A white paper on designing

“Authentication, Authorization & Accountability”

Raghavendra K S
Jan 2002

Introduction

A user management system is a necessary component in most software systems. A good and flexible user management system not only makes it easy to capture complex access controls/business policies but also helps in easy maintenance to change the rules any time. This is of immense importance in terms of reduced costs in development needed (if any) when new rules come up. Also, business rules can change whenever needed, with out worrying too much about restrictions of the underlying system.

Also, a well designed and well built user management is an ideal piece for software reuse.

This document comes up with a blue print for a simple user management system that gives a lot of flexibility in terms of administration of users.

Goals

The following will be the goals that the rest of this document will aim at:

1. Decoupling of authentication, authorization and accountability processes (compliance to the spirit of AAA standards)
2. Ease of administration
3. Care will be taken for administration to be easy and in turn give less room to human errors, which could lead the system vulnerable to attacks.

Most of this document will borrow concepts from RBAC (role based access controls) as opposed to the traditional ACL's. There are a few points that have been added to the traditional RBAC to make management of user more flexible and simple to use.

We will first look into authentication support from the system. The next section will then move onto Authorization and finally we will address accountability in the last section.

Authentication

Authentication of users should happen by challenge response because it is not desirable for the password to travel on the wire. Authentication module will just authenticate the user and will have nothing to do with authorization. Once the authentication is complete the user, depending on the "role" assumed would be authorized to a set of privileges by a separate module.

Further, the central repository where the keys reside will be protected both physically and by using encryption. The system should ensure that none of the passwords will be less than 8 characters nor will they be "weak". All the passwords on the system should be encrypted using a one-way hash.

Authorization

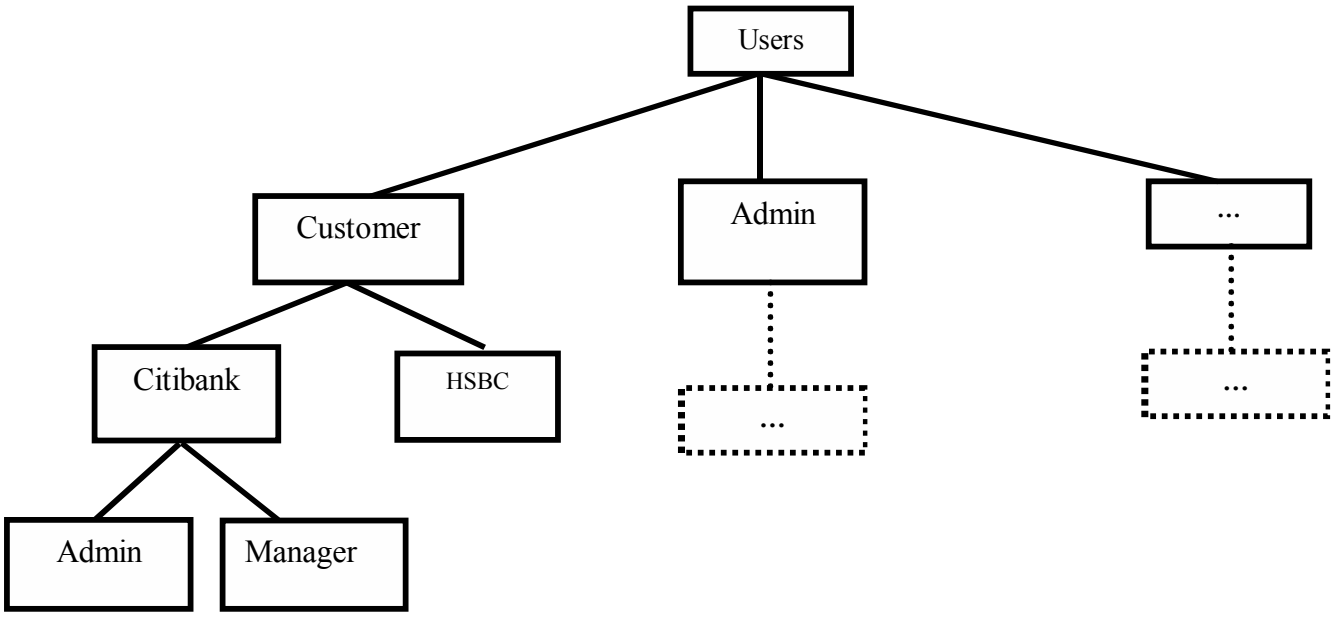
Once a user is authenticated, it needs to be decided what his/her "rights" are. For this we advocate to have a role-based model for authorization. Roles add a useful level of redirection between users and the resources. The fact that user/role associations change more frequently than role/permission associations results in reduced administrative costs when using a role-based authorization model compared to associating users directly with permissions. Each user belongs to one or more roles and roles are mapped to the set of privileges. In other words, access rights are grouped by role name and users are granted membership into roles based on their competencies and responsibilities. To support such a role based authorization, a sophisticated user management system needs to be in place. Properties of such a system are described in the following sub section.

User profiles and resource accesses

Depending on the application, it is suggested to either pick a MAC (Mandatory Access control) policy or a DAC (Discretionary access control) policy. In MAC, "resources" belong to the "organization" and not really to individual users. Users cannot sub-delegate access permissions onto other users/roles at their discretion. In a DAC scheme such sub-delegation by users is permitted.

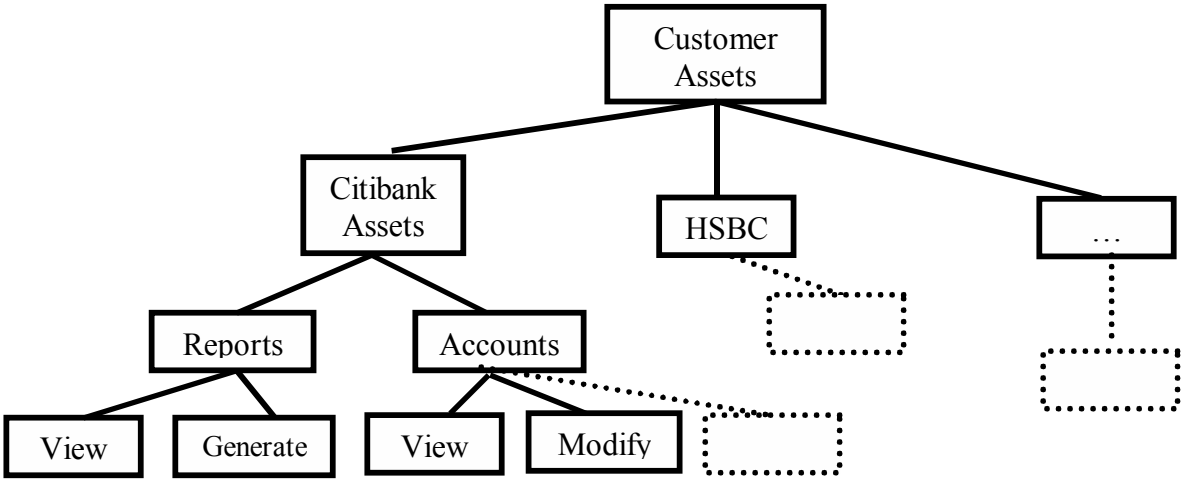
Whether it is decided to go forward with MAC or DAC policy, the following should be supported by the user management system.

1. Ability to define access rights.
2. Support to group access rights into roles.
   - ◊ Create a new role
   - ◊ Remove a role
     - ◊ Referential integrity shall be maintained (No user shall be mapped to a role that doesn't exist)
   - ◊ Modify a role
3. Each right/privilege can be negative or positive.
   - ◊ To "empower" or "debar"
4. Support to map users to a role.
   - ◊ Addition and revocation of users to a role should be facilitated.
   - ◊ Support to map groups to roles is not necessary for the project in hand. (grouping can be achieved by supporting hierarchy in the roles)
5. Hierarchy of roles should be supported. The following diagram should give a feel for the hierarchy of roles. It reflects a highly simplified role hierarchy for a company that hosts banking services of different banks (ASP model).



- ◊ Users belonging to different roles may need to perform common operations. For example, some general operations may need to be performed by all users of the system. In such a situation, it would be inefficient and administratively cumbersome to specify repeatedly these general operations for each role that gets created. Role hierarchies are a natural way to tackle these situations. Any user assuming a role, by default assumes the role of the "parent" role also.
- ◊ If the role hierarchy is viewed as a tree of nodes, in case of a clash of rights between the "ancestor" role node and the role node to which the user belongs to, the rights applying to the present role node shall prevail. The metaphor here is: "The properties of the subclass overrides those of the properties belonging to the parent class in case of a clash"
- ◊ If the user belongs to two roles and none of the roles is a clear ancestor of the other, clashes in access rights becomes a tricky issue to resolve. As a policy, the negative access right will dominate. This offers more security.

6. Hierarchy of resources should be supported for more flexibility in user administration. A way to classify resource rights could be as shown in the following figure.



Note that in the above picture, the leaf levels are actually actions that can be performed on the parent. Conceptually there is nothing wrong about considering these as resources themselves and placing them in the hierarchy tree (as shown above). If a role, Manager is attached to View "leaf" of "Reports" then it means that Manager can view the reports.

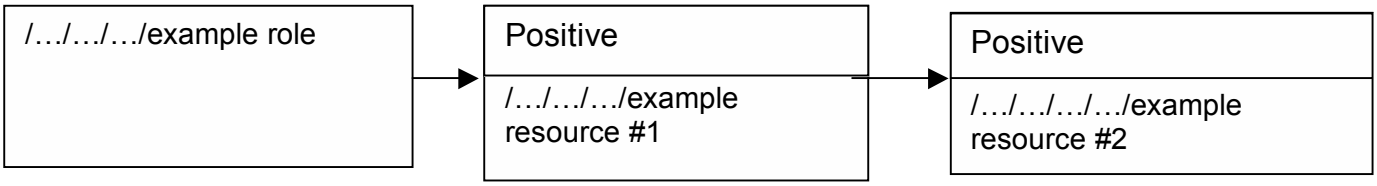
While implementing it may be convenient to stop in the hierarchy at one level higher and not consider "actions" that can be performed on a resource in the resource hierarchy. In the same example, if Manager wants to be given access to view the reports, it can be encoded at the time of associating Manager role with Reports. You may have a flag for this association to describe what all is possible (view, update, create etc). If this approach is followed, it may considerably decrease the row entries in the database (in the table that associates roles and resources)

7. Once the role and resource hierarchies are designed, it is a straight job t o associate roles and resources.

   Various data structures can be used for the same. A couple of examples to capture this relationship using some simple data structures are shown below.

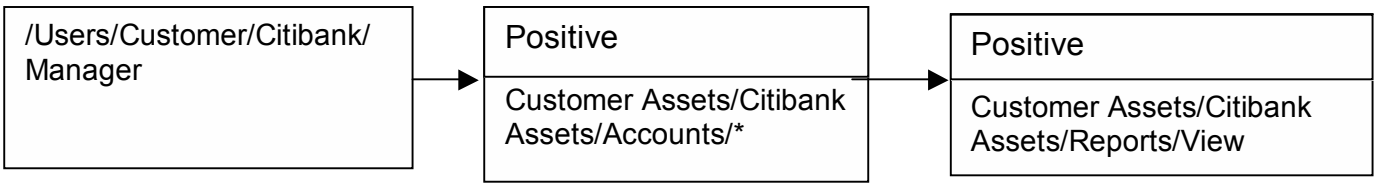
   XPATH notation and "linked-list" like structure are used for showing the examples for associations.

1. Role “example role” is associated to “example resource #1” and “example resource #2”



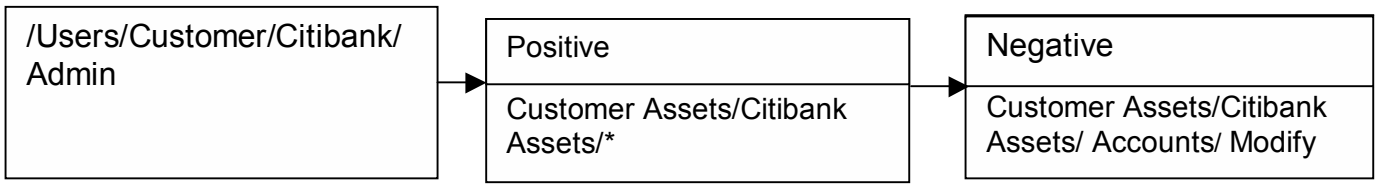
Example #1:

1. Citibank Manager should have full access (read/modify) anything related to accounts.
2. Citibank Manager should be able to view reports.



Example #2:

1. Citibank admin can do anything that is related to Citibank.
2. However, he is not entitled to modify accounts of Citibank.



8. When roles cannot be used to represent relationships, it is common to program access design logic directly into an application. This approach restricts the ability to change access policy in a timely manner. To take care of relationships and "competencies/responsibilities" that cannot be "burnt" into a role, there is an excellent paper "Supporting Relationships in

Access Control using Role Based Access Control by John Barkley, Konstantin Beznosov, Jinny Uppal. The designer of a user management module is urged to read this paper.

Accountability

Audits are much easier in a role-based model than in an ACL model. The former being a "user/role" centric model, checks can be made so that no user is given more "access rights" than is necessary. The role provides a clarification and concise identification of policies for given functions. This allows for the rights/authorities of a person to be clearly documented. Doing such a check in a "resource" centric ACL is more painful as one would need to search the entire set of rights for all the resources.

Whenever a user has placed a request for authorization, it should be logged for future reference. Further, in the entire solution, every important activity needs to be logged for inspection in the future.

References:

1. Role Based Access controls by Michael Lebkicher
2. Supporting Relationships in Access Control using Role Based Access Control by John Barkley, Konstantin Beznosov, Jinny Uppal
3. Role based access control website