Effective Detector Set Generation and Evolution for Artificial Immune System

Cholmin Kim[1], Wonil Kim[2], and Manpyo Hong[1]

[1] Internet Immune System Laboratory
Graduate School of Information and Communication
Ajou University
{ily, mphong}@ajou.ac.kr
[2] College of Electronics and Information Engineering
Sejong University
wikim@sejong.ac.kr

Abstract. Human bodies defend themselves against harmful invaders by the Natural Immune System (NIS). The NIS possesses a huge set of detectors against many different invaders. Based on the essential principles of the NIS, the idea of Artificial Immune System (AIS) can be used for protecting computer resources. The AIS can detect the nonself-activity using a self-adopted detector set. The detector set is generated randomly and evolved by a Genetic Algorithm (GA). However, the random generation and the blind GA adoption decrease the performance of detector set and generate unnecessary redundancy. To overcome this problem, we propose a more efficient detector generation and evolution scheme. Using the proposed idea, the AIS can produce more effective detector set with time advantages.

1 Introduction

Detecting intrusions and anomalies in a complex computer system have the same nature as the 'Achilles and Turtle' problem. Many researchers try to find a good protection mechanism for a secure computer system, but the new intrusion skill or tool that can circumnavigate the protection mechanism is also developed immediately [1]. Nowadays, many researchers have realized that a set of specific methods to counteract a specific type of intrusion is no longer the permanent solution. They found that a complete understanding of the behavior of the system to be protected is the best way to treat the intrusion [2]. One kind of technique in this paradigm is an Artificial Immune System (AIS) [3]. The main idea of AIS comes from the human body. The human body has a fundamental protection mechanism. The Natural Immune System (NIS) is the essential part of the human self-protection mechanism [2]. The NIS can distinguish self from nonself. Self comprises all components that are necessary for normal functioning of the human body whereas nonself is foreign material that could harm normal functioning of the human body.

The basic concept of AIS is the same as NIS. As in NIS, any kind of program that tries to accesses the computer system can be categorized into self and nonself [3]. In this case, nonself is a program which has harmful function to computer system. Thus

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3037, pp. 491–498, 2004.

a nonself detector is required to protect a system from illegal programs. These detectors are generated randomly and evolved adaptively. The GA (Genetic Algorithm) is the most frequently used technique to evolve these detectors. As evolution stages process, more accurate detector set for nonselves can be generated.

The current AIS, however, has some problems in randomness and blind GA adoption. By the random generation of detector, we can obtain larger detector set but it may not guarantee that every element of this set will be certainly being useful. In fact, large portions of generated detectors are not used. The blind crossover and mutation also have the similar problems. The current AIS randomly chooses crossover point and mutation value. Thus some meaningless gene relocation and mutation can occur. Due to these problems, the performance of detectors decreases and hence relatively large storages are needed. To overcome this, we propose improved techniques for initial detector generation and evolution.

There is a difference between the previous AIS and the proposed idea. The previous AIS protect the self-file by searching some nonself-file signatures. Thus, the detector is a binary string for the file. However, the detection target of the proposed system is a system call sequence. The proposed idea will generate meaningful system call sequences and evolve to accurate one.

The rest of this paper is organized as follows: In Section 2, we introduce the brief overview of NIS, AIS and GA method used in immune systems. In Section 3, we propose the idea to improve the performance of detector set and discuss experimental results. Section 4 concludes.

2 Ideas from Immune Systems and Natural Evolution

2.1 The Natural Immune System (NIS)

The NIS consists of a multitude of cells and molecules that interact in a variety of ways to detect and eliminate infectious agents (pathogens) [2, 3]. Detection and elimination of pathogens is the consequence of trillions of cells interacting through simple and localized rules. Consequently the NIS is very robust to failure of individual components and attacks on the NIS itself [2].

The problem of detecting pathogens is often described as distinguishing self from nonself as in previous section. However, many pathogens are not harmful, and an immune response to eliminate them may damage the body. In these cases it would be healthier not to respond. Therefore it would be more accurate to say that the problem faced by the NIS is distinguishing between harmful nonself and everything else [3].

2.2 The Artificial Immune System

Using the main idea of NIS and D'haeseleer, Forrest, and Helman suggested AIS [6]. They concentrated their interest on the Negative Selection (NS). NS is a kind of NIS behaviors. It searches for the protein that binds with detectors. Detectors are generated for a protein that does not belong to self and will detect all nonself objects. The matching algorithm detects sequence of contiguous bits containing particular position and length.

The detection range of AIS can be described as follows. The string space is a set of every string which can be occurred in the system. For host-based AIS, it can be every combination of system call sequences in certain length. The string space partitioned into self strings *S* and nonself strings *N*. Nonself strings indicate some intrusion or abuse. The detector set *R* is a subset of all possible candidate detectors and is the actual set the AIS can generate. Nonself strings can be further divided into detectable nonself strings (*N´*) and holes (*H, H = N - N´*). Limitation of memory size leads to detect not every nonself strings but some portions of them. Detectable nonself strings can be obtained depending on the choice of *R*.

2.3 Protein and Computer System Call

Detector set of NIS is collection of protein strings, whereas the set of proposed AIS is system call sequences [3, 7]. There are at least 10^{16} nonself proteins for human body. The NIS will store a portion of those. On the other hands, current Linux system has about 200 system calls. Thus, if we consider the detector as size 3 string the number of detector combinations will be 8 million (200 * 200 * 200). The details of detector size selection will be described in section 3. Detector of size 7 is also frequently used. In this case the number will reach to $1.28 * 10^{16}$. If we just compare the numbers, we can find that the negative selection problem for the NIS and AIS has the similar complexity. However, because of the parallelism, the NIS is more efficient. Each lymphocyte of NIS moves totally independently and carries an individual detector. In the contrast, since the number of processors of a computer is restricted, the detector set of AIS is activated in some serial manners and needs more operations than single lymphocyte of NIS. Consequently, we need more optimized generation and evolution mechanism for the AIS detector set.

2.4 The Genetic Algorithm

The NIS appears to be able to recognize at least 10^{16} nonselves. The human genome contains about 10^{5} genes. From an information-processing perspective, recognizing an almost limitless number of foreign cells and molecules, and distinguishing these from self molecules are formidable tasks. Thus, the individual cells that comprise the immune system encode and operate the control mechanism in parallel. The genetic mechanism involving the combinatorial association of a number of gene segments underlies the construction of the receptors, so that an NIS has the genetic capability of expressing over 10^{16} different receptors [4, 5]. To implement this in the AIS, the AIS model uses GA which is an idealized computational model of Darwinian evolution based on the principles of genetic variation and natural selection.

In order to use the GA to evolve a set of bit strings that have the property of target, the GA must concern the characteristics of the target gene. In our AIS, the target is a system call sequences.

GA is typically used to evolve a population in which each member specifies one candidate solution and the individual solution competes with each other. The majority of GA analysis has focused on these optimization applications. From a pattern-recognition perspective, the nonself recognition problem can be viewed as a string-

matching problem in which the task is to discover sets of common substrings that collectively the population of nonself strings. The learning task is thus to evolve a set of nonselves that meets the coverage requirement.

3 The Proposed Detector Set Generation and Implementation

We briefly introduce the AIS and the detectors used in AIS. Since the random generation does not guarantee the usefulness of detector, some unnecessary detectors could be generated. The detector generation system, however, could not decide the usefulness of the newly generated strings. By this reason, the previous AIS maintain garbage detectors and need relatively large storage space. In addition to these problems, the number of detectors determines the time that is needed to compare a detector with a target. Consequently, excessive number of detectors also decreases the performance of matching operation.

On the other hand, the GA in AIS has problems too. In GA, evolution is done by the crossover and the mutation. In the crossover shuffle process, input genes are divided into several sub-genes and these sub-genes change their positions each other. Since the detector strings in our system are consisting of system call sequences, they have some contextual meanings. Thus, swapping of sub genes can make brand new object gene and produce a useless detector. With the same reason the mutation has a problem too. A blind mutation will generate garbage too.

In order to overcome these problems we propose three techniques, filtered initial generation of detector, filtered detector crossover and biased detector mutation.

It is observed that 3 and 7 is the best size for identifying the system call sequences [7]. We use size 3 sequences in first heuristic and 7 in the others.

3.1 Class Based Detector Generation

To improve the initial detector generation phase, we uses some strategic selection of detector. Since the detector strings consist of system call sequences, they have contextual meanings and contain similar string with both normal and abnormal system call sequences. Thus we compare newly generated one with exist strings and if the new one is not similar with any of exist strings, we discard it.

The method we propose for initial filtering constructs the detector set based on a context. In this method we first generate a random system call number, and decide a detector class which the first system call number is belonging. Figure 1 describes the mechanism.

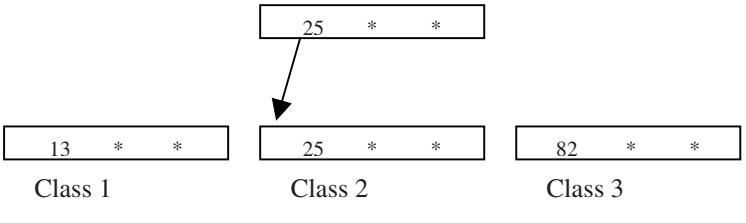

Fig. 1. Initial detector generation and classification

Since we know the belonging class of current detector after the generation of the first system call number, we can remove the system call numbers which are quite different from other members in the class during the remaining system call generation phase. Table 1 shows the system call sequences of length 3 during an intrusion. The sequences are sorted and duplications are eliminated. It is extracted from sunsendmailcp (SSCP) intrusion log files [8].

SSCP script uses a special command line option to cause sendmail to append an email message to a file. By using this script on a file such as /.rhosts, a local user may obtain root access.

Table 1. System call sequences of length 3 during an intrusion

Class 1	2	5	23	**Class 9**	19	128	3
	2	11	17		19	128	23
	2	50	27	**Class 10**	23	4	50
Class 2	3	3	5		23	45	3
	3	3	112		23	50	27
	3	4	14	**Class 11**	27	2	50
	3	19	32		27	4	27
	3	93	88		27	50	3
	3	112	19	**Class 12**	45	3	112
Class 3	4	3	3	**Class 13**	50	3	93
	4	14	112		50	27	4
	4	18	50		50	27	50
Class 4	5	4	18		50	50	27
	5	23	4	**Class 14**	88	167	17
	5	23	50	**Class 15**	112	3	19
Class 5	9	3	5		112	50	50
	9	9	3	**Class 16**	128	3	3
Class 6	11	17	4		128	3	4
Class 7	14	3	112		128	3	112
	14	112	19		128	4	18
Class 8	17	5	4		128	23	45
	17	112	19		128	112	3
Class 9	19	112	4	**Class 17**	167	17	5

Since these sequences are not found in normal system call sequences of sendmail daemon, they can be considered as nonself. As we can see in this case there are only 17 classes if we classify the sets by the first system call number. Logically there can be about 200 classes, but almost every class except these 17 classes are useless.

This classification can be adapted to more than one layer and will provide more hierarchical class architecture of detector.

3.2 Filtered Detector Crossover

In order to avoid generating unnecessary detectors in crossover, we also use context information of system call sequence. When we perform crossover operation, we divide the genes into substrings. By the consequence of shuffle substrings, the resulting string will have a quite different system call combination. If we compare the adjacent system calls with normal or abnormal system call sequences, we can

conclude whether the generated detector is a brand new one or not. Figure 2 shows the selected position for comparison.

In the system call sequence size 7, we can divide the sequence into two sub-gene of size 3 and 4. Thus the comparison locations are the 4th and the 5th. We will show the experimental result in section 4.

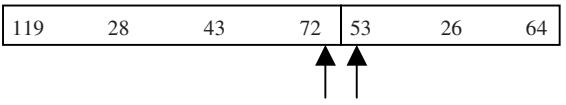

Fig. 2. The comparison location of system call

3.3 Biased Detector Mutation

There are about 200 different system call services. These system calls have different usage frequency. However, previous works performed uniform mutation. Due to the different frequencies, the random number selection must be biased. If it is not biased, the generated number can not represent the real system call number distribution.

We have used biased roulette for the random number. First we gather the frequencies of the system call number from intrusion log files. Figure 3 shows the frequencies of the system calls. As seen in the graph, intrusion log files have different distribution to the normal one.

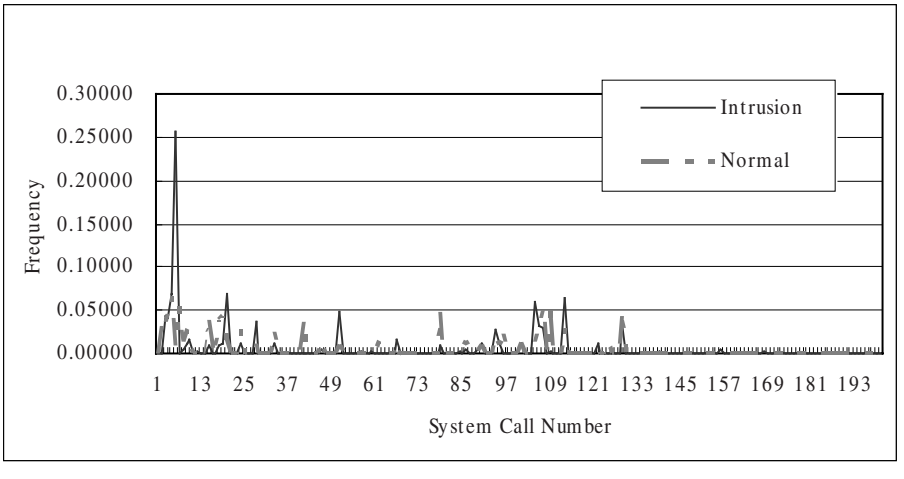

Fig. 3. System calls usage frequency distributions

3.4 Experiment and Evaluation

We experimented our ideas using the logged data of Immune System Laboratory in New Mexico University by the following steps [8]. First, we generated our detector set by the proposed ideas using single attack log file. Then we matched the generated set with another attack log file and normal log files to evaluate true positive (TP) and

false positive (FP) cases. We performed the same experiment with the blind detector generation scheme. Table 2 shows the result of the two experiments.

Table 2. The results of experiment 1 and 2.

	File size	# of matched detector in ex1 (Proposed Method)	# of matched detector in ex1 (Blind Method)	# of matched detector in ex2 (Proposed Method)	# of matched detector in ex2 (Blind Method)
Intrusion log files	1.1K	36/40	35/374	107/503	108/3013
	2.0K(1)	36/40	36/374	130/503	131/3013
	2.0K(2)	37/40	36/374	131/503	131/3013
Normal log files	1K	0/40	15/374	2/503	16/3013
	312K	1/40	30/374	4/503	35/3013
	5818K	1/40	55/374	7/503	56/3013

The first experiment shows the exactness of class based detector generation. We have generated detector set of size 40 with 20 classes. Then we search the generated detector set from three different sunsendmailcp intrusion log files that are not used in detector set generation. The log file sizes are 1.1K, 2.0K and another 2.0K bytes respectively. The intrusion log file format is a simple ASCII text file. At least 36 of 40 detectors were found in every intrusion log files. Three detectors are not founded in any intrusion log files. One detector is found in only the second log file of size 2.0K. We also generated the detector set in random manner without any class. It generates 374 detectors. But it found the same number of strings with the proposed method.

We matched the same detector sets with the three normal log files. The file sizes are 1.0K, 312.0K and 5818.0K bytes respectively. Normal log files have relatively large sizes than attack log files because it can be gathered in long period. Only 1 of our detectors is found in 2 log files. However, many detectors in blind scheme has found in each log files. It means our scheme decrees FP rate.

The second experiment show how effectively the proposed idea generates an accurate detector set by filtering the unnecessary detector in crossover and mutation steps. It was done for the system call sequences of size 7. We divided the original system call sequence of size 7 into two sub-genes. Sizes of sub-genes are 4 and 3. Thus the crossover point is offset 4 of the sequence. We have generated 503 of evolved detectors with our scheme. These detectors are evolved by following steps. First, we crossover genes then leave detectors that have frequently appeared system calls in position 4 and 5. Then mutate the detector with biased policy. The mutation rate is 1%. 3013 detectors are generated without our method. These detectors are crossover without filtering and mutated in random manner. As seen in Table 2, False Positive (FP) rates of the proposed method in both experiments are very low comparing with the blind method.

4 Conclusion

GA is one of popular methods used to detect nonselves in Artificial Immune System (AIS). However, unnecessary detectors are still generated due to the random initialization and blind GA adoption. We proposed a novel approach to solve these problems in this paper. The first one is classification of detector, the second one is crossover point comparison, and the third one is biased mutation. We used a system call sequence data provided by Immune System Laboratory in New Mexico University. By the experiments, we showed that the detector set that is generated and evolved by the proposed idea maintain high detection rate in spite of using small space. Our idea can be easily extended to other intrusion detection areas.

References

1. A. Acharya and M. Raje.: MAPbox: Using Parametererized Behavior Classes to Confine Aplications. In Computer Science Technical Report TRCS99-15, Department of Computer Science University of California, Santa Barbara (1999).
2. S. Forrest and S. Hofmeyr.: Engineering an immune system. Graft Vol. 4:5 (2001) 5-9.
3. S. Hofmeyr and S. Forrest.: Architecture for an Artificial Immune System. Evolutionary Computation 7(1), Morgan-Kaufmann, San Francisco, CA (2000) 1289-1296.
4. S. Forrest.: Genetic Algorithms. Computing Surveys Vol. 28:1 (1996) 77-80.
5. S. Forrest, B. Javornik, R.E. Smith, and A.S. Perelson.: Using Genetic Algorithms to Explore Pattern Recognition in the Immune System. Evolutionary Computation, Vol. 1, No. 3 (1993) 191-211.
6. P. D'haeseleer, S. Forrest, and P. Helman.: A Distributed Approach to Anomaly Detection (1997).
7. R. Sekar, M. Bender, D. Dhurjati and P. Bollineni.: A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors, Proceedings of 2001 IEEE Symposium on Security and Privacy (2000) 144-155.
8. Computer Immune System Data Sets, Computer Immune System Laboratory, New Mexico University, http://www.cs.unm.edu/~immsec/data-sets.htm.
9. R. Biischkes, M. Borning and D. Kesdogan.: Transaction-based Anomaly Detection. In Proceedings of the Workshop on Intrusion Detection and Network Monitoring (1999).
10. J. Fritzinger and M. Mueller.: Java security. Technical report, Sun Microsystems, Inc (1996).
11. M. Opera, S. Forrest.: How the immune system generates diversity: Pathogen space coverage with random and evolved antibody libraries. University of New Mexico (1998).
12. S. A. Hofmeyr, S. Forrest.: Immunity by Design. University of New Mexico (1999).