Enforcing Security in the AgentScape Middleware

Thomas B. Quillinan, Martijn Warnier, Michel Oey,
Reinier Timmer and Frances Brazier
Department of Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
{thomasq, warnier, michel, rjtimmer, frances}@few.vu.nl

ABSTRACT

Multi Agent Systems (MAS) provide a useful paradigm for accessing distributed resources in an autonomic and self-directed manner. Resources, such as web services, are increasingly becoming available in large distributed environments. Currently, numerous multi agent systems are available. However, for the multi agent paradigm to become a genuine mainstream success certain key features need to be addressed: the foremost being security. While security has been a focus of the MAS community, configuring and managing such multi agent systems typically remains non-trivial. Well defined and easily configurable security policies address this issue. A security architecture that is both flexible and featureful is prerequisite for a MAS.

A novel security policy enforcement system for multi agent middleware systems is introduced. The system facilitates a set of good default configurations but also allows extensive scope for users to develop customised policies to suit their individual needs. An agent middleware, AgentScape, is used to illustrate the system.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

Keywords

Multi Agent Systems, Security Policies, AgentScape

1. INTRODUCTION

Distributed multi agent systems provide a powerful paradigm for building large scale distributed middleware systems [11, 12]. Features such as mobility, autonomy and adaptivity make these systems attractive, particularly in highly dynamic environments such as e-government, e-health and e-commerce applications. However, mobility and autonomy also provide new challenges, especially if security is of key concern.

The security challenges in multi-agent systems are numerous: agent middleware platforms need to be protected from malicious agents. Such agents might steal sensitive information or use resources without paying for them. Agents also need to be protected from malicious hosts [18, 3, 16]. This is especially challenging as in essence the environment is not under the control of the agent owner and can, thus, not be trusted. Untrusted hosts can view and alter the state of the agent, or even delete the agent altogether.

These challenges have been recognised in the multi agent systems community and have been previously addressed in two areas. Firstly, they have been addressed through the development of agent systems that are especially tailored for security, such as SeMoA [17] and Mansion [22]. Secondly, dedicated extensions to general purpose agent middleware systems have been created, such as the SAgent framework [7] for JADE [2], and in mobile agent application systems, such as Ajanta [10]. While these systems provide the ability to enforce security policies for agents, defining, configuring and auditing these policies is not straightforward. This paper describes a security architecture that addresses some of the issues found in existing solutions.

Security policies allow users of agent systems to easily 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, a good default policy is one that will not prevent users from performing vital tasks, but will protect against some of the most common security issues. Another example is a 'high security' policy that should be used in security critical environments. Such policies would be much more restrictive. The multi agent middleware system, AgentScape [9], is used to illustrate the security policy framework.

The remainder of this paper is organised as follows: the following section gives some general background on the AgentScape middleware. The security threats that are common to all agent middlewares are described in Section 3. Section 4 discusses the security architecture. The paper concludes with a discussion.

2. BACKGROUND

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. The multi-

level AgentScape middleware infrastructure [14] has been designed to be extensible.

2.1 Agents

Intelligent software agents are mobile applications that are launched by a user or another agent and obtain rights and permissions to use resources and access data. Agents have the ability to be created; to migrate between hosts; to communicate with other agents and their owner, and to access resources and services.

Example 1
A simple example of an agent application is an application that seeks specific products for the agent's owner on the Internet. The agent migrates to different websites that the owner uses and determines the best price for the specific product. Then the agent relates this price and the details to the agent's owner. △

2.2 AgentScape

Within AgentScape, *agents* are active entities that reside within *locations*, and *services* are third-party 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. This communication is exclusively managed by the middleware. Agents can also migrate from one location to another. Locations are made up of one or more hosts. AgentScape

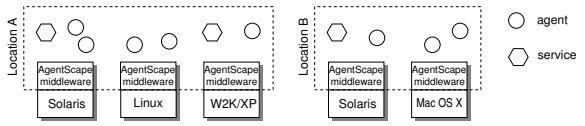

Figure 1: Conceptual model showing two AgentScape locations, each consisting of a number of hosts.

has been designed as a multi-layered architecture with (1) a small *middleware kernel*, called the AgentScape Operating System (AOS) kernel [21], that implements basic mechanisms and (2) high-level *middleware services* that implement agent platform specific functionality and policies (see Figure 2). The current set of middleware services includes agent servers, host managers, location managers, a lookup service and a web service gateway. The policies and mechanisms of the location and host manager infrastructure are based on negotiation and service level agreements [13]. Agent servers

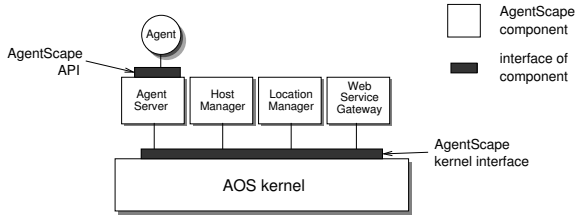

Figure 2: An AgentScape host operating as a location manager.

provide agent access to the AgentScape middleware Multiple code base support in AgentScape is managed through the provision of multiple agent servers, at least one per code base. All interactions between agents and locations/hosts are managed by Agent servers.

3. THREATS IN AGENT MIDDLEWARE

Securing agent middleware systems entails first identifying the security threats that these systems face. There are a number of basic threats with agents executing on remote hosts. These threats can be placed into two categories: (*i*) **Malicious Hosts** and (*ii*) **Malicious Agents**.

3.1 Malicious Hosts

Malicious hosts intentionally attack agents that are executing on the host, or possibly on other hosts. There are a number of attack types, ranging from denial of service attacks to data stealing or injection attacks. Hosts are assumed to have complete control over their own systems. The most straightforward solution is for agents to migrate to hosts that they 'trust' not to be malicious. Another approach uses trusted hardware to verify the actions of the host are not malicious. Finally, security protocols, either using a trusted third party [1], or a threshold scheme [20, 23] can be used to address the malicious host problem. This problem is not addressed further in this paper.

3.2 Malicious Agents

Malicious agents attempt to gain access to resources on a host that they are not authorised to use. Such access includes attempts to access private data or additional computational resources that have not been negotiated.

Securing hosts from malicious agents entails monitoring every action that the agent attempts on the host. Whenever an agent makes a call to the middleware API, it is first mediated upon 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 is not willing to allow an agent to migrate anywhere except directly back to the owner's host, or, perhaps, to another host that has been properly authenticated to the current host. This is known as the *home based approach* [22]. However, the future migration to 'untrusted' hosts cannot be guaranteed in this manner.

In Java [6], 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. Sandboxing typically encompasses network access as well as accessing the local filesystem. Applications that need access to these restricted functions have to use other approaches. The sandboxing technique is not exclusive to Java–other platforms [21] also use sandboxing.

3.3 Related Work

Several multi-agent systems have identified security as a concern, and have developed strategies towards addressing security threats. The Java Agent Development Platform (JADE) [2] 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 [7] and the JADE-S plugin [15].

Secure Mobile Agents (SeMoA) [17] 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. The Cougaar Multi Agent System [8] is designed to address scalability, reliability and survivability. Cougaar does not have an explicit access control mechanism, but does support role based authentication.

Each of these systems provides centralised access control. In contrast, in the following section, a security architecture is presented that addresses the security requirements of a mobile execution environment in a distributed manner. This architecture has a particular emphasis on design, configuration and management of security policies for an agent middleware.

4. SECURITY IN AGENTSCAPE

This section outlines a comprehensive solution to the malicious agent threat in an agent middleware system, AgentScape. Agent communication is described in Section 4.2. Section 4.3 then outlines how creation of agents and migration between hosts and resource and service access is managed.

In AgentScape, a Public Key Infrastructure (PKI) is installed. 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. The agent environment also provides that every agent has a globally unique identifier (GUID). This GUID is an identifying reference used by the middleware to address the agent and perform operations on it, such as deliver messages, stop and/or pause, migrate or even kill and/or remove the agent.

The GUID is private to the middleware. Thus, the GUID cannot be used to publicly address the agents and to allow agents to send messages to each other. This is achieved by introduction of additional references for agents called handles. An agent can have as many handles as it requires. Handles are generated by computing the secure hash (for example, SHA-256) of the GUID concatenated with a counter. Handles can be published publicly, making access to the agents possible. For example, by publishing the handle in a lookup service or by communicating the handle directly to other agents.

4.1 Identity Management

Identity management in the context of agent based systems requires that agents are somehow 'bound' to their owner. An agent consists of meta-data, (executable) code and data (that the agent has 'found' on a particular host). The meta-data of the agent contains, at least the following: the GUID of the agent, the name of the agent owner and a signed (by the owner) hash of the agent code. The signature ensures that agent and owner are bound to each other.

When an agent is injected, the agent platform checks if the agent code is indeed signed. If the verification is successful the agent obtains a GUID and an *owner 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.

4.2 Communication Security

AgentScape currently supports SSL-based communication between hosts and/or 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 an organised manner.

Integrity support is also currently provided by AgentScape. This is implemented through the use of cryptographic primitives to create a digitally signed checksum of data transmitted between AgentScape hosts/locations.

4.3 Access Control in AgentScape

Once the basic security features, such as authentication and identity management mechanisms, are in place, the next requirement is an authorisation mechanism. There are a number of principals involved in AgentScape, such as location and world administrators, resources and their administrators, and agents and their owners. There are also a number of basic actions that agents use to achieve their goals. This is a structure that can be readily managed using a Role Based Access Control (RBAC) [19, 24] mechanism.

RBAC is an access control architecture that models roles, users and permissions. RBAC is designed to reflect real-world relations between users and permissions. Roles define the logical tasks that users can perform. Users become members of roles and roles are assigned permissions.

Example 2
Defining roles, users and permissions can be straightforward. In this simple example, 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. In this case, each role has a number of permissions. These permissions reflect the capabilities of the role.

Role	Permission
BasicAgent	Migrate, Execute
TrustedAgent	Migrate, Execute, AccessRes
AgentOwner	Inject, GetResult
ResAdmin	AccessRes, ChangePerms, GetLogs

Table 1: RBAC Example Role Permission Table

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

Table 2: RBAC Example Role User Table

Extending this example, Table 2 represents a set of users who have been assigned to the roles shown in Table 1. These

users are shown as textual names, but are represented by a unique identifier in an AgentScape RBAC policy. △

Agent owners form the base of the trust mechanism. Owners are ultimately responsible for the actions of their agents. Therefore, by default, agents hold the permissions granted to their owners. As access to important resources must be specifically controlled, agent owners that wish to access such resources must be explicitly specified in the RBAC policy. Developing an RBAC system in AgentScape entails determining the permissions that will be supported. 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. However, as the enforcement mechanism is built into the AgentScape Security Manager (see Figure 3), security relevant actions are defined by the middleware, and the permissions reflect these actions. Managing these policies is the responsibility of the administrator of each location and/or host.

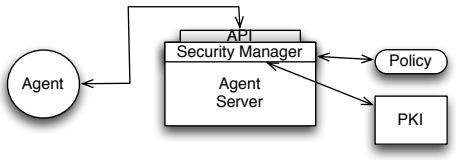

Figure 3: AgentScape Security Manager.

Whenever an agent attempts to perform a security relevant action, the Security Manager checks that the agent is authorised to perform the action. This is a two step process. First, the Security Manager determines the GUID of the agent and determines the role, or roles, that the GUID is a member of. Second, the Security Manager determines if any of these roles is authorised to perform the requested action.

A selection of the basic security relevant actions is shown in Table 3. This is not the complete set of actions, but are the actions that are most commonly used when writing security policies for locations.

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

The security relevant actions shown in Table 3 represent the basic set of actions that are found in AgentScape and reflect the abilities of agents as outlined in Section 2. These actions are extended with *parameters*. These 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 specific parameter, '*', is supported to allow *all* types of an action to be permitted by a role. This is utilised 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 permission to access a resource is not explicitly granted, access to it is denied.

Each location and host can have its own RBAC policy. Determining the agent owners and other hosts/locations to trust is a task for each AgentScape host and location administrator. A location administrator defines a set of roles and assigns trusted principals to these roles and assigns permissions to those roles for that location. Hosts can similarly have different restrictions to the locations. Both policies are enforced together; actions are only permitted when both policies agree.

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

Example 3
Consider the Role/Permission table shown in Table 4. In this table, normal (BasicAgent) agents are allowed to execute and access CPU and Memory resources. Trusted agents, that is, agents who are in the role TrustedAgent, are authorised 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

Parameterisation of permissions allows more fine-grained access to system resources to be defined. This can be utilised to define policies such as defining the locations that agents may migrate to from this location.

4.4 Accessing AgentScape Locations

When a principal wishes to inject an agent into an AgentScape location, the principal first contacts the location and they perform 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 authorised 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 and

assigned to, at most, the same roles as the owner. Owner roles are defined by each location individually. Typically, a default role is used for unknown agents and owners. This allows the efficient lookup of permissions, without requiring a separate lookup when performing security mediations. In order to manage the growth of the *Role-User* table, when an agent successfully migrates to another location, or completes its task, the entry is removed from the table. If owners are removed from a role, any agent belonging to that owner are also removed.

The owner generates a public/private keypair and then creates and signs an X.509 certificate [4] for the agent, including the GUID, and specifies any access control restrictions that the owner may wish to place on the agent. These restrictions are codified in the form of additional attributes, as a X.509 attribute certificate [5]. For example, the owner may wish to restrict the agent from accessing certain databases that the owner may have permission to use. These attributes are used during negotiation and migration to determine the roles to place the agents. The private key is held by the agent owner and they use it to sign a hash of the agent code to allow agents to authenticate themselves with remote locations. This signed hash is stored in the meta-data of the agent.

Example 4
Example 1 outlines an agent based solution to purchasing items on the Internet. In this system, the owner is authorised to purchase items at a large number of websites – namely the websites with whom the owner has an account. However, when an owner dispatches an agent for a specific item, they may not wish the agent to access certain websites. Therefore, while the owner is authorised to access these websites, the agent must not migrate to these sites. △

Migration of an agent from one location to another uses a similar protocol to injection. First, the locations must perform mutual authentication. The agent will then authenticate itself with the target location. This is achieved by sending the certificate of the owner of the agent to the target location, along with a hash of the agent GUID (the handle) and a hash of the agent code. These are all signed by the owner using the agent's private key created during injection. The target location can then verify the identity of the owner of the agent. The agent then negotiates for any resources on the target location that it will require. If this negotiation is successful, the target location's Role-User table is updated with the GUID of the agent. The agent then migrates to the target location. Whenever an agent attempts to access a resource on a location, the agent middleware intercepts that request and ensures that the agent is authorised to access the resource and that access to the resource has been negotiated.

4.5 AgentScape 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 define a set of 'good default' policies.

Good default policies range from simple policies, used for systems where agents are executing in a well known environment, to more restrictive policies, where agents are executing in a more hostile environment.

While a simple system is common in small, well-understood 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.

Example 5
In a hostile environment, locations are controlled by entities that are not always known by every principal. Agents are authenticated to their initial location as before, but the authorisation 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.

In a hostile environment migration is only authorised between the original 'home' host–the host where injection of the agent took place–and remote hosts. Therefore, migration from one remote host to another forces the 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. △

Within a hostile environment, agents should not only be constrained to a minimal set of actions that each location and host provides, but also the actions that the owner of the agent allows the agent to perform on their behalf. These actions include the ability to negotiate, migrate, inject and access resources.

The security architecture for AgentScape outlined in this section 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 authorised by the locations and hosts, are not desirable to the owner.

5. CONCLUSIONS AND FUTURE WORK

A security policy management system for multi agent systems was introduced. This system is based on the analysis of security threats that apply to all agent middlewares. The system facilitates a set of good default configurations, but also allows extensive scope for users to develop customised policies to suit their individual needs. The agent middleware AgentScape has been extended to support these security features.

While several existing agent middlewares provide a security architecture, they do not encompass the flexibility and control that the solution outlined in this paper provides. For example, SeMoA and Cougaar approach security from the premise of reliability. While SeMoA does provide an access

control mechanism, it is a centralised system and does not allow individual hosts and locations to define their own policies. JADE-S allows decentralised access control policies. However, these policies are defined in an ad hoc manner, using trust management credentials. Trust management provides a decentralised approach to access control. Typically, credentials are held by the principals that are authorised by the credential and only produced when needed to prove access rights. Therefore, it is difficult to determine a priori the permissions associated with each principal. While trust management credentials are also utilised to some extent by AgentScape, these are used to inform the RBAC system and are more explicitly defined. In the solution presented in this paper permissions are stored in a *distributed* manner, yet the permissions are held by the resource administrators, not the users. The trust management credentials serve to limit existing permissions. Therefore, determining the permissions associated with principals is much more straightforward.

Acknowledgments

This work is a result of support provided by the NLnet Foundation (`http://www.nlnet.nl`), the ALIVE project (FP7-IST-215890), the ACCESS project funded by the NWO Token program and the Dutch Ministry of Economic Affairs, grant no. BSIK03024 in the Interactive Collaborative Information Systems (ICIS) project (`http://www.icis.decis.nl/`). The authors would also like to thank the anonymous reviewers for their help.

6. REFERENCES

[1] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. *Proceedings of the 2001 Symposium of Security and Privacy*, 00:0002, 2001.

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

[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. RSA, 2002.

[4] CCITT Draft Recomendation. *The Directory Authentication Framework, Version 7*, Nov. 1987.

[5] S. Farrell and R. Housley. An internet attibute certificate profile for authorization. Request for Comment (RFC) 3281, IETF, April 2002.

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

[7] V. Gunupudi and S. R. Tate. Sagent: A security framework for jade. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS06).* ACM, 2006.

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

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

[10] N. M. Karnik and A. R. Tripathi. A security architecture for mobile agents in ajanta. *ICDCS*, 00:402, 2000.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] V. Roth. Programming Satan's agents. In *In Proceedings of the 1st International Workshop on Secure Mobile Multi-Agent Systems*, 2001.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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*, 2008.

[24] 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.