Available online at www.sciencedirect.com
ScienceDirect
Journal of Network and Computer Applications 30 (2007) 81–98
Journal of NETWORK and COMPUTER APPLICATIONS
www.elsevier.com/locate/jnca

D-SCIDS: Distributed soft computing intrusion detection system

Ajith Abraham[a,*], Ravi Jain[b], Johnson Thomas[c], Sang Yong Han[a]

[a]*School of Computer Science and Engineering, Chung-Ang University, Korea*
[b]*University of South Australia, Adelaide, Australia*
[c]*Computer Science Department, Oklahoma State University, OK 74106, USA*

Received 28 June 2005; accepted 28 June 2005

Abstract

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. A Distributed IDS (DIDS) consists of several IDS over a large network (s), all of which communicate with each other, or with a central server that facilitates advanced network monitoring. In a distributed environment, DIDS are implemented using co-operative intelligent agents distributed across the network(s). This paper evaluates three fuzzy rule-based classifiers to detect intrusions in a network. Results are then compared with other machine learning techniques like decision trees, support vector machines and linear genetic programming. Further, we modeled Distributed Soft Computing-based IDS (D-SCIDS) as a combination of different classifiers to model lightweight and more accurate (heavy weight) IDS. Empirical results clearly show that soft computing approach could play a major role for intrusion detection.

*Corresponding author.
E-mail addresses: ajith.abraham@ieee.org, , hansy@cau.ac.kr (A. Abraham), ravi.jain@unisa.edu.au (R. Jain), jpt@okstate.edu (J. Thomas).

1084-8045/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2005.06.001

1. Introduction

An intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource. Intrusion detection is classified into two types: misuse intrusion detection and anomaly intrusion detection (Mukkamala et al., 2005). 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. The behavior of the user is observed and any deviation from the constructed normal behavior is detected as an intrusion (Denning, 1987; Summers, 1997). In Distributed Intrusion Detection System (DIDS) conventional intrusion detection system are embedded inside intelligent agents and are deployed over a large network. In a distributed environment, IDS agents communicate with each other, or with a central server. By having these co-operative agents distributed across a network, incident analysts, network operations, and security personnel are able to get a broader view of what is occurring on their network as a whole. Distributed monitoring allows early detection of planned and coordinated attacks, thereby allowing network administrators to take preventive measures. DIDS also helps to control the spreading of worms, improves network monitoring and incident analysis, attack tracing and so on. It also helps to detect new threats from unauthorized users, back-door attackers and hackers to the network across multiple locations, which are geographically separated (Abraham and Thomas, 2005). In a DIDS it is important to ensure that the individual IDS are lightweight and accurate.

Data mining approaches for intrusion detection were first implemented in mining audit data for automated models for intrusion detection (Barbara et al., 2001; Cohen, 1996; Lee et al., 1999). Several data mining algorithms are applied to audit data to compute models that accurately capture the actual behavior of intrusions as well as normal activities. Audit data analysis and mining combine the association rules and classification algorithm to discover attacks in audit data. Soft Computing (SC) is an innovative approach to construct computationally intelligent systems consisting of artificial neural networks, fuzzy inference systems, approximate reasoning and derivative free optimization methods such as evolutionary computation, etc. (Zadeh, 1998). This paper introduces three fuzzy rule-based classifiers (Abraham et al., 2004) and compares its performance with Linear Genetic Programming (LGP) (Abraham, 2004), Support Vector Machines (SVM) (Vapnik, 1995) and Decision Trees (DT) (Brieman et al., 1984; Peddabachigari et al., 2004). Further, we modeled Soft Computing (SC)-based IDS (SCIDS) (Abraham et al., 2004) as a combination of different classifiers to model lightweight and more accurate (heavy weight) IDS. The rest of the paper is organized as follows. Section 2 provides a brief overview of the research on distributed intrusion detection systems. Soft computing for intrusion detection is introduced in Section 3 followed by the importance of attribute reduction (important feature selection) in Section 4.

Experimental results are also presented in Section 4 followed by conclusions in Section 5.

2. Distributed intrusion detection system (DIDS)

A number of IDSs have been proposed for a networked or distributed environment. Early systems included ASAX (Mouinji et al., 1995), DIDS (Snapp et al., 1999) and NSTAT (Kemmerer, 1997). These systems require the audit data collected from different places to be sent to a central location for analysis. NetSTAT (Vigna and Kemmerer, 1999) is another example of such a system. In NetSTAT attack scenarios are modeled as hypergraphs and places are probed for network activities. Although NetSTAT also collects information in a distributed manner, it analyses them in a central place. The scalability of such systems is limited due to their centralized nature. To improve scalability later systems such as EMERALD (Porras and Neumann, 1997), GriDS (Staniford–Chen et al., 1996) and AAFID (Spafford and Zamboni, 2000), deployed instruction detection systems at different locations and organized them into a hierarchy such that low-level IDSs send designated information to higher level IDSs. EMERALD uses both misuse detection and statistical anomaly detection techniques by having a recursive framework, which allows generic components to be deployed in a distributed manner. To detect intruders, GriDS aggregates computer and network information into activity graphs which reveal the causal structure of network activity. AAFID consists of agents, filters transceivers and monitors organized in a tree structure. The hierarchical approaches employed by theses schemes scale better than the previous centralized approach. However, the main problem with such an approach is that if two or more IDSs that are far apart in the hierarchy detect a common intruder, the two detection cannot be correlated until the messages from the different IDSs reach a common high-level IDS. This will require the messages to traverse multiple IDSs resulting in communication overheads. The Common Intrusion Detection Framework (CIDF) (Staniford-Chen et al., 1998) goes one step further as it aims to enable different intrusion detection and response components to interoperate and share information and resources in a distributed environment. The intrusion detection inter-component adaptive negotiation protocol helps cooperating CIDF components to reach an agreement on each other's needs and capabilities (Feiertag et al., 2000). MADAM ID uses CIDF to automatically get audit data, build models, and distribute signatures for novel attacks so that the gap between the discovery and detection of new attacks can be reduced (Lee et al., 2000). The coordinated and response system (CARDS) (Ning et al., 2002) aims at detecting distributed attacks that cannot be detected using data collected from any single location. CARDS decomposes global representations of distributed attacks into smaller units that correspond to the distributed events indicating the attacks. It then executes and coordinates the decomposed smaller units in the places where the corresponding units are observed. The message transmission between component IDSs is not determined by a centralized or hierarchical scheme, Instead, in CARDS, one component IDS sends a message to another only when the message is required by the later IDS to detect certain attacks. The communication cost is therefore reduced. Although JiNao (Jou et al., 2000) has been proposed as a distributed IDS for detecting intrusions network routing protocols, no specific mechanisms have been provided for doing so. JiNao focuses on the Open Shortest Path First (OSPF) protocol.

Software agents have been proposed as a technology for intrusion detection applications. Rationale for considering agents in an IDS ranges from increased adaptability for new threats to reduced communication costs. Since agents are independently executing entities, there is the potential that new detection capabilities can be added without completely halting, rebuilding, and restarting the IDS. Other potential advantages are described in Jansen et al. (1999) and Kruegel and Toth (2001). Kruegel and Toth (2001) also identify downside tradeoffs including increased design and operational complexity. The Autonomous Agents for Intrusion Detection (AAFID) framework (Spafford and Zamboni, 2000) employs autonomous agents for data collection and analysis. AAFID utilizes agents hosted on network nodes, filters to extract pertinent data, transceivers to oversee agent operation, and monitors to receive reports from transceivers. These entities are organized into a hierarchical architecture with centralized control. Cooperating Security Managers (CSMs) (White et al., 1996) enable individual distributed intrusion detection packages to cooperate in performing network intrusion detection without relying on centralized control. Each individual CSM detects malicious activity on the local host. When suspicious activity is detected, each CSM will report any noteworthy activity to the CSM on the host from which the connection originated. The local CSM will not notify all networked systems, but rather only the system immediately before it in the connection chain. Other agent-based hierarchical architectures include the Intelligent Agents for Intrusion Detection project (Helmer et al., 1998) with a centralized data warehouse at the root, data cleaners at the leaves, and classifier agents in between. Bernardes and dos Santos Moreira (2000) have proposed a hybrid framework with partially distributed decision making under the control of a centralized agent manager. Agents are deployed to observe behavior of the system and users. Agents communicate via messages to advise peers when an action is considered suspect. When an agent considers an activity to be suspect, an agent with a higher level of specialization for the suspected intrusion is activated. Agents then report their findings to a centralized manager. The main drawbacks with these systems, is that the use of one or more centralized repositories leave at least some portion of the network exposed to malicious attacks including tampering and denial of service attacks. Even if an autonomous mobile decision-making agent was to detect a problem, interlocking mechanisms would be necessary to preclude any accidental or malicious removal, delay, or spoofing the agent. The Tethered Agent and Collective Hive (TACH) architecture includes a centralized Hive to keep track of agents and collected data and an Agent Registry (AR) to track fingerprints of agents (Lu, 2000). An Aglet-based framework for TACH incorporates mobile agents for virus detection and misuse detection (Kapoor, 2000). Limitations of TACH include the use of a centralized entity for agent control and a period communication protocol between agents with time-out detection used to detect status changes in the agents. If the centralized entity is disabled then the entire TACH system has been compromised. The DIDS (Mukherjee et al., 1994) used a combination of host and LAN monitors to observe system and network activity. A centralized director obtained information from the monitors to detect intrusions. The CIDF nomenclature mentioned above includes reconnaissance agents for data gathering, analysis agents, and decision-response agents (Staniford-Chen et al., 1998). The Computer Immunology Project at the University of New Mexico (Forrest et al., 1997) explored designs of IDSs based on ideas gleaned by examining animal immune systems. Small, individual agents would roam a distributed system, identify intrusions, and resolve the intrusions. One portion of the project developed a sense of self for security-related computer programs by observing the normal sets of system calls executed by the programs. This sense of self can be used to detect intrusions by discovering when a program executes an unusual set of system calls. The JAM Project at Columbia University (Stolfo et al., 1997) uses intelligent, distributed Java agents and data mining to learn models of fraud and intrusive behavior that can be shared between organizations. Helmar et al. propose lightweight agents for intrusion detection (Helmer et al., 2003). Their multi-agent system includes agents that travel between monitored systems in a network of distributed systems, obtain information from data cleaning agents, classify and correlate information, and report the information to a user interface and database via mediators. Agent systems with lightweight agent support allow runtime addition of new capabilities to agents. DeMara and Rocke (2004) propose an IDS based on mobile agents for detecting malicious activity by people with legitimate access to resources and services. These include attacks such as spoofing, termination, sidetracking, alteration of internal data, and selective deception. Their system employs techniques such as encapsulation, redundancy, scrambling, and mandatory obsolescence.

DIDS are simply a superset of the conventional IDS implemented in a distributed environment. Due to the distributed nature the implementation poses several challenges. IDS could be embedded inside agents and placed in the network to be monitored. The individual IDS may be configured to detect a single attack, or they may detect several types of attacks. Each network component may host one or many IDS. Since there will be a large number of flag generators (detection of an attack, event, etc.), these must be abstracted, analyzed, and condensed by a suitable architecture before arriving at a final conclusion. Typically there would be a centralized analyzing and control facility. The most popular architecture is of the master–slave type which may be suitable for small networks. In a hierarchical architecture analysis and control are done at different layers because of the geographical distribution or due to the size of the network. Attacks/event detection information is passed to analyzer/controller nodes that aggregate information from multiple IDS agents. It is to be noted that the event information, which is detected by the IDS agents will follow a bottom-up approach for analysis and the various command and control flow will follow a top-down approach. The physical location of IDS agents will be fixed since they monitor fixed network segments. In the case of hierarchical architecture, the analyzer/controller nodes may exist at many locations

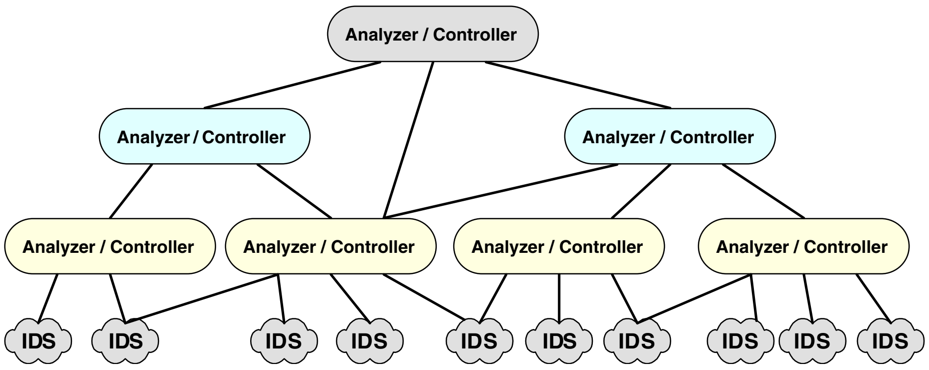

Fig. 1. Hierarchical architecture with free communication between layers.

in the network since they receive their input and give their output via network connections. Depending on the network environment the communication between the different layers could be implemented as depicted in Fig. 1 (Abraham and Thomas, 2005).

In the hierarchical architecture, the Central Analyzer and Controller (CAC) is the heart and soul of the DIDS. The CAC usually consists of a database and Web server, which allows interactive querying by the network administrators for attack information/analysis and initiate precautionary measures. CAC also performs attack aggregation, building statistics, identify attack patterns and perform rudimentary incident analysis. The co-operative intelligent agent network is one of the most important components of the DIDS. Ideally these agents will be located on separate network segments, and very often geographically separated. Communication among the agents is done utilizing TCP/IP sockets.

Agent modules running on the host machines are capable of data analysis and to formulate adequate response actions and are very often implemented as read only and fragile. In the event of tampering or modification the agent reports to the server agent and automatically ends its life. Agents residing in the individual analyzer/controllers consist of modules responsible for agent regeneration, dispatch, updating and maintaining intrusion signatures and so on. These agents control the individual IDS agents for monitoring the network, manage all the communication and life cycle of the IDS agents and also updates the IDS agents with detection algorithms, response and trace mechanisms.

3. Soft computing

Soft computing was first proposed by Zadeh (1998), to construct new generation computationally intelligent hybrid systems consisting of neural networks, fuzzy inference system, approximate reasoning and derivative free optimization techniques. It is well known that intelligent systems, which can provide human like expertise such as domain knowledge, uncertain reasoning, and adaptation to a noisy and time-varying environment, are important in tackling practical computing problems. In contrast with conventional Artificial Intelligence (AI) techniques which only deal with precision, certainty and rigor the guiding principle of hybrid systems is to exploit the tolerance for imprecision, uncertainty, low solution cost, robustness, partial truth to achieve tractability, and better rapport with reality.

3.1. Fuzzy rule-based systems

Fuzzy logic has proved to be a powerful tool for decision making to handle and manipulate imprecise and noisy data. The notion central to fuzzy systems is that truth values (in fuzzy logic) or membership values (in fuzzy sets) are indicated by a value in the range [0.0, 1.0], with 0.0 representing absolute falseness and 1.0 representing absolute truth. A fuzzy system is characterized by a set of linguistic statements based on expert knowledge. The expert knowledge is usually in the form of *if-then* rules.

Definition 1. Let X be some set of objects, with elements noted as x. Thus, $X = \{x\}$.

Definition 2. A fuzzy set A in X is characterized by a membership function which are easily implemented by fuzzy conditional statements. In the case of fuzzy statement if the antecedent is true to some degree of membership then the consequent is also true to that same degree.

A simple rule structure: *If* antecedent *then* consequent.

A simple rule: *If* variable$_1$ is low and variable$_2$ is high *then* output is benign *else* output is malignant.

In a fuzzy classification system, a case or an object can be classified by applying a set of fuzzy rules based on the linguistic values of its attributes. Every rule has a weight, which is a number between 0 and 1 and this is applied to the number given by the antecedent. It involves 2 distinct parts. First the antecedent is evaluated, which involves fuzzifying the input and applying any necessary fuzzy operators and second applying that result to the consequent known as inference. To build a fuzzy classification system, the most difficult task is to find a set of fuzzy rules pertaining to the specific classification problem. We explored three fuzzy rule generation methods for intrusion detection systems. Let us assume that we have a n-dimensional c-class pattern classification problem whose pattern space is an n-dimensional unit cube [0, 1]^n. We also assume that m patterns $x_p = (x_{pl}, \ldots, x_{pn})$, $p = 1, 2, \ldots, m$, are given for generating fuzzy *if-then* rules where $x_p \in [0, 1]$ for $p = 1, 2, \ldots, m$.

3.1.1. Rule generation based on the histogram of attribute values (FR_1)

In this method, use of the histogram itself is an antecedent membership function. Each attribute is partitioned into 20 membership functions $f_h(\cdot)$, $h = 1, 2, \ldots, 20$. The smoothed histogram $m_i^k(x_i)$of class k patterns for the ith attribute is calculated using the 20 membership functions $f_h(\cdot)$ as follows:

$$m_i^k(x_i) = \frac{1}{m^k} \sum_{x_p \in \text{class } k} f_h(x_{pi})$$

$$\text{for} \beta_{h-1} \leqslant x_i \leqslant \beta_h, \ h = 1, 2, ..., 20, \tag{1}$$

where m_k is the number of class k patterns, $[\beta_{h-1,}\beta_h]$ is the hth crisp interval corresponding to the 0.5-level set of the membership function $f_h(\cdot)$:

$$\beta_1 = 0, \ \beta_{20} = 1, \tag{2}$$

$$\beta_h = \frac{1}{20-1}\left(h - \frac{1}{2}\right) \text{for } h = 1, 2, ..., 19. \tag{3}$$

The smoothed histogram in (1) is normalized so that its maximum value is 1. A single fuzzy *if-then* rule is generated for each class. The fuzzy if-then rule for the kth class can be written as

$$\text{If } x_1 \text{ is } A_1^k \text{and} \ldots \text{and } x_n \text{ then class } k, \tag{4}$$

where A_i^k is an antecedent fuzzy set for the ith attribute. The membership function of A_i^k is specified as

$$A_i^k(x_i) = \exp\left(-\frac{(x_i - \mu_i^k)^2}{2(\sigma_i^k)^2}\right), \tag{5}$$

where μ_i^k is the mean of the ith attribute values x_{pi} of class k patterns, and σ_i^k is the standard deviation. Fuzzy *if-then* rules for the two-dimensional two-class pattern classification problem are written as follows:

$$\text{If } x_3 \text{ is } A_3^1 \text{ and } x_4 \text{ is } A_4^1 \text{ then class 2}, \tag{6}$$

$$\text{If } x_3 \text{ is } A_3^2 \text{ and } x_4 \text{ is } A_4^2 \text{ then class 3}. \tag{7}$$

Membership function of each antecedent fuzzy set is specified by the mean and the standard deviation of attribute values. For a new pattern $x_p = (x_{p3}, x_{p4})$, the winner rule is determined as follows:

$$A_3^*(x_{p3}).A_2^*(x_{p4}) = \max\{A_1^k(x_{p3}).A_2^k(x_{p4}) | k = 1, 2\}. \tag{8}$$

3.1.2. *Rule generation based on partition of overlapping areas (FR$_2$)*

Fig. 2 demonstrates a simple fuzzy partition, where the two-dimensional pattern space is partitioned into 25 fuzzy subspaces by five fuzzy sets for each attribute (S: small, MS: medium small, M: medium, ML: medium large, L: large). A single fuzzy *if-then* rule is generated for each fuzzy subspace. One disadvantage of this approach is that the number of possible fuzzy *if-then* rules exponentially increases with the dimensionality of the pattern space. Because the specification of each membership function does not depend on any information about training patterns, this approach uses fuzzy if-then rules with certainty grades. The local information about training patterns in the

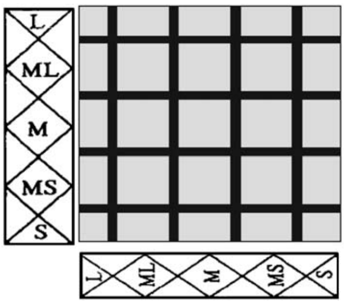

Fig. 2. An example of fuzzy partition.

corresponding fuzzy subspace is used for determining the consequent class and the grade of certainty. In this approach, fuzzy if-then rules of the following type are used:

$$\text{If } x_1 \text{ is } A_{j1} \text{ and} \ldots \text{and } x_n \text{ then class } C_j \\ \text{with CF} = \text{CF}_j,\ j = 1, 2, \ldots, N, \tag{9}$$

where j indexes the number of rules, N is the total number of rules, A_{ji} the antecedent fuzzy set of the ith rule for the ith attribute, C_j; is the consequent class, and CF_j is the grade of certainty. The consequent class and the grade of certainty of each rule are determined by the following simple heuristic procedure:

Step 1*:* Calculate the compatibility of each training pattern $x_p = (x_{p1}, x_{p2}, \ldots, x_{pn})$ with the jth fuzzy *if-then* rule by the following product operation:

$$\pi_j(x_p) = A_{j1}(x_{p1}) \times \cdots \times A_{jn}(x_{pn}), p = 1, 2, \ldots, m. \tag{10}$$

Step 2*:* For each class, calculate the sum of the compatibility grades of the training patterns with the jth fuzzy *if-then* rule R_j:

$$\beta_{\text{class } k}(R_j) = \sum_{x_p \in \text{class } k}^{n} \pi(x_p), k = 1, 2, ..., c, \tag{11}$$

where $\beta_{\text{class } k}(R_j)$ the sum of the compatibility grades of the training patterns in class k with the jth fuzzy if-then rule R_j.

Step 3*:* Find class A_j^* that has the maximum value$\beta_{\text{class } k}(R_j)$:

$$\beta_{\text{class } k_j^*} = \text{Max}\{\beta_{\text{class } 1}(R_j), \ldots, \beta_{\text{class } c}(R_j)\}. \tag{12}$$

If two or more classes take the maximum value or no training pattern compatible with the jth fuzzy *if-then* rule (i.e., if $\beta_{\text{class } k}(R_j) = 0$ for $k = 1, 2, \ldots, c$), the consequent class C_i can not be determined uniquely. In this case, let C_i be ϕ.

Step 4*:* If the consequent class C_i is 0, let the grade of certainty CF_j be $CF_j = 0$. Otherwise the grade of certainty CF_j *is* determined as follows:

$$CF_j = \frac{(\beta_{\text{class } k_j^*}(R_j) - \bar{\beta})}{\sum_{k=1}^{c} \beta_{\text{class } k}(R_j)}, \tag{13}$$

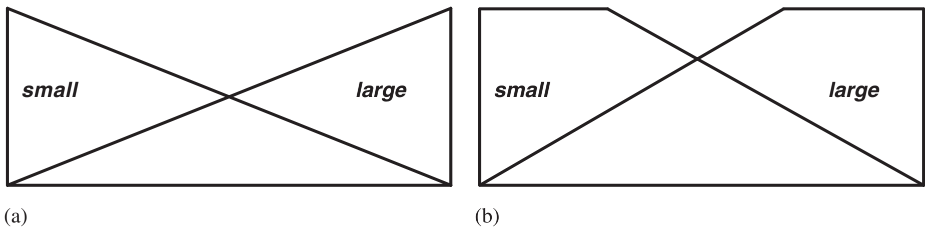

Fig. 3. Fuzzy partition of each attribute: (a) simple fuzzy grid approach; (b) modified fuzzy grid approach.

where

$$\bar{\beta} = \sum_{\substack{k=1 \\ k \neq k_j^*}} \frac{\beta_{\text{class } k}(R_j)}{(c-1)},$$

The above approach could be modified by partitioning only the overlapping areas as illustrated in Fig. 3.

This approach generates fuzzy *if-then* rules in the same manner as the simple fuzzy grid approach except for the specification of each membership function. Because this approach utilizes the information about training patterns for specifying each membership function as mentioned in Section 3.1, the performance of generated fuzzy *if-then* rules is good even when we do not use the certainty grade of each rule in the classification phase. In this approach, the effect of introducing the certainty grade to each rule is not so important when compared to conventional grid partitioning.

3.2. Neural learning of fuzzy rules (FR_3)

In a fused neuro-fuzzy architecture, neural network learning algorithms are used to determine the parameters of fuzzy inference system (membership functions and number of rules). An Evolving Fuzzy Neural Network implements a Mamdani-type FIS and all nodes are created during learning. Each input variable is represented here by a group of spatially arranged neurons to represent a fuzzy quantization of this variable. New neurons can evolve in this layer if, for a given input vector, the corresponding variable value does not belong to any of the existing MF to a degree greater than a membership threshold. Technical details of the learning algorithm are given in Kasabov (1998).

4. Experimental setup and results

Complex relationships exist between features, which are difficult for humans to discover. The IDS must therefore reduce the amount of data to be processed. This is very important if real-time detection is desired. The easiest way to do this is by doing an intelligent input feature selection. Certain features may contain false correlations, which hinder the process of detecting intrusions. Further, some features may be redundant since the information they add is contained in other features. Extra features can increase computation time, and can impact the accuracy of IDS. Feature selection improves classification by searching for the subset of features, which best classifies the training data (Chebrolu et al., 2005).

Feature selection is done based on the contribution the input variables made to the construction of the decision tree. Feature importance is determined by the role of each input variable either as a main splitter or as a surrogate. Surrogate splitters are defined as back-up rules that closely mimic the action of primary splitting rules. Suppose that, in a given model, the algorithm splits data according to variable 'protocol_type' and if a value for 'protocol_type' is not available, the algorithm might substitute 'flag' as a good surrogate. Variable importance, for a particular variable is the sum across all nodes in the tree of the improvement scores that the predictor has when it acts as a primary or surrogate (but not competitor) splitter. Example, for node i, if the predictor appears as the primary splitter then its contribution towards importance could be given as $i_{\mathrm{importance}}$. But if the variable appears as the nth surrogate instead of the primary variable, then the importance becomes $i_{\mathrm{importance}} = (p^n)^* i_{\mathrm{improvement}}$ in which p is the 'surrogate improvement weight' which is a user controlled parameter set between (0–1) (Shah et al., 2004).

The data for our experiments was prepared by the 1998 DARPA intrusion detection evaluation program by MIT Lincoln Labs MIT. The LAN was operated in a real environment, but was subjected to multiple attacks. For each TCP/IP connection, 41 various quantitative and qualitative features were extracted. The data set has 41 attributes for each connection record plus one class label as given in Table 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.

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.

Our experiments have three phases namely input feature reduction, training phase and testing phase. In the data reduction phase, important variables for real-time intrusion detection are selected by feature selection.

Table 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

In the training phase, the different soft computing 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 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 5092 and 6890 records, respectively (KDD Cup, 1999). Using all 41 variables could result in a big IDS model, which would result in substantial overhead for online detection. All the training data were scaled to (0–1). The decision tree approach described before resulted in to reducing the number of variables to 12 significant variables or features. The list of reduced variables is shown in Table 2.

Using the original and reduced data sets, 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. All the IDS models are trained and tested with the same set of data.

We examined the performance of all three fuzzy rule-based approaches (FR_1, FR_2 and FR_3) mentioned in Section 3. When an attack is correctly classified the grade of certainty is increased and when an attack is misclassified the grade of certainty is decreased. A learning procedure is used to determine the grade of certainty. Triangular membership functions were used for all the fuzzy rule based classifiers. For FR_1 and FR_2 two triangular membership functions were assigned and 2 and 2^{12} rules were learned respectively for the reduced data set. For FR_3, 4 triangular membership functions were used for each input variable. A sensitivity threshold $Sthr = 0.95$ and error threshold $Errthr = 0.05$ was used for all the classes and 89 rule nodes were developed during the one pass learning. For comparison purposes various other empirical results were adapted from Mukkamala et al. (2005), Peddabachigari et al. (2004), Shah et al. (2004), Chebrolu et al. (2005) and Abraham (2004).

The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population size was set at 120,000 and a tournament size of 8 is used for all the 5 classes. Crossover and mutation probability is set at 65–75% and 75–86%, respectively for the different classes (Abraham, 2004). Our trial experiments with SVM revealed that the polynomial kernel option often performs well on most of the datasets. We also constructed decision trees using the training data and then testing data was passed through the constructed classifier to classify the attacks (Mukkamala et al., 2005).

A number of observations and conclusions are drawn from the results illustrated in Tables 3 and 4. Using 41 attributes, the FR_2 method gave 100% accuracy for all the 5 classes, showing the importance of fuzzy inference systems. For the full data set, LGP outperformed decision trees and support vector machines in terms of detection accuracies (except for U2R class).

Using 12 attributes most of the classifiers performed very well except the fuzzy classifiers (FR_1, FR_2). For detecting U2R attacks FR_2 gave the best accuracy. However, due to the tremendous reduction in the number of attributes (about 70%

Table 2
Reduced variable set

C, E, F, L, W, X, Y, AB, AE, AF, AG, AI

Table 3
Performance comparison using full data set

Attack type	Classification accuracy on test data set (%)					
	FR_1	FR_2	FR_3	DT	SVM	LGP
Normal	40.44	**100.00**	98.26	99.64	99.64	99.73
Probe	53.06	**100.00**	99.21	99.86	98.57	99.89
DOS	60.99	**100.00**	98.18	96.83	99.92	99.95
U2R	66.75	**100.00**	61.58	68.00	40.00	64.00
R2L	61.10	**100.00**	95.46	84.19	33.92	99.47

Table 4
Performance comparison using reduced data set

Attack type	Classification accuracy on test data set (%)					
	FR_1	FR_2	FR_3	DT	SVM	LGP
Normal	74.82	79.68	99.56	**100.00**	99.75	99.97
Probe	45.36	89.84	99.88	97.71	98.20	**99.93**
DOS	60.99	60.99	98.99	85.34	98.89	**99.96**
U2R	94.11	**99.64**	65.00	64.00	59.00	68.26
R2L	91.83	91.83	97.26	95.56	56.00	**99.98**

less), we are able to design a computational efficient (lightweight) IDS. Since a particular classifier could not provide accurate results for all the classes, we propose to use a combination of different classifiers to detect different attacks. The D-SCIDS architecture using 41 attributes (heavy weight) and 12 attributes (lightweight) are depicted in Fig. 4. The proposed heavy weight model could detect with 100% accuracy while the lightweight model could detect all the attacks with high accuracy (lowest accuracy being 94.11% for U2R class using FR_1). It is to be noted that FR_1 classifier is preferred for the lightweight D-SCIDS (due to fewer number of rules) even though it has slightly lower accuracy when compared to FR_2 (Table 5).

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.

5. Conclusions

Effective intrusion detection and management systems are critical components of cyber infrastructure as they are in the forefront of the battle against cyber-terrorism. In this paper, we presented a framework for Distributed Intrusion Detection Systems (DIDS) using several soft computing paradigms. We also demonstrated the

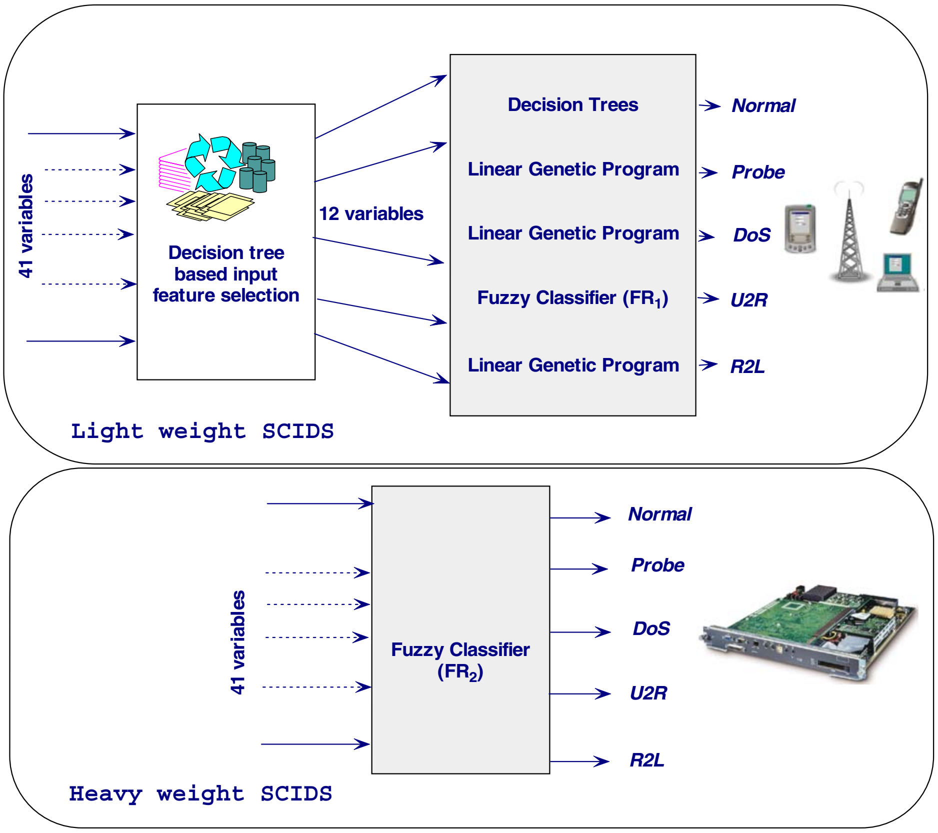

Fig. 4. Light/heavy-weight SCIDS architecture.

Table 5
Performance of the light weight DSCIDS

Attack type	Classification accuracy on test data (%)
Normal	100.00
Probe	99.93
DOS	99.96
U2R	94.11
R2L	99.98

importance of feature reduction to model lightweight intrusion detection systems. Finally, we propose a hybrid architecture involving ensemble and base classifiers for intrusion detection.

For real time intrusion detection systems, LGP would be the ideal candidate as it can be manipulated at the machine code level. Overall, the fuzzy classifier (FR_2) gave

100% accuracy for all attack types using all the 41 attributes. The proposed hybrid combination of classifiers requires only 12 input variables. While the lightweight SCIDS would be useful for MANET/distributed systems, the heavy weight SCIDS would be ideal for conventional static networks, wireless base stations etc. More data mining techniques are to be investigated for attribute reduction and enhance the performance of other soft computing paradigms.

With the increasing incidents of cyber attacks, building an effective intrusion detection models with good accuracy and real-time performance are essential. This field is developing continuously. More data mining techniques should be investigated and their efficiency should be evaluated as intrusion detection models.

Acknowledgments

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

References

Abraham A. Evolutionary computation in intelligent web management, evolutionary computing in data mining. In: Ghosh A, Lakhmi J, editors. Studies in fuzziness and soft computing. Germany: Springer; 2004. p. 189–210 [Chapter 8].

Abraham A, Thomas J. Distributed intrusion detection systems: a computational intelligence approach. In: Abbass HA, Essam D, editors. Applications of information systems to homeland security and defense. USA: Idea Group Inc. Publishers; 2005. p. 105–35 [Chapter 5].

Abraham A, Jain R, Sanyal S, Han SY. SCIDS: a soft computing intrusion detection system. Sixth international workshop on distributed computing (IWDC 2004). Lecture Notes in Computer Science, vol. 3326. Germany: Springer; 2004. p. 252–7. ISBN: 3-540-24076-4.

Barbara D, Couto J, Jajodia S, Wu N. ADAM: a testbed for exploring the use of data mining in intrusion detection. SIGMOD Rec 2001;30(4):15–24.

Bernardes MC, dos Santos Moreira E. Implementation of an intrusion detection system based on mobile agents. In: Proceedings of the international symposium on software engineering for parallel and distributed systems, 2000. p. 158–64.

Brieman L, Friedman J, Olshen R, Stone C. Classification of regression trees. Wadsworth Inc.; 1984.

Chebrolu S, Abraham A, Thomas J. Feature deduction and ensemble design of intrusion detection systems, computers and security, vol. 24/4. Amsterdam: Elsevier; 2005. p. 295–307.

Cohen W. Learning trees and rules with set-valued features. American Association for Artificial Intelligence (AAAI); 1996.

DeMara RF, Rocke AJ. Mitigation of network tampering using dynamic dispatch of mobile agents. Comput Security 2004;23:31–42.

Denning D. An intrusion-detection model. IEEE Trans Software Eng 1987;SE-13(2):222–32.

Feiertag R, Rho S, Benzingher L, Wu S, Redmond T, Zhang C, Levitt K, Peticolas D, Heckman M, Staniford S, McAlerney J. Intrusion detection inter-component adaptive negotiation. Comput Networks 2000;34:605–21.

Forrest S, Hofmeyr SA, Somayaji A. Computer immunology. CACM 1997;40(10):88–96.

Helmer G, Wong J, Honavar V, Miller L. Intelligent agents for intrusion detection. Available from http://citeseer.nj.nec.com/helmer98intelligent.html, 1998.

Helmer G, Wong JSK, Honavar V, Miller L, Wang Y. Lightweight agents for intrusion detection. J Systems Software 2003;67:109–22.

Jansen W, Mell P, Karygiannis T, Marks D. Applying mobile agents to intrusion detection and response. National Institute of Standards and Technology, Computer Security Division, 1999. Available from: http://csrc.nist.gov/publications/nistir/ir6416.pdf.

Jou YF, Gong F, Sargor C, Wu X, Wu SF, Chang HC, Wang F. Design and implementation of a scalable intrusion detection system for the protection of network infrastructure. DARPA information survivability conference and exposition, 2000.

Kapoor B. Remote misuse detection system using mobile agents and relational database query techniques. Master's thesis, University of Central Florida, 2000.

Kasabov N. Evolving fuzzy neural networks—algorithms, applications and biological motivation. In: Yamakawa T, Matsumoto G, editors. Methodologies for the conception, design and application of soft computing. Singapore: World Scientific; 1998. p. 271–4.

KDD Cup, 1999. Intrusion detection data set: 〈http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz〉

Kemmerer RA. NSTAT: a model-based real-time network intrusion detection system. Technical Report TRCS97-18, Reliable Software Group, Department of Computer Science, University of California at Santa Barbara, 1997.

Kruegel C, Toth T. Applying mobile agent technology to intrusion detection. Proceedings of the ICSE workshop on software engineering and mobility, 2001. Available from: http://citeseer.nj.nec.com/kr01applying.html.

Lee W, Stolfo S, Mok K. A Data Mining Framework for Building Intrusion Detection Models. In: Proceedings of the IEEE symposium on security and privacy, 1999.

Lee W, Nimbalker RA, Yee KK, Patil SB, Desai PH, Tran PP, Stolfo SJ. A data mining and CIDF based approach for detecting novel and distributed intrusions, Proceeings of the third international workshop on recent advances in intrusion detection, 2000.

Lu J. Mobile agent protocols for distributed detection of network intrusions. Master's thesis, University of Central Florida, 2000.

MIT Lincoln Laboratory. 〈http://www.ll.mit.edu/IST/ideval/〉

Mouinji A, Charlier BL, Zampunieris D, Habra N. Distributed audit trail analysis. Proceedings of the ISOC 95 symposium on network and distributed system security, 1995. p. 102–12.

Mukherjee B, Todd Heberlein L, Levitt KN. Network intrusion detection. IEEE Network 1994;8(3):26–41.

Mukkamala S, Sung A, Abraham A. Intrusion detection using ensemble of soft computing and hard computing paradigms. J Network Comput Appl 2005;28(2):167–82.

Ning P, Jajodia S, Wang XS. Design and implementation of a decentralized prototype system for detecting distributed attacks. Comput Commun 2002;25:1374–91.

Peddabachigari S, Abraham A, Thomas J. Intrusion detection systems using decision trees and support vector machines. Int J Appl Sci Comput USA 2004;11(3):118–34.

Porras PA, Neumann PG. EMERALD: event monitoring enabling response to anomalous live disturbances. Proceedings of the 20th national information security conference, NIST, 1997.

Shah K, Dave N, Chavan S, Mukherjee S, Abraham A, Sanyal S. Adaptive neuro-fuzzy intrusion detection system. IEEE international conference on information technology: coding and computing (ITCC'04), vol. 1, USA. Silver Spring, MD: IEEE Computer Society; 2004. p. 70–4.

Snapp SR, Bretano J, Diaz GV, Goan TL, Heberlain LT, Ho C, Levitt KN, Mukherjee B, Smaha SE, Grance T, Teal DM, Mansur D. DIDS (Distributed Intrusion Detection System)—motivation architecture and an early prototype. In: Proceedings of the 14th national computer security conference, Washington, DC, October, 1999. p. 167–76.

Spafford EH, Zamboni D. Intrusion detection using autonomous agents. Comput Networks 2000;34:547–70.

Staniford-Chen S, Cheung S, Crawford R, Dilger M, Frank J, Hoagland J, Levitt K, Wee C, Yipi R, Erkle DZ. GriDS—a large scale intrusion detection system for large networks. Proceedings of the 19th national information security conference 1996;1:361–70.

Staniford-Chen S, Tung SB, Schnackenberg D. The common intrusion detection framework (CIDF). Proceedings of the information survivability workshop, Orlando FL, October 1998.

Stolfo SJ, Prodromidis AL, Tselepis S, Lee W, Fan D, Chan PK. JAM: Java agents for meta-learning over distributed databases. In: Proceedings of the third international conference on knowledge discovery and data mining, Newport Beach, CA, USA, 1997. p. 74–81.

Summers RC. Secure computing: threats and safeguards. New York: McGraw Hill; 1997.

Vapnik VN. The nature of statistical learning theory. Berlin: Springer; 1995.

Vigna G, Kemmerer RA. NetSTAT: a network-based intrusion detection system. J Comput Security 1999;7(1):37–71.

White GB, Fisch EA, Wpooch U. Cooperating security managers: a peer-based intrusion detection system. IEEE Networks 1996.

Zadeh LA. Roles of soft computing and fuzzy logic in the conception, design and deployment of Information/Intelligent Systems. In: Kaynak O, Zadeh LA, Turksen B, Rudas IJ, editors. Computational intelligence: soft computing and fuzzy-neuro integration with applications, 1998. p. 1–9.