. Results are shown in Table 3.

Table 3 Training accuracy

Classifiers	2-g	3-g	4-g
Random forest (RF)	99.07	98.64	98.98
Support vector machine (SVM)	97.22	96.86	97.50
Extreme gradient boosting (GBDT)	98.64	98.43	98.21
Voting classifier	98.76	99.21	98.86

Table 4 Testing accuracy

Classifiers	2-g	3-g	4-g
Random forest (RF)	77.28	90.37	84.93
Support vector machine (SVM)	69.13	86.17	77.28
Extreme gradient boosting (GBDT)	61.97	89.87	80.24
Voting classifier	93.33	91.11	85.43

Table 5 Time required in seconds

Voting classifier	2-g	3-g	4-g
Calculating tf-idf value	76.54	79.04	83.53
Model fitting	82.14	120.23	168.37
Model fitting	1.96	2.16	3.08

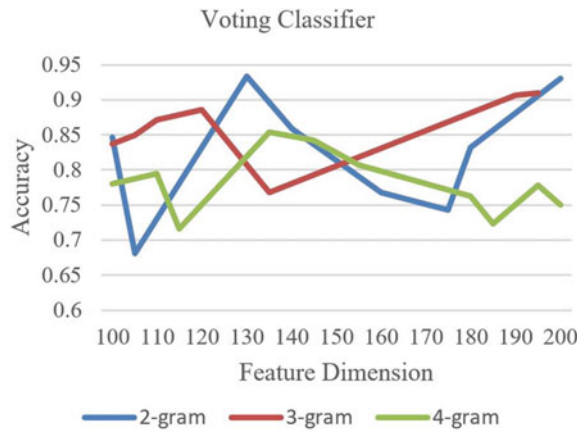
After training the model, we tested it using different sets containing a diverse combination of ransomware samples and benign files. We were able to obtain classification accuracy of 90% using the random forest (RF) algorithm. The ransomware samples used for testing were observed to contain different packers and had plenty of obfuscation methods employed in them which were ascertained by testing the samples using dynamic analysis methods. These results were further enhanced by employing the ensemble voting method after optimizing the hyperparameter values. We obtained model accuracy as high as 93.33%. Results are shown in Table 4.

Training this machine learning model using any one classifier algorithm takes almost negligible time. The time taken for each step, in seconds, is presented in Table 5. Since the disassembling step is common for all variations, it need not be considered.

Loading the dataset created using disassembled opcodes into the Python notebook takes an average 12 s. Vectorization and selection of feature vectors were observed to be the most time-consuming step during the implementation phase. It was noted to have consumed over 3 min for each variation. Time required for model fitting or training using a voting classifier increases drastically with the increase in the number of N-grams. Once we fit the model, it takes negligible time for making the prediction. We trained and tested the model with various feature dimensions for all three chosen N-gram sequences to select the optimum number of features. Then, further tests were conducted based on the chosen feature dimension and N-gram size. The results are shown in Fig. 5.

We can see from the results in above figures that in voting classifier, both 2 and 3 -g performances are better in classifying samples.

Fig. 5 Classification accuracy of voting classifier in varying feature dimensions



We have also compared our work with existing dynamic analysis-based methods as listed in Table 6. The results tabulated are from the respective published papers. The test samples that we have used in this study are active and live.

From the table, it is clear that our method proves to be more efficient than existing dynamic analysis models. This exempts us from designing complex sandboxes to analyze ransomware behavior. This method proved to be less time-consuming at the prediction phase. It will save countless working hours for any analyst who tries to dissect and figure out the heuristic signatures of a ransomware sample and then classify it manually. We were able to drastically reduce the false positive rate (FPR), which means very few benign files were wrongly classified. Though the false negativity rate (FNR) was negligible, while analyzing these samples that were wrongly classified as benign using various malware analysis tools, it was inferred that the samples employed code injection techniques. Those misclassified instances also exhibited advanced encryption techniques because of which the disassembler could not unpack the file properly.

5 Conclusion

Ransomware that can detect the virtualized sandboxing environment used to perform dynamic analysis will dodge detection by employing various evasive tactics. We proposed a static analysis method which uses natural language processing techniques to classify ransomware in order to address this disadvantage. In order to construct classifiers, we use an ensemble learning model called voting classifier consisting of four machine learning models (SVM, RF, LR, and GBDT). We use different lengths of N-gram opcode sequences (2, 3, and 4-g) with various threshold values to limit the number of features that represent each class. Tf-idf is used to determine the most potent N-gram features that are highly indicative with respect to each class, which

Table 6 Comparison with existing methods

Ransomeware detection model	Method	Dataset used	TPR (%)	FPR (%)	FNR (%)
Automated dynamic analysis of ransomeware: benefits, limitations, and use for detection [16]	Algorithm: regularized logistic regression feature selection: MIC	582-ransomeware 942-benign	96.34	0.16	3.66
Deep learning for ransomeware detection [17]	Algorithm: deep natural network. Detection before encryption begins	155 ransomeware Unknown benign	93.92	38	7.08
Leveraging machine learning techniques for windows ransomeware network traffic detection [18]	Algorithm: J48 decision tree. Feature selection: Tshark extractor	210-ransomeware 264-benign	97.1	1.6	2.9
Our method	Algorithm: ensemble model. Feature selection: N-gram, tf-idf	2983-ransomeware 2682-benign	98.82	0.31	1.18

results in yielding a high validation accuracy. Detailed tests on real datasets show that the other algorithms also perform as well as random forest, which prompted us to employ the voting classifier strategy. The findings conclude that classifiers using N-gram feature vectors of different lengths are a better model to effectively classify ransomware. The results then compared with existing ransomware detection methods that rely on dynamic analysis model prove that our static analysis model performs better.

As the future enhancements to this model, we can introduce into our dataset, malignant samples which have advanced code obfuscation capabilities such as code rearranging, injecting garbage/benign opcodes into the source code in order to avoid detection. We can also move to deep learning methods, probably, RNN for even better feature selection and improved classification efficiency.

Acknowledgements This work was carried out as part of a research project sponsored by the Science and Engineering Research Board (SERB), the Department of Science and Technology (DST), Government of India (GoI). We express our sincere gratitude to DST-SERB for the support they extended to us through the "Early Career Research" Award (Sanction No. ECR/2018/001709). We are much obliged to the Department of CSE for having facilitated the seamless research environment even during the adverse circumstances caused by COVID-19 pandemic. We sincerely thank all the faculties at the Department of CSE and CTS Laboratory for their meticulous support.

References

1. J. Fu, J. Xue, Y. Wang, Z. Liu, C. Shan, Malware visualization for fine-grained classification. IEEE Access **6**, 14510–14523 (2018)
2. A. Ali, Ransomware: a research and a personal case study of dealing with this nasty malware. Issues Inform. Sci. Inf. Technol. **14**, 087–099 (2017)
3. B. Eduardo, D. Morat Oss, E. Magana Lizarrondo, M. Izal Azcarate, A survey on detection techniques for cryptographic ransomware. IEEE Access **7**, 144925–144944 (2019)
4. B.A. Al-rimy, M.A. Maarof, S.Z. Shaid, Ransomware threat success factors, taxonomy, and countermeasures: a survey and research directions. Comput. Secur. **74**, 144–166 (2018)
5. K. DaeYoub, J. Lee, Blacklist versus whitelist-based ransomware solutions. IEEE Consumer Electron. Mag. **9**(3), 22–28 (2020)
6. A. Pekta, T. Acarman, Classification of malware families based on runtime behaviors. J. Inf. Secur. Appl. **37**, 91–100 (2017)
7. J.O. Kephart, W.C. Arnold, Automatic extraction of computer virus signatures, in *Proceedings of the 4th Virus Bulletin International Conference* (Abingdon, UK, 1994)
8. A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, UNVEIL: a largescale, automated approach to detecting ransomware, in *Proceedings of the 25th USENIX Conference on Security Symposium* (USENIX Security, 2016), pp. 757–772
9. 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
10. 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
11. G. Ramesh, A. Menen, Automated dynamic approach for detecting ransomware using finite-state machine. Dec. Support Syst. **138**, 113400 (2020)
12. R. Vinayakumar, K.P. 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
13. H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, A.K. Sangaiah, Classification of ransomware families with machine learning based on N-gram of opcodes. Future Gener. Comput. Syst. **90**, 211–221

14. 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
15. 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
16. D. Sgandurra, L. Muoz-Gonzlez, R. Mohsen, E.C. Lupu, Automated dynamic analysis of ransomware: benefits, limitations and use for detection (2016). arXiv:1609.03020
17. A. Tseng, Y. Chen, Y. Kao, T. Lin, Deep learning for ransomware detection. Internet Archit. IA2016 Workshop Internet Archit. Appl. IEICE Techn. Rep. **116**(282), 87–92 (2016)
18. O.M. Alhawi, J. Baldwin, A. Dehghantanha, *Leveraging machine learning techniques for windows ransomware network traffic detection, in Cyber Threat Intelligence* (Springer, Cham, 2018), pp. 93–106
19. D. Bilar, Opcodes as predictor for malware. Int. J. Electron. Secur. Digital Forensics **1**(2), 156–168 (2007)
20. R. Moskovitch, C. Feher, Y. Elovici, Unknown malcode detectiona chronological evaluation, in *IEEE International Conference on Intelligence and Security Informatics, 2008. ISI* (IEEE, 2008), pp. 267–268
21. R. Moskovitch, et al., Unknown malcode detection via text categorization and the imbalance problem, in *2008 IEEE International Conference on Intelligence and Security Informatics* (IEEE, 2008), pp. 156–161
22. R. Moskovitch et al., *Unknown malcode detection using opcode representation, in Intelligence and Security Informatics* (Springer, Berlin Heidelberg, 2008), pp. 204–215
23. I. Firdausi, et al., Analysis of machine learning techniques used in behavior-based malware detection, in *2010 Second International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)* (IEEE, 2010)
24. L. Yi-Bin, D. Shu-Chang, Z. Chao-Fu, B. Gao, Using multi-feature and classifier ensembles to improve malware detection. J. CCIT **39**(2), 57–72 (2010)
25. I. Santos et al., *Opcode-sequence-based semisupervised unknown malware detection, in Computational Intelligence in Security for Information Systems* (Springer, Berlin, Heidelberg, 2011), pp. 50–57
26. Z. Zhao, A virus detection scheme based on features of control flow graph, in *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)* (IEEE, 2011 Aug 8), pp. 943–947
27. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**, 436 (2015). http://dx.doi.org/10.1038/nature14539
28. Y. Lecun, Generalization and network design strategies, in *Connectionism in Perspective* (Elsevier, 1989)
29. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
30. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoderdecoder for statistical machine translation, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Association for Computational Linguistics, Doha, Qatar, 2014), pp. 1724–1734. http://dx.doi.org/10.3115/v1/D14-1179
31. M. Schuster, K.K. Paliwal, Bidirectional recurrent neural networks. IEEE Trans. Signal Process. **45**(11), 2673–2681 (1997)
32. N. Harini, T.R. Padmanabhan, 2CAuth: a new two factor authentication scheme using QR-code. Int. J. Eng. Technol. **5**(2), 1087–1094 (2013)
33. G. Ramesh, I. Krishnamurthi, K. Sampath Sree Kumar, An efficacious method for detecting phishing webpages through target domain identification. Dec. Support Syst. **61**, 12–22 (2014)
34. N. Harini, T.R. Padmanabhan, 3c-auth: a new scheme for enhancing security. Int. J. Netw. Secur **18**(1), 143–150 (2016)