3

Evolving Intrusion Detection Systems

Ajith Abraham[1] and Crina Grosan[2]

[1] School of Computer Science and Engineering, Chung-Ang University, 221, Heukseok-Dong, Dongjak-Gu, Seoul, 156-756, Korea
ajith.abraham@ieee.org, http://ajith.softcomputing.net

[2] Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş Bolyai University, Kogalniceanu 1, Cluj-Napoca, 3400, Romania
cgrosan@cs.ubbcluj.ro, http://www.cs.ubbcluj.ro/ cgrosan

An intrusion Detection System (IDS) is a program that analyzes what happens or has happened during an execution and tries to find indications that the computer has been misused. An IDS does not eliminate the use of preventive mechanism but it works as the last defensive mechanism in securing the system. This Chapter evaluates the performances of two Genetic Programming techniques for IDS namely Linear Genetic Programming (LGP) and Multi-Expression Programming (MEP). Results are then compared with some machine learning techniques like Support Vector Machines (SVM) and Decision Trees (DT). Empirical results clearly show that GP techniques could play an important role in designing real time intrusion detection systems.

3.1 Introduction

Computer security is defined as the protection of computing systems against threats to confidentiality, integrity, and availability [28]. Confidentiality (or secrecy) means that information is disclosed only according to policy, integrity means that information is not destroyed or corrupted and that the system performs correctly, availability means that system services are available when they are needed. Computing system refers to computers, computer networks, and the information they handle. Security threats come from different sources such as natural forces (such as flood), accidents (such as fire), failure of services (such as power) and people known as intruders. There are two types of intruders: the external intruders who are unauthorized users of the machines they attack, and internal intruders, who have permission to access the system with some restrictions. The traditional prevention techniques such as user authentication, data encryption, avoiding programming errors and firewalls are

A. Abraham and C. Grosan: *Evolving Intrusion Detection Systems*, Studies in Computational Intelligence (SCI) **13**, 57–79 (2006)
www.springerlink.com

used as the first line of defense for computer security. If a password is weak and is compromised, user authentication cannot prevent unauthorized use, firewalls are vulnerable to errors in configuration and ambiguous or undefined security policies. They are generally unable to protect against malicious mobile code, insider attacks and unsecured modems. Programming errors cannot be avoided as the complexity of the system and application software is changing rapidly leaving behind some exploitable weaknesses. intrusion detection is therefore required as an additional wall for protecting systems. Intrusion detection is useful not only in detecting successful intrusions, but also provides important information for timely countermeasures.

An intrusion is defined [10] as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource. This includes a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable. An attacker can gain illegal access to a system by fooling an authorized user into providing information that can be used to break into a system. An attacker can deliver a piece of software to a user of a system which is actually a trojan horse containing malicious code that gives the attacker system access. Bugs in trusted programs can be exploited by an attacker to gain unauthorized access to a computer system. There are legitimate actions that one can perform that when taken to the extreme can lead to system failure. An attacker can gain access because of an error in the configuration of a system. In some cases it is possible to fool a system into giving access by misrepresenting oneself. An example is sending a TCP packet that has a forged source address that makes the packet appear to come from a trusted host. Intrusions are classified [29] as six types.

Attempted break-ins, which are detected by typical behavior profiles or violations of security constraints. Masquerade attacks, which are detected by atypical behavior profiles or violations of security constraints. Penetration of the security control system, which are detected by monitoring for specific patterns of activity. Leakage, which is detected by atypical use of system resources. Denial of service, which is detected by a typical use of system resources. Malicious use, which is detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

3.2 Intrusion Detection

Intrusion detection is classified into two types: misuse intrusion detection and anomaly intrusion detection. Misuse intrusion detection uses well-defined patterns of the attack that exploit weaknesses in system and application software to identify the intrusions. These patterns are encoded in advance and used to match against the user behavior to detect intrusion.

Anomaly intrusion detection uses the normal usage behavior patterns to identify the intrusion. The normal usage patterns are constructed from the statistical measures of the system features, for example, the CPU and I/O

activities by a particular user or program. The behavior of the user is observed and any deviation from the constructed normal behavior is detected as intrusion.

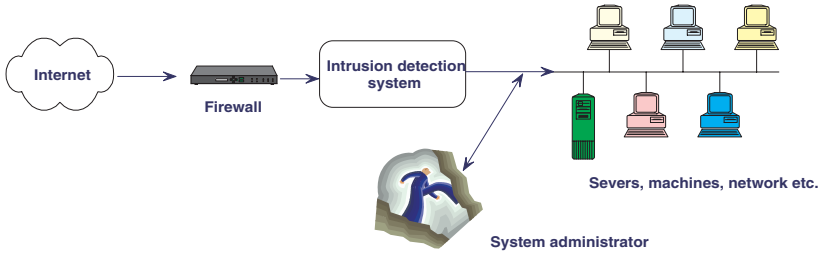

Fig. 3.1. Network protected by an IDS

Figure 3.1 illustrates a simple network, which is protected using IDS. We have two options to secure the system completely, either prevent the threats and vulnerabilities which come from flaws in the operating system as well as in the application programs or detect them and take some action to prevent them in future and also repair the damage. It is impossible in practice, and even if possible, extremely difficult and expensive, to write a completely secure system. Transition to such a system for use in the entire world would be an equally difficult task. Cryptographic methods can be compromised if the passwords and keys are stolen. No matter how secure a system is, it is vulnerable to insiders who abuse their privileges. There is an inverse relationship between the level of access control and efficiency. More access controls make a system less user-friendly and more likely of not being used. An Intrusion Detection system is a program (or set of programs) that analyzes what happens or has happened during an execution and tries to find indications that the computer has been misused. An Intrusion detection system does not eliminate the use of preventive mechanism but it works as the last defensive mechanism in securing the system.

Data mining approaches are a relatively new techniques for intrusion detection. There are a wide variety of data mining algorithms drawn from the fields of statistics, pattern recognition, machine learning, and databases. Previous research of data mining approaches for intrusion detection model identified several types of algorithms as useful techniques. Classification is one of the data mining algorithms, which have been investigated as a useful technique for intrusion detection models.

3.3 Related Research

James Anderson [2] first proposed that audit trails should be used to monitor threats. All the available system security procedures were focused on denying access to sensitive data from an unauthorized source. Dorothy Denning [7] first proposed the concept of intrusion detection as a solution to the problem of providing a sense of security in computer systems. The basic idea is that intrusion behavior involves abnormal usage of the system. The model is a rule-based pattern matching system. Some models of normal usage of the system could be constructed and verified against usage of the system and any significant deviation from the normal usage flagged as abnormal usage. This model served as an abstract model for further developments in this field and is known as generic intrusion detection model and is depicted in Figure 3.2 [15].

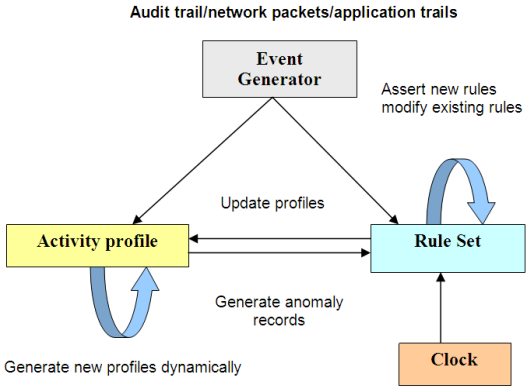

Fig. 3.2. A generic intrusion detection model

Statistical approaches compare the recent behavior of a user of a computer system with observed behavior and any significant deviation is considered as intrusion. This approach requires construction of a model for normal user behavior. IDES (Intrusion Detection Expert System) [17] first exploited the statistical approach for the detection of intruders. It uses the intrusion detection model proposed by Denning [7] and audit trails data as suggested in Anderson [2]. IDES maintains profiles, which is a description of a subject's normal behavior with respect to a set of intrusion detection measures. Profiles are updated periodically, thus allowing the system to learn new behavior as users alter their behavior. These profiles are used to compare the user behavior and informing significant deviation from them as the intrusion. IDES also uses the expert system concept to detect misuse intrusions. This system

has later developed as NIDES (Next-generation Intrusion Detection Expert System) [18]. The advantage of this approach is that it adaptively learns the behavior of users, which is thus potentially more sensitive than human experts. This system has several disadvantages. The system can be trained for certain behavior gradually making the abnormal behavior as normal, which makes intruders undetected. Determining the threshold above which an intrusion should be detected is a difficult task. Setting the threshold too low results in false positives (normal behavior detected as an intrusion) and setting too high results in false negatives (an intrusion undetected). Attacks, which occur by sequential dependencies, cannot be detected, as statistical analysis is insensitive to order of events.

Predictive pattern generation uses a rule base of user profiles defined as statistically weighted event sequences [30]. This method of intrusion detection attempts to predict future events based on events that have already occurred. This system develops sequential rules of the from E1 - E2 - E3 $\rightarrow$ (E4 = 94%; E5 = 6%) where the various E's are events derived from the security audit trail, and the percentage on the right hand of the rule represent the probability of occurrence of each of the consequent events given the occurrence of the antecedent sequence. This would mean that for the sequence of observed events E1 followed by E2 followed by E3, the probability of event E4 occurring is 94% and that of E5 is 6%. The rules are generated inductively with an information theoretic algorithm that measures the applicability of rules in terms of coverage and predictive power. An intrusion is detected if the observed sequence of events matches the left hand side of the rule but the following events significantly deviate from the right hand side of the rule. The main advantages of this approach include its ability to detect and respond quickly to anomalous behavior, easier to detect users who try to train the system during its learning period. The main problem with the system is its inability to detect some intrusions if that particular sequence of events have not been recognized and created into the rules.

State transition analysis approach construct the graphical representation of intrusion behavior as a series of state changes that lead from an initial secure state to a target compromised state. Using the audit trail as input, an analysis tool can be developed to compare the state changes produced by the user to state transition diagrams of known penetrations. State transition diagrams form the basis of a rule-based expert system for detecting penetrations, called the State Transition Analysis Tool (STAT) [23]. The STAT prototype is implemented in USTAT (Unix State Transition Analysis Tool) [11] on UNIX based system. The main advantage of the method is it detects the intrusions independent of audit trial record. The rules are produced from the effects of sequence of audit trails on system state whereas in rule based methods the sequence of audit trails are used. It is also able to detect cooperative attacks, variations to the known attacks and attacks spanned across multiple user sessions. Disadvantages of the system are it can only construct

patterns from sequence of events but not from more complex forms and some attacks cannot be detected, as they cannot be modeled with state transitions.

Keystroke monitoring technique utilizes a user's keystrokes to determine the intrusion attempt. The main approach is to pattern match the sequence of keystrokes to some predefined sequences to detect the intrusion. The main problems with this approach are lack of support from operating system to capture the keystroke sequences and also many ways of expressing the sequence of keystrokes for the same attack. Some shell programs like *bash*, *ksh* have the user definable aliases utility. These aliases make this technique difficult to detect the intrusion attempts unless some semantic analysis of the commands is used. Automated attacks by malicious executables cannot be detected by this technique as they only analyze the keystrokes.

IDES [17] used expert system methods for misuse intrusion detection and statistical methods for anomaly detection. IDES expert system component evaluates audit records as they are produced. The audit records are viewed as facts, which map to rules in the rule-base. Firing a rule increases the suspicion rating of the user corresponding to that record. Each user's suspicion rating starts at zero and is increased with each suspicious record. Once the suspicion rating surpasses a pre-defined threshold, an intrusion is detected. There are some disadvantages to expert system method. An Intrusion scenario that does not trigger a rule will not be detected by the rule-based approach. Maintaining and updating a complex rule-based system can be difficult. The rules in the expert system have to be formulated by a security professional which means the system strength is dependent on the ability of the security personnel.

Model-Based approach attempts to model intrusions at a higher level of abstraction than audit trail records. This allows administrators to generate their representation of the penetration abstractly, which shifts the burden of determining what audit records are part of a suspect sequence to the expert system. This technique differs from the rule-based expert system technique, which simply attempt to pattern match audit records to expert rules. Garvey and Lunt's [8] model-based approach consists of three parts: anticipator, planner and interpreter. The anticipator generates the next set of behaviors to be verified in the audit trail based on the current active models and passes these sets to the planner. The planner determines how the hypothesized behavior is reflected in the audit data and translates it into a system dependent audit trail match. The interpreter then searches for this data in the audit trail. The system collects the information this way until a threshold is reached, then it signals an intrusion attempt. The advantage of this model is it can predict the intruder's next move based on the intrusion model, which is used to take preventive measures, what to look for next and verify against the intrusion hypothesis. This also reduces the data to be processed as the planner and interpreter filter the data based on their knowledge what to look for, which leads to efficiency. There are some drawbacks to this system. The intrusion patterns must always occur in the behavior it is looking for otherwise it cannot detect them.

The Pattern Matching [14] approach encodes known intrusion signatures as patterns that are then matched against the audit data. Intrusion signatures are classified using structural interrelationships among the elements of the signatures. The patterned signatures are matched against the audit trails and any matched pattern can be detected as an intrusion. Intrusions can be understood and characterized in terms of the structure of events needed to detect them. A Model of pattern matching is implemented using colored petri nets in IDIOT [15]. Intrusion signature is represented with Petri nets, the start state and final state notion is used to define matching to detect the intrusion. This system has several advantages. The system can be clearly separated into different parts. This makes different solutions to be substituted for each component without changing the overall structure of the system. Pattern specifications are declarative, which means pattern representation of intrusion signatures can be specified by defining what needs to be matched than how it is matched. Declarative specification of intrusion patterns enables them to be exchanged across different operating systems with different audit trails. There are few problems in this approach. Constructing patterns from attack scenarios is a difficult problem and it needs human expertise. Attack scenarios that are known and constructed into patterns by the system can only be detected, which is the common problem of misuse detection.

3.4 Evolving IDS Using Genetic Programming (GP)

This section provides an introduction to the two GP techniques used namely Linear Genetic Programming (LGP) and Multi Expression Programming (MEP).

3.4.1 Linear Genetic Programming (LGP)

Linear genetic programming is a variant of the GP technique that acts on linear genomes [4]. Its main characteristics in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LISP), but the programs of an imperative language (like C/C++). An alternate approach is to evolve a computer program at the machine code level, using lower level representations for the individuals. This can tremendously hasten the evolution process as, no matter how an individual is initially represented, finally it always has to be represented as a piece of machine code, as fitness evaluation requires physical execution of the individuals. The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Since different instructions may have different sizes, here instructions are clubbed up together to form instruction blocks of 32 bits each. The instruction blocks hold one or more native machine code instructions, depending on the sizes of the instructions. A crossover point can occur only between instructions and is prohibited from

occurring within an instruction. However the mutation operation does not have any such restriction.

The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population space has been subdivided into multiple subpopulation or demes. Migration of individuals among the sub-populations causes evolution of the entire population. It helps to maintain diversity in the population, as migration is restricted among the demes. Moreover, the tendency towards a bad local minimum in one deme can be countered by other demes with better search directions. The various LGP search parameters are the mutation frequency, crossover frequency and the reproduction frequency: The crossover operator acts by exchanging sequences of instructions between two tournament winners. Steady state genetic programming approach was used to manage the memory more effectively.

3.4.2 Multi Expression Programming (MEP)

A GP chromosome generally encodes a single expression (computer program). By contrast, a Multi Expression Programming (MEP) chromosome encodes several expressions. The best of the encoded solution is chosen to represent the chromosome (by supplying the fitness of the individual). The MEP chromosome has some advantages over the single-expression chromosome especially when the complexity of the target expression is not known. This feature also acts as a provider of variable-length expressions. Other techniques (such as Grammatical Evolution (GE) [27] or Linear Genetic Programming (LGP) [4]) employ special genetic operators (which insert or remove chromosome parts) to achieve such a complex functionality. Multi Expression Programming (MEP) technique ([20], [21]) description and features are presented in what follows.

3.4.3 Solution Representation

MEP genes are (represented by) substrings of a variable length. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of lower values than the position of the function itself in the chromosome.

The proposed representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme, the first symbol of the chromosome must be a terminal symbol. In this way, only syntactically correct programs (MEP individuals) are obtained.

An example of chromosome using the sets F= {+, *} and T= {a, b, c, d} is given below:

1: a
2: b
3: + 1, 2
4: c
5: d
6: + 4, 5
7: * 3, 6

The maximum number of symbols in MEP chromosome is given by the formula:

Number_of_Symbols = (n+1) * (*Number_of_Genes* – 1) + 1,

where n is the number of arguments of the function with the greatest number of arguments.

The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments. The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.

The translation of a MEP chromosome into a computer program represents the phenotypic transcription of the MEP chromosomes. Phenotypic translation is obtained by parsing the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol.

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:

$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression: $E_3 = a + b$. Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression: $E_6 = c + d$. Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression: $E_7 = (a + b)$ * $(c + d)$. E_7 is the expression encoded by the whole chromosome.

There is neither practical nor theoretical evidence that one of these expressions is better than the others. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number

of genes). The chromosome described above encodes the following expressions:

$E_1 = a$,
$E_2 = b$,
$E_3 = a + b$,
$E_4 = c$,
$E_5 = d$,
$E_6 = c + d$,
$E_7 = (a + b)$ * $(c + d)$.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming and are stored in a conventional manner.

Due to its multi expression representation, each MEP chromosome may be viewed as a forest of trees rather than as a single tree, which is the case of Genetic Programming.

3.4.4 Fitness Assignment

As MEP chromosome encodes more than one problem solution, it is interesting to see how the fitness is assigned.

The chromosome fitness is usually defined as the fitness of the best expression encoded by that chromosome.

For instance, if we want to solve symbolic regression problems, the fitness of each sub-expression E_i may be computed using the formula:

$$f(E_i) = \sum_{k=1}^{n} |o_{k,i} - w_k|,$$

where $o_{k,i}$ is the result obtained by the expression E_i for the fitness case k and w_k is the targeted result for the fitness case k. In this case the fitness needs to be minimized.

The fitness of an individual is set to be equal to the lowest fitness of the expressions encoded in the chromosome:

When we have to deal with other problems, we compute the fitness of each sub-expression encoded in the MEP chromosome. Thus, the fitness of the entire individual is supplied by the fitness of the best expression encoded in that chromosome.

3.5 Machine Learning Techniques

For illustrating the capabilities of GP systems, two popular machine learning techniques were used namely Decision Trees (DT) and Support Vector Machines (SVM).

3.5.1 Decision Trees

Decision tree induction is one of the classification algorithms in data mining. The classification algorithm is inductively learned to construct a model from the pre-classified data set. Each data item is defined by values of the attributes. Classification may be viewed as mapping from a set of attributes to a particular class. The decision tree classifies the given data item using the values of its attributes. The decision tree is initially constructed from a set of pre-classified data. The main approach is to select the attributes, which best divides the data items into their classes. According to the values of these attributes the data items are partitioned. This process is recursively applied to each partitioned subset of the data items. The process terminates when all the data items in current subset belongs to the same class. A node of a decision tree specifies an attribute by which the data is to be partitioned. Each node has a number of edges, which are labeled according to a possible value of the attribute in the parent node. An edge connects either two nodes or a node and a leaf. Leaves are labeled with a decision value for categorization of the data.

Induction of the decision tree uses the training data, which is described in terms of the attributes. The main problem here is deciding the attribute, which will best partition the data into various classes. The ID3 algorithm [25] uses the information theoretic approach to solve this problem. Information theory uses the concept of entropy, which measures the impurity of a data items. The value of entropy is small when the class distribution is uneven, that is when all the data items belong to one class. The entropy value is higher when the class distribution is more even, that is when the data items have more classes. Information gain is a measure on the utility of each attribute in classifying the data items. It is measured using the entropy value. Information gain measures the decrease of the weighted average impurity (entropy) of the attributes compared with the impurity of the complete set of data items. Therefore, the attributes with the largest information gain are considered as the most useful for classifying the data items.

To classify an unknown object, one starts at the root of the decision tree and follows the branch indicated by the outcome of each test until a leaf node is reached. The name of the class at the leaf node is the resulting classification. Decision tree induction has been implemented with several algorithms. Some of them are ID3 [25] and later on it was extended into algorithms C4.5 [26] and C5.0. Another algorithm for decision trees is CART [5].

3.5.2 Support Vector Machines (SVMs)

Support Vector Machines [31] combine several techniques from statistics, machine learning and neural networks. SVM perform structural risk minimization. They create a classifier with minimized VC (Vapnik and Chervonenkis) dimension. If the VC Dimension is low, the expected probability of error is

low as well, which means good generalization. SVM has the common capability to separate the classes in the linear way. However, SVM also has another specialty that it is using a linear separating hyperplane to create a classifier, yet some problems can't be linearly separated in the original input space. Then SVM uses one of the most important ingredients called kernels, i.e., the concept of transforming linear algorithms into nonlinear ones via a map into feature spaces.

The possibility of using different kernels allows viewing learning methods like Radial Basis Function Neural Network (RBFNN) or multi-layer Artificial Neural Networks (ANN) as particular cases of SVM despite the fact that the optimized criteria are not the same [16]. While ANNs and RBFNN optimizes the mean squared error dependent on the distribution of all the data, SVM optimizes a geometrical criterion, which is the margin and is sensitive only to the extreme values and not to the distribution of the data into the feature space. The SVM approach transforms data into a feature space F that usually has a huge dimension. It is interesting to note that SVM generalization depends on the geometrical characteristics of the training data, not on the dimensions of the input space. Training a support vector machine (SVM) leads to a quadratic optimization problem with bound constraints and one linear equality constraint. Vapnik [31] shows how training a SVM for the pattern recognition problem leads to the following quadratic optimization problem:

$$\text{Minimize: } W(\alpha) = -\sum_{i-1}^{l} \alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{Subject to } \sum_{i=1}^{l} y_i \alpha_i, \quad \forall i : 0 \leq \alpha_i \leq C \quad ,$$

where l is the number of training examples $\alpha$is a vector of $l$variables and each component $\alpha_i$corresponds to a training example (x_i, y_i). SVM Torch software was used for simulating the SVM learning algorithm for IDS.

3.6 Experiment Setup and Results

The data for our experiments was prepared by the 1998 DARPA intrusion detection evaluation program by MIT Lincoln Labs [19]. The data set has 41 attributes for each connection record plus one class label as given in Table 3.1. The data set contains 24 attack types that could be classified into four main categories:

DoS: Denial of Service. Denial of Service (DoS) is a class of attack where an attacker makes a computing or memory resource too busy or too full to handle legitimate requests, thus denying legitimate users access to a machine.

Table 3.1. Variables for intrusion detection data set

Variable No.	Variable name	Variable type	Variable label
1	duration	continuous	A
2	protocol_type	discrete	B
3	service	discrete	C
4	flag	discrete	D
5	src_bytes	continuous	E
6	dst_bytes	continuous	F
7	land	discrete	G
8	wrong_fragment	continuous	H
9	urgent	continuous	I
10	hot	continuous	J
11	num_failed_logins	continuous	K
12	logged_in	discrete	L
13	num_compromised	continuous	M
14	root_shell	continuous	N
15	su_attempted	continuous	O
16	num_root	continuous	P
17	num_file_creations	continuous	Q
18	num_shells	continuous	R
19	num_access_files	continuous	S
20	num_outbound_cmds	continuous	T
21	is_host_login	discrete	U
22	is_guest_login	discrete	V
23	count	continuous	W
24	srv_count	continuous	X
25	serror_rate	continuous	Y
26	srv_serror_rate	continuous	X
27	rerror_rate	continuous	AA
28	srv_rerror_rate	continuous	AB
29	same_srv_rate	continuous	AC
30	diff_srv_rate	continuous	AD
31	srv_diff_host_rate	continuous	AE
32	dst_host_count	continuous	AF
33	dst_host_srv_count	continuous	AG
34	dst_host_same_srv_rate	continuous	AH
35	dst_host_diff_srv_rate	continuous	AI
36	dst_host_same_src_port_rate	continuous	AJ
37	dst_host_srv_diff_host_rate	continuous	AK
38	dst_host_serror_rate	continuous	AL
39	dst_host_srv_serror_rate	continuous	AM
40	dst_host_rerror_rate	continuous	AN
41	dst_host_srv_rerror_rate	continuous	AO

R2L: Unauthorized Access from a Remote Machine. A remote to user (R2L) attack is a class of attack where an attacker sends packets to a machine over a network, then exploits the machine's vulnerability to illegally gain local access as a user.

U2Su: Unauthorized Access to Local Super User (root). User to root (U2Su) exploits are a class of attacks where an attacker starts out with access to a normal user account on the system and is able to exploit vulnerability to gain root access to the system.

Probing: Surveillance and Other Probing. Probing is a class of attack where an attacker scans a network to gather information or find known vulnerabilities. An attacker with a map of machines and services that are available on a network can use the information to look for exploits.

Experiments presented in this chapter have two phases namely training and testing. In the training phase, MEP/LGP models were constructed using the training data to give maximum generalization accuracy on the unseen data. The test data is then passed through the saved trained model to detect intrusions in the testing phase. The 41 features are labeled as shown in Table 3.1 and the class label is named as *AP*.

This data set has five different classes namely *Normal, DoS, R2L, U2R* and *Probes.* The training and test comprises of 5,092 and 6,890 records respectively [13]. All the training data were scaled to (0-1). Using the data set, we performed a 5-class classification. The normal data belongs to class 1, probe belongs to class 2, denial of service belongs to class 3, user to super user belongs to class 4, remote to local belongs to class 5.

The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system [1]. The various parameter setting for LGP is depicted in Table 3.2

Table 3.2. Parameter settings for LGP

Parameter	Normal	Probe	DoS	U2Su	R2L
Population size	2048	2048	2048	2048	2048
Maximum no of tournaments	120000	120000	120000	120000	120000
Tournament size	8	8	8	8	8
Mutation frequency (%)	85	82	75	86	85
Crossover frequency (%)	75	70	65	75	70
Number of demes	10	10	10	10	10
Maximum program size	256	256	256	256	256

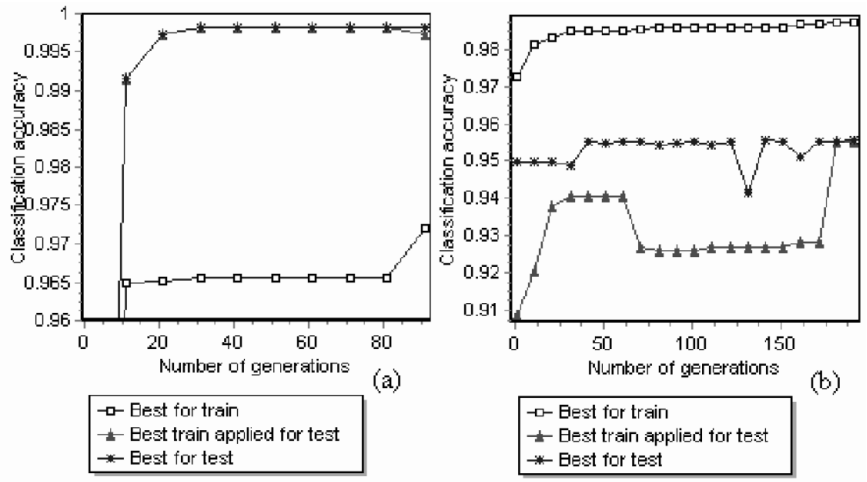

Fig. 3.3. Relation between accuracy and number of generations: (a) normal mode (b) probe attacks

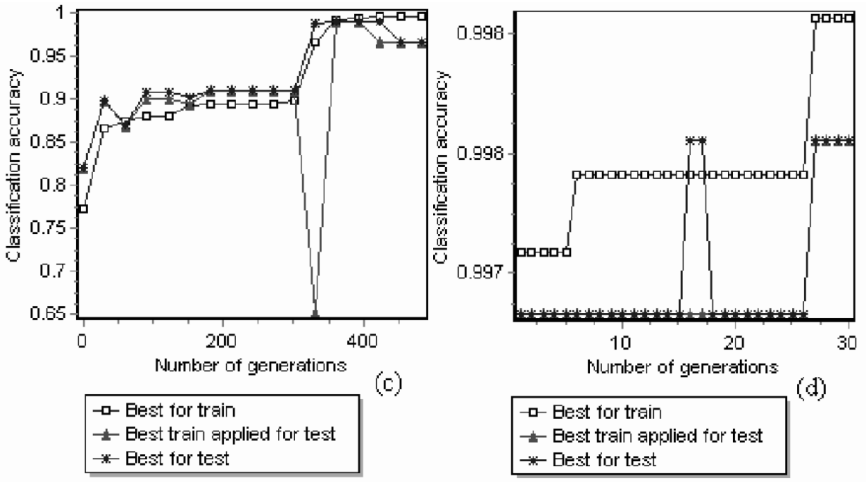

Fig. 3.4. Relation between accuracy and number of generations: (c) DoS attacks (d) U2R attacks

Our trial experiments with SVM revealed that the polynomial kernel option often performs well on most of the attack classes. We also constructed decision trees using the training data and then testing data was passed through the constructed classifier to classify the attacks [22].

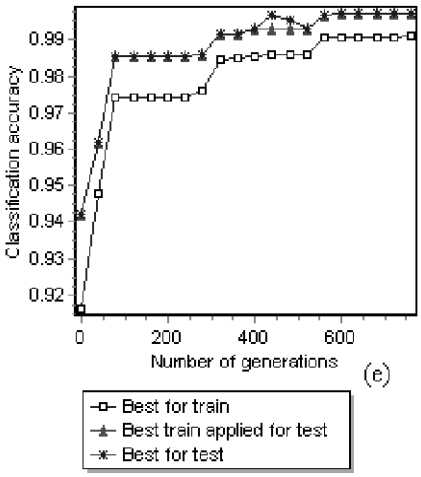

Fig. 3.5. Relation between accuracy and number of generations: (e) R2L attacks

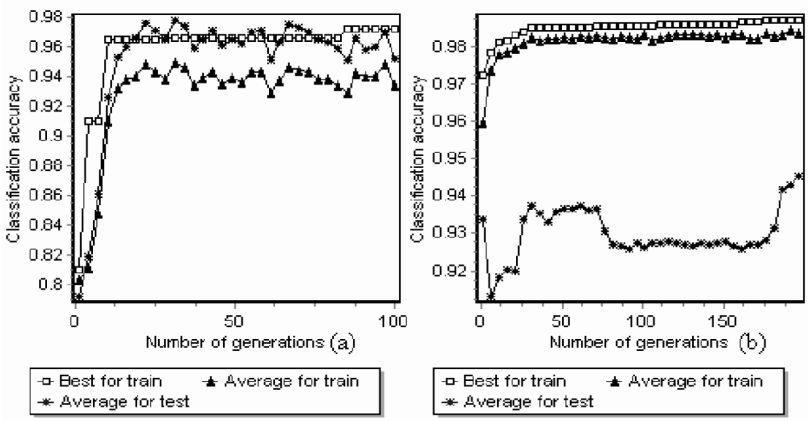

Fig. 3.6. Relationship between the best result obtained for training data set and the average of results obtained for training /test data: (a) normal mode (b) probe attacks

Parameters used by MEP are presented in Table 3.3 [9]. We made use of $+$, $-$, $*$, $/$, sin, cos, $sqrt$, ln, lg, $\log_2$, min, max, and abs as function sets.

Experiment results (for test data set) using the four techniques are depicted in Table 3.4. In Table 3.5 the variable combinations evolved by MEP are presented.

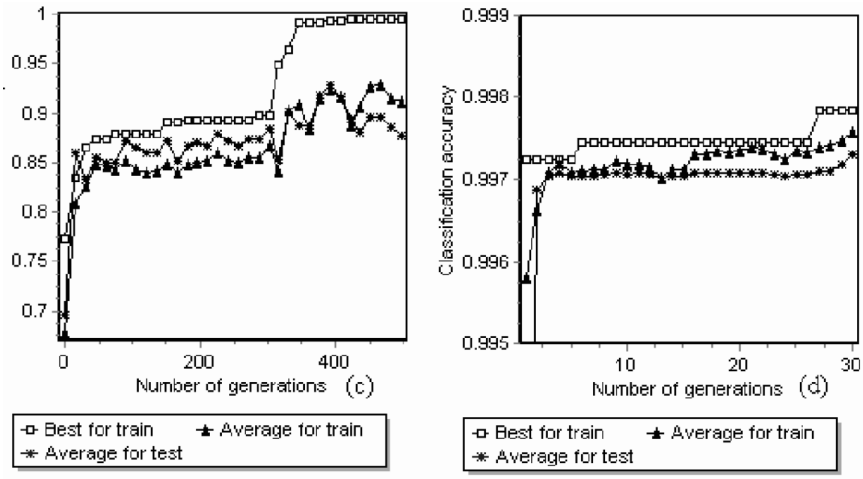

Fig. 3.7. Relationship between the best result obtained for training data set and the average of results obtained for training /test data: (c) DoS attacks (d) U2R attacks

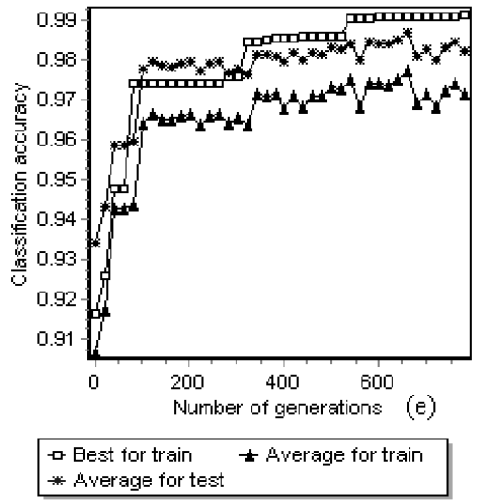

Fig. 3.8. Relationship between the best result obtained for training data set and the average of results obtained for training /test data: (e) R2L attacks

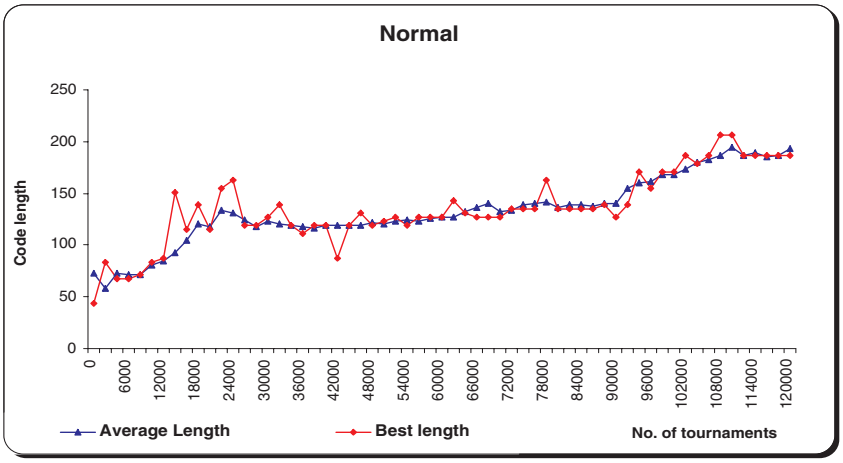

Fig. 3.9. Growth of program codes for normal mode

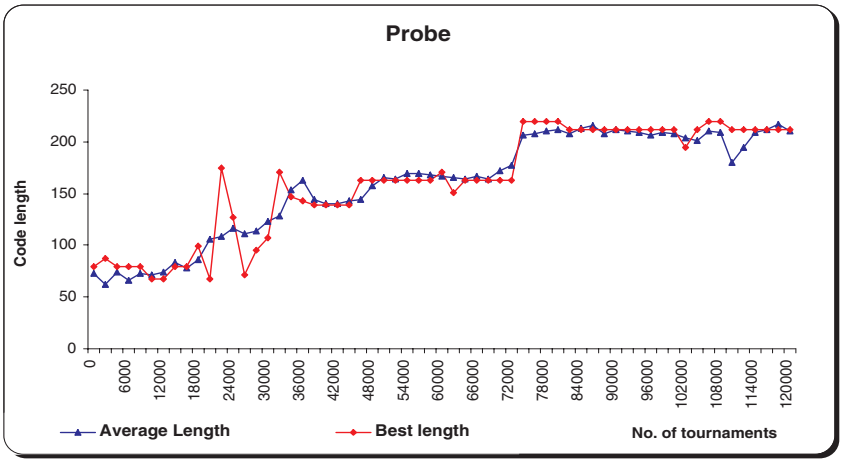

Fig. 3.10. Growth of program codes for probe attacks

As evident from Table 3.5, the presented GP techniques out performed some of the most popular machine learning techniques. MEP performed well for Classes 1, 4 and 5 while LGP gave the best test results Classes 2 and 3.

MEP performance is illustrated in Figures 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8. The classification accuracy for the best results obtained for training data, average of results obtained for the test data using the best training function and the best results obtained for the test data are depicted. Figures 3.3 (a) and (b) correspond to Normal, and Probe attacks, Figures 3.4 (c) and (d)

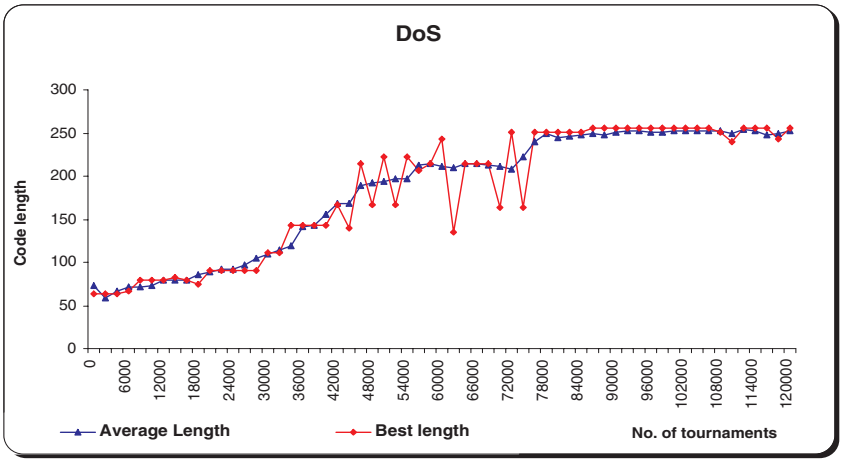

Fig. 3.11. Growth of program codes for DOS attacks

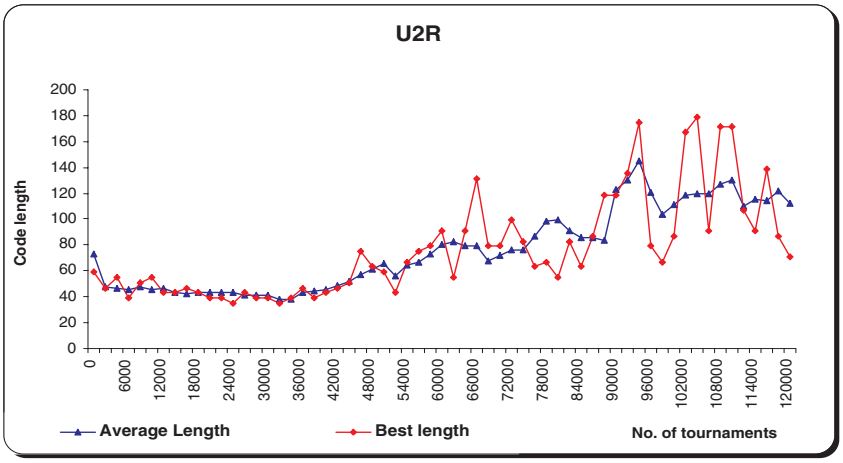

Fig. 3.12. Growth of program codes for U2R attacks

corresponds to DOS and U2R attacks and Figure 3.5 (e) corresponds to R2L attacks respectively.

In Figures 3.6- 3.8, the average of the classification accuracies for the results best results obtained for training data, results obtained for the test data and the obtained for the training data are depicted. Figures 3.6 (a) and (b) corresponds to Normal and Probe attacks, Figures 3.7 (c) and (d) corresponds DOS and U2R attacks and Figure 3.8 (e) corresponds to R2L attacks respectively.

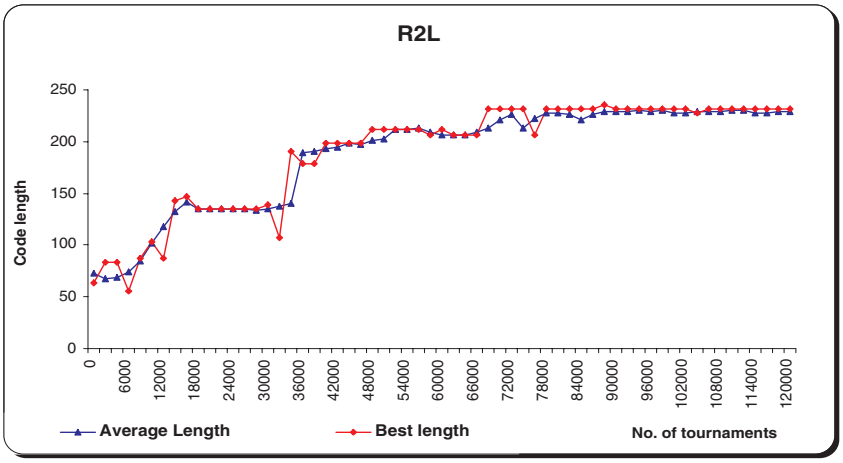

Fig. 3.13. Growth of program codes for R2L attacks

Table 3.3. Parameters used by MEP

Attack type	Parameter value				
	Pop. Size	Generations	Crossover (%)	No. of mutations	Chromosome length
Normal	100	30	0.9	3	30
Probe	200	200	0.9	4	40
DOS	250	800	0.8	5	40
U2R	100	20	0.9	3	30
R2L	100	800	0.9	4	40

Table 3.4. Functions evolved by MEP

Attack type	Evolved Function
Normal	$var12 * log_2(var10 + var3)$
Probe	$(fabs(var30 + var35)) < (var26 + var27)?(fabs(var30 + var35)) : (var26 + var27);$
DOS	$return(var38 - (Ln(var41 * var6) + sin(Lg(var30))) - (Lg(var30) - (var41 * var6))) > (0.3415 + var24 + var41 * var6)?(var38 - (Ln(var41 * var6) + sin(Lg(var30))) - (Lg(var30) - (var41 * var6))) : (0.3415 + var24 + var41 * var6) + var8$
U2R	$sin(var14) - var33$
R2L	$fabs((fabs(var8 > (var1 + (var6 > (Ln(var6))?var6 : (Ln(var6))) * var3)?var10 : (var1 + (var6 > (Ln(var6))?var6 : (Ln(var6))) * var3))) * (var12 + var6)) + var11$

Table 3.5. Performance comparison

Attack type	Classification accuracy on test data set (%)			
	MEP	DT	SVM	LGP
Normal	**99.82**	99.64	99.64	99.73
Probe	95.39	99.86	98.57	**99.89**
DOS	98.94	96.83	99.92	**99.95**
U2R	**99.75**	68.00	40.00	64.00
R2L	**99.75**	84.19	33.92	99.47

Figures 3.9 - 3.13 illustrate the growth in the program codes during the 120 tournaments during the development of LGP models. The best and average code length is depicted during the evolutionary learning.

In some classes the accuracy figures tend to be very small and may not be statistically significant, especially in view of the fact that the 5 classes of patterns differ in their sizes tremendously. For example only 27 data sets were available for training the U2R class. More definitive conclusions can only be made after analyzing more comprehensive sets of network traffic.

3.7 Conclusions

This chapter illustrated the importance of GP techniques for evolving intrusion detection systems. MEP outperforms LGP for three of the considered classes and LGP outperform MEP for two of the classes. MEP classification accuracy is grater than 95% for all considered classes and for three of them is greater than 99.75%. It is to be noted that for real time intrusion detection systems MEP and LGP would be the ideal candidates because of its simplified implementation.

Acknowledgements

This research was supported by the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the MIC (Ministry of Information and Communication), South Korea.

References

1. Abraham, A., Evolutionary Computation in Intelligent Web Management, Evolutionary Computing in Data Mining, Ghosh A. and Jain L.C. (Eds.), Studies in Fuzziness and Soft Computing, Springer Verlag Germany, Chapter 8, pp. 189-210, 2004. 70

2. J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980. 60
3. Barbara D., Couto J., Jajodia S. and Wu N., ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection. SIGMOD Record, 30(4), pp. 15-24, 2001.
4. Brameier M. and Banzhaf W, Explicit control of diversity and effective variation distance in Linear Genetic Programming. In Proceedings of the fourth European Conference on Genetic Programming, Springer-Verlag Berlin, 2001. 63, 64
5. Brieman L., Friedman J., Olshen R., and Stone C., Classification of Regression Trees. Wadsworth Inc., 1984. 67
6. Cohen W., Learning Trees and Rules with Set-Valued Features, American Association for Artificial Intelligence (AAAI), 1996.
7. Denning D., An Intrusion-Detection Model, IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp. 222-232, 1987. 60
8. T. D. Garvey and T. F. Lunt. Model based intrusion detection, In Proceedings of the 14th National Computer Security Conference, pages 372-385, October 1991. 62
9. Grosan C., Abraham A. and Han S.Y., MEPIDS: Multi-Expression Programming for Intrusion Detection System, International Work-conference on the Interplay between Natural and Artificial Computation, (IWINAC'05), Spain, Lecture Notes in Computer Science, Springer Verlag, Germany, pp. 163-172, 2005. 72
10. R. Heady, G. Luger, A. Maccabe, and M. Servilla, The Architecture of a Network level Intrusion Detection System. Technical report, Department of Computer Science, University of New Mexico, August 1990. 58
11. K. Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX, Master Thesis, University of California, Santa Barbara, November 1992. 61
12. T. Joachims. Making Large-Scale SVM Learning Practical. LS8-Report, University of Dortmund, LS VIII-Report, 1998.
13. KDD Cup 1999 Intrusion detection data set: http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz 70
14. S. Kumar and E. H. Spafford. An Application of Pattern Matching in Intrusion Detection. Technical Report CSD-TR-94-013, Purdue University, 1994. 63
15. S. Kumar. Classification and Detection of Computer Intrusions, PhD Thesis, Department of Computer Science, Purdue University, August 1995. 60, 63
16. Lee W. and Stolfo S. and Mok K., A Data Mining Framework for Building Intrusion Detection Models. In Proceedings of the IEEE Symposium on Security and Privacy, 1999.
17. T.F. Lunt, A. Tamaru, F. Gilham et al, A Real Time Intrusion Detection Expert System (IDES), Final Technical Report, Project 6784, SRI International 1990 60, 62
18. T. Lunt. Detecting intruders in computer systems. In Proceedings of the 1993 Conference on Auditing and Computer Technology, 1993. 61
19. MIT Lincoln Laboratory. http://www.ll.mit.edu/IST/ideval/ 68
20. Oltean M. and Grosan C., A Comparison of Several Linear GP Techniques, *Complex Systems*, Vol. 14, No. 4, pp. 285-313, 2004. 64
21. Oltean M. and Grosan C., Evolving Evolutionary Algorithms using Multi Expression Programming. *Proceedings of The 7th European Conference on Artificial Life*, Dortmund, Germany, pp. 651-658, 2003. 64
22. Peddabachigari S., Abraham A., Thomas J., Intrusion Detection Systems Using Decision Trees and Support Vector Machines, International Journal of Applied Science and Computations, Vol.11, No.3, pp.118-134, 2004. . 71

23. P. A. Porras. STAT: A State Transition Analysis Tool for Intrusion Detection. Master's Thesis, Computer Science Dept., University of California, Santa Barbara, 1992. 61
24. Provost, F. and T. Fawcett. Robust Classification for Imprecise Environments, Machine Learning 42, 203-231, 2001.
25. J. R. Quinlan. Induction of Decision Trees. Machine Learning, 1:81-106, 1986. 67
26. J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993. 67
27. C. Ryan C. J.J. Collins and M. O'Neill. Gramatical Evolution: Evolving programs for an arbitrary language, In Proceedings of the first European Workshop on Genetic Programming, Springer-Verlag, Berlin, 1998. 64
28. Summers R.C., Secure Computing: Threats and Safeguards. McGraw Hill, New York, 1997. 57
29. A. Sundaram. An Introduction to Intrusion Detection. ACM Cross Roads, Vol. 2, No. 4, April 1996. 58
30. H. S. Teng, K. Chen and S. C. Lu. Security Audit Trail Analysis Using Inductively Generated Predictive Rules. In Proceedings of the 11th National Conference on Artificial Intelligence Applications, pages 24-29, IEEE, IEEE Service Center, Piscataway, NJ, March 1990. 61
31. Vapnik V.N., The Nature of Statistical Learning Theory. Springer, 1995. 67, 68