An improved itinerary recording protocol for securing distributed architectures based on mobile agents

Guillaume Allée\textsuperscript{a}, Samuel Pierre\textsuperscript{a,*}, Roch H. Glitho\textsuperscript{b} and Abdelmorhit El Rhazi\textsuperscript{a}
\textsuperscript{a}Mobile Computing and Networking Research Laboratory (Larim), Canada
\textsuperscript{b}Ericsson Research Canada, Canada

Abstract. This paper proposes an improved itinerary recording protocol for securing distributed architectures based on mobile agents. The behavior of each of the cooperating agents is described, as well as the decision process establishing the identities of offenders when an attack is detected. Our protocol is tested on a set of potential attacks and the results confirm our assumption regarding offender designations and moments of detection. More precisely, the performance evaluation shows that our protocol detects the attack where there is collaboration between a platform on the cooperating agents’ itinerary and another on the mobile agent’s itinerary. As a result, this protocol constitutes a suitable option for electronic commerce applications where security concerns prevail over cost factors.

Keywords: Mobile agent, itinerary recording, distributed architecture, security protocol, electronic commerce, cooperating agents

1. Introduction

Wide-area networks allow the design of interesting shared applications that permit constant access to resources and information available on the Internet. However, programming these applications remains challenging: the main problem is due to the fact that the static features of the client-server architecture are not well adapted to the dynamic aspects and diversity of the systems that are available on the Internet. An alternative could be to replace the interactions between both entities by the mobility of an agent that migrates to hosts providing interesting data. Since the agents execute within the environment provided by the remote host, security concerns are paramount, especially for applications like electronic commerce. The problem we are addressing is that of verifying and recording the identities of the host platforms which have executed the agent. This problem is hard to solve with only one agent as malicious hosts can alter the agent’s data. One possible solution to detect these attacks would be to use co-operating agents in a protocol to record and verify the itineraries.

Greenberg et al. \cite{11} define the word agent as an autonomous application which has one or several purposes or a set of capabilities and could, if need be, collaborate or communicate with other agents.
or users. An agent is considered mobile when it can be transported from one machine to another on a network [12]. However, a mobile agent cannot transport itself: it is the executing platform that sends it to another platform through the network. The set of platforms where the agent is executed is called the agent itinerary.

A platform is concurrently a client and a server that allows the execution of an agent. The platform receives and sends the agent. It must be installed on all hosts where the agent stops. Platform protection techniques are required to shield the execution environment and platform resources against several types of attacks from a mobile agent. On the other hand, agent protection techniques must be used to protect the agent’s code and data from interference by malicious hosts [6,7,9,14,17].

Platforms are susceptible to a variety of possible agent attacks. A hostile agent can attempt to access resources without the platform authorization, it may consume too many resources (e.g. CPU or disk space) or disguise itself as another agent. Researchers have developed several techniques in order to protect the platform. Most of these are based on conventional protection approaches but there are some techniques that use new concepts, such as codes with proofs, which consists of asserting certain features of the agent’s code (such as memory protection) and supplying a proof of these features. When an agent arrives on a new platform, the latter verifies the compatibility of the proof and the agent’s code [2,3,8].

Platform attacks against the agent are even more critical since the platform has access to an agent’s components and can modify them in the absence of attack detection or prevention techniques. An attack [13] may take various shapes: a change that modifies the behavior of an agent or alters the stored results obtained from another platform, as well as the denial of services when a platform requests a task from an agent, namely when a platform refuses to send an agent to another platform. A simple yet effective attack on a mobile agent aims at preventing the agent from migrating to competitors’ servers. This particularly affects mobile agents with loose itineraries in comparison to agents whose itineraries are defined a priori [15]. The problem consists of verifying and recording the identities of the platforms, which have executed the agent in order to secure the mobile agent’s route. It is crucial to detect or protect the mobile agent against this type of attack in the field of e-commerce where the agent wants to visit all of the competitors’ servers.

Roth [16] identifies flaws in some cryptographic protocols that are targeted at protecting free-roaming mobile agents, e.g. mobile agents that are free to choose their next hop dynamically based on data they acquire during their execution. He found that some protocols succumbed to interleaving attacks which are impersonation or other deception involving selective combination of information from one or more previous or simultaneously ongoing protocol execution. He presented in [17] an approach which is robust against interleaving attacks. He also introduced algorithms and data structures meant to protect free-roaming mobile agents against attacks on data integrity and confidentiality.

In [4], Westhoff et al. propose a method to protect agent routes against attacks performed by a single platform as well as attacks conducted by collusions of cooperating malicious platforms. They define the route protection as the guarantee that “none of the visited stations can manipulate the route in a malicious ways, nor can they get an overview of the other sites the agent’s owner is contacting”. They suppose that the agent route is initially provided by the platform of origin and that it can be extended by a visited site. The information related to a mobile agent’s route is hidden in such a way that only the mobile agent’s platform of origin knows all the platforms to be visited. The platform of origin can be sure that all chosen platforms have been visited. Actually, this method differs from our method as we consider the agent route to be totally dynamic (i.e., the route is not fixed by the platform of origin).

Schneider [5] combines the replication and the voting concepts to ensure the proper functioning of the agent. The idea is that instead of having a computation performed solely by the mobile agent, multiple
copies are used to perform the same computation and the result is obtained by a result consensus from all of the platforms. If a malicious platform attacked the mobile agent, the existence of several clones would ensure the correct end result. The main advantage of this method is that it addresses the system failure. Its drawbacks is that additional resources are needed for agent replications. The method proposed in this paper also uses the concepts of replication to ensure security i.e. a cooperating agent executes partially the same code as the one executed by the mobile agent.

This paper proposes a security mechanism inspired by Roth [15] to protect mobile agents’ itineraries against a set of potential attacks from offender platforms in an electronic commerce context. Section 2 summarizes Roth’s protocol. Section 3 proposes a new itinerary recording protocol (based on Roth’s protocol) and assesses the effectiveness of this new protocol through the types of attacks it detects. Section 4 presents implementation details and results.

2. Summary and background of Roth’s protocol

Itinerary recording is carried out according to a protocol whose purpose is to record and verify the itinerary of a mobile agent with a cooperating agent, in an autonomous fashion, without permanently connecting a trusted platform to the network. The two agents’ owner platform connects to the network to send the agents to their first platforms, and then it disconnects itself until both agents return. As the itinerary is not pre-established, the mobile agent moves progressively while accomplishing its tasks and it chooses its next destinations during its execution on different platforms. The purpose of Roth’s protocol is to record and verify the mobile agent’s itinerary.

2.1. Definitions and main assumptions

Let \( H \) be the set of available platforms on a network. \( R \) is a subset of \( H \times H \) such as \((h_i, h_j) \in R \iff h_i \text{ and } h_j \text{ collaborate in order to attack the agent.} \) \( H_a \) and \( H_b \) are non-void subsets of \( H \) with \((H_a \times H_b) \cap R = \emptyset\). These two subsets are considered non-collaborating sets of platforms. Two agents, \( a \) and \( b \), are considered cooperating agents, when the itinerary of agent \( a \) contains only \( H_a \) platforms and agent \( b \)’s itinerary has only \( H_b \) platforms. \( h_a \) and \( h_b \) are current execution environments for agents \( a \) and \( b \). Basically, agent \( a \) could be attacked by platform \( h_a \) that executes it, but \( h_a \) could not directly attack cooperating agent \( b \). Moreover, platform \( h_b \) will not collaborate with \( h_a \) for this purpose. However, it is possible that two platforms on agent \( a \)’s itinerary collaborate to attack it.

Roth [14] justifies this choice by stating that it is as unrealistic to state that “all platforms are hostile and ready to collaborate with each other to attack the agent” as to claim that “all platforms could be securely used”. According to Roth, it is more realistic to state that:

- At a given moment in the life of an agent, a certain percentage of platforms could be considered dangerous;
- All dangerous platforms are not necessarily ready to collaborate with other platforms to attack an agent.

Roth [14] adds the following three assumptions to implement the concept of cooperating agents:

- The transport of agents from one platform to another is carried out through an authenticated channel (Assumption 1);
- The platforms supply authenticated communication channels to cooperating agents (Assumption 2);
- The agent has authenticated access to the identity of the remote platform with which it communicates, the platform on which it is executed, and to the previous platform on which it was executed (Assumption 3).
Definition: Let $h_i \in H$ be the platforms visited by agent $a$ and let $id(h_i)$ be the identity of platform $h_i$. Let $prev_i$ be the identity of the last platform where agent $a$ has been executed. Finally let $next_i$ be the identity of the next platform where the agent would like to go after platform $h_i$. The agent starts from platform $h_0$. Thus, if the agent moves a total of $n$ times, we must obtain $h_0 = h_n$, since agent $a$ returns to the platform of origin.

Initialization: Let $h_0$ be the platform of origin of agents $a$ and $b$. $h_0$ must be a trustworthy platform for agents $a$ and $b$. Each agent is sent to its first platform ($next_0$) through an authenticated communication channel (Assumption 1).

Step $i$, $i \in \{1, \ldots, n\}$:
Agent $a$ sends a message to agent $b$ containing $next_i$ and $prev_i$ through an authenticated channel (Assumption 2). Thus, agent $b$ learns $id(h_i)$ (Assumption 3) and verifies that

\[ id(h_i) = next_{i-1} \text{ and } prev_i = id(h_{i-1}) \]

If this is the case, $b$ records $next_i$ in agent $a$’s itinerary.

Fig. 1. Itinerary recording protocol.

2.2. Roth’s protocol

Let $a$ and $b$ be two cooperating agents, $H_a$ and $H_b$ two non-collaborating platform subsets of $H$. Each agent must return to its platform of origin at the end of its task. Agent $b$ records and verifies the itinerary of its cooperating agent $a$, according to the protocol described in Fig. 1.

The need for a cooperating agent is mainly justified by the fact that a malicious platform might modify the data held by an agent executing on that platform. Indeed, it could be argued that, if one supposes that the third assumption is valid, it is sufficient that the agent keeps the authenticated identity of each platform where it was executed. It could be supposed that it makes the same verifications as agent $b$ in Roth’s protocol. However, this solution is not secure. The following is a contrary example with an undetected attack. Host $h_i$ attacks the agent by sending it to host $h_f$, although the agent is supposed to go to platform $h_i+1$. The identity of host $h_f$ is added to the itinerary record, then host $h_f$ replaces the agent on its route by sending it to $h_{i+1}$, which is supposed to be honest. On $h_{i+1}$, the agent verifies the identity of the platform that sent it, which appears correct, since $h_f$ did not modify the itinerary record. On step $j > i + 1$, host $h_j$, which is allied with attacking platforms $h_i$ and $h_f$, modifies the itinerary record to remove the record of execution on host $h_f$. When the agent returns to its owner’s platform, its itinerary record appears correct although there are no records of the execution on host $h_f$.

3. Proposed new itinerary recording protocol

It should be noted that Roth’s protocol does not specify the manner by which agent $b$ moves. In our approach, we propose to move agent $b$ every time agent $a$ moves, i.e., when agent $a$ contacts $b$ to announce the identities of its next and previous execution platforms. Once this information is obtained,
agent \( b \) moves to another platform and waits to communicate with agent \( a \). We will specify the required assumptions for our protocol, describe the possible attacks, and indicate the manner in which they will be detected by the protocol. Finally, we will show the originality of our protocol in relation to Roth’s.

3.1. Assumptions

As stated above, agent \( b \) will move. There are two possibilities for agent \( b \)’s itinerary: it is either pre-determined or not. We propose to fix the itinerary of agent \( b \) at the beginning of the protocol. The advantage of this choice is that, with a fixed itinerary, we simplify the protocol and increase the likelihood of detecting the attack.

We suppose that agent \( a \) has a chronological list of platforms to be visited by agent \( b \). We call this list \( p_f, p_f[i] \) indicating the platforms visited by agent \( b \), later called \( h_b[i] \). Since a movement from agent \( a \) triggers the movement of agent \( b \), the number of platforms visited by agent \( b \) must be identical to the number of platforms visited by agent \( a \). However, the number of hosts visited by agent \( a \) cannot be predicted. To solve this problem, the number of platforms visited by agent \( a \) will be limited. This type of estimate is possible when “looking for the best price for a product” type of applications, where we know that the agent will visit a number of vendors, but we know neither the platforms (because a vendor has several platforms) nor the order in which the visits will take place. Therefore, it is impossible to establish an itinerary before the departure from the owner’s platform. Agent \( b \)’s itinerary consulted by agent \( a \) must be protected from modifications. This itinerary could, for example, be part of the agent’s data and thus, with the agent’s code, could be protected from modifications. This is possible with the owner’s signature guaranteeing the integrity of the agent and publishing its identity for the benefit of the platforms where the agents will eventually be executed.

We suppose that the set of platforms visited by agent \( a \) and the set of platforms visited by agent \( b \) are disjoint. The order in which agent \( b \) visited the platforms is irrelevant. It is nevertheless important to minimize the probability of an attack from a platform on agent \( b \)’s itinerary with a collaboration from a platform on agent \( a \)’s itinerary. However, it is difficult to quantify this risk, since \( a \)’s itinerary is not pre-determined. This is why agent \( b \)’s itinerary should be chosen randomly, from a list of platforms that do not offer the services sought by agent \( a \).

Figure 2 illustrates these assumptions using a collaboration example considered impossible in Roth’s protocol, which becomes possible in the protocol we suggest. We use the same assumptions as Roth’s (cf. Section 2.1). This figure shows the assumptions that we put forth concerning the platform collaboration to perform an attack. In our assumption, the platforms in \( a \)’s itinerary and \( b \)’s itinerary can collude to attack mobile agent \( a \). It is important to note that Roth’ assumption stipulates that the platforms in \( a \)’s itinerary and \( b \)’s itinerary cannot collude to attack mobile agent \( a \). For example, on the left of the figure, the platforms \( h_{a_i} \) and \( h_{b_{i+1}} \) can collude to attack agent \( a \).

The impossible collaboration between platforms in the itinerary of agent \( a \) and platforms in the itinerary of agent \( b \) is simply an assumption put forth by Roth. Our protocol treats the case where this collaboration is not supposed. Consequently, our protocol supposes a reduced number of assumptions compared to Roth’s protocol and treats more realistic cases.

3.2. Itinerary recording protocol

Definition

Let \( h_{a_i} \) be the platform visited by agent \( a \), \( id(h_{a_i}) \) the identity of this platform, \( h_{b_i} \) the platform visited by agent \( b \) and \( id(h_{b_i}) \) the identity of this platform. Let \( prev_{ai} \) be the identity of the last platform
where agent \( a \) was executed. According to Assumption 3, the agent has access to this identity in an authenticated manner. Finally, let \( \text{nextai} \) be the identity of the next platform that agent \( a \) would like to visit after visiting platform \( ha_i \). The agent begins its itinerary on platform \( ha_0 \). Thus, if agent \( a \) moves a total of \( n \) times, we obtain \( ha_0 = ha_n \) as the assumption stating that the agent finally returns to the platform of origin.

**Initialization**

Let \( h_0 \) be the platform of origin of both agents \( a \) and \( b \). \( h_0 \) must be a trusted platform for both agents \( a \) and \( b \). For agent \( a \), \( \text{nexta}_0 \) has the value of the first platform where the agent will be executed. Each agent is sent to its respective destination.

**Step** \( i, i \in \{1, \ldots, n - 1\} \)

Agent \( b \) waits for communication from agent \( a \). If agent \( a \) does not communicate after a certain period of time, agent \( b \) would consider that agent \( a \) was kidnapped, and it would return to its owner’s platform. The offender cannot be identified with certainty: it could be either that platform \( ha_{i-1} \) did not actually send the agent, or that platform \( ha_i \) hindered communication.

Agent \( a \) attempts to communicate with agent \( b \) on platform \( hb_i \).

If agent \( b \) is not on this platform, it is due to the fact that the previous platform \( hb_{i-1} \) did not send it and that platform \( hb_i \) did not execute it, or it did not grant network access to agent \( b \) in order to receive communication from agent \( a \). Therefore, we cannot record or verify agent \( a \)’s itinerary. In this case, agent \( a \) returns to its platform of origin.
\begin{verbatim}
b is on \( h_0 \)
i := 1
continue := true
\textbf{while} continue \textbf{do}
\begin{verbatim}
wait_communication (
if (verify_signature ( prevai || nextai , signhai) = false 
or id(hai) \neq nextai_{-1} 
or prevai \neq id (hai_{-1})) then
migrate ( h_0 )
find_offender()
continue := false
else
record_itinerary ( prevai, nextai, signhai (prevai || nextai),
signhbi (prevai || nextai||signhai))
if nextai = h_0 then
migrate ( h_0 )
continue := false
else
i := i + 1
migrate ( hb_i )
end if
end if
\end{verbatim}
\end{verbatim}
end while

Fig. 3. Algorithm for agent \( b \).

Otherwise, agent \( a \) sends agent \( b \) the identities of \( nextai_i \) and \( prevai_i \), signed by platform \( ha_i \) and by the authenticated channel, \( b \) learns \( id (ha_i) \). It verifies the signature of \( ha_i \) and the following equalities:

\[ id(ha_i) = nextai_{i-1} \text{ and } prevai = id(ha_{i-1}) \]

If these equalities are confirmed, agent \( b \) records \( nextai_i, prevai_i, ha_i \)’s signature, then requires a signature from \( hb_i \). We use \( \text{sign}_Y(X) \) to designate the result of the cryptographic signature of platform \( Y \) on object \( X \) and we use \( || \) to designate concatenation. Here is a typical entry from the itinerary record.

\[
\begin{array}{cccc}
prevai_i & nextai_i & \text{sign}_{hai} (nextai_i || prevai_i) & \text{sign}_{hbi} (prevai_i || nextai_i||\text{sign}_{hbi}) \\
\end{array}
\]

Non-confirmed identities indicate that an attack has occurred. In this case, agent \( b \) returns to its owner’s platform.

Agent \( a \) performs its task on the platform before migrating to its subsequent destination. Figures 3 and 4 describe the algorithms for agents \( a \) and \( b \) respectively.

\textbf{Step} \( n \)

When the agents return to their owner’s platform without detecting an attack, the owner verifies whether the itinerary has been modified. For each entry \( i \in \{1, \ldots, n\} \), it verifies whether the signature \( \text{sign}_{hai}(prevai_i || nextai_i) \) is valid and that \( ha_i = nextai_{i-1} = prevai_{i+1} \). It repeats the operation for \( \text{sign}_{hbi} (prevai_i || nextai_i||\text{sign}_{hai}) \). An invalid entry indicates the occurrence of an attack on the agent’s itinerary and the results are considered invalid. The offender was not identified.

Once the integrity of the itinerary record has been confirmed, the agent’s owner verifies whether agent \( a \)’s itinerary was executed on the designated platforms. A negative outcome indicates an attack has occurred and messages exchanged between \( a \) and \( b \) have been modified.

\end{verbatim}
\( a \) is on \( h_0 \)
\[ i := 1 \]
while \( h_a \neq h_0 \) do
\[ \text{communicate to } h_b(\text{preva}, \text{nexta}, \text{sign}_a(\text{preva} \| \text{nexta})) \]
if \( \text{communication\_succeed()} = \text{false} \) then
\[ \text{migrate (} h_b) \]
else
\[ \text{task ()} \]
\[ i := i + 1 \]
\[ \text{migrate (} h_a) \]
end if
end while

Fig. 4. Algorithm for agent \( a \).

offender1: = person
offender2: = person
offender3: = person
validity: = true
\[ i := 1 \]
while (offender1 = person and \( i < n + 1 \) and validity = true) do
\[ \text{if (verify\_signature (} \text{preva} \| \text{nexta} \| \text{sign}_a) = \text{true} \]
\[ \text{and verify\_signature (} \text{preva} \| \text{nexta} \| \text{sign}_a \| \text{sign}_b) = \text{true} \) then
\[ \text{If id(sign}_a) = \text{preva} + 1 \text{ then}
\[ \text{offender1: = } h_a \]
\[ \text{offender2: = } h_b_{i+1} \]
end if
\[ \text{if id(sign}_a) \neq \text{nexta} \text{ then}
\[ \text{offender1: = } h_{a_{i-1}} \]
\[ \text{offender2: = } h_b \]
\[ \text{offender3: = } h_{b_{i+1}} \]
end if
\[ \text{if (id(sign}_a) = \text{preva} \text{ and id(sign}_a) = \text{nexta} \text{ and platform\_conform(id(sign}_a)) = \text{false} \) then
\[ \text{offender1: = } h_{a_{i-1}} \]
else
\[ \text{validity: = false} \]
end if
\[ i := i + 1 \]
end if
end while

Fig. 5. Algorithm Step \( n \).

Figure 6 describes the function \textit{find\_offender()} executed by agent \( b \) when it detects an attack before step \( n \), while Fig. 5 presents the algorithm executed on step \( n \) in the absence of attack detection in the previous steps. When identifying the offender, the algorithm does not consider the man-in-the-middle attack as the protocol supposes that the communication channels between agents \( a \) and \( b \) are authenticated and secured (e.g., using SSL [3]).
offender1: = person
offender2: = person
if verify_signature (prevai \parallel nextai , signhai) = false then
   offender1: = hai
else
   if id(hai) \neq nextai then
      offender1: = hai–1
   else
      if prevai = id(hai –1) then
         offender1: = prevai
         offender2: = hai–1
      else
         offender1: = hai–2
         offender2: = hb i–1
      end if
   end if
end if

Fig. 6. Function find_offender().

3.3. Protocol security

The protocol we propose functions similarly to Roth’s protocol [15], in that it detects the same attacks when we uphold Roth’s assumptions, i.e., if we suppose there is no collaboration. However, with the weaker assumptions that we stated, we obtain identical results at the level of detected attacks. The possibilities for collaboration between two hosts hai and hbj fall into one of the four subsets: i < j – 1, i = j – 1, i = j, and i > j. We will also consider the case of collaboration among the three hosts hai, hbi+1 and hbi+2. The other attacks with this type of collaboration can be modeled as two different attacks with collaboration between a’s itinerary platform and agent b’s itinerary platform.

3.3.1. Attack with Collaboration between hai and hbj, with i < j – 1

In this scenario, the platforms hai and hbj (i < j – 1) collude in order to attack the mobile agent by modifying its itinerary. Let’s suppose that the mobile agent decides to migrate to platform h’ (refer to Fig. 7) while running on platform hai. Platform hai attacks the mobile agent a by sending it to platform hai+1 and modifies the message sent by mobile agent a to the cooperating agent b (i.e. replaces the identity of platform h’ by the identity of platform hai+1 in the variable nexta_i). With this modification, cooperating agent b cannot detect the attack. However, it will be detected by the platform of origin upon the agents’ return to it because the platform of origin will detect that agent a is executed on platform hai+1 instead of h’. In order to prevent the platform of origin from detecting this attack, platform hai will ask platform hbj to modify the entry corresponding to platform hai+1 in agent a’s itinerary.

Replacing an entry in the itinerary record is impossible, since this latter resists to the modifications. An entry is simultaneously signed by platform hai and hbj. If platform hbj attempts to modify an entry with j > i, it could not create a valid entry at the signature level as it cannot imitate the signature of platform hbj. Therefore, if it modifies nextai; the signature will be invalid and the attack will be detected when signatures are checked. However, the platform of origin cannot find the offender. If hbj collaborates with


Fig. 7. Attack with collaboration between $ha_i$ and $hb_j$ ($i < j - 1$).

If $hb_j$ to counterfeit its signature, the attack will, nevertheless, be detected upon the agent’s return, since the correspondence between $nexta_i$ and the identity of the platform that signed entry $i + 1$ is invalid.

If platform $hb_j$ ($j > i$) could create its own agents and collect suitable records from the correct itinerary of agent $a$, it could replace the entry $i$. Consequently, the protocol cannot detect this scenario of attack due to the fact that the information route recorded by agent $b$ are not linked to a particular agent instance.

3.3.2. Attack with collaboration between $ha_i$ and $hb_{i+1}$

If $ha_i$ sends agent $a$ to platform $ha_{i+1}$ with $id(ha_{i+1}) \neq nexta_i$, platform $hb_{i+1}$ will receive $id(ha_{i+1})$, $prev_{a_{i+1}}$ and $nexta_{i+1}$ information, then verify the equality of $id(ha_{i+1}) = nexta_i$. However, unless the authentication key was stolen, this equality cannot be confirmed. Therefore, agent $b$ (on platform $hb_{i+1}$) detects a problem on the itinerary. However, it is possible that this platform collaborates with $ha_i$ to have agent $b$ believe that this equality is true. Therefore, $ha_i$ collaborates with $hb_{i+1}$. Agent $b$ carries out equality verifications at step $i + 1$, although the error remains undetected due to manipulation from the platform on which it is executed, $hb_{i+1}$. Agent $b$ finds that the two equalities are verified. Once on platform $hb_{i+2}$, the verification of the equality $prev_{a_{i+2}} = id(ha_{i+1})$ indicates an error since platform $ha_i$ did not send the agent to $nexta_{i+1}$. Unmasking the offender would be problematic since this equality is not verified. Moreover, this corresponds to the case where host $ha_{i+1}$ sends the agent to a wrong platform, which would attempt to put it on its track. Figure 8 illustrates this attack scenario. Indeed, if $ha_{i+1}$ sends agent $a$ to a different platform than $nexta_{i+1}$, and that this platform sends agent $a$ towards identity platform $nexta_{i+1}$ to put the agent on its track, once on platform $ha_{i+2}$, the equality $prev_{a_{i+2}} = id(ha_{i+1})$ will not be verified, because the latter authentically knows the identity of the platform that sent it to agent $a$. 


Thus, finding a culprit is problematic: we know it is possible that the offender would be $h_{a_i}$, this is the case we studied with $h_{b_{i+1}}$ collaboration, or it could be that $h_{a_{i+1}}$ (with an accomplice) tried to put the agent on its track. Agent $b$ verifies $next_{a_{i+1}}$ signature by $h_{a_{i+1}}$ and compares this identity with $next_{a_i}$. This is carried out in the function $find_{offender}()$, which is executed when the agent detects the attack. If this is a first attack, the identity of the platform that signed $next_{a_{i+1}}$ is different from both $next_{a_i}$ and offender $h_{a_i}$, with the collaboration of $h_{b_{i+1}}$. In the case of a second attack, the result is an equality between these two entities and the offender is $h_{a_{i+1}}$.

Another attack with collaboration between $h_{a_i}$ and $h_{b_{i+1}}$ can occur. Platform $h_{a_i}$ sends agent $a$ to $h'$ whose identity is different from the host where it would really like to send agent $a$. Then, host $h'$ would try to put the agent on its track by sending it to $h_{a_{i+1}}$. Once on $h_{a_{i+1}}$, agent $a$ communicates with agent $b$ to give it $prev_{a_{i+1}}$ and $next_{a_{i+1}}$. Agent $b$ carries out its verifications and would normally deduce that the agent deviated from its trajectory since $prev_{a_{i+1}} \neq id(h_{a_i})$. However, $h_{b_{i+1}}$ manipulates it in order to confirm this equality.

Although this attack is not immediately detected, it will be noticed when the agent returns to its platform of origin. On step $n$, the verification of the itinerary record will establish that $id(sign_{h_{a_i}}) \neq prev_{a_{i+1}}$ and offenders $h_{a_i}$ and $h_{b_{i+1}}$ will be unmasked. If $h_{b_{i+1}}$ modifies the record of entry $i+1$, it will be incapable of reproducing $h_{a_{i+1}}$ signature. Thus, $h_{b_{i+1}}$ cannot do so without detection upon the agent’s return as the validity of the signatures is verified on step $n$. However, in this case, the identity of the offender is revealed.
3.3.3. Attack with collaboration between $ha_i$ and $hb_j$, for $j = i$ and for $i > j$

Suppose that $ha_i$ collaborates with $hb_i$ so that agent $b$ records $h_f$ (where the attacking platform $ha_i$ would like to send agent $a$) as the identity of platform $i + 1$ visited by agent $a$ instead of $h'$ (where $a$ would like to go). Then, agent $b$ records a $nexta_i$ which is false, and platform $ha_i$ sends the agent to the platform that was recorded. In this case, when $b$ will be on platform $hb_{i+1}$, (that we suppose to be honest), it will receive the following information from $a$: $nexta_{i+1}, preva_{i+1}, id(ha_{i+1})$. The verification of equality $nexta_i = id(ha_{i+1})$ is carried out and does not allow immediate detection of the attack since platform $hb_i$ recorded where platform $ha_i$ sent it ($ha_{i+1}$) rather than agent $a$’s original destination ($h'$). However, this type of fraud will be detected upon the agent’s return to its platform of origin, since agent $b$ kept the complete itinerary of $a$ and the agent’s owner verifies this itinerary. Figure 9 shows this scenario of attack. This is not a drawback of our protocol: this type of fraud is similar to the case where a malicious platform counterfeits $nexta_i$, in the communication between $a$ and $b$ to send agent $a$ toward another platform than its destination, without $b$ being able to detect the attack. This attack was called “attack with modification of communication between $a$ and $b$” in Roth’s protocol. The latter also does not allow for an immediate detection of this attack, although it detects the attack on step $n$, by verifying agent $a$’s itinerary upon agent $b$’s return. If $ha_{i+1}$ is dishonest, the platform must attempt to replace the agent on its track. This is detected with $hb_{i+1}$ since the equality $preva_i + i + 2 = id(ha_{i+1})$ will not be verified. A collaboration between $ha_i$ and $hb_j$ for $i > j$ prevents an attack. Indeed, agent $b$ is executed on host $hb_j$ before agent $a$ migrated to host $ha_i$. 

![Fig. 9. Attack with collaboration between $ha_i$ and $hb_i$.](image)
3.3.4. Attack with collaboration between $ha_i$, $hb_{i+1}$ and $hb_{i+2}$

The principle of this attack is identical to the attack with collaboration between $ha_i$ and $hb_{i+1}$: $ha_i$ sends agent $a$ towards a different platform than its original destination; once on the platform, agent $a$ communicates with agent $b$ to access the identities of the platform, as specified in the protocol. Figure 10 illustrates this scenario of attack. Usually, $b$ would detect the attack, but in this instance, it has been manipulated by platform $hb_{i+1}$ that collaborates with $ha_i$. Therefore, it does not detect the attack and migrates towards host $hb_{i+2}$. In the case where $hb_{i+2}$ was honest, we would have immediately detected the attack and found the offender. Here, $hb_{i+2}$ collaborates and manipulates agent $b$ to have it believe that equality $prev_{a_i+2} = id(ha_{i+1})$ is verified. In this case, the agents’ return to their owner’s platform should be expected. The algorithm executed upon the return of the agent to their owner’s platform (step $n$) compares the identity of the platform on agent $a$’s itinerary, who signed the entry $i+1$ with next$ai$. next$ai$ is different from the identity of the platform that signed entry $i+1$ of itinerary ($ha_{i+1}$). If this attack was previously undetected, it is due to the collaboration between $ha_i$, $hb_{i+1}$ and $hb_{i+2}$. As indicated earlier, if $hb_{i+2}$ is honest, the platform will detect the attack and unmask the offenders.

3.3.5. Other attacks

Some other attack scenarios could be imagined. However, they will involve the collaboration of several platforms, and are consequently not addressed in this paper. For example, spoofing agents $a'$ and $b'$ could be created by $ha_{n-1}$ and $hb_{n-1}$. The aim of these agents would be to revisit the itinerary and correct it. This scenario involves the collaboration of $ha_{n-1}$, $hb_{n-1}$, $ha_1$, $hb_1$ and $hb_i$ (the platform that modifies the agent’s itinerary). The protocol cannot detect the modification of agent $a$’s itinerary. Nevertheless, even if this latter could be corrected, the results obtained inside the visited platform remain invalid in the case where they are signed by the platform.

4. Implementation and Results

The platform used to implement our protocol is Grasshopper2.2.3, which uses Java as the agent programming language. To carry out the tests, we used 5 machines running Windows 2000. The machines are identified with the name used for the shopping application. The network used is a local network, Ethernet 100 Mbps. The mobile agents are implemented using JDK 1.3.2 The security package IAIK-JCE 3.01 [10] was used; it offers a set of different cryptographic algorithms implemented in pure Java.

It was deemed important to test our protocol in a relatively realistic environment and we decided to implement an application where the recording and itinerary verification could help secure the application. Our agent shops for a certain number of products on the user’s behalf. Provided with a description of the products required by the user, it moves to the platform of many vendors to inquire about their prices. In order to simplify the presentation, the list of products is supposed to be known by the vendors. The itinerary is dynamic: initially, the user starts the search by entering a certain number of vendor addresses. However, while shopping, if a vendor does not have the listed products, the agent is referred to another vendor with whom trade agreements have been established. These agreements contain financial compensation clauses for the benefit of the vendor who routed the agent to another address.

1 http://www.grasshopper.de/ (July 2003).
2 http://java.sun.com/j2se/1.3/ (July 2003).
The mobile agent records the three best offers for each product sought. It could save the entire set of offers, but if they are too numerous, the result size would become too large. In this case, the agent could lose some of its advantages over the client-server approach. We chose to save the best three results instead of a single one, since factors other than prices may influence the user's choice (the user could choose a slightly more expensive product if it came from a more reputed company for instance). The Grasshopper’s transport service allows for the agents’ transport via SSL (Secure Sockets Layer).

4.1. Implementation tests

First, we tested the agent’s authenticated transport communication. For the client, as well as for the server, we tested three different cases:

- the client or the server has a signed certificate issued by the certification authority, which is considered trustworthy by the other entity;
- the client or the server has a signed certificate issued by the certification authority that is not considered trustworthy by the other entity;
- the client or the server does not have a certificate.

The SSL connection is refused when the client’s or the server’s certificate is not trustworthy. Since the mutual authentication was forced through our SSL connection, this confirms our assumptions.

Then, we tested the authenticated communication between the two agents. The agent transport is carried out only if the agent and server have a certificate signed by a trustworthy certification authority. In the other cases, the SSL connection is refused and the message is not sent from agent $a$ to agent $b$. 
Finally, we tested the detection of the attacks on the itinerary. We carried out the attack where platform $ha_i$ sends agent $a$ to a different platform than agent $a$’s original destination. In the same way, we tested the same attack with an attempt to put the agent on its track. In both cases, the attack was detected during the same step $i$. However, the tests carried out on the other attacks were limited in terms of security features to be verified. Indeed, attacks where there is collaboration between a host on agent $a$’s itinerary and a host on agent $b$’s itinerary consist of manipulating the agents. The purpose of this manipulation is to modify the equality result or to alter communication between the agents.

We chose to measure the cost of our protocol on a prototypical mobile agent application – the shopping application described earlier. As seen earlier, the agent departs with a shopping list of three products. It only knows the address of one vendor, called supplier1. The first vendor sends the agent to another vendor since he does not carry one of the products and has trade agreements that allow forwarding the agent to another vendor, called supplier2. Agent $a$ has a list of hosts that agent $b$ will visit.

Five machines are used to conduct the experimental study. A Grasshopper platform is installed on each machine. Each platform represents a different site. Two platforms, called Supplier1 and Supplier2 constitute agent $a$’s itinerary. Two other platforms, called $B1$ and $B2$ constitute agent $b$’s itinerary. A fifth platform, called Home, represents the platform of origin. Consequently, agents $a$ and $b$ are created within the Home platform which sends agent $a$ to Supplier1’s platform and agent $b$ to $B1$’s platform. Agent $a$ moves from Supplier1 to Supplier2 before returning to Home platform. Agent $b$ moves from $B1$ to $B2$ before returning to Home platform. Figure 11 illustrates this experimental environment.

Our tests measured both:

- the execution time for each vendor machine visited by the shopping agent;
- the total execution time for the shopping agent and its cooperating agent.

We chose to measure these variables as they reflect some of the advantages of the mobile agent paradigm over the client-server approach. We measured the execution time with the Java method $System.currentTimeMillis()$. For the first measurement, we call this method upon the arrival of the agent.
on the platform and before it migrates to the next platform. The second measurement corresponds to
the difference between the time the agents were created and and the moment where agent b finished the
verifications after having returned to its owner’s platform. Hence, this measurement corresponds to the
total time the task required. We did measurements with three types of agents:

– the shopping agent and its cooperating agent, using our itinerary recording protocol;
– the shopping agent and its cooperating agent, using Roth’s protocol;
– only the shopping agent.

Figure 12 shows the execution time for these three situations. It should be noted that there are some
important differences in a given category. For example, measurement 3 of our protocol indicates the
highest results. We noticed (via Grasshopper) that the thread executing the agent used a minimal priority.
The fact that the other threads were executed on the process platform explains the variations in execution
time. Moreover, while the execution time differs little between our protocol and Roth’s (around 7 s in
both cases), it is much lower for the shopping agent. This is explained by the fact that, in both our
protocol and Roth’s, the shopping agent establishes an SSL connection with its cooperating agent, while
this is not required by the shopping agent alone. However, establishing this connection requires time,
since the virtual machine must load many classes in memory and initialize the SSL session.
Figure 13 shows the total execution time for the three cases. Just as with the execution time on a vendor platform, although our protocol and Roth’s have similar values, the shopping agent alone requires less execution time. These results are related to the platform execution time. Note that the total execution time reflects both the execution time on each platform and the total migration time. Since the network’s features do not change, the migration time would vary little from one measurement to another. Similarly, the difference among the migration times of three agents is small, since the agents are of similar sizes in relation to the network flow; moreover, in all three cases, we use an SSL connection to transport the agents.

We also find a large difference between the relative gap of our protocol and the shopping agent, for the execution on the vendor’s platform (70%) and the total execution time (70%). As mentioned above, this is due to the SSL connection which requires much more time.

5. Conclusion

This paper presented a mobile agent itinerary recording protocol. This protocol gives a secure method of recording the identities of the platforms on which a mobile agent is executed. This problem is difficult to solve with a single agent since the data transmitted are prone to manipulation by malicious platforms, which is the reason why Roth [15] introduced the concept of cooperating agents.

We have designed an itinerary recording protocol using the concept of cooperating agents and described in details the behavior of each of the two cooperating agents, as well as the procedures selected to establish the identities of the offenders when an attack is detected, as already done by Roth. To show the security properties of our protocol, we reviewed a set of potential attacks. For each attack, we described the moment of detection and, when possible, we designated the offenders. Thus, we have been able to verify that our protocol detects the attack in cases where there is collaboration between a platform on the cooperating agents’ itinerary and another on the agent’s itinerary. This type of attack is outside the scope of those addressed by Roth.

Finally, we implemented our protocol and Roth’s on a mobile agent platform. To test these implementations, we chose a mobile agent shopping application with a random itinerary. This is a classical application using the mobile agent paradigm where itinerary recording is critical. We tested our protocol on a set of possible attacks and the results confirm our assumption in terms of offender designation and moment of detection. We compared three implementations (the shopping agent with our protocol, with Roth’s protocol, and the shopping agent alone) and measured the execution time of the shopping agent on each platform and the total execution time. Results show that our protocol costs slightly more than Roth’s in terms of execution time; however, our protocol detects more attacks than Roth’s. The use of an itinerary recording protocol (either ours or Roth’s) adds a significant cost in terms of execution time as well as in terms of network load. Thus, our protocol is not suitable in cases where cost is critical.

Future research could strive to create an algorithm that would allow unmasking the offender of a kidnapped mobile agent: the case in which the cooperating agent’s timer expires. With our protocol, we know that the kidnapping offender is either the platform from which the agent communicates before the latter could migrate, or the platform where the agent was meant to be sent the last time it communicated with the cooperating agent.

References


