Distributed and Scalable Intrusion Detection System Based on Agents and Intelligent Techniques

Aly M. El-Semary* and Mostafa Gadal-Haqq M. Mostafa**

Abstract—The Internet explosion and the increase in crucial web applications such as e-banking and e-commerce, make essential the need for network security tools. One of such tools is an Intrusion detection system which can be classified based on detection approachs as being signature-based or anomaly-based. Even though intrusion detection systems are well defined, their cooperation with each other to detect attacks needs to be addressed. Consequently, a new architecture that allows them to cooperate in detecting attacks is proposed. The architecture uses Software Agents to provide scalability and distributability. It works in two modes: learning and detection. During learning mode, it generates a profile for each individual system using a fuzzy data mining algorithm. During detection mode, each system uses the FuzzyJess to match network traffic against its profile. The architecture was tested against a standard data set produced by MIT's Lincoln Laboratory and the primary results show its efficiency and capability to detect attacks. Finally, two new methods, the memory-window and memoryless-window, were developed for extracting useful parameters from raw packets. The parameters are used as detection metrics.

Keywords—Data-Mining, Fuzzy Logic, IDS, Intelligent Techniques, Network Security, Software Agents

1. INTRODUCTION

Intrusion Detection Systems (IDSs) are software systems that monitor computers or networks to detect attacks. After they were introduced by Anderson in 1980 [1] and formalized by Denning in 1987 [2], intrusion detection systems have become an active area of research. They are evaluated according to their ability to minimize false negatives and false positives. False negatives occur when an IDS fails to detect attacks. On the other hand, false positives occur when benign activities are classified as attacks. IDSs are classified according to how attacks are detected into two main approaches: signature-based and anomaly-based detections. Signature-based intrusion detection systems (also known as misuse or pattern matching) employ general pattern matching models based on attack signatures, such as rules, state-modeling and string matching. NIDES [3] and Snort [4] use rules to define attack signatures. USTAT [5] uses state

※ This work is supported by the *Deanship* of Scientific *Research, Taibah University*, KSA.
Manuscript received May 10, 2010; first revision August 27, 2010; accepted September 8, 2010.
Corresponding Author: Aly M. El-Semary
* Dept. of Systems and Computer Engineering, Faculty of Engineering, Al-Azhar University, Cairo, Egypt (alyelsemary@azhar.edu.eg, alyelsemary@ieee.org, aelsemary@taibahu.edu.sa, aly2semary@yahoo.com)
** Dept. of Computer Science, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt (mgmostafa@asu.edu.eg)

transition diagrams to detect access control violations; IDIOT [6] uses colored Petri nets to represent attacks. Signature-based IDSs are vulnerable to novel attacks for which signatures have not yet been created. Therefore, they have elevated false negative rates. On the other hand, anomaly-based intrusion detection systems involve identifying activities that deviate from what is considered normal system use; attacks are viewed as deviations from normal activities. Anomaly-based systems employ various techniques including artificial intelligence [3], statistical analysis [2], machine learning [7] and data mining [8, 9]. Recently, fuzzy logic together with data mining IDSs [10-16] have been successfully used to identify anomalies. Anomaly detection systems are capable of detecting attacks for which well-defined patterns do not exist (such as new attacks or variations of existing attacks). However, defining and maintaining "normal" profiles are not easy tasks.

Intrusion Detection Systems can be further classified as host-based or network-based. Host-based intrusion detection involves detecting malicious activity within a single system. A host-based intrusion detection system uses log information, system activity, process accounting information (e.g., processor time, memory, disk usage), and file integrity to determine whether or not a host is the target of an attack. A host-based system may be stand-alone or a part of a distributed intrusion detection system. Denning [2] was the first to propose a real-time, general-purpose expert system for detecting attacks; the system modeled normal system behavior using audit records, and monitored the system audit records to detect abnormal activities. Lee [9] developed a model that uses data mining of audit data records to create normal system profiles. A network-based intrusion detection system monitors network traffic to detect malicious activities, such as denial-of-service attacks, port scans, pings of death, or attempts to break into a system. For example, a large number of TCP connection requests to a very large number of different ports could indicate a "port scan" reconnaissance probe. Researchers have developed various network-based intrusion detection models. Roesch [4] developed Snort, a network misuse detection system employing a rule base. Qin [17] used data mining to profile normal network behavior. Bridges and Vaughn [18], Dickerson et al. [10, 11], Luo and Bridges [16] used fuzzy logic and data mining to model normal network behavior, and Elsemary et al [13-15] use fuzzy association rules to model both network behavior and attack signature but their model is neither scalable nor distributed. Last, but not least, Ming-Yang Su [19] uses the frequency episode rules implemented by finite state machines to design a real-time network-based intrusion prevention system for Probe/Exploit intrusion.

In this paper, we propose not only a new architecture for intrusion detection systems but also two new methods for analyzing useful parameters to be used as detectiom metrics. The architecture is based on agents and intelligent techniques and it is scalable and distributed. The rest of this paper is organized as follows: Section 2 presents the general research methodolgy. Section 3 introduces the proposed system architecture and its details; Section 4 presents data analysis methods that are used to extract useful parameters for detection metrics. Section 5 analyzes the simulation results and finally Section 6 concludes the paper and future work.

2. General Research Methodology

Fig. 1 describes the general research methodology. It continually collects raw data from the underlying network and then extracts useful parameters that are used as detection metrics. Next,

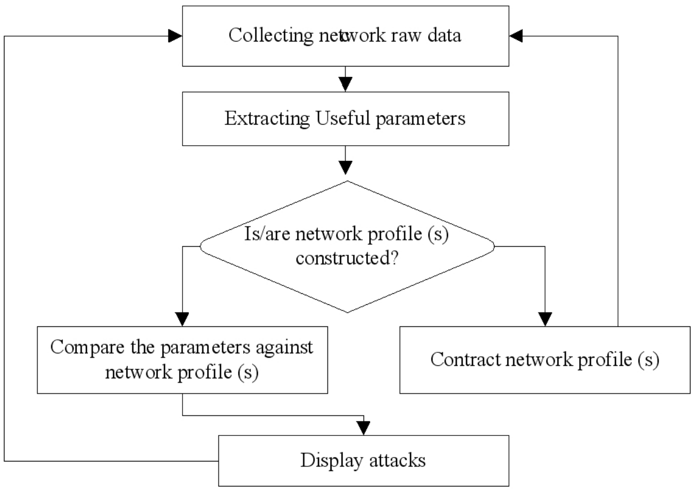

Collecting network raw data
Extracting Useful parameters
Is/are network profile (s) constructed?
Compare the parameters against network profile (s)
Contract network profile (s)
Display attacks

Fig. 1. General research methodology flowchart

if network profiles (sets of rules) are not contracted yet, they should be built first. Otherwise, the parameters are compared against the network profiles and if there is an attack, it should be announced. The network profiles are contracted offline from training data while the detection is online.

3. Proposed System Architecture

The proposed novel intrusion detection system, according to our knowledge, embodies a modular design that involves seven stages: 1) Data capturing, 2) packets queuing, 3) data analysis, 4) data mining, 5) network profiling, 6) attacks detection, and 7) attacks fusion. These stages provide functional separation and enable developers to incorporate new elements into the system (Fig. 2). In addition, this architecture enables the system to be either on one machine or distributed all over a private network. It works in two modes of operation: learning mode and detection mode. In the learning mode, the system inspects network traffic and uses a fuzzy data mining algorithm to produce a set of fuzzy rules which represent the network behavior or profile. In the detection mode, the system matches the current network behavior against the network profiles produced during the learning mode. In other words, the system uses the network profile to decide whether or not an intrusion has occurred.

3.1 Data Capturing

In this stage, capture agents are responsible for capturing raw packets from the private network that needs to be protected. It passes the raw packets to the packet queuing stage which in turn holds them in specific queues. This stage comprises several agents that can be run either on one machine or distributed on more than one machine at different places in the network. Each agent has the ability to be configured to capture specific types of network packets such as TCP packets and ICMP packets.

3.2 Packets Queuing

In this stage, the raw packets captured by captured agents are queued into buffers. This stage contains a number of queues equal to the number of captured agents; each queue is associated with one agent as shown in Fig. 2. The association is represented by a labeled arrow from an agent to its corresponding queue. The label on the arrow shows the types of packets captured by the agent. E.g., the TCP capture agent captures TCP packets and puts them into the TCP queue. Therefore, the arrow between them is labeled ‘TCP packets’.

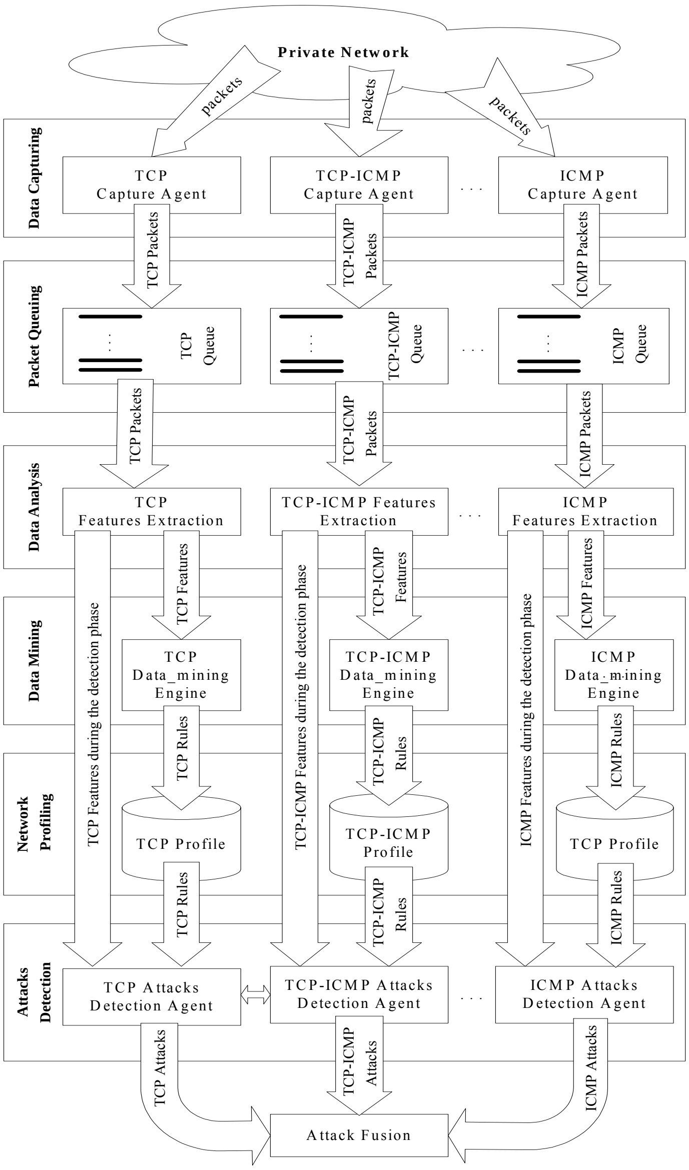


Fig. 2. Novel IDS architecture

3.3 Data Analysis

The Data Analysis stage is responsible for extracting features from the packets kept in the packet queues. This stage passes the extracted features to either the Data mining engines or attack detection agents depending on the mode of operation. During the learning mode, the extracted features are passed to the Data mining engines while during the detection mode, they are passed to the attack detection agents (Fig. 2). Each data analysis module in this stage is associated with a specific packet queue in the previous stage to get its inputs. It extracts features from the raw packets using either one of two analysis methods: the memory-window or memoryless-window. These two methods are described in more detail in Section 4.

3.4 Data Mining

The data mining stage, which is active only during the learning mode, consists of a set of separate data mining modules. Each module works independently to extract a set of fuzzy rules from the analyzed data received from the corresponding module in the features extraction stage. It passes the extracted fuzzy rules to the corresponding module in the network profiling stage. For example, the TCP data mining module receives its input from the TCP features extraction module and passes its output-- fuzzy rules, to the TCP profiling (Fig. 2). The extracted fuzzy rules describe either the normal behavior of the underlying network if the system is used for anomaly detection or the attack behavior or signatures if the system is used for signature based detection. The first step for extracting the rules is to describe the associated attributes. Each attribute or feature is characterized by a fuzzy variable that is defined by three trapezoidal functions: L (Low), M (Medium), and H (High). Before defining each function, four statistical parameters called Q_1 (1st Quarter), median, Q_3 (3rd Quarter), and x_{max}, are calculated from the data associated with each attribute. The value x_{max} is the maximum value received from attribute x. Next, each parameter is divided by $1.1x_{max}$ to normalize it to a value between 0 and 1 (Fig. 3)

After defining the membership functions for each attribute, each data mining engine (agent) in the data mining stage is used to extract a set of fuzzy rules from the data. Each engine deploys a fuzzy association rule algorithm that is used to discover the hidden relationships among attributes and represent these relationships in a form of fuzzy association rules. The fuzzy data- mining algorithm used to extract rules takes the data associated with each attribute and two thresh-

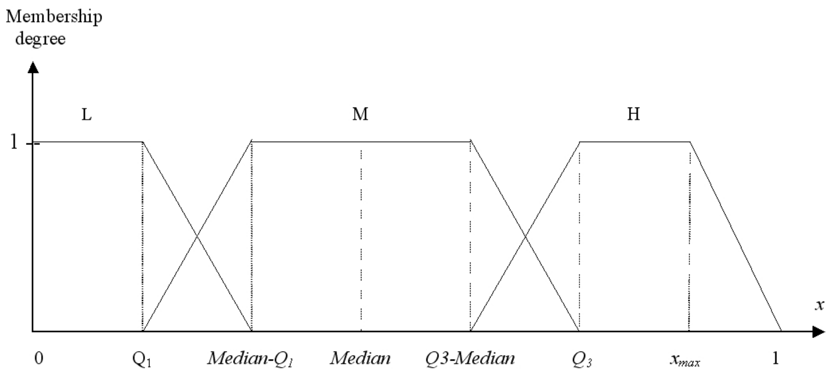


Fig. 3. Fuzzy sets defining term functions of an attribute x

olds called s_{min} and c_{min} as an input, where s_{min} represents the minimum support while c_{min} represents the minimum confidence. These thresholds enable the algorithm to exclude rules that have support and confidence values less than s_{min} and c_{min}, respectively. The algorithm produces fuzzy rules as an output. Each fuzzy rule will be in the form:

$$p \rightarrow q: s, c;$$

Where p and q are fuzzy predicates in conjunctive form and are called the rule antecedent and rule consequent, respectively; while s and c are the rule support and confidence, respectively. Specifically,

p has the form: (a_1 is t_1) $\wedge$ (a_2 is t_2) $\wedge \ldots \wedge$ (a_m is t_m), and
q has the form: (a_{m+1} is t_{m+1}) $\wedge$ (a_{m+2} is t_{m+2}) $\wedge \ldots \wedge$ (a_n is t_n).

Each rule satisfies the following conditions: $a_i \in$ set of attributes A, $t_i \in$ terms of attribute ai,

$$\{a_1, a_2, \ldots, a_m\} \cap \{a_{m+1}, a_{m+2}, \ldots, a_n\} = \phi, \text{ and } s_{min} \leq s \leq 1, \text{ and } c_{min} \leq c \leq 1.$$

For more details about this algorithm refer to [13].

3.5 Network Profiling

Network profiling consists of a number of databases equal to the number of capture agents in the data capturing stage. Each database is used to keep the network behavior or attack signatures extracted from the raw packets captured by the corresponding capture agent during the learning mode. That is, the TCP Profile database is used to keep rules extracted from raw packets captured by the TCP capture agent during the learning mode. The rules in this stage are used by the attack detection agent stage during the detection mode to check whether or not an attack occurs. This stage provides an interface to query the rules of each profile.

3.6 Attack Detection

The Attack detection stage consists of a number of attack detection agents equal to the number of captured agents in the first stage. Agents of this stage are active only during the detection mode. Each agent discovers attacks by matching data received from its associated feature extraction module in the data analysis stage against the fuzzy rules in the corresponding network profile produced during the learning mode. For example, the TCP attack detection agent receives data from the TCP features extraction module and matches it against the rules stored in the TCP profile (Fig. 2). The data is represented in a record structure produced either by the memory-window method (Fig. 7) or memoryless-window method (Fig. 11). The rest of this section describes in more detail the TCP attack detection agent as an example. The TCP matching algorithm is implemented with FuzzyJess [20]. FuzzyJess is a fuzzy inference engine or specifically, a rule-based expert system shell that integrates the fuzzy capabilities of the FuzzyJ Toolkit with Jess, a Java version of CLIPS [21].

The algorithm works as follows: It receives the facts or data from its corresponding feature extraction module in the data analysis stage and then sets up a counter i with a value of 1 to keep

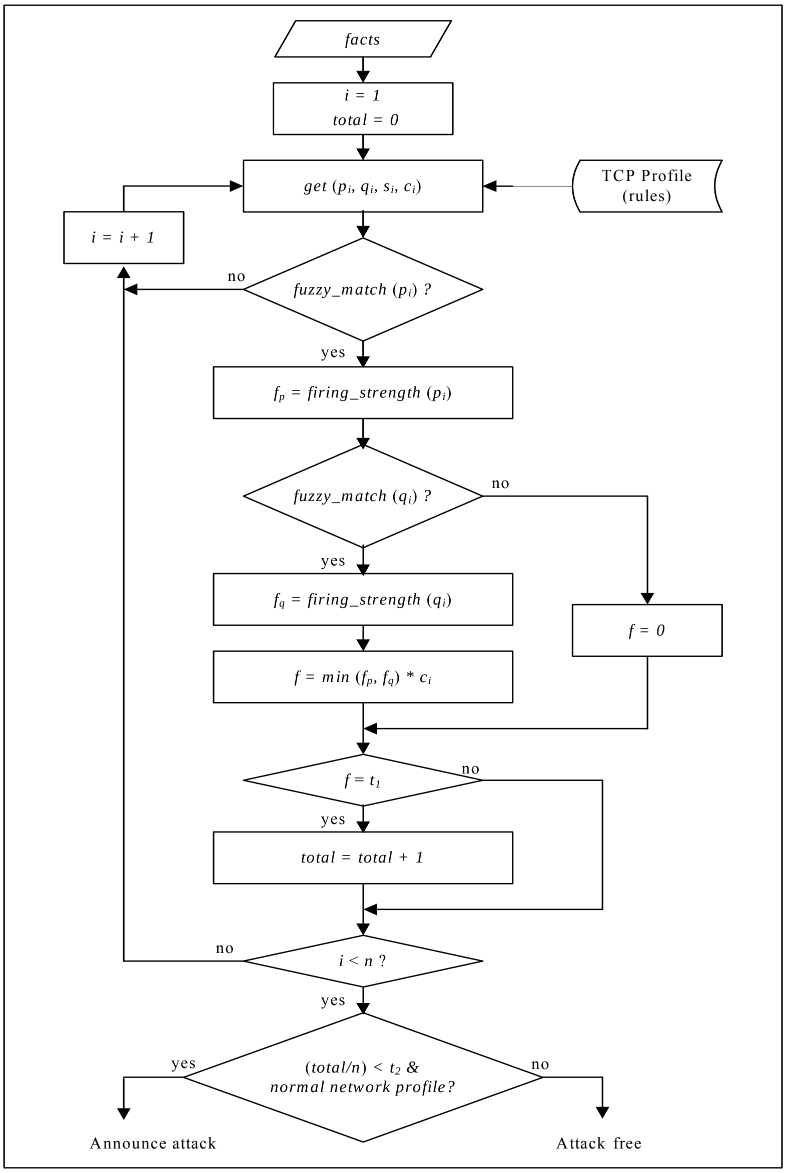


Fig. 4. Attacks detection algorithm

track of the rules. Next, it gets the rule number *i* from the TCP profile and evaluates its antecedent that is denoted by p_i. If p_i has a fuzzy match, the firing strength f_p of p_i is computed and then evaluates the consequent part q_i of the rule. If q_i has a fuzzy match, the firing strength f_q of q_i is computed. Next, the overall firing strength, f, of the rule equals the minimum value of f_p and f_q multiplied by the confidence, c_i, of the rule. If q_i does not have a fuzzy match, f is computed as the value of f_p multiplied by the support, s_i, of the rule.

After computing the value of f of the underlying rule, it is compared against a threshold t_1

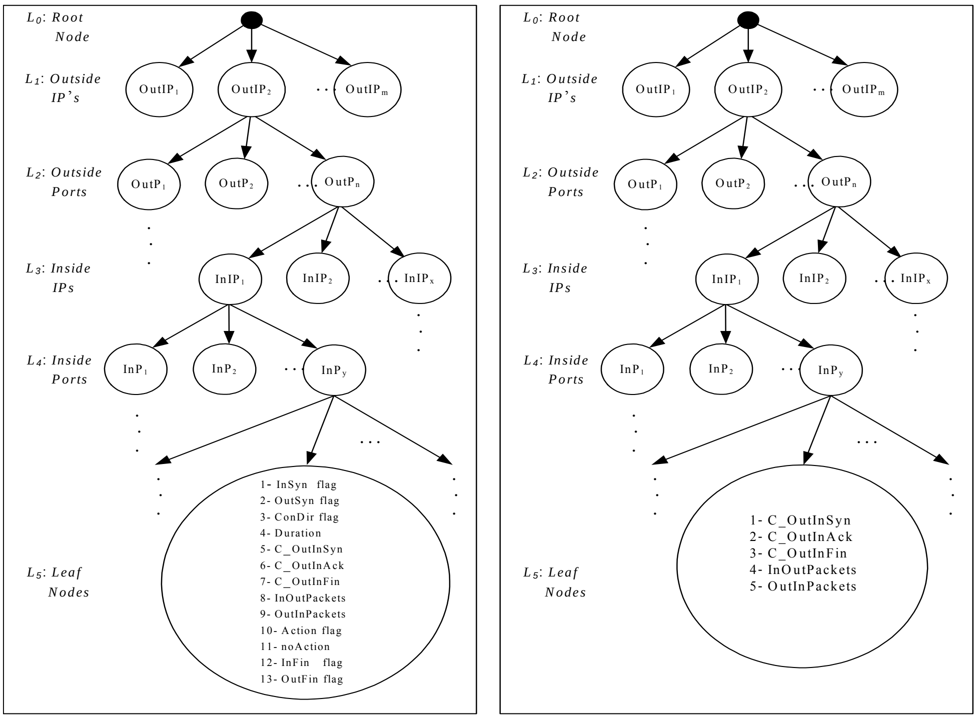
which is a value between zero and one. If the value of f is greater than or equal to t_1, the variable *total* should be incremented. The variable *total* is used to count the number of rules that have firing strength greater than or equal t_1. Next, the value of i should be checked against the value of the variable n, which refers to the number of rules in the corresponding network profile. If the last rule is not reached, i should be incremented and the process is repeated until the last rule is reached. Otherwise, the value of *total* is compared to a threshold t_2, where t_2 has a value between zero and one, too. If the value of *total* is less than the value of t_2, it means that the underlying facts do not match the predefined behavior. If the predefined behavior is the normal behavior of the network under consideration, the corresponding detection agent will announce the existence of an attack. Otherwise, the predefined behavior is the behavior of possible attacks and in this case, the agent will consider the facts as a benign activity. On the other hand, if the value of *total* is greater than or equal to the value of t_2, the detection agent will announce the opposite result to the first case. *i.e.*, if the predefined behavior represents the network behavior, the facts are considered as normal traffic. If it represents attack behavior, the agent considers the facts as an attack activity. In case there is an attack, the detection engine conveys the attack to the attacks fusion in the next stage. Also, it may convey the attack to the other detection agents in its stage to guard against a cooperative attack being undertaken. As attacks can be cooperative, these detection agents can cooperate to detect these kinds of attacks.

3.7 Attacks Fusion

In this stage, the attacks fusion agent collects or receives the decisions from the detection stage and displays the events (connections) that are classified as attacks; it displays the outside IP addresses that were used to attack the network and the inside IP addresses that are under attack. These addresses can be used for further processing, for example, it can be used in the Intrusion Prevention Systems (IPS) to cut the connections with these addresses and to prevent them from connecting to the network again.

4. Data Analysis Methods

This section focuses on analyzing collected data or raw packets to extract parameters that are used to measure underlying network activities. Two methods for this purpose are introduced, the memory-window and memoryless-window analysis methods. The structure of the trees used by the memory-window and memoryless-window is depicted in Figs. 5a and 5b, respectively. Each tree is organized into six levels: L_0, L_1, L_2, L_3, L_4, and L_5. L_0 is the root node that defines a window. L_1 represents the outside IP addresses (i.e., IP addresses that do not belong to the private network). Each IP address is considered a child (e.g., OutIP_1, OutIP_2, …, or OutIP_m) of the root node. L_2 represents the outside ports that are currently connected with the private network. The ports are organized as follows: if outside ports (e.g., OutP_1, OutP_2, …, and OutP_n) are associated with the IP address OutIP_2, then these ports will be the children of the OutIP_2. Level L_3 of the tree consists of the inside IP addresses that are currently connected with the outside IP addresses. The IP addresses in this level will be arranged as follows: suppose that the IP address OutIP_2 in L_1 is connected to the inside IP addresses; InIP_1, InIP_2, ..., and InIP_x through the outside port OutP_n. In this case, these IP addresses will be the children of OutP_n. In this level, an inside IP can be repeated with different ports in L_2.



a) Tree structure of memory-window method. b) Tree structure of memoryLess-window method.

Fig. 5. Construction of trees used in data analysis methods

Level L_4 includes the inside ports that currently connect inside IP addresses with outside IP addresses. These ports are arranged as pictorially described in Fig. 5. Suppose that the IP address OutIP_2 in L_1 is connected through the port OutP_n in L_2 to different inside ports in L_4 (e.g., InP_1, InP_2, …, and InP_y) on the IP address InIP_1 in L_3. In this case, the ports InP_1, InP_2, …, and InP_y will be the children of InIP_1. A path from the root node to a node in L_4 represents a unique TCP connection. Finally, the two trees will differ only in Level L_5 which represents the tree leaves that contain variables by which the values of the parameters used in both methods, memory-window and memoryless-window, can be measured.

4.1 Memory-Window Analysis

In the memory-window analysis method, a window of size *n* (expressed in seconds) is moved over raw packets in non overlapping order. Features or parameters from the set of raw packets within the window are extracted. Some of these parameters memorize information from previous windows, hence the name memory-window. After measuring values of the parameters, the window is moved by *n* and then the new values of the parameters are calculated and so on. Fifteen crucial parameters are carefully selected to feature the network behavior. These parameters and their descriptions are listed in Table 1. The memory-window method uses three different modules to measure the values of these parameters: 1) tree construction, 2) feature extraction, and 3) tree pruning. For each window, these three modules are executed sequentially.

Table 1. Parameters used in memory-window

Parameters	Descriptions
C_OutIPs	It holds the number of outside IPs that is currently connected to the network.
C_Max_OutPs_OutIP	It holds the number of currently connected ports associated with an outside IP that has the maximum number of connections.
C_InIPs	It holds the number of unique-inside IP addresses that are currently connected with outside IP addresses.
C_Max_InPs_InIP	It holds the number of currently connected ports associated with an inside IP that has the maximum number of connections.
C_Full_Con	It holds the number of TCP connections that are currently established.
N_Full_Con	It holds the number of new TCP connections established during the current window.
Max_Con_Duration	It holds duration time of the TCP connection that has the maximum duration. The time is expressed as the number of windows during which the connection is still connected.
Half_Con	It holds the number of half-open connections that are not turned into full-connections until the current window.
Illegal_Con	It holds the number of illegal connections (i.e., connections that do not follow 3-way handshake) during the current window.
C_Max_OutSyn_InIP	It holds the number of Syn packets received by an inside IP and receives the highest number of such packets until the current window.
C_Max_OutAck_InIP	It holds the number of Ack packets received by an inside IP and receives the highest number of such packets until the current window.
C_Max_OutFin_InIP	It holds the number of Fin packets received by an inside IP and receives the highest number of such packets until the current window.
C_InOutPackets_IPs	It holds the total number of packets sent from the network (inside) to outside until the current window.
C_OutInPackets_IPs	It holds the total number of packets received by the network (inside) from outside until the current window.
N_Max_Illegal_Packets	It holds the number of illegal TCP packets received by an inside IP with the maximum number of such packets until the current window.

Tree Construction Module

The tree construction module is responsible to construct or update a tree structure based on the received raw packets during a window size. The tree structure used in the memory-window method is shown in Fig. 5a. Each leaf node includes variables discussed in Table 2. This module

Table 2. Leaf variables used in memory-window method

Variables	Descriptions
InSyn	a flag which is set when a syn packet is sent from inside to outside.
OutSyn	a flag which is set when a syn packet is sent from outside to inside.
ConDir	a flag which is set when a connection is initiated from inside
Duration	holds the number of windows while the connection lasts.
C_OutInSyn	holds the number of Syn packets sent from outside into inside.
C_OutInAck	holds the number of Ack packets sent from outside into inside.
C_OutInFin	holds the number of Fin packets sent from outside into inside.
InOutPackets	holds the number of packets sent from inside to outside.
OutInPackets	holds the number of packets sent from outside to inside.
Action	a flag which is set when a connection does not receive or send at least one packet during the current window. Otherwise, it will be reset.
NoAction	holds the number of windows in which a full connection is ideal.
InFin	a flag which is set when a fin packet is sent from inside into outside.
OutFin	A flag which is set when a fin packet is sent from outside into inside.

reads a set of raw packets within the current window and then updates the tree according to the given structure. Each packet is represented in a record structure depicted in Fig. 6. When the module finishes updating the tree, it calls the features extraction module.

Features Extraction Module

After the tree is updated for the current window, the features extraction module is called to extract the fifteen features discussed in Table 1. The module takes as an input the tree resulting from the tree construction module and outputs the features represented in a record structure like the one depicted in Fig. 8. Next, the tree pruning module is called. This will be demonstrated through a complete example after introducing the tree pruning module.

Tree Pruning Module

The tree pruning module is responsible for pruning the tree by removing the useless connections, which might be one of the following (Fig. 5a):

1. Terminated connections. That is, when *InFin* =1 and *OutFin* =1.
2. Illegal connections. That is, when *InFin* = 0 and *OutFin* = 0.
3. Half-open connections that last for a specific number of windows *w*. That is, if either *InSyn* or *OutSyn* are set and *Duration* ≥ w.
4. Full connections that have no packets sent from either sides for a specific number of windows *z*, That is, if *InSyn* = 1, *OutSyn* = 1, *Action*=1, and *NoAction* ≥ *z*).

This module checks all the connections in the tree and removes any of the connections described above. Then the window is moved by its size and again the tree construction module is called. The rest of this section is devoted to a complete numerical example to demonstrate the memory-window analysis method in more detail.

Suppose the set of raw packets shown in Fig. 6 is received. The necessary information about each packet is described in a record. Each record includes the attributes: *Captured time*, *SIP*,

	Captured Time	SIP	Sport	DIP	DPort	Syn	Ack	Fin
	898171251150	172.16.114.169	1025	195.73.151.50	25	1	0	0
	898171251151	195.73.151.50	25	172.16.114.169	1025	1	1	0
	898171251152	172.16.114.169	1025	195.73.151.50	25	0	1	0
	898171251153	195.73.151.50	25	172.16.114.169	1025	0	1	0
	898171251156	172.16.114.169	1025	195.73.151.50	25	0	1	0
	898171251158	195.73.151.50	25	172.16.114.169	1025	0	1	0
	898171251161	195.73.151.50	25	172.16.114.169	1025	0	1	1
W1	898171251162	172.16.114.169	1025	195.73.151.50	25	0	1	0
	898171251164	193.73.150.45	1030	172.73.151.25	21	0	1	0
	898171251165	193.73.150.45	1033	172.73.151.20	80	1	0	0
	898171251166	172.16.114.169	1025	195.73.151.50	25	0	1	1
	898171251167	195.73.151.50	25	172.16.114.169	1025	0	1	0
	898171251167	172.16.114.169	1026	194.73.248.153	25	1	0	0
	898171251168	194.73.248.153	25	172.16.114.169	1026	1	1	0
	898171251168	172.16.114.169	1026	194.7.248.153	25	0	1	0
	898171251169	194.73.248.153	25	172.16.114.169	1026	0	1	0
	898171251171	194.73.248.153	25	172.16.114.169	1026	0	1	1
	898171251172	172.16.114.169	1026	194.7.248.153	25	0	1	0
	898171251175	172.16.114.169	1026	194.7.248.153	25	0	1	1
	898171251177	194.73.248.153	25	172.16.114.169	1026	0	1	0
W2	898171251178	207.200.73.34	1027	172.16.113.105	80	1	0	0
	898171251180	172.16.113.105	80	207.200.73.34	1027	1	1	0
	898171251181	207.200.73.34	1027	172.16.113.105	80	0	1	0
	898171251183	172.16.113.105	80	207.200.73.34	1027	0	1	0
	898171251185	207.200.73.34	1027	172.16.113.105	80	0	1	0

Fig. 6. Set of raw packets

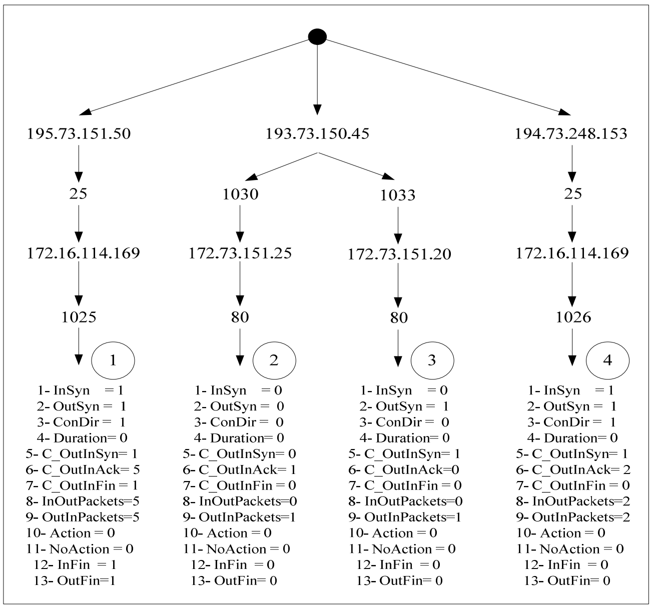
Sport, *DIP*, *DPort*, *Syn*, *Ack*, and *Fin*. The captured time measured in milliseconds is the time at which the packet is captured. The SIP and DIP are the source and destination IP addresses while the Sports and Dports are the source and destination ports, respectively. Finally, the Syn, Ack, and Fin represent the TCP flags.

Let the size of the window be 20 *ms*. Therefore, the first window, w_1, will start from the first packet captured at 898171251150 *ms* and ends 20 *ms* later (i.e., at 898171251170 *ms*). When the tree construction module is executed over w_1, it produces the tree shown in Fig. 7a. The tree has four different connections. The first and the fourth connections are full connections because the *InSyn* and *OutSyn* flags in each connection are sets. Also, they are initiated from inside (*i.e.*, the private network which has the class "A" address 172) because the *ConDir* flags of each connection are set. The second connection is illegal because the *InSyn* and *OutSyn* are reset. The third connection is a half-open connection since the *InSyn* is reset and *OutSyn* is set. The second and third connections are initiated from outside because their *ConDir* flags are reset. Only the first connection received a final packet from each side. Therefore, *InFin* and *OutFin* flags are set. Finally, the *Duration* variable of each connection is equal to zero, which denotes that all these connections are started in this window, w_1.

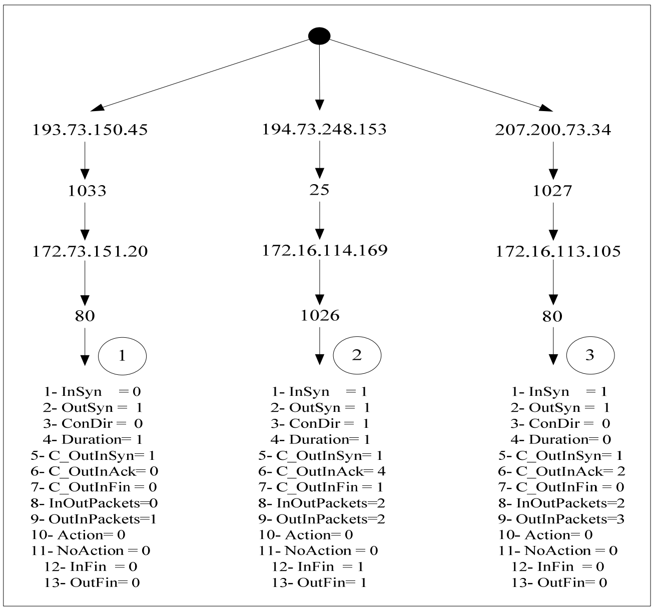
Next, the features extraction module is called to measure the desired parameters. Since each window produces one record of features, w_1 produces the first record shown in Fig. 8. The *C_OutIPs* comes from the first level which contains three outside IP addresses (*i.e.*, C_OutIPs = 3). The IP address 193.73.150.45 has two connections through the ports 1030 and 1033. Since it has the maximum number of ports, *C_Max_OutPs_OutIP* takes the value 2. *C_InIPs* comes from the third level which has four IP addresses: 172.16.114.169, 172.73.151.25, 172.73.151.20, and 172.16.114.169 but the 1st and 4th ones are the same. Therefore, *C_InIPs* takes the value 3. The 172.16.114.169 has two ports: 1025 in the 1st connection and 1026 in the 4th connection. The other IP addresses have only one port each. In this case, *C_Max_InPs_InIP* takes the value 2. The tree shows that there are only two full connections, the 1st and 4th connections, because their *InSyn* and *OutSyn* flags are sets. As a result, the feature *C_Full_Con* takes on the value 2. The feature *N_Full_Con* is equal to the number of full connections that have the *Duration* value zero. In this case, it takes on the value 2. Since all connections have equal durations which equal zero, the feature *Max_Con_Duration* takes on the value 0. Only one connection is a half connection, which is the 3rd connection because *InSyn* is 0 and *OutSyn* is 1. As a result, *Half_Con* is equal to 1. Also, there is only one illegal connection, which is the 2nd connection since both flags *InSyn* and *OutSyn* are equal 0 (i.e., *Illegal_Con*= 1).

The IP address 172.16.114.169 is the address that received the highest number of *syn* packets from outside. It received one through the 1st connection and the other one through the 4th connection. Therefore, the value 2 is given to the feature *C_Max_OutSyn_InIP*. Also, the same IP address is the address that received the highest number of outside Acknowledge packets and final packets through the 1st and 4th connections. Accordingly, values 7 (5+2) and 1(1+0) are assigned to the parameters *C_Max_OutAck_InIP* and *C_Max_OutFin_InIP*, respectively. The total number of packets sent from and received at the private network by all connections are 7 and 9, respectively. Therefore, the parameters *C_InOutPackets_IPs* is equal to 7 and *C_OutInPackets_IPs* is equal to 9. Finally, only one illegal packet is received through the 2nd connection for which the value of 1 is devoted to the parameter *N_Max_Illegal_Packets*.

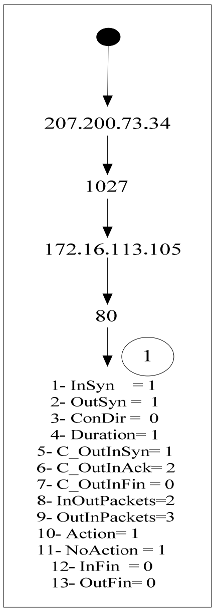
After extracting the features associated with w_1, the tree pruning module is called to remove the unwanted connections. The constructed tree in Fig. 7a resulting from w_1 has two unwanted connections, the 1st and 2nd , that need to be removed. The 1st connection is terminated, since *InFin* = 1 and *OutFin* = 1. The 2nd connection is illegal because *InSyn* = 0 and *OutSyn* = 0. Consequently, the program will increment the duration of the remaining connections. Afterwards,



a) Tree resulted from W_1.



b) Tree resulted from w_2.



c) Tree resulted from W_3.

Fig. 7. Produced trees based on the memory-window method

	C_OutIPs	C_Max_OutPs_OutIP	C_InIPs	C_Max_InPs_InIP	C_Full_Con	N_Full_Con	Max_Con_Duration	Half_Con	Illegal_Con	C_Max_OutSyn_InIP	C_Max_OutAck_InIP	C_Max_OutFin_InIP	C_InOutPackets_IPs	C_OutInPackets_IPs	N_Max_Illegal_Packets
W_1	3	2	3	2	2	2	0	1	1	2	7	1	7	9	1
W_2	3	1	3	1	2	1	1	1	0	1	4	1	4	6	0
W_3	1	1	1	1	1	0	1	0	0	1	2	0	2	3	0

Fig. 8. Features produced by the memory-window method

the window is moved by its size over the next set of packets; the new window is called w_2 as shown in Fig. 6. Finally, the tree construction module is called again and the whole process is repeated. As a result, the new tree is produced as depicted in Fig. 7b.

After creating the tree in Fig. 7b from w_2, the feature extraction module is called to produce the 2nd record in Fig. 8 associated with w_2. Then, the tree pruning module is called to remove the unwanted connections. In the tree in Fig. 7b, there are two connections which need to be removed. The 1st one is a half open connection but its duration is equal to 1 (say the threshold is 1). The 2nd one is terminated so it is removed as in w_1 and the duration of the remaining connection is incremented. The window is then moved over the next set of packets (where no more packets exist in our example). However, the tree in Fig. 7c is produced because there is still memorized information from previous windows. In the same way, the feature extraction module produces the features associated with w_3 in the 3rd record in Fig. 8. Next, the tree pruning module removes this connection because it is a full connection and at the same time, it has no interaction with one window (say the value of 1 is the threshold) as *Action*=1 and *NoAction* = 1.

4.2 Memoryless-Window Analysis

Unlike the memory-window method, the memoryless-window method does not memorize any information from the previous windows. It depends only on the information contained in the raw packets within the current window. Therefore, the same features are used in the memoryless-window method except for the features that depend on information from the previous windows. In this case, the features that will be used in the underlying method include:

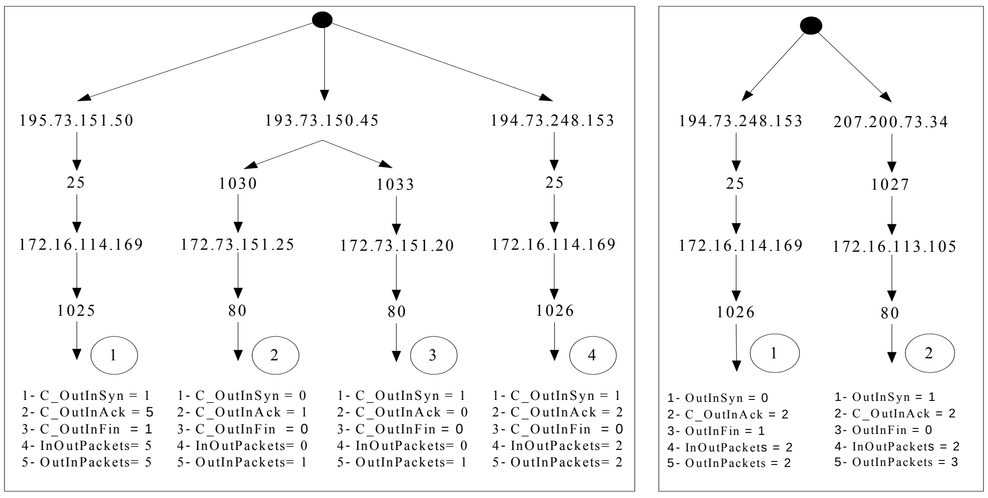
1.C_OutIPs
2.C_Max_OutPs_OutIP
3.C_InIPs
4.C_Max_InPs_InIP
5.C_Con
6.C_Max_OutSyn_InIP
7.C_Max_OutAck_InIP
8.C_Max_OutFin_InIP
9.C_InOutPackets_IPs

10.C_OutInPackets_IPs

These features, except *C_Con*, have the same definitions and computations as in the memory-window method. The *C_Con* denotes the total number of connections which can be full, half, or illegal connections.

The memoryless-window method has only two modules: the tree construction module and the features extraction module. The tree construction module is the same as in the memory-window method but it constructs the tree from scratch for each window. Therefore, the tree pruning module is not needed. Also, the features extraction module calculates the features in the same way as in the memory-window method except for the feature *C_Con*. The value of *C_Con* is calculated as the number of connections in the current window. After measuring the values of these features, the tree of the current window is deleted and a new one is created for the next window. The tree structure used in the memoryless-window method is shown in Fig. 5b. It is the same structure used in the memory-window method except the last level, L_5. It contains only the variables whose values depend only on the current window. The rest of this section applies the memoryless-window method to the same set of raw packets depicted in Fig. 6.

Using this method, only two trees are produced. The first tree shown in Fig. 9a resulted from the tree construction module after processing the set of packets within the window w_1. Then the features extraction module is called which in turn, produces the features depicted in the first record of Fig. 10. After it extracts the features of the window w_1, the tree is deleted and the window is moved by 20 *ms* over the next set of packets. The new window is labeled with w_2 as shown in Fig. 6. Then the second tree shown in Fig. 9b is produced when the tree construction module is called again. Finally the feature extraction module is called to measure the features associated with this tree. It produces values described in the second record of Fig. 10.



a)Tree resulted from w_1. b)Tree resulted from w_2.

Fig. 9. Trees produced based on the memoryless-window method

	C_OutIPs	C_Max_OutPs_OutIP	C_InIPs	C_Max_InPs_InIP	C_Con	C_Max_OutSyn_InIP	C_Max_OutAck_InIP	C_Max_OutFin_InIP	C_InOutPackets_IPs	C_OutInPackets_IPs
W_1	3	2	3	2	4	2	7	1	7	9
W_2	2	1	2	1	2	1	2	1	4	5

Fig. 10. Features produced by the memoryless window method

5. Results

This section presents the experimental results that we obtained from implementing the underlying intrusion detection system. The results were based on a standard data set provided by MIT's LincolnLaboratory [22].

5.1 Data Set

A subset of the DARPA intrusion detection data set from MIT's Lincoln Laboratory was used to train and test the underlying architecture. We decided to use this data set because it is a standard and well known data set for testing intrusion detection systems. The original data set contains network traffic captured in 1998 and 1999 [22]. 1998's data was captured over the course of seven weeks and 1999's data was captured over the course of three weeks. Network traffic corresponding to the first hour of Thursday of the first week of 1999's data was used for training all four intrusion detection systems.

5.2 Experiment Design

We have implemented the present system to analyze the TCP and the IP data packets in the underlying data set. The underlying system architecture composes four different anomaly intrusion detection systems, namely: TCP-MW, TCP-MLW, IP-MW, and IP-MLW. The TCP-MW and TCP-MLW work only on TCP raw packets while IP-MW and IP-MLW work on IP raw packets. The TCP-MW and IP-MW use the memory-window method while the TCP-MLW and IP-MLW use the memoryless-window method. Finally, the agents of the four intrusion detection systems implemented in the underlying architecture are configured to run on the same machine.

The system was tested with two data subsets which combine both normal and attack traffic. The first data set includes the Neptune attack while the second data set includes the Smurf attack. The normal traffic is extracted from 1999's data; specifically, the first thirty minutes of traffic captured from the second hour of Thursday of the first week. This traffic is divided into two files: the first fifteen minutes of the traffic is saved into a *tcpdump* file called “normal-1” while the second fifteen minutes of the traffic is saved into another *tcpdump* file called “normal-2.” The Neptune attack is a SYN flood denial of service on one or more ports. This attack is extracted from 1999's data; specifically, it starts at 11:04:16 on Thursday of the second week and it lasts for about five minutes. It is saved into a *tcpdump* file called “neptuneAttack.” The Smurf attack is a denial of service ICMP echo reply flood. This attack is extracted from 1998's data; specifically, it starts at 8:16:59 on Thursday of the fifth week and it lasts for about five minutes. This attack is saved into a *tcpdump* file called "smurfAttack." The first data set called testDataSet$_1$ is formed by concatenating the traffic in the *tcpdump* files: "normal-1," "neptuneAttack," and "normal-2," respectively. The second data set called testDataSet$_2$ is formed in the same way but the "neptuneAttack" file is replaced with the "smurfAttack" file.

5.3 Experimental Results

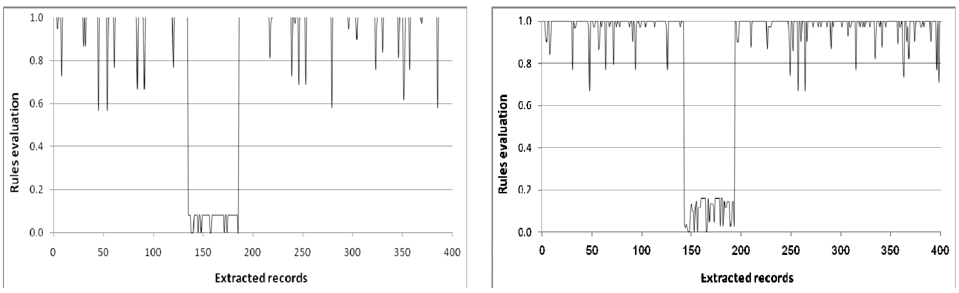
The four intrusion detection systems implemented in the underlying architecture are trained with a support of 0.85 and confidence of 0.85 except the IP-MLW which uses the values of 0.7 and 0.65 for support and confidence, respectively. Different values for the support and confidence are used but the assigned values showed better results. Also, during the detection phase, the threshold t_1 is set to the value of 0.1 for all systems while the threshold t_2 is set to the values 0.3 for the IP-MLW and to the value of 0.4 for the other systems.

TCP Results

The two systems, TCP-MW and TCP-MLW that depend on TCP traffic are tested with testDataSet$_1$. The results of the TCP-MW and TCP-MLW are shown in Fig. 15a and Fig.15b, respectively. The x-axis represents the records extracted from the traffic and the y-axis represents the percentage of rules that have firing strength greater than or equal to the threshold t_1. Since the testDataSet$_1$ is formed from normal traffic injected with the Neptune attack, TCP-MW detected the Neptune attack which started at window 142 and lasted until window 193 as shown in Fig. 11a. Also, the TCP-MLW detected the Neptune attack which started at window 134 and lasted until window 185 as shown in Fig. 11b. Since the memoryLess-window produces fewer windows than the memory-window, the Neptune attack was detected at the window.

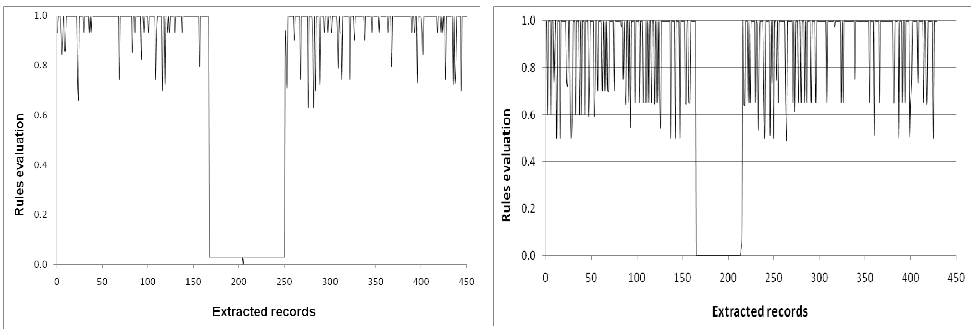
IP Results

In the same way, the IP-MW and IP-MLW that depend on IP traffic were tested using testDataSet$_2$ which includes the Smurf attack. The results of the IP-MW and IP-MLW are described in Fig. 12a and Fig. 12b, respectively. Also, the x-axis represents the records extracted from the traffic and the y-axis represents the percentage of rules that have firing strength greater than or



(a) TCP-MW (b) TCP-MLW

Fig. 11. Test results using the Neptune attack data



(a) TP-MW (b) IP-MLW

Fig. 12. Test results using the Smurf attack data

equal to the threshold t_1. The IP-MW detected the Smurf attack at the window or record 167 and lasted until window 250 while the IP-MLW detected it at window 154 and lasted until window 215.

The results described in Fig. 11 and Fig. 12 show that there is no false negative and false positive associated with the underlying attacks. This shows the efficiency and capability of our proposed architecture and the two developed methods for extracting metrics parameters to detect attacks.

6. Conclusion and Future Work

In this paper, we proposed a new scalable and distributed architecture for network intrusion detection systems. The architecture enables different or the same types of intrusion detection systems to cooperate in detecting attacks. It combined different intelligent techniques such as fuzzy logic and data-mining together with software agents to make up the overall architecture. Also, two data analysis methods: memory-window and memoryless-window were introduced to extract the features used as detection metrics. The memory-window method memorizes information from previous windows while the memoryless-window considers the information associated with the current window. The window is expressed in milliseconds and it is moved over the received set of processed network packets. The proposed architecture together with the data analysis methods was tested using the standard DARPA IDS data set produced by MIT's Lincoln Laboratory. The primary results showed that it is able and efficient to detect network attacks.

In future projects, we plan to extend the novel data representation developed in this work to a system of network visualization that can allow system administrators to interactively manage their networks. Also, even though the developed architecture allows detection models to cooperate, the language that allows them to talk together is not implemented, this too shall be considered for future work.

References

[1] J. Anderson. Computer security threat monitoring and surveillance. Technical Report, James P. Anderson Company, Fort Washington, Pennsylvania, 1980.

[2] D. Denning. An intrusion detection model. IEEE Transactions on Software Engineering, 13(2):222-232, 1987.

[3] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (NIDES). Technical Report SRI-CSL-95-07, SRI International, Computer Science Laboratory, Menlo Park, California, 1995.

[4] M. Roesch. Snort−lightweight intrusion detection for networks. In Proceedings of the 13th Systems Administration Conference, Seattle, Washington, 1999, pp.229-238.

[5] K. Ilgun, R. Kemmerer, and P. Porras. State transition analysis: A rule-based Intrusion detection approach. IEEE Transactions on Software Engineering, 21(3):181-199, 1995.

[6] S. Kumar and E. Spafford. Software architecture to support misuse intrusion detection. Technical Report, The COAST Project, Department of Computer Science, Purdue University, West Lafayette, Indiana, 1995.

[7] T. Lane. Machine Learning Techniques for Computer Security. Ph.D. Dissertation, Purdue University, West Lafayette, Indiana, 2000.

[8] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, 1998.

[9] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection model. In Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, 1999.

[10] J. E. Dickerson and J. A. Dickerson. Fuzzy network profiling for intrusion detection. In Proceedings of the North American Fuzzy Information Processing Society, Atlanta, Georgia, 2000, pp.301-306.

[11] J. E. Dickerson, J. Juslin, J. A. Dickerson, and O. Koukousoula. Fuzzy intrusion detection. In Proceedings of North American Fuzzy Information Processing Society 2001, Vancouver, Canada, 2001.

[12] G. Florez, S. Bridges, and R. Vaughn. An improved algorithm for fuzzy data mining for intrusion detection. In North American Fuzzy Information Processing Society Conference (NAFIPS 2002), (New Orleans, Louisiana), June, 2002.

[13] Aly El-Semary, J. Edmonds, J. Gonzalez, and M. Papa. Framework for hybrid fuzzy logic intrusion detection systems. In Proceedings of the 2005 IEEE International Conference on Fuzzy Systems, Reno, Nevada, May 22-25, 2005, pp.325-330.

[14] Aly El-Semary, J. Edmonds, J Gonzalez, and M. Papa. Implementation of a hybrid intrusion detection system using FuzzyJess. In Proceedings of the 7th International Conference on Enterprise Information Systems, Miami, Florida, 2005, pp.390- 393.

[15] Aly El-Semary, J. Edmonds, J. Gonzalez and M. Papa. Applying data mining of fuzzy association rules to network intrusion detection. In Proceedings of the 7th Annual IEEE Information Assurance Workshop, United States Military Academy, West Point, NY, 2006, pp.100-107.

[16] J. Luo and S. Bridges. Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection. International Journal of Intelligent Systems, 15(8):687-703, 2000.

[17] M. Qin and K. Hwang. Frequent episode rules for intrusive anomaly detection With Internet data mining. In Proceedings of the 13th USENIX Security Symposium, 2004.

[18] S. Bridges and R. Vaughn. Fuzzy data mining and genetic algorithms applied to intrusion detection. In Proceedings of the 23rd National Information Systems Security Conference, Baltimore, Maryland, 2000.

[19] Ming-Yang Su. Discovery and prevention of attack episodes by frequent episodes mining and finite state machines. Journal of Network and Computer Applications, Vol.33, Issue 2, March, 2010, pp.156-167.

[20] The FuzzyJess toolkit. http://www.cs.vu.nl/~ksprac/2002/doc/fuzzyJDocs/FuzzyJess.html.

[21] The C Language Integrated Production System (CLIPS). http://clipsrules.sourceforge.net/.

[22] DARPA Intrusion Detection Data Set. http://www.ll.mit.edu/ mission/communications/ist/corpora/ideval/data/index.html,

Aly M. El-Semary
He received his B.S. degree in Systems and Computer Engineering from Al-Azhar University, Cairo, Egypt in 1992, and M.S. and Ph.D. degrees in Computer Science from Tulsa University, USA in 2001 and 2004, respectively. He works for the Department of Systems and Computer Engineering, Faculty of Engineering, Al-Azhar University, where he is currently an assistant Professor. However, he is currently working as a visitor for Computer Science and Engineering College, Taibah University, Saudi Arabia. His current interests include network and computer security, sensor networks, fuzzy logic, data-mining, and neural networks.

Mostafa Gadal-Haqq M. Mostafa
He received his B.Sc. (Honor) in 1984, M.Sc. in 1989, and Ph.D. in 1996, all from the Faculty ofScience, Ain Shams University, Cairo, Egypt. He is now a professor of computer Science at the Faculty of Computer and Information Sciences. Prior to joining the Faculty of Computer and Information Science, he was with the Faculty of Science, Ain Shams University. He also joined the College of Computer Science and Engineering, Taibah University, Madinah, K.S.A., from 2001 to 2009. He has several publications in pattern recognition letter, CVPR, ICIP and other respected journals and conferences. His research interests include pattern recognition, computer vision, image processing, speech processing, and computer security.