Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Automatic malware classification and new malware detection using machine learning*

Liu LIU‡, Bao-sheng WANG, Bo YU, Qiu-xi ZHONG

(*College of Computer, National University of Defense Technology, Changsha 410073, China*)

E-mail: hotmailliuliu@163.com; wbshengnudt@163.com; BoYUnudt@sina.com; Qiuxizhong@163.com

Received June 12, 2016; Revision accepted Sept. 14, 2016; Crosschecked Sept. 15, 2017

Abstract: The explosive growth of malware variants poses a major threat to information security. Traditional anti-virus systems based on signatures fail to classify unknown malware into their corresponding families and to detect new kinds of malware programs. Therefore, we propose a machine learning based malware analysis system, which is composed of three modules: data processing, decision making, and new malware detection. The data processing module deals with gray-scale images, Opcode *n*-gram, and import functions, which are employed to extract the features of the malware. The decision-making module uses the features to classify the malware and to identify suspicious malware. Finally, the detection module uses the shared nearest neighbor (SNN) clustering algorithm to discover new malware families. Our approach is evaluated on more than 20 000 malware instances, which were collected by Kingsoft, ESET NOD32, and Anubis. The results show that our system can effectively classify the unknown malware with a best accuracy of 98.9%, and successfully detects 86.7% of the new malware.

Key words: Malware classification; Machine learning; *n*-gram; Gray-scale image; Feature extraction; Malware detection
https://doi.org/10.1631/FITEE.1601325 **CLC number:** TP309.5

1 Introduction

Malware, also known as malicious software, refers to any software that causes damage to users, computers, or networks in some way. Malware contains viruses, worms, backdoors, Trojan horses, or other malicious programs (Gandotra *et al*., 2014). Currently, malware is an important challenge in the field of information security (Kong and Yan, 2013). According to a report from Kaspersky Labs (2015), in the past year, 58% of corporate computers were attacked and 29% of companies suffered from network attacks. Although there are hundreds of thousands of new malware found every day, most of them are derived from the known families of malware (Egele *et al*., 2012). Malware obfuscation techniques (Lee *et al*., 2010) include mainly packing, metamorphosis, and virtual technologies. These technologies have been used widely to evade the detection of anti-virus software (Musale *et al*., 2015). Most malware detection systems adopted by anti-virus manufacturers are based on signatures and anomaly detections. Although the signature technique has a high accuracy, it cannot detect new malware programs and has to update its feature library in real time (Yan *et al*., 2013) manually. New malware can be found by anomaly detection; however, the false alarm rate is high.

Based on the state of the malware that has been analyzed, malware analysis can be divided into static and dynamic analysis (Damodaran *et al*., 2017). Static analysis refers to the analysis of executable files without executing the program (Jain and Meena, 2011). The advantages of static analysis are its ability to find an author's style and profile the code flow. The disadvantage is that it is thwarted easily by

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61303264) and the National Basic Research Program (973) of China (Nos. 2012CB315906 and 0800065111001)

ORCID: Liu LIU, http://orcid.org/0000-0002-6523-1454

obfuscation techniques. Dynamic analysis, on the other hand, enables the observation of the running state of a program in a safe and controlled environment. This approach is able to reflect accurately the behavioral characteristics of the program. It is not affected by encryption, compression, metamorphosis, etc. However, this method spends time not only on debugging a program, but also in tracking and recording the running process of the program. Therefore, dynamic analysis is usually far more inefficient than static analysis. In addition, it is subject to some restrictions in the running environment (Russo and Sabelfeld, 2010). Malware may contain activation conditions; thus, some malicious actions may not be observed correctly.

With the successful application of machine learning in the fields of image processing, speech recognition, and software engineering (Yu *et al*., 2016), machine learning has become an important method for analyzing malware in the last 10 years (Wong and Stamp, 2006; Yao *et al*., 2012; Liu *et al*., 2015). Generally, by machine learning the process for detecting malware can be divided into three steps. First, the features of the execution file, which are extracted by static and dynamic analysis, are mapped into the machine learning input (Shabtai *et al*., 2009). Second, a forecasting model is trained using these features, meaning that a type of higher-level signature is obtained. Finally, the unknown software is predicted. Machine learning is able to probe and discover the intrinsic properties of the software automatically; thus, it can distinguish between benign software and malware (Yan *et al*., 2013; Pascanu *et al*., 2015). Machine learning can also be used to assign an unknown malware to a known family (Tsyganok *et al*., 2012; Kong and Yan, 2013). However, malware authors can easily generate a large quantity and diversity of malware variants using automatic tools. Therefore, we are encouraged to use machine learning to solve the following problems: (1) How to assign variants quickly and effectively to corresponding families? (2) How to detect new malware?

In this study, we propose an incremental malware detection system to classify malware families and to detect new malware. This system is divided into three main parts: feature extraction and selection, decision making, and new malware detection. We have made the following contributions: (1) We propose a feature extraction method based on gray-scale images, Opcode *n*-gram, and import functions. The features are mapped into the feature vector for machine learning. (2) We use an improved information gain to reduce the high-dimensional features. (3) We create a decision-making system to assign the unknown malware to a corresponding family and to screen out suspicious software. The advantage of this system is that it can probe the capabilities of different classifiers. (4) We apply shared nearest neighbor (SNN) to find new malware. Compared with previous clustering methods, SNN is more suitable for dealing with high-dimensional data. (5) We prove through experiments that the accuracy and robustness of ensemble classifiers are better than those of other classifiers.

2 Related work

Lin *et al*. (2015) proposed a method to alleviate the time-consuming training of the model for malware classification. Support vector machines (SVMs) were used to classify malicious software. The high-dimensional aspects of the *n*-gram features were constructed by dynamic analysis. Therefore, they put forward a two-stage dimensionality reduction method. First, they applied term frequency–inverse document frequency (TF-IDF) (Salton and McGill, 1986) to select the important features. Then, they used principal component analysis (PAC) (Jolliffe, 2002) and kernel PCA (KPAC) to extract features to reduce the number of random variables. Multiple groups of experiments showed the effectiveness and efficiency of their method.

Lin and Stamp (2011) explored the vulnerabilities in hidden Markov model (HMM) based malware detection approaches. They created a virus-generating tool, which can produce metamorphic copies. They also found that some of the copies can evade HMM-based detection. Annachhatre *et al*. (2015) demonstrated a novel method for malware classification. It combines HMMs and an improved *k*-means algorithm. The innovative feature is that seven models of HMMs were generated by four different compilers and handwritten assembly code, as well as two metamorphic malware generators. The seven models were GCC, MinGW, TurboC, Clang,

TASM, NGVCK, and MWOR. After the models were trained by Opcode sequences, each malware sample received a 7-tuple of scores from the HMM models. A malware sample, which is more similar to the training dataset, should obtain higher scores. Then, the 7 tuples were mapped into the input of *k*-means, which separates the malware samples into clusters. Finally, the experiment results demonstrate that their method is an effective way to automate the classification of malicious code. Visualization techniques have also been applied widely in many fields of computer science. In recent years, these techniques have also been used to analyze malicious code. Han *et al*. (2013) converted binary files into bitmap images. Then, the bitmap image was employed to generate an entropy graph by calculating the entropy values. They used the entropy graph to calculate the similarity of the malware. Nataraj *et al*. (2014) computed the texture features of malware to find similarities using generalized search trees (GIST), which uses a wavelet decomposition of an image. The methods proposed by Han *et al*. (2013) and Nataraj *et al*. (2014) apply different image features, and their classification accuracies are between 0.97 and 0.98. However, the calculation time of the former is less than that of the latter, meaning that the former is more efficient.

Previous studies have proved that Opcode *n*-gram patterns can be used to extract the features of malware effectively (Ding *et al*., 2014; Kapoor and Dhavale, 2016). For example, Ding *et al*. (2014) proposed the control flow graph (CFG) to find all the execution paths that are Opcode sequences. Then, *n*-grams are extracted from the Opcode stream, and used as features which are the top *n*-grams, based on a document frequency score. Kapoor and Dhavale (2016) used a similar approach to extract *n*-gram features. Bi-normal separation is used for feature scaling. Meanwhile, they used information gain and χ^2 test to select features. Compared with the information gain, a χ^2 test can not only detect malware, but also achieve an accuracy of muticlassification up to 0.972.

Many methods based on Windows application programming interfaces (APIs) have been proposed (Tian *et al*., 2010; Iwamoto and Wasaki, 2012). They use API to extract knowledge about the behavior of the executable file. Tian *et al*. (2010) used a trace tool called 'HookMe' to collect all the trace reports of the malware. Then, they dealt with all the API call strings and parameters as separate strings. They set a list to store all the strings in all the malware. Moreover, each file receives a feature vector that has the same length as the list. In the vector, '1' denotes that a string is contained in the list and '0' means that a string is not present. Their experiments showed that the random forest (RF) classifier achieves the best result. Iwamoto and Wasaki (2012) created an API-sequence graph for each sample. They compared the graphs to calculate Dice's coefficient, which is employed to classify the samples. Although their accuracy is lower than that reached by Tian *et al.* (2010), their dataset is closer to the real situation.

Cheng *et al*. (2013) applied information retrieval (IR) theory to classify malicious programs. They employed a dynamic analysis method to extract the sequences of API calls and system calls, providing the discriminatory features from different layers. Compared with previous methods, they recorded all parameters of the API calls and system calls stored in documents. Then, TF-IDF weighting and IR theory were applied to describe and retrieve malware families. Experiments showed that the accuracy of their method is better than those of other methods.

Besides dynamic and static analysis methods, hybrid analysis has attracted attention in academia. Roundy and Miller (2010) proposed a new method which integrates static and dynamic techniques to construct control- and data-flow analysis. To detect the obfuscated malware, Islam *et al*. (2013) also integrated two features. Experiments proved that their method has not only a high detection rate, but also a very high robustness.

3 Our methods

3.1 System overview

As shown in Fig. 1, our system can be divided into three parts: data processing, decision making, and clustering. Data processing handles feature extraction and selection. The system uses Opcode *n*-gram, gray-scale images, and the import function to extract malware features. The selection method reduces the dimension of the feature space to prevent a dimension disaster and to improve classifier performance. In the decision-making system, several

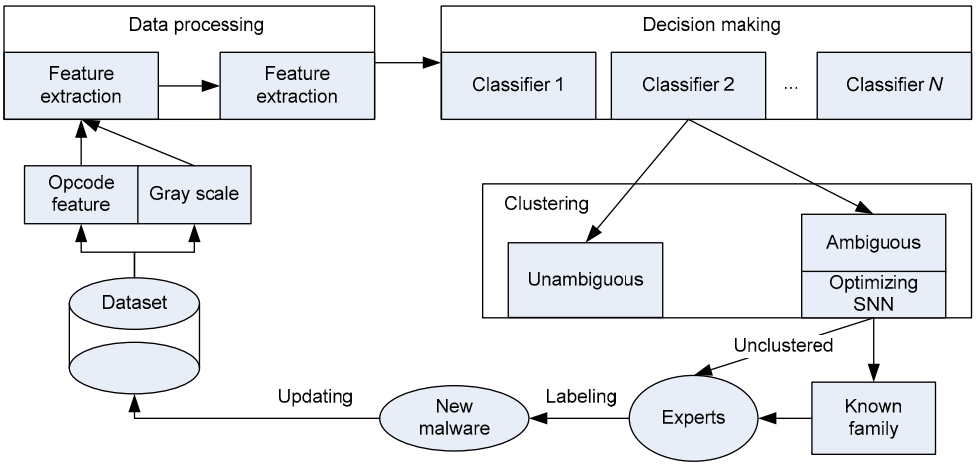

Fig. 1 Schematic of our system which is composed of three modules: data processing, decision making, and clustering (SNN: shared nearest neighbor)

classifiers are trained by a large number of instances to predict the unknown samples. Meanwhile, they decide jointly whether the detected samples are questionable. In the clustering phase, the suspicious samples are assigned to their corresponding family. If a sample cannot be clustered as a known malware, it will be stored in the database for the next clustering.

3.2 Feature extraction

Malware classification based on gray-scale imaging is a novel approach. It has been proved an effective static analysis tool (Kancherla and Mukkamala, 2013; Nataraj *et al*., 2014). A gray-scale image is an image that is expressed in gray scale. The brightness of white to black is divided into 256 grades according to a logarithmic relationship. The different physical information from the graphs can cause a corresponding gray-scale difference, and textures confirm the reflection of the gray-scale difference in the visual field. To exploit the texture difference of malware, we first use the interactive disassembler (IDA) to obtain binary files of the malware. Then, we divide the content of the binary files into many small units, each of which contains eight bits and is converted to an unsigned integer in the range of 0–255. In a gray-scale image, 0 and 255 represent black and white, respectively. Finally, the transformed file is mapped to a matrix called the 'gray-scale matrix'. The width of the matrix is usually initialized to 2^n. In our experiments, n=8.

In Fig. 2, four types of malware belonging to three malware families are converted to gray-scale images. The two types of malware in Figs. 2a and 2b, whose textures are similar, belong to the same family. The malware in Fig. 2c is from family Virus.Win32.Afgan. The malware in Fig. 2d is from family Trojan-Banker.Win32. From Fig. 2, we arrive at the following conclusions: the textures of malware are different for different families, and similar within the same family. However, the matrix is not suitable to be used as our feature expression. To adapt it for further use, the gray-scale matrix is mapped into a 1D vector called a 'gray-scale vector'.

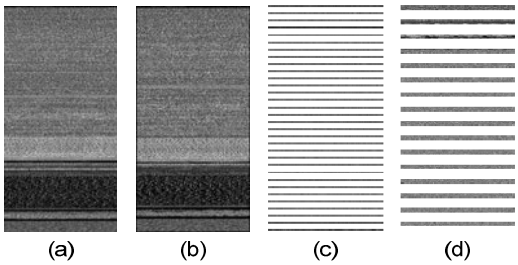

Fig. 2 Malware textures: (a) and (b) belong to the same family; (c) is from family Virus.Win32.Afgan; (d) is from family Trojan-Banker.Win32

3.3 Opcode *n*-gram

We use IDA Pro to reverse the analysis of malware. IDA Pro is a powerful interactive disassembler that Hex-Rays released (Tian *et al*., 2009; Ye *et al*., 2010). It is not only applied to obtain the assembly file from malware, but also used to identify the function blocks and describe the function flow chart, gain import functions, etc.

3.3.1 *n*-gram

In this study, we employ an *n*-gram model to extract Opcode features from a malware program. The model is an effective method for text feature extraction. It is based on the hypothesis that the appearance of the *n* words is related only to the previous *n*−1 words, wherein *n* represents the length of a characteristic sequence. If we have a set that contains *L* Opcodes, the set will be divided into *L*−*n*+1 feature sequences. The model finds feature sequences in a sliding window way. For example, a 3-gram model is employed to obtain feature sequences from {push, call, add, mov, xor, inc, pop}. As shown in Fig. 3, we extract five short sequences, and each sequence includes three Opcodes.

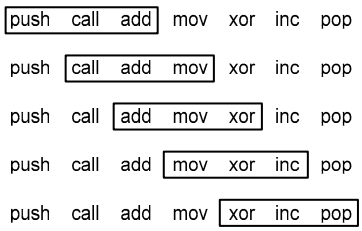

Fig. 3 Opcode 3-gram

3.3.2 Control flow graph

The programs, written by a high-level language, are composed of conditional judgment, loop statement, call function, etc.; thus, the relationship between the functions is very important in the analysis of the program (Kinable and Kostakis, 2011). Instructions cannot be limited to local relationships. However, they should be put into the program flow. Therefore, we analyze the malware by constructing a CFG. Fig. 4 shows how we construct the CFG of functions using IDA Pro. Functions are a part of VIRUS.Wind32.Afgan.c. To express the function call relationship, we need to count the times each function is called. To describe the program features correctly, we make the function block a basic unit, so we can extract only feature sequences in the block.

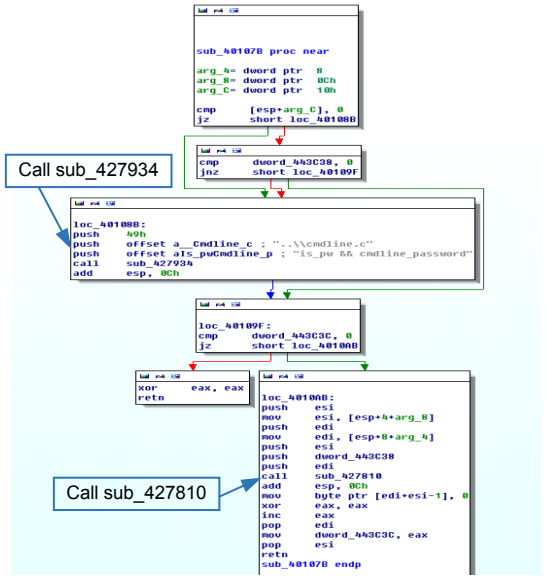

Fig. 4 Control flow graph

3.3.3 Opcode 3-gram extraction

We combine CFG with 3-gram to extract the Opcode features of the malware. Fig. 5 describes the process of an Opcode feature extraction. First, we distill a set of Opcodes with the CFG sequentially.

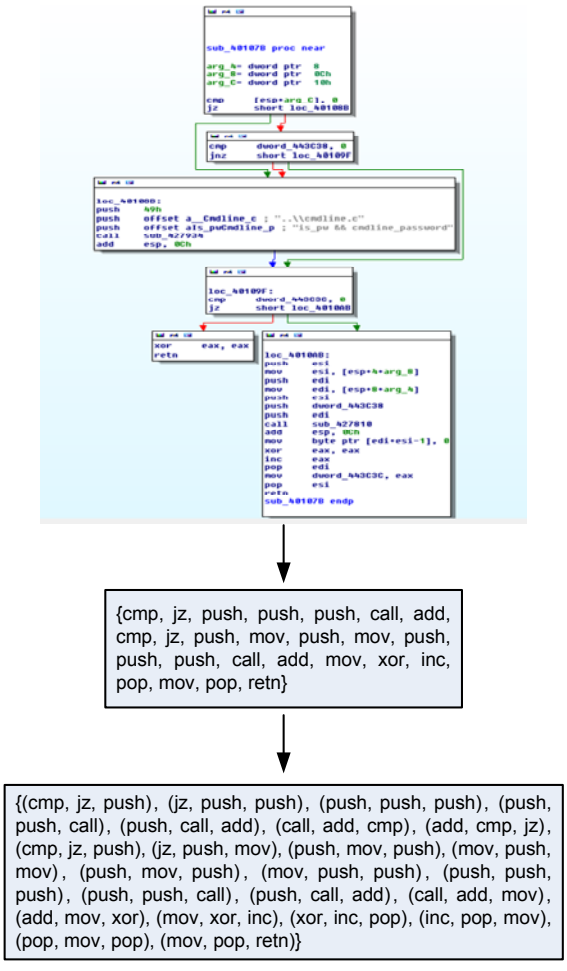

Fig. 5 Opcode 3-gram extraction

The set contains 25 Opcodes. Then we apply 3-gram to extract 23 feature sequences. There are two functions named 'sub_427934' and 'sub_427810', which are called in sub_40107B. The counters are added to a unit automatically. This means that a function is called m times, and its characteristic sequences will be counted m times.

3.4 Import functions

Import functions for a program are contained mainly in dynamic link libraries. If a malware program hopes to achieve a specific function, it needs to call a specific library function. For example, wsock32.dl can perform the relevant network task, and Kernel32.dll contains access and memory operations, files, hardware, and other functions. Advapi32.dll provides functions which can access core Windows components. Thus, the information for these import functions can help us analyze the malware's purpose. Therefore, we count the times each DLL appears in the import functions, and the statistical values will be a part of the malware features. Fig. 6 shows the import functions for VIRUS.Wind32.Afgan.c, where the 'Name' column contains names of the call functions, and the 'Library' column is the dynamic link library (DLL) of the corresponding function. The frequency of each import function will be an important part of malware features.

Address	Or	Name	Library
00000000006B777C		ExitProcess	KERNEL32
00000000006B7778		GetProcAddress	KERNEL32
00000000006B7774		LoadLibraryA	KERNEL32
00000000006B7784		RegCloseKey	advapi32
00000000006B778C		ImageList_Add	comctl32
00000000006B7794		ChooseFontA	comdlg32
00000000006B779C		SaveDC	gdi32
00000000006B77A4		IsEqualGUID	ole32
00000000006B77AC		VariantCopy	oleaut32
00000000006B77B4		OleCreatePropertyFrame	olepro32
00000000006B77BC		AMGetErrorTextA	quartz
00000000006B77C4		DragFinish	shell32
00000000006B77CC		GetDC	user32
00000000006B77D4		VerQueryValueA	version
00000000006B77DC		mixerOpen	winmm
00000000006B77E4		OpenPrinterA	winspool
00000000006B77EC		send	wsock32

Fig. 6 Import functions

3.5 Feature selection

It is important to choose features that have the ability to distinguish malware families. The features are extracted by the n-gram model as high-dimensional data. To improve classification accuracy and reduce time consumption, a novel method is applied to reduce the dimensionality of the data. To assist in the description, we review a few concepts. $Y=\{0, 1, \ldots\}$ represents the label of the malware family. s_i refers to the feature sequence. The frequency of the sequence is

$$f(s_i)=\mathrm{sum}(s_i \mid y_j)\Big/\sum_{i\le n_j}\mathrm{sum}(s_i \mid y_j), \tag{1}$$

where $\mathrm{sum}(s_i|y_j)$ denotes the quantity of the sequence belonging to family y_j, and $F(s_i)$ is the frequency of the sequence in Y:

$$F(s_i)=\frac{\sum_{j\le n}\mathrm{sum}(s_i \mid y_j)}{\sum_{j\le n}\sum_{i\le n_j}\mathrm{sum}(s_i \mid y_j)}. \tag{2}$$

The information gain of the sequence is

$$I_{\mathrm{w}}(S;Y)=\sum_{y_i\in Y}\sum_{s_i\in S}p(s_i y_i)\log\frac{p(s_i,y_i)}{p(s_i)\cdot p(y_i)}. \tag{3}$$

$p(s_i, y)$ stands for the joint probability distribution of s_i and y, and $p(s_i)$ and $p(y)$ are the marginal probability distribution functions of S and Y, respectively. The information gain is used to measure the ability of a sequence to distinguish the malware. We employ a two-step strategy to reduce the dimensionality.

If feature s_i satisfies condition $|1-F(s_i)/f_j(s_i)|\le\zeta$, s_i is deleted. This shows that the feature does not have the ability to discriminate the malware family. Define a new information gain as

$$I_{\mathrm{w}}(S;Y)=\frac{1}{F(S)}\sum_{y_i\in Y}\sum_{s_i\in Y}p(s_i,y_i)\log\frac{p(s_i,y_i)}{p(s_i)\cdot p(y_i)}. \tag{4}$$

The expression has a weight term. This term is aimed at enhancing the importance of certain features that have low frequency and high discrimination. By calculating the information gain, we retain 500 features with larger values.

4 Decision-making system

We propose a decision-making system to capture suspicious software, which may belong to a known family or be a new malware program. This system

determines the label of a sample through integrating results of multiple classifiers. Compared with previous ensemble systems (Hu *et al*., 2007; Tao *et al*., 2013), we design a weight vector for each classifier. The weight vector contains n weight values, where n denotes the quantity of the sample family. In Fig. 7, there are N classifiers and N weight vectors. Each classifier is trained by a training set, which is sampled by Bootstrap. Testing set T_1 is used to verify the capability of each classifier. The capability is not only to gain a higher accuracy rate, but also to classify the special family precisely. Different classifiers may have different effects on different families. Therefore, each classifier is able to provide a higher degree of confidence for the specific classification results.

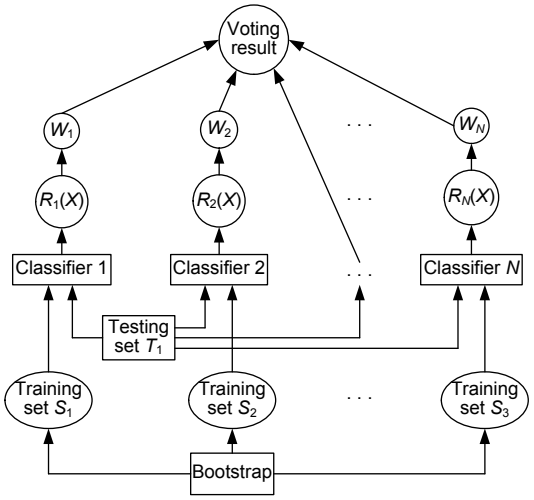

Fig. 7 Decision-making system

Fig. 8 refers to weight vectors. C and F represent the classifier and the sample family, respectively, W indicates the ability of classifier C_i to identify family F_j, and W_{ij}=TP/N_j, where TP is the number of samples that are classified correctly into F_j, and N_j is the quantity of samples included in F_j. In the voting phase, the system combines all W_{ij}'s to decide the result. The voting result is $R = W_{\max} - \sum_{l \neq \max} W_l$, where $W_l = \sum_{j=1}^{N} a \cdot W_{ij}$, $W_{\max} = \max(W_l)$, and $l \in [1, N]$. When l=i, a=1; otherwise, a=0. If the result satisfies $|R| \geq \eta$, the sample belongs to the target label; otherwise, it is a suspicious malware. Threshold η is chosen based on our experience.

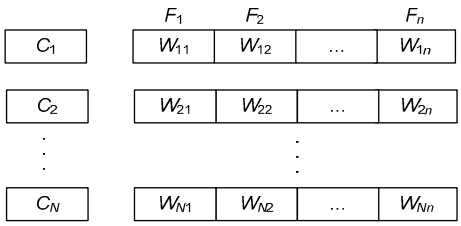

Fig. 8 Weight table, where C_i and F_j represent the classifier and the sample family, respectively, and W_{ij} indicates the ability of classifier C_i to identify family F_j (i=1, 2, …, N; j=1, 2, …, n)

5 Shared nearest neighbor

Although we have designed an effective method that can reduce attributes, there are still thousands of dimensions. The performance of classical clusters is low in the high-dimensional space, because the Euclidean distance fails in the high-dimensional space, and the triangle inequality does not hold (Jarvis and Patrick, 1973). Consider the similarities of the three samples listed in Table 1.

Table 1 Similarities of three samples

Sample	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
P_1	1	1	0	0	0	0	0	0	0	0
P_2	0	0	0	0	0	0	0	0	1	1
P_3	0	0	0	0	1	1	1	1	1	1

We calculate the Euclidean distance: $D(P_1, P_2)$=2, $D(P_2, P_3)$=2, and $D(P_1, P_3)$=$\sqrt{6}$. Two conclusions can be obtained: (1) P_1 and P_3 have the same similarity for P_2; i.e., they have the same distance to P_2. (2) P_1 and P_3 are close to P_2; thus, P_1 is close to P_3. Assume that these three points are malware samples. If the attribute value is not zero, it is contained in this sample. Table 1 demonstrates that P_1 and P_2 have no common properties, while P_2 and P_3 have two shared properties. This explains why the Euclidean distance does not exactly depict the similarity of the samples in a high-dimensional space.

In view of the above problems, we adopt the SNN method, which has good performance in the high-dimensional space. The method was first proposed by Jarvis and Patrick (1973). The similarity of two points is defined in that they jointly have a large neighborhood C which includes at least k points. The

advantage of this method is that it is able to cluster points with different densities. As shown in Fig. 9, circles of different sizes describe the clustering with different densities.

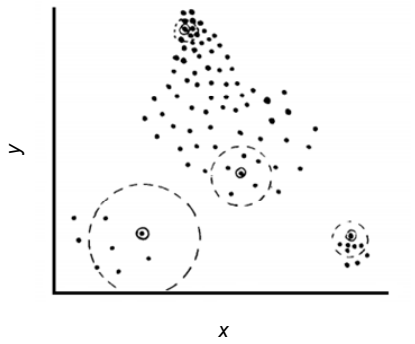

Fig. 9 Clustering with five neighborhoods of each point

The position relationship M between two points is stored in each row of a similarity matrix. $M(A, B)=1$ denotes that point B is the closest to point A. Each row saves only k minimum values, and other values are set to 0. The matrix is used to construct the K-nearest neighbor (K-NN) graph. In Fig. 10, points O and P are noise or outliers, which, however, are not separated by the graph. To address the problem, the link strength is calculated by

$$\mathrm{str}(O,P)=\sum(k+1-m)\cdot(k+1-n). \qquad (5)$$

If the strength of a point is less than the special threshold, all of its edges are removed. In Fig. 10, points O and P are classified as outliers. Ertoz *et al*. (2002) adopted the link strength to select the core points that have larger link strength in each clustering. In each clustering, a point is either one of the core points or connected to the core points.

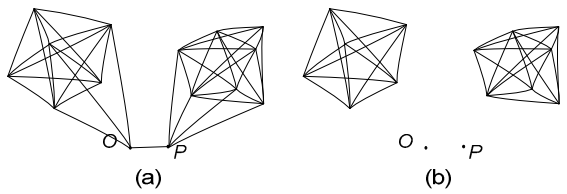

Fig. 10 Near neighbor graph (a) and weighted shared near neighbor graph (b)

The SNN model can be described as follows: (1) calculate the similarity matrix; (2) construct the K-NN graph; (3) compute the link strength and set the threshold to find the noises and outliers with a low strength; (4) select the core points that have a high strength; (5) assign a new point to the clustering, or mark it as an outlier.

6 Experiment

6.1 Dataset

A large amount of malware information has been collected using ESET NOD32, VX Heavens, and Threat Trace Security from our campus network. We capture 21 740 malware instances which belong to nine families (viruses, worms, Trojan horses, backdoors, etc.). They are detected mainly from Windows 7. The data set is divided into a training set and a test set which have 19 740 and 2000 samples, respectively. IDA Pro transforms each malware instance into a binary file and assembly file. The features of each malware program are extracted in the third part of our method. We use the Oracle database, Python 2.7, and Scikit-learn, where Scikit-learn is the machine learning module for Python containing most of the classification algorithms. In the experiments, the RF, K-NN, gradient-boosting (GB), Naïve Bayes (NB), logistic regression (LR), SVM-poly (SP), and decision tree (DT) algorithms are used.

6.2 Classification

6.2.1 n-gram (n=2, 3, …, 9)

n-gram is an efficient method for text feature extraction, where n denotes the length of the feature sequence. The length determines the performance of the algorithm. Therefore, we employ nine different lengths to abstract features, which are applied in seven classification algorithms. As shown in Table 2, the accuracies of all the algorithms show that the one that uses n-gram (n=2, 3) is the best. We find that the performance of classifiers may be worse when employing n-gram (n>4).

6.2.2 Combination of three features

We have an integration feature that combines the gray-scale image, n-gram (n=3), and the frequencies of import functions. Table 3 shows the results of seven classifiers based on different feature extraction methods. RF and GB algorithms have the best

Table 2 The *n*-gram comparisons among different classifiers

Classifier	Accuracy								
	2-gram	3-gram	4-gram	5-gram	6-gram	7-gram	8-gram	9-gram	Average
RF	0.928	0.947	0.936	0.925	0.922	0.883	0.878	0.769	0.899
K-NN	0.883	0.881	0.850	0.811	0.797	0.781	0.742	0.706	0.806
GB	0.925	0.928	0.914	0.911	0.900	0.878	0.858	0.794	0.889
NB	0.725	0.747	0.714	0.719	0.675	0.686	0.669	0.622	0.695
LR	0.900	0.900	0.892	0.856	0.867	0.831	0.786	0.744	0.847
SP	0.819	0.803	0.792	0.733	0.689	0.667	0.639	0.553	0.712
DT	0.738	0.767	0.743	0.645	0.677	0.638	0.629	0.602	0.680
Average	0.845	0.853	0.834	0.800	0.790	0.766	0.743	0.684	0.790

RF: random forest; *K*-NN: *K*-nearest neighbor; GB: gradient-boosting; NB: Naïve Bayes; LR: logistic regression; SP: support vector machine-poly; DT: decision tree

performance. In addition, they represent ensemble learning. RF is made up of many DTs, but its average performance is 7.4% higher than that of DT. Specifically, in applying the 3-gram, its accuracy is 16.7% higher than that of DT. The reason is that ensemble learning is able to improve the classification ability of a single classifier by integrating multiple weak classifiers (Zhou *et al*., 2002). Our experiments prove that the gray-scale image is the Opcode *n*-gram, and the accuracy of the combined method is higher than those of the others. In Fig. 11, we find that the performances of NB and DT achieve the greatest improvement.

Table 3 Accuracy rates of different features based on different classifiers

Classifier	Accuracy			
	Image	3-gram	Combined	Average
RF	0.958	0.936	0.989	0.961
K-NN	0.872	0.831	0.881	0.861
GBC	0.961	0.922	0.975	0.953
NB	0.775	0.772	0.953	0.833
LR	0.928	0.872	0.958	0.919
SP	0.936	0.800	0.969	0.902
DT	0.917	0.769	0.975	0.887
Average	0.907	0.843	0.957	–

RF: random forest; *K*-NN: *K*-nearest neighbor; GB: gradient-boosting; NB: Naïve Bayes; LR: logistic regression; SP: support vector machine-poly; DT: decision tree

To evaluate our method effectively, we present a comparison with that proposed by Nataraj *et al*. (2014). We use the LMgist module (Oliva and Torralba, 2001) based on Matlab R2012b to extract the GIST feature from our gray-scale images. We use seven classifiers to test its accuracy for malware classification. The results show that the average accuracy is 0.914, and the best accuracy is 0.965 based on RF. Although their method is better than the gray-scale method, the accuracy of our combined method is 4.3% higher than that of their method. Opcode *n*-gram is similar to the method proposed by Ding *et al*. (2014). Our experiments show that when *n* is equal to three, our combined method achieves a best accuracy of 0.853.

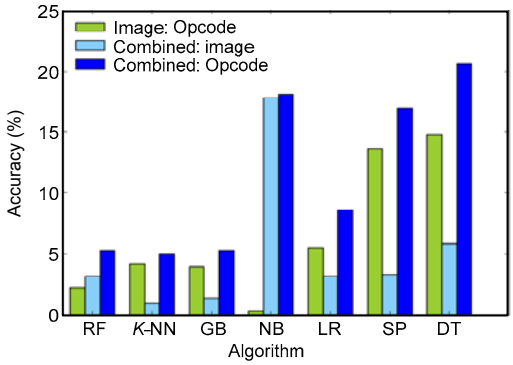

Fig. 11 Improved accuracies of different classifiers
RF: random forest; *K*-NN: *K*-nearest neighbor; GB: gradient-boosting; NB: Naïve Bayes; LR: logistic regression; SP: support vector machine-poly; DT: decision tree

6.3 New malware

In this experiment, our dataset is divided into known and unknown malware. Assume that the unknown malware is regarded as a new family belonging to the Trojan family. Then, the classifiers are trained using the known malware. The test results of these classifiers are shown in Table 4, where each element represents the accuracy of a classifier on a particular family. The expression for accuracy is TP/N_i.

To discover new malware, we construct a decision-making system which is composed of multiple classifiers. Because the performance of classifiers is different, they have different effects on the

result of the decision. Therefore, a scoring method is used to combine the different results of the classifiers.

In the experiment, we use $S=\{S_1, S_2, \ldots, S_{10}, \ldots, S_N\}$ as the sample set, where $S'=\{S_1, S_2, \ldots, S_{10}\}$ belongs to the new malware and the other samples belong to the known families. We train seven models to validate the samples. For example, the labels for S' are {(4, 4, 1, 8, 1, 4, 5, 1, 1, 4), (5, 6, 2, 4, 4, 4, 1, 6, 1, 4), (5, 4, 4, 5, 6, 4, 4, 4, 1, 6), (8, 1, 1, 1, 8, 1, 8, 1, 1, 1), (1, 4, 5, 6, 5, 8, 5, 8, 5, 5), (6, 1, 6, 2, 2, 6, 2, 2, 2, 6), (8, 5, 1, 7, 5, 4, 1, 5, 6, 7)}, where label(S_i)=$\{s_{i1}, s_{i2}, \ldots, s_{i10}\}$. Then, we employ the scoring method to calculate the scoring table of labels. Table 5 is the scoring table, which denotes the weights of classifiers for the decision results.

In the decision phase, we select the target label which has the most votes. If the result of sample S_i satisfies $|R_i|\geq 0.16$, S_i belongs to the target label. If $|R_i|<0.16$, S_i is the suspicious malware. By calculating $|R_i|$, 10 samples are the suspicious malware. We employ the SNN method to cluster the suspicious malware. In Fig. 12, the suspicious samples are considered to be outliers. The samples are too far away from the core points to be clustered. Moreover, there are few outliers to form a new group. Then, these samples are stored in the database.

We reselect 900 samples which contain nine labeled samples. The samples include 810 known samples and 90 new samples which are from the same family as the suspicious samples. The results of the experiment are shown in Fig. 13. A new group appears in the lower right corner of the figure. The group is composed of 78 samples from the unknown malware. There are 12 samples assigned wrongly to other categories. Therefore, the accuracy of this experiment is 0.867.

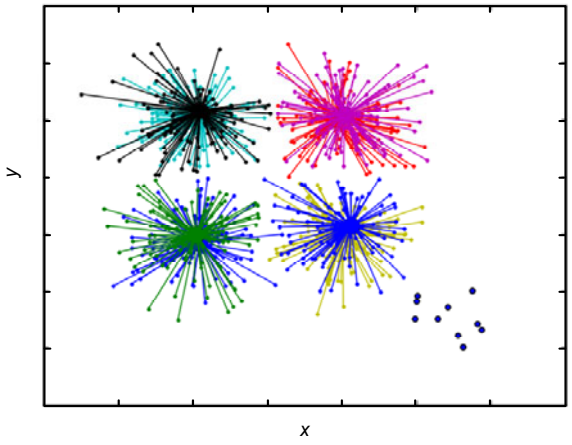

Fig. 12 Unknown malware detection with eight estimated clusters

Table 4 Accuracies of different classifiers on different malware families

Classifier	Accuracy								
	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9
RF	1.00	1.00	1.00	1.00	1.00	0.97	0.98	1.00	0.96
K-NN	0.74	1.00	0.96	0.89	0.63	0.85	1.00	1.00	0.92
GB	1.00	1.00	1.00	0.96	0.94	0.97	0.97	0.96	1.00
NB	1.00	1.00	0.95	1.00	0.86	0.88	0.95	1.00	0.95
LR	0.97	0.98	0.98	1.00	0.89	0.86	1.00	0.97	0.97
SP	1.00	1.00	1.00	0.96	0.93	0.89	1.00	1.00	0.97
DT	1.00	1.00	1.00	0.96	0.90	0.87	1.00	1.00	0.98

Table 5 Scoring table

Classifier	Accuracy									
	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
RF	0.149	0.149	0.146	0.149	0.150	0.150	0.152	0.150	0.154	0.151
K-NN	0.126	0.127	0.146	0.133	0.134	0.133	0.112	0.126	0.113	0.134
GB	0.144	0.143	0.140	0.142	0.146	0.144	0.146	0.144	0.154	0.146
NB	0.140	0.149	0.146	0.149	0.150	0.150	0.152	0.150	0.154	0.150
LR	0.146	0.149	0.130	0.129	0.134	0.146	0.134	0.145	0.137	0.134
SP	0.149	0.149	0.146	0.149	0.150	0.133	0.152	0.150	0.154	0.134
DT	0.146	0.134	0.146	0.149	0.136	0.144	0.152	0.135	0.134	0.151

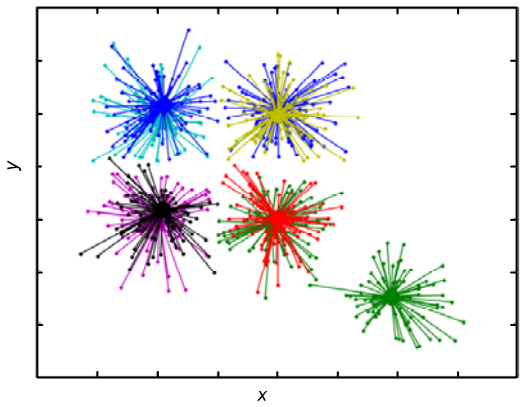

Fig. 13 Unknown malware clustering with nine estimated clusters

7 Conclusions

The system is composed mainly of three parts: data processing, decision making, and clustering. The system can detect accurately not only known malware, but also unknown malware. In the data processing phase, we presented a combined feature extraction method, which is based on a gray-scale image, Opcode *n*-gram, and the import function. The method uses the texture of the malware to effectively describe the features of the different programs. In the decision-making system, we proposed a decision-making system applied to find the suspicious malware. Due to the high dimensionality of the malware, SNN is used to cluster the samples. Finally, experiment results showed that our system not only improves the accuracy of malware classification effectively, but also discovers new malware effectively.

References

Annachhatre, C., Austin, T.H., Stamp, M., 2015. Hidden Markov models for malware classification. *J. Comput. Virol. Hack. Tech.*, **11**(2):59-73. https://doi.org/10.1007/s11416-014-0215-x

Cheng, J.Y.C., Tsai, T.S., Yang, C.S., 2013. An information retrieval approach for malware classification based on Windows API calls. Int. Conf. on Machine Learning and Cybernetics, p.1678-1683. https://doi.org/10.1109/ICMLC.2013.6890868

Damodaran, A., di Troia, F., Visaggio, C.A., *et al*., 2017. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hack. Tech.*, **13**(1): 1-12. https://doi.org/10.1007/s11416-015-0261-z

Ding, Y.X., Dai, W., Yan, S.L., *et al*., 2014. Control flow-based Opcode behavior analysis for malware detection. *Comput. Secur.*, **44**:65-74. https://doi.org/10.1016/j.cose.2014.04.003

Egele, M., Scholte, T., Kirda, E., *et al*., 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, **44**(2): Article 6. https://doi.org/10.1145/2089125.2089126

Ertoz, L., Steinbach, M., Kumar, V., 2002. A new shared nearest neighbor clustering algorithm and its applications. Workshop on Clustering High Dimensional Data and Its Applications at the 2nd SIAM Int. Conf. on Data Mining, p.105-115.

Gandotra, E., Bansal, D., Sofat, S., 2014. Malware analysis and classification: a survey. *J. Inform. Secur.*, **5**(2):44440. https://doi.org/10.4236/jis.2014.52006

Han, K.S., Lim, J.H., Im, E.G., 2013. Malware analysis method using visualization of binary files. Proc. on Research in Adaptive and Convergent Systems, p.317-321. https://doi.org/10.1145/2513228.2513294

Hu, Q.H., Yu, D.R., Xie, Z.X., *et al*., 2007. EROS: ensemble rough subspaces. *Patt. Recogn.*, **40**(12):3728-3739. https://doi.org/10.1016/j.patcog.2007.04.022

Islam, R., Tian, R.H., Batten, L.M., *et al*., 2013. Classification of malware based on integrated static and dynamic features. *J. Netw. Comput. Appl.*, **36**(2):646-656. https://doi.org/10.1016/j.jnca.2012.10.004

Iwamoto, K., Wasaki, K., 2012. Malware classification based on extracted API sequences using static analysis. Proc. Asian Internet Engineering Conf., p.31-38. https://doi.org/10.1145/2402599.2402604

Jain, S., Meena, Y.K., 2011. Byte level *n*-gram analysis for malware detection. *In*: Venugopal, K.R., Patnaik, L.M. (Eds.), Computer Networks and Intelligent Computing. Springer, Berlin, p.51-59. https://doi.org/10.1007/978-3-642-22786-8_6

Jarvis, R.A., Patrick, E.A., 1973. Clustering using a similarity measure based on shared near neighbors. *IEEE Trans. Comput.*, **C-22**(11):1025-1034. https://doi.org/10.1109/T-C.1973.223640

Jolliffe, I.T., 2002. Principal Component Analysis (2nd Ed.). Springer, New York. https://doi.org/10.1007/b98835

Kancherla, K., Mukkamala, S., 2013. Image visualization based malware detection. IEEE Symp. on Computational Intelligence in Cyber Security, p.40-44. https://doi.org/10.1109/CICYBS.2013.6597204

Kapoor, A., Dhavale, S., 2016. Control flow graph based multiclass malware detection using bi-normal separation. *Defen. Sci. J.*, **66**(2):138-145. https://doi.org/10.14429/dsj.66.9701

Kaspersky Labs, 2015. Security Bulletin 2015. https://securelist.com/files/2015/12/KSB_2015_Statistics_FINAL_EN.pdf

Kinable, J., Kostakis, O., 2011. Malware classification based on call graph clustering. *J. Comput. Virol.*, **7**(4):233-245. https://doi.org/10.1007/s11416-011-0151-y

Kong, D.G., Yan, G.H., 2013. Discriminant malware distance learning on structural information for automated malware

classification. Proc. 19th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.1357-1365. https://doi.org/10.1145/2487575.2488219

Lee, J., Jeong, K., Lee, H., 2010. Detecting metamorphic malwares using code graphs. Proc. ACM Symp. on Applied Computing, p.1970-1977. https://doi.org/10.1145/1774088.1774505

Lin, C.T., Wang, N.J., Xiao, H., *et al*., 2015. Feature selection and extraction for malware classification. *J. Inform. Sci. Eng*., **31**(3):965-992. https://doi.org/10.6688/JISE.2015.31.3.11

Lin, D., Stamp, M., 2011. Hunting for undetectable metamorphic viruses. *J. Comput. Virol*., **7**(3):201-214. https://doi.org/10.1007/s11416-010-0148-y

Liu, X.W., Wang, L., Huang, G.B., *et al*., 2015. Multiple kernel extreme learning machine. *Neurocomputing*, **149**: 253-264. https://doi.org/10.1016/j.neucom.2013.09.072

Musale, M., Austin, T.H., Stamp, M., 2015. Hunting for metamorphic JavaScript malware. *J. Comput. Virol. Hack. Tech*., **11**(2):89-102. https://doi.org/10.1007/s11416-014-0225-8

Nataraj, L., Karthikeyan, S., Jacob, G., *et al*., 2014. Malware images: visualization and automatic classification. Proc. 8th Int. Symp. on Visualization for Cyber Security. https://doi.org/10.1145/2016904.2016908

Oliva, A., Torralba, A., 2001. Modeling the shape of the scene: a holistic representation of the spatial envelope. *Int. J. Comput. Vis.*, **42**(3):145-175. https://doi.org/10.1023/A:1011139631724

Pascanu, R., Stokes, J.W., Sanossian, H., *et al*., 2015. Malware classification with recurrent networks. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, p.1916-1920. https://doi.org/10.1109/ICASSP.2015.7178304

Roundy, K.A., Miller, B.P., 2010. Hybrid analysis and control of malware. *In*: Jha, S., Sommer, R., Kreibich, C. (Eds.), Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, p.317-338. https://doi.org/10.1007/978-3-642-15512-3_17

Russo, A., Sabelfeld, A., 2010. Dynamic vs. static flow-sensitive security analysis. 23rd IEEE Computer Security Foundations Symp., p.186-199. https://doi.org/10.1109/CSF.2010.20

Salton, G., McGill, M.J., 1986. Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, USA.

Shabtai, A., Moskovitch, R., Elovici, Y., *et al*., 2009. Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. *Inform. Secur. Tech. Rep*., **14**(1):16-29. https://doi.org/10.1016/j.istr.2009.03.003

Tao, H., Ma, X., Qiao, M., 2013. Subspace selective ensemble algorithm based on feature clustering. *J. Comput*., **8**(2): 509-516.

Tian, R.H., Batten, L., Islam, R., *et al*., 2009. An automated classification system based on the strings of Trojan and virus families. 4th Int. Conf. on Malicious and Unwanted Software, p.23-30. https://doi.org/10.1109/MALWARE.2009.5403021

Tian, R.H., Islam, R., Batten, L., *et al*., 2010. Differentiating malware from cleanware using behavioural analysis. 5th Int. Conf. on Malicious and Unwanted Software, p.23-30. https://doi.org/10.1109/MALWARE.2010.5665796

Tsyganok, K., Tumoyan, E., Babenko, L., *et al*., 2012. Classification of polymorphic and metamorphic malware samples based on their behavior. Proc. 5th Int. Conf. on Security of Information and Networks, p.111-116. https://doi.org/10.1145/2388576.2388591

Wong, W., Stamp, M., 2006. Hunting for metamorphic engines. *J. Comput. Virol*., **2**(3):211-229. https://doi.org/10.1007/s11416-006-0028-7

Yan, G.H., Brown, N., Kong, D.G., 2013. Exploring discriminatory features for automated malware classification. *In*: Rieck, K., Stewin, P., Seifert, J.P. (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment. Springer Berlin Heidelberg, p.41-61. https://doi.org/10.1007/978-3-642-39235-1_3

Yao, W., Chen, X.Q., Zhao, Y., *et al*., 2012. Concurrent subspace width optimization method for RBF neural network modeling. *IEEE Trans. Neur. Netw. Learn. Syst*., **23**(2): 247-259. https://doi.org/10.1109/TNNLS.2011.2178560

Ye, Y.F., Li, T., Chen, Y., *et al*., 2010. Automatic malware categorization using cluster ensemble. Proc. 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, p.95-104. https://doi.org/10.1145/1835804.1835820

Yu, Y., Wang, H.M., Yin, G., *et al*., 2016. Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? *Inform. Softw. Technol*., **74**:204-218. https://doi.org/10.1016/j.infsof.2016.01.004

Zhou, Z.H., Wu, J.X., Tang, W., 2002. Ensembling neural networks: many could be better than all. *Artif. Intell*., **137**(1-2):239-263. https://doi.org/10.1016/S0004-3702(02)00190-X