IEEE 8th International Conference on Computer and Information Technology Workshops

A Virus Prevention Model Based on Static Analysis and Data Mining Methods

Tzu-Yen Wang
Dept. of Computer Sci., National Chiao Tung Univ., Hsinchu, Taiwan
tzu-yen.wang@hotmail.com

Chin-Hsiung Wu[*]
Dept. of Info. Tech. & Comm., Shih Chien Univ. Kaohsiung Campus, Taiwan
chwu@mail.kh.usc.edu.tw

Chu-Cheng Hsieh
Dept. of Computer Sci., Univ. of California, Los Angeles
chucheng@ucla.edu

Abstract

Owing to the lack of prevention ability of traditional anti-virus methods, a behavior-based virus prevention model for detecting unknown virus is proposed in this study. We first defined the behaviors of an executable by observing its usage of dynamically linked libraries and Application Programming Interfaces. Then, information gain and support vector machines were applied to filter out the redundant behavior attributes and select informative feature for training a virus classifier. The performance of our model was evaluated by a dataset contains 1,758 benign executables and 846 viruses. The experiment results are promising, and the overall accuracies are 99% and 96.66% for detecting the known viruses and the previously unseen viruses respectively.

1. Introduction

According to Symantec® security reports [9, 10], during July 2006 to June 2007, the virus threats have been getting serious so far. There were total 8,258 new Win32 variants and 136 previously unseen malware reported on the second half year of 2006. The number of virus without anti-virus signatures increased 22 percent over the first half of 2006. Furthermore, 212,101 new malicious code threats appeared in the first half year of 2007. It was a 185 percent increase over the second half of 2006. The studies in Computer virus detection have been worked for many years; however, the records above show that they are apparently not well enough.

Traditional signature-based anti-virus software catches malware based on "known" signature, but it is unrealistic for unknown virus or variants of existed ones. Therefore, our aim is to propose a virus prevention model (VPM) based on novel data mining methods to fix the drawbacks of traditional anti-virus methods. The rest of this paper is organized in the following way. Section 2 is devoted to some related researches of malicious executable detection. Section 3 states the main ideas of our VPM with experiment results given in Section 4. Finally, a few concluding remarks are sketched in Section 5.

*Corresponding author.
This work was partially supported by the National Science Council under the contract no. NSC-96-2221-E-158-001.

2. Related work

In general, there are two kinds of approaches for virus detection: static analysis and dynamic analysis. Static analysis utilizes the information in suspected executable programs without running it; and dynamic analysis monitors software activities after loading it into memory. Dynamic approach pays more attention on activities, but it is hard to prevent damage in advance since the virus it detects is "running". Our work only focuses on static analysis because we try to catch viruses before they cause the damage.

Here we just briefly state some of researches in recent years. J. Bergeron *et al.* proposed a slicing algorithm [1, 2] and transformed binary codes into different graphs such as control-flow graph, data-flow graph, call graph, and critical-API graph for further analysis. Also, J. Z. Kolter *et al.*[5] used a hex-dump utility to convert each executable to hexadecimal code in an ASCII format and produced n-gram features by combining each four-byte sequence into a single term. They used information gain to select valued features and tested the performance of various kinds of learning classifiers. M. G. Schultz *et al.*[8] applied three feature extracting methods including binary profiling, string sequences, and Naïve Bayes to generate classifiers to detect malware. [6] aimed at the detection of obfuscated viruses by examining their control flow graphs.

By observing the usage of dynamically linked libraries (DLLs) and Application Programming Interfaces (APIs) of executable files, we combined statically behavior-based analysis with data mining methods such as information gain, linear and nonlinear support vector machines to construct a virus prevention model provided against both the known viruses and the unknown ones.

3. Virus prevention model

Because viruses and benign programs are built for opposite intent, they differ on behaviors in nature. The distinctive behaviors between them could be used as indicators to recognize virus. On the other hand, merely looking code-block (signature) is easily evaded by obfuscated viruses; however, their intent is hard to manipulate. This is another good reason to detect viruses by their program behaviors rather than their code signatures. We have illustrated the process of our VPM in Figure 1. The ideas of VPM stepwise describes in the following sections.

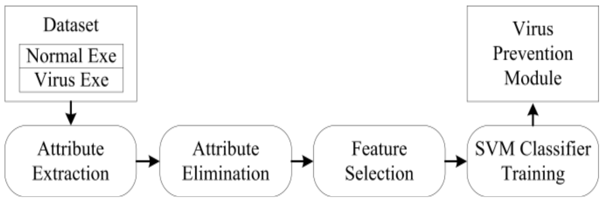


Figure 1. The process of VPM

3.1. Attribute extraction

For modern operation system, programs run in the protection mode. In other words, they need to request the operating system to execute critical operations such as file management, resource access, and device control etc. Microsoft® Windows operation system provided many Application Programming Interfaces (APIs) in all kinds of Dynamically Linked Libraries (DLLs) for applications to interact with it. Utilizations of DLLs and APIs could be defined as behaviors of the executable and considered as attributes, which provide some clues to discriminate viruses from benign programs.

The attributes of an executable can be statically extracted from its Portable Executable (PE) file format [7] without running the program. PE file format is a data structure, which records necessary information for the Windows OS loader to manage the wrapped executable code. A utility named "Dependency Walker" can parse the PE file format and output in tree structure. Figure 2 is an example of a tree diagram, which illustrates the relationships of the invoking DLLs in an executable. In this tree structure, each parent DLL invokes some APIs exported from its child DLLs. Basing on the tree structure, we defined three types of attributes: T1, T2 and T3. T1 indicates the APIs directly used by the main program, i.e., the APIs reside in the first layer DLLs. T2 is the APIs contained in the first layer DLLs indirectly invoked by other DLLs except the main executable. T3 consists of the entire dependency paths to represent the relationship between DLLs. Each of them is a complete downward path, starting from a first layer DLL to each of its leaf DLL. Moreover, each DLL records a hash value to indicate the content of the APIs directly exploited by its parent DLL. An example of three different types of attributes is given in Figure 2.

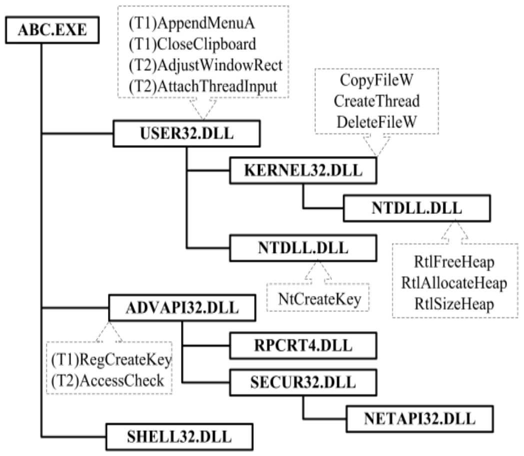
3.2. Redundant attribute elimination

When dealing with very high-dimensional data, eliminating redundant attributes is necessary to keep the informative ones called features. Information gain(IG)[12], which computes the discrimination of an attribute, is a kind of filter approach of feature selection.

Given a random variable X, if there are k values of X, the entropy of X is defined in (1), where p_i is the proportion of $X = v_i$.

$$H(X) = -\sum_{i=1}^{k} p_i(X = v_i) \log_2 p_i(X = v_i) \quad (1)$$

High entropy implies X to abide by a uniform distribution and lack of discrimination. Furthermore, the conditional entropy, which is denoted in (2), implies the entropy of X conditioned by the values of Y.



T1 : USER32-AppendMenuA( ), USER32-CloseClipboard( ),..., ADVAPI32-RegCreateKey(), ...

T2 : USER32-AdjustWindowRect( ), USER32-AttachThreadInput( ),..., ADVAPI32-AccessCheck( ),...

T3 : USER32(2785) \ KERNEL32(2155) \ NTDLL(2121), USER32(2785) \ NTDLL(2312), ADVAPI32(2379) \ RPCRT4(2545) , ...

Figure 2. An example of dependency tree

Then, information gain (3) is the amount by which the entropy of X decreases reflects additional information about X provided by Y.

$$H(X|Y) = -\sum_{j=1}^{m} p_j(Y = v_j) \times H(X|Y = v_j) \quad (2)$$

$$IG(X|Y) = H(X|Y) - H(X) \quad (3)$$

In our study, there were only two classes in the dataset and $H(X)$ is equal to the equation below: (v=virus, b=benign)

$$-\frac{|v|}{|S|}\log_2\frac{|v|}{|S|} - \frac{|b|}{|S|}\log_2\frac{|b|}{|S|} \quad (4)$$

, where |.| denotes the number of particular kind of executable and $|S| = |v| + |b|$. In addition, Y is an attribute with Boolean value such that $Y = 1$ indicates the subset S_1 in which each instance has attribute Y and

$Y = 0$ represents the subset $S_0 = S - S_1$. $H(X|Y)$ is denoted in (5) and $|S_k| = |v_k| + |b_k|$.

$$-\sum_{k=0}^{1} \frac{|S_k|}{|S|}\left(-\frac{|v_k|}{|S_k|}\log_2\frac{|v_k|}{|S_k|} - \frac{|b_k|}{|S_k|}\log_2\frac{|b_k|}{|S_k|}\right) \quad (5)$$

According to the definition of IG, the higher the IG, the better discrimination the attribute has. It means that attributes with equal IG are also the same in discrimination power that could be highly related. Therefore, it is unnecessary to reserve all of them to increase the burden of the data mining process especially when dealing with high-dimensional problem. Basing on this simple idea, the attribute elimination was performed by discarding all but one of attributes with the same IG. Finally, according to the attributes that we had reserved, every executable was represented by a Boolean vector. Each vector element corresponds to an attribute and its Boolean value indicates the corresponding attribute exists in the executable or not. An example is given in Table 1.

Table 1. Transfer executables to vectors

	A_1	A_2	A_3	A_4	A_n	Vector
E_1	T	F	T	T	F	1011…0
E_2	T	T	F	F	F	1100…0
E_m	F	F	T	F	T	0010…1

(E: executable; A: attribute)

3.3. Feature selection and data classification using support vector machines

Support vector machines (SVM)[4] is a kind of machine learning algorithm with high generalization ability and substantial theory[11]. Giving a training dataset $(x_1, y_1), (x_2, y_2),…, (x_m, y_m) \in R^n \times \{\pm1\}$, the main idea of SVM is to find a separating hyperplane $w^T x + b = 0$,where w is the normal vector of the hyperplane and b is the functional distance from the hyperplane to the origin, to linearly separate the two classes data in the original space. If the data are linearly inseparable, the nonnegative slack variable ξ and positive weight parameter C are introduced to control the trade-off between the maximization of generalization ability and minimization of the classification error at the same time. As was mentioned above, SVM with linear kernel (L-SVM) leads to the optimization problem:

$$\begin{aligned} &\min_{(\mathrm{w,b})\in \mathrm{R}^{n+1}} \ \tfrac{1}{2}\mathrm{w}^{\mathrm{T}}\mathrm{w} + \mathrm{C}|\xi| \\ &\text{s.t. } y_i(w^{\mathrm{T}}x_i + b) \geq 1 - \xi_i, \\ &\quad \xi_i \geq 0 \, , i = 1, …, \mathrm{m} \end{aligned} \quad (6)$$

By solving (6), we can obtain w and b to construct a decision function $D(x)$ for the classification problem.

$$D(x) = \mathrm{sgn}(w^{\mathrm{T}}x + b) \quad (7)$$

If L-SVM cannot linearly separate the data in the original space, mapping data to a high-dimensional feature space by a nonlinear function ϕ would be helpful. This mapping process can be done with a kernel function that reduces the effort of data mapping by ϕ and dot product in the high-dimensional feature space. After considering the complication of virus detection problem, we used the SVM with RBF kernel (8) to establish a virus detector where γ is the parameter of the RBF kernel.

$$k(x, y) = \exp\left(-\gamma\|x - y\|_2^2\right) \quad (8)$$

For the equation (6) of L-SVM, if the square of any element w_k of w equals to zero or approximates to zero, the deletion of the associated kth attribute does not affect the optimal hyperplane. By this observation, the values of all w_k^2 of L-SVM could be criterions for feature selection. Suppose that ***A*** is the set of attributes, which has been filtered out the redundant attributes by IG. ***P*** is the performance of RBF-SVM. The procedure of feature selection can be performed as follows:

1. Train an L-SVM by ***A*** and obtain the normal vector w. Rank all the attributes in ***A***, according to their corresponding w_k^2 value.

2. Delete one or some of the attributes with lower rank from ***A*** and denote remaining attribute set as ***A'***.

3. Use ***A'*** to train a RBF-SVM and get ***P'***. If ***P'*** > ***P***, then set ***P***=***P'*** and ***A***=***A'*** and go to Step 2. Otherwise, stop this procedure and define ***A'*** as the feature set.

4. Experiment results

Our dataset consisted of 846 virus and 1,758 benign executables and all of them were Windows PE formatted and unpacked. We obtained some benign programs from the folders of pure Windows XP operating system. Besides, we downloaded additional ones from the web site PChome® Downloads (http://toget.pchome.com.tw/). On the other hand, viruses were downloaded from the web site VX Heavens (http://vx.netlux.org). The SVM classifier was obtained by LIBSVM[3].

Since both of the detection rate of virus and the false alert rate of benign program are important, we evaluated them at the same time by the viewpoint of Overall Accuracy (OA) indicates the proportion of correctly classified executables. The experiment contained two parts: the detection of known viruses and the prediction of previously unseen viruses. The prediction rate was calculated by five-fold cross-validation (CV). In other words, the dataset were randomly divided into five disjoint portions with an equal number of sizes in each portion, one of them was chosen as the testing set, and the remaining four portions were merged into a training set. This procedure was repeated five times by choosing different partition as a testing set. We calculated the average OA as the generalization ability of our system.

After parsing the PE file format of each executable in the dataset, we got 93,116 attributes in the beginning. The experiment results shown in Table 2. reveal that even some of the attributes with lower IG were discarded; the performance of RBF-SVM remained steady. Therefore, we successfully reduced the number of attributes from 93,116 to 1,398 after the process of attribute elimination. Continuously, Figure 3 presents the results of the feature selection procedure stated in Section 3.3. Evidently, we not only succeeded in feature reduction but also made further progress in classification performance. Finally, the top 429 features was chosen to train the RBF-SVM and achieved a proper performance with 96.66% in prevention rate and 99.00% in detection rate (True Positive=98.35%, False Positive=0.68%).

5. Conclusion

In summary, the following are the main concepts of VPM. First, executable behaviors are examined by the usage of DLLs and APIs and the DLL dependency examination is novel. Moreover, both attribute elimination and feature selection were performed by using IG and L-SVM respectively for collecting informative features. Finally, these features are used to train a RBF-SVM that is well performed in the evaluations of known and unknown virus detection.

Future research will extend this model to detect other kinds of Win32 malware such as email worm, Trojan horse, backdoors. In addition, we are hopeful that there might be an opportunity to cooperate with COTS anti-virus software.

Table 2. Results of attribute elimination by IG

IG Threshold	Attribute #		CV %	
	Before	After	Before	After
0.2	20	20	83.91	83.91
0.1	1,234	293	86.64	87.10
0.05	1,469	510	88.90	88.71
0.025	3,104	693	90.98	91.17
0.01	6,684	903	93.78	93.98
0	93,116	1,398	95.41	95.62

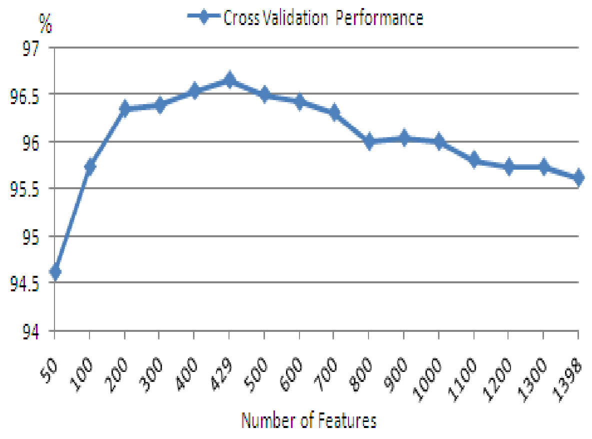


Figure 3. Results of feature selection by L-SVM

6. References

[1] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi, "Static Detection of Malicious Code in Executable Programs," in *Symposium on Requirements Engineering for Information Security (SREIS' 01)*, 2001.

[2] J. Bergeron, M. Debbabi, M. M. Erhioui, and B. Ktari, "Static analysis of binary code to isolate malicious behaviors," in *Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1999, pp. 184-189.

[3] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," 2001.

[4] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*: Cambridge University Press, 2000.

[5] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* Seattle, WA, USA: ACM, 2004.

[6] C. Mihai and J. Somesh, "Static analysis of executables to detect malicious patterns," in *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12* Washington, DC: USENIX Association, 2003.

[7] M. Pietrek, "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format," *Microsoft Systems Journal,* vol. 9, pp. 15-34, 1994.

[8] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 38-49.

[9] Symantec, "Symantec Internet Security Threat Report," vol. 11, March 2007.

[10] Symantec, "Symantec Internet Security Threat Report," vol. 12, September 2007.

[11] V. N. Vapnik, *Statistical Learning Theory*: John Wiley & Sons, 1998.

[12] I. H. Witten and E. Frank, *Data Mining- Practical Machine Learning Tools and Techniques with JAVA Implementations*, 2 ed.: Morgan Kaufmann, 2005.