Privilege Management of Mobile Agents

Wayne Jansen, Tom Karygiannis
National Institute of Standards and Technology

Abstract: *Most mobile agent systems use internal data structures within an agent to control and specify its security requirements and properties. These structures typically contain authorization information regarding access to computational resources on distributed systems and conceptually serve as an internal passport for the agent. While these structures are often very similar semantically, they differ greatly in their implementation, depending to a large extent on the mechanisms used to protect their contents. This paper considers a general scheme for managing privileges using attribute certificates. An attribute certificate can be viewed as an external, digitally signed agent passport, which allows greater flexibility in meeting the needs of an application and overlaying a suitable management scheme. The paper presents the benefits of this approach and gives an example of how an agent system could be enhanced with this mechanism.*

Introduction

A software agent is loosely defined as a program that exercises an individual's or an organization's authority, and autonomously meets and interacts with other agents and its environment while working toward a goal. Possible interactions among agents include such things as contract and service negotiation, auctioning, and bartering. Software agents may be either stationary or mobile. Stationary agents remain resident at a single platform, while mobile agents travel among platforms by suspending their execution, moving themselves to another platform, and resuming execution upon arrival.

A number of models exist for describing agent systems. For the purpose of discussing mobile agent security issues, however, a simple model consisting of only two components, the agent and the agent platform, is sufficient. An agent consists of the code, data, and state information needed to carry out some computation. The agent platform provides the computational environment in which an agent operates. Multiple agents can cooperate with one another to carry out some task and are able to move or hop among agent platforms. The platform where an agent is instantiated and commences activity is referred to as the home platform, and normally is the most trusted environment for an agent. One or more hosts may make up an agent platform, and an agent platform may support multiple locations or meeting places where agents can interact. Since some of these details do not affect the discussion of security issues, they are omitted from the agent system model illustrated in Figure 1, which depicts the movement of an agent among several platforms.

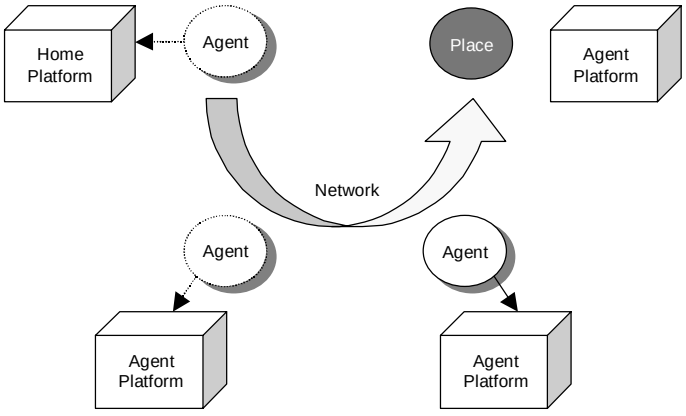

Figure 1: Agent System Model

Mobile agent computing is an extreme form of distributed computing, which poses a severe challenge to the security of an application. One difficult class of threats introduced by mobility is the possibility that the computational environment (i.e., the agent platform) may attempt to subvert the agents comprising the application. Other threats include agents attacking the agent platform itself or other agents at a platform, and outside entities attacking the overall agent framework. A wide range of techniques, both conventional and newly developed for this paradigm, are

available as technical countermeasures to the security threats encountered in deploying agent-based applications [Jan99]. Agent systems typically incorporate such techniques into their design, where internal data structures reflect if, how, and when the mechanism should be applied. Besides control settings on security mechanisms, internal data structures also convey the permissions needed by an agent, such as resource access or consumption. This paper describes a general-purpose mechanism for managing the privileges (e.g., resource permissions and security mechanism settings) of mobile agents based on the use of *attribute certificates*. An attribute certificate allows various principles to govern the activities of mobile agents through selective privilege assignment. This paper also outlines the motivation for using attribute certificates and describes how to implement this mechanism within an agent system.

Background

Security policy, in general, refers to a documented set of information system security decisions. A technical security policy defines the rules needed to be enforced by the security mechanisms and controls of the underlying hardware and software that comprise a system. Technical security policies must be mapped to the specific security mechanisms provided by an agent system. Privileges, as described above, are intimately tied to security policy, since they provide the context for the safe execution of mobile code on a host platform. Agent systems can express and handle security policy in a variety of ways. By briefly looking at the past work on three agent systems, Aglets Workbench [Lan98, Kar97], Ara [Pei98], and Telescript [Tel95, Tar96], we can compare their similarities and differences with regard to policy and privileges, and attempt to answer several fundamental questions about policy handling and expression. The discussion uses the term attributes to refer to security-relevant privileges associated with an agent system.

Who Expresses Policy?

In an automated system, the principals who need to express security policy must have an authenticated identity. Their authorizations must also be consistent with the expressed policy. The security model for the Aglets Workbench suggests that the following principals have a strong interest in defining the security policy of an agent:

- Agent manufacturer – wants to limit product liabilities and ensure that no damage can be caused by a malfunction,
- Agent owner – wants to ensure that the results are trustworthy and no unwanted actions occur as side effects,
- Platform owner – wants to ensure the safety and security of the context, fair and controlled consumption of underlying resources, and controlled interaction among agents being hosted, and
- Domain authority – wants to ensure the safety and security of a collection of agent platforms [Kar97].

The Ara system suggests three policy-setting principals:

- Agent user – wants to ensure that the results are trustworthy and no unwanted actions occur as side effects and may or may not be the agent owner,
- Agent manufacturer – the same as with Aglets, and
- Platform owner – the same as with Aglets [Pei98].

Telescript suggests three similar, but slightly different, policy-setting principals than those above:

- Agent creator – the instantiator of the agent; similar to agent user in Ara,
- Regional authority – similar to the domain authority in Aglets if the region includes multiple platforms; otherwise, it is similar to the platform owner, and
- Local authority – similar to the platform owner in Aglets and Ara if the platform supports a single context; otherwise it is a new type of context authority [Tar96].

What Kind of Policy is Expressed?

Agent systems need policy controls to function effectively and safely. Some agent systems allow users to express policy controls when the agent is instantiated. These policy controls are expressed by the assignment of values to attributes defined in a data structure within an immutable part of an agent (e.g., in Ara, the structure is called an agent passport.). The attributes are then checked and applied at each agent platform visited. This is a workable approach, since the user can easily assign appropriate values, but not necessarily a flexible one that can be easily extended and adapted. Let's look more closely at the attributes used in Ara and Telescript.

The Ara *agent passport* includes the following information:

- Global identifier of the agent,
- Date of birth,
- Allowance – access rights to various system resources, including files, CPU time (i.e., maximum time to live), memory (i.e., maximum size), network domain connections,
- Manufacturer and user certificates,
- Signature over code (by manufacturer),
- Signature over arguments supplied to the agent (by user),
- Signature over entire passport (by user), and
- Host trace – immutable part of agent containing a list of sites visited so far incrementally signed by receiving platforms.

Telescript uses *permits* to limit resource consumption and restrict the capabilities of executing code [Tel95]. Permit attributes include the following boolean permit attributes:

- CanGo – Go to another place,
- CanSend – Construct clones and send each to another place,
- CanCreate – Construct a peer process,
- CanRestart – Be restarted upon termination,
- CanCharge – Charge another process teleclicks,
- CanGrant – Grant a capability to any of certain other processes, and
- CanDeny – Deny a capability to any of certain other processes.

Non-boolean permit attributes in Telescript include the following:

- Age – The number of seconds since the subject was constructed,
- Charges – The permit's allowance (i.e., the number of teleclicks that are charged to the subject after its construction),
- Extent – The number of octets that is momentarily the subject's size,
- Authenticity – The subject's recognized level of authenticity, and
- Priority – The subject's recognized level of priority.

The Aglet Workbench, which is implemented in Java, relies on the JDK 1.2 policy file definitions in its security design to represent security policy [Lan98]. Although the Aglet Workbench security aspects are not yet fully implemented in recent distributions, the design captures enough detail to be useful for comparison purposes. The first group of attributes deals with resource types derived from JDK 1.2 and includes the following:

- File – entries in the local file system,
- Net – network access,
- Awt – the local window system,
- System – system resources such as memory and CPU time,
- QoP – quality of protection,
- Context – resources of the context, and
- Aglet – resources of the Aglet.

The second group of Aglet attributes defines owner preferences of an agent's capabilities to perform key activities. These security preferences are to be honored by the platforms visited by the agent and include restricting or allowing the following activities:

- Process-related operations including cloning, deactivating, dispatching, retracting, and disposing,
- Messaging operations including sending, subscribing, and unsubscribing, and
- Critical value operations including the setting of an agent's itinerary and properties, and the reading of context information.

How is Policy Expressed?

In order for policy to be acted upon by an agent system, it must be expressed in a manner that allows automatic processing. There is a significant distinction between the complexity of natural language expressions of technical policy envisioned by users and the simplified syntax of attributes supplied to an agent. It appears that a majority of

attributes can be expressed very simply by using attribute type-value pairs of boolean or integer variables. The agent systems under comparison have the following policy expression capabilities:

- Telescript – integer or boolean type-value pairs
- Ara – integer, boolean, or character string type-value pairs
- Aglets – authorization rules involving logical expressions of type-value pairs

How is Policy Conveyed?

Once a policy is expressed, it must be placed at appropriate processing points and maintained in a secure fashion. Each of the three agent systems conveys policy slightly differently. For each agent system, consider the perspective of a user launching an agent from its home platform to another platform:

- In Ara, the user's attributes are conveyed internally via the agent passport and are validated at the receiving platform using an admission function programmed by the platform owner. The admission function returns either the local allowance granted to the agent or a denial of admission.
- In Telescript, user attributes are conveyed via a native permit, which appears to be internal to the agent although it could be external. The agent's privileges are determined by intersecting the native permit with the regional and local permits in effect.
- In Aglets, attributes appear to be external and associated with the codebase of the agent. The receiving platform maintains a policy database against which the attributes of the agent's codebase are compared.

Rationale for Attribute Certificates

If mobile software agents are to operate on behalf of humans, they must follow a prescribed security policy established by principals. Rather than embody a prescribed policy as privileges within an agent, it is possible to push the policy-related information to an external object – an attribute certificate. The issuer of the attribute certificate adjusts the content in order to govern the agent's use of computational resources and security mechanisms. An attribute certificate must be signed by the issuer to protect the security-relevant information about the agent from alteration. Elements of an attribute certificate are pictured in Figure 2. They include the identity of the owner (formed by a secure hash over the agent's code and information), the identity of the issuer, the identifier of the algorithms used to protect the certificate, the lifetime of the certificate, and the subject attributes, which may be expressed as simple type-value pairs or more complex syntactical expressions. These elements can be used to establish the validity of the certificate and the binding between the attribute certificate and the agent. The reader is referred to the Related Work section of this paper for pointers to on-going efforts on standardizing the form and content of attribute certificates.

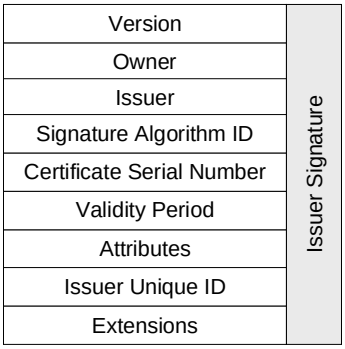

Figure 2: Attribute Certificate Elements

As an agent moves among agent platforms, it carries along its issued attribute certificate(s) and the identity certificate(s) of the issuer(s) who assigned the privileges to this agent (e.g., user or other policy-setting authority). A platform receiving the agent determines the relevancy of the certificate(s), verifies the issuer's identity, perhaps with the assistance of a Public Key Infrastructure (PKI), and determines whether compliance exists between the privilege assignment conveyed in the certificates and the platform's prevailing policy. Policy-setting principals can assign privileges via attribute certificates by either jointly signing and issuing a single attribute certificate to an agent, or individually issuing attribute certificates to an agent and having them interpreted and validated by the policy engine of the agent platform.

While there is variability in the number and types of policy-setting principals needed, it appears that in practice only a few are required. Unfortunately, the policy-setting principals may differ, depending on the application and the security framework in which they operate. For example, an application, intended for general use within an open electronic commerce environment, is likely to be significantly different in this regard from one intended for a command and control system within a military environment. As a result, it is difficult to build into an agent system a general-purpose privilege mechanism that meets the needs of all applications. Using attribute certificates to express policy in terms of privileges assigned by identified principals, however, does allow for a tailorable framework that can potentially meet the needs of most applications. In addition, because an attribute certificate is a signed object whose tampering can be detected, it can serve as an additional countermeasure for those agent systems that are vulnerable to attacks against an agent by a malicious platform via privilege modification.

Drawing on the various kinds of attributes reviewed earlier, several general categories can be delineated for application to agent systems:

- *System resources* – System resources include things such as processing time, memory space, disk storage, access to the network, access to the local window system, and execution priority. Although Java-based agent systems have problems with accounting for thread resource consumption of memory and CPU cycles, platforms based on other languages or enhanced virtual machines may not have such limitations [Sur00].
- *Agent capabilities* – Agent capabilities include the ability to perform certain agent-oriented functions such as cloning, issuing messages, subscribing to messages, changing itinerary, etc. A user or other principal may also want to constrain the agent's activities to some maximum number of hops among agent platforms, a maximum lifetime before returning to the home platform, or terminating, or a maximum dollar amount on financial transactions.
- *Security services* – Security services entail the use of attributes to invoke protection mechanisms at the platform such as performing host tracing, performing partial result encapsulation, protecting inter-platform communication for confidentiality, integrity, or mutual authentication, and restricting travel to certain security domains or to fixed itineraries.
- *Agent communication* – Besides services at the platform level, other attributes could apply at the agent communication level, for example, to control whether or not an agent should directly encapsulate the agent communication language dialogue itself. Moreover, given a particular ontology, attribute certificates may specify what types of communicative acts an agent can issue or respond to, and in what types of conversations it can participate.

The Aglet authorization rules hint at the possible richness of policy expression that might be desirable. Similar forms of expression are also echoed in somewhat related efforts in distributed system management. Matchmaker [Ram98] uses a complex form of policy expression similar to that of Aglet's to broker between service providers and consumers via a classified advertisement (or classads) data model. A classad allows logical expressions to be used within attributes to qualify the service offered or required and also supports arithmetic expressions and computations involving real numbers. A more general scheme devised by Koch et ali. entails the use of a Policy Definition Language (PDL) to specify executable rules suitable for automation of management policy [Koc96]. The PDL supports logical expressions used as the precondition for triggering management actions. The IETF's Security Policy Specification Language [Con00], although specialized for security and Internet communications, generally follows the PDL in terms of functionality. Here the management actions involve the form of protection to be applied to the communications.

What this seems to indicate is that while policy expression can be complex, for most applications simple attribute type-value pairs involving integers, booleans, or character strings, and logical expressions are sufficient to express agent policy. The main difficulty is determining a common syntax and representation that support semantic differences associated with the various policy-setting principals one can envision.

Attribute Certificate Processing & Agent Systems

Consider a simple scenario involving two policy-setting principals, a user and a platform owner. Both the user and the platform owner are issued attribute certificates conveying privileges assigned by the attribute authorities who administer their respective domains. In turn, the user who launches the agent issues an attribute certificate, which delegates a subset of those privileges to the agent. When the agent arrives at a platform, either inside or outside the

user's domain, the receiving platform processes the agent's credentials against the policy of the platform, resulting in the allowable set of privileges under which the agent must operate. This procedure is reminiscent of the intersection of native and regional permits under Telescript. In a similar fashion, the policy engine for the agent platform must calculate the intersection of pertinent privileges. Accordingly, the calculation must take into account any hierarchical relationship among policy-setting principals. For example, we would expect that the policy of the platform owner to take precedence over that of the agent user, should a discrepancy occur.

As one can see, the agent-platform, which provides the computational environment of an agent system, inherently shoulders responsibility for the processing of policy. This is quite understandable, given the design goals for an agent system regarding security (e.g., perform the requisite authentication and access control of other entities). Policy computation is a security-relevant mechanism and in classical security terms, must be part of the trusted computing base. Therefore, the aforementioned policy engine must be implemented as a trusted component of the agent-platform, and its results, the privilege set computed for the agent, must be enforced by other platform security mechanisms within the trusted computing base. While the policy engine must remain inside the protection boundary, it can be implemented in a flexible manner so as not to preclude the ability to use external certificates to express policy within a particular agent system.

The policy engine's job is twofold: to validate the chain of certificates associated with an agent, and to render a verdict on whether to allow processing by the agent and under what set of privileges. Validation of certificate chains can be in itself a complicated process involving the possibility of expired or revoked certificates and the necessity to retrieve supporting information. Rendering a verdict adds to the difficulty, since the policy engine needs to know about the principals involved and how they relate to one another from an authorization standpoint. That is, not only must the type of principal be indicated within the certificates, but also any precedence relationships among principals must be supplied to the policy engine.

The Mobile Code Toolkit (MCT) [Mob97] is a typical example of the present-day agent systems that are available, and one we are considering as a possible vehicle for developing the aforementioned security enhancements. MCT is an open-source agent system designed with an eye toward network management applications. MCT is implemented in the Java language and somewhat similar to the more widely known Aglets Workbench, another suitable vehicle for enhancement, if it were eventually to become an open source product. Besides mobile code support, MCT allows the placement of static code among platforms and automatic replacement of older versions, useful for controlling configuration. Figure 3 provides an illustration of the needed enhancements to the MCT to enable processing of attribute certificates. The sections that follow discuss these enhancements in greater detail.

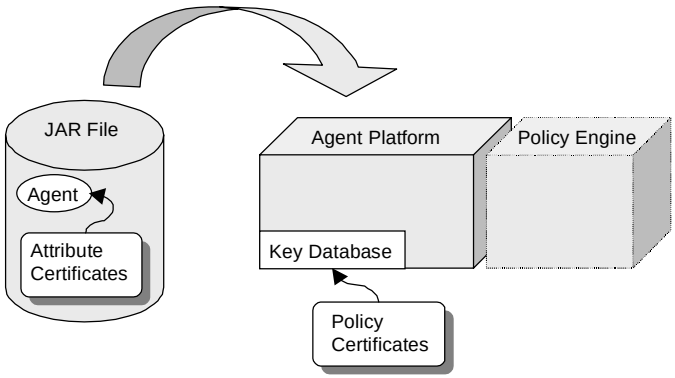

Figure 3: Attribute Certificate Processing

Coupling Agents with Attribute Certificates

The responsibility for maintaining relevant attribute certificates with an agent as it moves among platforms falls to the agent system. The current version of MCT allows movement of mobile code in the form of either individual class files or Java Archive (JAR) files. For incorporating security features, it is necessary to use JAR files, since they are the prescribed means within the Java framework for signing and verifying a codesource using digital signature techniques. In addition to the archived code, a JAR file contains a pair of files, a signature instruction and a digital

signature file, for each signer of one or more of the files contained in the archive. Additional information, such as the identity certificates of the entity that signed the code, may also be included within the archive to simplify the verification processing of the mobile code by a recipient.

JAR files could be used as a container for attribute certificates as well, by adding the attribute certificates to the JAR and requiring all the agent's code to be placed within the JAR file. However, it is important not to confuse the standard Java security features regarding signed JAR files with those of attribute certificates. In principle they are distinct, even though some redundancy would exist in situations where a policy-setting authority, such as a manufacturer or branding authority, issues an attribute certificate for an agent in addition to signing its JAR (i.e., a signed message digest that includes the agent's code affords similar protection against modification).

Policy as a Certificate

Policy certificates are used to express the policy that has been set by the relevant authorities. A policy certificate is similar to an attribute certificate insofar as it is issued by a policy-setting authority and conveys assigned rules regarding privileges, which are used to govern the entity being issued the certificate. The policy certificate is an external representation of the policy that pertains to a platform. Depending on the agent system, external representation may need to be translated into an internal form for processing by the policy engine. Assuming that policy certificates can be held and applied directly by a platform, the policy engine needs only to apply the attribute certificates of an incoming agent against the policy certificates held at a platform. Since policy certificates may be unique to a specific context or place on an agent platform, they must be capable of being installed on each platform by the platform authority. In the case of MCT, the platform already relies on a key database, which contains the set of public keys and identity certificates used to authenticate mobile code and ensure that signed code has not been tampered with during transport. To enhance MCT for policy certificate processing, each platform must retain applicable policy certificates in conjunction with the key database, as illustrated in Figure 3.

Because of the similarity in function, the format and structure of the policy certificate closely follow that of the attribute certificate. One significant difference between the attribute and the policy certificate is the binding of the certificate to the entity that is issued the certificate. While an attribute certificate has a clear and singular subject – the agent delegated the privileges – a policy certificate may apply to a broader range of subjects. For example, in the case of a policy certificate issued by a domain authority, it would be desirable to apply the designated privileges to many platforms (i.e., those comprising the domain). A simple solution is to rely on the platform to retain and utilize the prevailing policy certificates that are installed through configuration parameters during initialization. Note that policy certificates may reside elsewhere, other than the platform itself, provided that the location doesn't provide an avenue for attack (e.g., it should be trusted and its communications secure).

Policy Engine

The agent platform must be extended to invoke certificate verification for an agent when it arrives. The policy engine is a new object class whose job is to perform the needed computations and to establish the security context. The security context consists of an allowable set of privileges, representing the intersection of attribute and policy certificates. A null intersection implies that no processing is permitted (i.e., a null security context). The policy engine is made configurable by instantiating a selected policy engine as a trusted component of the platform during the platform's installation. Like many agent systems, MCT has the ability to extend the platform's capabilities in this manner. Moreover, since the policy engine is an instance of a mobile code object class, it should be possible to replace the engine with an updated version at run time. It would also be possible to support more than one policy engine at a platform, if multiple places or application contexts were being supported.

While the policy engine determines policy, it does not enforce it. Enforcement is the responsibility of the mobile code manager, an existing part of the platform responsible for code migration, communications, and other administrative functions. The mobile code manager must restrain the new thread of code by asserting the security context associated with the mobile code of the agent, using features of the Java security manager and class loader.

Related Work

Work is progressing within standardization bodies [Pro98, Enh98] to compliment the X.509 identity-based certificate standards with standards for privilege management. The framework for privilege management generally follows X.509 principles whereby a trusted party, referred to as an authorization authority, issues attribute

certificates to human or machine entities that may in turn delegate that authorization. In addition to the issuing and delegation of privileges via attribute certificates, their revocation is also addressed within the framework. The framework includes definition of the information objects needed for a privilege management infrastructure, including attribute certificates, privilege policy format and attribute certificate revocation list. Work is also being done within the IETF [Far00] to establish an interoperability profile of these standards, intended for generic applications, such as electronic mail, which have limited special-purpose requirements.

The Anchor Toolkit is a mobile agent system that provides for the secure transmission and management of mobile agents [Mud99]. The toolkit protects the agents being dispatched between hosts through encrypted channels. A mobile agent's host platform is required to sign the agent's persistent state before dispatching the agent to the next platform. The signed persistent state can be used later to detect potential problems with the agent's state. The toolkit uses another security tool, called Akenti, developed by the authors to provide access control to a mobile agent's hosts' resources. Akenti uses public/private key signed certificates to express user identity, resource use-conditions, and user attributes. Use-conditions are used to express platform policy, while user attributes typically represent a single privilege granted to the mobile code by some authority. Akenti makes access control decisions for each trusted agent and allows execution only after it authenticates the agents, the server that dispatched the agent, and all the hosts the agent visited in attaining its current state. This scheme relies on a level of trust between mobile agent platforms to make access control decisions in order to mitigate the risk associated with accepting mobile agents.

SESAME is a multi-domain distributed-system security architecture built around the use of authentication and privilege certificates [Ash97]. Both users and applications are controlled in the same way when accessing protected resources - they must first obtain proof of their privileges in the form of a Privilege Attribute Certificate (PAC) and then present it to a target application when requesting resource access. The target application may in turn access another target using the delegated privileges. Access control information is represented in a generic fashion to support mapping to the different types of access controls on targeted resources. SESAME follows a delegation-only model for authorization. PAC revocation is avoided by relying on relatively short delegation periods. While the focus of SESAME is solely on static client-server type applications, it provides a good example of the underlying framework needed when applying certificate-based solutions for distributed-system security.

State Appraisal defines a security mechanism for protection of mobile agents. The goal of State Appraisal is to ensure that an agent has not been somehow subverted due to alterations of its state information [Far96]. Both the author and owner of an agent produce appraisal functions that become part of an agent's code. Appraisal functions are used to determine what privileges to grant an agent, based both on conditional factors and whether the identified state invariants hold. An agent whose state violates an invariant can be granted no privileges, while an agent whose state fails to meet some conditional factors may be granted a restricted set of privileges. When the author and owner each digitally sign an agent, their respective appraisal functions are protected from undetectable modification. One way of looking at this in comparison with attribute certificates is that state appraisal conveys both the policy engine and the prescribed policy internal to the agent. An agent platform uses the functions to verify the correct state of an incoming agent and to determine what privileges the agent can possess during execution. Privileges are issued by a platform based on the results of the appraisal function and the platform's security policy.

Conclusions

Attribute certificates are a flexible way to express the privileges associated with a mobile agent, in accordance with the principle of least privilege. Attribute certificates also provide a natural alternative to using internal data structures commonly found in most agent systems. When combined with the ability for most agents systems to instantiate a policy engine at system initialization, they provide a useful framework that is tailorable to meet the security policy of an application. The degree of tailorability includes the ability to define various policy-setting principals, precedence relationships among them, application specific attributes, and policy processing algorithms.

References

[Ash97] Paul Ashley, "Authorization For a Large Heterogeneous Multi-Domain System," Australian Unix and Open Systems Group National Conference, 1997, pp. 159-169.

[Enh98] Enhanced Management Controls Using Digital Signatures and Attribute Certificates, X9.45-199x, Working Draft, American National Standards Institute X9 Committee, August 25, 1998.

[Con00] Matthew Condell, Charles Lynn, John Zao, "Security Policy Specification Language," Internet Engineering Task Force (IETF) Internet Draft, March 10, 2000, <URL: http://www.ietf.org/internet-drafts/draft-ietf-ipsp-spsl-00.txt>.

[Far96] William Farmer, Joshua Guttman, Vipin Swarup, "Security for Mobile Agents: Authentication and State Appraisal," Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS '96), September 1996, pp. 118-130.

[Far00] Stephen Farrell, Russel Housley, "An Internet Attribute Certificate Profile for Authorization," Internet Engineering Task Force (IETF) Internet Draft, May 2000, <URL: http://www.ietf.org/internet-drafts/draft-ietf-pkix-ac509prof-03.txt>.

[Jan99] Wayne Jansen, Tom Karygiannis, "Mobile Agent Security," National Institutes of Standards and Technology, NIST SP 800-19, August 1999.

[Kar97] Günter Karjoth, Danny B. Lange, and Mitsuru Oshima, "A Security Model for Aglets," IEEE Internet Computing, pp. 68-77, August 1997.

[Koc96] Thomas Koch, Christoph Krell, Bernd Krämer, "Policy Definition Language for Automated Management of Distributed Systems," IEEE Computer Society, June 1996.

[Lan98] Danny Lange, Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, 1998.

[Mob97] Mobile Code Toolkit, Carleton University, May 1997, <URL: http://www.sce.carleton.ca/netmanage/mctoolkit/ >.

[Mud99] Srilekha Mudumbai, Abdeliah Essiari, William Johnston, "Anchor Toolkit: A Secure Mobile Agent System," Proceedings of Mobile Agents '99 Conference, October 1999.

[Pei98] Holger Peine, "Security Concepts and Implementation in the Ara Mobile Agent System," Proceedings of the 7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford University, June 1998.

[Pro98] Proposed Draft Amendment on Certificate Extensions, Collaborative ITU and ISO/IEC Meeting on the Directory, Vancouver, British Columbia, Canada, September 10-12, 1998.

[Ram98] Rajesh Raman, Miron Livny, Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July 28-31, 1998.

[Sur00] N. Suri et ali., "NOMADS: Toward a Strong and Safe Mobile Agent System," Proceedings of the 4th International Conference on Autonomous Agents (Agents 2000) Barcelona, Catalonia, Spain, June 3-7, 2000.

[Tar96] Joseph Tardo, Luis Valente, "Mobile Agent Security and Telescript," Proceedings of IEEE COMPCON '96, Santa Clara, California, pp. 58-63, February 1996, IEEE Computer Society Press.

[Tel95] The Telescript Language Reference, Version 1.0, October 1995, General Magic, Inc., <URL: http://science.gmu.edu/~mchacko/Telescript/docs/telescript.html>.