Model Generation for an Intrusion Detection System Using Genetic Algorithms

Adhitya Chittur
November 27, 2001
Ossining High School
Ossining, NY

Chittur, Adhitya

Summary

As malicious intrusions (commonly termed “hacks”) into computer systems have become a growing problem, the need for accurately detecting these intrusions has risen. This paper presents a novel approach to detecting these intrusions by using a complex artificial intelligence method known as a genetic algorithm applied to an Intrusion Detection System. For this experiment, a genetic algorithm was written to learn how to detect malicious intrusions and separate them from normal use. The algorithm was then tested in a real-world simulation to gauge its effectiveness under unpredictable conditions.

Chittur, Adhitya

Abstract

This experiment analyzed the effectiveness of a genetic algorithm applied to the detection of computer intrusions and malicious computer behavior. The use of genetic algorithms to detect malicious computer behavior is a novel approach to the computer network intrusion detection problem presented in designing an Intrusion Detection System. A genetic algorithm is a method of artificial intelligence problem-solving based on the theory of Darwinian evolution applied to mathematical models. The genetic algorithm designed for this experiment promoted a high detection rate of malicious behavior and a low false positive rate of normal behavior classified as malicious. The genetic algorithm was given "training data" from which an empirical model of malicious computer behavior was generated. This model was then tested over previously unseen data to gauge its real-world performance. The results presented show that the genetic algorithm was successfully able to generate an accurate empirical behavioral model from training data and then able to successfully apply this empirical knowledge to data never seen before. The final model produced had an overall accuracy level of 97.8%, which showed both a high detection rate and an extremely low false positive rate. From these results, it was concluded that genetic algorithms are a viable method for empirical model generation for computer intrusion detection. Genetic algorithms are now a possible alternative for the detection of malicious intrusions.

Introduction

As reliance upon the use of digitally transmitted data over computer networks such as the Internet has increased, so has the need for protecting these networks from malicious users (commonly called "hackers" or "crackers"). Many methods for detecting malicious intruders (e.g.,

firewalls, password protected systems) currently exist. However, these traditional methods are becoming increasingly vulnerable and inefficient due to their inherent problems. As a result, new methods for intrusion detection that are not hampered by vulnerability and inefficiency must be developed (Eskin et al, 2001). This research sought to design such a detection method through the use of a genetic algorithm.

Traditional systems in place for intrusion detection primarily use a method known as "fingerprinting" to identify malicious users. Fingerprinting requires the compilation of the unique traits of every type of attack on a computer system. Each generated fingerprint is first added to the attack database of a detection system and then compared to all subsequent user connections for classification as either a malicious or normal connection (Lane, 1998). This trait compilation is typically accomplished through human analysis by the creators of the system. The resulting fingerprint updates must then be manually installed on each individual system in use (Eskin, 2001; Stolfo, 2000).

There are several inherent problems with this method: a system must first be compromised by an attack for a fingerprint to be generated; a separate fingerprint is required for each different type of attack; and as the number of fingerprints grows, more computer resources must be allocated to detection, degrading overall system performance. In addition, to gain protection from new attacks, there is a significant waiting period from the time a new attack is first reported to the time that a fingerprint is generated. During this waiting period, a system is left vulnerable to the new attack and may be compromised. Moreover, in extreme scenarios, a fingerprint-based system may be unable to allocate all required resources to detect attacks because of the number of fingerprints, resulting in undetected attacks (Lee et al, 2001).

As an alternate solution for protecting computers from malicious users, a model-based Intrusion Detection System (IDS) may be used. Instead of using a fingerprinting method of user classification, an IDS compares learned user characteristics from an empirical behavioral model to all users of a system. User behavior is generally defined as the set of objective characteristics of a connection between a client (e.g., a user's computer) and a server. Using a generalized behavioral model is theoretically more accurate, efficient, and easier to maintain than a fingerprinting system. This method of detection eliminates the need for an attack to be previously known to be detected because malicious behavior is different from normal behavior by nature (Sinclair et al, 1999). Also, a model based system uses a constant amount of computer resources per user, drastically reducing the possibility of depleting available resources. Furthermore, while actual attack types by malicious users may vary widely, a model-based IDS does not require the constant updates typical of fingerprint-based systems because the characteristics of any attack against a system will not significantly change throughout the lifetime of the system because attacks are inherently different from normal behavior (Eskin et al, 2001; Lee et al, 2001; Sinclair et al, 1999).

In previous research, the options for model generation have been to base it on normal users or to base it on malicious users (Eskin et al, 2001). Models based on normal users, known as Anomaly Detection models, use an empirical behavioral model of a normal user and classifies any computer activity that does not fit this model as malicious. Models based on malicious users are known as Misuse Detection models. These models look for a pattern of malicious behavior, and behavior that fits this model is classified as malicious (Eskin et al, 2001). In this research, neither model was explicitly specified, allowing the genetic algorithm to generate the best model.

An Intrusion Detection System must first be able to detect malicious user connections, for

which it must have a generalized model of user behavior for comparison to users of a system. The most efficient method for generating a user model is to apply a data analysis algorithm to given "training data," which is representative of real world data (Stolfo et al, 2000), and then generate an empirical model of either type of user based on this training data. Previous research into empirical model generation has used data analysis algorithms such as generalized data mining techniques (Lee et al,1998, 2001), sparse Markov transducers (Eskin et al, 2001), and genetic algorithms (Cedex, 1993; Crosbie & Spafford, 1995). Moreover, previous research using genetic algorithms as a method for intrusion detection has either been theoretical (Cedex, 1993) or become obsolete and is no longer applicable to current intrusion detection research (Crosbie & Spafford, 1995). The experiment presented in this paper seeks to test the viability of genetic algorithms as a method for generating empirical user behavioral models.

A genetic algorithm is a method of data analysis that works analogously to Darwinian evolution (Koza, 1992). Within a computer simulation, a population of many individuals is created, each individual representing a possible mathematical model. Each individual has one or more chromosomes that function as basic instructions to the individual in a cause (e.g., input data) and effect (e.g., user classification) manner. An individual is measured by the aggregate performance of its chromosomes. An initial population is created by complete randomization of the chromosomes, and individuals of subsequent generations go through mutations, which are also randomized (Moriarty et al, 1999). As in Darwinism, a population that goes through many generations eliminates poor performing individuals and allows better performing individuals to replicate and mutate themselves during each generation. This genetic algorithm was designed so that each individual represented a possible behavioral model.

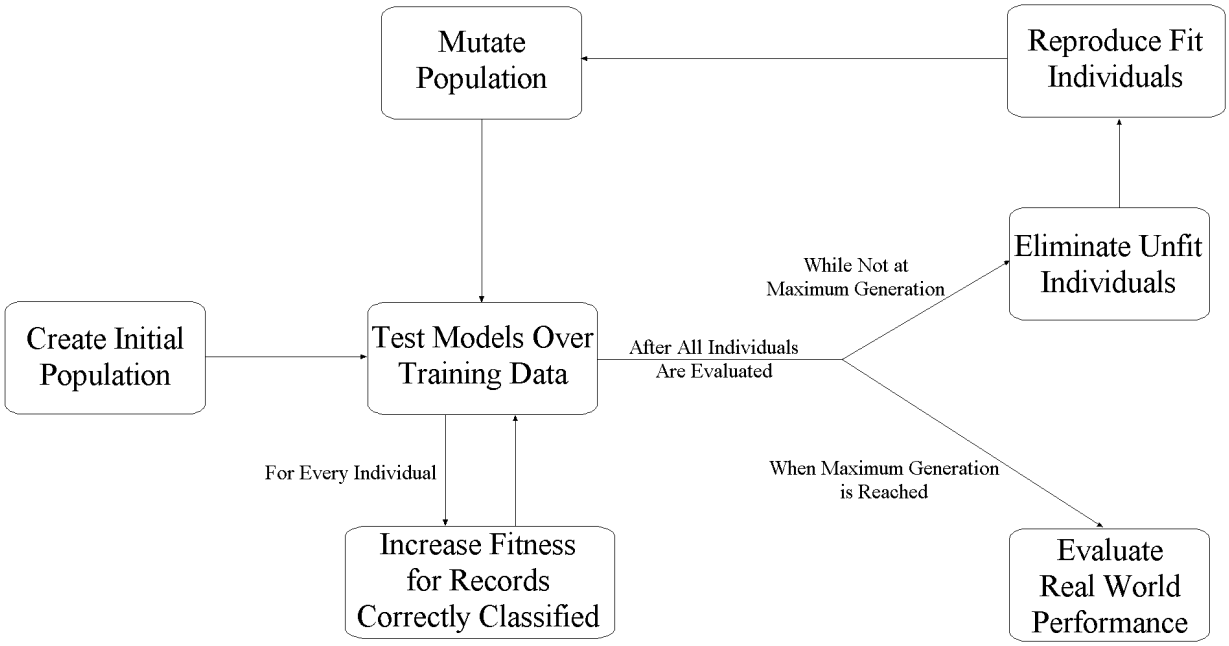

Figure 1: Genetic algorithm theory. This is an application to the Intrusion Detection problem.

The performance of an individual is measured by a fitness function. A fitness function rates the performance of an individual in its environment by comparing the results of the individual's chromosomes with the desired results of the problem as defined by the author of the algorithm (Koza, 1992). The fitness is generally expressed within the algorithm as a floating point (i.e., decimal) number with a predefined range of values, from best performing to worst performing. As in Darwinian evolution, low-performing individuals are eliminated from the population and high-performing individuals are cloned and mutated, replacing those that were eliminated. Analogous to biological mutations, some of the randomly mutated individuals theoretically improve and return superior results until an individual returning a perfect fitness score, known as an ideal individual, is found. If such an individual is not found, the genetic algorithm stops when a predefined maximum number of generations is reached.

The goal of this research is to test whether genetic algorithms are a viable option for model generation in an artificial intelligence-based Intrusion Detection System, designed to replace or reinforce fingerprinting systems. The genetic algorithm written for this experiment isolates forty-one different characteristics of user connections (Stolfo, 2000) to classify users. For comparison, former research has analyzed only five or fewer characteristics (Crosbie, 1995). This genetic algorithm also applies the use of a novel randomized weighting system, based on randomized coefficients in input data, to determine if a user is malicious or normal.

Methodology: The Dataset

In this study, a genetic algorithm was designed to be run over the 1999 Knowledge Discovery in Database (KDD) Cup data, supplied by the Defense Advanced Research Projects Agency (DARPA) and the Massachusetts Institute of Technology's Lincoln Labs (Lippmann, 2000). This data set was selected both for its modernity and similarity to data an Intrusion Detection System would see in a real-world scenario. The data set was generated via a simulated U.S. Air Force local-area network set up at Lincoln Labs, which was run and operated similarly to a standard Air Force network, excepting for planned and recorded attacks.

Originally, the data consisted of nine weeks of raw Transmission Control Protocol (TCP) dump data from the network. From this raw data, which was roughly four gigabytes in size, connection records were established based upon sequences of TCP packets (Stolfo et al, 2000). Forty-one unique attributes were compiled from each raw TCP packet sequence. These included symbolic attributes, such as "protocol type," with values "TCP," "ICMP," and "UDP," as well as continuous attributes, such as error rates over the closed interval [0,1] and bytes transferred over the

half-open interval [0,∞). Each resulting connection record was then labeled as either normal or malicious. In the complete data set, there were approximately five million separate connection records, totaling over seven hundred twenty megabytes.

The genetic algorithm was run over a ten percent subset of the data, called the training data, and then tested over the entire data set to test real-world performance. In the real world, an empirical behavior model would rarely see any data which directly corresponds to training data. Instead, the model must be able to extrapolate conclusions learned from the given training data and return an appropriate decision. Generating a model from the ten percent subset was meant to test whether it was able perform this extrapolation over the entire data set.

Methodology: The Genetic Algorithm

In this study, the fitness of an individual was dependent upon how many attacks were correctly detected and how many normal use connections were classified as attacks. Correct detections were expressed as a positive ratio of total attacks while false positives were expressed as a negative ratio of total normal connections. The fitness function developed for this experiment, *F*, of specific individual d_i was:

$$F(d_i) = \frac{a}{A} - \frac{b}{B}$$

Equation 1: Fitness formula.

where a is the number of correctly detected attacks, A the number of total attacks, b the number of

false positives, and *B* the total number of normal connections. The range of fitness values for this function was over the closed interval [-1,1] with -1 being the poorest possible fitness and 1 being the ideal. A high correct detection rate and a low false positive rate yielded a high score on the fitness function for an individual. Low detection rates or high false positive rates yielded low scores on the fitness function.

The model generated by this genetic algorithm was based on a new method of data analysis for the intrusion detection problem. Each node in the model's decision tree was designed to hold a randomized coefficient for the data, so that this coefficient multiplied by the data would yield a weight for the certainty of whether a certain record was an attack or not. The coefficients were based on an Ephemeral Random Constants (ERC) (Koza, 1992), random numbers generated by the genetic algorithm specific to mathematical modeling. These numbers' slight change in value was the basis for mutation in this genetic algorithm. For symbolic connection attributes (e.g., connection type), different weights were established for each symbol based on an ERC. For continuous connection attributes (e.g., bytes sent), ERC coefficients were randomly established for the data. In continuous attributes that contained data of magnitudes apart, such as bytes sent, separate ERC coefficients were established for each magnitude of data.

The certainty formula developed for this experiment, C_i, of whether record *c* was classified as an attack by model *i* was:

$$C_i(c) = \sum_{j=1}^{n} \left(\Re_{i,j} \times c_j \right)$$

Equation 2: Certainty formula.

where $\Re_{i,j}$ is the Ephemeral Random Constant-based coefficient for attribute c_j and n is the number of attributes. An arbitrary threshold value was established, and any certainty values which exceeded this threshold value were classified as malicious attacks.

The genetic algorithm was run for one hundred generations with one hundred individuals. Forty-one different types of nodes were established, one for each of the forty-one connection record attributes. The genetic algorithm package ECJ 7 was used for this research (Luke, 2001). It provided the necessary population breeding, randomizing, and statistics gathering functions, from which this genetic algorithm was written. The genetic algorithm was written in Java, and the Webgain Visual Café 4.1 Expert Edition interface development environment was used to run the experiment. This experiment was run on a Dell computer with an Intel Pentium III 800 megahertz microprocessor and 256 megabytes of random access memory on Microsoft Windows 2000 using Sun Microsystem's Java Development Kit (JDK) version 1.3.1.

Information collected on each generation consisted of the mean fitness of all of the individuals within the generation, the fitness of the best performing individual, the correct detection rate and the false positive rate.

Results and Conclusion

The experiment took approximately forty-eight hours to complete. The algorithm was run twice with the same parameters. Both runs returned impressive results: the best fitness value returned was 0.967817, very close to the ideal fitness value of 1. The breakdown of the fitness value showed that 97.4694% of attacks in the training data (the ten percent subset) were correctly detected and 0.6877% of normal connections were incorrectly classified as attacks. This corresponds to the

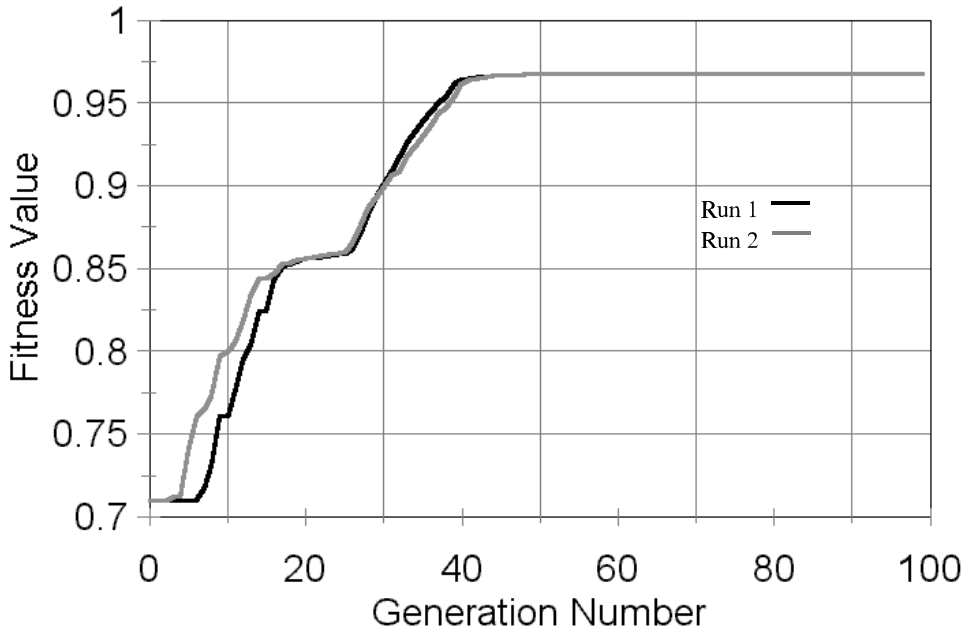

Figure 2: Best individual's fitness value calculated by *F(d)* vs. generation number

desired results of a high detection rate and a low false positive rate.

Both separate runs of the experiment produced the same final individual, yielding the same statistics. When these final individual's models were applied to the full data set, both returned a fitness value of .97454, with 97.7601% of attacks correctly detected and 0.306% of normal connections classified as attacks. This demonstrated that the best model generated using training data successfully was able to detect unknown attacks over previously unseen data.

Figure 2 shows the evolution of the best individual of each generation. In the initial generation, an individual with a fitness value of 0.70997 was created in run 1 and an individual with a fitness value of 0.710028 was created in run 2. The fitness value of the best individual for each generation had an approximate steady increase until generation forty, at which point it is apparent that the best possible individual possible by the current methods had been created. This demonstrated the ability of the genetic algorithm to successfully evolve an individual's model.

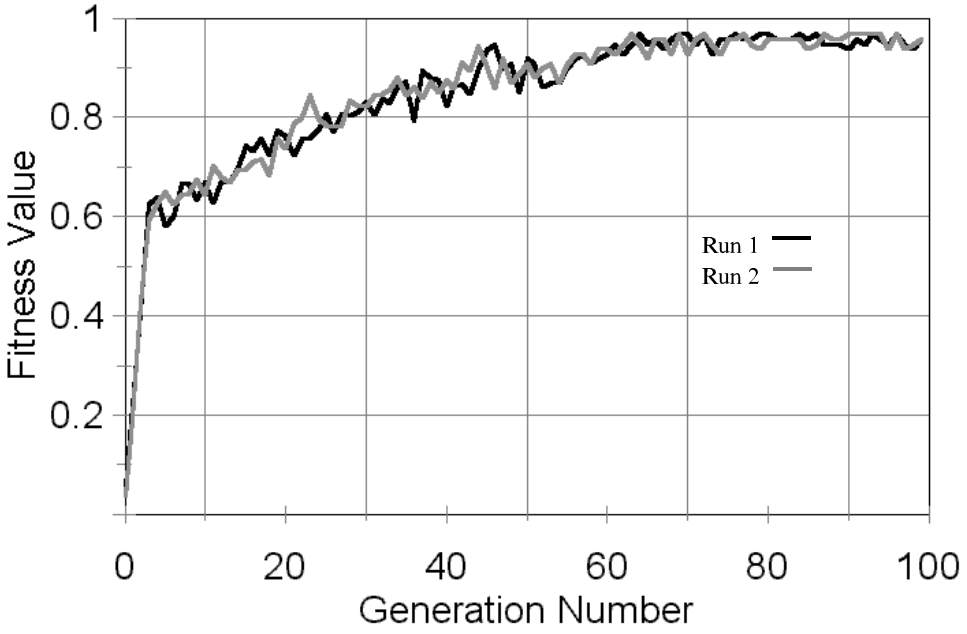

Figure 3: Mean fitness value for the population vs. generation number

By correlating this information with Figure 3, which demonstrates the mean fitness value of the population, it is apparent that the best individual created in the first run was initially unique, but its high fitness value caused it to replaced much of the population's unfit individuals with mutated copies of itself. This can be concluded from the mean fitness of the initial populations being 0.053734 and 0.039588 for run 1 and run 2, respectively, while the fitness values for the best individuals were 0.70997 and 0.710028, respectively. It is apparent that by the fourth generation of the genetic algorithm for each run, mutated copies of the superior individual(s) from the previous generations flourished in the population. Because of these numerous randomly mutated copies, more fit individuals were created, as evidenced by the best individual's fitness value increase in Figure 2. From the steady rate of the population's mean fitness value in both runs after generation sixty-five, it is apparent that the best individual possible with the current parameters made up nearly the entire population by generation sixty-five. It did not make up the whole population because the

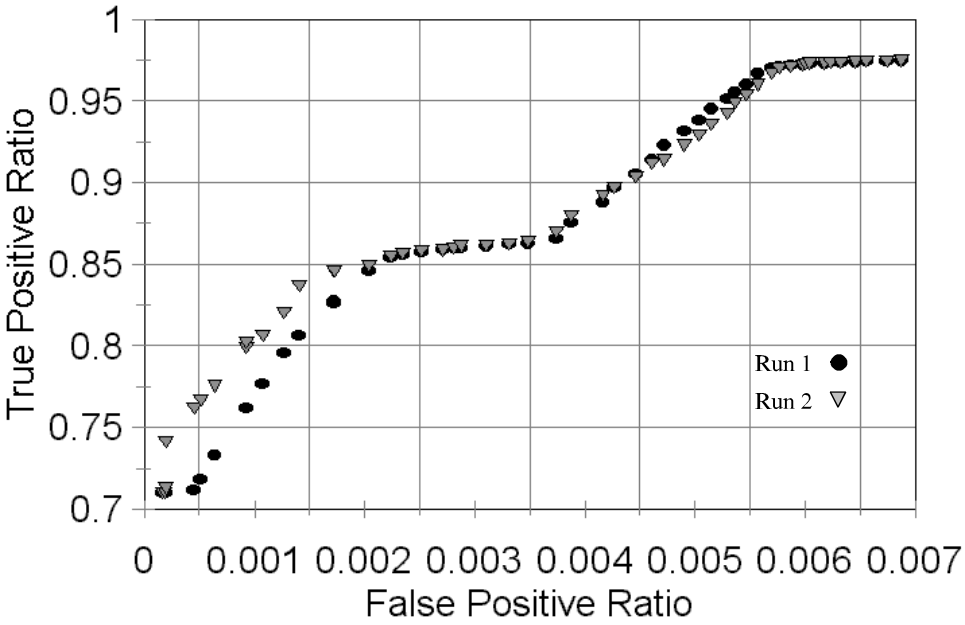

Figure 4: Scatter plot of the best individual of each generation's correct detection rate vs. its false positive rate

genetic algorithm was designed to randomly introduce new individuals for each generation.

Figure 4 demonstrates the trade-off of a higher correct attack detection rate, computed by the ratio of the number of correctly detected attacks to the number of total attacks, to a higher false positive rate, computed by the ratio of the number of normal connections classified as attacks to the number of total normal connections. As expected, there is a positive correlation between these two variables; as more attacks were correctly detected, more normal connections were incorrectly classified as attacks. However, the largest false positive ratio is .006877, meaning that only 0.6877% of normal connections were classified as attacks, a very low false positive rate. The best individual generated by the genetic algorithm for both runs was able to detect 386,703 of the 396,743 attacks in the training data and classified only 669 of the 97,276 normal connections as attacks. When the individual's model was applied to the entire data set, it was able to detect

3,825,703 of the 3,925,650 attacks and classified only 2,977 of the 972,779 normal connections as attacks. These ratios are approximately equal, demonstrating that the empirical model generated by the genetic algorithm from learning over training data was successfully able to correctly classify users in previously unseen data.

From the results obtained, it is evident that the genetic algorithm designed for this experiment was successfully able to generate a model with the desired characteristics of a high correct detection rate and a low false positive rate from learning over training data. Likewise, the genetic algorithm was able to perform the mutation and evolution strategies according to the fitness function. The genetic algorithm then successfully applied what it had learned to a real-world test case.

Discussion

The success of the genetic algorithm shows that this method of model generation for an Intrusion Detection System is a viable alternative. All hypothesis were proven: the genetic algorithm successfully evolved an individual's model through randomized mutation and the model generated over training data was successfully able to apply its empirical knowledge to data not seen before. This supports the hypothesis that the characteristics of malicious computer connections are inherently dissimilar to normal connections.

The quality of the results presented warrants future research in the area of genetic algorithms applied to the Intrusion Detection problem. In this experiment, one hundred individuals were evolved through one hundred generations, allowing for a maximum possibility of ten thousand unique individuals. This may have been a limiting factor, and given more individuals and/or more

generations, it is possible that an even better model could be generated from the given data set. Also, the fitness function *F(d)* presented in Equation 1 can be changed to provide weighting for desired results. The negative influence of false positives on an individual's fitness can be increased or reduced by introducing coefficients of the two ratios into the equation, thus weighting the desired characteristics of a high detection rate and low false positive rate. As another modification to the fitness function, the amount of computer resources used may be used in the fitness function. Additionally, the certainty formula *C(c)* presented in Equation 2 could be exchanged for a different method of certainty of whether a given connection is an attack.

The next step of this research will be the implementation of such a model generated by a genetic algorithm to real-time intrusion detection. By doing this, the efficiency of a model generated could be gauged in the real-world and compared to traditional methods of intrusion detection. Also, the efficiency of a model generated by a genetic algorithm could be compared to models generated using other techniques.

The results presented in this paper show that genetic algorithms are a promising method for the detection of malicious intrusions into computer systems. The model generated may be installed on an existing Intrusion Detection System for further analysis of its performance. The model generation presented in this research may be used to supplement or possibly replace current intrusion prevention methods.

Acknowledgments

I would first like to thank Dr. Eleazar Eskin for his mentorship and guidance on my

experiment. I would also like to thank Mr. Angelo Piccirillo for his personal and administrative support of my research project. I would like to extend a special thanks to Ms. Susan Fahey for reviewing my research paper.

References

1. Cedex CS. Genetic Algorithms, an Alternative Tool for Security Audit Trails Analysis. *Lodovic Me, SUPELEC (France), 1993.*
2. Crosbie M, Spafford E. Applying Genetic Programming to Intrusion Detection. *Proceedings of the AAAI 1995 Fall Symposium, 1995.*
3. Eskin E, Miller M, Zhong ZD, Yi G, Lee W, Stolfo S. Adaptive Model Generation for Intrusion Detection Systems. *Workshop on Intrusion Detection and Prevention, 7th ACM Conference on Computer Security, 2001.*
4. Eskin E, Lee W, Stolfo SJ. Modeling System Calls for Intrusion Detection with Dynamic Window Sizes. *Proceedings of DISCEX II, 2001.*
5. Koza JR. Genetic Programming: On the Programming of Computers by Means of Natural Selection. *The MIT Press, 1992.* [1]
6. Lane T. Filtering Techniques for Rapid User Classification. *Proceedings of the AAAI98 /ICML-98 Joint Workshop on AI Approaches to Time-series Analysis, 1998.*
7. Lee W, Stolfo S, Chan PK, Eskin E, Fan W, Miller M, Hershkop S, Zhang J. Real Time Data Mining-based Intrusion Detection. *Proceedings of DISCEX II, 2001.*
8. Lee W, Stolfo S, Mok K. Mining Audit Data to Build Intrusion Detection Models. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD '98), 1998.*
9. Lippmann R, Haines JW, Fried DJ, Korba J, Das K. The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks, 34, 579-595, 2000.*
10. Luke, S. ECJ 7. *http://www.cs.umd.edu/projects/plus/ec/ecj/ , 2001.* [2]

11. Moriarty DE, Schultz AC, Grefenstette JJ. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research 11:199-229, 1999.*

12. Sinclair C, Pierce L, Matzner SP. An Application of Machine Learning to Network Intrusion Detection. *15th Annual Computer Security Applications Conference, 1999.*

13. Stolfo SJ, Fan W, Lee W, Prodromidis A, Chan P. Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX '00), 2000.*

[1]Book

[2]Genetic Algorithm package on which code written for this experiment was based