A Reliable Protection Architecture for Mobile Agents in Open Network Systems

Ibharalu Friday Thomas
Department of Computer Science
University of Agriculture
Abeokuta, Nigeria

Sofoluwe Adetokunbo Babatunde
Department of Computer Science
University of Lagos
Lagos, Nigeria

Akinwale Adio Taofiki
Department of Computer Science
University of Agriculture
Abeokuta, Nigeria

ABSTRACT
A mobile-agent system is one where user programs (the agent) may voluntarily and autonomously migrate from one computer (the host) to another (the mobile agent server). A large deployment of mobile agent systems is not possible without satisfying security architecture. The major obstacle facing wide deployment of mobile agents is the attack of a visiting code by a malicious host. The fact that host computers have complete control over all the programs of a visiting agent makes it very hard to protect agents from un-trusted hosts. This has resulted to restricted deployment of mobile agents to known hosts in closed networks where the security of the agents is guaranteed. However, this restriction negates the original major concept of autonomy on which mobile agent technology is established. In this paper we propose dynamic protection architecture for mobile agents systems, using Travel Diary Protection Scheme and Platform Registry. The scheme protects and allows mobile agents to roam freely in open networks environment without being compromised in a malicious hosts.

General Terms
Security of Mobile Agent on host platform

Keywords
Mobile Agents, Security, Travel Diary, Platform Registry

1. INTRODUCTION
Mobile agent can be simply thought as an entity which can run in dynamic environment with autonomous ability and mobility. This technology has great potential in e-Commerce, network management, distributed computing, data mining, intrusion detection system, etc. But security is the main problem that prevents mobile agent from being widely deployed. Generally, security problems lie in two aspects: host security and agent security. The first problem has many common characteristics of traditional computer security; thus corresponding traditional methods have been used to solve it with satisfactory results. But the latter is still a challenging problem. Our study focuses on the security of mobile agent on a host platform. Furthermore, an agent can embark on two types of journey: a journey with a diary containing pre-defined itinerary and a journey where the mobile agent has no fore knowledge of the host to visit. This is called a free-roaming mobile agent which is more difficult to protect. Methods used to protect an agent (its data and state) count on the type of the agents' journey. Free-roaming agent without traveling dairy specifying where to visit may face more complex attacks such as colluded truncation attack, replay attacks or many other forms of host attacks on a visiting agent.

This paper focuses on securing free-roaming agents in open network environments and presents a novel security protocol which has fine function in preventing attacks.

2. LITERATURE REVIEW
Since the beginning of mobile agent research, many security issues have been identified. In [15], issues were classified according to the source of the attack and the entity being attacked: agents against agents, agents against hosts, and hosts against agents.

In the first category - agents against agents - we can find attacks in which agents modify or access another agent's data, disguise their identity in order to falsify a transaction, or repeatedly send messages to another agent in order to launch a denial of service attack, among others. The second category - agents against host platforms - includes threats in which agents perform some malicious action on a resource they can access (e.g. deleting a file), consume an excessive amount of system resources, gain access to a service to which they are not entitled, and so on.

With regard to these two first categories, in which the attacker is an agent, sound solutions have already been proposed. Among the solutions that provide an acceptable level of protection, the most efficient one is called Software-Based Fault Isolation [3]. This mechanism, also known as sandboxing is based on limiting program accessibility to a closed domain, in such a way that the program address space and available resources are confined within this domain.

Other mechanisms proposed for these kinds of attacks include: using safe code interpretation [4], where the set of available instructions prevents the agent from attacking the host: signing the code in order to authenticate the agent owner, together with some mechanism to determine the level of trust of this owner [10], sending logical demonstrations along with the code, in order to proof that the execution of that code is secure [11].

Regarding the second category - others against host platforms - the source of the attack can be any external entity that is not part of the agent platform. This external entity can perform attacks against the platform resources (files, communication ports, etc.) or against the host's communications with the outside. In these cases, security greatly depends on the mechanisms provided by the operating system. Additionally, a secure communication channel, established using mechanisms such as Transport

Layer Security [12], can be used to secure the communication between the host and other parties

Regarding the third category, that is host against agents - is the most difficult to prevent. It is obvious that if a host is to execute an agent, it must have complete access to the agent code, state and data. There is nothing to prevent the host from analyzing the agent code, from corrupting its state or data, from manipulating its execution environment, or from executing it multiple times in order to, for example, generate multiple purchases in a shopping scenario. If some agent data is to be kept secret from the host, it must be stored in a way that even the agent itself cannot directly access encrypted with the key of a different host platform, for instance.

Several mechanisms have been proposed to address the malicious host server problem. Some of the better known solutions to the malicious host problem are impractical. They have been designed for particular scenarios that are actually rarely found in real-life applications. Some of the better known ones are:

- Execution tracing
- Obfuscation
- Computing with encrypted functions
- Tamper-proof devices

2.1 Execution Tracing

Execution tracing [13] is a technique that allows unauthorized modifications of an agent to be detected upon completion of the agent execution. The protocol proposed in [14] is based on recording the agent's behavior on each platform in order to build a trace of its execution. The trace is composed of a sequence of identifiers corresponding to the operations executed by the agent. Platforms must produce and maintain traces of all executed agents, so that agent owners can request these traces after the agent has terminated its execution, and verify that the agent code or state has not been maliciously modified. This approach has several drawbacks, such as the size and the number of logs to be kept by platforms, or the possible lack of connection between the owner and the platforms once the agent has returned to the home platform. Besides, the verification mechanism is too expensive to be applied systematically, and can only be used when the owner has a suspicion that the agent execution has been corrupted.

2.2 Obfuscation

Code obfuscation [13] aims at generating executable agents which cannot be attacked by reading or manipulating their code. This technique is based on transforming the agent code in such a way that it is functionally identical to the original one, but it is impossible to understand it. The approach also establishes a time interval during which the agent and its sensitive data are valid. After this time elapses, any attempt to attack the agent becomes worthless.

The major drawback of these techniques is the difficulty in establishing the time required by an attacker to understand an obfuscated code. Similarly, no mechanism is currently known for quantifying the amount of time required by an agent to accomplish its task, especially in heterogeneous environments. As a result, restricting the lifetime of a mobile agent is not feasible in practice.

2.3 Computing with Encrypted Functions

Computing with encrypted functions is a technique proposed by Sander and Tschudin [12] to achieve code privacy and code integrity. Their technique is based on creating encrypted programs that can be executed without decrypting them. Supposing that a mobile agent has to execute a certain function f then f is encrypted to obtain E (f) and a program is created that implements E (f). Platforms execute E (f) on a clear text input value x, without knowing what function they actually computed. The execution yields E (f(x)), and this value can only be decrypted by the agent owner to obtain the desired result f(x). The main problem of this technique is that the authors have only found encryption schemes for polynomials, using homomorphism encryption and function composition techniques. Thus, their proposal is not suitable for general programming.

2.4 Tamper-Proof Devices

The use of tamper-proof devices is based on performing part or the entire agent execution on a physically sealed environment, which can be trusted to execute the agent correctly. Tamper-proof devices can be provided by a trusted third party and, if necessary, they can be inspected periodically to verify that their security has not been compromised. Tamper - proof devices can be used to carry out cryptographic operations with a private key that must be kept secret from the remote host. They can also have their own private key, for example, to sign partial results generated by the agent. This approach suffers from two areas: the cost of tamper-proof device on every platform. Secondly, the approach is only suitable for closed environments, such as corporate networks such as within in a group of banks in a geographic political area. As a result, the technique implies a loss of agent autonomy. Hence this paper focuses on realistic protocol that solves the malicious host problem.

3. OVERVIEW OF MOBILE AGENT ARCHITECTURE

Figure 1 to 5 shows the structure of mobile agent systems. Mobile agents travel around in a network environment visiting computers, hopping from one host to others. The execution of the program code and the dispatching agents to different computers is handled by mobile agent servers. Each agent has its own thread, which is executed by the host server. Any communication between different servers or agents is done by messages. So messages are quite universal in the agent environment.

3.1 Mobile Agents Interactions an a Server

Figure 1 shows the relationships between various agents on a given server as they communicate using messages to accomplish the execution tasks. The host plays a crucial row of providing the resources needed by visiting agents.

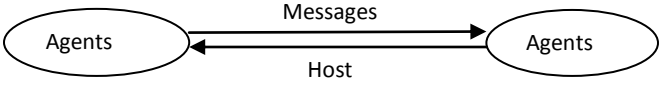

Figure 1: Agents on Servers

3.2 Mobile Agent Server Architecture

Figure 2 shows the structure of a mobile agent server. The server coordinates the activities of the visiting quest agents. Messages passing form an important component in the mobile agent communication.

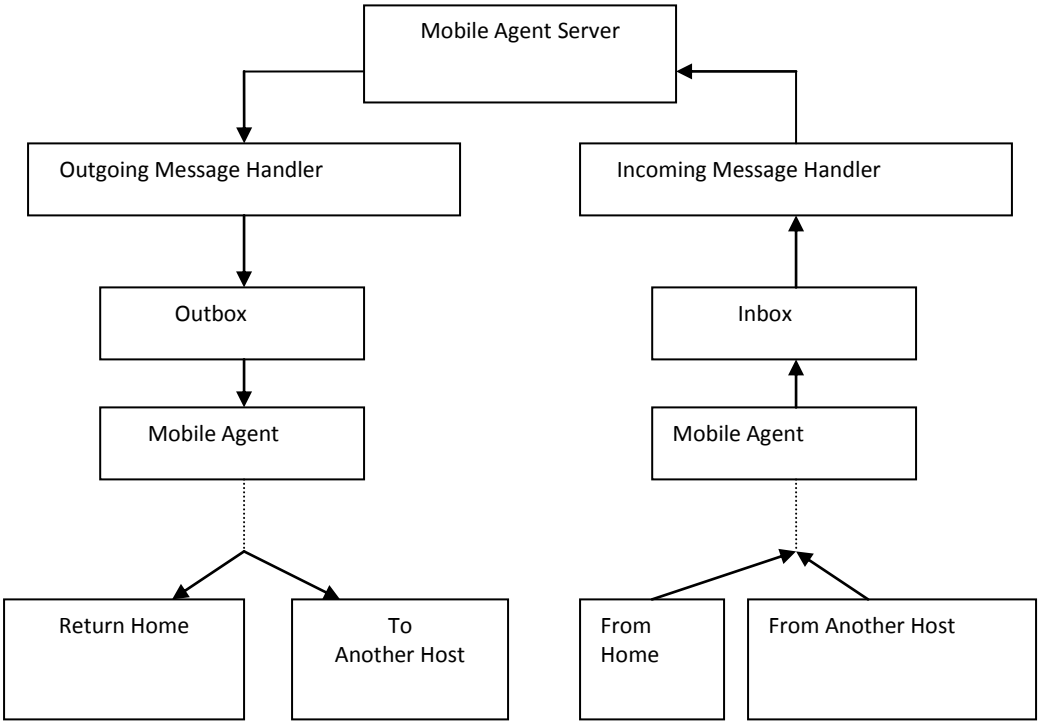

Figure 2: Mobile Agent Server Architecture

3.3 Transfer of Mobile Agents between Servers

When an agent finishes its task on a host, it either migrates to another host platform or returns to its home host. Figure 3 shows how agents are dispatched from one host to another using object serialization/de-serialization. Object serialization is the process of converting a data structure or object into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and "resurrected" later in the same or another computer environment [14]. When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. The process of restoring the object is known as de-serialization.

3.4 Message Exchange between Agents on the Same Server

The host platform facilitates exchange of messages between two or more agents on the host and between different hosts. Figure 4 shows the roles of the server in intra/inter server agents' communication while figure 5 illustrates message exchange between agents on different servers.

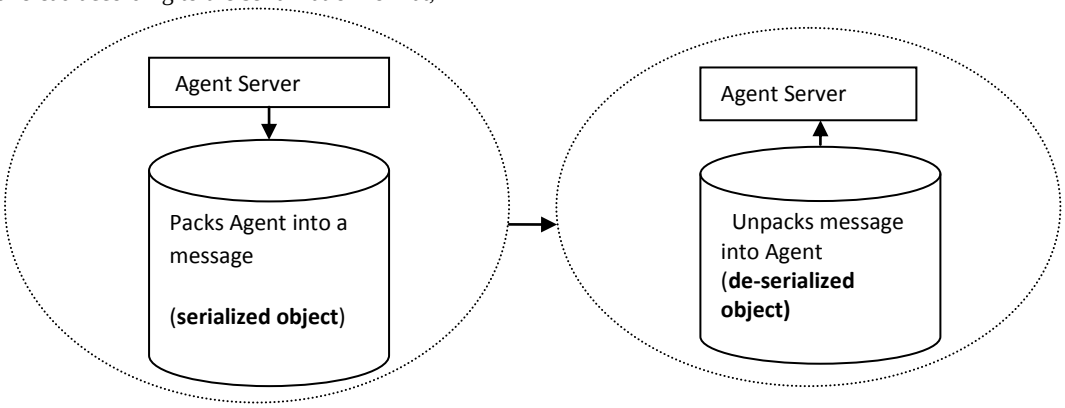

Figure 3: Transfer of mobile agents between servers

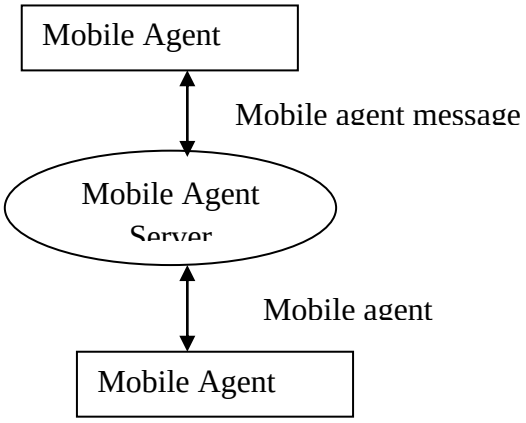

Figure 4: **Message exchange between agents on the same server**

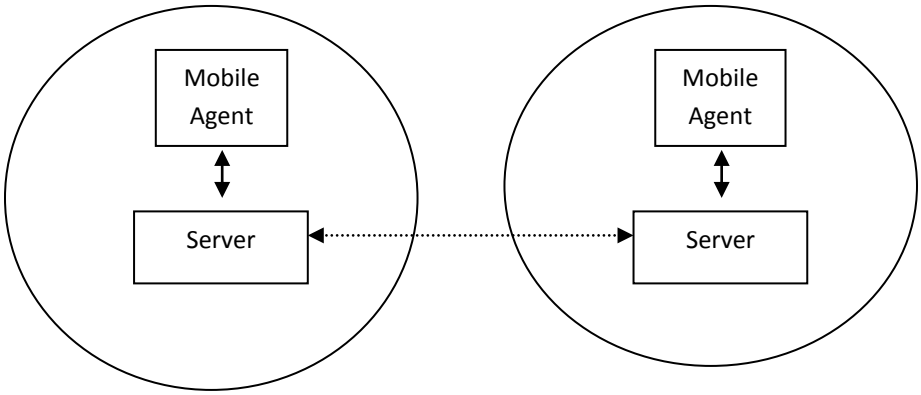

Figure 5: Message exchange between agents on different servers

3.5 Agent's Security Challenge

The mobile agent architecture given in figures 1 to 5 shows that agent is exposed to a lot of abuses and security chalenges while roaming the network to perform its duties. A lot of research has been dedicated to address the security problems in mobile agent systems. This research differs in its aim, emphasis, base, and technique. Some works are towards building the foundations for the security of a mobile agent system; some propose security mechanisms following different approaches; some focus on introducing security mechanisms into the architectures of mobile code systems; and others implement real applications with security features.

However, there has been no successfull research dedicated to provide an intuitive protection framework for protecting mobile agents on the host server they execute on. This is the problem that this work addressed.

4. THE APPROACH

4.1 Agent's Itinerary

Several protocols have been proposed for the protection of the agent's itinerary. These protocols are usually based on storing the itinerary information in a separate data structure, and then use cryptographic mechanism to protect this data structure. When the itinerary information is stored and maintained outside the main agent code, the itinerary is said to be explicit, and its protection is significantly simplified.

The itinerary protection protocols presented to date do not support the protection of free-roaming agents. Agents are thus forced to travel static itineraries, that is, itineraries in which all host are known in advance. However, most useful and practical mobile agent-based applications should be based on using dynamic itineraries for free roaming agents, in which some host platforms are discovered at runtime.

4.2 Securing Dynamic Itineraries

In order to support free-roaming agents, we present a protection scheme based on introducing trusted locations into the agent's route. Introducing some trusted hosts into the itinerary makes it possible for our architecture to secure the information associated with dynamically located host.

4.2.1 Platform Registries

This paper proposes platform registries, which are digital security infrastructures, maintained by trusted certificate authorities, such as Baltimore Cyber Trust, Entrust Secure

Server Certification Authority, Equifax Secured Certificate Authority, RSA Data Security Inc, e.t.c., for the registration and insurance of trusted digital certificates to public mobile agents' host platforms

4.2.1.1 Assumptions Made with Regard to Trusted Platforms

With our introduction of trusted platform registries into agent's itinerary, we believe that the agent's task will be executed on that platform as expected. The architecture presented in this proposal assumes that a trusted platform will execute the agent's task honestly. Moreover, it is assumed that the trusted agent platforms are protected with appropriate mechanisms so as to prevent attacks from third parties that might alter the agent execution. In this case, security greatly depends on the mechanisms provided by the operating system and the good design of associated protocols. The proposed protocol also assumes the existence of a security infrastructure that allows agent developers and users to determine whether a platform is trustworthy or not. An example of such infrastructure can be found in [14]. In this work, the authors describe a security framework for a mobile agent system which incorporates a simple trust model. Such model is based on establishing trust relationships in a manner similar to that used in public key infrastructures to handle distributed authentication.

The identification of trustworthy platforms can also be grounded on simpler mechanisms, such as relying on real-world trust relationships. For example, the platform associated with a bank where the user has an account, or the platform from which the agent was first launched, can be safely introduced into the agent's itinerary as trusted platforms.

4.3 The Protection Architecture

This protection architecture aimed at protecting flexible dynamic mobile agent itineraries. The architecture pursues three main objectives:

Integrity: Platforms must not be able to modify the agent's itinerary undetectably.

Confidentiality: Platforms must not be able to access itinerary information of other platforms.

Authenticity: Platforms must be able to verify the identity of the agent owner.

4.3.1 The Idea

The general idea behind this protection architecture is to construct a chain of digital envelopes, each of which containing two elements: the data, and the encrypted key that allows decrypting the following envelope. The scheme is illustrated in figure 6 below.

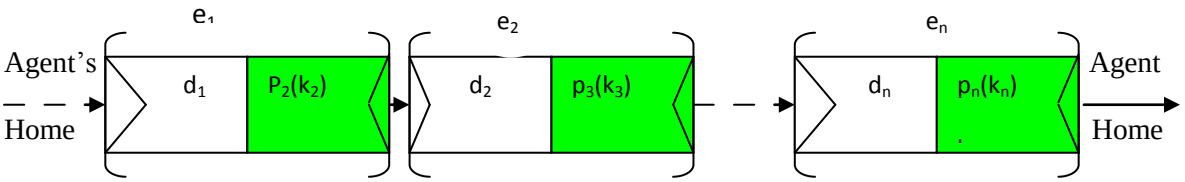

Figure 6: Chain of digital envelops for static itinerary

The envelopes shown in this figure 6 represent the entries of the protected itinerary. Every envelope, (ej) is encrypted using a random symmetric key (kj), and this symmetric key is in turn encrypted using the public key, (pj) of the host j, entitled to open the envelope. Thus, each envelope can only be decrypted by the intended host. Additionally, the envelopes can only be opened in the correct order, since the symmetric key used to decrypt an envelope is protected inside the previous envelope.

4.4 Support for Dynamic Itinerary

The problem of protecting dynamic itineraries in which not all public keys are known in advance makes it impossible to build a chain of digital envelopes as the one previously described in sub-section 4.3.1 above. More specifically, the hosts that will be visited to execute an itinerary are dynamically discovered by the agent at runtime. Therefore, the public keys of such platforms are not available when the itinerary is created. In order to solve this problem, we developed a novel protection scheme that is based on protecting the agent itinerary, on the dynamically discovered platforms, using the public keys that will be obtained from their corresponding platform registries. This scheme involves the use of platform registries discussed above by changing the chain of digital envelopes, as shown in figure 7.

It should be noted that, when an agent asks a dynamically located platform for its platform registry identifier, for the purpose of obtaining its public key, and this host falsely gives a wrong id, then it is either the registry will be unable to supply its id or the host itself will be unable to decrypt the message meant for it. In either of the two cases, the agent is protected from this malicious host.

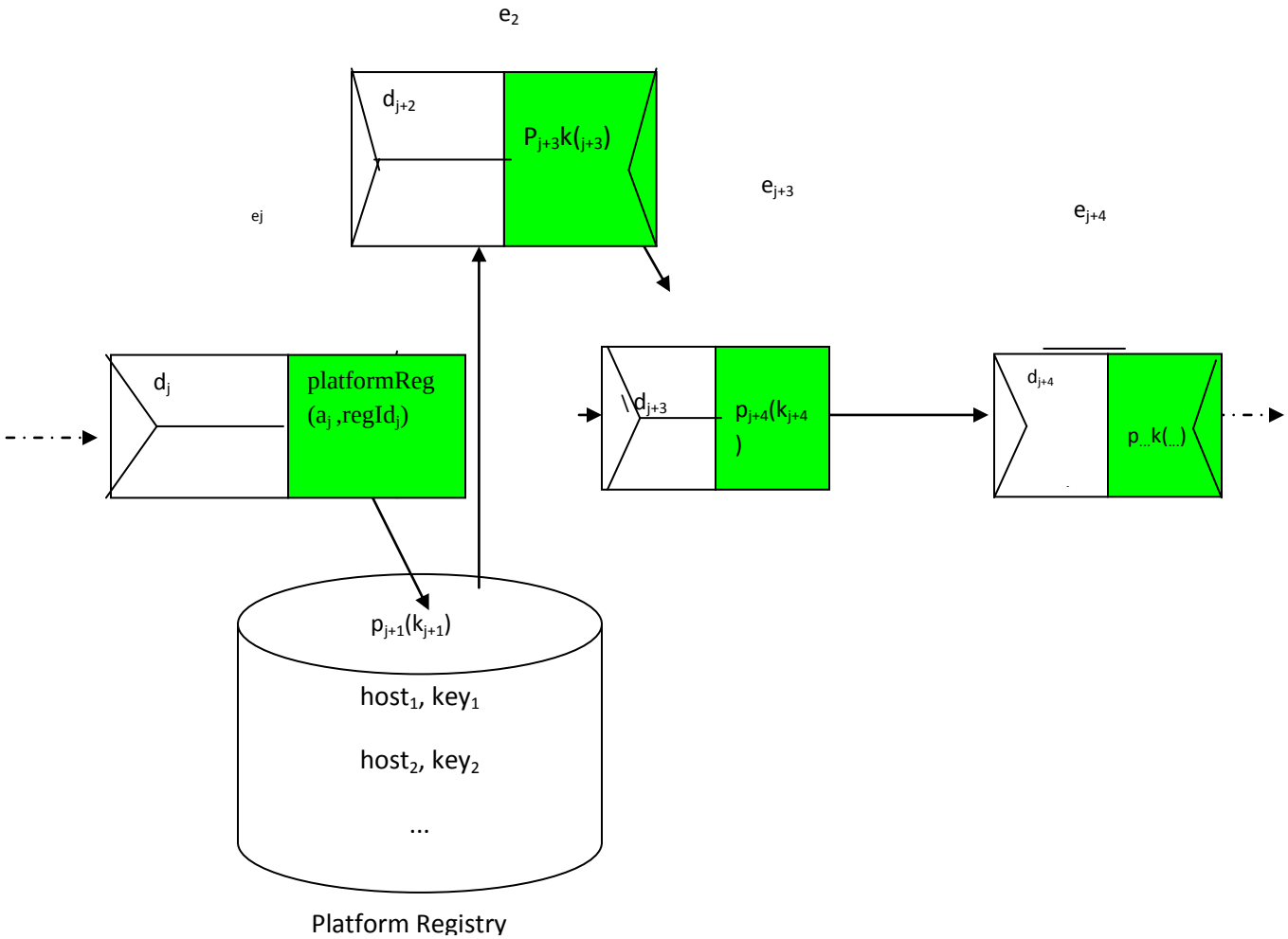

Figure 7: Chain of digital envelopes with *Platform Registry* to support dynamic itineraries.

4.5 Securing Itinerary

To secure an itinerary, a random symmetric key $k_1, k_2, ..., k_n$ is created for every itinerary host. Next, each possible migration from a host i to j denoted by t_{ij} is constructed as

$$t_{ij} = a_j, p_j(s_i(id, k_j, a_j)) \quad (1)$$

Where p_i denotes an asymmetric encryption function using the public key of platform i

a_j denotes the address of host j

s_i denotes a digital signature function using the private key of host i

From the equation, the transition from host i to host j contains the random symmetric key k_j associated with host j. This key will be used to encrypt the envelope of the protected itinerary when going to host j. In order to ensure that only platform j has access to k_j it is encrypted using the public key of host j. Finally, t_{ij} include a unique agent identifier id that is used to prevent replay attacks. It should also be noted that both id and k_j are signed by the agent's owner so that host j will be able to verify the agent's identity and integrity of the information it carries.

The equation in 1 used to build agent transitions t_{ij} from host i to j is useful only when the host j is not dynamically located at runtime, that is a_j and the public key of host j is

known at the agent home before the start of migration. If host j is located at runtime, then

$$t_{ij} = a_j, ? \quad (2)$$

Thus $p_j(s_i(id, k_j, a_j))$ is replaced with ?, an unknown value. This is due to the fact that a_j is not known at the time of creating the itinerary, and the public key needed to compute $s_i(id_j, k_j, a_j)$ is not available. In this case, when the host is located at runtime, its address a_j and its corresponding agent Platform Registry Identifier $(regId_j)$ will be obtained from this host for the purpose of getting its corresponding public key needed to compute k_j

The transition from the current host i to the dynamically located host j is now computed as

$$p_j = platformReg(a_j, regId_j)$$
$$t_{ij} = a_j, p_j(s_i(id, k_j, a_j)) \quad (3)$$

This equation 3 is equivalent to equation 1 above. The major difference is that k_j is replaced with the random symmetric key for the new host, a_j, generated from the public platform registry access function that takes an agent host address and its corresponding platform registry identifier and return the host public key, if the host is registered with the registry and null otherwise. It is not safe to obtain a host's public key, p_j directly from the host. The symmetric keys $k_1, k_2, ..., k_n$, which are used to encrypt the entries of the protected itinerary, are digitally signed by the agent owner. It ensures that attackers can neither generate their own itinerary entries nor modify existing ones.

Also, the unique agent identifier id prevents the reuse of entries previously generated by the same owner. Thus the integrity of the protected itinerary is guaranteed. Additionally, every transition to a host j includes address a_j of the host. Hence the hosts can verify that they were indeed part of the itinerary.

4.6 Simulation

In order to prove the viability of the proposed architecture, we implemented and performed two multi-phased experiments. The first part of the experiments was based on the proposed security architecture, simulating a simple mobile agent-based application on a hotel search and reservation system, using a local area network (LAN) of thirteen computers, with ten serving as host servers and the other three serving as platform registries. Each of the ten computers configured .with appropriate programs to make them malicious and very hostile to visiting mobile agents, was setup to act as mobile agent server to their respective hotels.

The system allows an individual to find the cheapest hotel in a given destination, taking into account the user preferences with regard to room facilities and guest services. The application allows the user to define search criteria. After defining the search criteria, a mobile agent is started that, first of all, queries a remote hotel search engine to obtain a list of the five cheapest hotels in the destination. The agent then visits each one of these hotels and checks their room availability for the desired rates, their room facilities, services, etc. In addition, the agent can also negotiate a special discount for long stays. Our dispatched agent randomly visited eight of the ten servers and eventually returned home with execution log on each server visited.

The second part of the experiment was identical to the first except that the dispatched agent employed obfuscation methods for its itinerary without the proposed new protection scheme.

Number of Mobile Agents on Hotel Reservation Assignment	Mobile Agents without Platform Registry Protection Protocols			Mobile Agents with Platform Registry Protection Protocols		
	Altered	Unaltered	Killed	Altered	Unaltered	Killed
13	9	2	2	0	9	4

Table 1: Analysis of Mobile Agents with and without Platform Registry

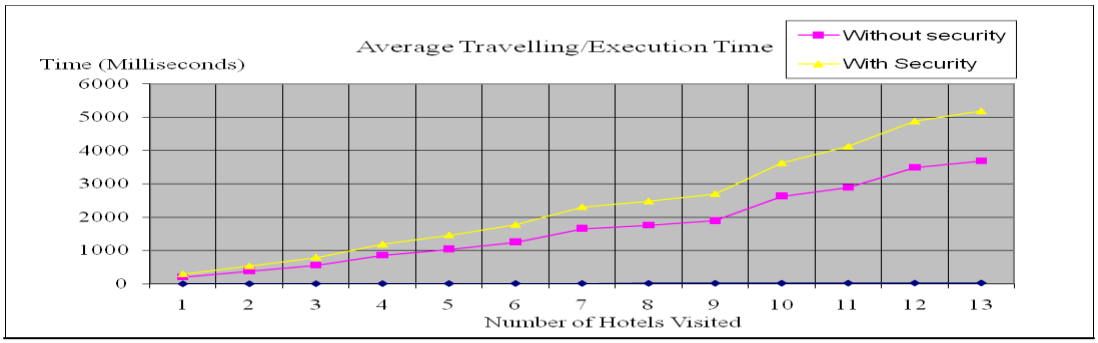

Figure 8: Agent on malicious hosts with and without security measures

4.7 Analysis of the Log Files

The agent code in our experiments was designed to return its execution log on each server visited. This enabled us to analyze its venerability to attacks by its hosts. The execution log files at each server were encrypted using this same proposed protocol with the public key of the agent home platform. The analysis of the log files showed that no successful attempts were made to read the agent's itinerary which included packets from the agent's previously visited servers and the packets for the next host to be visited from the current server. The current host server could only open the packet meant for it that was previously encrypted with its own public key obtained from one of the platform registries.

On the other hand, in our second experiment, where the obfuscation methods were used without our proposed protection architecture, the analysis of the returned log files showed that four of the host servers visited were able to access the agent's data packets that were not meant for these hosts. Table 1 illustrates mobile agents with and without platform registry protection scheme. As shown in table 1, our scheme made it difficult for the host to alter the packet.

On time performance factor, the execution time of the agents with our proposed scheme was compared to the execution time of the agents in our second experiment, that is, the roaming agents without our proposed protection framework, to determine if the proposed protection architecture increased the execution times considerably. We found out that the execution time of the agents with our protocol increased by approximately 40.6% of the unprotected agent's execution time as illustrated in figure 8. This increase is largely due to the time required to execute complex cryptographic protection protocol at platform registries and on each of the host platforms visited. We also found out that the time increase is a linear function of the number of hosts visited. The increase in time would be negligible if the actual task to be performed by an agent on each server is itself complex and time consuming.

5. CONCLUSION

The paper proposes the use of a chain of digital envelopes with platform registries to support dynamic agents' itineraries in open network environment. The scheme is capable of preventing a host server from gaining access to the information carried by a mobile agent that is not meant for it, that is, the current host. The proposed scheme exhibited better performance when compared to the results obtained from obfuscation methods in terms of data integrity and security. However, the proposed scheme consumes a little more time visiting platform registries and executing complex cryptographic functions than the obfuscation methods.

6. REFERENCES

[1] Wahbe R., S, Lucco, T.E. Anderson and Graham S.L., 1993, Efficient Software Based Fault Isolation, In Proceedings of the 14th ACM Symposium on Operating Systems Principle, pp 203-216, ACM

[2] Jacob Y. Levy, John K. Ousterbhout and Brent B Welch, 1997, The safe Tcl Security Model Technical Report, Sun Microsystems

[3] Sreekanth V., S Ramchandram and A. Govardhan, 2010, Mobile Agent Security and Key Management Technique, Journal of Computing, Vol. 2, Issue 9, ISSN 2151-9617

[4] Neelesh Kumar Panthi and Chaudhari Neelesh Kumar Panthi, 2010, Securing Mobile Agent using Dummy and Monitoring Mobile Agent, International Journal of Computer Science and Information Technologies, Vol 1 (4), pp 208-211

[5] Sarvarnl Islam Rizvi, Zinat Sultana, Bio Sun and Mid Washiqul Islam, 2010, Security of Mobile Agent in Ad Hoc Network using Threshold Cryptography, World Academy of Science, Engineering and Technology, Vol 30, pp 424-427

[6] Sreekanth V., Ranchandra S., and Gavardhan A., 2008, A Novel Approach for Securing and Integrity of Mobile Agents, ICCBN, IISC, Bangalore

[7] Tomas Sander and Christian F. Tschudin, 1998, Protecting Mobile Agent against Malicious Hosts, In Giovanni Vigna, Mobile Agent Security, pp. 44-60, Springer-Verlag, Herdeberg Germany

[8] Gray R.S., 1995, A Transportable Agent System, In proceedings of CIKM 95 Workshop on Intelligent Information Agents

[9] Dierks T. and Rescorla E., 2006, The Transport Layer Security Protocol Version, In RFC 4344, IETF

[10] Vigna G., 1998, Cryptographic Traces for Mobile Agents, In Mobile Agent and Security, Vol., 1419 of Lecture Notes in Computer Science, pp 137-153, Springer Verlag

[11] Hohl F., 1998, Time Limited Blackbox Security: Protecting Mobile Agent from Malicious Hosts, In Mobile Agent and Security, Vol., 1419 of Lecture Notes in Computer Science, pp 92-113, Springer Verlag

[12] Sander T. and Tschudin C.F., 1998, Protecting Mobile Agents against Malicious Host, In Mobile Agent and Security, Vol., 1419 of Lecture Notes in Computer Science, Springer Verlag

[13] Tan H.K. and Morean L 2001, Trust Relationships in a Mobile Agent System, In Mobile Agent, Vol., 2240 of Lecture Notes in Computer Science, pp 15-30, Springer Verlag

[14] Carles Garrigne Ollivera Bellaterra, 2008, Contribution to Mobile Agent Protection, PhD Thesis, Universtat Ant Onoma, De Barcelona

[15] Wikipedia the Free Encyclopedia Serialization, http://en.wikipedia.org/wiki/serialization