International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.2, No. 1 (2006), pp. 100-104
© Research India Publications http://www.ijcir.info

Using Support Vector Machine to Detect Unknown Computer Viruses

Bo-yun Zhang[1,2], Jian-ping Yin[1], Jin-bo Hao[1], Ding-xing Zhang[1] and Shu-lin Wang[1]

[1]School of Computer Science, National University of Defense Technology,
Changsha 410073, China
hnjxzby@yahoo.com.cn

[2]Department of Computer Science, Hunan Public Security College,
Changsha 410138, China

***Abstract*: A novel method based on support vector machine (SVM) is proposed for detecting computer virus. By utilizing SVM, the generalizing ability of virus detection system is still good even the sample dataset size is small. First, the research progress of computer virus detection is recalled and algorithm of SVM taxonomy is introduced. Then the model of a virus detection system based on SVM and virus detection engine are presented respectively. An experiment using system API function call trace is given to illustrate the performance of this model. Finally, comparison of detection ability between the above detection method and other is given. It is found that the detection system based on SVM needs less priori knowledge than other methods and can shorten the training time under the same detection performance condition.**

Keywords: computer virus, API function calls, Support Vector machine, virus detection.

I. Introduction

Excellent technology exists for detecting known malicious executables. Software for virus detection has been quite successful, and programs such as McAfee Virus Scan and Norton AntiVirus are ubiquitous. These programs search executable code for known patterns, and this method is problematic. One shortcoming is that one must obtain a copy of a malicious program before extracting the pattern necessary for its detection. Obtaining copies of new or unknown malicious programs usually entail them infecting or attacking a computer system.

In this paper, we propose a novel method to detect computer viruses through SVM [1]. Our efforts to address this problem have resulted in a fielded application, built using techniques from statistical pattern recognition and machine learning. The Viruses Classification System currently detects unknown malicious executables code without removing any obfuscation. To our knowledge, the experiment is the first time to established methods based on SVM applying to detect unknown computer viruses.

In the following sections, we first describe related research in the area of computer viruses detection. Then we illustrate the architecture of our detect model in section III. Section IV introduces the classification algorithm. Section V details the method of extraction feature from program, and stating the detection engine work procedure. Section VI shows the implementation and experiment results. We state our conclusion in Section VII.

II. Related work

So far, there have been few attempts to use machine learning and data mining for the purpose of identifying new or unknown malicious code.

In an early attempt, Lo et al. [2] conducted an analysis of several programs evidently by hand and identified tell-tale signs, which subsequently be used to filter new programs. Researchers at IBM's T.J.Watson Research Center have investigated neural networks for virus detection [3] and have incorporated a similar approach for detecting boot-sector viruses into IBM's Anti-Virus software.

More recently, instead of focusing on boot-sector viruses, Schultz et al. [4] used data mining methods, such as naïve Bayes, to detect malicious code. The Bloodhound technology of Symantec Inc., uses heuristics for detecting malicious code [5]. Szappanos [6] used code normalization techniques to detect polymorphic viruses. Normalization techniques remove junk code & white spaces, and comments in programs before they generate virus signature.

To improve the performance of the detector mentioned above, lots of malicious and benign codes as training dataset should be collected. And they would consume lots of times when training the classifiers. Aid to solve these problems, the SVM was utilized in our experiments.

III. Model Structure

We first describe a general framework for detecting malicious executable code. Figure 1 illustrates the proposed architecture. The framework is divided into 4 parts: (1) application server, (2) Virtual Computer, (3) detection server, and (4) virus detection firewall based on character code scanning. Once detected, the operating system can be designed to observe the behavior of the computer viruses. This would require the system to contain a virtual environment or machine. So the virtual operating system-VMWare[7] was used in our experiments. The computer virus would be executed in the virtual environment to learn its behavior. In this environment, the virus would not destroy the real detection system.

Before a file save to the application server, it will be scanned by the virus detection firewall. If the file is infected with virus then quarantine it. Otherwise if there is no malicious information about the file, it will be replicated 2 copies. Then, one copy will be sent to the application server, another one will be sent to the detect server based on SVM detection engine after extracting feature in the virtual computer. At the following stage, the SVM detector check the copy again. If the file is infected with unknown viruses, the application server will be remind to remove the copy from its application database. And then quarantine it in a special database or sent it to an expert to analyze it by hand.

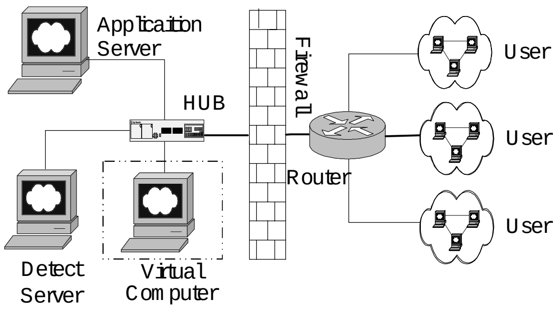


Figure 1. Architecture

IV. Classification algorithm

Virus detecting can be look as a binary classification problem. Our detecting model is based on SVM, which is a kind of machine learning method based statistical learning theory. The advantage of this method is that its general capability can be improved by using structural risk minimization principle. That is to say, even using limited training sets, we can get a relatively small error rate on independent testing sets. In addition, since SVM is a convex problem, the local optimum found by this method is also global optimum. Here we mainly discuss the method of calculating decision hyperplane and sample classification. First we label the training data $(x_1, y_1), ..., (x_l, y_l) \in R_k \times \{\pm 1\}$, $i = 1, ..., l$, $y_i \in \{+1, -1\}$, $x_i \in R_k$. Suppose we have some hyperplane

$$w \cdot x + b = 0 \tag{1}$$

in which separates the positive from the negative examples. Where w is normal to the hyperplane, $|b| / \|w\|$ being the perpendicular distance from the hyperplane to the origin, $\|w\|$, the Euclidean norm of w , and $w \cdot x$, the dot product between vector w and vector x in feature space. The optimal hyperplane should be a function with maximal margin between the vectors of the two classes , which subject to the constraint as :

$$\max W(a) = \sum_{i=1}^{l} \alpha - \frac{1}{2} \sum \alpha_i y_i \alpha_j y_j K(x_i, x_j)$$
$$s.t. \quad \sum_{i=1}^{l} \alpha_i y_i = 0, \ \alpha_i \in [0, C], \ i = 1, ..., l. \tag{2}$$

where α_i is Lagrange multipliers, $K(x_i, x_j)$ is kernel function, C is a constant.

For a test sample $x \bullet$ we could use decision function :

$$f(x) = \text{sgn}(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b) \tag{3}$$

to determine which class it is.

V. Viruses Detection Engine

A. Basic hypothesis

The area of coverage is characterized by the assumptions mentioned below.

Assumption 1. Computer virus interacts with Operating System at runtime.

Most types of malicious activities, such as accessing network or file services, involve interactions with the OS. However, some malicious activities, such as denying service or corrupting data, can be done without interactions with the OS, virus that limits itself to such activities will not be detected by our system.

Assumption 2 In interacting with the Operating System, Viruses use the Win32 APIs.

Instead of using the Win 32 APIs, it is possible to interact with the OS through the Windows NT native API functions. Our detection system can be extended to cover this type of interaction.

Assumption 3. Once a infected program begins to execute, the infection procedure ofvirus must immediately usurp control of the computer, whereas the other procedure of the virus may not always run.

Assumption 4. The Win32 APIs used by software executables can be effectively monitored at runtime.

B. Sample extracting

Our first intuition into the problem was to extract information from the PE executables that would dictate its behavior. Professor Forrest [8] had studied thoroughly about the role of system call sequence of Unix OS in intrusion detection. According to their studies, we choose the Windows API function calls as the main feature in our experiments. To extract resource information from Windows executables we used GNU's Bin–Utils [9]. Because more and more sophisticated computer virus, which using polymorphic and obfuscation techniques foil the commercial virus scanner. Sometimes one cannot extract API function calls only using the method above, so a tracing tool --APISPY.EXE was designed in our experiments.

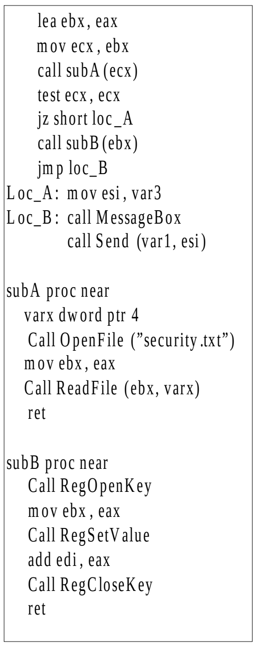
1) Size of window

By monitor the behavior of each sample in the Virtual Computer, we could trace the API function calls of them. After extracting the system call sequence , index the API function in system call mapping file, then each function has a index value, for example ‘35’ assign to function ‘openfile( )’, show as Fig 2(b). We save the numerical sequence correspond to the api function trace into a data file and slide a window of size *k* across the trace, recording each unique sequence of length *k* that is encountered. For example, if *k*=4, one get the unique sequences show detail as Fig 2(c).

Short sequence of system call represent the order of system call by executing process, then, which value is the best for short sequence length? Wenke Lee and Steven A. Hofmey[10] found that one can not get useful message from system call sequence when window size larger than 30. If take conditional entropy and computational consumption into consideration, short sequence length should be 6 or 7.

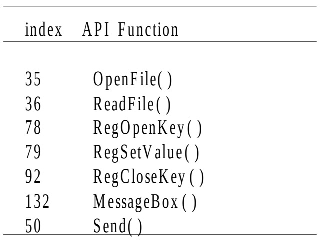
2) Extracting abnormal samples

We decide parameter value of SVM through training, so both normal short sequence and abnormal sequence samples are needed. System call short sequence sample can be gotten by scanning the history of system call using *k* length slide window, these samples are saved in a sample database. Generally, the record in sample database is not larger than $|\sum|^k$, where $\sum$ being API call sets, $|\sum|$, API function number and k , the size of slide window.



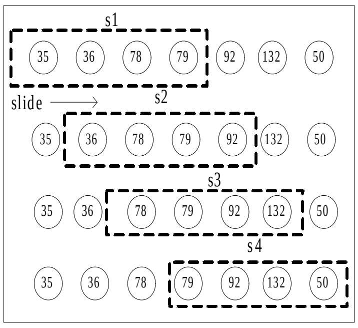
```
    lea ebx, eax
    mov ecx, ebx
    call subA (ecx)
    test ecx, ecx
    jz short loc_A
    call subB (ebx)
    jmp loc_B
Loc_A:  mov esi, var3
Loc_B:  call MessageBox
        call Send (var1, esi)

subA proc near
   varx dword ptr 4
    Call OpenFile ("security.txt")
   mov ebx, eax
   Call ReadFile (ebx, varx)
    ret

subB proc near
    Call RegOpenKey
    mov ebx, eax
    Call RegSetValue
    add edi, eax
    Call RegCloseKey
    ret
```

(a)



index	API Function
35	OpenFile( )
36	ReadFile( )
78	RegOpenKey( )
79	RegSetValue( )
92	RegCloseKey( )
132	MessageBox( )
50	Send( )

(b)



• c)

Figure 2. (a) An malicious code snippet (b) API call trace (c) When window size *k*=4,obtaion short API call traces s1=35 36 78 79,s2=36 78 79 92,s3=78 79 92 132,s4=79 92 132 50.

When using *k* length slide windows to scan system call history by the program that contain malicious code, we can get a set of short sequences which including normal and abnormal system calls. Since few activities are illegal, abnormal short sequences is only a small part of the whole short sequences. We use the following procedure to judge whether API call sequence is legal or not.

After got short sequence of malicious program, we compare it with records in sample database, if it matches with any item, it will be deleted. Otherwise, the Hamming distance between normal samples will be calculated. The similarity between two sequences can be computed using a matching rule that determines how the two sequences are compared. The matching rule used here is based on Hamming distance, i.e. the difference between two sequences *i* and *j* is indicated by the Hamming distance $d(i, j)$ between them. For each new sequence *i,* we determine the minimal Hamming distance $d_{\min}(i)$ between *i* and the set of normal sequences:

$$d_{\min}(i) = \min\{d(i, j), \forall normal\ \ sequence\ \ j\}$$

The $d_{\min}(i)$ value represents the strength of the anomalous signal, i.e. how much it deviates from a known pattern. Note that this measure is not dependent on trace length and is still amenable to the use of thresholds for binary decision making. If the rate of anomalous to normal sequences is R_A , then the average complexity of computing $d_{\min}(i)$ per sequence is $N(k-1)R_A + (k-1)(1-R_A)$,which is $O(k(R_A N+1))$, where *k* is the size of window, *N* is the number of normal sample in dataset.

For a mismatched sequence *i*, we set thresholds on the values, It will be regarded as anomalous any sequence *i* for which $d_{\min}(i) \geq D$, where $1 \leq D \leq k$ being the threshold value. It is say if a sequence *i* of length *k* is sufficiently different from all normal sequences, it can be flagged as anomalous. So we obtain empirically the abnormal dataset.

C. Detection Method

The virus detecting method presented in this paper is a supervising learning algorithm. Firstly, we marked the API call sequence, then got the parameters value of SVM by training. To Check a file, we trace the API call sequence, then use *k* length slide window to divide the sequence into several short sequences, these short sequences can be judged by SVM classifier, abnormal short sequence will be marked, finally, we can judge whether the file contains virus or not based on the output of SVM classifier.

In reality, it is likely to be impossible to collect all normal variations in behavior, so we must face the possibility that our normal database will provide incomplete coverage of normal behavior. If the normal were incomplete, false positives could be the result. Moreover, the inaccuracy of the SVM itself also needs to set some judge rules to improve the performance of the detecting systems. So we judge whether a file contains virus based on the number of abnormal API call short sequence. If the number is larger than a predefined threshold, the file has been infected by virus, otherwise not. We decide the threshold value by training.

VI. Experiment Results

We estimate our results over data set in table 1. The malicious executables were downloaded from http://vx.netlux.org and http://www.cs. Columbia. edu / ids/mef/, the clean programs were gathered from a freshly installed Windows 2000 server machine running MS Office 2000 and labeled by a commercial virus scanner with the correct class label(malicious or benign) for our method. The pretreating procedure of experiment data details as follow step:

(1) Tracing API call sequences from viruses set and benign set. An api call tracing tool is programmed in our experiment. It can hook all API function calls in Windows 2000 server platform. (2) Disparting each API call sequence into short trace by k –the size of sliding window. (3) Identifying the abnormal short traces in the short sequence set of viruses. (4) Choose parts of normal and abnormal short trace as training data to train SVM, and the other as testing set. At last, we get the distribution of the short traces database used in training and testing in our experiment, show in table 2.

During the experiment, we use the software LIBSVM [11]. To evaluate our system we were interested in several quantities: (1). False Negative, the number of malicious executable examples classified as benign; (2). False Positives, the number of benign programs classified as malicious executables.

There are some common kernels mentioned in SVM, we must decide which one to try first. Then the penalty parameter C and kernel parameters are chosen. After compare with other kernel, at last we choose Radial Basic Function:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right)$$

There are two parameters while using RBF kernel: $1/\sigma^2$ and C. It is not known beforehand which C and $1/\sigma^2$ are the best for one problem. Consequently some kind of parameter search must be done. So we try some variable group value of $(1/\sigma^2, C)$ to test the classification performance of SVM.

In our experiments, there are 100 files in training dataset, and 532 files in testing dataset. The dimension of feature vector is k, which is the length of short API traces. We set the value of threshold D is 3, then we test the detection engine when k=6, 7 respectively. And the detail experiment result shows in table 3 . In another experiment [12], we had used a algorithm based on Fuzzy Pattern Recognition algorithm(FPR) to classify the data set in table 1. That algorithm had the lowest false positive rate, 4.45%. The present method has the lowest false positive rate, 3.21%, which has better detection rates than the algorithm based on FPR. Notice that the detection rates of these two methods is nearly equal, but the FPR algorithm use more training samples than SVM algorithm. This shows that SVM algorithm is fit to detect computer viruses when the viruses sample gathered is difficult.

	Sample space	Training set	Testing set
Benign file	423	50	373
Malicious file	209	50	159
Sum	632	100	532

Table1. Sample data in Experiment.

Training data set		Testing data set	
Number of normal traces	Number of abnormal traces	Number of normal traces	Number of abnormal traces
496	242	2766	876

Table 2. Distribution of the traces database used in the experiment

C	σ^2	False Negative		False Positive	
		k = 6	k = 7	k = 6	k = 7
50	10	3.21%	4.02%	5.66%	7.54%
100	1	4.82%	5.63%	6.28%	5.66%
200	0.5	6.97%	7.50%	10.06%	11.32%

Table 3. Experimental result of detection system

VII. Conclusion

We presented a method for detecting previously undetectable computer viruses. As our knowledge, this is the first time that using support vector machine algorithm to detect malicious codes. We showed this model's detect accuracy by comparing our results with other learning algorithms. Experiment result shows that the present method could effectively use to discriminate normal and abnormal API function call traces. The detection performance of the model is still good even the virus sample dataset size is small.

Acknowledgment

This work supported by the National Natural Science Foundation of China under Grant No.60373023 , Hunan Provincial Natural Science Foundation of China under Grant No.04JJ6032 and the Scientific Research Fund of Hunan Provincial Education Department of China under Grant No.05B072. The authors would like to thank Dr. Hui Xia of Hokkaido University for his helpful comments.

References

[1] Vapnik,V.N.: Statistaical learning theory. Springer, New York(1998)

[2] Lo,R., Levitt,K., Olsson,R.: MCF: A Malicious Code Filter. Computers & Security.14(1995)541-566

[3] Tesauro,G., Kephart,J., Sorkin,G.: Neural networks for computer virus recognition.IEEE Expert. 8(1996)5-6

[4] Schultz,M., Eskin,E., Zadok,E., Stolfo,S.: Data mining methods for detection of new malicious executables. In: Needham,R., Abadi M, (eds):. Proceedings of the 2001 IEEE Symposium on Security and Privacy. Oakland, CA: IEEE Computer Society Press (2001) 38-49

[5] Symantec. http://www. symantec. com / avcenter / reference/ heuristc.pdf. (Last accessed: Jun 1.2004)
[6] Szappanos,G.: Are There Any Polymorphic Macro Viruses at ALL (and What to Do with Them).in Proceedings of the 12th International Virus Bulletin Conference(2001)
[7] VMware. http://www.vmware.com. (Last accessed: 10, Nov. 2003)
[8] Forrest,S., Hofmeyr, S. A., Somayaji, A.: Computer immunology. Communications of the ACM. 10 (1997) 88–96
[9] Cygnus. http://sourceware. cygnus.com / cygwin (Last accessed: 20, Dec. 2004)
[10] Lee,W., Dong,X.: Information-Theoretic measures for anomaly detection. In: Needham,R., Abadi M, (eds):. Proceedings of the 2001 IEEE Symposium on Security and Privacy. Oakland, CA: IEEE Computer Society Press (2001)130-143.
[11] LIBSVM. http://www.csie.ntu.edu.tw/~cjlin/. (Last accessed: 18,Nov. 2004)
[12] Zhang,B., Yin,J., Hao,J.: Using Fuzzy Pattern Recognition to Detect Unknown Malicious Executables Code. In: Wang,L.,Jin,Y.(eds.):Fuzzy Systems and Knowledge Discovery. LNAI,Vol.3613. Springer-Verlag, Berlin Heidelberg New York(2005) 629-634

Author Biographies

Boyun Zhang born in YongZhou, Hunan, China on 1972, received his B.S. degrees in physics education from Xiangtan Normal University , Xiangtan, China, in 1994 and M.S. degree in applied computer science from Hunan University, Changsha, China in 2002. He is currently pursuing his PhD degree at School of Computer Science, National University of Defense Technology, Changsha, China. His current research interests include network security and machine learning.