International Research Journal of Computer Science (IRJCS) ISSN: Applied
Volume 1 Issue 1 (September 2014) www.irjcs.com

Implementation of Intrusion Detection System using GA

Ms.Lata Jadhav[1]
Dept. of Comp. Science & Engg.
Aurangabad, India.

Prof.C.M.Gaikwad[2]
Dept.of Information Technology Govt. College of Engg.
Aurangabad, India.

ABSTRACT: ***In recent years, computer systems are facing increased number of security threats because of rapid expansion of computer networks. Different soft computing techniques have been proposed in recent years to develop the Intrusion Detection System. This paper presents an effective genetic algorithm (GA) approach for intrusion detection and the software implementation. The Genetic algorithm is used to derive the set of classification rules from audit data and support confidence framework is utilizes as fitness function to judge the quality of each rule. Then the generated rules are used to detect or classify network intrusions. The proposed method is easy to implement while providing the flexibility to either generally detect network intrusions. Experimental results show the more effective detection rates based on benchmark DARPA data sets on intrusions.***

Keywords: Genetic algorithm, Intrusion Detection, support confidence framework.

1. INTRODUCTION:

When an intruder attempts to break into an information system or performs an action not legally allowed, we refer to this activity as an intrusion. Intruders can be classified into two groups, external and internal. The external intruders refer to those who do not have authorized access to the system and who attack by using various penetration techniques. The internal intruders refer to those with access permission who wish to perform unauthorized activities [2]. Intrusion prevention technique like firewall, filtering router policies fails to stop much type of attacks. Intrusion detection systems are becoming an important part of your computer and network security. An intrusion detection system is used to detect several types of malicious behaviours that can compromise the security and trust of a computer system. This includes network attacks against defenceless services, data driven attacks, host based attacks like unauthorized logins and access to sensitive files, and malware (viruses, trojan horses and worms). There are basically two categories of intrusion detection systems (IDSs): misuse detection and anomaly detection Misuse detection systems detect intruders with known patterns, where as anomaly detection systems identify deviations from normal network behaviours and alert for potential unknown attacks. The IDSs can also be classified into two categories depending on where they look for intrusions. A host-based IDS monitors activities associated with a particular host, and a network-based IDS listens to network traffic.

In this paper, we present a GA-based approach to network misuse detection. GA is chosen because of some of its nice properties, e.g., robust to noise, no gradient information is required to find a global optimal or sub-optimal solution etc. Using GAs for network intrusion detection has proven to be a cost-effective approach. The software is experimented using DARPA data sets on intrusions, which has become the de facto standard for testing intrusion detection systems[3].

2. GENETIC ALGORITHM:

Genetic Algorithms is an optimization technique using an evolutionary process[4][5]. A solution of a problem is represented as a data structure known as chromosome. An evaluation function is used to calculate the goodness of each chromosome according to the desired solution; this function is known as “Fitness Function”. GA process begins with series of initial solutions is initially generated (random population) and through a combination of algorithms similar to an evolutionary process (often a combination of elitism, crossover, and mutation) the process works towards evolving solutions having better “goodness” as evaluated by the fitness function.

In every generation the fitness of these chromosomes is checked. To determine the fitness of the chromosomes fitness function is used and then fittest chromosomes are selected. The chromosomes which have poor fitness value are discarded. The selected fit chromosomes undergo crossover, mutation to form a new population. This new population is used for the next generation. Normally, the algorithm terminates when either a set number of generations or a satisfactory fitness level has been achieved. Genetic algorithm is composed of three operators. They are reproduction or selection, crossover or recombination and mutation.

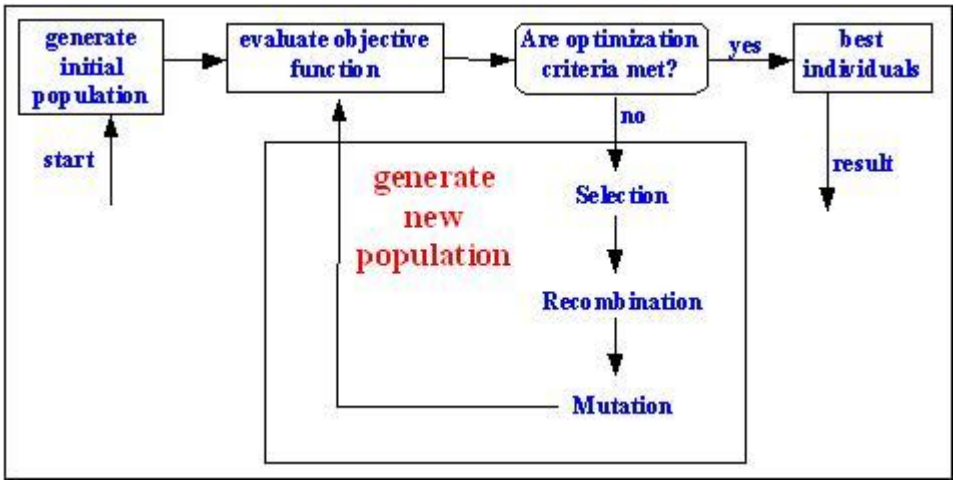


Figure 1.Structure of simple genetic algorithm

The basic concepts of Genetic Algorithms are simple, yet the process of choosing the gene representation, a good fitness function, and even application of the recombination [Whitley] can be the key to successful use of Genetic Algorithms.

2.1 DARPA DATA SET:

A key dependency of the work done by Gong and Li and as will be shown with netGA is the usage of DARPA data sets for training data. Creating this training data is not a trivial task and is considered beyond the scope of this project. The MIT Lincoln laboratory provides an excellent description of the process followed for creating the data. This DARPA training data is actually a result of test network traffic data, a Sun Microsystems Solaris and the use of Sun's Basic Security Module[Sun]. The data sets used in both papers were created in 1998[3]. Today's attacks have changed with regard to rule based systems, but the training data still works well for developing Genetic Algorithms.

3. GA WITH IDS:

3.1 Data Representation

The way that Genetic Algorithms are used with netGA is that rules are randomly created to match attacks encoded as a integer array with the seven elements shown in Figure 2. The first six attributes of the chromosome match the gene characteristics of an attack. The seventh attribute describes the attack type that the first six rules identify when they match. This representation uses the same approach as used by Gong[2].

Feature Name	Format	Number of genes
Duration	h:m:s	3
Protocol	Int	1
Source_port	Int	1
Destination_port	Int	1
Source_IP	a.b.c.d	4
Destination_IP	a.b.c.d	4
Attack_name	Int	1

Figure 2: Chromosome Representation for Rule

In order to evaluate a rule represented by a chromosome, the DARPA audit data is parsed and loaded into a list of audit connections. The attributes loaded from the DARPA audit data directly match the attributes used in the chromosome representation. The gene representation follows the simple rule if A then B, where if the first six attributes are logically and-ed together are true(A), then the rule matches the attack (B**).** Following is the sample example[4] that classifies a network connection as the denial-of-service attack Neptune.

if (duration = “0:0:1” and protocol = “finger” and
source_port = 18989 and destination_port = 79 and
source_ip = “99.19.99.19” and destination_ip =

"192.168.254.10") then (attack_name = "Neptune")

Above rule specifies that if a network packet is originated from IP address 99.19.99.19 and port number 18989 and send to IP address 192.168.254.10 at port number 79 using finger protocol for duration of connection 1 second then most likely it is Neptune attack which eventually make destination host out of service.

3.2 Fitness Function:

Every chromosome is selected after applying fitness function to them. To determine the fitness of a rule, the support confidence framework [6] is used. If a rule is represented as if A then B [4] then the fitness of the rule is as follows:

support = |A and B| / N
confidence = |A and B| / |A|
fitness = w1 * support + w2 * confidence

Here, *N* is the total number of network connections in the audit data, |*A*| stands for the number of network connections matching the condition *A*, and |*A and B*| is the number of network connections that matches the rule *if A then B*. The weights *w1* and *w2* are used to control the balance between the two terms and have the default values of *w1*=0.2 and *w2*=0.8.

3.3 Crossover and Mutation

Crossover is one of the important steps in GA. There are three types of crossover techniques. They are one point, two point and uniform cross over technique. In this paper we used two point crossovers. Crossover involves splitting two chromosomes and then combining first part of a chromosome with the second part of the other chromosome.

Each gene in each chromosome is checked for possible mutation by generating a random number between zero and one and if this number is less than or equal to the given mutation probability then the gene value is changed. Mutations create diversity to search in domain regions that may otherwise be excluded.

3.4 Detection Algorithm Overview

Listing 1 shows the major steps of the employed detection algorithm as well as the training process. It first generates the initial population, sets the defaults parameters, and loads the network audit data. Then the initial population is evolved for a number of generations.

Algorithm: Rule set generation using genetic algorithm.
Input: Network audit data, number of generations, and population size.
Output : A set of classification rules.
1. Initialize the population
2. *W1* = 0.2, *W2* = 0.8, *T* = 0.5
3. *N* = total number of records in the training set
4. For each chromosome in the population
5. *A* = 0, *AB* = 0
6. For each record in the training set
7. If the record matches the chromosome
8. *AB* = *AB* + 1
9. End if
10. If the record matches only the "condition" part
11. *A* = *A* + 1
12. End if
13. End for
14. Fitness = *W1* * *AB* / *N* + *W2* * *AB* / *A*
15. If Fitness > *T*
16. Select the chromosome into new population
17. End if
18. End for
19. For each chromosome in the new population
20. Apply crossover operator to the chromosome
21. Apply mutation operator to the chromosome
22. End for
23. If number of generations is not reached, then goto line 4

Listing 1. Major steps of the detection algorithm.

In each of the qualities rules are firstly calculated, then a number of best-fit rules are selected, and finally the GA operators are applied to the selected rules. The training process starts by randomly generating an initial population of rules (line 1). The weights and fitness threshold values are initialized in line 2. Line 3. Calculates the total number of records in the audit data. Lines 4-18 calculate the fitness of each rule and select the best-fit rules into new population. Lines 19-22 apply the crossover and mutation operators to each rule in the new population. Finally, line 23 checks and decides whether to terminate the training process or to enter the next generation to continue the evolution process.

4. IMPLEMENTATION AND RESULTS

The genetic algorithm for rule generation is implemented using Java language (JDK6) in NetBeans7.4.The front end development environment used is NetBeans7.4. Two subsets were developed from DARPA 1998 data.

Table 1 gives the distributions of record types in both training and testing data set. The first row gives the number of normal network records. The second row gives the distributions of Smurf attack whereas the third row gives the distribution of Neptune attack.

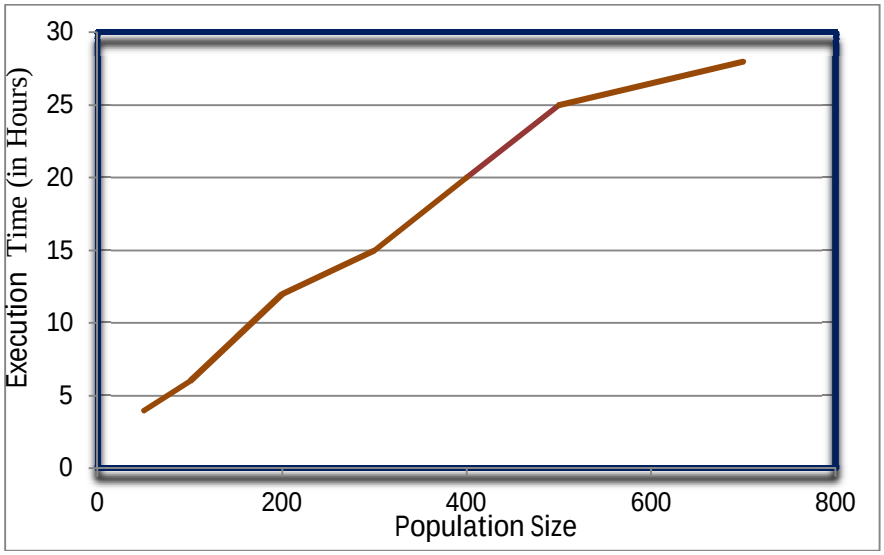
The implementation is done in two phases. In the first phase the classification rules are generated using genetic algorithm. Support confidence function as fitness function. The GA parameters used were w1 = 0.2, w2 = 0.8, 200 generations, population of 2000 rules, mutation rate of 0.001. In the second (testing / detection) phase, for each test data, an initial population is made using the data and occurring mutation in different features. This population is compared with each chromosomes prepared in training phase. Portion of population, which are more loosely related with all training data than others, are removed. Crossover and mutation occurs in rest of the population which becomes the population of new generation. The process runs until the last generation finished. The group of the chromosome which is closest relative of only surviving chromosome of test data is returned as the predicted type.

Table 1.Results

Record Type	Training	Testing	Match %
Normal	73	64	87%
Smurf	799	790	98%
Neptune	96	96	100%

4.1 Execution time

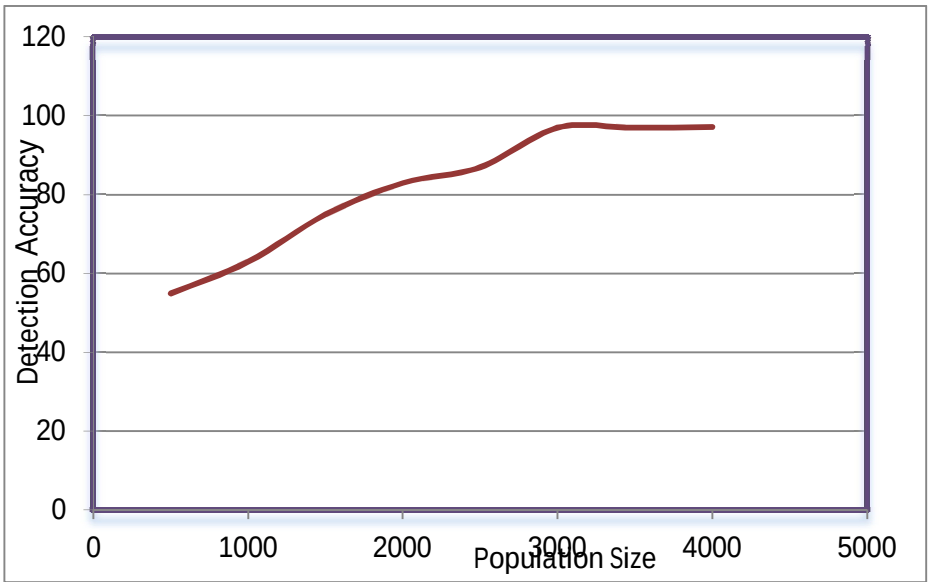
The time taken by a genetic algorithm to reach to a required solution is an important aspect. This execution time increases linearly as the population size increases for the equal number of generations. Graph 4.1 shows the population size and corresponding execution time taken by the GA. The maximum number of generations is set to 200.



Graph4.1 Effects of GA population size on Execution Time

4.2 Population Size

Graph 4.2 shows the percentage detection for different number of generations of GA. As the number of generations is increased, the detection rate is improved at the cost of increased time required for the generation of rules. The best results are obtained after 200 generations.



Graph 4.2 Effects of Population on Detection Accuracy

Conclusion:

IDS is implemented using GA in two steps. In the first step, GA is used to generate classification rules where as in the second step these rules are used for intrusion detection. This reduces the search space and yields more accurate results while using smaller population and lesser number of generations compared to Gong et al.'s approach. This has reduced the time required for the generation of fittest rules. The given system is run for different generations. As the number of generations is increased, more accurate intrusion detection rates are obtained.

References

[1] W. Li, "A Genetic Algorithm Approach to Network Intrusion Detection", SANS Institute, USA, 2004.

[2] Li, Wei. 2002. "The integration of security sensors into the Intelligent Intrusion Detection System (IIDS) in a cluster environment." Master's Project Report. Department of Computer Science, Mississippi State University.

[3] MIT Lincoln Laboratory, DARPA datasets, MIT, USA, in November r 2004). http://www.ll.mit.edu/IST/ideval/data/data_index.html

[4] H. Pohlheim, "Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms", http://www.geatbx.com/docu/index.html (accessed in January 2005).

[5] M. Crosbie and E. Spafford, "Applying Genetic Programming to Intrusion Detection", Proceedings of the AAAI Fall Symposium, 1995

[6] W. Lu and I. Traore, "Detecting New Forms of Network Intrusion Using Genetic Programming", *Computational Intelligence,* vol. 20, pp. 3, Blackwell Publishing, Malden, pp. 475-494, 2004.

