VOL. 3, NO. 8 Aug, 2012 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2012 CIS Journal. All rights reserved.

http://www.cisjournal.org

Genetic Algorithm Rule-Based Intrusion Detection System (GAIDS)

[1] A.A. Ojugo, [2] A.O. Eboka, [3] O.E. Okonta, [4] R.E Yoro (Mrs), [5] F.O. Aghware
[1]Department of Mathematics /Computer Sci., Federal University of Petroleum Resources Effurun, Delta State
[2,3] Department of Computer Science Education, Federal College of Education (Technical) Asaba, Delta State
[4] Department of Computer Science, Delta State Polytechnic Ogwashi-Uku, Delta State
[5] Department of Computer Science Educations, College of Education, Agbor, Delta State
[1] ojugo_arnold@yahoo.com, [1] arnoldojugo@yahoo.com, [1] maryarnoldojugo@gmail.com
[2] an_drey2k@yahoo.com, [3] okeyokonta@yahoo.com
[4] rumerisky@yahoo.com, [5] aghwarefo@yahoo.com

ABSTRACT

This study examines the detection of attacks or network intrusion by users referred to as hackers (whose aim is to gain illegal entry as well as access to a network system and resources. Network and data security has become a pertinent issue with the advent of the Internet; though the Internet comes with a lot of merits on its own. Traditional used methods for data security includes the use of passwords, cryptography to mention few. The approach considered here is Intrusion Detection System, which is a software, driver or device used to prevent an unauthorized or illegal access to data in a networked system. Most of the existing IDS are implemented via rule-based systems where new attacks are not detectable. This study thus, presents a genetic algorithm based approach (with its driver implementation), which employs a set of classification rule derived from network audit data and the support-confidence framework, utilized as fitness function to judge the quality of each rule. The software implementation is aimed at improving system security in networked settings allowing for confidentiality, integrity and availability of system resources.

Keywords: *Soft computing, intrusion detection systems, network, security, genetic algorithm, attacks.*

1. INTRODUCTION

Our society's dependence on digitally transmitted data is ever-growing with an growing need to desuade intruders – leading to advances in cryptography and data security methods such as firewalls, application gateways etc. Despite these, to ensure data integrity is still a herculean task. Thus, attention **drifts** to intrusion detection system (IDS), which monitors network traffic so as to identify resources misuse, unauthorized use as well as its abuse.

Artificial Intelligence aims to create intelligent system capable of human reasoning; while Soft Computing aims to merge AI with other fields, creating a synergy that solve tasks by exploiting numeric data and human knowledge via mathematical models and reasoning – to yield a method tolerant to partial truth, imprecision and noise in data via optimization process that assures users of a solution guaranteed of high quality even with noise at its data input. Pushing further the bounds of SC has led to development of Evolutionary Programming Algorithms that aims at quantitative (numeric) data processing so as to ensure qualitative knowledge/experience in form of natural languages – spanning across several branches inspired by evolution and behavioural patterns in biological populations. Example of SC and EPA includes Genetic Algorithm, Neural Network etc. They simply mimics natural population seeking food and space (Shi, 2004; Abarghouei, Ghanizadeh and Shamsuddin, 2009), which have proven efficient in complex, constraint satisfaction problems (Hu, Eberhart and Kennedy, 2007; Sedighizadeh and Masehian, 2009). In its attempt to explore dynamic processes, optimization has three feats namely:

a. **Robustness** helps to estimate system's effectiveness even with noise implementation.
b. **Continuous adaptation** (yields a result void of local minima with random immigrants of high diversity to slow convergence in the space and balances data exploitation and exploration, so that in learning the feats of change, its solution is biased accordingly).
c. **Flexibility** – decisions made based on uncertainty has its impacts in a system's state. Optimization aims to predicts a system's future with an algorithm that focuses on both its objective function (to make the system **flexible**) and facilitate adaptation (if necessary, with the ease of system's blackbox integration).

Genetic Algorithm (GA) is a search algorithm that mimics the evolution process in solving various tasks. They are considered an effective heuristic search technique inspired by concepts of biology via evolutionary computations (Vollmer, Alves-Foss and Manic, 2011). They are based on the principles of natural selection and genetics.

The study aims to generate signatures for a rule-based IDS via GA that will hopefully, create better rules.

2. IDS AND SOFT-COMPUTING METHODS

Intrusion is the set of actions that attempts to compromise integrity, confidentiality or availability of network resources; while an intruder is any user or group of users who initiates such intrusive action (Olusegun, Oluwatobi and Adewale, 2010). An **IDS** is engineered to generate an alert when it observes potentially malicious traffic (Kurose and Ross, 2010). It monitors packets from network connections and determines if it is an intrusive activity or not. Once an intrusion is detected, the IDS simply performs one of the following actions: (a) Logs in a message into system audit file to be later analyzed by network security experts, (b) Send email alert to a network **administrators**, and (c) Stops such connection to end an intruder's attack (as placed under Intrusion Prevention System) amongst many other functions.

Security threats are initiated **externally** or **internally** (Gong, Zulkernine, and Abolmaesumi, 2005; Kandeeban and Rajesh, 2007) to result in two types of intruders namely: (a) External Intruders with no authorized access to network resources. Thus, they attack via various penetration means, and (b) Internal Intruders who have authorized access to network resources.

IDS can be classified into four namely:

i. **Rule-Based/Signature-based IDS** – maintains an extensive database of attack signatures. A signature or rule may be a list of characteristics about a single packet (e.g. source/destination port numbers, protocol type, specific string bits in a packet payload, or series of packets) – normally created by a network security experts. This IDS sniffs each packet passing through, compares them against its signature database – so that if a packet(s) match, it generates an alert; otherwise, it proceeds (Kurose et.al, 2010). As the most widely used IDS – its demerits are: (a) it requires previous knowledge of the attack to generate an accurate signature and thus, is completely oblivious/blind to new attacks yet to be recorded by it, (b) if signatures match, it may be false alarm from a number of other feats and not a result of an attack, and (c) each packet must be compared against an extensive signature set – that may overwhelm the IDS with processing so that it fails to actually detect many malicious packets (consider a network where gigabits of packet flow in per seconds, the rule-based IDS may find it difficult to compare all packets against its gigantic database).

ii. **Anomaly-IDS** creates traffic profile via aiming for statistically unusual, packet streams like inordinate percentage of Internet Control Message Protocol (ICMP), ping sweeps and/or the port scan's sudden exponential growth. It does not rely on previous knowledge about existing attacks and can potentially detect new, undocumented attacks. Also, it is very challenging to distinguish between normal traffic and statistically unusual traffic (Kurose et.al, 2010).

iii. **Host-IDS** monitors file and process activities related to a software environment associated with a specific host and listens to network traffic to identify attacks to host – as data from a host is used to detect signs of intrusion as packets enters/exits a host (Li, 2004).

iv. **Network-IDS** identifies intrusive acts via monitoring traffic through network devices.

Diaz-Gomez and Hougen (2005) notes that IDS has three main components namely:

- **Sensors/Network Probe** tracks network traffic, system behavior and log files. It then translates raw data into events usable by an IDS monitors and taps in to access all the network connections.
- **Analysis Console** takes sensor output (network connections) as input, analyze them looking for signs of intrusion. It is the most critical component as it decides whether or not, a connection is an intrusion.
- **Policy Control/Response** generates reactions based on analysis engine's outcome. If the Analysis Console flags a network connection as an intrusion, this control performs several actions depending on the set policy by the network administrator. Examples includes actions as simple as Logging a particular network connection, or sending alert to administrator's e-mail etc. It also handles what action(s) to be taken when an intrusion is detected.

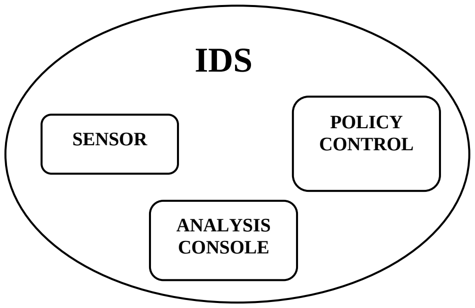


Fig 1: A Generalized Framework

2.1 GENERTIC ALGORITHM

GA is a search method that finds an approximate solution to an optimization task – inspired from biological, evolution process and natural genetics and proposed by

Holland (1975). GA uses hill climbing method from an arbitrary selected number of genes. GA has four operators: **initialization**, **selection**, **crossover** and **mutation**.

The system starts with randomly-initialized population that evolves, where each chromosome represents a solution in the space. Each chromosome has a number of genes, whose quality is a measured of its fitness function and the quantitative representation of each rule's adaptation (Kandeeban et.al, 2010). These three (seelction, crossover and mutation) operators are applied to the pool to gradually improve the quality of each chromosome. Competition for food and space allows evolves the population, as stronger genes dominate weaker ones – a desired improvement over various iterations. Chromosomes with better fitness value are more likely to reproduce offspring. Thus, only the fittest genes survive and reproduce. Its reproductive process creates diversity in the pool via evolution (combination of two individual chromosomes) as new genes are created from previous ones to generate a new pool. The exchange of genetic material between two parent chromosomes results in a **crossover** and creation of a better, **fitter** individual. Repeated **selection** (survival of fittest) and **crossover** causes continuous evolution of a generation that will better survive in a competitive state. **Mutation** causes the genes' sporadic and random alteration as well as helps regenerate lost genetic materials (Srinivas and Patnaik, 1994). If the newly generated pool contains an output close enough to the desired value, the solution is found; Else, the new pool goes through the same process and continues until a solution is reached, or until a certain number of generations have been produced.

GA's strength resides in parallel transversing of a system by proposing solutions whose initial population is randomly generated and are continuously evaluated via a fitness function (Diaz-Gomez and Hougen, 2005). GAs have successfully been applied in areas like AI, Biology, Engineering, Hydrology, to mention a few with variants developed to suit the nature of the task at hand – to yield new generation of strings via genetic operators (crossover and mutation) with its cycle or iterations repeated until a termination state is reached (Back, 1996). The algorithm is thus:

```
Genetic Algorithm() { initialize population via randomness;
                      evaluate randomness of population;
WHILE termination conditions not met do
        select solutions for next population;
        perform crossover and mutation;
        evaluate population;
END while };
```

2.2 GA and IDS (GAIDS)

Gong et al (2005) notes that GA has been applied to IDS directly to derive **classification rules** or via SC methods (GA used to select appropriate network connection features or determine the optimal parameters of some functions for the acquisition of rules).

Early effort in using GAs for IDS dates back to 1995, as applied in multiple agent technology and Genetic Programming (GP) to detect network anomalies. Each agent monitors one network parameter of the audit data, and Genetic Programming (GP) is used to find a set of agents that collectively determine network behavior anomalies. Its merit was in using many small autonomous agents, but, they had communication defect and its **training** was time-consuming, if agents were not appropriately initialized. The method is **obsolete** and unapplicable to current research (Chittur, 2001; Crosbie and Spafford, 1995).

Chittur (**2001)** in his rule-based IDS set the goal to test if GA was feasible for model generation, designed to replace/reinforce rule–based IDSs. He notes that network connection and its related behaviour can be translated to represent a rule – so that the set of rules are used to judge whether or not, a real-time connection is intrusive. The rules are modeled as chromosomes, so that the **generated** rule set becomes knowledge to judge whether a network connection is a potential intrusion. Fitness value is dependent on how many connections are grouped as attacks and how many attacks were correctly detected. The fitness value for each chromosome is evaluated in a closed range of [- 1, 1], with -1 as the poorest value and 1 is the ideal fitness. A threshold value was established and any certainty value exceeding this threshold value was classified as a malicious attack.

GA was then tested using data from raw TCP dump data from US Air Force DARPA dataset. It returned impressive results with its best rule having fitness value close to 1. The system presented about 97% of attacks correctly detected and 0.69% of normal connections were incorrectly classified as attacks. There was a correlation between these two variables (higher correct attack detection rate and a higher false positive rate) – so that as more attacks were correctly detected, more normal connections were incorrectly classified as attacks. Its optimal solution noted 386,703-of-396,743 attacks in its training data with 669-of-97,276 normal connections that was falsely classified as attacks. Also, GA successfully evolved an individual's model via randomized mutation so that the model generates a new pool over a subset data not previously known (retraining). Thus, GA ia able to generate a model with the desired characteristics of low false positive and high correct detection rates for an IDS. Its major bottleneck is difficulty in establishing a threshold value that may lead to detect novel or unknown attacks (Kandeeban et.al, 2010).

Li (2004) focused only on using GA to generate rules for the IDS knowledge-base – used to differentiate normal connections from intrusive attacks with rules in the form below. **Condition** is the matching of current network connection to rules in the knowledge base, **act** refers to actions defined by security policies within an organization such as reporting alert to system administrator, stopping the connection, logging a message into system audit files, or all of the above.

If {condition} then {act}
For example, a rule can be defined as:

if {the connection has following information:
source IP address 124.12.5.18;
destination IP address: 130.18.206.55;
destination port number: 21;
connection time: 10.1 seconds}
then {stop the connection}

Explained as thus: if there exists a network connection request with source IP address 124.12.5.18, destination IP address 130.18.206.55, destination port number 21, and connection time 10.1 seconds, then stop the connection establishment – since IP address 124.12.5.18 is recognized by the IDS as a blacklisted IP address. Thus, service request initiated from it, is rejected. The rules are tested on historical connections and to filter new connections to find suspicious network traffic. In its implementation, the network traffic used for GA was pre-classified data set (DARPA dataset 1999) that differentiates normal network connections from intrusive ones. Chromosomes encoded for the GA in Li's research was based on nine network connection feats namely: (a) source IP, (b) destination IP, (c) source port, (d) destination port, (e) state, (f) duration, (g) protocol, (h) number of bytes sent by originator, and (i) number of bytes sent by responder.

Each attribute over a range is encoded into chromosome with 57-genes via integer representation; while IP addresses are encoded in HEX for simplicity. The population of the chromosomes randomly-initialized for rule selection, evolves via crossover and mutation operators. Its fitness is a function of the **weighted sum model**, in which weights were used to indicate the significance of each network feature. Some network connection feat had more weight than others, especially as it becomes a **critical factor**, depended upon to detect an intrusion. For practical implementation, additional methods like neural network, used to accurately determine the weights of network connections (Gong et. al, 2005).

Gong et. al (2005) and **Li (2004)** had similarities and also used the same dataset to train their algorithm – though Gong clearly stated two (2) areas where the research differs from Li's: (i) **representation of rules** and (ii) **definition of fitness function**. Gong notes that in rule representation, Li's approach encodes only the "**condition**" part of the rules so that the method is only suitable for detecting network anomalies. However, Gong in their work encoded both "**condition**" and "**outcome**" – giving their work the precise benefit of detecting and classifying the types of network intrusion as well as detect network anomalies. Also, Li's work used **nine** network connection feats to encode chromosomes; where Gong used **six** feats from the same DARPA dataset, plus a classification of the attack type (manually added), making a total of seven (7) features. Network connection features encoded includes – destination IP, source IP, destination Port, source Port, duration, protocol, and finally attack type (manually added by experts). In fitness function, Gong shuns the weighted sum used by Li, but used the support – confidence framework that allowed the GA designed to generate rules used to detect anomalies or precisely classify various network types of network intrusions. Other works includes Kandeeban et.al, 2007; Al-Anni et al, 2009; Olusegun et.al, 2010; Lavender, 2010 etc., These made slight variations to the studies of Li or Gong.

Vollmer et al (2011) presents a combined approach that uses GA and anomaly-based IDS to create rules for a signature-based IDS. The study retrieved network packets as training data for GA as with previous studies. Its main difference is rather than using network audit data logs, network packets originate from network traffic identified by anomaly based IDS as being anomalous. The study produced set of optimal rules (rule-based) for a specific, anomalous-instance previously detected by an anomaly IDS. Thus, bridging rule and anomaly IDS. Its fitness function was defined for the study is the **three part fitness function.** The algorithm was demonstrated on anomalous ICMP network packets (input) and Snort rules (output of the algorithm). Output rules were sorted according to a fitness value and any duplicates were removed. The experimental results on ten test cases demonstrated a 100 percent rule alert rate. Out of 33,804 test packets 3 produced false positives.

3. STATEMENT OF THE PROBLEM

To ensure data integrity and privacy – security methods need be considered. The firewall's demerit is in its faulty packet filtering methodology. Application gateway's demerit is its **cost** and bottleneck caused as it slows down the network. Thus, IDS aims to provides a solution that tries to bridge existing network security technologies.

4. PURPOSE/OBJECTIVES OF STUDY

The paper's objective is to: (a) deploy a GA, Rule-Based IDS that creates rule for IDS, and (b) testing generated signatures with exsiting IDS benchmarks. The purpose is to employ GA to speed up rule generation, counters new attacks, and proffer security via a rule-based IDS. The study will also potentially reduce human effort of rule generation for a rule-based IDS as identified by Vollmer (2010). Its frees security experts from rule-creation, and allows LAN administrators to generate customized rules for the specific attacks they face at that level. Generating these new rules will hopefully detect new forms of attacks to a reasonable extent (its merits).

5. GAIDS FRAMEWORK

To generate new rules, GA uses sample data to train and test the newly created.

5.1 Training

Involves the following steps and a greater subset of the data is used evolve rules over a number of iterations as network feats are properly selected.

Table 1: Chromosome Representation of a Rule

S/no	Feat Name	Format	Number of Genes
1	Duration	H:M:S	3
2	Protocol	Numeric	1
3	Source Port	Numeric	1
4	Destination Port	Numeric	1
5	Source IP	a.b.c.d	4
6	Source IP	a.b.c.d	4
7	Attack Name and type	String	1

Training data contains analyzed logs of cconnection such that it knows apriori, connections that are attacks. Feat selection is a major task in designing the rule-based IDS (Chou et al, 2008; Kayacık et al, 2005) as in table 1. Source IP originatees an intrusion; while Destination IP is its target. Destination port shows **applications** that the target system is running (e.g. FTP service runs on port 21). Some IPs are more probable targets for intrusions such as Military Domain IPs (Li, 2004).

The training phase is further subdivided into these:

1. **Encoding Scheme** – Training phase uses a subset of DARPA dataset, which contains all 7-feats so that a connection considered intrusive, has an attack name. The data analysis prior to its use notes that normal connection in the training data contains no attack name. Each chromosome is a **rule** within which the 7-feats are encoded via fixed length vector, and each feat encoded as one or more genes of different types as in table 1. Each rule uses an if-then clause with a **"condition"** and **"outcome"** part**.** The first 6-feats are connected via logical AND to form **"condition"** part; while attack name is the "**outcome**" to show network record **classification** (during **training**) or **connection** (during intrusion detection) if a rule is matched. Below, a rule classified as port scan attack:
   IF (duration="0:0:1" **AND** protocol="telnet" **AND** source port=8982 **AND** destination port=23 **AND** source IP="9.9.9.9" **AND** destination IP="172.16.12.50") **then** (attack name="port-scan"). In the above example, given that the numbers 1 and 2 represents protocol telnet and attack port-scan as:
   {0, 0, 1, 2, 18982, 79, 9, 9, 9, 9, 172, 16, 012, 50, 1}

   Wild-card are allowed and may be used to make the rules more general. In case a wildcard is used, then the corresponding gene is encoded as -1. Thus, our previous example is a generalized rule applicable to all packets originating from network 9.9.*.* as:
   {0, 0, 1, 2, 18982, 79, 9, 9, -1, -1, 172, 16, 112, 50, 1}

2. **Fitness Function** – Chromosomes are evaluated at training to determine its goodness (attacks detected). If the chromosome correctly classifies an attack, it is considered **good**; else, it is **bad** and not be selected for crossover to produce offspring. Thus, the more attacks a chromosome detects, the higher its fitness value. The fitness models adopted is: **support and confidence model**. If we have the rule:

   If A then B,
   support = |A and B| / N
   confidence = |A and B| / |A|
   fitness = w1 * support + w2 * confidence
   N = Number of connections in training data
   |A| = Number of connections matching condition A.
   |A and B| = Connections matching rule if A and B
   w1, w2 = Weights to balance/control the two terms.

A merit of using fitness function is that, by changing the weights w1 and w2, the approach is simply used for either identifying network intrusions or precisely classifying types of intrusions. In the latter, w1 is set to 0 and w2 set to 1; while in the former, w1 is set to 1 and w2 is set to 0. Unlike other fitness functions, the selection criteria w1/w2 is not crucial factor to the performance of the approach.

Table 2: Workings of the Fitness Value Framework

Duration	Protocol	S.Port	D.Port	S. IP	D. IP	Attk name
0.0.11	ftp	1892	21	192.168.1.30	192.168.1.20	-
0 .0 .0	smtp	1900	25	192.168.1.30	192.168.1.20	-
0.0.2	**rsh**	**1023**	**1021**	**192.168.1.30**	**192.168.1.20**	**Rcp**
0.0.23	telnet	1906	23	192.168.1.30	192.168.1.20	guess
0.0.14	rlogin	1022	513	192.168.1.30	192.168.1.20	rlogin
0.0.2	**rsh**	**1022**	**1021**	**192.168.1.30**	**192.168.1.20**	**Rsh**
0.0. 15	ftp	43549	21	192.168.1.40	192.168.1.20	-

Table 2 is list of audit data with sample chromosomes to represent matching rules that identifies attack with each connection. Chromosomes match are seen in lines 3 and 6, to match attack type, rsh, only on line 6. Chromosome fitness is evaluated below if:

w1 = 0.2, w2 = 0.8, N = 10, |A| = 2, |A and B| = 1.

Thus, Fitness = w1 * support + w2 * confidence

Support = | A and B | / N = 1 / 10 = 0.1
Confidence = |A and B | / A = 1 /2 = 0.5
Fitness = (0.2 x 0.1) + (0.5 x 0.8) = 0.42

3. **Selection** – The study adopts **tournament selection** in which, chromosomes are **randomly** chosen from current generation – so that with next iteration, a lesser number is chosen. This continues until one is chosen from the last two/three chromosome, to be parents that creates the new **offspring**. This scheme is selected based on its reputation of maintaining population diversity and we recall, the goal of this study is not to create a **best rule (global optimum)**, but to create bunch of rules that are good enough to detect intrusion **(local optimums).** Thus, population diversity needs to be maintained. The algorithm is:

Algorithm: Tournament Selection {}

Input: Population of chromosome
Output: Selected Chromosome for crossover

a. Randomly select 3-chromosomes from pool
b. Pick best 2-chromosome based on fitness value
c. Return the selected two chromosomes
d. Apply Crossover | Select best chromosome to be parent

4. **Crossover** – algorithm chooses two random cross section points from the chromosome as in table 3, and exchanges the midsection between the parents

as in lines 1 and 2, to form two new children seen in lines 3 and 4. This is a two point crossover in GA.

Table 3: 2 point Crossover in GA from parent to children

Duration	Protocol	S.Port	D.Port	S. IP	D. IP	Attk name
0.-1.-1	rsh	-1	1021	192.168.-1.-1	192.168.0.-1	Rsh
0.0.5	tel- net	2020	23	9.9.9.9	172.16.112.50	Port scan
				Crossover Points		
0.-1.-1	rsh	-1	1021	192.9.9.9	172.16.112.-1	Rsh
0.0.5	telnet	2020	23	9.168.-1.-1	192.168.0.50	Port scan

5. **Mutation:** Each gene in a chromosome may or may not change depending on the probability of mutation rate. Mutation improves population **diversity** needed in this work, and its algorithm is as thus:

Algorithm: Mutation of rules (chromosome)

```
Input: A chromosome rule
Output: Same or Mutated chrom,., a fns of mutation rate
1. Set mutation threshold (between 0 and 1)
2. For each network attribute in chromosome
3. Generate a random number between 0 and 1
4.  If random number > mutation threshold then
5. Generate random value w.r.t data properties
6. Set chromosome attribute value with
   generated attribute value
7. End if
8. End For Each
```

Thus, the complete training phase algorithm is as thus:

Algorithm Rule set Generation via GA-Training

```
Input: Audit data, generations and population size.
Output: A set of classification rules.

1. Randomly Initial they created chromosome
   population.
2. Set w1 = 0.2, w2 = 0.8, Max Generations =
   400 (epoch)
3. Set N = total number of record in training set
4. Set generation Counter = 0
5. For each chromosome in population
6. Set A = 0, AB = 0
7. For Each record in training set
8. If record matches chromosome
9. AB = AB + 1   //AB++
10. End If
11. If record matches only condition part
12.  A = A+1  //A++
13.  End if
14.  End For Each record
15.  End For Each Chromosome
16.  Select 30-best fitted chromosome into new
   pool
17.  For each chromosome in new pool/population
18.  Select chromosome for breeding
19.  Apply crossover and mutation to new
   offspring
20. Place newly created chromosome into
   population
21. End For each
22. Kill old pool, new pool now current pool
23. Increment generation Counter by 1
24. If generation Counter < Max Generation then
25. Goto line 5
26. Else goto line 27
27. End
```

5.2 Testing Phase

The generated rules are used to evaluate the remaining dataset, and the aim of testing is to gather information of how well the rules created, can detect attacks. Two methods are used for testing namely: (a) use existing rules in the rule-based IDS, and/or (b) build tailored rule-based IDS. The proposed design requires tailored rules in that rules created from here are fed back into the IDS for detection purpose as the remaining part of the DARPA dataset are used as incoming connection to see if generated rules can distinguish between normal and intrusive connections. If they can, it means the GA, ruled based IDS can detect possibly new intrusions using previous knowledge of existing ones. Its algorithm is thus:

Algorithm: Intrusion Detection

```
Input: Inflowing network connection
Output: Decision if connection is intrusive or not

1.  Loop Forever {fetch incoming packet
   (network probe)}
2.  For each rule in base
3. Match rule to network connection (analysis
   console)
4. If rules match then
5.  Mark connection as intrusion (policy control)
6.  End if
7. End For Each
8. End Loop Forever
```

6. METHODS AND MATERIALS

The training and testing data-set used for the study is the DARPA (Defense Advanced Research Project Agency) dataset as in Appendix A1 and A2.

7. FINDINGS AND DISCUSSION

From Appendix A3, the top rules havee almost same fitness range [0.8, 0.8065]. Thus, the rules are estimated 80% good to be used in detection. This result also shows that the achievement of generating a bunch of good rules, rather than a single optimum rule – is better in intrusion detection. 10-out-of-22 rules have destination port as -1, so that the rules looks out for cconnections from any destination port. This increasees the chances of detecting intrusion on any port in the network and improves the generality of rules. The rule generator used a population of 400, w1 = 0.2, w2 = 0.8, 5000 (epochs) evolutions and 0.05 probability of a gene to be mutated respectively.

At testing, the rule are used as thus:

TCP: 192.168.1.30:1754 → 192.168.0.20:23
(test data network connection)
duration hours 0 minutes 0 seconds 23(connection duration)
rule hours -1 minutes 0 seconds 23 (rule duration)
Src Rule IP 192.168.1.30 (source ip address for rule)
Src Test IP 192.168.1.30 (source ip for net-connection)
Dst Rule IP 192.168.0.20 (destination IP for rule)
Dst Test IP 192.168.0.20 (destination IP for connection)
Src Rule Port -1 (source port number for rule)
Src Test Port 1754 (source port for net-connection)
Dst Rule Port 23 (destination port number for rule)
Dst Test Port 23 (destination port for net-connection)
An Intrusion Is Detected <<<

8. SUMMARY AND CONCLUSION

The study implements a GA, Ruled-based IDS used to generate a set of classification rules from network audit data with 7-network features when encoding such rules. A simple, efficient and flexible fitness function (**support-confidence**) was used to evaluate ggoodness of each rule (chromosome). Depending on the selection of fitness function weight values (w1 and w2), the generated rules can be used to either generally detect network intrusions or precisely classify the types of intrusions.

The training and testing data set as in used was the DARPA 1998 MIT Lincoln laboratory. It became the first standard corpora for evaluation, training and testing IDS. The system's architecture was divided into to two namely training and testing the IDS via GA. The study implemented GAIDS using C (programming language) in Linux operating system platform. The choice of C was due to its interaction as a system programming language, simplicity and speed of code execution, which are all critical factors in IDS. An IDS goes through millions of network connection each day, and it is expected that it inspect each one of them in order to determine which is an intrusion and which is not, hence the need for speed.

The implemented system was able to detect intrusion during testing phase – proving the capability of GA to generate rules via a training data. However, some limitations of the method are also observed. First, the generated rules were biased to the training data set. This was resolved by carefully selecting either the number of generations during **training** so as not to overtrain or the number of top best-fit rules so as not to overfit at intrusion detection phase. Secondly, the support-confidence framework may be simple to implement and provide improved accuracy to final rules, it requires the whole training data to be loaded into memory before any computation. For large training datasets, it is neither efficient nor feasible. Thus, system requires enough **cache** memory to hold the data.

9. RECOMMENDATIONS

The following recommendations were made:

- ✓ For the DARPA dataset, attack planning and verification were performed by hand due to parameter selection. These tasks proved to be very time intensive and hindered the timely completion of the evaluation. In the future, this process should be automated. All that should be required of the human operator is to supply a database of attacks and their variants. Once this database has been collected, an automated system should be able to plan out a schedule. Another approach to dataset building might be combining knowledge from different security sensors into a standard rule base is another promising area in this work.
- ✓ As a future work, other learning algorithms suitable for optimization can be implemented in a bid to achieve a secured environment for distributed computing. As there will always be the need to share resources, the issue of intrusion is unavoidable and as such the security of such systems becomes an important issue.

REFERENCES

[1] Abarghouei, A., Ghanizadeh, A and Shamsuddin, S., (2009): Soft computing methods in edge detection, J Soft Comp., 1(2), ISSN:2074-8523, PP.163-203.

[2] Al-Anni, M. K. and Sundararajan, V., (2009): Detecting a denial of service via AI tools and GSA, Indian J. Sci., 2(2), ISSN: 0974-6846, PP.16-21.

[3] Crosbie, M., and Spafford, G., (1995): Applying genetic programming to intrusion detection, http://www.aaai.org/Papers/Symposia/Fall/1995/FS-95-01/FS95-01-001.pdf.

[4] Chittur, A., (2001): Model generation for an intrusion detection system via genetic algorithms, http://www.hacktory.cs.columbia.edu/sites/default/files/gaids-thesis01.pdf.

[5] Chou, T. S., Yen K. K. and Lou, J., (2008): Network intrusion detection design using feature selection of soft computing paradigms, World Academy of Science, Engineering and Technology 47.

[6] Diaz-Gomez, P. and Hougen, D., (2005): Improved off-line intrusion detection using a genetic algorithm,

http://cameron.edu/~pdiaz-go/Art_ICEIS.pdf.

[7] Fausett, L., (1994): Fundamentals of Neural Networks, NJ: Prentice Hall, USA, 1st edition, ISBN-10: 0133341860.

[8] Gong, R. H., Zulkernine, M. and Abolmaesumi, P., (2005): A software implementation of GA based approach to network intrusion detection, http://www.cse.msu.edu/~cse848/2011/Student_papers/Tavon_Pourboghrat.pdf.

[9] Kandeeban, S. S. and Rajesh, R. S., (2007): GA for framing rules for intrusion detection, J. Comp. Sci and Security., 7(11), ISSN:1738-7906, PP.285-290.

[10] Kandeeban, S. S. and Rajesh, R. S., (2010): Integrated intrusion detection system using soft computing, J. Network Security, 10(2), ISSN:1816-353X, PP.87–92.

[11] Kayacik, H. G., Zincir-Heywood, A. N. and Heywood M. I., (2005): Selecting features for IDS: a feature relevance analysis on KDD 99 IDS dataset, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.7574&rep=rep1&type=pdf.

[12] Kurose, J. F. and Ross, K. N., (2010): Computer network a top down approach, Pearson publisher, ISBN-10: 0-13-136548-7.

[13] Lavender, B. E., (2010): Implementation of GA into IDS and integration into nprobe, http://brie.com/brian/netga/Lavender_Report.pdf.

[14] Li, W., (2004): A GA approach to network IDS, http://www.security.cse.msstate.edu/docs/Publications/wli/DOECSG2004.pdf.

[15] Olusegun, F., Oluwatobi, O. A. and Adewale O. O., (2010): ID-SOMGA: A self organizing migrating genetic algorithm-based solution for Intrusion Detection, Computer and information science, (3)(4), ISSN: 1913-8989, PP 80-92.

[16] Openshaw, W. R., (2003): Rainfall runoff processes: A workbook to accompany online module, http://www.engineering.usu.edu/dtarb/rrp.html.

[17] Schafer, J. D., (1985): Multiple objective optimization with vector evaluated genetic algorithms, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.5689&rep=rep1&type=pdf.

[18] Shanmugam, B. and Idris, N. B., (2011): Hybrid intrusion detection systems (HIDS) using fuzzy logic, http://www.intechopen.com/download/pdf/14361.

[19] Sreedharan, S., (2012): EE04 804(B) soft computing ver. 1.2, http://sudhinpk.files.wordpress.com/2012/03/class-1-on-21st-22nd-of-feb.pdf.

[20] Vollmer, T., Alves-Foss, J. and Manic, M., (2011): Autonomous rule creation for intrusion detection, http://www.inl.gov/technicalpublications/Documents/5025964.pdf.

APPENDIX A

Table A1: shows the DARPA Dataset used for Training

S/No	Duration (h:m:s)	Protocol	Source Port	Dest Port	Source IP	Dest IP	Attack Type
1	0:01:26	telnet	1754	23	192.168.1.30	192.168.0.20	-
2	0:00:14	ftp	1755	21	192.168.1.30	192.168.0.20	-
3	0:01:00	telnet	1769	23	192.168.1.30	192.168.0.20	-
4	0:00:03	finger	1772	79	192.168.1.30	192.168.0.20	-
5	0:00:03	smtp	1778	25	192.168.1.30	192.168.0.20	-
6	0:00:03	smtp	1783	25	192.168.1.30	192.168.0.20	-
7	0:01:11	telnet	43496	23	192.168.0.40	192.168.0.20	-
8	0:00:19	ftp	43497	21	192.168.0.40	192.168.0.20	-
9	0:00:12	ftp	1787	21	192.168.1.30	192.168.0.20	-
10	0:01:40	telnet	43501	23	192.168.0.40	192.168.0.20	-
11	0:00:12	ftp	43511	21	192.168.0.40	192.168.0.20	-
12	0:00:18	rlogin	1023	513	192.168.0.40	192.168.0.20	-

13	0:00:01	finger	1811	79	192.168.1.30	192.168.0.20	-
14	0:00:01	finger	1820	79	192.168.1.30	192.168.0.20	-
15	0:00:03	smtp	43517	25	192.168.0.40	192.168.0.20	-
16	0:00:03	smtp	43518	25	192.168.0.40	192.168.0.20	-
17	0:00:01	smtp	1826	25	192.168.1.30	192.168.0.20	-
18	0:00:03	smtp	1832	25	192.168.1.30	192.168.0.20	-
19	0:00:02	finger	1834	79	192.168.1.30	192.168.0.20	-
20	0:00:01	finger	1841	79	192.168.1.30	192.168.0.20	-
21	0:00:04	finger	1847	79	192.168.1.30	192.168.0.20	-
22	0:00:18	ftp	1850	21	192.168.1.30	192.168.0.20	-
23	0:00:01	finger	1855	79	192.168.1.30	192.168.0.20	-
24	0:00:22	telnet	1867	23	192.168.1.30	192.168.0.20	guess
25	0:00:03	smtp	43533	25	192.168.0.40	192.168.0.20	-
26	0:00:44	telnet	1876	23	192.168.1.30	192.168.0.20	-
27	0:00:00	smtp	43538	25	192.168.0.40	192.168.0.20	-
28	0:00:23	telnet	1884	23	192.168.1.30	192.168.0.20	guess
29	0:00:01	smtp	43541	25	192.168.0.40	192.168.0.20	-
30	0:01:40	telnet	1890	23	192.168.1.30	192.168.0.20	-
31	0:00:11	ftp	1892	21	192.168.1.30	192.168.0.20	-
32	0:00:00	smtp	1900	25	192.168.1.30	192.168.0.20	-
33	0:00:02	rsh	1023	1021	192.168.1.30	192.168.0.20	rcp
34	0:00:23	telnet	1906	23	192.168.1.30	192.168.0.20	guess
36	0:00:14	rlogin	1022	513	192.168.1.30	192.168.0.20	rlogin
36	0:00:02	rsh	1022	1021	192.168.1.30	192.168.0.20	rsh
37	0:00:15	ftp	43549	21	192.168.0.40	192.168.0.20	-
38	0:00:40	telnet	1914	23	192.168.1.30	192.168.0.20	guess
39	0:01:24	telnet	43560	23	192.168.0.40	192.168.0.20	-
40	0:00:13	ftp	43566	21	192.168.0.40	192.168.0.20	-
41	0:00:12	ftp	1932	21	192.168.1.30	192.168.0.20	-
42	0:00:02	finger	1933	79	192.168.1.30	192.168.0.20	-
43	0:00:02	finger	1939	79	192.168.1.30	192.168.0.20	-
44	0:00:01	finger	1946	79	192.168.1.30	192.168.0.20	-
45	0:00:20	ftp	43573	21	192.168.0.40	192.168.0.20	-
46	0:00:48	telnet	1959	23	192.168.1.30	192.168.0.20	-
47	0:00:53	telnet	1967	23	192.168.1.30	192.168.0.20	-
48	0:00:01	smtp	1976	25	192.168.1.30	192.168.0.20	-
49	0:00:11	ftp	43582	21	192.168.0.40	192.168.0.20	-
50	0:00:53	telnet	43587	23	192.168.0.40	192.168.0.20	-
51	0:00:36	telnet	1978	23	192.168.1.30	192.168.0.20	-
52	0:00:11	ftp	1984	21	192.168.1.30	192.168.0.20	-
53	0:00:57	telnet	43598	23	192.168.0.40	192.168.0.20	-
54	0:00:42	telnet	2016	23	192.168.1.30	192.168.0.20	-
55	0:00:34	telnet	43603	23	192.168.0.40	192.168.0.20	-

56	0:00:05	telnet	2020	23	192.168.1.30	192.168.0.20	port-scan
57	0:00:04	ftp	2022	21	192.168.1.30	192.168.0.20	port-scan
58	0:00:03	finger	2023	79	192.168.1.30	192.168.0.20	port-scan
59	0:00:04	rsh	2030	1021	192.168.1.30	192.168.0.20	port-scan
60	0:00:04	rlogin	2031	513	192.168.1.30	192.168.0.20	port-scan
61	0:00:05	exec	2032	512	192.168.1.30	192.168.0.20	port-scan
62	0:00:37	telnet	1042	23	192.168.1.30	192.168.0.20	-
63	0:00:01	smtp	1048	25	192.168.1.30	192.168.0.20	-
64	0:00:01	finger	1050	79	192.168.1.30	192.168.0.20	-

Table A2: DARPA Dataset for Testing

S/No	Source IP	Source Port	Dest IP	Dest Port	Protocol	Duration
1.	192.168.1.30	1754	192.168.0.20	23	ftp	0hour, 0min, 12secs
2.	192.168.0.20	20	192.168.1.30	1767	ftp	0hour, 0min, 2secs
3.	192.168.1.30	1767	192.168.0.20	20	ftp-data	0hour, 0min, 2secs
4.	192.168.1.30	1876	192.168.0.20	23	telnet	0hour, 0min, 23secs
5.	192.168.0.40	43494	192.168.1.30	25	smtp	0hour, 0min, 5secs
6.	192.168.1.30	1754	192.168.0.20	23	telnet	0hour, 0min, 12secs
7.	192.168.0.20	20	192.168.1.30	1767	ftp-data	0hour, 0min, 2secs
8.	192.168.1.30	1767	192.168.0.20	20	ftp-data	0hour, 0min, 2secs
9.	192.168.0.40	43494	192.168.1.30	25	smtp	0hour, 0min, 5secs
10.	192.168.1.30	1762	192.168.0.20	20	ftp-data	0hour, 0min, 5secs
11.	192.168.1.30	1755	192.168.0.20	21	ftp	0hour, 0min, 10secs
12.	192.168.1.30	1884	192.168.0.20	23	telnet	0hour, 0min, 5secs
13.	192.168.0.40	43493	192.168.1.30	25	smtp	0hour, 0min, 8secs
14.	192.168.1.30	1755	192.168.0.20	21	ftp	0hour, 0min, 11secs
15.	192.168.1.30	1762	192.168.0.20	20	ftp-data	0hour, 0min, 5secs
16.	192.168.1.30	1755	192.168.0.20	21	ftp	0hour, 0min, 10secs
17.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 2secs
18.	192.168.1.30	1755	192.168.0.20	21	ftp	0hour, 0min, 18secs
19.	192.168.0.20	20	192.168.1.30	1762	ftp-data	0hour, 0min, 13secs
20.	192.168.1.30	1769	192.168.0.20	23	telnet	0hour, 0min, 7secs
21.	192.168.1.30	1755	192.168.0.20	21	ftp	0hour, 0min, 17secs
22.	192.168.1.30	1768	192.168.0.20	20	ftp-data	0hour, 0min, 6secs
23.	192.168.1.30	1770	192.168.0.20	20	ftp-data	0hour, 0min, 5secs
24.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 12secs
25.	192.168.1.30	1893	192.168.0.20	20	ftp-data	0hour, 0min, 23secs
26.	192.168.1.30	1754	192.168.0.20	23	telnet	0hour, 0min, 6secs
27.	192.168.1.30	1900	192.168.0.20	25	smtp	0hour, 0min, 11secs
28.	192.168.1.30	1906	192.168.0.20	23	telnet	0hour, 0min, 2secs
29.	192.168.1.30	1892	192.168.0.20	21	ftp	0hour, 0min, 27secs
30.	192.168.0.40	43546	192.168.1.30	21	ftp	0hour, 0min, 8secs

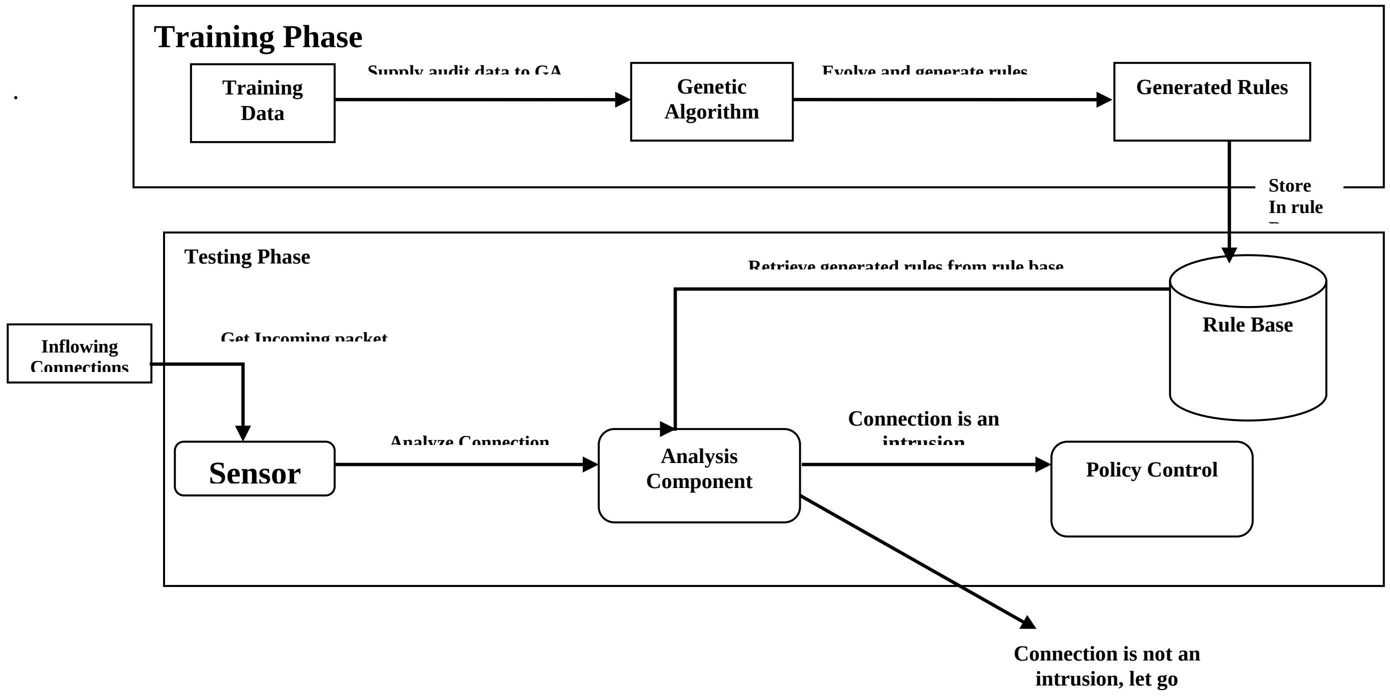
31.	192.168.1.30	21	192.168.0.40	43546	ftp	0hour, 0min, 8secs
32.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 0secs
33.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 3secs
34.	192.168.1.30	1876	192.168.0.20	23	telnet	0hour, 1min, 4secs
35.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 5secs
36.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 1secs
37.	192.168.1.30	1876	192.168.0.20	23	telnet	0hour, 1min, 6secs
38.	192.168.1.30	1884	192.168.0.20	23	telnet	0hour, 0min, 43secs
39.	192.168.0.20	20	192.168.1.30	1893	ftp-data	0hour, 0min, 25secs
40.	192.168.1.30	1892	192.168.0.20	21	ftp	0hour, 0min, 30secs
41.	192.168.1.30	1890	192.168.0.20	23	telnet	0hour, 0min, 34secs
42.	192.168.0.20	20	192.168.1.30	1895	ftp-data	0hour, 0min, 20secs
43.	192.168.1.30	21	192.168.0.40	43555	ftp	0hour, 0min, 18secs
44.	192.168.1.30	1022	192.168.0.20	514	rsh	0hour, 0min, 7secs
45.	192.168.1.30	1908	192.168.0.40	80	http	0hour, 0min, 2secs
46.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 7secs
47.	192.168.0.40	43550	192.168.1.30	20	ftp-data	0hour, 0min, 23secs
48.	192.168.1.30	1867	192.168.0.20	23	telnet	0hour, 0min, 23secs
49.	192.168.0.40	43555	192.168.1.30	21	ftp	0hour, 0min, 18secs
50.	192.168.1.30	20	192.168.0.40	43548	ftp-data	0hour, 0min, 26secs
51.	192.168.1.30	1754	192.168.0.20	23	telnet	0hour, 0min, 8secs
52.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 23secs
53.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 22secs
54.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 3secs
55.	192.168.1.30	1890	192.168.0.20	23	telnet	0hour, 0min, 55secs
56.	192.168.1.30	1906	192.168.0.20	23	telnet	0hour, 0min, 25secs
57.	192.168.0.20	1022	192.168.1.30	1021	rcp	0hour, 0min, 5secs
58.	192.168.1.30	1876	192.168.0.20	23	telnet	0hour, 0min, 5secs
59.	192.168.1.30	21	192.168.0.40	43546	ftp	0hour, 0min, 31secs
60.	192.168.1.30	20	192.168.0.40	43563	ftp-data	0hour, 0min, 9secs
61.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 23secs
62.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 22secs
63.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 5secs
64.	192.168.1.30	1890	192.168.0.20	23	telnet	0hour, 0min, 55secs
65.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 23secs
66.	192.168.1.30	1022	192.168.0.20	513	rlogin	0hour, 0min, 22secs
67.	192.168.1.30	1023	192.168.0.20	514	rsh	0hour, 0min, 18secs
68.	192.168.1.30	1890	192.168.0.20	23	telnet	0hour, 0min, 55secs
69.	192.168.1.30	1906	192.168.0.20	23	telnet	0hour, 0min, 25secs
70.	192.168.0.20	1022	192.168.1.30	1021	rcp	0hour, 0min, 5secs
71.	192.168.0.40	43546	192.168.1.30	21	ftp	0hour, 0min, 31secs

VOL. 3, NO. 8 Aug, 2012 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2012 CIS Journal. All rights reserved.

http://www.cisjournal.org

Table A3: Result of Rules and Attack names implemented in Linux

S/no	Duration (h,m,s)	Protocol	Source Port	Dest Port	Source IP	Dest IP	Attack	Fitness
1	-1,0,23	telnet	-1	23	192.168.1.30	192.168.0.20	guess	0.8063
2	-1,0,23	-1	-1	-1	192.168.1.30	192.-1.0.20	guess	0.8063
3	0,0,5	-1	-1	-1	192.168.1.30	192.168.0.20	port-scan	0.8063
4	0,0,5	-1	-1	-1	192.168.1.30	192.-1.0.20	port-scan	0.8063
5	-1,0,23	telnet	-1	23	192.-1.1.30	192.168.0.20	guess	0.8063
6	0,0,5	-1	-1	-1	192.168.1.30	192.168.0.20	port-scan	0.8063
7	-1,0,23	telnet	-1	23	192.168.1.30	192.168.0.20	guess	0.8063
8	0,0,5	-1	-1	-1	192.168.1.30	192.168.0.20	port-scan	0.8063
9	0,0,23	telnet	-1	-1	192.168.1.30	192.168.0.20	guess	0.8063
10	-1,0,23	telnet	-1	23	192.168.1.30	192.168.0.20	guess	0.8063
11	0,0,5	-1	-1	-1	192.168.1.30	192.-1.0.20	port-scan	0.8063
12	-1,0,23	telnet	-1	23	192.168.1.30	192.168.0.20	guess	0.8063
13	0,0,-1	-1	1023	1021	192.-1.1.30	-1.168.0.20	rcp	0.8031
14	-1,0,-1	-1	1023	-1	192.168.1.30	192.168.0.-1	rcp	0.8031
15	0,0,14	-1	-1	513	192.168.1.30	192.168.0.-1	rsh	0.8031
16	0,0,14	-1	-1	513	192.168.1.30	192.168.0.20	rsh	0.8031
17	0,0,14	-1	-1	513	-1.168.1.30	192.168.0.20	rsh	0.8031
18	0,0,14	-1	-1	513	192.168.1.30	192.168.0.-1	rsh	0.8031
19	-1,0,-1	-1	1023	-1	192.168.1.30	192.168.0.-1	rcp	0.8031
20	0,0,5	-1	-1	23	192.168.1.30	192.168.0.20	port-scan	0.8031
21	-1,0,-1	-1	1023	-1	192.168.1.30	192.168.-1.20	rcp	0.8031
22	0,0,14	-1	-1	513	192.168.1.30	192.168.0.-1	rsh	0.8031


Training Phase
Training Data
Supply audit data to GA
Genetic Algorithm
Evolve and generate rules
Generated Rules
Store In rule
Testing Phase
Retrieve generated rules from rule base
Rule Base
Inflowing Connections
Get Incoming packet
Sensor
Analyze Connection
Analysis Component
Connection is an intrusion
Policy Control
Connection is not an intrusion, let go