Controlling Database Access by Providing Access Permissions on Database Objects

[1]Manushi Majumdar, [2]Anu Unni, [3]Hrishikesh Babar, [4]Abhishek Birjepatil, [5]Prof. Anand Magar
Sinhgad Academy of Engineering,
(University of Pune),
Kondhwa-Bk, Pune

Abstract— The paer focuses on securing a table in the database, to the row level, implying that not all the rows in a database can be accessed by the end user. A user can access and perform operations on only those rows which he has been assigned right to access. Traditionally, there are various ways and techniques to achieve this. The approach adopted by us is analogous to the NTFS security and Permission model, where every row in the database table corresponds to an NTfs object, i.e a folder or file. Permissions controlling access are applied to every database object. Moreover, a hierarchial structure of the objects can be enabled using the Role Based Access Policy (RBAC). Using RBAC principles, multiple hierarchy could also be supported, where in an object could act as a parent object to another object, thus implementing inheritance of permissions from parent to child object. The resultant permission obtained on the child object could be calculated and resultantly enforced when a user wants to access it.

Index Terms— NTFS, Permission, RBAC, Access Control, Database, Hierarchy, Database Object

1 INTRODUCTION

In this era of Information, a very huge amount of information is stored in the form of a database. For instance, in a business environment, databases are maintained to store client details, financial information, human resource details, i.e. all the data that keeps your company in business. Hence it is extremely important to secure the data in the database to make sure that the data is not accessed by people unauthorized to do so. Also, the database needs to be secured even at the row-level of a table, so as to prevent all the data contained in the table to be accessed by even those unauthorized for the same. An example of a table storing the employee details of an organization could be considered for the above case. Not every employee must be allowed to see the details of a particular employee. To achieve this, access to the database has to be controlled on an extremely granular level.

The New Technology File System(NTFS), incorporated by the Microsoft Corporation for it's Windows Operating Systems was meant to replace their initial File System FAT[1]. NTFS has many advantages over FAT, including reliability, increased storage capacity and efficiency. But a striking feature of NTFS is its security, i.e. NTFS offers a secure environment and flexible control over what can be accessed by which users, to allow for many different users and groups of users to be networked together, with each able to access only the appropriate data.[2] Our paper aims to follow the NTFS security and permission model to adopt their security principles at the database level, so as provide for controlled access. Also the Role Based Access Control technique is extended on the database model, to limit access to users based on the roles they have and the permissions and access rights granted to the roles.

Details of Authors: *1,2,3,4 are Students of B.E (Information Technology) at Sinhgad Academy of Engineering. 5 is a professor at the Information Technology Department of Sinhgad Academy of Engineering, and is the faculty guide of the afro mentioned students who have performed the research.*
Email-id: *manushi.majumdar@gmail.com, anu33unni@gmail.com, hrishikeshbabar@gmail.com, abhishek_birjepatil@hotmail.com, anand7375@gmail.com*

2 LITERATURE SURVEY

2.1 NTFS

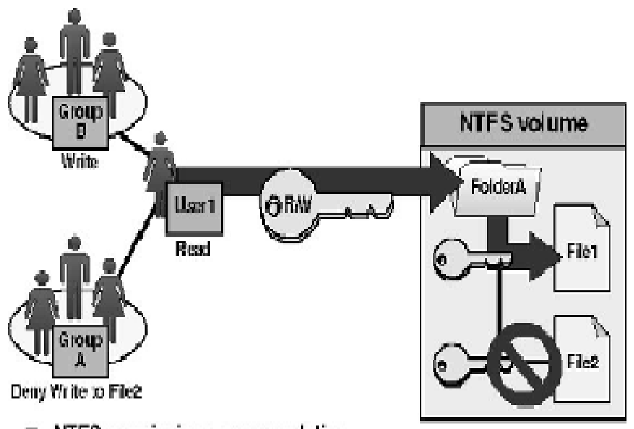
Security in NTFS s oriented around the key concept of assigning rights to specific users or groups of users. As per requirement, a number of user accounts are created, each comprising of a user or a group of users. The user accounts are classified based on some common credentials held by the users, and the access rights that the respective account has on the files and folders present. A set of predefined permissions are enabled on the contained files and folders based on the respective accounts. Access control lists (ACLs) are used to manage which users and groups of users are allowed to access different files and folders (objects) within NTFS volumes.

These ACLs contains entries that specify what rights each user or group has for the object in question. These access rights are called permissions. The six basic permissions created for NTFS objects are READ(R), WRITE(W), EXECUTE(X), DELETE(D), TAKE OWNERSHIP(O), CHANGE PERMISSIONS(P). Additionally one can even apply a FULL CONTROL to a group where the group is granted all the permissions and complete access. NTFS permissions can be "granted" or "denied". Another feature of NTFS security is inheritance of permissions, where the child file or folder inherits the permissions applied on it's parent folder or other folders up the hierarchy.

But for any file or folder, the explicitly applied permissions have higher precedence over the inherited permissions. Similarly, permissions denied have a higher precedence over permissions granted.
The following Rules apply in the process of permission resolution:

1. "Deny" permissions take precedence over "allow" permissions.
2. Permissions applied directly to an object take precedence over permissions inherited from a parent object.
3. Permissions inherited from near relatives take precedence over permissions inherited from distant predecessors. So permissions inherited from the object's parent folder take precedence over permissions inherited from the object's "grandparent" folder, and so on.
4. Permissions from different user groups that are at the same level (in terms of being directly-set or inherited, and in terms of being "deny" or "allow") are cumulative. So if a user is a member of two groups, one of which has an "allow" permission of "Read" and the other has an "allow" of "Write", the user will have both read and write permission--depending on the other rules above, of course)

Hence the hierarchy followed by the permissions is as follows: Explicit Deny -> Explicit Allow -> Inherited Deny -> Inherited Allow [3]



- NTFS permissions are cumulative.
- File permissions override folder permissions.
- Deny overrides other permissions.

Fig. NTFS Permission Enforcement

2.2RBAC

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file. Roles are defined according to job competency, authority, and responsibility within the enterprise. RBAC is a specialized permission model that applies permissions on the predefined roles, and roles are assigned to one or more users. It is a technique that brings the set of users on one side and the set of permissions on the other.[4] Thus, in order to access an object, a user needs to hold a particular role that contains permissions to access the object. The four components of the RBAC system are as follows:

A. CORE RBAC

It embodies the essential aspects of RBAC. The basic concept of RBAC is that users are assigned to roles, and users acquire permissions by being members of roles. Core RBAC includes requirements that user-role and permission-role assignment can be many-to-many. It includes requirements for user-role review whereby the roles assigned to a specific user can be determined as well as users assigned to specific role. A similar requirement for permission-role review is imposed as an advanced review feature. It allows includes the concept of user sessions, which allows selective activation and deactivation of roles. Finally it requires that users be able to simultaneously exercise permission of multiple roles. This precludes products that restrict users of activation of one role at a time.[5]

B. HIERARCHICAL RBAC

It adds requirements for supporting role hierarchies. A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby the seniors roles acquire the permission of their juniors, and junior roles acquire the user membership of their seniors. This standard recognizes two types of role hierarchies

1.General Hierarchical RBAC: In this case, there is support for an arbitrary partial order to serve as role hierarchy, to include the concept of multiple inheritance of permissions and user membership among roles.

2.Limited Hierarchical RBAC: Some systems may impose restrictions on the role hierarchy. Most commonly, hierarchies are limited to simple structures such as trees and inverted trees[5]

C.SEPARATION OF DUTY RELATIONS

Separation of duty relations are used to enforce conflict of interest policies. Conflict of interest in a role-based system may arise as a result of a user gaining authorization for permissions associated with conflicting roles. One means of preventing this form of conflict of interest is though *static separation of duty* (SSD), that is, to enforce constraints on the assignment of users to roles. The SSD policy can be centrally specified and then uniformly imposed on specific roles. Because of the potential for inconsistencies with respect to static separation of duty relations and inheritance relations of a role hierarchy, we define SSD requirements both in the presence and absence of role hierarchies. SoD policies deter fraud by placing constrains on administrative actions and there by restricting combinations of privileges that are available to users

1.Static Separation of Duty Relations:

Static Separation of Duty. SSD relations place constraints on the assignments of users to roles. Membership in one role may prevent the user from being a member of one or more other roles, depending on the SSD rules enforced.

Static Separation of Duty in the Presence of a Hierarchy. This type of SSD relation works in the same way as basic SSD except that both inherited roles as well as directly assigned roles are considered when enforcing the constraints.

2. Dynamic Separation of Duty Relations:

Dynamic separation of duty (DSD) relations, like SSD relations, limit the permissions that are available to a user. However DSD relations differ from SSD relations by the context in which these limitations are imposed.

DSD requirements limit the availability of the permissions by placing constraints on the roles that can be activated within or across a user's sessions. DSoD policies deter fraud by placing constrains on the roles that can be activated in any given session there by restricting combinations of privileges that are available to users.[4]

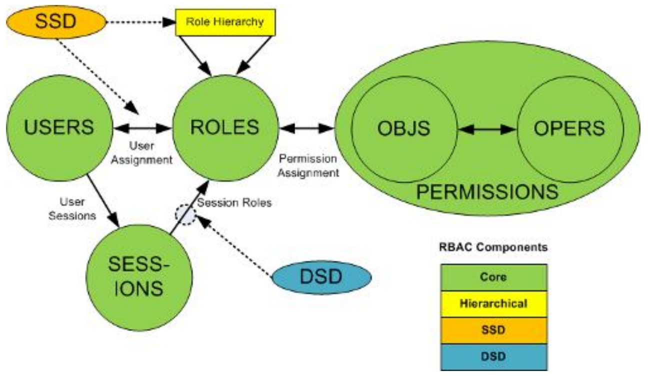


Fig. RBAC Components[6]

3 PROPOSED WORK

All the necessary objects that require controlled access must be stored in the database. RBAC can be implemented by maintaining a hierarchy of those database objects, i.e organizing them in a parent-child structure. This is analogous to NTFS where, a particular folder could have multiple parent folders in a hierarchy as well as multiple file/folders as children in hierarchy. The parent and child of every object, if any, could be maintained in another table that holds the inheritance details. This table could reference the original table that stores all the necessary objects, with the help of its primary key. Thus, for every tuple, the hierarchy table would store the primary key of the objects acting as parent and child. Multiple inheritance, could also be taken care of by this table structure where the child object of a particular parent-child pair, could act as a parent object of another parent-child pair, thus implying the presence of Limited Hierarchy or General Hierarchy RBAC. Also, as in the NTFS model, a particular file system object could belong to multiple

groups, similarly one database object could have multiple parent objects that it is associated to. Hence, as per the demand of the system to be implemented, Static or Dynamic Separation of Duties could be carried out to control access for a session or permanently.
Taking this further, permissions can also be set on the database objects, which further limit the access one will have on the database object, as well as the tasks one is authorized to perform on them. If we consider the example of the basic permissions provided by NTFS, and prepare a structure as follows:

Position	Permission
1	Read
2	Write
3	Execute
4	Delete

Table 1: HierarchyTable

In the above table structure, position is a primary key numeric value indicating the position that the particular Permission has in the PermissionNameTbl. A permission string could be associated with every object in the database. For instance, an Obejct A is assigned permissions Read and Delete, then the resultant permission string for Object A will be "1001" where
1 = permission at the respective position is "assigned"
0 = permission at the respective position is "not assigned"

Additionally, stored procedures could be written to eforce the permissions on the database objects. Whenever the user of the system wishes to access a particular object, a permission check procedure should be called that will evaluate the permissions available on the object by the particular user and thus grant or deny the access request made. This system would also successfully implement RBAC as effective permissions available on an object could vary based on the role held by the particular user/object accessing the object to be accessed. The role of the accessing object will determine the permissions owned by it over the accessed object, which can vary from session to session. Thus the effective permissions over the accesed object will be calculated, thereby restricting access to database objects.

4 CONCLUSION

Thus we have seen how the NTFS permission principles can be extended on the database. This approach not only has benefits of access control that we are looking to achieve, but also provides the additional benefit of grouping of users that can simplify delegation of tasks. Additionally, RBAC has its own benefits when the database has to be created for an enterprise or for a scenario where users distinguished based on their roles.In a scenario like this, distinguishing people based on identity proves to be expensive as well as inconvenient. RBAC provides for security based on the roles one has, naturally introducing a hierarchy on the system designed. Moreover, introducing permissions that a user has on an object, helps regulate access checks down to the database level.

5 REFERENCES

[1] www.ntfs.com

[2] http://www.pcguide.com/ref/hdd/file/ntfs/sec.htm

[3] MCSE Self-Paced Training Kit: Microsoft® Windows® 2000 Core Requirements, Exams 70-210, 70-215, 70-216, 70-217, 2nd Edition

[4] Role Based Accesss Control, Second Edition by David F. Ferrialo, D. Richard Kuhn, Ramaswamy Chandramouli

[5]Role Based Access Control Models by Dr.Saeed Rajput and Reena Cherukuri.

[6]http://www.joshuatreesoftware.us/iamfortress/javadocs/api/com/jts/fortress/AccessMgr.html