Unit 3: Access Control

Introduction:

Access control is a data security process that enables organizations to manage who is authorized to access corporate data and resources. Secure access control uses policies that verify users are who they claim to be and ensure appropriate control access levels are granted to users.

Basically access control is of 2 types: Physical Access Control: Physical access control restricts entry to campuses, buildings, rooms and physical IT assets. Logical Access Control: Logical access control limits connections to computer networks, system files and data.

Access controls limit access to information and information processing systems. When implemented effectively, they mitigate the risk of information being accessed without the appropriate authorization, unlawfully and the risk of a data breach.

Access control protects data by ensuring that only authorized entities can retrieve data from an organization's data repositories. When effectively implemented, access controls prevent unauthorized and compromised users from accessing sensitive data.

Two definitions of access control are useful in understanding its scope

Access control can be defines as the process of granting or denying specific requests to obtain and use information and related information processing services; and enter specific physical facilities.

Access control defines the 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.

Access controls are the 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.

Access Control Principles:

Access control defines computer security as follows: Measures that implement and assure security services in a computer system, particularly those that assure access control service. 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:

Access control context involves the following entities and functions:

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

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.

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.

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): DAC provides access rights depending upon the rules already set by the administrators. In this type of access control model, each resource has an owner or admin that decides to whom to give access and at what level.

DAC decentralizes security decisions, allowing administrators and resource owners to give access to users at specified levels. It uses ACLs (access control lists), which define at what level to give users permission to a particular resource.

Mandatory access control (MAC): Mandatory access control is access control policies that are decided by the system and not the application or data owner. Mandatory Access Control (MAC) is a group of security policies constrained according to system classification, configuration and authentication. MAC policy management and settings are created in one secure network and defined to system administrators.

MAC defines and provides a centralized enforcement of confidential security policy parameters. Mandatory access control creates strict security policies for single users and the resources, systems, or data they are enabled to access. These policies are controlled by a management; single users are not given the authority to set, alter, or revoke permissions in a method that contradicts current policies.

Role-based access control (RBAC): Role-Based Access Control (RBAC) is a method for restricting network access based on the roles of individual users. RBAC allows employees to access only the information they need to do their job. Employee roles in an organization determine the privileges granted to individuals and prevent lower-level employees from accessing sensitive information or performing higher-level tasks.

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, MAC is a concept that evolved out of requirements for military information security and is best covered in the context of trusted system. Both RBAC and ABAC have become increasingly popular;

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.

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 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.

Discretionary Access Control:

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, it is say that of an **access matrix**. The access matrix concept was formulated by Lampson and subsequently refined by Graham, Denning and by Harrison.

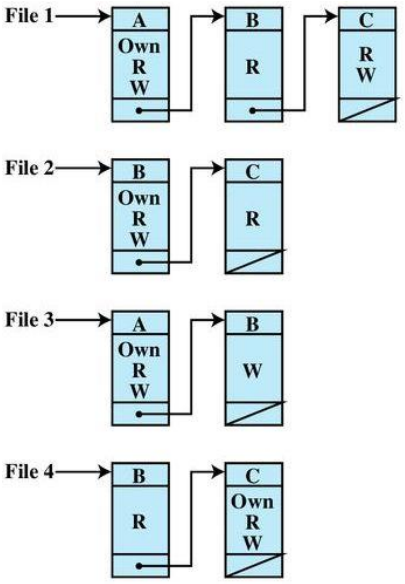
One dimension of the matrix consists of identified subjects that may attempt data access to the resources. Typically, this list will consist of individual users or user groups, although access could be controlled for terminals, network equipment, hosts, or applications instead of or in addition to users. 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.

AS shown in below figure, it 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.

		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

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).

AS shown in following fig. 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.



Access control lists for files of part (a)

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.

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.

Proposes a data structure that is not sparse, like the access matrix, but is more convenient than either ACLs or capability lists. 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.

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. 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.

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.

Example: UNIX File Access Control:

For 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 based on, the file access. 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. The execute bit grants the right to descend into the directory or search it for a filename.

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.

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.

This access scheme is adequate when file access requirements align with users and a modest number of groups of users. 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.

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 as shown fig.

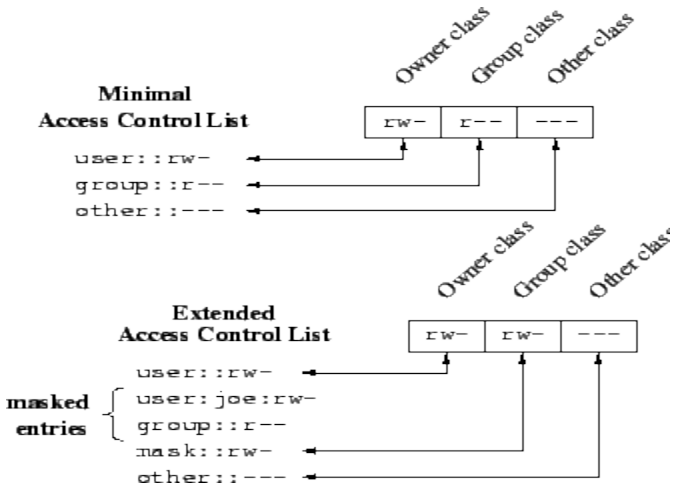


Figure 4.5 UNIX File Access Control

The owner class and other class entries in the 9-bit permission field have the same meaning as in the minimal ACL case.

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.

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.

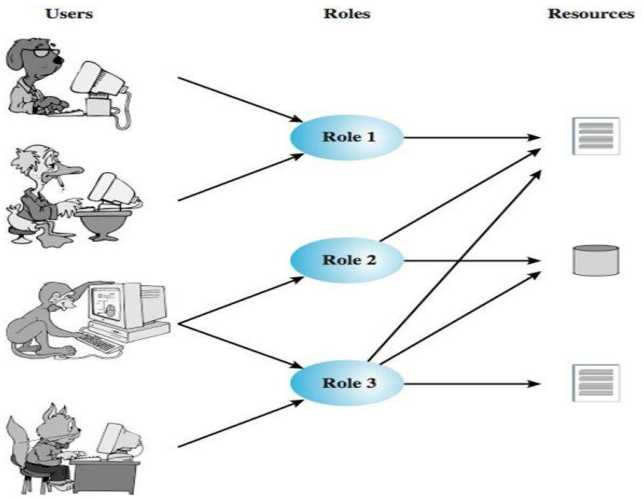
Step 2 checks if the matching entry contains sufficient permissions. A process can be a member in more than one group; so more than one group entry can match.

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.

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.

The relationship of users to roles is many to many, as is the relationship of roles to resources, or system objects as shown in following fig. The set of user's changes, in some 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. 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 that a single user may be assigned multiple roles (more than one mark in a row) and that multiple users may be assigned to a single role (more than one mark in a 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 that a role can be treated as an object, allowing the definition of role hierarchies.

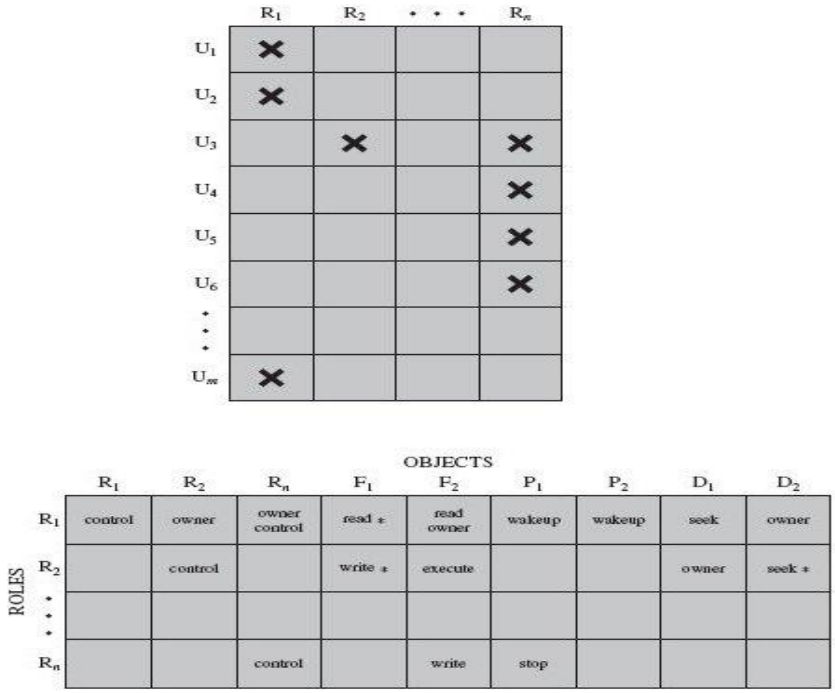


Fig :-Access Control Matrix Representation of RBAC

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 4.3. RBAC0 contains the minimum functionality for an RBAC system. RBAC1 includes the RBAC0 functionality and adds role hierarchies, which enable one role to inherit permissions from another role. RBAC2 includes RBAC0 and adds constraints, which restrict the ways in which the components of a RBAC system may be configured. RBAC3 contains the functionality of RBAC0, RBAC1, and RBAC2.

Base Model—RBAC0 Figure 4.8b, without the role hierarchy and constraints, contains the four types of entities in an RBAC0 system:

• **User:** An individual that has access to this computer system. Each individual has an associated user ID.

• **Role:** A named job functions 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.

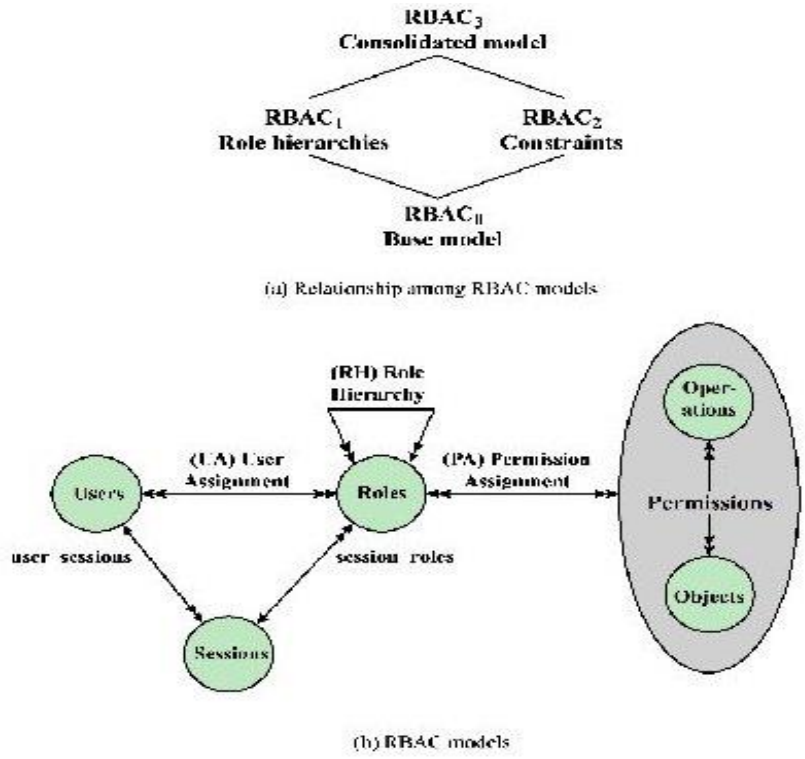


Figure 4.8 A Family of Role-Based Access Control Models.

RBAC0 is the minimum requirement for an RBAC system. RBAC1 adds role hierarchies and RBAC2 adds constraints. RBAC3 includes RBAC1 and RBAC2. 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.

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.

Role Hierarchies—RBAC1 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.

Constraints—RBAC2 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.

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 system wide 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 4.4a 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 4.4b. 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.

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

(a) Functions and Official Positions

(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) PA 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
•••	•••	•••

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. 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 the applications the user invokes. This is illustrated in Table 4.4b. 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.