A Security Scheme for Mobile Agent Platforms in Large-Scale Systems

Michelle S. Wangham, Joni da Silva Fraga, and Rafael R. Obelheiro

Department of Automation and Systems
Federal University of Santa Catarina
C. P. 476 – 88040-900 – Florianópolis – SC – Brazil
{wangham,fraga,rro}@das.ufsc.br

Abstract. Mobile agents have recently started being deployed in large-scale distributed systems. However, this new technology brings some security concerns of its own. In this work, we propose a security scheme for protecting mobile agent platforms in large-scale systems. This scheme comprises a mutual authentication protocol for the platforms involved, a mobile agent authenticator, and a method for generation of protection domains. It is based on SPKI/SDSI chains of trust, and takes advantage of the flexibility of the SPKI/SDSI certificate delegation infrastructure to provide decentralized authorization and authentication control.

1 Introduction

A mobile agent in a large-scale network can be defined as a software agent that is able to autonomously migrate from one host to another in a heterogeneous network, crossing various security domains. In order for these agents to exist within a system or to form themselves a system, they require a computing environment—an agent platform—for deployment and execution.

The ability to move agents (code + state) allows deployment of services and applications in a more flexible, dynamic, and customizable way with respect to the client-server paradigm [1]. Despite its many benefits, the mobile agent paradigm introduces new security threats from malicious agents and platforms [2]. Due to these threats, security mechanisms should be designed to protect the communications infrastructure, agent platforms and agents themselves. This paper concentrates on mechanisms for protecting agent platforms against malicious agents, considering large-scale distributed systems.

One of the main concerns with an agent platform implementation is ensuring that agents are not able to interfere with one another or with the underlying agent platform [3]. A common approach for accomplishing it is to establish isolated execution domains (protection domains) for each incoming mobile agent and platform, and to control all inter-domain access. Protection against malicious agents is not restricted to confining their execution to their own execution domains in agent platforms; other issues need to be considered when distributed large-scale systems are the focus. For instance, generation of these protection

A. Lioy and D. Mazzocchi (Eds.): CMS 2003, LNCS 2828, pp. 104–116, 2003.

domains depends on distributed authentication and authorization mechanisms, which makes it a difficult task.

The Java platform is quickly becoming the language of choice for implementing mobile agent systems. Besides being considered a *de facto* standard for programming distributed applications, Java has several properties that make it a good fit for this task. However, although Java is very convenient for the creating mobile agents, its static and centralized access control model poses some limitations with regard to security policy definition.

These considerations have led us to design a security scheme for protecting agent platforms against malicious mobile agents in large-scale systems. This scheme takes advantage of the flexibility of the SPKI/SDSI certificate delegation mechanisms to accomplish decentralized authentication and authorization. It includes a protocol for establishing secure channels, an algorithm for authentication of incoming mobile agents, and a scheme for generating isolated protection domains for agents that aims to overcome some limitations of the Java 2 access control model.

2 Security in Mobile Agent Platforms

A mobile agent platform provides an environment where agents can execute themselves and interact with other agents. A malicious agent can attack the platform it is currently visiting or other agents on the same platform, thus posing a significant threat to this platform. Possible security threats facing mobile agent platforms include [3]: *masquerading*, when an agent poses as an authorized agent in an effort to gain access to services and resources to which it is not entitled or to deceive the agent with which it is communicating; *denial of service*, when an agent attempts to consume an excessive amount of the agent platform's computing resources or to send repeatedly messages to another agent; *unauthorized access*, for example when an agent obtains read or write access to data for which it has no authorization, including access to agent's state or code, and *repudiation*, when a agent participates in a transaction or communication and later claims that it did not happen.

Establishing isolated domains for agents is the most common technique for protecting agent platforms resources and functionalities against malicious agents. In addition to this approach, other techniques were proposed based on conventional security techniques. Some of these techniques are: safe code interpretation [4][5], digital signatures [4][6][7], path histories [8], State Appraisal [2], and Proof-Carrying Code (PCC) [9].

Many of these mechanisms offer an effective security to agent platforms and their resources for some classes of applications, particularly when techniques are combined. For instance, the domain isolation technique combined to code signing (as provided in the Java 2 platform [4]) makes it possible to implement run-time access control based on the authority of an agent (the owner)[1]. However, these

[1] Limitations of the Java 2 access control model are described in section 6.

techniques are not really suitable to large-scale applications. Moreover, access control based solely on owner of the agent does not seem to be appropriate when multi-hop agents with free destinations are taken into consideration, since the trust in an agent depends not only on the owner and the forwarding platform but also on all platforms visited by the agent [8]. Therefore, an effective authentication and authorization scheme for large-scale mobile agent systems should be based on agent credentials, on the identities of agent owners and on the lists of platforms visited by agents. Identities and credentials are usually represented by, and stored in, digital certificates, such as those used in SPKI/SDSI.

3 SPKI/SDSI Infrastructure

The *Simple Public Key Infrastructure/Simple Distributed Security Infrastructure*(SPKI/SDSI) specification defines a simple and flexible distributed authentication and authorization infrastructure based on digital certificates and local name spaces [10].

SPKI/SDSI uses an egalitarian model of trust. The subjects (or principals) are public keys and each public key is a certificate authority [11]. There is no hierarchical global infrastructure as in X.509. Two types of certificates are defined by SPKI/SDSI: name and authorization certificates. An authorization certificate grants specific authorizations from the issuer to the subject of the certificate; these authorizations are bound to a name, a group of names or a key. The issuer of the certificate can also allow a principal to delegate the received permissions to other principals.

The SPKI/SDSI delegation model allows the construction of chains of trust that begin with a service guardian and arrive in principals' keys. When a given subject desires to obtain access to some resource, it must present a signed request and a chain of authorization certificates for checking the needed permissions [11].

Since SPKI/SDSI follows a decentralized approach to authentication and authorization, it is suitable for large-scale systems. Access rights can be delegated to form a chain of certificates (controlled distribution of authorization). Authorizations and permissions can be freely defined and are not restricted to any predefined set [10]. The issuer of a certificate can specify certain conditions under which the certificate is valid; this provides for finer-grained control of delegation. Since certificates are bound to keys instead of names, they eliminate the need for finding the public key corresponding to a given name [12], and can be also used in situations where anonymity is desired. Due to these advantages, SPKI/SDSI certificates were chosen to represent agent credentials in our security scheme.

4 A Proposal for Authentication and Authorization Based on Chains of Trust

We now present an authentication and authorization scheme for large-scale mobile agent systems. We assume that agents have free itineraries and are multi-hop, that is, the number of platforms they traverse in a given itinerary is not

fixed. SPKI/SDSI chains of trust and the concept of federations provide scalability and flexibility to this scheme.

4.1 Platform Federations

In this scheme, we use the concept of *federation* introduced in [13], which emphasizes the grouping of principals with common interests. With that, mobile agent platforms can group according to their service classes, constituting service federations. For example, the Mold Enterprise Federation may group the mobile agent platforms from enterprises that manufacture molds and matrices. The purpose of a federation is to assist its members on reducing principal names and on building new chains of trust through its Certificate Manager (CM) [13]. These chains of trust between client and service are quickly and efficiently established from name and authorization certificates available at the certificate repository of the service federation. By storing name and authorization certificates in these repositories, the services available in the associated platforms can be announced. The inclusion of a platform in one of these federations should be negotiated with the association that controls this certificate storage service. The CM offers a certificate search alternative, either for name reduction or for creating new authorization chains.

Besides, a member of a federation can join other federations and different federations can establish trust relationships. The certificate managers can be associated to each other, linking those who, for affinity, can better represent the needs of their members, creating webs of federations with global scope. The main function of a web of federations is to help a client, through its agents, in the search for access privileges that link it to the guardian of a service (another platform). Further details on the concept of federations can be found in [13].

4.2 Authentication and Authorization Scheme

Fig. 1 shows the procedures defined in the security scheme, composed by prevention and detection techniques that emphasize the protection of agent platforms and their resources. In this scheme, after the mobile agent creation process, the source and destination platforms (the ones which send and receive agents, respectively) first go through a mutual authentication protocol so that a secure channel between them can be established. After that, the agent will be sent with its credentials to be authenticated by the destination platform and then its domain of execution can be created. In other words, when an agent arrives in a platform it presents its public key and a chain of SPKI/SDSI certificates to a verifier that performs the authorization checks. From these information, this verifier must generate the permissions required for the agent to be run in a protection domain on its destination platform. This dynamic generation of permissions provides flexibility and follows the principle of least privilege. Chains of trust also help to achieve the necessary scalability for Internet-based applications. The following section analyzes some aspects referring to the mechanisms in the proposed scheme.

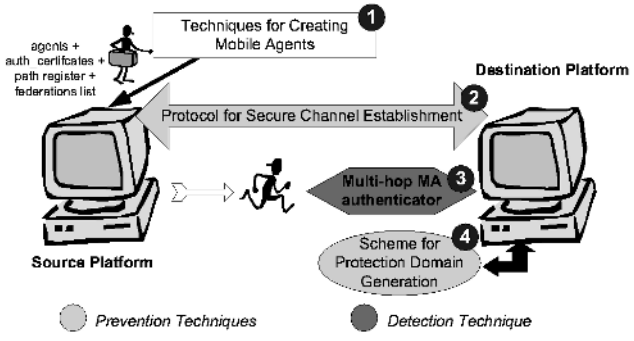

Fig. 1. Security Scheme for Agent Platform Protection

Creation of Mobile Agents. During the mobile agent creation process (see Fig. 1, procedure 1), the owner, being the authority that an agent represents, provides a set of SPKI/SDSI authorization certificates defining the agent's credentials. It should be noted that this initial set of authorization certificates may not be sufficient to grant access to certain resources in a given platform. So, new certificates can be provided to the agent during its visits to other agent platforms. For example, suppose that an agent needs to visit a transport enterprise associated to a mold enterprise. The agent may not have the certificates needed to be received in this platform. Thus, the platform that represents the mold enterprise can delegate certificates to this agent enabling it to access the associated enterprise. The trust model proposed in SPKI/SDSI determines that a client is responsible for finding certificates that enable it to access a given service. Therefore, an agent can search for certificates on the webs of federations and negotiate them when they are found.

The owner of the agent has to put in an object that will contain the list of previously visited platforms (called the *path register*) a signature indicating its identity and the identity of the first platform to be visited. This object is attached to the agent. A list that indicates service federations whose member platforms are authorized to execute the agent can be (optionally) defined and attached to the agent. The path register and the list of federations are used to analyze the history of the agent's travels in the agent authentication process.

Finally, the agent's owner must sign the code of the agent and its read-only data, and then create the agent in its home platform to send it through the network.

Secure Channel Establishment. In the proposed scheme, mutual authentication between the platforms involved must be established before agents can be transferred. This authentication creates a secure channel in the communications infrastructure between the authenticated parties that is used for agent transfers.

In accordance to the SPKI/SDSI model, identification is not done with names, but with public keys, with digital signatures as the authentication mech-

anism. Thus, in platform authentication, for a digital signature to be verified, a public key and a signed request must be present at the receiver.

Mutual authentication is performed during the secure channel establishment between agents platforms. Fig. 2 shows the mutual authentication performed with a challenge-response protocol, based on SPKI/SDSI certificates of the owners (managers) of the platforms. The basis for authentication in SPKI/SDSI are chains of authorization certificates [10].

In step 1, Fig. 2, the source platform sends a signed message containing a request (*establish_trust*) and a nonce (*nonceSP*), without any certificates. From this request, the destination platform builds a signed challenge and sends it to the source platform so that it can prove it has the required permissions (step 2). The challenge is composed by information from the resource's ACL, by *nonceSP* and by a nonce generated by the destination platform (*nonceDP*). In step 3, the source platform verifies the signature of the challenge to confirm the authenticity of the destination platform. Then, it sends a signed response with the request, *nonceDP*, and the authorization certificates for the desired operation. From the chain of authorization certificates, the destination platform can check the requester's signature, finishing the authentication process (step 4).

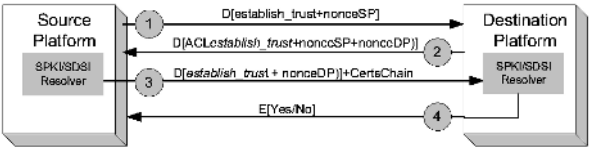

Fig. 2. Protocol for Mutual Authentication of Platforms

It is important to note that the process of mutual authentication of platforms is concluded with the establishment of a secure channel. This channel will be used for all agents that are transferred between the two platforms, without need for subsequent platform authentication.

For secure channel establishment, an underlying security technology—Secure Sockets Layer (SSL)—is used to ensure confidentiality and integrity of communications between agent platforms. When a secure channel is established, the source platform sends the agent to the destination platform along with its its credentials for building an execution domain that is appropriate for the agent.

Mobile Agents Authentication. Before instantiating a thread to an agent, the destination platform must authenticate the received agent. In order to protect against an agent, a platform depends not only on the verification of the agent's owner authenticity, but also on the degree of trust in the platforms already visited by the agent, since a mobile agent can become malicious by virtue of its state having been corrupted by previously visited platforms [2]. One of the

contributions of this paper is the definition of a multi-hop authenticator that establishes trust on an agent, based on the authenticity of the owner of the agent, on the authenticity of the platforms visited by the agent and on the federations defined by the owner of the agent.

Consider the authenticator shown in Fig. 3; upon receiving a mobile agent, a platform must first check, through verification of the agent's signature (code and read-only data), that this agent has not been corrupted and confirm its association to a principal, its owner (step 1). Thus, modifications introduced by malicious platforms can be easily detected by any platform visited by the agent.

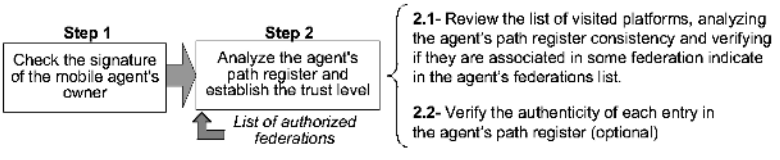

Fig. 3. Multi-hop Agents Authenticator

For one-hop agents, the technique proposed in step 1 ensures the integrity of an agent, but for multi-hop agents this technique is insufficient. For detecting possible modifications and checking the agent's traveling history, the destination agent platform must analyze the agent's path register (step 2). For that purpose, each platform visited by the agent should add to the agent's path register a signed entry containing its identity (public key) and the identity (public key) of the next platform to be visited, forming a history of the path followed by the agent. In step 2, Fig. 3, the platform administrator has to define how the agent's path register is analyzed and how the trust level is established. The possibilities shown in Fig 3 include only step 2.1, only step 2.2, or both steps[2].

Moreover, we suggest that platform-generated sensitive data (write-once data) should be stored in a container to be carried by the agents (as proposed by Karnik in [7]). These sensitive data should be signed by the generating platform so that possible modifications can be detected. This approach is vulnerable to some attacks, however [14][15][16]. For instance, Roth [16] describes an attack where a malicious platform, which is visited a second time by an agent or which colludes with another platform, deletes all items that were added to the agent's container since one of its previous visits or since the agent's departure from the first platform. In this paper, we focus on protecting agent platforms, but it is our intention to address the protection of agents in future work.

Generation of Protection Domains. Protection domains and the permissions assigned to them are defined after trust in an agent has been established

[2] An inconvenient is that analyzing an agent's path becomes costlier as the path register grows.

(a result from previous procedures). They are based on the agent's SPKI/SDSI authorization certificates and on trust and risk levels. The platform guardian verifies the agent's certificates in order to define the set of permissions. This decouples the privileges granted to agents (their credentials) from the privileges required to access resources (the access control policy), which provides flexibility and scalability to the security scheme.

Some extensions to the Java 2 security model are needed for generating the protection domain where an agent will be run. These extensions, represented by grey boxes in Fig. 4, are: *SPKISecureClassLoader*, required for extracting certificates from the incoming agent and for creating a protection domain of a thread; *SPKIPolicy*, an object that represents a policy that defines, from the certificates carried by an agent, which Java permissions will be associated to the application domain; and *SPKIVerifier*, required for verifying the authenticity of SPKI certificates.

Following the dynamics depicted in Fig. 4, the platform administrator describes the security policy of the agent platform by mapping the authorization granted from SPKI/SDSI certificates to Java permissions, defining for that purpose a policy file. When an agent is received in a platform, its credentials are forwarded by *SPKISecureClassLoader* to the *SPKIPolicy* object which interprets them. When SPKI permissions are mapped to Java permissions, the Java support generates the corresponding protection domain for the thread that runs the agent; the Java permissions are made available through *PermissionCollection*.

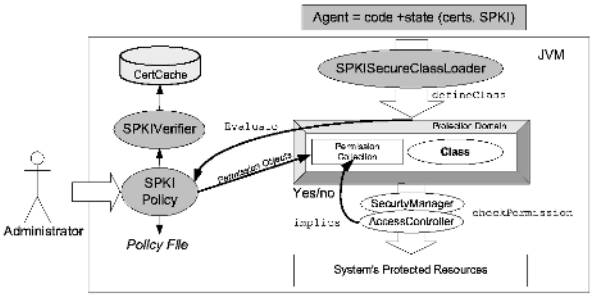

Fig. 4. Dynamics for Protection Domain Generation

If a thread (agent) makes an access request to a resource outside of its application domain, that is, in a system domain, *AccessController* is activated to check whether the access must be allowed. It must verify, in the created protection domain, whether the principal has a corresponding *Permission* object in its collection of permissions. If it does, the thread can exchange domains, entering in the system domain.

5 Implementation

A prototype of the security scheme for protection of agent platforms has been defined and implemented in order to demonstrate its suitability. The architecture of this prototype is shown in Fig. 5.

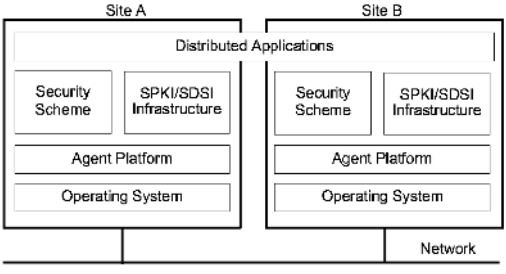

Fig. 5. Architecture of the Prototype

For the mobile agents support layer we have chosen IBM Aglets[3], an open-source platform that uses the Java platform. Aglets provides mechanisms for code and state information mobility, and an environment (called Tahiti) which supports creation, cloning, dispatching, and retraction of agents. Aglets supports both Agent Transfer protocol (ATP) and Java Remote Method Invocation (RMI) as communication infrastructures. In our work, we have chosen to use only RMI because it is better suited to purely Java-based distributed systems.

The SPKI/SDSI Infrastructure component of the prototype architecture is responsible for the creation of SPKI/SDSI certificates for agent platforms and mobile agents. For this component, we have adapted an existing Java library that implements SDSI 2.0 [17] and provides the necessary functionalities.

A tool for assisting the agent creation process was implemented. This GUI-based application allows an owner to define the credentials and list of federations for an agent, to sign the code and read-only data, and to initialize the path register and the write-once container.

The protocol for secure channel establishment (see Fig. 2) was implemented with the SDSI 2.0 library and with Java 2 cryptographic tools. SSL support is provided by the iSaSiLk toolkit[4]. The Aglets platform was adapted to optionally use RMI over SSL.

The multi-hop authenticator described in section 4.2 is being implemented with the Java 2 cryptographic tools and the SDSI 2.0 library. Presently, all entries in an agent's path register are analyzed considering only step 2.1 (Fig. 3). Only two levels of trust were defined according to the list of federations: *authorized* or *non-authorized* platforms. That is, the platform either is or is not a member

[3] http://aglets.sourceforge.net/
[4] http://jce.iaik.tugraz.at/products/02_isasilk/

of a federation present in the list of federations previously defined for the agent mission. The multi-hop algorithm as currently defined is shown in Fig. 6.

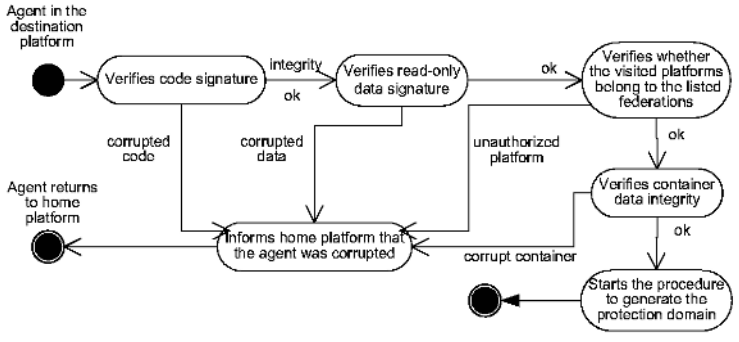

Fig. 6. Multi-hop Authenticator

The scheme for generation of protection domains (see Fig. 4) has been fully designed but only partially implemented in the Aglets Platform. When a agent is dispatched, the Aglets Platform attaches the certificates defined in the agent creation to the serialized agent through *SPKIAgletWriter*. When the agent is received in the destination platform, *SPKISecureClassLoader* calls *SPKIAgletReader* to extract these certificates.

6 Related Work

Most Java-based agent platforms take advantage of the Java security models (especially the Java 2 version) to implement part of their security mechanisms. Among these platforms, there are commercial ones, such as IBM Aglets and GMDFokus Grasshopper, and academic ones, such as SOMA (University of Bologna) and Ajanta (University of Minnesota).

These mobile agent platforms extend the Java Security Manager to provide a more flexible and adequate solution to agent platforms and to implement protection domains that isolate mobile agents, preventing malicious attacks from them. The difference between authorization schemes in these platforms lies in the information used to determine the set of access rights for an incoming agent. Aglets and Ajanta use only the agent's owner identity. Grasshopper uses access control policies based on the owner's identity or on the name of its group (group membership). In SOMA the principals are associated to roles that identify the operations allowed on the resources of the system. All these approaches are not really suitable to large-scale systems.

Besides, it is important to note that the Java 2 access control model has some limitations that need to be analyzed. Instead of following the distributed

nature of its execution model, the Java 2 security model uses a centralized authorization scheme[5]. When running, each code is labeled as belonging to one or more protection domains. Each domain has a set of permissions associated from a policy configuration file. Therefore, this file defines a static mapping between each mobile component and the permissions granted to it for execution in a local environment. In addition to a number of difficulties related to programming, development of a distributed and dynamic environment is constrained by limitations that stem from the concentration of trust on a single configuration file, which demands an up-front static definition of all distributed components in the system and their corresponding security attributes.

Agent authentication is essential for implementing an effective authorization scheme in mobile agent systems. The Aglets and Grasshopper platforms do not have mechanisms for mobile agent authentication. SOMA authenticates agents based on several data contained in its credentials: domain and place of origin, class which implements the agent and user responsible for the agent. Before migration, these information, the initial state of agent and its code are digitally signed by the user that creates the agent. When an agent arrives at a remote site, its credentials are verified with regard to authenticity by checking the signature of the agent's owner. The Ajanta platform uses a challenge-response authentication protocol with random nonce generation to prevent replay attacks, based on the signature of the agent's owner.

In comparison to the static model in Java 2 and to the platforms discussed above, our scheme has the advantage of decoupling privilege attributes (credentials) from control attributes (policies), its use of some Java security features notwithstanding. This means that, although a policy configuration file still needs to be statically defined, the proposed mechanisms add the flexibility offered by SPKI certificates to domain generation. That is, domains are dynamically defined when an agent aggregates to its credentials the delegated certificates received during its itinerary.

Besides, in the agent authentication process described in section 5, the information used to determine an agent's set of access rights is based not only on the identity of the agent's owner, but also on the public keys of the owner and of the visited platforms, which avoids global name resolutions in large-scale systems.

Two related proposals use SPKI/SDSI certificates to improve access control in Java 2. The first, developed by Nikander and Partnen [12], uses SPKI authorization certificates to delegate Java permissions that directly describe possible permissions associated to a protection domain. In this work, the authorization tag of the SPKI certificate was extended to express Java permissions. This solution has the disadvantage of using modified SPKI certificates. The second work [18] proposes two improvements to access control in Java 2: association of access control information to each mobile code segment (applet) as attributes, and the introduction of intermediate elements in the access control scheme for assisting

[5] This centralization refers to the fact that all access control is done from a single configuration file that defines the whole security policy of a machine. Thus, there is only one ACL relating all subjects and objects in the machine.

the configuration of access control attributes of incoming mobile programs and of access control information located in the runtime environments. The SPKI/SDSI group mechanism is implemented through name certificates and makes these improvements possible. Molva and Roudier's work [18] does not provide details on how their proposal can be implemented nor on how to combine it to current Java 2 security model.

Both works discussed above do not deal with mutual authentication between source and destination platforms nor analyze the history of the visited platforms to establish trust on mobile code. Only the first proposal has flexibility characteristics similar to the ones proposed in the present work, in which the domains are formed according to the certificates delegated to an agent in its creation and throughout its itinerary. Nikander and Partanen propose that the search for forming new chains should be responsibility of the server. However, as mentioned before, this is not in accordance to the SPKI/SDSI model.

7 Concluding Remarks

Security issues still hamper the development of applications with mobile systems. Current security mechanisms do not present satisfactory results for protecting mobile agent platforms. There are even more limitations when we consider large-scale systems, which impose stronger requirements with regard to flexibility and scalability.

Our security scheme was motivated by perception of these limitations and a concern with aspects of security specific to large-scale applications. Its purpose is to prevent mobile agent attacks against platforms, defining a procedure that employs a combination of prevention and detection techniques. This scheme is based on decentralized authorization and authentication control that is suitable for large-scale systems due to its use of SPKI/SDSI authorization certificates. The mechanism of authorization certificate delegation allows a separation between agent credentials and security policy definition. The scheme for generation of protection domains is more flexible than those of the related works.

The work described in this paper, although not fully implemented yet, already presents satisfactory results. As soon as it is concluded, its performance will be properly evaluated. This prototype is currently being integrated to a distributed Internet-based application in order to demonstrate its usefulness [19]. Considering the protection of platforms from agents and of the communication channel, the proposed security scheme effectively mitigates the perceived security threats, albeit further work is still needed to define mechanisms for protection of mobile agents against malicious agent platforms.

Acknowledgments. The authors thank the "IFM (Instituto Fábrica do Milênio)" and "Chains of Trust" project (CNPq 552175/01-3) members for their contributions, especially Elizabeth Fernandes, Galeno Jung, Ricardo Schmidt, and Rafael Deitos. We also thank the anonymous reviewers for their helpful

comments. The first and second authors are supported by CNPq (Brazil). The third author is supported by CAPES (Brazil).

References

1. Vigna, G., ed.: Mobile Agents and Security. LNCS 1419. Springer-Verlag (1998)
2. Farmer, W., Guttman, J., Swarup, V.: Security for mobile agents: Issues and requirements. In: Proc. 19th National Information System Security Conference. (1996)
3. Jansen, W., Karygiannis, T.: Mobile agent security. Technical Report NIST Special Publication 800-19, National Institute of Standards and Technology (1999)
4. Sun: Java 2 SDK security documentation. (2003) `http://java.sun.com/security/`.
5. Levy, J., Ousterhout, J., Welch, B.: The Safe-Tcl security model. Technical Report SMLI TR-97-60, Sun Microsystems (1997)
6. Gray, R., Kotz, D., Cybenko, G., Rus, D.: D'Agents: Security in a multiple-language, mobile agent systems. In Vigna, G., ed.: Mobile Agents and Security. LNCS 1419. Springer-Verlag (1998) 154–187
7. Karnik, N.: Security in Mobile Agent Systems. PhD thesis, University of Minnesota (1998)
8. Ordille, J.: When agents roam, who can you trust? In: 1st Conference on Emerging Technologies and Applications in Communications. (1996)
9. Necula, G., Lee, P.: Safe, untrusted agents using proof-carrying code. In Vigna, G., ed.: Mobile Agents and Security. LNCS 1419. Springer-Verlag (1998) 61–91
10. Ellison, C.M., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylönen, T.: SPKI requirements. RFC 2693, Internet Engineering Task Force (1999)
11. Clarke, D.E.: SPKI/SDSI HTTP server/certificate chain discovery in SPKI/SDSI. Master's thesis, Massachusetts Institute of Technology (MIT) (2001)
12. Nikander, P., Partanen, J.: Distributed policy management for JDK 1.2. In: Proc. 1999 Network and Distributed Systems Security Symposium. (1999)
13. Santin, A., Fraga, J., Mello, E., Siqueira, F.: Extending the SPKI/SDSI model through federation webs. In: Proc. 7th IFIP Conference on Communications and Multimedia Security. (2003)
14. Yee, B.: A sanctuary for mobile agents. In: Secure Internet Programming. LNCS 1603. Springer-Verlag (1997) 261–273
15. Karjoth, G., Asokan, N., Gulcu, C.: Protecting the computing results of free-roaming agents. In: Proc. 2nd International Workshop on Mobile Agents. (1998)
16. Roth, V.: On the robustness of some cryptographic protocols for mobile agent protection. In Picco, G.P., ed.: Mobile Agents. LNCS 2240. Springer-Verlag (2001) 1–14
17. Morcos, A.: A Java implementation of Simple Distributed Security Infrastructure. Master's thesis, Massachusetts Institute of Technology (1998)
18. Molva, R., Roudier, Y.: A distributed access control model for Java. In: European Symposium on Research in Computer Security (ESORICS). (2000)
19. Rabelo, R., Wangham, M., Schmidt, R., Fraga, J.: Trust building in the creation of virtual enterprises in mobile agent-based architectures. In: 4th IFIP Working Conference on Virtual Enterprises. (2003)