Distributed Authorization Framework for Mobile Agents*

G. Navarro[1], J. A. Ortega-Ruiz[2], J. Ametller[1], and S. Robles[1]

[1] Dept. of Information and Communications Engineering
Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain.
[2] Institute for Space Studies of Catalonia (IEEC),
80034 Barcelona, Spain.
gnavarro@ccd.uab.es,jao@gnu.org,{jametller,srobles}@ccd.uab.es

Abstract. Mobile agent systems provide new perspectives for distributed e-commerce applications. These applications may present specific restrictions, making mobile agent systems a feasible solution. Even so, mobile agents present some security related problems. An important one is resource access control. The ability for mobile agents to provide a simple, scalable, flexible, and secure access control system is a key point for the widespread adoption of mobile agents. In this paper we propose a mechanism to safely assign roles to mobile agents and an access control method based on *Role-based Access Control* (RBAC). The access control method provides a simple, lightweight and distributed model for mobile agent applications. It is proposed as an extension of the MARISM-A (*An Architecture for Mobile Agents with Recursive Itineraries and Secure Migration*) project, a secure mobile agent platform.

Keywords: Access Control, Mobile Agents, Trust management, Security.

1 Introduction

During the last years, mobile agent technologies have witnessed an steady, if not fast, increase in popularity. Probably, the main hurdle to a wider adoption are the security issues that mobility brings to the picture [1]. Among them, an outstanding one is resource access control. Traditional access control methods rely on the use of centralized solutions based on the authentication of global identities (for example, via X.509 certificates). These methods allow to explicitly limit access to a given resource through attribute certificates or Access Control Lists, and rely on a centralized control via a single authority. Despite providing effective means of protection, these techniques suffer from serious drawbacks; in particular, they give raise to closed and hardly scalable systems. Practical mobile agent systems demand lightweight, flexible and scalable solutions for access control, in order to cope with the highly heterogeneous nature of their clients. In the same vein, solutions depending on centralized entities (such as traditional Certification Authorities) should be avoided.

* This work has been partially funded by the Spanish Ministry of Science and Technology (MCYT) though the project TIC2003-02041.

There are alternatives based on *trust management*, which allow to assign authorizations (permissions or credentials) to concrete entities, as well as trust delegation among entities. Well-known implementations of these infrastructures are the *Simple Public Key Infrastructure/Simple Distributed Secure Infrastructure* (SPKI/SDSI)[2] and KeyNote[3], and several security frameworks are based upon it. Recent developments in this area, in an attempt to further ease access control management, have brought into the picture Role-based Access Control (RBAC) [4]. In these schemes, privileges of principals requesting access to a resource are determined by their membership to predefined roles. The use of RBAC greatly simplifies access control management and is specially suited to mobile agents scenarios, where agents privileges are subsumed in a possibly more general RBAC system.

This article presents an access control framework for mobile agents. We combine RBAC and *trust management* to provide a flexible, lightweight methodology for access control in such scenarios. In our approach, mobile agents do not carry any explicit information regarding resources access, avoiding the privacy concerns associated with sensitive data embedded in mobile code. In addition, our framework allows dynamical binding of authorizations to agents, providing thus great flexibility when it comes to define resource access control policies based on it. We have implemented our proposed scheme on MARISM-A, a JADE-based agent platform. Our framework is called DAFMA (*Distributed Authorization Framework for Mobile Agents*).

In Section 2 we introduce the main motivations behind DAFMA. Sections 3 and 4 introduce the main concepts and the base to our systems. We describe DAFMA and its components in Section 5. Finally, Section 6 summarizes our conclusions.

2 Motivations

DAFMA is intended to provide a suitable access control framework for distributed environments, and more specifically, mobile agent systems. There are not much proposals for access control systems for multiagent systems supporting agent mobility. Most of the security works in mobile agents deal with protecting communications, itineraries, and so on, but less of them deal with the protection of resources and the management of access control rights. Usually, proposed system rely on ad-hoc and centralized solutions. Just as an example, in [5] the authors propose a solution based on credentials, which are carried by the agent as authorizations. This credentials are combined with the Java protection domains to provide resource access control. The solution seem interesting but it relies too much on the platform itself, and specially Java. The system cannot be applied to other platforms and it is not distributed, since it requires a centralized policy.

In [6], the author presents a framework for building a secure infrastructure for agent systems based on SPKI/SDSI. Related to delegation it considers: chain-ruled delegation [7], threshold delegation, and conditional delegation. This approach does not consider mobile agent systems or even heterogeneous multiagent systems. The fact that agents are able to sign certificates with private keys, makes the whole system too much dependent in the trust relation between the agent and the execution platform.

JADE (`http://jade.tilab.it`) also provides a security add-on[8], which is also based on the Java security architecture and does not support agent mobility. FIPA (Foundation for Intelligent Physical Agents: `http://fipa.org`) also began to consider agent security through a technical committee, but the work has not been finished, and there is no FIPA recommendation at the moment on security related topics.

Other mobile agent platforms, such as NOMADS [9], provide access control based on Java-specific solutions and eventually used KAoS to provide high level policy languages and tools. KAoS [10] is a high level policy language currently focused on semantic web services. Despite its complexity, KAoS does not explicitly support mobile agents.

Our proposal provides a simple distributed architecture for the management of authorizations in mobile agent environments. Its ideas are, indeed, applicable to generic mobile code systems. A key factor is simplicity. We tried to simplify as much as possible the components and interactions, so the system becomes more easily implementable and scalable. DAFMA does not rely in implementation-dependent solutions such as the security architecture of Java, what makes it suitable for heterogeneous systems. And at the same time provides enough flexibility to accommodate a wide range of applications.

3 Distributed Naming Scheme

In DAFMA, we use a distributed naming scheme strongly influenced by SPKI/SDSI, which is used to create and manage roles and groups. We denote each entity (agent, platform, human user, etc.) as a *principal*, which has a pair of cryptographic keys (this is not exactly the case for mobile agents, see Section 5.5). The public key acts as a global identifier of the principal. In order to make it more manageable one can use the *hash* of the public key as an abbreviation for the public key. Each principal generates by itself the keys and is responsible of its own identity, so there is no need for a centralized CA, although it can be used if it is needed.

A principal can define local names of entities under its own responsibility. In order to do that, an entity has an associated local name space called *name container*, which can be made public to the rest of the world. The name container has entries of the type: `(<principal>,<local-name>)`. The *principal* corresponds to the principal for whom, the local name is being defined, and *local-name* is an arbitrary string. The *principal* can be specified as a public key or as a *fully qualified name* (see below).

For example, consider a principal with public key PK_0, which creates an agent with public key PK_1 and wants to name it *my-agent*. The name container of the entity will have an entry of the form: `(`PK_1`, my-agent)`. Now on, the agent PK_1 can be referenced by the name *my-agent* in the local name space of PK_0. One of the most interesting issues is that a third party can make a reference to a name defined in other name containers through a *fully qualified name*. A name container is identified by the public key of the owner, so the fully qualified name PK_0 *my-agent* makes reference to the name *my-agent* defined in the name container of PK_0. It is important to note that given the properties of cryptographic keys, it is commonly assumed the uniqueness of the public key, so fully qualified names are globally unique.

These names, make it very easy for a principal to create groups or roles. For instances, a user PK_{admin} can create a group *employees* with members PK_a, PK_b and

the agent PK_1, by adding the following entries in its name container:

$$(PK_1, \textit{employee})$$
$$(PK_2, \textit{employee})$$
$$(PK_0\ \textit{my-agent}, \textit{employee})$$

In order to support role hierarchies, we use group or role inclusion. That is, to declare a role as member of another role. For example consider the role *security-staff*, which is a super-role of *employees*. That is, members of *security-staff* also have the permissions or authorizations associated to *employees*. This could be expressed as:

$$(PK_{admin}\ \textit{security-staff}, \textit{employee})$$

In order to make public an entry of a name container, we need to ensure that it cannot be forged. A *name certificate* is a name container entry signed by the owner of the container and including a validity specification. We denote such a certificate as:

$$\{(PK_{admin}\ \textit{security-staff}, \textit{employee})\}_{PK_{admin}^{-1}}$$

where PK_{admin}^{-1} denotes the private key corresponding to the public key PK_{admin}, which digitally sings the certificate determining the issuer of the certificate or the owner of the name container where the name is defined. We do not show the validity specification for clarity reasons.

4 Authorization Certificates

The base to our proposal are *authorizations*. Authorizations are expressed through SPKI/SDSI *authorization certificates*. An authorization certificate has the following fields:

- Issuer (I): *principal* granting the authorization.
- Subject (S): *principal* receiving the authorization.
- Authorization tag (tag): specific authorization granted by the certificate.
- Delegation bit (p): if it is active, the subject may forward delegate the authorization received.
- Validity specification (V): validity of the certificate (time range and on-line tests).

The certificate is signed by the issuer. The on-line tests from the validity specification field, provide the possibility of checking, at verification time, the validity or revocation state of the certificate. For clarity reasons we will denote an authorization certificate as:

$$\{(K_s,\ p, +)\}_{K_i^{-1}}$$

Where the subject K_s received the authorization p from the issuer K_i. Again, K_i^{-1} denotes the private key associated with K_i, which signs the certificate. The plus symbol

denotes that the delegation bit is active otherwise a single dash is used. Note that we represent an authorization certificate as a 3-tuple, while a name certificate is a 2-tuple.

Delegation introduces a lot of flexibility to the system. One principal can delegate authorizations to others without the intervention of a third party, authority, or without needing to modify any policy of the system. Despite its benefits, delegation comes with a cost. Suppose we have a bunch of name and authorization certificates and want to find if some principal is authorized to do something. Finding a certificate chain from a source of authority to the principal, that is, finding an *authorization proof* may become very complex, as it is in most sophisticated policy languages. In our case, we take as a base the SPKI/SDSI certificate chain discovery algorithm, which has a polynomial complexity and is easily implementable. It is based on simple reduction rules, such as:

$$\frac{\{(K_B, a, +)\}_{K_A^{-1}};\ b \preceq a}{\{(K_B, b, +)\}_{K_A^{-1}}}; \quad \frac{\{(K_B, a, +)\}_{K_A^{-1}};\ \{(K_C, b, +)\}_{K_B^{-1}}}{\{(K_C, a \wedge b, +)\}_{K_A^{-1}}}$$

Where '$\preceq$' denotes a partial order on authorizations (such as in lattice-based permissions), and '$\wedge$' denotes the intersection of authorizations. Previous to the reduction rules, there is a *name resolution* phase, which resolves names to public keys making use of name certificates. The reader may note that there is a little informality in this notation, for example the result of the first rule does not mean that there is a certificate signed by K_A granting permission b to K_B, but that this certificate (or its consequences) could be deduced from the previous statements.

5 DAFMA

The DAFMA system is implemented on top of MARISM-A, a secure mobile agent platform implemented in Java [11]. It provides extensions on top of the JADE system, a Java multiagent platform, which follows the standards proposed by FIPA. Mobility is achieved by the MARISM-A mobility component which is integrated into JADE. On top of JADE there are the main MARISM-A components such as the authorization framework presented in this paper, and other MARISM-A services such as: cryptographic service, directory service, service discovery, etc.

One of the first problems we found when planning the authorization model, is if a mobile agent should have a cryptographic key pair and be considered as a principal. A mobile agent cannot trivially store a private key, and cannot perform cryptographic operations such as digital signatures. There are some proposals to store sensitive information (private keys) in mobile agents [12]. But the problem arises when the mobile agent uses the private key to compute a cryptographic operation. The agency where the agent is in execution will be able to see the private key or at least, reproduce the operation. As a result we consider that a mobile agent should not have a private key.

Our solution is to establish the role membership of a mobile agent directly, in a way that the agent does not need to carry authorization related information, making the agent more simple and lightweight. This issue will be discussed in Section 5.5.

DAFMA is made up of four independent components, which interact to perform all the required functionality. This components are implemented as static agents. The interaction between this components is done using the FIPA ACL (*Agent Communication*

Language) and FIPA ontologies. Figure 1 shows the components, which are described in the next sections.

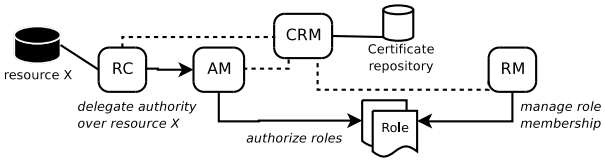

Fig. 1. DAFMA Architecture

5.1 Authorization Manager (AM)

The *Authorization Manager* (AM) manages the delegation of authorizations, issuing authorization certificates. It follows a *local authorization policy*, and its main responsibility is to delegate authorizations to specific roles following its local policy. It may also provide the ability to delegate authorizations to other AM in order to distribute the authorization management. Since the authorization policy is local to the AM agent, it does not need to follow any specification and its format can be implementation-dependent.

We propose an authorization policy, expressed as a sort of ACL (*Access Control List*) entry in S-expression format, similar to the SPKI/SDSI ACLs. An ACL entry is an authorization certificate without the issuer and it does not need to be signed because it is local to the AM and is not going to be transmitted. It has the following fields:

- Subject: the principal, who can request the AM to issue a certificate. Note that it does not need to be the principal receiving the authorization. This allows to separate the management of the authorization certificates (who makes the request) from the authorization itself (who receives the authorization).
- Authorization tag: the authorization tag matches the *authorization-request* issued by the principal asking for the certificate.
- Delegation bit: it indicates if the *Subject* can delegate the right to request the certificate.
- Validity: indicates the validity conditions (time range, ...) under which the certificate can be requested.

For example, suppose principal K_a wants to request to the authorization manager AM the certificate $\{(K_b, \mathit{dummy}, -)\}_{AM^{-1}}$, K_a will issue the following *authorization-request* S-expression:

```
(authorization-request
  (issuer AM)
  (subject K_b )
  (tag dummy)
```

```
  (validity V1 ))
```

Then AM could have the following SPKI ACL:

```
(acl
  ...
  (entry
    (subject Ka )
    (tag (authorization-request ...))
    (validity V2 )))
```

Note that the subject of the ACL entry and the certificate are not the same (a principal may request a certificate issued to another principal). The validity specifications V_1, and V_2 may also differ. V_1 indicates the validity for the request of the certificate and V_2 indicates the validity for the authorization granted by the certificate.

5.2 Role Manager (RM)

The *Role Manager* (RM) manages the roles (mainly role membership) by issuing name certificates following a *local role policy*. It can also assign a role to another role defined by itself or by another RM. Thus allowing the definition of role hierarchies or the delegation of role membership management. Section 5.5 details how roles are assigned to mobile agents.

Each RM has a local role policy, which determines what roles does it manage. It also includes rules to determine if a given principal requesting a role membership has to be granted or not. This is done by using a *membership-request*, which is equivalent to an *authorization-request*, and specifies the name certificate requested. If we choose to describe the role policy as a S-expression ACL, it is analogous to an authorization policy. The policy will determine which principal has authority to request memberships for a given role.

Now the subject of the SPKI ACL entry is a principal or another role, and the authorization tag determines the role that the subject can have.

An agent or principal, may belong to several roles. This is possible and adds lot of flexibility to the system. Since the system only uses positive authorizations or permissions, it is not possible to find conflicting authorizations for the same agent.

5.3 Resource Controller (RC)

The *Resource Controller* (RC) main task is to control the access to a resource. It holds the master SPKI key to access the resource, delegates authorizations to AMs, and verifies that an agent requesting access to the resource has a proper authorization.

It delegates authorizations to one or more AM following a local authorization policy. Note that this policy is quite simple because the main authorization management is performed by the AM. For example a RC controlling resource R_1 may delegate full control to AM_1 and read authorizations to AM_2.

5.4 Certificate Repository Manager (CRM)

The *Certificate Repository Manager* (CRM) implements and manages a certificate repository. For example, one agency may have one CRM to collect all the certificates issued by agents inside the agency. The CRM provides the repository and all the services needed to query, store or retrieve the certificates in the repository. It also provides a certificate chain discovery service. A principal can make a query to the CRM to find a specific certificate chain. This way we solve the problems derived from certificate distribution and leave the task to perform chain discoveries to the CRM and not to the other principals. It decreases communication traffic, certificates do not need to travel from one principal to another, and reduces the task that generic principals need to perform.

It is important to note that a distributed version of this module presents some problems mainly regarding with performance. Existing solutions [13] to provide distributed certificate chain discovery introduce limitations and complexity to the system. Another feasible approach is to use other mechanisms like a distributed LDAP directory, or use discovery services from the underlying agent platform [14].

5.5 Establishing Mobile Agents Role Membership

Since mobile agents cannot have private keys, we can not delegate authorizations to the mobile agent or make it member of a role. Our approach is to set, as member of the role, a hash of the agent's code. A principal can be identified by a a public key or a hash of a public key. So a hash may be seen as a principal, subject of a certificate. This is even supported by certificate frameworks, where a hash of an object can be considered as a principal, such as SPKI/SDSI, or even X.509.

In order to establish the role membership of a mobile agent we consider two different approaches. To show them we use a simple example where a user K_u is member of the role $physician$ defined by a Role Manager RM of the agency i, which has a resource controlled by a RC. The role $physician$ allows its members to access the given resource in agency i. The user has a mobile agent with the code m_i to be executed in platform i. The goal is to set the hash of m_i as member of the role $physician$.

User-managed role The RM makes member of the role $physician$ a role (or group) defined by the user K_u, say $agent$. Then, the user K_u can make member of its role $agent$ any hash of agent's code.

$$\{(K_u \textit{ agent, physician})\}_{RM^{-1}}$$
$$\{(hash(m_i),\ agent)\}_{K_u^{-1}}$$

RM-managed role The RM makes member of the role $physician$ the user K_u. Then the users sends a request to the RM to set the agent code's hash as member of the role.

$$\{(K_u,\ \textit{physician})\}_{RM^{-1}}$$
$$\{(hash(m_i),\ \textit{physician})\}_{RM^{-1}}$$

The *user-managed role*, is quite straightforward, gives the user full flexibility to manage the role $K_u\ agent$, and does not require any special mechanism or protocol. The user may add to the role any agent (or even user) she wants. The main problem with this approach is related to the accountability of the system. In e-commerce application there will be different degrees of trust between users. For example an hospital may trust an internal physician to manage its role $agent$, but a client from an external research center may not be so trusted. In the first case we will use *user-managed role*, while in the second one we will use *RM-managed roles*.

This last approach requires an additional protocol, which is implemented as an Agent Communication Language Interaction Protocol. It can be described in three steps:

1. The client sends an agent role assignment request (*ara-request*) to the role manager RM. Including m_i and the role she wants to obtain for the agent.
2. The RM requests the CRM to verify the client's role, and checks if he is member of the role requested in the *ara-request*.
3. If it succeeds, the RM computes the hash of m_i and issues a name certificate such as:

$$\{(hash(m_i),\ \textit{physician})\}_{RM^{-1}}$$

In the last step, the RM stores the code m_i, which may be used for further security audits. Note that this approach does not allow the user to manage the role and extend it to other agents. The user needs to send a request for each agent.

Later, the agent arrives to agency i, it will send an access request to the RC controlling the resource. The RC just has to compute the hash of the code m_i and check, through a request to the CRM, if the agent is member of a role authorized to access the resource.

The *RM-managed* approach can be used to directly authorize an agent. If the operation is carried out by an authorization manager it can delegate an authorization directly to the agent hash. The agent does not need to belong to a role. This does not introduce any conflict or problem to the system, since it is fully supported by SPKI/SDSI resolution and reduction algorithms.

The main drawback of this approach is that a mobile agent is no capable of issuing certificates, since the agent can not sign them. But note that this does not mean, that the agent cannot issue or delegate an authorization, which may be certified by a trustee.

Note that the authorizations associated to an agent may be determined by its role membership. This way we can say that the agent will have *dynamically assigned authorizations* during its lifetime. If the authorizations associated with a role change, the authorizations related to the agent also change.

6 Conclusions

We have proposed an access control system for a mobile agent platform. It provides a simple, flexible and scalable way of controlling the access to resources. It takes the ideas from RBAC and *trust management*. The proposed model is an extension of the MARISM-A project, a secure mobile agent platform based on JADE.

Currently we have implemented the main MARISM-A platform components (mobility, services, etc.), and a prototype of the authorization framework for access control proposed in this paper. Our solution provides a secure migration for agents with protected itineraries and we solve the secure resources access control and the authorization management. We provide a lightweight mechanism to authorize mobile agents, where the agent does not need to carry any kind of information regarding the access control. It introduces a method to establish and manage roles, which may be used in an open system with unknown users. This kind of users are normally untrusted and by the way authorizations are assigned, it will be feasible to introduce some accountability mechanisms to the system. The definition of access control policies also allows to separate the right of requesting an authorization from the authorization itself.

As a future work we plan to incorporate support for standard security languages, such as SAML to introduce more interoperability. Given the simplicity of the current framework this should not be a difficult issue.

References

1. Chess, D.: Security issues of mobile agents. In: Mobile Agents. Volume 1477 of LNCS., Springer-Verlag (1998)
2. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: RFC 2693: SPKI certificate theory. IETF (1999)
3. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The KeyNote Trust Management System. RFC 2704, IETF (1999)
4. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. In: ACM Transactions on Information and System Security. Volume 4. (2001)
5. Tripathi, A., Karnik, N.: Protected resource access for mobile agent-based distributed computing. In: Proceedings of the ICPP workshop on Wireless Networking and Mobile Computing. (1998)
6. Hu, Y.J.: Some thoughts on agent trust and delegation. In: Proceedings of the fifth International Conference on Autonomous Agents. (2001)
7. Aura, T.: Distributed access-rights management with delegation certificates. In: Secure Internet Programming: Security Issues for Distributed and Mobile Objects. Volume 1603 of LNCS. Springer-Verlag (1999)
8. JADE Board: Jade security guide. JADE-S Version 2 add-on (2005)
9. Suri, N., Bradshaw, J., Breedya, M., Groth, P., Hill, G., Jeffers, R., Mitrovich, T.: An overview of the nomads mobile agent system. In: Proceedings of 14th European Conference on Object-Oriented Programming. (2000)
10. Bradshaw, J. M., D.S.B.P..W.J.D.: KAoS: Toward an industrial-strength open agent architecture. Software Agents (1997)
11. Robles, S., Mir, J., Ametller, J., Borrell, J.: Implementation of Secure Architectures for Mobile Agents in MARISM-A. In: Fourth Int. Workshop on Mobile Agents for Telecommunication Applications. (2002)
12. Cartrysse, K., van der Lubbe, J.: Privacy in mobile agents. In: First IEEE Symposium on Multi-Agent Security and Survivability. (2004)
13. Li, N., Winsborough, W., Mitchell, J.: Distributed credential chain discovery in trust management. In: ACM Conference on Computer and Communications Security. (2001)
14. FIPA TC Ad Hoc: Fipa agent discovery service specification (2003)