IDReAM: Intrusion Detection and Response executed with Agent Mobility Architecture and Implementation

Noria Foukia
Information Sciences Institute (USC)
Network Systems Laboratory
foukia@isi.edu

ABSTRACT

This paper deals with a new approach to build a completely distributed and decentralized Intrusion Detection and Response System (*IDRS*) in computer networks. This approach is called Intrusion Detection and Response executed with Agent Mobility or *IDReAM* for short. Conceptually, IDReAM combines Mobile Agents (*MAs*) with self-organizing paradigms inspired by natural life systems. The Intrusion Detection System (*IDS*) borrows mechanisms from the immune system that protect the human body against external aggressions. The Intrusion Response System (*IRS*) borrows mechanisms from the stigmergic paradigm of a colony of ants. The two natural systems exhibit a social life by the organization of their entities (immune cells and ants) which is not possible without the functionality of mobility. Thus, in a natural way, MAs are good candidates to provide this property of mobility. After having presented IDReAM's conceptual model in a previous paper, the present paper concretely describes IDReAM's architecture and the corresponding implementation based on the conceptual model. The implementation is carried out with J-Seal2, a pure Java MA platform. This paper also provides IDReAM's assessment in term of resource consumption and intrusion detection and intrusion response efficiency.

Categories and Subject Descriptors

C.2.4 - D.4.6 - D.4.7 - K.6.5

General Terms

Security

Keywords

Mobile Agents, Intrusion Detection and Response

1. INTRODUCTION AND MOTIVATION

The advent of large-scale open networks and the recent emergence of distributed systems enable cooperative entities to perform intrusions in a more effective scenario, targeting the attack on several machines through several links. This trend in distributed intrusions led to the consideration some adaptation of the *Intrusion Detection Systems* (IDSs) to circumvent new intrusive behaviors and to adapt them to the new network orientation because in such new context traditional security mechanisms demonstrated severe weaknesses. One of the principal weaknesses of the protection systems is the lack of robustness inherent in their centralized nature. Even though most of the existing IDSs use distributed data collection (host-based or network-based), many of them continue to perform data analysis centrally, thereby limiting scalability. Moreover, even if the IDS is distributed in the network, its deployed elements generally remain static. With the means available to modern attackers, such as automated intrusion tools, these static elements are easily accessible, making the protection system itself an easy target of attacks. Unfortunately, system administrators cannot always react to frequent intrusion alerts within adequate time limits. Therefore, the need of efficient response tools is also crucial; however, the majority of IDSs that also deal with response possess static response components, insufficient to respond to distant attacks. Moreover, frequently the responses provided with current Intrusion Response Systems (*IRSs*) exhibit a lack of flexibility which can have a negative effect on the system to be protected. An example given in [15], is the changes of firewall rules which do not only defend against the detected attack but may also block legitimate user access or stop legitimate services. However, to date, only a few investigations have been undertaken to develop dynamic and flexible response systems. Thus automatic IRSs have to take over this task.

Obviously, many issues should still be solved when dealing with Intrusion Detection (*ID*) and *Intrusion Response* (IR); we are trying to use new techniques and solutions to solve some of them. Notably, we propose a new *Intrusion Detection and Response System* (IDRS) called *Intrusion Detection and Response executed with Agent Mobility* or *IDReAM* for short. The *Mobile Agents* (MAs) are envisioned as a solution to build a distributed IDRS with a high dynamics. To our understanding, a MA is a piece of software that is first autonomous because it exercises control over its own actions. It owns its own execution code and some resources such as its own data. It can move during its execution to use resources located at another place of the network or to execute at a remote location. IDReAM is embodied in a population of MAs[1] in perpetual movement whose furtiveness and dynamics make it more reliable. Less occupied by its own protection, the IDRS is more available to protect the network. In fact, several other motivations led us to

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

[1]IDReAM is implemented with J-Seal 2 which allows weak mobility only. See section 3.1

use MAs. Among them are the fact that MAs impose a relatively low resource overhead in the host and the hope that these small software entities will be able to execute in near real-time, which is highly desirable for ID and IR. Also, their dispersion through the network makes the IDRS as a whole more resistant to attacks and to automated tools. Regarding the response task, MAs as autonomous entities are good candidates to perform automatic responses. But MAs don't have a global vision of the network activity each time; hence ID and IR could be more difficult. Each MA is responsible for sensoring, independently from the others MAs, the part of the network that it is currently visiting. The cooperation between agents is inspired by the cooperative behaviors found in natural life systems. More precisely it is inspired by the mechanisms of self-organization because, one way to act autonomously while maintaining a global coordinated task is what self-organizing natural systems achieve.

The next section contains related works to agent-based approach for ID. Section 3 describes IDReAM's architecture and implementation. Section 4 evaluates IDReAM in term of resource consumption and protection system. Section 5 draws the conclusion.

2. RELATED WORKS

Several research works have been achieved in IDRSs. To date, only a few investigations have been undertaken to develop MA-based IDRSs even though MA technology seems to exhibit good properties for that: light-weight entities can react rapidly; moving entities can react at different locations. Wayne Jansen [9] followed by Stefano Martino [10], acted as a precursor to devote a population of MAs to the system's protection. In [9], he explored new research ways and new paradigms to use MA technology to enhance existing IDRSs. (AAFID) [14] proposed in 1998 an agent-based approach where the key idea is to involve many small co-operating autonomous agents[2] whose responsibility is hierarchically decomposed instead of a single large IDS. Each agent, dispatched to a host, monitors a small aspect (thus, a small quantity) of the traffic and collaborates with the other agents to detect intrusion: if an agent believes that there is a suspicious activity, it notifies the other agents in the hierarchy about this by sending a suspicion broadcast. Each broadcast about the same possible intrusion raises a suspicion level. After several suspicion broadcasts about the same possible intrusion, the suspicion level can exceed a pre-defined threshold and engage an alert. IDReAM is basically and conceptually different from AAFID because it does not have any hierarchy. In IDReAM, all MAs are at the same level and they notify other MAs about a suspicious activity by depositing in the nodes they visit any information related to this activity. This is made possible thanks to the mobility which is not present in AAFID. More recently, [13] proposes a new version of AAFID (namely AAFID 3) by adding mobility; however, the heavy hierarchical structure is maintained, which is a weakness because if one function of the hierarchy is compromised, the entire IDS could be compromised. IDReAM avoids overly strong hierarchical operations. Each operation should stay simple and each MA is rather specific to one type of detection or response. The MAs collaborate through the environment to accomplish the global protection task.

Another approach called *MICAEL* [11] extended the idea of Crosbie and Spafford to use autonomous agents for ID, by adding mobility. Actually, the majority of the agents are rather static during a normal processing of the IDS and the mobility is mainly used to periodically check the integrity of the other active static agents; a *Headquarter* centralizes and controls all the agents and can escape from its location if the hosting resources decrease or if the host is infested. Moreover, even if the data analysis is distributed among the static agents located at different hosts, the correlation between the different analyses also seems to be centralized on the Headquarter. Like IDReAM, MICAEL tries to prevent the protection system from being itself the target of an intrusion. In both cases, MAs are used for that. However, in IDReAM the MAs are in perpetual movement which increases their furtiveness whereas in MICAEL the mobility is used only when it is specifically required to check the agent's integrity.

A recent interesting approach proposed in [12] is quite close to ours because it also uses collaborating MAs for providing the ID and IR. One of the major issues of [12] is to avoid any central coordination and static hierarchical architecture. This IDS uses a peer-to-peer solution where MAs periodically dispatched from each site look for suspicious activities in their neighborhood and report back. But at each site the analysis is accomplished by a static agent called *Detective* that coordinates the MAs and correlates the different reports received from the previously sent MAs. In IDReAM, every dispatched MA walks randomly between the different nodes and the correlation between the collected information is due to the deposits made by previous MAs in the current node. In [12], the response task is based on an auction mechanism where each neighbor of the suspected site takes part in deciding if the action should be handled or not. This auction mechanism seems particularly heavy and could slow the response process.

[1] developed a method to trace intruders using MAs. MAs autonomously migrate to target systems to collect only information related to intrusions, eliminating the need to transfer useless system logs to the analyzer server. In each network segment, a manager dispatches a tracing MA to the target system which activates an information-gathering MA. This latter MA collects evidence about the intrusion on the target system and returns to the manager, whereas the first tracing MA migrates to another site in an effort to trace the path of the intrusion and identify its point of origin. The major difference with IDReAM is the existence of one manager per network segment which analyzes reports made by the information MAs in order to make a decision. This also constitutes a major weakness because the decision about potential intrusion is centralized. One similarity with IDReAM is the information left by each tracing agent on a so-called message board on each target system. The current tracing agent refers to this message board to see if its trace has already been followed by another tracing agent and to decide where to go.

In [4], Serge Fenet studied the social insect foraging behavior for distributed applications. He took as a case study the ID application and he also investigated how to use MAs to detect and respond to attacks. His approach is quite similar to ours since the behavior of MAs is also inspired by self-organizing ants. In [4] the MAs also refer to pheromonal information centralized in a so-called pheromone server[3]. This information is not diffused, and here again the weakness of the approach is the pheromone server the MAs have to refer to in order to get intrusion information. This approach is not totally distributed even if MAs are completely dispatched in the network.

Apart from these agent-based approaches, we also relate to the Cooperative Security Managers (CSM) [16] because rather early its creators proposed a completely distributed architecture based on

[2] AAFID is based on Crosbie and Spafford's work [3] who previously defined an autonomous agent as a lightweight program that observes only one small aspect of the overall system.

[3] A pheromone server plays the role of a communication interface between MAs and their hosts. The pheromone server gathers all the information deposited locally by MAs on the host they visited.

a cooperation model between the hosts housing local IDSs. The collaboration is done through direct exchange of information about users moving through the network. This allows the tracing of the users; however, CSM does not use any agent-based approach and does not take advantage of the mobility.

3. ARCHITECTURE AND IMPLEMENTATION

To summarize, IDReAM's conceptual model, described in [6] and [5], borrows some functionalities that are found in natural life. More precisely, the IDS is inspired by the behavior of the human immune system and the IRS is inspired by the behavior of a colony of foraging ants. Schematically, there are two main populations of MAs:
1) The **Intrusion Detection Agents** (*IDAs*) which borrow mechanisms from the immune cells. To detect attacks happening in the network, IDAs have to be able to discriminate between normal and abnormal activity. In the immune system it is done by T cells distinguishing good proteins named "selfs" from bad ones named "non-selfs". In the IDS the normal self activity is viewed as a good sequence of events, whereas the abnormal non-self activity is viewed as a bad sequence of events. IDAs move randomly in the network they protect like T cells in the body. When entering hosts, IDAs check for bad sequences of events as T cells check for non-self sequences. Finally, IDAs report any suspicious activity by diffusing a message through the network as T cells diffuse the sign of a disease through the glandular system and the lymphoid organs.
2) The **Intrusion Response Agents** (*IRAs*) whose task is to respond to detected intrusions. For that, they must locate the place where the alert was given by the IDAs and go there. To trace the source of the alert they mimic the mechanism employed by the foraging ants to trace the source of food. Ants use the chemical pheromone deposited by the other ants in the environment to trace the source of food. The IRAs use an *electronic version of the pheromone* which indicates the route to the infested node. This pheromone is built by an IDA when it detects an intrusion and is randomly diffused from the infested node through different nodes of the network. Moreover, like in ants' colonies, an evaporation process and an inhibition process of the electronic pheromone limit the number of IRAs following the same pheromonal trails if a response has already been provided. IRAs could also move directly to the source of the alert if the information was provided in the pheromone. But we did not insert the source address because it could be easily used by an intruder to build a fake pheromone. It is more difficult for an intruder to reproduce the entire pheromonal path with consistent pheromonal fields in each deposit.

The current prototype of IDReAM is composed of a set of MAs and services that allow us to carry out specific ID and IR tasks on the network. In IDReAM's architecture, each node has a MA platform, called *J-Seal2* [2], running on it. The platform hosts the IDAs, the IRAs and the needed services in protection domains or *seals* and provides resource control. Apart from the platform, the node also houses a *monitor* that provides the administrator with an interactive access to the IDRS through a graphic interface. This allows the administrator to consult the current agents' activity and to send instructions to the agents, such as an order to terminate. The pheromones and the different profiles are stored locally as well as in a database for administrative operations. For each J-Seal2 platform, there are two seals called *AgentHosts*: one houses and manages the IDAs and their services and the other houses and manages the IRAs and their services (Figure 1). The roles and functionalities of the

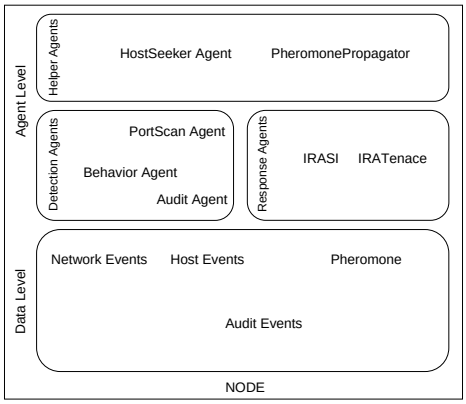

Figure 1: Mobile Agent Architecture.

different types of MAs are taken from the conceptual model and have been implemented:
1) An IDA follows the sequence of actions below:

- It moves randomly. When entering a machine it probes the incoming events and computes the *Suspicion Index* (SI), taking into account the information deposited by other IDAs and the information it previously memorized.
- If SI $\geq$ threshold then it launches an alert, builds and diffuses a pheromone.
- Then it continues its random walk to compute a new deviation.
- When returning to the AgentHost that created it, it updates the SI in a non-self profile of prohibited events.

Different populations of IDAs have been implemented:
The *AuditIDA* audits logs using an internal state machine for detecting a signature of intrusion. Indeed, a set of states are interconnected by possible actions and only some are "final", i.e. set off an alert. The incoming log events determine which state is considered at any given moment.
The *BehaviorIDA* compares a sequence of traced system calls against a clean profile. The current implementation uses the Hamming distance and the r-continuous bit distance.
The *PortScanIDA* is an implementation for detecting port scans. It listens in raw mode on the TCP sockets of the host. In order to limit resource use it obtains copies of such packets on all ports for a limited time only.
2) An IRA follows the sequence of actions below:

- In its normal quiet state, it moves randomly, searching for pheromonal information in the network.
- If it finds a pheromone, it switches to a tracking state and follows the pheromone back to its source. Following the trail, it speeds the pheromone's evaporation as explained in [7] on each node belonging to the current trail.
- As soon as it reaches the source of the alert, it initiates a response to the original attack and switches to the quiet state. Then it continues its random walk to discover another pheromone.

Different IRAs have been implemented, executing different tracing strategies. An *IRATenace* always follows the same pheromone until the source of the alert, whereas an *IRASI* can switch to another trail if it crosses a pheromone with a higher SI.
3) Other helper agents and services needed for the realization of the conceptual model are described below:
The *SimpleHostSeeker*: its provides the IDA (or the IRA) with an asynchronous way to obtain a travel destination. While the IDA (or the IRA) is operating on the current host, its specially created *SimpleHostSeeker* selects the next neighbor to visit according to the moving strategy and reports back to the IDA (or the IRA). Using the information provided by the *SimpleHostSeeker* the IDA (or the IRA) decides autonomously to choose preferably one direction. Each *AgentHost* maintains a table of the agent visit frequencies which is used by the *AgentHost* to control the different populations of agents.
The *SimplePheromonePropagator*: an IDA which detects an intrusion builds the corresponding pheromone to diffuse the alert through its neighborhood. The pheromone diffusion required the implementation of the *SimplePheromonePropagator*. This MA deposits the pheromone in each host of a randomly chosen path of the network. On each host, the intensity of the deposit is governed by an exponential function of $-t$, where t is the time since the pheromone's creation. This latter function was chosen because it satisfies the monotonous decreasing requirement of the pheromonal gradient of the pheromone randomly deposited by the *SimplePheromonePropagator*. This will allow the IRAs to go up the pheromonal gradient in the opposite direction.
The *PheromoneStorage*: When the pheromone is found by an IRA, this service modifies the pheromone in the local *Pheromone* "conveyor". This service decreases the pheromonal gradient in case of evaporation and destroys the pheromone in case of inhibition.

3.1 Implementation Specificity

A set of packages and classes for the detection and response tasks has been implemented. A simple description of the main packages and classes is given below:
1) Package ***MobileHost***
This subsystem presents a thin service layer that allows weak autonomy in the agents' movements and inter-agent communication through method calls. This is mainly implemented here by two classes, *AgentHost* and *MobileAgent*.
AgentHost is the topmost seal instantiated as an agent, an instance of this class is created by the framework at start-up. *AgentHost* uses three sub-classes for the creation and reception of the agents:

1. *Initializer* is launched at the start of *AgentHost*. *Initializer* reads in a configuration file the number of IDAs or IRAs to create, depending on the *AgentHost* and creates them using *MobileAgentStarter*. It stops immediately thereafter.

2. *MobileAgentStarter* creates and activates all agents.

3. *NetworkListener*, launched in parallel to *Initializer* is permanently listening to the network in order to receive and unwrap the incoming agents.

MobileAgent is a common abstraction of all the *IDA* and the *IRA* derived classes. *MobileAgent* receives all the necessary elements for its creation from the *AgentHost* class, most notably its name. J-Seal2 provides synchronous communication via channels and also an asynchronous *send* method for asynchronous communication in a default *net* service. This method is rather rudimentary; therefore, *MobileAgent* provides an implementation of the *moveTo* method which enhances the autonomy of a MA. The *send* method does not allow an agent to decide when it moves whereas the *moveTo* allows an agent to do it. Moreover, *moveTo* accepts the name of a seal as parameter whereas *send* accepts only the content of a channel called a *capsule*. Diagram 2 shows how the various objects interact to allow agents to semi-autonomously move around network hosts. Diagram 3 explains the inter-agent communication (between an *IDA* and its *SimpleHostSeeker*), required by the collaborative conceptual model.

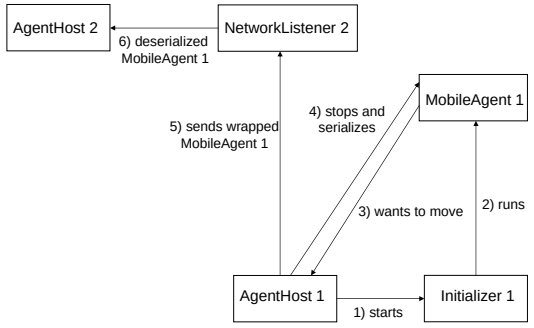

Figure 2: Agent Moving Mechanism.

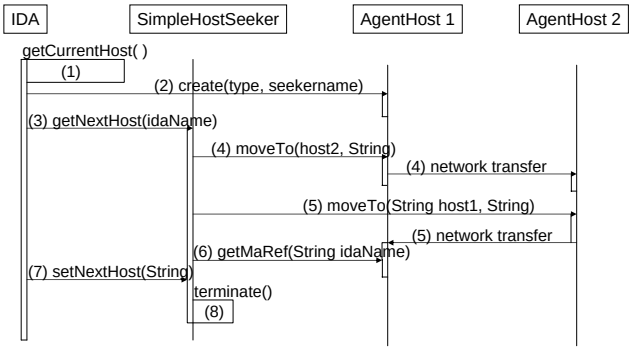

Figure 3: Inter-Agent Communication.

2) Package ***IDSHost***
This package provides the implementation of the various services provided by stationary agents related to ID and to IR. These services mainly include the storage of the pheromone, administrative alerts, system logs that IDAs feed, as well as a network topology informational service. For instance, the *Neighborhood* service, initiated in the start-up provides the local view of the network to all agents.

3.2 Mobile Agent Security Implementation Issues

A major issue of MA-based systems is the protection of the MAs and the platform from intruders who can corrupt them. This has to be absolutely discarded in IDReAM because this would definitively compromise the protection system. The MA authentication and the MA encryption have been implemented and integrated into IDReAM, ensuring a secure transfer of MAs between the hosts.

1) **MAs' Authentication**

The authentication is based on a certification scheme using a combined RSA/MD5 cipher scheme. This authentication ensures that the agent has been created by a trusted *AgentHost* that owns a certificate from the Certification Authority (CA) called *UnigeCA* [8]:

1. Each *AgentHost* receives a certificate from the CA and a key pair sent in a PKCS#12 file. This file also contains the CA self-signed root certificate used by each intermediate *AgentHost* to authenticate its personal certificate as being generated and signed by UnigeCA.

2. Each agent MA_1 has a copy C_1 of the certificate of its original *AgentHost* (AH_1) which created it. This copy plays the role of (MA_1) source certificate.

3. AH_1 signs MA_1 with its private key and sends MA_1, previously encrypted, the signature ($Sign_1$), plus C_1 signed by UnigeCA. MA_1 has to deliver C_1 to each intermediate *AgentHost* that wants to transfer it.

4. The receiver *AgentHost* (AH_2) verifies that C_1 has been generated and signed by UnigeCA and verifies AH_1's signature on MA_1. If all the verifications are valid, MA_1 is executed; otherwise MA_1 is destroyed.

5. Before MA_1 is transferred to the next destination, AH_2 wraps MA_1 and signs the wrapped MA_1 plus the original certificate C_1 together with its private key and adds its certificate C_2. Then AH_2 sends the whole to AH_3. This ensures that at any moment of MA_1's life, its creator (AH_1) can be identified.

6. AH_3 receives the packets and verifies C_2 with UnigeCA root certificate. It verifies AH_2's signature ($Sign_2$) with the public key included in C_2. If $Sign_2$ or C_2 are not valid, MA_1 is destroyed. Otherwise, C_1 is verified with UnigeCA root certificate. If C_1 is valid, MA_1 is unwrapped and executed.

Step 5 and step 6 are repeated at each new transfer. An intruder who wants to inject a bad MA requires an original *AgentHost* certificate signed by UnigeCA because it is verified by each intermediate *AgentHost*. Moreover, each certificate is stored locally in a protected PKCS#12 file and the administrator should ensure that the certificates are in order before starting the MAs.

2) **MAs' Encryption**

The encryption of the agents uses 3-DES. The wrapped MA, its signature and the certificate are encrypted with a symmetric key shared by the *AgentHosts*. A first step negotiation between AH_1 and AH_2 allows the sharing of the common symmetric key generated for AH_1 stored in a secure location and also protected in a PKCS#12 format; AH_1 recovers AH_2's certificate and encrypts the symmetric key with AH_2's public key with RSA and sends it to AH_2.

4. APPROACH'S ASSESSMENT

The port scan demonstration scenario used for IDReAM is detailed in [6]. This section presents IDReAM's assessment in terms of resource consumption and discusses its scalability. It also observes IDReAM's reaction when facing an increasing number of port scans and the effect of the pheromonal information on the IRAs' behavior according to the moving strategy.

4.1 Mobile Agents' Performance

First, the CPU overhead introduced by the main agents' classes has been measured in a machine housing IDReAM, named Pctelecom69 which has the following charateristics: Pentium 2 - 400MHz - RAM 128 MB - Disk 4 GB - Linux Mandrake 9.0 - Java version 2.

A special MA called *SyntheticIDA* has been configured so that it is greedier in resource than a classical *PortScanIDA*. In comparison with *PortScanIDA*, it does not sleep between two consecutive probes of the TCP/IP packets' arrival, but it spends its time computing the successive dividers of a big natural number. Table 1 shows the effect of increasing the number of *SyntheticIDAs* on PCtelecom69: the average percentage of *SyntheticIDAs* which succeed in executing their computing task when their number increases. The measure is repeated 20 times for each population of *SyntheticIDAs* running for 20 minutes on Pctelecom69. When the population of *SyntheticIDAs* increases, we also increase the time allotted to carry out their work. Table 2 gives the different sizes in bytes of a sample of MAs chosen among the biggest ones. This size has been obtained from the serialization of the wrapped agent in Pctelecom69 sent to another machine when calling the *moveTo* method. Table 3 gives the CPU consumption of the population of *SyntheticIDAs* of the Table 1 when the JVM has been used in profiling mode. The result expresses the consumption of all the processes relating to the *SyntheticIDAs* compared to all the processes of the JVM. This measure is an average of 20 runnings of 20 minutes each performed in

Number of *SyntheticIDAs*	Success in %
1	95.21
20	72.22
60	48.40
80	45.99

Table 1: Scale's Effect of an IDAs' Population.

Name of the Agent	Size in Bytes
PortScanIDA	39226
SyntheticIDA	16329
IRATenace	23291
IRASI	23875

Table 2: Agents' Size in Bytes.

Number of SyntheticIDAs	% of CPU
1	-
20	4.97
60	18.51
80	50.25

Table 3: CPU Consumption of *SyntheticIDAs*.

Pctelecom69.
Comparing Table 1 with Table 3, it seems that when the number of greedy MAs managed by a single *AgentHost* increases, the success rate decreases because the CPU allocated by the JVM and by J-Seal2 is controlled. In fact, the detail of the profiling mode, not presented here, shows that the majority of the resources are consumed by J-Seal2 micro-kernel, the *AgentHost* and the first created MAs. The following MAs are attributed a smaller part of the CPU. Under 20 MAs, the consumption is negligible and does not appear during the profiling. Moreover, Table 2 shows that the bandwidth consumed by the MAs is not very high even for the biggest MAs. All these measures lead us to conclude that the issue of scalability of IDReAM depends mainly on the number of MAs currently present a the same location because they are managed by the same *AgentHost*. Since we optimized the code, we think that this is mainly due to the way J-Seal2 manages the resources consumed by the seals. However, in IDReAM the MAs are completely distributed and furtive, with a small chance to have 20 MAs simultaneously at the same host; this can help to make IDReAM more scalable.

4.2 Detection and Response Performance

In [6], an IBM mainframe has been used for the port scan scenario which runs 15 Linux virtual machines. These machines have been configured such that the logical network of neighbor nodes is a meshed network. In the following, IDReAM is observed in the presence of an increasing number of intrusions generating many pheromonal traces.

1) **Average Detection Time**

Table 4 shows the average time needed by a population of *PortscanIDAs* between two alerts when a distributed port scan is remotely targeted to the 15 virtual machines of the IBM mainframe, from 3 real machines configured like Pctelecom69. Nmap is used as scanner on each of the 3 real machines. Nmap has been configured to scan all the 15 machines with a frequency of 6 seconds (center) between two scans and also with a frequency of 1 second (right) between two scans in an intensive port scan scenario. From this

PortScanIDAs	Time in Seconds	Time in Seconds
1	21.23	5.32
2	14.93	4.4
5	12.62	1.87

Table 4: *PortScanIDAs* Average Time Between 2 Alerts with Nmap (6s) and (1s).

second series of measures, obviously the *PortScanIDAs* are more stressed with an intensive port scan. However, this test shows the positive effect of increasing the population of *PortScanIDAs* which decreases the average time between two alerts, meaning that the whole population detects more scans since the frequency of Nmap is maintained fixed during the test.

2) **Average Response Time**

Table 5 gives the average response time of a population of *IRATenaces* and a population of *IRASIs* according to the number of pheromonal traces. The different traces of pheromones have been obtained from a population of *PortScansIDAs*. The pheromone has been diffused at a distance of 3 hops in the same meshed network of 15 virtual machines. The measure with each IRA population has been repeated 20 times, replacing at each time the pheromonal traces at the same locations. The time is started when the first pheromone is discovered and is stopped when all the pheromonal traces have evaporated.

IRAs	Traces	*IRATenace* Time	*IRASI* Time
1	1	2.48	2.32
5	5	2.86	9.18
10	10	2.91	20.8

Table 5: Average Response Time (in Sec) According to the Number of Traces and IRAs.

With the *IRATenaces*, on average each pheromone seems to have the same propensity to be chosen. The fact that the average response time is not exactly the same is certainly due to the time needed by each *IRATenace* to find a pheromonal trace because the population is randomly dispatched in the network. This is amplified by the non-homogeneous dispersion of the pheromone through the network. With the *IRASIs*, the suspicion indices vary from 4 to 9 in the case of 5 pheromonal traces and from 1 to 10 to in the case of 10 pheromonal traces. The extreme effect of the *IRASI* strategy appears obvious: the average response time increases considerably when the number of traces increases, even if the population of *IRASI* increases . This is due to the fact that the *IRASIs* are permanently changing their direction to trace a new pheromone. This is accentuated by the amplitude of the suspicion indices.

3) **Effect of the Control Mechanism**

A mechanism to control the population of MAs has been implemented in IDReAM, which destroys and regenerates MAs according to their frequency at each node. Table 6 shows the effect of the control mechanism when all the pheromone is concentrated at a part of the network. For this test, only 10 virtual machines have been used. The pheromonal traces have been deposited artificially such that there are uniquely 5 pheromonal traces leading to an alert located at each of the machines 1 to 5 (group 1). The distance of the pheromone' s diffusion is 3. There is no pheromone located at the machines 6 to 10 (group 2). Before the pheromones are deposited, two IRAs are launched: one from the machine 5 and one from the machine 10. Each IRA duplicates every time it crosses a node with several pheromones. Table 6 expresses the percentage of destruction and generation of the IRAs by the population control mechanism and per group of machines. The results are collected during the time spent between the first pheromonal deposit and the disappearance of the last pheromone. The measure is repeated 20 times. The parameters have been chosen so that during one minute a machine is visited by 6 IRAs: if there are more than 6 IRAs per minute, the surplus of MA is destroyed by the host. If there are less than 6 IRAs per minute, new IRAs are generated by the host. Obviously, the nodes where the MAs are destroyed are mainly

	Group 1	Group 2
Percentage of Destruction	68.57	31.43
Percentage of Generation	28.8	71.2

Table 6: Percentage of Destruction and Generation of IRAs at the 2 Groups of Machines.

those where the pheromones have been concentrated because the IRAs have a greater propensity to go there. The nodes where new IRAs are created are mainly those where the pheromones are absent because the IRAs have a smaller propensity to go there. However, we note here the dangerous effect that the pheromone can have if it is controlled by an intruder. An injection of a false pheromone

to increase its concentration at certain locations of the network can destabilize the system. Therefore, the control mechanism enforced by the permanent process of evaporation and inhibition inherent in the way the pheromone has been built is crucial for the "social" regulation of the MAs' populations, which of course reinforces the survival of IDReAM.

5. CONCLUSION

The combination of the MA paradigm with two natural life paradigms is the driving force behind IDReAM. In the present paper the entire description of IDReAM architecture and implementation is given. The IDS and the IRS are totally separated in an architectural point of view but can communicate through primitives and mechanisms derived from the conceptual model such as the communication through the pheromone. The implementation of the different agents and services is quite generic, with mechanisms dedicated to the two main populations of agents (IDAs and IRAs). This is the case, for instance, of the moving mechanism. But there is also a specialization of the agents' role such as the *PortScanIDA*'s role. The implementation has been thought to be sufficiently flexible and modular: new populations of agents can be easily declined from the generic classes according to the protection's needs. We also assessed IDReAM's reliability to detect and respond to intrusions and we measured its cost in term of resource consumption: the services and agents are lightweight software entities which avoid too much CPU and bandwidth consumption. In an intensive port scan, *PortScanIDAs* are more stressed and detect more scans, and the IRAs' response time differs considerably with the tracing strategy. Moreover, the population control mechanism avoids the network being flooded by its own protection system.

6. REFERENCES

[1] M. Asaka, S. Okazawa, A. Taguchi, and S. Goto. A method of tracing intruders by use of mobile agents. INET 99, June 1999.

[2] W. Binder. Design and implementation of the j-seal2 mobile agent kernel. *ECOOP 2000*.

[3] M. Crosbie and E. Spafford. Defending a computer system using autonomous agents. Technical report, Department of Computer Sciences, Purdue University, 1995.

[4] S. Fenet. Vers un paradigme de programmation écologique basé sur une architecture mobile distribuée : Application à la sécurité des réseaux. *Laboratoire d'Ingénierie des Systèmes d'Information - Lyon, France*, December 2001.

[5] N. Foukia. Idream: Intrusion detection and response executed with agent mobility - the conceptual model based on self-organizing natural systems. *Proceedings of the Engineering Self-Organising Applications Workshop (ESOA'04) in AAMAS conference*, July 2004.

[6] N. Foukia, S. Hassas, S. Fenet, and P. Albuquerque. Combining immune systems and social insect metaphors: a paradigm for distributed intrusion detection and response system. *Mobile Agents for Telecommunication Applications (MATA) - Morocco*, 8-10 October 2003.

[7] N. Foukia, S. Hassas, S. Fenet, and J. Hulaas. An intrusion response scheme: Tracking the alert source using stigmergy paradigm. *SEMAS 2002, Bologna - Italy*, July 2002.

[8] A. Hugentobler. Réalisation d'une authorité de certification pour l'université de genève. Master's thesis, Centre Universitaire d'Informatique, June 2000.

[9] W. Jansen, P. Mell, T. Karygiannis, and D. Marks. Applying mobile agents to intrusion detection and response. Technical report, National Institut of Standard and Technology, Interim Report 6416, October 1999.

[10] S. Martino. A mobile agent approach to intrusion detection. Joint Research Centre-Institute for Systems, Informatics and Safety, Italy, June 1999.

[11] J. D. Queiroz, L. F. R. da Costa Carmo, and L. Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. *In 2nd Annual Workshop on Recent Advances in Intrusion Detection - Rio de Janeiro - Brasil*, September 1999.

[12] G. Ramachandran and D. Hart. A p2p intrusion detection system based on mobile agents. *In Proceedings of the 42nd Annual Southeast Regional Conference*, April 2004.

[13] J. Skaita and F. Mourlin. A mobile approach for the intrusion detection. Technical report, Laboratory of algorithmic, Complexity and Logic - Communicating Systems Group - Paris 12, France, 2002.

[14] E. H. Spafford and D. Zamboni. Intrusion detection using autonomous agents. *Computer Networks 34(4):547-570*, October 2000.

[15] T. Toth and C. Kruegel. Evaluating the impact of automated intrusion response mechanisms - 18th annual computer security applications conference. December 9-13, 2002.

[16] G. B. White, E. A. Fisch, and U. W. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network, 10(1):20-23*, January/February 1996.