Determining Privileges of Mobile Agents

Wayne A. Jansen
National Institute of Standards and Technology
Jansen@nist.gov

Abstract

This paper describes a method for controlling the behavior of mobile agent-system entities through the allocation of privileges. Privileges refer to policy rules that govern the access and use of computational resources and services by mobile agents. Our method is based on extending the platform processing environment, using the capabilities present in most mobile agent systems, and applying two forms of privilege management certificates: attribute certificates and policy certificates. Privilege management certificates are digitally signed objects that allow various policy-setting principals to govern the activities of mobile agents through selective privilege assignment. The approach overcomes a number of problems in existing agent systems and provides a means for attaining improved interoperability of agent systems designed and implemented independently by different manufacturers. The paper also describes applying the scheme to Java-based agent systems.

1. Introduction

A mobile agent is a program that executes autonomously over a set of network hosts, on behalf of an individual or organization. An agent moves among hosts to execute parts of its program and to interact with its execution environment and other agents it encounters, in working toward some goal. An agent may also remain fixed on a host for an indefinite period, while conducting its activities. The sequence of hosts that an agent visits may be predetermined when the agent's program is written, or determined dynamically when the agent is launched by its user or as it acquires information at a host. For example, an agent may need to access information at a specific network host, or prefer to execute parts of its program on network hosts offering certain types of services.

Mobile agent computing is a radical form of distributed computing, which poses significant challenges to the security of the agents that form an application and of the hosts on which they execute. One difficult class of threats introduced by mobility is the possibility that the computational environment (i.e., the host and supporting software) may attempt to subvert visiting agents. Solution to this problem is an active area of research. Other threats include agents attacking the computational environment or other agents visiting the host, and outside entities attacking the overall agent framework. A wide range of security techniques, both conventional and newly developed for this paradigm, are available as technical countermeasures against the security threats encountered in deploying agent-based applications [1]. Agent systems typically incorporate some basic countermeasures into their design.

The actions an agent can take at a host are dictated by its privileges, which are controlled through either capabilities or access control authorizations. Capabilities are permissions or access rights assigned to and conveyed with an agent, while access control authorizations are similar, but statically configured at a host. In the most general sense, they can be viewed as policy rules that govern the behavior of an agent at a host. Most agent systems support two categories of privilege: those involving basic host resources such as processing time, memory space, disk storage, network access, and file access, and those involving agent middleware resources, such as cloning, changing itinerary, issuing or subscribing to messages, limiting conversation dialogues, and controlling security services. Agent systems maintain privileges mainly within internal data structures, as opposed to some external form or representation. While these structures are often very similar semantically, they differ in their implementation, in such things as the mechanisms used to protect them from tampering or forgery.

In reviewing a number of agent systems, we noted several shortcomings in the way in which they manage the privileges of agents:

- Among applications, the number of policy-setting principals and the trust relationships that are needed can vary considerably. However, within an agent system those representations are typically fixed and unchangeable. This dichotomy forces developers of agent-based

applications to conform to the imposed scheme, which may or may not match well the intended security policy of their application.

- Policy expression varies among agent systems in terms of granularity, language, and resource entities, and is often difficult for an application developer to modify or extend. When combined with the previous shortcoming, the overall result is to constrain a developer into a rigid framework that may require an elaborate work-around to express the intended policy or, at worst, may be totally inadequate for the needs of the application.
- The means of protecting policy, once expressed and residing in an internal data structure, also varies among agent systems, particularly regarding strength of protection. Each agent system must be closely reviewed to decide whether the expressed policy is satisfactorily protected for the risk environment of the application.
- Because the internal policy-related data structures, trust relationships, policy expression, and strength of policy protection as a whole differ widely among agents developed for different agent systems, the opportunity for interoperability of agent systems is severely limited.

To overcome these noted shortcomings, we devised a method for allocating, managing, and applying security policies in a flexible manner that allows freedom in determining the granularity, language, and entities of the policy expression, as well as the relevant policy-setting principals and their precedence relationships. The remainder of the paper describes our scheme, beginning with an in-depth overview, followed by a detailed look at its use in Java-based agent systems, and then ending with a review of related work.

2. Overview

Mobile agent systems can be implemented in various ways. Interpreted scripting languages or virtual machine-based interpretive language compilers are frequently used for their inherent flexibility in adapting heterogeneous platforms to support agents uniformly. Depending on the agent system, individual agents may be represented as independent processes or lightweight threads. Similarly, the computational environment for an agent may involve a single host computer or multiple hosts. Our method for privilege management applies to a variety of agent systems, despite these kinds of implementation differences. The approach taken also provides a means to work independently of or, if available, in conjunction with a Public Key Infrastructure (PKI), including one built in compliance with the X.509 public key certificate framework [2].

A simple model of an agent system is sufficient for describing the overall scheme. It consists of two main components: the agent and the agent platform. An agent represents the code and state information needed to carry out some computation. The agent platform provides the computational environment in which an agent operates. Multiple agents can interact with one another at an agent platform and use services offered by the platform, such as transport to another agent platform. The platform where an agent is instantiated and commences activity is distinguished as the home platform, and is normally the most trusted environment for an agent. An agent platform comprises one or more hosts and may support multiple places where agents can interact. Figure 1 illustrates the agent and agent platform components along with other components needed for the privilege management enhancements described below.

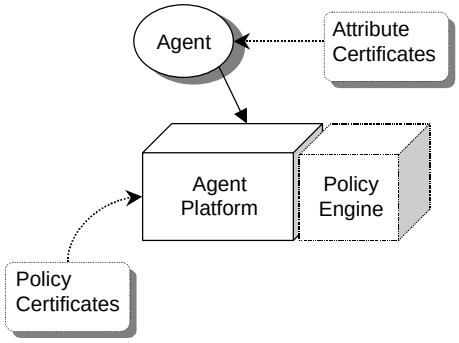

Figure 1: Agent system with enhancements

2.1 Privilege Management Certificates

If mobile agents are to operate on behalf of individuals and organizations, they must follow prescribed security policies established by principals who have the requisite authority. Rather than embody policy rules within an agent, it is possible to push the policy information to an external object – an attribute certificate. Two variants of attribute certificates exist to distinguish those certificates issued to an agent's code from those issued to an instance of an agent (i.e., its code and state information). The distinction is subtle, but important, since for the latter, the certificate includes the values of any instance variables of the agent considered immutable by the agent system. At a minimum, this must include the globally unique identifier of the agent, assigned by the agent system.

To govern an agent's use of computational resources and security mechanisms, the issuer of an attribute certificate assigns the according privileges within the certificate to the agent. 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 as 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. Efforts to standardize the form and content of attribute certificates are ongoing. Their focus, however, has been mainly on stationary communicating programs (e.g., client-server systems) or programs having limited mobility (e.g., applet-like movement from a server to a client) and conveying no state information or computation data.

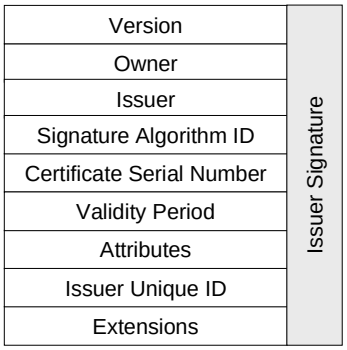

Figure 2: Attribute certificate elements

Policy certificates are counterparts to attribute certificates, but express policy rules assigned to an agent platform instead of an agent. Policy certificates follow the same structure as attribute certificates. While an attribute certificate conveys the policy rules associated with an agent, the policy certificate conveys policy rules governing the behavior of all agents that may attempt to visit an agent platform or a specific place on an agent platform. Policy certificates also convey information about the precedence relationships of policies set by different policy-setting principals, which affect the policy processing of ambiguous or contradictory rules. Since many agent systems take advantage of the security mechanisms provided by the underlying operating system or virtual machine, maintaining the platform policy rules separate from the certificate, in the system access control and authorization files where they normally reside, can be advantageous. Therefore, policy certificates are designed to accommodate such external file references, where needed, by including the location of the file and a cryptographic hash of the file's contents into the policy certificate, for later access and validation during certificate processing.

Although the format and structure of the policy certificate closely follow that of the attribute certificate, one significant difference between them is the binding of the certificate to the entity issued the certificate. While an attribute certificate has a clear and singular subject – the agent assigned the privileges – a policy certificate, in general, may apply to a broader range of subjects than an individual platform. For example, having a policy certificate issued by a domain authority apply to many agent platforms (i.e., those comprising the domain) would be desirable in some situations. Such multi-platform policy allocation can be accomplished through an appropriate choice of an entity name for the owner of the certificate (e.g., a DNS domain name). As with most security policy information, the platform administrator or security officer is relied upon to apply the relevant policy certificates for the platform by setting configuration parameters. Policy certificates may reside elsewhere, other than the platform itself, if the location does not provide an avenue for attack. Policy certificates are validated by an agent platform during its initialization and the policy content can be applied at that time or later, on demand.

Privilege management certificates can be represented in a variety of ways – typically using the Abstract Syntax Notation 1 (ASN.1) Distinguished Encoding Rules. While an ASN.1 encoding does work sufficiently, it has a serious drawback in not being a human readable representation. Moreover, ASN.1 parser tools are neither widely available nor platform independent. To overcome these limitations, we elected to use an eXtended Markup Language (XML) representation for privilege management certificates.

The XML certificate representation closely follows the basic certificate structure in Figure 2. Although the structure of a privilege management certificate is fixed, portions of it were purposely left open to refinement and substitution. The "attributes" and the "extensions" elements are intended to convey policy represented in various types of specification languages and certificate handling information, respectively. The contents of the "attributes" element are determined by a syntax identifier, making it easy to select among different forms of policy representation. The "extensions" element by its very nature supports adding additional elements to a privilege management certificate to meet the needs of an agent system or an application running over it. Convention dictates that each extension element contains a criticality flag and that the processing platform, upon encountering a critical extension it does not recognize, rejects the certificate.

Consider the types of policy expression that might be conveyed within privilege management certificates by examining related efforts in distributed system management. Matchmaker [3] uses a complex form of

policy expression 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 in determining a match. 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 [4]. The PDL supports logical expressions used as the precondition for triggering management actions. The IETF's Security Policy Specification Language [5], 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.

2.2 Policy-Setting Principals

The types of policy-setting principals supported among existing agent systems vary in both number and definition. We discovered, however, that these policy-setting principals could be mapped into one of three distinct classes of principals: branding, using, or hosting classes. Branding principals are those entities involved in attesting for characteristics of the agent's code. A branding principal could be, for example, the manufacturer who develops the code, an evaluator who reviews the code, or an owner who purchases the code. Using principals represent the individual or organization on whose behalf the agent operates, and cause the agent to be launched. Typically, a single using principal exists for each agent, the user of the agent, but multiple users could be involved in situations where concurrence of other individuals is required, such as in some military command and control operations where a two-person rule applies. Hosting principals are those entities having resource authorization control over the agent platform. A hosting principal could be, for example, the system administrator, the system security officer, the owner of the platform, or an authorization authority for the domain in which the platform operates.

With our method, a policy-setting principal is, in effect, any entity that issues privilege management certificates. Therefore, the relationship between policy-setting principals of existing agent systems and those represented under our method must be established through the types of certificates a principal can issue. Specifically, hosting principals issue policy certificates, while branding and using principals issue attribute certificates, which differ only insofar as those issued by the latter convey additional information peculiar to a specific instance of an agent. Certificate assignment is illustrated in Figure 3. Joint signing and issuing of certificates by multiple policy-setting principals is an option not explored here.

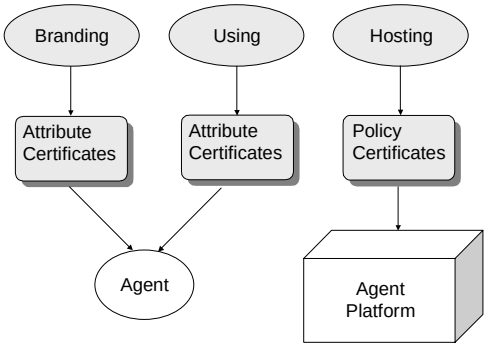

Figure 3: Certificate assignment

As with identity certificates, privilege management certificates may involve chains of delegation, whereby the privileges of an issued certificate are derived from those held by the issuer in the form of a privilege management certificate. That is, besides agents and agent platforms, privilege management certificates can be issued to designated individuals, if needed, who in turn redelegate their assigned privileges to other entities through certificate issuance, forming verifiable authorization chains. Thus, the scheme can support many different styles of privilege authorization and delegation, from a push-style, where a policy-setting principal first gains privileges from an authorization authority before allocating them to an agent, to a pull style where the visited platforms contact an authorization authority to confirm the legitimacy of privileges allocated to an agent by a policy-setting principal. Agent systems, which typically have fixed principals, cannot match the flexibility afforded through privilege management certificates and the ability to chain authorizations.

This is not meant to imply that privilege management certificates should be applied unconditionally. On the contrary, policy-setting principals should be designated judiciously and their certificate issuing applied selectively to minimize policy-processing overhead, yet meet the intended security policy requirements. For example, not all agents need to carry issued certificates, only those needing special privileges for their actions. Similarly, not all three classes of policy-setting principals need be involved in an application, only those having relevance.

2.3 Policy Processing

An agent moves among agent platforms carrying along any issued attribute certificate(s). To simplify processing at a platform, an agent may optionally carry information about the issuer(s) of its attribute certificate(s) (e.g., a user or other policy-setting principal), such as any attribute certificate(s) and identity

certificate(s) held. The policy certificates for a platform must be validated when the platform is initialized and used to establish the platform policy. A platform receiving the agent determines the validity and relevancy of the agent's certificate(s), verifies the issuer's identity, perhaps with the assistance of a PKI, and determines whether the agent's privileges conveyed in its attribute certificates and the platform's prevailing policy established through the policy certificates form a compatible security context for the agent.

The agent platform, which provides the computational environment of an agent system, inherently shoulders responsibility for the processing of policy. This is quite reasonable, 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, policy computation 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 the platform security mechanisms within the computing base. Most agent systems have a means for extending the agent platform with additional program components, for example, in the form of static agents, which is needed for flexible implementation of the policy computation.

Policy computation can be characterized as a policy engine component that in turn is divided into two parts: an outer and an inner policy engine. The operations of the outer policy engine are generic, while those of the inner engine can be tailored to the specific contents of the privilege management certificates. Figure 4 illustrates this characterization.

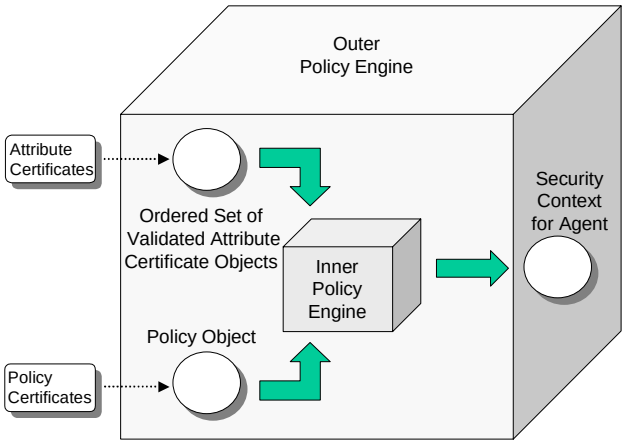

Figure 4: Policy engine organization

The outer policy engine is responsible for parsing and verifying the well-formedness of those certificates associated with an agent, validating the certificates' contents including the signature and any certificate chains, eliminating any certificates not applicable at the platform, and ordering the validated attribute certificates according to the policy precedence hierarchy among issuers. It also fetches the platform policy regarding the agent, established from the policy certificates of the platform at initialization. Validation of certificate chains can be a complicated process involving the possibility of expired or revoked certificates and the necessity to retrieve supporting information. In ordering attribute certificates, the outer engine must ascertain the principals involved, their role in the process (i.e., as an issuer of specific certificate types and variants) and the precedence relationships existing among principals.

The job of the inner engine is to determine the security context for an agent at a platform using the information provided by the outer engine. It does this by, processing the contents of validated attribute certificates against the platform policy and rendering a verdict on whether to allow processing by the agent and under what set of privileges to do so. The two-part organization allows the inner engine to be relatively simple and tailorable to the needs of the application, while the outer engine handles the complex yet common interpretation and validation work.

From this overview, we see that the scheme relies on the placement of policy rules within certificates bound to agents and agent platforms, and on the placement of policy processing capability at an agent platform. This approach provides sufficient flexibility to encompass a very broad range of policies, suitable for most agent-based applications. The benefits of this approach are derived from the kinds of certificates supported and their form and content.

3. Java-Based Agent Systems

Many agent systems rely on the Java programming language and runtime environment for their implementation. While not an agent system itself, Java supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialization, platform heterogeneity, and other features that make it an ideal foundation for an agent system. Java follows a so-called sandbox security model, used to isolate memory and method access, and maintain mutually exclusive execution domains. Java enforces strong type safety using a variety of mechanisms. Static type checking in the form of byte code verification is used to check the safety of downloaded code. Some dynamic checking is also performed during runtime. A distinct name space is maintained for untrusted downloaded code, and linking of references between modules in different name spaces is restricted to public methods. A security manager mediates all accesses to system resources, serving in effect as a reference monitor.

A Java language compiler produces byte codes for an abstract computer called the Java Virtual Machine, which interprets the codes for the host computer on which it executes. More than one Java Virtual Machine (JVM) may be operating simultaneously on a host computer. Typically, a single JVM is used to support the execution environment for multiple agents (e.g., Aglets [6]), each as an independent thread, rather than multiple JVMs (e.g., Nomads [7]). Dynamic class loading and method invocation features of the JVM provide a simple, but effective way to support agent platform extensions.

Another feature supported by Java, is the Java Archive (JAR) file format, which is based on the de facto, standard ZIP archive format and useful for managing collections of Java class files and resources. It is a convenient way for packaging an agent's classes for initial distribution and subsequent movement among visited platforms. The contents of JAR files may also be signed for authentication and integrity protection purposes. Thus, many Java-based mobile agent systems incorporate this format in their design to protect and simplify management of an agent's code. A special password-protected database of private keys and their associated digital certificates, called the key store, is supported by Java and its contents used when signing JAR files.

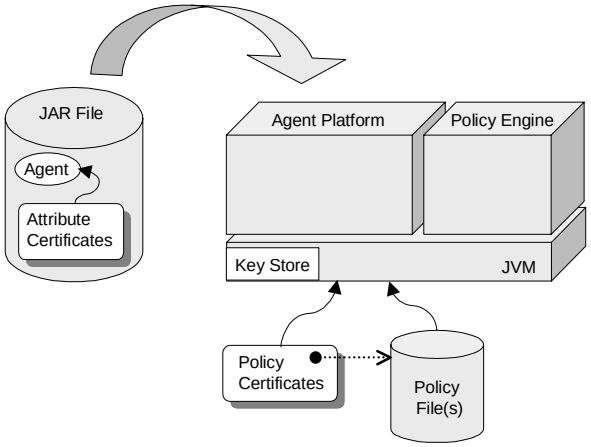

Figure 5: A Java-based agent system with enhancements

Java provides a single system-wide policy file and an optional user policy file, as well as a tool for specifying other policies. Each entry in a policy file indicates the set of permissions authorized for code from a specified code source. Policy rules are expressed using a grant-style policy specification language, whereby all permissions are denied unless explicitly assigned to a code source. Permissions represent authorized actions on system objects. The loader uses the assigned permissions to manage the name space and form a protection domain for any loaded code. Actions attempted by the code are checked against the domain permissions via the security manager. Besides standard Java permissions, developers may also define permissions specific to an application.

Figure 5 illustrates a Java-based mobile agent system and the needed enhancements to enable processing of privilege management certificates. Note that each policy-setting principal, as a certificate issuer, must hold a cryptographic public key pair for certificate signing. In a Java-based agent system, this requirement results in a key store entry for each principal. The sections that follow discuss an implementation of the enhancements in detail.

3.1 Policy Certificates

For any Java-based agent system, the agent platform is a specialized application that runs over the JVM. Rather than inventing a solution for policy specification and enforcement, these systems normally rely on the security policy mechanism afforded by the JVM via the policy authorizations in the standard Java policy file(s). As noted earlier, the simplest way to capture extant policy information residing within system files is to have the issuer of a policy certificate encapsulate them (i.e., the according Java policy files), by reference, within the certificate. Besides encapsulated policy files, the policy certificate conveys additional policy information related to policy-setting principals and permissions conveyed externally with agents. To illustrate the kind of information useful in policy processing, the following features were included in the policy certificate:

- The ability to specify a policy hierarchy based on the class of policy-setting principal,
- The ability to govern certificate occurrence,
- The ability to stratify permissions into mutually exclusive sets corresponding to and controlled by each class of policy-setting principal, and
- The ability for non-hosting principals to both lower and raise privilege.

The policy hierarchy specification was augmented with the ability to set the minimum and maximum occurrences of each type of certificate. This information allows the policy engine to determine whether an agent has sufficient certificates to begin processing and the order in which to apply the policy rules. For example, the hierarchy specification – $\text{Hosting}_1^1 >> \text{Using}_0^1 >> \text{Branding}_1^1$ – indicates that policies issued by hosting principals dominate policies issued by using principals, which in turn dominate policies issued by branding principals. The subscripts and superscripts respectively denote the minimum and maximum certificate occurrences for the class of principal. Therefore, the example specification also indicates that every agent instance may have zero (i.e., the minimum) or one (i.e., the maximum) certificates issued by a using principal, and exactly one certificate issued by a branding principal.

The policy hierarchy specification is conveyed in the "extensions" element of the policy certificate. It would be possible, of course, to introduce finer granularity into the hierarchy specification if required.

To account for hosting principals needing to maintain some control over their computational resources, as opposed to relinquishing them wholesale, hosting principals, by default, dominate other policy-setting principals in the policy hierarchy. However, hosting principals can perform selective allocation of privilege adjustment to other policy-setting principals through a sparse authorization matrix (principals x permissions) within the "attributes" element of the policy certificate. Each class of policy-setting principal can be granted rights to raise or lower an indicated permission. If that right is withheld, any unauthorized attempts to adjust the permission are ignored and a security notification issued. The approach is flexible and allows the stratification of privileges into mutually exclusive sets for each class of policy-setting principle to control. For example, using principals may be limited to controlling features of the agent system, while branding principals, such as manufacturers, may be limited to certain virtual machine resources.

The current version of Java is designed with a number of features that allow controlled modification and extension to the runtime environment. They include the ability to define new security properties, to specify a replacement class for the standard policy class, to define new permissions, and to place trusted code in a directory where it is treated as part of the virtual machine for class loading and operations. These features were used in implementing our privilege management scheme. By replacing the standard policy class, a new policy certificate aware handler can be instantiated during initialization of the JVM. By defining new security properties, the handler is able to locate, validate, and translate the appropriate policy certificates into an internal form suitable for processing by the policy engine. By defining a new permission, the ability to adjustment permissions can be controlled through a standard policy entry (see discussion below). Finally, by locating privilege management components within the virtual machine directory for trusted extensions, they obtain complete access to system level resources.

Java policy is by nature platform-centric. Standard policy rules do not take into account any policy-setting principals except those associated with the platform, namely the system administrator and home user, which are often synonymous. Under the grant-styled policy mechanism of Java, the most direct means of having external policy rules associated with an agent accepted and incorporated at a platform is to define a permission that allows the granting of those external permissions. Such a privilege-adjustment permission allows a platform authority to control the privilege not only with respect to a code source, but also with respect to a specific set of policy-setting principals who issue one or the other variant of an attribute certificate. The information conveyed by privilege-adjustment permissions complements the more detailed information within the policy certificate regarding the specific permissions a class of policy-setting authority can adjust. One analogy is that granting a privilege-adjustment permission to an agent opens the doorway to the room where specific permission adjustments may occur.

The form of the privilege-granting permission we used begins with its name, "privilegeAdjustment," followed by the key store aliases of the permitted certificate issuers (may be any, represented by "*"), and completed by either of the actions, "sealedBy" for branding principals or "launchedBy" for using principals. In simple terms, the permission grants an agent's code source the right to gain the privileges expressed within an attribute certificate issued by some policy-setting principal to either the agent or an instance of the agent. For example, to permit any agent's code base, sealed by a trusted reviewer (i.e., the enterprise security officer (ESO)) and launched by any trusted user (i.e., one having an entry in the key store), to adjust its platform privileges (i.e., be accepted for privilege adjustment processing), the Java policy rule would be

```
Grant {
    Permission privilegeAdjustment "*"
                                   "launchedBy";
    Permission privilegeAdjustment "ESO"
                                   "sealedBy";
    };
```

Thus, the standard Java policy mechanisms can be extended in a manner suitable for meeting the security policy requirements of most agent systems. Implementing the scheme as described, does not affect the syntax or structure of Java policy files, which remain the primary means for expressing platform policy. Instead of replacing policy files, policy certificates incorporate their contents through reference. This approach allows policy certificates to be issued to any standard Java policy file by a policy-setting principal in the same manner as attribute certificates are issued to an agent – via a cryptographic hash of the contents of the policy file.

3.2 Attribute Certificates

An attribute certificate is an external XML representation of the policy rules assigned to an agent. For a Java environment, the policy is represented as standard Java permissions conveyed within the "attributes" element of the attribute certificate. The responsibility for maintaining relevant attribute certificates with an agent as it moves among platforms

falls to the agent system. Java-based mobile agent systems usually allow movement of mobile code as either individual class files or a JAR file. Because JAR files are the prescribed means within the Java framework for signing and verifying code, most security-conscious designers incorporate them into the agent system. In addition to the archived code, a signed 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. These files are maintained in a special directory – the META-INF directory. Additional meta-information, such as the identity certificates of the entity that signed the code, may also be included within the META-INF directory to simplify the verification processing of the JAR contents by a recipient.

Because of its flexibility for conveying meta-information, an agent's JAR file also makes a suitable container for attribute certificates issued to the agent. Once the agent's code resides within the JAR, it can be cryptographically bound to a certificate, and the certificate placed within the META-INF directory for subsequent use. Multiple certificates can be accommodated to support policies involving multiple policy-setting principals. Not confusing the standard Java security features regarding signed JAR files with those of attribute certificates is important. In principle, they are distinct and can be applied either individually or jointly. The described JAR extensions follow this principle. However, some redundancy exists in situations where a branding principal issues an attribute certificate for an agent in addition to signing its JAR, since the certificate's message digest over the agent's code affords similar protection. One advantage of using attribute certificates is that their expiration date can be set much shorter than the validity period of the signing key, effectively enabling the lifetime of the authorization to be limited to an appropriate period.

As with the policy certificate, other useful information can appear in the "extensions" portion of an attribute certificate. They include a constraint indicator to determine whether the entity issued the certificate is a terminal policy-setting principal or an intermediate one able to reassign privileges for a designated number of decedents, and a renewal service location to determine where an expiring attribute certificate can be extended and for what duration and number of times. In the case where an attribute certificate is issued to an instance of an agent, the "extensions" portion of the attribute certificate is used to convey values of immutable instance variables and, thus, protect them from tampering via the certificate's signature. Recall that the globally unique identifier of an agent must be treated this way.

3.3 Policy Engine

Besides verifying the prevailing policy certificates and the binding to their associated policy file, the agent platform must be extended to invoke attribute certificate processing for an agent when it arrives. The policy engine is a pair of new object classes whose job is to perform the needed computations and determine the allowable privileges for the agent's code. The privileges consist of an authorized set of Java permissions, representing the amalgamation of policy rules within the attribute and policy certificates. Null privileges imply that no processing is permitted (i.e., no permissions granted). The policy engine classes can be located and loaded by the agent platform as a trusted component at initialization time, through an entry in either the standard Java security properties file or the properties file of the agent system. Since the inner and outer parts of the policy engine are Java object classes, their instantiation is straightforward. It would, therefore, be possible to support more than one policy engine at a platform, if support for multiple places or application contexts were needed. An optional but critical extension, the policy identifier, is defined within the "extensions" element of all privilege management certificates specifically for use in matching certificates to a policy engine.

The outer policy engine supports a single method, called to verify the relevant attribute certificates and return the allowable set of privileges. As explained earlier, the outer policy engine invokes the inner engine once all the certificates are evaluated and ordered. Since the replacement policy class keeps intact all standard Java policy mechanisms, the policy engine is able to rely on them during processing. This simplifies the implementation of the policy engine. For example, when the policy engine needs to determine the baseline permissions assigned to a particular protection domain, it can use the standard Java Application Programming Interface (API) to ask the currently installed Java policy object for that information. Similarly, through an existing API, it can determine whether a permission asserted in an attribute certificate is implied by those baseline permissions. For a grant-style policy mechanism, the policy engine computation essentially becomes the logical union of the baseline permissions with any permissions conveyed in the attribute certificates, subject to the precedence of policy-setting principals who issued the certificates and any privilege adjustment constraints imposed by the prevailing platform policy.

While the policy engine determines allowable privileges, it does not enforce them. Enforcement is the responsibility of the program component of the agent platform that is also responsible for code migration and other administrative functions. This component must be augmented to restrain the agent's code by asserting the

associated security permissions within a protection domain using features of the Java security class loader.

4. Related Work

Work has progressed within standardization bodies [2, 8] to compliment the original X.509 identity-based certificate standards with standards for privilege management. The framework for privilege management generally follows X.509 principles by which a trusted party, called 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 [9] to establish an interoperability profile of these standards, intended for generic applications, such as electronic mail, involved in client-server types of transactions.

The Anchor Toolkit is a mobile agent system that provides for the secure transmission and management of mobile agents [10]. 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 the resources of a mobile agent's host platform. 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 [11]. 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 generically to facilitate 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 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.

Nikander and Partanen [12] describe a method for enhancing the Java language environment with policy expression and processing via Simple Public Key Infrastructure (SPKI) certificates. SPKI certificates are a proposed alternative to using X.509 certificates, emphasizing key, rather than individual, identities. The motivation for the enhancement was to make it possible to distribute Java security policy management fully in a way that does not affect the local configuration. They accomplished this by assigning permissions to class files bundled within a JAR file, using SPKI certificates in a manner similar to our use of attribute certificates. No counterpart to policy certificates appears in their method, however. The approach essentially goes from one extreme (i.e., platform-centric policy specification) to another (i.e., a code-centric policy specification), and relies completely on the contents of validated certificate chains for determining the permissions for a protection domain of a given class. As with SESAME, the focus here is on client-server type applications. However, because of the method's flexibility and grounding in Java, it could be extended for use in Java-based agent systems.

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 [13]. 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 to 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.

5. Summary

Attribute certificates are a convenient way to express the privileges associated with a mobile agent, in accordance with the principle of least privilege. Attribute certificates also provide a flexible alternative to using fixed predefined policy structures commonly found in most agent systems. When combined with policy certificates for the agent platform and the ability for most agents systems to extend the agent platform with a policy engine, they collectively form 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 the policy set by those principals, application specific attributes, and policy processing algorithms.

6. References

[1] Wayne Jansen, "Countermeasures for Mobile Agent Security," *Computer Communications*, 23 (2000), Elsevier, October 2000, pp. 1667-1676.

[2] ITU-T Recommendation X.509 | ISO/IEC 9594-8: *Information Technology - Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks*, March 2000.

[3] Rajesh Raman, Miron Livny, Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 1998.

[4] Thomas Koch, Christoph Krell, Bernd Krämer, "Policy Definition Language for Automated Management of Distributed Systems," *Proceedings of the Second International Workshop on Systems Management*, IEEE Computer Society, June 1996.

[5] Matthew Condell, Charles Lynn, John Zao, "Security Policy Specification Language," Internet Engineering Task Force (IETF) Internet Draft, March 10, 2000.

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

[7] Niranjan Suri et ali., "NOMADS: Toward a Strong and Safe Mobile Agent System," *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, June 3-7, 2000.

[8] American Bankers Association, *Enhanced Management Controls Using Digital Signatures and Attribute Certificates*, ANS X9.45-1999, American National Standards Institute (ANSI) X9 Committee, February 11, 1999.

[9] Stephen Farrell, Russel Housley, "An Internet Attribute Certificate Profile for Authorization," Internet Engineering Task Force (IETF) Internet Draft, June 2001.

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

[11] Paul Ashley, "Authorization for a Large Heterogeneous Multi-Domain System," *Proceedings of the Australian Unix and Open Systems Group National Conference*, 1997, pp. 159-169.

[12] Pekka Niklander, Jonna Partanen, "Distributed Policy Management for JDK 1.2," *Proceedings of the 1999 Network and Distributed Systems Security Symposium*, February 1999, pp. 91-102.

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