GROUP HIERARCHIES WITH DECENTRALIZED USER ASSIGNMENT IN WINDOWS NT[*]

RAVI SANDHU and GAIL-JOON AHN
George Mason University

Abstract

The notion of groups in Windows NT is much like that in other operating systems. Rather than set user and file rights individually for each and every user, the administrator can give rights to various groups, then place users within those groups. In this paper we describe an experiment to extend the Windows NT group mechanism in two significant ways that are useful in managing group-based access control in large-scale systems. The goal of our experiment is to demonstrate how group hierarchies (where groups include other groups) and decentralized user-group assignment (where administrators are selectively delegated authority to assign certain users to certain groups) can be implemented by means of Microsoft remote procedure call (RPC) programs. In both respects the experimental goal is to implement previously published models (RBAC96 for group hierarchies and URA97 for decentralized user-group assignment). Our results indicate that Windows NT has adequate flexibility to accommodate sophisticated access control models to some extent.

Keywords: Security, RBAC, Windows NT

1 INTRODUCTION

Groups have been used for access control ever since the first time-sharing systems were implemented in the early 1970s. A group is a collection of users and serves as a convenient unit for granting and revoking access.

Every account in NT's user database contains a group membership list indicating which groups the account belongs to [4]. Users belonging to a group are explicitly displayed with the User Manager program. Windows NT notably lacks a facility for including one group in another.[1] In practice, it is often desirable that groups bear some relationship to each other. By allowing membership in a group to automatically imply membership in some other groups we can reduce the number of explicit access decisions that need to be made by users and administrators. Many commercial database management systems, such as Informix, Oracle and Sybase, provide facilities for hierarchical groups (or roles). Commercial operating systems, however, provide limited facilities at best for this purpose.

Another limitation of Windows NT groups is that membership is exclusively controlled by built-in administrator groups such as *Account Operators*, *Administrators*, and *Domain Admins* [4]. This is a centralized model which does not scale gracefully to systems with large numbers of groups and users. More generally, it is possible to decentralize user-group assignment by allowing administrators to selectively delegate authority to assign certain users to certain groups.

In this paper we describe an experiment to extend the Windows NT group mechanism to include group hierarchies and decentralized user-group assignment can be implemented by means of Microsoft RPC programs.

Our model for group hierarchies is based on the RBAC96 model for role-based access control [7].[2] The model for decentralized user-group assignment, called URA97, is adapted from [5]. Neither model was designed with Microsoft RPC programs in mind. There are numerous papers in the literature on hierarchical groups and alternate models for this purpose including [2, 3, 6].

The example of figure 1 is taken from [5]. URA97 distinguishes between regular groups and administra-

[*]This work is partially supported by grant CCR-9503560 from the National Science Foundation and by the University Research Program of NSA at the Laboratory for Information Security Technology at George Mason University.

All correspondence should be addressed to Ravi Sandhu, ISE Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22030, `sandhu@isse.gmu.edu`, `www.list.gmu.edu`.

[1]Even though a local group can include global group(s) as a member, Windows NT doesn't support the hierarchical relationship between global groups or between local groups.

[2]The notion of a role is similar to that of a group, particularly when we focus on the issue of user-role or user-group membership. For our purpose in this paper we can treat the concepts of roles and groups as essentially identical.

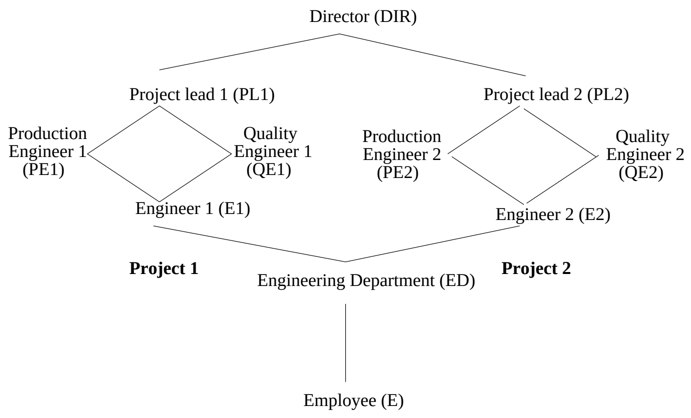

Figure 1: AN EXAMPLE GROUP HIERARCHY

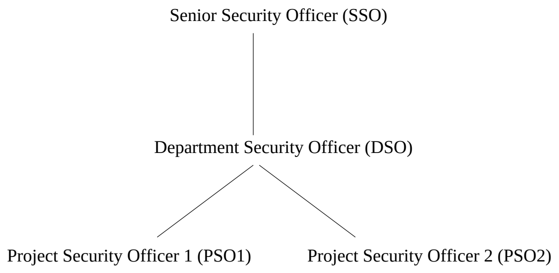
Senior Security Officer (SSO)
Department Security Officer (DSO)
Project Security Officer 1 (PSO1)
Project Security Officer 2 (PSO2)

Figure 2: AN EXAMPLE ADMINISTRATIVE GROUP HIERARCHY

tive groups. Figure 2 shows a hierarchy of administrative groups. We will use this example throughout this paper. These administrative groups are authorized to grant and revoke membership of users in the regular groups of figure 1, as we will see shortly.

The rest of the paper is organized as follows. In section 2, we discuss how to implement group hierarchies in Windows NT. In section 3 we review the URA97 model and discuss its implementation in Windows NT. Implementation overview is described in section 4. Section 5 concludes the paper.

2 GROUP HIERARCHIES

We show how group hierarchies can be simulated in Windows NT. The basic idea is that when a user added to a senior group the assign program automatically adds the user to all junior groups. Similarly, when a user is removed from a senior group the revoke program automatically removes the user from appropriate junior roles.

To maintain the group hierarchy we use the file

Group Name	Parent Group(s)	Child Group(s)
DIR	-	PL1, PL2
PL1	DIR	PE1, QE1
PL2	DIR	PE2, QE2
PE1	PL1	E1
QE1	PL1	E1
PE2	PL2	E2
QE2	PL2	E2
E1	PE1, QE1	ED
E2	PE2, QE2	ED
ED	E1, E2	E
E	ED	-

Table 1: THE EXAMPLE GROUP HIERARCHY OF FIGURE 1

`grouphr.txt` to store the children and parents of each group. The group hierarchy of figure 1 is represented in `grouphr.txt` as shown in table 1. The first column gives the group name, the second column gives the (immediate) parent groups of that group, and the third column gives the (immediate) children. The null symbol "−" means that the group has no parent or child as the case may be.

Using `grouphr.txt`, we can find all seniors and juniors for a group by respectively chasing the parents and children.

We say a user is an *explicit* member of a group if the user is explicitly designated as a member of the group. A user is an *implicit* member of a group if the user is an explicit member of some senior group. A user can simultaneously be an explicit and implicit member of the same group.[3] To simulate a group hierarchy we use information about explicit and implicit membership in `account database`. If Alice belongs explicitly or implicitly to a group she will be added to that group's member list in `account database`. However, `account database` is not sufficient to distinguish the case where Alice is both an explicit and implicit member of some group from the case where she is only an implicit member of the group. For this purpose we introduce another file `explicit.txt` that keep information about explicit membership only.

The implementation of group hierarchy by explicitly assigning a member of a senior group to be a member of all junior group in `account database` may raise a scalability issue. For example, many Unix implementations

[3]This is a property of the RBAC96 and URA97 models on which our experiment is based. There are other models which do not allow this.

limit the number of groups activated in a process to a fairly small number such as 32 or 16, so this approach does not scale for Unix. We conducted a small experiment to ascertain how many group can be activated in a process on Windows NT. Our experiment indicated that Windows NT can accommodate up to 993 groups simultaneously activated in a single process.[4] This is a sizable number so large group hierarchies can be accommodated by means of this approach.

3 DECENTRALIZED GROUPS

Windows NT centralizes user-group assignment and revocation entirely in hands of built-in administrator groups. However, this simple approach does not scale to large systems. Clearly it is desirable to decentralize user-group assignment to some degree so that expensive system administrators do not need to spend valuable time on routine tasks. In particular we can use administrative groups for this purpose.

Sandhu and Bhamidipati [5] recently introduced the URA97 model for decentralized administration of user-role membership (URA97 stands for user-role assignment 1997). This section reviews URA97 and the next one describes our approach to implementing it in Windows NT. In our review of URA97 we will use the term group rather than role. Our description of URA97 is informal and intuitive. A formal statement of URA97 is given in [5]. We emphasize that URA97 was defined in earlier work independent of any consideration of its implementation in NT.

3.1 User-Group Assignment

There are two issues that need to be addressed in decentralized management of group membership. Firstly we would like to control the groups that an administrative group has authority over. Recall figures 1 and 2 which respectively show the regular and administrative groups of our example. We would like to say, for example, that the PSO1 administrative group controls membership in project 1 groups, i.e., E1, PE1, QE1 and PL1. Secondly, it is also important to control which users are eligible for membership in these groups.

URA97 addresses these two issues respectively by means of a *group range*[5] and a *prerequisite group* or more generally a *prerequisite condition*. URA97 has a *can_assign* relation which we store in the file `can_assign.txt`.

Assignment of a user to a group in URA97 means explicit assignment. Implicit assignment to junior groups happens as a consequence and side-effect of explicit assignment. In other words `can_assign.txt` applies only to explicit membership.

[4]It is actually possible to assign a user upto 1000 global groups but this causes logon failure due to too many SIDs. As we reduced the number of groups, we found that the Windows NT system works successfully with 993 groups simultaneously assigned to a user. This result must depend on the internal data structure of `account database` in the NT kernel.

Administrative Group	Group Range
PSO1:	[E1,PL1):
PSO2:	[E2,PL2):
DSO:	(ED,DIR):
SSO:	[ED,DIR]:

Table 2: EXAMPLE OF `can_revoke.txt`

3.2 User-Group Revocation

URA97 authorizes revocation by the *can_revoke* relation which we store in the `can_revoke.txt` file. An example is shown in table 2. The meaning of each row in `can_revoke.txt` is that a member of the administrative group can revoke membership of a user from any regular group in group range. We would typically expect some correlation between the range authorized for an administrative group in `can_assign.txt` and in `can_revoke.txt`, but this is not required by the model.

URA97 defines two notions of revocation called *weak* and *strong*. Weak revocation is straightforward and has impact only on explicit membership in the group in question. Strong revocation requires revocation of both explicit and implicit membership.

4 IMPLEMENTATION OVERVIEW

We use Microsoft RPC to enforce desired behavior of URA97 with respect to different administrative groups. The RPC mechanism is the simplest way to implement client-server applications, because it keeps the details of network communications out of the application code. The security of RPC is part of the operating system that uses it. Therefore, Microsoft RPC on Windows NT can use the Windows NT security built in as part of the operating system. The Windows NT security

[5]In our actual implementation, we use group set which is identical to group range. i.e., group set for group range [E1,PL1] is E1, PE1, QE1, PL1.

FILENAME	OWNER	PERMISSION
`RPCserver.exe`	rbac	Everyone: X
`RPCclient.exe`	rbac	Everyone: X
`assign.exe`	rbac	rbac : X
`weak_revoke.exe`	rbac	rbac : X
`strong_revoke.exe`	rbac	rbac : X
`explicit.txt`	rbac	rbac : RW
`can_assign.txt`	rbac	rbac : RW
`can_revoke.txt`	rbac	rbac : RW
`grouphr.txt`	rbac	rbac : RW

Table 3: THE PERMISSION OF REFERENCE FILES AND PROCEDURES

model is designed for C2-level security, as defined by the U.S. Department of Defense [1]. One of the most important requirements of C2-level security is that the owner of a resource (such as a file) must be able to control access to the resource. In order to use this aspect, we use named pipes as RPC's transport mechanism, and since pipes are part of the file system, we can use NT security mechanisms using Microsoft RPC. We reiterate, the security is implemented by the OS (Windows NT) and not by the RPC mechanism.[6]

To implement URA97 in Windows NT we also use several reference files introduced in the previous sections and set their permission bits as shown in table 3.[7] These procedures can read and write the four reference files. We previously described the structure of files `explicit.txt` and `grouphr.txt` in section 2, and `can_assign.txt` and `can_revoke.txt` in section 3. For simplicity all these files in our implementation are owned by user rbac.

Three procedures (`assign, weak_revoke, and strong_revoke`) are called at the Windows NT command line prompt(which is actually DOS prompt). An alternate GUI interface for these procedures is also provided.

5 CONCLUSION

In this paper we have described our experiment to provide two useful extensions to the Windows NT group mechanism by means of Microsoft RPC programs. First we have added hierarchical groups by means of explicit assignment to junior groups. When a user is assigned to a senior group the system *automatically* adds the user to all junior groups. Similarly, when a user's membership is revoked from a group, revocation from appropriate junior groups is *automatically* carried out. This behavior is adapted from the RBAC96 model. Secondly we have adapted the URA97 model for decentralized user-group assignment and implemented it in Windows NT. Our implementations use Microsoft RPC programs to enforce authorization to add and remove users from groups. Our results indicate that Windows NT has adequate flexibility to accommodate sophisticated access control models to some extent. We also indicated that the Windows NT has better scalability in simulating group hierarchies by explicit assignment to junior groups, as compared with Unix.

[6] Usually, a protocol sequence in RPC contains options for network communications protocols. Named pipes (`ncacn_np`) is one option of transport protocols for communications.

[7] Each entry has the name of either a user account or a group, and a set of permissions that apply to that account or accounts in that group. `R W X` stand for `READ`, `WRITE`, and `EXECUTE` respectively.

References

[1] Microsoft Press. *Microsoft Windows NT Server Networking Guide.* Microsoft Press, 1997.

[2] Matunda Nyanchama and Sylvia Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgernstern, and C. Landwehr, editors, *Database Security VIII: Status and Prospects.* North-Holland, 1995.

[3] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, 16(1), 1991.

[4] Charles B. Rutstein. *Windows NT Security.* McGraw-Hill, 1997.

[5] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects.* North-Holland, 1997.

[6] Ravi S. Sandhu. The NTree: A two dimension partial order for protection groups. *ACM Transactions on Computer Systems*, 6(2):197–222, May 1988.

[7] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.