

Malware Classification Using N-grams Sequential Pattern Features

¹Ohm Sornil and ²Chatchai Liangboonprakong

¹*Department of Computer Science, National Institute of Development Administration,
Bangkok, Thailand, osornil@as.nida.ac.th*

²*Department of Computer Science, Suan Sunandha Rajabhat University,
Bangkok, Thailand, chatcchai.li@ssru.ac.th*

Abstract

Malware family identification is a complex process involving extraction of distinctive characteristics from a set of malware samples. Malware authors employ various techniques to prevent the identification of unique characteristics of their programs, such as encryption and obfuscation. In this paper, we present n-gram based sequential features extracted from content of the files. N-grams are extracted from malicious program files; sequential n-gram patterns are determined; pattern statistics are calculated and reduced by the sequential floating forward selection method; and a classifier is used to determine the family of malware. Three classification models which are C4.5, multilayer perceptron, and support vector machine are studied. Experimental results on a standard malware test collection show that the proposed method performs well, with the classification accuracy of 96.64%.

Keywords: *Malware Classification, N-Gram, Sequential Pattern, Sequential Floating Forward Selection, C4.5, Multilayer Perceptron, Support Vector Machine.*

1. Introduction

A common method of launching computer attacks is by means of malware such as backdoors, Trojan horses, viruses and worms, which can cause severe damages to security and privacy of computer systems and networks worldwide. Malware has evolved into a powerful instrument for illegal commercial activities, and a significant effort is made by its authors to thwart detection by anti-malware products.

General analysis techniques for detecting malware are commonly classified into dynamic and static approaches. In dynamic analysis (also known as behavioral analysis), detection of malware relies on information that is collected from the operating system at runtime (i.e., during the execution of the program) such as system calls, network and file accesses [19]. This approach has several disadvantages. Firstly, it is difficult to simulate suitable conditions for malicious functions of a program, such as the vulnerable applications that the malware will be activated. Secondly, it is not clear what the required period of time is needed to observe the malicious activities of a program. In static analysis, information about a program or its expected behaviors employs explicit and implicit observations in its binary code. The main advantage of static analysis is its ability to examine a suspected file without actually executing it and thus provide rapid classification [22].

Our goal in this research is to introduce a novel set of features for effective classification of malware families. The features are based on n-gram sequential patterns, extracted from disassembling files. Three classification models are explored with the proposed features which consist of C4.5, multilayer perceptron, and support vector machine. Experimental evaluations on a standard malware data collection are performed to evaluate the proposed technique.

2. Related Work

Several techniques have been studied in the past for malware detection. Cohen [8], and Chess and White [5] use sandboxing to detect viruses. They show that in general the problem of virus detection is undecidable. Christodorescu and Jha [6] detect malicious code in executable files. Their implementation, called SAFE, handles most common types of obfuscations used by malware writers, such as insertion of NOPs between instructions, to evade detection.

In [7], Christodorescu et al. exploit semantic heuristics to detect obfuscated malware. Although, their approach works well for obfuscated malicious programs, the time taken (over a minute to classify) by their approach makes it impractical for use in commercial antivirus scanners. Kruegel et al. [14] use control flow graph information and statistical methods for disassembling obfuscated executables. Bergeron et al. [2] consider critical API calls and security policies to test for the presence of malicious code. Their approach does not work for obfuscated malicious executables. Zhang et al. [26] use fuzzy pattern recognition to detect unknown malicious code, however, the approach does not handle obfuscated program binaries. Martignoni et al. [16] use real-time program monitoring to detect obfuscation in memory. Their implementation OmniUnpack detects obfuscation for both known and unknown packers. Zhang and Reeves [27] identifies patterns of system or library functions called by a malware sample to detect their metamorphic versions. Bilar [3] uses statistical structures such as opcode frequency distribution and graph structure fingerprints to detect malicious programs.

An approach to represent malware content is the use of n-grams which are substrings of a larger string with length n . N-grams exist in constant malware parts although obfuscation techniques are used. Representations of malware by using n-gram profiles have been presented in the literature, see for example [1], [11], [21], and [25]. In these studies some promising results towards malware detection are presented. However, malware domain has been evolving due to survivability requirements. Malware has to evade anti-virus scanners to perform its functions. Obfuscation techniques have been developed in order to avoid detection by anti-virus scanners. And these techniques affect n-gram features in the binary form of malware used by previous work. Similar methodologies have been used in source authorship, information retrieval, and natural language processing [4][9]. Abou-Assaleh et al. [1] contribute to the ongoing research while using common n-gram profiles. The K nearest neighbor algorithm with $k=1$ is used. A feature set is constituted by using n-grams and occurrence frequency. Tests have been done with different values of n (ranging from 1 to 10) and frequencies (ranging from 20 to 5,000). The data set used in their experiments comprises 25 malware and 40 benign files. The test results show 98% of success. Using the data in [1], the accuracy slightly drops to around 94%. Kolter and Maloof [11] use 4-grams as features and select top 500 n-grams through information gain measure. They use instance based learners, TF-IDF, naïve Bayes, support vector machines, and decision trees, and also boost the learners. The boosted decision tree outperforms all other and gives promising results, such as ROC curve of 0.996. Walenstein et al. [24] explore the use of n-grams and unordered n-perms to disassembly. This has the benefit of using features from specific malware, however, because all n-perms of a file are considered, there is no opportunity to select those features that may distinguish a family in question from other similarly constructed files.

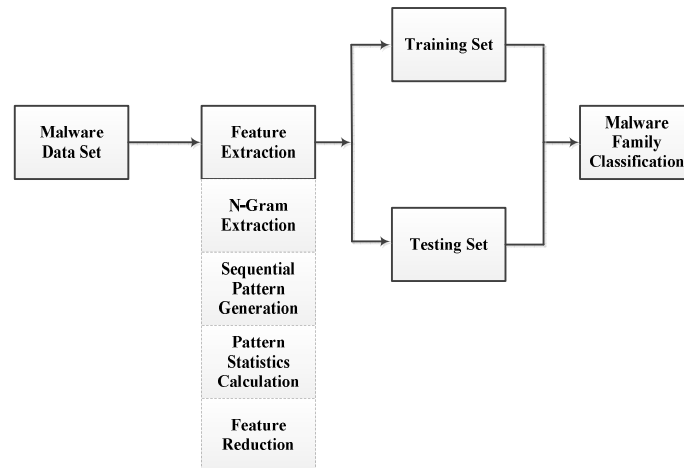


Figure 1. Architecture of the proposed method.

3. Proposed Methodology

In this section, the proposed malware detection process (as shown in Figure 1) is described.

3.1. Feature Extraction

Features are created in 4 steps: n-gram extraction, sequential pattern extraction, pattern statistics calculation, and feature reduction.

3.1.1. N-Grams Extraction

An n-gram is an n-character slice of a longer string. To extract n-grams, first IDA-Pro [10], a tool that disassembles files, is used to extract contents of a file into a long string of hexadecimals. The string is then processed into a set of overlapping n-grams. In our study, we explore n-grams of several different lengths. The kfgm tool [12] is employed to generate n-gram slices. In the experiments, our tests are run with $n=1$, $n=2$, $n=3$ and $n=4$. The resulting n-gram representation is shown in Figure 2.

Malware	N-Grams					
mw_1	t_1	t_2	t_3	t_9	.	.
mw_2	t_3	t_4	t_6	t_8	.	.
mw_3	t_3	t_4	t_6	t_7	.	.
mw_4	t_3	t_4	t_6	t_8	.	.
mw_5	t_1	t_2	t_4	t_6	.	.
mw_6	t_1	t_2	t_4	t_9	.	.
mw_7	t_2	t_5	t_6	t_7	.	.
mw_8	t_2	t_5	t_6	t_7	.	.
mw_9	t_3	t_4	t_5	t_8	.	.
...						
mw_n	t_i	t_k

Figure 2. The n-gram representations of malware.

3.1.2. Sequential Pattern Extraction

There are a very large number of n-grams, but they lack order information necessary to capture characteristics of a program. This information can be provided by sequential patterns. A sequential pattern extraction technique [28] is used to generate frequently occurred sequences of n-grams to represent the data and reduce response time. After being processed, malware is represented by a set of frequent patterns.

Let $T = \{t_1, t_2, \dots, t_m\}$ be a set of terms (n-grams) which are resulted from n-gram extraction. Given X be a set of terms (*termset*) in malware m , $coverset(X)$ denotes the covering set of X for m , which includes all malware file $mw \in F(m)$, where $X \subseteq mw$, i.e., $coverset(X) = \{mw | mw \in F(m), X \subseteq mw\}$, the absolute support of X is the number of occurrences of X in $F(m)$: $sup_a(X) = |coverset(X)|$, the relative support of X is the fraction of the malware file that contains the pattern $sup_r(X) = |coverset(X)| / |F(m)|$. The relationships between frequent patterns and covering sets are shown in Figure 3. A termset X called a frequent pattern if $sup_a \geq min_sup$, a minimum support.

Frequent Pattern	Covering Set
$\{t_3, t_4, t_6\}$	$\{mw_2, mw_3, mw_4\}$
$\{t_3, t_4, t_5\}$	$\{mw_4, mw_9\}$
$\{t_1, t_2, t_4\}$	$\{mw_5, mw_6\}$
$\{t_2, t_5, t_6, t_7\}$	$\{mw_7, mw_8\}$
$\{t_3, t_4, t_6, t_8\}$	$\{mw_2, mw_4\}$
$\{t_3, t_4\}$	$\{mw_2, mw_3, mw_4\}$
$\{t_4, t_6\}$	$\{mw_2, mw_3, mw_4\}$
...	
$\{t_i, \dots, t_k\}$	$\{mw_n, \dots, mw_m\}$

Figure 3. Frequent patterns and covering sets.

A sequential pattern is an ordered list of terms

$$S = \langle t_1, \dots, t_r \rangle : (t_i \in T)$$

$$sup_a \geq min_sup$$

A sequence $s_1 = \langle x_1, \dots, x_i \rangle$ is a sub-sequence of another sequence $s_2 = \langle y_1, \dots, y_j \rangle$ denoted by $s_1 \subseteq s_2$, if $\exists j_1, \dots, j_y$ such that $1 \leq j_1 < j_2 < \dots < j_i < j_y$ and $x_1 = y_{j_1}, x_2 = y_{j_2}, \dots, x_i = y_{j_i}$. Given $s_1 \subseteq s_2$, we usually say s_1 is a sub-pattern of s_2 , and s_2 is a super-pattern of s_1 . To simplify the explanation, we refer to sequential patterns as patterns.

3.1.3. Pattern Statistics Calculation

After n-gram patterns are generated, they will be organized in the following fashion:

$$mw_i = \langle (s_{i1}, w_{i1}), (s_{i2}, w_{i2}), \dots, (s_{im}, w_{im}) \rangle$$

$$Y = \{mw_1, mw_2, \dots, mw_n\}$$

where s_{ij} in pair (s_{ij}, w_{ij}) denotes a pattern s_j in malware mw_i , and w_{ij} is s_j 's significance weight in mw_i , thus the results are a set of malware vectors. In each malware $mw_i \in Y$, the pattern significance can be calculated using term frequency-inverse document frequency (TF-IDF) weighting, where a term represents an n-gram sequential pattern, and a document represents a record of malware. A TF-IDF weight can be calculated as follows:

$$w_{ij} = tf(s_{ij}) \times idf(s_i)$$

As a result, TF-IDF filters out common n-gram sequential patterns by giving low weights to patterns that appear frequently in the data set.

3.1.4. Feature Reduction

There can be a large number of patterns. Although all of these patterns constitute the inputs of a classifier, they have different impacts to the classification performance. Some patterns may not increase the discriminative power of the classification among pattern classes. Vice versa some patterns may be highly correlated, and some may even be irrelevant for a specific classification. The sequential floating forward selection (SFFS) procedure [17] is thus applied to find a minimum feature set. It consists of applying after each forward step a number of backward steps as long as the resulting subsets are better than the previously evaluated ones at that level. The SFFS method can be described, as follows:

Input: $Y = \{\vec{d}_i \mid j = 1, \dots, n\}$

Output: $X_k = \{x_i \mid j = 1, \dots, k, x_j \in Y\}, k = 0, 1, \dots, n$

Initialize feature set: $Y_0 = \{\phi\}; m = 0; J(m) = 0$

Step 1: Find the best feature and update Y_m .

$$x^+ = \arg \max [J(Y_m - x)]; x \notin Y_m$$

$$Y_m = Y_m + x^+; m = m + 1$$

Step 2 : Find the worst feature

$$x^- = \arg \max [J(Y_m - x)]; x \notin Y_m$$

If $J(Y_m - x^-) > J(Y_m)$ then $Y_{m+1} = Y_m - x^-; m = m + 1$.

Go to Step 1

Else go to Step 2

where x_i is a candidate feature (pattern), Y_m is the set of selected features (patterns) of size m , and $J(\cdot)$ is the criterion function which is conditional mutual information in this research.

3.2. Malware Family Classification

A classification model accepts a feature vector and returns the family of the malware. Three learning algorithms are studied in this research, which consist of C4.5, multilayer perceptron, and support vector machine. They are available in KNIME [13], a data mining software. The data set is randomly split into two partitions: 80% for training and 20% for testing.

3.2.1. C4.5 Decision Tree

The C4.5 decision tree [18] is a powerful and popular tool for classification. The algorithm uses gain ratio as the impurity measure for split calculation which can be calculated as:

$$\text{Information Gain} = I(\text{parent}) - \sum_{j=1}^k N(v_j) / N * I(v_j)$$

$$\text{Split Info} = - \sum_{i=1}^k P(v_i) \log_2 P(v_i)$$

$$\text{Gain Ratio} = \text{Information Gain} / \text{Split Info}$$

where $N(v_j)$ is the number of records with value v_j , N is the total number of records, and k is the number of values for the variable. A decision tree has three main components: nodes, arcs and leaves. Each node is labeled with a feature attribute which is the most informative among the attributes not yet considered in the path from the root. Each arc out of a node is labeled with a feature value for the node, and each leaf is labeled with a category or class. A decision tree can then be used to classify a data point by starting at the root of the tree and moving through it until a leaf node is reached. The leaf node would then provide the classification of the data point.

3.2.2. Artificial Neural Network

For artificial neural network, RProp MLP implementation is used in this study [20]. The RProp algorithm is a learning algorithm for multilayer feedforward networks. The following pseudo-code fragment shows the kernel of the RPROP adaptation and learning process.

For all weights and biases {

If $(\partial E / \partial w_{ij}(t-1) * \partial E / \partial w_{ij}(t) > 0)$ then {

$$\Delta_{ij}(t) = \text{minimum}(\Delta_{ij}(t-1) * n^+, \Delta_{max})$$

$$\begin{aligned}
 & \Delta w_{ij}(t) = -\text{sign}(\partial E / \partial w_{ij}(t)) * \Delta_{ij}(t) \\
 & w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \\
 & \} \\
 & \text{Else if } (\partial E / \partial w_{ij}(t-1) * \partial E / \partial w_{ij}(t) < 0) \text{ then } \{ \\
 & \quad \Delta_{ij}(t) = \text{maximum}(\Delta_{ij}(t-1) * n^-, \Delta_{min}) \\
 & \quad \Delta w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1) \\
 & \quad \partial w_{ij}(t) = 0 \\
 & \} \\
 & \text{Else If } (\partial E / \partial w_{ij}(t-1) * \partial w_{ij}(t) = 0) \text{ then } \{ \\
 & \quad \Delta w_{ij}(t) = -\text{sign}(\partial E / \partial w_{ij}(t)) * \Delta_{ij}(t) \\
 & \quad w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \\
 & \} \\
 & \}
 \end{aligned}$$

where E is the error function, and w_{ij} is the weight of the connection between nodes i and j . The minimum and maximum operators are supposed to deliver the minimum and maximum of two numbers, respectively; the sign operator returns +1 if the argument is positive, -1 if the argument is negative, and 0 otherwise. To overcome the inherent disadvantages of the pure gradient-descent, RPROP performs a local adaptation of the weight updates according to the behavior of the error function. In substantial difference to other adaptive techniques, the effect of the RPROP adaptation process is not blurred by the unforeseeable influence of the size of the derivative but only dependent on the temporal behavior of its sign. This leads to an efficient and transparent adaptation process.

3.2.3. Support Vector Machine

A support vector machine (SVM) is a supervised learning technique suitable for solving classification problems with high dimensional feature space. In this study, we use the LIBSVM implementation [15] with a polynomial kernel function to train SVM. Although the basic technique is conceived for binary classification, several methods for multi-class problems have been developed. Being a supervised method, it relies on two phases: during the training phase, the algorithm acquires knowledge about the classes by examining the training set that describes them. During the testing phase, a classification mechanism examines the test set and associates its data to the classes that are available. The target of the algorithm is the estimation of boundaries between classes. Given training vectors $x_i \in R^n$, $i = 1, 2, \dots, l$ in two classes, and a vector $y \in R^l$ such that each $y_i \in \{+1, -1\}$, an SVM for non-separable data considers the following optimization problem:

$$\begin{aligned}
 & \min \frac{1}{2} * w^T w + C \sum_{i=1}^l \alpha_i y_i (w^T K(s_i, x) + b), \\
 & \text{subject to } \alpha_i \geq 0, i = 1, 2, \dots, l
 \end{aligned}$$

In the objective function, w is perpendicular to the hyperplane that separates the positive and negative points, C is a parameter that is used to cost the α_i , $K(s_i, x)$ is a non-linear kernel that maps the input data to another (possibly infinite dimensional) Euclidean space, and s_i are the points called the support vectors that maximize the separation between the positive and negative. Default settings are chosen for all other parameters.

4. Experimental Results

The VX Heavens virus collection [23] database is used as the dataset for evaluation. In our study, we classify malware into 10 families. Backdoors contains one family with 2,014 files. Trojan horses consists of 4 families which are Trojan.BAT (834 files), Trojan.DOS (858 files), Trojan-

Downloader.win32 (624 files), and Trojan.win32 (818 files). Viruses contains 3 families which are Virus.MSWord (712 files), Virus.DOS (984 files), and Virus.BAT (590 files). Worms consists of 2 families which are Worm.win32 (1,907 files) and Email-worm.win32 (2,854 files). Altogether, there are 10 malware families with a total of 12,199 files.

Table 1. Feature characteristics

N	Feature Characteristics		
	<i>N-Grams</i>	<i>Sequential Patterns</i>	<i>Final Features</i>
1	256	2,169	1,356
2	35,491	22,313	6,714
3	104,213	55,023	10,482
4	218,469	87,285	14,908

The dataset is divided into two subsets 80% for training and 20% for testing. Three different classification models are studied.

Table 2. Malware classification performance

N-GRAMS	Accuracy		
	<i>C4.5</i>	<i>ANN (RProp MLP)</i>	<i>SVM</i>
1-gram	71.83%	72.29%	75.44%
2-gram	82.10%	82.85%	83.91%
3-gram	88.74%	84.82%	92.96%
4-gram	91.25%	88.31%	96.64%

Table 1 shows the numbers of n-grams, sequential patterns, and final features while varying the values of n from 1 to 4 since from experiments the value of n greater than 4 does not yield significantly different results. We can see that a higher value of n yields more n-grams; more n-gram yields more sequential patterns, thus a larger set of final features after feature reduction. The final feature sets consist of: 1,356 patterns for 1-gram, 6,714 patterns for 2-gram, 10,482 patterns for 3-gram, and 14,908 patterns for 4-gram. These features are then used in the model. For the classification process, we have 10 malware families or 10 classes to be determined. C4.5, ANN, and SVM are tested with each n-gram size. The classification results are shown in Table 2.

We can observe that SVM gives the best classification accuracy in every size of n-gram; the receiver operating characteristics (ROC) curve (Figure 4) also shows the higher performance of SVM relative to the other two classifiers. However, the relative performance between C4.5 and ANN is unclear. With $n = 1$ and 2, ANN gives higher accuracy while the opposite is observed with $n = 3$ and 4. The larger n-gram yields the higher accuracy in general to capture unique characteristics of different families in the presence of various obfuscations.

The experimental results for classification of malware families show that the proposed features have the ability to achieve high classification accuracy of 96.64%

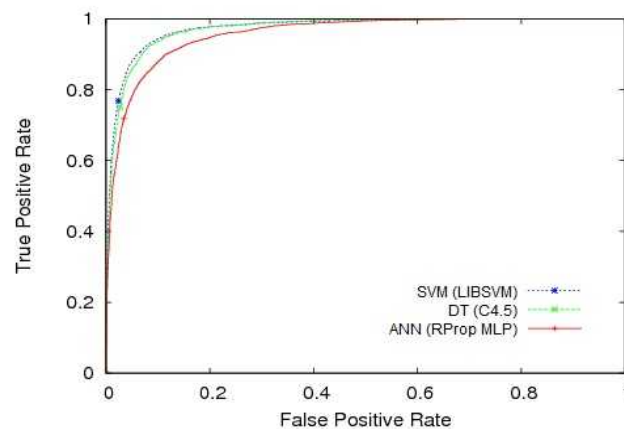


Figure 4. ROC plot for classification.

5. Conclusions

Malware family identification is a complex process involving extraction of distinctive characteristics from a set of malware samples while authors employ various obfuscation techniques to prevent the identification of unique characteristics of their programs. In this paper, we propose n-grams sequential pattern features for classifying malware into 10 families. N-grams are created from the binary contents of files; n-gram sequential patterns are formed and reduced to a minimal set by the sequential floating forward selection procedure. Four different sizes of n-grams ($n = 1, 2, 3$, and 4) are studied; and 3 classification models (C4.5 decision tree, artificial neural network, and support vector machine) are studied. Due to the complexities of malware, the larger n-gram size yields the higher accuracy. The proposed features yield 96.64% in accuracy with 4-gram and support vector machine.

6. References

- [1] T. Abou-Assaleh, V. Keselj, R. Sweidan, "N-gram based detection of new malicious code", In Proceedings of the 28th Annual International Computer Software and Applications Conference, IEEE Computer Society, pp.41-42, 2004.
- [2] J. Bergeron, M. Debbabi, J. Desharnais, M.M Erhoui, Y. Lavoie, N. Tawbi, "Static Detection of Malicious Code in Executable Programs", In Symposium on Requirements Engineering for Information Security, 2001.
- [3] D. Bilar, "Statistical Structures: Tolerant Fingerprinting for Classification and Analysis", Las Vegas, NV. Blackhat Briefings USA, 2006.
- [4] S. Burrows, S. M. Tahaghoghi, "Source code authorship attribution using n-grams", In Proceedings of the 12th Australasian Document Computing Symposium, A. Spink, A. Turpin, and M. Wu, eds. RMIT University, Melbourne, Australia, pp.32-39, 2007.
- [5] D.M. Chess, S. R. White, "An undetectable computer virus", In Proceedings of Virus Bulletin Conference, 2000.
- [6] M. Christodorescu, S. Jha, "Static Analysis of Executables to Detect Malicious Patterns", In Proceedings of the 12th USENIX Security Symp Security, pp.169-186, 2003.
- [7] M. Christodorescu, S. Jha, A. Seshia, D. Song, R. E. Bryant, "Semantics-Aware Malware Detection", In Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 32-46, 2005.
- [8] F. Cohen, "Computer Virus Theory and experiments", Computers and Security, 6, pp. 22-35, 1987.
- [9] G. Frantzeskou, E. Stamatatos, S. Gritzalis, S. Katsikas, "Effective identification of source code authors using byte-level information", In Proceedings of the Twenty-Eighth International Conference on Software Engineering, ACM Press, Shanghai, China, pp.893-896, 2006.
- [10] IDA-Pro Tool, available at <http://www.hex-rays.com>
- [11] J. Z. Kolter, M. A. Maloof, "Learning to Detect Malicious Executable in the Wild", In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, USA, pp.470-478, 2004.
- [12] KfNgram, available at <http://www.kwicfinder.com/kfNgram>.
- [13] KNIME Data Mining Tool, available at <http://www.knime.org>.

- [14] C. Kruegel, W. Robertson, F. Valeur, G. Vigna, "Static disassembly of obfuscated binaries", In Proceedings of USENIX Security, San Diego, CA, pp.255–270, 2007.
- [15] LIBSVM Tool, available at <http://www.csie.ntu.edu.tw/~cjlin/>.
- [16] L. Martignoni, M. Christodorescu, S. Jha, "OmniUnpack: Fast, Generic, and Safe Unpacking of Malware", In 23th Annual Computer Security Applications Conference (ACSAC), Miami Beach, 2007.
- [17] P. Pudil, F. J. Ferri, J. Novovicova, J. Kittler, "Floating search methods for feature selection with nonmonotonic criterion functions", International Conference on Pattern Recognition, 1994, pp. 279-283.
- [18] J. R. Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufman, 1993.
- [19] K. Rieck, T. Holz, P. Dussel, P. Laskov, "Learning and classification of malware behavior", In Conference on Detection of Intrusions and Malware & Vulnerability Assessment Heidelberg, Springer, pp.108-125, 2007.
- [20] H. Riedmiller, M. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm", Proceedings of the IEEE International Conference on Neural Networks (ICNN), pp.586-591, 1993.
- [21] I. Santos, Y. K. Penya, J. Devesa, P. G. Bringas, "N-Grams-Based File Signatures For Malware Detection", In Proceedings of the 11th International Conference on Enterprise Information Systems, Volume AIDSS, pp. 317-320, 2009.
- [22] A. Shabtai, D. Potashnik, Y. Fledel, R. Moskovitch, E. Elovici, "Monitoring analysis and filtering system for purifying network traffic of known and unknown malicious content", Security and Communication Networks, 2010.
- [23] VX Heavens Virus Collection, VX Heavens website, available at <http://vx.netlux.org>
- [24] A. Walenstein, M. Venable, M. Hayes, C. Thompson, A. Lakhota, "Exploiting similarity between variants to defeat malware" In Proceeding of BlackHat 2007 DC Briefings, 2007.
- [25] T. Ying, Z. Pentao, "Immune based computer virus detection approaches", CAAI Transactions on Intelligent Systems, pp.80-94, 2013.
- [26] B. Zhang, J. Yin, J. Hao, "Using Fuzzy Pattern Recognition to Detect Unknown Malicious Executables Code", Fuzzy Systems and Knowledge Discovery. LNCS (LNAI), vol.3613, pp.629–634, 2005.
- [27] Q. Zhang, D. S. Reeves, "MetaAware: Identifying Metamorphic Malware", In Annual Computer Security Applications Conference, 2007.
- [28] N. Zhong, Y. Li, S. T. Wu, "Effective Pattern Discovery for Text Mining" IEEE Transactions on Knowledge and Data Engineering, 2012.