ISSN 1870-4069

Analysis of Feature Sets for Malware Family Classification

Jesús Javier Reyes Torres, Eleazar Aguirre Anaya, Ricardo Menchaca Méndez, Nareli Cruz Cortés, David Alejandro Robles Ramírez

Instituto Politécnico Nacional, Centro de Investigación en Computación, Mexico
{eaguirre, ric, nareli}@cic.ipn.mx
{b151223, b151224}@sagitario.cic.ipn.mx

Abstract. Malware families are evolving constantly, in order to evade the detection mechanisms, the authors modify the order of functions, add random useless code and add new features, it has been observed that these new variations share common characteristics with previous versions, these existing patterns in the malware allows us to generate characteristics to describe it, for later use in a machine learning algorithm. In this paper, we analyze different feature sets extracted from malicious portable executables which are used as input to a machine learning algorithm, these features are extracted using n-grams, and are used in three classification models: Logistic Regression, Random Forest and Support Vector Machines.

Keywords: malware classification, N-grams, portable executable, machine learning, logistic regression, random forest, support vector machine.

1 Introduction

Malicious software currently represents a computational security problem, so it is important to recognize the prevalence and continuing growth of malicious software. In the year 2015, 16.7 million of new variants of malware were found, according to Symantec [1]. Most of the new malicious software is designed to evade the anti-malware systems which use signature based methods to detect it. The authors modify the order of the functions, add random useless code, obfuscation techniques as packing in order to evade the detection methods.

The malicious software analysis techniques are classified into dynamic and static approach. In dynamic analysis, malware's information is collected from the operating system at runtime, such information can be system calls, network access and files, in this approach it is hard to simulate appropriate conditions in which the malicious software can execute its malicious functions, and at the beginning we do not know the time period needed to observe the malicious activity of the program. In static analysis, the information is gathered from the executable file without executing it, this information represents the expected behavior, in this approach features are extracted by an analyst, these features can be from the disassemble code or from the hexadecimal view of the binary.

In this paper we address the classification, using static features extracted from nine malicious software families obtained from a Microsoft database. Our approach is a non-signature based method in which we use machine learning to generate a classification model to classify a given sample in one of the nine families used to generate the model.

1.1 Scope

The most commonly used operating system is the Microsoft Windows operating system, this make the operating system an attractive target for malware authors. The principal executable format in Windows is the Portable Executable format (PE), which is a Microsoft standard. In our work, all the samples are malicious software written in the Portable Executable format, some of them for 32-bit systems and others for 64-bit systems.

2 Related Work

Different approaches for malicious software detection have been presented for years, at first F. Cohen [2] showed that in general the problem of virus detection is undecidable. In [3] Harris and Miller establish that the characteristics extracted through the analysis of the binary code provide information about the content and structure, for example instructions, basic blocks, functions and modules.

Machine learning for malware detection has been widely used, Siddiqui et al. extract variable length instructions sequences that can identify trojans from clean programs using data mining techniques [4]. Schultz et al. in [5] present a data mining framework to detect new malicious executables, they use different algorithms to generate classification models, the Multi-Naïve Bayes method was reported as the one with the best accuracy and detection rate over unknown programs with the value of 97.76%.

In [6] different patterns were used to detect the presence of malicious content in executable files. The analysis is made principally taking in consideration the bytecode as in [7] where they compute statistical and information-theoretic features in a block-wise manner to quantify the byte-level file content, and in [8] the authors use a static analysis methodology for representing malicious codes, their framework seeks to acquire the most important files, benign and malicious, in order to improve classifier performance.

N-grams is a way to represent the malicious software content, it consists in generate substrings with length *n* from a larger string, since n-gram overlap, they do not capture just statistics about substrings of length *n*, but they implicitly capture frequencies of longer substrings as well [9], in [10] Santos et al. demonstrate that a n-gram-based methodology signatures can achieve detection of new or unknown malicious software, Abou-Assaleh *et al.* [9] demonstrates that applying the CNG method based on byte n-gram analysis good results can be achieved, in [11] the authors use the n-grams to develop an automatic malware categorization system (AMCS) by observing the common characteristics shared by different malicious software families.

The operational code (OpCode) also has been used as static information to represent malicious software, in [12] O'Kane et al. use OpCode to detect encryped malware using SVM, Santos *et al*. in [13] propose a method to detect unknown malware families, using

the frequency of appearance of OpCode sequences as a base, and in [14] Bashari Rad et al. use static analysis to generate histograms of machine instructions frequency to be used as a features to classify the obfuscated version of metamorphic viruses.

Narayanan *et al.* in [15] obtained suggestions that every malware software belonging to a family has a distinct pattern, these patterns are quite similar between a family and distinguishable across other families.

In [16] Srakaew *et al.* compare two malware classification methods using data mining, they use two different types of features: statistical features and abstract assembly features, they observed that the abstract assembly approach is more promising, giving high accuracy with less complicated model.

3 Data Set

The data set was obtained from a Microsoft repository, this data set contain nine different malware families which are Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY and Gatak, the data set is unbalanced as can be observed in Fig. 1.

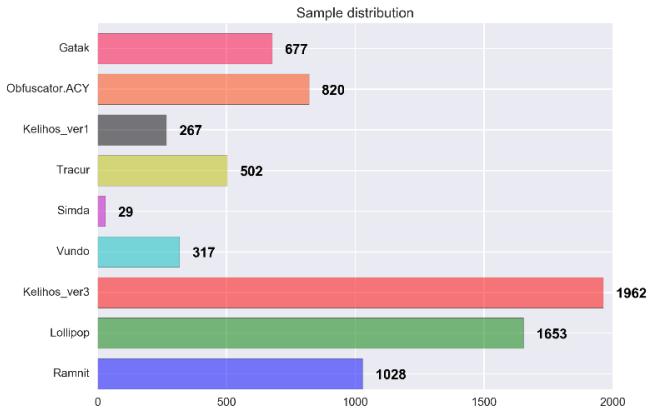

Fig. 1. Sample distribution.

4 Methodology

In this section, we describe the malicious software classification process, in our work we use static features generated from the malicious samples. In general, there are three distinct stages when a machine learning malicious software detection approach is used, these sections are: Feature extraction, sometimes feature selection is applied in order to reduce the dimensionality of the file's representation, and the generation of a classifier model using a machine learning algorithm.

The flow followed in this work is shown in Fig. 2. in each stage, different methods are used and a different data representation is obtained. In the malware analysis process, the analyst mostly of the time only have the malicious executable to start with, is very

improbable to have the source code of a malware sample. So, it is convenient to transform the executable into a more manageable representation, in this work the ASM and hexadecimal view are used as the representation of the malicious files.

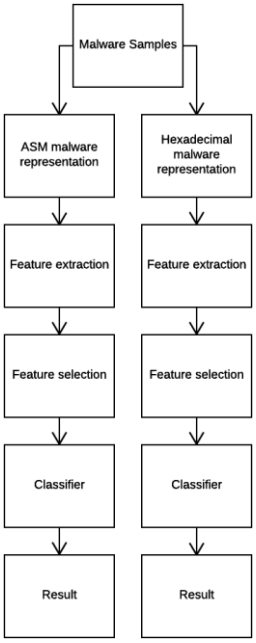

Fig. 2. General framework of malware classification system.

These new representations are processed to extract the information with which we want to work with, from the hexadecimal representation we extract only the hexadecimal code, the memory addresses are not taken into account.

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Fig. 3. Hexadecimal view from an executable file.

From the ASM representation we only use the OpCode field, ignoring the label field, the operand field and comment field:

$$[Campo\ de\ etiqueta]\ \ [Campo\ de\ Opcode\ \ [Campo\ del\ operando]]\ [Campo\ de\ comentarios]$$

The complete set of mnemonics present in the samples is not used, instead we use a reduced set of mnemonics, considered to be good in describe the malware samples. These instructions were selected using previous works in this topic.

Table 1. Reduced set of mnemonics**.**

and	fstcs	test	nop
int	setle	lea	shld
jnz	xor	jz	jb
fild	sub	jmp	std
imul	fdvip	mov	sbb
pop	retn	movzx	setb
loopd	ja	bt	push
jnb	add	lods	dec
pushf	call	rdtsc	rep
inc	adc	je	cmp

5 Feature Extraction

The features are created with n-grams applied to the final representation of the sample, the final executable representation with the hexadecimal representation is shown in Figure 5. and the final executable representation with the ASM representation is shown in Fig. 6.

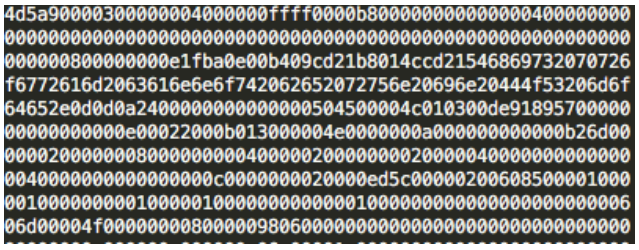

Fig 4. Hexadecimal representation of an executable file.

To these representation, the n-gram extraction method is applied, for the case of the hexadecimal we consider each character of the hexadecimal alphabet as a unit.

```
xor retn int int push lea sub mov push push
push mov mov call mov lea push push call mov
push call push call push push call push call
mov mov imul mov mov mov add mov mov mov lea
mov inc test jnz mov sub mov add mov mov mov
mov test jz mov cmp jz cmp jz cmp jz xor jmp
mov mov mov add mov add mov add mov mov mov
mov sub add mov mov mov mov add mov mov mov
mov mov test jz mov mov xor pop pop pop add
mov pop retn push push push push mov call mov
```

Fig 5. ASM representation of an executable file.

In Fig. 7. can be observed the process to generate the n-grams from a string with a value of *n* equal to 4, the number of n-grams generated from a string with length *L* can be calculated as follows:

$$Gr = L - (n - 1), \tag{1}$$

where *n* is the length of the sub-strings.

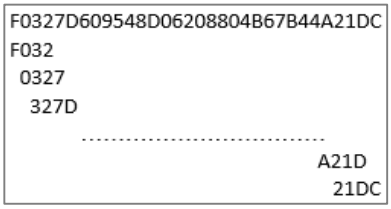

Fig. 6. N-grams generation process in the hexadecimal representation.

For the case of the ASM representation we consider each mnemonic as a unit, in Fig. 8. can be observed the process to generate the n-grams in this representation.

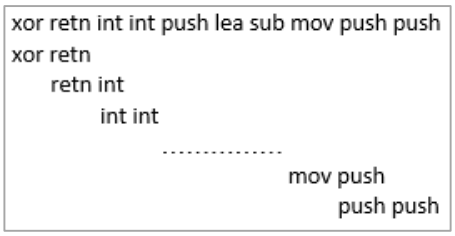

Fig. 7. N-gram generation process in the ASM representation.

After processing the samples into n-grams, the set of samples *S* can be represented as a matrix where each column represents a n-gram term and each row represent a sample $s \in S$. As we are using the supervised learning approach we also have to include the label of each sample, in our case we have nine levels, each one representing a malware family.

As can be observed in Fig. 9. between all the k samples which compose $S = \{s_1, s_2, \cdots, s_k\}$ we generate a set of terms (n-grams) $T = \{t_1, t_2, \cdots, t_m\}$ which is composed for all the unique strings whit length n generated by the extraction over all

the sample space, each cell in the terms space is filled with the normalized term frequency, showed in Equation 2:

$$tf(t,d) = \frac{f(t,d)}{max\{f(w,d) : w \in d\}}. \quad (2)$$

Here we divide the frequency of a term t in a determined document d by the maximum frequency of some term w in the same document.

Sample label	**Terms (N-gram)**					
s_1	t_1	t_2	t_3	t_4	$\cdots$	t_m
s_2	t_1	t_2	t_3	t_4	$\cdots$	t_m
s_3	t_1	t_2	t_3	t_4	$\cdots$	t_m
s_4	t_1	t_2	t_3	t_4	$\cdots$	t_m
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_k	t_1	t_2	t_3	t_4	$\cdots$	t_m

Fig. 8. Samples representation matrix.

6 Feature Selection

With the n-grams a large number of features is generated, this number may vary according to the unique number of sub-strings that can be found in the extraction process over the sample space, this number depends in the alphabet used to represent the data, in our case for the hexadecimal representation the alphabet is $A_H = \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$ which are the characters used to represent hexadecimal digits, and for the case of the ASM representation, the alphabet A_{ASM} is composed by the forty mnemonics in Table 1.

The number of substrings which represent the features in our machine learning approach has as an upper bound all the possible unique strings whit length n that can be generated with our alphabet.

$$UB = (Alphabet)^n. \quad (3)$$

As we are using machine learning to approximate the functional relationship $f()$ between an input $X = \{x_1, x_2, \cdots, x_M\}$ and an output Y, based in a tuple relation $\{X_i, Y_i\}$, $i = 1, \cdots, N$, sometimes the output Y is not determined by the complete set of the input features $\{x_1, x_2, \cdots, x_M\}$, instead, it is decided only by a subset of them $\{x_{(1)}, x_{(2)}, \cdots, x_{(m)}\}$, where $m < M$.

The irrelevant features may lead in increase the computational cost and overfitting, a rule of thumb about the relation between the number of samples and the features to describe them, this rule claim that the numbers of samples needed to make a good generalization has to be more or equal to ten times the number of effective features.

$$N \geq 10(effective\ features). \quad (4)$$

Our sample space is composed of 7255 malicious software samples, and if for instance we generate the n-grams with a value of $n = 4$ over the hexadecimal

representation, the upper bound number of features is 65536, which is a large number of features compared with the number of samples.

7 Feature Sets

We generate different feature sets and use them as an input of a machine learning algorithms to generate different classification models. The different feature sets generated are shown below:

- Features generated using hexadecimal representation and the n-gram method with a value of $n = 2$.
- Features generated using hexadecimal representation and the n-gram method with a value of $n = 4$.
- Features generated using opcode representation and the n-gram method with a value of $n = 2$.

Also, a feature vector is generated using those generated by the hexadecimal representation with a value of $n = 2$ and opcode representation with a value of $n = 2$. The number of features generated with the different representations are shown in Table 2.

Table 2. Number of features generated.

Features	Number of features
Hexadecimal $n = 2$ (D1)	257
Hexadecimal $n = 4$ (D4)	65537
Opcode $n = 2$ (D2)	1138
Hexadecimal $n = 2$ and Opcode $n = 2$ (D6)	1394

Taking into account the rule of thumb presented in (4) in three cases this is not fulfilled, so it was proposed to use feature selection to reduce the dimensions of those vectors.

The feature selection used in this work is the univariate feature selection, using the $\mathcal{X}^2$ metric which is used to test the independence of two events, the method examines each feature independently to determine the strength of the relationship of the feature with the response variable.

Table 3. Number of features generated by applying feature selection.

Feature	Number of features
Hexadecimal $n = 4$ (D5)	725
Opcode $n = 2$ (D3)	725
Hexadecimal $n = 2$ and Opcode $n = 2$ (D7)	725

In Table 3 is shown the number of features generated for each representation taking into consideration that we only have 7255 samples to use in the model generation. The

algorithms in which we prove those different feature sets are Logistic Regression, Random Forest, K Nearest Neighbors and Support Vector Machines.

Important parameters of the machine learning algorithms:

- Logistic regression: L2 for penalty, balanced class weight, primal formulation.
- Random Forest: ten estimators, gini impurity, two as minimum sample splits, one sample required to be a leaf node, bootstrap samples to build a tree, balanced class weight.
- K nearest neighbors: three neighbors, minkowski metric with p=2.
- Support Vector Machine: L2 for penalty, square of the hinge loss, error term equal 10, dual problem of optimization, radial kernel, gamma equal to 0.5.

We used the cross-validation of K-Fold as validation method, as shown in Fig. 1, the samples representing each family of malware are unbalanced so we use the stratified method of the method, to maintain the representativeness of each family when the model is generated.

8 Experiments and Results

The experiments were made in a computer system with the following characteristic: processor Intel Core i5 at 2.8 GHz, 16 Gb of RAM memory and macOS Sierra as operative system. The software used were programmed in python 2.7.

The metrics used to analyze the results were Precision, Recall and F1-score, the reason why we do not use accuracy is that our database is unbalanced and we have to make sure that the family with the less number of samples (Simda) is detected by the model generated.

Below are the results obtained for each feature set with the different machine learning algorithms using K-Fold Cross validation with a value of $K = 10$.

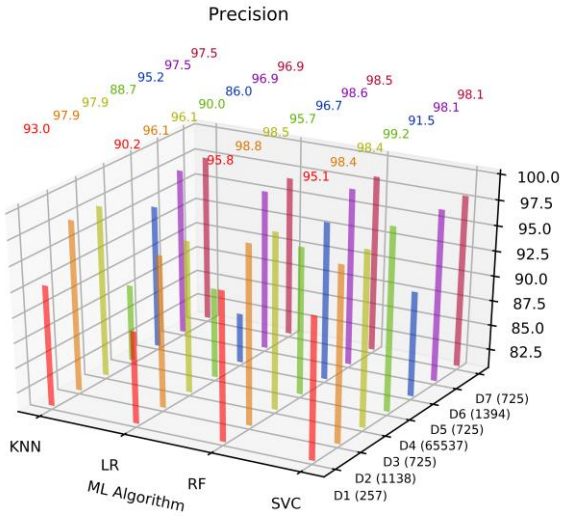

Fig. 9. Precision metric for all the feature sets.

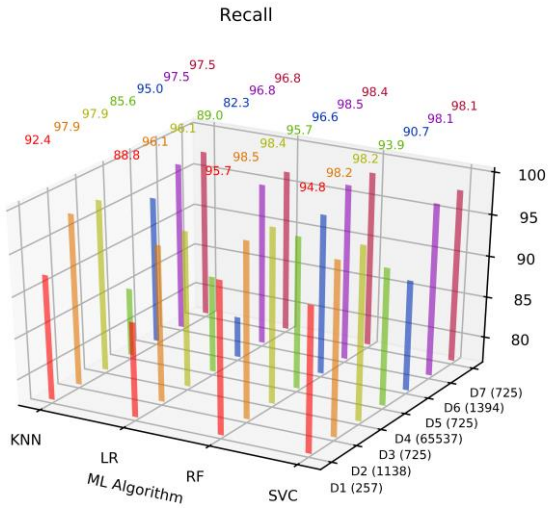

Fig 10. Recall metric for all the feature sets.

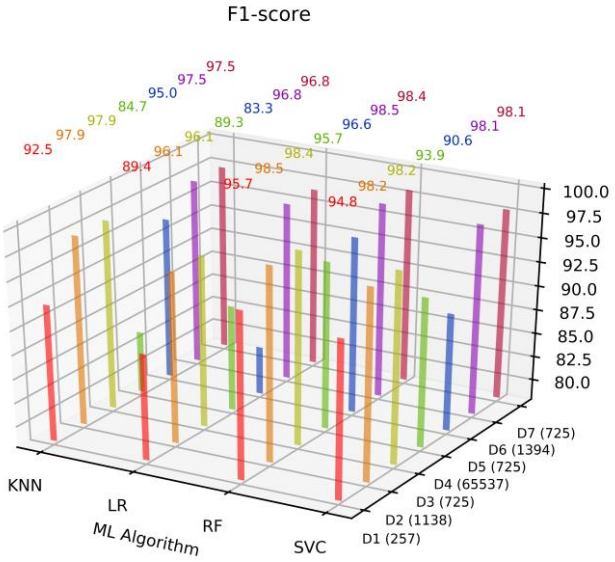

Fig. 11. F1-score metric for all the feature sets.

In the state of the art the accuracy is the metric used to evaluate the model performance, below is shown a table with some works, and their performance.

It is possible to make a comparison but it is not strict, because the metrics are not the same, we prefer not to use the accuracy due to the characteristics of the database, in addition to that in our dataset there are samples that present obfuscation. Even so the result that we obtained is competent with those mentioned in table 4.

Table 4. Results obtained in the state of the art.

	Approach	Results
N-gram-based detection of new malicious code	Using n-grams with a value of n from 1 to 10. CNG classification method.	Maximum accuracy of 98%
Pattern based feature selection	Using 4-gram and selecting fea-tures with PCA. Support Vector Machine.	Accuracy of 97.7%
Malware detection using sta-tistical analysis of byte-level file content	Features extraction using statisti-cal and information theory fea-tures. Boosted J48 algorithm.	Area Under the curve of 96.2%
Proposal of n-gram based al-gorithm for malware classifi-cation	Using 4-gram and the top 60 fea-tures. Similarity function to calculate.	Accuracy of 98%
Selecting features to classify malware	Using features from metadata in the PE header, data about 10 sec-tions and all import and exports. J48 algorithm.	Accuracy of 98%

9 Discussion

In all the feature sets the algorithm which the best performance is obtained is Random Forest, between them can be observed that using the features generated by the opcode representation with a value of $n = 2$ (D2) and the features generated using the hexadecimal representation with a value of $n = 2$ plus the generated using the opcode representation with a value of $n = 2$ (D6) highest performance is obtained, in both cases the number of features do not fulfil the rule of thumb presented in (4) so it is possible that we are overfitting the model to the data, if we look at the results obtained using the same feature sets but with the feature selection applied (D3 and D7) it can be observed that the values are not very different and It is possible that we are reducing overfitting in order to make better predictions outside the training/validation dataset.

Generally, the most difficult family to model is the is Obfuscator.ACY, the detection rate of this family in the two models with highest performance is very close to hundred percent, in the case when the feature selection was applied to the features generated, the number of samples well classified decrease at most by six percent.

Another factor taken into consideration is that the family Simda has very few samples, in the four models generated using Random Forest one third of the samples belonging to this family were classified correctly. With the four sets mentioned above also were obtained the best results for the other three algorithms compared with the remaining feature sets.

The worst results were obtained using the features generated from the hexadecimal representation with a value of n=4 for the n-grams, the performance is improved by using these features applying feature selection in K Nearest Neighbor and Random Forest algorithms, but it gets worse with Logistic Regression and Support Vector Machines.

10 Conclusions

As can be observed in the graphs some of the feature sets are good to describe our database, the features with which the best performance is achieved are they generated using the opcode representation with a value of n=2 and the case in which to these features are aggregated the features generated using the hexadecimal representation with a value of n=2, applying feature selection to both cases we achieve almost the same results, the algorithms in which the best results are obtained are Random Forest and Support Vector Machines, that can be because in Random Forest the number of features can be large, and in SVM due to the regularization parameter the algorithm is more resistant to overfitting.

In the algorithms KNN and Logistic regression we observe a improvement when the features generated with the opcode representation is used compared with these obtained using the features generated with the hexadecimal representation, so it can be say that the opcode is better to describe our data.

Acknowledgements. The authors thank the Instituto Politécnico Nacional an CONACYT for their support in the realization of this work.

References

1. Symantec reports. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf [Last accessed: 11 08 2017]
2. Cohen, F.: Computer viruses: theory and experiments. Computers & security 6(1), 22–35 (1987)
3. Harris, L.C., Miller, B. P.: Practical analysis of stripped binary code. ACM SIGARCH Computer Architecture News 33(5), 63–68 (2005)
4. Siddiqui, M., Wang, M.C., Lee, J.: A survey of data mining techniques for malware detection using file features. In: Proceedings of the 46th annual southeast regional conference on xx, pp. 509–510, ACM (2008)
5. Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Security and Privacy 2001 (S&P 2001), IEEE Symposium on, pp. 38–49 (2001)
6. Liangboonprakong, C., Sornil, O.: Classification of malware families based on n-grams sequential pattern features. In: Industrial Electronics and Applications (ICIEA), 2013 8th IEEE Conference on., pp. 777–782. IEEE (2013)
7. Tabish, S.M., Shafiq, M.Z., Farooq, M.: Malware detection using statistical analysis of byte-level file content. In: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatic, pp. 23–31, ACM (2009)
8. Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., Elovici, Y.: Detecting unknown malicious code by applying classification techniques on opcode patterns. Security Informatics 1(1), 1 (2012)
9. Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.: N-gram-based detection of new malicious code. In: Computer Software and Applications Conference (COMPSAC 2004). Proceedings of the 28th Annual International, vol. 2, pp. 41–42. IEEE (2004)
10. Santos, I., Penya, Y.K., Devesa, J., Bringas, P.G.: N-grams-based File Signatures for Malware Detection. ICEIS 2(9), 317–320 (2009)

11. Ye, Y., Li, T., Chen, Y., Jiang, Q.: Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 95–104 (2010)
12. O'Kane, P., Sezer, S., McLaughlin, K.: N-gram density based malware detection. In: Computer Applications & Research (WSCAR), 2014 World Symposium on, pp. 1–6. IEEE (2014)
13. Santos, I., Brezo, F., Nieves, J., Penya, Y.K., Sanz, B., Laorden, C., Bringas, P.G.: Idea: Opcode-sequence-based malware detection. In: International Symposium on Engineering Secure Software and Systems. Springer, Berlin, pp. 35–43, Heidelberg (2010)
14. Rad, B.B., Masrom, M., Ibrahim, S.: Opcodes histogram for classifying metamorphic portable executables malware. In: e-Learning and e-Technologies in Education (ICEEE), 2012 International Conference on, pp. 209–213. IEEE (2012)
15. Narayanan, A., Yang, L., Chen, L., Jinliang, L.: Adaptive and scalable android malware detection through online learning. In: Neural Networks (IJCNN), 2016 International Joint Conference on, pp. 2484–2491. IEEE (2016)
16. Piyanuntcharatsr, S.S.W., Adulkasem, S., Chantrapornchai, C.: On the Comparison of Malware Detection Methods Using Data Mining with Two Feature Sets. International Journal of Security and Its Applications 9(3), 293–318 (2015)