Journal of Computer Science 3 (1): 35-42, 2007
ISSN 1549-3636
© 2007 Science Publications

A Secure Protocol Based on a Sedentary Agent for Mobile Agent Environments

Abdelmorhit El Rhazi, Samuel Pierre and Hanifa Boucheneb
Mobile Computing and Networking Research Laboratory (LARIM)
C.P. 6079, Station succ. Centre-ville, Montréal, Québec, Canada H3C 3A7

Abstract: The main challenge when deploying mobile agent environments pertains to security issues concerning mobile agents and their executive platform. This paper proposes a secure protocol which protects mobile agents against attacks from malicious hosts in these environments. Protection is based on the perfect cooperation of a sedentary agent running inside a trusted third host. Results show that the protocol detects several attacks, such as denial of service, incorrect execution and re-execution of the mobile agent code. Results also indicate that the traffic generated and run time are barely affected.

Key words: Mobile agent security, mobile computing, denial of service

INTRODUCTION

The ubiquitous use of the Internet and the integration of mobile computing devices in our lives raise issues pertaining to the organization and communication of information in wide-area networks. Several architectures have been developed, among them those based on the mobile agent concept that offers to solve specific client/server architecture problems, such as the number of messages exchanged between clients and servers, which is a stumbling block to the growth of services offered over the Internet. However, deploying this new architecture in commercial applications requires secure protocols. Indeed, the mobile agent code and data are vulnerable to malicious attacks on the host where the agent is running.

The term *mobile agent* includes two different concepts: agent and mobility. An agent is a program which is often implemented as multiple threads that have certain features. It operates without direct human intervention through certain controls of its actions and of its internal state. It can interact with other agents using an inter-agent communication language. An agent is said to be mobile when it can be transported from one machine to another within a network.

A mobile agent system has several advantages compared to its rival client/server paradigm. Indeed, when the mobile agent moves to the area where the desired resource resides, it interacts with it without transmitting any data through the network, thus significantly reducing bandwidth consumption. In the same manner, when the mobile agent moves to the area where the user resides, it can respond quickly to the user's actions. In such cases, the agent can continue its execution even if the network connection has failed. Moreover, the majority of distributed applications naturally agree with the mobile agent model since an agent can migrate through several platforms, send an offspring agent to simultaneously visit another platform or remain stationary and interact with remote resources. Finally, agent mobility hides network specifications from developers.

However, mobile agent-based architectures are not widely deployed in commercial applications. This is mainly due to security issues concerning the mobile agents and the hosts which execute them. Generally, the mobile agent systems subdivide security problems into two different parts. The first one concerns the protection of the platform that runs the mobile agent against attacks which can harm its resources. This security aspect has already been addressed by the prolific research conducted upon the appearance of mobile code (Applet), viruses and worms through Internet[1,2]. The second security concern is related to the protection of mobile agents from malicious platform attacks. The importance of this type of protection lies in the fact that in its itinerary, the mobile agent could visit platforms which are not necessary reliable. Consequently, the mobile agent's code and/or data could be tampered with, re-executed in order to obtain illegal results. Its execution could be simply delayed or totally destroyed. Therefore, mobile agents must be provided with some protection mechanisms to prevent these attacks.

The current research tasks focused their efforts on protecting and detecting attacks aimed at the mobile agent code and data. Some of these approaches use cryptographic notions to hide the mobile agent code and data inside platforms[3]. Others use reference states and protection delays in order to detect and protect the agent against attacks[4]. However, none of these approaches have yet succeeded in designing a realistic, secure protocol that prevents the various attacks. Among the attacks which have yet to be addressed are

Corresponding Author: Samuel Pierre, Department of Computer Engineering, École Polytechnique de Montréal
C.P. 6079, Station Centre-ville, Montréal, Québec, Canada H3C 3A7, Tel.: (514) 340-4711 Ext. 4685, Fax: (514) 340-4658

the attacks which consist of re-executing the mobile agent code, or parts of its code, so that the attacking platform can obtain illegal privileges such as the re-execution of a purchasing order or the re-execution of the agent with different data. The one-second attack, which has yet to arouse the interest of researchers, refers to the attack called *denial of service,* which consists of killing or delaying the execution of the mobile agent. For example, consider a mobile agent from an electronics store which offers a promotion that is about to expire. In this case, the assailing platform could delay the execution of the agent until the offer expires. The importance of providing the mobile agent with protection mechanisms against these attacks is of the utmost importance, especially for e-commerce applications where security issues are critical. Indeed, these attacks (agent re-execution, denial of service, etc.) have a direct influence on the degree of reliability of the systems using mobile agent technology in the field of e-commerce and m-commerce (mobile electronic commerce).

Hence, a secure protocol was designed to detect attacks which harm the mobile agent code and data, as well as code re-execution and denial of service attacks. The protocol was implemented within a mobile agent system.

MOBILE AGENT SECURITY: BACKGROUND AND RELATED WORK

According to Walsh *et al.*[2], mobile agent system security covers four different aspects:

* Security of agent transmissions;
* Protection of the platform against malicious agents;
* Agent protection against attacks from other malicious agents;
* Agent protection against attacks from malicious platforms.

Mobile agent systems must be equipped with safe transfer mechanisms since malicious adversaries can capture the agent and analyze its data and code while it is being transmitted. The platform that runs the mobile agent is vulnerable to several attacks due to the execution of the mobile code. Those attacks include the unauthorized reading of significant platform information, resource damages (CPU, memory, etc.) and attacks called *denial of service* (i.e. a partial or complete halt of the services provided by the platform). At the same time, reliable agents should have access to the platform resources in order to normally execute its code. Thus, the system should provide the platform with authentication mechanisms that enable it to specify the access rights for the mobile agents. The problem[5] of platform protection against the attacks of malevolent mobile agents, such as imminent threats from viruses, worms and Trojan horses received considerable attention from the research community.

Mobile agents are vulnerable to several types of attacks from malicious platforms where they run their codes as well as attacks performed by other agents executed within the same environment. This includes passive attacks, where neither data nor the code of the mobile agent are corrupted, such as traffic analyses, as well as active attacks once the mobile agent data and/or code are modified.

A description of malicious platform attacks: Hohl[6] defines, in general, a malicious platform as being a party able to execute an agent that belongs to another party and tries to attack that agent in some way. He also provides a list of the various platform attacks against a mobile agent: code, data and control flow spying out; code, data and control flow manipulation; incorrect code execution, platform masquerading, denial of service, spying out interaction with other agents, manipulation of interactions with other agents, returning wrong results of system calls issued by the agent and re-execution of the agent. Further details for each of these attacks are presented in[5,7].

Mobile agents "protection" against the attacks: Whereas the countermeasures aiming to protect the platforms are largely inspired by the conventional systems using preventive methods[5], the mobile agent protection techniques provide for the detection of an attack. This is due to the fact that the agent is completely dependent upon the platform on which it runs its code and cannot, by itself, prevent an attack. However, it is possible to detect the attacks or render them less harmful.

Mobile agent protection from platform attacks has been researched prolifically due to the different difficulties encountered by traditional systems. Here are some of the most important protection techniques:

The protection of mobile agents using cryptographic functions[3]: This approach ensures the confidentiality of the mobile agent computations by encrypting its code. Platforms execute the encrypted mobile agent code without decrypting it; hence they cannot modify it or spy on it. The objective is to encrypt functions and add their transformations to a program. The resulting program can be executed by a processor. The meaning of the functions cannot be revealed to the processor. The limitation of this approach is that it cannot be applied to all types of functions.

Cryptographic security of mobile codes[5]: This approach is based on the usage of a reliable third party (RTP) that executes a part of the code on the behalf of the mobile agent. The mobile agent code is divided into several fragments. Certain fragments are not transported by the mobile agent, but implemented within the RTP. Mobile agents need to contact the RTP to be able to execute these special fragments. The RTP does not know the meanings of the functions which represent mobile agent code fragments, as its purpose is to ensure reliable computing confidentiality.

Table 1: Comparative view of the approaches

N°	Types of Attacks	Crypto. Function	Crypto. Security	Data Security	Reference States	Time Limited Blackbox	Co-Operating Agents
1	Code spying out	P	P			P	
2	Data spying out			P		P	
3	Control flow spying out	P	P				
4	Code manipulation	P	P		D	P	
5	Data manipulation			D		P	
6	Control flow manipulation	P	P	D	D		D
7	Incorrect code execution	P	P		D	P	
8	Platform masquerading			P			
9	Denial of service					D	
10	Spying out interactions with other agents					P	
11	Manipulation of interaction with other agents				D		
12	Returning wrong results of system calls by the agent				D	P	
13	Re-execution of the agent						

P = Protection D = Detection

The approach proposed in this paper rests upon the same principle, that is, a protocol using a third party to protect the mobile agent and a mobile agent code that is split into several fragments. The limitation of this approach pertains to the size of the code fragments implemented within the RTP which must remain low to ensure that system performance is not severely hindered.

Cryptographic security of data[1]: This approach ensures the security of mobile agent data using cryptographic systems. Three mechanisms are implemented within a mobile agent to protect data. The first one consists of declaring a part of the data as '*read only'*. This section contains data which cannot be modified or deleted by visited platforms without attack detection, i.e., mobile agent identification. The second mechanism allows a mobile agent to create an '*append only'* section where a visited platform can only add one record and cannot update or delete existing records without being detected. The last mechanism allows a mobile agent platform of origin to select from a list of visited platforms within which certain data becomes visible.

Mobile agent protection with reference states[8-10]: This approach detects an attack by analyzing the discrepancies in the behaviour of the attacking platform and the platform of reference. The mean mechanisms which use this approach are '*state appraisal'*[8], '*replicated server'*[9] and '*execution trace'*[10]. The protocol proposed in this paper uses this approach to detect attacks aimed at modifying the mobile agent code by replicating segments of the code.

Time limited blackbox security[4]: The principle of this approach is to generate a mobile agent code which cannot be manipulated by a malicious platform during a certain period of time. Visited platform can access the input and output of mobile agent execution, although they cannot know the meaning of its computing. This approach uses the cryptographic functions described in the first approach and is consequently subject to the same drawbacks. Moreover, the security time remains difficult to adequately determine.

Mutual protection of co-operating agents[11,12]: The basic principle behind this approach is to use a cooperating agent to protect mobile agent data. Roth[12] uses a cooperating agent to protect mobile agent itinerary which is defined as a list of visited platforms. The new protocol herein proposed widens the cooperating agent's tasks in order to protect mobile agents from several types of attacks.

Table 1 presents the possible type of protections against the various attacks listed at the beginning of this section and the current approaches which ensure. The approaches investigated may be classified into two different categories: those which detect or protect mobile agents against code attacks and those which detect or protect mobile agents against data attacks. Only the fifth approach, the time limited blackbox security, combines both types of protection, although the way in which it uses time is its main disadvantage: it implies that all platforms must be synchronized by a global clock.

The comparative analysis of these approaches indicates that none of the suggested solutions treat attacks which consist of re-executing the agent. Moreover, denial of service and attacks consisting of interactions manipulated by other agents are only detected. Also, generally, all these approaches protect only one aspect of the mobile agent: either its code or its data. Hence, it was deemed important to design a protocol that provides mobile agents with more security.

THE PROPOSED PROTOCOL

The proposed protocol combines the existing protection techniques in order to enhance mobile agent security. It uses a notion of third party and mobile agent code fragmentation as does the second approach

'Cryptographic security of mobile codes'. It also uses the technique called 'the use of reference states to protect mobile agents'. Indeed, the protocol uses a replica of a mobile agent code fragment which should be executed by the platform of reference. The use of a time counter (e.g. *Timeout*) was inspired by the fifth technique, 'Time-limited blackbox security'. Finally, the last technique, 'Mutual protection of co-operating agents', constitutes the driving principle of the proposed protocol.

The core of the proposed protocol is built around the complete cooperation of a sedentary agent in order to protect the mobile agent which moves between several platforms P_i (i=1, …, L) all over a network. For each execution step within a platform, the mobile agent sends three types of messages to the cooperating agent (*SA*) which is in charge of detecting an attack against the mobile agent data and code. The cooperating agent is called *sedentary* as it moves only once from the origin platform (*O*) to the trusted third platform (*T*), as shown in Figure 1. The sedentary agent does not constitute a copy of the mobile agent. It only contains the "critical code'' (i.e. the part of the mobile agent code which the protocol was designed to protect).

Concepts and Assumptions: The protocol[13] aims:

* to detect the attack which modifies a part of the mobile agent code and the incorrect executions (Attacks 4 and 7 in Table 1);
* to detect the attacks which tampers with the intermediate results obtained from the visited platforms (Attack 5 in Table 1);
* to detect the re-execution of a mobile agent code (Attack 13 in Table 1);
* to detect denial of service attacks (Attack 9 in Table 1);
* to record the mobile agent itinerary in order to detect attacks which modified its route (Attack 5 in Table 1).

The mobile agent code is split into two parts. A crucial part, the critical code, contains the mobile agent code whose handling is susceptible to influence the integrity of the mobile agent. For example, this part of the code compares offers from visited platforms. The second part, called the non-critical code, contains the remainder of the mobile agent code. Malicious hosts have no interest in attacking this part of the code. The security of the critical code is ensured by its duplication into two agents (mobile and sedentary) that simultaneously run this part of the code and compare both of their results. The protocol detects tampering attacks on the intermediate results by saving the sedentary agent data at each execution phase of the mobile agent.

The protocol detects attacks such as re-executions of the mobile agent code and denials of service by using two time counters: a *Timeout* and a *Supplementary Waiting Interval* (SWI).

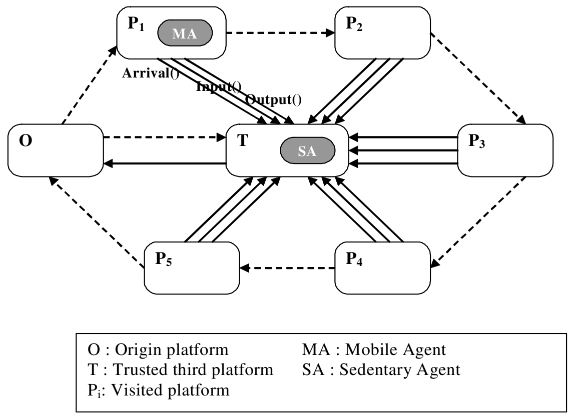

Fig. 1: Protocol concepts

It also allows for the preservation of the mobile agent itinerary and detects modification attacks according to Roth's Protocol[12].

In order to devise a secure protocol, two assumptions were adopted:

* the availability of a trusted third platform;
* the existence of an asymmetric cryptographic system.

The first assumption supposes that a trusted third platform (*T*) is available to all platforms in order to run the cooperating agent (*SA*). This platform must execute the code correctly. It must not collude with an attacker nor with the platform of origin (*O*) against a platform (P_i), nor with a platform (P_i) against the platform of origin (*O*) or against a platform P_j ($i \neq j$). *T* should be protected from denial-of-service (DoS) attacks. The second assumption stipulates that platforms *O*, P_i (i=1, …, L) and *T* use an asymmetric cryptographic system (private key/public key). All platforms P_i (i=1, …, L) allow the mobile agent to encrypt/decrypt the data that is sent. The mobile agent does not transport any private keys. It uses the cryptographic services provided by each P_i.

A description of the protocol: The proposed protocol can be described as a three step process: initialization, where the origin platform generates different agents, the execution inside each platform step, where mobile agents communicate with their cooperating agents and the final step, where the platform of origin validates the results obtained by the mobile agents.

Step 1: Initialization: At the beginning of the protocol, the original platform (*O*) generates the mobile agent and its cooperating agent. It initializes the parameters of both agents, sends the mobile agent to the first platform (P_1) on its itinerary and the cooperating agent (*SA*) to the third platform (*T*) where it remains stationary until the end of the protocol.

Step 2: Execution on each platform: When the mobile agent reaches the ith platform P_i, it sends a message type *Arrival()* to its cooperating agent (*SA*) in order to

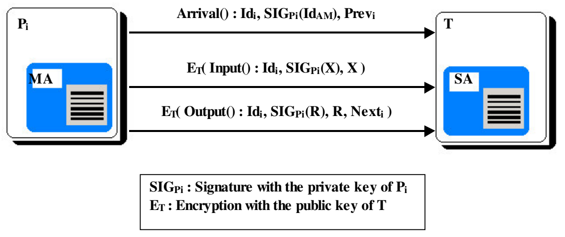
Pi
MA
Arrival() : Idi, SIGPi(IdAM), Previ
ET(Input() : Idi, SIGPi(X), X)
ET(Output() : Idi, SIGPi(R), R, Nexti)
T
SA
SIGPi : Signature with the private key of Pi
ET : Encryption with the public key of T

Fig. 2: Messages sent from mobile agent to sedentary agent

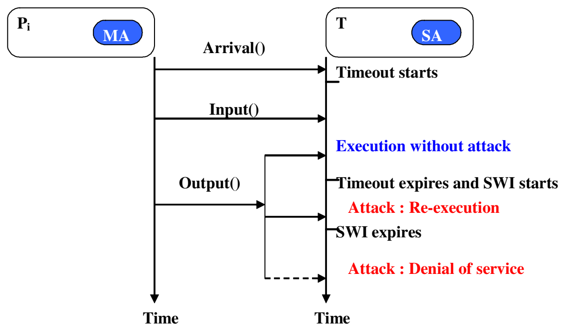

Fig. 3: Using timers

inform it of its arrival on platform P_i. When the sedentary agent receives the *Arrival* () message, it corroborates the authenticity of the message and validates the mobile agent itinerary. If verification results indicate the absence of corruption, the sedentary agent initializes a time counter *Timeout* used to limit the execution time of the mobile agent inside the platform P_i.

Before the execution of the critical code, the mobile agent sends an *Input*() message to its cooperating agent. This message contains all of the input provided by the present platform P_i, which is required to run this part of the code. As soon as the sedentary agent receives the *Input*() message, it extracts the data which was sent and begins to execute the cloned part of the mobile agent code. Once the mobile agent has completed the execution of its code, it sends an *Output*() message to the sedentary agent. The *Output*() message contains the results obtained by the mobile agent on platform P_i.

When the sedentary agent (*SA*) receives the *Output*() message, it extracts the results. Then, it authenticates the critical code execution by comparing the results obtained by the mobile agent on platform P_i to the results, deemed trustworthy, from the third platform (*T*) who was supposed to execute the code accurately. If the results obtained by the cooperating agent differ from those obtained by the mobile agent, the sedentary agent concludes that an attack was carried out by the present platform P_i and marks this platform as housing an attack. If both results are identical , the cooperating agent concludes that the execution of the critical code was completed without any attacks. In both cases, the mobile agent continues its itinerary to the next platform.

The sedentary agent records the mobile agent itinerary by saving the identity (*Next*$_i$) of the platform where the mobile agent wants to migrate. Figure 2 illustrates the three types of messages exchanged between the mobile and the sedentary agent. The *Arrival()* message contains the ID of the current platform P_i, which is an electronic signature using the private key of $P_{i.}$ It also includes the mobile agent ID to ensure message authentication and the ID of the previous platform which specifies where the mobile agent came from (*Prev*$_i$) to allow for verifications of the itinerary. *Input()* messages contain the ID of the current platform, the input required to run the critical code and an electronic signature to ensure its authentication. *Ouput()* messages contain the ID of the current platform, the results obtained by the mobile agent on platform P_i , an electronic signature to ensure its authentication and the ID of the following platform (*Next*$_i$) to keep track of the mobile agent itinerary. *Input()* and *Output()* messages are encrypted using the public key for platform *T* to ensure the privacy of certain specific platform data when messages are being transferred. A malicious entity cannot forge any of the messages nor can it read critical data.

As soon as the sedentary agent receives the *Arrival*() message, it initializes a *Timeout* with an appropriate value that represents the maximum required time to run the entire mobile agent code and transfer both *Input*() and *Output*() messages between the mobile and the sedentary agents. A *Timeout* which expires (*Timeout* = 0) before the sedentary agent receives the *Output*() message indicates that the present platform has attacked the mobile agent, i.e., the mobile agent code has been re-executed. The sedentary agent continues to wait for the *Output*() message during another Supplementary Waiting Interval (SWI). Once this additional time has expired, the sedentary agent concludes that a denial-of-service attack has occurred. Figure 3 illustrates the use of timers to detect attacks.

Step 3: The end: When the mobile agent ends its run on the last platform P_L, it migrates to the original platform *O* which, in turns, contacts the sedentary agent to request the list of attacking platforms and the various results obtained before it ended its run on the third platform *T*. The original platform *O* validates the results obtained for each platform visited. The corrupted results, i.e., results obtained on the platforms marked as attackers by the sedentary agent, are ignored.. The platform *O* updates its list of attack platforms so as to avoid them during the next mobile agent run.

The messages sent from the mobile agent to the sedentary agent are encrypted using asymmetric security systems or contain digital signatures. Consequently, their contents cannot be modified by malicious platforms performing a person-in-the-middle attack without detecting it, unless the attacker has obtained private keys from the platforms. Another attack scenario would be to delete these messages: in this case, the sedentary agent would detect this attack after the allotted time has expired.

RESULTS

The protocol was implemented in order to test its ability to detect attacks and to measure its performance in terms of traffic generated and run time. A method was also devised and implemented to estimate the temporal counter *Timeout* that allows the sedentary agent to detect re-execution attacks.

Estimation of the temporal counters timeout and SWI: The *Timeout* and *SWI* (*supplementary waiting interval*) values must be either calculated or estimated using timers in order to analyze performance. *Timeout* represents the maximum amount of time required to run the entire mobile agent code and transfer both *Input*() and *Output*() messages between the mobile and sedentary agents. *SWI* represents the time interval which has elapsed after the sedentary agent detects a denial-of-service attack. An agent, called Estimator Agent (*EA*), was implemented to allow the original platform *O* to estimate the value of *Timeout*. The *SWI* value was not estimated as it depends on the type of application under which the protocol will be implemented. The application is designed so that administrators can specify the time delay that they consider to be an indicator of a denial-of-service attack.

At the beginning of the protocol, the original platform *O* sends the sedentary agent *SA* to the trusted platform *T*. Then, it runs the *EA agent* that estimates the *Timeout* value and communicates the result to the *SA* agent. In order to calculate this estimation, the *EA* agent simulates an execution stage of the mobile agent *MA* on a platform.

Several experiments to detect mobile agent code re-execution were undertaken by directly assigning the value estimated by the *EA* agent to *Timeout*. The results reveal that this simple assignment does not provide satisfactory rates. The studied rates indicate the percentage when the sedentary agent *SA* detects the re-execution attacks and the percentage when it marks an attacking platform whereas this one runs the code only once. In order to study this phenomenon, the following experiment was undertaken:

The mobile agent *MA* moves to two platforms P_1 and P_2. Platform P_1 was forced to re-execute the critical code of the mobile agent whereas P_2 platform runs it only once. The run time of the *EA* and both run times of the *MA* inside platforms P_1 and P_2 were calculated. Figure 4 shows the results obtained.

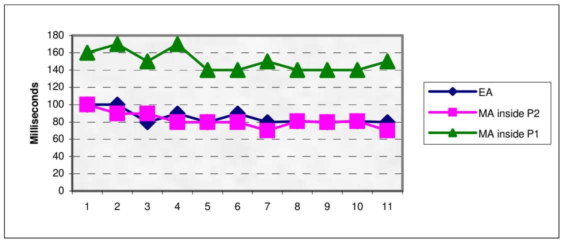

Fig. 4: Run times of agents *MA* and *EA*

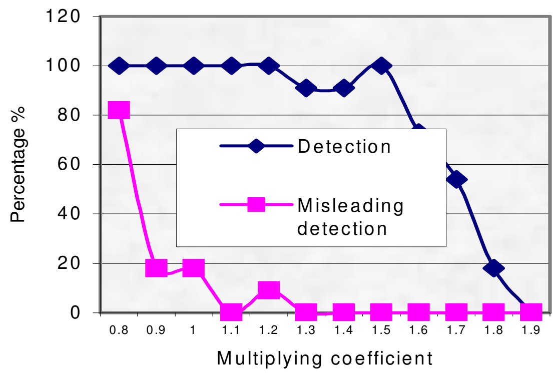

Fig. 5: Impact of the coefficient on the two rates

Note that the two lines representing the run time of agent *EA* and the run time of the *MA* inside platform P_2 are similar. This is due to the fact that both agents run the same code within similar environments. The difference between these two lines justifies the unsatisfactory rates obtained by the first experiment. However, the line that represents the run time of the *MA* inside the attacking platform P_1 is certainly far away from the other two lines. This is due to the fact that platform P_2 re-executes the code, hence requiring more time.

Given the results from the previous experiment, the *Timeout* value was calculated with the following formula:

Timeout = coefficient × run time of *EA*

The coefficient varied between 0.8 and 2 in order to investigate the impact of this coefficient on both rates, i.e., the rate for detecting the re-execution and the rate for the misleading detection of the re-execution. The results are presented in Figure 6.

Notice that the detection rate for low coefficient values equals 100%. However, the rate of misleading detection varies goes from 80% to 0%. This is justified by the fact that the timeout counter expires before the mobile agent is executed on both platforms P_1 and P_2. However, when the coefficient value is high, both rates become null. The cooperating agent *SA* is not mistaken about the reliability of platform P_2, although it does not detect that P_1 re-executes the code. This is due to the fact that the high timeout value does not expire until the mobile agent is fully executed. The best interval is

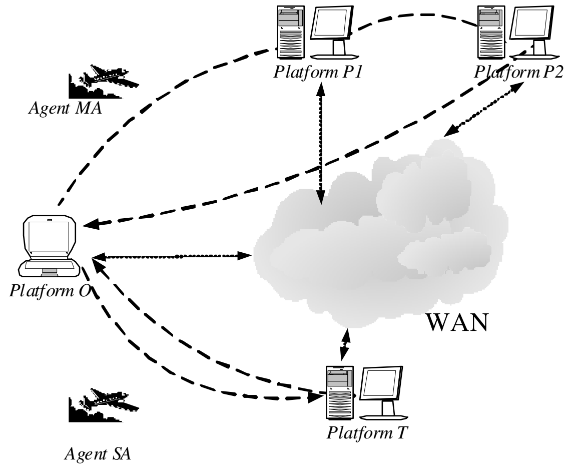

Fig. 6: The experimental environment

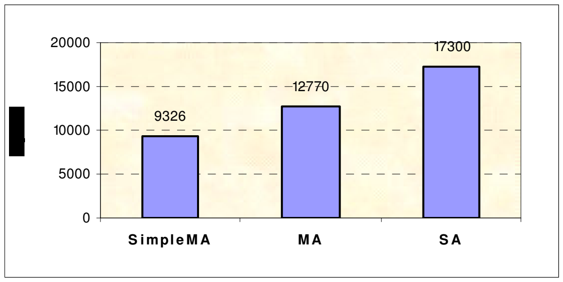

Fig. 7: Comparing the agent sizes

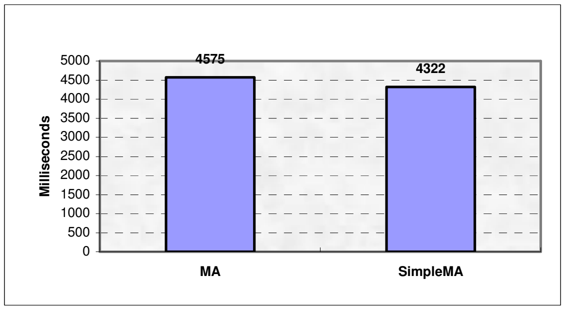

Fig. 8: Comparing the run time of the agents

located between 1.3 and 1.5 where the detection rate is higher than 90% and the rate of misleading detection equals 0%. This result is justified by the fact that the execution environments vary between the P_1, P_2 and O platforms. Indeed, the run time of a mobile agent depends on the number of activated processes on the platform, on the number of activate agents, on the technical characteristics of the computers and on the network's state at the time the *MA* and messages are transferred.

Measurements and results: The implementation of the protocol[13], an application where mobile agent *MA* visited two platforms P_1 and P_2 was used as an example. It ran the critical code on each platform and returned to its platform of origin. The performance of the implemented protocol were compared to those of a simple mobile agent, called *SimpleMA*, which behaves in the same manner as the mobile agent *MA*, although it is not secure, i.e., it does not implement the protocol. The metrics considered were the traffic generated by the mobile agent and its run time. These metrics were chosen as the protocol, although generates additional traffic during the message transfer between the mobile and sedentary agents and increases the mobile agent run time because of the operations designed to detect the attacks.

Four machines were used to conduct the experimental study. Each machine was equipped with a mobile agent platform and each platform represented a different site. Two platforms, called *P1* and *P2* appeared upon the *MA* itinerary. The third platform, called *T*, represented the trusted third platform. The last platform, called *O*, represented the platform of origin. Consequently, agents *MA* and *SA* were created within platform *O* which sent *MA* to platform *P1* and *SA* to platform *T*. Agent *MA* moved from *P1* to *P2* and returned to platform *O*. Agent *SA* returned to platform *O* at the last phase of the protocol. The experimental environment is depicted in Figure 6.

Both agents were implemented using a mobile agent platform, which used Java as the agent programming language. IAIK-JCE 3.01[14] was selected as the security package as it offers a pure Java implementation of different cryptographic algorithms. A RSA using a key length of 1024 bits was selected to sign and encrypt messages.

Analysis of generated traffic: The number of bytes transmitted between the platforms (O, T, P_1 and P_2) were measured. Figure 7 illustrates the average size of the three agents (*MA*, *SA* and *SimpleMA*), indicating that the average size of the secure agent *MA* (about 12 kilobytes) is not significantly greater than the average size of the simple agent *SimpleMA* (about 9 kilobytes). This is justified by the fact that the sedentary agent supports most of the security mechanisms.

The second revelation is the significant size of the sedentary agent *SA* (about 17 kilobytes), which is justified by the same reason, i.e., the *SA* is entirely responsible for security. However, even if the size of the sedentary agent is considerable, it does not have a meaningful effect upon the traffic generated as it migrates only once between the platform of origin (*O*) and the third platform (*T*).

Run time analysis: The second metric was used to assess the performance of the protocol by analyzing the run time of the *MA*. The results pertaining to the global run time of the MA and the *SimpleMA* were measured and compared. Figure 8 illustrates this comparison.

Overall, both run times are very close: a difference of 250 milliseconds, which represents only an increase of 5.8% for the *MA* run time compared to that of the *SimpleMA*. Two factors explain this result. First, the mobile agent uses asynchronous communication to send messages to its cooperating agent. Indeed, the manner in which the protocol is designed allows the *MA* to execute its code without waiting for a response from its cooperating agent, thus avoiding the possible deadlock between it and the *MA*. The second factor is that, in this scenario, the migration mechanism and the mobile agent instantiation are more costly in terms of execution time than the mobile agent executed on the platforms.

CONCLUSION AND DIRECTIONS FOR FUTURE RESEARCH

A secure protocol to protect mobile agents from the attacks of malicious hosts was designed and tested. This protection is based on the complete cooperation of a sedentary agent that runs inside a trusted third platform. The main concept consists of expanding the tasks of the cooperating agent to make it more powerful than it has been in previous approaches[11,12] which, although exploiting the concept of the cooperating agent, limited its use to the protection of the mobile agent itinerary.

This new protocol offers several advantages. First, it allows for message transfers between the mobile agent and the cooperating agent without debasing the performance of the mobile agent. Another considerable advantage of the protocol lies in its avoidance of cryptographic functions[3] which increase the vulnerability of the platforms that carry encrypted code and possibly malevolent instructions.

The effectiveness of the timers depends on the estimation or calculation methods employed. A method that generated significant results for similar platforms was proposed and implemented. A real time mechanism should be devised to allow for the calculation of the *Timeout* used to detect re-execution attacks. This mechanism, implemented within the cooperating sedentary agent, should take into consideration the fluctuating platform and network environments. Indeed, these two factors directly influence the selection of the initial value of the timer. This would aim to increase the detection rate of the cooperating agent for this type of attack and decrease the rate when it is mistaken about the reliability of non-malicious platforms.

REFERENCES

1. Karnik, N.M. and A.R. Tripathi, 2000. A security architecture for mobile agents in ajanta. Proc. 20th Intl. Conf. Distributed Computing Systems of IEEE Computer Society, Los Alamitos (Ed.), California, pp: 402-409.
2. Walsh, T., N. Paciorek and D. Wong, 1999. Security and Reliability in Concordia. Mobility: Processes, Computers and Agents, Addison-Wesley (Ed.), pp: 525-534.
3. Sander, T. and C.F. Tschudin, 1998. Protecting Mobile Agents Against Malicious Hosts. Mobile Agents and Security, G. Vigna (Ed.), Lecture Notes in Computer Science, vol. 1419.
4. Hohl, F., 2000. A framework to protect mobile agents by using reference states. Proc. 20th Intl. Conf. Distributed Computing Systems of IEEE Computer Society, Los Alamitos (Ed.), California, pp: 410-417.
5. Algesheimer, J., C. Cachin, J. Cameniscsh and G. Karjoth, 2000. Cryptographic Security for Mobile Code. IBM Research Report, Zurich, Switzerland.
6. Hohl, F., 1998. Time Limited Blackbox Security: Protecting Mobile Agents from Malicious Hosts", in Mobile Agents and Security, LNCS 1419, Springer-Verlay, pp: 92-113.
7. E l Rhazi, A., 2003. Sécurité des Agents Mobiles : Protocole Sécuritaire Basé sur la Coopération Parfaite d'un Agent Sédentaire, Master Thesis, École Polytechnique de Montréal, Canada.
8. Farmer, W.M., J.D. Guttman and V Swarup, 1996. Security for Mobile Agents: Issues and Requirements. Proc. 19th Natl. Information Systems Security Conf., pp: 591-597.
9. Minsky, Y., R. Van Renesse, F. Scheinder and S. Stoller, 1996. Cryptographic support for fault-tolerant distributed computing. Proc. 17th ACM SIGOPS, European Workshop, pp: 109-114.
10. Vigna, G., 1998. Cryptographic Traces for Mobile Agents. Mobile Agents and Security, G. Vigna (Ed.), Springer-Verlag, pp: 137-153.
11. Allée, G., S. Pierre, R.H. Glitho and A. El Rhazi, 2005. An improved itinerary recording protocol for securing distributed architectures based on mobile agents. Intl. J. Mobile Information Systems, 1: 127-149.
12. Roth, V., 1998. Mutual protection of co-operating agents. Secure Internet Programming, Vitek and Jensen (Ed.), Springer-Verlag, Berlin, Germany, pp: 26-37.
13. El Rhazi, A., S. Pierre and H. Boucheneb, 2003. Secure protocol in mobile agent environment. Proc. Canadian Conf. Electrical and Computer Engineering (CCECE2003), Montreal, Canada.
14. Institute for Applied Information Processing and Communications, Javadoc for IAIK-JCE 3.01, Online Documentation, 2002. http://jce.iaik.tugraz.at/products/01_jce/documentation/index.php