Enforcing Integrity of Agent Migration Paths by Distribution of Trust

Martijn Warnier*, Michel Oey, Reinier Timmer, and Frances Brazier

Intelligent Interactive Distributed Systems group,
Department of Computer Science,
VU University Amsterdam,
de Boelelaan 1081a, 1081 HV Amsterdam
The Netherlands
E-mail: {warnier,michel,rjtimmer,frances}@cs.vu.nl
*Corresponding author

Benno Overeinder

NLnet Labs,
Kruislaan 419,
1098 VA Amsterdam,
The Netherlands
E-mail: benno@NLnetLabs.nl

Abstract Agent mobility is the ability of an agent to migrate from one location to another across a network. Though conceptually relatively straightforward, in practice security of mobile agents is a challenge: from transport layer security to preservation of integrity in open environments. This paper discusses the security issues involved and proposes protocols for secure agent migration. AgentScape, an agent platform for mobile agents, is used to illustrate the feasibility of the implementation of these protocols.

Keywords: Agent Systems, Mobile Agents, Agent Security

Reference to this paper should be made as follows: M. Warnier, M.A. Oey, R.J. Timmer, B.J. Overeinder and F.M.T. Brazier: 'Enforcing Integrity of Agent Migration Paths by Distribution of Trust', *Int. J. of Intelligent Information and Database Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Martijn Warnier received his PhD in Computer Science at the Radboud University Nijmegen, the Netherlands. He is currently a Postdoctoral Researcher in the IIDS group at the VU University Amsterdam.
Michel Oey received his PhD in Computer Science at the VU University Amsterdam and is currently a scientific programmer in the IIDS group.
Reinier Timmer received his Masters of Science at the VU University Amsterdam and is currently a scientific programmer in the IIDS group.
Frances Brazier is a full professor at the VU University Amsterdam and chairs the IIDS group. IIDS' research focus is on (self-)management of

large scale distributed autonomous systems. AgentScape is the middleware designed to support large scale distributed autonomous systems. Benno Overeinder received his PhD at the University of Amsterdam and is currently a senior researcher at NLnet Labs, a non-profit organization that develops open source Internet software. His former affiliation was assistant professor in the IIDS group.

All authors have, in one form or another, contributed to the design and development of the open source AgentScape platform.

1 Introduction

The agent paradigm supports the design of autonomous distributed networked systems in service-oriented environments. Agents are autonomous, pro-active, communicative, and goal-directed, often capable of learning and sometimes equipped with the ability to migrate from one location to another. Agents equipped with the ability to migrate, so-called mobile agents, traverse the network to access the services and resources they need to achieve the goals they pursue. The potential of mobile agent technology in sectors such as E-Commerce (8; 9), E-Health (18) and E-Governance (4; 33), is well recognized. In these sectors, in particular, privacy and copyright, are of utmost importance. Data access control is mandatory: by moving agents to the location at which data is stored, data access and processing can be done locally and controlled.

Mobility and autonomy, however, also provide new challenges, especially with respect to security issues. Preservation of agents' integrity and their data—data acquired whilst traversing a network—is the challenge addressed in this paper. This challenge is of explicit importance in open environments: environments that are not under the control of agents' owners. The hosts on which an agent resides may be malicious, and temporarily have complete control of the agent's runtime environment.

Complementary is preservation of the integrity of agent platforms, i.e., agent middleware: protecting platforms from malicious agents, i.e., agents that attack the host on which they reside. This is not the focus of this paper, as solutions to this problem exist. Sandboxing (as in Java) or jailing (30), are examples of solutions with which agents are run in contained environments limiting the amount of damage they can cause to the systems on which they run. A more fine-grained, though less practical, approach is the use of proof carrying code (19) (applied to the mobile agent paradigm described in (20)). Agents carry a 'proof' with them that specifies their expected and acceptable behaviour. Each host is equipped with a theorem prover to ensure that an agent's code indeed adheres to its specification. This approach is very labour intensive (12). Protecting agent platforms against malicious hosts is not further discussed in this paper.

The remainder of this paper is organized as follows: the next section introduces the AgentScape platform, the platform used throughout the paper to illustrate the technological issues concerning secure agent migration.[a] Section 3 explains

[a]Note that a new migration protocol recently has been introduced (5) that aims to standardize the migration protocol further. See (6) for an implementation of this protocol in AgentScape. In

the basics of agent migration. It also discusses the simplest security protection mechanism: transport layer security. An overview of more extended mobile agent security approaches is given in Section 4, with the main focus on the protection of the integrity of an agents' migration path. Section 5 introduces a new approach to this problem that is both more scalable and more secure than existing approaches. The paper ends with a discussion and conclusions.

2 AgentScape

AgentScape is an agent platform that provides the middleware infrastructure needed to support mobility, security, fault tolerance, distributed resource and service management, and services access, to agent applications.

Within AgentScape, *agents* are active entities that reside within *locations*, and *services* are external software systems accessed by agents hosted by the AgentScape middleware (see Figure 1). Agents in AgentScape can communicate with other agents and can access services. Agents may migrate from one location to another.

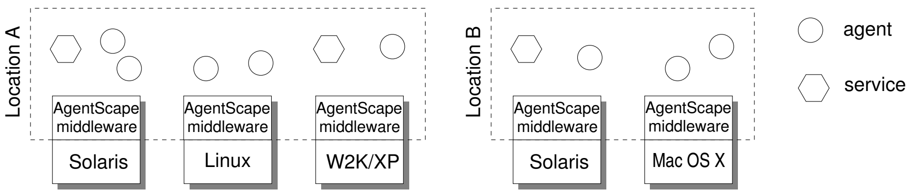

Figure 1 Conceptual model of the AgentScape middleware environment.

All operations of agents are modulo authorization and security precautions. For example, an agent may be required to have the appropriate credentials (ownership, authorization, access to resources, and so on) to access a specific service.

The leading principle in the design of the AgentScape middleware (22) has been to develop a minimal but sufficient open agent platform that can be extended to incorporate new functionality or adopt (new) standards into the platform. This design principle has resulted in a multi-layered architecture with (1) a small *middleware kernel*, called the AgentScape Operating System (AOS) kernel (32, in this issue), that implements basic mechanisms and (2) high-level *middleware services* that implement agent platform specific functionality and policies (see Figure 2).

AgentScape's middleware services implement agent specific functionality. The current set of middleware services include:

- **Location Manager** - Every *location* has a Location Manager, which runs on one of the hosts within that location and manages the location's hosts. Locations typically represent a same single administrative domain.
- **Host Manager** - Each host (typically, one physical machine) runs its own Host Manager to manage the middleware processes for which it is responsible, regulating and guarding access to its resources.

the remainder of this paper the original, more mature, AgentScape agent migration protocol is used.

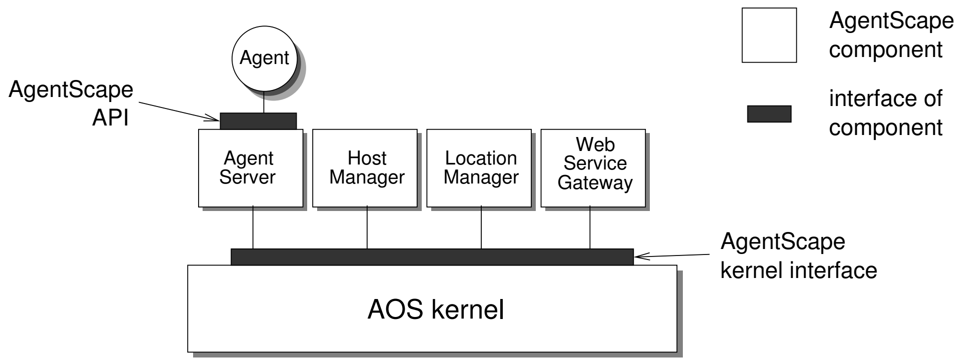

Figure 2 The AgentScape software architecture.

- **Agent Server** - An AgentServer provides a programming language specific runtime environment for agents. Each host can run one or more AgentServers to host agents supporting e.g., different programming languages.
- **Web Service Gateway** - The Web Service Gateway enables agents to communicate with web services using the SOAP/XML protocol (24).
- **Lookup Server** - The Lookup Server keeps track of the current location of agents. Strictly speaking, this service is not part of the AgentScape middleware as it can be run as a stand-alone application. Two versions exist, a centralized, unsecured version and a decentralized secured version (23).

Agent servers provide agents access to the AgentScape middleware (see Figure 2). Agent servers are programming language, i.e., code base, specific. A host can support multiple code bases, by providing multiple code base specific agent servers, at least one per code base. Agent servers 'sandbox' or jail agents, to limit access to the underlying computer system.

The location manager is the coordinating entity in an AgentScape location (managing one or more hosts in its location). Note that for fault tolerance a location manager may be replicated, but that is beyond the scope of the paper, see (16). Agent creation, migration, and all policy related issues relevant in the context of a location, are managed and coordinated by the location manager. The host manager manages and coordinates activities on a host, in interaction with the location manager. and is responsible for local (at the host) resource access and management. The policies and mechanisms of the location and host manager infrastructure are based on negotiation and service level agreements (17).

3 Agent Migration

This section illustrates the basic implementation of agent migration in AgentScape. Other agent platforms that support migration implement a similar form of agent migration, see for example (10; 27). Each host in an AgentScape location runs an instance of the AOS kernel, and manages the data and code of the agents it hosts. *Agent containers* (14), store all data and code associated with an agent: the agent's (executable) code, meta-data (agent id, owner information, authorization data, etc.), and (optionally) data the agent may want to store (obtained at some remote location, etc.).

The following protocol is used in AgentScape for agent migration:

Protocol 1
The protocol for standard agent migration as implemented in AgentScape.

1. *An agent sends a 'migration request' to its local host manager.*

2. *This host manager contacts its local location manager, specifying the identifier of the agent and the destination to which it wishes to migrate, i.e., the destination location.*

3. *The local location manager contacts the (remote) location manager of the destination location with a request to accept the migrating agent.*

4. *If the location manager of the destination location is willing to accept the agent the remote location manager selects a host within its location and instructs the corresponding host manager to prepare to receive a migrating agent. Information on how to contact this chosen host manager is sent back via the location managers to the host manager of the migrating agent.*

5. *The local host manager of the migrating agent suspends the agent and finalizes the state of the agent in an agent container. Next, the local host manager contacts the remote host manager directly and sends the agent container.*

6. *Once the transfer is completed, the receiving host manager verifies the integrity of the agent container. It then sends a message to the originating host, containing the migration status of the agent. The originating host manager then deletes the suspended agent from its host, and the receiving host starts the migrated agent.*

If any step fails in Protocol 1 above, the migration process as a whole halts and the suspended agent is restarted at the originating host. The same applies to the secure migration protocols described in the remainder of the paper. Also, note that, as AgentScape only provides weak migration, an agent is responsible for storing useful information in data entries in the agent container. The AOS kernel guarantees the integrity of the data in agent containers by computing checksums of each entry in the agent container.

This form of agent migration does not automatically provide security against outside attackers. An attacker can alter the data stream between two AOS kernels, if no transport layer security is present. Possible risks include altered agents, where the attackers performs a man-in-the-middle attack, alter the agent's code and recompute the checksums. It is even possible to insert a completely new agent or to simply destroy the agent during migration.

A first step towards a secure migration process is achieved by adding transport layer security.

3.1 Transport Layer Security

A normal connection does not provide any security features and is vulnerable to both active (man-in-the-middle) and passive (eavesdropping) attacks. Transport layer security can be obtained by using a secure connection (SSL (7)). The complete data stream that is transferred over an SSL connection is encrypted, thereby eliminating all passive attacks.

For authentication the SSL protocol uses certificates (15). It is possible to use *self-signed* certificates to set up an SSL connection, but in this case proper authentication between hosts is not guaranteed. Active attacks are still possible as an attacker can set up two SSL connections: one between the originating host and the attacker and another between the attacker and the receiving host. This is another form (2) of the well-known man-in-the-middle attack.

Authentication between hosts can only be ensured if *signed certificates* are used. The identity of the host, for example a URL or IP-address, together with a public key are signed by a root certificate, together forming a signed certificate. The public key of the root certificate is stored in a PKI server. When two hosts set up an SSL connection the signatures of their certificates are checked, using the public key stored in the root certificate, and only if the certificates match with their identity is the SSL connection finalized. This prevents all active attacks from outsiders.

Most agent platforms, including AgentScape, use SSL in their implementation of agent migration. However, mobile agent security should also deal with the far trickier issue of *insider attacks*. Especially when an agent platform is deployed in an open environment. What can be done against attacks from malicious hosts? Can malicious hosts be identified or is it possible to limit what such hosts can do to agents? The next sections explore this issue further.

4 Secure Agent Migration

Insider attacks form the main security risk in mobile agent systems, in particular when agents are used in open environments such as the Internet. In closed environments in which all hosts are trusted this is not a problem: all hosts are known to behave correctly. 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. The problem stems from the intrinsic features of mobile agent systems: agents are executed on hosts that can view and alter their (internal) state, or even delete the agent altogether. Protecting agents from malicious hosts can be achieved by ensuring the following four points:

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) forms the basis for detecting malicious hosts and/or preventing them from doing any harm. For example, a number of techniques (3; 13; 26; 28) have integrity of agent 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 encryption of sensitive data.

The main focus of this and the following sections is *the detection of breaches of integrity in migration paths of mobile agents.* The general problem of protecting the agent from malicious hosts that could alter the agent's code or data is not addressed directly. However, as stated before, techniques to detect whether an agent has been altered can be based on the integrity of an agent's migration path.

One fundamental (and unsolvable) problem with agent migration is that a malicious host can always delete an agent in its entirety. This can never be prevented. However, it is possible to detect *which* host deleted an agent. The only thing that is needed for this is the preservation of the integrity of the migration path of an agent. An agent owner can then simply follow the migration path of an agent and conclude which host deleted the agent. Of course for this to work, the malicious host should not be able to forge the migration history of an agent. Once a malicious host is identified as such, the host can be put on a black list, thereby preventing further malicious behaviour of the host in question.

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 able to read and alter information stored inside agents.

Although agents can decide to only migrate to trusted hosts, i.e., hosts that have a valid (signed) certificate, a trust relationship does not always give full guarantees on the correct behaviour and intentions of hosts. An agent's migration path provides a means to detect breaches of integrity.

There are roughly two different approaches known in the literature 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 (29). 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 (digital) signature of host A over some content, in this case, agent x. In the remainder of this paper the word 'host' can be read as meaning either a normal host (computer) or an (AgentScape) location - a number of hosts administered and hosted at the same domain. The additional steps needed when dealing with locations, as explained in Protocol 1, are left implicit since they do not add any essential details to the proposed protocols and do not increase readability.

4.1 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).

Protocol 2 below gives a more detailed explanation of a migration step, in which agent x migrates from host A to host B, using a trusted third party.

Protocol 2
The protocol for the migration of agent x from $A \to B$ 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. If verification fails, the migration is not authorized and aborted altogether.*
6. *host B starts the suspended agent.*[b]

Figure 3 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 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, cannot detect whether an agent has migrated from a malicious host to multiple malicious hosts returning

[b]For the sake of clarity, some details are omitted from this and following protocols. 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.

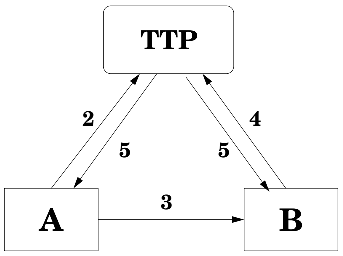

Figure 3 Agent migration using a central authority (trusted third party).

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.[c] This is known as a *home based approach.* The agent platform Mansion uses a dedicated service: the Agent Location Service (31) that implements this use of the initial host as a trusted third party.

The central point of failure and performance problems of this approach can, potentially, also be removed by exploring the option of using multiple TTPs. However, this leads to new problems such as replication of data and consistency between TTPs. This problem is left for future research.

4.2 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 (15). 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).

Protocol 3
The protocol describing 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: $[x, B \rightarrow C, [x, A \rightarrow B]_A]_B$. (Note that this signature will become a new link in the signature chain in step 4.)*

2. *host B sends the agent with the old signature chain, along with the new link of the chain to C.*

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

3. *host C verifies the signature chain stored with agent x and the new link it has received from B.*

4. *if the verification of the signatures is successful, host C adds the new link 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 that breaches of integrity of the migration path are noticed. The method, however, has some drawbacks: verifying the complete chain of signatures for long chains is 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 re-use 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.

In the next section a new distributed approach to integrity protection of migration paths is discussed. It uses an alternative signature chain concept that improves on the approaches discussed above. 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.

5 Scalable and Secure Agent Migration using Distributed Trust

This section introduces a new approach for the preservation of the integrity of migration paths of agents. One of the main design goals is scalability, as well as enhanced security features, improving on known solutions.

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

Briefly, the algorithm works as follows: Suppose agent x migrates along the path $A \to B \to C$. Each migration step is recorded within the agent. Each step is signed by a host. When agent x migrates from B to C, host B asks host A to sign the migration step ($B \to C$). The resulting signature is stored within agent x's agent container. When host C receives the agent and the signatures, it confirms receipt to host A. Protocol 4 below provides a detailed application of the algorithm to a single migration step.

Protocol 4
The protocol describes 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 $n \in \mathbb{N}$. The sequence number is stored inside the meta-data in agent x's agent container.

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 seqnr $(n+1)$ is in line with the seqnr (n), which was used when the agent migrated from host A to host B in the previous migration step. 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 agent x's agent container 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 if and when host C ships agent x to another host.*
6. *host C receives agent x's agent container, verifies the migration chain stored in this container, and contacts host A to acknowledge the receipt of agent x's agent container 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 to accept agent x's agent container; otherwise, C can accept the container and start agent x. After this step, assuming the migration from $B \rightarrow C$ is successful, host A can delete all info concerning agent x, or better still it can keep a record of the waiver so the migration path can be reconstructed if the agent is deleted by a malicious host in the future.*

Figure 4 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 Protocol 4, host C contacts host A to confirm receipt of an 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 an agent to the same host twice, 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.

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 cannot cut out a middle piece, because the sequence numbers will not match, ii) hosts cannot replace a middle piece, because the signatures in the chain of hosts will not match, iii) hosts cannot cut out or replace the tail of a migration chain (including cycles),

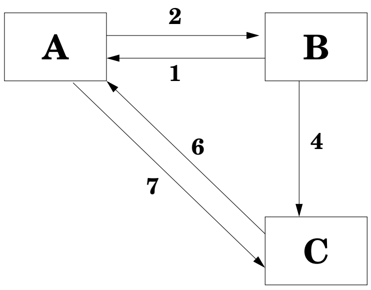

Figure 4 Agent migration using distributed trust.

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 liability of a single migration step is spread over two hosts: the previous host and the next receiving host, i.e., A and C, in the migration chain $A \rightarrow B \rightarrow C$. By this unique overlapping migration signature pattern by all participants, the migration of trusted to untrusted (or untrustworthy) is always registered (and signed).

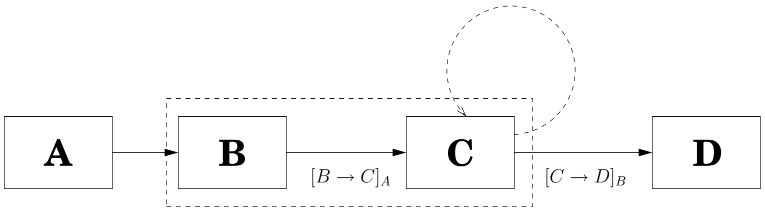

Figure 5 Conspiring hosts altering the migration path.

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 $A \rightarrow B \rightarrow C \rightarrow \cdots \rightarrow C \rightarrow D$, as depicted in Figure 5 (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 host in a migration step assumes that at least one of the two preceding hosts 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.

6 Discussion and Conclusions

Agent migration has been discussed in detail in this paper. Starting from unsecured agent migration, through migration that is secure against outsider attacks and ending with a discussion of migration protocols that can be used against insider (malicious hosts) attacks.

The main contribution of the paper is the introduction of a protocol that ensures that a breach of integrity in migration paths of mobile agents in large scale distributed agent systems will be detected. This approach distributes trust over two (signing) 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, as 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 additional value. Reputation and trust models (1) have been studied in the context of agent systems by, for example, (25; 11). Such models are currently unrelated to our migration path integrity protocol. Investigating if and how these results could benefit from each other is future work.

Our protocol 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 do conspire together the situation becomes more complex. A possible solution is to enforce that agents alternate between trusted and untrusted hosts (26) 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' (21), 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, though this is hard to enforce in practice, for example, host can have multiple IP-addresses, thereby 'cloaking' there real identity.

The mechanisms needed to instrument this approach are currently implemented and tested in the agent platform AgentScape (22).

Acknowledgments

This research is conducted as part of the ACCESS project[d] funded by the NWO TOKEN program. The authors also thank Stichting NLnet for their support.

[d]`http://www.iids.org/access`

References

[1] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *Proceedings of the 1997 workshop on New security paradigms*, pages 48–60. ACM Press, 1998.

[2] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-Middle in Tunnelled Authentication Protocols. In *Security Protocols: 11th International Workshop, Cambridge, UK, April 2-4, 2003: Revised Selected Papers*. Springer, 2005.

[3] 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.

[4] A. Csetenyi. Electronic government: perspectives from e-commerce. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, pages 6–8. IEEE Computer Society Washington, DC, USA, 2000.

[5] J. Cucurull, R. Mart, S. Robles, J. Borrell, and G. Navarro-Arribas. Fipa-based interoperable agent mobility. In *CEEMAS07*, volume 4696 of *LNAI*, pages 319–321. Springer Verlag, September 2007.

[6] J. Cucurull, B. J. Overeinder, M. A. Oey, J. Borrell, and F. M. T. Brazier. Abstract software migration architecture towards agent middleware interoperability. In *Proceedings of the 2nd Int'l Multiconference on Computer Science and Information Technology (IMCSIT)*, volume 2, pages 27–37, October 2007.

[7] A. Freier, P. Karlton, and P. Kocher. Secure Socket Layer. *Netscape*, 1996.

[8] R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(02):147–159, 2001.

[9] M. He, N. R. Jennings, and H. F. Leung. On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985–1003, 2003.

[10] A. Helsinger, M. Thome, and T. Wright. Cougaar: a scalable, distributed multi-agent architecture. In *IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[11] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt. Developing an integrated trust and reputation model for open multi-agent systems. In *Proceedings of the 7th International Workshop on Trust in Agent Societies*, pages 65–74, 2004.

[12] B. Jacobs, M. Oostdijk, and M. Warnier. Source Code Verification of a Secure Payment Applet. *Journal of Logic and Algebraic Programming*, 58(1-2):107–120, 2004.

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

[14] N. Karnik and A. Tripathi. Security in the Ajanta mobile agent system. *Software-Practice and Experience*, 31(4):301–29, 2001.

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

[16] D. G. A. Mobach. *Agent-Based Mediated Service Negotiation.* PhD thesis, Computer Science Department, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, May 2007.

[17] D. G. A. Mobach, B. J. Overeinder, and F. M. T. Brazier. WS-Agreement based resource negotiation framework for mobile agents. *Scalable Computing: Practice and Experience*, 7(1):23–36, 2006.

[18] A. Moreno and J. L. Nealon. *Applications of Software Agent Technology in the Health Care Domain.* Birkhauser, 2003.

[19] G. C. Necula and P. Lee. Proof-carrying code. In *Proceedings of the 24th Symposium on Principals of Programming (POPL).* ACM, 1997.

[20] G. C. Necula and P. Lee. Safe, untrusted agents using proof-carrying code. *Special Issue on Mobile Agent Security*, pages 61–91, 1997.

[21] J. 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.

[22] B. J. Overeinder and F. M. T. 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.

[23] B. J. Overeinder, M. A. Oey, R. J. Timmer, R. van Schouwen, E. Rozendaal, and F. M. T. Brazier. Design of a secure and decentralized location service for agent platforms. In *Proceedings of the Sixth International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2007)*, May 2007.

[24] B. J. Overeinder, P. D. Verkaik, and F. M. T. Brazier. Web service access management for integration with agent systems. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing, Mobile Agents and Systems Track*, 2008.

[25] S. D. Ramchurn, C. Sierra, L. Godo, and N. R. Jennings. A computational trust model for multi-agent interactions based on confidence and reputation. In *Proceedings of the 6th International Workshop of Deception, Fraud and Trust in Agent Societies*, pages 69–75, 2003.

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

[27] V. Roth and M. Jalali. Concepts and architecture of a security-centric mobile agent server. In *Proc. Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001)*, pages 435–442. IEEE Computer Society, 2001.

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

[29] 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.

[30] G. van 't Noordende, A. Balogh, R. F. H. Hofman, F. M. T. Brazier, and A. S. Tanenbaum. A Secure Jailing System for Confining Untrusted Applications. In *Proc. 2nd International Conference on Security and Cryptography (SECRYPT)*, pages 414–423, July 2007.

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

[32] G. van 't Noordende, B. J. Overeinder, R. J. Timmer, F. M. T. Brazier, and A. S. Tanenbaum. Building Secure Mobile Agent Middleware using the AOS Mobile Agent Kernel. *Int. J. of Intel ligent Information and Database Systems*, 2(3), 2008.

[33] M. Warnier, F. M. T. Brazier, and A. Oskamp. Security of distributed digital criminal dossiers. *Journal of Software (Academy Publisher)*, 3(3), March 2008.