CHAPTER 4

Access Control

Learning Objectives

After studying this chapter, you should be able to:

- Explain how access control fits into the broader context that includes authentication, authorization, and audit.
- Define the three major categories of access control policies.
- Distinguish among subjects, objects, and access rights.
- Describe the UNIX file access control model.
- Discuss the principal concepts of role-based access control.
- Summarize the RBAC model.
- Discuss the principal concepts of attribute-based access control.
- Explain the identity, credential, and access management model.
- Understand the concept of identity federation and its relationship to a trust framework.

Two definitions of access control are useful in understanding its scope.

1. NISTIR 7298 (*Glossary of Key Information Security Terms*, May 2013), defines access control as the process of granting or denying specific requests to: (1) obtain and use information and related information processing services; and (2) enter specific physical facilities.
2. RFC 4949, *Internet Security Glossary*, defines access control as a process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy.

We can view access control as a central element of computer security. The principal objectives of computer security are to prevent unauthorized users from gaining access to resources, to prevent legitimate users from accessing resources in an unauthorized manner, and to enable legitimate users to access resources in an authorized manner. Table 4.1, from NIST SP 800-171 (*Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations*, August 2016), provides a useful list of security requirements for access control services.

We begin this chapter with an overview of some important concepts. Next we look at three widely used techniques for implementing access control policies. We then turn to a broader perspective of the overall management of access control using identity, credentials, and attributes. Finally, the concept of a trust framework is introduced.

4.1 ACCESS CONTROL PRINCIPLES

In a broad sense, all of computer security is concerned with access control. Indeed, RFC 4949 defines computer security as follows: measures that implement and assure security services in a computer system, particularly those that assure access control

Table 4.1 Access Control Security Requirements (SP 800-171)

Basic Security Requirements	
1	Limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems).
2	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.
Derived Security Requirements	
3	Control the flow of CUI in accordance with approved authorizations.
4	Separate the duties of individuals to reduce the risk of malevolent activity without collusion.
5	Employ the principle of least privilege, including for specific security functions and privileged accounts.
6	Use non-privileged accounts or roles when accessing nonsecurity functions.
7	Prevent non-privileged users from executing privileged functions and audit the execution of such functions.
8	Limit unsuccessful logon attempts.
9	Provide privacy and security notices consistent with applicable CUI rules.
10	Use session lock with pattern-hiding displays to prevent access and viewing of data after period of inactivity.
11	Terminate (automatically) a user session after a defined condition.
12	Monitor and control remote access sessions.
13	Employ cryptographic mechanisms to protect the confidentiality of remote access sessions.
14	Route remote access via managed access control points.
15	Authorize remote execution of privileged commands and remote access to security-relevant information.
16	Authorize wireless access prior to allowing such connections.
17	Protect wireless access using authentication and encryption.
18	Control connection of mobile devices.
19	Encrypt CUI on mobile devices.
20	Verify and control/limit connections to and use of external information systems.
21	Limit use of organizational portable storage devices on external information systems.
22	Control CUI posted or processed on publicly accessible information systems.

CUI = controlled unclassified information

Source: From NIST SP 800-171 Protecting Controlled Unclassified Information in Nonfederal Information Systems and Organizations, December 2016 National Institute of Standards and Technology (NIST), United States Department of Commerce.

service. This chapter deals with a narrower, more specific concept of access control: Access control implements a security policy that specifies who or what (e.g., in the case of a process) may have access to each specific system resource, and the type of access that is permitted in each instance.

Access Control Context

Figure 4.1 shows a broader context of access control. In addition to access control, this context involves the following entities and functions:

- **Authentication:** Verification that the credentials of a user or other system entity are valid.

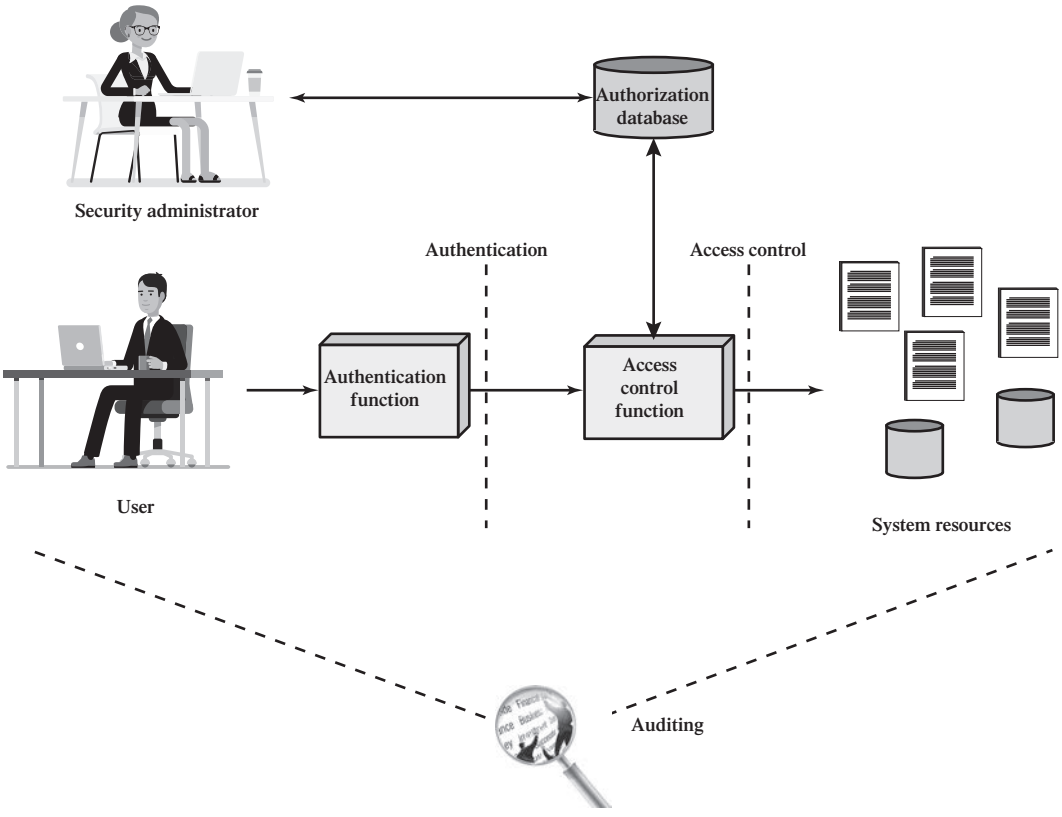


Figure 4.1 Relationship Among Access Control and Other Security Functions
Source: Based on [SAND94].

- **Authorization:** The granting of a right or permission to a system entity to access a system resource. This function determines who is trusted for a given purpose.
- **Audit:** An independent review and examination of system records and activities in order to test for adequacy of system controls, to ensure compliance with established policy and operational procedures, to detect breaches in security, and to recommend any indicated changes in control, policy, and procedures.

An access control mechanism mediates between a user (or a process executing on behalf of a user) and system resources, such as applications, operating systems, firewalls, routers, files, and databases. The system must first authenticate an entity seeking access. Typically, the authentication function determines whether the user is permitted to access the system at all. Then the access control function determines if the specific requested access by this user is permitted. A security administrator maintains an authorization database that specifies what type of access to which resources is allowed for this user. The access control function consults this database to determine whether to grant access. An auditing function monitors and keeps a record of user accesses to system resources.

In the simple model of Figure 4.1, the access control function is shown as a single logical module. In practice, a number of components may cooperatively share the access control function. All operating systems have at least a rudimentary, and in many cases a quite robust, access control component. Add-on security packages can supplement the native access control capabilities of the operating system. Particular applications or utilities, such as a database management system, also incorporate access control functions. External devices, such as firewalls, can also provide access control services.

Access Control Policies

An access control policy, which can be embodied in an authorization database, dictates what types of access are permitted, under what circumstances, and by whom. Access control policies are generally grouped into the following categories:

- **Discretionary access control (DAC):** Controls access based on the identity of the requestor and on access rules (authorizations) stating what requestors are (or are not) allowed to do. This policy is termed *discretionary* because an entity might have access rights that permit the entity, by its own volition, to enable another entity to access some resource.
- **Mandatory access control (MAC):** Controls access based on comparing security labels (which indicate how sensitive or critical system resources are) with security clearances (which indicate system entities are eligible to access certain resources). This policy is termed *mandatory* because an entity that has clearance to access a resource may not, just by its own volition, enable another entity to access that resource.
- **Role-based access control (RBAC):** Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.
- **Attribute-based access control (ABAC):** Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions.

DAC is the traditional method of implementing access control, and is examined in Sections 4.3 and 4.4. MAC is a concept that evolved out of requirements for military information security and is best covered in the context of trusted systems, which we deal with in Chapter 27. Both RBAC and ABAC have become increasingly popular, and are examined in Sections 4.5 and 4.6, respectively.

These four policies are not mutually exclusive. An access control mechanism can employ two or even all three of these policies to cover different classes of system resources.

4.2 SUBJECTS, OBJECTS, AND ACCESS RIGHTS

The basic elements of access control are: subject, object, and access right.

A **subject** is an entity capable of accessing objects. Generally, the concept of subject equates with that of process. Any user or application actually gains access to an object by means of a process that represents that user or application. The process takes on the attributes of the user, such as access rights.

A subject is typically held accountable for the actions they have initiated, and an audit trail may be used to record the association of a subject with security-relevant actions performed on an object by the subject.

Basic access control systems typically define three classes of subject, with different access rights for each class:

- **Owner:** This may be the creator of a resource, such as a file. For system resources, ownership may belong to a system administrator. For project resources, a project administrator or leader may be assigned ownership.
- **Group:** In addition to the privileges assigned to an owner, a named group of users may also be granted access rights, such that membership in the group is sufficient to exercise these access rights. In most schemes, a user may belong to multiple groups.
- **World:** The least amount of access is granted to users who are able to access the system but are not included in the categories owner and group for this resource.

An **object** is a resource to which access is controlled. In general, an object is an entity used to contain and/or receive information. Examples include records, blocks, pages, segments, files, portions of files, directories, directory trees, mailboxes, messages, and programs. Some access control systems also encompass, bits, bytes, words, processors, communication ports, clocks, and network nodes.

The number and types of objects to be protected by an access control system depends on the environment in which access control operates and the desired tradeoff between security on the one hand, and complexity, processing burden, and ease of use on the other hand.

An **access right** describes the way in which a subject may access an object. Access rights could include the following:

- **Read:** User may view information in a system resource (e.g., a file, selected records in a file, selected fields within a record, or some combination). Read access includes the ability to copy or print.
- **Write:** User may add, modify, or delete data in system resource (e.g., files, records, programs). Write access includes read access.
- **Execute:** User may execute specified programs.
- **Delete:** User may delete certain system resources, such as files or records.
- **Create:** User may create new files, records, or fields.
- **Search:** User may list the files in a directory or otherwise search the directory.

4.3 DISCRETIONARY ACCESS CONTROL

As was previously stated, a discretionary access control scheme is one in which an entity may be granted access rights that permit the entity, by its own volition, to enable another entity to access some resource. A general approach to DAC, as exercised by an operating system or a database management system, is that of an **access matrix**. The access matrix concept was formulated by Lampson [LAMP69, LAMP71],

and subsequently refined by Graham and Denning [GRAH72, DENN71] and by Harrison et al. [HARR76].

One dimension of the matrix consists of identified subjects that may attempt data access to the resources. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, network equipment, hosts, or applications instead of or in addition to users. The other dimension lists the objects that may be accessed. At the greatest level of detail, objects may be individual data fields. More aggregate groupings, such as records, files, or even the entire database, may also be objects in the matrix. Each entry in the matrix indicates the access rights of a particular subject for a particular object.

Figure 4.2a, based on a figure in [SAND94], is a simple example of an access matrix. Thus, user A owns files 1 and 3 and has read and write access rights to those files. User B has read access rights to file 1, and so on.

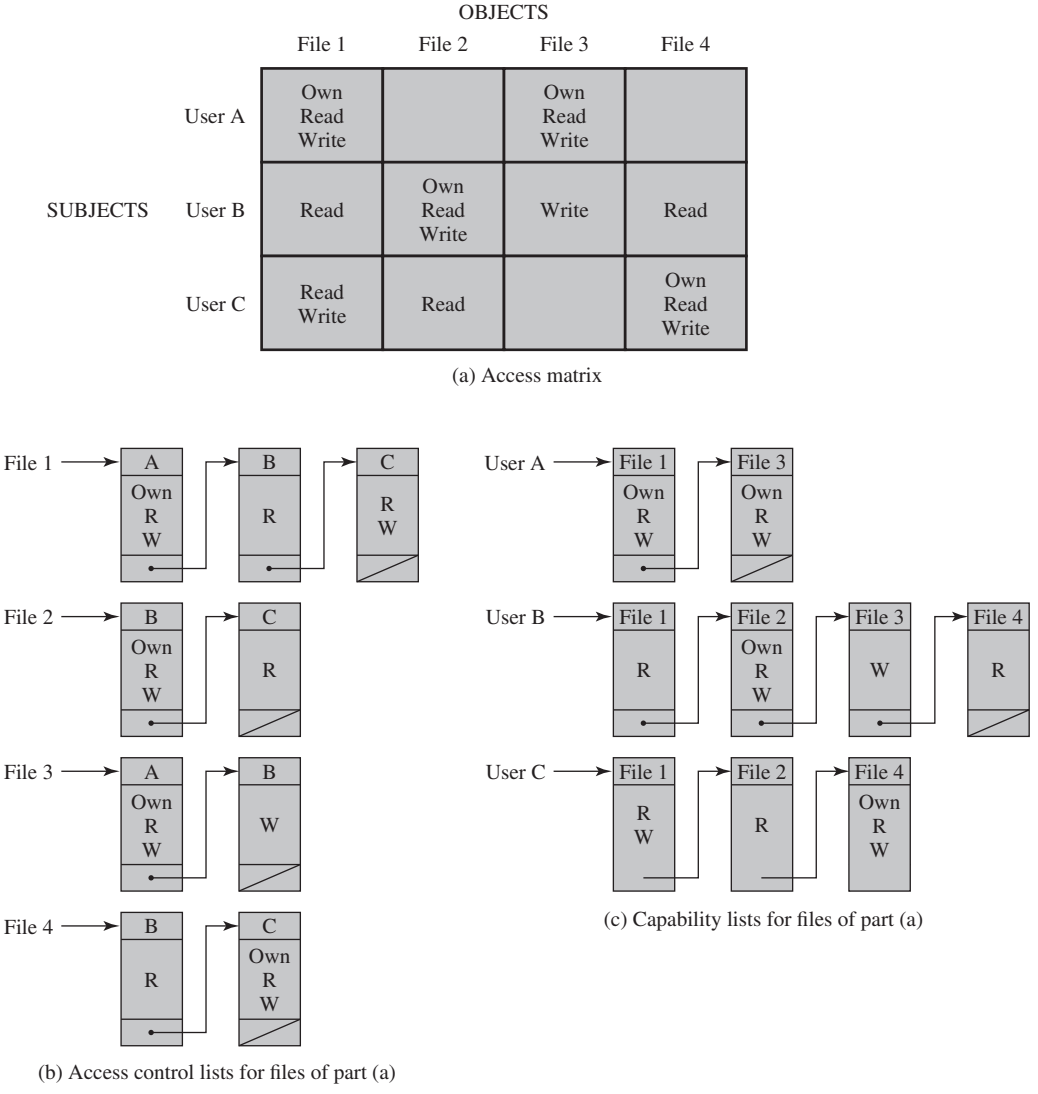
In practice, an access matrix is usually sparse and is implemented by decomposition in one of two ways. The matrix may be decomposed by columns, yielding **access control lists** (ACLs) (see Figure 4.2b). For each object, an ACL lists users and their permitted access rights. The ACL may contain a default, or public, entry. This allows users that are not explicitly listed as having special rights to have a default set of rights. The default set of rights should always follow the rule of least privilege or read-only access, whichever is applicable. Elements of the list may include individual users as well as groups of users.

When it is desired to determine which subjects have which access rights to a particular resource, ACLs are convenient, because each ACL provides the information for a given resource. However, this data structure is not convenient for determining the access rights available to a specific user.

Decomposition by rows yields **capability tickets** (see Figure 4.2c). A capability ticket specifies authorized objects and operations for a particular user. Each user has a number of tickets and may be authorized to loan or give them to others. Because tickets may be dispersed around the system, they present a greater security problem than access control lists. The integrity of the ticket must be protected, and guaranteed (usually by the operating system). In particular, the ticket must be unforgeable. One way to accomplish this is to have the operating system hold all tickets on behalf of users. These tickets would have to be held in a region of memory inaccessible to users. Another alternative is to include an unforgeable token in the capability. This could be a large random password, or a cryptographic message authentication code. This value is verified by the relevant resource whenever access is requested. This form of capability ticket is appropriate for use in a distributed environment, when the security of its contents cannot be guaranteed.

The convenient and inconvenient aspects of capability tickets are the opposite of those for ACLs. It is easy to determine the set of access rights that a given user has, but more difficult to determine the list of users with specific access rights for a specific resource.

[SAND94] proposes a data structure that is not sparse, like the access matrix, but is more convenient than either ACLs or capability lists (see Table 4.2). An authorization table contains one row for one access right of one subject to one resource. Sorting or accessing the table by subject is equivalent to a capability list. Sorting or accessing the table by object is equivalent to an ACL. A relational database can easily implement an authorization table of this type.



(a) Access matrix

(b) Access control lists for files of part (a)

(c) Capability lists for files of part (a)

Figure 4.2 Example of Access Control Structures

An Access Control Model

This section introduces a general model for DAC developed by Lampson, Graham, and Denning [LAMP71, GRAH72, DENN71]. The model assumes a set of subjects, a set of objects, and a set of rules that govern the access of subjects to objects. Let us define the protection state of a system to be the set of information, at a given point in time, that specifies the access rights for each subject with respect to each object. We can identify three requirements: representing the protection state, enforcing access rights, and allowing subjects to alter the protection state in certain ways. The model addresses all three requirements, giving a general, logical description of a DAC system.

Table 4.2 Authorization Table for Files in Figure 4.2

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

To represent the protection state, we extend the universe of objects in the access control matrix to include the following:

- **Processes:** Access rights include the ability to delete a process, stop (block), and wake up a process.
- **Devices:** Access rights include the ability to read/write the device, to control its operation (e.g., a disk seek), and to block/unblock the device for use.
- **Memory locations or regions:** Access rights include the ability to read/write certain regions of memory that are protected such that the default is to disallow access.
- **Subjects:** Access rights with respect to a subject have to do with the ability to grant or delete access rights of that subject to other objects, as explained subsequently.

Figure 4.3 is an example. For an access control matrix A, each entry $A[S, X]$ contains strings, called access attributes, that specify the access rights of subject S to object X. For example, in Figure 4.3, S_1 may read file F_1, because 'read' appears in $A[S_1, F_1]$.

From a logical or functional point of view, a separate access control module is associated with each type of object (see Figure 4.4). The module evaluates each

		OBJECTS							
		Subjects			Files		Processes		Disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
SUBJECTS	S_1	control	owner	owner control	read*	read owner	wakeup	wakeup	seek	owner
	S_2		control		write*	execute			owner	seek*
	S_3			control		write	stop			

* = copy flag set

Figure 4.3 Extended Access Control Matrix

request by a subject to access an object to determine if the access right exists. An access attempt triggers the following steps:

1. A subject S_0 issues a request of type α for object X.
2. The request causes the system (the operating system or an access control interface module of some sort) to generate a message of the form (S_0, α, X) to the controller for X.
3. The controller interrogates the access matrix A to determine if α is in $A[S_0, X]$. If so, the access is allowed; if not, the access is denied and a protection violation occurs. The violation should trigger a warning and appropriate action.

Figure 4.4 suggests that every access by a subject to an object is mediated by the controller for that object, and that the controller's decision is based on the current contents of the matrix. In addition, certain subjects have the authority to make specific changes to the access matrix. A request to modify the access matrix is treated as an access to the matrix, with the individual entries in the matrix treated as objects. Such accesses are mediated by an access matrix controller, which controls updates to the matrix.

The model also includes a set of rules that govern modifications to the access matrix, as shown in Table 4.3. For this purpose, we introduce the access rights 'owner' and 'control' and the concept of a copy flag, as explained in the subsequent paragraphs.

The first three rules deal with transferring, granting, and deleting access rights. Suppose the entry α* exists in $A[S_0, X]$. This means S_0 has access right α to subject X and, because of the presence of the copy flag, can transfer this right, with or without copy flag, to another subject. Rule R1 expresses this capability. A subject would transfer the access right without the copy flag if there were a concern that the new subject would maliciously transfer the right to another subject that should not have that access right. For example, S_1 may place 'read' or 'read*' in any matrix entry in the F_1 column. Rule R2 states that if S_0 is designated as the owner of object X, then S_0 can grant an access right to that object for any other subject. Rule R2 states that

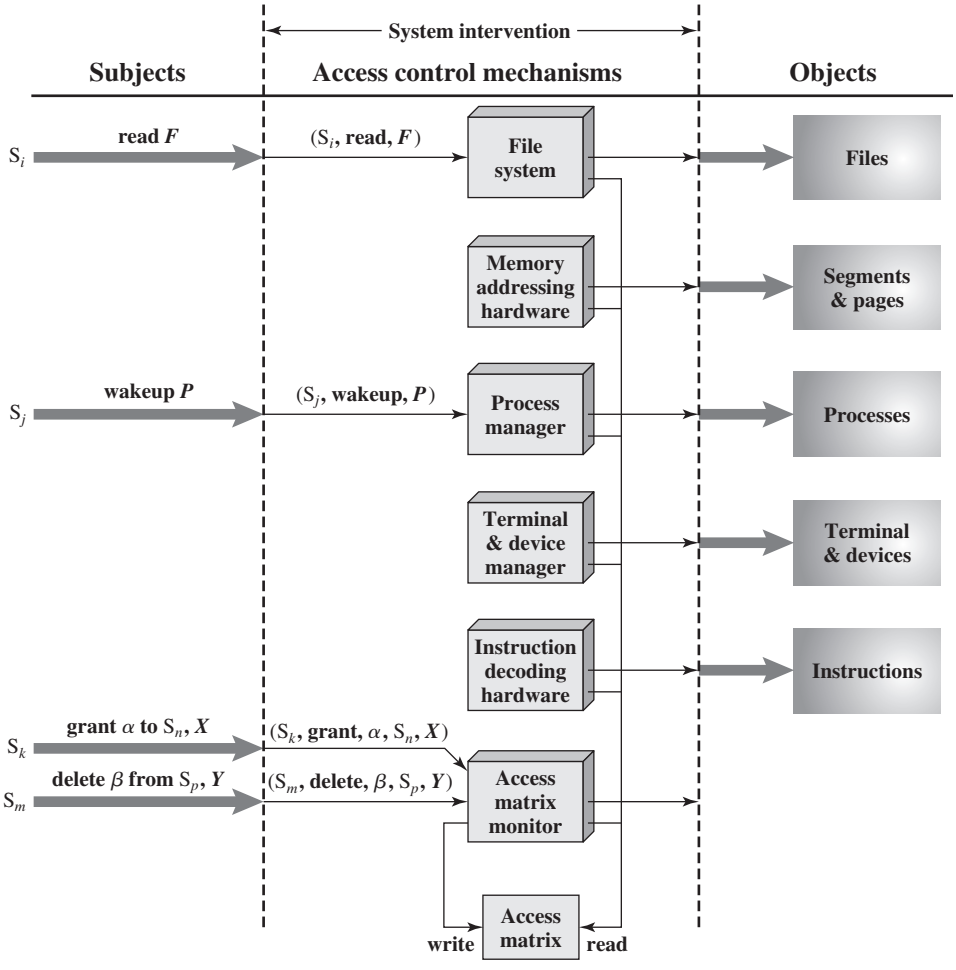


Figure 4.4 **An Organization of the Access Control Function**

S_0 can add any access right to $A[S, X]$ for any S, if S_0 has 'owner' access to X. Rule R3 permits S_0 to delete any access right from any matrix entry in a row for which S_0 controls the subject, and for any matrix entry in a column for which S_0 owns the object. Rule R4 permits a subject to read that portion of the matrix that it owns or controls.

The remaining rules in Table 4.3 govern the creation and deletion of subjects and objects. Rule R5 states that any subject can create a new object, which it owns, and can then grant and delete access to the object. Under Rule R6, the owner of an object can destroy the object, resulting in the deletion of the corresponding column of the access matrix. Rule R7 enables any subject to create a new subject; the creator owns the new subject and the new subject has control access to itself. Rule R8 permits the owner of a subject to delete the row and column (if there are subject columns) of the access matrix designated by that subject.

The set of rules in Table 4.3 is an example of the rule set that could be defined for an access control system. The following are examples of additional or alternative

Table 4.3 **Access Control System Commands**

Rule	Command (by S_0)	Authorization	Operation
R1	**transfer** $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ **to** S, X	"$\alpha*$" in $A[S_0, X]$	store $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R2	**grant** $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ **to** S, X	'owner' in $A[S_0, X]$å	store $\left\{\begin{matrix}\alpha* \\ \alpha\end{matrix}\right\}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_0, S]$ or 'owner' in $A[S_0, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_0, X]$
R6	**destroy object** X	'owner' in $A[S_0, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_0, S]$	delete row for S from A; execute **destroy object** S

rules that could be included. A transfer-only right could be defined, which results in the transferred right being added to the target subject and deleted from the transferring subject. The number of owners of an object or a subject could be limited to one by not allowing the copy flag to accompany the owner right.

The ability of one subject to create another subject and to have 'owner' access right to that subject can be used to define a hierarchy of subjects. For example, in Figure 4.3, S_1 owns S_2 and S_3, so S_2 and S_3 are subordinate to S_1. By the rules of Table 4.3, S_1 can grant and delete to S_2 access rights that S_1 already has. Thus, a subject can create another subject with a subset of its own access rights. This might be useful, for example, if a subject is invoking an application that is not fully trusted and does not want that application to be able to transfer access rights to other subjects.

Protection Domains

The access control matrix model that we have discussed so far associates a set of capabilities with a user. A more general and more flexible approach, proposed in [LAMP71], is to associate capabilities with protection domains. A protection domain is a set of objects together with access rights to those objects. In terms of the access matrix, a row defines a protection domain. So far, we have equated each row with a specific user. So, in this limited model, each user has a protection domain, and any processes spawned by the user have access rights defined by the same protection domain.

A more general concept of protection domain provides more flexibility. For example, a user can spawn processes with a subset of the access rights of the user, defined as a new protection domain. This limits the capability of the process. Such a scheme could be used by a server process to spawn processes for different classes of users. Also, a user could define a protection domain for a program that is not fully trusted, so its access is limited to a safe subset of the user's access rights.

The association between a process and a domain can be static or dynamic. For example, a process may execute a sequence of procedures and require different access rights for each procedure, such as read file and write file. In general, we would like to minimize the access rights that any user or process has at any one time; the use of protection domains provides a simple means to satisfy this requirement.

One form of protection domain has to do with the distinction made in many operating systems, such as UNIX, between user and kernel mode. A user program executes in a **user mode**, in which certain areas of memory are protected from the user's use and in which certain instructions may not be executed. When the user process calls a system routine, that routine executes in a system mode, or what has come to be called **kernel mode**, in which privileged instructions may be executed and in which protected areas of memory may be accessed.

4.4 EXAMPLE: UNIX FILE ACCESS CONTROL

For our discussion of UNIX file access control, we first introduce several basic concepts concerning UNIX files and directories.

All types of UNIX files are administered by the operating system by means of inodes. An inode (index node) is a control structure that contains the key information needed by the operating system for a particular file. Several file names may be associated with a single inode, but an active inode is associated with exactly one file, and each file is controlled by exactly one inode. The attributes of the file as well as its permissions and other control information are stored in the inode. On the disk, there is an inode table, or inode list, that contains the inodes of all the files in the file system. When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.

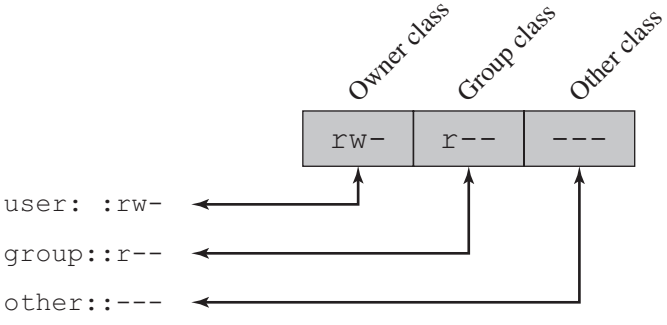
Directories are structured in a hierarchical tree. Each directory can contain files and/or other directories. A directory that is inside another directory is referred to as a subdirectory. A directory is simply a file that contains a list of file names plus pointers to associated inodes. Thus, associated with each directory is its own inode.

Traditional UNIX File Access Control

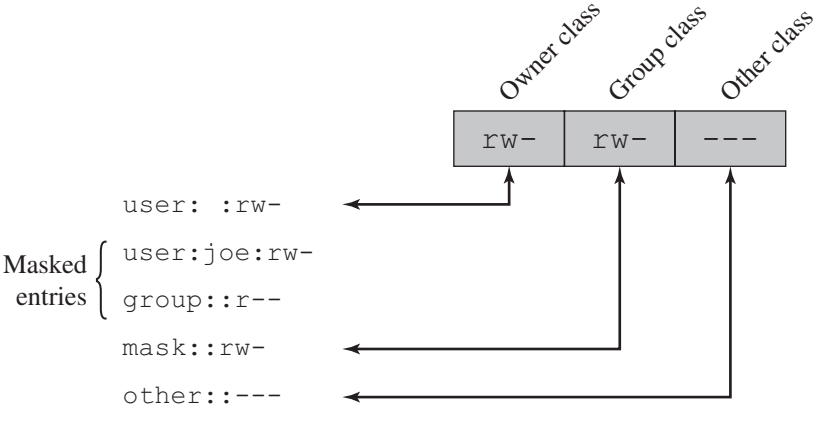
Most UNIX systems depend on, or at least are based on, the file access control scheme introduced with the early versions of UNIX. Each UNIX user is assigned a unique user identification number (user ID). A user is also a member of a primary group, and possibly a number of other groups, each identified by a group ID. When a file is created, it is designated as owned by a particular user and marked with that user's ID. It also belongs to a specific group, which initially is either its creator's primary group, or the group of its parent directory if that directory has SetGID permission

set. Associated with each file is a set of 12 protection bits. The owner ID, group ID, and protection bits are part of the file's inode.

Nine of the protection bits specify read, write, and execute permission for the owner of the file, other members of the group to which this file belongs, and all other users. These form a hierarchy of owner, group, and all others, with the highest relevant set of permissions being used. Figure 4.5a shows an example in which the file owner has read and write access; all other members of the file's group have read access; and users outside the group have no access rights to the file. When applied to a directory, the read and write bits grant the right to list and to create/rename/delete files in the directory.[1] The execute bit grants the right to descend into the directory or search it for a filename.



(a) Traditional UNIX approach (minimal access control list)



(b) Extended access control list

Figure 4.5 UNIX File Access Control

[1]Note that the permissions that apply to a directory are distinct from those that apply to any file or directory it contains. The fact that a user has the right to write to the directory does not give the user the right to write to a file in that directory. That is governed by the permissions of the specific file. The user would, however, have the right to rename the file.

The remaining three bits define special additional behavior for files or directories. Two of these are the "set user ID" (SetUID) and "set group ID" (SetGID) permissions. If these are set on an executable file, the operating system functions as follows. When a user (with execute privileges for this file) executes the file, the system temporarily allocates the rights of the user's ID of the file creator, or the file's group, respectively, to those of the user executing the file. These are known as the "effective user ID" and "effective group ID" and are used in addition to the "real user ID" and "real group ID" of the executing user when making access control decisions for this program. This change is only effective while the program is being executed. This feature enables the creation and use of privileged programs that may use files normally inaccessible to other users. It enables users to access certain files in a controlled fashion. Alternatively, when applied to a directory, the SetGID permission indicates that newly created files will inherit the group of this directory. The SetUID permission is ignored.

The final permission bit is the "sticky" bit. When set on a file, this originally indicated that the system should retain the file contents in memory following execution. This is no longer used. When applied to a directory, though, it specifies that only the owner of any file in the directory can rename, move, or delete that file. This is useful for managing files in shared temporary directories.

One particular user ID is designated as "superuser." The superuser is exempt from the usual file access control constraints and has systemwide access. Any program that is owned by, and SetUID to, the "superuser" potentially grants unrestricted access to the system to any user executing that program. Hence great care is needed when writing such programs.

This access scheme is adequate when file access requirements align with users and a modest number of groups of users. For example, suppose a user wants to give read access for file X to users A and B, and read access for file Y to users B and C. We would need at least two user groups, and user B would need to belong to both groups in order to access the two files. However, if there are a large number of different groupings of users requiring a range of access rights to different files, then a very large number of groups may be needed to provide this. This rapidly becomes unwieldy and difficult to manage, if even possible at all.[2] One way to overcome this problem is to use access control lists, which are provided in most modern UNIX systems.

A final point to note is that the traditional UNIX file access control scheme implements a simple protection domain structure. A domain is associated with the user, and switching the domain corresponds to changing the user ID temporarily.

Access Control Lists in UNIX

Many modern UNIX and UNIX-based operating systems support access control lists, including FreeBSD, OpenBSD, Linux, and Solaris. In this section, we describe FreeBSD, but other implementations have essentially the same features and interface. The feature is referred to as extended access control list, while the traditional UNIX approach is referred to as minimal access control list.

[2]Most UNIX systems impose a limit on the maximum number of groups to which any user may belong, as well as to the total number of groups possible on the system.

FreeBSD allows the administrator to assign a list of UNIX user IDs and groups to a file by using the `setfacl` command. Any number of users and groups can be associated with a file, each with three protection bits (read, write, execute), offering a flexible mechanism for assigning access rights. A file need not have an ACL but may be protected solely by the traditional UNIX file access mechanism. FreeBSD files include an additional protection bit that indicates whether the file has an extended ACL.

FreeBSD and most UNIX implementations that support extended ACLs use the following strategy (e.g., Figure 4.5b):

1. The owner class and other class entries in the 9-bit permission field have the same meaning as in the minimal ACL case.
2. The group class entry specifies the permissions for the owner group for this file. These permissions represent the maximum permissions that can be assigned to named users or named groups, other than the owning user. In this latter role, the group class entry functions as a mask.
3. Additional named users and named groups may be associated with the file, each with a 3-bit permission field. The permissions listed for a named user or named group are compared to the mask field. Any permission for the named user or named group that is not present in the mask field is disallowed.

When a process requests access to a file system object, two steps are performed. Step 1 selects the ACL entry that most closely matches the requesting process. The ACL entries are looked at in the following order: owner, named users, (owning or named) groups, others. Only a single entry determines access. Step 2 checks if the matching entry contains sufficient permissions. A process can be a member in more than one group; so more than one group entry can match. If any of these matching group entries contain the requested permissions, one that contains the requested permissions is picked (the result is the same no matter which entry is picked). If none of the matching group entries contains the requested permissions, access will be denied no matter which entry is picked.

4.5 ROLE-BASED ACCESS CONTROL

Traditional DAC systems define the access rights of individual users and groups of users. In contrast, RBAC is based on the roles that users assume in a system rather than the user's identity. Typically, RBAC models define a role as a job function within an organization. RBAC systems assign access rights to roles instead of individual users. In turn, users are assigned to different roles, either statically or dynamically, according to their responsibilities.

RBAC now enjoys widespread commercial use and remains an area of active research. The National Institute of Standards and Technology (NIST) has issued a standard, FIPS PUB 140-3 (*Security Requirements for Cryptographic Modules*, September 2009), that requires support for access control and administration through roles.

The relationship of users to roles is many to many, as is the relationship of roles to resources, or system objects (see Figure 4.6). The set of users changes, in some

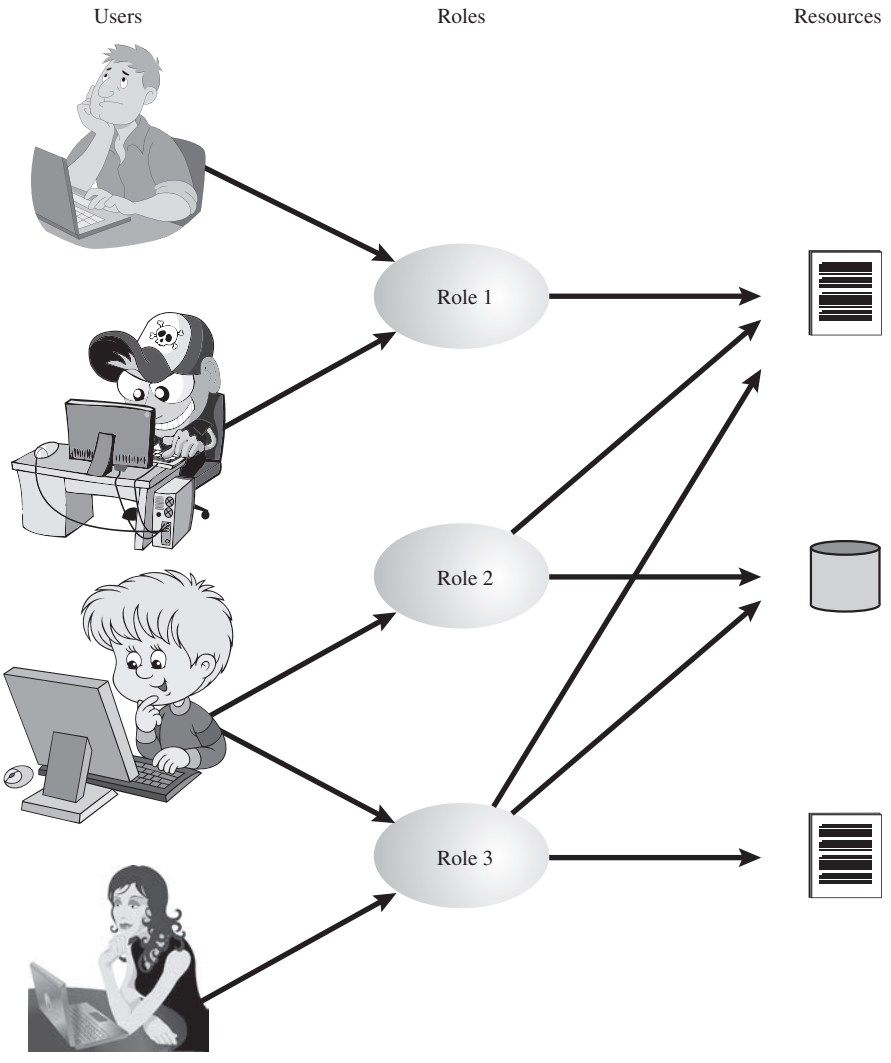
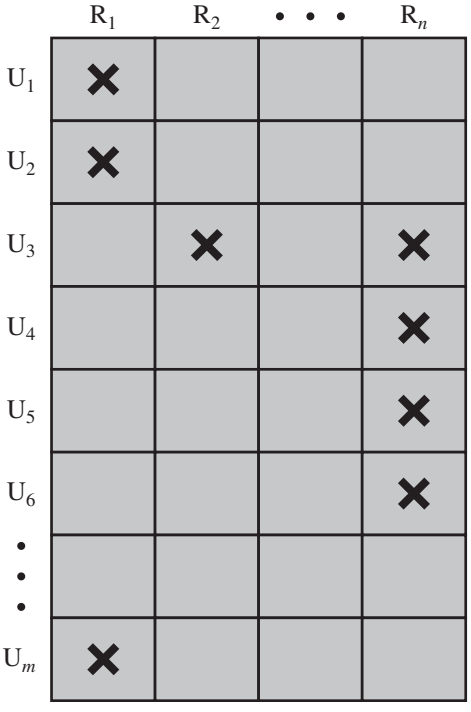


Figure 4.6 **Users, Roles, and Resources**

environments frequently, and the assignment of a user to one or more roles may also be dynamic. The set of roles in the system in most environments is relatively static, with only occasional additions or deletions. Each role will have specific access rights to one or more resources. The set of resources and the specific access rights associated with a particular role are also likely to change infrequently.

We can use the access matrix representation to depict the key elements of an RBAC system in simple terms, as shown in Figure 4.7. The upper matrix relates individual users to roles. Typically there are many more users than roles. Each matrix entry is either blank or marked, the latter indicating that this user is assigned to this role. Note a single user may be assigned multiple roles (more than one mark in a row) and multiple users may be assigned to a single role (more than one mark in a



OBJECTS

ROLES	R_1	R_2	R_n	F_1	F_2	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
• • •									
R_n			control		write	stop			

Figure 4.7 Access Control Matrix Representation of RBAC

column). The lower matrix has the same structure as the DAC access control matrix, with roles as subjects. Typically, there are few roles and many objects, or resources. In this matrix, the entries are the specific access rights enjoyed by the roles. Note a role can be treated as an object, allowing the definition of role hierarchies.

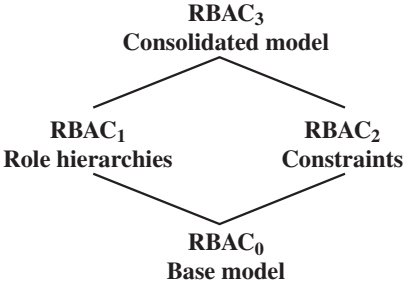
RBAC lends itself to an effective implementation of the principle of least privilege, referred to in Chapter 1. Each role should contain the minimum set of access

rights needed for that role. A user is assigned to a role that enables him or her to perform only what is required for that role. Multiple users assigned to the same role enjoy the same minimal set of access rights.

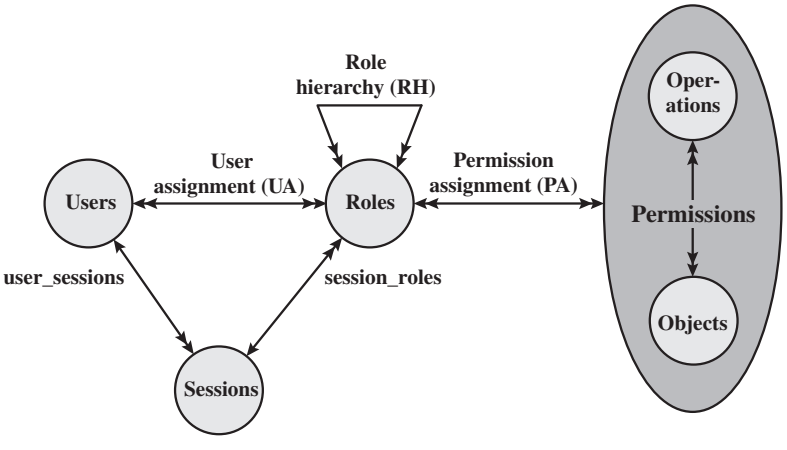
RBAC Reference Models

A variety of functions and services can be included under the general RBAC approach. To clarify the various aspects of RBAC, it is useful to define a set of abstract models of RBAC functionality.

[SAND96] defines a family of reference models that has served as the basis for ongoing standardization efforts. This family consists of four models that are related to each other, as shown in Figure 4.8a and Table 4.4. $RBAC_0$ contains the minimum functionality for an RBAC system. $RBAC_1$ includes the $RBAC_0$ functionality and adds role hierarchies, which enable one role to inherit permissions from another role. $RBAC_2$ includes $RBAC_0$ and adds constraints, which restrict the ways in which the



(a) Relationship among RBAC models



(b) RBAC models

Figure 4.8 A Family of Role-Based Access Control Models $RBAC_0$ is the minimum requirement for an RBAC system. $RBAC_1$ adds role hierarchies and $RBAC_2$ adds constraints. $RBAC_3$ includes $RBAC_1$ and $RBAC_2$.

Table 4.4 **Scope RBAC Models**

Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes

components of an RBAC system may be configured. $RBAC_3$ contains the functionality of $RBAC_0$, $RBAC_1$, and $RBAC_2$.

Base Model—RBAC$_0$ Figure 4.8b, without the role hierarchy and constraints, contains the four types of entities in an $RBAC_0$ system:

- **User:** An individual that has access to this computer system. Each individual has an associated user ID.
- **Role:** A named job function within the organization that controls this computer system. Typically, associated with each role is a description of the authority and responsibility conferred on this role, and on any user who assumes this role.
- **Permission:** An approval of a particular mode of access to one or more objects. Equivalent terms are *access right*, *privilege*, and *authorization*.
- **Session:** A mapping between a user and an activated subset of the set of roles to which the user is assigned.

The arrowed lines in Figure 4.8b indicate relationships, or mappings, with a single arrowhead indicating one, and a double arrowhead indicating many. Thus, there is a many-to-many relationship between users and roles: One user may have multiple roles, and multiple users may be assigned to a single role. Similarly, there is a many-to-many relationship between roles and permissions. A session is used to define a temporary one-to-many relationship between a user and one or more of the roles to which the user has been assigned. The user establishes a session with only the roles needed for a particular task; this is an example of the concept of least privilege.

The many-to-many relationships between users and roles and between roles and permissions provide a flexibility and granularity of assignment not found in conventional DAC schemes. Without this flexibility and granularity, there is a greater risk that a user may be granted more access to resources than is needed because of the limited control over the types of access that can be allowed. The NIST RBAC document gives the following examples: Users may need to list directories and modify existing files without creating new files, or they may need to append records to a file without modifying existing records.

Role Hierarchies—RBAC$_1$ Role hierarchies provide a means of reflecting the hierarchical structure of roles in an organization. Typically, job functions with greater responsibility have greater authority to access resources. A subordinate job function may have a subset of the access rights of the superior job function. Role hierarchies make use of the concept of inheritance to enable one role to implicitly include access rights associated with a subordinate role.

Figure 4.9 is an example of a diagram of a role hierarchy. By convention, subordinate roles are lower in the diagram. A line between two roles implies the upper role includes all of the access rights of the lower role, as well as other access rights not available to the lower role. One role can inherit access rights from multiple subordinate roles. For example, in Figure 4.9, the Project Lead role includes all of the access rights of the Production Engineer role and of the Quality Engineer role. More than one role can inherit from the same subordinate role. For example, both the Production Engineer role and the Quality Engineer role include all of the access rights of the Engineer role. Additional access rights are also assigned to the Production Engineer Role, and a different set of additional access rights are assigned to the Quality Engineer role. Thus, these two roles have overlapping access rights, namely, the access rights they share with the Engineer role.

*CONSTRAINTS—**RBAC$_2$*** Constraints provide a means of adapting RBAC to the specifics of administrative and security policies in an organization. A constraint is a defined relationship among roles or a condition related to roles. [SAND96] lists the following types of constraints: mutually exclusive roles, cardinality, and prerequisite roles.

Mutually exclusive roles are roles such that a user can be assigned to only one role in the set. This limitation could be a static one, or it could be dynamic, in the sense that a user could be assigned only one of the roles in the set for a session. The mutually exclusive constraint supports a separation of duties and capabilities within an organization. This separation can be reinforced or enhanced by use of mutually exclusive permission assignments. With this additional constraint, a mutually exclusive set of roles has the following properties:

1. A user can only be assigned to one role in the set (either during a session or statically).
2. Any permission (access right) can be granted to only one role in the set.

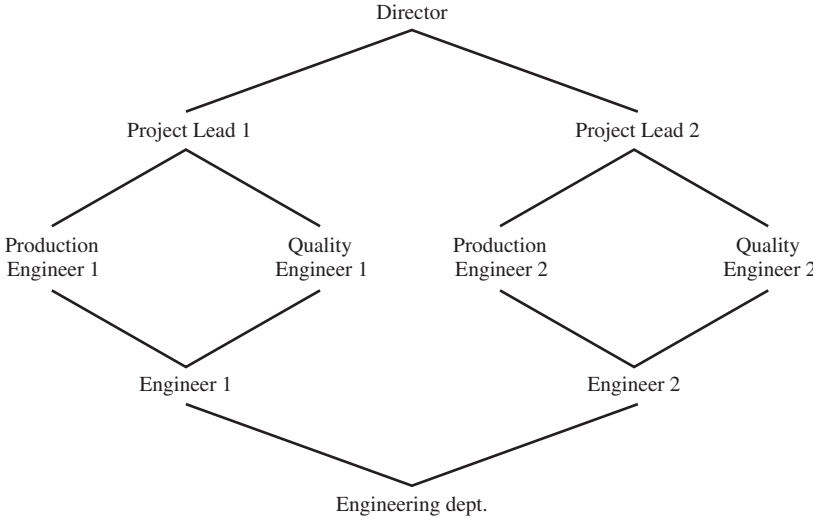


Figure 4.9 Example of Role Hierarchy

Thus, the set of mutually exclusive roles have non overlapping permissions. If two users are assigned to different roles in the set, then the users have non overlapping permissions while assuming those roles. The purpose of mutually exclusive roles is to increase the difficulty of collusion among individuals of different skills or divergent job functions to thwart security policies.

Cardinality refers to setting a maximum number with respect to roles. One such constraint is to set a maximum number of users that can be assigned to a given role. For example, a project leader role or a department head role might be limited to a single user. The system could also impose a constraint on the number of roles that a user is assigned to, or the number of roles a user can activate for a single session. Another form of constraint is to set a maximum number of roles that can be granted a particular permission; this might be a desirable risk mitigation technique for a sensitive or powerful permission.

A system might be able to specify a **prerequisite role**, which dictates a user can only be assigned to a particular role if it is already assigned to some other specified role. A prerequisite can be used to structure the implementation of the least privilege concept. In a hierarchy, it might be required that a user can be assigned to a senior (higher) role only if it is already assigned an immediately junior (lower) role. For example, in Figure 4.9 a user assigned to a Project Lead role must also be assigned to the subordinate Production Engineer and Quality Engineer roles. Then, if the user does not need all of the permissions of the Project Lead role for a given task, the user can invoke a session using only the required subordinate role. Note the use of prerequisites tied to the concept of hierarchy requires the $RBAC_3$ model.

4.6 ATTRIBUTE-BASED ACCESS CONTROL

A relatively recent development in access control technology is the attribute-based access control (ABAC) model. An ABAC model can define authorizations that express conditions on properties of both the resource and the subject. For example, consider a configuration in which each resource has an attribute that identifies the subject that created the resource. Then, a single access rule can specify the ownership privilege for all the creators of every resource. The strength of the ABAC approach is its flexibility and expressive power. [PLAT13] points out that the main obstacle to its adoption in real systems has been concern about the performance impact of evaluating predicates on both resource and user properties for each access. However, for applications such as cooperating Web services and cloud computing, this increased performance cost is less noticeable because there is already a relatively high performance cost for each access. Thus, Web services have been pioneering technologies for implementing ABAC models, especially through the introduction of the eXtensible Access Control Markup Language (XAMCL) [BEUC13], and there is considerable interest in applying the ABAC model to cloud services [IQBA12, YANG12].

There are three key elements to an ABAC model: attributes, which are defined for entities in a configuration; a policy model, which defines the ABAC policies; and the architecture model, which applies to policies that enforce access control. We will examine these elements in turn.

Attributes

Attributes are characteristics that define specific aspects of the subject, object, environment conditions, and/or requested operations that are predefined and preassigned by an authority. Attributes contain information that indicates the class of information given by the attribute, a name, and a value (e.g., Class = HospitalRecordsAccess, Name = PatientInformationAccess, Value = MFBusinessHoursOnly).

The following are the three types of attributes in the ABAC model:

- **Subject attributes:** A subject is an active entity (e.g., a user, an application, a process, or a device) that causes information to flow among objects or changes the system state. Each subject has associated attributes that define the identity and characteristics of the subject. Such attributes may include the subject's identifier, name, organization, job title, and so on. A subject's role can also be viewed as an attribute.
- **Object attributes:** An object, also referred to as a **resource**, is a passive (in the context of the given request) information system–related entity (e.g., devices, files, records, tables, processes, programs, networks, domains) containing or receiving information. As with subjects, objects have attributes that can be leveraged to make access control decisions. A Microsoft Word document, for example, may have attributes such as title, subject, date, and author. Object attributes can often be extracted from the metadata of the object. In particular, a variety of Web service metadata attributes may be relevant for access control purposes, such as ownership, service taxonomy, or even Quality of Service (QoS) attributes.
- **Environment attributes:** These attributes have so far been largely ignored in most access control policies. They describe the operational, technical, and even situational environment or context in which the information access occurs. For example, attributes, such as current date and time, the current virus/hacker activities, and the network's security level (e.g., Internet vs. intranet), are not associated with a particular subject nor a resource, but may nonetheless be relevant in applying an access control policy.

ABAC is a logical access control model that is distinguishable because it controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment relevant to a request. ABAC relies upon the evaluation of attributes of the subject, attributes of the object, and a formal relationship or access control rule defining the allowable operations for subject-object attribute combinations in a given environment. All ABAC solutions contain these basic core capabilities to evaluate attributes and enforce rules or relationships between those attributes. ABAC systems are capable of enforcing DAC, RBAC, and MAC concepts. ABAC enables fine-grained access control, which allows for a higher number of discrete inputs into an access control decision, providing a bigger set of possible combinations of those variables to reflect a larger and more definitive set of possible rules, policies, or restrictions on access. Thus, ABAC allows an unlimited number of attributes to be combined to satisfy any access control rule. Moreover, ABAC systems can be implemented to satisfy a wide array of requirements from basic access control lists through advanced expressive policy models that fully leverage the flexibility of ABAC.

ABAC Logical Architecture

Figure 4.10 illustrates in a logical architecture the essential components of an ABAC system. An access by a subject to an object proceeds according to the following steps:

1. A subject requests access to an object. This request is routed to an access control mechanism.
2. The access control mechanism is governed by a set of rules (2a) that are defined by a preconfigured access control policy. Based on these rules, the access control mechanism assesses the attributes of the subject (2b), object (2c), and current environmental conditions (2d) to determine authorization.
3. The access control mechanism grants the subject access to the object if access is authorized, and denies access if it is not authorized.

It is clear from the logical architecture that there are four independent sources of information used for the access control decision. The system designer can decide which attributes are important for access control with respect to subjects, objects, and

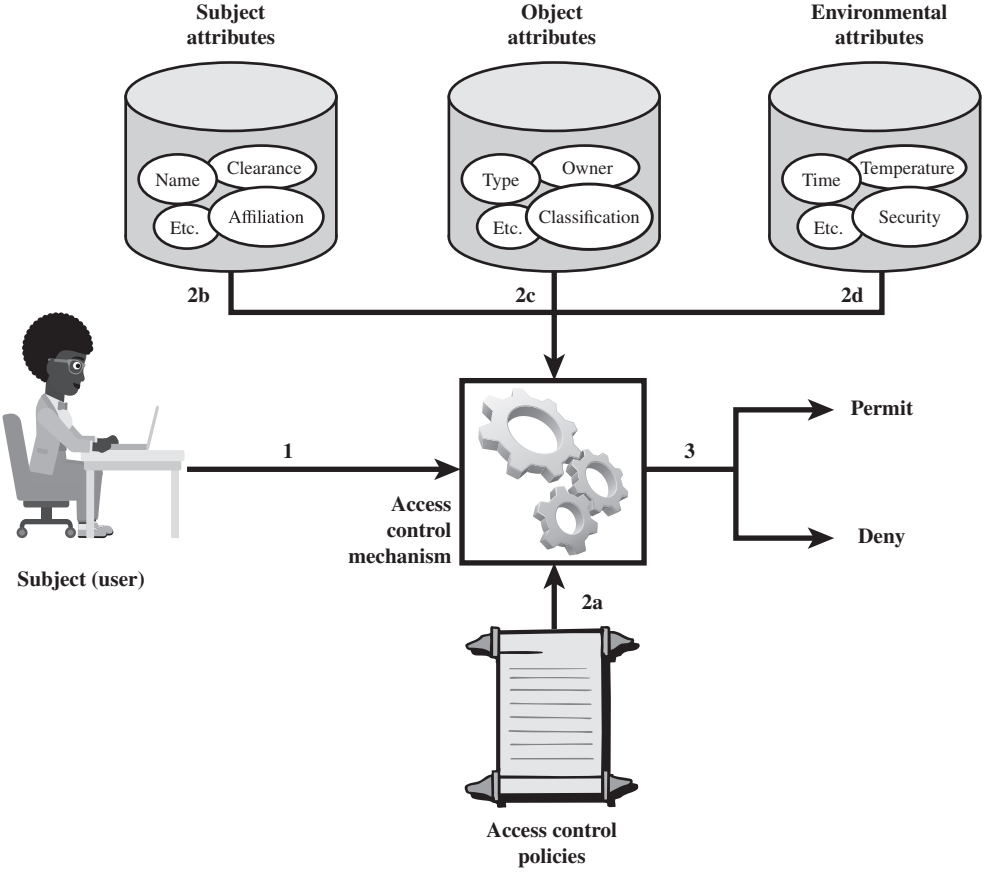


Figure 4.10 **ABAC Scenario**

environmental conditions. The system designer or other authority can then define access control policies, in the form of rules, for any desired combination of attributes of subject, object, and environmental conditions. It should be evident that this approach is very powerful and flexible. However, the cost, both in terms of the complexity of the design and implementation, and in terms of the performance impact, is likely to exceed that of other access control approaches. This is a trade-off that the system authority must make.

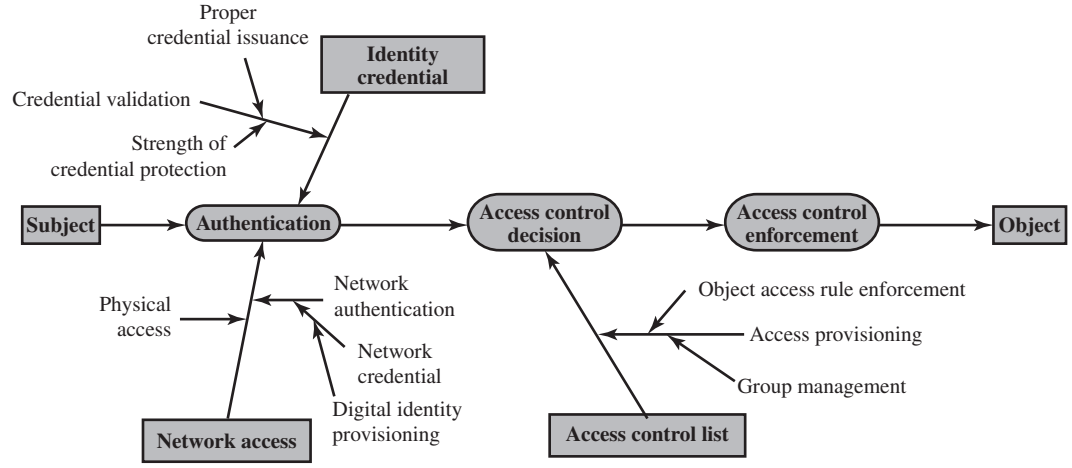
Figure 4.11, taken from NIST SP 800-162 [*Guide to Attribute Based Access Control (ABAC) Definition and Considerations,* January 2014], provides a useful way of grasping the scope of an ABAC model compared to a DAC model using access control lists (ACLs). This figure not only illustrates the relative complexity of the two models, but also clarifies the trust requirements of the two models. A comparison of representative trust relationships (indicated by arrowed lines) for ACL use and ABAC use shows that there are many more complex trust relationships required for ABAC to work properly. Ignoring the commonalities in both parts of Figure 4.11, one can observe that with ACLs the root of trust is with the object owner, who ultimately enforces the object access rules by provisioning access to the object through addition of a user to an ACL. In ABAC, the root of trust is derived from many sources of which the object owner has no control, such as Subject Attribute Authorities, Policy Developers, and Credential Issuers. Accordingly, SP 800-162 recommended that an enterprise governance body be formed to manage all identity, credential, and access management capability deployment and operation and that each subordinate organization maintain a similar body to ensure consistency in managing the deployment and paradigm shift associated with enterprise ABAC implementation. Additionally, it is recommended that an enterprise develop a trust model that can be used to illustrate the trust relationships and help determine ownership and liability of information and services, needs for additional policy and governance, and requirements for technical solutions to validate or enforce trust relationships. The trust model can be used to help influence organizations to share their information with clear expectations of how that information will be used and protected and to be able to trust the information and attribute and authorization assertions coming from other organizations.

ABAC Policies

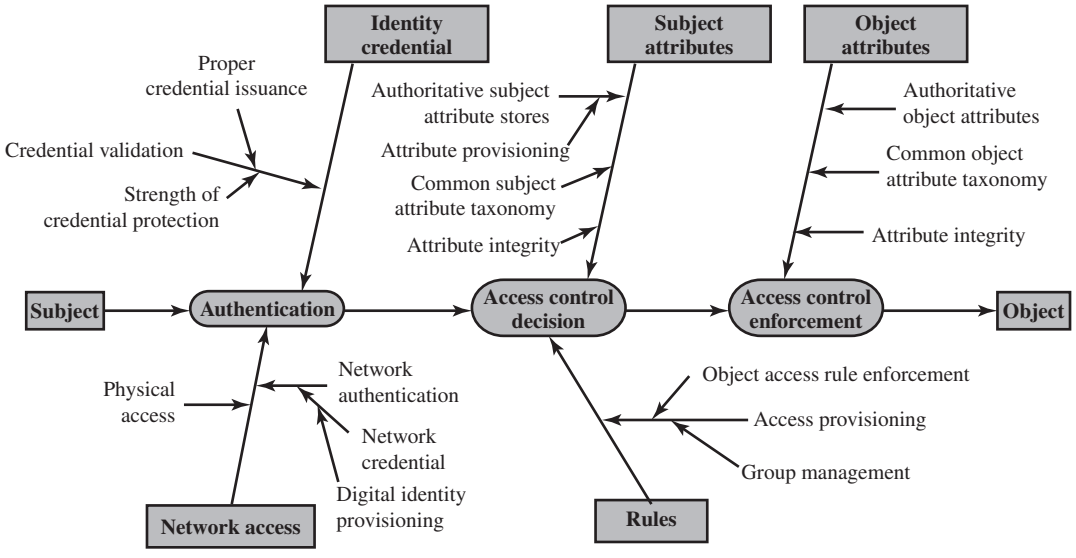
A **policy** is a set of rules and relationships that govern allowable behavior within an organization, based on the privileges of subjects and how resources or objects are to be protected under which environment conditions. In turn, **privileges** represent the authorized behavior of a subject; they are defined by an authority and embodied in a policy. Other terms that are commonly used instead of privileges are **rights**, **authorizations**, and **entitlements**. Policy is typically written from the perspective of the object that needs protecting, and the privileges available to subjects.

We now define an ABAC policy model, based on the model presented in [YUAN05]. The following conventions are used:

1. S, O, and E are subjects, objects, and environments, respectively;
2. SA_k $(1 \leq k \leq K)$, OA_m $(1 \leq m \leq M)$, and EA_n $(1 \leq n \leq N)$ are the predefined attributes for subjects, objects, and environments, respectively;



(a) ACL Trust Chain



(b) ABAC Trust Chain

Figure 4.11 ACL and ABAC Trust Relationships

3. ATTR(s), ATTR(o), and ATTR(e) are attribute assignment relations for subject s, object o, and environment e, respectively:

$$\text{ATTR}(s) \subseteq \text{SA}_1 \times \text{SA}_2 \times \ldots \times \text{SA}_K$$
$$\text{ATTR}(r) \subseteq \text{OA}_1 \times \text{OA}_2 \times \ldots \times \text{OA}_M$$
$$\text{ATTR}(o) \subseteq \text{EA}_1 \times \text{EA}_2 \times \ldots \times \text{EA}_N$$

We also use the function notation for the value assignment of individual attributes. For example:

```
Role(s) = "Service Consumer"
ServiceOwner(o) = "XYZ, Inc."
CurrentDate(e) = "01-23-2005"
```

4. In the most general form, a Policy Rule, which decides on whether a subject *s* can access an object *o* in a particular environment *e*, is a Boolean function of the attributes of *s*, *o*, and *e*:

   $\text{Rule: can\_access } (s, o, e) \leftarrow f(\text{ATTR}(s), \text{ATTR}(o), \text{ATTR}(e))$

   Given all the attribute assignments of *s*, *o*, and *e*, if the function's evaluation is true, then the access to the resource is granted; otherwise the access is denied.

5. A policy rule base or policy store may consist of a number of policy rules, covering many subjects and objects within a security domain. The access control decision process in essence amounts to the evaluation of applicable policy rules in the policy store.

Now consider the example of an online entertainment store that streams movies to users for a flat monthly fee. We will use this example to contrast RBAC and ABAC approaches. The store must enforce the following access control policy based on the user's age and the movie's content rating:

Movie Rating	Users Allowed Access
R	Age 17 and older
PG-13	Age 13 and older
G	Everyone

In an RBAC model, every user would be assigned one of three roles: Adult, Juvenile, or Child, possibly during registration. There would be three permissions created: Can view R-rated movies, Can view PG-13-rated movies, and Can view G-rated movies. The Adult role gets assigned with all three permissions; the Juvenile role gets Can view PG-13-rated movies and Can view G-rated movies permissions, and the Child role gets the Can view G-rated movies permission only. Both the user-to-role and permission-to-role assignments are manual administrative tasks.

The ABAC approach to this application does not need to explicitly define roles. Instead, whether a user *u* can access or view a movie *m* (in a security environment *e* which is ignored here) would be resolved by evaluating a policy rule such as the following:

$$
\begin{aligned}
&\text{R1:can\_access}(u, m, e) \leftarrow \\
&\quad (\text{Age}(u) \geq 17 \wedge \text{Rating}(m) \in \{\text{R, PG-13, G}\}) \vee \\
&\quad (\text{Age}(u) \geq 13 \wedge \text{Age}(u) < 17 \wedge \text{Rating(m)} \in \{\text{PG-13, G}\}) \vee \\
&\quad (\text{Age}(u) < 13 \wedge \text{Rating}(m) \in \{\text{G}\})
\end{aligned}
$$

where Age and Rating are the subject attribute and the object attribute, respectively. The advantage of the ABAC model shown here is that it eliminates the definition and management of static roles, hence eliminating the need for the administrative tasks for user-to-role assignment and permission-to-role assignment.

The advantage of ABAC is more clearly seen when we impose finer-grained policies. For example, suppose movies are classified as either New Release or Old Release, based on release date compared to the current date, and users are classified as Premium User and Regular User, based on the fee they pay. We would like to enforce a policy that only premium users can view new movies. For the RBAC model, we would have to double the number of roles, to distinguish each user by age and fee, and we would have to double the number of separate permissions as well.

In general, if there are K subject attributes and M object attributes, and if for each attribute, Range() denotes the range of possible values it can take, then the respective number of roles and permissions required for an RBAC model are:

$$\prod_{k=1}^{K} Range\,(SA_k) \quad \text{and} \quad \prod_{m=1}^{M} Range\,(SA_m)$$

Thus, we can see that as the number of attributes increases to accommodate finer-grained policies, the number of roles and permissions grows exponentially. In contrast, the ABAC model deals with additional attributes in an efficient way. For this example, the policy R1 defined previously still applies. We need two new rules:

$$\begin{aligned}
&R2: can_access(u, m, e) \leftarrow \\
&\quad (MembershipType(u) = Premium) \vee \\
&\quad (MembershipType(u) = Regular \wedge MovieType(m) = OldRelease) \\
&R3: can_access(u, m, e) \leftarrow R1 \wedge R2
\end{aligned}$$

With the ABAC model, it is also easy to add environmental attributes. Suppose we wish to add a new policy rule that is expressed in words as follows: *Regular users are allowed to view new releases in promotional periods*. This would be difficult to express in an RBAC model. In an ABAC model, we only need to add a conjunctive (AND) rule that checks to see the environmental attribute *today's date* falls in a promotional period.

4.7 IDENTITY, CREDENTIAL, AND ACCESS MANAGEMENT

We now examine some concepts that are relevant to an access control approach centered on attributes. This section provides an overview of the concept of identity, credential, and access management (ICAM), and then Section 4.8 will discuss the use of a trust framework for exchanging attributes.

ICAM is a comprehensive approach to managing and implementing digital identities (and associated attributes), credentials, and access control. ICAM has been developed by the U.S. government, but is applicable not only to government agencies,

but also may be deployed by enterprises looking for a unified approach to access control. ICAM is designed to:

- Create trusted digital identity representations of individuals and what the ICAM documents refer to as nonperson entities (NPEs). The latter include processes, applications, and automated devices seeking access to a resource.
- Bind those identities to credentials that may serve as a proxy for the individual or NPE in access transactions. A credential is an object or data structure that authoritatively binds an identity (and optionally, additional attributes) to a token possessed and controlled by a subscriber.
- Use the credentials to provide authorized access to an agency's resources.

Figure 4.12 provides an overview of the logical components of an ICAM architecture. We will examine each of the main components in the following subsections.

Identity Management

Identity management is concerned with assigning attributes to a digital identity and connecting that digital identity to an individual or NPE. The goal is to establish a

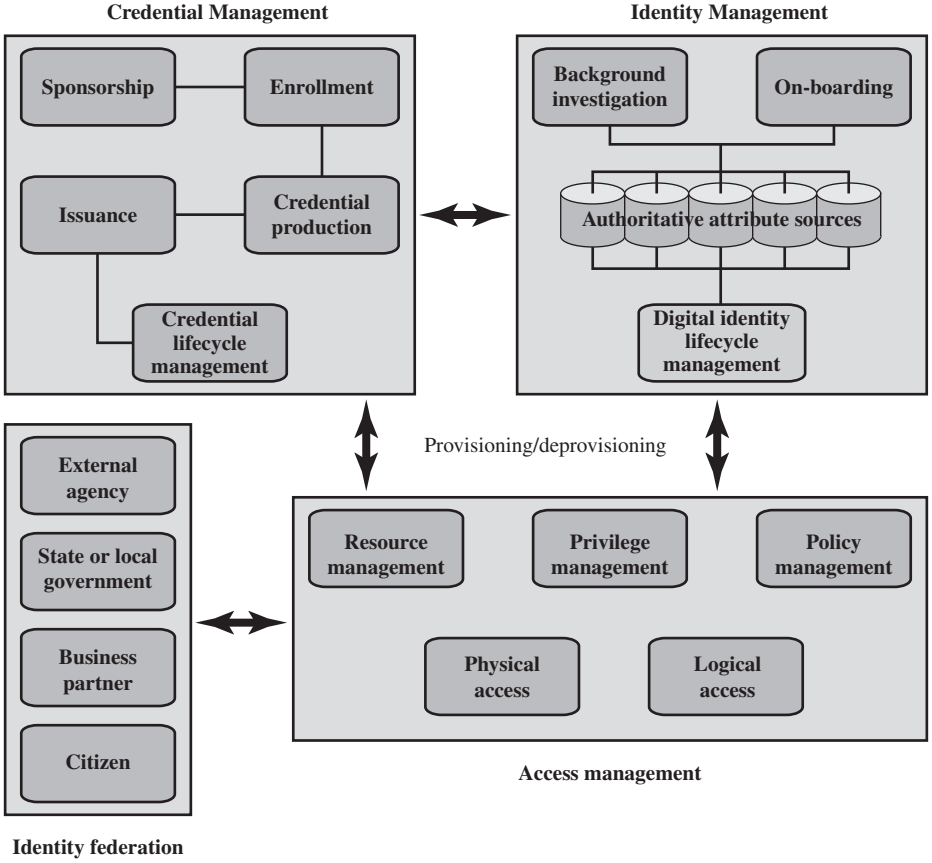


Figure 4.12 Identity, Credential, and Access Management (ICAM)

trustworthy digital identity that is independent of a specific application or context. The traditional, and still most common, approach to access control for applications and programs is to create a digital representation of an identity for the specific use of the application or program. As a result, maintenance and protection of the identity itself is treated as secondary to the mission associated with the application. Further, there is considerable overlap in effort in establishing these application-specific identities.

Unlike accounts used to log on to networks, systems, or applications, enterprise identity records are not tied to job title, job duties, location, or whether access is needed to a specific system. Those items may become attributes tied to an enterprise identity record, and may also become part of what uniquely identifies an individual in a specific application. Access control decisions will be based on the context and relevant attributes of a user—not solely their identity. The concept of an enterprise identity is that individuals will have a single digital representation of themselves that can be leveraged across departments and agencies for multiple purposes, including access control.

Figure 4.12 depicts the key functions involved in identity management. Establishment of a digital identity typically begins with collecting identity data as part of an enrollment process. A digital identity is often comprised of a set of attributes that when aggregated uniquely identify a user within a system or an enterprise. In order to establish trust in the individual represented by a digital identity, an agency may also conduct a background investigation. Attributes about an individual may be stored in various authoritative sources within an agency and linked to form an enterprise view of the digital identity. This digital identity may then be provisioned into applications in order to support physical and logical access (part of Access Management) and de-provisioned when access is no longer required.

A final element of identity management is lifecycle management, which includes the following:

- Mechanisms, policies, and procedures for protecting personal identity information
- Controlling access to identity data
- Techniques for sharing authoritative identity data with applications that need it
- Revocation of an enterprise identity

Credential Management

As mentioned, a credential is an object or data structure that authoritatively binds an identity (and optionally, additional attributes) to a token possessed and controlled by a subscriber. Examples of credentials are smart cards, private/public cryptographic keys, and digital certificates. Credential management is the management of the life cycle of the credential. Credential management encompasses the following five logical components:

1. An authorized individual sponsors an individual or entity for a credential to establish the need for the credential. For example, a department supervisor sponsors a department employee.
2. The sponsored individual enrolls for the credential, a process which typically consists of identity proofing and the capture of biographic and biometric data. This

step may also involve incorporating authoritative attribute data, maintained by the identity management component.

3. A credential is produced. Depending on the credential type, production may involve encryption, the use of a digital signature, the production of a smartcard, or other functions.
4. The credential is issued to the individual or NPE.
5. Finally, a credential must be maintained over its life cycle, which might include revocation, reissuance/replacement, reenrollment, expiration, personal identification number (PIN) reset, suspension, or reinstatement.

Access Management

The access management component deals with the management and control of the ways entities are granted access to resources. It covers both logical and physical access, and may be internal to a system or an external element. The purpose of access management is to ensure that the proper identity verification is made when an individual attempts to access security-sensitive buildings, computer systems, or data. The access control function makes use of credentials presented by those requesting access and the digital identity of the requestor. Three support elements are needed for an enterprise-wide access control facility:

- **Resource management:** This element is concerned with defining rules for a resource that requires access control. The rules would include credential requirements and what user attributes, resource attributes, and environmental conditions are required for access of a given resource for a given function.
- **Privilege management:** This element is concerned with establishing and maintaining the entitlement or privilege attributes that comprise an individual's access profile. These attributes represent features of an individual that can be used as the basis for determining access decisions to both physical and logical resources. Privileges are considered attributes that can be linked to a digital identity.
- **Policy management:** This element governs what is allowable and unallowable in an access transaction. That is, given the identity and attributes of the requestor, the attributes of the resource or object, and environmental conditions, a policy specifies what actions this user can perform on this object.

Identity Federation

Identity federation addresses two questions:

1. How do you trust identities of individuals from external organizations who need access to your systems?
2. How do you vouch for identities of individuals in your organization when they need to collaborate with external organizations?

Identity federation is a term used to describe the technology, standards, policies, and processes that allow an organization to trust digital identities, identity attributes, and credentials created and issued by another organization. We will discuss identity federation in the following section.

4.8 TRUST FRAMEWORKS

The interrelated concepts of trust, identity, and attributes have become core concerns of Internet businesses, network service providers, and large enterprises. These concerns can clearly be seen in the e-commerce setting. For efficiency, privacy, and legal simplicity, parties to transactions generally apply the need-to-know principle: What do you need to know about someone in order to deal with them? The answer varies from case to case, and includes such attributes as professional registration or license number, organization and department, staff ID, security clearance, customer reference number, credit card number, unique health identifier, allergies, blood type, Social Security number, address, citizenship status, social networking handle, pseudonym, and so on. The attributes of an individual that must be known and verified to permit a transaction depend on context.

The same concern for attributes is increasingly important for all types of access control situations, not just the e-business context. For example, an enterprise may need to provide access to resources for customers, users, suppliers, and partners. Depending on context, access will be determined not just by identity, but by the attributes of the requestor and the resource.

Traditional Identity Exchange Approach

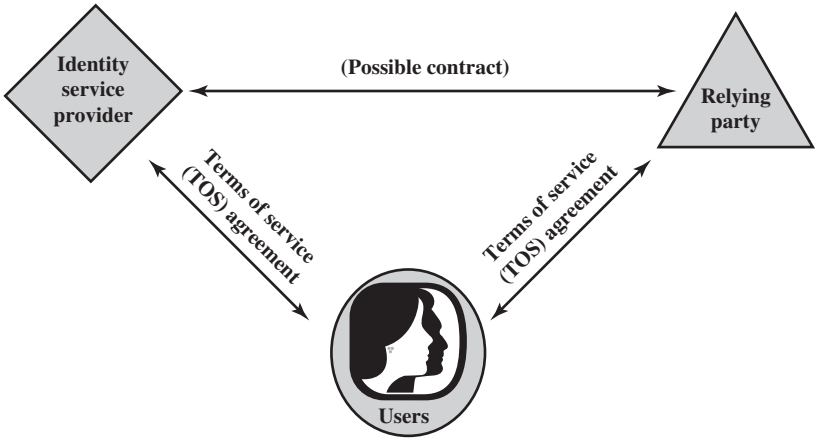
Online or network transactions involving parties from different organizations, or between an organization and an individual user such as an online customer, generally require the sharing of identity information. This information may include a host of associated attributes in addition to a simple name or numerical identifier. Both the party disclosing the information and the party receiving the information need to have a level of trust about security and privacy issues related to that information.

Figure 4.13a shows the traditional technique for the exchange of identity information. This involves users developing arrangements with an **identity service provider** to procure digital identity and credentials, and arrangements with parties that provide end-user services and applications and that are willing to rely on the identity and credential information generated by the identity service provider.

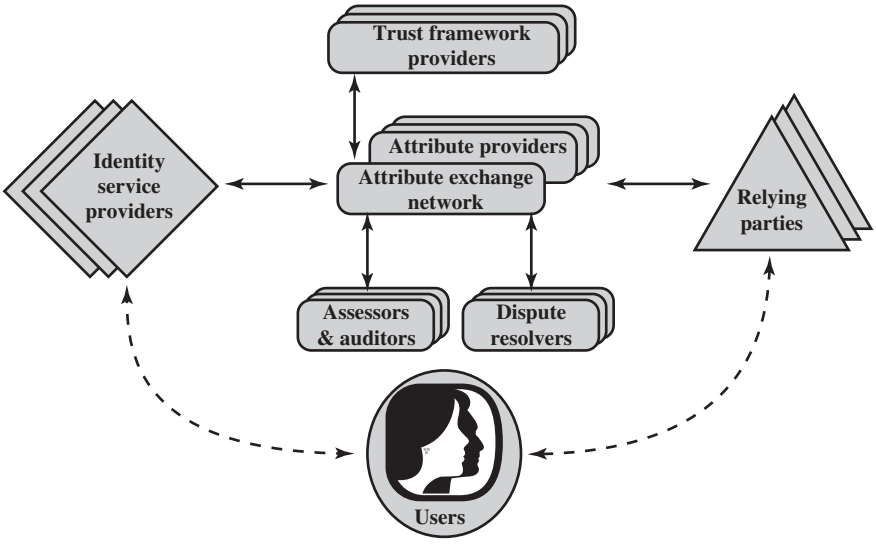
The arrangement of Figure 4.13a must meet a number of requirements. The **relying party** requires that the user has been authenticated to some degree of assurance, that the attributes imputed to the user by the identity service provider are accurate, and that the identity service provider is authoritative for those attributes. The identity service provider requires assurance that it has accurate information about the user and that, if it shares information, the relying party will use it in accordance with contractual terms and conditions and the law. The user requires assurance that the identity service provider and relying party can be entrusted with sensitive information and that they will abide by the user's preferences and respect the user's privacy. Most importantly, all the parties want to know if the practices described by the other parties are actually those implemented by the parties, and how reliable those parties are.

Open Identity Trust Framework

Without some universal standard and framework, the arrangement of Figure 4.13a must be replicated in multiple contexts. A far preferable approach is to develop an open,



(a) Traditional triangle of parties involved in an exchange of identity information



(b) Identity attribute exchange elements

Figure 4.13 **Identity Information Exchange Approaches**

standardized approach to trustworthy identity and attribute exchange. In the remainder of this section, we examine such an approach that is gaining increasing acceptance.

Unfortunately, this topic is burdened with numerous acronyms, so it is best to begin with a definition of the most important of these:

- **OpenID:** This is an open standard that allows users to be authenticated by certain cooperating sites (known as Relying Parties) using a third party service, eliminating the need for Webmasters to provide their own ad hoc systems and allowing users to consolidate their digital identities. Users may create accounts with their preferred OpenID identity providers, then use those accounts as the basis for signing on to any Web site that accepts OpenID authentication.

- **OIDF:** The OpenID Foundation is an international nonprofit organization of individuals and companies committed to enabling, promoting, and protecting OpenID technologies. OIDF assists the community by providing needed infrastructure and help in promoting and supporting expanded adoption of OpenID.
- **ICF:** The Information Card Foundation is a nonprofit community of companies and individuals working together to evolve the Information Card ecosystem. Information Cards are personal digital identities people can use online, and the key component of identity metasystems. Visually, each Information Card has a card-shaped picture and a card name associated with it that enable people to organize their digital identities and to easily select one they want to use for any given interaction.
- **OITF:** The Open Identity Trust Framework is a standardized, open specification of a trust framework for identity and attribute exchange, developed jointly by OIDF and ICF.
- **OIX:** The Open Identity Exchange Corporation is an independent, neutral, international provider of certification trust frameworks conforming to the Open Identity Trust Frameworks model.
- **AXN:** An Attribute Exchange Network (AXN) is an online Internet-scale gateway for identity service providers and relying parties to efficiently access user-asserted, permissioned, and verified online identity attributes in high volumes at affordable costs.

System managers need to be able to trust that the attributes associated with a subject or an object are authoritative and are exchanged securely. One approach to providing that trust within an organization is the ICAM model, specifically the ICAM components (see Figure 4.12). Combined with an identity federation functionality that is shared with other organizations, attributes can be exchanged in a trust-worthy fashion, supporting secure access control.

In digital identity systems, a **trust framework** functions as a certification program. It enables a party who accepts a digital identity credential (called the relying party) to trust the identity, security, and privacy policies of the party who issues the credential (called the identity service provider) and vice versa. More formally, OIX defines a trust framework as a set of verifiable commitments from each of the various parties in a transaction to their counter parties. These commitments include (1) controls (including regulatory and contractual obligations) to help ensure commitments are delivered and (2) remedies for failure to meet such commitments. A trust framework is developed by a community whose members have similar goals and perspectives. It defines the rights and responsibilities of that community's participants; specifies the policies and standards specific to the community; and defines the community-specific processes and procedures that provide assurance. Different trust frameworks can exist, and sets of participants can tailor trust frameworks to meet their particular needs.

Figure 4.13b shows the elements involved in the OITF. Within any given organization or agency, the following roles are part of the overall framework:

- **Relying parties (RPs):** Also called service providers, these are entities delivering services to specific users. RPs must have confidence in the identities and/or

attributes of their intended users, and must rely upon the various credentials presented to evince those attributes and identities.

- **Subjects:** These are users of an RP's services, including customers, employees, trading partners, and subscribers.
- **Attribute providers (APs):** APs are entities acknowledged by the community of interest as being able to verify given attributes as presented by subjects and which are equipped through the AXN to create conformant attribute credentials according to the rules and agreements of the AXN. Some APs will be sources of authority for certain information; more commonly APs will be brokers of derived attributes.
- **Identity providers (IDPs):** These are entities able to authenticate user credentials and to vouch for the names (or pseudonyms or handles) of subjects, and which are equipped through the AXN or some other compatible Identity and Access Management (IDAM) system to create digital identities that may be used to index user attributes.

There are also the following important support elements as part on an AXN:

- **Assessors:** Assessors evaluate identity service providers and RPs and certify that they are capable of following the OITF provider's blueprint.
- **Auditors:** These entities may be called on to check that parties' practices have been in line with what was agreed for the OITF.
- **Dispute resolvers:** These entities provide arbitration and dispute resolution under OIX guidelines.
- **Trust framework providers:** A trust framework provider is an organization that translates the requirements of policymakers into an own blueprint for a trust framework that it then proceeds to build, doing so in a way that is consistent with the minimum requirements set out in the OITF specification. In almost all cases, there will be a reasonably obvious candidate organization to take on this role, for each industry sector or large organization that decides it is appropriate to interoperate with an AXN.

The solid arrowed lines in Figure 4.13b indicate agreements with the trust framework provider for implementing technical, operations, and legal requirements. The dashed arrowed lines indicate other agreements potentially affected by these requirements. In general terms, the model illustrated in Figure 4.13b would operate in the following way. Responsible persons within participating organizations determine the technical, operational, and legal requirements for exchanges of identity information that fall under their authority. They then select OITF providers to implement these requirements. These OITF providers translate the requirements into a blueprint for a trust framework that may include additional conditions of the OITF provider. The OITF provider vets identity service providers and RPs and contracts with them to follow its trust framework requirements when conducting exchanges of identity information. The contracts carry provisions relating to dispute resolvers, and auditors for contract interpretation and enforcement.

4.9 CASE STUDY: RBAC SYSTEM FOR A BANK

The Dresdner Bank has implemented an RBAC system that serves as a useful practical example [SCHA01]. The bank uses a variety of computer applications. Many of these were initially developed for a mainframe environment; some of these older applications are now supported on a client-server network, while others remain on mainframes. There are also newer applications on servers. Prior to 1990, a simple DAC system was used on each server and mainframe. Administrators maintained a local access control file on each host and defined the access rights for each employee on each application on each host. This system was cumbersome, time-consuming, and error-prone. To improve the system, the bank introduced an RBAC scheme, which is systemwide and in which the determination of access rights is compartmentalized into three different administrative units for greater security.

Roles within the organization are defined by a combination of official position and job function. Table 4.5a provides examples. This differs somewhat from the concept of role in the NIST standard, in which a role is defined by a job function. To some extent, the difference is a matter of terminology. In any case, the bank's role structuring leads to a natural means of developing an inheritance hierarchy based on official position. Within the bank, there is a strict partial ordering of official positions within each organization, reflecting a hierarchy of responsibility and power. For example, the positions Head of Division, Group Manager, and Clerk are in descending order. When the official position is combined with job function, there is a resulting ordering of access rights, as indicated in Table 4.5b. Thus, the financial analyst/Group Manager role (role B) has more access rights than the financial analyst/Clerk role (role A). The table indicates that role B has as many or more access rights than role A in three applications and has access rights to a fourth application. On the other hand, there is no hierarchical relationship between office banking/Group Manager and financial analyst/Clerk because they work in different functional areas. We can therefore define a role hierarchy in which one role is superior to another if its position is superior and their functions are identical. The role hierarchy makes it possible to economize on access rights definitions, as suggested in Table 4.5c.

In the original scheme, the direct assignment of access rights to the individual user occurred at the application level and was associated with the individual application. In the new scheme, an application administration determines the set of access rights associated with each individual application. However, a given user performing a given task may not be permitted all of the access rights associated with the application. When a user invokes an application, the application grants access on the basis of a centrally provided security profile. A separate authorization administration associated access rights with roles, and creates the security profile for a use on the basis of the user's role.

A user is statically assigned a role. In principle (in this example), each user may be statically assigned up to four roles and select a given role for use in invoking a particular application. This corresponds to the NIST concept of session. In practice, most users are statically assigned a single role based on the user's position and job function.

All of these ingredients are depicted in Figure 4.14. The Human Resource Department assigns a unique User ID to each employee who will be using the system.

Table 4.5 Functions and Roles for Banking Example

(a) Functions and Official Positions

Role	Function	Official Position
A	financial analyst	Clerk
B	financial analyst	Group Manager
C	financial analyst	Head of Division
D	financial analyst	Junior
E	financial analyst	Senior
F	financial analyst	Specialist
G	financial analyst	Assistant
. . .	. . .	. . .
X	share technician	Clerk
Y	support e-commerce	Junior
Z	office banking	Head of Division

(b) Permission Assignments

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	1, 2, 3, 4, 7
	derivatives trading	1, 2, 3, 7, 10, 12, 14
	interest instruments	1, 4, 8, 12, 14, 16
	private consumer instruments	1, 2, 4, 7
. . .	. . .	. . .

(c) Permission Assignment with Inheritance

Role	Application	Access Right
A	money market instruments	1, 2, 3, 4
	derivatives trading	1, 2, 3, 7, 10, 12
	interest instruments	1, 4, 8, 12, 14, 16
B	money market instruments	7
	derivatives trading	14
	private consumer instruments	1, 2, 4, 7
. . .	. . .	. . .

Based on the user's position and job function, the department also assigns one or more roles to the user. The user/role information is provided to the Authorization Administration, which creates a security profile for each user that associates the User ID and role with a set of access rights. When a user invokes an application, the application consults the security profile for that user to determine what subset of the application's access rights are in force for this user in this role.

A role may be used to access several applications. Thus, the set of access rights associated with a role may include access rights that are not associated with one of

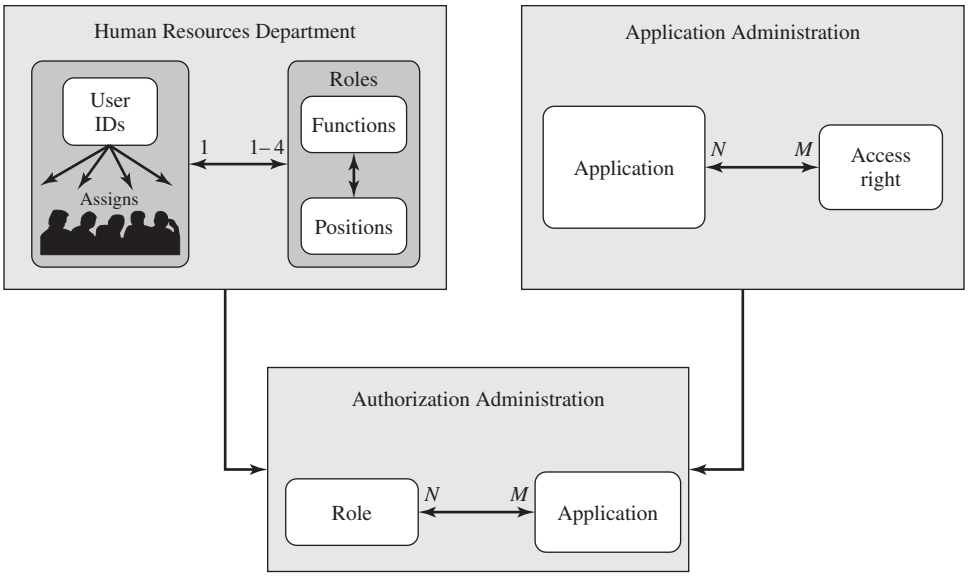


Figure 4.14 **Example of Access Control Administration**

the applications the user invokes. This is illustrated in Table 4.5b. Role A has numerous access rights, but only a subset of those rights are applicable to each of the three applications that role A may invoke.

Some figures about this system are of interest. Within the bank, there are 65 official positions, ranging from a Clerk in a branch, through the Branch Manager, to a Member of the Board. These positions are combined with 368 different job functions provided by the human resources database. Potentially, there are 23,920 different roles, but the number of roles in current use is about 1,300. This is in line with the experience other RBAC implementations. On average, 42,000 security profiles are distributed to applications each day by the Authorization Administration module.

4.10 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

access control	attribute-based access control (ABAC)	authorizations
access control list	Attribute Exchange Network (AXN)	assessor
access management	attribute provider	capability ticket
access matrix	auditor	cardinality
access right		closed access control policy
attribute		credential

credential management	kernel mode	prerequisite role
discretionary access control (DAC)	least privilege	privilege
dispute resolver	limited role hierarchy	protection domain
dynamic separation of duty (DSD)	mandatory access control (MAC)	relying part
entitlements	mutually exclusive roles	resource
environment attribute	object	rights
general role hierarchy	object attribute	role-based access control (RBAC)
group	open access control policy	role constraints
identity	Open Identity Exchange Corporation (OIX)	role hierarchies
identity, credential, and access management (ICAM)	Open Identity Trust Framework (OITF)	separation of duty
identity federation	OpenID	session
identity management	OpenID Foundation (OIDF)	static separation of duty (SSD)
identity provider	owner	subject
Information Card Foundation (ICF)	permission	subject attribute
	policy	trust framework
		trust framework provider
		user mode

Review Questions

4.1 What is the difference between authentication and authorization?
4.2 How does RBAC relate to DAC and MAC?
4.3 List and define the three classes of subject in an access control system.
4.4 List and briefly explain the three basic elements of access control.
4.5 What is ABAC?
4.6 What is the difference between an access control list and a capability ticket?
4.7 List some of the main types of access control.
4.8 Briefly define the four RBAC models of Figure 4.8a.
4.9 What is meant by mutually exclusive roles in the $RBAC_3$ model?
4.10 Describe three types of role hierarchy constraints.
4.11 In the NIST RBAC model, what is the difference between SSD and DSD?

Problems

4.1 For the DAC model discussed in Section 4.3, an alternative representation of the protection state is a directed graph. Each subject and each object in the protection state is represented by a node (a single node is used for an entity that is both subject and object). A directed line from a subject to an object indicates an access right, and the label on the link defines the access right.
- **a.** Draw a directed graph that corresponds to the access matrix of Figure 4.2a.
- **b.** Draw a directed graph that corresponds to the access matrix of Figure 4.3.
- **c.** Is there a one-to-one correspondence between the directed graph representation and the access matrix representation? Explain.

4.2 **a.** Explain, with an appropriate example, how protection domains provide flexibility.
b. How is the concept of protection domains related to operating systems? Explain by quoting an example from the UNIX operating system.

4.3 The VAX/VMS operating system makes use of four processor access modes to facilitate the protection and sharing of system resources among processes. The access mode determines:

- **Instruction execution privileges:** What instructions the processor may execute
- **Memory access privileges:** Which locations in virtual memory the current instruction may access

The four modes are as follows:

- **Kernel:** Executes the kernel of the VMS operating system, which includes memory management, interrupt handling, and I/O operations
- **Executive:** Executes many of the operating system service calls, including file and record (disk and tape) management routines
- **Supervisor:** Executes other operating system services, such as responses to user commands
- **User:** Executes user programs, plus utilities such as compilers, editors, linkers, and debuggers

A process executing in a less-privileged mode often needs to call a procedure that executes in a more-privileged mode; for example, a user program requires an operating system service. This call is achieved by using a change-mode (CHM) instruction, which causes an interrupt that transfers control to a routine at the new access mode. A return is made by executing the REI (return from exception or interrupt) instruction.

a. A number of operating systems have two modes: kernel and user. What are the advantages and disadvantages of providing four modes instead of two?
b. Can you make a case for even more than four modes?

4.4 The VMS scheme discussed in the preceding problem is often referred to as a ring protection structure, as illustrated in Figure 4.15. Indeed, the simple kernel/user scheme is a two-ring structure. A disadvantage of a ring-structured access control system is that it violates the principle of least privilege. For example if we wish to have an object accessible in ring X but not ring Y, this requires that $X < Y$. Under this arrangement all objects accessible in ring X are also accessible in ring Y.

a. Explain in more detail what the problem is and why least privilege is violated.
b. Suggest a way that a ring-structured operating system can deal with this problem.

4.5 UNIX treats file directories in the same fashion as files; that is, both are defined by the same type of data structure, called an inode. As with files, directories include a nine-bit protection string. If care is not taken, this can create access control problems. For example, consider a file with protection mode 644 (octal) contained in a directory with protection mode 730. How might the file be compromised in this case?

4.6 In the traditional UNIX file access model, which we describe in Section 4.4, UNIX systems provide a default setting for newly created files and directories, which the owner may later change. The default is typically full access for the owner combined with one of the following: no access for group and other, read/execute access for group and none for other, or read/execute access for both group and other. Briefly discuss the advantages and disadvantages of each of these cases, including an example of a type of organization where each would be appropriate.

4.7 Consider user accounts on a system with a Web server configured to provide access to user Web areas. In general, this uses a standard directory name, such as 'public_html,' in a user's home directory. This acts as their user Web area if it exists. However, to allow the Web server to access the pages in this directory, it must have at least search (execute) access to the user's home directory, read/execute access to the Web directory, and read access to any webpages in it. Consider the interaction of this requirement

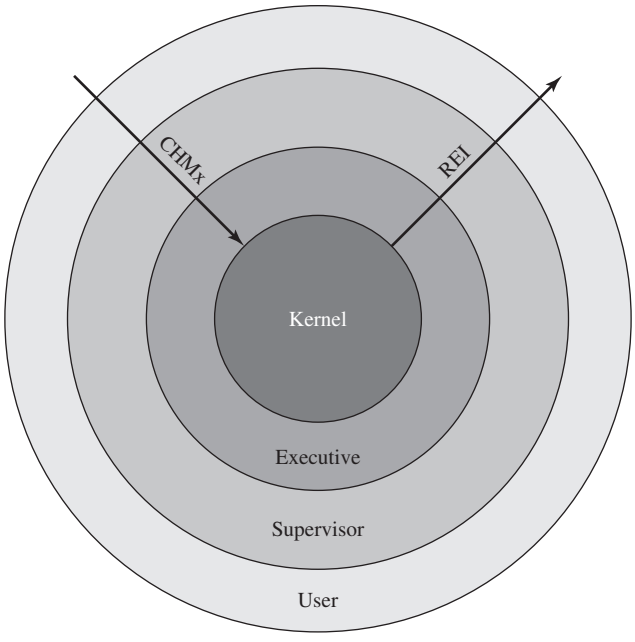


Figure 4.15 VAX/VMS Access Modes

with the cases you discussed for the preceding problem. What consequences does this requirement have? Note a Web server typically executes as a special user, and in a group that is not shared with most users on the system. Are there some circumstances when running such a Web service is simply not appropriate? Explain.

4.8 Assume an application requires access control policies based on the applicant's age and the type of funding to be provided. Using an ABAC approach, write policy rules for each of the following scenarios:

a. If the applicant's age is more than 35, only "Research Grants (RG)" can be provided.

b. If the applicant's age is less than or equal to 35, both "RG and Travel Grants (TG)" can be provided.

4.9 Assume a system with K subject attributes, M object attributes and Range () denotes the range of possible values that each attribute can take. What are the number of roles and permissions required for an RBAC model? What is the problem with this approach if additional attributes are added?

4.10 For the NIST RBAC standard, we can define the general role hierarchy as follows:

RH $\subseteq$ ROLES $\times$ ROLES is a partial order on ROLES called the inheritance relation, written as $\geq$, where $r_1 \geq r_2$ only if all permissions of r_2 are also permissions of r_1, and all users of r_1 are also users of r_2. Define the set *authorized_permissions*(r_i) to be the set of all permissions associated with role r_i. Define the set *authorized_users*(r_i) to be the set of all users assigned to role r_i. Finally, node r_1 is represented as an immediate descendant of r_2 by $r_1 \gg r_2$, if $r_1 \geq r_2$, but no role in the role hierarchy lies between r_1 and r_2.

a. Using the preceding definitions, as needed, provide a formal definition of the general role hierarchy.

b. Provide a formal definition of a limited role hierarchy.

4.11 In the example of Section 4.9, use the notation *Role(x)*. *Position and Role(x)*. *Function* to denote the position and the function associated with role *x*.

a. We can define the role hierarchy for this example as one in which one role is superior to another if its position and functions are both superior. Express this relationship formally.

b. An alternative role hierarchy is one in which a role is equal to another if its position is equal, regardless of the function. Express this relationship formally.

4.12 In the example of the online entertainment store in Section 4.6, with the finer-grained policy that includes premium and regular users, describe the ABAC policy rules for accessing a movie, and list all the advantages of an ABAC control policy.