Securing Mobile Agents for Electronic Commerce: an Experiment

Anthony H. W. Chan, Caris K. M. Wong, T. Y. Wong, Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N. T.
Hong Kong
Fax: (852)-2603-5302
Email: {hwchan1, kmwong1, tywong, lyu}@cse.cuhk.edu.hk

Abstract
Mobile software agents are becoming a major trend of distributed systems in the next decade. Electronic commerce and information retrieval are two prospective applications of mobile agents. Nevertheless, security is a crucial concern for such systems. Attacks to agents by malicious hosts are the most challenging part of the problem unsolved. In this paper, we build a *Shopping Information Agent System (SIAS)* based on mobile agent technology. We discuss possible security attacks by malicious hosts to agents in the system, and present our solutions to prevent these attacks. We analyze the security of our solutions, and evaluate the performance overhead introduced.

1. INTRODUCTION

Mobile agents are autonomous software agents that travel in a computer network to execute and perform tasks on different hosts for their owners. Several reasons for deploying mobile agents have been suggested, such as [1]:

a) they reduce the network load;
b) they overcome network latency;
c) they encapsulate protocols;
d) they execute asynchronously and autonomously;
e) they adapt dynamically;
f) they are naturally heterogeneous; and
g) they are robust and fault-tolerant;

A lot of mobile agent platforms have been developed around the world, such as Aglets [2] from IBM, Concordia [3] from Mitsubishi, and the Mole [4] from University of Stuttgart. Prospective applications of mobile agents include electronic commerce, information retrieval and network management. Nevertheless, security is one of the blocking factors of the development of these systems. The main unsolved security problem lies on the possible existence of malicious hosts that can manipulate the execution and data of agents [5].

In this paper, we build a *Shopping Information Agent System (SIAS)* using the Concordia architecture. The system is useful to collect and compare the prices of a set of products specified by users from different seller hosts in a electronic market. We address the security issues of the system, describe possible attacks by malicious hosts to the system, and devise and implement our solutions to protect the system against these attacks.

The paper is organized in the following way: Section 1 (this section) is an introduction of the paper. Section 2 discusses the security issues of mobile agents in general, with focus on the problem of malicious hosts. Section 3 gives an overview of SIAS. Section 4 addresses the security problems and solutions of SIAS. An evaluation of the security solutions for SIAS is given in Section 5. Finally, Section 6 concludes the paper and suggests some directions for future work.

2. SECURITY ISSUES OF MOBILE AGENTS

Any distributed system is subject to security threats such as eavesdropping, corruption, masquerading, denial of service, replaying, and repudiation, so is a mobile agent system. Therefore, issues such as encryption, authorization, authentication, non-repudiation should be addressed in a mobile agent system. Moreover, a secure mobile agent system must protect the hosts as well as the agents from being tampered by malicious parties.

2.1 Host Security

In a mobile agent system, hosts continuously receive agents and execute them. Hosts may not be sure where an agent comes from, and are at the risk of being damaged by malicious code or agents (Trojan horse attack). This problem can be effectively solved by strong authentication of the code sources, verification of code integrity, and limiting the access rights of incoming agents to local resources of hosts, such that damages to hosts by malicious agents are limited to the resources available to agents. The solution is realized in the Java security model [6].

2.2 Agent Security

The main security challenge of mobile agent systems lies on the protection of agents. When an agent executes on a remote host, the host is likely to have access to all the data and code carried by the agent. If by chance a host is malicious and abuses the code or data of an agent, the privacy and secrecy of the agent and its owner would be at risk.

There can be seven types of attack by malicious hosts [5]:

i. Spying out and manipulation of code
ii. Spying out and manipulation of data
iii. Spying out and manipulation of control flow
iv. Incorrect execution of code
v. Masquerading of the host
vi. Spying out and manipulation of interaction with other agents
vii. Returning wrong results of system calls to agents

There are a number of solutions proposed to protect agents against malicious hosts [7], which can be divided into three streams:

i. *Establishing a closed network*: limiting the set of hosts among which agents travel, such that agents travel only to hosts that are trusted.
ii. *Agent tampering detection*: using specially designed state-appraisal functions to detect whether agent states have been changed maliciously during its travel.
iii. *Agent tampering prevention*: hiding from hosts the data possessed by agents and the functions to be computed by agents, by messing up code and data of agents, or using cryptographic techniques

None of the proposed solutions solve the problem completely. A closed network effectively decreases the chance of an agent being attacked by unknown malicious hosts, however, it also limits the mobility and ability of agents. Agent tampering detection is possible but requires subsequent efforts to recover from attacks, and is not effective enough for agents that carry out critical missions. Agent tampering prevention would be most effective and useful, but is not yet feasible for all functions. Most researchers in the area are seeking a better solution, and there is no general methodology suggested to protect agents. In the mean time, developers of mobile agent systems have to develop their own methodologies according to their own needs.

Apart from attacks by malicious hosts, it is also possible that an agent attacks another agent. However, this problem, when compared with the problem of malicious hosts, is less important, because the actions of a (malicious) agent to another agent can be effectively monitored and controlled by the host on which the agent runs, if the host is not malicious.

3. OVERVIEW OF THE SHOPPING INFORMATION AGENT SYSTEM (SIAS)

This section presents an overview of *SIAS*, the *Shopping Information Agent System* that we have implemented. SIAS is a web-based mobile agent system that provides users with information of products for sale in an electronic marketplace. Advantages of SIAS include such properties as reduction of communication costs and delegation of tasks, which are the intrinsic advantages of a mobile agent system. It is written in the Java programming language and on top of the Concordia [3] application-programming interface (API). Here in this section we describe only the basic design, functionality and implementation of SIAS. The security issues of the system would be considered in Sections 4 and 5.

3.1 What the System Does

SIAS implements mobile agents to retrieve product information in an electronic market for users. An electronic market consists of hosts that sell products on the network. Each seller maintains a database that stores the prices and quantities in stock of different products available at that host.

SIAS keeps a roster of all hosts in the electronic market and a list of all products available in the market. It allows users to specify a set of products and the corresponding quantities they want to buy from the list. An agent is created for the user who has specified the list of products and quantities. The agent, on behalf of the user, will collect information about availability and price from hosts in the network. The path of the agent is determined before the agent is launched, according to the roster of hosts kept by the system. After the agent visits all hosts specified in its itinerary, it returns to its sender and reports the lowest prices and corresponding sellers. The design of the system is described in details in the next subsection.

3.2 Design

SIAS is designed using the object-oriented paradigm because the concept of objects is useful to describe agents. There are three main types of objects in the system, namely Agents, Launch Servers and Database Servers. We describe the object details and control flow of the system in this subsection.

3.2.1 Object Description

The three objects are designed as follows:

i. The *Agent* object: it keeps a list of product identification numbers (IDs) and a list of the corresponding quantities specified by users. It is responsible to travel around the network and collect product information for users from different hosts.

ii. The *Launch Server* object: it is responsible for creating agents for users, sending the agents to the network, and receiving the agents when they finish visiting all the hosts specified in their itineraries.

iii. The *Database Server* object: it stores the information of products available at a particular host, (each host has its own instance of this object) and is responsible for retrieving required information for an agent when it arrives to the host.

Figures 1, 2 and 3 show the details of the objects respectively.

```
The Agent :

   attributes:
      - List of product IDs
          To store the product IDs inputted by users
      - List of product quantities
          To store the quantities of the corresponding products
      - List of product entries
          To store the product entries retrieved from the Data
          Base Server.

   methods:
      - doNothing
          When arrives at a host, the agent do nothing and
          then leaves.
      - queryServer
          When this method is invoked by the Data Base
          Server, the agent queries the Data Base.
      - reportCheapest
          When this method is invoked by the Launch Server,
          the agent calculates the cheapest purchasing
          combination and reports the result as a string.
```

Figure 1: Object details of Agent.

```
The Launch Server :

   attributes:
      - HashTable info
          It is used to map agent's ID to a string. The string is
          a report generated by the agent.

   methods:
      - createAgent
          Creates an agent with attributes initialized according
          to users' input
      - handleAgent
          When an agent arrives at the Launch Server, the
          server will invoke the "reportCheapest" of the
          incoming agent and stores the result string to the
          hashtable for the user to query.
```

Figure 2: Object details of Launch Server.

```
The Data Base Server :

   methods:
      - handleAgent
          When an agent arrives at the Data Base Server, the
          server will invoke a series of methods which may be
          methods of the incoming agent or not.
```

Figure 3: Object details of Database Server.

3.2.2 Flow Description

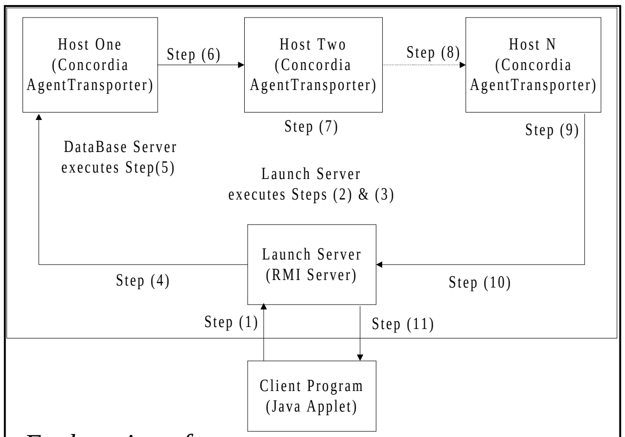
When user makes a request for product information, an Agent is constructed with the product and quantity lists initialized properly by the Launch Server, and the agent will start its tour on the network. Whenever it reaches a host with a Database Server, it stays there, collects information of user-selected products, and then goes to another host. When it has visited all the hosts that are specified in its itinerary, it will calculate the lowest prices, and finally reports to user. The detailed control flow of the system is illustrated in Figure 4.

3.3 Implementation

SIAS is implemented using the Java programming language with the support of the Concordia API [3]. The choices of programming language and supporting API, together with some other implementation details, are discussed in this subsection.

3.3.1 Choice of programming language

We choose Java to be the programming language for implementation of SIAS with two main reasons, apart from its object-orientation and portability features. First, most mobile agent APIs currently available, including Concordia and Aglets [2], are built on top of Java. Second, Java provides an API that helps us to implement security measures for our system.



Host One (Concordia AgentTransporter)
Step (6)
Host Two (Concordia AgentTransporter)
Step (8)
Host N (Concordia AgentTransporter)
Step (7)
Step (9)
DataBase Server executes Step(5)
Launch Server executes Steps (2) & (3)
Launch Server (RMI Server)
Step (4)
Step (10)
Step (1)
Step (11)
Client Program (Java Applet)

Explanation of steps:

1. *Client program launches a request to the Launch Server object upon user input using Java Remote Method Invocation (RMI);*
2. *Launch Server creates an Agent object;*
3. *Launch Server initializes the agent with user-specified products and quantities, and the itinerary of agent;*
4. *Launch Server sends the agent to the network;*
5. *Database Server on Host One retrieves the required information for the incoming agent;*
6. *Agent goes to the next destination;*
7. *Database Server on Host Two repeats Step (5);*
8. *Agent goes to other hosts in the itinerary;*
9. *Database Server on each host repeats Step*

Figure 4: Control flow of SIAS.

3.3.2 Why Choose Concordia

We choose the Concordia mobile agent API, among others like IBM Aglets Software Development Kit (ASDK) because it is simple and easy-to-use. This saves us a lot of time from developing the system. However, communication between agents would be difficult to implement with Concordia, yet it does not affect our choice because there is little communication between agents in SIAS.

Another important point in choosing Concordia is that it allows easy manipulation of execution of agent codes. Therefore, we can simulate a malicious host that does not execute an agent in the intended way easily.

3.3.3 Other implementation details

Agent objects are instantiated by the Launch Server object. The Launch Server object fills the product list and quantity list of the created agent, determines the itinerary of agent and then sends the agent out to the network.

Referring to Figure 4, there is an object on each host called AgentTransporter. This is introduced by the Concordia API, and it is responsible to listen for incoming agents. When an agent arrives, the AgentTransporter raises an event signal, and invokes the Database Server or Launch Server to handle the agent. The Database Server use Java Database Connectivity (JDBC) to handle the connectivity between agents and the database that store the product information at each host.

3.3.4 Running SIAS

We have made SIAS accessible from the World Wide Web, at the following URL:

http://www.cse.cuhk.edu.hk/~lyu9905/sias/enter.html

However, for technical reasons, we cannot keep the Concordia server running all time of the day. Interested parties may send us an email so that we can turn the server up for running SIAS.

4. SECURITY DESIGN OF SIAS

SIAS is a web-based system, attacks from the Web to the system are likely, and security is an important issue of the system design. Moreover, system security is of crucial importance to applications in an electronic marketplace, where money transaction is concerned. This section describes the security challenges of SIAS, and presents a simple but original approach to solve the problems.

SIAS is a mobile agent system, and is therefore subject to all kinds of attacks described in Section 2. Both host security and agent security would be issues of SIAS. However, since we have built SIAS using the Java programming language, which provides strong security mechanisms to protect hosts against malicious programs or agents through the use of Java Virtual Machine (JVM) and sandbox, the host security problem is very much simplified and solved. On the other hand, agent security needs much more concerns. In what follows, only agent security of SIAS against malicious hosts would be discussed.

4.1 Security Problems of SIAS

We start our discussion by giving a set of security requirements for SIAS. There are three primary requirements:

1. *Integrity*: the query results reported by an agent

must truly represent the market prices of the products and at the quantities specified by the user.

2. *Confidentiality*: information collected from a store by an agent should not be revealed to other hosts or agents.
3. *Authenticity*: an agent must visit and collect information truly from the list of stores specified by users.

Without special design, all these requirements can be violated by actions of a malicious host. There are four possible types of such attacks to agents that can compromise the security of the system, namely modification of the query products of an agent, modification of the query quantities of an agent, spying out and modification of query results, and modification of the itinerary of the agent.

4.1.1 Modification of query products

The list of products specified by user is stored as the product ID list attribute of an Agent object, in plain text form. When an agent goes to a malicious host, the malicious host can change the product list the agent wants to query. When the agent later go to another host, the later host will respond to the changed products of query and report wrong information. This violates the integrity of the queries.

4.1.2 Modification of query quantities

Similar to the modification of query products, when an agent goes to a malicious host, the malicious host can change the quantities of products the agent want to query, which is simply in plain text form. When the agent goes to another host, the later host will respond to the modified quantities of query, and report wrong information. This also violates the integrity of queries.

4.1.3 Spying out and modification of query results

Agents carry query results also in plain text form. Therefore, when an agent goes to a malicious host, the malicious host can spy out and modify the results that the agent has collected from previous hosts in such a way that the changed results would favor the malicious host itself. For example, a malicious host may raise the prices quoted by other hosts, to convince the user that it is selling at the lowest price, which is not true. This violates the confidentiality and integrity of query results.

4.1.4 Modification of itinerary of an agent

The itinerary of an agent is accessible to hosts that have control over the Concordia platform where the agent lands and executes. When an agent goes to a malicious host, the malicious host can modify the path of the mobile agent so that the agent will go to a host not specified by user. This violates the authenticity requirement of the system.

The above are only a subset of possible attacks. There are other attacks such as replaying of query results and masquerading of hosts. However, these attacks are more complex, and require more efforts for both attack and defense. For the time being, we consider the four simple attacks only.

4.2 Our Solutions to problems

Having figured out the four system vulnerabilities described above, we have to implement mechanisms to protect our systems against exploitation of these vulnerabilities. As stated in Section 2, there is currently no good solution to mobile agent security in general. Therefore, we have to devise our own mechanisms to defend against possible attacks.

We develop a simple but original approach to protect agents in SIAS against attacks from malicious host, based on cryptographic techniques. It is actually a mixed approach of the solutions, i.e., *establishing a closed network, agent tampering prevention and agent tampering detection,* discussed in Section 2.

1. *Closed network*: we introduce a new object, namely key server or KeyServer, into our system, which provides a public key infrastructure for agents and hosts in the system. Each agent or host should have a public key certificate registered to the key server for encryption or decryption purposes later on. The Launch Server generates a pair of keys for each agent created, and registers the public key of the agent with a unique agent identification number to the key server at run-time. On the other hand, each host must identify itself and register its public key to the key server before, by such means as a formal paper writing. This in effect establishes a closed set of hosts registered and known to the key server. Agents are then confined to travel among a closed network form by these hosts.

2. *Agent tampering prevention*: to protect query integrity, an agent can digitally sign its list of products and quantities using its private key, before it is launched. A host receiving the agent should verify the product and quantity lists with

the signatures. Since only the Launch Server possess the private key for the agent, malicious hosts would not be able to fake the signature of the product and quantity lists.

Moreover, each host should encrypt the query results returned to the agent with the public key of the agent. Therefore, only the Launch Server can decrypt the query result, and confidentiality of query results is achieved. Furthermore, each host should digitally sign the query result it provides to the agent to ensure integrity and authenticity of the query result returned.

3. *Agent tampering detection*: the itinerary of an agent is an variable hidden by the Concordia system and normally not accessible. However, hosts can actually have access to the itinerary of an incoming agent by controlling the execution of the Concordia agent transporter. A malicious host would be able to change the itinerary of the agent. As before, the straightforward method of protecting the itinerary is to encrypt it. However, this requires modification of the agent transporter of Concordia, which is not desirable to us.

   We work around the problem by making the itinerary of an explicit attribute of an agent. When an agent arrives at a host, the host should read the itinerary of the agent, and encrypt the itinerary using its own private key to form encrypted itinerary EI_1. Then when the agent arrives at a second host, the second host should encrypt, with its own private key, EI_1 concatenated with the itinerary it reads from the agent. This keeps on to form a chain of encrypted itineraries. When the agent returns to the Launch Server, the Launch Server will decrypt the chain of encrypted itineraries using the public keys of the hosts to check the consistency of all itineraries and check with a copy of the original itinerary it saves before launching the agent. If a malicious host ever changes the itinerary of the agent, it is likely to be reflected in the encrypted itinerary chain and detected finally.

Figure 5 illustrates the changes introduced to SIAS for the security solutions described above, and Figure 6 illustrates the control flow of security-enhanced SIAS. Note that the encryption algorithm chosen is the most common RSA algorithm [8].

5. EVALUATION OF THE SECURE SIAS

In this section, we evaluate the security design we implemented in Section 4. There are two aspects to

I. {Product ID list} *changed to*:
{Product ID list}•sig_A({Product ID list})

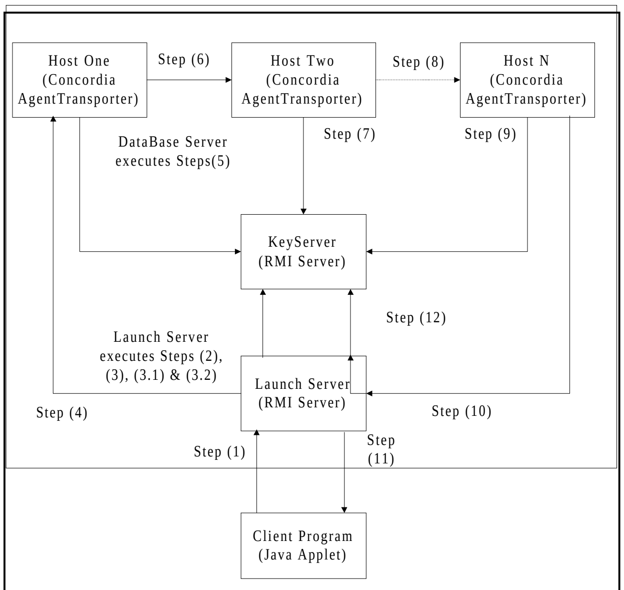
II. {Product Quantity list} *changed to*:
{Product Quantity list}•sig_A({Product Quantity list})

III. {Query result} *changed to*:
D_A({Query result}•sig_H({Query result}))

IV. *New attribute (chain of encrypted itineraries)*:
E_{HN}($E_{H(N-1)}$(…E_{H2}(E_{H1}(Itinerary at Host 1) • Itinerary at Host 2) • … Itinerary at Host N-1) •Itinerary at Host N)

Key
A: agent;
H: host;
H(k): k-th host visited by the agent;
sig_X(Y): digital signature of Y using the private key of X;
E_X(Y): the ciphertext of Y encrypted by the private key of X;
D_X(Y): ciphertext of Y encrypted by the public key of X.

Figure 5: Changes introduced to secure SIAS


Host One (Concordia AgentTransporter)
Step (6)
Host Two (Concordia AgentTransporter)
Step (8)
Host N (Concordia AgentTransporter)
DataBase Server executes Steps(5)
Step (7)
Step (9)
KeyServer (RMI Server)
Step (12)
Launch Server executes Steps (2), (3), (3.1) & (3.2)
Launch Server (RMI Server)
Step (4)
Step (10)
Step (1)
Step (11)
Client Program (Java Applet)

Explanation of additional / modified steps:
3.1. Launch Server generates a key pair for agent;
3.2. Launch Server signs the product and quantity lists for agents and registers the public key of agent to Key Server;
5. Database Server on Host One retrieves public key of agent from Key Server, and verify the signatures of product and quantity lists of agents. Then, the Database Server retrieves the required information for the incoming agent, signs the results using its own private key, and encrypt the results using the public key of agent, and also starts the chain of encrypted itineraries for agent;
11. Launch Server decrypts the query results, and verifies the signatures of the query results. It also detects change of agent itinerary by decrypting the chain of encrypted itineraries, and finally reports the cheapest purchasing combination to client program.
12. Launch Server deletes the public key entry of the finished agent from the key server.

Figure 6: Control flow of security-enhanced SIAS.

evaluate. First, we analyze the security provided to SIAS by the additional measures. Then, we measure the performance overhead introduced to the system by such measures.

5.1 Security Analysis

The security of the additional measures lies mainly on the introduction of a key server that facilitates the use of public key cryptography. Assuming the key server and the communication channel with the it are secure enough, which can be justified by the popularity of Kerberos [9] and Secure Socket Layer [10], the closed network we want can be built effectively.

Furthermore, if the keys of agents are managed properly, the prevention of modification of the signed product and quantity lists of an agent by a malicious host is supported by the security of the RSA encryption algorithm, of which the difficulty to break is equivalent to the factoring problem. The time complexity for breaking the system depends on the length of the key in number of bits. The longer the key is, the more secure would be the system. In our implementation, we have chosen a key length of 128 bits. This would be sufficiently secure for domestic purpose.

Similarly, a malicious host would understand or modify the encrypted query results collected by an agent from another host at the same complexity. Therefore, integrity of queries, and confidentiality and integrity of query results, as described in Section 4, can be achieved by prevention of tampering.

For the detection of modification to itinerary of an agent by a malicious host, suppose there is only a single malicious host, out of N hosts, that wants to modify the itinerary of an agent. Since the encrypted itineraries are chained together, with one encapsulating another, the malicious host would need to fake all the (N-1) encrypted itineraries from other hosts to avoid being detected, which would be too complex to an ordinary attacker. Therefore, the itinerary of the agent can be assured, and authenticity achieved.

However, as mentioned in Section 4, there do exist other attacks that we have not considered completely, such as replaying attacks, timing attacks, and repeated cipher-text attacks. Protection against these attacks would be a direction for future work on SIAS.

5.2 Performance measurement

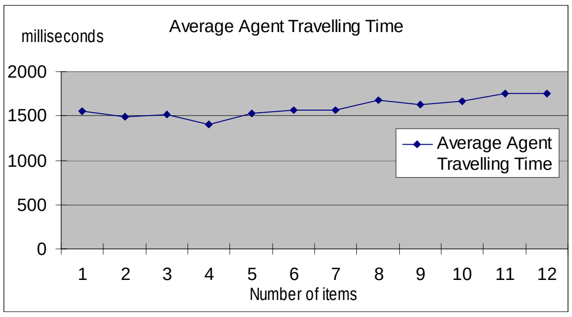
We have tested the times for SIAS to launch a single agent before and after implementation of the security mechanisms described in Section 4. Round trip times (RTTs) required for an agent to travel around an electronic market, consisting of three hosts, are measured under different situations. Queries of different sizes (number of product items) have been tested. RTTs measured are plotted against the query sizes in Figure 7. Each value represents the average of three corresponding measurements.

Figure 7(a) shows the results for the SIAS implementation without security measure implemented. The RTT increases very slightly with the size of query. The overhead introduced by each additional item in average is only about 250/6 = 41.7 milliseconds. This can be explained by the small change in delay of database query with different query sizes.

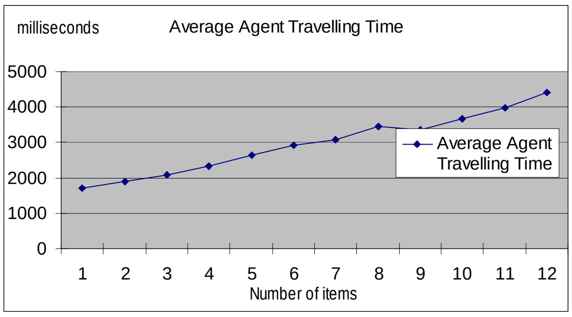
On the other hand, Figure 7(b) shows that for the security-enhanced SIAS, the RTT increases very fast and linearly with the size of query. The overhead introduced by each additional item of query is about 250 milliseconds, which is about six times the overhead of the system without security measure. This significant overhead can be explained by the extensive use of the RSA algorithm to encrypt and decrypt each item, which is time consuming, especially when the key is long. However, a longer key gives stronger protection to the system. Therefore, we see a trade-off between performance and security for SIAS.

In addition to measuring the performance overhead introduced by the security measures, we also simulate malicious hosts trying to modify the product list and itinerary of an agent in SIAS, and measure the overheads introduced by the actions of malicious hosts. The results are reported in Figures 7(c) and 7(d).

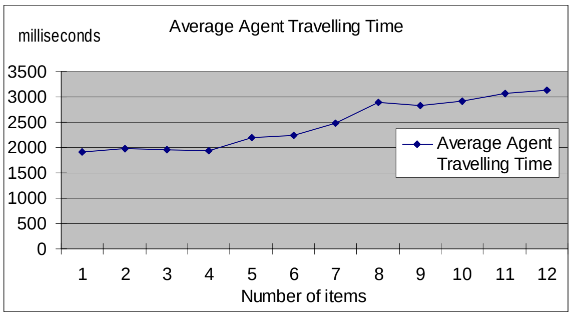
Both graphs show that an agent takes more time to travel around when there is attack from malicious host, compared with the measurements in Figure 7(a). The RRTs in (d) is slightly larger than those in (c) in general, because agent itinerary is actually an internal property of an agent, and it takes the malicious host extra time to access the itinerary. The delays of agents by malicious hosts suggest that the agent round trip time may also be used as a measure for tampering detection.



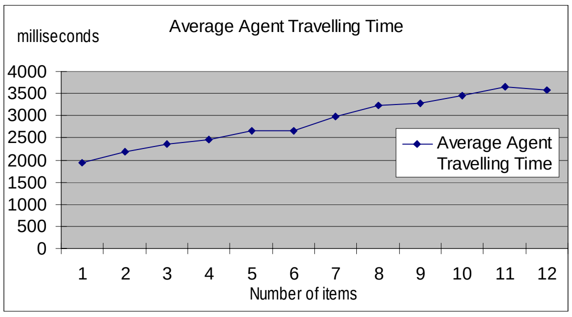
(a): SIAS without security measures.



(b): SIAS with security measures implemented.



(c): SIAS with a malicious host trying to modify product list of agent.



(d): SIAS with a malicious host trying to modify the itinerary of agent.

Figure 7: Round trip time measurements for an agent in SIAS with different configurations.

6. Conclusion

We studied the technology of autonomous mobile agents and discussed the problem of malicious hosts in a mobile agent system. We implemented SIAS as a sample application of mobile agents, which reduces communication cost and allows delegation of tasks. We addressed some security problems of malicious hosts in SIAS, and developed a primitive approach to protect the agents. We analyzed the security of our approach, and believe it is strong enough for domestic purpose. We measured the performance overhead of the security measures, saw a trade-off between performance and security for SIAS, and learned that it takes time for a malicious host to attack an agent.

In the future, we would keep on improving the security of SIAS, and seek a general methodology to solve the problem of malicious hosts. We would scale up SIAS, and evaluate the performance of SIAS with different numbers of hosts (sizes of the electronic market). We believe with enhanced security mechanisms, mobile agents can serve as an important technology in future distributed systems.

References

[1] Danny B. Lange and Mitsuru Oshima. "Seven Good Reasons for Mobile Agents", *Communications of the ACM*, p.88 - 89, 1999 Mar.

[2] "IBM Aglets Software Development Kit Homepage". http://www.trl.ibm.co.jp/aglets/

[3] "Concordia - Java Mobile Agent Technology". http://www.meitca.com/HSL/Projects/Concordia/

[4] "The Home of the Mole ". http://mole.informatik.uni-stuttgart.de/

[5] F. Hohl. "A Model of Attacks of Malicious Hosts Against Mobile Agents", *Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations*, p. 105 - 120, INRIA, France, 1998.

[6] "Java Security Architecture". http://java.sun.com/products//jdk/1.2/docs/guide/security/spec/security-specTOC.fm.html

[7] C. Tschudin. "Mobile Agent Security", *Intelligent Information Agents: Agent Based Information Discovery and Management in the Internet*, p. 431 - 446, Springer, 1999.

[8] R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, 1978 Feb.

[9] "Kerberos: The Network Authentication Protocol". http://web.mit.edu/kerberos/www/

[10] Alan O. Freier, Philip Karlton, and Paul C. Kocher, "The SSL Protocol Version 3.0". Internet Draft.1996 Nov 18.