Suppose a member $w$ of an organization $W$ wants to provide access to a document $d$ to members of organization $Y$, but the document is not to be shared with members of organization $X$ or $Z$. So, $d$ cannot be in category $C$ because if it were, members $x \in X$ and $z \in Z$ could access $d$. Another category containing $d$, $W$, and $Y$ must be created. Multiplying this by several thousand possible relationships and documents creates an unacceptably large number of categories.

A second problem with mandatory access controls arises from the abstraction. Organizations that use categories grant access to individuals on a "need to know" basis. There is a formal, written policy determining who needs the access based on common characteristics and restrictions. These restrictions are applied at a very high level (national, corporate, organizational, and so forth). This requires a central clearinghouse for categories. The creation of categories to enforce ORCON implies local control of categories rather than central control, and a set of rules dictating who has access to each compartment.

ORCON abstracts none of this. ORCON is a decentralized system of access control in which each originator determines who needs access to the data. No centralized set of rules controls access to data; access is at the complete discretion of the originator. Hence, the MAC representation of ORCON is not suitable.

A solution is to combine features of the MAC and DAC models. The rules are

1. The owner of an object cannot change the access controls of the object.
2. When an object is copied, the access control restrictions of that source are copied and bound to the target of the copy.
3. The creator (originator) can alter the access control restrictions on a per-subject and per-object basis.

The first two rules are from mandatory access controls. They say that the system controls all accesses, and no one may alter the rules governing access to those objects. The third rule is discretionary and gives the originator power to determine who can access the object. Hence, this hybrid scheme is neither MAC nor DAC.

The critical observation here is that the access controls associated with the object are under the control of the *originator* and not the owner of the object. Possession equates to only some control. The owner of the object may determine to whom he or she gives access, but only if the originator allows the access. The owner may not override the originator.

## 7.4 Role-Based Access Control

The ability, or need, to access information may depend on one's job functions.

EXAMPLE: Allison is the bookkeeper for the Department of Mathematics. She is responsible for balancing the books and keeping track of all accounting for that

The forms of these axioms restrict the transactions that can be performed. They do not ensure that the allowed transactions can be executed. This suggests that role-based access control (RBAC) is a form of mandatory access control. The axioms state rules that must be satisfied before a transaction can be executed. Discretionary access control mechanisms may further restrict transactions.

EXAMPLE: Some roles subsume others. For example, a trainer can perform all actions of a trainee, as well as others. One can view this as containment. This suggests a hierarchy of roles, in this case the trainer role containing the trainee role. As another example, many operations are common to a large number of roles. Instead of specifying the operation once for each role, one specifies it for a role containing all other roles. Granting access to a role $R$ implies that access is granted for all roles containing $R$. This simplifies the use of the RBAC model (and of its implementation).

If role $r_i$ contains role $r_j$, we write $r_i > r_j$. Using our notation, the implications of containment of roles may be expressed as

$$(\forall s \in S)[\ r_i \in authr(s) \wedge r_i > r_j \rightarrow r_j \in authr(s)\ ]$$

EXAMPLE: RBAC can model the separation of duty rule. Our goal is to specify separation of duty centrally; then it can be imposed on roles through containment, as discussed in the preceding example. The key is to recognize that the users in some roles cannot enter other roles. That is, for two roles $r_1$ and $r_2$ bound by separation of duty (so the same individual cannot assume both roles):

$$(\forall s \in S)\ [\ r_1 \in authr(s) \rightarrow r_2 \notin authr(s)\ ]$$

Capturing the notion of mutual exclusion requires a new predicate.

**Definition 7–11.** Let $r$ be a role, and let $s$ be a subject such that $r \in authr(s)$. Then the predicate $meauth(r)$ (for *mutually exclusive authorizations*) is the set of roles that $s$ cannot assume because of the separation of duty requirement.

Putting this definition together with the above example, the principle of separation of duty can be summarized as

$$(\forall r_1, r_2 \in R)\ [\ r_2 \in meauth(r_1) \rightarrow [\ (\forall s \in S)\ [\ r_1 \in authr(s) \rightarrow r_2 \notin authr(s)\ ]\ ]\ ]$$

## 7.5 Summary

The goal of this chapter was to show that policies typically combine features of both integrity and confidentiality policies. The Chinese Wall model accurately captures requirements of a particular business (brokering) under particular conditions (the

department. She has access to all departmental accounts. She moves to the university's Office of Admissions to become the head accountant (with a substantial raise). Because she is no longer the bookkeeper for the Department of Mathematics, she no longer has access to those accounts. When that department hires Sally as its new bookkeeper, she will acquire full access to all those accounts. Access to the accounts is a function of the job of bookkeeper, and is not tied to any particular individual.

This suggests associating access with the particular job of the user.

**Definition 7–7.** A *role* is a collection of job functions. Each role $r$ is authorized to perform one or more transactions (actions in support of a job function). The set of authorized transactions for $r$ is written $trans(r)$.

**Definition 7–8.** The *active role of a subject* $s$, written $actr(s)$, is the role that $s$ is currently performing.

**Definition 7–9.** The *authorized roles of a subject* $s$, written $authr(s)$, is the set of roles that $s$ is authorized to assume.

**Definition 7–10.** The predicate $canexec(s, t)$ is true if and only if the subject $s$ can execute the transaction $t$ at the current time.

Three rules reflect the ability of a subject to execute a transaction.

**Axiom 7–1.** Let $S$ be the set of subjects and $T$ the set of transactions. The *rule of role assignment* is $(\forall s \in S)(\forall t \in T)[\ canexec(s, t) \rightarrow actr(s) \neq \varnothing\ ]$.

This axiom simply says that if a subject can execute *any* transaction, then that subject has an active role. This binds the notion of execution of a transaction to the role rather than to the user.

**Axiom 7–2.** Let $S$ be the set of subjects. Then the *rule of role authorization* is $(\forall s \in S)[\ actr(s) \subseteq authr(s)\ ]$.

This rule means that the subject must be authorized to assume its active role. It cannot assume an unauthorized role. Without this axiom, any subject could assume any role, and hence execute any transaction.

**Axiom 7–3.** Let $S$ be the set of subjects and $T$ the set of transactions. The *rule of transaction authorization* is $(\forall s \in S)(\forall t \in T)[\ canexec(s, t) \rightarrow t \in trans(actr(s))\ ]$.

This rule says that a subject cannot execute a transaction for which its current role is not authorized.