Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090260

Computer Malicious Executables Detection based on Real-Valued Negative Selection Algorithm

*Jinquan Zeng**

School of Computer Science & Engineering, University of Electronic Science and Technology of China, 610054 Chengdu, China

Received: 1 Jul. 2014, Revised: 30 Sep. 2014, Accepted: 1 Oct. 2014
Published online: 1 Mar. 2015

Abstract: How to detect computer malicious executables is an important research direction of computer security, especially, unknown malicious executables and new variants. Inspired by biological immune systems, a based on real-valued negative selection algorithm approach to detect malicious executables is proposed in this paper, which is referred to MEDRNS. In order to avoid detectors covering self space, some of benign executables are used to build the profile of the system, and then based on the built profile of the system, the detectors are generated. At the same time, using the variable-sized self radius to represent the self space, detectors have the more quality. The approach can increase true-positive rate and decrease false-positive rate, and experimental results show that MEDRNS has better detecting ability than that of the previous techniques.

Keywords: Artificial immune systems, malicious executables, anomaly detection

1 Introduction

The computer malicious executable codes have a long history and can be categorized into three kinds based on their transport mechanism. The first is called viruses that always infect other benign programs, which become infected, and in turn, propagate the virus to other programs when executed. The second is called Trojans that always masquerade its malicious executable code inside a useful utility or freeware program, but perform malicious functions. And the last but not the least is called worms that can replicate and distribute itself automatically around the network, usually by exploiting vulnerabilities in the software running on the networked computers. With the fast development of Internet, security threats of malicious executable code are getting more serious. Staniford introduces a worm that can spend the whole Internet within 30 seconds [1]. How to detect malicious executables, specially unknown malicious executables, has become one of the prime research interests in the field of computer security [1,2,3,4].

Current anti-virus systems with a large number of virus signatures can only detect known viruses and cannot detect unknown viruses and the variants of known viruses [5]. Although these anti-virus systems use the word virus in their names, they also detect worms and Trojans. In order to avoid to be detected by anti-virus systems, the authors of virus make viruses change their structures when the viruses copy themselves [6]. Some researchers have found that the method can succeed in escaping from anti-virus systems [7]. At the same time, eight to ten malicious programs are created every day and most cannot be detected until signatures have been generated for them [8]. During this time period, systems protected by signature-based anti-virus systems are vulnerable to attacks. These challenges have prompted some researchers to investigate learning methods for detecting new or unknown viruses, and more generally, malicious codes.

In order to detect new or unknown malicious executables, some researchers begin to investigate learning methods. Early, Lo et al [9] proposed the filter for the viruses that can escape from signature-based methods; however, no experiment was conducted to validate the method. Tesauro et al [10] investigated the neural network for detecting boot-sector viruses and incorporated it into IBM's Anti-virus software. This method can efficiently detect boot-sector viruses, however, not other viruses. Mihai and Somesh[11] presented a static analyzer for executables; however, the machine instruction sequence in the executables has to be known and it costs very much time. Some researchers

* Corresponding author e-mail: zengjq@uestc.edu.cn

used data mining methods to detect malicious executables, but the false-positive rate is high [7,11,12,13].

Biological immune systems (BIS) have many characteristics such as uniqueness, autonomous, recognition of foreigners, distributed detection, and noise tolerance [14]. Inspired by BISs, Artificial Immune Systems (AIS) have become one of the relatively new areas of soft computing [15,16,17,18,23,29] and AISs generally include clonal selection based algorithms, negative selection based algorithms and artificial immune network models [19,20,21,22]. One of the major algorithms developed within AISs is the negative selection algorighm (NSA), proposed by Forrest et al. [24]. The NSA can only use self samples to train detectors for classifying unseen data as self or non-self and its typical applications include anomaly detection, fault detection, especially, network security. Early works in NSAs used the problem in binary representation [24]. However, many applications are natural to be described in real-valued space and cannot be processed by NSAs in binary representation [25]. Recently, more concerns focused on real-valued negative selection algorithm (RNS) [25,26]. The algorithms use a real-valued representation of the self/non-self space and can speed up the detector generation process [27].

AIS [15] is considered as a new way to defeat fast-proliferating malicious executables. In order to detect malicious executables, especially, new or unknown malicious executables, a based on real-valued negative selection algorithm to detect malicious executables is proposed in this paper, which is referred to MEDRNS. Quantitative description of the model is given. Experimental results show that MEDRNS has better detecting ability.

2 Model Theories

In BIS, antibodies play an important role in protecting the host from external antigens. In order to cover the external antigens, the repertoire in BIS is huge [30] and the antibody is made up of multi-gene segments to attain the diversity [28]. In computer systems, the executables are made up of the binary strings, and the string of the binaries decides the function of the executables and makes the executables behave benign or malicious. Inspired by the principles of the antibody diversity in BIS, the variable-length instructions is extracted from benign executables and makes up of the benign instruction library (BIL), and the BIL is used to extract the characters of the executables in MEDRNS, including benign and malicious executables. Furthermore, the profile of the benign executables can be built by using the characters of the benign executables and then detectors are generated to cover the space of the malicious executables.

2.1 BIL

In this paper, Define instructions as the binary strings extracted from benign executables and the variable-length instructions set consists of the BIL. Let BI_l devote the benign instructions set given by:

$$AI_l = \{bs \,|bs \in B_l, |bs| = l, l \in N\} \tag{1}$$

where l is the instruction length (the number of bytes), N is the natural number and B_l is the instructions extracted from the benign executables.

The benign instruction library (BIL) is given by:

$$BIL = BI_{l_1} \cup BI_{l_2} \cup, \cdots, \cup BI_{l_n} \tag{2}$$

where $l_i \in N$, $i = 1, \cdots, n$ is the instruction length, and N is the natural number. The equation (2) shows that the BIL is made up of variable-length antibody genes, and the antibody gene library is used to extract the characteristics of the executables.

2.2 Antigen Presenting

Antigens are defined as the executables, including benign and malicious executables. Simulating the antigen presenting cells in BIS, and the characteristics of an executable are extracted from the BIL. Let c devote the characteristics of an executable, described by the equation (3). Where $0 \leq x_{l_i} \leq 1, i = 1, \ldots, n$, x_{l_i} is the executable characteristics extracted from the benign instructions set BI_{l_1}, n is the dimension, and the extracting method is described by the equation (4). Where the function $f_e(e, j, l_i)$ extracts the binary string from the benign executable e, j is the extracted position and l is the number of extracted bytes, respectively.

$$c = < x_{l_1}, x_{l_2}, \cdots, x_{l_n} > \tag{3}$$

The equation (3) shows that the state vector of the executable is made up of the characteristics extracted from the whole benign instructions set $BI_{l_i}, i = 1, \cdots, n$.

$$x_{l_i} = \frac{\left| BI_{l_i} \cap \left\{ \bigcup_{l_i \in N, j=0}^{|e|} \{f_e(e, j, l_i)\} \right\} \right|}{\left| \bigcup_{l_i \in N, j=0}^{|e|} \{f_e(e, j, l_i)\} \right|} \tag{4}$$

The executables include malicious executables and benign executables. The malicious executables set is denoted as $C_n \subset C$ and let $C_s \subset C$ be the benign executables, such that:

$$C_s \cap C_n = \Phi, C_s \cup C_n = C \tag{5}$$

2.3 Benign Executable Space

Let S the benign executables set given by:

$$S = \{\langle c, r\rangle \,|c \in C_s, r \in R\} \tag{6}$$

where c is the characteristics of the benign executables, C_s is the set of the characteristics of the benign executables, r is the self radius of benign executables, and R is the real number, respectively. The characteristics extracted from the benign executables are used to build the profile of the benign executables.

How to build the profile of the benign executables is very important work in NSAs. In facts, we cannot collect all of benign executables, and so we have to use some of benign executables to express the space of benign executables. In order to carry out the aim, the self-radius is introduced. The traditional self radius[31] is constant-sized and cannot build an appropriate profile of benign executables. Comparing with the version of constant-sized self radius, a variable-sized self radius is introduced in this paper. We count the distance among the set of benign executables and assign a variable-sized self radius based on the total distance of every benign executable to other benign executables. Now that we let each benign executable in the training set has its own self radius in addition to the distance to other benign executables. Big distance means that the benign executable is far from other benign executable, and so the number of benign executables near the benign executable is little and low self radius is assigned to the benign executable. If the distance is little, it shows that the number of the benign executables near the benign executables is high and so the big self sample is assigned to the benign executable. The self radius of the benign executable s is given by:

$$s.r = \sum_{c \in C_s} f_d(s,c) \Big/ (|C_s| - 1) \tag{7}$$

Where the function $f_d(s,c)$ is the Euclidean distance between s and c.

2.4 Malicious Executable Detectors

A NSA consists of two phases, training and detecting phase. In training phase, the detectors are generated randomly and those that match any benign executable using Euclidean distance matching rule are eliminated. Let D denote the detector set given by:

$$\begin{aligned} D = \{d \,|d.c \in U, d.r \in R, \exists s \in C_s, \forall s' \in C_s, \\ f_d(d,s') > f_d(d,s), d.r = f_d(c,s) - s.r\} \end{aligned} \tag{8}$$

where $d =< c, r >$, c is the characteristics of the detectors, $U = [0,1]^n$, is a n-dimensional space, r is the detection radius of the detectors, and R is the real number, respectively.

2.5 The Evolution of Model

In an actual application, the benign executables often vary, for example, the user installs or uninstalls application soft, and the evolution of the benign executables is given by:

$$B(t) = \begin{cases} B(0), t = 0 \\ B(t-1) \cup B_{new}(t) - B_{delete}(t), t > 0 \end{cases} \tag{9}$$

where $B(t)$, $B(t-1)$ are, respectively, the benign executables at time t and $t-1$. $B(0)$ is the initial benign executables. $B_{new}(t)$ is the new benign executables added into B at time t. $B_{delete}(t)$ are the mutated benign executables deleted at time t, which includes three parts: 1) the unloaded software; 2) the elements recognized by new detectors; 3) the benign executables infected by malicious executables. Because of the evolution of the benign executable, the instructions extracted from the benign executables also evolves and responses these variations, and the evolution of the BIL is given by:

$$BI(t) = \begin{cases} BI(0), t = 0 \\ BI(t-1) \cup BI_{new}(t) - BI_{delete}(t), t > 0 \end{cases} \tag{10}$$

where $BI(t), BI(t-1) \subset BIL$ are, respectively, the instructions at time t and $t-1$. $BI(0)$ is the initial instructions. $BI_{new}(t)$ is the new instructions added into the BIL, which are extracted from $B_{new}(t)$. $BI_{delete}(t)$ is the instructions deleted from the BIL, which are extracted from $B_{delete}(t)$.

Adding or deleting the instructions can response the variants of the benign executables in a computer system. Furthermore, according to the new executables and BIL, the profile of the benign executables is also updated given by:

$$C_b(t) = \begin{cases} C_b(0), t = 0 \\ \{c_b = \langle x_{l_1}, x_{l_2}, \cdots, x_{l_n}\rangle \,|0 \le x_{l_i} \le 1, e_b \in B(t) \\ \quad , i = 1, \cdots, n\}, t > 0 \end{cases} \tag{11}$$

where $B(t)$ is the benign executables. $C_b(0)$ is the initial profile of the benign executables, and $C_b(t)$ is the profile of the benign executables at time t. Based on the new profile of the benign executables, the detectors set D also update given by:

$$\begin{aligned} D(t) = \begin{cases} D(0), t = 0 \\ D(t-1)_{reserve} \cup D(t)_{update} \cup D_{new}(t) \\ \qquad - D_{delete}(t), t > 0 \end{cases} \end{aligned} \tag{12}$$

$$\begin{aligned} D_{reserve}(t-1) = \{x \,|x \in D_{reserve}(t-1), \forall s \in S(t), \\ f_d(x,s) \ge x.r + s.r\} \end{aligned} \tag{13}$$

$$\begin{aligned} D_{update}(t) = \{x \,|x \in D(t-1), \exists s \in S(t), \\ f_d(x,s) < x.r + s.r, x.r = f_d(s,x) - s.r\} \end{aligned} \tag{14}$$

$$\begin{aligned} D_{delete}(t) = \{x| \, x \in D(t-1), \exists s \in S(t), \\ f_d(x,s) \le s.r\} \end{aligned} \tag{15}$$

where $D(t), D(t-1) \subset D$, are, respectively, the detectors at time t and $t-1$. $D(0)$ is the initial detectors. $D_{reserve}(t-1)$ is the reserved detectors at $t-1$, which do not cover the space of the benign executables and are the valid detectors. $D_{update}(t)$ is the updated detectors, which cover partly the space of the benign executables and decrease their detection radius. $D_{new}(t)$ is the new detectors added into the detectors set, which are tolerant to C_s. $D_{delete}(t)$ is the detectors eliminated from the detectors set, which lie within the benign executables set C_s. Reserving the valid detectors decreases the cost of training detectors, updating and eliminating detectors can efficiently decrease the false position rate, and adding the new detectors can increase the true-positive rate.

The traditional techniques only can build a static profile of the system and cannot adapt the varieties of benign/malicious executables space, and so they produce the high false position rate and low true position rate. In MEDRNS, the evolution of model is efficient and active learning mechanism, and increases its self-adaptation and self-learning capability in a variable self/nonself surrounding.

2.6 Suspicious Executable Detection

After the characteristics $c(c \in C)$ of an executable e is extracted, its characteristics are presented to the detectors for detecting and the detecting process is given by:

$$f_{detect}(c) = \begin{cases} 0, & \text{iff } \forall d \in D \wedge f_d(c,d) > d.r \\ 1, & \text{iff } \exists d \in D \wedge f_d(c,d) \leq d.r \end{cases} \quad (16)$$

if the executable lies within the detection radius of a detector, the function $f_{detect}(c)$ returns 1 and then the executable is malicious. Otherwise, the function $f_{detect}(c)$ returns 0 and then the executable is benign, the function $f_d(x,y)$ is the Euclidean distance between x and y.

3 Simulations and Experimental Results

In order to test the proposed method, we need benign and malicious executables. We gathered 992 benign executables and 4192 malicious executables. The benign executables were gathered from a freshly installed Windows XP installed MSOffice 2000 and Visual C++ and the malicious executables were downloaded from some FTP sites, composed of Trojans, viruses and worms. There were no duplicate programs in the data set and every example was labeled either malicious or benign by the commercial virus scanner.

Table 1 is the comparison of the experimental data set between Schultz [12] and MEDRNS (the dataset in [32] is same to [12]). Table 1 shows that the number and the types of malicious executables in MEDRNS are bigger than that of Schultz [12], and the number of benign executables is almost equal. We use more malicious executables to test the performance of our proposed approach and then confirm the detection capability of MEDRNS.

To evaluate our method, we compared MEDRNS with the methods used by Schultz [12] and Peng [32], including signature method, RIPPER, Naive Bayes, Multi-Naive Bayes and Support Vector Machine. Table 2 shows the comparison for detection performance, where the vector dimensions were four in MEDRNS. From the table 2, we can see that the signature-based method has the worst true-positive rate, but the lowest false-positive rate, and the learning-based methods have better detection performance than that of signature-based methods. It also shows that the learning-based methods can detect unknown malicious executables. At the same time, the table 2 shows that MEDRNS is a good method to detect unknown malicious executables and has higher detection performance than that of Naive Bayes, Multi-Naive Bayes and SVM, for example, the true-positive and false-positive rates of Multi-Naive Bayes are 97.76% and 6.01%, however, the true-positive and false-positive rates of MEDRNS are 98.21% and 2.22%.

4 Conclusion

Detecting unknown malicious executables is a challenging task. In this paper, a based on real-valued negative selection algorithm to detect malicious executables is proposed, which is referred to MEDRNS. Using the variable-sized self radius to represent the self space, it can construct the appropriate profile of the system, and then generates the more quality detectors, which can increase the true positive rate and decrease the false positive rate. The experiment results show that MEDRNS is an efficient method to detect malicious executables.

Acknowledgement

This work is supported by 863 High Tech Project of China under Grant No. 2013AA01A213, special technology development fund for research institutes of the Ministry of Science and Technology of China (2011EG126038 and 2013EG126063), China Postdoctoral Science Foundation (20100480074) and Applied Basic Research Program of Sichuan Provincial Science and Technology Department.
The author is grateful to people for a careful checking of the details and for helpful comments that improved this paper.

References

[1] S. Staniford, V. Paxson and N. Weaver, How to own the internet in your spare time, Proceedings of the 11th

Table 1: The comparison for datasets of Schultz [12] and MEDRNS

Data set	The number of executables	The number of malicious executables	The type of malicious executables	The number of benign executables
Schultz [12]	4266	3265	Trojan and virus	1001
MEDRNS	5184	4192	Trojan, virus and worm	992

Table 2: The comparison for detection performance of Schultz [12] and Peng [32] and MEDRNS

	TP	TN	FP	FN	Detection Rate	False Positive Rate
Signature Method [12] -Bytes	1102	1000	0	2163	33.75%	0%
RIPPER [12] -DLLs used	22	187	19	16	57.89%	9.22%
-DLL function calls	27	190	16	11	71.05%	7.77%
-DLLs with counted function calls	20	195	11	18	52.63%	5.34%
Naive Bayes [12] -Strings	3176	960	41	89	97.43%	3.80%
Multi-Naive Bayes [12] -Bytes	3191	940	61	74	97.76%	6.01%
SVM [32]	3188	938	63	77	97.64%	6.29%
MEDRNS	4177	970	22	75	98.21%	2.22%

USENIX Security Symposium. San Francisco Marriott, 149-167, August, (2002).

[2] A.K.Sood and R.J. Enbody, Targeted Cyberattacks: A Superset of Advanced Persistent Threats, IEEE Security & Privacy, **11**, 54-61, (2013).

[3] M. Xu,L. F Wu,S. H. Qi,J Xu,H. P. Zhang,Y. Z. Ren and N. Zheng,A similarity metric method of obfuscated malware using function-call graph, Journal of Computer Virology and Hacking Techniques,**9**, 35-47, (2013).

[4] K. Hassan, M. Fauzan, and A. K. Syed, Determining malicious executable distinguishing attributes and low-complexity detection, Journal in Computer Virology, **7**, 95-105, (2011).

[5] S. R. White, Open Problems in Computer Virus Research. Proceedings of Virus Bulletin Conference, (1998).

[6] P. Ször P and P. Ferrie, Hunting for Metamorphic, Proceedings of Virus Bulletin Conference September, 123-144, (2001).

[7] M. Christodorescu and S. Jha, Static Analysis of Executables to Detect Malicious Patterns. Proceedings of the 12th USENIX Security Symposium, 169-186, (2003).

[8] R. W. Steve, S. Morton and J. P. Edward, et al, Anatomy of a Commercial-Grade Immune System. IBM Research White Paper, (1999).

[9] R. W. Lo, K. N. Levitt and R. A. Olsson, MCF: A malicious code filter, Computers & Security, **14**, 541-566, (1995).

[10] G. Tesauro, J. O. Kephart and G. B. Sorkin, Neural networks for computer virus recognition. IEEE Expert, **11**, 5-6, (1996).

[11] J. Zico Kolter and Marcus, A. Maloof, Learning to Detect and Classify Malicious Executables in the Wild. Journal of Machine Learning Research, **7**, 2721-2744, (2006).

[12] M.G. Schultz, E. Eskin, F. Zadok and S. J. Stolfo, Data mining methods for detection of new malicious executables, Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, 1207-1217, (2001).

[13] M. Robert, G. Ido, P. Shay and S. Dima, et al. Detection of Unknown Computer Worms Activity Based on Computer Behavior using Data Mining, Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications, 169-177, (2007).

[14] T. Li, Computer Immunology. Publishing House of Electronics Industry, Beijing, 2004.

[15] L. de Castro and F. Zuben, Artificial Immune Systems: Part I - Basic Theory and Applications. TR - DCA 01/99, (1999)

[16] F. B. Zhang, X. Yue, D. W. Wang and L. Xi, A Principal Components Weighted Real-valued Negative Selection Algorithm, International Journal of Digital Content Technology and its Applications. **5**, 313-324, (2011).

[17] F. Y. Zhang and D. Y. Qi, Run-time malware detection based on positive selection, Journal in Computer Virology, **7**, 267-277 (2011).

[18] J. Greensmith, U. Aickelin and S. Cayzer, Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection, Proceedings of the ICARIS. LNCS, **3627**, 153-167, (2005).

[19] S. Manzoor, M.Z.Shafiq, S.M.Tabish and M. Farooq, A sense of 'danger' for windows processes, Proceedings of ICARIS, LNCS, **5666**, 220-233, (2009).

[20] D. Dervovic and J. C. Zuniga-Pflucker, Positive selection of T cells, an in vitro view. Semin. Immunol, **22**, 276-286, (2010).

[21] L. de. Castro and F. Zuben, Learning and Optimization Using the Clonal Selection Principle. IEEE Transactions on Evolutionary Computation, **6**, 239-251, (2002).

[22] H. U. Berna, K. K. Sadan, A review of clonal selection algorithm and its applications. Artificial Intelligence Review, **36**, 117-138, (2011).

[23] J. R. Al-Enezi and M. F. Abbod and S. Alsharhan, Artificial Immune Systems-models, algorithms and applications, International Journal of Research and Reviews in Applied Sciences, **3**, 118-131, (2010).

[24] S. Forrest, A. S. Perelson, L. Allen and R. Cherukuri, Self-nonself discrimination in a computer. Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press, (1994).
[25] H. G. Dai, Y. Yang and C. H. Li, Distance Maintaining Compact Quantum Crossover Based Clonal Selection Algorithm, JCIT, **5**, 56-65, (2010).
[26] T. Stibor, J. Timmis and C. Eckert, A comparative study of real-valued negative selection to statistical anomaly detection techniques, Proceedings of ICARIS'2005, LNCS **3627**, 262-275, (2005).
[27] J. Zhou and D. Dasgupta, Applicability issues of the real-valued negative selection algorithms, Proceedings of Genetic and Evolutionary Computation Conference, (2006).
[28] F. A. Gonzalez and D. Dasgupta, Anomaly detection using real-valued negative selection. Genetic Programming and Evolvable Machines, **4**, 383-403, (2003).
[29] L. N. De Castro and J. I. Timmis, Artificial immune systems as a novel soft computing paradigm. Soft Computing journal, **7**, 526-544, (2003).
[30] S. Tonegawa, The Molecules of the Immune System. Scientific American, **253**, 104-113, (1985).
[31] J. Zhou and D. Dasgupta, Real-valued negative selection algorithm with variable-sized detectors, Proceedings of Genetic and Evolutionary Computation - GECCO-2004, Part I. LNCS, **3102**, Seattle, WA, USA, Springer-Verlag 287-298, (2004).
[32] H. Peng, and J. Wang, Research of Malicious Executables Detection Method Based on Support Vector Machine, ACTA ELECTRONICA SINICA, **33**, 276 -278, (2005).

Jinquan Zeng received his M.S. degree in computer science from ZhengZhou University in 2004, and the Ph.D. degree in computer science from Shichuan University in 2008. He is currently a lecturer at School of Computer Science & Engineering, University of Electronic Science and Technology of China. His research interests include network security,computer network etc.