Paper: ASAT-13-CE-14

13th International Conference on
AEROSPACE SCIENCES & AVIATION TECHNOLOGY,
ASAT- 13, May 26 – 28, 2009, E-Mail: asat@mtc.edu.eg
Military Technical College, Kobry Elkobbah, Cairo, Egypt
Tel : +(202) 24025292 – 24036138, Fax: +(202) 22621908

Performance Evaluation of a Genetic Algorithm Based Approach to Network Intrusion Detection System

B. Abdullah*, I. Abd-alghafar**, Gouda I. Salama** and A. Abd-alhafez**

Abstract: The purpose of the work described in this paper is to provide an intrusion detection system (IDS), by applying genetic algorithm (GA) to network intrusion detection system. Parameters and evolution process for GA are discussed in detail and implemented**.** This approach uses information theory to filter the traffic data and thus reduce the complexity. We use a linear structure rule to classify the network behaviors into normal and abnormal behaviors. This approach applied to the KDD99 benchmark dataset and obtained high detection rate up to 99.87% as well as low false positive rate 0.003%. Finally the results of this approach compared with available machine learning techniques.

Keywords: Intrusion Detection System, Genetic Algorithm, Open Source Weka software.

1. Introduction

Internet and local area networks are expanding at an amazing rate in recent years, not just in the terms of size, but also in the terms of changing the services offered and the mobility of users that make them more vulnerable to various kinds of complex attacks. While we are benefiting from the convenience that new technology has brought us, computer systems are exposed to increasing number and complexity of security threats.

Of particular importance, thus, is the ability of applying rapidly new network security policies in order to detect and react as quickly as possible to the occurring attacks.
Different techniques have been developed and deployed to protect computer systems against network attacks (anti-virus software, firewall, message encryption, secured network protocols, password protection). Despite all the efforts, it is impossible to have a completely secured system. Therefore, intrusion detection is becoming an increasingly important technique that monitors network traffic and identifies network intrusions such as anomalous network behaviors, unauthorized network access, or malicious attacks to computer systems.

Intrusion detection systems are typically classified with respect to placement as: host based or network based [1]. A host based IDS will monitor resources such as system logs, file systems and disk resources; whereas a network based intrusion detection system monitors the data passing through the network.

There are two general categories of intrusion detection systems (IDSs) [2] as: misuse detection and anomaly based. Misuse detection systems are most widely used and they detect

* Yemeni Armed Forces , belalarh@gmail.com
** Egyptian Armed Forces

intruders with known patterns as: network packet, like source address, destination address, source and destination ports or even some key words of the payload of a packet. These systems exhibit a drawback in the sense that only the attacks that already exist in the attack database can be detected, so this model needs continuous updating, but they have a virtue of having very low false positive rate. Anomaly detection systems identify deviations from normal behavior and alert to potential unknown or novel attacks without having any prior knowledge of them. They exhibit higher rate of false alarms, but they have the ability of detecting unknown attacks and perform their task of looking for deviations much faster.

Genetic algorithm (GA) field is one of the up-coming fields in computer security, especially in intrusion detection systems (IDS) [3, 4, 5]. GA operates on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to the solution of the problem that GA is trying to solve. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness value in the problem domain and breeding them together using the operators borrowed from the genetic process performed in nature, i.e. crossover and mutation. This process leads to the evolution of populations of individuals that are better adapted to their environment than the individuals that they were created from, just as it happens in natural adaptation.

Anup Goyal and Chetan Kumar [18], generate a rule using the principles of evolution in a GA to classify all types of smurf attack labels in the training data set. With false positive rate is also quite low at 0.2% and accuracy rate is as high as 100%.

The early effort of using GAs for intrusion detection can be dated back to 1995, when M. Crosbie and E. Spafford, [19] applied the multiple agent technology and GP to detect network anomalies. Each agent monitors one parameter of the network audit data and GP is used to find the set of agents that collectively determine anomalous network behaviors. This method has the advantage of using many small autonomous agents, but the communication among them is still a problem. Also the training process can be time-consuming if the agents are not appropriately initialized.

Li [20] propose a GA-based method to detect anomalous network behaviors. Both quantitative and categorical features of network data are included when deriving classification rules using GA. The inclusion of quantitative features may lead to increased detection rates. However, no experimental results are available yet.

Xiao *et al.* [21] present an approach that uses information theory and GA to detect abnormal network behaviors. Based on the mutual information between network features and the types of network intrusions, a small number of network features are closely identified with network attacks. Then a linear structure rule is derived using the selected features and a GA. The use of mutual information reduces the complexity of GA, and the single resulting linear rule makes intrusion detection efficient in real-time environment. However, the approach considers only discrete features.

Rest of the work is organized as follows. Section 2 gives the overview of the genetic algorithm. In section 3 genetic algorithm based IDS (GAIDS) employed. In section 4 the preprocessing and features extraction phase employed. In section 5 training and testing phase using GA is employed**.** Experimental setup is given in Section 6. Discussion and experimental results is written in Section 7. Conclusions are written in Section 8.

2. Genetic Algorithm Overview

Genetic algorithms (GA) are search algorithms based on the principles of natural selection and genetics. The bases of genetic algorithm approach are given by Holland [8] and it has been deployed to solve wide range of problems.

GA evolves a population of initial individuals to a population of high quality individuals, where each individual represents a solution of the problem to be solved. Each individual is called chromosome, and is composed of a predetermined number of genes [9]. The quality of each rule is measured by a fitness function as the quantitative representation of each rule's adaptation to a certain environment. The procedure starts from an initial population of randomly generated individuals. Then the population is evolved for a number of generations while gradually improving the qualities of the individuals in the sense of increasing the fitness value as the measure of quality. During each generation, three basic genetic operators are sequentially applied to each individual with certain probabilities, i.e. selection, crossover and mutation. The algorithm flow is presented in Fig. 1.

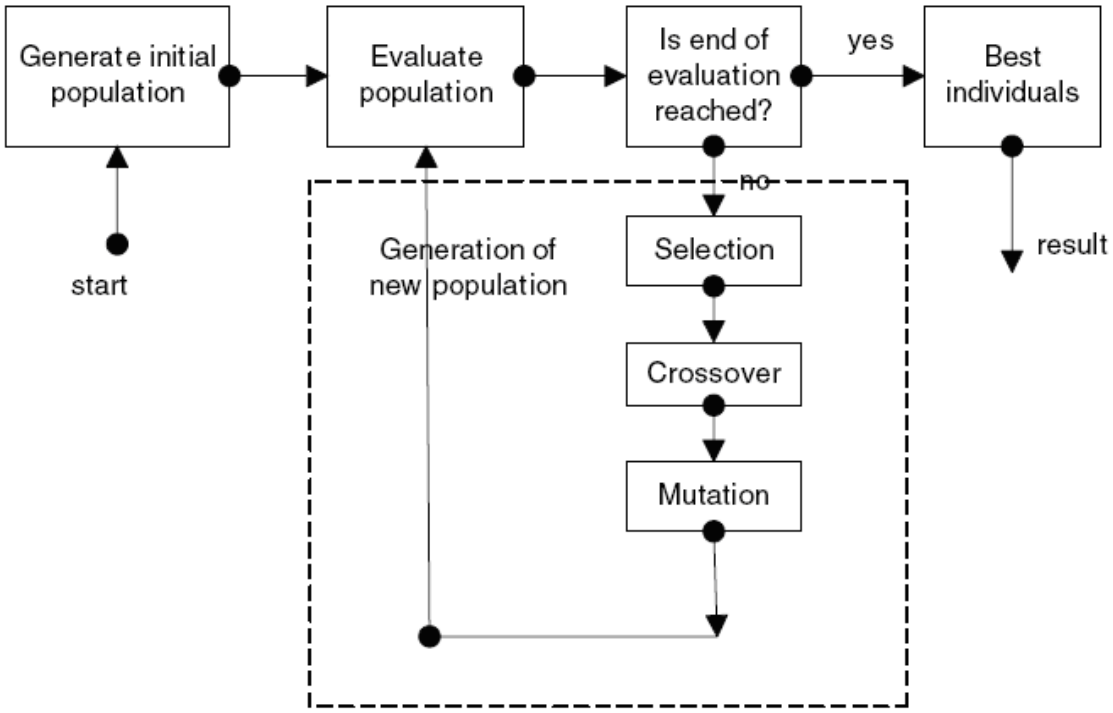

Fig.1. Genetic algorithm flow [2].

A genetic algorithm is quite straightforward in general, but it could be complex in most cases. For example, during the crossover operation, there could be one-point crossover or even multiple point crossovers. There are also parallel implementations of genetic algorithms. Sometimes series of parameters (for example, mutation rate, crossover rate, population size, chromosome size, number of evolutions or *generations*, and how the selection is done) needs to be considered with specific selection process that the **Selection**[11,12,13,15] is the stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (recombination or crossover). There are several generic selection algorithms, such as tournament selection, fitness proportionate selection (also known as *roulette-wheel selection*)**,** linear Ranking Selection and exponential Ranking Selection. The final goal is to search the solution space in a relatively short period of time [10].

3. Genetic Algorithm based IDS (GAIDS)

The proposed Genetic Algorithm Intrusion Detection System (GAIDS) is depicted in fig.2 which consists of the following two phases:

- **Pre-processing and Features Extraction Phase:** randomly select two separated training and testing data sets from the full DARPA data set, convert the symbolic features into numerical ones, normalize the data set, and select most suitable features.
- **Training and Testing Phase:** Initialize population, evaluate this population, create new generation, evaluate new generation, and apply genetic operators on the new generation until the most suitable individual is reached, and we use the most suitable individual for training and testing phase as following:

  Training Phase: In this phase GAIDS is trained using the training data set.

  Testing Phase: Measures the performance of the system to the testing data set.

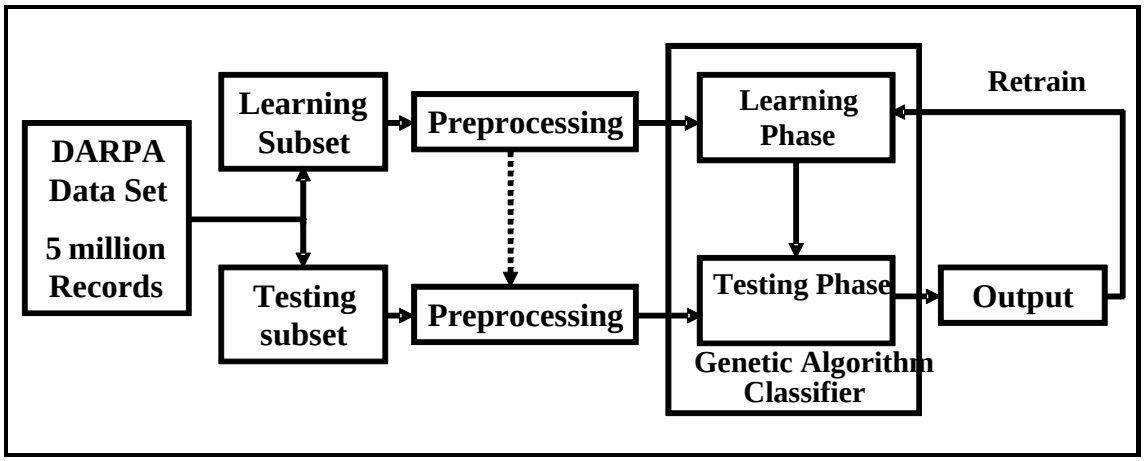

Fig.2.The simple structure of the proposed model

4. The Features Extraction and preprocessing phase

Figure.3 shows a detailed block diagram of the GAIDS preprocessing phase.

- **Data Sets Extraction:** Two subsets of the full data set were randomly selected and used as training and testing data sets.
- **Symbolic to Numerical Conversion:** Some features have symbolic form (e.g. protocol type). These features were converted into numerical ones by assigning a unique number for each feature. The resulting map is used to do the same for the testing data set.
- **Normalization:** It is often useful to scale the inputs to fall within a specific range, in the proposed system Equation 1 was used to normalize the training and testing data sets, [17].

$$X_n = 2 * (X - X_{min}) / (X_{max} - X_{min}) - 1 \quad (1)$$

where: X_{min}, X_{max}: are the Minimum and maximum value of the original inputs, respectively.
X_n: is the normalized output.

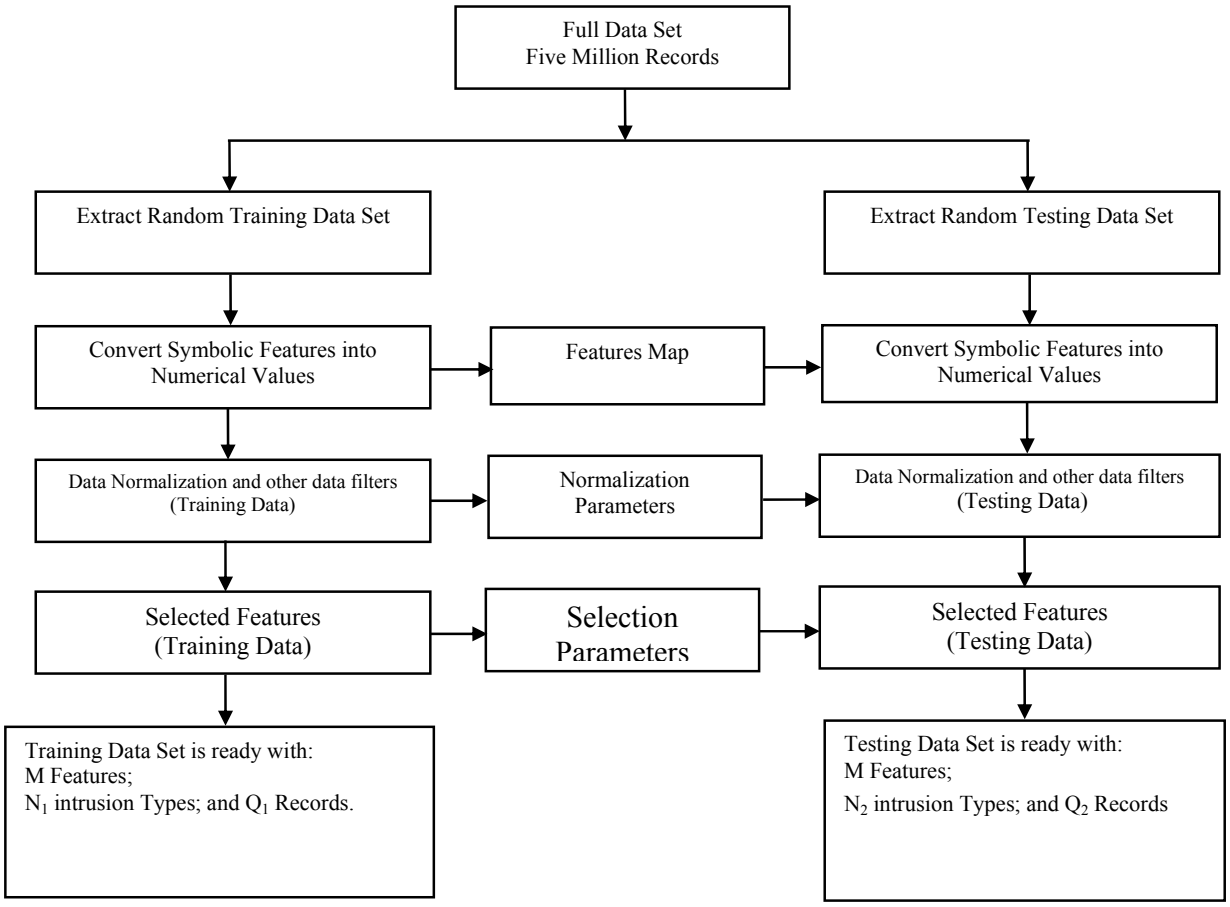

Fig 3. GAIDS preprocessing phase Major components

The resulting output will fall in the range [-1, +1]. The training data set will have M features, N_1 intrusion type, and Q_1 records. The testing data set will have M features, N_2 intrusion type, and Q_2 records.

Features Selection

The all 41 features in the KDD cup data set are listed with its descriptions and its numbers in the appendix A.

This experiment reduces the dimension of the 41 features of the data set size that can be used in classifications by select two types of features:

First type, 18 features are selected over 41 features based on the target operating system i.e. (features that used in windows operating system) that showing below:

(duration, protocol_type {tcp,udp,icmp}, service {red_i,netbios_ns,urh_i,netbios_dgm,etc.), flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate). Second type 31 features are selected over 41 features based on the information gain that explained below.

-Information Gain Let S be a set of training set samples with their corresponding labels. Suppose there are *m* classes and the training set contains *si* samples of class *I* and s is the total number of samples in the training set. Expected information needed to classify a- given sample is calculated as [1]:

$$I(S_1, S_2, ..., S_m) = -\sum_{i=1}^{m} \frac{S_i}{S} \log_2 \left(\frac{S_i}{S}\right) \quad (2)$$

A feature *F* with values *{ f1, f2, …, fv }* can divide the training set into V subsets *{ S1, S2, …, Sv }* where *Sj* is the subset which has the value *fj* for feature *F*. Furthermore let *Sj* contain *sij* samples of class *i*. Entropy of the feature *F* is [1]:

$$E(F) = \sum_{j=1}^{v} \frac{S_{1j} + ... + S_{mj}}{S} * I(S_{1j}, ..., S_{mj}) \quad (3)$$

Information gain for F can be calculated as [1]:

$$\text{Gain(F)} = I(S_1, \ldots, S_m) - E(F) \quad (4)$$

In [1], information gain is calculated for class labels by employing a binary discrimination for each class. That is, for each class, a dataset instance is considered in-class, if it has the same label; out-class, if it has a different label. Consequently, as opposed to calculating one information gain as a general measure on the relevance of the feature for all classes, we calculate an information gain for each class. Thus, this signifies how well the feature can discriminate the given class (i.e. normal or an attack type) from other classes. Figure.4 in the appendix B shows the maximum information gain for each feature.

In addition, Table.1 details the most discriminative class label for each feature. For majority of the features (31 over 41), normal, smurf and Neptune are the most discriminative classes. That is to say, there are many features that can discriminate these classes accurately. There are 9 features with very small maximum information gain (e.g. smaller than 0.001), which contribute very little to intrusion detection. Moreover features 20 and 21 (outbound command count for FTP sessions and hot login, respectively) do not show any variations in the training set therefore they have no relevance to intrusion detection.

Table 1. List of features for which the class is selected most relevant[1]

Class Label	Relevant Features
normal	1, 6, 12, 15, 16, 17, 18, 19, 31, 32, 37
smurf	4, 25, 26, 29, 30, 33, 34, 35, 38, 39
Neptune	2, 3, 5, 23, 24, 27, 28, 36, 40, 41
back	10, 13
Land	7
Teardrop	8
ftp_write	9
guess_pwd	11
buffer_overflow	14
warezclient	22

5. Training and Testing Phase using GA

The proposed GA-based intrusion detection approach contains two modules where each work in a different stage. In the training stage, a set of classification rules are generated from network audit data using the GA in an offline environment. In the intrusion detection stage, the generated rules are used to classify incoming network connections in the real time environment. Once the rules are generated, the intrusion detection is simple and efficient. In the following sections, we focus our discussions on deriving the set of rules using GA. To determine a fitness value of each rule, the following fitness function is deployed [16]:

$$fitness = \frac{\alpha}{A} - \frac{\beta}{B} \quad (5)$$

Where α is the number of correctly detected attacks, A is the total number of attacks in the training dataset, β is the number of normal connections incorrectly characterized as attacks, i.e. false-positives, and B is the total number of normal connections in the training dataset. Scale of fitness values is [-1, 1], where -1 is the lowest and 1 the highest value. High detection rate and low rate of false-positives result in a high fitness value. On the other side, low detection rate and high rate of false-positives result in a low fitness value.

5.1. Detection Algorithm Overview

List 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. In each generation, the qualities of the 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 4-19 calculate the fitness of each rule and select the best-fit rules into new population. Lines 20-23 apply the crossover and mutation operators to each rule in the new population. Finally, line 24 checks and decides whether to terminate the training process or to enter the next generation to continue the evolution process.

6. Experimental Setup

6.1. Objective

KDDCUP 99[6] data set used to train and test the system classifier. The dataset has been provided by MIT Lincoln Labs [6]. It contains a wide variety of intrusions simulated in a military network environment set up to acquire nine weeks of raw TCP/IP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. The LAN was operated as if it were a true Air Force environment, peppered with multiple attacks. Hence, this is a high confidence and high quality data set. They set up an environment to collect TCP/IP dump rows from a host located on a simulated military network. Each TCP/IP connection is described by 41 discrete and continuous features (e.g. duration, protocol type, flag, etc.) that listed in appendix A, and labeled as either normal, or as an attack, with exactly one specific attack type (e.g. Smurf, Perl, etc.). These 22 attacks and normal labels concluded in the available 10% of the DARPA data set are listed with number of samples for each class (attack, normal) and category for each class in Table.2. Attacks fall into four main categories:

List 1. Major steps of the detection algorithm.

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 Apply the 4 different selection algorithms
17. Select the chromosome into new population
18. End if
19. End for
20. For each chromosome in the new population
21. Apply crossover operator to the chromosome
22. Apply mutation operator to the chromosome
23. End for
24. If number of generations is not reached, go to line 4

(i) Denial of Service Attacks (DOS) in which an attacker overwhelms the victim host with a huge number of requests.
(ii) User to Root Attacks (U2R) in which an attacker or a hacker tries to get the access rights from a normal host in order, for instance, to gain the root access to the system.
(iii) Remote to Local Attacks (R2L) in which the intruder tries to exploit the system vulnerabilities in order to control the remote machine through the network as a local user.
(iv) Probing in which an attacker attempts to gather useful information about machines and services available on the network in order to look for exploits.

The scope of these experiments was focused to generate classifiers or rules for different attack types and one normal type belonging to different classes. The training data set contains 42674 connection records of the available 10% of the training set containing as published by Lincoln Labs which contains 494,021 connections for training of the attacks and normal type and the testing data set contains 31016 connection records of the attacks and normal type, as shown in Table 3. Hence, we wanted to create a rule that can classify all of these connections with a

minimal false positive rate. Although we would have apply our implementation to the attacks and normal types and connection features, the enormous training time complexity of the algorithm, very large data sets and lack of time restricted us. The training and testing connection records are classified in our experiments with the two types of the selected features that showed in section (7). From the Table 2, clearly that the attacks in the denial of service(DOS) category and normal category represent 98.93% of the available 10% KDD data set, i.e. attacks in the denial of service(DOS) category and normal category contains 488735 connection records from the 10% KDD data set that contains 494020 connection records, so in our experiments we focused the our classification on the two category (Normal & DOS).

6.2. Tools

For our implementation, we have used the GALIB java library [23] especially suited to develop Gas and java runtime environment jre1.5. Owing to the large hypothesis search space and high time complexity, we wanted to use a tool or library that is high on performance and computing speed. After a comprehensive survey of many tools, we decided to use GALIB since it is a java library, has been widely used by other researchers and well documented. We used a windows operating system with a Pentium 4 processor, 160GB of hard disk space and 1 GB of RAM to execute the computer program.

Table 2. Class labels that appears in 10% KDD" dataset work [1].

.Attack	#Sample	Category.
Smurf.	280790	DoS
Neptuno	107201	DoS
Back.	2203	DoS
Teardrop.	979	DoS
Pod.	264	DoS
Land.	21	DoS
Normal.	97277	normal
Satan.	1589	probe
Ipsweep.	1247	probe
Portsweep.	1040	probe
Nmap.	231	probe
Warezclient.	1020	r2l
Guess_passwd.	53	r2l
Warezmaster.	20	r2l
Imap.	12	r2l
ftp_write.	8	r2l
multihop.	7	r2l
phf.	4	r2l
spy.	2	r2l
buffer_overflow	30	u2r
rootkit	10	u2r
loadmodule.	9	u2r
perl.	3	u2r

Table 3. Class distribution for training and test data for the our dataset

	Normal	Intrusions (DOS)	Total
Train	32957	9717	42674
Test	26782	4234	31016

The University of Waikato in New Zealand [14], implements all the machine learning algorithms in the open source weka software called Weka. Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

6.3. Performance measures

To evaluate our system, besides the classical accuracy measure, the two standard metrics of detection rate and false positive rate developed for network intrusions, have been used. Table 4 shows these standard metrics [22]. Detection rate (DR) is computed as the ratio between the number of correctly detected intrusions and the total number of intrusions, that is

$$DR = \frac{\#\text{TruePositive}}{\#\text{FalseNegative}+\#\text{TruePositive}} \quad (6).$$

False positive(FP) (also said false alarm) rate is computed as the ratio between the numbers of normal connections that are incorrectly classifies as intrusions and the total number of normal connections, that is $\text{FP} = \frac{\#\text{FalsePositive}}{\#\text{TrueNegative}+\#\text{FalsePositive}}$ (7).

Table 4. Standard metrics to evaluate intrusions [22].

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	True Negative		False Positive	
	Intrusions (DOS)	False Negative		True Positive	

7. Discussion and Experimental Results

This section report two different experiments to show the performance of the GA compared with the available machine learning algorithms in open source weka software. The first experiment used 18 out of 41 features and the second experiment used 31 out of 41 features.

7.1. Experiment 1

For the first experiment that used 18 out of 41 features that explained in section 4. In GAIDS we were able to create a rule that could successfully classify data with 99.9695% training

accuracy, and with 99.971% testing accuracy, that shown in Table 13. And with the false positive rate is 0.003% and the detection rate of intrusions is 99.87%**,** that shown in Table 5**:**

In the open source weka software, we use three algorithms of the available algorithm in this software, and these algorithms that we used: decision tree (j48 algorithm), Bayes Network algorithm (BayesNet), and Support vector machine (SMO (Sequential Minimal Optimization)). And after we applied the same training and testing data that used in our work (GAIDS) to these algorithms, we obtain the following: In the j48 algorithm we obtained classification with 99.98% training accuracy, and with 99.958% testing accuracy, that shown in Table 13. And the false positive rate is 0.006% and the detection rate of intrusions is 99.98%**,** that shown in Table 6.

Table 5. Evaluate intrusions using GA in GAIDS exp 1.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	32956	99.99	1	0.003
	Intrusions (DOS)	12	0.12	9705	99.87

Table 6. Evaluate intrusions using j48 in weka for exp 1.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	32955	99.993	2	0.006
	Intrusions (DOS)	2	0.02	9715	99.98

In the BayesNet algorithm we obtained classification with 99.97% training accuracy, and with 99.983% testing accuracy, that shown in Table 13. And the false positive rate is 0.021% and the detection rate of intrusions is 99.98%**,** that shown in Table 7**:**

Table 7. Evaluate intrusions using BayesNet in weka for exp 1.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	32950	99.97	7	0.021
	Intrusions (DOS)	2	0.02	9715	99.98

In the SMO algorithm we obtained classification with 100% training accuracy, and with 99.994% testing accuracy, that shown in Table 13. And the false positive rate is 0% and the detection rate of intrusions is 100%**,** that shown in Table 8**:**

Table 8. Evaluate intrusions using SMO in weka for exp 1.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	32957	100	0	0
	Intrusions (DOS)	0	0	9717	100

7.2. Experiment 2

For the second experiment that used 31 out of 41 features that explained in section 4. In GAIDS we were able to create a rule that could successfully classify data with 99.582% training accuracy, and with 99.67% testing accuracy, that shown in Table 13. And with the false positive rate is 0.5% and the detection rate of intrusions is 99.82%**,** that shown in Table 9**:**

Table 9. Evaluate intrusions using GA in GAIDS exp 2.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	31208	99.5	158	0.5
	Intrusions (DOS)	20	0.177	11287	99.82

After we applied the same training and testing data that used in our work (GAIDS) to the algorithms in weka software, we obtain the following:. In the j48 algorithm we obtained classification with 99.997% training accuracy, and with 99.987% testing accuracy, that shown in Table 13. And the false positive rate is 0% and the detection rate of intrusions is 99.991%**,** that shown in Table 10**:**

Table 10. Evaluate intrusions using j48 in weka for exp 2.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	31366	100	0	0
	Intrusions (DOS)	1	0.0088	11306	99.991

In the BayesNet algorithm we obtained classification with 99.75% training accuracy, and with 99.67% testing accuracy, that shown in Table 13. And the false positive rate is 0.293% and the detection rate of intrusions is 99.90%**,** that shown in Table 11**:**

Table 11. Evaluate intrusions using BayesNet in weka for exp 2.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	31274	99.7	92	0.293
	Intrusions (DOS)	11	0.097	11296	99.90

In the SMO algorithm we obtained classification with 100% training accuracy, and with 99.9903% testing accuracy, that shown in Table 13. And the false positive rate is 0% and the detection rate of intrusions is 100%**,** that shown in Table 12**:**

Table 12. Evaluate intrusions using SMO in weka for exp 2.

		Predicted label			
		Normal		Intrusions (DOS)	
		Samples	Rates%	Samples	Rates%
Actual class	Normal	31366	100	0	0
	Intrusions(DOS)	0	0	11307	100

Table 13 compares our approach with the three algorithms (j48, BayesNet, SMO) in the weka software for the Exp 1 and Exp 2. In particular we show the training accuracy, testing accuracy, detection rate, and the false positive rate. In exp 1 the testing accuracy in our GAIDS is better than the BayesNet algorithm in weka but the best training and testing accuracy 100%, 99.994% respectively, obtained in the SMO algorithm in weka, and the false positive rate in our GAIDS 0.003% is very low, better than the false positive rate in the j48 and BayesNet algorithms in weka, but the false positive rate in SMO algorithm in weka is the best that is 0%. And the detection rate 99.87% in our GAIDS is very good but lower than detection rate in the j48, BayesNet, and SMO algorithms in weka, but the best detection rate 100% obtained in the SMO algorithm in weka.

Table 13. Comparisons between our GAIDS and three algorithms in weka software.

	Exp 1				**Exp 2**			
Algorithms	Training accuracy %	Testing accuracy%	Detection rates%	FP rates %	Training accuracy%	Testing accuracy%	Detection rates%	FP rate%
GAIDS	**99.9695**	**99.971**	**99.87**	**0.003**	**99.582**	**99.67**	**99.82**	**0.5**
J48 in weka	99.983	99.58	99.98	0.006	99.9977	99.987	99.991	0
BayesN-et in weka	99.978	99.9839	99.98	0.021	99.7586	99.6744	99.90	0.293
SMO in weka	100	99.994	100	0	100	99.9903	100	0

In exp 2 the testing accuracy in our GAIDS is the same as the BayesNet algorithm in weka but the best testing and training accuracy 100%, 99.9903% respectively, obtained in the SMO algorithm in weka, and the false positive rate in our GAIDS 0.5% is low, but lower than the false positive rate in the j48, BayesNet, and SMO algorithms in weka, but the false positive rate in SMO and j48 algorithm in weka is the best that is 0%. And the detection rate 99.82% in our GAIDS is very good but lower than detection rate in the j48, BayesNet, and SMO algorithms in weka, but the best detection rate 100% obtained in the SMO algorithm in weka.
In general the our results obtained from our GAIDS in experiment 1 is better than the our results obtained from our GAIDS in experiment 2, more specifically in the false positive rate that in exp 1 is 0.003% but in exp 2 is 0.5%, so that the selected 18 out of 41 features is most used and better than the selected 31 out of 41 features.

8. Conclusions

In this work genetic algorithm approach was deployed to intrusion detection. Genetic algorithm was used to obtain classification rules for intrusion detection. GA-approach demonstrated that can be used either to classify network connections as either normal or intrusive or further classify attacks by their type.

Our system is using two types selected features of the network connections 18 out of 41 features that used for the target operating system (windows) and 31 out of 41 features selected using information theory to identify the most important features of network connections, that maintaining high detection rates, so it can perform intrusion detection process fast and could be applied to high speed networks.

High attack detection rate and low false-positive rate demonstrate advantages of applying GA technique to intrusion detection, that by using 18 out of 41 features in experiment 1 this approach obtain high detection rate of intrusion is 99.87% and low false positive rate that is 0.003%, but when 31 out of 41 features used in experiment 2 we obtain detection rate and false positive rate lower than experiment 1, so the that 18 out of 41 features that used in windows OS is better than the 31 out of 41 features obtained by information gain.

References:

[1] H. Güneş Kayacık, A. Nur Zincir-Heywood, Malcolm I. Heywood,. " Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets". Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax, Nova Scotia. B3H 1W5 2006. http://www.cs.dal.ca/projectx/ .

[2] Zorana Bankovic, Dus an Stepanovic, Slobodan Bojanic, Octavio Nieto- Taladriz, "Improving network security using genetic algorithm approach". Computers and Electrical Engineering 33 (2007) 438–451.

[3] Gong RH, Zulkernine M, Abolmaesumi P. "A Software Implementation of a Genetic Algorithm based approach to Network Intrusion Detection". In: Proceedings of the sixth international conference on software engineering, artificial intelligence, networking and parallel/distributed computing and first ACIS international workshop on self-assembling wireless networks (SNPD/SAWN‘05), 2005.

[4] Chittur A. "Model Generation for an Intrusion Detection System Using Genetic Algorithms, publications/gaids-thesis01.pdf, accessed in 2006. http://www1.cs.columbia.edu/ids/

[5] Folino G, Pizzuti C, Spezzano G. "GP ensemble for Distributed Intrusion Detection Systems". In ICAPR, 3rd international conference on advances in pattern recognition, LNCS, Springer Verlag, 3686/2005, Bath, UK, August 2005.

[6] KDD Cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, October 1999.

[7] Lu W, Traore I. "Detecting new forms of network intrusion using genetic programming". Comput Intell;20(3):470–90, 2004

[8] Holland J. "Adaptation in natural and artificial system. Ann Arbor". The University of Michigan Press; 1975.

[9] S.N.Sivanandam · S.N.Deepa. "Introduction to Genetic Algorithms". Springer-Verlag Berlin Heidelberg 2008.

[10] Pohlheim, Hartmut.. "Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms". 30 Oct. 2003.
URL: http://www.geatbx.com/docu/algindex.html.

[11] Brad L. Miller and David E. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise"1995.

[12] http://en.wikipedia.org/wiki/Fitness_proportionate_selection

[13] Kwang Y. Lee, Mohamed A. El-Sharkawi, "Modern heuristic optimization techniques"2008.

[14] Weka: Data Mining Software, http://www.cs.waikato.ac.nz/ml/weka/

[15] Topias Blickle, Loather Thiele." A Comparison of Selection Schemes used in Genetic Algorithm", Swiss Federal Institute of Technology (ETH) Gloriastresse 35, 8092 Zurich,Nr. 11,December 1995.

[16] Chittur A. "Model Generation for an Intrusion Detection System Using Genetic Algorithms", 2006.
http://www1.cs.columbia.edu/ids/publications/gaids-thesis01.pdf

[17] Khaled Labib and V. Rao Vemuri, "NSOM: A Real-time Network-Based Intrusion Detection System Using Self-Organizing Maps", *University of California, Davis, 2002.*

[18] Anup Goyal and Chetan Kumar "GA-NIDS: A Genetic Algorithm based Network Intrusion Detection System", 2008.

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

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

[21] T. Xiao, G. Qu, S. Hariri, and M. Yousif, "An Efficient Network Intrusion Detection Method Based on Information Theory and Genetic Algorithm", *Proceedings of the 24th IEEE International Performance Computing and Communications Conference (IPCCC '05),* Phoenix, AZ, USA. 2005.

[22] Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano, "GP Ensemble for Distributed Intrusion Detection Systems". ICAPR 54-62, 2005.

[23] Genetic Programming Classifiers Library,
http://sourceforge.net/project/downloading.php?group_id=211385&filesize=10861307&filename=WekaGP-3-4-12.zip&90497168

Appendix A. Description of KDD 99 Intrusion Detection Dataset Features

Table A.1. List of features with their descriptions and data types (summarized from [1])

Feature	Description	Type	Feature	Description	Type
1. duration	Duration of the connection.	Cont.	22. is guest login	1 if the login is a "guest" login; 0 otherwise	Disc.
2. protocol type	Connection protocol (e.g. tcp, udp)	Disc.	23. Count	number of connections to the same host as the current connection in the past two seconds	Cont.
3. service	Destination service (e.g. telnet, ftp)	Disc.	24. srv count	number of connections to the same service as the current connection in the past two seconds	Cont.
4. flag	Status flag of the connection	Disc.	25. serror rate	% of connections that have "SYN" errors	Cont.
5. source bytes	Bytes sent from source to destination	Cont.	26. srv serror rate	% of connections that have "SYN" errors	Cont.
6. destination bytes	Bytes sent from destination to source	Cont.	27. rerror rate	% of connections that have "REJ" errors	Cont.
7. land	1 if connection is from/to the same host/port; 0 otherwise	Disc.	28. srv rerror rate	% of connections that have "REJ" errors	Cont.
8. wrong fragment	number of wrong fragments	Cont.	29. same srv rate	% of connections to the same service	Cont.
9. urgent	number of urgent packets	Cont.	30. diff srv rate	% of connections to different services	Cont.
10. hot	number of "hot" indicators	Cont.	31. srv diff host rate	% of connections to different hosts	Cont.
11. failed logins	number of failed logins	Cont.	32. dst host count	count of connections having the same destination host	Cont.
12. logged in	1 if successfully logged in; 0 otherwise	Disc.	33. dst host srv count	count of connections having the same destination host and using the same service	Cont.
13. # compromised	number of "compromised" conditions	Cont.	34. dst host same srv rate	% of connections having the same destination host and using the same service	Cont.
14. root shell	1 if root shell is obtained; 0 otherwise	Cont.	35. dst host diff srv rate	% of different services on the current host	Cont.
15. su attempted	1 if "su root" command attempted; 0 otherwise	Cont.	36. dst host same src port rate	% of connections to the current host having the same src port	Cont.
16. # root	number of "root" accesses	Cont.	37. dst host srv diff host rate	% of connections to the same service coming from different hosts	Cont.
17. # file creations	number of file creation operations	Cont.	38. dst host serror rate	% of connections to the current host that have an S0 error	Cont.
18. # shells	number of shell prompts	Cont.	39. dst host srv serror rate	% of connections to the current host and specified service that have an S0 error	Cont.
19. # access files	number of operations on access control files	Cont.	40. dst host rerror rate	% of connections to the current host that have an RST error	Cont.
20. # outbound cmds	number of outbound commands in an ftp session	Cont.	41. dst host srv rerror rate	% of connections to the current host and specified service that have an RST error	Cont.
21. is hot login	1 if the login belongs to the "hot" list; 0 otherwise	Disc.			

Appendix B: Information gain of each feature

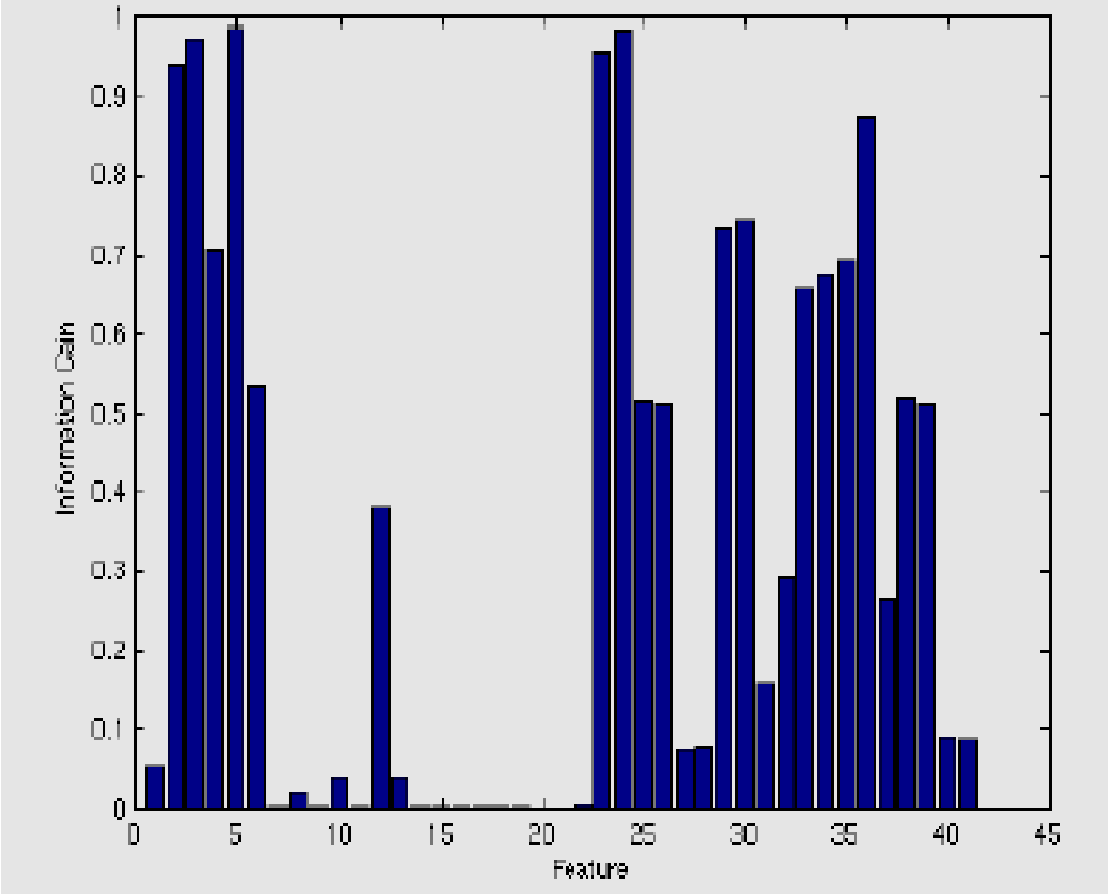

Fig 4. Information gain of each feature