Managing Role/Permission Relationships Using Object Access Types

John Barkley
Anthony Cincotta
National Institute of Standards and Technology
Gaithersburg MD 20899
jbarkley@nist.gov

July 24, 1998

Abstract

The role metaphor in Role Based Access Control (RBAC) is particularly powerful in its ability to express access policy in terms of the way in which administrators view organizations. Much of the effort in providing administrative tools for RBAC has been devoted to tools for associating users with roles and roles with roles. This paper introduces the concept of an "Object Access Type" and describes the tool "RGP-Admin" for administering associations between roles and permissions using Object Access Types. RGP-Admin is applicable to most RBAC mechanisms and Access Control List mechanisms which support groups. A prototype demonstration of RGP-Admin was developed to illustrate how Object Access Types are used to manage associations between Windows NT groups, representing roles, and file permissions within the Windows NT File System.

1 Introduction

Administering users' access to resources is often accomplished by directly associating users with permissions. This approach can be particularly difficult, error-prone, and costly to administer when users enter and leave an organization, and when users' responsibilities change within an organization. Role Based Access Control is designed to address this problem. Simple RBAC mechanisms and Access Control List (ACL) mechanisms which support groups are often equivalent in their functionality and their ability to describe policy[1].

With RBAC, users are assigned to roles. Roles are assigned permissions and when users' responsibility changes, their role assignment changes. As a result, a user's permissions change. The advantage of this approach is that once an organization's role permissions are defined, then administering permission assignments when user responsibility changes is no longer necessary.

Because permissions are usually attributes stored with objects and objects are usually widely dispersed among storage media, administering permission assignments can be inherently expensive. Within an organization, changes in user responsibility typically occur more often than changes in role permissions. Thus, by associating permissions with roles and by moving users in and out these roles, the amount of permission assignment administration can be reduced and consequently, the total cost of security administration can be reduced.

With RBAC, much of the effort in providing administrative tools has been devoted to tools for associating users with roles and roles with roles. This paper introduces the concept of an "Object Access Type" and describes the tool "RGP-Admin" for administering associations between roles and permissions using Object Access Types. The use of Object Access Types is applicable to most RBAC and ACL mechanisms. A prototype demonstration of RGP-Admin which runs on Windows NT[1] was developed to illustrate the use of RGP-Admin with Windows NT groups, which represent roles, and file permissions.

Much of this paper has been derived from the experiences of the NIST team which implemented RBAC on the World Wide Web (RBAC/Web)[2] for Unix servers and adapted the RBAC/Web Admin Tool for use directly with Windows NT. A version of the Admin Tool has also been developed for use in a relational database environment. RBAC has several advantages over ACLs. Even a very simple RBAC model affords an administrator the opportunity to express an access control policy in terms of the way that the organization is viewed, i.e., in terms of the roles that individuals play within the organization. With RBAC, it is not necessary to translate a natural organizational view into another view in order to accommodate an access control mechanism. In addition, most RBAC models have features which most ACLs do

[1] Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.

not. In particular, many RBAC models[3][4] have role hierarchies where one role can "inherit" another.

Section 2 introduces the concept of an Object Access Type. Section 3 describes in general terms the features of RGP-Admin. Section 4 describes a prototype demonstration implementation of RGP-Admin for Windows NT.

2 Object Access Types

A permission can be described as authorization to perform an operation on an object. An access control policy which uses roles defines an association between a role and the permissions for that role. This association can be represented as a 3-tuple:

(role, object, permitted operations on object)

which means that a user assigned to the role is authorized to perform an operation on the object only if the operation is a member of the set of permitted operations for that object.

This representation is isomorphic to a representation of the form:

(object, role, permitted operations on object)

where the first and second elements of the 3-tuple are interchanged. In this equivalent representation, for each object, there is a list of roles and associated permitted operations for those roles on that object. This list is the RBAC information for the object. For each object, this information may be different. However, for many objects this information may be the same. An "Object Access Type" is an access control information specification which can be manipulated as an entity separate from the object with which it is associated. Object Access Types can be created, edited, deleted, assigned to objects, and removed from objects.

3 RGP-Admin

RGP-Admin manages the association of roles and permissions by means of views of Object Access Types, roles, objects, and permissions and an Object Access Type Editor. RGP-Admin:

- Provides views of associations between Object Access Types and objects, and between role permitted operations and objects; and
- Sets or removes an object's Object Access Type; and,
- Defines, saves, and recalls Object Access Types and Object Access Type collections.

RGP-Admin has three components: the Object Access Type View, the Object Access Type Editor, and the Role Permission View.

3.1 Object Access Type View

Object Access Type assignment may be viewed by selecting an Object Access Type and a set of objects. For each object in the selected set, RGP-Admin displays the object icon as green, if the object has access control information which is an instance of the selected Object Access Type, or red, if the object does not. The selected Object Access Type may be set to a selected object and optionally, to objects inherited by that object, if the object is displayed red, or removed from a selected object and optionally, objects inherited by that object, if the object is displayed as green. The Object Access Type View also:

- creates, saves, and recalls Object Access Type collections; and,
- obtains the access control information for an object and adds it to an Object Access Type Collection.

Figure 2 shows a sample Object Access Type View as it is implemented in the RGP-Admin Prototype Demo for Windows NT.

3.2 Object Access Type Editor

The Object Access Type Editor creates and edits Object Access Types. Object Access Types are managed by adding or removing a role from the Object Access Type and by modifying the permissions associated with that role. RGP-Admin's Object Access Type View sets objects to Object Access Types or removes Object Access Types from objects. Figure 3 shows a sample Object Access Type Editor as it is implemented in the RGP-Admin Prototype Demo for Windows NT.

3.3 Role Permission View

RGP-Admin graphically displays object access by role in order to verify the access permissions set by means of the RGP-Admin Object Access Type View or by means of some other tool, such as, Windows NT Explorer in the case of the prototype implementation. RGP-Admin defines object access by means of Object Access Types in the Object Access Type View. When an Object Access Type definition is not required, e.g., the number of objects to be set to specific access control information is small, then tools other than RGP-Admin might be used to set the access control information. In the Role Permission View, RGP-Admin displays the access associated with selected objects for a selected role. Different kinds of access may be displayed in order to answer the questions in sections 3.3.1 and 3.3.2. Figure 4 shows a sample Role Permission View as it is implemented in the RGP-Admin Prototype Demo for Windows NT.

3.3.1 To which objects does a selected role have specifically selected access?

This question is answered by selecting a set of objects, a role, and specific permissions. For each object in the selected set, RGP-Admin displays the object icon as green, if the selected role has all of the selected permissions, or red, if the selected role does not have all of the selected permissions.

3.3.2 To which objects does a selected role have any access?

This question is answered by selecting a set of objects, a role, and by leaving all specific permissions unselected. For each object in the selected set, RGP-Admin displays the object icon as blue, if the selected role has any access, i.e., any permission to access the object, or red, if the selected role has no access to the object.

3.3.3 Hierarchy Mode

The capability for one role to inherit another role is a common feature of RBAC models, e.g., the Sandhu $RBAC_1$ Model[5], the NIST model[3], the SQL3 RBAC model[4]. A role hierarchy is a strict partial ordering[6] (i.e., like "<", asymmetric and transitive) on the set of roles. One can think of role inheritance as the capability for one role to be authorized for (or ``included in") another role. SQL3 implements role hierarchies in this manner.

When roles or groups have hierarchies, it can be important to know whether a given role *effectively* has access to an object. With hierarchies, access is permitted either as a result of the permissions associated with that role or as a result of the given role inheriting some other role that has permission to access the object. It can also be important to know whether access is permitted to the object as a result of the permissions defined for the role itself or is based on permissions associated with inherited roles.

In the Role Permissions View, if hierarchy mode is selected, then the questions of sections 3.3.1 and 3.3.2 are answered based on the effective access of the selected role, i.e., based on the permissions authorized for the selected role and any of its inherited roles. In addition, the roles inherited by the selected role is displayed. If hierarchy mode is not selected, then the questions of sections 3.3.1 and 3.3.2 are answered based only on the permissions authorized for the selected role.

3.3.4 Path Mode

Like roles, objects, such as, files and processes, can be organized into hierarchies. In such object hierarchies, it is important to know not only the access of a role to an object, but also, to know whether the path in the hierarchy to the object can be traversed.

In Role Permission View, if Path Mode is selected, then for nodes in the hierarchy not shown as end nodes, the questions of sections 3.3.1 and 3.3.2 are answered based on whether the selected role has permission to traverse these intermediate nodes. Nodes in the hierarchy shown as end nodes are displayed normally. If Path Mode is not selected, then the questions of sections 3.3.1 and 3.3.2 are answered for all nodes based on the selected role. Note that when no permissions are selected, Path Mode is irrelevant since the permission to traverse an intermediate node is included in the concept of any access.

4 RGP-Admin Prototype Demo

In the prototype demonstration of RGP-Admin for Windows NT:

- Objects are NT File System (NTFS) files or directories.
- Roles are Windows NT groups.
- Object Access Types are NTFS ACL specifications.
- Permissions are the NTFS file permissions: Read(R), Write(W), Execute(X), Delete(D), Change Permissions(P), Take Ownership(O).

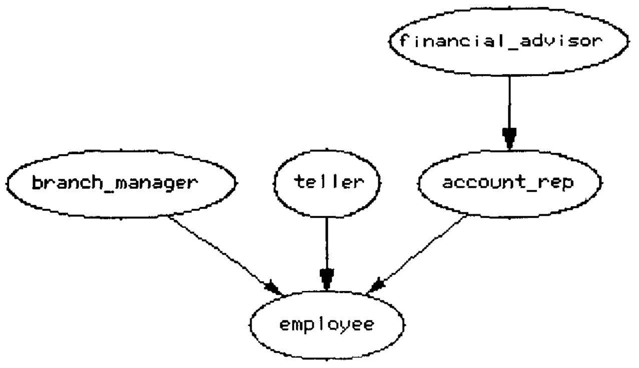

Figure 1: Roles in a bank.

The Prototype Demo illustrates a small portion of a banking environment where the roles and their hierarchy are shown in figure 1. The roles branch_manager and teller are two that one might expect at a bank's branch office. The role account_rep is authorized for the bank's account representatives who sit at the desks outside of the teller windows. The role financial_advisor is authorized for an account representative who is trained in recommending non-insured investment products. The role financial_advisor inherits account_rep because financial_advisor needs to be able to open and close accounts. The roles account_rep, branch_manager, financial_advisor, and teller inherit the role employee since any user authorized for these roles is a bank employee.

Table 1 shows the role permissions defined for each of the Object Access Types. There are four Object Access Types:

- The Object Access Type account applies to files that contain individual account information and to directories that hold such files.

- The Object Access Type cd_to_dir provides all roles with the capability of traversing a directory in order to access files in the directory.

- The Object Access Type employee applies to files readable by all employees and to directories which contain employee related files. Employees need to be able to write in such directories in order to create suggestions files.

- The Object Access Type suggestions applies to files created by employees containing suggestions for more efficient bank operations.

The RGP-Admin Prototype Demo uses a pair of parenthesized permission lists for describing file and directory permissions. This notation is also used in Windows NT Explorer. For example, table 1 shows that role branch_manager has permissions "(RX) (R)" for the accounts Object Access Type. When a file is the accounts Object Access Type, then branch_manager has Read permission for that file. When a directory is an accounts Object Access Type, then branch_manager has Read and Execute permission for the directory. When a directory or file is created within a directory of Object Access Type accounts, then the file or directory created has Object Access Type accounts.

In support of bank policy, the role permissions for each of the Object Access Types are:

- The role account_rep must be able to create and delete account files. Thus, account_rep has permission to traverse and write into the accounts directory, and read and delete account files. Note that account_rep does not have permission to read the accounts directory because bank policy is that account_rep does not need to be able to create a list of all account holders. Also, the account_rep does not need to write into an account file. When an account is created or deleted, the initial deposit or final withdrawal from the account must be performed by a teller.

- The role branch_manger has permission to read the account directory and account files, and to read and delete suggestion files.

	Object Access Type			
Role	accounts	cd_to_dir	Employee_ read	suggestions
account_ rep	(WX)(RD)	(X)()		
branch_ manager	(RX)(R)	(X)()		(R)(D)
employee		(X)()	(RWX)(R)	(X)(R)
financial_ advisor	(R)()	(X)()		
teller	(X)(RW)	(X)()		

Table 1: Role permissions for each Object Access Type

- The role employee has permission to read all employee information but does not have any permissions to access information about accounts.

- The role financial_advisor is able to read the accounts directory, thus, obtaining a list of all account holders. This permission is necessary in order for financial_advisor to be able to derive marketing information about current account holders in order to identify account holders who might be interested in the bank's uninsured investments. Since financial_advisor inherits account_rep, financial_advisor is has the permissions necessary to function as an account_rep. A financial_advisor needs permission to open and close both insured and uninsured accounts.

- The role teller must have the permission to make changes (deposits and withdrawals) from the account files. Thus, teller has permission to traverse the account directory and read/write account files. Note that teller has no permission for creating or deleting accounts. That is the responsibility of the account_rep.

In order for all roles to be able to traverse the directory tree to access files for which they have permissions, all roles have Execute permission on directories with cd_to_dir Object Access Type.

4.1 Example Displays

Figure 2 shows a sample Object Access Type Window of the RGP-Admin Prototype Demo. This view color-codes file icons green if the file has the selected Object Access Type or red if not as shown. In figure 2, the Object Access Type accounts is selected and the icons for the directory accounts and the files *.acc are displayed in green indicating that those files have Object Access Type accounts. The icons for the files a:\, bank_files, and empl_info are displayed in red indicating that those files have some other Object Access Type. On a black and white display, "red" is the darker shading.

Files shown in red can be set to the accounts Object Access Type. If the file to be set is a directory, then files and/or directories in the directory or in the directory tree can be set to accounts. In addition, using this display, a file's Object Access Type can be copied and added to a collection.

Figure 3 shows a sample of the Object Access Type Editor Window. Using this display, an Object Access Type can be created and/or edited from a collection. In figure 3, the accounts Object Access Type has been selected from the collection for editing. Figure 3 shows that employee is the only role available to be added to the accounts Object Access Type. The Permissions checkboxes are used to set the permissions for a selected role in accounts.

Figure 4 shows a sample of the Role/Group Permission View Window. Because the Prototype Demo illustrates a Windows NT implementation, the title "Role/Group Permission View" serves as a reminder that roles are Windows NT groups. This view color-codes file icons green if the selected role has the selected permission for the file and red if not. If no permissions are selected, then file icons are color-coded blue if the selected role has any access. If Hierarchy Mode is selected, then the color-coding is based on the selected role and any additional permissions that the selected role inherits from the roles which it inherits.

In figure 4, the role financial_advisor, Hierarchy Mode, and Read permission are selected. All files except a:\ and bank_files are shown in green indicating that financial_advisor has read permission on those files. On a black and white display, "green" is the lighter shading. Note that financial_advisor has Read permission for the accounts directory as a result of the fact that the directory has Object Access Type accounts (see tab. 1 and fig. 2). The role financial_advisor has Read permission on all of the other files displayed in green as a result of inheritance.

If Path Mode were also selected, then the directories a:\ and bank_files would also be shown in green indicating that financial_advisor has either Read or Execute permission. If so, financial_advisor is able to traverse those directories. If no permissions are selected, then file icons are colored blue if the selected role has any access.

4.2 Windows NT Implementation Issues

Windows NT does not support multilevel group hierarchies. Windows NT Local Groups can contain Domain Groups as members, but Domain Groups cannot contain Local Groups or other Domain Groups. The RGP-Admin Prototype Demo includes multilevel group hierarchy in order to demonstrate how RGP-Admin could be implemented with such a feature. Most RBAC mechanisms support role hierarchies. The use of hierarchies in administering the user/role relationships can significantly reduce administrative costs.

One approach to implementing multilevel role hierarchies in Windows NT is to represent role hierarchies by manipulating user membership in NT groups according to a hierarchy representation external from the NT User Manager Database. This is the approach taken in the adaptation of the RBAC/Web Admin Tool for Windows NT. If role A inherits role B, then all users in the NT group representing role A are members of the NT group representing role B.

Windows NT supports users as entries in ACLs and negative permissions associated with ACL entries. These features present a problem with regards to what it means for a role, i.e., an NT group, "to have a permission for a file." The preferred meaning would be: "any user who is authorized for a role (i.e., a member of an NT group which is an entry with positive permissions in a file's ACL) has the access indicated by the permissions associated with the entry."

With ACLs that can have entries with negative permissions overriding entries with positive permissions, this preferred meaning is not possible in general. For example, an ACL may have an entry for a role with positive permissions and also an entry negating those permissions for a user authorized for that role. The result is that although the user is authorized for the role and the role associated with permission, that particular user does not have permission while all other users authorized for that role have permission. Thus in general, the meaning of the statement "role having a permission for a file" becomes: "the role has an entry with positive permissions in the file's ACL."

The nature of Windows NT ACLs also implies an alternative meaning for Hierarchy Mode. Hierarchy Mode becomes "Effective User Access" mode which only applies to users. When this mode is selected, the *effective* access of a selected user (i.e., whether this user has access to the file as determined by Windows NT criteria) color-codes the file icons according to the selected permissions. When this mode is not selected, then the color-coding of the file icons indicates that the selected user or role has an entry in the ACL with the selected positive permissions.

5 Summary

The use of roles in administering access policy is usually less error-prone and less costly than directly associating users with permissions. Much of the effort in developing administrative tools to support the use of roles has been devoted to tools which associate users with roles. RGP-Admin is a tool that manages the association of roles and permissions by means of Object Access Types.

An Object Access Type is an access control information specification which can be manipulated as an entity separate from the object with which it is associated. Just as roles and objects can have hierarchies, hierarchical relationships could also be defined for Object Access Types based on permission inheritance. This might be useful for reusing access specifications across applications, projects, and/or organizational units. The concept of an Object Access Type hierarchy is a subject for further work.

RGP-Admin allows Object Access Types to be created, edited, and associated with objects. In addition, RGP-Admin displays graphically the association of Object Access Types and objects, and the association of roles, permissions, and objects.

References

[1] John Barkley. Comparing Simple Role Based Access Control Models and Access Control Lists. In *Second ACM Workshop on Role Based Access Control*, November 1997.

[2] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrilla, and D.R. Kuhn. Role Based Access Control for the World Wide Web. In *20th National Information System Security Conference*. NIST/NSA, 1997.

[3] D. Ferraiolo, J. Cugini, and D.R. Kuhn. Role Based Access Control: Features and Motivations. In *Annual Computer Security Applications Conference*. IEEE Computer Society Press, 1995.

[4] ISO/IEC 9075, (Working Draft) Database Language SQL - Part 2: Foundation. Document ISO/IEC JTC1/SC21 N10489, July 1996.

[5] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role Based Access Control Models. *IEEE Computer*, 29(2), February 1996.

[6] Patrick Suppes. *Axiomatic Set Theory*. Van Nostrand, Princeton, New Jersey, 1960.

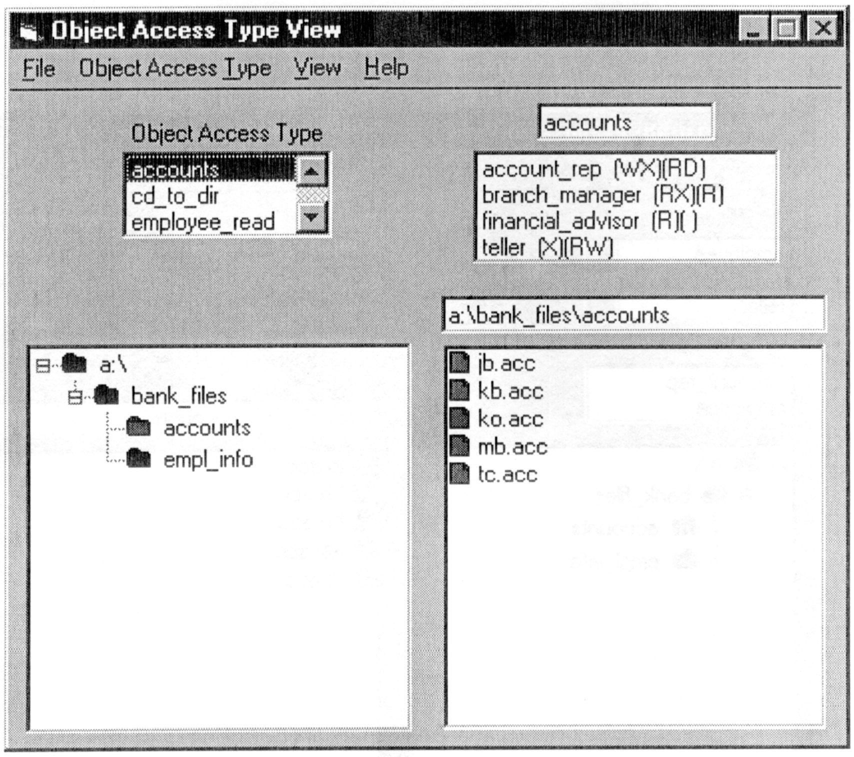

Figure 2: Object Access Type View Window

Figure 2 shows that the Object Access Type accounts is selected and that the icons for the directory accounts and the files *.acc are displayed in green indicating that those files have Object Access Type accounts. The icons for the files a:\, bank_files, and empl_info are displayed in red indicating that those files have some other Object Access Type. On a black and white display, "red" is the darker shading.

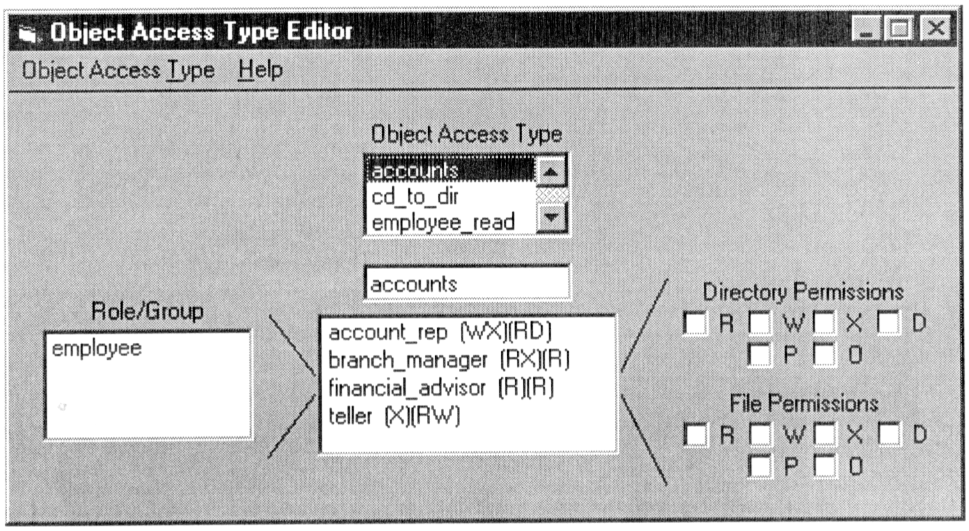

Figure 3: Object Access Type Edit Window

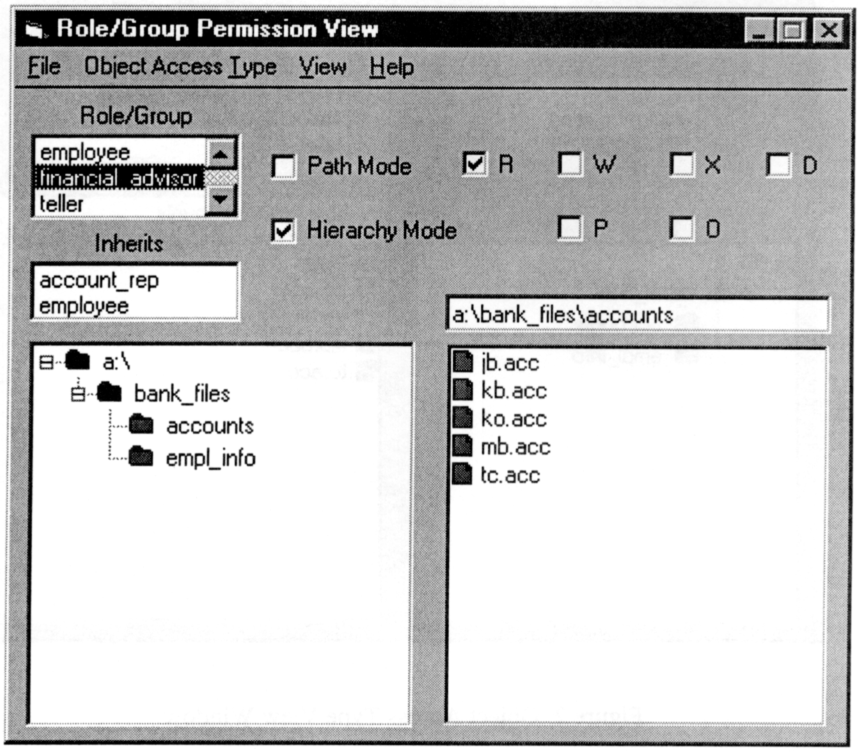

Figure 4: Role/Group Permission View Window.

Figure 4 shows that the role financial_advisor, Hierarchy Mode, and Read permission are selected and that all files except a:\ and bank_files are shown in green indicating that financial_advisor has read permission on those files. On a black and white display, "green" is the lighter shading.