Detecting New Forms of Network Intrusion Using Genetic Programming

Wei Lu
Department of Electrical and Computer Engineering
University of Victoria
PO Box 3055 STN CSC
Victoria, BC, Canada
wlu@ece.uvic.ca

Issa Traore
Department of Electrical and Computer Engineering
University of Victoria
PO Box 3055 STN CSC
Victoria, BC, Canada
itraore@ece.uvic.ca

Abstract - How to find and detect novel or unknown network attacks is one of the most important objectives in current intrusion detection systems. In this paper, a rule evolution approach based on Genetic Programming (GP) for detecting novel attacks on network is presented and four genetic operators namely reproduction, mutation, crossover and dropping condition operators are used to evolve new rules. New rules are used to detect novel or known network attacks. A training and testing dataset proposed by DARPA is used to evolve and evaluate these new rules. The proof of concept implementation shows that the rule generated by GP has a low false positive rate (FPR), a low false negative rate (FNR) and a high rate of detecting unknown attacks. Moreover, the rule base composed of new rules has high detection rate (DR) with low false alarm rate (FAR).

1. Introduction

Intrusion detection has been extensively studied since the seminal report written by Anderson [1]. Traditionally, intrusion detection techniques are divided into misuse detection and anomaly detection. Misuse detection techniques mainly focus on developing models of known attacks, which can be described by specific patterns or sequences of events and data. Anomaly detection techniques model system or users' normal behaviors, and any deviation from the normal behaviors is considered as an intrusion. Misuse detection techniques have low false detection rates (FDR), but their major weakness is that novel or unknown attacks will go unnoticed until corresponding signatures are added to the database of the Intrusion Detection System (IDS). Anomaly detection techniques have the potential to detect novel attacks, but quite often they tend to have high false detection rates because it is very difficult to discriminate between abnormal and intrusive behavior.

In this paper we propose a rule evolution approach based on Genetic Programming (GP) [2][3] for detecting known or novel attacks on network. GP extends the fundamental idea of Genetic Algorithm (GA), and evolves more complex data structures. To do so, it uses parse trees to represent initial populations instead of chromosomes. Moreover, the GP technique can be used to evolve a population of individuals whereas GA searches the best solution in all possible solutions. Initial rules are selected based on background knowledge from known attacks and can be represented as parse trees. GP will evolve these initial rules to generate new rules. New rules are used to detect novel or known attacks. To evolve and evaluate these new rules, we use the training and testing dataset proposed by DARPA [4], which includes almost all known network based attacks namely *land, synflood, ping of death (pod), smurf, teardrop, back, neptune, ipsweep, portsweep* and *UDPstorm* attacks. The proof of concept implementation shows that the GP based approach can detect smurf and UDPstorm attacks, which are absent from the training dataset. The average false negative rate for each rule is 5.04% and the average false positive rate is 5.23%. The average rate of detecting unknown attacks for each rule is 57.14%. Moreover, we plot a receiving operator characteristic (ROC) curve of false alarm rate and detection rate when we apply the testing dataset to evaluate our rule base. The ROC curve shows that the detection rate will be close to 100% when the false alarm rate falls in the range between 1.4% and 1.8%.

The rest of the paper is organized as follows. Section 2 presents an overview of related works. Section 3 provides background information on genetic programming. Section 4 discusses how to use GP to generate new rules for detecting known or novel attacks on network. Section 5 presents the evaluation of the new rules and discusses the experimental results. Finally, section 6 makes some concluding remarks.

2. Related Works

In [5], Frank described and categorized several Artificial Intelligence (AI) techniques that can be used for intrusion detection; use of AI techniques for intrusion detection is categorized according to two dimensions: behavior classification and data reduction. Behavior classification assumes that intrusion can be decided by a given set of known behaviors and data reduction is typically used to analyze the large amount of audit-log data produced so as to reduce the amount of data handled by human experts. However, explicit knowledge of known behaviors is difficult to establish. Any mistake occurring in the process of defining patterns of known behaviors will increase false alarm rate and decrease the effectiveness of intrusion detection.

Some early application of neural networks for user behavior modeling was proposed by Fox et al. [6]. Ghosh et al. later extended their idea by using back propagation algorithm for anomaly detection [7]. They established that randomly generating anomalous input data increases the performance of anomaly detection. The biggest limitation of this method is the difficulty of choosing the input parameters. Any mistake in input data selection will increase the false alarm rate. Further, how to initialize the weights of the neural network is still an open question.

Lodovic et al. initially proposed another application of AI to intrusion detection by using Genetic Algorithm (GA) for misuse detection [8]. They defined a n-dimensional hypothesis vector H, where Hi = 1 if attack i was taking place according to the hypothesis, otherwise Hi = 0. Thus, the aim of intrusion detection was reduced to the problem of finding the H vector that maximizes the product W*H, subject to the constraint $(AE.H)_i <= O_i$. W refers to the n-dimensional weight vector; AE refers to an attacks-events matrix; O refers to the observed n-dimensional audit trail vector. They showed that GA applied to misuse detection has a low false alarm rate. However, the biggest limitation of their approach is that it cannot identify attacks precisely.

A. Chittur extended their idea by using GA for anomaly detection [9]. Random numbers were generated using GA. A threshold value was established and any certainty value exceeding this threshold value was classified as a malicious attack. The experimental result showed that GA successfully generated an accurate empirical behavior model from training data. The biggest limitation of this approach was the difficulty of establishing the threshold value, which might lead to a high false alarm rate when used to detect novel or unknown attacks.

More works on using GA for intrusion detection are described in [10], [11] and [12]. J. Gomez et al. [10] proposed a linear representation scheme for evolving fuzzy rules using the concept of complete binary tree structure. GA is used to generate genetic operators for producing useful and minimal structure modification to the fuzzy expression tree represented by chromosomes. The biggest drawback of the proposed approach was that the training was time consuming. S.M. Bridge et al. [11] employed GA to tune the fuzzy membership functions and select an appropriate set of features in their prototype IIDS (Intelligent IDS). B. Balajinath [12] used GA to learn individual user behavior. Active user behavior is predicted by GA based on past observed user behavior, and used to detect intrusion. Common limitation on both approaches is that the training process is time consuming and they can only be used to detect anomalous behaviors at the host level.

In [13], Crosbie and Spafford employed GP and agent technology to detect anomalous behaviors in a system. The autonomous agents are used to detect intrusions using log data of network connections. Each autonomous agent is used to monitor a particular network parameter and autonomous agents that are predicting correctly are given higher weight value in deciding whether a session is intrusive or not. There are a number of advantages to having many small agents instead of a single large one. But how to handle the communication among these agents is still an issue. Moreover, the training process may be time consuming if the proper primitive for each agent is not chosen.

3. Overview of Genetic Programming

3.1 GP Algorithm

Genetic Programming [2] is an extension of Genetic Algorithm (GA). It is a general search method that uses analogies from natural selection and evolution. The main difference between them is the solution encoding method. GA encodes potential solutions for a specific problem as a simple population of fixed-length binary strings named chromosomes and then apply reproduction and recombination operators to these chromosomes to create new chromosomes. In contrast to GA, GP encodes multi potential solutions for specific problems as a population of programs or functions. The programs can be represented as parse trees. Usually, parse trees are composed of internal nodes and leaf nodes. Internal nodes are called primitive functions and leaf nodes are called terminals. The terminals can be viewed as the inputs to the specific problem. They might include the independent variables and the set of constants. The primitive functions are combined with the terminals or simpler function calls to form more complex function calls. For instance, GP can be used to evolve new rules from general ones. The rules are represented as *if condition 1 and condition 2...and condition N then consequence*. In this case, the primitive function corresponds to *AND* operator and the terminals are the conditions (e.g. *condition1, condition2,..., condition N*).

GP randomly generates an initial population of solutions. Then, the initial population is manipulated using various genetic operators to produce new populations. These operators include reproduction, crossover, mutation, dropping condition, etc.. The whole process of evolving from one population to the next population is called a generation. A high-level description of GP algorithm can be divided into a number of sequential steps, as follows: .

1. Create a random population of programs or rules using the symbolic expressions provided as the initial population.

2. Evaluate each program or rule by assigning a fitness value according to a pre-defined fitness function that can measure the capability of the rule or program to solve the problem.
3. Use reproduction operator to copy existing programs into the new generation .
4. Generate the new population with crossover or mutation or other operators from a randomly chosen set of parents.
5. Repeat steps 2 onwards for the new population until a pre-defined termination criterion has been satisfied or a fixed number of generations have been completed.
6. The solution to the problem is the genetic program with the best fitness within all the generations.

3.2 Genetic Operators

In GP, crossover operation is achieved firstly by reproduction of two parent trees. Then two crossover points are randomly selected in the two offspring trees. Exchanging sub-trees, which are selected according to the crossover point in the parent trees, generates the final offspring trees. The obtained offspring trees are usually different from their parents in size and shape. Figure 1 describes a crossover operation between function $x^2+x+x-2x$ and function $2x^2$, they produce two offspring functions $2x^2+x$ and x^2-x.

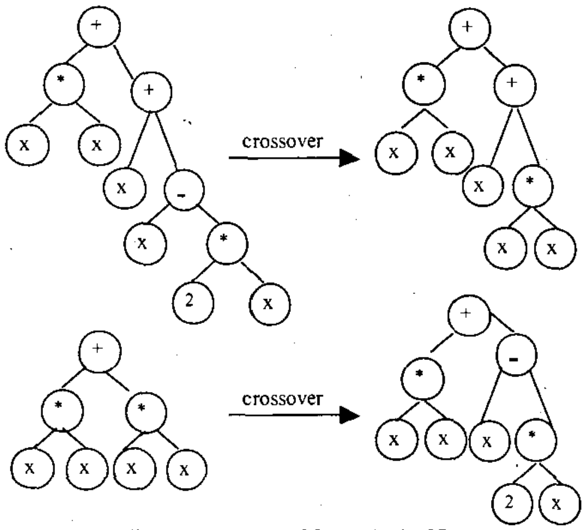

Figure 1. Example of Crossover in GP

Mutation operation is also considered in GP. A single parental tree is firstly reproduced. Then a mutation point is randomly selected from the reproduction, which can be either a leaf node or a sub-tree. Finally, the leaf node or the sub-tree is replaced by a new leaf node or sub-tree generated randomly. Figure 2 describes a mutation operation on function $2x^2$, the produced mutation offspring function is x^2+2x.

A new operator named dropping condition is proposed to evolve new rules in this paper. It randomly selects one condition in the rule and then turns it into any. That is this particular condition is no longer considered in the rule. For example, the rule

if condition1 and condition2 and condition3 then consequence

can be changed to

if condition1 and condition2 and any then consequence

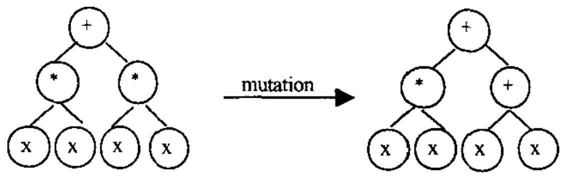

Figure 2. Example of mutation in GP

3.3 Fitness Function

Fitness functions ensure that the evolution is toward optimization by calculating the fitness value for each individual in the population. The fitness value evaluates the performance of each individual in the population. We use a fitness function defined in [3] that is based on the support-confidence framework. Support is a ratio of the number of records covered by the rules to the total number of records. Confidence factor (cf) represents the accuracy of rules, which is the confidence of the consequent to be true under the conditions. It is the ratio of the number of records matching both the consequent and the conditions to the number of records matching only the conditions. If a rule is represented as if A then B and the size of the training dataset is N, then

$$cf = |A \text{ and } B| / |A|\ ;\ support = |A \text{ and } B| / N$$

$|A|$ stands for the number of records that only satisfy condition A. $|B|$ stands for the number of records that only satisfy consequent B. $|A \text{ and } B|$ stands for the number of records that satisfy both condition A and consequent B.

A rule with a high confidence factor does not necessarily behave significantly different from the average. Thus, normalized confidence factor is defined to consider the average probability of consequent denoted *prob*.

$$normalized\_cf = cf * \log (cf / prob),\ \ prob = |B| / N$$

To avoid wasting time to evolve those rules with a low support value, a strategy is defined: if support is below a user-defined minimum threshold (min_support), the confidence factor of the rule should not be considered.

Thus, the fitness function is defined as follows:

$$raw\_fitness = \begin{cases} support & \text{if } support < min\_support \\ w1 * support + w2 * normalized\_cf & \text{otherwise} \end{cases}$$

where the weights $w1$ and $w2$ are user-defined, and used to control the balance between the confidence and the support during the searches.

Token competition is used to increase the diversity of solutions [14]. The idea is as follows: In the natural environment, once an individual finds a good place to live, then it will try to protect this environment and prevent newcomers from using it unless the newcomers are stronger than this individual. Other weaker individuals are hence forced to search their own place. In this way, the diversity of the population is increased. A token is allocated to each record in the training dataset. If a rule matches a record, its token will be seized by the rule. The priority of receiving the token is determined by the strength of the rules. Thus, a rule with high raw_fitness score can acquire as many tokens as possible. The modified fitness is defined as follows:

$$modified\_fitness = raw\_fitness * count / ideal$$

Where count is the number of tokens that the rule has actually seized, ideal is the total number of tokens that it can seize, which is equal to the number of records that the rule matches.

4. Generating New Rules Using GP

The use of GP to detect unknown attacks is based on the belief that new rules will have better performance than initial ones based on known attacks. Better performance means the new rules obtained after evolving the initial ones using GP will not only cover known attacks but also possibly detect the novel ones.

Individual solution in a population is represented as a derivation tree that we describe using a string data structure. For example, a tree can be represented as *"AabAcdAceI"*. *A* means 'and' operator; *a,b,c,d* and *e* correspond to the conditions in the rules. *I* is the consequence, which means intrusion. The redundant conditions in the rule will be deleted after the evolution and thus *"AabAcdAceI"* can be interpreted as *if a and b and c and d and e then intrusion*. The attribute values of *a,b,c,d,e* are selected from known attacks.

New rules are generated in two phases. In the first step, temporary new rules are composed of new rules generated by four operators including mutation, reproduction, crossover and dropping condition and additional rules directly generated from previous populations. Thus, the number of temporary new rules is doubled. In the second phase, one half of the temporary new rules with the highest fitness scores after token competition are retained and passed to the next generation.

To assess the feasibility and efficiency of GP for intrusion detection, we have selected an initial population of 40 rules that cover a series of network based attacks. Table 1 shows 10 instances of the initial rules, the rest of the rules are given in the appendix.

We calculate the fitness value for each rule based on the training dataset. Currently, the most widely used training and testing dataset for anomaly detection is provided by DARPA Intrusion Detection Evaluation Program [4], which consists of the raw TCP dump data of nine weeks activity in a local area network simulating a typical U.S. Air Force LAN. The training dataset is labeled as either normal or intrusive. The test dataset is similar with the training dataset. The only difference is that the test dataset includes some unknown attacks not occurring in the training dataset.

In our case, 10,000 network connection records provided by DARPA training dataset are used to train the rules, each connection lasting 2s. Eleven parameters defined in DARPA dataset are used to describe the attacks in the training dataset. Table 2 describes these parameters and their meaning.

Rules	**Meaning**
Afp	if land=1 and wrong_fragment=0 then intrusion
AAgg	if wrong_fragment>1 then intrusion
Aab	if protocol_type=tcp and count>3 then intrusion
bAbA	if srv_count>3 then intrusion
AhAg	if protocol_type=icmp and wrong_fragment>1 then intrusion
rA	if synflood=1.00 then intrusion
AatA	if protocol_type=tcp and num_compromised>1 then intrusion
Aaav	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
wwww A	if diff_srv_rate>0.33 then intrusion
Aajt	if count<3 and num_compromised>1 then intrusion

Table 1. Initial Rules

Parameters	**Meaning**
protocol_type	type of protocol
land	flag to identify whether connection is from/to the same host/port
wrong_fragment	number of wrong fragments in the connection
synflood	connections that have "SYN" errors
num_compromised	number of compromised conditions
same_srv_rate	percentage of connections to the same services
diff_srv_rate	percentage of connections to the different services
count	number of connections from the same source host to the same destination host
srv_count	number of connections from the same source service to the same destination service
dst_host_count	number of connections from the same destination host to the same source host
dst_host_srv_count	number of connections from the same destination service to the same source service

Table 2. Representation of Parameters

The rules in the initial population are evolved using mutation, crossover and dropping condition operators. The rates of crossover, mutation and dropping_condition operations are respectively 0.6, 0.01 and 0.001 for each rule. Fourty offspring rules are evolved from the previous fourty parent rules. Based on token competition, combining offspring rules with parent rules generates temporary new rules. One half of the temporary new rules with highest fitness scores after token competition are selected as the new rules.

The evolution will not be terminated until we have executed 5000 runs or the fitness value for each rule is bigger than a threshold equal to 0.95. Table 3 describes 10 instances of obtained new rules. To view the rest of new rules, please refer to the appendix.

The initial and new rules are composed of attribute descriptors. Table 4 shows attribute descriptors representations and meanings.

Rules	Meaning
Ag	if wrong_fragment>1 then intrusion
Afpq	if land=1 and wrong_fragment=0 and synflood=0 then intrusion
Aav	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
Ahcq	if protocol_type=icmp and dst_host_srv_count>160 and synflood=0 then intrusion
Aikq	if protocol_type=udp and srv_count>367 and synflood=0 then intrusion
Aat	if protocol_type=tcp and num_compromised>1 then intrusion
At	if num_compromised>1 then intrusion
Ag	if wrong_fragment>1 then intrusion
Ajlpr	if count<412 and dst_host_count<810 and wrong_fragment=0 and synflood>1 then intrusion
Ail	if protocol_type=udp and dst_host_count>203 then intrusion

Table 3. New Rules

Terminal	Meaning	Terminal	Meaning
s	num _compromised=0	i	protocol _type=udp
b	count>R1	f	land=1
c	srv_count>R1	o	land=0
d	dst_host_count >R1	j	count<R2
k	srv_count<R2	q	synflood=0
p	wrong _fragment=0	r	synflood>1
e	dst_host_srv _count>R1	a	protocol _type=tcp
l	dst_host_count <R2	h	protocol _type=icmp
m	dst_host_srv _count<R2	u	same_srv _rate=0.00
t	num _compromised>1	v	same_srv _rate=1.00
g	wrong_fragment >1	w	diff_srv _rate>R3

Table 4. Representation of Terminals

Note:R1,R2,R3 are random values

5. Evaluation of New Rules

Evaluation of intrusion detection approaches for detecting novel attacks is an important and multi-faceted problem. The training dataset we use is one day's connection records provided by DARPA, that is 10000 connection records. Eight kinds of network attacks are included in the training dataset, namely *land, synflood, pod, teardrop, back, neptune, ipsweep and portsweep*. The testing dataset we use is another one-day activity consisting of 10000 connection records. Ten kinds of network attacks are included in the testing dataset, namely *smurf, UDPstorm, land, synflood, pod, teardrop, back, neptune, ipsweep* and *portsweep*. *Smurf* and *UDPstorm* attacks are the novel attacks which are absent from the training dataset. Detection of attacks involved in the test dataset and not occurring in the training dataset assesses the potential ability to detect novel attacks. We use three performance metrics to evaluate the new rules, namely false positive rate (FPR), false negative rate (FNR) and unknown attack detection rate (UADR). A false positive occurs when a rule classifies normal traffic as intrusive. A false negative occurs when a rule characterizes an intrusion as normal. UADR measures the capability of a new rule to detect novel attacks.

For each rule, we calculate its FPR, FNR and UADR independently. We find that every time we use GP to evolve the rules, the number of generated rules is different and thus the FPR, FNR and UADR for each rule is also different. Therefore, to statistically evaluate the efficiency of our GP based approach, FPR, FNR and UADR are defined as the arithmetical average value of all new rules's rates:

$FPR=average(\sum (FPR)_{rules});$

$FNR=average(\sum (FNR)_{rules});$

$UADR=average(\sum (UADR)_{rules});$

For instance, consider a rule base that includes two new rules: rule1 and rule2. The FPR, FNR and UADR of rule1 is 0.001, 0.015 and 0.56 respectively. The FPR, FNR and UADR of rule2 is 0.002, 0.03 and 0.78 respectively. Thus, according to the definition:

average FPR for each rule=(0.001+0.002)/2=0.0015;

average FNR for each rule=(0.015+0.03)/2=0.0225;

average UADR for each rule=(0.56+0.78)/2=0.67.

Since the number of new rules is different in each run, the average FPR, FNR and UADR for each run is also different. We execute 10,000 runs and plot the probability distribution of FPR, FNR and UADR. Figure 3 and Figure 4 illustrate the FPR's probability distribution and the log scale probability distribution. Figure 5 and Figure 6 illustrate the FNR's probability distribution and the log scale probability distribution. Figure 7 and Figure 8 illustrate the UADR's probability distribution and the log scale probability distribution

The standard deviation of FPR for each rule over 10000 runs is 0.0944 and the average value of FPR for each rule over 10000 runs is 0.0523. The confidence interval is 0.0019. In the figure of Log Scale Distribution of FPR, we amplify the probability difference of different FPR scales, and then conclude that the probability of the rules whose FPRs fall in a range between 0 and 0.1 is about 10 times greater than the probability of rules whose FPRs fall in other ranges, such as between 0.1 and 0.2, 02 and 0.3, etc. Table 5 summarizes this information.

The standard deviation of FNR for each rule over 10000 runs is 0.0801 and the average value of FNR for each rule over 10000 runs is 0.0504. The confidence interval is 0.0016. In the figure of Log Scale Distribution of FNR, we amplify the probability difference of different FNR scales, and then conclude that the probability of the rules whose FNRs fall in a range between 0 and 0.1 is about 10 times greater than the probability of rules whose FNRs fall in other ranges, such as between 0.1 and 0.2, 02 and 0.3, etc.. Table 6 summarizes this information.

The standard deviation of UADR for each rule over 10000 runs is 0.397 and the average value of UADR for each rule over 10000 runs is 0.2509. The confidence interval is 0.00794. In the figure of Log Scale Distribution of UADR, we amplify the probability difference of different UADR scales, and then conclude that the probability of the rules whose FPRs fall in a range between 0 and 0.1 is about 10 times greater than the probability of rules whose FPRs falls in other ranges except 0.9 and 1.0. The UADR for each rule in about 20% of the runs are bigger than 0.9 and rates in 70% of the runs fall in a range between 0 and 0.1. Table 7 summarizes this information.

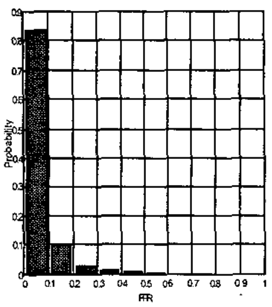

Figure 3. Distribution of FPR

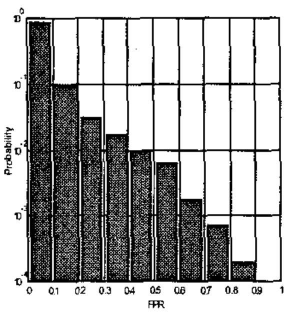

Figure 4. Log Scale Distribution of FPR

FPR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0
Num. Of Run	8200	1000	400	200	99	70	20	8	3	0

Table 5. Scale Distribution Over 10000 runs

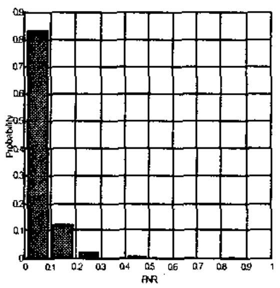

Figure 5. Distribution of FNR

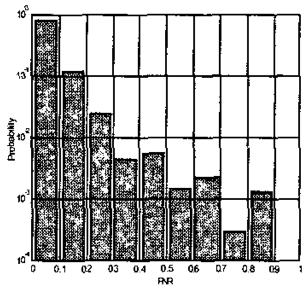

Figure 6. Log Scale Distribution of FNR

FNR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8 -	0.8-0.9	0.9-1.0
Num. Of Run	8300	1100	300	100	100	30	40	5	25	0

Table 6. Scale Distribution Over 10000,runs

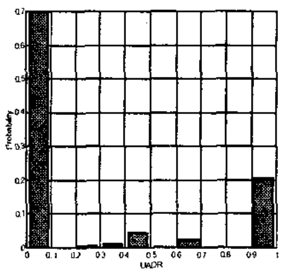

Figure 7. Distribution of UADR

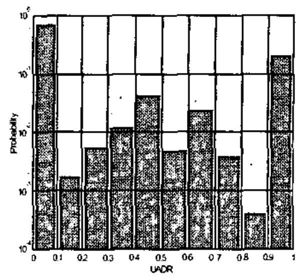

Figure 8. Log Scale Distribution of UADR

UADR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6 - 0.7	0.7-0.8	0.8-0.9	0.9-1.0
Num. Of Run	6835	25	75	105	550	60	250	50	5	2055

Table 7. Scale Distribution Over 10000 runs

In practical evaluation, we usually use the rule base instead of single rule to test the performance of intrusion detection system based on GP. We execute 10000 runs to evaluate the statistical performance of our system since we get different rules every time. We use as evaluation metrics the false alarm rate (FAR) and the detection rate (DR). Generally speaking, we say that an intrusion detection approach is good if it has high detection rate with low false alarm rate. The probability distribution of FAR for the rule base over 10000 runs is illustrated in Figure 9. Figure 10 illustrates the same information for FAR between 0 and 0.1

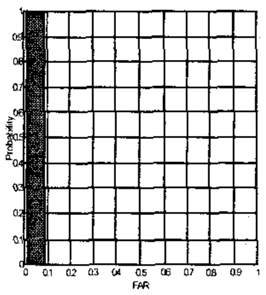

Figure 9. Distribution of FAR

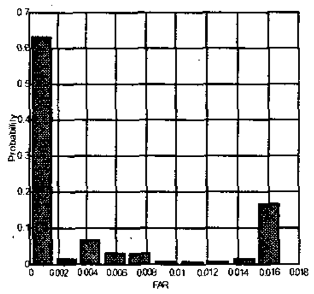

Figure 10. Distribution of FAR between 0 and 0.1

The average value of FAR over 10000 runs is 0.41% and the standard deviation value of FAR over 10000 runs is 0.0063.

The probability distribution of DR for the rule base in 10000 runs is illustrated in Figure 11. Figure 12 illustrate the log scale probability distribution of DR.

The average value of DR over 10000 runs is 0.5714 and the standard deviation value of DR over 10000 runs is

0.4068. Figure 13 is the ROC curve plotting the false alarm rate and the detection rate.

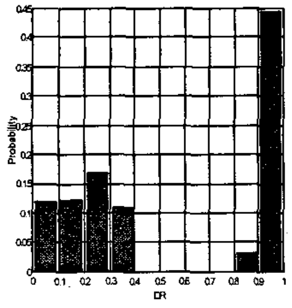
Figure 11. Distribution of DR

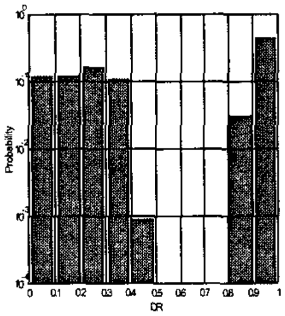
Figure 12. Log Scale of Distribution of DR

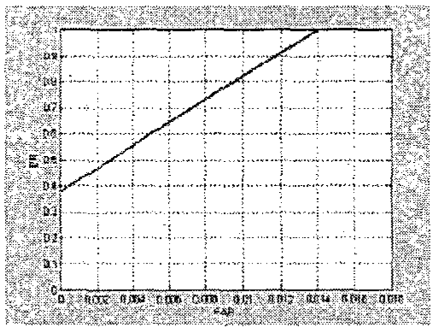
Figure 13. ROC curve of FAR and DR

The DR increases as the FAR does the same. The DR is close to 100% when the FAR is in the range between 1.4% and 1.8%. However, when the false alarm rate is close to 0%, the DR is only about 40%. The reason why the DR falls in a broad range from 40% to 100% is because the number of rules in the rule base is different for each run. When there are more rules in the rule base, we have a high detection rate and thus will possibly have a low false alarm rate. There are some other approaches used to detect intrusion using the DARPA's dataset as testbed. For example, the ROC curve plotted by Eskin et al. [15] is illustrated in Figure 14.

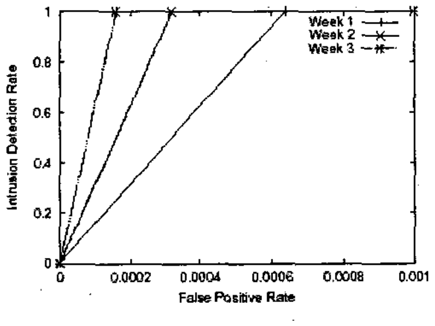

Figure 14. ROC curve of FAR and DR plotted by Eskin et al. [15]

Figure 14 shows that DR increases as the FAR does the same. The DR is close to 100% when the FAR falls in the range between 0.06% and 0.1%. It is obvious that Eskin's result is better than ours considering the ROC curve comparison. However, the curve plotted by Eskin et al. is for only one kind of attack, which is a *ftpd attack*, while our ROC curve is for ten attacks. Our approach can detect ten kinds of attacks when the FAR is smaller than 1.8%. The approach proposed by Eskin et al. can be used to detect only one kind of attack when its FAR is smaller than 0.1%.

6. Conclusions

In this paper, we've presented and evaluated a GP-based approach for detecting known or novel attacks on a network. The proof of concept implementation shows that new rules generated by GP have the potential capability to detect novel forms of attacks. However, the detection result is not good for some runs. The main reason is that the selection of crossover and mutation points in corresponding operations is random. And also deciding the probability of genetic operators selection is experience based. In our implementation, the probability of mutation, crossover is 0.01 and 0.6 individually.

The purpose of the work reported in this paper was mainly to assess the efficiency of GP for known or novel attacks detection. The next step in our work will consist of extending the scope of the rules involved, and using alternative training and testing data sources.

References

1. A J.P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical Report, James P. Anderson Co., Fort Washington, PA, 1980.
2. John R. Koza. Genetic Programming. MIT Press, 1992.
3. Man Leung Wong, Kwong Sak Leung, Data Mining Using Grammar based Genetic Programming and Applications, Kluwer Academic Publishers, 2000.
4. Lippmann, R., et al., The 1999 DARPA Off-Line Intrusion Detection Evaluation. Computer Networks, 34(4) 579-595, 2000.
5. Jeremy Frank. Artificial Intelligence and Intrusion Detection: Current and Future Directions. In Proceedings of the 17th National Computer Security Conference, pages 11-21, Oct. 1994.
6. K.L.Fox, R.R. Henning, J.H. Reed, and P.R. Simonian. A Neural Network Approach towards Intrusion Detection. In Proceedings of 13th National Computer Security Conference, pages 125-134, 1990.
7. A.K.Ghosh, J.Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions against Programs. In Proceedings of the 14th Annual Computer Security Applications Conference, pages 259-267, December 1998.
8. Lodovic Me. Genetic Algorithms, an Alternative Tool for Security Audit Trails Analysis, Technical report, Supelec, France, 1992.

9. Adhytia Chittur, Model Generation for an Intrusion Detection System using Genetic Algorithm. High School Honors Thesis, 2002.
10. J. Gó mez, D. Dasgupta, O. Nasaroui, and F. Gonzá lez. Complete Expression Trees for Evolving Fuzzy Classifiers Systems with Genetic Algorithms and Application to Network Intrusion Detection. *In Proceedings of NAFIPS-FLINT joint Conference*, pages 469-474, New Orleans, LA, June 2002.
11. Bridges, Susan M., and Rayford M. Vaughn. Fuzzy Data Mining and Genetic Algorithms Applied to Intrusion Detection. Proceedings of the Twenty-third National Information Systems Security Conference, Baltimore, MD, October 2000.
12. B. Balajinath and S. Raghavan. Intrusion Detection Through Learning Behavior Model. pp. 1202-1212, Computer Communications, Vol. 24, No. 12, July 2001.
13. Mark Crosbie and Gene Spafford. Applying Genetic Programming to Intrusion Detection. Technical Report FS-95-01, AAAI Fall Symposium Series, AAAI Press, 1995.
14. Leung, K., S., Y., So, L., and Yam, K. F., Rule Learning in Expert Systems Using Genetic Algorithms: 1, Concepts. In proceeding of the 2nd International Conference on Fuzzy Logic and Neural Networks, pp. 201-204.
15. Eleazar Eskin, Matthew Miller, Zhi-Da Zhong, George Yi, Wei-Ang Lee, Sal Stolfo. Adaptive Model Generation for Intrusion Detection Systems. *Workshop on Intrusion Detection and Prevention, 7th ACM Conference on Computer Security*, Athens, GR, November, 2000.

Appendix

Rules	Meaning
AfpAqA	if land=1 and wrong_fragment=0 and synflood=0.00 then intrusion
AaggqA	if wrong_fragment>1 and synflood=0.00 then intrusion
Aigq	if protocol_type=udp and wrong_fragment>1 and synflood=0.00 then intrusion
AabrA	if protocol_type=tcp and count>3 and synflood>1 then intrusion
Aarc	if srv_count>3 and synflood>1 then intrusion
Ahcr	if protocol_type=icmp and srv_count>3 and synflood>1 then intrusion
AaaAfj	if protocol_type=tcp and land=1 and count<3 then intrusion
AaAoc	if protocol_type=tcp and srv_count>3 and land=0 then intrusion
gaAjA	if protocol_type=tcp and count<3 and wrong_fragment>1 then intrusion
capAA	if protocol_type=tcp and srv_count>3 and wrong_fragment=0 then intrusion
Abc	if count>3 and srv_count>3 then intrusion
AatA	if protocol_type=tcp and num_compromised>1 then intrusion
Akvq	if srv_count<3 and synflood=0 and same_srv_rate=1.00 then intrusion
uagjA	if protocol_type=tcp and same_srv_rate=0.00 and wrong_fragment>1 and count<3 then intrusion
AAvA	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
AAwv	if protocol_type=tcp and diff_srv_rate>0.33 and same_srv_rate=1.00 then intrusion
AqsujaA	if protocol_type=tcp and count<3 and synflood=0 and num_compromised=0 and same_srv_rate=0.00 then intrusion
Aasf	if num_compromised=0 and land=1 then intrusion
AsAg	if wrong_fragment>1 and num_compromised=0 then intrusion
AAAtfg	if num_compromised>1 and land=1 and wrong_fragment>1 then intrusion
Aaib	if protocol_type=udp and count>3 then intrusion
Attt	if num_compromised>1 then intrusion
Avfrg	if land=1 and wrong_fragment>1 and same_srv_rate=1.00 and synflood>1 then intrusion
AAAtiA	if protocol_type=udp and num_compromised>1 then intrusion
AAAvA	if same_srv_rate=1.00 then intrusion
cwAA	if srv_count>25 and diff_srv_rate>0.33 then intrusion
Aaiog	if protocol_type=udp and land=0 and wrong_fragment>1 then intrusion
Ar	if synflood>1 then intrusion
Aagg	if wrong_flagment>1 then intrusion
AaffA	if land=1 then intrusion

Table 8. Initial Rules (Ctd.)

Rules	Meaning
Agr	if wrong_fragment>1 and synflood>1 then intrusion
Agq	if wrong_fragment>1 and synflood=0 then intrusion
Agiq	if protocol_type=udp wong_fragment>1 and synflood=0 then intrusion
Aw	if diff_srv_rate>0.33 then intrusion
Aagq	if protocol_type=tcp and wrong_fragment>1 and synflood=0 then intrusion
Aqv	if synflood=0 and same_srv_rate=1.00 then intrusion
Agu	if wrong_fragment>1 and same_srv_rate then intrusion
Aadr	if protocol_type=tcp and dst_host_count>88 and synflood>1 then intrusion
Ahmq	if protocol_type=icmp and dst_host_srv_count<160 and synflood=0 then intrusion
Aicq	if protocol_type=udp and srv_count>367 and synflood=0 then intrusion
Aha	if protocol_type=icmp and count>160 then intrusion
Afs	if land=1 and num_compromised=0 then intrusion
Aags	if protocol_type=tcp and wrong_fragment>1 and num_compromised=0 then intrusion
Av	if same_srv_rate=1.00 then intrusion
Af	if land=1 then intrusion
Afg	if land=1 and wrong_fragment>1 then intrusion
Aflp	if land=1 and sat_host_coun<203 and wrong_fragment=0 then intrusion
Afq	if land=1 and wrong_fragment=0 then intrusion
Agh	if protocol_type=icmp and wrong_flagment>1 then intrusion
Abc	if count>120 and srv_count>250 then intrusion
Aad	if protocol_type=tcp and dst_host_count>320 then intrusion
Aijv	if protocol_type=udp and count<10 and same_srv_rate=1.00 then intrusion
Akr	if srv_count<60 and synflood>1 then intrusion
Aabo	if protocol_type=tp and count>450 and land=0 then intrusion
Aeh	if protocol_type=icmp and dst_host_srv_count>255 then intrusion
Aopt	if land=0 and wrong_fragment>1 and num_compromised>1 then intrusion
Agr	if wrong_fragment>1 and synflood>1 then intrusion
Ahfg	if protocol_type=icmp and land=1 and wrong_fragment>1 then intrusion
Afw	if land=1 and diff_srv_rate>0.5 then intrusion
Act	if srv_count>46 and num_compromised>1 then intrusion

Table 9. New Rules (Ctd.)