LAMAIDS: A Lightweight Adaptive Mobile Agent-based Intrusion Detection System

Mohamad Eid, Hassan Artail, Ayman Kayssi, and Ali Chehab
(Corresponding author: Hassan Artail)

Electrical and Computer Engineering Department, American University of Beirut
P.O.Box: 11-0236 Riad El-Solh, Beirut 1107 2020, Lebanon (Email: hartail@aub.edu.lb)

(Received May 11, 2006; revised and accepted Aug. 1, 2006)

Abstract

Intrusion detection system (IDS) has become an essential component of a computer security scheme as the number of security-breaking attempts originating inside organizations is increasing steadily. The idea of filtering the traffic at the "entrance door" (by firewalls, for instance) is not completely successful since it does not allow monitoring of local traffic. This paper presents a lightweight and adaptive mobile agent-based intrusion detection system (LAMAIDS) that detects intrusion from outside the network as well as from inside. A main machine, being a typical intrusion detection system residing at a secure location, creates mobile IDS agents and dispatches them into the network. The mobile IDS agents are equipped with lightweight IDS capabilities and decision-making. On each hop, the agents sniff the network traffic and look for abnormal activities using a set of rules supplied by the main machine. Simulation results based on real-world scenarios demonstrate significant improvements in terms of detection rate, network overhead, and adaptability, scalability, and fault tolerance.

Keywords: Defense systems, distributed systems, intrusion detection systems, mobile Agents

1 Introduction

The number of information warfare attacks has become increasingly sophisticated. Annual reports from the Computer Emergency Response Team (CERT) indicate a significant increase in the number of computer security incidents each year, with 137,529 in 2003 [5]. Added to this, recent research has demonstrated that approximately 70% of attacks originate inside organizations and are made by inside users [9]. The field of Intrusion Detection has seen a considerable amount of research and development in response to the above challenges aimed at making Intrusion Detection Systems (IDSs) more effective in detecting unauthorized and malicious access by computers or computer users. Ideally, an IDS is characterized by the following features [7]: 1) it runs continually with minimal supervision and intervention from the end user, 2) is able to operate in a hostile computing environment while exhibiting a high degree of fault-tolerance, 3) can be configured to adapt to changes in the system and to user behavior over time, 4) imposes a minimal overhead on the system, 5) is able to perform data fusion and correlate information from multiple sources.

This paper treats three main challenges that face an IDS, namely the ability to monitor local traffic and detect local intrusions, the dynamic evolvement of the detection rule-sets, and the immunity of the intrusion detection system itself. In response to these issues, we propose the Lightweight Adaptive Mobile Agent-based Intrusion Detection System (LAMAIDS) with the following features: 1) providing a highly distributed IDS, with mobile processing units to capture and analyze relevant data asynchronously and independently from the main machine, 2) roaming the internal network, mobile agents are capable of detecting attacks from within the network, 3) securing against attacks targeting the IDS itself since attackers do not know the exact locations of the mobile IDS agents, 4) using dynamic and centrally-controlled rule-sets, meaning that these sets can adapt to the state of the network, 5) adapting to the severity level of the attack by increasing the degree of monitoring traffic.

IDS systems are usually categorized according to misuse or anomaly detection models. With the first type, detection is performed by exploiting known vulnerabilities and attack signatures while with anomaly-based models, detection is based on flagging when intrusive activities lead to deviations from what is considered normal operations. IDS systems are also classified as host-based, network-based, hybrid, or distributed. Host-based systems base their decisions on information obtained from a single host (usually audit trials). Network-based systems deduce malicious behavior based on the format and content of data packets on the network. A hybrid system combines both approaches while a distributed one consists of multiple IDS units in the network and cooperate to provide a global view of an attack.

The rest of the paper is organized as follows: Section 2 presents a review of related research to intrusion detection with a focus on mobile agent-based systems. Section 3 describes the various entities of LAMAIDS while Section 4 provides a discussion of its design and implementation. Section 5 presents the performance evaluation of the system while Section 6 presents a conclusion of the performed work and ideas for future work.

2 Related Work

Current intrusion detection systems still face issues that include centralization or partial distribution, static reconfiguration, high false positive/negative rates, vulnerability to direct attacks, limited flexibility, and lack of adaptability and extensibility [27]. Given the characteristics of mobile agents (e.g., autonomous execution, dynamic adaptation, and scalability) and their potential to overcome a number of limitations intrinsic to existing IDS system [17], many mobile agent-based intrusion detection mechanisms have been proposed. These systems focus on distributed data collection [11, 12, 13, 14, 18], autonomous behavior [2, 6, 8, 15, 24], and fully distributed solutions [3, 4, 10, 22, 25, 26]. None of these efforts however shares LAMAIDS's abilities for dynamically updating the detection mechanisms and adaptation to network security states, nor do they include provisions to ensure their own survivability in hostile environments.

Distributed data collection for intrusion detection systems were introduced to overcome the susceptibility of single point data collection and detection. In [12], a scheme is proposed where lightweight agents travel between monitored systems in a network of distributed systems, obtain information from data-processing agents, classify and correlate information, and report the information to both a user interface and a database, via mediators. A new Mobile Agent Distributed Intrusion Detection System (MADIDS) was proposed to process the great flow of intrusion detection data transfer in high-speed networks [11]. MADIDS comprises specialized agents: event generation agents for collecting intrusion data, event analysis agents to do data analysis, event tracking agents that track intrusions based on input from the analysis agents, and agent server, which supervises and assigns tasks to the agents. The NADIR system [13] performs distributed data collection by employing the service nodes on the Los Alamos National Laboratory's Integrated Computer Network (ICN) to collect audit information, which is then analyzed by a central expert system. This work presents many interesting results and considerations regarding the collection, storage, reduction and processing of data in large computer networks.

The approach of using autonomous agents introduced the idea of lightweight, independent entities operating in concert for detecting anomalous activities. The approaches described in [24] and [2] propose an architecture for a distributed intrusion detection system based on multiple independent entities. Agents are used mainly for forming a set of lightweight software components, which can be easily reconfigured. They look for interesting events and report their findings to a single transceiver that oversees their operations and reports their results to one or more monitors that are responsible for the network.

The work in [3, 22, 25] presents a fully distributed architecture where data collection and information analysis are performed locally without referring to a central management unit. For instance, the architecture in [22] comprises two components: IDS agents and a stationary secure database that supplies misuse signatures. The agents are responsible for detecting intrusion based on local audit data and by collaborating together to decide if the network is being attacked. Each agent has a local database that warehouses information such as signature files and users patterns. This system requires that an agent resides on every host, thus resulting in a potentially large number of IDS agents in the network. In a similar approach, a large number of small-size mobile agents are deployed in the network to do monitoring, decision-making, notification, and reaction to attempted intrusions [4]. When an agent considers an activity suspicious, it advises the other agents in order to activate agents with higher level of specialization for the suspected intrusion type. Once there is a consensus among agents about the existence of an intrusion, a message is sent to an operator, who is supposed to launch one or more reactive agents. Another architecture that employs collaboration is proposed in [26]. Distributed intrusion detection is implemented by means of Cooperative Security Managers (CSMs) that correlates data collected from local IDSs and other CSMs. The security managers are implemented as stationary agents, thus imposing a continuous overhead on the hosts they reside on and presenting a challenge for configuring and updating the system.

3 LAMAIDS Architecture

The system administrator initially starts the main intrusion detection processor (MIDP) stationary component which in turn creates the user interface agent. The latter prompts the user for the startup conditions of the system (number of startup agents and their visit lists, rules sets, severity lists, among others). The MIDP then creates agents, configures them through briefcases they carry around, and dispatches them into the network. Once launched, the agents perform intrusion detection and take local measures as well as notifying the MIDP when attacks are suspected. The MIDP may perform further analysis of the received data and inform the user if an attack is deemed real. Agents primarily respond to suspected attacks by means of cloning to increase the level of monitoring in the network. When the suspicious activity subsides, the cloned agents in the network become subject to gradual disposal.

LAMAIDS comprises three primary components: the

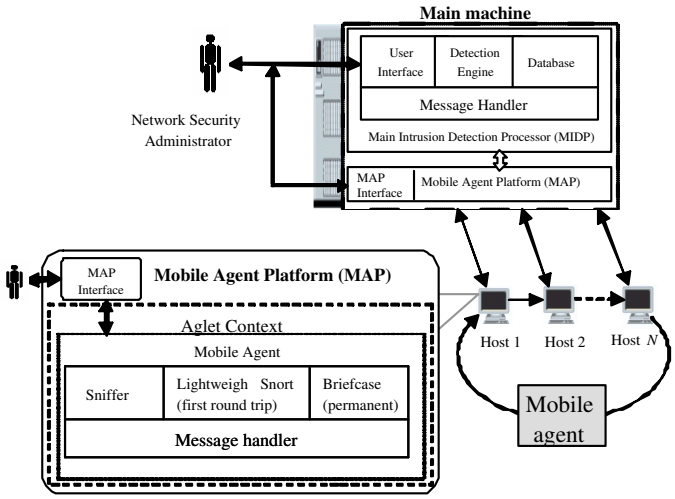

Figure 1: General architecture of LAMAIDS

Attack Name (keyword)	Severity value
PORT SCAN	1
Large ICMP	2
Back Orifice	2
DNS zone transfer	3
FINGER root	3
WEB-IIS	1
DOS Bay/Nortel Nautica Marlin	1

Visit list (Vector)

atp://pc01r1:9000
atp://pc02r1:900
atp://pc03r1:9000
atp://pc04r1:9000
atp://pc05r1:9000
atp://pc06r1:9000

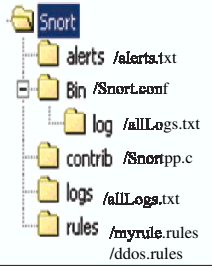

Figure 2: A sample briefcase

MIDP, mobile agent platform, and distributed mobile IDS agents. A high level view of the architecture is given in Figure 1.

The MIDP acts as the manager of the proposed distributed framework and represents the central intrusion detection processing and analysis unit. It cross-relates and analyzes the multiple log files sent by the dispatched agents, provides and updates the rule sets and severity lists for them, and interfaces the IDS to the system administrator, among other things. The MIDP comprises four components: detection engine, database, user interface, and message handler. The major function of the detection engine is the collection and correlation of IDS data from the agents. The MIDP is mostly concerned with linking events across the network and providing the organization with a heuristic analysis of the status of the network. The database contains a secure trusted repository for the mobile agents to obtain latest information about attacks. It contains attack traces or signatures (rule set) and severity level associated with each attack (severity list). A severity level defines the response mechanism that agents should use when particular attacks are detected. The database also contains credentials of existing agents in the system, including the agent ID, its child and parent IDs (if they exist), the agent visit list, the agent proxy, and the host at which the agent is currently residing.

The Mobile Agent Platform (MAP) can create, interpret, execute, transfer, and terminate/kill agents. The platform is responsible for accepting requests made by the MIDP, generating mobile IDS agents, and dispatching them into the network. The platform is a small server program that listens for incoming agents and resides in each host on which an agent is expected to run.

The mobile IDS agent has three primary functions: sniffing the network traffic, performing intrusion detection, and executing cloning mechanisms. Sniffing is done using a platform-specific library of functions while intrusion detection is performed by a mobile and lightweight intrusion detection program that moves with the agent. Cloning is the primary mechanism used by agents when intrusion is suspected in order to increase the degree of monitoring and hence, to increase the level of certainty about a probable intrusion. Sniffing and detection work in parallel, with the former continuously writing raw network packets to a specific log file and the latter reading from it. If the header or the contents of a decoded packet matches one or more detection rules, the alerting subsystem logs the alerting message to an alert text file.

As the agent moves from one host to another, it carries with it a briefcase that contains data comprising the severity list, the visit list, the attack list, and the lightweight IDS plus its rule set (A sample is shown in Figure 2). The severity list identifies what the agent should do upon detection of an attack, determines the type and volume of logged data that the agent should send to the MIDP, and more importantly, defines the cloning strategy. The visit list includes the URLs of the hosts on the agent's itinerary and is usually updated when cloning and disposal take place.

Agents communicate by message passing through proxies. An agent message comprises two fields: name of the message and optional arguments that are used for attaching data. Basically, messages are exchanged when an agent is created, an agent's briefcase is updated, an attack is suspected, an agent is disposed, or when information is requested by the MIDP.

Cloned or child agents are created temporarily, with a limited lifetime, to aid in detecting intrusions. A child agent is terminated when its life time expires but it has to request permission from the main machine (MIDP). If termination is granted, the agent is terminated after handing over its network segment to its parent. A child agent can become a parent when it clones itself, in which case it cannot be terminated until all its children are ter-

Table 1: A cloning matrix example (n is the size of the original visit list and 1 means that cloning should definitely occur)

		Visit list size (original = n)					
		n	$[n\text{-}1, \lceil n/2 \rceil]$	$[\lceil n/2 \rceil\text{-}1, \lceil n/4 \rceil]$	$[\lceil n/4 \rceil\text{-}1, \lceil n/8 \rceil]$	$[\lceil n/8 \rceil\text{-}1, 2]$	1
Severity level	1	1	>T/16	>T/8	>T/4	>T/2	-
	2	>T/16	>T/8	>T/4	>T/2	>T	-
	3	>T/8	>T/4	>T/2	>T	-	-
	4	>T/4	>T/2	>T	-	-	-
	5	>T/2	>T	-	-	-	-
	6	>T	-	-	-	-	-

minated successfully. Although requesting permission for termination from the MIDP causes slight delays in the termination process, it was deemed necessary to ensure consistency in the system and to protect against miscommunications.

The agent uses a cloning strategy that mainly involves three factors. These are the visit list size (a visit list size of 1 means that the agent is stationary and cannot be cloned), the severity level of the attack (a value between 1 (most severe) and 6 (least severe)), and the residence time (the longer an agent has to stay at each host, the more necessary it is to have more agents in the network). The strategy is represented in the form of a matrix that is shown in Table 1 for an example of 16 hosts. As shown cloning decisions are mostly influenced by the determined severity level, which is closely tied to the type of the attack and its probability. This is however based on the size of the visit list of the agent and its residence time, as was justified above. The initial visit list and the residence time (as set by a suitable threshold, T) were divided into intervals with exponentially-increasing widths to give the agent more freedom to clone in response to the severity level. For example, for a severity level 1 and for a list of n hosts, cloning is a must. For half the visit list, cloning is to take place if the residence time is greater than T/16.

Given that the child agent inherits its capabilities from its parent, the visit list is split in half between itself and its parent. This will produce the effect of a binary search when trying to localize an intrusion in the network upon sensing that a potential one exists.

4 Implementation

The mobile agent platform that resides at each host was implemented using the Aglets version 2.0 agent system, chosen because of its availability, ease of use, reliable messaging, dynamic routing, and support for mobile agents. The runtime consists of a server process to handle incoming and outgoing aglets. The fundamental operations in an Aglets system are creation, cloning, dispatching, retraction, activation/deactivation, and disposal. The Aglets system has three primary elements: the context, the proxy, and the aglets [16]. The context is a stationary object that provides a uniform execution environment for aglets in an otherwise heterogeneous environment. One node may run multiple server processes where each process can host multiple contexts that are able to manage multiple aglets. Incoming aglets are received and inserted into the context by the server. Every context has a security manager that protects the underlying host from malicious aglets. The proxy interface, which is the mean by which location transparency is achieved, provides a common way of accessing an aglet and also acts as a shield object that protects an aglet from malicious aglets. The aglet is an autonomous mobile Java object that has its own thread of control. It is event driven and allows the developer to add customized listeners into the aglet. Listeners catch particular events in the life cycle of an aglet and allow for taking corresponding actions. There are three types of listeners, namely, clone listener, mobility listener, and persistence listener.

4.1 MIDP Implementation

The MIDP was implemented as a java class that extends the Aglets class. It runs *snort* version 2.0 [23] and includes a repository database with severity and rules information plus a helper class agents that keeps track of running agents. One method in MIDP creates startup agents and a second method processes incoming messages from agents and takes corresponding actions. There are seven basic messages that are employed to indicate agent cloning, agent disposal requests, data requests by and from the MIDP, attack detection, and configuration updates.

4.2 Mobile IDS Agent Implementation

The mobile agent is implemented as a class whose data members consist of two vectors for the severity list and the attack list, a string to save the rule set file name, the proxy

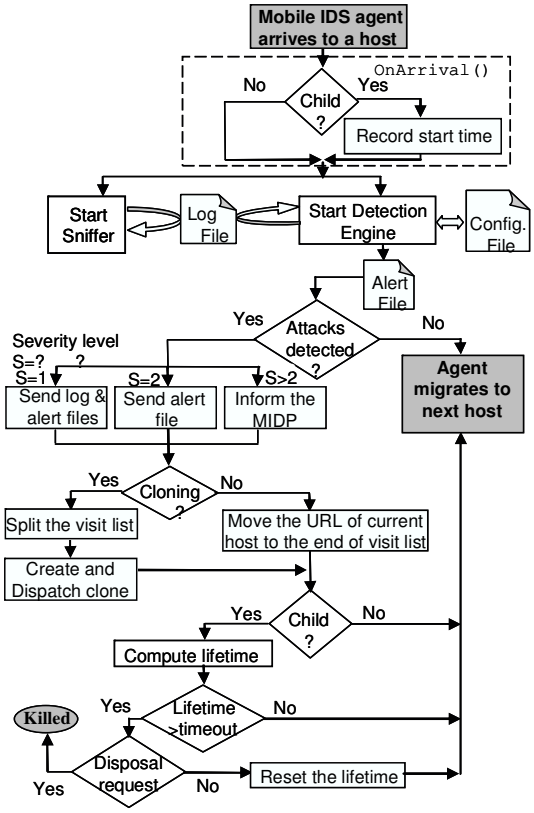

Figure 3: Life cycle of an agent on a host

of the MIDP, and an instance of the `SeqItinerary.java` that implements the `Itinerary.java` class. The latter defines a model for an itinerary with two abstract methods, `go` and `hasMoreDestinations`. The aglet uses the `go` to dispatch itself while `hasMoreDestinations` is used to check if the destinations vector contains more URLs. `SeqItinerary.java` implements the abstract methods of `Itinerary` and keeps track of the current destination of the aglet. The class uses functions that return the visit list, insert/delete a URL into/from the itinerary, and return the number of hosts on the agent's itenerary. Concerning function members, the agent's class has six functions (described in Table 2) used to catch and respond to specific events during the lifecycle of the agent. Figure 3 represents a flow diagram for the agent lifecycle and indicates the sequence of events it goes through.

4.2.1 Sniffing and Detection

Packet sniffing was implemented using the *Jpcap* Java API, which is a set of Java classes that provide access to low-level network data through packet capture and processing. The sniffer may be started upon the agent's first visit to the host within a daemon process that runs continuously. That is, the sniffer is invoked periodically, where it runs and sleeps for a configurable time on a repeating basis. When it runs, it writes the sniffed network packet data to a specified log file, regardless of whether an agent is currently running on the host or not. By controlling the sleep time of the sniffer, agents can control the frequency of the sniffer's execution and hence allows for inspecting the packets on the network more or less frequently.

Each agent carries with it a lightweight snort (LWS) engine [20] that detects intrusions by inspecting the entries in the log file that the sniffer writes to. LWS is a scaled-down version of the full-fledged snort system that runs against a limited rule set and dumps alerts into a text file. It measures roughly 100 KB in its compressed source distribution and takes around 0.5 second to be configured and activated by an agent. During its first itinerary, the agent installs and configures LWS on every host and thus it can readily launch it on subsequent visits. LWS can be easily configured and it can be instructed to deactivate sniffing since it is performed by a native application and to log alerts in ASCII format. The agent can attach new rule files to LWS and detach others that are deemed inapplicable since unnecessary rules lead to false positive alerts. A rule consists of two parts: the rule header, which comprises an action type field (log, alert, or pass), type of packet field (TCP, UDP, etc.), and source and destination IP addresses and ports.

4.2.2 Agent Cloning

When an agent determines that an intrusion is in progress, it calls the `onClone` method. Using this method, the parent retrieves the proxy of the clone and gives it a copy of its own configuration. The parent keeps half of its itinerary and encapsulates the other half in the UPDATE message and sends it to the clone. Upon its creation, the agent informs the MIDP about its existence and about its ID, parent's ID, and the URL of the host where it was created. When the clone receives an acknowledgement from the MIDP, it sets the expiration time and starts its itinerary.

The aglet terminates after it has fulfilled its tasks and after it has received permission from the MIDP. It does so by invoking the `dispose` method which removes the agent from its current context and kills all associated threads. A diagram that shows the methods applicable to the clone and its parent are shown in Figure 4.

5 Performance Evaluation

To study the feasibility of the LAMAIDS architecture, we have conducted a series of experiments to evaluate its effectiveness. Our experiments have shown that the system can rapidly reach an all-snort state when continuous attacks are launched against the home network, and return to the idle state when the attacks are terminated.

5.1 Simulation Setup and Procedure

The prototype network comprises a main station, 32 Windows and 8 Linux workstations on which the Aglet server was installed. The 40 computers are connected via 2

Table 2: Mobile IDS agent methods and descriptions

Method	Description
`Oncreation()`	Executed when the mobile IDS agent is initially created. The agent sends to the MIDP a NEW_AGENT message that contains the credentials of the agent. The MIDP sends briefcase data as a reply.
`OnArrival()`	If child, the agent saves the arrival time in a variable so it calculates the time spent at this host before departure.
`onClone()`	Used by the clone to set the boolean flags _child and _clone to true.
`onCloned()`	After the clone is created, the parent sends half of its visit list via an UPDATE message to its clone. The clone replies with an ack.
`onDispatch()`	If child, the agent computes the residence time at the current host and subtracts it from the timeout interval. If the result is less or equal to zero, the agent sends the MIDP a DISPOSAL_REQUEST. If an intrusion is detected or if the MIDP replies with a negative ack., the remaining lifetime is reset to the timeout.
`Run()`	After creation, the agent sends the MIDP a NEW_AGENT message with its credentials, sets the timeout interval, and resets the _clone field if child. The agent then creates and runs a thread for LWS. Meanwhile, it checks for alerts and sends an ATTACK_DETECTED message to the MIDP with logged and alert data if applicable. Using the cloning decision matrix, the agent decides whether to clone or not. Upon the expiration of its residence timeout on the host, the agent executes the go method if the visit list has more than one host.

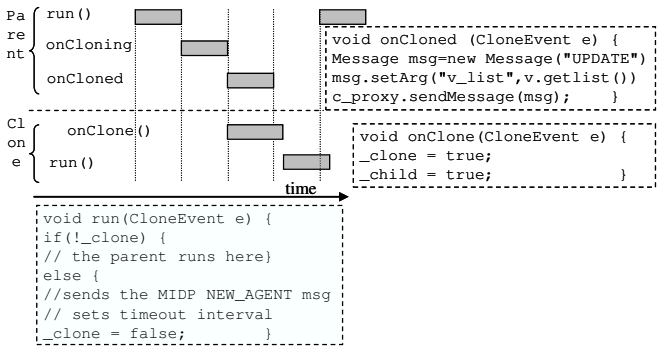

Figure 4: Collaboration diagram for aglet cloning

switches, thus forming a switched network. All the experiment results represent average values from 10 experiment runs. For each run, all Aglet servers were restarted and all log files were cleared.

For a realistic testing environment, attacks needed to be interjected into a volume of background traffic. It was prohibitively expensive to run an exhaustive test of all known attacks, but instead, a representative attack subset from each category was generated (equivalence partitioning [21]). The seven different attack types shown in Table 3 were used in the evaluation. The normal (background) traffic is generated using the WINJET packet generator [19] while attacks are simulated using the BLADE Software [23]. The attack sessions that were simulated lasted for 600 seconds.

In conducting the evaluation, the following procedure was followed:

1) Configure the MIDP to specify the number of startup agents and visit lists.
2) Start the MIDP and the main IDS (snort).
3) MIDP launches the specified number of agents into the 40-host network.
4) Generation of background traffic and verification with the network analyzer.
5) Launch specific attacks against target host(s).
6) Record and audit attack detection performance.
7) 600 seconds later, terminate the traffic generation.
8) Cleaning the alert files.
9) Repeat Steps 3 through 8 for a total of 10 trials.
10) Average and record the results.

5.2 Performance Metrics

We investigate the performance of our system along three major dimensions, namely the ability of detection, system adaptability, and workload capacity. Errors in detection occur usually in two forms: false positives and false negatives. System adaptability is the ability of the system to change dynamically in response to the status of the network by means of adjusting the agent population to reach a compromise between detection capabilities

Table 3: Simulated attacks and their description

Attack Name	Description
DoS Smurf	ICMP echo reply flood, caused by an ICMP echo packet with spoofed address (of victim) sent to a network broadcast address.
Backdoor Back Orifice	A remote administration tool that allows almost complete control over a computer by the remote attacker.
Nmap TCP Scan.	Scans many ports to determine available services on a single host using UDP packets
Finger User	Allows an attacker to disrupt your network using the redirection capability in the finger daemon.
RPC Linux Statd Overflow	Buffer overflow vulnerability exists making it possible for malformed requests by an attacker to be devised giving root privileges.
DNS Zone Transfer	DNS server provides information for all DNS resource records registered with DNS server that can be used by attackers to better understand a network
HTTP IIS Unicode	An attacker could send a specially crafted URL containing Unicode characters to access files and folders on the Web server with the privileges of the IUSR account. Allow the attacker to add, delete, or modify files, or execute commands on the server.

and workload. Finally concerning network load, we examine CPU time and network bandwidth consumption. The CPU time depends mainly on the residence time of the agent at a host and the number of agents. The network bandwidth consumption is directly related to the number of deployed agents.

5.3 Analytical Analysis

For the three performance metrics mentioned above, we derive expressions for the probability of detection as a function of agent population and residence time, for the host and network resource consumption, and for the agent population as a function of time under attack and no attack situations. To start with, given a network of N hosts, we let P be the probability that an agent and the attack meet at a single host assuming that when they line up, the agent will certainly detect the attack. Let L be the number of hosts assigned to each agent, T_r be the residence time of the agent at every host, and α be the number of agents in the network at any time. It follows that:

$$\alpha = \frac{N}{L}.$$

5.3.1 Probability of Detection

Suppose a particular attack on one or more hosts lasts for a period of T seconds, then a measure that indicates the fraction of time in which the attack and the agent meet at the same host can be expressed as:

$$\frac{\min(T, T_r)}{L \times T_r}.$$

Now, if we designate by P^i_{LWS} the probability that lightweight snort can detect an attack of type i when running on a host for the duration of the attack, then the probability of detection when the agent and the attack meet at the same host is (assuming that $T < T_r$):

$$P^i_{ad} = \min[\frac{P^i_{LWS} T}{L \times T_r}, 1].$$

We note that P^i_{LWS} is a function of the rule set of LWS and may be computed experimentally by subjecting the host on which LWS is running to K instances of a particular attack and then count the number of entries k in the log file that correspond to detected attacks. In this case, P^i_{LWS} can simply be computed as k over K.

The probability of detection of an attack in the network is a function of mainly two variables: L, the number of hosts per agent (or equivalently, α, the agent population for a given network) and the residence time T_r. More specifically, the probability that an attack of type i is detected at one of the L hosts is the probability that the agent detects the attack after meeting at one host and not meeting at all the remaining hosts. Therefore, the probability that an attack is detected while the agent is roaming the segment of L hosts is:

$$P^i_d(L, T_r) = P^i_{ad}(1 - P^i_{ad})^{(L-1)} = P^i_{ad}(1 - P^i_{ad})^{(\frac{N}{\alpha}-1)}.$$

Now we can compute the value of the residence time T_r that renders the maximum probability of detection with respect to residence time. We do this by taking the derivative of P^i_d with respect to T_r and setting it to zero. Afterwards, we derive the corresponding expression for P^i_d and

get

$$T_{r_{opt}} = P^i_{LWS} \times T \quad (1)$$

$$P^i_{d_{\max}}(L) = \frac{(L-1)^{L-1}}{L^L}. \quad (2)$$

From the above, one can determine that the probability of detection increases as function of T_r until it reaches a maximum after which it starts to decrease. This behavior reflects the fact that for small values of T_r the agent does not have sufficient time to capture attack patterns, thus yielding a lower P_d. After a period of Tr_{opt} the agent would be residing at the host more than necessary, thus missing potential attacks at other hosts, thus causing P_d to decrease as well. On the other hand and as evident from Equation (2), $P_d(\alpha)$ increases when the agent population α increases.

5.3.2 Resources Consumption

Resource consumption comprises two components: CPU time consumption and network bandwidth consumption. The host component H may be expressed as a linear function of α and Tr. It is safe to assume that the CPU time increases linearly as the number of running agents increases. If K_1 is the average CPU time consumption during the execution of a mobile IDS agent for a unit time, then we can express H as

$$H(\alpha, T_r) = K_1 \alpha T_r. \quad (3)$$

On the other hand, the network component $N(\alpha)$ involves several elements. Agents migrate from one host to another; they send log files and alerts to the MIDP, and exchange messages among them and with the MIDP. Log and alert files are normally sent when attacks are suspected and therefore can be tied to the probability of detection P_d. Agent migration normally occurs after the agent spends a period of T_r on the host. Lastly, an agent will exchange M_1 messages with other agents (including its parent if it is a child) and M_2 messages with the MIDP during its stay at a host. If K_2 is the average network load consumed by the agent migration and a single connection between two agents or between the agent and the MIDP, we can then express $N(\alpha)$ by the following equation:

$$N(\alpha) = 2 * K_2 * \alpha. \quad (4)$$

If the average size of a log/alert file is S_{LAF}, the average size of the agent including its briefcase is S_{AB}, and finally, the average size of the message is S_M, then we can express N as:

$$N(\alpha, T_r) = (S_{LAF} \times P_d + S_{AB} + (M_1 + M_2) \times S_M) \times \frac{\alpha}{T_r} Mbps.$$

The constant K_1 is computed as follows: we run snort and find out the memory it consumes (determined to be 1.66 MB), then we divide by the total CPU memory (256 MB) to get $K_1 = 0.648\%$. On the other hand, to calculate K_2 we need to determine the migration bandwidth and the messaging bandwidth. The former is determined by dividing the bandwidth required to transport the agent code (4.8 Kbps) to the overall network bandwidth (100 Mbps), thus agent migration consumes $4.8Kbps/100Mbps = 0.048\%$. The agent communication part is the bandwidth consumed by a typical message (along with its attachments) divided by the network bandwidth. It was determined to be: $29.6Kbps/100Mbps = 0.0296\%$. Therefore, K_2 becomes the summation: $0.0296\% + 0.048\% = 0.0776\%$.

5.3.3 Adaptability - Agent Population

The adaptability of the system is defined as the change in agent population in response to the conditions of the network from an intrusion point of view. During attacks, the number of agents must increase to provide better monitoring of the attacks and decrease during no-attack situations. We derive expressions for two scenarios: when the attack is targeting one host and when it is targeting all hosts.

In the first scenario, the agent population is assumed to be constant at $\frac{N}{L}$ at startup time ($t = 0$). Immediately after ($t = ta$), attacks are launched at one particular host causing the number of agents to double roughly every residence time period (T_r). Consequently, the system reaches the all-snort state after a short period of time (i.e., the agent population becomes equal to the number of hosts in the network: $Na(t) = N$). Therefore the transient interval starts at $t = 0$ and ends at $t = (\frac{L}{2} - 1) * T_r$ during which the agent population is doubled every T_r. This behavior can be expressed in the following equation:

$$Na(t) = \begin{cases} \frac{N}{L}, & t = 0 \\ 2\lfloor \frac{t}{T_r} \rfloor - 1 + \frac{N}{L}, & 0 < t \leq \lceil \log_2(L) \rceil \times T_r \\ \frac{N}{L} + L - 1, & t > \lceil \log_2(L) \rceil \times T_r. \end{cases}$$

When targeting all hosts, it is assumed that continuous attacks are launched against all hosts. The time $t = 0$ corresponds to the instant when the attacks start. Afterward, the agent population increases linearly with time (every residence time interval T_r) till the point where an agent resides all the time on the victim host, after which the agent population remains constant. Mathematically, this can be represented by the following equation (Here, we assume that $L \leq 2^n$, such that: $2^{n-1} \leq L \leq 2^n$):

$$P(t) = \begin{cases} \frac{N}{L}, & t \leq 0 \\ \frac{N}{L} \times \lfloor 2^{\frac{1}{T_r}} \rfloor, & 0 < t \leq \lceil \log_2(L) \rceil \times T_r \\ N, & t > \lceil \log_2(L) \rceil \times T_r. \end{cases}$$

For every attack type in Table 3, 50 experiments were run. Each experiment corresponded to one value of T_r starting from 1 second and finished with $T_r = 50$ seconds, and in which attacks were launched continuously using *BLADE* while subjecting all hosts to simulated traffic using *WINJET*.

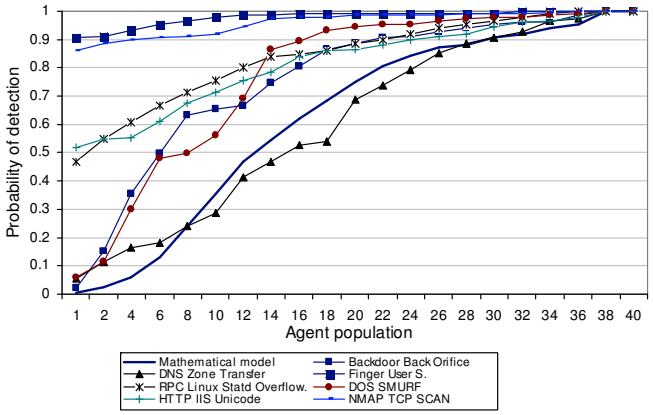

Figure 5: Probability of detection vs. agent population ($po = 0.986, Tr = 50s, N = 40$)

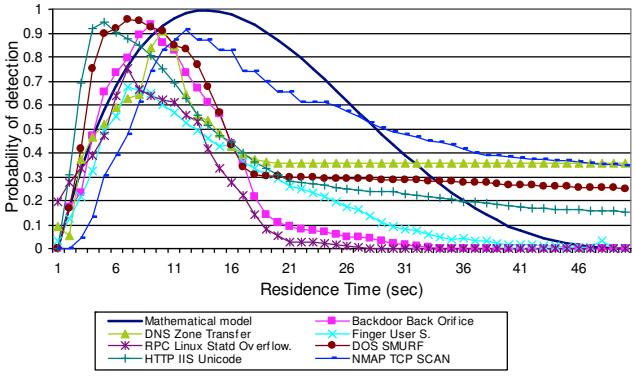

Figure 6: Probability of detection vs residence time ($\alpha = 10, N = 40$, and $po = 0.0196$)

5.4 Experimental Results

5.4.1 Probability of Detection

The detection rate is determined as follows: every attack type in Table 3 is launched continuously while targeting one host per segment, and the total number of alarms reporting the launched attack from the alert files is collected and then associated with the number of deployed agents and the residence time. This number of detected attacks is then divided by the number of actual attacks that were launched. Finally, the average detection rate is computed by summing the detection rates across all hosts and dividing by the number of hosts. For every value of α, a value of T_r is computed in accordance with Equation (1). This computed value is then used to configure the time that each agent should spend on each host under no attack situations. We produced a set of curves as shown in Figure 5, which also includes the probability of detection as calculated in Equation (2). Notice that the ideal curve produced from the mathematical model derived above is also plotted.

The residence time increases as the number of missed attacks decreases since agents will spend enough time at a host to capture attacks, thus the probability of detection increases. When the residence time increases more than necessary (after $T_r = 14$ sec. in our case), the agent will be wasting time on some hosts while attacks are on others. For this reason, the number of missed attacks starts to increase again, thus causing the probability of detection to decrease. This behavior was obtained from the performance data presented in Figure 6. The optimum T_r ranges from 6 to 13 seconds while ideally T_r was computed to be 12.755 seconds.

5.4.2 Resources Consumption

The computed average CPU time is based on the following calculations: we determine the total CPU percentage required to run threads of all existing mobile agents. Then

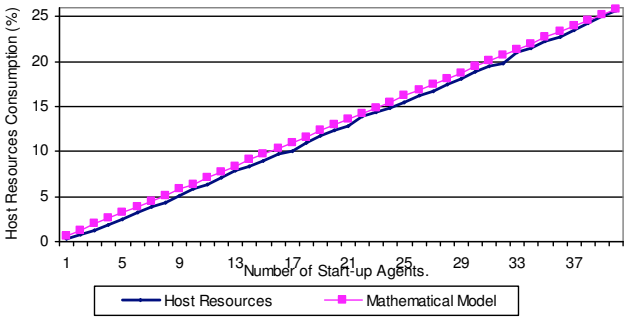

Figure 7: Host resources consumption versus number of start-up agents

we add up and divide by the total number of hosts in the network (If we have 5 agents running and each consumes 12% of the CPU time, then the average CPU time will be: $(5 * 12\%)/40 + 20\%$, where 20% is the CPU time consumed by running the MIDP at the central machine. The results are shown in Figure 7 where, on the same figure, Equation (3) is plotted.

The network resources are computed as follows: we determine the bandwidth required to transmit a single typical message from an agent to the MIDP, then divide by the bandwidth of the network and multiply by 100. The results and Equation (4) are plotted in Figure 8.

5.4.3 Adaptability - Agent Population

Targeting All Hosts: This experiment tests the hypothesis that the agent population increases linearly as a function of $\log_2(t)$. At $t = 0$ we launch continuous attacks that target all hosts. Whenever an agent clones, we record the time and the resulting number of agents; this continues until we reach the all-snort state after which the agent population remains constant (at 20). Two startup cases were tested: the first case is when every agent was initially assigned 20 hosts ($\alpha = 2$ and $N = 40$) and in the second case every agent was initially assigned 6 hosts ($\alpha = 7$ and

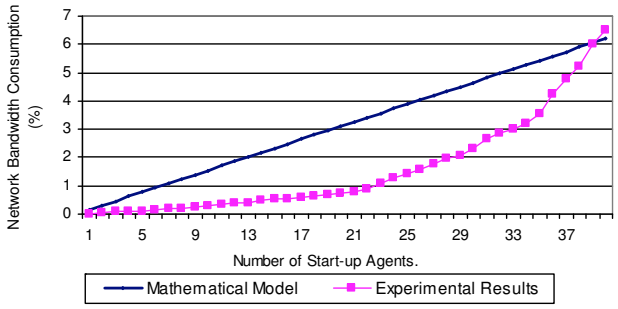

Figure 8: Network resources consumption versus number of start-up agents

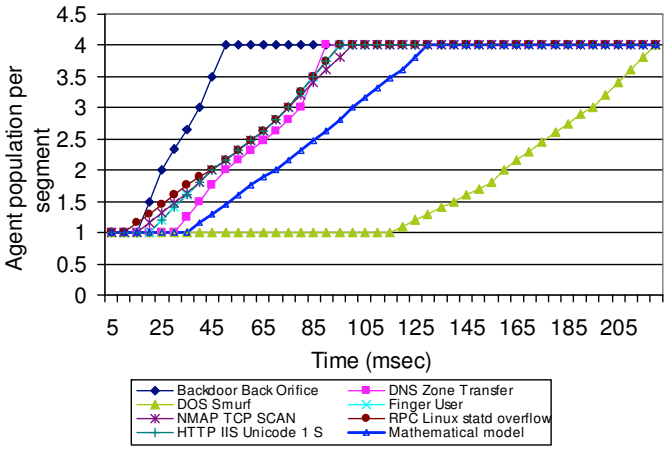

Figure 10: Agent population when one host was targeted

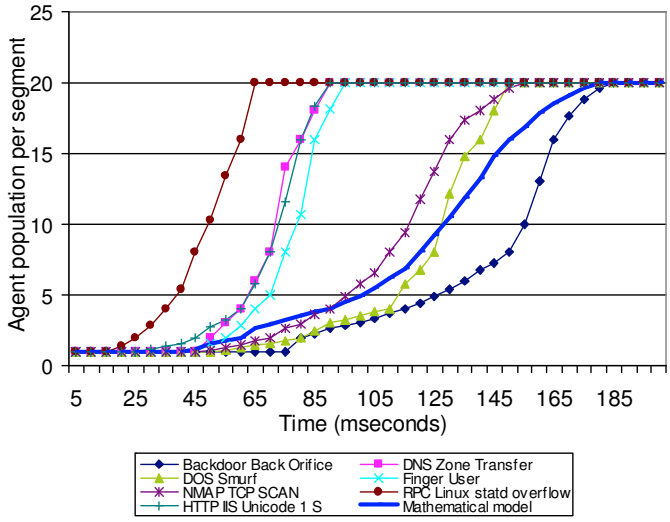

Figure 9: Agent populations vs. time (targeting all hosts): $N = 40, L = 10$

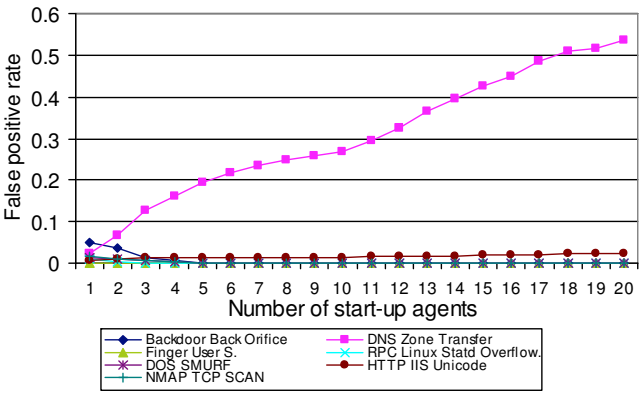

Figure 11: False positive rate versus the number of start-up agents ($T_r = 20sec.$)

$N = 40$). The plots for the two cases are shown in Figure 9 and include the ideal pattern of agent population versus time in each case.

Targeting One Host: Initially, an attack was launched against one host where $N = 40$. The residence interval T_r was 30 seconds. Whenever cloning takes place, we record the time and plot the number of agents versus time when the agent roams a segment of 20 hosts (illustrated in Figure 10).

5.4.4 False Positive Rates

The false positive rate is defined as the percentage of decisions in which normal data are flagged as anomalous. It is calculated as follows: we determine the total number of false alarms stored in the alert files collected by dispatched agents and divide by the total number of alerts recorded in the alert file. Finally, we compute the average false positive rate by summing such false positive rates at every host in the segment and divide by the number of hosts in the network. We obtain the curves in Figure 11.

We notice that all the tested attacks produced negligible false positive rates. However, the detection of DNS Zone Transfer attack results in high false positive rate. This event indicates that an outside host requested a zone transfer from an internal DNS server, which can be legitimate traffic from a secondary DNS server, or an attacker gathering information about your domain, thus making the detection rules false positive prone.

False positive rates were also plotted against the residence time. The attacks were launched manually and randomly, and the averages of 10 runs were recorded. The results are shown in Figure 12. We noticed that almost all the tested attacks produced constant false positive rates versus residence time (the exception is the Backdoor Back Orifice attack).

5.4.5 LAMAIDS versus All-snort Configuration

In this section, the performance of LAMAIDS has been compared with that of all-snort configuration. All-snort

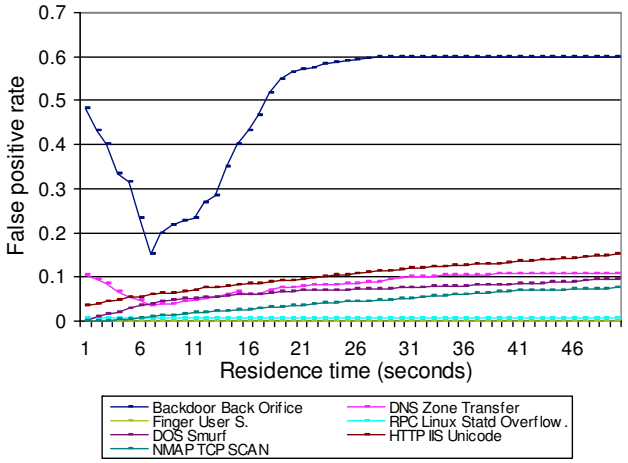

Figure 12: False positive rate versus residence time ($\alpha = 4, N = 40$)

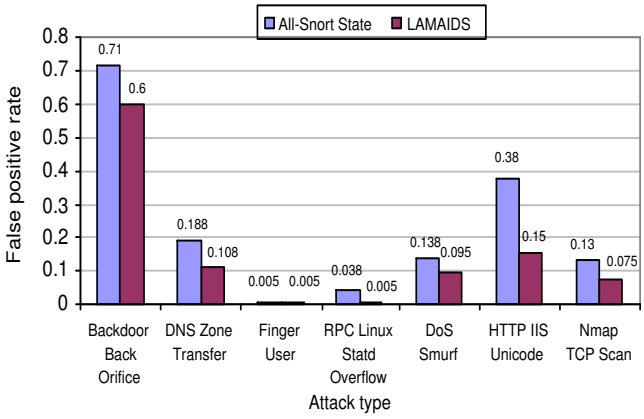

Figure 13: False positive rates: LAMAIDS vs. all-snort state ($\alpha = 2, N = 20, Tr = 50sec$)

configuration means that snort exists on every host in the network and is running at all times. We focus our comparison on three criteria: the detection ratio, the false positive ratio, and the CPU time consumption.

From Figure 13, we can see that the false positive ratio of our adaptive system is significantly lower compared to that of the all-snort system. Adaptive mechanisms used by the agents can change normal profiles correspondingly, enabling our IDS to suit the environment better. False positives can be reduced correspondingly. If an agent detects an attack, it will report it to the MIDP, which in turn informs neighbor agents about the specific attack. Therefore, agents will be focused on specific patterns and can expect the type of attacks taking place in real time.

We observe from Figure 14 that the detection ratio of LAMAIDS is slightly less than that of the all-snort configuration. This is due to the fact that while agents are snorting some hosts, other hosts are being susceptible to

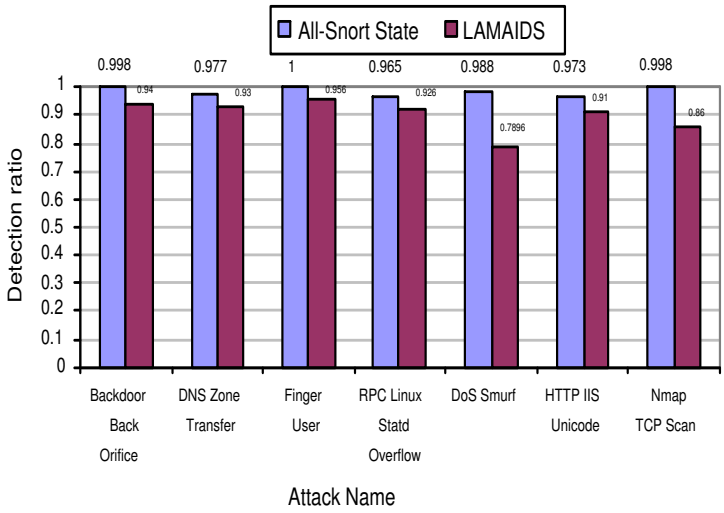

Figure 14: Detection rates: LAMAIDS vs. all-snort ($\alpha = 2, N = 20, Tr = 50sec$)

attacks and left unprotected. Therefore, LAMAIDS will miss some of the launched attacks. We observe that attacks requiring shorter running time are more likely to be detected by LAMAIDS than attacks with longer running time. This is due to the effect of the residence time of the agent at any host.

From the two graphs in figures 13 and 14, we get the detection-to-false-positive ratio as a function of the attack types. We realize that this ratio is higher in LAMAIDS than the all-snort configuration for almost all the attacks. Therefore, it is clear that the gain we obtained from false positive improvements is much larger than the loss due to detection ratios.

Concerning the network resources consumption, LAMAIDS outperforms the all-snort configuration since it does not necessitate running snort all the times but on a need basis. Knowing that a main challenge of existing intrusion detection systems is to decrease the false positive rates, the main benefit of our adaptive system is to lower the false positive ratio and network resources consumption, while maintaining a good detection rate.

6 Conclusion

We presented an architecture for distributed intrusion detection and defense system based on mobile agents that detects intrusions from outside and inside a network segment. The system is shown to be efficient, robust, and flexible. The system potentially reduces the massive amount of distributed log data moved among the inner nodes of a conventional IDS. Having mobile IDS agents visit hosts and doing intrusion detection locally is well suited to the ability of mobile agents to move the computation to the data, thus reducing network load. Fur-

thermore, the developed architecture implements robust and attack-resistant IDS (inherited from agent mobility). There is no single vulnerable point of failure. As agents are less susceptible to direct attacks, can clone for redundancy or replacement, and operate independently and autonomously from where created.

References

[1] J. S. Balasubramaniyan, J. O. G. Fernandez, D. Isaco, E. Spafford, and D. Zamboni, "An architecture for intrusion detection using autonomous agents," in *Proceedings of the 14th IEEE Computer Security Applications Conference (ACSAC '98)*, pp. 13-24, Dec. 1998.

[2] J. Balasubramaniyan, J. G. Fernandez, D. Isacoff, E. Spafford, and D. Zamboniy, "An architecture for intrusion detection using autonomous agents," in *Proceedings of the 14th Annual Computer Security Applications Conference*, pp. 13-24, 1998.

[3] J. Barrus and N. Rowe, "A distributed autonomous-agent network-intrusion detection and response system," in *Proceedings of the 1998 Command and Control Research and Technology Symposium*, pp. 577-586, 1998.

[4] M. Bernardes and E. Moreira, "Implementation of an intrusion detection system based on mobile agents," in *Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems*, pp. 158-164, 2000.

[5] CERT Coordination Center, *CERT/CC Statistics for 1988 through 2004*. (http://www.cert.org/stats/cert_stats.html#incidents)

[6] M. Crosbie and G. Spafford, *Active Defense of a Computer System Using Autonomous Agents*, Technical Report 95-008, pp. 47907-1398, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, Feb. 1995.

[7] M. Crosbie and G. Spafford, *Active Defense of a Computer System Using Autonomous Agents*, Technical Report 95-008, pp. 47907-1398, COAST Group, Department of Computer Sciences, Purdue University, West Lafayette, Feb. 1995.

[8] M. Crosbie and E. Spafford, "Defending a computer system using autonomous agents," in *Proceedings of the 18th National Information Systems Security Conference*, Oct. 1995.

[9] CSI/FBI, *Issues and Trends: 1999 CSI/FBI Computer Crime and Security Survey, 1999*, Aug. 30 2004. (http://www.gocsi.com)

[10] S. Fenet and S. Hassas, "A distributed intrusion detection and response system based on mobile autonomous agents using social insects communication paradigm," *Electronic Notes in Theoretical Computer Science*, vol. 63, pp. 43-60, 2001.

[11] L. Guangchun, L. Xianliang, L. Jiong, and Z. Jun, "MADIDS: A novel distributed IDS based on mobile agent," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 1, pp. 46-53, Jan. 2003.

[12] G. Helmer, J. Wong, V. Honavar, L. Miller, and Y. Wang, "Lightweight agents for intrusion detection," *Journal of Systems and Software*, vol 67, no 2, pp. 109-122, Aug. 15, 2003.

[13] J. Hochberg, K. Jackson, C. Stallings, J. F. McClary, D. DuBois, and J. Ford, "NADIR: An automated system for detecting network intrusion and misuse," *Computers & Security*, vol. 12, no. 3, pp. 235-248, May 1993.

[14] W. Hunteman, "Automated information system-(AIS) alarm system," in *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, pp. 394-405, 1997.

[15] J. Koza, *Genetic Programming: On the Programming of Computer by means of Natural Selection*, MIT Press, 1992.

[16] D. Lange and M. Oshima, *Programming and Deploying Java Mobile: Agents with Aglets*, Addison-Wesley, 1998.

[17] *Objectspace Voyager Core Package Version 1.0 Technical Overview*, 1997. (http://www.objectspace.com/voyager/whitepapers/VoyagerTechOview.pdf)

[18] P. A. Porras and P. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proceedings of the 20th National Information System Security Conference*, pp. 353-365, 1997.

[19] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th Systems Administration Conference-LISA '99*, pp. 229-238, USENIX, Nov. 1999.

[20] M. Roesch, *Snort - Lightweight Intrusion Detection for Networks*, A white paper on the design features of Snort 2.0, 2004. (http://www.sourcefire.com /technology/whitepapers.html)

[21] Smashing the Stack for Fun and Profit, Aleph1, Phrack #49, 2004. (http://www.phrack.com)

[22] A. Smith, "An examination of an intrusion detection architecture for wireless ad hoc networks," in *Proceedings of the 5th National Colloquium for Information System Security Education*, May 2001.

[23] The Snort Intrusion Detection System, Mar.15 2004. (http://www.snort.org)

[24] E. Spafford and D. Zamboniy, "Intrusion detection using autonomous agents," *Computer Networks*, vol. 34, no. 4, pp, 547-570, Oct. 2000.

[25] G. White, E. Fisch, and U. Pooch, "Cooperating security managers: A peer-based intrusion detection system," *IEEE Network Magazine*, vol. 10, no. 1, pp. 20-23, 1996.

[26] G. White, E. Fisch, and U. Pooch, "Cooperative security managers: A peer-based intrusion detection system," *IEEE Network Magazine*, vol. 10, no. 1, pp. 20-23, Jan. 1996.

[27] R. Zhang, D. Qian, C. Ba, W. Wu, and X. Guo, "Multi-agent based intrusion detection architecture," in *Proceedings of the IEEE International Conference on Computer Networks and Mobile Computing*, pp. 494-504, 2001.

Mohamad Eid is currently a PhD student in the Multimedia and Communication Research Lab (MCRLab) at the University of Ottawa, where he is doing research in the field of haptic technologies and applications. He received his Bachelor of Engineering in Electronics and Communications from Beirut Arab University (BAU) in 2002. He received and the Master of Engineering in Computer and Communication Engineering from the American University of Beirut (AUB) in 2005.

Hassan Artail Worked as a system development supervisor at the Scientific Labs of DaimlerChrysler, Michigan before joining AUB in 2001. At DaimlerChrysler, he worked for 11 years in the field of software and system development for vehicle testing applications, covering the areas of instrument control, computer networking, distributed computing, data acquisition, and data processing. He obtained a B.S. and M.S. in Electrical Engineering from the University of Detroit in 1985 and 1986 respectively and a Ph.D. from Wayne State University in 1999. His research is in the areas of Internet and mobile computing, distributed computing and systems, and computer plus network security.

Ayman I. Kayssi was born in Lebanon in 1967. He received his BE with distinction in 1987 from the American University of Beirut, Lebanon and the MSE in 1989 and PhD in 1993 from the University of Michigan, Ann Arbor, USA, all in electrical engineering. He is currently professor and chairman of electrical and computer engineering at the American University of Beirut, where he has been working since 1993. His research and teaching interests are in the areas of internet engineering, wireless networking, CAD for VLSI, modeling and simulation.

Ali Chehab received his Bachelor degree in EE from the American University of Beirut (AUB) in 1987, the Master's degree in EE from Syracuse University, and the PhD degree in ECE from the University of North Carolina at Charlotte, in 2002. From 1989 to 1998, he was a lecturer in the ECE Department at AUB. He rejoined the ECE Department at AUB as an assistant professor in 2002. His research interests are VLSI design and test, mobile agents, and wireless security.