Secure Migration of Mobile Agents based on Distributed Trust

Martijn Warnier Michel Oey Reinier Timmer Frances Brazier
Intelligent Interactive Distributed Systems
VU University, Amsterdam
The Netherlands
{warnier,michel,rjtimmer,frances}@cs.vu.nl

Abstract

Integrity of mobile agents in open environments in which no guarantees can be provided on the integrity of the hosts on which they run, is an open challenge. This paper presents a method with which tampering can be detected. This method is based on the notion of distributed trust; trust distributed over the hosts involved.

Keywords: *Agents Systems, Mobile Agents, Agent Security, Migration Path Integrity*

1. Introduction

Distributed agent systems provide a powerful paradigm for building large scale distributed systems [5, 6]. Mobility and autonomy are attractive features of such systems. In dynamic environments such as e-government, e-health and e-commerce applications mobile agents provide a means to process data locally, respecting intellectual property rights [2].

Mobility and autonomy, however, also provide new challenges, especially if security is of key concern. The challenge is to preserve integrity of an agent and the data it collects in open environments: environments that are not under the control of an agent's owner.

In closed environments in which all hosts are trusted this is not a problem: all hosts are known to behave correctly. Similarly, in more open systems, where agents only migrate to hosts they trust, the integrity of the agent and its data are not at significant risk. The use of a separate trust infrastructure, e.g., a X509-based PKI [4], can, for example, be used to establish which hosts can be trusted. However, mobile agents can also be deployed in open systems. In such dynamic circumstances it is not always feasible to determine the trustworthiness of hosts in advance.

In such open environments the *malicious host problem* forms a serious threat to the integrity of an agent. It is closely related to the intrinsic features of mobile agent systems: agents are executed on hosts that can view, alter their state, or even delete the agent altogether. This leads to a number of security problems:

1. Protecting the integrity of the migration path
2. Protecting the integrity of the agent itself
3. Ensuring confidentiality of the agent's data and (binary) code
4. Ensuring integrity of the agent's control flow

The migration of an agent from one host to another is called a *migration step*. A *migration path* is a sequence of multiple migration steps that identifies all the hosts, in order, an agent has visited.

The integrity of the migration path (item 1, above) is the basis for detecting malicious hosts and/or preventing them from doing any harm. For example, a number of techniques [1, 3, 10, 9] have integrity of agents migration paths as a premise, and can be used to detect tampering with the agent (items 2 & 4). Confidentiality (item 3) can be ensured by using encryption of sensitive data.

The main focus of this paper is the detection of breaches of integrity in migration paths of mobile agents. This paper does not directly address the general problem of protecting the agent from malicious hosts that alter the agent's code or data. However, as stated before, techniques to detect whether an agent has been altered on the basis of its migration path.

How migration works is the topic of the first section. Some known solutions for integrity protection are discussed in the context of their strengths and weaknesses in this context. The next section introduces a new technique with which the integrity of an agent's code and data can be derived on the basis of its migration path. This method is based on distributing trust among the individual hosts of an agent platform. The paper ends with a discussion and conclusions.

2. Agent Migration

This paper assumes that a secure distributed mobile agent system provides the following basic properties: an agent runs on one single host at a time, is aware of its current host, has the ability to *migrate* to other hosts in the system. In addition this paper assumes that the environment provides a public-key infrastructure with which agents and hosts can be authenticated. The host on which an agent is initialized, is assumed to be trusted by the agent's owner. This host can be traced by all other hosts at any arbitrary moment in time. Hosts are assumed to have full control over the agents they run. The consequence of this assumption is that hosts are assumed to be able to read and alter information stored inside agents.

Agents preferably only migrate to trusted hosts. However, a trust relationship does not always give full guarantees on the correct behavior and intentions of hosts. An agent's migration path provides a means to detect breaches of integrity.

The simplest form of migration in a secure agent system requires sending and receiving hosts to mutually authenticate themselves using a PKI. The integrity of a migrating agent is ensured by having the sending host create a (digital) signature of an (hash of the) agent's code. This signature is transmitted together with an agent's code, and data (including state). The receiving host can then verify the integrity of an agent's code before re-initializing the agent process.

3. Migration Paths

There are roughly two different approaches for recording agent migration paths: (i) use a centralized trusted third party (TTP) to authorize and keep track of migration paths or (ii) store (a signature of) migration paths *inside* the agents themselves [11]. Both approaches are discussed. The following notation is used in this paper: capital letters A,B,C,... denote hosts, small letters x,y,z,... denote agents, arrows ($\rightarrow$) represent migration steps between hosts and $[x]_A$ denotes the signature of agent x by host A.

3.1. Centralized Trusted Third Party

A centralized approach requires all migration paths to be registered. Both sending and receiving hosts register each and every migration step before and after migration. The trusted third party authorizes the migration and stores the migration step in its database together with the commitments of the hosts. This makes it possible to prove, at a later point in time, that both parties had agreed with the migration step.

This approach has the advantage that it is relatively easy to monitor hosts over a period of time. Thus, for example, if agents tend to disappear on one specific host, this host is known to be unreliable, possibly malicious. A centralized trusted third party may prevent agents from migrating to untrusted and/or unreliable hosts (simply by not authorizing the migration).

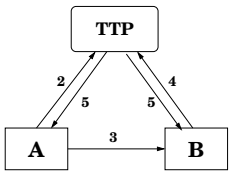

Figure 1. Agent migration using a central authority (trusted third party).

Example 1 below gives a more detailed explanation of a migration step, using a TTP, by an agent from host A to host B.

Example 1
A detailed example of the migration of agent x from $A \rightarrow B$ using a trusted third party. The migration protocol starts when:

1. *host A suspends and signs agent x: $[x]_A$*
2. *host A reports to the trusted third party (TTP) that agent x will migrate from A to B. A sends $[x]_A$ along with the report.*
3. *host A sends agent x to host B*
4. *host B receives x and computes $[x]_B$ which it sends to the TTP.*
5. *the TTP verifies that A and B have both signed the same agent. If the verification passes, the TTP notifies both A and B that the migration has succeeded, and adds this migration step to the migration path it keeps for agent x.*
6. *host B starts the suspended agent*[1].

Figure 1 shows the messages that are sent between the parties. The numbers used in the figure correspond to the numbers above.

Note that this approach does not prevent hosts from conspiring to forge migration paths. For example, malicious hosts can simply decide to migrate an agent between themselves without using the trusted third party. However, after such an illegal migration, the agent cannot migrate to

[1] Note for the sake of clarity, some details are omitted from this and following examples. This protocol has, for instance, not been secured against replay attacks. To solve this problem, other techniques, such as adding freshly generated random numbers to each signature must be applied.

a non-malicious host without being noticed. The trusted third party, having not been informed of the previous migration step, will immediately detect something is wrong and will therefore not authorize the migration. The trusted third party, however, can not detect whether an agent has migrated from a malicious to multiple malicious hosts returning to the first malicious host (a cycle), before moving to a non-malicious host using the trusted third party.

Note also that a centralized approach such as this creates a central point of failure, and a potential performance bottleneck (i.e., a scalability problem). This problem can be partially solved by using an agent's initial host as its trusted third party[2]. This is known as a *home based approach*. The agent platform Mansion uses a dedicated service: the Agent Location Service [12] that implements this use of the initial host as a trusted third party.

3.2. Signature Chain

A method that stores an agent's migration path *together* with an agent's code and data does not suffer the drawbacks of a single centralized server. The stored migration path can be protected against tampering using digital signatures [4]. A chain of such signatures can protect a whole migration path of an agent. In this method, the migration process includes that a host signs a migration step together with all (signed) previous migration steps (the chain, signed by other hosts).

Example 2
A detailed example of the migration step of agent x from $B \rightarrow C$, where B received agent x from A earlier using signature chaining. Agent x's signature chain at host B is $[x, A \rightarrow B]_A$.

1. *host B suspends and signs agent x and signs $B \rightarrow C$ along with the already existing signature chain: $[x, B \rightarrow C, [x, A \rightarrow B]_A]_B$. This signature will become a new link in the signature chain.*
2. *host B sends the agent with the old signature chain, along with the new link of the chain to C.*
3. *host C checks the signature chain stored with agent x and the new link it has received from B.*
4. *if the verification of the signatures was successful, host C adds the new entry to the agent's signature chain and starts the suspended agent x.*

As signatures can only be generated by individual hosts and verified by all other hosts, this method ensures breaches of integrity of the migration path are noticed. The method, however, has some drawbacks: for long chains verifying the complete chain of signatures becomes computationally intensive. A more serious problem is that a malicious host can remove arbitrary cycles from a migration path. If an agent (accidentally) visits the same malicious host for a second time, the malicious host can remove the part of the migration path between the first time it was hosted by the malicious host and the second. The malicious host can then reuse the old signature chain in the agent for other purposes. This cannot be detected by other hosts nor by an agent's owner. Finally, if agents disappear, e.g., by having been killed by a malicious host, outside hosts are not capable of identifying where something went wrong.

Note that with respect to cycles there is a difference between the trusted third party approach and the signature chaining approach. In both approaches a malicious host can remove cycles from the migration path. In the trusted third party approach, malicious hosts could remove the migration path between conspiring malicious hosts ending with the initial malicious host. In signature chaining a malicious host can remove a similar cycle which may also have included non-malicious hosts.

This paper proposes a distributed approach to integrity protection of migration paths that can be used to detect unauthorized modification. The advantages of a distributed approach are that the solution scales better in a distributed system and does not have a single point of failure.

4. Secure Migration using Distributed Trust

In essence, the proposed approach for integrity preservation of migration paths entails the distribution of trust to several hosts on a migration path. This approach assumes that agents are not allowed to migrate to the host on which it currently runs (irreflexivity), and that the migration path can be recorded *together with* an agent's code and data. Tampering of the migration path can be detected by either the agent owner or one ore more receiving hosts.

Briefly, the algorithm works as follows: Suppose agent x migrates along the path $A \rightarrow B \rightarrow C$. Each migration step is recorded with the agent. Each step is signed by the host from which it's originated. When agent x migrates from B to C, host B asks host A to sign the migration step ($B \rightarrow C$). The resulting signature is stored with the agent. When host C receives the agent and the signatures, it confirms receipt to host A. Example 3 below provides a detailed application of the algorithm to a migration step.

Example 3
A detailed example of the migration step $B \rightarrow C$ of agent x. Host B received agent x from host A earlier, using a distributed trust algorithm. The sequence number (hop count) used in the migration step $A \rightarrow B$ is n with $n \in \mathbb{N}$:

[2] If all agents are started from the same location these two approaches are equivalent in terms of scalability.

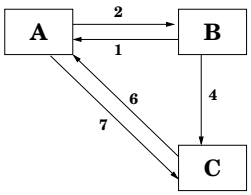

Figure 2. Agent migration using distributed trust.

1. *host B suspends agent x and asks host A to sign $(x, B \rightarrow C, n+1)$ (a* waiver*).*
2. *host A checks whether the seqnr $(n+1)$ corresponds to the seqnr (n) corresponding to the agent being on host A. If all is correct, host A signs the waiver and sends it back to host B. Host A remembers that agent x has migrated to host C with seqnr n+1.*
3. *host B adds the signature $[x, B \rightarrow C, n+1]_A$ to agent x.*
4. *host B sends the agent to host C.*
5. *host B records that agent x has migrated from itself to host C with seqnr $n+1$, and waits for C to request further information. Host C will contact host B whenever host C ships agent x to another host.*
6. *host C receives agent x, verifies the migration chain stored in the agent, and contacts host A to acknowledge the receipt of the agent with seqnr $n+1$.*
7. *host A then checks whether the info from C corresponds to the information it has recorded previously. If there is a mismatch, it notifies host C of this, so C can refuse the agent; otherwise, C can start agent x. After this step, assuming the migration from $B \rightarrow C$ was successful, host A can delete all info concerning agent x.*

Figure 2 shows the messages that are sent between the parties, the numbers used in the figure correspond with the numbers above.

Each *waiver* is issued only once, and can be used only once because the receiving host will 'consume' the waiver by contacting the host that issued the waiver. In Example 3, host C contacts host A to confirm receipt of the agent. Consequently, a malicious host cannot send an agent to different hosts, as it can only acquire one waiver for the agent. Furthermore, a malicious host cannot send the agent twice to the same host (i.e., a replay attack) by reusing a waiver obtained before, because the waiver would have already been used. In other words, host B cannot tamper with the migration path.

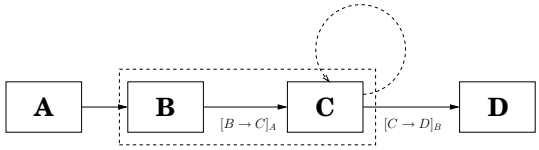

Figure 3. Conspiring hosts altering the migration path.

The migration path cannot be tampered with by a single malicious host, because the entries are linked together via host and sequence number: i) hosts can not cut out a middle piece, because the sequence numbers will not match, ii) hosts can not replace a middle piece, because the signatures in the chain of hosts will not match, iii) hosts can not cut out or replace the tail of a migration chain (including cycles), as the next receiving host will check the migration with the previous host in the migration chain, using both the host and the sequence number. The essence of this approach is that the responsibility of a migration is spread over two hosts: the previous host and the next receiving host, i.e., A and C, in the migration chain $A \rightarrow B \rightarrow C$.

The proposed migration protocol can also withstand two conspiring malicious hosts trying to manipulate the migration path between them. For example, suppose that agent x migrated along the path: $P \rightarrow A \rightarrow Q \rightarrow B$. Now, suppose that B conspires with A to try to remove host Q from the migration path pretending that the agent migrated directly from A to B. However, host P, the predecessor of A, will not issue the waiver $[x, A \rightarrow B]_P$: it has already issued the waiver $[x, A \rightarrow Q]_P$ and will refuse to issue another one.

Unfortunately, it is still possible for two adjacent conspiring malicious hosts to remove a cycle from the migration path. For example, consider an agent that migrates through malicious hosts B and C along the path $A \rightarrow B \rightarrow C \rightarrow \ldots \rightarrow C \rightarrow D$, as depicted in Figure 3 (note, the migration loop at C). In this case, C can remove the cycle from the migration path with the help of B before sending the agent to D. C simply has B issue another waiver for the migration $C \rightarrow D$. Note that C can only remove a cycle if the agent actually returns to C, which the agent typically cannot be forced to do. If cycles are created, then, contrary to signature chaining, two adjacent conspiring malicious hosts are always necessary to remove a cycle.

The main reason two malicious hosts can still alter the migration path is that the receiving hosts assumes that at least one of these is not malicious. To guard against n malicious hosts, this scheme can be extended to incorporate at least the $n+1$ preceding hosts in handing out the waivers.

5. Discussion and Conclusions

This paper introduces a mechanism to ensure that breach of integrity in migration paths of mobile agents in large scale distributed agent systems will be detected. This approach distributes trust over three hosts during each migration step. The combination of sequence numbers with signatures guarantees that one or more hosts can detect if part of the migration path, including cycles, has been removed.

Spreading trust over multiple hosts in an agent system clearly has benefits in terms of scalability and it strengthens the security mechanism, since a 'single point of failure' no longer exists. Orthogonally, a dedicated trust model that can distinguish the –relative– trustworthiness of hosts in multiple agent systems can be of much value as well.

The approach works well in situations with only one (unknown) malicious host in a migration path, or in environments with multiple malicious hosts that do not conspire together. However, if multiple malicious hosts conspire together the situation becomes more complex. A possible solution is to enforce that agents alternate between trusted and untrusted hosts [9] in their migration path. In the extreme situation where only the initial host can be trusted this solution is equivalent to 'two-hop boomerang agents' [7], in which agents always return to their initial host after each migration step to another host. Another possible solution is to disallow cycles in migration paths altogether.

The mechanisms needed to instrument this approach are currently being implemented in in the agent platform AgentScape [8].

Acknowledgments

This research is conducted as part of the ACCESS project[3] funded by the NWO TOKEN program. The authors thank Stichting NLnet for their support and Benno Overeinder for useful comments on earlier drafts of this paper.

References

[1] E. Bierman and E. Cloete. Classification of malicious host threats in mobile agent computing. In *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 141–148. South African Institute for Computer Scientists and Information Technologists Republic of South Africa, 2002.

[2] D. de Groot, M. Boonk, F. Brazier, and A. Oskamp. Issues in a mobile agent-based multimedia retrieval scenario. In *Proceedings of The 4th Workshop on the Law and Electronic Agents (LEA 2005)*, pages 33–43, June 2005.

[3] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. *Personal Technologies*, 2(2):92–99, 1998.

[4] C. Kaufman, R. Perlman, and M. Speciner. *Network Security, PRIVATE Communication in a PUBLIC World*. Prentice Hall, 2nd edition, 2002.

[5] D. Kotz and R. Gray. Mobile Agents and the Future of the Internet. *Operating Systems Review*, 33(3):7–13, 1999.

[6] M. Luck, P. McBurney, and C. Preist. *Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing)*. AgentLink, 2003.

[7] J. Ordille. When agents roam, who can you trust? In *In the Proceedings of the First Annual Conference on Emerging Technologies and Applications in Communications*, pages 188–191, 1996.

[8] B. Overeinder and F. Brazier. Scalable middleware environment for agent-based internet applications. In *Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04)*, volume 3732 of *LNCS*, pages 675–679, Copenhagen, Denmark, 2004. Springer.

[9] V. Roth. Mutual protection of co-operating agents. In J. Vitek and C. Jensen, editors, *Secure Internet programming: security issues for mobile and distributed objects*, volume 1603 of *LNCS*, pages 275–285. Springer-Verlag, 2001.

[10] T. Sander and C. Tschudin. Protecting Mobile Agents Against Malicious Hosts. *Mobile Agents and Security*, 60, 1998.

[11] A. Saxena and B. Soh. Authenticating mobile agent platforms using signature chaining without trusted third parties. In *In the Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, (EEE'05).*, pages 282–285, 2005.

[12] G. van 't Noordende, F. Brazier, and A. Tanenbaum. Security in a mobile agent system. In *Proceedings of the First IEEE Symposium on Multi-Agent Security and Survivability*, Philadelphia, 2004.

[3] http://www.iids.org/access