An Evaluation of Negative Selection in an Artificial Immune System for Network Intrusion Detection

Jungwon Kim and Peter J. Bentley

Department of Computer Science
University College London
Gower Street, London
U.K.
Email: {J.Kim, P.Bentley}@cs.ucl.ac.uk

Abstract

This paper investigates the role of negative selection in an artificial immune system (AIS) for network intrusion detection. The work focuses on the use of negative selection as a network traffic anomaly detector. The results of the negative selection algorithm experiments show a severe scaling problem for handling real network traffic data. The paper concludes by suggesting that the most appropriate use of negative selection in the AIS is as a filter for invalid detectors, not the generation of competent detectors.

1 INTRODUCTION

The biological immune system has been successful at protecting the human body against a vast variety of foreign pathogens (Tizard, 1995). A growing number of computer scientists have carefully studied the success of this competent natural mechanism and proposed computer immune models for solving various problems including fault diagnosis, virus detection, and mortgage fraud detection (Dasgupta, 1998; Kephart et al,1995).

Among these various areas, intrusion detection is a vigorous research area where the employment of an artificial immune system (AIS) has been examined (Dasgupta, 1998; Kim and Bentley, 1999b; Hofmeyr, 1999; Hofmeyr and Forrest, 2000; Forrest and Hofmeyr, 2000). The main goal of intrusion detection is to detect unauthorised use, misuse and abuse of computer systems by both system insiders and external intruders. Currently many network-based intrusion detection systems (IDS's) have been developed using diverse approaches (Mykerjee et al, 1994). Nevertheless, there still remain unresolved problems to build an effective network-based IDS (Kim and Bentley, 1999a). As one approach of providing the solutions of these problems, previous work (Kim and Bentley, 1999a) identified a set of general requirements for a successful network-based IDS and three design goals to satisfy these requirements: being distributed, self-organising and lightweight. In addition, Kim and Bentley (1999a) introduced a number of remarkable features of human immune systems that satisfy these three design goals. It is anticipated that the adoption of these features should help the construction of an effective network-based IDS.

An overall artificial immune model for network intrusion detection presented in (Kim and Bentley, 1999b) consists of three different evolutionary stages: negative selection, clonal selection, and gene library evolution. This model is not the first attempt to develop an AIS for network intrusion detection. Various approaches to build an AIS have been attempted mainly by implementing only a small subset of overall human immune mechanisms (Dasgupta, 1998). This is because the nature of human immune systems is very complicated and sophisticated and thus it is very difficult to implement perfect human immune processes on a computer. However, as seen from other immunology literature (Paul, 1993; Tizard, 1995), an overall immune reaction is the carefully co-ordinated result of numerous components such as cells, chemical signals, enzyme, etc. Therefore, the omission of crucial components in order to make the development of AIS simpler and more applicable may detrimentally affect the performance of an AIS. This implies that appropriate artificial immune responses can be expected only if the roles of crucial components of human immune systems are correctly understood and they are implemented in the right way.

In this paper, we continue our effort to understand the roles of important components of artificial immune systems especially for providing appropriate artificial immune responses against network intrusions. Following our previous work identifying three different evolutionary stages: negative selection, clonal selection, and gene library evolution, of AIS by extensive literature study (Kim and Bentley, 1999a; 1999b), this paper focuses on the investigation of the roles of first stage: negative selection. With implementation details of this stage, this work presents how and which aspects of negative selection can contribute to the development of an effective network-based IDS.

2 BACKGROUND

2.1 NEGATIVE SELECTION OF THE HUMAN IMMUNE SYSTEM

An important feature of the human immune systems is its ability to maintain diversity and generality. It is able to detect a vast number of antigens with a smaller number of antibodies. In order to make this possible, it is equipped with several useful functions (Kim and Bentley, 1999a). One such function is the development of mature antibodies through the gene expression process. The human immune system makes use of gene libraries in two types of organs called the thymus and the bone marrow. When a new antibody is generated, the gene segments of different gene libraries are randomly selected and concatenated in a random order, see figure 1. The main idea of this gene expression mechanism is that a vast number of new antibodies can be generated from new combinations of gene segments in the gene libraries.

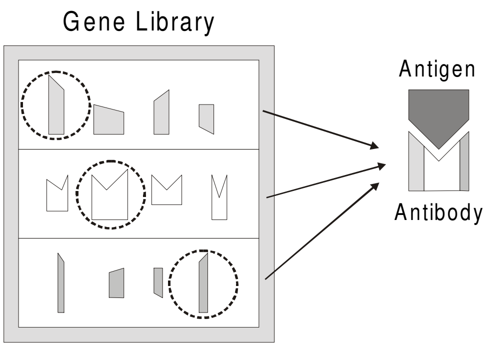

Figure 1 Gene Expression Process

However, this mechanism introduces a critical problem. The new antibody can bind not only to harmful antigens but also to essential self cells. To help prevent such serious damage, the human immune system employs negative selection. This process eliminates immature antibodies, which bind to self cells passing by the thymus and the bone marrow. From newly generated antibodies, only those which do not bind to any self cell are released from the thymus and the bone marrow and distributed throughout the whole human body to monitor other living cells. Therefore, the negative selection stage of the human immune system is important to assure that the generated antibodies do not to attack self cells.

2.2 THE NEGATIVE SELECTION ALGORITHM

Forrest et al (1994; 1997) proposed and used a negative selection algorithm for various anomaly detection problems. This algorithm defines 'self' by building the normal behaviour patterns of a monitored system. It generates a number of random patterns that are compared to each self pattern defined. If any randomly generated pattern matches a self pattern, this pattern fails to become a detector and thus it is removed. Otherwise, it becomes a 'detector' pattern and monitors subsequent profiled patterns of the monitored system. During the monitoring stage, if a 'detector' pattern matches any newly profiled pattern, it is then considered that new anomaly must have occurred in the monitored system.

This negative selection algorithm has been successfully applied to detect computer viruses (Forrest et al., 1994), tool breakage detection and time-series anomaly detection (Dasgupta, 1998) and network intrusion detection (Hofmeyr, 1999; Hofmeyr and Forrest, 2000; Forrest and Hofmeyr, 2000). Besides these practical results, D'haeseleer (1997) showed several advantages of negative selection as a novel distributed anomaly detection approach.

3 ALGORITHM OVERVIEW

This work used a negative selection algorithm to build an anomaly detector. This was achieved by generating detectors containing non-self patterns. The overview of this algorithm is provided in figure 2 and 3. The negative selection algorithm for network intrusion detection used in this paper follows the algorithm of Forrest et al (1994, 1997), described in the previous section. 'Self' was built by profiling the activities of each single network connection. The detail of self profiling is described in the next section.

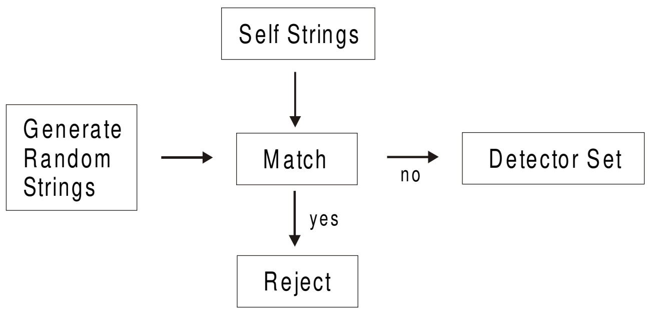

Figure 2 Detector Set Generation of a Negative Selection Algorithm (Forrest et al, 1995)

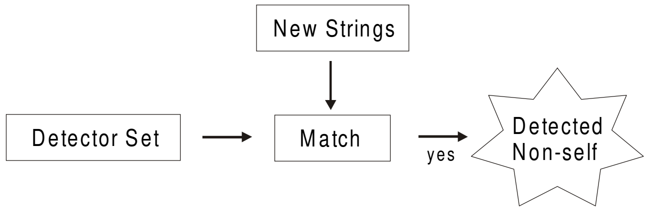

Figure 3 Non-Self Detection by a Detector Set

Even though this work follows the implementation details of Forrest et al's negative selection algorithm, there are two implementation details different from Forrest et al (1994, 1997). In the encoding of detectors, each gene of a detector has an alphabet of cardinality 10 with values from '0' to '9' and the allele of this gene indicates the 'cluster number' of corresponding field of profiles. As presented in the next section, the self profile built from the first data set has 33 fields and this number determines the total number of corresponding genes in the detectors. From these 33 fields, the values of 28 fields are continuous and the values of the other 5 fields are discrete. Specifically, the continuous values of 28 fields show a wide range of values. In order to handle this various and broad range of values, an overall range of real values for each field is sorted. Then, this range is discretised into a predefined number of clusters. The lower bound and higher bound of each cluster are determined by ensuring that each cluster contains the same number of records. This modification is necessary in order to save the length of encoded detector.

Furthermore, our implementation of measuring the similarity between a generated detector and a self profile is operated at the phenotype level while Forrest et al's (1994, 1997) is performed at the genotype level. In order to measure the similarity between a given detector and a self, the genotype of a detector is mapped onto a phenotype. The phenotype mapped from the evolved genotype is represented in a form of a detector pattern. As shown in figure 5, a field of a detector phenotype is represented by an interval having a lower bound and a higher bound while a field of a self phenotype is described by one specific value. Hence, the first step of measuring the similarity checks whether a value of each field of a self pattern belongs to a corresponding interval of a detector phenotype. When any value of a self pattern field is not included in its corresponding interval of a detector phenotype, these two fields are not matched. Similarly, for a nominal type of field, two fields match when the values of fields are identical.

The final degree of similarity between a given detector and self example follows the same matching function of Forrest et al (1994), the r-contiguous matching function. Thus, the degree of similarity is measured simply by counting the matching corresponding fields. For instance, if an activation threshold, r, is set as 2, the detector phenotype and self phenotype in the figure 4 will match since two contiguous fields, "Number of Packet" and "Duration", match and this number of contiguous matching fields equals to the activation threshold. However, if this threshold is set as 3, it is regarded that two phenotypes do not match.

Detector Phenotype =
(Number of Packet = [10, 26], Duration = [0.3, 0.85],
Termination = `half closed', … etc)

Self Phenotype =
(Number of Packet = 14, Duration = 0.37,
Termination = `normal', ….etc)

Figure 4: A Detector Phenotype and a Self Phenotype

4 NETWORK TRAFFIC DATA VS NETWORK INTRUSION SIGNATURE

The data chosen for this work was collected for a part of the 'Information Exploration Shootout', which is a project providing several data sets publicly available for exploration, discovery and collecting the results of participants[1]. The set used here was created by capturing TCP packet headers that passed between the intra-LAN and external networks as well as within the intra-LAN. This set consists of five different data sets. The TCP packet headers of the first set were collected when no intrusion occurred and the other four sets were collected when four different intrusions were simulated. These intrusions are: *IP spoofing attack, guessing rlogin or ftp passwords, scanning attack* and *network hopping attack.* The details of attack signatures and attack points of the four different attacks are not available.

The data originally had the fields of network packets capturing tool's format such as time stamp, source IP address, source port, destination IP address, destination port, etc. However, the primitive fields of captured network packets were not enough to build a meaningful profile. Consequently, it was essential to build a data-profiling program to extract more meaningful fields, which can distinguish "normal" and "abnormal". Many researchers have identified the security holes of TCP protocols (Porras and Valdes, 1998; Lee, 1999) and so the fields used by our profiles were selected based on the extensive study of this research. They were usually defined to describe the activities of each single connection.

The automated profile program was developed to extract the connection level information from TCP raw packets and it was used to elicit the meaningful fields of the first data set.

For each TCP connection, the following fields were extracted:

- Connection identifier: each connection is defined by four fields, initiator address, initiator port, receiver address and receiver port. Thus, these four fields are included in the profile first in order to identify each connection.
- Known port vulnerabilities: many network intrusions attack using various types of port vulnerabilities. There are fields to indicate whether an initiator port or a receiver port potentially holds these known vulnerabilities.
- 3-way handshaking: TCP protocol uses 3-way handshaking for a reliable communication. When some network intrusions attack, they often violate the 3-way handshaking rule. Thus, there are fields to check the occurrences of 3-way handshaking errors.
- Traffic intensity: network activities can be observed by measuring the intensity over one connection. For example, number of packets and number of kilobytes

[1] Available at http://iris.cs.uml.edu:8080/ network.html.

for one specific connection can describe the normal network activity of that connection.

Thus, in total, self profile fields had 33 different fields for the data set. Even though the network profile fields were extracted to describe a single connection activity, the data used in this research was too limited to apply this initial profile. The limit was that the data was collected for a quite short time, around 15~20 minutes. During this brief period, most different connections were established only once. An insufficient quantity of data was collected to build different connection profiles. Therefore, it was necessary to group different connections into several *meaningful* categories until each category had a *sufficient number* of connections to build a profile. Consequently, a total number of connections for each potential profile category were counted.

First of all, the data was categorised into two different groups: 'inter-connection' and 'intra-connection'. Inter-connection was the group of connections that were established between internal hosts and external hosts, and intra-connection was the group of connections that were established between internal hosts. Furthermore, to preserve anonymity, all internal hosts had a single fake address '2' and any extra information about external hosts and network topology was not provided. Therefore, the profiles according to specific hosts were insufficient. Instead, in this research, only the profiles of specific ports on any hosts were considered.

According to various possible categories, the established connection number of each profile was counted. From each case, apart from a profile class that had more than 100 connections, other profile classes were again grouped into other different classes until each class had more than 100 connections. Finally, 13 different self profiles were built. Their class names and the number of established connections are shown in table 1.

In table1, the class column of inter-connection is shown as: {(a,b),(c,d)}, where 'a' is an internal host, 'b' is a internal port number, 'c' is a external host address and 'd' is an external port number. Hence, the connection is established between (a,b) and (c, d). For the class column of intra-connection, 'a' is an internal host address, 'b' is an internal port number, 'c' is an internal host address and 'd' is an internal, port number. * indicates 'any' host address and 'any' port number. In addition, "well-known" shows the ports in the range 0 to 1023 are trusted ports. These ports are restricted to the superuser: a program must be running as root to listen to a connection. The port numbers of commonly used IP services, such as *ftp, telnet, http*, are fixed and belong to this range. But, many common network services employ an authentication procedure and intruders often use them to sniff passwords. It is worthwhile to monitor these ports separately from the other ports. Therefore, if the number of connections for any profile category, which is based on a specific port on any hosts, is not sufficient, these categories are regrouped into two new classes, a "well-known" port and a "not well-known" port.

Table 1: Self Profiles

Inter-connection	
Class	Number of Connection
{(2, *), (*, 80)}	5292
{(2, *), (*, 53)}	919
{(2, *), (*, 113)}	255
{(2, *), (*, 25)}	192
{(2, *), (*, well-known)}	187
{(2, *), (*, not well-known)}	756
{(2, 53), (*, *)}	940
{(2, 25), (*, *)}	352
{(2, 113), (*, *)}	145
{(2, well-known), (*, *)}	114
{(2, not well-known), (*, *)}	6050
Intra-connection	
{(2, *), (2, well-known)}	190
{(2, *), (2, not well-known)}	189

5 EXPERIMENT OBJECTIVE

Although previous work using a negative selection algorithm for anomaly detection (Forrest et al 1994; Dasgupta 1998; Hofmeyr, 1999) showed promising results, there had been little effort to apply this algorithm on vast amounts of data. One distinctive feature of a network intrusion detection problem is that the size of data, which defines "self" and "non-self", is enormous. In order for this algorithm to be adopted to a network-based IDS, it is important to understand whether this algorithm is capable of generating detectors in a reasonable computing time. In addition, it is essential to examine whether its tuning method, which derives an appropriate number of detectors to gain a good non-self detection rate, works when it is used on the huge size of real network data. Therefore, a series of experiments were performed to investigate these two significant features of the negative selection algorithm.

6 DATA AND PARAMETER SETTING

6.1 SETTING

As presented in section 4, the data used in this work produced thirteen different self profiles. From 13 different self sets, one self set, {(2, *), (*, 25)} in table 1, which has relatively smaller number of examples, 192, was selected for the following experiments. From the total of 192 examples of the selected self profile, 154 examples were used for generating detectors and 38 examples were applied for testing generated detectors. In addition, the detectors were tested on five different test sets. The first four sets were collected when four different intrusions were simulated (as explained in section 4) and the last set was created by generating random strings. These five sets have 273, 190, 1151, 273 and 500 examples respectively.

As described in section 3, the negative selection algorithm used in this paper employed the r-contiguous matching function. For the following experiments, its matching threshold should be defined. In order to define this number, the formulas to approximate the appropriate number of detectors when a false negative error is fixed (D'haeseleer, 1997; Forrest et al, 1994) were used. These formulas are as follows (Forrest et al, 1994) :

$$P_m \approx m^{-r}[(l-r)(m-1)/m+1] \text{ .... (1)}$$

$$P_m \approx \frac{1}{N_s} \text{ .... (2)}$$

where,

P_m = the matching probability between a detector string and a randomly chosen self string,

N_s = the number of self strings,

m = the detector genotype alphabet cardinality,

l = the detector genotype string length and

r = the threshold of r-contiguous matching function.

Since N_s, m, l are already known, r can be calculated by using equation (1) and (2). The calculated r was used in the following equation in order to derive an appropriate number of detectors, N_r, and a total number of trials to generate these detectors, N_{r_0}, when the false negative error, P_f, is fixed (Forrest et al, 1994).

$$N_r = \frac{-\ln P_f}{P_m} \text{ ....(3) and}$$

$$N_{r_0} = \frac{-\ln P_f}{P_m \times (1-P_m)^{N_S}} \text{ ....(4)}$$

The selected self set, {(2, *), (*, 25)}in table 1, was used for calculating N_r and N_{r_0} when P_f is fixed. Table 2 shows calculated N_r and N_{r_0} using (3) and (4) when P_f and r have various values.

(D'haeseleer, 1997; Forrest, et al, 1994) showed that the larger matching threshold drives the creation of less general detectors and thus it requires a larger number of detectors but a smaller number of detector generation retrials. This is because less general detectors are easier to avoid the matching a self profile. N_r and N_{r_0} in table 2 follows the same tendency.

Table 2 Number of required detectors, N_r and number of trials to generate required number of detectors, N_{r0} when false negative error, P_f, and the threshold r of r-contiguous matching function are given. These numbers are calculated when a self string length, l = 33, an alphabet cardinality, m = 10 and the number of self strings, N_S = 192.

P_f	$r = 3$		$r = 4$	
	N_r	N_{r_0}	N_r	N_{r_0}
0.2	51	21953	535	955
0.1	73	31382	766	1366
0.05	95	40829	997	1777
0.01	146	62765	1532	2733

Even though this formula is clearly useful to predict the appropriate number of detectors and its generation number, its predicted number showed how infeasible this approach is when it is applied on a more complicated but more realistic search space. For instance, when the expected false negative error rate is fixed as 20%, its predicted detector generation trial number is 51 and the appropriate number of generated detectors is 21935 for the matching threshold is 3. Similarly, when we define the matching threshold as 4, it predicted 535 for the former and 955 for the latter. In addition, it was observed that when we fixed the matching threshold number as four and ran the system, the system could not manage to generate any single valid detector after one day. None of these cases seem to provide any feasible test case in terms of computing time. This results certainly did not follow the predicted detector generation trial number.

Thus, for the following experiments, we generated valid detectors by setting a matching threshold number that allowed a system to generate a valid detector in a reasonable time. It was observed that the average time of single successful detector generation took about 70sec CPU time and the average number of trials to generate a valid detector was 2~3 when a matching threshold was nine. These results were gained after running the negative selection algorithm for preliminary experiments. This number is used as the matching threshold for the following experiments. The details of these experiment results are described in the next section.

7 EXPERIMENT RESULT

Five different sets of detectors were generated after the AIS with the negative selection was run five times. Even though the matching threshold, 9, gave reasonable computing time to generate a valid detector, it requires a large number of detectors to gain a good non-self

Table 3 The mean and variance values of intrusion and self detection rates when detector set size varies
The means values are followed by the variances in the parentheses.

Num. Of Detectors	Intrusion1 (%)	Intrusion 2 (%)	Intrusion 3 (%)	Intrusion 4 (%)	Intrusion 5 (%)	Test Self Set (%)
100	9.45(2.11)	10.11(8.50)	11.14(9.44)	10.62(4.03)	0.48(0.012)	7.89(17.31)
200	11.72(5.37)	11.58(13.71)	12.98(11.52)	12.89(10.43)	0.88(0.092)	9.47(36.70)
300	12.53(4.25)	11.89(13.24)	13.73(9.48)	13.63(9.15)	1(0.12)	10(29.08)
400	13.33(2.79)	12.32(11.30)	14.58(10.18)	14.36(6.87)	1.28(0.112)	10.53(31.16)
500	13.55(3.15)	12.74(13.63)	14.89(10.40)	14.51(7.35)	1.36(0.068)	11.05(25.62)
600	13.77(3.80)	13.16(11.91)	15.07(10.24)	14.65(8.12)	1.68(0.412)	11.58(29.78)
700	13.77(3.80)	13.16(11.91)	15.26(9.46)	14.65(8.12)	2.04(0.388)	11.58(29.78)
800	13.92(4.09)	13.26(11.27)	15.45(10.09)	14.80(8.22)	2.04(0.388)	11.58(29.78)
900	14.14(4.13)	13.47(10.47)	15.67(9.69)	15.02(8.52)	2.08(0.352)	12.63(46.40)
1000	**14.21(4.32)**	**14.08(11.52)**	**15.90(8.71)**	**15.09(8.68)**	**2.28(0.312)**	**12.63(46.40)**

detection rate. After taking into account practically reasonable time to generate a whole data set, up to 1000 valid detectors were generated per run. All experiments were run on a PC with AMD K6-2 400Mhz processor and 128M RAM.

Table 4 Time is an avarage time of single detector generation and Trial is an average trial number to generate a single detector. The average values are followed by the standard deviations in parentheses.

System Run	**Time (Sec)**	**Detector Generation Trial**
1	58.71(26.85)	2.80(2.16)
2	67.29(28.88)	2.21(1.65)
3	73.75(33.72)	2.81(2.22)
4	78.48(39.86)	3.12(2.69)
5	69.64(26.62)	2.72(2.07)
Average	**71.81(32.75)**	**2.63(2.14)**

Table 3 shows the average time of single successful detector generation and the average number of trials to generate a valid detector. Compared to the result when the matching threshold is four, which did not generate any single detector after 24 hours, these results certainly look more applicable. We monitored five different non-self sets and one previously unseen self sets after every 100 detector generation and the monitor results of five different runs are shown in table 4. The overall non-self detection rate was very poor: less than 16%. In particular, the non-self detection rate for the last intrusion set, which was artificially generated by random strings, is extremely low and its maximum average non-self detection rate reaches only 2.28%. In addition, its average false positive detection rate, which is self detection rate by a detector set, shows 12.63% and this rate is not hugely different from the other four average non-self detection rates except intrusion 5. This implies that the collected self and non-self sets perhaps have some overlapping patterns because they showed quite similar detection rates. Thus generated detector sets completely failed to distinguish the hidden self and non-self patterns.

These poor results were anticipated. This is because the matching threshold was set in order to obtain a reasonable detector generation time. If, for example, we wanted a more usable 80% non-self detection rate, 643775165 detectors would be required (this number is also obtained from equation 3). The largest size of a generated detector set, 1000, was much smaller than this number and this caused such poor results. In addition, each run already took about 20 hours[2] to generate 1000 detectors. If we wished to generate 643775165 detectors, it would require 12517850.4 hours, or about 1,429 years on the same computer. According to Moore's Law, the processing speed of computers doubles every 18 months. We would have to wait around 35 years before the average processing speed of computers became fast enough to generate these detectors in an hour - and this is for just 15~20 minutes of a tiny subset of the network traffic data.

8 ANALYSIS

In contrast to the promising results shown in Hofmeyr's negative selection algorithm for network intrusion detection (Hofmeyr, 1999; Hofmeyr and Forrest, 2000), the results of these experiments raise doubt whether this algorithm should be used for network intrusion detection. In order to answer this question, the negative selection algorithm for network intrusion detection is analysed in detail.

The main problem of the negative selection algorithm is a severe scaling problem. Unlike previous work using

[2] Since it took, on average, 72 seconds to generate each detector, 72000 seconds were needed to produce 1000 detectors. 72000 seconds are 20 hours.

the negative selection algorithm for anomaly detection, here we apply a much larger "self" set to the negative selection algorithm. The definition of larger "self" set was essential to cover diverse types of network intrusions. For instance, (Hofmeyr 1999; Hofmeyr and Forrest, 2000) defines "self" as a set of normal pairwise connections between computers. These include connections between two computers in the LAN and between one computer in the LAN and external computers. The connection between computers is defined by "data-path-triple": (the source IP address, the destination IP address, the port called for this connection). This self definition is chosen based on the work by (Heberlein, et al, 1990). However, as other IDS literature pointed out (Lee, 1999), this self definition is very limited in order to detect various types of network intrusions and it will certainly be impossible to detect some intrusions that occur within a single normal connection such as unauthorised access from a remote machine.

However, as observed in section 4, when the self definition widens, a binary string to encode a detector lengthens. As the result of long length of binary detectors, an appropriate number of detectors to gain an acceptable false negative error becomes huge and thus requires an unacceptably long computation time. Our previous experiment results clearly show this problem.

It should be noted that Hofmeyr (1999) developed a refined theory and multiple secondary representations and these help to reduce the number of trials to generate detectors on structured self as much as three orders magnitude less. These methods made the distribution of a self set clump and it resulted in the reduction of the number of detector generation trials. However, the refined theory and secondary representations add extra space and computing time. More importantly, all of the suggested secondary representations, such as pure permutation, imperfect hashing and substring hashing, are matching rules which check matching only on genotypes. Unfortunately, matching rules that operate only at the genotype level have a weakness to be applied for a network intrusion detection problem. This deficiency can be explained by unravelling the problem of r-contiguous matching function.

We used the r-contiguous rule to check the match between a given detector and antigen. The main purpose of using it was in order to employ the formula to approximate an appropriate number of detectors to gain a certain non-self detection rate. However, the r-contiguous matching rule is too simple to determine the matching between rather complicated and high-dimensional patterns. It has been already known that most rules to represent intrusion signatures describe correlation among significant network connection events and temporal co-occurrences of events (Lee, 1999; Porras, 1998). Since the r-contiguous bit matching only measures the contiguous bits of genotypes of given two strings, it is hard to guarantee that the r-contiguous bit matching can catch this kind of correlation from given self and non-self patterns. The wider range of self definition shown in section 4 is also suggested in order to extract this type of correlation from given self and non-self network traffic examples.

But, if any new matching function is employed, D'haeseleer's (1997) formula is no longer valid. There is no way to tune the right number of detectors for negative selection. Therefore, this difficulty may force the negative selection algorithm to adopt an arbitrary number of detectors and this may cause an unexpectedly low detection accuracy or inefficient computation by generating more than sufficient number of detectors. In addition, D'haeseleer's (1997) new detector generation algorithms using a linear-time algorithm and a greedy algorithm that guarantees a liner time of detector generation is also not applicable when a different matching function is used.

In summary, it is necessary to use a more sophisticated matching function to determine the degree of correlation among significant network connection events and temporal co-occurrences of events. This requires deriving a new way to tune an appropriate number of detectors, which can be used for more sophisticated matching function.

These drawbacks of the negative selection algorithm made the AIS struggle to monitor vast amount of a network self set despite its other important features[3]. Consequently, the initial results of our experiments motivated us to re-define the role of negative selection stage within an overall network-based IDS and design a more applicable negative selection algorithm, which follows a newly defined role. As much of the other immunology literature (Tizard, 1995) addresses that the antigen detection powers of human antibodies rise from the evolution of antibodies via a clonal selection stage. While the negative selection algorithm allows the AIS to be an invaluable anomaly detector, its infeasibility to be applied on a real network environment is caused from allocating a rather overambitious task to it. To be more precise, the job of a negative selection stage should be restricted to tackle a more modest task that is closer to the role of negative selection of human immune system. That is simply filtering the harmful antibodies rather than generating competent ones. This view has been corroborated by further work (Kim and Bentley, 2001) which has recently shown how succesful the use of clonal selection with a negative selection operator can be for this type of problem.

---

[3] Hofmeyr and Forrest (2000)'s final system employs some other extensions to support the operation of AIS under a real network environment. Among them, affinity maturation and memory cell generation follow the clonal selection concept and these provide a kind of evolution of a detector set distributed on monitored hosts. However, it still uses only the negative selection algorithm to generate an initial detector set. Even though it may conform to human immune systems more closely, this approach could require excessive computation time to generate the initial detector set, if a broader definition of self is used. In addition, the usefulness of initial detectors is not proven before they are distributed to other hosts. This may also cause a waste of other computing resources.

9 CONCLUSIONS

This paper has investigated the role of negative selection in an artificial immune system (AIS) for network intrusion detection. The negative selection stage within our AIS was implemented following the algorithm created by Forrest et al (1994; 1997) and applied to real network data. The experiments showed the infeasibility of this algorithm for this application: the computation time needed to generate a sufficient number of detectors is completely impractical.

This result directs this research to re-define the role of negative selection algorithm within our overall artificial immune system framework. Current work is now investigating the intrusion detection mechanism of the clonal selection stage. A new understanding of the task of the clonal selection stage has now resulted in the development of a more appropriate use for negative selection as an operator within a novel clonal selection algorithm (Kim and Bentley, 2001).

References

D'haeseleer, P, (1997), "A Distributed Approach to Anomaly Detection", *ACM Transactions on Information System Security*. http://www.cs.unm.edu/~patrik/

Dasgupta, D., (1998), "An Overview of Artificial Immune Systems and Their Applications", In Dasgupta, D. (editor). *Artificial Immune Systems and Their Applications*, Berlin: Springer-Verlag, pp.3-21.

Forrest, S. et al, (1994) "Self-Nonself Discrimination in a Computer", *Proceeding of 1994 IEEE Symposium on Research in Security and Privacy,* Los Alamos, CA: IEEE Computer Society Press.

Forrest, S., et al, (1997), "Computer Immunology", *Communications of the ACM*, 40(10), 88-96.

Forrest, S and Hofmeyr, S. (2000) "Immunology as Information Processing", in *Design Principles for Immune Systems and Other Distributed Autonomous Systems*, (Ed) Segal, L.A. & Cohen, I. R. eds., Oxford University Press.

Heberlein, L. T., et al. (1990), "A Network Security Monitor", *Proceeding of 1990 Symposium on Research in Security and Privacy*, Oakland, CA, pp.296-304, May, 1990.

Hofmeyr, S., (1999) *An Immunological Model of Distributed Detection and Its Application to Computer Security*, Phd Thesis, Dept of Computer Science, University of New Mexico.

Hofmeyr, S., and Forrest, S., (2000), "Architecture for an Artificial Immune System", *Evolutionary Computation*, vol.7, No.1, pp.45-68.

Kephart, J. O., et al, (1995), "Biologically Inspired Defenses Against Computer Viruses", *the Proceeding of 14th Intl. Joint Conf. on Artificial Intelligence, Montreal, August*, pp.985-996.

Kim, J. and Bentley, P. (1999a), "The Human Immune System and Network Intrusion Detection", *7th European Conference on Intelligent Techniques and Soft Computing (EUFIT '99), Aachen, Germany*.

Kim, J. and Bentley, P. (1999b), "The Artificial Immune Model for Network Intrusion Detection, *7th European Conference on Intelligent Techniques and Soft Computing* (*EUFIT'99), Aachen, Germany*.

Kim, J. and Bentley, P. (2000), "Negative Selection within an Artificial Immune System for Network Intrusion Detection", *the 14th Annual Fall Symposium of the Korean Information Processing Society, Seoul, Korea*.

Kim, J. and Bentley, P. (2001), The Artificial Immune System for Network Intrusion Detection: An Investigation of Clonal Selection with a Negative Selection Operator. Submitted to CEC2001, the Congress on Evolutionary Computation, Seoul, Korea, May 27-30, 2001.

Lee, W., (2000) *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*, PhD Thesis, Dept of Computer Science, Columbia University.

Mykerjee, B., et al, (1994), "Network Intrusion Detection", *IEEE Network,* Vol.8, No.3, pp.26-41.

Paul, W. E., (1993), "The Immune System: An Introduction", in *Fundamental Immunology* 3rd Ed., W. E. Paul (Ed), Raven Press Ltd.

Porras, P. A., (1992), *STAT: A State Transition Analysis Tool for Intrusion Detection*, MSc Thesis, Department of Computer Science, University of California Santa Babara.

Porras, P. A. and Valdes, A., (1998), "Live Traffic Analysis of TCP/IP Gateways", *Proceeding of ISOC Symposium of Network and Distributed System Security*. http://www.csl.sri.com/emerald/downloads.html

Tizard, I. R., (1995), *Immunology: Introduction*, 4th Ed, Saunders College Publishing.