Classification of Malware based on Data Mining Approach

[1]Ankita K Tiwari
[1]Department of Computer Science, IT Systems and Network Security
[1]Gujarat Technological University, India

Abstract— In recent years, the number of malware families/variants has exploded dramatically. Automatic malware classification is becoming an important research area. Using data mining, we identify seven key features within the Microsoft PE file format that can be fed to machine learning algorithms to classify malware. In this paper, resting on the analysis of Windows API execution sequences called by PE files, we develop the Intelligent Malware Detection System (IMDS) using Objective-Oriented Association (OOA) mining based classification. IMDS is an integrated system consisting of three major modules: PE parser, OOA rule generator, and rule based classifier. An OOA_Fast_FP Growth algorithm is adapted to efficiently generate OOA rules for classification. Promising experimental results demonstrate that the accuracy and efficiency of our IMDS system outperform popular anti-virus software such as Norton Antivirus and McAfee Virus Scan, as well as previous data mining based detection systems which employed Naive Bayes, Support Vector Machine (SVM) and Decision Tree techniques.

I. INTRODUCTION

Malware refers to a broad class of malicious software that threatens computer and information systems and networks. Such software may modify, destroy or steal data, may obtain unauthorized access to confidential information and exploit vulnerabilities in applications. Authors of malware often obfuscate the executable components so as to make the malware more difficult to identify. In [2], the authors identify four ways in which this is commonly done. These are: the insertion of dead code that does not change the original code in any way, interchanging the uses of registers, replacing one sequence of instructions with an equivalent one, and permuting instruction sequences in the code without changing the code behavior. In this paper, we do not assume that the malware under analysis has been DE obfuscated. However, we do assume that the malware we consider has been unpacked, that is, it has not been compressed or encrypted in any way.

Still, the traditional approach to analyze malware requires that a human analyst manually performs the tests and extracts the information in order to classify the sample (Moser et al., 2007). Unfortunately, due to the tremendous growth of malicious code, antivirus companies receive everyday thousands of suspicious files that have to be analyzed and classified as malware or benign software. Hence, a reliable and fast automation of the analysis and classification is a crucial point to be able to cope with this threat.

With this scenario in mind, there are two malware analysis approaches: static analysis (Carrera and Erd´elyi, 2004) which is performed without actually executing the file, only observing the binary looking for suspicious patterns, and dynamic analysis (Christodorescu et al., 2007; Rieck et al., 2008) which implies running the sample in an isolated and controlled environment monitoring its behavior.

Our efforts for detecting polymorphic and new, previously unseen malicious executable lead to the Intelligent Malware Detection System (IMDS), which applies Objective-Oriented Association (OOA) mining based classification [10, 14]. The IMDS rests on the analysis of Windows Application Programming Interface (API) execution sequences which recollect the behavior of program code pieces. The associations among the APIs capture the underlying semantics for the data are essential to malware detection. Our malware detection is carried out directly on Windows Portable Executable (PE) code with three major steps: (1) first constructing the API execution sequences by developing a PE parser; (2) followed by extracting OOA rules using OOA Fast FP-Growth algorithm; (3) and finally conducting classification based on the association rules generated in the second step. So far, we have gathered 29580 executable, of which 12214 are referred to as benign executable and 17366 are malicious ones. This executable called 12964 APIs in total. The collected data in our work is significantly larger than those used in previous studies on data mining for malware detection [13, 9, 18]. The experimental results illustrate that the accuracy and efficiency of our IMDS outperform popular anti-virus software such as Norton Antivirus and McAfee Virus Scan, as well as previous systems using data mining approaches such as Naïve Bayes, SVM and Decision Tree.

In summary, our main contributions are: (1) we develop an integrated IMDS system based on analysis of Windows API execution calls. The system consists of three components: PE parser, rule generator and classifier; (2) We adapt existing association based classification techniques to improve the system effectiveness and efficiency; (3)We evaluate our system on a large collection of executable including 12,214 benign samples and 17,366 malicious ones; (4)We provide a comprehensive experimental study on various anti-virus software as well as various data mining techniques for malware detection using our data collection; (5) Our system has been already incorporated into the KingSoft's Antivirus software.

II. RELATED WORK

As mentioned earlier the signature based method , there is still some work to improve in this method [5,18,23,26] and also a few attempts to apply data mining and machine learning techniques, such as Naive Bayes method, support vector machine (SVM) and Decision Tree classifiers, to detect new malicious executable.

Theoretical studies on computer viruses can be found in [34, 35]. Sung et al. [26] developed a signature based malware detection system called SAVE (Static Analyzer of Vicious Executable) which emphasized on detecting polymorphic malware by calculating a similarity measure between the known virus and the suspicious code. The basic idea of this approach is to extract the signatures from the original malware with the hypothesis that all versions of the same malware share a common core signature. Schultz et al. [24] applied Naive Bayes method to detect previously unknown malicious code. The authors downloaded 1,001 benign executable and 3,265 malicious executable from several FTP sites and labeled them by a commercial virus scanner. Decision Tree was studied in [15,30]. In [30], the authors used the same data set described in [24], and worked on a subset of the data which consists of 125 benign programs and 875 malicious codes. Their results also showed that the ROC curve of the Decision Tree method dominated all others.

The detail classification methodologies of Naive Bayes, SVM and Decision Tree are described in Sect. 6. Although results were generally good, it would be interesting to know how these classifiers performed on larger collections of executable. In the field of data mining, frequent patterns found by association mining carry the underlying semantics of the data; therefore, we applied OOA mining technique to extract the characterizing frequent patterns to achieve accurate malware detection.

III. THE SYSTEM ARCHITECTURE

Our IMDS system is performed directly on Windows PE code. The functionality of the PE parser is to generate the Windows API execution calls for each benign/malicious executable. The system consists of three major components: PE parser, OOA rule generator, and malware detection module, as illustrated in Fig. 1.

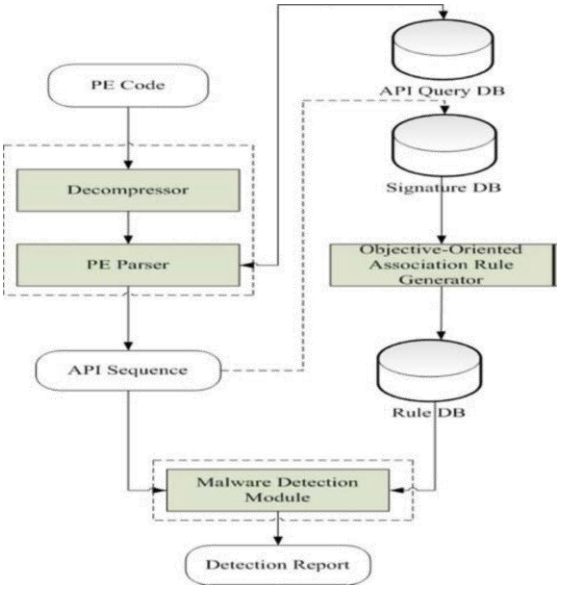

Fig. 1 IMDS system architecture

Some famous viruses such as CIH, CodeRed, CodeBlue, Nimda, Sircam, Killonce, Sobig, and LoveGate all aim at PE files. If a PE file is previously compressed by a third party binary compress tool such as UPX and ASPack Shell or embedded a homemade packer, it needs to be decompressed before being passed to the PE parser and we use the dissembler W32Dasm developed by KingSoft Anti-Virus Laboratory to dissemble the PE code and output the assembly instructions as the input of our PE parser. Through the API query database, the API execution calls generated by the PE parser can be converted to a group of 32-bit global IDs which represents the static execution calls of the corresponding API functions. For example, the API "KERNEL32.DLL, Open- Process" executes the function that returns a handle to an existing process object and it can be encoded as 0x00500E16. To finally determine whether a PE files is malicious or not, we pass the selected API calls together with the rules generated to the malware detection module to perform the association rule based classification.

IV. ANALYSIS OF DATA

Since PE is designed as a common file format for all flavors of Windows operating system, Some famous viruses such as CIH, CodeRed, CodeBlue, Nimda, Sircam, Killonce, Sobig, and LoveGate all aim at PE files. The malicious executables mainly consisted of backdoors, worms, and Trojan horses and all of them were provided by the Anti-virus laboratory of KingSoft Corporation. The benign executables were gathered from the system files of Windows 2000/NT and XP operating system. Since a virus scanner is usually a speed sensitive application, in order to improve the system performance, we developed a PE parser as described in the previous section to construct the API execution calls of PE files instead of using a third party disassemble.

Let us firstly look at the general outline of PE file format. Figure 2 is the general layout of a PE file. All PE files must start with a simple DOS MZ header, thus DOS can verify whether a file is a valid executable or not when the program is running under DOS system. Next to the DOS header, there is a PE header which contains essential information used to load a PE file. The content of a PE file is divided into sections and each section stores data with common attributes.

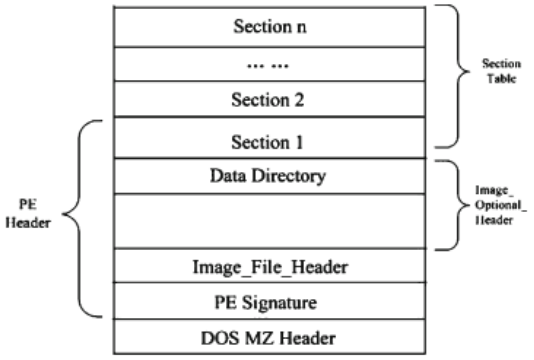

Fig. 2 General layout of a PE file

If the PE file calls another piece of executable code, we call it an import function. For example, the code of a Windows API is always in the correlative Dynamic Linked Library (DLL) file of the "system32" directory. The pointers to the names of the import functions and their resident DLL are recorded in the Imported Address Table (IAT). In order to extract statically the API execution calls of a PE file, our

PE parser performs the following three major steps. First of all, it locates the IAT which contains the pointers to the hints and names of the imported API, and then builds a binary lookup tree from all imported APIs. The second step is to scan the code section(s) to extract the CALL instructions and their target addresses, and then the PE parser searches the target addresses in the binary lookup tree to find the corresponding API. In the final step, we map the API name along with its module name to a 32-bit unique global API ID. For example, the API "MAPI32.MAPIReadMail" is encoded as 0x00600F12.

The implementation of our PE parser involves the following procedures:

A. verifying if the file is a valid PE file;

B. Locating the PE header by examining the DOS header;

C. Obtaining the address of the data directory in Image_Optional_Header;

D. Extracting the value of Virtual Address in the data directory;

E. Finding Image_Import_Descriptor structures using the value of Virtual Address;

F. Checking the RVA value in each Image_Import_ Descriptor structure found in step 5 to locate the arrays which contain the API function names;

G. Extracting API function names.

V. CLASSIFICATION BASED ON OOA MINING

Classification is "the task of learning a target function that maps each feature set to one of the predefined class labels" [28]. For association rule mining, there is no predetermined target. Given a set of transactions in the database, all the rules that satisfied the support and confidence thresholds will be discovered [3]. In our IMDS system, we adapted OOA mining techniques [25] to generate the rules.

A. OOA definitions

In our IMDS system, the goal is to find out how a set of API calls supports the specific objectives: *Obj*1 = *(Group = Malicious)*, and *Obj*2 = *(Group = Benign)*.

- *Definition 1 (support and confidence)*

Let $I = \{I1, \ldots, Im\}$ be an itemset and $I \rightarrow Obj$ *(os%, oc%)* be an association rule in OOA mining. The support and confidence of the rule are defined as:

$os\% = supp(I, Obj) = count(I \cup \{Obj\}, DB) / |DB| \times 100\%$

$oc\% = conf(I, Obj) = count(I \cup \{Obj\}, DB) / count(I, DB) \times 100\%$

where the function $count(I \cup \{Obj\}, DB)$ returns the number of records in the dataset *DB* where $I \cup \{Obj\}$ holds.

- *Definition 2 (OOA frequent itemset)*

Given *mos*% as a user-specified minimum support. *I* is an OOA frequent itemset/pattern in DB if *os*% ≥ *mos*%.

- *Definition 3 (OOA rule)*

Given *moc*% as a user-specified confidence. Let $I = \{I1, \ldots, Im\}$ be an OOA frequent itemset. $I \rightarrow Obj$ *(os%, oc%)* is an OOA rule if *oc*% ≥ *moc*%.

B. OOA_FP-Growth algorithm

OOA mining algorithm called OOA_FP-Growth is Designed based on FP-Growth algorithm [10,11]. This algorithm encodes the dataset using an FP-tree structure and extracts frequent itemsets from it. There are two steps in the OOA_FP-Growth algorithm:

1. Construct the FP-tree:

(a) First of all, select all records satisfying the objective as a sub-dataset.

(b) Scan the sub-dataset to determine the support count of each item; sort the frequent items in decreasing order of their support counts.

(c) Create a set of nodes for each record; form a path to encode the record and increase the frequency count of each node along the path by 1.

(d) Algorithm stops when every record has been mapped onto one of the paths in the FP-tree.

2. Generate frequent itemsets: similar to the FP-Growth algorithm, OOA_FPGrowth generates frequent itemsets by exploring the FP-tree in a bottom-up fashion.

Table 1 Sample dataset

ID	Called API	File type
1	API_1, API_5	Benign
2	API_1, API_3	Malicious
3	$API_1, API_2, API_4, API_5$	Benign
4	$API_1, API_2, API_3, API_4, API_5$	Malicious
5	API_3, API_5	Malicious
6	API_2, API_4	Benign
7	API_2, API_4, API_5	Malicious
8	API_2, API_5	Benign

Now, we use the example data file shown in Table 1 to illustrate the OOA_FP-Growth algorithm. Given that the minimum support count is 2 and the objective is *Obj*1 = *(Group = Malicious)*, we obtain the sub-dataset as shown in Table 2. Based on the sub-dataset we construct the OOA_FPtree as illustrated in Fig. 3. Searching the tree in a bottom-up fashion, we generate the following frequent itemsets: {*API*2, *API*4}, {*API*4, *API*5}, {*API*2, *API*5}, {*API*2, *API*4, *API*5}, {*API*1, *API*3}, {*API*3, *API*5}.

C. OOA_Fast_FP-Growth algorithm

Table 2 Sub-dataset from Table 1

ID	Called API	File type
2	API_1, API_3	Malicious
4	$API_1, API_2, API_3, API_4, API_5$	Malicious
5	API_3, API_5	Malicious
7	API_2, API_4, API_5	Malicious

OOA_FP-Growth generates a huge number of conditional FP-trees recursively, which is time and space consuming. Our malware detection relies on finding frequent patterns from large collections of data, therefore, the efficiency is an essential issue to our system. In our IMDS system, we extend a modified FP-Growth algorithm

proposed in [6] to conduct the OOA mining. This algorithm greatly reduces the costs of processing time and memory space, and we call it OOA_Fast_FP-Growth algorithm.

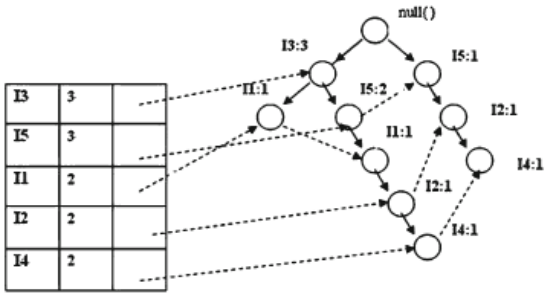

Fig. 3 FP-tree constructed from Table 2

Similar to OOA_FP-Growth algorithm, there are also two steps in OOA_Fast_FP-Growth algorithm: constructing an OOA_Fast_FP-tree and generating frequent patterns from the tree. But the structure of an OOA_Fast_FP-tree is different from that of an OOA_FP-tree in the following way:

(1) The paths of an OOA_Fast_FP-tree are directed, and there is no path from the root to leaves. Thus, fewer pointers are needed and less memory space is required. (2) In an OOA_FP-tree, each node is the name of an item, but in an OOA_Fast_FP-tree, each node is the related order of the item, which is determined by the support count of the item. Based on the sub-dataset shown in Table 2, we construct the OOA_Fast_FP-tree as illustrated in Fig. 4. The support counts from API1 to API5, respectively, are 3, 3, 2, 2, 2 and their related orders are 3, 4, 1, 5, 2.

The rule generation procedure of the algorithm relies on the definition of the constrained subtree. Let ki < …< k2 < k1 be a set of the items' orders and P is a subpath from the root to the node N in the OOA_Fast_FP-tree, we say path P is constrained by the itemset {ki , . . . , k2, k1} if the following two conditions are satisfied: (1) There exists a node M, which is a child node of N, and the orders of ki , . . . , k2, k1 appear in the subpath from node N to M; (2) ki appears in the children of the node N, and k1 appears in the

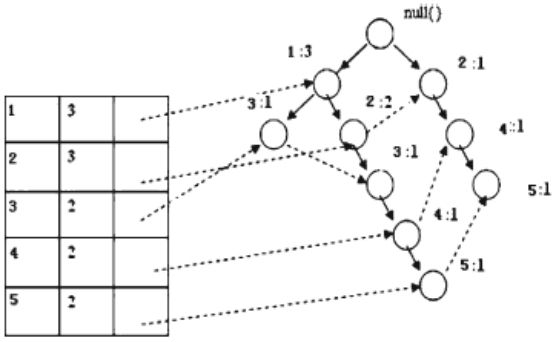

Fig. 4 Fast FP-tree constructed from Table 2

node M. If a subtree consists of all the subpaths which are constrained by {ki , . . . , k2, k1},we call it constrained subtree by {ki , . . . , k2, k1}, and denote it as ST (k1, k2, . . . , ki).

The constrained subtree ST (k1, k2, . . . , ki) is a special subtree of the OOA_Fast_FP-tree, so we need to record the sum of the frequency counts of the nodes with the same order. Then, the constrained subtree ST (k1, k2, . . . , ki) can be obtained by searching the OOA_Fast_FP-tree in a bottomup fashion. The detailed description can be referred in [6].

VI. RESULT AND ANALYSIS

In this paper we will conduct three sets of experiments using the data collected. In the first set of experiments, we evaluate the efficiency of different OOA mining algorithms. The second set of experiments is to compare the abilities to detect polymorphic/ metamorphic and unknown malware of our IMDS system with current widely used anti-virus software. Finally, we compare our IMDS system with other classification based methods. All the experiments are conducted under the environment of Windows 2000 operating system plus Intel P4 1 GHz CPU and 1 GB of RAM.

A. Evaluation of different OOA mining algorithms

In our experiments, we implemented OOA_FP Growth, and OOA_Fast_FP-Growth algorithms under Microsoft Visual C++ environment. By using different support and confidence thresholds, we compare the efficiency of the three algorithms. The results are shown in Table 3. From Table 3, we observe that the time complexity increases exponentially as the minimum support threshold decreases.

Table 3 Running time of different OOA mining algorithms (min)

Experiment	1	2	3	4
Mos	0.355	0.35	0.34	0.294
Moc	0.9	0.95	0.9	0.98
OOA_Apriori	25	260.5	∞	∞
OOA_FP-Growth	8	16.14	60.5	280.2
OOA_Fast_FP-Growth	4.1	7.99	28.8	143.5

Mos and Moc represent the minimum support and minimum confidence for each experiment

B. Comparisons of different anti-virus scanners

In this part, we are examining the abilities of detecting polymorphic malware and unknown malware of our system with popular software tools such as Norton AntiVirus 2007, Dr. Web 2007, McAfee VirusScan 2007 and Kaspersky Anti-Virus 2007.

C. Polymorphic/metamorphic virus detection

For each virus, we apply the encryption and obfuscation techniques described in [26], including flow modification, data segment modification and insertion of dead code, to create a set of polymorphic/metamorphic versions. Then we compare our system with current most widely used anti-virus software.

D. Unknown malware detection

In order to examine the ability of identifying new and previously unknown malware of our IMDS system, we use 1,000 malware for test. These malware are not simple modifications of well known malware and we do not have any information about their nature. They are analyzed by the experts in KingSoft Anti-virus laboratory and their signatures have not been recorded into the virus signature

database. Comparing with other anti-virus software, our IMDS system performs
most accurate detection.

E. Comparisons of different classification methods

In this set of experiments, we compare our system with other classification methods. We randomly select 2,843 executables from our data collection, in which 1,207 files are benign and 1,636 executables are malicious. Then we convert the transactional sample data in our signature database into a relational table, in which each column corresponds to an API and each row is an executable. This transformation makes it easy to apply feature selection methods and other classification approaches.

VII. CONCLUSION

In this paper, we describe our research effort on malware detection based on window API calls. We develop an integrated IMDS system consisting of PE parser, OOArule generator and rule based classifier. First, a PE parser is developed to extract the Windows API execution calls for each Windows portable executable. Then, the OOA_Fast_FP-Growth algorithm is adapted to generate association rules with the objectives of finding malicious and benign executable. This algorithm achieves much higher efficiency than previous OOA mining algorithms. Finally, classification is performed based on the generated rules. Experiments on a large real data collection from anti-virus lab at Kingsoft Corp. demonstrate the strong abilities of our IMDS system to detect polymorphic and new viruses. And the efficiency, accuracy and the scalability of our system outperform other current widely used anti-virus software and other data mining based malware detection methods. Our work results in a real malware detection system, which has been incorporated into the scanning tool of KingSoft's AntiVirus software.

REFERENCES

[1] Adleman, L.: An abstract theory of computer viruses (invited talk). In: CRYPTO '88: Proceedings on Advances in Cryptology, pp. 354–374, New York, NY, USA. Springer, New York (1990)
[2] Agrawal, R., Imielinski, T.:Mining association rules between sets of items in large databases. In: Proceedings of SIGMOD (1993)
[3] Agrawal, R., Srikant, R.: Fast algorithms for association rule mining. In: Proceedings of VLDB-94 (1994)
[4] Cheng, H.,Yan,X., Han, J.,Hsu,C.: Discriminative frequency pattern analysis for effective classification. In: Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE-07) (2007)
[5] Christodorescu,M., Jha, S.: Static analysis of executables to detect malicious patterns. In: Proceedings of the 12th USENIX Security Symposium (2003)
[6] Fan, M., Li, C.: Mining frequent patterns in an fp-tree without conditional fp-tree generation. J. Comput. Res. Dev. **40**, 1216– 1222 (2003)
[7] Filiol, E.: Computer Viruses: from Theory to Applications. Springer, Heidelberg (2005)
[8] Filiol,E.: Malware pattern scanning schemes secure against blackbox analysis. J. Comput. Virol. **2**(1), 35–50 (2006)
[9] Filiol, E., Jacob, G., Liard, M.L.: Evaluation methodology and theoretical model for antiviral behavioural detection strategies. J. Comput. Virol. **3**(1), 27–37 (2007)
[10] Han, J., Kamber, M.: DataMining: Concepts and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2006)
[11] Han, J., Pei, J.,Yin,Y.:Mining frequent patternswithout candidate generation. In: Proceedings of SIGMOD, pp. 1 12, May (2000)
[12] Hsu, C., Lin, C.: A comparison of methods for multiclass support vector machines. IEEE Trans. Neural Netw. **13**, 415–425 (2002)
[13] Jain, A., Duin, R., Mao, J.: Statistical pattern recognition: A review. IEEE Trans. Pattern Anal. Mach. Intell. **22**, 4–37 (2000)
[14] Kephart, J., Arnold, W.: Automatic extraction of computer virus signatures. In: Proceedings of 4th Virus Bulletin International Conference, pp. 178–184 (1994)
[15] Kolter, J., Maloof, M.: Learning to detect malicious executables in the wild. In: Proceedings of KDD'04 (2004)
[16] Kwak, N., Choi, C.: Input feature selection by mutual information based on parzen window. IEEE Trans. Pattern Anal. Mach. Intell. **24**, 1667–1671 (2002)
[17] Langley, P.: Selection of relevant features in machine learning. In: Proceedings of AAAI Fall Symposium (1994)
[18] Lee, T., Mody, J.: Behavioral classification. In: Proceedings of 2006 EICAR Conference (2006)
[19] Liu, B., Hsu,W., Ma, Y.: Integreting classification and association rule mining. In: Proceedings of KDD'98 (1998)
[20] Lo, R., Levitt, K., Olsson, R.: Mcf: A malicious code filter. Comput. Secur. **14**, 541–566 (1995)
[21] McGraw, G., Morrisett, G.: Attacking malicious code: report to the infosec research council. IEEE Softw. **17**(5), 33–41 (2002)
[22] Peng, H., Long, F., Ding, C.: Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. 27 (2005)
[23] Rabek, J., Khazan, R., Lewandowski, S., Cunningham, R.: Detection of injected, dynamically generated, and obfuscated malicious code. In: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 76–82 (2003)
[24] Schultz,M., Eskin, E., Zadok, E.: Data mining methods sfor detection of new malicious executables. In: Security and Privacy, 2001 Proceedings. 2001 IEEE Symposium on 14–16 May, pp. 38–49 (2001)
[25] Shen, Y., Yang, Q., Zhang, Z.: Objective-oriented utility-based association mining. In: Proceedings of IEEE International Conference on Data Mining (2002)
[26] Sung, A., Xu, J., Chavez, P., Mukkamala, S.: Static analyzer of vicious executables (save). In: Proceedings of the 20th Annual Computer Security Applications Conference (2004)

[27] Swets, J., Pickett, R.: Evaluation of Diagnostic System: Methods from SignalDetection Theory. Academic Press, NewYork (1982) 28. Tan, P., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison Wesley, Reading (2005)
[28] Vapnik, V.: The Nature of Statistical Learning Theory. Springer, Heidelberg (1999)
[29] Wang, J., Deng, P., Fan, Y., Jaw, L., Liu, Y.: Virus detection using data mining techniques. In: Proceedings of IEEE International Conference on Data Mining (2003) 31. Witten, H., Frank, E.: Data Mining: Practical Machine Learning Tools with Java Implementations. Morgan Kaufmann, San Francisco (2005)
[30] Xu, J., Sung, A., Chavez, P., Mukkamala, S.: Polymorphic malicious executable sanner by api sequence analysis. In: Proceedings of the International Conference on Hybrid Intelligent Systems (2004)
[31] Ye, Y.,Wang, D., Li, T., Ye, D.: IMDS: Intelligent malware detection system. In: Proccedings of ACM International Conference on Knowlege Discovery and Data Mining (SIGKDD 2007) (2007) .
[32] Yin, X., Han, J.: Cpar: Classification based on predictive association rules. In: Proceedings of 3rd SIAM International Conference on Data Mining (SDM'03), May (2003)
[33] Zuo, Z., Tian Zhou, M.: Some further theoretical results about computer viruses. Comput. J. **47**(6), 627–633 (2004)
[34] Zuo, Z., Zhu, Q.-x., Zhou, M.-t.: On the time complexity of computer viruses. IEEE Trans. Inf. Theory **51**(8), 2962–2966 (2005).