Security in Large-Scale Open Distributed Multi-Agent Systems

M.A. Oey M. Warnier F.M.T. Brazier

Delft University of Technology, The Netherlands

1 Introduction

Designing large-scale distributed multi-agent systems that operate in open environments, such as the Internet, creates new challenges, especially with respect to security issues. Agents are autonomous, pro-active, communicative, goal-directed, often capable of learning, and sometimes mobile [8]. Mobile agents traverse the network to access services and resources they need to achieve the goals they pursue. The potential of mobile agent technology in sectors such as E-Commerce [17, 18], E-Health [29] and E-Governance [10, 52] is well recognized. In these sectors, security issues such as authentication, authorization, 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.

Many security requirements need to be addressed for large-scale distributed multi-agent systems in open environments. The focus of this chapter lies on security requirements specific for agent systems rather than security requirements for distributed computers systems in general. Section 2 identifies the most relevant security requirements for agent systems. This set of requirements is a minimum that needs to be fulfilled for secure agent systems in open environments. Sections 3 through 7 discuss the security requirements and possible solutions in detail. The solutions are illustrated within the context of the AgentScape [20] agent platform. This platform has been chosen as it has been specially designed to be used in a large-scale, distributed, open environment. However, similar implementations of these solutions are possible in other agent platforms.

The chapter closes with an overview of a number of well-known agent platforms, such as AgentScape [20], Ajanta [23], SeMoA [41], and JADE [5] with its security extensions JADE-S [34] and S-Agent [16]. The discussion focuses on what techniques these agent systems have used to solve some of the discussed security requirements.

2 Security Issues in Agent Systems

An agent system, a specific type of distributed computer system, needs to address not only security requirements related to distributed computer systems, but also multi-agent system specific security requirements. This section identifies a minimum set of security requirements specific to multi-agent systems that needs to be fulfilled for it to operate securely in an open environment.

2.1 Principals in an Agent System

Conceptually, multi-agent systems are distributed, networked, computer systems in which **agent owners** run communicating **agents** that access **resources** on hosts, each of which runs an **agent platform** (i.e., an instance of an agent middleware) that is under the control of a **platform administrator**.[1] The bold terms are the major **principals** in a multi-agent system.

Each of these principals faces security threats. A secure agent system must protect these principals and their communication with other principals against security threats. For example, **secure communication** is needed to protect the communication between two agents, but also between two platforms and between an agent and a platform or the agent's owner. In a large scale, distributed system, such as the Internet, communication is usually over long distances and can be intercepted or monitored. In a closed environment, all principals in an agent system are known in advance and usually trusted, therefore, security measures are often implicit. However, in a more open environment, more explicit security measures are needed to guard against security threats.

Traditionally, security threats are described using the terms **confidentiality**, **integrity**, and **availability**: the CIA-triad [46]. Confidentiality refers to the ability to prevent access to those that are not authorized. Integrity refers to the ability to prevent any unauthorized modification. Availability refers to keeping resources accessible at all times to authorized parties. A prerequisite for guarding confidentiality, integrity, and availability is identity management [9], which encompasses **naming** and **authentication**. Naming is the ability to identify each individual principal in an agent system. Authentication is the ability to verify a principal's identity. For example, reliable authentication is needed as malicious parties may want to impersonate certain principals in order to gain access to that principal's privileges.

The next sections look at security threats in an agent system from the viewpoint of the two most important stakeholders in an agent system: the **agent owner** and the **agent platform's administrator**.

2.2 Security Threats for the Agent's Owner

An agent performs its actions on behalf of its owner, which is usually a legal entity, such as a human or an organization: the **agent owner**. The main security concerns for an agent owner are confidentiality and integrity of his agent, any data it carries, and any communication to and from the agent. Confidentiality is directly related to guarding the privacy of an agent's owner. For example, in an e-health environment, agents acting on behalf of patients carry privacy-sensitive information that should not be revealed to others.

Agent mobility introduces extra security risks, as agents run on hosts that are out of the control of the agent's owner. For example, malicious parties can start agent platforms with the intent to eavesdrop or manipulate agents that they host. This **malicious host** problem is hard to solve, as platforms in general have full control over the agents that run on them. The most effective solutions involve the use of *trusted hardware*.

[1] The term *agent platform* or *middleware* refers to software running on hosts to support agents; *agent system* refers to the whole system of agents, agent owners, agent platforms, platform administrators, etc.

Unfortunately, these solutions are usually also the more costly solutions to implement. Software-only solutions give less protection but are more practical to implement. The malicious host problem exists foremost in open environments. It is reasonable to assume that in closed environments all hosts are trusted to behave well and that adequate **authorization** mechanisms have been installed to prevent unauthorized users of the platform to have access to an agent's private data.

Availability of an agent is a requirement for an agent owner that can be implemented by an agent owner himself, possibly supported by a platform. For example, to make an agent more fault tolerant, an agent owner can start two (or more) copies of an agent and send them to different platforms, so that if one agent dies, the other can continue, keeping the agent available to its owner. Alternatively, an agent owner can trust a platform owner to take adequate measures to guarantee the availability of an agent platform.

2.3 Security Threats for the Platform's Administrator

A host's administrator can run an agent platform (i.e., an instance of the agent middleware) on his host. An agent platform enables visiting agents to (paid) access to a host's resources. The main security concerns for an agent platform's administrator (who is not necessarily the same as the host's administrator) are confidentiality, integrity, and availability of the agent platform, its resources, and any communication from and to the agent platform.

In open environments, a platform must prepare for deliberate attacks, from outside, as well as inside. Mobile **malicious agents** can first migrate to a platform and try to attack a platform from the inside. Attacks typically include gaining unauthorized access to a host's resources or accessing the data of other agents running on that host. To protect against the threat of malicious agents a **resource access control** mechanism must be installed that enforces an authorization mechanism that determines who is allowed to access which resource and to what extent.

A typical resource in an agent system that may be the target of availability threats is the **lookup service**. The lookup service is a database that keeps track of the current locations of all agents in an agent system. An agent system needs this information, for example, to deliver messages to agents sent from other agents. In an open environment, an attacker could start an agent platform, join the agent community and subsequently fill the lookup service with false information about locations of agents. This attack renders the information in a lookup service useless and consequently paralyzes an agent system as a whole. This specific attack is a form of a **Denial-of-Service** attack and illustrates the necessity of a **secure lookup service** which guarantees the correctness of its information. Without it, a platform administrator cannot guarantee the availability of the agent platform.

2.4 Summary

The next list summarizes the security requirements discussed in this section. Each requirement is either a prerequisite for security or is associated with a threat for one of the two main principals in an agent system: agent owner or platform administrator.

- *Prerequisite*: **Naming and Authentication** – the ability to verify the identity of principals
- *Prerequisite*: **Communication Security** – confidentiality and integrity of data sent between agents, services, hosts, etc. must be guaranteed.
- *Agent owner*: **Malicious Host** protecting an agent's confidentiality and integrity even if it runs on a malicious host.
- *Platform administrator*: **Malicious Agent** – protecting a host's confidentiality and integrity from malicious agents
- *Platform administrator*: **Secure Lookup Service** – guarding the information in the lookup service.

This set of security requirements forms a bare minimum for agent systems in open environments. In addition to these security requirements other requirements common to all distributed computer systems need to be addressed, such as fault tolerance, availability, backups, traceability, etc. For agent systems in specific domains more stricter security requirements may apply as well. For example, in privacy sensitive environments **anonymity** may be an important requirement.

The remainder of this chapter focuses on the specific security requirements in order. Each requirement is discussed in more detail and one or more possible solutions are presented. Sections 3 and 4 discuss the prerequisites naming and authentication, and communication security. Next, Section 5 focuses on the main security threat to an agent owner: the malicious host. Finally, Sections 6 and 7 look at threats to a platform administrator and discuss the malicious agent and a secure lookup service.

3 Naming and Authentication

As mentioned above, identity management is an important security requirement in an open, distributed agent system. The ability to name principals and authenticate them is an important part of identity management.

3.1 Naming

Before authentication can be done, principals in an agent system must first have a (unique) identifier: a name. This name does not have to be human-readable; it can be a meaningless string, as long as it is machine-readable. In principle, names can be static, which means they do not change over the lifetime of a principal, or dynamic. For humans and organizations static names are a more logical choice, however for (mobile) agents in an agent system, dynamic names have their use. For example, agent names could contain a reference to the location where an agent resides (*location-dependent names*, see also Section 7), which makes locating the agent trivial. However, for the remainder of this chapter it is assumed that principals have **globally unique identifiers** (GUIDs), which are static names. The term global does not necessarily have to imply that the identifier is unique in the universe, but it suffices that the identifier is unique

within an instance of a running agent system. It can be assumed that in any agent system, something similar to GUIDs is used to name principals.

Another property of naming is whether principals can have more than one name. For example, if an agent has multiple names, it can use these names as *pseudonyms*. Pseudonyms can be used to implement **anonymity** [51]: an agent can use a different pseudonym for each interaction with another agent.

To illustrate, AgentScape [20] (see Section 8) actually has two naming schemes. First, agents are identified internally by GUIDs, which are kept private to the middleware. Second, agents are externally visible through their (static) **handles**. Each agent can have more than one handle at a time, which allows them to implement a form of anonymity as each handle is a *pseudonym*.

Note that naming is not sufficient for authentication as there is no mechanism to verify that a name corresponds to the correct principal. Authentication is discussed in the next section.

3.2 Authentication: a Public Key Infrastructure

Many ways of authentication are known and used in the world. One well-known method is the use of username and password combinations. Only if the correct password is supplied is the user authenticated. A more elaborate scheme requires a PKI, a **Public Key Infrastructure**, that uses asymmetric key encryption also known as public-key cryptography [27]. Every principal (agent, user, host, etc.) that needs to be able to be authenticated creates a key-pair, consisting of a *public* and a *private* key. These keys have the property that data encrypted with one key can be decrypted by the other, and given one key it is computationally infeasible to derive the other key. Every principal publishes its public key to the world, but keeps its own private key private. The identity of a principal can now be verified by checking whether the principal can correctly decrypt a message encrypted with the principal's public key. Only the real owner of that public-private key pair can decrypt the message assuming the private key has been kept private. Whether the public key is indeed the public key of the correct principal and not of an imposter impersonating that principal using its own generated keypair is the task of the PKI.

The public key infrastructure is used to securely publish public keys of principals. A public key is published together with the corresponding principal's personalia. This combination is called a **certificate**. This certificate is also (digitally) signed [25] by a **Certificate Authority** (CA), after it has verified that the public key and principal are indeed legitimate, which, for example, involves showing a passport to an official of the CA. All principals publish their public key in the form of signed certificates. Anyone who trusts the signing CA can use that certificate and be confident that the public key and the principal both stated in the certificate are valid and belong to each other. In short, an agent system is able to solve the authentication problem by using a PKI, where all principals create a public/private keypair and a trusted CA signs all corresponding certificates.

For completeness, signing a certificate is done by adding an encrypted version of the certificate (actually, a hash of it) to the certificate. Encryption is done with the private key of the CA, which means that everyone can verify the signature with the

public key of the CA, but nobody can forge the signature. The public key of the CA is assumed to have been distributed securely to all participants. Note that safely distributing the certificates of a handful of CAs is more feasible than distributing the certificates of all participants.

In AgentScape, a public key infrastructure is installed. Agent owners, locations, and hosts have public and private key pairs. This ensures that locations and hosts can mutually authenticate and set up secure communication channels, using SSL (see Section 4).

3.3 Linking an Agent and its Owner

In many situations, an agent must be uniquely and undeniably linked to its owner (e.g., a human or organization). This link is part of authenticating an agent and is necessary, for example, to charge the owner if agents make purchases on the web or to help determine liability whenever agents misbehave. This section discusses, in the context of agent based systems, how agents can be 'bound' to their owner.

As mentioned before, it is assumed that an agent can be identified by a GUID. Conceptually, an agent consists of meta-data, (executable) code, and data that an agent has 'found' on a particular host. The meta-data of an agent contains at least the following: the GUID of this agent, the name of this agent's owner, and a signed (by the owner) hash of this agent's code. The signature ensures that agent and owner are bound to each other. For authentication to succeed, it is important that the public key of an agent owner is stored in a PKI.

For example, in AgentScape, when an agent is injected, the agent platform checks if the agent code is indeed signed. If verification is successful the agent obtains a GUID and a handle is returned to the agent owner. Assuming the owner keeps this handle secret, it can be used to communicate between agent and owner. Next, the injected agent is started by the agent platform. If the agent misbehaves in some way, the owner can be contacted and be held responsible for the agent's actions. The agent injection procedure is similar in other agent systems.

4 Communication Security

In distributed agent systems communication is manifold. The (distributed) components that make up the agent system's middleware need to communicate with each other to maintain a running agent platform, and the agents themselves communicate with each other, with (external) services, and with the platform. Confidentiality of communication between agents, services, hosts, etc. must be guaranteed. Threats can be external or internal. External eavesdroppers may want to listen in on agents to find out privacy-related information, may want to disrupt the agent platform, or may want to impersonate other agents or services, etc.

4.1 Common Security Attacks

Many types of attacks are known that target communication channels. Two very common attacks are man-in-the-middle attacks and replay attacks. This section briefly explains these two attacks to illustrate the kind of attacks possible on communication channels. For clarity, the names Alice, Bob, and Mallory, which are commonly used in cryptography, are used to explain these security attacks.

With a **man-in-the-middle attack**, an attacker (Mallory) tries to put himself in between the communication path of two others (Bob and Alice). When Bob tries to contact Alice, Mallory steps in posing as Alice, and forwards the request to Alice, but now pretending to be Bob. As a result, Bob and Alice both *think* they are talking privately to each other, while in fact Mallory is able to intercept all data that is sent by them. This form of attack succeeds if Mallory is able to impersonate Bob and Alice successfully. A **replay attack** is a threat where an attacker deliberately resends or delays messages that were sent previously. Since the attacker does not alter messages, the receiving party does not have any reason to refuse incoming messages, unless it has the ability to detect that a message is a resent copy or an old delayed message. To see the effect of a replay attack, consider the consequences of a message that contains a money-transfer order for an online bank application.

4.2 Encryption

A common technique to guarantee confidentiality and integrity of communication is encryption. Two well-known techniques are **SSL-based communication** [32] and **IPsec** [26]. SSL is widely used to provide secure connections to webservers (e.g., the *https* protocol). All data sent over a connection between two parties is encrypted with a shared-key. The key is exchanged in a hand-shake phase during the setup of the connection. Authentication, that is, the method to ensure that a party actually is who he/she claims to be, usually involves a **certificate** signed by a trusted third party (i.e., a certificate authority) whom both communicating parties trust. After the hand-shake successfully completes, both parties can be assured that their communication remains confidential. In agent systems, the setup of the encrypted SSL-connection is usually done by the agent middleware. As a consequence, the agent middleware's internal communication is also secure. In addition, all agent-to-agent communication is automatically encrypted transparently, under the assumption that communication is supported by the agent middleware, which is almost always the case.

The other technique is IPsec. This protocol uses encryption at a much lower level than SSL does. SSL uses encryption at the application level, which means the encryption is performed by the application, an agent platform. In contrast, IPsec is performed by the underlying operating system. The advantage of this technique is that both agent application developers and agent system developers have secure communication available to them automatically. However, most agent platforms provide their own secure communication (usually via SSL) as it is relatively simple to implement and they then do not have to rely on the underlying operating system to support IPsec.

For example, AgentScape currently supports SSL-based communication between hosts and locations. This provides the basis for hosts/locations to authenticate each

other. Furthermore, all messages transmitted between hosts/locations, including migration of agents, are encrypted to ensure confidentiality. The PKI is used to link host/location identities in a secure manner.

5 Malicious Hosts

To an agent owner, protecting an agent's code and the data it has acquired while traversing a network is his main security concern. Especially, when agents are used in open environments such as the Internet, where agents execute outside the control of the agent's owner. Hosts on which an agent resides may be malicious, yet temporarily have complete control of the agent's runtime environment. It is often infeasible to determine the trustworthiness of hosts in advance in open environments.

Unfortunately, in practice, it is almost impossible to protect a migrating agent if it runs on hosts that are outside the control of an agent's owner. Such a **malicious host** can view and alter an the agents (internal) state, or even delete the agent altogether. However, some hardware and software solutions exist that try to provide security guarantees or at least allow others to detect that an agent has been tampered with by a malicious host. Below some of these solutions are discussed.

In principle, protecting agents from malicious hosts requires [39]:

1. Protecting the integrity of the migration path of an agent
2. Protecting the integrity of the agent's data and (binary) code
3. Ensuring confidentiality of the agent's data
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. In principle, 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 [6, 22, 39, 43] have integrity of agent migration paths as a premise, and can be used to detect tampering with the agent (items 2 & 4). Solutions to protect an agent's migration paths are discussed in more detail at the end of this section (Section 5.5). Before that, some solutions to protect an agent's integrity, confidentiality, and control flow are briefly presented.

5.1 Trusted Hardware

A technique that in principle can offer the most protection is using trusted hardware (Trusted Computing [49]). Trusted hardware, such as the Trusted Platform Module (TPM), provides guarantees of the hardware's behavior. A TPM is a piece of hardware within a computer that cannot be tampered with. It can perform cryptographic functions and store cryptographic keys securely. Software manufacturers can use a TPM to guarantee users that their software running on a host has not been tampered with. A TPM can create a hash of the hardware and software of a computer and check whether

anything has been modified. Agents can use this information to detect whether to trust a host or not, depending on whether they trust the software manufacturer who created the agent middleware running on the host.

Another use of a TPM is for an agent to let certain critical operations be performed by a TPM. An agent sends any input encrypted to the TPM, the TPM then operates on the data and sends the result back to the agent. The result is encrypted in such a way that only the agent's owner can decrypt it after the agent returns to its owner. Unfortunately, both uses of the TPM require specialized hardware. Requiring all computers to have specialized hardware restricts the use of it for agent systems in an open environment. Therefore, the remainder of this section focuses on software-only techniques.

5.2 Protecting an Agent's Integrity

An agent needs to protect both it's agent code as well as any data it carries. As mentioned before, without trusted hardware, an agent cannot protect this data from being modified by a malicious host. However, it is possible for an agent to detect, after a migration from a potentially malicious host, whether that host has made any unwanted modifications to the agent's code and/or any data that the agent carried. The solution is the use of digital signatures.

To protect an agent's code, the agent carries a signature from the agent owner over a *hash* of the agent's code. After migration, an agent platform checks whether the agent owner is authorized (trusted) to run agents and whether the signed hash in the agent matches the actual hash of the agent's code it received. If not, then the agent has been modified and the agent platform can notify the agent's owner and refuse to start the agent. Since only the agent owner can generate this signature, a malicious host cannot modify the agent's code without being detected.

The data that an agent carries can be protected as follows. A hash is calculated of each piece of data that needs to be protected. Then all these hashes are stored in a table together with some meta-data on each piece of data, such as its location within an agent. This table is then signed and stored within the agent. If a malicious host modifies or removes a part of the protected data or the table, the signature will not match and the modification will be detectable by the agent or the agent owner.

Unfortunately, an agent cannot carry its own private key to sign data, because a malicious host would then also have access to it and be able to fake signatures. Consequently, an agent cannot sign its own data. Instead, an agent owner or a trusted third party should sign the table. The agent has to migrate to the agent owner's host or to the trusted third party's host first to get the signature. Migration to a trusted host makes this scheme a little cumbersome. If, however, the migration path of an agent can be securely tracked (migration path integrity), other solutions become possible [6, 22, 39, 43].

5.3 Protecting an Agent's Confidentiality

To protect a malicious host from reading confidential data that an agent carries, it is sufficient to encrypt that data with the public key of the agent's owner, which ensures that only the agent's owner can read the data after the agent has returned to the owner. Encryption can be done by an agent itself on the (trusted) host where it has acquired

the data. Unfortunately, after encryption an agent itself does not have access to the data either. If it needs access to encrypted data and it trusts the host it is on, it can set up a secure connection to the agent's owner and ask it to decrypt the data.

5.4 Protecting an Agent's Control Flow

Unfortunately, protecting an agent's control flow on a malicious host is virtually impossible without dedicated trusted hardware. Basically, an agent would need to control (or at least monitor) the runtime environment of the host on which it runs, which is impossible as the host controls it. For example, a malicious host could deny or limit access to resources that an agent has previously negotiated for. If the agent does not check for this, it would never notice the fraud. Even worse, even if an agent checks for fraud, a really malicious host could change the control-flow of the agent to skip this check.

The best an agent can do is to use the techniques described above to protect the confidentiality and integrity of the data it carries, to at least detect whether the agent has been tampered with. The agent can then can redo its operation again at a more trusted host after migration.

5.5 Protecting an Agent's Migration Path

One fundamental (and unsolvable) problem for 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, a 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 behavior of the host in question. The main focus of this section is the *detection* of *breaches* of integrity in migration paths of mobile agents.

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, that is, hosts that have a valid (signed) certificate, a trust relationship does not give full guarantees with respect to a host's behavior and intentions.

A number of solutions exist to protect the integrity of an agent's migration path. A possible solution uses a centralized **trusted third party** (TTP) [15] to authorize and keep track of migration paths of agents. The trusted third party can be physically located elsewhere and does not have to be part of the agent system itself. However, all users of an agent system must trust that the trusted third party is not malicious and cannot be compromised. Secure communication channels (see Section 4) to the TTP and digital signatures [25] (see also 5.2) are used to secure the migration protocol against

fraud. Unfortunately, malicious hosts can simply migrate an agent between them without informing the TTP. Furthermore, a centralized TTP forms a single point of failure and can become a performance bottleneck for large-scale agent systems. Multiple TTPs can be used to improve scalability. For example, in the **home based approach**, each agent uses its own initial (trusted) host as its TTP. Alternatively, Roth [39] uses *co-operating agents* that use each other as TTP.

A decentralized solution to secure the migration path of an agent is **signature chaining** [45], which stores an agent's migration path in an agent itself, together with an agent's code and data. Digital signatures are used to protect the migration path against tampering by a malicious host. In this method, each host adds the next migration step to the migration path that was already stored in the agent and signs the entire path, including the signatures of previous hosts in the migration paths. By signing the entire migration path the signatures of all participating hosts are chained together. Each new migration step adds another connected link to the signature chain. Unfortunately, verifying long signature chains is computationally intensive, and a malicious host can remove arbitrary cycles from a migration path if an agent (accidentally) visits the same malicious host for a second time [45].

Another scalable solution that uses the notion of distributed trust to secure migration paths is described in [53]. In this solution, other hosts in the migration path authorize and check each following migration step. Increasing the number of hosts required to authorize a migration makes the migration protocol more resistant to cooperating malicious hosts. 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, [36, 19].

6 Malicious Agents

The previous section discusses the malicious host problem. This section focuses on the complementary problem: **malicious agents**. Just as agent owners want to protect their agents against potentially malicious hosts, so do platform administrators want to protect their hosts against potentially malicious migrating agents. Malicious agents typically attempt to gain access to resources on a host they are not authorized to use. Such access includes attempts to access private data of the host, private data of other agents, or to use additional computational resources that have not been negotiated. Fortunately, there are a number of techniques that a platform administrator can apply to reduce the threat of malicious agents and control their access to a host's resources. This section discusses a few of these techniques and subsequently focuses on the subject on how to configure and manage access to resources for agents.

6.1 Sandboxing Agents

Most solutions to securing hosts from malicious agents entail monitoring every action that an agent attempts on a host. Whenever an agent makes a call to the middleware API, it is intercepted by a **security manager**. The security manager checks the system policy to determine if an action, such as migration and resource access, should be allowed or denied. For example, a host could decide that it does not allow agents to use remote web-services (i.e., not running on the local host). Every attempt to contact a remote web-service will be blocked by the security manager.

Many agent platforms are Java-based [14], and in Java one of the primary solutions towards securing mobile code is to execute any remote code in a protection domain or **sandbox**. A sandbox limits the set of operations that the remote code may call. For example, sandboxing typically restricts network access as well as access to the local filesystem. Java provides agent system programmers the tools to define sandboxes by using a *security manager* and/or custom *class loaders*. In Java the actual sandbox is enforced and implemented by the underlying JVM, for interpreted scripting languages such as Python and Safe-Tcl the sandbox is implemented by the interpreter. For C or C++ (binary code) agents are 'jailed' [50].

Sandboxing and jailing 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. An alternative solution is to only run agents of *trusted* owners. Whom to trust is up to the platform administrator. In this solution, agents are only trusted if they are signed by a reputable software manufacturer, whom the user trusts not to provide malicious agents. The simplicity of this scheme is also its weakness: the security of the system lies in the belief that the signer is trustworthy. The weakness of this system has already been shown as digital signing certificates have been issued to people masquerading as a representative of a well known software maker [12]. Furthermore, small and open source software makers may not have the financial capability to purchase such signing certificates. Of course, digital signatures can be combined with sandboxing to create a more robust security solution.

Finally, a more elaborate security approach is the use of proof-carrying code [30] (applied to the mobile agent paradigm described in [31]). Agents carry a *machine-verifiable proof* with them that specifies their expected and acceptable behavior. Each host is equipped with a theorem prover to ensure that an agent's code indeed adheres to its specification. Unfortunately, constructing the proof is very labor intensive [21], which makes this approach less practical.

Sandboxes and security managers restrict an agent's actions. However, a security manager first needs to know when to allow or deny an agent's request to access a resource: **access control**. In a flexible environment, principals may first want to negotiate about which resources they need, to what extent, and at what price. The outcome of this *resource negotiation* is input to the security manager that monitors and authorizes access to resources as negotiated. For example, the WS-Agreement standard [3] which provides a negotiation protocol for the domain of web services can be used. Mobach [28] has applied and extended this standard in the field of distributed agent systems.

Specifying security permissions can be an elaborate job, prone to mistakes. The

remainder of this section discusses how the combination of **roles** and sets of predefined **policies** simplify this task. Security policies allow users of agent systems to manage the security features of the multi-agent system of their choice. Developers of agent systems have the opportunity to ship a number of security policies with their software. For example, an effective default policy is one that will not prevent users from performing vital tasks, but will protect the host against some of the most common security issues. In contrast, 'high security' policies should be used in security critical environments. Such policies are very restrictive. Below a security policy framework is discussed and illustrated within AgentScape [20].

6.2 Resource Access Control

Once the basic security features, such as an agent naming scheme and authentication (see Section 3), are in place, the next requirement is an authorization mechanism. Conceptually, an authorization mechanism needs to specify who is allowed to do what and to what extent. There are a number of principals involved in any agent platform. For example, principals in AgentScape are locations, world administrators, resources and their administrators, and agents and their owners. Similar principals can be identified in any other agent platform. In any agent platform agents can perform a number of basic actions to achieve their goals, such as communication, migration, access to resources, etc. Controlling which principal can perform which action is a structure that can be readily managed using a **Role Based Access Control** (RBAC) [44, 54] mechanism.

6.3 Roles, Users, and Permissions

RBAC is an access control architecture that models roles, users and permissions. RBAC is designed to reflect real-world relations between users and permissions. Each role is associated with a set of permissions corresponding to logical tasks that users can perform. Users are assigned one or more roles. The advantage of this setup is that changing the permissions of a whole group of users with a specific role can be easily done by simply changing the permissions of the corresponding role.

Defining roles, users and permissions can be straightforward. First a number of permissions are defined and assigned to roles. Users are then associated with these roles. Table 1 shows some example (Role, Permission) pairs, denoting the capabilities of each role. Note that each role can have multiple permissions. Table 2 assigns roles to a set of users. These users are shown as textual names, but would in practice be represented by a unique identifier.

Role	**Permission to perform action**
BasicAgent	Migrate, Execute
TrustedAgent	Migrate, Execute, AccessRes
AgentOwner	Inject, GetResult
ResourceAdmin	AccessRes, ChangePerms, GetLogs

Table 1: RBAC Example Role Permission Table

Role	User
BasicAgent	SimpleAgent1, SimpleAgent2
TrustedAgent	ClaireTradingAgent, DaveStockAgent
DatabaseAccess	Alice, Claire
ResourceAdmin	Trent, Steve

Table 2: RBAC Example Role User Table

Agent owners form the base of the trust mechanism. They are ultimately responsible for the actions of their agents. Therefore, by default, agents hold the permissions granted to their owners, but these permissions can be further restricted when appropriate. Access to resources is explicitly specified in an RBAC policy.

The RBAC system can be dynamically updated, that is, roles can be changed, users can be added or removed from roles, and permissions can be assigned and removed from roles. Determining, specifying, and managing roles, users, and permissions is the responsibility of an administrator of each host. Part of this management can be delegated to (privileged) users to keep the task manageable. For example, a database administrator can be given the right to manage permissions to databases for which he is responsible. Agent owners can manage the rights of their own agent. Note that an agent owner cannot give its agents more rights than he himself has been given by a platform's administrator.

In an open system, every agent platform is autonomous. Therefore, each host can have its own RBAC policy. In addition, if multiple hosts cooperate and one single administrative domain (called a *location* in AgentScape terminology) each administrator of a host can define different (e.g., stricter) restrictions for its resources than a location administrator and vice versa. Both policies are enforced together; actions are only permitted if both policies agree.

6.4 Security Manager

To enforce resource access control, every action of an agent must first be authorized by an RBAC system before the action can be executed. Whenever an agent attempts to perform a security relevant action, a Security Manager checks whether the agent is authorized to perform this action. This check is a two-step process. First, the Security Manager determines the GUID of the agent and determines the role, or roles, of which the GUID is a member. Second, the Security Manager determines if one or more of these roles is authorized to perform the requested action.

It is worthwhile to note that not only a platform's administrator, but also an agent owner needs to trust the security manager. After an agent owner has negotiated for resources and possibly paid for access, an agent owner expects the security manager to grant access as negotiated. Similar to monitoring of Service Level Agreements (SLA) a trusted third party module can be used to monitor and log the communication between client (agent) and service provider (host) [37].

6.5 Parameterization of Permissions

A selection of the basic security relevant actions used in AgentScape is shown in Table 3. In most agent systems similar actions can be identified. These actions reflect the basic abilities of agents. The permissions for these actions can be extended with *parameters*. Parameters are used to further refine the granularity of permissions. For example, negotiation can be restricted to specific types of resources. Parameters are defined in parentheses. A special parameter, '*', is supported to allow *all* types of an action to be permitted by a role. This notation is used to avoid having to explicitly specify every type of resource and every location when wishing to grant access to them. Permissions are positive, that is, if access to a resource is not explicitly granted, access is denied.

Action	**Principal**	**Description**
Migrate	Agent	Migrate from one Location to another.
Inject	Owner	Launch an Agent in a Location.
AccessRes	Agent	Access a resource provided by a location.
Negotiate	Agent	Negotiate access to a remote location.
Lookup	Agent	Access yellow or white pages lookup service.
SendMsg	Location/Agent	Send a message to a remote location.
RecvMsg	Location/Agent	Receive a message from a remote location.

Table 3: Common Security Relevant Actions

In most cases, locations and hosts typically utilize generic policies for all agents. That is, most locations and hosts are not expected to specifically restrict access to resources, unless these resources are of specific importance. For example, most hosts will allow all agents access to CPU and memory resources, but access to special databases are more carefully controlled.

Parameterization simplifies expressing permissions for roles, and also allows more fine-grained access for system resources to be defined. This can be used, for example, to define policies that limit the locations to which agents may migrate. To illustrate parameterization consider the Role/Permission table shown in Table 4. In this table, normal agents (BasicAgent) are allowed to execute and access CPU and Memory resources. Only trusted agents, that is, agents with the role *TrustedAgent*, are authorized to access the price database.

Role	**Permission**
BasicAgent	Migrate(*), Execute, AccessRes(CPU,Memory)
TrustedAgent	Migrate(*), Execute, AccessRes(CPU,Memory,PriceDB)

Table 4: Database Resource Role-Permission Table

6.6 Agent Injection

RBAC requires all users (agents, humans, etc.) to be associated with one or more roles. New human users are usually entered into an RBAC system by a location's administrator. However, new agents injected by human users can be automatically added by an agent platform in an RBAC system with the corresponding permissions as described by an agent platform's administrator. The agent injection protocol in AgentScape is as follows. When a principal wishes to inject an agent into an AgentScape location, the principal first contacts the location and they perform a two-way authentication. Once authenticated, a location will accept agents injected into that location by a specific principal if, and only if, the principal is authorized to perform injections.

Once an agent is injected into a location, the location assigns a GUID to the agent instance. This GUID is also automatically entered as a new user into the *Role-User* table of both the location and the host that is going to run the agent, and is assigned to, at most, the same roles as the owner. Owner roles are defined by each location individually. In addition, default roles can be used for unknown agents and owners. To limit the growth of a *Role-User* table, an agent's entry can be removed as soon as the agent finishes or successfully migrates to another location. After successful migration, the GUID of the agent will be entered into the *Role-User* table of the receiving location and host. If owners are removed from a role, any agent belonging to that owner loses that role.

6.7 Security Policies

While security can be a major concern for resource and location administrators, it is not always the case that these principals are either particularly interested, or trained to, define their own security policies. For this reason, it is advisable to have a set of predefined default policies. These predefined policies range from simple, non-restrictive policies, used for agent systems deployed in a well known environment, to stronger, restrictive policies, where agent systems operate in a more hostile environment. These two extremes are described in the context of the following two case studies: a closed world and a hostile world.

In a simple *closed world environment*, locations are controlled by well known entities and are all trusted. Communication between locations is cryptographically secured and each location is known and trusted by every other location. The major threat to the middleware is that of malicious agents. Agent owners must be authenticated. Once authenticated, agents are authorized to perform any and all actions. Therefore, the authorization mechanism is not used for access control, but is instead used for auditing purposes: whenever an agent performs a security relevant action, it is logged for possible later examination by the location administrator. While a simple system is common in small, closed environments, the provision of services on the web, with the associated access of these services by software agents demonstrates that such an environment cannot be assumed.

In a *hostile environment* locations are controlled by entities that are not always known by every principal. Agents are authenticated by their initial location as before, but the authorization mechanism is now used to enforce location-specific restrictions.

The security manager monitors usage of specified resources and ensures that all accesses are restricted by the negotiated limits. Any breaches of these limits are logged and execution of the agent responsible is immediately suspended. Migration is only authorized between the original 'home' host–the host where the agent started–and remote hosts. Therefore, migration from one remote host to another forces an agent to first return to the home host. This is enforced to prevent malicious hosts attempting to inject or read data developed from a prior migration. For example, the result of a price check from a prior website should not be available when performing a price check at a competitor's site.

Within a hostile environment, not only locations and hosts may want to constrain the actions of agents, but also agent owners may want to restrict the actions their agents are allowed to perform on their behalf. These actions include the ability to negotiate, migrate, inject, access resources, purchase items on the web, etc.

In summary, the security architecture outlined in this section and illustrated within the AgentScape agent system provides a flexible means to define and manage agent access to specific functionality. Flexibility is provided in two areas: firstly, hosts and locations have the ability to control access to resources that they control. Secondly, owners can constrain their agents from performing actions that, while they are authorized by the locations and hosts, are not desirable to the owner. For more information see [35].

7 Secure Lookup Service

Every distributed agent system has some way of naming agents, and a way of mapping agent names to their location. Finding the location of an agent is useful, for example, for co-locating agents, that is, migrating agents to run on the same host to improve performance by reducing communication costs. Sometimes the names of agents already contain a reference to their location (*location-dependent names*), in which case, resolving the name to a location becomes trivial. However, with location-dependent names, agents do not have stable names as after a migration their names will have changed. Such agents are more difficult to track for other agents. With *location-independent* names, the names remain stable after migration, but the agent system needs a **lookup service** to map an agent's name to its current location.

A **lookup service** is a generic name for a global service that keeps track of where each agent is located and how to communicate with it. Another name often used is **white pages**. To prevent agents and services from impersonating other agents and services, the information in a lookup service must be trustworthy. However, in an open environment, where anyone can join the agent community, guarding the information in a lookup service is a challenge.

Scalable location services are essential in distributed systems and, in particular, for multi-agent systems. Domain Name System (DNS) is a very successful realization of a location service that resolves symbolic names to contact addresses (IP addresses). DNSSEC (Secure DNS) has been designed to support authentication preventing spoofing and man-in-the-middle attacks [4]. Both DNS and DNSSEC, however, are not designed to deal with highly dynamic entities such as mobile agents. The dynamic na-

ture of mobile agents in Internet-scale, open network systems requires a different type of approach for registering, deregistering, and retrieving location information. Scalability and integrity are of utmost importance as (up-to-date) agent location information is a prerequisite of successful agent mobility.

This section presents an approach for a scalable and secure location service.

7.1 Information in the Lookup Service

To make a lookup service secure, the service should store not only agent-ids and their current location, but also provide ways for its users to determine the validity (i.e., trustworthiness) of that information. In an open environment, users of a lookup service know that a lookup service may be compromised and may contain false information. One way to solve this problem is to have information published in the lookup service be signed by its publisher. The validity of the information returned by a lookup server depends on the level of trust placed in the signing publisher. Signing is done with public-key cryptography. This system requires a public-key infrastructure (PKI). The PKI ensures that public-keys are published in a secure and authenticated manner.

It is possible to integrate (a simple version of) a PKI and the lookup service. In this case, the lookup service holds two types of information: *Agent-Location* information and *Certificates*. The first piece of information is simply an (Agent-id, Location) pair, denoting the current location of a specific agent. This information is signed by the platform that currently holds the agent. Certificates are signed (location, public-key) pairs denoting that the specified public key is the public key of the platform running on that specified location. Note that it is possible for platforms to sign their own certificates: *self-signed* certificates. However, the trustworthiness of self-signed certificates is questionable in an open, hostile environment.

Each certificate is signed by a principal, which is either a root certificate authority or another platform. By allowing platforms to sign certificates containing public keys of other platforms a **web of trust** [13] can be build. Platforms should only sign a certificate for another platform if it trusts that the other platform is not malicious. Users of the lookup service can follow the chain of signatures in the certificates until they find a signature of a platform that they trust. This principle assumes that trust is transitive, that is, you trust everyone that is trusted by someone you trust. This principle may be too naive for some and they can restrict themselves to only trust information that is signed by someone they trust directly.

7.2 Using the Secure Lookup Service

This section describes how a secure lookup service is used in an agent platform, such as AgentScape. Agents are identified by a GUID and locations are identified by their name (locationname). Each location is responsible for publishing the location information for all of the agents it currently hosts. Furthermore, when a location starts, it first publishes its public-key via a certificate so others can verify the signature of all information published by this platform. This certificate is signed by a (root) certificate authority. Note that it is assumed that the public keys of root certificate authorities are well-

known and that everyone has obtained a copy of them in a secure manner. For example, platform administrators could exchange certificates in person.

In addition, the started platform can sign certificates for other platforms, indicating that it trusts and 'endorses' the information signed by those platforms. Which platforms to trust is usually determined by a platform's administrator and is typically stored in a list by the agent platform.

Below, the main functionality of a location service is briefly discussed.

Registering an Agent. When an agent is injected into an agent system its location is registered by the lookup service. First, the hosting agent platform creates a (agent GUID, locationname) pair. This information is signed by the hosting platform and published in the lookup service for others to find.

Deregistering an Agent. Deregistering is done by explicitly publishing that the agent does not have a current location anymore, indicating that the agent no longer exists. To prevent the information in a lookup service from growing too much, information in the lookup service could have an expiration time, that is, a lookup service automatically removes expired information automatically, unless the information is republished periodically. In this case, an alternative solution for deregistering an agent is to simply let an agent's location information expire from the lookup service, that is, to not republish the information for that agent. Note, that until the information expires, the lookup service will errantly report an agent's location, but this is not severe, as any attempt to contact the agent will simply fail with an error that the agent does not exist anymore. Choosing smaller expire times decreases this problem, but requires valid information to be republished more often.

Lookup of an Agent's Location. Agent lookup is done by searching the lookup service for all information pairs concerning an agent's GUID.

- If no information is found an agent does not exist (anymore).
- If multiple pairs are found, the platform filters the pairs by only looking at information signed by known and trusted platforms. The most recently published information indicates the current location of the agent. The recentness of information can be determined by including version numbers (e.g., timestamps) with each published piece of information.

A less strict trust-model would allow a recursive search for certificates of signing platforms until a certificate is found that is signed by a trusted platform.

Agent Migration. Agent migration is the most complicated scenario: care must be taken to ensure the agent is not accidentally 'dropped' or duplicated, for example, when one of the locations crashes or network connectivity is lost. Another important issue is to correctly update an agent's location in the lookup service.

The basic agent migration procedure is as follows, given an agent A, and locations X and Y.

- Agent A, running on location X, indicates its wish to migrate to location Y.
- Location X contacts location Y and transfers agent A.
- Location Y acknowledges to location X that agent A has been received.
- Location X stops republishing location information for agent A, but maintains a forwarding pointer for agent A to location Y in case other agents try to contact agent A on the old location.
- Location Y publishes that agent A is now located at location Y. As this piece of information has a higher version number than the previous information published by location X, this marks location Y as the current location of agent A.

7.3 Scalability

The previous section focused on the problem that the information in a lookup service must be authenticated and its integrity guaranteed. Another problem to tackle is scalability. In a distributed environment with potentially many hundreds of thousands of agents (or more) and many migrations, the lookup service can quickly become a performance bottleneck, especially if a centralized lookup service is used. One technique for scalability is Peer-to-Peer technology. For example, a **distributed hash table** (DHT) [38, 47, 42] is a decentralized lookup datastructure that is similar to a hashtable and aimed at performance.

A DHT stores (key, value) pairs and allows quick retrieval of the value associated with a particular key. The data can be spread over the participating nodes, but can also be replicated to increase lookup performance and/or to make the system more fault tolerant. A DHT is a self-managed datastructure. The nodes themselves are responsible for balancing the load and maintaining the data. Nodes can dynamically join and leave the DHT without disrupting the service. These properties make a DHT very scalable, and therefore, make it a good candidate for implementing a distributed lookup service.

In a lookup service based on a DHT, the (key, value) pairs stored in the DHT are the signed (agent-id, location) information pairs. An agent's location can be quickly retrieved via the DHT. Furthermore, each platform's certificate is stored as a (location, certificate) pair, making verifying signatures straightforward. Note that certificates are relatively static which means that they are easily cached at each host, making lookups in the lookup service necessary only for unknown public-keys, or when the cached copy is too old. Caching increases the performance of the distributed lookup service even further. Experiments in AgentScape with a secure lookup service based on a DHT, as described in this section, have shown promising results with respect to performance [33].

8 Agent Systems Overview

Many dozens of agent systems have been designed and developed over the last ten years or so. Some of them have reached quite a mature state and have an active community

supporting and using the agent system. This section briefly introduces and discusses a few representative agent systems: AgentScape [20], Ajanta [23], SeMoA [41], and Jade [5]. These agent systems are chosen because they are well-known and/or have a focus on security aspects. Each of these systems provides centralized access control. In contrast, the security solutions presented in the previous sections all emphasize a distributed solution.

The discussion of each agent system focuses on their security architecture and the different approaches taken by these agent systems to deal with individual security requirements. An extensive and detailed discussion of each agent system is out of the scope of this chapter.

8.1 AgentScape

AgentScape [20] is a middleware layer that supports open, distributed, large-scale agent systems. It was designed especially to be used in a large scale, distributed, heterogeneous, open environment. Its design provides minimal but sufficient support for agent applications within an open environment, and can be extended to incorporate new functionality or adopt (new) standards into the platform. AgentScape is written in Java and therefore runs on multiple operating systems. It also supports agents written in different programming languages, such as Java, Jason [7], and C.

Within AgentScape, *agents* are active entities that reside within *locations*, consisting of multiple *hosts*, and *services* are external software systems accessed by agents. Each host runs an instance of the AgentScape middleware. AgentScape uses a Public Key Infrastructure (PKI). Agent owners, locations and hosts have public key pairs. This ensures that locations and hosts can mutually authenticate and set up secure communication channels, using SSL.

Furthermore, every agent has a GUID that is assigned by the agent platform. This GUID is an identifying reference used by the middleware to address an agent and perform operations, such as deliver messages, stop and/or pause, migrate or even kill and/or remove the agent. A GUID is private to the middleware. Externally, agents use *handles*. An agent can have as many handles as it requires. Handles can be published publicly, making access to the agents for others possible. An agent's handles are uniquely linked to its GUID, but the agent's GUID cannot be deduced from its handles, which makes them suitable as *pseudonyms* (see Section 3.1).

8.2 Ajanta

Ajanta [24] is a mobile-agent system based on the Java programming language. Security and robustness have been primary concerns in Ajanta's development. Ajanta platforms can guard themselves against malicious agents. An Ajanta system consists of several *AgentServers* running on hosts. Each agent server creates a confined execution environment for visiting agents and provides them controlled access to local resources. Agents can migrate to other agent servers, communicate with each other, query their environment, etc. The implementation of Ajanta's security architecture is based on *proxies* and Java's security model to restrict, control, and (remotely) monitor running agents. Agents do not have direct references to a host's resources. Instead they

have to go through proxies, which check whether the agent has the authorization to access that resource. Furthermore, agent owners can use encryption to secure parts of the agent's data, thereby guaranteeing the data's confidentiality and integrity.

8.3 SeMoA

Secure Mobile Agents (SeMoA) [40] is an extensible Agent platform, written in Java, designed to counter certain protocol attacks and malicious agents. SeMoA has a RBAC-based access control architecture. SeMoA is also designed to load agents in a secure manner, as each agent is loaded in a separate class loader. This enforces separation between agents, and prevents agents interfering with other code executing within a location. Execution of agents is managed explicitly, with access to features such as threads and resources mediated upon.

8.4 Jade with Jade-S and S-Agent

The Java Agent Development Platform (JADE) [5] is a popular FIPA-compliant agent middleware platform. There are a number of extensions to JADE that provide a security architecture to the system, in particular S-Agent [16] and the JADE-S plugin [34].

S-Agent extends JADE with the intention of providing data confidentiality and addressing the malicious host problem, described in Section 5. S-Agent provides two solutions to the malicious host problem without the need for secure hardware. These solutions are implementations of two different security protocols, the ACCK protocol [2] and the TX protocol [48]. ACCK uses a trusted third party to ensure that a host does not act maliciously. The TX protocol uses a threshold scheme, where two or more agents must agree that an action is authorized before that action will be allowed. This eliminates the need for a trusted third party. However, it can require more protocol interactions, depending on the number of parties required for the threshold to be met.

JADE-S is an extension to JADE providing decentralized access control. It uses the SPKI [11] trust management system. Trust management systems have a number of advantages compared to the traditional identity-based systems created using X.509. Policies and certificates are created and maintained separately from the application. The terminology used within the policies and/or credentials is application defined. They are represented in an application specific fashion, allowing the application designer to decide what characteristics are required. Agents are explicitly granted permissions, and only agents trusted by the location are authorized to execute code at that location.

9 Summary

Security in multi-agent systems is a major concern, particularly in multi-agent systems deployed in a large-scale, distributed, and open environment. Finding a balance between restricting access to resources and allowing enough openness to let the whole system function efficiently and effectively is the challenge.

This chapter has identified threats to the two main stakeholders in an agent system: the agent owner and the platform administrator. The security requirements looked at

included identity management, secure communication, and maintaining confidentiality, integrity, and availability for the stakeholders. These requirements need to be fulfilled for any secure agent system. Each security requirement has been discussed in detail and solutions have been illustrated in the AgentScape agent platform.

Acknowledgments

This work is a result of support provided by the NLnet Foundation (`http://www.nlnet.nl`). The authors wish to thank Benno Overeinder, David Mobach, Thomas Quillinan, Kassidy Clark, Reinier Timmer, and Reinout van Schouwen for their contributions.

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] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, pages 2–11, 2001.

[3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement negotiation specification (WS-AgreementNegotiation) (draft). https://forge.gridforum.org/projects/graap-wg, 2004.

[4] D. Atkins and R. Austein. Threat analysis of the domain name system. IETF RFC 3833, Aug. 2004.

[5] F. Bellifemine, A. Poggi, and G. Rimassa. JADE–A FIPA-compliant agent framework. *Proceedings of PAAM*, 99:97–108, 1999.

[6] 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. RSA, 2002.

[7] R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[8] P. Braun and W. Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[9] S. Clauß and M. Köhntopp. Identity management and its support of multilateral security. *Computer Networks*, 37(2):205–219, 2001.

[10] 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.

[11] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. Request for Comment (RFC) 2693, Internet Engineering Task Force, September 1999.

[12] B. Fonseca. VeriSign issues false Microsoft digital certificates. `http://www.infoworld.com/articles/hn/xml/01/03/22/010322hnmicroversign.html`, March 2001. Infoworld.

[13] S. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[14] L. Gong. *Inside Java™2 Platform Security*. The Java™Series. Addison Wesley, June 1999. ISBN: 0-201-31000-7.

[15] H. Guan, H. Zhang, P. Chen, and Y. Zhou. *Integration and Innovation Orient to E-Society Volume 1*, volume 251 of *IFIP International Federation for Information Processing*, chapter Mobile Agents Integrity Research, pages 194–201. Springer, 2008.

[16] V. Gunupudi and S. R. Tate. SAgent: A Security Framework for JADE. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS06)*, 2006.

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

[18] 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.

[19] 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.

[20] IIDS. AgentScape Agent Middleware. `http://www.agentscape.org`.

[21] 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.

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

[23] N. M. Karnik and A. R. Tripathi. Agent Server Architecture for the Ajanta Mobile-Agent System. In *Proceedings of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 66–73, July 1998.

[24] N. M. Karnik and A. R. Tripathi. Design Issues in Mobile Agent Programming Systems. *IEEE Concurrency*, 6(6):52–61, July–September 1998.

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

[26] S. Kent and R. Atkinson. Security architecture for the internet protocol. IETF RFC 2401, Nov. 1998.

[27] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.

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

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

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

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

[32] Netscape Inc. Secure sockets layer website. `http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt`.

[33] 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.

[34] A. Poggi, M. Tomaiuolo, and G. Vitaglione. Security and trust in agent-oriented middleware. In R. Meersman and Z. Tari, editors, *OTM Workshops 2003*, number 2889 in LNCS, pages 989–1003. Springer-Verlag, 2003.

[35] T. B. Quillinan, M. Warnier, M. A. Oey, R. J. Timmer, and F. M. T. Brazier. Enforcing security in the agentscape middleware. In *Proceedings of the 1st International Workshop on Middleware Security (MidSec)*. ACM, December 2008.

[36] 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.

[37] O. Rana, M. Warnier, T. B. Quillinan, and F. M. T. Brazier. Monitoring and reputation mechanisms for service level agreements. In *Proceedings of the 5th International Workshop on Grid Economics and Business Models (GenCon)*, Las Palmas, Gran Canaria, Spain., August 2008. Springer Verlag.

[38] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[39] 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.

[40] 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.

[41] V. Roth and M. Jalali-Sohi. Concepts and architecture of a security-centric mobile agent server. In *Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems*, pages 435–442, Dallas, Texas, U.S.A., March 2001.

[42] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer-Verlag, Berlin, Germany, 2001.

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

[44] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

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

[46] W. Stallings. *Cryptography and network security: principles and practice*. Prentice Hall, 2006.

[47] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Feb. 2003.

[48] S. R. Tate and K. Xu. Mobile agent security through multi-agent cryptographic protocols. In *Proceedings of the 4th International Conference on Internet Computing*, pages 462–468, Las Vegas, NV., 2003.

[49] Trusted Computing Group. TPM main specification. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, July 2007.

[50] 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.

[51] M. Warnier and F. M. T. Brazier. Organized anonymous agents. In *Proceedings of The Third International Symposium on Information Assurance and Security (IAS'07)*. IEEE, August 2007.

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

[53] 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*, 3(4), 2009.

[54] X. Zhang, S. Oh, and R. Sandhu. PDBM: A flexible delegation model in RBAC. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, Como, Italy, 2003.