ENGINEERING ACCESS CONTROL IN DISTRIBUTED APPLICATIONS

Konstantin Beznosov
School of Computer Science
Florida International University
Miami, FL, 33199
E-mail: beznosov@cs.fiu.edu

Yi Deng[1]
Department of Computer Science
School of Engineering and Computer Science
University of Texas at Dallas
Richardson, TX 75083
E-Mail: yideng@utdallas.edu

[1] Contacting author

ABSTRACT

This paper discusses issues of engineering access control solutions in distributed applications for enterprise computing environments. It reviews application-level access control available in existing middleware technologies, discusses open problems in these technologies, and surveys research efforts to address the problems.

Keywords: Software security, access control, authorization, distributed systems, software engineering, middleware.

1. INTRODUCTION

Security of computer systems is conventionally achieved via *protection* and *assurance*. The former is usually provided by some security subsystems or mechanisms, which are designed to protect the system from specific threats. A threat is any potential occurrence that can have an undesirable effect on the assets and resources associated with a computer system. Protection is based on the premise that it is possible to list most of the threats which can happen, and to build mechanisms that can prevent the threats [13]. The protection mechanisms can be classified in to three groups: accountability, availability and authorization. Accountability mechanisms make sure that users and other system active entities (conventionally called *subjects*) are held accountable for their actions towards the system resources and services. Availability mechanisms ensure either service continuity or service and resource recovery after interruption. Authorization mechanisms ensure that the rules governing the use of system resources and services are enforced. The mechanisms are further qualified as either access control or data protection ones.

Access control mechanisms allow system owner to enforce those rules when rules check and enforcement are possible. The term "authorization" also implies the process of making access control decisions. When checking and enforcement of the rules are not possible, data protection mechanisms, such as data encryption, are used. A reference monitor is a part of the security subsystem, responsible for enforcing the rules via mediation of access by subjects to system resources (traditionally called *objects*).

Access control has been exercised at different places and levels of abstraction, e.g. network, database, operating system and middleware controls, each with different emphasis. Control to protected resources can also be addressed from a single system or an organization point of view. The objective of this paper, however, is to survey the design of access control mechanisms from an organization and application point of view. Modern information systems are increasingly interconnected to form information enterprise, which consists of many self-contained, heterogeneous and yet integrated application systems. The problem of access control in such an environment is to enforce organization-wide security policies across these applications. How to ensure the secure interoperation of interconnected software systems in an information enterprise is a complex task and emerges as a central issue in software development and operation. In this paper, we survey the issues, problems and solutions in engineering software to handle access control for enterprise-oriented application systems, and discuss future solutions and technology in this important area.

The rest of the paper is organized as follows. Section 2 provides background information on access control in application systems and defines evaluation criteria. We discuss the issues of access control in distributed systems in Section 3. Section 4 gives background information about access control in software systems. Mechanisms available in various distributed security technologies are described in Section 5. Section 6 overviews approaches on application-level access control reported in research literature. Conclusions are provided in Section 7.

2. BACKGROUND AND EVALUATION CRITERIA

In this section, we give background information on access control, explain main concepts and terms, and then introduce the issue of application-level access control. In addition, we define criteria for evaluating existing technologies and research.

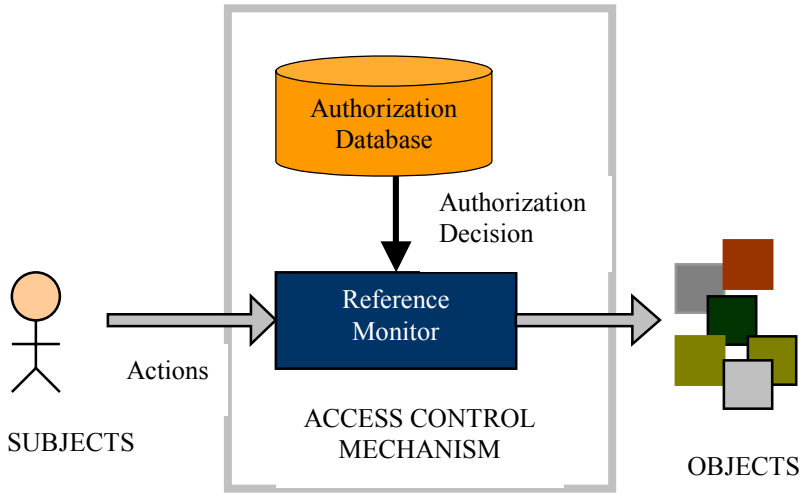

FIGURE 1. CONVENTIONAL ACCESS CONTROL.

2.1. BACKGROUND

The structure of traditional access control (AC) mechanisms can be viewed using the conceptual model of *reference monitor*. A reference monitor is a part of the security subsystem, responsible for mediating access by subjects to system resources (traditionally called *objects*), as illustrated in Figure 1.

The mediation consists of making authorization decisions, by checking access requests against authorization rules from the authorization database -- a storage of such rules -- and enforcing them. A set of the rules is sometimes called a *policy*. Authorization rules commonly have a subject-action-object structure, which specifies what subject(s) can perform what action(s) on what object(s). Permitted actions are called *access rights*. Thus a subject has a particular access right to an object if it can perform the action towards that object. Furthermore, all authorization rules can be conceptualized into *access matrix* [39], where a row corresponds to a subject and a column to an object, and each cell specifies access rights granted to the subject for the corresponding object.

To make an authorization decision, a reference monitor takes authorization rules and three groups of information: 1) the access request, 2) the subject who made the request, and 3) the object to be accessed.

The information about access request usually carries the request type, for example “read”. However, some application domains have a need for AC decisions based on additional attributes of the request. For instance, a banking system might deny a withdrawal request if its amount exceeds a pre-determined threshold.

Information about the subject can be divided in two types -- related or unrelated to security. Originally, only security-related information was used in AC decisions. Controlled by security or user administrators, this information describes subject’s identity, group membership, clearance, and other

security attributes. Some times, the term *privilege attributes* is used to refer to those security attributes intended solely for the purpose of AC.

In some application domains, security-unrelated information about the subject needs to be taken into account. For example, access to rated materials in public libraries could be granted according to a person's age. Another example is information derived from organizational work-flow process. This information is not controlled by security or user administrators and it is not always provided to the reference monitor in the form of subject security attributes. The information about the object to be accessed can also be divided into categories related and unrelated to security. An example of an object security attribute is its security level.

Depending on the capabilities of a particular AC mechanism and the availability of information about the subject, request and object, either limited or elaborate information are available for making authorization decisions. This information availability will be used as a criterion for evaluating expressiveness (or power) of AC mechanisms.

AC mechanisms are part of most operating, database management (DBM) and middleware systems. They are also present in such control systems as firewalls, and many applications.

Application resources can be in the form of data processed by applications, their services (e.g. Telnet, SMTP or WWW servers), particular operations performed on them (e.g. GET access requested from a WWW server via HTTP protocol, operation invocation on a CORBA-based application server), or even menus of the application interface.

Some application resources, such as files, database records or network sockets, can be protected by an operating system, DBMS, or middleware system. However, there are resources that are application-specific and are recognized only by the application itself, for example the execution of particular parts of the application business logic. The granularity of application-specific resources is finer than of general-purpose computing systems. This is one essential distinction between application-level and general purpose AC. Another vital difference is that authorization rules used for application-level AC require the use of such information about access operations, subjects, or objects, that is specific to the application domain or more elaborate (more expressive) than the information used by AC mechanisms of general purpose systems. In order to meet the requirements, applications commonly have their own AC mechanisms in addition to the use of those provided by the underlying general purpose systems. And this practice is increasingly common.

2.2. Evaluation Criteria

There are a wide variety of technologies and techniques for handling AC, which differ from each other in various ways. To make meaningful comparisons, we define the following evaluation framework.

1. **Granularity of protected resources.** We will use the following granularity hierarchy: application, interface, method, arbitrary resource. If a solution does not allow authorization decisions on fine-grain resources, it cannot be used for protecting application resources.
2. **Support for policies specific to an organization or application domain.** There is a wide range of AC models and policies, as shown in Chapter 3. At one end are AC mechanisms that support only one model (and the corresponding policies), for example lattice-based mandatory AC (MAC). At the other end are solutions that allow implementation of any authorization logic and their support for policies is limited only by the interface to the logic. In general, the more AC policy types a mechanism can support the easier it is to configure for required organizational policies. We will look at the range of supported AC models.

3. **Information used for making authorization decisions.** As we discussed above, authorization decisions are made based on information about the subject and object, as well as the operations to be performed. Some technologies allow obtaining only authenticated identity of the subject but not the information about group membership or activated roles, which ultimately limits the functional capabilities of the AC mechanism based on such a technology. We will look into what information is available and what information is used in authorization decisions.
4. **Use of application-specific information.** The use of information that is application-specific and becomes available only while the application processes the client request is critical for some application domains (e.g. health care). If a solution does not allow the use of such information, full automation of protecting application resources would not be possible.
5. **Support for consistency of policies across multiple applications.** It was discussed earlier that in the enterprise environment, the issue of consistent policy enforcement is a critical one. We will consider the support for enterprise-wide consistent AC policy enforcement while examining the available and proposed approaches.
6. **Support for changes.** No matter how functionally perfect the support for the AC is, if it is ineffective to accommodate changes, such as insertion or deletion of application systems and underlying computation environment, it is not suitable in enterprise settings. Most available approaches support the changes to some degree. Unfortunately, there is no objective quantitative criteria for determining the level of support. We will evaluate the level of support by comparing the solutions.
7. **Solution scalability.** Performance and administration scalability highly affects the utility of AC solutions. If a technology does not scale well it can not be more then just an academic exercise. Since there is not any benchmark available for evaluating the scalability of AC solutions, we will use common knowledge to reason about the scalability. For instance, we will examine the amount of data that needs to be modified, in order to accommodate a policy change. Another commonly known measure that we will use is the communication complexity, which is still regarded as the major factor in the performance of distributed systems.

3. PROBLEMS OF ACCESS CONTROL IN DISTRIBUTED SYSTEMS

From software engineering point of view, design and implementation of software security in distributed enterprise environment must address several concerns. First, an information enterprise consists of many self-contained and yet interconnected or integrated application systems. The problem of access control is to enforce organization-wide security policies across these applications. Consequently, ways must be provided to ensure the consistency of enforcement across application boundaries. Second, changes occur frequently in enterprise environment. These may include changes to existing systems, insertion or deletion of applications, changes in business processes and security policies, changes in hardware/software platforms, etc. An attractive security system design must effectively support the evolution of enterprise systems. Last but certainly not least, the above qualities need to be achieved at reasonable cost during the development, operation, and evolution of application systems.

The industry has achieved considerable results on access control of operating system, databases and middleware in such a way to make the security mechanisms as relatively independent and self-contained components in the systems. Most of operating systems implement authorization logic in the security part of their kernels [9, 18, 19, 23, 25, 30, 31, 35, 42, 44, 45,48, 53, 54, 60, 64]. There are also special-purpose ad-on security software packages that furnish authorization decisions for operating systems [9, 15, 16, 32]. Abadi et al. [1] and Lampson et al [39] developed a unified theory of authentication and access control in distributed systems. Practical implementations reflecting some results of the theory

have been implemented in security architectures of some distributed environments [38, 47, 50, 51, 57]. However, fine-grain control of application resources, except when those resources are stored in a consolidated database, is done traditionally in ad hoc manner [65], and there are no automated means to ensure enterprise-wide consistency of such controls.

Generally speaking, current solutions to access control have two problems. First, access to protected resources is controlled at several points. They are network controls (e.g. firewall), middleware controls (access control mechanisms enforced by middleware environments such as CORBA [50], EJB [43], DCE [22], DCOM [46]), database and operating system controls. Making all these controls to work in concert and consistently enforce enterprise-wide access control policies is a daunting task, when there are hundreds of application and supporting systems (e.g. operating systems).

The second problem is that traditional access control mechanisms [61] provide limited capabilities for handling complex policies and authorization decisions that are based on factors specific to an application domain. The complexity of access control policies in some application areas, e.g. health care, requires exercising access control policies that are more sophisticated and of finer granularity than the general ones used in operating systems, databases, and security services of such distributed environments as Java [38], DCOM, DCE, SESAME [52], and CORBA. All but last types of access control points provide only coarse grain level of control. In addition, security requirements in some domains mandate domain-specific factors (e.g. relationship between the user and the patient [8], emergency context) to be used in access control policies. This complexity and granularity level often force application designers to embed domain-specific authorization logic inside their applications. Some even document patterns of designing "application security" [69]. As a result, it both increases the complexity of software design and makes it difficult to ensure system integrity and quality. It also significantly increases the difficulty and cost in system administration and management, especially in the face of rapidly changing business, technology, as well as system requirements and functionality. These problems are less severe in stand-alone systems. However, the distributed environments of enterprise systems with dozens of heterogeneous systems exacerbate them drastically.

4. DESIGN APPROACHES TO APPLICATION-LEVEL ACCESS CONTROL

The research community has been working towards systematic ways of controlling access to resources in distributed and heterogeneous application systems. The three main research approaches are policy agents, interface proxies and interceptors, and enterprise-wide authorization services. They are discussed in this section.

4.1. POLICY AGENTS

The concept of policy agents [29] is motivated mainly by the goal of accommodating the existing body of products and technologies already deployed in organizations. The key idea is centralized AC management via rule translation into languages supported by local AC mechanisms, and distribution of the rules across the systems, as shown in Figure 2. The distribution is achieved with the help of policy agents that reside on computers hosting applications. They could be the OS AC or add-on packages, AC provided by the middleware, by DBMS security layers, or even by the AC mechanisms integrated in the application. Consistency of authorization policies across application boundaries is achieved by the centralized AC management via translation of authorization rules into languages supported by local mechanisms through policy agents. The distributed management architecture based on such agents provides the infrastructure necessary to map domain-wide authorization rules into rules specific to particular mechanisms.

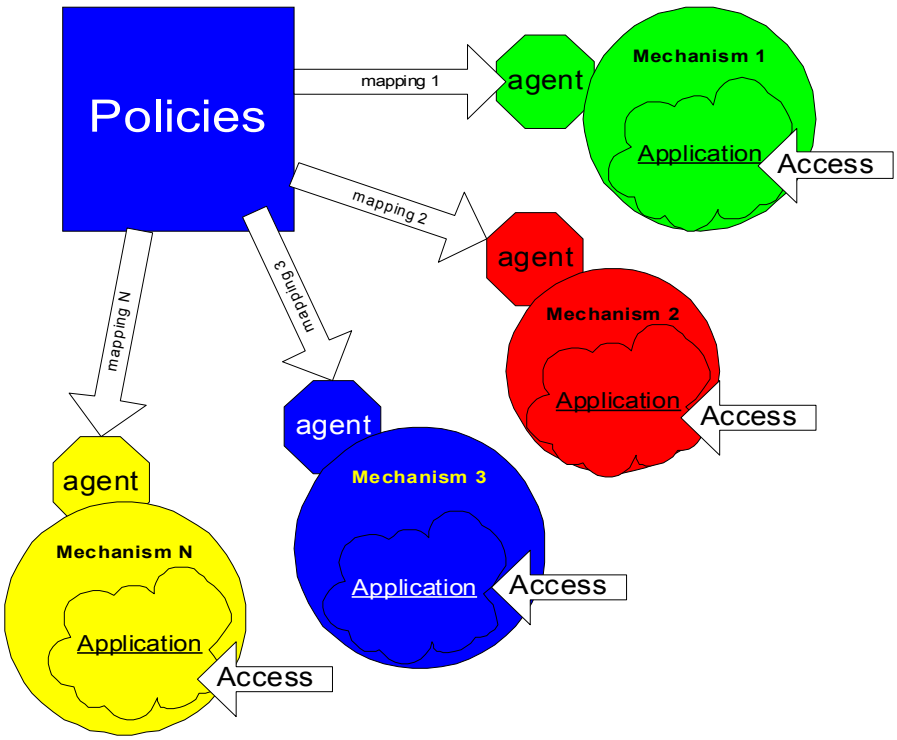

FIGURE 2. POLICY AGENTS

This approach have several advantages, including fault tolerance, compartmentalization of enterprise security without penalizing run-time performance. There is a high degree of run time autonomy between the local mechanisms.

The main problems of this approach are the difficulties in achieving the consistency of enforced global policy across different applications, and in automating the mapping of the global policy into the languages and representations specific to various AC mechanisms. This approach also suffers from some innate limitations. First, the granularity and expressiveness of domain-wide AC policies can be only as good as the policies supported by the most coarse-grain and least expressive mechanism in that domain. Second, the distribution of policy updates can be very slow, which could easily make policies based on periodic authorizations [10] unaffordable. However, this direction becomes irreplaceable if other approaches, such as proxies and interceptors, fail.

4.2. PROXIES AND INTERCEPTORS

Solutions under this approach employ either interface proxies [28], role classes [8], security meta-objects [55,56], or interceptors of intersystem communications, as in CORBA, DCOM, SafeBots [20, 21] and Legion system [26, 27, 67]. The idea is illustrated in Figure 3. Access to an application is controlled externally and authorization decisions are made and enforced before an application gains control and/or after it dispatches an invocation to another system. Invocations are intercepted either in the communication, middleware, or application layers.

No changes to the application system are required in these solutions, which is their main advantage. The reference monitor is implemented externally to the application system. Security developers can control the behavior and size of the monitor. This makes it a good alternative to policy agents in controlling access to resources of already deployed applications. Moreover, if an existing application

does not have any AC mechanism, proxies and interceptors become the only choice. Another advantage is the ability to make all the decisions locally to an application system, which improves performance scalability.

However, it is difficult to ensure the consistency of enforced policy and authorization data coherence because there are as many instances of access control proxies as applications. There are other significant limitations. First, the granularity of access control can not be finer than (object) method and its arguments. Second, the decisions must always be made either before or after an application controls invocation execution. Third, variables, whose values become available at some point after the method is invoked but before an AC decision is made, can not be used.

4.3. AUTHORIZATION SERVERS

When an application enforces its own access control policies, a reference monitor is embedded in the application. In this case, if an access control decision is performed by a dedicated authorization server external to the application system, the reference monitor spans the application and the authorization server. This allows decoupling of the reference monitor decision part from an application without losing the capability for an application to define its own space of protected resources and its semantics. This is the main idea behind solutions employing authorization services [63, 66, 67, 70].

The effort of developing a generalized framework for access control (GFAC) [2, 3, 4, 5] at the MITRE Corporation is an early representative of this approach. The project endeavored to build a theoretical framework that explicitly recognizes the main information components for access control -- subject and object (security-related) attributes, access context, authorities, and rules. They showed that "the rules for access control are an entity that is separate from, although necessarily related to, the model of the TCB (trusted computing base) interface" [40]. Moreover, La Padula suggests [40] that in a networking environment "one can conceive of an access control engine realized as a server, with access requests handled via a remote procedure call mechanism."

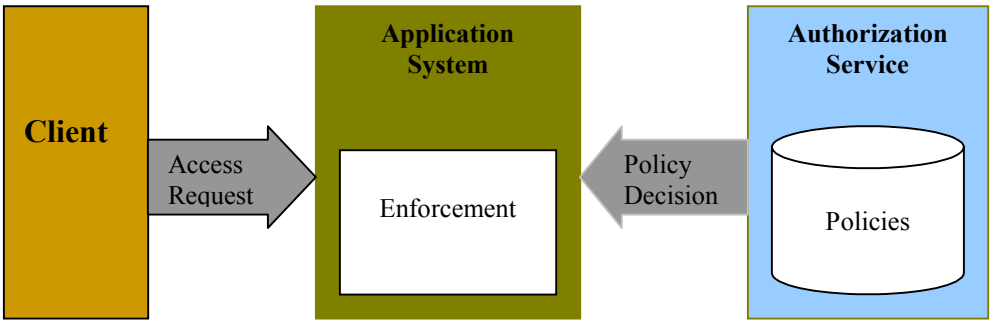

Figure 3. Authorization Services

The main advantages of this approach include:

- Logical centralization of access control rules, which supports inherent consistency and coherence of authorization policies enforced throughout a policy domain.
- Ease of policy change and update because authorizations are made at a (logically) single place.
- Since authorization logic is centralized and decoupled from the application logic, it is possible to change policy type without affecting application systems.

This approach can also significantly reduce the cost of access control administration. This is made possible by the centralized administration of authorization rules for all systems belonging to one policy domain.

Since an application system determines when to obtain an authorization decision from the server, it can do so right when such a decision is needed. The authorization service is to make authorization decisions for access to the application resources that can not be protected by middleware AC mechanisms, as shown in Figure 3. Authorization decisions on resources of any granularity level can be obtained from the server because an application uses the server while it is processing a request, as shown in Figure 3. This lifts the limitation of the Proxies and Interceptors.

For this approach to be feasible, several issues must be addressed. First, it is much more challenging to design an authroization server so that it does not become a bottleneck in terms of performance. Second, if the server fails, all application systems served by it will have to resort to a simplistic and very limiting policy such as "always deny" or "always grant." This would render systems unoperational. Thus the provision of high fault-tolerance degree needs to accompany such servers.

5. APPLICATION ACCESS CONTROL IN MIDDLEWARE

In this section, we review access control mechanisms provided by the main-stream middleware technologies, and discuss how these technologies can be used by distributed application systems, as well as what their limitations are.

5.1. JAVA AUTHENTICATION AND AUTHORIZATION SERVICE

The release of Java 2 platform introduced a new security architecture [24] that uses a security policy to decide the granting of individual access permissions to running code. The Java Authentication and Authorization Service (JAAS) [38] is designed to provide a framework and standard API for authenticating and assigning privileges to users. Together with Java 2, an application can provide access control based on the attributes of code and its source (code-source-centric), or user (user-centric), or a combination of both. In JAAS, access control is enforced using proxies approach only on system resources, such as files, sockets, etc., whereas Java objects constituting an application system are not protected. Access control to these objects has to be implemented by application itself. Since JAAS specifies a flexible mechanism for defining application-specific protected resources of any granularity, an application can reuse these mechanisms for determining permissions granted to an invoking subject using JAAS decision logic as an authorization service. This would allow having a uniform interface to the authorization logic. JAAS also has a generic and extensible support for different authorization factors. Privilege attributes in JAAS need not be predefined.

New attributes can be easily defined via new classes, and this could introduce confusion because even semantically the same attributes are considered dissimilar by JAAS if they are implemented as different classes. A notion of attribute type, like in CORBA or SESAME, would address the problem.

JAAS supports authorization decisions based on the source code origination, the identity of the code signer, the value of a privilege attribute possessed by a subject. These are all used to determine permissions for a particular resource. An application system can retrieve permissions from Policy object and compare them with the required ones. In order for an application to enforce its own AC, it needs to determine the resource name and the required permissions. Afterwards, it would obtain from Policy permissions granted to the subject in question and make comparison for the final verdict. This is not much different from other middleware technologies such as CORBA, DCE, SESAME and DCOM. The flexibility comes from the way AC policies can be specified in JAAS. Since Policy is just an interface,

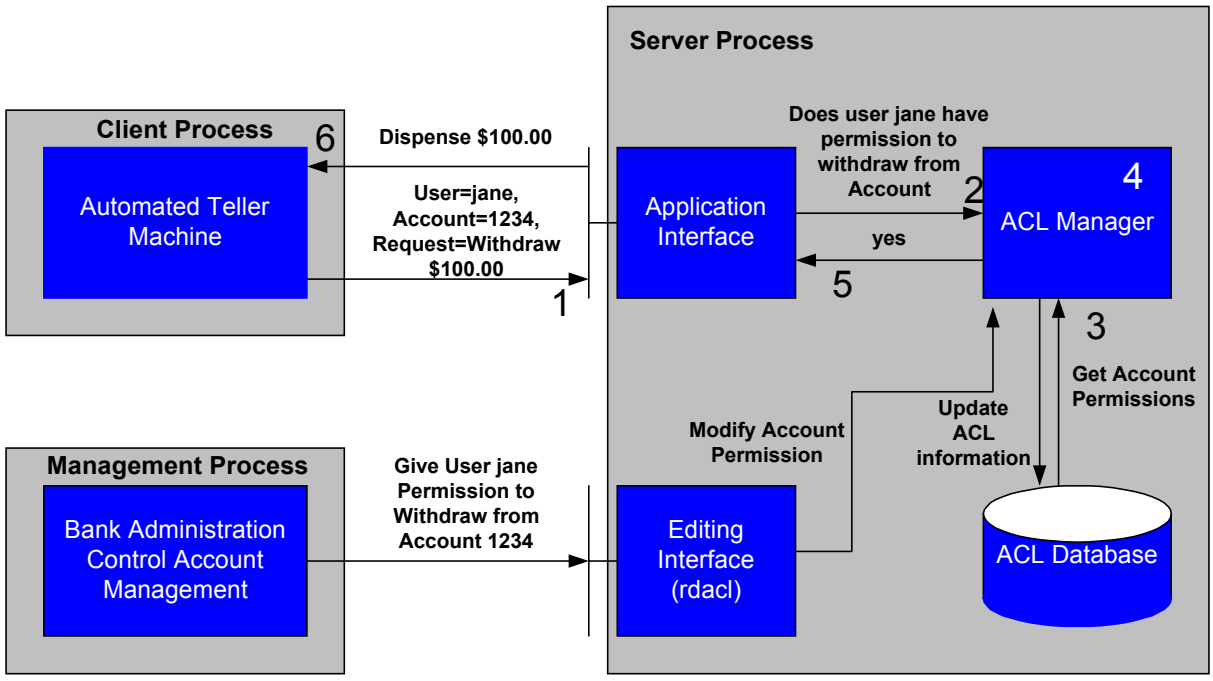

FIGURE 4. AUTHORIZATION PROCESS AND ACL MANAGEMENT IN DCE-BASED APPLICATION [[17]SYSTEMS (FROM [37])

behind which almost any functionality could be implemented, JAAS does not constrain implementers of AC policies to any particular mechanism.

5.2. DISTRIBUTED COMPUTING ENVIRONMENT

In the Open Software Foundation's (OSF) Distributed Computing Environment (DCE) [37], the service does not control access to applications or their resources. DCE applications are expected to enforce and provide administrative access to authorization policies on their own. To do so, an application has to implement AC functionality, including an access control list (ACL) manager and ACL storage, as shown in Figure 4.

In order for an application to use DCE security service for AC, it needs 1) to determine the DCE object ID (OID) of the resource in question, and 2) to obtain authorization decision from its ACL manager using the OID.

If an application uses the DCE ACL model for authorization, it associates an ACL with a protected resource identified by the OIDs, which are used by the ACL manager to determine right ACLs. The exact definition of "resource" is entirely at the discretion of the application. For example, an object could be an item of stored data (such as a file), or could be a purely computational operation (such as matrix inversion). Thus, the concept of OIDs enables any granularity of protected resources.

DCE ACLs support a limited number of privilege attribute types -- identities of the user, who is the resource owner, the owner group, and other group(s). DCE ACL language is also considerably limited allowing security administrators to either explicitly grant or deny rights to the subject based only on its identity or group attributes. The language capability to support policies specific to application or organization remains to be seen.

The following simple example from [17] illustrates (Figure 4) how access control is expected to be implemented by an application system. User *jane* makes a request to withdraw $100.00 from her account number 1234. The application interface passes this information to the ACL manager asking for an authorization decision. The ACL manager retrieves the authorization policy for account 1234 from ACL database and applies the policy to derive the answer. If user *jane* is authorized, the withdrawal is performed. DCE security service does not provide transparent access control for application systems. If an application system needs to protect the resources it accesses, it has to implement access control functionality, including ACL manager and ACL storage shown in Figure 4.

In terms of administration of ACL associated with application resources, the DCE applications are also expected provide means for performing the task. They can implement DCE standard ACL administration interface (rdacl). When Jane's account is first set up, a bank employee would use an administrative tool to give user *jane* permission to withdraw money from account 1234. `Rdacl` interface seems to be the only means of ensuring the consistency of authorization policies across application boundaries unless access to the ACL database is implemented as a global service. In the latter case, policy and application changes could also be accommodated by the DCE environment easier than in the basic configuration shown in Figure 4.

As seen from the discussion above, DCE security service provides rudimentary help to applications to make AC decisions, and it enforces no AC externally to an application. Comparatively to its predecessor, Kerberos [33, 49], it advances privilege attribute management by enabling attribute types other than subject identity. However, the expressiveness of DCE ACL language is fairly limited, and we could not determine how application-specific factors could be used in authorization decisions. It seems that the increase of the application client or server population would not drastically affect overall performance of the application systems in a DCE environment because AC decisions are made using local data. However, administration scalability is poor because policy changes have to be reflected in the ACL database for every application unless the database is centralized, in which case the performance scalability would suffer.

5.3. Generic Authorization and Access Control API

Generic Authorization and Access Control API (GAA API) is published as an IETF Internet draft [59]. It defines a framework for application authorization. GAA API aims to address the lack of standard authorization API for applications using GSS API [41]. Kerberos [33, 49] was the first security technology providing GSS API functionality, and it did influence the model behind GSS API. It had only rudimentary authorization support for access control in networked applications. If a client does not have authenticated ticket for a particular network server, it will be denied access. This is why GAA API attempts to build an authorization model that would fit into the existing GSS API model.

It reuses security context from GSS API. Since the GSS API provided only subject identity, GAA API uses group membership service. GSS API provides very generic low-level abstraction. Its usage by application systems requires significant integration efforts. This prompted new generations of security technologies for distributed application systems such as CORBA and DCOM, in which an application can be developed without any notion of underlying security, including access control, unless it requires enforcement of complex security policies. However, if an application does use GSS API and it requires protection of fine grain resources or enforcement of complex access control policies, then GAA API gives more than most application systems would need for interfacing with an authorization service. However, the service, or at least its proxy, should be co-located in the same process because the only language binding, available as of May 2000, is defined in C language [58]. The main advantage of the API over the other models reviewed here is its support for very flexible and powerful concept of additional conditions that may need to be enforced by the application system or met by the client.

5.4. MICROSOFT DISTRIBUTED COMPONENT OBJECT MODEL

Since DCOM RPC is a derivation of DCE RPC, it is not surprising that its security model resembles DCE security. As with DCE, ACLs are used for coding authorization policies. In DCOM, they are named Discretionary ACLs to signify the default right of the owner to modify access control entries (ACE). However, this does not change the essence of the model. What it does, though, is the capability of enforcing policies outside of a DCOM object (proxies and interceptors approach) and a hierarchy of policies.

DCOM provides two choices for controlling access to applications and their resources. With "declarative security," DCOM can enforce AC without any cooperation on behalf of the object or the object's caller. The policies for an application can be externally configured and enforced. The declarative security policies can be divided into default policies and component-specific ones. A default policy specifies the default launch and access settings for all components running on the local machine that do not override these settings. Component security settings can be used to provide security for a specific component, thereby overriding the default security settings.

With "programmatic security," DCOM exposes its security infrastructure to a developer via security APIs[2] so that both clients and objects can enforce their own application-specific authorization policies of any granularity, and using any information as input for the decisions. Programmatic security can be used to override both default and component security settings in the registry.

Every invocation on a DCOM object is subject to control according to the policy hierarchy: 1) policy encoded in the component implementation, 2) the declarative process-specific policies, and 3) the declarative host-specific policies. Policies 2 and 3 are exercised before the call is dispatched to the object. This is a considerable advantage over DCE authorization model, where the security environment enforces no control and an application has to implement its own enforcement of the policies.

A significant hindrance of the authorization model is its granularity of "component-specific" policy (policy type 2). The granularity is per OS process, and there is no distinction among different object methods. That is, the policy uses the same discretionary access control list (DACL) to control access to all objects and their methods that a system process implements. Blakley [13] argues that OO security systems should let security administrators control access to individual methods of an object. Neither of DCOM process-wide or host-wide authorization policies supports such level of granularity.

Process-wide and host-wide policies (types 2 and 3) implicitly introduce the notion of access policy domains for DCOM objects. Unfortunately, the partitioning of objects is based only on their locations and not on their sensitivity or the value of other parameters. The limitation of authorization policy domains to the host boundaries restricts the administration scalability of DCOM-based distributed applications because it has to be performed individually on each host or even for each process.

As we have shown, no application-specific information can be used or application-specific policies are enforced when declarative AC is exercised. Declarative authorization policies and their changes have to be administered on a machine-by-machine basis, which hinders administration scalability and rules out automatic policy consistency across application boundaries unless applications are located on the same host.

---

[2]For example, calling subject identity can be obtained using methods `IObjectContext::IsCallerInRole()` and `ISecurityProperty::GetCallerSID()`.

5.5. SESAME SECURITY

SESAME (a Secure European System for Applications in a Multi-vendor Environment) is a European project started in late 1980s, and funded in part by the European Commission under its RACE program [34]. SESAME technology is not a middleware. Instead, it is an architecture for security services. It does not provide a means of communication such as ORB in CORBA, RPC layer in DCE or DCOM. Thus it can not control pre/post invocation events. This is why access control and other security functionality has to be specifically invoked by an application system. This prevents SESAME from providing access control using proxies or interceptors approach. Instead, AC logic is provided, as an authorization service, to an application system by SESAME-compliant infrastructure, as opposed to DCE, where an application system has to implement ACL storage, as well as run-time and management functionality.

Another drawback of SESAME AC is the lack of capabilities, or at least there is no straightforward way, for applying one AC policy to several application systems. It means that AC policies have to be configured for applications individually.

The unit of AC check provided by SESAME is an application system. Therefore, either access is granted to the whole system or access is denied at all. For distributed application systems, which commonly expose their functionality via several operations with different AC requirements, such granularity is insufficient. Consequently an application system has to implement additional functionality to exercise per-operation AC. This makes SESAME less attractive then JAAS, DCE, DCOM or CORBA technologies discussed here from software engineering point of view. However, SESAME is neutral to the underlying communication protocols, and is known for its advanced model of privilege attributes management and propagation that is best suitable for large multi-domain heterogeneous environments [6]. This makes it suitable for building heterogeneous, multi-vendor distributed application systems that require authorization based on privilege attributes, other than user identity.

5.6. CORBA SECURITY

CORBA Security [50], like DCOM Security, strictly follows the direction of interceptors. Everything, including obtaining information necessary for making authorization decisions, is done before the method invocation is dispatched. The decisions are based on subject privilege attributes, required rights of the method, and the access control policy of the domains to which the object belongs. AC decisions can be specific for each object, if the object is located in a separate domain, or for a large group of objects associated with one policy domain. This means that the model scales very well without loosing fine granularity. Unlike DCOM, CORBA objects residing on different computers can be associated with the same domains.

The expressive power of CORBA access control mechanisms was analyzed by Karjoth [36], where it was shown to be capable of support lattice-based mandatory access control (MAC). Beznosov and Deng show [11] that it also possible to configure CORBA AC mechanisms to support role-based access control (RBAC0 – RBAC3), which means that discretionary access control (DAC) models can be also implemented, as Sandhu and Munawer show in [62].

Because CORBA Security defines advanced concepts of privilege attributes, similar to SESAME, it enables AC policies based on roles, groups, clearance, and any other security-related attributes of subjects and their sessions, i.e. principals. User grouping via privilege attributes, object grouping via policy domains, and method grouping via the concept of required rights make CORBA Security highly scalable in terms of security administration, an important factor in object-oriented enterprise distributed environments.

Given the power and flexibility of CORBA Security AC mechanisms, there are still applications, in which additional access control needs to be exercised. If an application system is to enforce its own AC, it can do so with the help of CORBA Security interfaces. For enforcing conventional access control policies, an application system needs to know who wants to access what protected resource and in what way. CORBA Security interfaces, available to an application system, contain a method for obtaining subject security attributes, including privilege attributes. This information is sufficient for enforcing application-specific AC on application resources.

6. SUMMARY AND DISCUSSION

Middleware technologies provide several means to control the use of distributed resources exposed via application interfaces. There are two groups of technologies used for securing distributed software systems. One group merely provides party authentication, communication protection, and access control independently of the underlying communication technology, which includes Kerberos [33,49], SESAME [34, 52] and GAA API [59]. This enables using and mixing any desired communication protocols and media, but developers are overburdened with significant efforts to integrate the security technology with the underlying communication mechanisms.

Another group is middleware technologies, such as CORBA [50], DCE [22], Java [38], and DCOM [46], which provide the underlying communication infrastructure along with the security subsystem. They enjoy reasonable integration of both and much more seamless use of the former by developers. Moreover, some of them enable basic access control completely outside of an application system because access decision and enforcement occur before the remote call is dispatched to the application server.

Access control in Java Authentication and Authorization Service (JAAS) is enforced only on system resources, such as files, sockets, etc., but not on Java objects and other application resources. JAAS has very generic and extensible support for different privilege attributes that can be easily defined via new classes. The source code base, the identity of the code signer, and the value of the subject privilege attribute are passed to the authorization code via `Policy` class interface for authorization decisions. JAAS allows any granularity of authorization decisions, and it does not constrain implementers of authorization policies to any particular mechanism or to the information used for the decisions. It also enables seamless change of policies. However, the architecture does not address the consistency of authorization policies across multiple applications. Nor does it have any provisions for achieving performance and administration scalability.

In the DCE, application systems are expected to enforce and provide administrative access to authorization policies themselves. An application system can use DCE access control list (ACL) but it has to implement most of access control functionality, including ACL storage and manager, as well as its administration. DCE Security supplies an application with only the caller's subject and group identities. Cross-application administration of authorization logic is not directly supported although administrative interface for doing the administration on per-application basis is defined, yet it is not a scalable solution.

The security model of DCOM resembles DCE security. As with DCE, ACLs are used to code authorization policies. The main advance of DCOM is the capability of enforcing policies outside of objects with the presence of process and host-specific policies in addition to the capability for an application to use DCOM Security API for its own AC. The authorization model is significantly hindered by the granularity of the so-called "component-specific" policy where there is no distinction among different objects and their methods in the same OS process. Component- and host-wide policies implicitly introduce the notion of access policy domains; still it is not clear if such domain partitioning

is an administratively scalable and functionally successful solution. The administration has to be performed individually on each host or even for each process, which is better than the DCE solution but still limited. Although DCOM Security provides ways for application systems to exercise fine grain AC in an application-specific way, application-specific policies cannot be enforced and only security-related attributes of subjects and objects can serve as input for external AC.

SESAME is an architecture for security services which does not specify a communication layer. Thus it cannot control pre/post invocation events. This is why AC and other security functionality has to be specifically activated by an application. This prevents SESAME from providing AC externally to applications. Another drawback of SESAME authorization is the lack of support for applying one policy to several application systems located on separate hosts. The unit of authorization check is an application system. All these, especially the granularity of AC, make SESAME less attractive then JAAS, DCE, DCOM or CORBA technologies for providing access control to application resources. However, SESAME is neutral to the underlying communication protocols, and is known for its advanced model of privilege attributes management and propagation. This makes it indispensable for building heterogeneous, multi-technology and multi-organization distributed applications that require authorization based on privilege attributes, other than user identity, and the use of different communication technologies.

In CORBA Security, access control can be enforced completely outside of an application system. AC decisions are based on subject privilege attributes, required rights of the method, and the access control policies of the domains to which the object belongs. The AC model scales very well without losing fine granularity, for the decisions could be specific to each object or to a large group of objects associated with the same policy domain. Unlike DCOM, CORBA objects residing on different computers can be associated with the same policy domain. Because CORBA Security defines advanced concepts of privilege attributes, it enables AC policies based on roles, groups, clearance, and any other security-related attributes of subjects. User grouping via privilege attributes, object grouping via policy domains, and method grouping via the concept of required rights improve administration and performance scalability. If an application system is to enforce its own AC, it can do so with the help of CORBA Security API, which allows it to obtain subject security attributes, including privilege attributes. However, application-specific policies are difficult to enforce and the use of application-specific information in the CORBA AC is limited.

The Generic Authorization and Access Control API (GAA API) defines a framework for application authorization. The API aims to address the lack of standard authorization interfaces for those applications which use the generic security service (GSS) API. This is why GAA API's authorization model specifically fits into the existing GSS API. If an application uses GSS API, which provides very generic low-level abstraction, and it requires the protection of fine grain resources or the enforcement of complex authorization policies, GAA API defines interface with enough capabilities for most applications. The main advantage of the API over the other reviewed models is the support for the very flexible and powerful concept of additional conditions that can support application-specific policies. The drawbacks of the API are that it only defines the interface between an application and an authorization mechanism, and the model addresses neither administration scalability nor the consistency of authorization policies across multiple applications.

Ideally, all security functionality should be engineered outside of an application system, thereby allowing the application to be so called "security unaware." However, this is difficult to achieve for the majority of applications, where access control and other security policies are too complex, or require too fine-grain control, to be supported by the general-purpose middleware security technologies. This is an ongoing subject of research.

We expect that contemporary information enterprises with their dynamics and complexity have to have application AC engineered at the middleware level as well as at the application level. For application-level access control, successful architectural solutions most probably will employ a combination of proxies and interceptors, policy agents, as well as authorization services because they complement each other. For systems with the existing AC mechanisms tightly integrated into applications, the approach of policy agents is the only choice. In those systems, where AC mechanisms are missing, weak, or too coarse grained, interceptors and proxies, combined with the ideas from policy agents and authorization services could cure the problem. New applications with requirements for fine-grain access control, complex or very dynamic AC policies, or to be deployed in organizations of different types (e.g. military, government, finance, health care, telecommunications) and sizes, will be best constructed with the use of authorization service.

REFERENCES

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A Calculus for Access Control in Distributed Systems," DEC August 1991.

[2] M. D. Abrams, A. B. Jeng, and I. M. Olson, "Generalized Framwork for Access Control: An Informal Description," The MITRE Corporation, Springfield, VA, USA MTR-89W00230, September 1989.

[3] M. D. Abrams, K. W. Eggers, L. J. LaPadula, and I. W. Olson, "A Generalized Framework for Access Control: an Informal Description," The MITRE Corporation, McLean, Virginia, USA MP-90W00043, August 1990.

[4] M. D. Abrams, L. J. LaPadula, and I. M. Olson, "Building Generalized Access Control on UNIX," In Proceedings of UNIX Security II: USENIX workshop proceedings, Portland, Oregon, USA, 1990, pp. 65-70.

[5] M. Abrams, J. Heaney, O. King, L. J. LaPadula, M. Lazear, and I. Olson, "A Generalized Framework for Access Control: Towards Prototyping the Orgcon Policy," In Proceedings of National Computer Security Conference, 1991, pp. 257-266.

[6] P. Ashley, "Authorization for a Large Heterogeneous Multi-Domain System," In Proceedings of Australian Unix and Open Systems Gorup National Conference, 1997.

[7] J. Barkley, "Implementing Role-based Access Control Using Object Technology," In Proceedings of The First ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1995, pp. 93-98.

[8] J. Barkley, K. Beznosov, and J. Uppal, "Supporting Relationships in Access Control Using Role Based Access Control," In Proceedings of ACM Role-based Access Control Workshop, Fairfax, Virginia, USA, 1999, pp. 55-65.

[9] M. Benantar, R. Guski, and K. M. Troidle, "Access control systems: From host-centric to network-centric computing," IBM Systems Journal, vol. 35(1), pp. 94-112, 1996.

[10] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "Supporting Periodic Authorizations and Temporal Reasoning in Database Access Control," In Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India, 1996, pp. 472-483.

[11] K. Beznosov and Y. Deng, "A Framework for Implementing Role-based Access Control Using CORBA Security Service," In Proceedings of Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1999, pp. 19-30.

[12] K. Beznosov, Y. Deng, B. Blakley, C. Burt, and J. Barkley, "A Resource Access Decision Service for CORBA-based Distributed Systems," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 310-319.

[13] B. Blakley, *CORBA Security: an Introduction to Safe Computing with Objects*, First ed. Reading: Addison-Wesley, 1999.

[14] J. Bloomer, *Power Programming with RPC*. Sebastopol, CA: O'Reilly & Associates, 1992.

[15] CA, "CA-ACF2 for OS/390," Computer Associates, 1998.

[16] CA, "CA-Top Secret for OS/390," Computer Associates International, 1998.

[17] D. L. Caswell, "An Evolution of DCE Authorization Services," *Hewlett-Packard Journal: technical information from the laboratories of Hewlett-Packard Company*, vol. 46(6), pp. 49--54, 1995.

[18] D. A. Curry, *UNIX System Security: A Guide for Users and System Administrators*: Addison-Wesley, 1992.

[19] DEC, "Guide to VAX/VMS System Security --- Version 5.2," Digital Equipment Corporation, 1989.

[20] R. Filman and T. Linden, "SafeBots: a Paradigm for Software Security Controls," In Proceedings of New Security Paradigms Workshop, Lake Arrowhead, CA USA, 1996, pp. 45-51.

[21] R. Filman and T. Linden, "Communicating Security Agents," In Proceedings of The Fifth Workshop on Enabling Technologies: Infrastructure for Collabarative Enterprises, Stanford, CA, USA, 1996, pp. 86-91.

[22] F. Gittler and A. C. Hopkins, "The DCE Security Service," *Hewlett-Packard Journal*, vol. 46(6), pp. 41-48, 1995.

[23] V. Gligor, C. Burch, R. Chandersekaran, L. Chanpman, M. Hecht, W. Jiang, G. Luckenbaugh, and N. Vasudevan, "On the Design and the Implementation of Secure Xenix Workstations," In Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA, 1986, pp. 102-117.

[24] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2," In Proceedings of The USENIX Symposium on Internet Technologies and Systems, Monterey, California, 1997, pp. 103-112.

[25] F. T. Grampp and R. H. Morris, "UNIX Operating System Security," *AT& Bell Laboratories Technical Journal*, vol. 63(8), pp. 1649-1672, 1984.

[26] A. S. Grimshaw, M. J. Lewis, A. J. Ferrari, and J. F. Karpovich, "Architectural Support for Extensibility and Autonomy in Wide-Area Distributed Object Systems," Department of Computer Science, University of Virginia CS-98-12, 1998.

[27] A. S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40(1), pp. 39-45, 1997.

[28] B. Hailpern and H. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Transactions on Software Engineering*, vol. 16(11), pp. 1247-1257, 1990.

[29] J. Hale, P. Galiasso, M. Papa, and S. Shenoi, "Security Policy Coordination for Heterogeneous Information Systems," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 219-228.

[30] A. Heydon and J. D. Tygar, "Specifying and Checking UNIX Security Constraints," *Computing Systems*, vol. 7(1), pp. 9-12, 1994.

[31] R. Hommes, "VMS Security Architecture," In Proceedings of DECUS Europe Symposium, Cannes, France, 1990.

[32] IBM, *Resource Access Control Facility (RACF). General Information*: IBM Red Books, 1976.

[33] IETF, "RFC 1510, The Kerberos Network Authentication Service, V5," 1993.

[34] P. Kaijser, "A Review of the SESAME Development," *Lecture Notes in Computer Science*, vol. 1438, pp. 1-8, 1998.

[35] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn, "A Retrospective on the VAX VMM Security Kernel," *IEEE Transactions on Software Engineering*, vol. 17(11), pp. 1147-1165, 1991.

[36] G. Karjoth, "Authorization in CORBA Security," In Proceedings of Fifth European Symposium on Research in Computer Security (ESORICS), 1998, pp. 143-158.

[37] M. M. Kong, "DCE: An Environment for Secure Client/Server Computing," *Hewlett-Packard Journal*, vol. 46(6), pp. 6-15, 1995.

[38] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers, "User Authentication And Authorization In The Java Platform," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 285-290.

[39] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practics," In Proceedings of ACM Symposium on Operating Systems Principles, Asilomar Conference Center, Pacific Grove, California, 1991, pp. 165-182.

[40] L. J. LaPadula, "Formal modeling in a Generalized Framework for Access Control," In Proceedings of Computer Security Foundation Workshop III, 1990, pp. 100-109.

[41] J. Linn, “Generic Security Service Application Program Interface,” Internet Engineering Task Force, Internet Draft RFC 1508, September 1993.

[42] G. L. Luckenbaugh, V. D. Gligor, L. J. Dotterer, and C. S. Chandersekaran, “Interpretation of the Bell-LaPadula Model in Secure Xenix,” In Proceedings of DoD-NBS Conference on Computer Security, 1986.

[43] V. Matena and M. Hapner, “Enterprise JavaBeans,” Sun Microsystems Inc., Palo Alto, CA., Specification March 21 1998.

[44] E. J. McCauley and P. J. Drongowski, “KSOS -- The Design of a Secure Operating System,” In Proceedings of National Computer Conference, 1979.

[45] M. J. McInerney, *Windows NT Security*, Prentice Hall, 1999.

[46] Microsoft, “DCOM Architecture,” : Microsoft, 1998.

[47] R. Monson-Haefel, *Enterprise JavaBeans*, O'Reilly and Associates, 1999.

[48] S. J. Mullender, G. v. Rossum, A. S. Tanenbaum, R. v. Renesse, and H. v. Staveren, “Amoeba: A Distributed Operating System for the 1990s,” *Computer*, vol. 23(5), pp. 44-53, 1990.

[49] B. C. Neuman and T. Y. Ts'o, “Kerberos: an Authentication Service for Computer Networks,” University of Southern California, Information Sciences Institute ISI/RS-94-399, 1994.

[50] OMG, “Security Service Specification,” in *CORBAservices: Common Object Services Specification*: Object Management Group, 1996.

[51] OSF, “Authentication and Security Services,” Open Software Foundation, 1996.

[52] T. Parker and D. Pinkas, “SESAME V4 - Overview,” SESAME December 1995.

[53] C. P. Pfleeger, *Security in Computing*, Prentice-Hall, 1989.

[54] J. S. Quarterman, A. Silberschatz, and J. L. Peterson, “4.2BSD and 4.3BSD as Examples of the UNIX System,” *ACM Computing Surveys*, vol. 17(4), pp. 379-418, 1985.

[55] T. Riechmann and F. J. Hauck, “Meta Objects for Access Control: Extending Capability-based Security,” In Proceedings of New Security Paradigms Workshop, Langdale, Cumbria, UK, 1997, pp. 17-22.

[56] T. Riechmann and F.J. Hauck, “Meta Objects for Access Control: A Formal Model for Role-based Principals,” In Proceedings of New Security Paradigms Workshop, Charlottesville, VA USA, 1998, pp. 30-38.

[57] W. Rubin and M. Brain, *Understanding DCOM*, P T R Prentice Hall, 1999.

[58] T. Ryutov and C. Neuman, “Access Control Framework for Distributed Applications (Work in Progress),” Internet Engineering Task Force, Internet Draft draft-ietf-cat-acc-cntrl-frmw-03, March 9 2000.

[59] T. Ryutov and C. Neuman, “Generic Authorization and Access control Application Program Interface: C-bindings,” Internet Engineering Task Force, Internet Draft draft-ietf-cat-gaa-bind-03, March 9 2000.

[60] J. H. Saltzer, “Protection and the Control of Information Sharing in Multics,” *Communications of the ACM*, vol. 17(7), pp. 388-402, 1974.

[61] R. Sandhu and P. Samarati, “Access Control: Principles and Practice,” *IEEE Communications Magazine*, vol. 32(9), pp. 40-48, 1994.

[62] R. Sandhu and Q. Munawer, “How to Do Discretionary Access Control Using Roles,” In Proceedings of ACM Workshop on Role-based Access Control, Fairfax, Virginia, USA, 1998, pp. 47-54.

[63] *V. Varadharajan and C. C. a. J. Pato*, “Authorization in Enterprise-wide Distributed System: A Practical Design and Application,” In Proceedings of 14th Annual Computer Security Applications Conference, 1998.

[64] B. J. Walker, R. A. Kemmerer, and G. J. Popek, “Specification and Verification of the UCLA Unix Security Kernel,” *Communications of the ACM*, vol. 23(2), pp. 118, 1980.

[65] W. Wilson and K. Beznosov, “CORBAmed Security White Paper,” Object Management Group corbamed/97-11-03, November 1997.

[66] T. Y. C. Woo and S. S. Lam, “Designing a Distributed Authorization Service,” University of Texas at Austin, Computer Sciences Department TR93-29, September 1993.

[67] T. Y. C. Woo and S. S. Lam, “Designing a Distributed Authorization Service,” In Proceedings of IEEE INFOCOM, San Francisco, 1998.

[68] W. A. Wulf, C. Wang, and D. Kienzle, “A New Model of Security for Distributed Systems,” In Proceedings of New Security Paradigms Workshop, Lake Arrowhead, CA USA, 1996, pp. 34-43.

[69] J. W. Yoder and J. Barcalow, “Archictectural Patterns for Enabling Application Security,” In Proceedings of Pattern Languages of Programming, Monticello, Illinois, USA, 1997.

[70] M. E. Zurko, R. Simon, and T. Sanfilippo, “A User-Centered, Modular Authorization Service Built on an RBAC Foundation,” In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, 1998.