An Immunogenetic Technique to Detect Anomalies in Network Traffic

Fabio A. González* **and Dipankar Dasgupta**
Intelligent Security Systems Research Lab, Computer Science Division
The University of Memphis
Memphis, TN 38152
*and Universidad Nacional de Colombia
Bogotá, Colombia
{fgonzalz, ddasgupt}@memphis.edu

Abstract

The paper describes an immunogenetic approach which can detect a wide variety of intrusive activities on networked computers. In particular, this technique is inspired by the negative selection mechanism of the immune system that can detect foreign patterns in the complement (non-self) space. The novel pattern detectors (in the complement space) are evolved using a genetic search, which could differentiate varying degrees of abnormality in network traffic. The paper demonstrates the usefulness of such a technique in intrusion/anomaly detection. A number of experiments are performed using intrusion detection data sets (DARPA IDS evaluation program) and tested for validation. Some results are reported along with their analysis and concluding remarks.

1 INTRODUCTION

There are many approaches to anomaly detection, and most of them work on building a model or profile of the system that reflects its normal behavior. A simple approach is to define thresholds (upper and lower) for each monitored parameter of the system, and if a parameter exceeds this range, it is considered an abnormality. The most common approach uses a statistical model (Denning, 1986; Eskin, 2000) to calculate the probability of occurrence of a given value, lesser the probability, higher is the possibility of an anomaly. In general, statistical approaches model individually the different variables that represent the state of the system, which could make it difficult to detect complex multivariable temporal patterns.

Other approaches also build models to predict the future behavior of systems or processes based on the present and past states (Crosbie and Spafford, 1995; Dagupta and Gonzalez, 2001). Accordingly, if the actual state of the system differs considerably from the predicted state, an anomaly alarm is raised. These approaches are more successful in capturing temporal and multiple variable correlations. However, more time is needed for training the model, and in some cases its application can be infeasible because of the size of data sets involved (generally, this is the case with network security).

Therefore, the challenge is to build an anomaly detection system that can capture multi-variable correlations, and is capable of dealing with large amounts of data generated in a computer network environment. Data mining techniques have been applied with some success to this problem (Lane, 200; Lee and Stolfo, 1998). This approach has the advantages of dealing with large data sets and being able to get useful knowledge (generally expressed in terms of rules). For these techniques, it is important that the data have some degree of structure. In several works, the network traffic data (packet level) is processed to get connection information (type, duration, number of bytes transmitted, etc). In some cases, it is necessary to have the information regarding whether a connection is normal or it is an attack.

This paper proposes an approach that does not rely on structured representation of the data and uses only normal data to build a profile of the system. Although, it is applied to perform anomaly detection for network security, it is a general approach that can be applied to different anomaly detection problems.

The technique is inspired by artificial immune systems ideas, and it attempts to extend Forrest's (self/non-self) (Forrest et al., 1994) two-class approach to multiple classes. Specifically, the non-self space will be further classified in multiple sub-classes to determine

the level of abnormality. The core of the technique is a genetic algorithm that evolves rules to cover the abnormal space (non-self). Experiments are performed and the results are compared with the ones produced by a positive characterization technique.

2 BACKGROUND AND PREVIOUS WORKS

The idea of using immunological principles in computer security (Dasgupta, 1999; Hofmeyr and Forrest, 2000; Kephart, 1994) started since 1994. Stephanie Forrest and her group at the University of New Mexico have been working on a research project with a long-term goal to build an artificial immune system for computers. In their approach, the problem of protecting computer systems from harmful viruses was viewed as an instance of the more general problem of distinguishing *self* (legitimate users, uncorrupted data, etc.) from the dangerous *other* (unauthorized users, viruses, and other malicious agents). This method (called the *negative-selection* algorithm (Forrest et al., 1994)) was used to detect changes in the protected data and program files. In another work, they applied the algorithm to monitor UNIX processes where the purpose is to detect harmful intrusions in a computer system. Kephart (Kephart, 1994) suggested another immunologically inspired approach (decoy program) for virus detection. In this approach, known viruses are detected by their computer-code sequences (signatures) and unknown viruses by their unusual behavior within the computer system.

In (Kim and Bentley, 2001), the negative-selection algorithm is evaluated as a mechanism to perform network intrusion detection. The conclusion is that the algorithm presents "severe scaling problems for handling real traffic data", and it is suggested to use it only as a filter for invalid detectors. In the present work, we show that it is possible to overcome these issues and generate effective detectors by changing the representation scheme.

3 ANOMALY DETECTION PROBLEM DEFINITION

The purpose of anomaly detection is to identify which states of a system are normal and which are abnormal. The states of a system can be represented by a set of features. Accordingly,

Definition 1. System states space. A state of the system is represented by a vector of features, $x^i = (x_1^i, ..., x_n^i) \in [0.0, 1.0]^n$. The space of states is represented by the set $S \subseteq [0.0, 1.0]^n$. It includes the feature vectors corresponding to all possible states of the system.

The features can represent current and past values of system variables. The actual values of the variables could be scaled or normalized to fit a defined range $[0.0, 1.0]$.

Definition 2. Normal subspace (crisp characterization). A set of feature vectors, $Self \subseteq S$ represents the normal states of the system. Its complement is called Non_Self and is defined as $Non\_Self = S - Self$. In many cases, we will define the $Self$ (or Non_Self) set using its characteristic function $\chi_{self} : [0.0, 1.0]^n \rightarrow \{0, 1\}$

$$\chi_{self}(\overrightarrow{x}) = \begin{cases} 1 & \text{if } \overrightarrow{x} \in Self \\ 0 & \text{if } \overrightarrow{x} \in Non\_Self \end{cases}$$

The terms *self* and *non-self* are motivated by the natural immune system. In general, there is no sharp distinction between the normal and abnormal states, instead there is a degree of normalcy (or conversely, abnormality). The following definition tries to reflect this:

Definition 3. Normal subspace (non-crisp characterization). The characteristic function of the normal (or abnormal) subspace is extended to take any value within the interval $[0.0, 1.0] : \mu_{self} : [0.0, 1.0]^n \rightarrow [0.0, 1.0]$. In this case, the value represent the degree of normalcy: 1 indicates normal, 0 indicates abnormal, and intermediate values represent elements with some degree of abnormality[1].

The non-crisp characterization allows a more flexible distinction between normalcy and abnormality. However, in a real system it is necessary to decide when to raise an alarm. In this case, the problem becomes again a binary decision problem. It is easy to go from the non-crisp characterization to the crisp one by establishing a threshold:

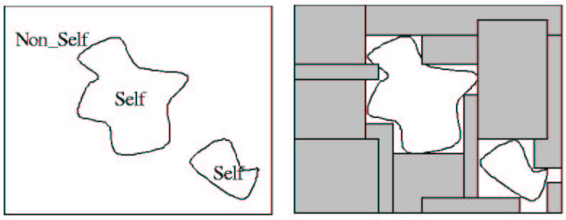
$$\mu_{self,t}(\overrightarrow{x}) = \begin{cases} 1 & \text{if } \mu_{self}(\overrightarrow{x}) > t \\ 0 & \text{if } \mu_{self}(\overrightarrow{x}) \leq t \end{cases}$$

[1] This definition is basically a fuzzy set specification. In fact, the function μ_{self} is a membership function. However, we chose not to refer it directly, because of the different emphasis of this work.

Definition 4. Anomaly detection problem. Given a set of normal samples $Self' \subseteq Self$, build a good estimate of the normal space characteristic function χ_{self} (or μ_{self} in the non-crisp case). This function should be able to decide whether the observe state of the system is anomalous or not.

4 PROPOSED APPROACH

The original negative selection algorithm proposed by Forrest (Forrest et al., 1994) used binary encoding for representing self/non-self strings. In a previous work (Dasgupta and Gonzalez, 2002), a real-valued representation to characterize the descriptor space (self/non-self space) was proposed. A genetic algorithm with sequential niching scheme was used to evolve detectors on the complement (non-self) space (as shown on Figure 1). The present work proposes another niching technique (deterministic crowding (Mahfoud, 1992)) to evolve the non-self-covering detectors, which appears to perform better in covering the complement space.



(a) (b)

Figure 1: (a) Self and non-self division of the descriptor space (b) Approximation of the non-self space by interval rules.

The detectors correspond to hyper-rectangles in a multidimensional space. These detectors are represented by rules with the following structure:

$$\begin{array}{lll} R^1\text{: If} & Cond_1 & \text{then } \mathbf{non\_self} \\ \vdots & \vdots & \vdots \\ R^m\text{: If} & Cond_m & \text{then } \mathbf{non\_self} \end{array}$$

where,

- $Cond_i = x_1 \in [low_1^i, high_1^i]$ and $\ldots$ and $x_n \in [low_n^i, high_n^i]$
- $(x_1, ..., x_n)$ is a feature vector
- $[low_i^j, high_i^j]$ specifies the lower and upper values for the feature x_i in the condition part of the rule R^j.

The condition part of each rule defines a hypercube in the descriptor space ($[0.0, 1.0]^n$). Then, a set of these rules tries to cover the non-self space with hypercubes. For the case $n = 2$, the condition part of a rule represents a rectangle. Figure 1.b illustrates an example of this kind of coverage for $n = 2$.

The non-self characteristic function (crisp version) generated by a set of rules $R = \{R^1, ..., R^m\}$ is defined as follows:

$$\chi_{non\_self,R}(\vec{x}) = \begin{cases} 1 & \text{if } \exists R^j \in R \text{ such that } \vec{x} \in R^j \\ 0 & \text{otherwise} \end{cases}$$

where $\vec{x} \in R^j$ means that $\vec{x}$ satisfies the condition part of the rule R^j.

A rule is considered good if it does not include positive samples and covers a large area. This criteria guides the evolution process performed by the genetic algorithm.

As was discussed previously, a good characterization of the abnormal (non-self) space should be non-crisp. Then, the non-self space is further divided into different levels of deviation. We can think of these levels as concentric regions around the self zones. The farther a level is from the self, the more abnormal it is.

In order to characterize the different levels of abnormality, we considered a variability parameter (called v) to the set of normal descriptors samples, where v represents the level of variability that we allow in the normal (self) space. A higher value of v means more variability (a larger self space); a lower value of v represents less variability (a smaller *self* space). Figure 2 shows two sets of rules that characterize self spaces with a large and small value of v. Figure 2.a shows a covering using a small variability parameter v. Figure 2.b shows a covering using a larger value of v. The variability parameter can be assumed as the radius of a hyper-sphere around the self samples. Figure 2.c shows the levels of deviation defined by the two coverings.

In the non-self space, we use a genetic algorithm with different values of v to generate a set of rules that can provide complete coverage. In general, a set of rules has the following structure:

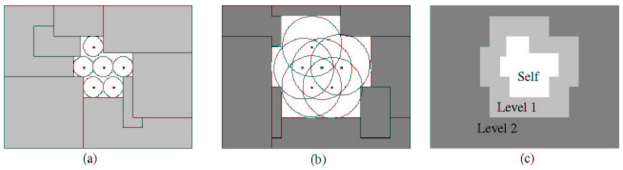


Figure 2: A set of normal samples is represented as points in 2-D space. The circle around each sample point represents the allowable deviation. (a) Rectangular rules cover the non-self (abnormal) space using a small value of v. (b) Rectangular rules cover the non-self space using a large value of v. (c) Level of deviation defined by each v, where level 1 corresponds to non-self cover in (a) and level 2 corresponds to non-self cover in (b)

$$
\begin{array}{rlll}
R^1\text{: If} & Cond_1 & \text{then } \textbf{Level 1} \\
\vdots & \vdots & \vdots \\
R^i\text{: If} & Cond_i & \text{then } \textbf{Level 1} \\
R^{i+1}\text{: If} & Cond_{i+1} & \text{then } \textbf{Level 2} \\
\vdots & \vdots & \vdots \\
R^j\text{: If} & Cond_j & \text{then } \textbf{Level 2} \\
\vdots & \vdots & \vdots
\end{array}
$$

The different levels of deviation are organized in a hierarchical way such that, the level 1 contains the level 2, the level 2 contains the level 3, and so forth. This means that a descriptor can be matched by more than one rule, but the highest level reported will be assigned. This set of rules generates a non-crisp characteristic function for the non-self space:

$$\mu_{non\_self}(\overrightarrow{x}) = \max\{l \mid \exists R^j \in R\ ,\ \overrightarrow{x} \in R^j \text{and } l = level(R^j)\} \cup \{0\},$$

where $level(R^j)$ represent the deviation level reported by the rule R^j.

4.1 GENETIC ALGORITHM FOR DETECTOR GENERATION

The purpose of the genetic algorithm is to evolve 'good' rules to cover the non-self space. In general, one rule is not enough, instead, a set of rules that solve the problem cooperatively is necessary. In our previous work, we used a genetic algorithm combined with a sequential niching technique (Beasley et al., 1993). That approach was useful in evolving good detector rules. The main drawback of that approach is that the genetic algorithm must be run multiple times to generate multiple rules. The approach proposed by this paper, uses a niching technique, deterministic crowding (Mahfoud, 1992), that allows the generation of multiple rules in a single run.

The input to the GA is a set of n-dimensional feature vectors $S = \{x^1, ..., x^m\}$, which represents samples of the normal behavior of the parameter, the number of different levels of deviation (*numLevels*), and the allowed variability for each level $\{v_1, ..., v_{numLevels}\}$. The algorithm is shown as follows:

---

for $i = 1$ **to** *numLevels*
 initialize population with random individuals
 for $j = 1$ to *numGenerations*
  for $k = 1$ to *population_size*/2
   select two individuals with uniform probability
    and without replacement
   apply crossover to generate a child
   mutate the child
   if *dist(child,parent1)* $<$ *dist(child,parent2)*
     and *fitness(child)* $>$ *fitness(parent1)*
    substitute parent1 with child
   elseif *dist(child,parent1)* $\geq$ *dist(child,parent2)*
     and *fitness(child)* $>$ *fitness(parent2)*
    substitute parent2 with child
   endif
  endfor
 endfor
 extract the best individuals from the population
 and add them to the final solution
endfor

Each individual (chromosome) in the genetic algorithm represents the condition part of a rule, since the consequent part is the same for all the rules (the descriptor belongs to non-self). However, the levels of deviation in non-self space are considered by the variability factor (v_i) that is used by the fitness function.

The condition part of the rule is determined by the low and high limits for each dimension. The chromosome that represents these values consists of an array of floats. The crossover operator used is a uniform crossover and the mutation operator is Gaussian mutation.

4.1.1 Fitness Evaluation

Given a rule R with condition part ($x_1 \in [low_1, high_1]$ **and** ... **and** $x_n \in [low_n, high_n]$), we say that a feature vector $x^j = (x_1^j, ..., x_n^j)$ satisfies the rule (represented for $x^j \in R$) if the hyper-sphere with center x^j and radius v_i intercepts the hyper-rectangle defined by the points $(low_i, ..., low_n)$ and $(high_1, ..., high_n)$.

The fitness of a rule is calculated taking into account the following two factors:

- The number of elements in the training set S, that belongs to the subspace represented by the rule:

$$num\_elements(R) = \{x^i \in S \mid x^i \in R\}$$

- The volume of the subspace represented by the rule:

$$volume(R) = \prod_{i=1}^{n} (high_i - low_i)$$

The fitness is defined as:

$$fitness_R = volume(R) - C \cdot num\_elements(R)$$

where, C is the coefficient of sensitivity. It specifies the amount of penalization that a rule suffers if it covers normal samples. The bigger the coefficient, the higher is the penalty value. The fitness can also take negative values.

4.1.2 Individual's Distance Calculation

A good measure of distance between individuals is important for deterministic crowding niching, since it allows the algorithm to replace individuals with closer individuals. This allows the algorithm to preserve the forming niches.

The distance measure used in this work is the following:

$$dist(c, p) = \frac{volume(p) - volume(p \cap c)}{volume(p)},$$

where,

- c : child individual
- p : parent individual

Note that the distance measure is not symmetric. The idea is that we give more importance to the area of the parent that is not covered. The justification is as follows: if the child covers a high proportion of the parent that means that the child is a good generalization of it, but if the child covers only a small portion, then it is not so.

5 EXPERIMENTATION AND RESULTS

5.1 TESTING DATA

We performed experiments with intrusion data obtained from the MIT-Lincoln Lab (MIT-Lincoln-Labs, 1999). These data represents both normal and abnormal information collected in a test network, where some simulated attacks were performed. The purpose of the data is to test the performance of intrusion detection systems. The data sets (corresponding to the year 1999) contain complete weeks with normal data (not mixed with attacks). This allows us to get enough samples to build the self profile and generate detectors.

The test data set is composed of network traffic data (tcpdump, inside and outside network traffic), audit data (bsm) and file systems data. For our experiments, we used only the outside tcpdump network data for a specific computer (e.g., hostname: marx), then we applied the tool *tcpstat* to get traffic statistics. We used the first week's data for training (attack free), and the second week's data for testing, which include some attacks. Some of these were network attacks the others were inside attacks. Only the network attacks are considered for our testing. The attack time line is shown in Figure 3.

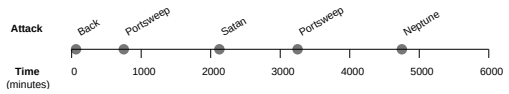


Figure 3: Network attacks on the second weekend

Three parameters were selected to detect some specific type of attacks. These parameters were sampled each minute (using *tcpstat*), and normalized. Table 1 lists six time series S_i and T_i for training and testing, respectively.

The set S of normal descriptors is generated from a time series $R = \{r_1, r_2, ..., r_n\}$ in an overlapping sliding window fashion:

$$S = \{(r_1, ..., r_w), (r_2, ..., r_{w+1}), ..., (r_{n-w+1}, ..., r_n)\},$$

where w is the window size. In general, from a time series with n points, a set of $n-w+1$ of w-dimensional

Table 1: Data sets and parameters used

Name	Description	Week	Type
S1	# of bytes / sec	1	Training
S2	# of packets / sec	1	Training
S3	# of ICMP packets / sec	1	Training
T1	# of bytes / sec	2	Testing
T2	# of packets / sec	2	Testing
T3	# of ICMP packets / sec	2	Testing

descriptors can be generated. In some cases, we used more than one time series to generate the feature vectors. In those cases, the descriptors are put side-by-side in order to produce the final feature vector.

5.2 EXPERIMENTAL SETTINGS

We used as the training set the time series $S1$, $S2$ and $S3$, and as the testing set the time series $T1$, $T2$ and $T3$, with a window size of 3. This means that the size of the feature vectors was 9.

The parameters for the genetic algorithm were: population size 200, number of generations 2000, mutation rate 0.1, and coefficient of sensitivity: 1.0 (high sensitivity).

The genetic algorithm was run with radius equal to 0.05, 0.1, 0.15 and 0.2 respectively. Then the elements in the testing set are classified using rules generated for each level (radius). This process is repeated 10 times and the results reported correspond to the average of these runs.

In order to evaluate the ability of the proposed approach to produce a good estimation of the level of deviation, we implemented a simple (but inefficient) anomaly detection mechanism. It uses the actual distance of an element to the nearest neighbor in the *Self* set as an estimation of the degree of abnormality. Formally, the characteristic function of the *non-self* set is defined as:

$$\begin{aligned} \mu_{non\_self}(\vec{x}) &= D(\vec{x}, Self) \\ &= \min\{d(\vec{x}, \vec{s}) : \vec{s} \in Self\} \end{aligned}$$

Here, $d(x, s)$ is a Euclidean distance metric (or any Minkowski metric[2]). $D(\vec{x}, Self)$ is the nearest neighbor distance, that is, the distance from x to the closest point in *Self*.Then, the closer an element $\vec{x}$ is to the self set, the closer the value of $\mu_{non\_self}(\vec{x})$ is to 0.

In the section results and discussion, we refer to this technique as *positive characterization* (PC) approach, to differentiate it from the proposed approach that we call *negative characterization* (NC) approach.

[2]In our experiments, we also used the D_∞ metric defined by: $D_\infty(\vec{x}, \vec{y}) = \max(|x_1 - y_1|, ..., |x_n - y_n|)$

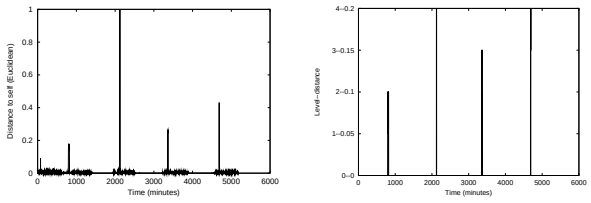
5.3 RESULTS AND DISCUSSION

Table 2: Number of generated rules for each deviation level

Level	Radius	Avg. Num. Rules	
		Seq. Niching	Det. Crowding
1	0.05	19.5	7.75
2	0.1	20.7	8.25
3	0.15	26	10
4	0.2	28	10

Table 2 shows the number of rules generated by the genetic algorithm for the previous technique (NC with sequential niching) and the new one (NC with deterministic crowding). The new technique produces less rules. This suggests the possibility that the new technique is discarding some good rules and therefore ignoring some niches. However, the performance of the two set of rules is the same. The conclusion is that the new technique is able to find a set of rules that is more compact but without compromising on performance. This can be explained by the fact that sequential niching is more sensitive to the definition of the distance between individuals than deterministic crowding.

Another notable point is the efficiency of the new technique. The new technique only needed four runs (one per level) to generate this set of rules. For the previous technique, it is necessary to run the GA as many times as the number of rules we want to generate. This is a clear improvement on computational time.



(a) PC output (b) NC output

Figure 4: Detection function $\mu_{non\_self,t}(\vec{x})$ generated by PC and PC applied to testing set : (a) deviation in testing set reported by PC (b) deviations in the testing set detected by evolved rule set.

Figure 4(a) shows the output of the PC algorithm when applied to the testing data. The five peaks correspond to the 5 attacks. Figure 4(b) shows a typical

attack profile produced by the application of the genetic generated rules to the testing set. Four of five attacks are detected.

As was shown in our previous work (Dasgupta and Gonzalez, 2002), the negative characterization (NC) is clearly more efficient (in time and space) compared to the positive characterization (PC). There seems to be a trade-off between compactness of the rule set representation and accuracy. Validity of these arguments are observed on our results. Figure 5 shows how the true positives rate changes according to the value of the threshold t. The PC technique has better performance than the NC, but only by a small margin. In general, the NC technique shows detection rates similar to the more accurate (but more expensive) PC technique. Table 3 summarizes the best true positive rates (with a maximum false alarm of 1%) accomplished by the two techniques.

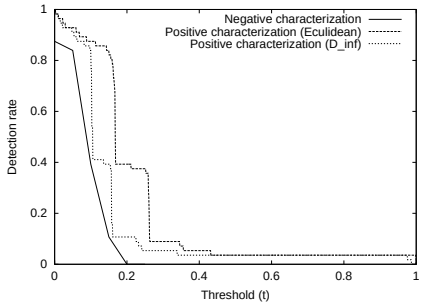


Figure 5: Comparison of the true positives rate of the detection function $\mu_{non\_self,t}(\vec{x})$ generated by positive characterization (PC) and negative characterization (NC) for different values of t.

Table 3: Best true positive rates for the different techniques with a maximum false alarm rate of 1%.

Detection Technique	**True Positive Rate**
Positive Charact. (Euclidean)	96.4%
Positive Charact. (D_∞)	92.8%
Negative Characterization	87.5%

Our previous work (Dasgupta and Gonzalez, 2002) shows that the NC technique produces a good estimate of the level of deviation when this is calculated using D_∞ distance. Table 4 shows the confusion matrix for the NC technique using sequential niching and deterministic crowding. For each element in the testing set, the function $\mu_{non\_self}(\vec{x})$ generated by the NC is applied to determine the level of deviation. This level of deviation is compared with the distance range reported by the PC algorithm (using D_∞ distance). Each row (and column) corresponds to a range or level of deviation. The ranges are specified on square brackets. A perfect output from the NC algorithm will generate only values in the diagonal.

Table 4: Confusion matrix for PC and NC reported deviations. The values of the matrix elements correspond to the number of testing samples in each class. The diagonal values represent correct classification.

PC output level	**NC output level**				
	seq. niching				
	0	1	2	3	4
1: [0.0,0.05]	5132	0	0	0	0
2: [0.05,0.1]	3	7.8	0.2	0	0
3: [0.1,0.15]	0	18.1	3.9	0	0
4: [0.15,0.2]	0	0	6.9	9.5	0.6
5: [0.2,..]	0	0	0	1	9
	det. crowding				
	0	1	2	3	0
1: [0.0,0.05]	5132	0	0	0	0
2: [0.05,0.1]	3	4	4	0	0
3: [0.1,0.15]	0	0	22	0	0
4: [0.15,0.2]	0	0	0	17	0
5: [0.2,..]	0	0	0	0	10

In the two cases, the values are concentrated around the diagonal. The two techniques produced a good estimate of the distance to the self set. However, the NC approach with deterministic crowding appears to be more precise. One possible explanation of this performance difference seems to be the fact that the sequential niching requires derating the fitness function for each evolved rule. This arbitrary modification in the fitness landscape can prevent evolving better rules.

6 CONCLUSIONS

In this paper, we investigated a technique to characterize and identify different intrusive activities by analyzing network traffic. The technique is based on artificial immune systems ideas and uses a genetic algorithm to generate good anomaly detectors rules. We used a real world data set (MIT-Lincoln lab) that has been used by other researchers for testing. The following are some preliminary observations:

- Our approach measures the anomaly of a system as the distance of a descriptor vector to a normal profile represented by descriptors collected during the normal operation. This approach appears to be very useful, since it was able to detect attacks in real test data set.

- The immunogenetic algorithm was able to produce good detectors that give a good estimation of the amount of deviation from the normal. This shows that it is possible to apply the negative selection algorithm to detect anomalies on real network traffic data. The real representation of the detectors was very useful in this work.

- The proposed algorithm is efficient; it was able to detect four of the five attacks detected by the positive characterization (with a detection rate of 87.5% and a maximum false alarms rate of 1%), while only using a fraction of the space (when compared to positive characterization).

- The use of deterministic crowding as niching technique improved the results obtained using sequential niching. While keeping the performance in terms of a high detection rate, the new algorithm generated a smaller set of rules that estimated in a more precise way the amount of deviation. The new technique is also more efficient in terms of computational power since it is able to evolve multiple rules for each individual run of the GA.

As part of our ongoing research we are exploring different covering strategies of the non-self space (for instance, using hyper-spheres), developing new algorithms to generate non-self covering rules and experimenting with other intrusion detection data sets.

Acknowledgments

This work was funded by the Defense Advanced Research Projects Agency (contract no. F30602-00-2-0514) and National Science Foundation (grant no. IIS-0104251).

References

Beasley, D., Bull, D. R., and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125.

Crosbie, M. and Spafford, E. (1995). Applying genetic programming to intrusion detection. In Siegel, E. V. and Koza, J. R., editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 1–8, MIT, Cambridge, MA, USA. AAAI.

Dagupta, D. and Gonzalez, F. (2001). *Information Assuarance in Computer Networks*, chapter An intelligent decision support system for intrusion detection and response, pages 1–14. Lecture Notes in Computer Science. Springer-Verlag.

Dasgupta, D. (1999). *Artificial Immune Systems and Their Applications*. Springer-Verlag, New York.

Dasgupta, D. and Gonzalez, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *To appear in IEEE Transactions on Evolutionary Computation*, 6(2).

Denning, D. (1986). An intrusion-detection model. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 118–31.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *Proc. 17th International Conf. on Machine Learning*, pages 255–262. Morgan Kaufmann, San Francisco, CA.

Forrest, S., Perelson, A., Allen, L., and Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *Proc. IEEE Symp. on Research in Security and Privacy*.

Hofmeyr, S. and Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473.

Kephart, J. (1994). A biologically inspired immune system for computers. In *Proceedings of Artificial Life*, pages 130–139, Cambridge, MA.

Kim, J. and Bentley, P. J. (2001). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1330–1337, San Francisco, California, USA. Morgan Kaufmann.

Lane, T. (200). *Machine Learning Techniques For The Computer Security*. PhD thesis, Purdue University.

Lee, W. and Stolfo, S. (1998). Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX.

Mahfoud, S. W. (1992). Crowding and preselection revisited. In Männer, R. and Manderick, B., editors, *Parallel problem solving from nature 2*, pages 27–36, Amsterdam. North-Holland.

MIT-Lincoln-Labs (1999). Darpa intrusion detection evaluation. http://www.ll.mit.edu/IST/ideval/index.html.