(11) NetWare 4 as an Example of Role-Based Access Control

Jeremy Epstein, Cordant, Inc., and Ravi Sandhu, George Mason University

jepstein@cordant.com, 11400 Commerce Park Drive, Reston VA 22091

sandhu@isse.gmu.edu, Department of Information and Software Systems Engineering, Fairfax, VA 22030

1.0 Introduction

In [SAND96a], the second author describes a taxonomy of role-based access control (RBAC) models, divided into four classes shown in Figure 11-1(a), *Taxonomy of RBAC Models*. A complementary set of models is used for administrative role-based access control (ARBAC), as shown in Figure 11-1(b), *Taxonomy of ARBAC Models*. In this position paper, we describe how the RBAC and ARBAC models can be partially implemented using unmodified NetWare 4 servers.

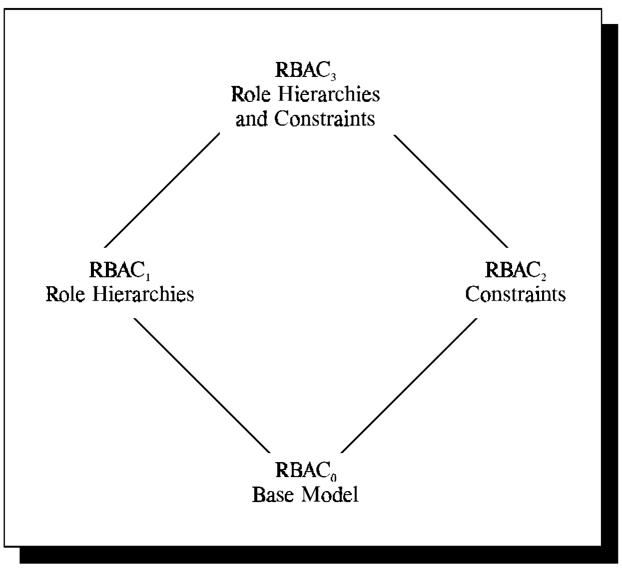

Figure 11-1(a). Taxonomy of RBAC Models

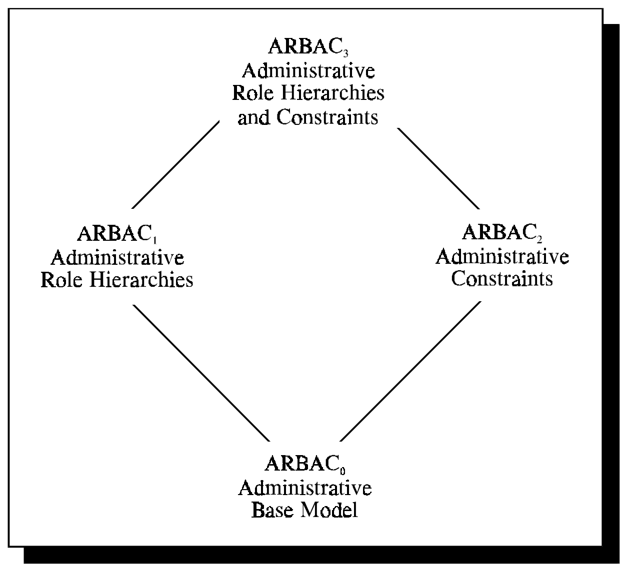

Figure 11-1(b). Taxomony of ARBAC Models

2.0 NetWare Access Control Policies

NetWare includes two major types of objects with separate but interrelated security policies. *NetWare Directory Services (NDS) objects* represent abstractions such as users, roles, groups, and computers, while *file system objects* provide a traditional hierarchical file system. Both file and NDS objects are supported by sophisticated access control mechanisms that allow assignment of rights to users, groups, and other entities. In addition, they allow rights assignments to be inherited in a hierarchical fashion. Most importantly, they allow assignment of access rights in a granular fashion: rights to a file or directory can be controlled independently from the ability to change file or directory access rights. That is, a user can, through a role, be granted the ability to read, write, create, or delete files without having any ability to share those files with others.

2.1 NDS Object Access Control Policy

NDS is an X.500-like system for managing data that represents an organization's assets. Every object in NDS has a class, which is defined in the *schema*. The schema contains approximately 20 built-in classes (e.g., User, Organization), and can be extended by authorized users. Depending on the class of an object, it will have one or more *attributes*, also known as *properties*. Attributes are used to store information about some aspect of an object. For example, an object of class User has attributes to represent the person's name, home directory, login script, etc. Some attributes are security relevant (e.g., those relating to passwords) while others are not (e.g., the user's telephone number).

NDS objects are organized in a tree, much as many file systems organize files and directories into a tree[1]. NDS objects are either *container* objects, which correspond to directories in a file system, or *leaf* objects, which correspond to files in a file system. Whether an object is a container or leaf object is determined by its class, as defined in the NDS schema. Figure 11-2, *Sample NDS Structure*, shows an NDS structure that might represent an organization. Objects are named by their complete path to the root, starting at the leaf. For example, Sally.Finance.Acme is the complete name of the left-most node in the tree.

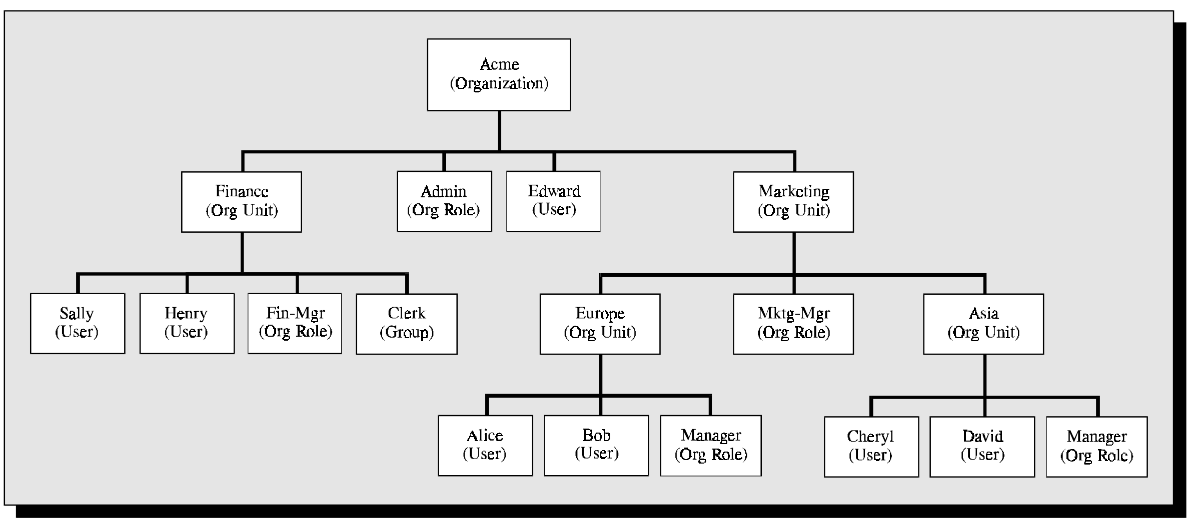

Figure 11-2. Sample NDS Structure

Users log in to NetWare servers by providing the name of their NDS User object. When presented with the proper password (and meeting other restrictions, such as an unexpired account), the user is logged in, and his or

[1] The tree structure does not include anything akin to a "hard" link in a UNIX file system. However, objects of class Asias can be used to provide a symbolic link-like capability.

her *security equivalence*[2] list, which is used in calculating access rights, is computed as follows:

- All users are security equivalent to the pseudo-object named [Public].
- Users are security equivalent to all container objects in the path from their User object to the root of the tree, designated [Root], including both themselves and [Root].
- Users are security equivalent to those objects to which they are explicitly made security equivalent by the administrator.

All rights in NDS are passed through security equivalence. Group membership is just one example of security equivalence, but does not play any special role (contrary to what is stated in [SCHA94]).

Note that users cannot choose a subset of security equivalences to be used for a session: they gain all security equivalences. Thus, a given user may be represented by several NDS User objects to reflect different uses of the system (e.g., as an administrator or as an ordinary user). The user would select which NDS User object to use, depending on the task to be accomplished.

While NDS can be configured in many different ways, typical configurations allow administrative users (but not ordinary users) to create and delete NDS objects in one or more containers. Administrative users can usually modify the attributes of NDS objects that they are responsible for. Non-administrative users may be able to modify some of the attributes of their User objects, such as the Login Script or Telephone Number attributes.

2.1.1 NDS Object Access Control Algorithm

Every NDS object has an Access Control List (ACL), which is stored as an attribute of the NDS object. An ACL is a list of triples, where the elements of the triple are as follows:

- The subject name, which is either the name of an NDS object (e.g., a specific User, Group, or Organization), or a pseudo-ID such as [Root], [Public], or [Inherited Rights Filter] (abbreviated IRF).
- Protected attribute, which for NDS object rights calculation is the reserved symbol [Entry Rights].
- Access rights, which is zero or more of *Supervisor, Create, Delete, Rename,* and *Browse*. The *Supervisor* right implies all other rights.

An object O1's rights to object O2 is computed as follows:

- For each object O' to which O1 is security equivalent, compute the rights for O' to O2 by searching from the root of the tree toward O2.

---

[2] "Security equivalence" is a misnomer. Security equivalences are neither transitive nor reflexive.

R, which is initially empty, represents the object rights O' has to O2. For each node N along the path, perform the following steps:

- If N's ACL includes an entry with the subject name [Inherited Rights Filter] and a protected attribute of [Entry Rights], then remove all access rights from R except those listed in the IRF.
- If N's ACL contains an entry with the subject name of O' and a protected attribute of [Entry Rights], then set R to the access rights for the entry.

- Union the value of R calculated in the first step for each O' together, yielding the rights of O1 to O2.

This algorithm allows setting an ACL at one location in the NDS tree and allowing the rights to flow down using *inheritance*. For example, in Figure 11-1, if Edward.Acme has the *Supervisor* right to the Acme object, then he will have the *Supervisor* right (and therefore all rights) to all objects in the NDS tree (unless they are blocked with an IRF). If Sally.Finance.Acme is made security equivalent to Edward.Acme, then she will also have the *Supervisor* right to all objects. An alternate method is to assign Admin.Acme the *Supervisor* right to Acme, and make Edward.Acme security equivalent to Admin.Acme. In this case, Sally.Finance.Acme would not gain the *Supervisor* right to Acme unless she is security equivalent to Admin.Acme (i.e., she does not obtain the rights transitively through Edward.Acme).

Note that groups are simply a particular case of security equivalence in this scheme: assigning rights to an NDS Group object and making NDS User objects security equivalent to the Group object is no different than assigning rights to any other class of NDS object and establishing security equivalence of NDS user object to object of that other class.

2.1.2 NDS Attribute Access Control Algorithm

The ACL for an NDS object is also used for NDS attributes, with the following changes:

- The protected attribute can be either the name of a specific attribute (e.g., Home Directory) or the pseudo-attribute [All Properties Rights].
- The access rights are zero or more of *Supervisor*, *Compare, Add or Delete Self*[3], *Read*, and *Write*. Note that the *Supervisor* attribute right is different from the Supervisor object right.

An object O1's rights to attribute A of object O2 is computed as follows:

1. For each object O' to which O1 is security equivalent, compute the rights for O' to all attributes of O2 by searching from the root of the tree toward O2. R, which is initially empty, represents the attribute rights O' has to O2. For each node N along the path, perform the following steps:

[3] *Add or Delete Self* is one right, not two (i.e., the word “or” does not indicate there are two rights).

a. If N's ACL includes an entry with the subject name [Inherited Rights Filter] and a protected attribute of [All Properties Rights], then remove all access rights from R except those listed in the IRF.

b. If N's ACL contains an entry with the subject name of O' and a protected attribute of [All Properties Rights], then set R to the access rights for the entry.

2. For node O2 *only*, perform the following steps:

   a. If O2's ACL includes an entry with the subject name [Inherited Rights Filter] and a protected attribute of A, then remove all access rights from R except those listed in the IRF.

   b. If O2's ACL contains an entry with the subject name of O' and a protected attribute of A, then set R to the access rights for the entry.

3. Union the value of R calculated in steps 1 and 2 for each O' together, yielding the rights of O1 to attribute A of O2.

4. For each object O' to which O1 is security equivalent, determine whether O' has the *Supervisor* object right to O2 by searching from the root of the tree toward O2. S, which is initially false, represents whether O' has the *Supervisor* object right to O2. For each node N along the path, perform the following steps:

   a. If N's ACL includes an entry with the subject name [Inherited Rights Filter], a protected attribute of [Entry Rights], and the IRF does not include the *Supervisor* right, then clear S.

   b. If N's ACL contains an entry with the subject name of O', a protected attribute of [Entry Rights], and the access rights include the *Supervisor* right, then set S.

5. If the value of S for any of the values of O' computed in step 4 is set, then O1 has all rights to attribute A of O2, regardless of the results of step 3.

There are certain attributes, which are flagged in special ways, that are not modified by the ACL. For example, attributes may be marked as Read-Only, which precludes modification to the attribute, even if the user has adequate rights. Other attributes are marked as Public-Read, which is equivalent to the ACL entry <[Public], *A*, Read> (where *A* is the name of the attribute). Attributes are marked as Read-Only or Public-Read as part of the attribute definition, and not as part of the ACL for the object.

An important aspect of the above policy is that rights to *individual* attributes are not inherited, but rights to all attributes (as represented by the symbol [All Properties Rights]) are inherited. Thus, a user could be given the Read and Write rights to [All Properties] at the root of the NDS tree, which would provide access to all lower objects (unless modified by IRFs or subsequent trustee assignments), but giving the *Read* and *Write* rights to the Telephone Number attribute at the root would only affect access to the attribute of that particular object.

2.2 File System Object Access Control Policy

Files in a NetWare file system are organized in a hierarchical tree, much as any traditional file system. Files are organized into *volumes*, which typically represent disk drives. File system rights rely on many of the same concepts as NDS rights: security equivalence, inheritance, and inherited rights filters. The file system access control policy is similar, but not identical, to the NDS object and NDS attribute policy. Every file system object (file or directory) may have a *trustee* list, which is equivalent to an ACL. Elements of a trustee list are pairs[4], where the first element is the subject name and the second element is the access rights (zero or more of *Supervisor, Read, Write, Create, Erase, Modify, File Scan,* or *Access Control*[5]). Any NDS object with at least one right to a file system object is called a *trustee* of the object, indicating that it has (partial) responsibility for the data contained in the file or directory. An object O1's rights to a file or directory F is computed as follows:

- For each object O' to which O1 is security equivalent, compute the rights for O' to F by searching from the root of the volume toward F. R, which is initially empty, represents the object rights O' has to F. For each node N along the path, perform the following steps:

  - If N's trustee list includes an entry with the subject name [Inherited Rights Filter], then remove all access rights from R except those listed in the IRF[6].

  - If N's ACL contains an entry with the subject name of O', then set R to the access rights for the entry, unless R already contains the *Supervisor* right, in which case R is unchanged.

- Union the value of R calculated in the first step for each O' together, yielding the rights of O1 to F.

Just as inheritance is used to assign rights in a relatively small number of locations in NDS, so too can it be used in the file system. For example, assigning the single trustee entry <[Public], {Read, File Scan}> to the \PUBLIC directory will allow all users access to all files in that directory (and all subdirectories) without assigning any rights to individual files in the directory.

Note that because of inheritance, rights are typically not assigned at the root of a volume, because that would provide rights to the whole volume (unless blocked by an IRF).

3.0 Using NetWare for RBAC

4 There is no "protected attribute" field in a trustee list entry, whereas there is in an NDS ACL entry.
5 The *Access Control* right allows changing the trustee list, except to add an entry with the Supervisor right. Note that there is no Executive right, because users execute programs on workstations over which the server has no control. Similarly, there is no "setuid" concept as in UNIX for protected subsystems.
6 The IRF for a file cannot block inheritance of the Supervisor right.

NetWare 4 can be used to enforce portions of the RBAC0, RBAC1, ARBAC0, and ARBAC1 policies described in [SAND96a]. The objects to be protected for RBAC0 and RBAC1 are files and directories, while the objects to be protected for ARBAC0 and ARBAC1 are NDS objects. We do not believe that NetWare can be used for implementation of role constraints (RBAC2 and ARBAC2) and, therefore, it cannot be used for the consolidated model (RBAC3 and ARBAC3), which presumes the presence of role constraints.

3.1 RBAC0: Base Model

RBAC0 provides basic RBAC features. The objects we wish to protect using RBAC0 are files and directories in the file system. The users of RBAC0 are equivalent to users in NetWare, and the permissions are the NetWare file rights (Supervisor, Read, Write, Create, Erase, Modify, File Scan, and Access Control). Roles can be implemented using any NDS object, although the Organizational Role object is most suitable for the purpose because of its name.

3.1.1 What Can be Done

RBAC0 calls for a many-to-many relationship between roles and users and between roles and permissions. In NetWare, users may be security equivalent to an arbitrary number of other objects, and objects may have an arbitrary number of users that are security equivalent to them. This allows us to establish a many-to-many relationship between users and roles. Similarly, the same permission (right) can be assigned to any number of roles and vice versa.

The essence of RBAC0 in NetWare is the ability to assign access rights independently from access control rights. That is, a role could have the ability to create, delete, read, and write files in a directory without the ability to grant others access to that directory. This would be done by not assigning the Access Control right to the role. In turn, user's rights are limited by the roles to which they are security equivalent.

3.1.2 What Cannot be Done

As noted above, NetWare has no concept of sessions operating in different roles as called for in RBAC0. Users obtain those rights associated with all objects to which they are security equivalent. Thus, there is no capability for dynamic activation and deactivation of roles during a session; a user must log out from one NetWare User account and log in as a different one to change their role. This is a weakness of NetWare, as it forces users to either have their maximum rights available at all times or to maintain multiple accounts, each of which is used for a different purpose (e.g., user or administrator).

A client operating system could maintain a mapping of user identities to roles and transparently log the user in and out as necessary. For example, the user might present a name and a role, and the client would map that to an NDS User object. Similarly, given sufficient client operating system support, users could have multiple windows each of which is logged in to a

NetWare server as a different user ID, thus presenting the facade of having multiple concurrent sessions. We are unaware of any implementation of this mechanism. In addition, maintaining multiple synchronized identities would be administratively cumbersome.

3.1.3 Possible Extensions

NetWare has no concept of a granularity below files. For example, it might be desirable to have RBAC to records in a database. This can be accomplished by extending the NetWare server using NetWare Loadable Modules (NLMs), which extend the server operating system[7]. Additional messages could be defined between clients and servers to provide access to database records. Such messages could rely on the authentication services provided by NetWare and could "piggy-back" by using the existing access controls to enforce RBAC on a row or column basis.

3.2 RBAC1: Role Hierarchies

The purpose of role hierarchies is to allow structuring of rights as they are typically done in an organization to reflect authority and responsibility. NetWare's rights inheritance coupled with NDS hierarchy works well for such a concept. Container objects, which are used for grouping NDS objects, can be trustees of a file just as any other NDS object. Because users are security equivalent to all containers they are transitively contained in, assigning rights to a container assigns those rights to all users (and other NDS objects) in that container.

However, NDS containers are inverted with respect to the usual organizational model that individuals near the top (i.e., root) have more authority and responsibility and authority than individuals closer to the bottom (i.e., the leaves).

A second difficulty with mapping NetWare access controls to RBAC1 is the notion of transitivity. [SAND96a] suggests that access controls should be transitive, so a Vice President would obtain not only those rights assigned to the Department Head role, but also transitively the rights associated to the Engineer role. However, security equivalence is not transitive, so this concept must be implemented administratively (e.g., either by assigning the Vice President role all of the rights of Department Head and Engineer roles, or by making each instance of a user who is a Vice President security equivalent to all three roles). [SAND96a] also describes the notion of inheriting rights from multiple roles. This is done easily in NetWare by making a User object security equivalent to an arbitrary number of other NDS objects.

NetWare does not meet the proposed requirement of role hierarchies being partial orders. Partial orders are reflexive, transitive, and anti-symmetric. NetWare's security equivalence mechanism provides reflexivity and anti-symmetry, but not transitivity.

---

[7] Commercial database systems (e.g., ORACLE) that run on NetWare use this technology.

As with RBAC0, RBAC1 includes the concept of sessions that can be used for a role. RBAC1 extends the concept further by requiring that users be able to assume any subset of the roles to which they are authorized, given the hierarchical nature of roles. This is impossible in NetWare, short of creating a separate user account for each unique combination of roles that a user might wish to exercise.

3.3 ARBAC0: Administrative Base Model

The notion of ARBAC0 is identical to that of RBAC0, except that it is concerned with administrative controls rather than access to files and directories. Just as NetWare's file access control policy can be used to provide roles with access to files and directories, so can the NDS access control policy be used to provide roles with access to NDS objects and their attributes. For example, by providing a role with the Supervisor object right to a container, individuals security equivalent to that role can administer objects within the container, subject to access blocked by IRFs. The role-based administrative access can be divided at an arbitrarily fine-grained level. For example, a Telephone-Manager role could be defined that has the Read and Write rights to the Telephone Number attribute of all NDS objects. However, to do this, the role would have to be listed on the ACL for every object in the NDS tree, because attribute-specific rights are not inherited.

3.4 ARBAC1: Administrative Role Hierarchies

The relationship of ARBAC1 to RBAC1 is the same as ARBAC0 to RBAC0. Just as hierarchies of users can be established to provide access to file system objects, so too can hierarchies be used for access to NDS objects. As with RBAC1, though, the lack of transitivity in the security equivalence mechanism limits the ability to meet the criteria established in [SAND96a].

4.0 Examples

In this section we provide several examples of how the NetWare mechanisms can be used to implement an RBAC policy.

4.1 File System Examples

Consider the NDS structure as shown previously in Figure 11-2 and the file system structure as shown in Figure 11-3, *Sample File System Structure*.

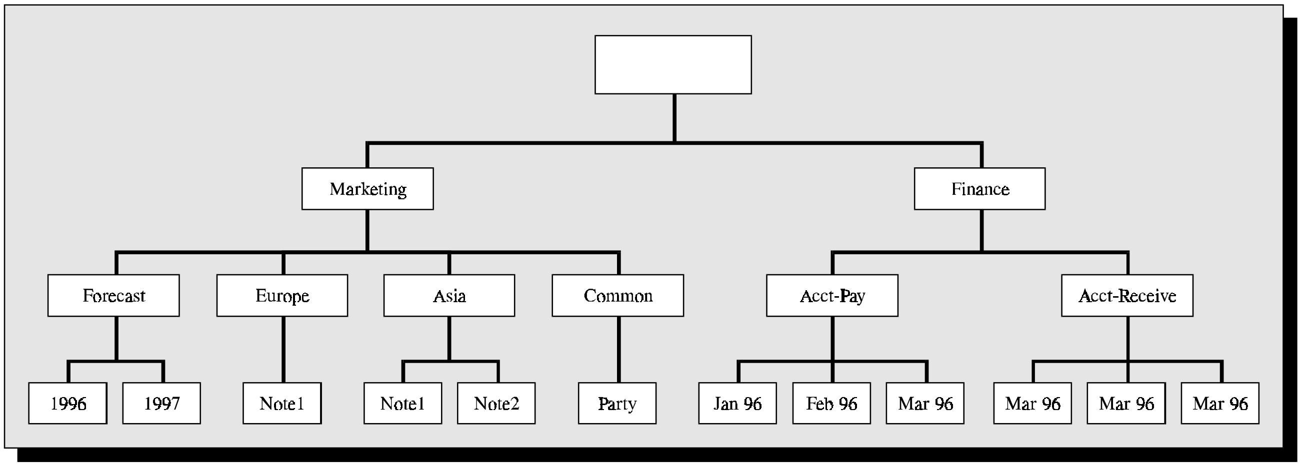

Figure 11-3. Sample File System Structure

Table 11-1, *Sample File System Trustee Assignments*, shows sample trustee assignments for this configuration. Recall that all users are automatically security equivalent to each container in which their user object is located and that users obtain the union of rights available to each object to which they are security equivalent. Thus, with no additional assignments, users Alice and Bob will have *File Scan, Create, Read*, and *Write* rights to all files and directories in \MKTG\EUROPE (by virtue of being security equivalent to Europe.Marketing.Acme, which is a trustee of the directory). Similarly, users Cheryl and David will have the *File Scan, Create, Read*, and *Write* rights to all files and directories in \MKTG\ASIA (by virtue of being security equivalent to Asia.Marketing.Acme, which is a trustee of the directory). Alice, Bob, Cheryl, and David will all have the *File Scan, Create, Read*, and *Write* rights to all files in \MKTG\COMMON (by virtue of being security equivalent to Marketing.Acme). Note that none of these assignments allow the users to propagate permissions, because no one has the *Access Control* or *Supervisor* right. Without any explicit security equivalences, no one has rights to \MKTG\FORECAST.

Table 11-1. Sample File System Trustee Assignments

File/Directory Name	Trustee	Rights
\MKTG	Mktg-Mgr.Marketing.Acme	Supervisor
\MKTG\EUROPE	Europe.Marketing.Acme	File Scan, Create, Read, Write
	Mgr.Europe.Marketing.Acme	Access Control
\MKTG\ASIA	Asia.Marketing.Acme	File Scan, Create, Read, Write
	Mgr.Asia.Marketing.Acme	Access Control
\MKTG\COMMON	Marketing.Acme	File Scan, Create, Read, Write
\MKTG\FORECAST	Mgr.Europe.Marketing.Acme	File Scan, Read, Write
	Mgr.Asia.Marketing.Acme	File Scan, Read, Write

Now assume that Bob.Europe.Marketing.Acme is made security equivalent to Mgr.Europe.Marketing.Acme, and similarly Cheryl.Asia.Marketing is made security equivalent to Mgr.Asia.Marketing.Acme. By this assignment, each will obtain the *Access Control* right to the respective \MKTG\EUROPE or \MKTG\ASIA directory, and the *File Scan, Read*, and *Write* rights to the \MKTG\FORECAST directory. If Bob goes on vacation, Cheryl can be made security equivalent to Mgr.Europe.Marketing.Acme and will instantly obtain the rights usually exercised by Bob. Note that it is not sufficient for Cheryl to be made security equivalent to Bob, because Bob is not directly a trustee, and security equivalence is not transitive.

By making Edward.Acme security equivalent to Mktg-Mgr.Marketing.Acme, he will obtain the Supervisor right to the marketing portion of the file system. Note that no one has access to the root of the file system tree: because of inheritance, access to the root is rarely granted.

In an analogous fashion, we could assign rights to the \FINANCE portion of the file system. There is, of course, no reason why objects in Finance.Acme could not have rights to files in \MKTG, or vice versa.

Thus, by using security equivalence and inheritance, a small number of access control assignments are sufficient for controlling a large file system tree. Using Organizational Role and Organizational Unit objects as trustees simplifies the management of the file system, which is a key goal of RBAC.

4.2 NDS Examples

Again consider the NDS structure as shown previously in Figure 11-2. Table 11-2, *Sample NDS Trustee Assignments*, shows sample ACL assignments for this configuration. With these trustee assignments, all users in Finance.Acme will have the *Browse* right to the Finance container, while all users in Marketing.Acme will have the *Browse* right to the Marketing container. The Finance organization has an administrator who has the *Supervisor* right to that portion of the NDS tree, while the Marketing organization has a less powerful administrator with *Create* and *Delete* rights, but not the *Supervisor* right. In addition, the organization as a whole has an administrator who has the *Supervisor* right to the entire tree.

Table 11-2. Sample NDS Trustee Assignments

Object Name	Trustee	Rights
Finance.Acme	Finance.Acme	Browse
	Manager.Finance.Acme	Supervisor
Marketing.Acme	Marketing.Acme	Browse
	Mktg-Mgr.Marketing.Acme	Create, Delete
Acme	Admin.Acme	Supervisor

By making Edward.Acme security equivalent to Admin.Acme, he will obtain the *Supervisor* right to the whole tree. If Sally.Finance.Acme is made security equivalent to Manager.Finance.Acme, then she will have the

Supervisor right to the Finance part of the NDS tree. Using an Inherited Rights Filter, Admin.Acme could be blocked from having any rights in Finance.Acme, thus allowing only Sally to administer those portions of the tree.

Because of security equivalence, any user can take over administration of the tree simply by being made security equivalent to Admin.Acme (or Manager.Finance.Acme, for that portion of the tree). As in the file system examples, because security equivalence is not transitive, it is not sufficient to make a user security equivalent to Sally, because her rights are not assigned directly, but rather come from security equivalence.

Thus, using assignment of rights to Organizational Unit and Organizational Role objects, we can configure access rights in the NDS tree with a bare minimum of configuration settings.

5.0 Conclusions

RBAC can be partially implemented using existing commercial products. The inability to provide some of the features suggested by the [SAND96a] family of models suggests that perhaps a finer-grained distinction of features would be desirable, rather than an all-inclusive definition of meeting a given set of RBAC criteria. By analogy, this is similar to defining a security target using the ITSEC [ITSEC91] or the proposed Common Criteria [COMM96] and comparing a product to the target, rather than using a one-size-fits-all approach to security as in the Orange Book [DOD85].

NetWare provides many useful features for implementing RBAC. It would be significantly more useful if it provided the ability for users to select sessions by selecting at login time what objects they want to be security equivalent to (as a subset of their authorized set), and transitivity in security equivalence.

6.0 References

[COMM96] Common Criteria Editorial Board, *Common Criteria for Information Technology Security Evaluation*, Version 1.0, January 1996. Available from: http://csrc.nist.gov/nistpubs/cc/read_me.ccl.

[DOD85] U.S. Department of Defense, *Trusted Computer Systems Evaluation Criteria*, DOD 5200.28-STD, Washington, DC, December 1985.

[ITSEC91] *Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonised Criteria*, Version 1.2, Luxembourg: Office for Official Publications of the European Communities, June 1991.

[SAND96a] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, “Role-Based Access Control,” *IEEE Computer*, 29:2, February 1996, 38-47.

[SCHA94] M. Schaefer, G. Grossman, and J. Epstein, “Using a Semiformal Model 2C a C2 Better,” *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD, 11-14 October 1994, 153-164.