RL-TR-91-17
Final Technical Report
March 1991

DISCRETIONARY SECURITY FOR OBJECT-ORIENTED DATABASE SYSTEMS

SRI International

Teresa F. Lunt

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-17 has been reviewed and is approved for publication.

APPROVED: Emilie J. Siarkiewicz

EMILIE J. SIARKIEWICZ
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Ronald Raposo

RONALD RAPOSO
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(COAC) Griffiss AFB, NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	Final Mar 89 - Aug 90

4. TITLE AND SUBTITLE
DISCRETIONARY SECURITY FOR OBJECT-ORIENTED DATABASE SYSTEMS

5. FUNDING NUMBERS
C - F30602-88-D-0026
Del. Order 0007
PE - 35167G
PR - 1068
TA - 01
WU - P2

6. AUTHOR(S)
Teresa F. Lunt

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SRI International
Computer Science Laboratory
333 Ravenswood Ave
Menlo Park CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Cent
Rome Laboratory (COAC)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-91-17

11. SUPPLEMENTARY NOTES
RL Project Engineer: Emilie J. Siarkiewicz/COAC/(315) 330-3241
Prime Contractor for this effort is CALSPAN-UB Research Center, P O Box 400,
(See reverse)

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The discretionary access controls in today's computer systems are designed to enforce a specific access control policy. An application whose access control policies do not easily match the policy that is "wired" into the system is forced to "work around" that wired-in policy. As a result, the application itself must enforce discretionary security and cannot make use of the assurances of the computer system's discretionary access controls. This report presents a flexible approach to discretionary access control that allows the implementation of arbitrary access control policies. The generality of the approach allows a user to implement a discretionary access control policy that is tailored to an application, rather than having to work around a specific policy that is wired into the computer system. The report focuses on discretionary controls for object-oriented systems. Object-oriented systems are an emerging technology of great import for applications in business, industry, and the military. Many of these applications must share information among users with different needs and authorizations. The specific access rules desired will vary from application to application. Thus, a flexible approach to discretionary access control for such systems is proposed.

NOTE: Rome Laboratory/RL (formerly Rome Air Development Center/RADC)

14. SUBJECT TERMS
Computer Security, Discretionary Access Control, Database Management, Object-Oriented Database Systems

15. NUMBER OF PAGES
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Block 11 (Cont'd)

Buffalo NY 14225

Contents

Acknowledgments

This research was supported by the U.S. Air Force, Rome Air Development Center (RADC), who funded SRI through Subcontract C/UB-07 with the Calspan - UB Research Center (CUBRC), under U.S. Government Contract F30602-88-D-0026. The author gratefully acknowledges RADC for making this work possible.

Abstract

The discretionary access controls in today's computer systems are designed to enforce a specific access control policy. An application whose access control policies do not easily match the policy that is "wired" into the system is forced to "work around" that wired-in policy. As a result, the application itself must enforce discretionary security and cannot make use of the assurances of the computer system's discretionary access controls.

Here we present a flexible approach to discretionary access control that allows the implementation of arbitrary access control policies. The generality of the approach allows a user to implement a discretionary access control policy that is tailored to an application, rather than having to work around a specific policy that is wired into the computer system.

We focus on discretionary controls for object-oriented systems. Object-oriented systems are an emerging technology of great import for applications in business, industry, and the military. Many of these applications must share information among users with different needs and authorizations. The specific access rules desired will vary from application to application. Thus, we propose a flexible approach to discretionary access control for such systems.

Chapter 1

Introduction

Traditional discretionary security mechanisms are usually based on either access control lists or capability lists. These mechanisms are cumbersome if one wishes to impose further restrictions based on, for example, time, day of week, date, or location. Additionally, these mechanisms tend to restrict one's thinking about access control policies to the abilities of the mechanisms.

In addition to the usual simple access control lists and capability lists, there are other dimensions to discretionary access control policies whose interpretations are ambiguous and which have largely been ignored in today's commercial systems. However, some of these dimensions, such as support for user groups and for specific denial of authorization, are required at the higher evaluation classes of the *DoD Trusted Computer System Evaluation Criteria* [1], and others, such as support for role-based access controls, are commonly required for military applications. Others, such as ownership policies, are commonly implemented, but in a seemingly haphazard way. For database systems, there are even more policy choices. The result is that it is apparently impossible for a single general-purpose system to satisfy the discretionary access control requirements of all, or even most, applications. Thus, until now, vendors have been forced to make arbitrary choices, and the users of such systems have had to force their security policies into the vendor-supplied access control mechanisms.

Because so many alternative formulations of discretionary access control policies are possible, it is unnecessarily limiting to have to "wire" a specific policy into a system. It is appealing to envision a system that could enforce any of a number of discretionary access control policies, where each

installation would instantiate the particular policy to be enforced. Such a mechanism should be designed into a system in such a way that high assurance could be obtained that the security policy that the user selects will be enforced. The benefit of this is obvious: with a wired-in policy, only the single policy that is wired into the system can be enforced with any assurance. If an application works around the wired-in policy to implement an application-specific policy, there will be no assurance that this application-specific policy will be correctly enforced. With a general approach such as is proposed here, in which a user can express a wide variety of security policies, whatever policy the user expresses can be enforced with the same degree of assurance.

In this report we first discuss some of the discretionary access control policy dimensions for which flexibility and choice are desired. (Many of these issues were first raised by Lunt [2].) Then we present a general approach for specifying arbitrary application-specific discretionary access control policies in a single general-purpose database system. We pay particular attention to discretionary access control policies for object-oriented database systems.

This report is organized as follows. Chapter 2 introduces some generic issues of discretionary access control. Chapter 3 introduces some additional issues that arise in database systems. Chapter 4 provides some background and definitions for object-oriented database systems. Chapter 5 discusses discretionary security issues for object-oriented databases. Chapter 6 proposes some extensions to the approach to allow users an additional degree of flexibility in specifying access control policies. Chapter 7 contains our conclusions.

Chapter 2

General Issues in Discretionary Access Control

Discretionary security policies for most operating systems and file systems are fairly simple and straightforward. These policies can be easily modeled using the Graham-Denning access matrix model [3]. This model defines an access matrix in which the rows represent *subjects* (users, processes), the columns represent *objects* (e.g., files, programs, subsystems), and the intersection of a row and a column contains the access modes (e.g., read, write, execute) that the subject has authorization for with respect to the object.

The Graham-Denning model leaves many questions unanswered, however. Particularly troubling are some of the requirements in the *DoD Trusted Computer System Evaluation Criteria* [1] for support for such things as group authorization and explicit denial of authorization. This chapter discusses some of these issues.

2.1 Groups

Discretionary security policies are concerned not only with which subjects may obtain access to which objects, but also with the granting, revoking, and denying of authorizations to and from users and groups. Given the set of authorizations for users and groups, some rule must be applied for deriving authorizations for *subjects*.

In the general case, a user may belong to more than one group. In assigning privileges to subjects acting on behalf of a user, one can choose to

- Have the subject operate with the union of the privileges of all the groups to which the user belongs as well as all his or her individual privileges
- Have the subject operate with the privileges of only one group at a time as well as all his or her individual privileges
- Allow the subject to choose whether to operate with its user's privileges or with the privileges of one of the groups to which its user belongs
- Implement some other policy

The second and third options above provide a means to support the concept of least privilege.

Note that even if a subject S is constrained to be associated with at most one group to which its associated user U belongs, a user is still not constrained to operate with the authorizations of only one group at a time. For example, if user U belongs to a group G_1 that is authorized for a relation or view R, and U also belongs to another group G_2 that has been specifically denied authorization for R, then U can still gain access to R by employing a subject whose associated group is G_1 (unless U has also been individually denied authorization for R). Thus, this choice of policy constrains subjects rather than users, and can be thought of as a form of least privilege.

2.? Roles

Some applications may require that discretionary access controls be specified on the basis of user roles. Many systems have some built-in roles (e.g., system administrator, database administrator, system security officer). However, different users are likely to have different requirements and definitions for such roles. In addition, many applications require that arbitrary user job access control requirements be formalized in terms of roles (for example, the secure military message system [4]). Thus, a generic capability for application-defined roles is desirable.

The relationship between a user's role authorizations and his or her user and group authorizations probably depends on the application. Whether

a user acting in a certain role is to be prohibited from granting some of his or her role privileges to a user acting in another role is also probably application dependent.

Recently a roles-like extension to the SQL query language for relational database systems has been adopted by ANSI for a future version of the ANSI SQL standard [5]. This extension aims to improve database system security by simplifying security administration. Discretionary security in SQL is based on access control lists. In most database applications, the users interact with the system at a much higher conceptual level than that of individual relations. For example, a typical database application might present the user with a set of menus and fill-in-the-blank screens, and provide the user with a set of function keys with specific operations defined for each screen. Users who interact with such an application are unlikely to be aware of how the data is organized into relations. Thus, an access control list mechanism for discretionary security requires users to specify the security attributes at too low a conceptual level in the system. The application's security policy is likely to be expressed in terms of the entities defined for the application, rather than in terms of the underlying relations. Typically, in these applications, managing the access control lists on the underlying data is performed by a security administrator. But managing a large collection of access control lists is a hard problem [6]. Part of what makes the problem so hard is that the security administrator must maintain a conceptual mapping in his or her head for how the application is mapped onto individual relations or columns of relations. In addition, typical application-specific security policies may be of the form "this user can access that data only if he is running one of these application programs" [5]; such policies are impossible to express in terms of simple access control lists.

The ANSI SQL roles facility addresses these problems by introducing the concept of *named protection domains* (NPDs) [5]. A named protection domain is a way of grouping privileges and assigning the resultant collection of privileges to specific individuals. A named protection domain is introduced to simplify security administration. For example, the ability to update the SALARY column in an EMPLOYEES table can be granted to a named protection domain called SALARY-CLERK. All the other specific authorizations required by a salary clerk would also be granted to the SALARY-CLERK named protection domain. Then, when a new salary clerk joins the organization, it is a simple matter to grant the new individual authorization for the SALARY-CLERK named protection domain. The NPDs can be designed by the applications designers and set up by the applications builders;

then the administration of security becomes a relatively simple task that can be performed by someone with little or no knowledge of the underlying implementation of the application in terms of relations and views. Only one NPD can be active at any one time, making it possible to enforce a separation-of-duties policy with this facility.

The ORACLE database management system implements the ANSI SQL roles facility described above.

2.3 Ownership

Most systems implement some concept of ownership of objects; each object has a defined owner. Ownership policies are generally implemented so that the owner of an object is the only individual authorized for certain operations on the object, for example, the authorization to delete the object, or to grant and revoke authorizations for the object.

The requirements for the specific definition of object ownership in terms of the special authorizations that are held only by the owner of an object may vary from application to application. Whether a certain class of user, such as a database administrator or security officer, is able to revoke such special authorizations from the owner of an object may also be an application-dependent choice. A facility that allows an appropriate ownership policy for the organization to be defined at system installation would allow vendors to provide the comprehensiveness and flexibility of control to cover most applications' access control requirements while avoiding having to wire in a fixed ownership policy.

2.4 Explicit Denial of Authorization

In the higher evaluation classes of the *DoD Trusted Computer System Evaluation Criteria* [1], users must be able to specify which users and groups are authorized for specific modes of access to named objects, as well as which users and groups are explicitly *denied* authorization for particular named objects. Note that explicit denial of authorization is *not* the same as simple *lack* of authorization. For example, the set of users and groups authorized for an object might be implemented as an ACL (access control list) and the set of users and groups explicitly denied authorization as an XACL (*exclusionary* access control list), as in the naval surveillance model [7]. Because the set of users and groups authorized for an object may be independent

of the set of users and groups denied authorization, there may be apparent conflicts between the two sets. For example, consider the following ACL and XACL for a relation R:

ACL: U_1, U_2, G_1, G_2
XACL: U_1, G_2, U_3, G_3

Now consider the following questions:

- Is U_1 authorized for R?
- If $U_2 \in G_3$, is U_2 authorized for R?
- If $U_1 \in G_1$, is U_1 authorized for R?
- If $U_1 \in G_1$ *and* $U_1 \in G_2$, is U_1 authorized for R?
- If $U_3 \in G_1$, $U_3 \notin G_2$, and $U_3 \notin G_3$, is U_3 authorized for R?

The answers to questions such as these are not provided by the *DoD Trusted Computer System Evaluation Criteria*. Specific choices have been made by particular systems designed for these evaluation classes; however, such choices are arbitrary and may not be suitable for all applications. Lunt [2, 8] goes into detail about a number of specific alternative approaches to these questions. However, choices about the meaning of denial and about how to reconcile the authorizations granted to users individually and as members of groups are application dependent. Thus, such choices could be specified at system-installation time using a mechanism such as is proposed in this report.

Chapter 3

Discretionary Access Control Issues Arising in Database Systems

In a relational database system, additional discretionary access control issues become important. These issues include how to control the propagation of authorizations and revocations, and how view authorizations are related to authorizations on relations.

3.1 Propagation of Authorization

Several database systems, for example ORACLE, use *grantflags* to control the propagation of authorizations. In these systems, grantflags are specified for each user for a relation or view and access mode. The grantflag can have the value "grant" or "nogrant." The grant flag allows a user to grant and revoke the corresponding access mode. In addition, a user with a grant flag for a relation or view R and mode m can give and rescind that grantflag.

In SeaView [9, 10], the propagation of access modes is controlled through the access modes "grant" and "give-grant." If a user U is authorized the grant access mode for a relation or view R, then U can grant or revoke any access mode other than grant and give-grant for R. A user U that is authorized the give-grant access mode for R can additionally grant and revoke the grant and give-grant access modes for R.

SeaView's inclusion of the give-grant mode enables a greater degree of control over the propagation of authorizations than does ORACLE's grant-

flag approach. In ORACLE, if user A grants user B mode m for relation or view R with the grantflag, then A cannot prevent B from granting the grantflag to other users. In SeaView, however, A could grant B the grant mode while withholding the give-grant mode.

There are other means, such as ownership policies, for controlling the propagation of authorizations. The choice, however, is likely to be application specific.

3.2 Propagation of Revocations

In System R [11], when a user A revokes an access mode from another user B, the mode is also revoked from all users to whom B had granted the mode (which in turn starts several other chains of revocations). This policy is called *cascading revocation*. If user B had granted many authorizations over a long period of time, as would be the case if user B were, say, a system security administrator or a database administrator, then the revocation of B's authorizations can have far-reaching, unpredictable, and undesired effects. Managing these difficulties is sufficiently complex that the vendors of the products that implement this cascading revocation policy recommend that, rather than assign individual usernames to security administrators and database administrators, a single role name be used for all such users. That is, these vendors recommend that all system administrators login under a single SYSADMIN username, and all database administrators login under a single DBA username. This has the obvious drawback that such users cannot then be held individually accountable for their actions.

To lessen these difficulties, some database system vendors provide two types of revocation: cascaded and simple. Cascaded revocation is as described above; simple revocation simply revokes the specified authorization from the named group or individual. In SeaView, for example, revocation is not propagated. Moreover, if an ownership policy is in effect, propagation of revocation may be inappropriate.

Whether to propagate revocation of authorization is an application-specific choice.

3.3 Authorizations for Views

In many relational database systems, a user may be authorized for a view without being authorized for the underlying relation(s). In such systems,

granting authorization for a view but not for the underlying relation is a means of restricting authorization to a subset of the data contained in the relation.

SeaView does not require that a user be authorized for a relation in order to access a view defined on that relation. Instead, SeaView includes a *reference* mode that can be used to control which users and groups can gain access to stored data through views. In SeaView, a user can exercise access mode m for a view only if he or she is authorized for the reference mode on all referenced relations at the time the view is accessed. A user can withhold the ability to reference a relation through a view by not granting the reference mode.

The Sybase Secure Dataserver takes another alternative, in which users cannot obtain data through a view unless they have the corresponding authorizations for all the referenced relations. No authorization information is kept for views [12].

A consequence of the SeaView approach is that when a user creates a view, he or she becomes authorized for only those access modes for it for which the user is authorized for each underlying relation. The set of users and groups authorized for a view is modified as access modes are subsequently granted and revoked for that view, independently from the granting and revoking of modes for the underlying relation(s). As a result, if a user or group G is later granted additional authorizations for the underlying relations, G does not thereby gain the corresponding authorizations for the views defined on those relations. Another consequence is that view authorizations are not revoked when authorization for an underlying relation is revoked.

With the Sybase approach, view authorizations are computed from the authorizations for the underlying relations at the time the view is accessed. With this approach, if a user's authorization for an underlying relation is revoked, the user can no longer access the view.

Which of these approaches is desired is an application-dependent choice.

Chapter 4

Object-Oriented Database Systems

Object-oriented databases are a powerful means for organizing and managing very large applications, which would otherwise be impossibly complex. Object-oriented database systems are organized around *objects*, which model real-world entities. The concept of an object can be used to model a simple item, such as a number, or a complex item, such as an aircraft. Each object has some *state* and a defined set of operations that can be performed on it. An object's state is represented by a set of *instance variables* that are part of the object definition. The value of an instance variable can also be the identifier of another object (indicating a relationship with another object). Operations on objects are handled by *methods*, which are executed in the context of the object's state upon the receipt of *messages*.

A group of objects with similar properties forms a *class*, which is also an object. A class can be *system-defined* (for example, a class of integers) or *user-defined* (for example, a class of customers).

4.1 Objects

An *object* represents either a class or an individual, and stores named attributes in *instance variables*. For example, the object CUSTOMER could have instance variables CUSTOMER-ID, ORGANIZATION-NAME, AUTHORIZED-REP, ADDRESS, PHONE-NUMBER, and CALLING-HOURS.

Object-oriented systems typically support both a subclass and an instance relation, where an instance object is an individual, meaning it can

have no instances of its own. Both subclasses and instances inherit the variables of the parent class object, and may have additional variables of their own.

For the purposes of this report, we will not make use of the distinction between subclass objects and individual objects. We will use the term "instance" ambiguously to refer either to a subclass or to an individual instance. The difference between a subclass and an individual instance is significant primarily for reasons of implementation efficiency; it does not affect security policy. Thus, our approach is equally valid for systems that make this distinction and those that do not.

4.2 Methods

An object has *methods* defined for it. Methods encapsulate the behavior of an object, in that an object can be acted upon only through executing the methods defined for the object. Methods consist of executable code. Methods are invoked by sending *messages* to an object. A message consists of a *command*, which selects the appropriate method, and some *arguments*, if necessary.

A method performs three sorts of activities: (1) it may read and write the variables of the object where it resides; (2) it may send messages to other objects, to invoke methods there; and (3) it may, when it terminates, return a value to the sender of the message that invoked it. Ability (2) accounts for much of both the usefulness and complexity of object-oriented systems.

For example, a Bank-account object would have methods for the commands Withdraw, Deposit, and Query-balance. To withdraw or deposit funds in a specific bank account, a message of the form *command value* is sent to the object representing that bank account, where *command* is either Withdraw or Deposit, and *value* is the amount to be deposited or withdrawn. To query the balance for a specific account, the message Query-balance would be sent to the account object.

Methods are inherited by instances. This means that a method placed in a class object is automatically made available to its descendants. The Withdraw method for bank accounts, for example, need only be placed in the Bank-account class object, to permit all individual bank accounts to respond to a Withdraw message. Inheritance of methods is one of the greatest benefits of the object-oriented approach.

4.3 Inheritance

One of the greatest benefits of the object-oriented approach is inheritance. Inheritance allows the object structure, names and default values of instance variables, and methods to be inherited from the object's parent or class object. Thus, in the object model, a *class hierarchy* designates the structure of *subclasses* and *superclasses*.

For example, the class CUSTOMER is a subclass of the class COMPANY. A subclass *inherits* the instance variables and methods of its superclass. If a class has more than one superclass, and two or more of its superclasses have instance variables of the same name, then the instance variable that is inherited by the subclass depends on some *a priori* rule.

Figure 4.1 illustrates the class hierarchy. The class SKI has subclasses NORDIC-SKI and ALPINE-SKI. NORDIC-SKI and ALPINE-SKI inherit the instance variables and methods of SKI, and may define some additional instance variables and methods of their own. Class ALPINE-SKI in turn has subclasses RACING-SKI and RECREATIONAL-SKI, and class NORDIC-SKI has subclasses TELEMARK-SKI, SKATING-SKI, and TOURING-SKI. RACING-SKI and RECREATIONAL-SKI in turn inherit the instance variables and methods of ALPINE-SKI, and TELEMARK-SKI, SKATING-SKI, and TOURING-SKI inherit the instance variables and methods of NORDIC-SKI. In addition to the instance variables and methods they inherit, RACING-SKI, RECREATIONAL-SKI, TELEMARK-SKI, SKATING-SKI, and TOURING-SKI may define some additional ones of their own. In addition, they may override the default values for instance variables that were inherited from their parent.

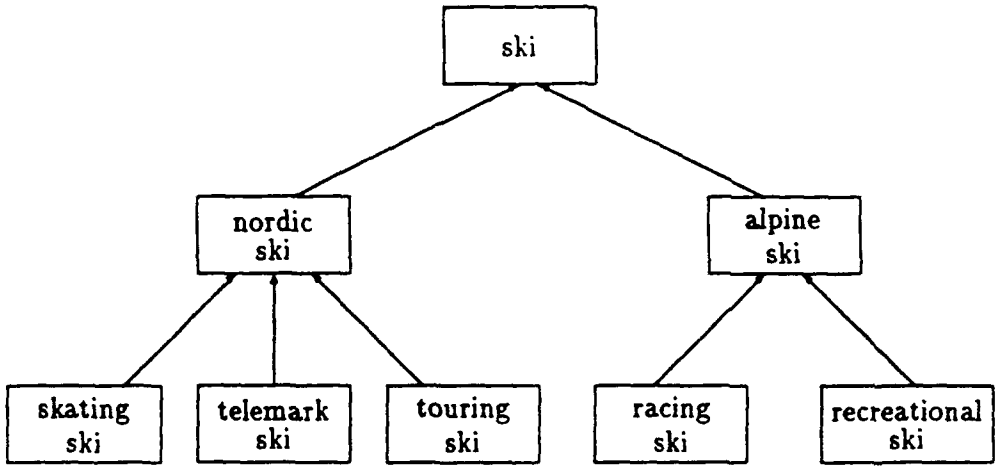

Figure 4.1: The *is-a* class hierarchy

Chapter 5

Discretionary Access Controls for Object-Oriented Database Systems

Discretionary access control models can be represented by the access matrix model developed by Lampson in 1971 [13] and further refined by Graham and Denning [3]. To make it more useful for database systems, the model was extended with predicates and other components [14, 15, 16]. For example, predicates can represent content-dependent access controls that may be implemented by views. There has been considerable study of the application of these models to relational databases [6, 15, 17, 11, 16].

Object-oriented database models are much richer and more complex than the relational model. Thus, many new discretionary security issues are introduced. For example, a fundamental property of object-oriented systems is inheritance. If a user is authorized to access a class, should that user also be authorized to access all of its instances or descendants? Should this inherited authorization also include authorization for attributes that do not exist in the parent class but were defined in the descendants [18]?

If authorizations are inherited, that is, if authorization for a class implies authorization for all instances of the class, then we have the undesirable result that individuals who are authorized for objects that are high up in the object hierarchy (near the root object) will be consequently authorized for much or all of the database. Thus, there is a need to override these inherited

authorizations. One way to do this is to use negative authorization, which is given a higher priority than positive authorization. Another way is to use explicit authorization to override implicit inherited authorization.

In this chapter we present an approach for discretionary security for object-oriented database systems. This approach could also be specialized for use for object-oriented operating systems, which typically do not support inheritance or support only a limited type of inheritance.

5.1 Related Work

Several research groups have been studying discretionary security for object-oriented database systems. Won Kim and his group at MCC have developed a detailed formal model for object-oriented systems using ORION as an illustration (the ORION object-oriented database system was also developed at MCC) [19]. Implied authorizations are inherited through the object hierarchy. They introduce the notion of weak and strong authorizations, where strong authorizations can override weak authorizations. Their model also includes positive and negative authorizations, where negative authorizations are used to represent explicit denial of authorization. Their approach makes an important contribution by proposing algorithms for reconciling negative and positive authorizations through the use of strong and weak authorizations. Here we have borrowed many of their ideas in an attempt to make use of inheritance in the discretionary security policy.

Udo Kelter at the University of Hagen in West Germany is developing models for distributed structurally object-oriented database systems [20]. A similar model was developed by K. Dittrich and his group at the University of Karlsruhe in West Germany, for the DAMOKLES database system [21]. The model defines *complex objects*, which are composed of sets of other objects as component parts. For example, an object of class BICYCLE is a composite of objects of class FRAME, WHEEL, HANDLEBARS, FITTINGS, and SEAT. An object in the *wheel* class is in turn a composite of objects of class TIRE, TUBE, RIM, SPOKE, and HUB. A HUB object may be composed of objects of class BEARING, SCREW, and so forth. Such a database is useful for describing a CAD or CASE application. In Kelter's model, complex objects are the units of access control. Authorization conflicts may arise if complex objects share components. The model also includes a hierarchy of user groups, and subgroups inherit authorizations from supergroups.

Eduardo Fernandez and his group at Florida Atlantic University have

developed an authorization model for object-oriented semantic databases specially tailored to OSAM*, a CAD/CAM database system being implemented at the University of Florida [22, 18]. Fernandez's approach divides the network of objects into administrative domains. The intent of this approach is to make the database suitable for large financial applications.The model defines authorization inheritance through the object hierarchy. Recent work also considers how to handle negative authorizations and the use of predicates [23].

J. D. Moffet and M. Sloman at the Imperial College of London use a model that treats administrative users differently from ordinary users of the data [24]. Their model also includes the notion of object ownership; owners can grant administrative authorizations. Recently they have considered how their model may be implemented in a decentralized fashion [25].

5.2 Representing Discretionary Authorizations

One of the first decisions to be made is whether discretionary authorizations should be attached to objects or stored in separate authorization structures. Most relational database systems use separate authorization structures. In an object model, such separate authorization structures could be represented as a set of objects. With this approach, we would represent authorizations as attributes of users and groups. Thus, users and groups are represented as objects, and their authorizations are represented as instance variables, as shown in Figure 5.1. Figure 5.1 shows class user to be a subclass of class group. Instances of the class user would be user objects representing the individual users of the system. Other instances of the group class would be the specific user groups that are defined. Each of these group objects would have a set of instances that would represent the individuals belonging to the group; these would be object-ids for the user objects representing the individuals. Instances of a group object could also represent subgroups of the group; these would be object-ids for other group objects.

With this approach, a user inherits the authorizations of all the groups he or she belongs to. In addition, subgroups inherit the authorizations of their parent groups. This approach has the drawback that it would be difficult to implement a least-privilege sort of policy.

We take a different approach here, by attaching discretionary authorizations to the objects they refer to. This is illustrated in Figure 5.2.

In Figure 5.2, a list of individuals and groups would appear as the autho-

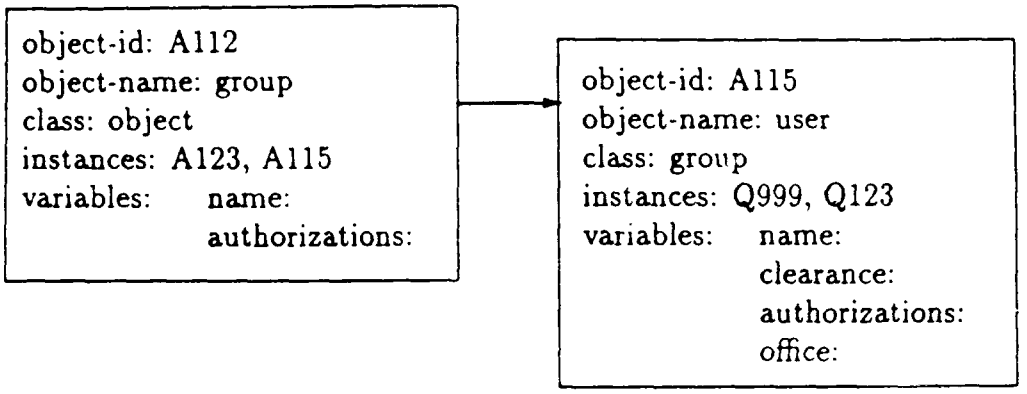

Figure 5.1: Users and groups as objects

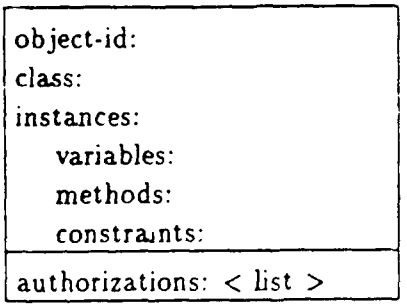

Figure 5.2: Authorizations attached to objects

rization list. (An individual can be thought of as a group with one member.)

5.3 Inheritance of Discretionary Authorizations

With our approach of attaching discretionary authorizations to the objects they refer to, an object inherits the authorizations of its parents. What this means is, if a user is authorized for an object, then the user is also authorized for the descendants, or instances, of the object. This approach has the convenience that a user can be authorized for an entire set of objects simply by authorizing the user for an appropriate class object; it is not necessary to grant the user authorization for all of the dependent objects individually.

In our approach, discretionary access control attributes are inherited just as are instance variables and methods. As for instance variables and methods, an object inherits the discretionary access control attributes not only of its immediate parent object but also of its parent's parents, and so on up the object hierarchy. For an object, additional discretionary access control attributes can be defined and applied in addition to those that are inherited. In the case of multiple inheritance, an object inherits the discretionary authorization attributes of each of its parents.

However, intuitively it seems that the type of policy that many applications would need to implement would be one in which more and more privilege is needed in order to access objects further down the class hierarchy. This is true of the mandatory security models that have been proposed for object-oriented database systems [26, 27, 28]. These mandatory models typically include a property that requires that the classification of an object must dominate that of its parent object(s). Thus, in addition to the convenience of inherited authorizations, we need a way to restrict or override the inherited authorizations so that users can assign the precise authorizations desired for an object. We discuss a means of doing this in the next section.

5.4 Negative and Positive Authorizations

One of the requirements in the *DoD Trusted Computer System Evaluation Criteria* [1] that appears at Class B3 for discretionary access control is that users be able to specify which users and groups are authorized for specific modes of access to named objects, as well as which users and groups are explicitly *denied* authorization for particular named objects (see Section 2.4). Following the work of Kim et al. [29, 19], we will call the explicit

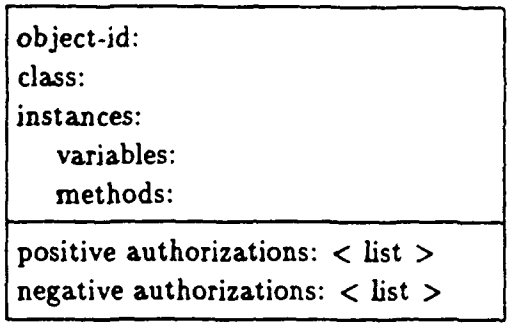

Figure 5.3: An object with its discretionary authorizations

discretionary authorizations *positive authorizations* and the explicit denials of authorization *negative authorizations.*

Just as an object's positive authorizations are attached to the object, as shown in Figure 5.2, we also attach the object's negative authorizations to the object. Figure 5.3 shows the template of an object with its positive and negative discretionary authorizations.

In Figure 5.3, a list of individuals and groups would appear under positive authorizations and negative authorizations. The positive authorizations are interpreted as "must belong to." That is, in order to be authorized for the object, a user must belong to one of the groups in the positive authorizations list. The negative authorizations are interpreted as "must not belong to." That is, if a user belongs to one of the groups on this list, the user cannot access the object.

Negative authorizations can be used to nullify the effects of positive authorizations. For example, in the object structure shown in Figure 5.4, the astronauts group is authorized for object O234. Since authorizations are inherited, this would imply that the members of the astronauts group are also authorized for all instances of object O234, that is, for all spaceships. However, if we would like to restrict access to the hubble spaceship, which is an instance of spaceship, so that user glenn of the astronauts group is not authorized for the hubble spaceship, we can put glenn on the negative authorizations list for the hubble spaceship. This is shown in Figure 5.4. In the figure, the entry for the astronauts group that is shown for the hubble spacecraft object (object O458) has been inherited from the parent object, object O234.

Thus, the use of negative authorizations in conjunction with positive authorizations allows users to implement policies in which more privilege is

needed in order to access objects further down the class hierarchy. Negative authorizations provide a way to restrict the inherited authorizations so that users can assign the precise authorizations desired for an object.

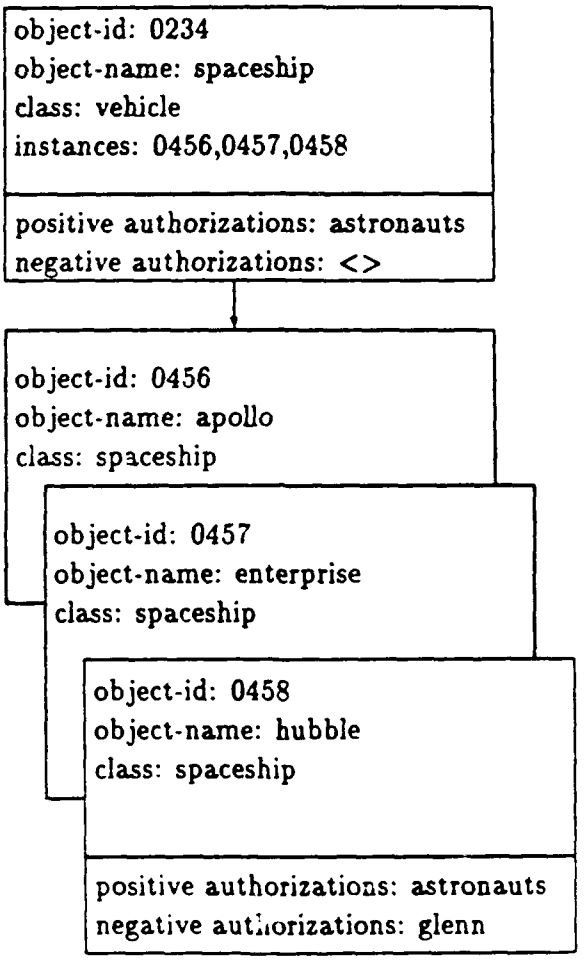

Figure 5.4: The use of negative authorizations

As we mentioned in Section 2.4, there can be difficulties in reconciling conflicting negative and positive authorizations. In [29], the concept of strong and weak authorizations is introduced to deal with this problem. We adopt this concept as well, as we describe in the next section.

5.5 Strong and Weak Authorizations

Our approach includes both weak and strong discretionary authorizations, similar to [29] and [19]. A *strong authorization* is one that cannot be overridden by another authorization. A *weak authorization* can be overridden by a strong authorization.

In [2] various means of reconciling positive and negative authorizations are discussed. One of these is the *most-specific rule*, which requires that if an individual user is specifically granted or denied authorization for an object, this takes precedence over any authorizations for the object that are granted or denied to groups to which the user belongs. This is the approach taken in Multics and proposed for SeaView. With the most-specific rule, negative authorizations are simply a convenience in forming the access control lists. They can be used as follows: if user A wants to make object O available to everyone in group G except user B, then instead of enumerating everyone except B in the access control list, A could grant authorization to G and specifically deny authorization to B.

Another approach discussed in [2] is *denials take precedence*. With this approach, a user or group's denial of authorization for an object takes precedence over any authorizations that the user or group may have been granted for the object. Under the interpretation that denials take precedence, if a user A explicitly denies user U authorization for object O (by granting a negative authorization), and a user B later grants a positive authorization for O to U, then U will not become authorized. Thus, denial of authorization is a strong measure that can be taken to ensure that specific users and groups cannot obtain authorization to an object.

To illustrate the difference in these two policies, consider the object shown in Figure 5.5. The figure shows an object representing the spaceship enterprise. The figure also shows that user kirk is on the positive authorization list for the object, and the captains group is on the negative authorization list for the object. Since user kirk happens also to be a member of group captains, there is a potential conflict between the two lists. With the most specific rule, since user kirk is a more specific entry than group captains, then kirk's positive authorization takes precedence over the captains group's negative authorization; thus, kirk is allowed access to the spaceship enterprise. Thus, with the most specific rule, the two authorization lists can be interpreted as meaning that no captains are authorized for the spaceship enterprise with the exception of captain kirk. With the denials-take-precedence policy, on the other hand, kirk's negative authorization as

a member of the captains group takes precedence over his positive authorization as an individual; thus, kirk is not allowed access to the spaceship enterprise.

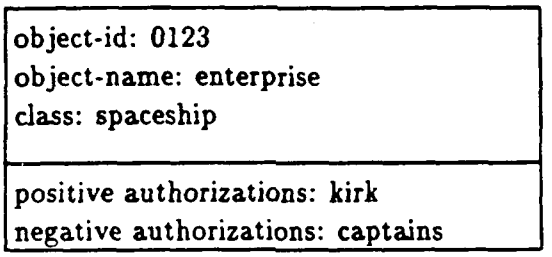

Figure 5.5: Example: conflicting authorizations

The denials-take-precedence rule could be implemented with our approach by making all negative authorizations strong and all positive authorizations weak. Thus, with inheritance of discretionary access control attributes, specifying a negative authorization for a group for a parent object ensures that no access to any of the object's offspring is granted to members of the group. On the other hand, specifying a positive authorization for a group for a parent object does not guarantee that the group will be authorized for the offspring; in fact, the group will be authorized for the offspring unless the child object contains a negative authorization for that group.

The most-specific rule could be implemented with our approach by making individual authorizations strong and group authorizations weak.

Both the denials-take-precedence policy and the most-specific policy are simply two possible policies out of very many possible ways of resolving conflicts among the positive and negative authorizations. How such conflicts should be resolved, whether it be through the use of the denials-take-precedence policy or the most-specific policy, is an application-specific choice. Rather than wire in one such choice, it is desirable to allow the application to specify how such conflicts are to be resolved. To take advantage of the full degree of flexibility possible, we allow specific negative and positive authorizations to be individually labeled strong and weak, so that many more policy variations than the two discussed above are possible. Figure 5.6 illustrates how strong and weak authorizations could be represented

for individual positive and negative authorizations.

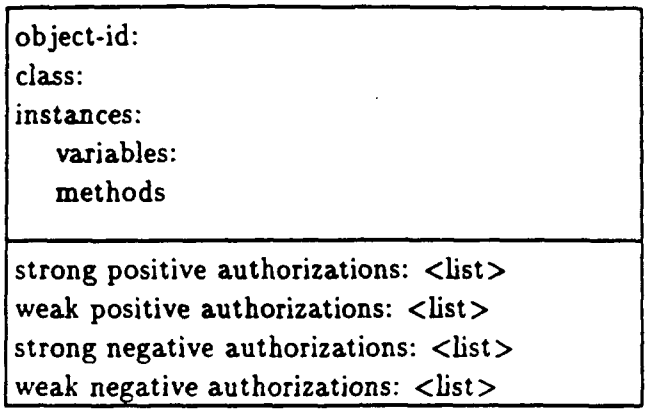

Figure 5.6: Strong and weak authorizations

Figure 5.7 illustrates the use of strong and weak positive and negative authorizations to implement a policy that says that all captains except captain kirk are to be denied authorization to the spaceship enterprise.

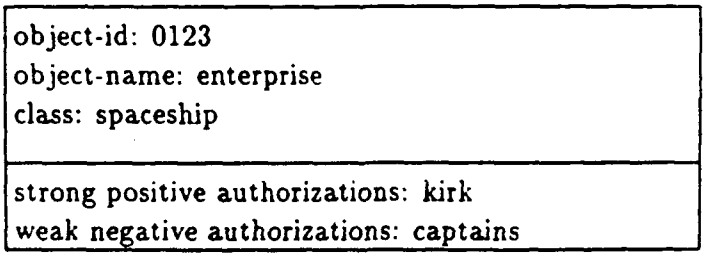

Figure 5.7: Negative and positive authorizations

5.6 Direct Conflicts of Authorizations

It is possible for negative and positive authorizations to conflict, if they are both weak or both strong. For example, in Figure 5.7, if a user could list

the captains group as a strong negative authorization, rather than as a weak negative authorization, there would still be an unresolved conflict. Ideally, the proper assignment of weak and strong authorizations would be used to avoid this situation. However, it may not be obvious at the time that an authorization is granted or revoked that a resulting conflict may occur. Thus, it would be useful and appropriate to include in the object-oriented database system a tool that would analyze the authorizations for an object to detect conflicts. A user would be notified if the result of his or her grant or revoke operation would result in a conflict, so that either the grant/revoke operation would not take effect or the situation would be remedied. One mechanism for doing this would be to have integrity constraints on the authorization lists. These constraints would be part of the root object and inherited by every object in the system.

Since it is possible for such a tool to detect and prevent such conflicts, our approach will assume that such conflicts do not exist, and that strong and weak negative and positive authorizations have been assigned appropriately so as to eliminate conflicts.

5.7 Representing Groups

To implement the policy described in the preceding sections, the system must know which users belong to which groups, and which groups are subgroups to which other groups. We represent users and groups as objects, as shown in Figure 5.8. Figure 5.8 is similar to Figure 5.1 except that discretionary authorizations are no longer attributes of user and group objects. Thus, the figure shows class user to be a subclass of class group. Instances of the class user would be user objects representing the individual users of the system, as shown in Figure 5.9. Other instances of the group class would be the specific user groups that are defined, as shown in Figure 5.10. Each of these group objects would have a set of instances that would represent the individuals belonging to the group; these would be object-ids for the user objects representing the individuals, as shown in Figure 5.11. Instances of a group object could also represent subgroups of the group; these would be object-ids for other group objects; see Figure 5.10.

Note that in Figure 5.11 the user objects E123,E 234, and E456 have two parent objects: payroll and user. Thus, Figure 5.11 shows that the user objects inherit the instance variables of the user class as well as those of the group class.

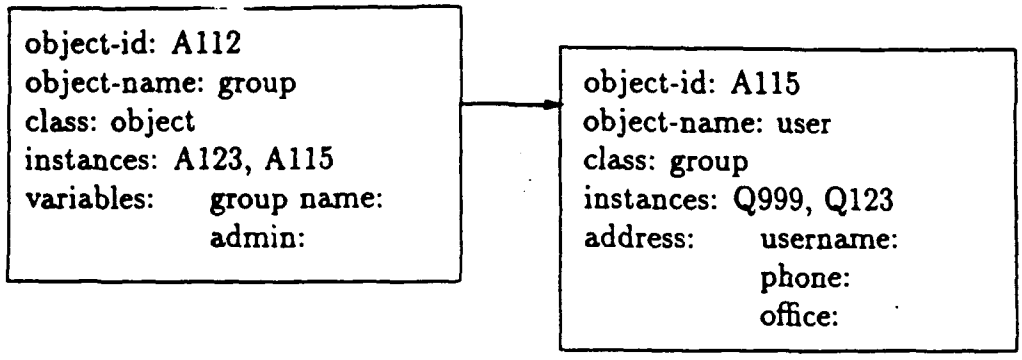

Figure 5.8: Representing users and groups

Arbitrary user attributes can be specified as part of the user class; those shown in Figure 5.8 are simply examples. Some other user attributes that the application might desire to specify could be job, department, projects, location, integrity level, and hire-date. Thus, the system should allow the specification of the user class to be customized by the system security administrator or database designer.

5.8 Access Modes

The discussion above of negative and positive and weak and strong authorizations was simplified somewhat, in that we assumed that the authorization lists contain simply lists of users and groups. However, rather than assign blanket authorization for an object to a user or group, most security policies require the ability to assign specific modes of access for the object to users and groups. Thus, in the authorization lists attached to objects, there must be not only a designation of which users and groups are so authorized, but an indication of which specific modes of access they are authorized.

Thus, each of an object's authorization lists is actually a set of lists, each for a specific access mode, as shown in Figure 5.12. Some access modes, such as create-instance, are relevant for all objects; slots for these are inherited from the root object. Other access modes are relevant to a particular object class and its methods. The particular access modes may correspond one

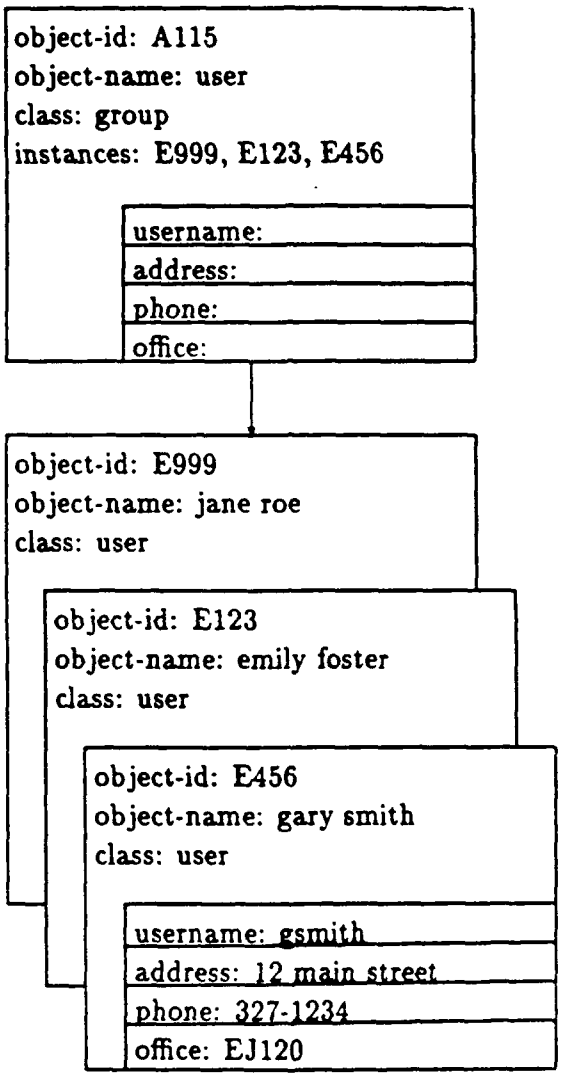

Figure 5.9: Users as objects

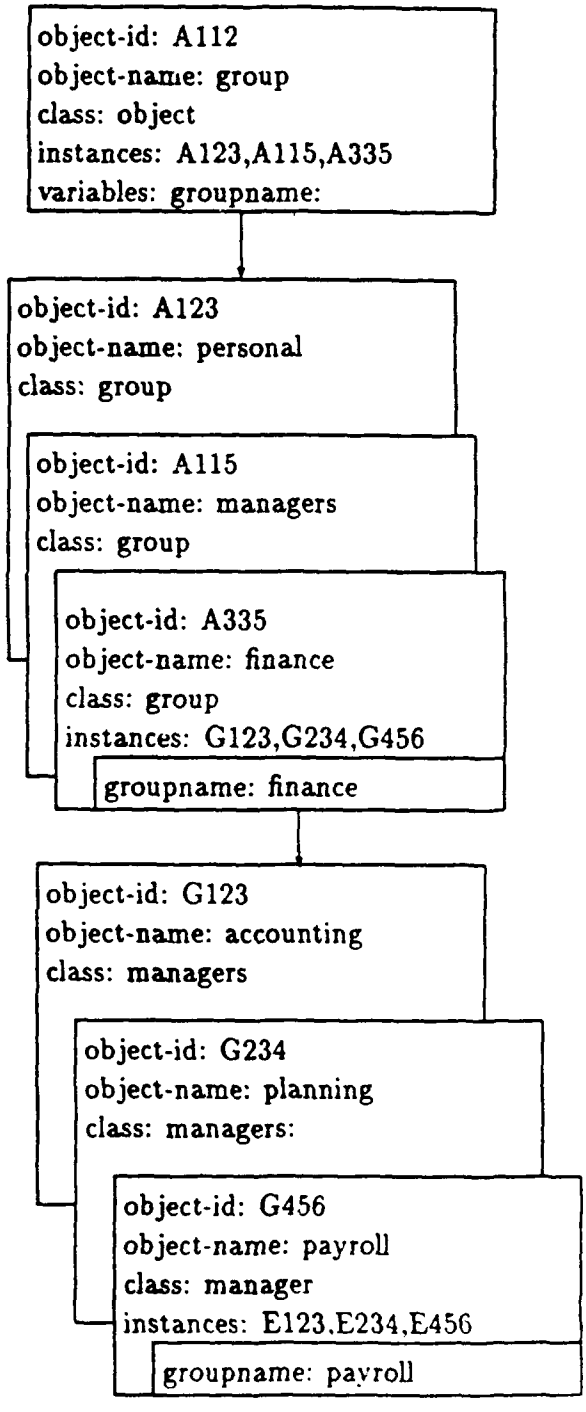

Figure 5.10: Groups and subgroups

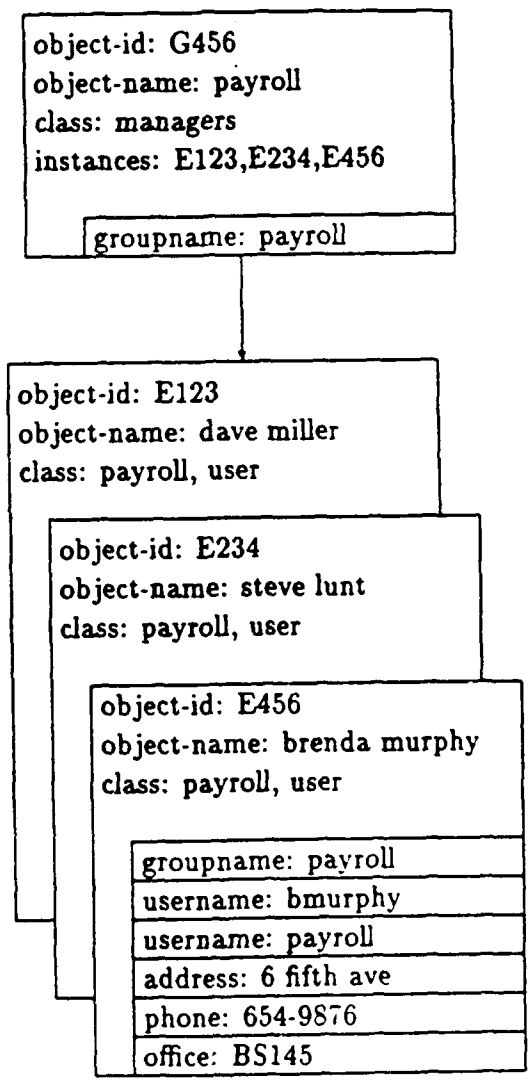

Figure 5.11: Members of a group

to one with the methods defined for the object. Alternatively, a smaller number of access modes could be defined, with each access mode relevant to a set of an object's methods. Figure 5.12 illustrates the structure of the authorization lists. In Figure 5.12, each list is a list of user and group identifiers.

Although the amount of authorization information that must be stored with each object appears from Figure 5.12 to be enormous, in practice the use of inheritance eliminates the need for storage of redundant data. Because most of the authorization information is inherited from the parent objects, only the additions to the authorization need be actually attached to any instance.

5.9 Control of Propagation of Authorizations

The topic of the control of propagation of discretionary authorizations is concerned with who has the authorization to change an object's authorization lists. This is controlled through the use of the special access mode "modify_ACL." Only subjects authorized for the modify_ACL mode can alter the weak and strong positive and negative authorization lists (with the exception of the modify_ACL authorization lists) for the object in question. We also propose the use of a give-grant mode, as in SeaView, to control the propagation of discretionary authorizations (see Section 3.1). Only subjects authorized for the give-grant mode can alter the modify_ACL authorization lists for the object in question.

To provide the user with a choice when revoking authorizations, separate functions would be provided for simple revocation and cascaded revocation.

5.10 Default Access Control Rules

Because authorizations are inherited, authorizations that are defined for the root object can be considered default access control rules. Weak and strong negative and positive authorizations assigned to the root object will be inherited by all other objects in the system. These default authorizations can be restricted or overridden by authorizations that are defined for objects lower in the object hierarchy.

Similarly, different default access control rules can be specified for large portions of the object hierarchy simply by making entries in the authorization lists for objects sufficiently high up in the object hierarchy.

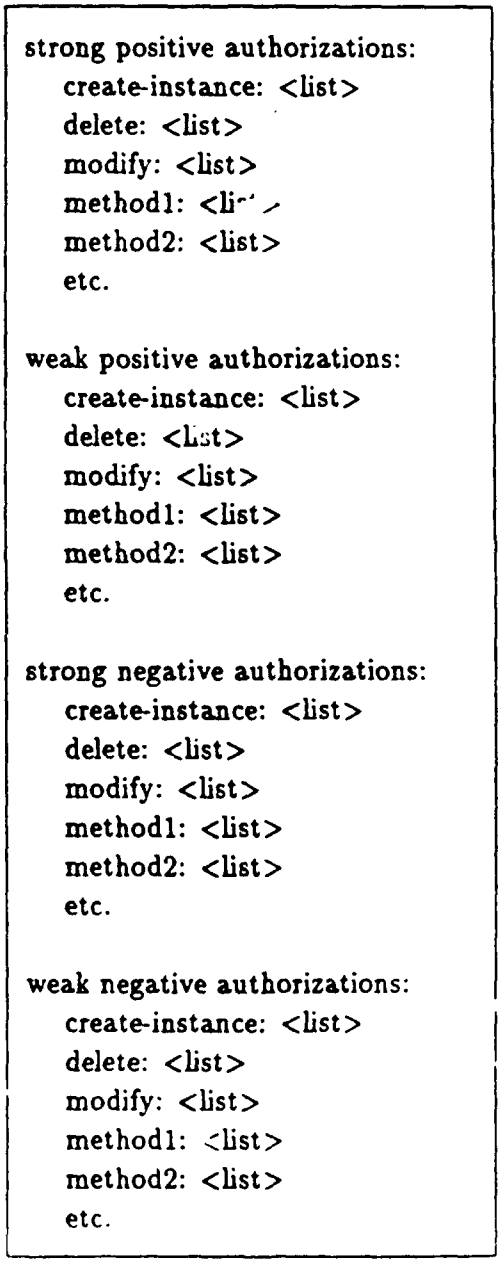

Figure 5.12: Authorization lists

Because the authorizations that are defined for the root object affect the rules of access to all the objects in the system, it is important that they not be changed capriciously. Thus, authorization to modify such authorizations should be restricted to a few special users. This should be controlled by listing only one small group on the positive authorization list for the modify_ACL access mode for the root object.

5.11 Discretionary Constraints

The discretionary access controls discussed in the previous sections can be extended to include named *discretionary constraints*. These discretionary constraints could be named on an object's authorization lists in the same way as user and group names. The names could refer to rules, defined elsewhere in the system, that can be value dependent, contain expressions having group-id variables, contain expressions that refer to other related objects, use logical connectives (group1 or group2), and so forth.

Possibly conflicting discretionary constraints can be resolved in the same way as is done for users and groups. That is, constraint names can be placed on the strong positive, weak positive, strong negative, and weak negative access control lists. Then, the constraints on the positive lists are used to compute who has authorization, while the constraints on the negative lists are used to compute who does *not* have authorization; the constraints on the weak lists can be overridden by the constraints on the strong lists.

The discretionary constraints would be inherited just as the other user and group authorizations are inherited. If an object has more than one direct parent, then the issue arises of how to combine the effects of the rules inherited from the different parents; this is a topic for further research.

Discretionary constraints can be used to allow or deny access on the basis of user identity, user attributes, user role, user group, location, object name, object attributes, object value the value of related objects, time of day, date, application name, and so on. Constraints may refer to other constraints and may use system-defined or externally defined functions.

Chapter 6

Some Extensions

We have presented a flexible approach to specifying discretionary security policies in object-oriented database systems. In this chapter we present a means of providing some additional flexibility to users, and we also advocate the use of a roles mechanism such as that described by Robert Baldwin [5] (see Section 2.2) for simplifying security management in the resulting system.

6.1 Associating Subjects with User and Group Privileges

Given the set of strong and weak negative and positive authorizations for users and groups, some rule must be applied for deriving authorizations for *subjects*. This is especially important in the general case in which a user may belong to several different groups. The rule that is used is likely to depend on the specific needs of the application.

Thus, in a system such as we have described in this report, users should be given a choice of how subject authorizations are derived from user and group authorizations.

We have already discussed the use of strong and weak negative and positive authorizations to control the interaction of individual and group authorizations. However, the following choices still remain. A subject can:

- Assume the union of the user's individual authorizations and those of all the groups to which the user belongs

- Assume some subset of the union of the user's individual authorizations and those of all the groups to which the user belongs (this choice must be designated by the user or application at the time of subject invocation or must be predesignated by the application)
- Assume the union of the user's individual authorizations and of a single group to which the user belongs (this group must be designated by the user or application at the time of subject invocation or must be predesignated by the application)
- Assume either the user's individual authorizations or those of *one* of the user's groups (this choice must be designated by the user or application at the time of subject invocation or must be predesignated by the application)

The system should present users with the appropriate choices, rather than wiring in a particular choice that may not be suited to the needs of all or most applications.

Note that unless the system can restrict a user to having a single subject active at a time, these choices cannot be used to implement roles or separation of duty simply by using groups. To implement roles, an additional capability is needed. This is discussed in the next section.

6.2 Roles

In Section 2.2 we discussed the difficulty of administering security in a relational database system. As is evident from the discussion in the preceding chapters, administering security is far more complex in object-oriented database systems, especially given the degree of flexibility we propose in this report. Unfortunately, added flexibility implies added complexity. Thus, a system taking the approach we advocate here also needs some tools to simplify the administration of security.

We advocate the use of tools similar to the ANSI SQL roles facility and the use of named protection domains (NPDs) described in Section 2.2. For example, a named protection domain can be defined for each application-specific role. The NPD would contain the strong and weak positive and negative authorizations that are required to define each role. The application designer would design each NPD so that it contains only those specific authorizations required by the role.

For example, the named protection domain SALARY-CLERK could contain a weak positive authorization to invoke the increment-salary method on the salary object for the nonmanager instances of salary; it could contain strong negative authorizations for the increment-salary and review-salary methods on the manager instances of salary and for the increment-salary method on the clerk's own salary. The named protection domain SALARY-CLERK could then be granted to the relevant individuals or groups. Thus, the authorizations associated with a salary clerk can be granted and revoked by a security administrator in a straightforward manner, without knowledge of the underlying object hierarchy and its authorization lists. If the system is designed so that only one NPD can be active at any one time, then it becomes possible to enforce a separation-of-duties policy with this mechanism.

6.3 Ownership

Ownership policies generally authorize only the owner of an object for certain operations on the object. The specific set of operations that define ownership may, however, be application dependent. Thus, a facility that allows users to define object ownership in terms of specific strong and weak negative and positive authorizations is desirable. This facility could use the named protection domain concept described above.

In addition to ownership, most database systems include a number of other system-defined roles, such as system administrator, security administrator, and database administrator. The named protection domain facility could also be used to allow users to customize these roles for their needs.

Chapter 7

Conclusions

This report presents a flexible approach to discretionary access control for object-oriented database systems. This approach allows users to implement arbitrary application-specific access control policies. The approach is sufficiently general to allow users to implement policies that are tailored to their application, rather than having to work around a specific policy that is wired into the computer system. Because each such specifiable policy is implemented with a common mechanism, each can be enforced with the same degree of assurance. The capability for a single system to enforce arbitrary application-specific discretionary security policies would materially increase the security of database systems by allowing users to rely on the evaluated system-provided mechanisms to enforce their specific policies, rather than having to encode such policies in the (perhaps untrustworthy) applications themselves.

Future work could take several directions. The model suggested here could be formalized. A specific object-oriented system could be selected for which a design of the discretionary security mechanisms outlined here could be produced. The concept of discretionary constraints could be explored further.

Bibliography

[1] *Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD.* Department of Defense, December 1985.

[2] T. F. Lunt. Access control policies: Some unanswered questions. *Computers and Security*, February 1989.

[3] G. S. Graham and P. J. Denning. Protection—principles and practice. In *Proceedings of the Spring Joint Computer Conference*, volume 40, Montvale, New Jersey, 1972. AFIPS Press.

[4] C. E. Landwehr, C. L. Heitmeyer, and J. McLean. A security model for military message systems. *ACM Transactions on Computer Systems*, 2(3), August 1984.

[5] Robert W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, May 1990.

[6] D. D. Downs, J. R. Rub, K.C. Kung, and C.S. Jordan. Issues in discretionary access control. In *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, 1985.

[7] R. D. Graubart and J. P. L. Woodward. A preliminary naval surveillance DBMS security model. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, April 1982.

[8] T. F. Lunt. Access control policies for database systems. In C. E. Landwehr, editor, *Database Security II: Status and Prospects*. North Holland, 1989.

[9] T. F. Lunt, D. E. Denning, R. R. Schell, W. R. Shockley, and M. Heckman. The SeaView security model. *IEEE Transactions on Software Engineering*, June 1990.

[10] T. F. Lunt, R. R. Schell, W. R. Shockley, M. Heckman, and D. Warren. A near-term design for the SeaView multilevel database system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 1988.

[11] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3), September 1976.

[12] P. A. Rougeau and E. D. Sturms. Sybase secure dataserver: A solution to the multilevel secure DBMS problem. In *Proceedings of the 10th National Computer Security Conference*, September 1987.

[13] B. W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Science and Systems*, March 1971. Reprinted in *ACM Operating Systems Review*, Vol. 8 (1), January 1974.

[14] R. W. Conway, W. L. Maxwell, and H. L. Morgan. On the implementation of security measures in information systems. *Communications of the ACM*, 15(4), April 1972.

[15] E.B. Fernandez, R.C. Summers, and C.D. Coleman. An authorization model for a shared data base. In *Proceedings of the 1975 ACM SIGMOD International Conference*, 1975.

[16] H.R. Hartson and D.K. Hsiao. A semantic model for database protection languages. In *Proceedings of the Second International Conference on VLDB*. North-Holland, 1976.

[17] E. B. Fernandez, R. C. Summers, and C. Wood. *Database Security and Integrity*. Addison-Wesley, Reading, Massachusetts, 1981.

[18] H. Song, E. B. Fernandez, and E. Gudes. Administrative authorization in object-oriented databases. In *Proceedings of the EISS Workshop on Database Security*, European Institute for System Security, Karlsruhe, W. Germany, April 1990.

[19] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. *A Model of Authorization for Next Generation Database Systems*. Technical Report ACA-ST-395-88, MCC, November 1988.

[20] U. Kelter. *Group-Oriented Discretionary Access Controls for Distributed Structurally Object-Oriented Database Systems.* Informatics Report N 93, Fern Universitat Hagen, Hagen, Germany, 1990.

[21] K. R. Dittrich, M. Hartig, and H. Pfefferle. Discretionary access control in structurally object-oriented database systems. In *Proceedings of the 2nd IFIP WG11.3 Workshop on Database Security*, October 1988.

[22] M.M. Larrondo-Petrie, E. Gudes, H. Song, and E. B. Fernandez. Security policies in object-oriented databases. In *Database Security III: Status and Prospects, D.L. Spooner and C. Landwehr (Eds.).* Elsevier, 1990.

[23] E. Gudes, H. Song, , and E. B. Fernandez. Evaluation of negative and predicate-based authorization in object-oriented databases. In *Proceedings of the 4th IFIP WG11.3 Workshop on Database Security*, September 1990.

[24] J. D. Moffet and M. S. Sloman. The source of authority for commercial access control. *Computer*, 21(2), February 1988.

[25] J. D. Moffet and M. S. Sloman. *Delegation of Authority.* Domino rept. B1/IC/4, Dept. of Computing, Imperial College of Science and Technology, London, July 1990.

[26] T. D. Garvey and T. F. Lunt. Multilevel security for knowledge-based systems. In *Proceedings the EISS Workshop on Database Security*, European Institute for System Security, Karlsruhe, W. Germany, April 1990.

[27] T. F. Lunt. Multilevel security for object-oriented database systems. In D. L. Spooner and C. E. Landwehr, editors, *Database Security III: Status and Prospects.* Elsevier, 1990.

[28] J. K. Millen and T. F. Lunt. *Secure Knowledge-Based Systems.* Technical Report SRI-CSL-90-04, Computer Science Laboratory, SRI International, Menlo Park, California, August 1989.

[29] F. Rabitti, D. Woelk, and W. Kim. A model of authorization for object-oriented and semantic databases. In *Proceedings of the International Conference on Extending Database Technology*, 1988.

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.