Role based RBAC Administration model: Extension to ARBAC97 and Administration of Separation of Duty Constraint

Ashish Gupta, Alka Jindal

Department of I.T.
Punjab Engineering College (DU)
INDIA

Abstract

In role-based access control (RBAC) permissions are associated with roles, and users are made members of roles thereby acquiring the roles permissions. ARBAC97 model provides a platform to administer role hierarchy. The motivation behind this extension is to simplify administration. This paper discusses the existing administrative model ARBAC97 and some of its featured problems. This paper provides a view at those problems and proposes equations, which provide relatively better administration and control. The concept of administration extent is provided to realize decentralized administration of RBAC. A list called character role list is also introduced to secure the integrity of role hierarchy. This paper also highlights the concept of administration of separation of duty constraint and its effect on RBAC. Operation like adding and deleting separation of duty constraint are specified. A series of conflict checking rules to maintain consistency and the administration of authorization constraints are provided in this proposed solution. The main advantage of proposed solution is its simplicity and practicality.

Keywords- ARBAC97, Separation of duty, Access control, RBAC, administration.

1. Introduction

Role-based access control has become widely accepted as an attractive access control technology which separates users and permissions logically by a three-layer user-role-permission structure, thereby simplifies the administration of authorization efficiently. By means of defining constraints, it can also realize the principle of least privilege and implement the management policies of enterprises flexibly in a practical application. For a large system, with hundreds of roles, thousands of users and millions of permissions, how to manage these roles, users, permissions and their interrelationships is very important for the correct implementation of RBAC security policies. An appealing characteristic is to use RBAC itself to manage RBAC, to further provide administrative convenience, especially in decentralizing administrative authority, responsibility and chores.

Access control is mechanism which is concerned with determining the allowed activities of legitimate users, mediating every attempt by a user to access a resource in the system. The objectives of an access control system are often described in terms of protecting system resources against inappropriate or undesired user access [13].

Role-based access control (RBAC) models have been the subject of considerable research in recent years resulting in several important models: the NIST model [1]; the role graph model [2]; the RBAC96 model [3]; the graph-based formalism for RBAC [4] and the unified NIST RBAC model [5][12]. It has been suggested that such models provide an attractive theoretical framework for multi-domain, distributed systems [6] [14]. The features that make RBAC attractive include policy neutrality, principle of least privilege and ease of management. Gligor [7] provides a good introduction to the characteristics
and advantages of RBAC. Management of RBAC models is also an important concept. Previous approaches to role-based administration have either been centralized, such as the NIST model [1] and the role graph model [2], or decentralized, such as the RBAC96 model [3], the ARBAC97 model [8][9], the graph-based formalism for RBAC [4] and SARBAC model. Many other papers in this field are presented [21] [23] [24].

In the next section, we present background of RBAC models. Explaining some of them helps in understanding the whole concept of RBAC. In third section, we move to administration of access control models and see what ARBAC97 model lacks. In section four, problem statement is given, which is the basis of this paper. In fifth section, solution to problem is explained and we also introduce concept of administration of separation of duty.

2. Journey of Access Control

A basic characterization of role-based access control was given in the call for papers of the First ACM Workshop on Role-based Access Control:

“... the essence of Role-Based Access Control (RBAC) is that rights and permissions are assigned to roles rather than to individual users. Users acquire these rights by virtue of being assigned membership in appropriate roles.”

A role is properly viewed as a semantic construct around which access control policy is formulated. The role is more stable because an organization's activities or functions usually change less frequently. The concept of role-based access control (RBAC) began with multi-user and multi-application on-line systems pioneered in the 1970s. In early 90s, Ferraiolo and Kuhn [11] in their paper explained in detail the concept of RBAC. Later other authors like Mohammed and Ditts in 1994; Nyanchama and Osborn in 1993; Ting in 1992; von Solms and van der Merwe in 1994 also presented papers using idea of role based access control. A breakthrough came when Ravi Sandhu with co-authors in 1996 presented RBAC96 model [3]. It is foundation of every model being followed these days. RBAC96 suggested four sub-models RBAC0, RBAC1, RBAC2 and RBAC3.

Since 1995 the formal development of role-based access control models has taken place in North America, notably at George Mason University, NIST and the SETA Corporation, and in Canada at the University of Western Ontario. As organizations grew larger their role hierarchy also increased, the need to manage them became more important. ARBAC97 i.e. administrative role based access control model was solution to this problem introduced in 1997. The authors of ARBAC97 believe that the ability of role-based access control to reduce the administrative burden of access control in a large decentralized organization can usefully be employed in the administration of role-based access control itself. ARBAC97 was further extended to ARBAC99 in 1999. In 2002, Crampton introduced SARBAC model. However, ARBAC97 remained popular with industry and is implemented in various products. In next section, brief discussion of ARBAC97 is presented. [8]

3. ARBAC97

As above said, ARBAC97 has the ability of role-based access control to reduce the administrative burden of access control in a large decentralized organization can usefully be employed in the administration of role-based access control itself [15]. The model identifies three aspects of role-based administration that need consideration: administration of user-role assignment (URA97); administration of permission-role assignment (PRA97); and administration of role-role assignment (RRA97) - that is, administration of the role hierarchy. Consider an example to get a clear view of the concept. This is the same example as used by authors of ARBAC97.
In role hierarchy, permissions are inherited from junior roles by senior roles.

3.1. **URA97; User role assignment**

URA97 is concerned with the management of the user assignment to roles by administrative roles. Administrative roles can assign users to roles and revoke assignments of users to roles provided they have authority to do so. A URA97 constraint is defined recursively as follows:

- $r$ and $\overline{r}$ are constraints, where $r \in R$. Here $R$ represents range.
- $c_1 \land c_2$ and $c_1 \lor c_2$ are constraints, where $c_1$ and $c_2$ are constraints.

A constraint is evaluated with respect to a user $u$. In particular,

- $r$ is satisfied if $r \in \downarrow R(u)$. Here $\downarrow R(u)$ represents hierarchy under $u$.
- $\overline{r}$ is satisfied if $\overline{r} \not\in \downarrow R(u)$;
- $c_1 \land c_2$ is satisfied if $c_1$ and $c_2$ are satisfied;
- $c_1 \lor c_2$ is satisfied if $c_1$ or $c_2$ is satisfied.

URA97 defines two relations, \textit{can-assign} $\subseteq AR \times C \times R$ and \textit{can-revoke} $\subseteq AR \times R$, where $C$ is the set of constraints and $R$ is the set of ranges in $R$ and $AR$ is the set of administrative roles. If $(a, c, R') \in \textit{can-assign}$, then $(u, r)$ can be added to UA by the administrative role $a$ provided $r \in R'$ and $u$ satisfies constraint $c$. Similarly, if $(a, R') \in \textit{can-revoke}$, then $(u, r)$ can be removed from UA by a provided $r \in R'$.

<table>
<thead>
<tr>
<th>Administrative role</th>
<th>Prerequisite role</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>ED</td>
<td>[E1, PL1]</td>
</tr>
<tr>
<td>PSO2</td>
<td>ED</td>
<td>[E2, PL2]</td>
</tr>
<tr>
<td>DSO</td>
<td>$ED \land \overline{PL1}$</td>
<td>[PL2,PL2]</td>
</tr>
</tbody>
</table>

\textit{can-assign}
This is an example of can-assign and can-revoke. Roles are specified in URA97 by following range notation.

\[
\begin{align*}
[x,y] &= \{ r \in R \mid x \geq r \land r \geq y \} \\
(x,y) &= \{ r \in R \mid x > r \land r \geq y \} \\
[x,y) &= \{ r \in R \mid x \geq r \land r > y \} \\
(x,y) &= \{ r \in R \mid x > r \land r > y \}
\end{align*}
\]

According to given example, PSO1 can assign users in ED to the roles E1, PE1 and QE1, and similarly for PSO2 with respect to E2, PE2 and QE2. DSO can assign a user in ED to PL1 provided that user is not already in PL2, and similarly for PL2 with respect to PL1. In next sub section, permission assignment model is considered.

### 3.2. PRA97; Permission role assignment

PRA97 is concerned with the management of the permission assignment, PA relation. Administrative roles are able to assign permissions to roles and revoke assignments of permissions to roles. A PRA97 constraint has the same structure as a URA97 constraint but is evaluated with respect to a permission, p. Specifically we have

- r satisfies if \( r \in \uparrow R(p) \) (note that this is an inversion of the condition for the satisfaction of \( r \) in URA97);
- \( r' \) satisfies if \( r \not\in \uparrow R(p) \);
- \( c_1 \land c_2 \) and \( c_1 \lor c_2 \), where \( c_1 \) and \( c_2 \) are constraints, is satisfied if \( c_1 \) and \( c_2 \) are satisfied;
- \( c_1 \lor c_2 \), where \( c_1 \) and \( c_2 \) are constraints, is satisfied if \( c_1 \) or \( c_2 \) is satisfied.

PRA97 defines two relations can-assignp \( \subseteq AR \times C \times R \) and can-revokep \( \subseteq AR \times R \) which are analogous to the URA97 relations can-assign and can-revoke, respectively. Specifically, if \((a, c, R') \in can-assignp\), then \((p, r)\) can be added to PA by the administrative role \( a \) provided \( r \in R' \) and \( p \) satisfies constraint \( c \); and if \((a, R') \in can-revokep\), then the administrative role \( a \) can remove \((p, r)\) from PA, provided \( r \in R' \). Following example explains the permission assignment to roles.

<table>
<thead>
<tr>
<th>Administrative Role</th>
<th>Prerequisite Condition</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSO</td>
<td>DIR</td>
<td>[PL1, PL2]</td>
</tr>
<tr>
<td>DSO</td>
<td>DIR</td>
<td>[PL2, PE2]</td>
</tr>
<tr>
<td>PSO1</td>
<td>PL1 \land QE1</td>
<td>[PE1, PE2]</td>
</tr>
<tr>
<td>PSO1</td>
<td>PL1 \land PE1</td>
<td>[QE1, QE2]</td>
</tr>
<tr>
<td>PSO2</td>
<td>PL2 \land OE2</td>
<td>[PE2, PE2]</td>
</tr>
<tr>
<td>PSO2</td>
<td>PL2 \land PE2</td>
<td>[QE2, QE2]</td>
</tr>
</tbody>
</table>
These relations are analogous to the URA97 relations. Earlier these were for user assignment, now they are for role assignment. The most important and complex part of ARBAC97 model is RRA97. RRA97 model applies for role hierarchy management by administrative roles.

3.3. RRA97; Role role assignment

RRA97 considers the administration of the role hierarchy. RRA97 addresses these operations: role insertion, role deletion, edge insertion and edge deletion. The model also addresses the following problems [10].

- No cycles should be introduced into the hierarchy.
- Successive changes to the role hierarchy “should not lead to undesirable side effects”.
- What is the effect of role hierarchy operations on the other relations in ARBAC97?
- How should permissions and users that are assigned to a deleted role be dealt with?

However, these questions are out of scope of this paper. The central idea in RRA97 is the relation can-modify \( \subseteq AR \times R \). If \((a, R') \in \text{can-modify}\), i.e. administrative role a can make changes to the hierarchy within the range \( R' \). \( R' \) is referred to as an authority range. An example of can-modify in RRA97.

<table>
<thead>
<tr>
<th>Administrative Role</th>
<th>UP- Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>(E1, PL1)</td>
</tr>
<tr>
<td>DSO</td>
<td>(ED, DIR)</td>
</tr>
</tbody>
</table>

The meaning of can-modify(x,Y) is that a member of the administrative role x (or a member of an administrative role that, is senior to x) can create and delete roles in the range Y, except for the endpoints of Y, and can modify relationships between roles in the range Y. In short, we can explain operations like AddRole, DeleteRole, AddEgde and DeleteEgde as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>ARBAC97</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRole (a,r, (\Delta r, \nabla r))</td>
<td>(\Delta r = {r'}, \nabla r = {r''})</td>
</tr>
<tr>
<td></td>
<td>((a, R') \in \text{can-modify})</td>
</tr>
<tr>
<td></td>
<td>(r', r'' \in R')</td>
</tr>
<tr>
<td></td>
<td>((r', r'')) is a create range</td>
</tr>
<tr>
<td>DeleteRole (a,r)</td>
<td>((a, R') \in \text{can-modify})</td>
</tr>
<tr>
<td></td>
<td>(r \in R')</td>
</tr>
<tr>
<td></td>
<td>(r) is not the end point of any range in any ARBAC97 relation</td>
</tr>
</tbody>
</table>
AddEdge (a,r,r') 
(a,R') \in can - modify
r, r' \in R'
(r) = (r')

DeleteEdge (a,r,r') 
(a,R') \in can - modify
r, r' \in R'

So, ARBAC97 provided a basis of further research in administrative field. However, certain problems persisted in ARBAC97, we will focus on them.

Before moving further here are some doubtful points in URA97 and PRA97. Comments on URA97 are also broadly applicable on PRA97.
- Revocation requires more careful consideration.
- It is not clear whether a user-role assignment can only succeed if a tuple in the can-assign relation is satisfied. In other words, is the default behavior in ARBAC97 that a user-role assignment fails if there is no appropriate tuple in the can-assign relation?
- URA97 does not consider whether users (acting in administrative roles) can assign roles to themselves.
- Does URA97 apply to the assignment of administrative roles to users?

These issues need consideration to improve flexibility and security of both URA97 and PRA97. One another important issue is of separation of duty (SOD). SOD means, a single duty is distributed among various users to authorize the task.

3.4. Separation of duty policies

Separation of duty is the partitioning of a sensitive task into sub-tasks assigned to different users so that the co-operation of two or more users is required to complete the task. The purpose of separation of duty is to prevent a single user compromising the security requirements of an organization. A typical example in commercial environments is to require that one user prepares a cheque and a different user validates that cheque. In a military context, the launch of a missile, for example, may require two authorizations, one each from a different user [16].

In role-based access control a separation of duty requirement is often modeled as a pair of roles. A distinction is made between static separation of duty, where the set of role assignments for each user must not contain both roles in the separation of duty requirement, and dynamic separation of duty, where the roles in each session must not contain both the roles in the separation of duty requirement [17]. Clearly static separation of duty is a stronger constraint on role assignment than dynamic separation of duty.

However, the structure of role-based access control admits several approaches to separation of duty. For example, operational separation of duty is the separation of duty at the level of permissions [18], where sensitive combinations of permissions are identified. Such a combination of permissions cannot be assigned to any single user, even if the set of roles which have those permissions assigned to them do not conflict. Therefore, operational separation of duty is claimed to provide a higher level of assurance of separation of duty requirements than static separation of duty, and hence dynamic separation of duty [19].

Administration of SoD is an important task. This paper also presents administration model for separation of duty. But first, ARBAC97 needs consideration. So in next section, various drawbacks in this model are presented and in later section, solution to the same problems is proposed.
4. Problem Foundation

In administrative RBAC models, major concern is how to provide effective and efficient administrative control over role hierarchy. Administrative roles were introduced to provide an optimum solution to this problem. Management needs are distributed among administrative roles. Every administrative role manages the role hierarchy under his authority. This way, each role of the hierarchy is managed by senior administrative role. However, certain checks have to be applied on these administrative operations. Operations like AddRole, DeleteRole, AddEdge and DeleteEdge are the basic operations performed by any administrative role.

While referring above example and administrative range, certain problems in RRA97 which make it more complex and difficult to manage are presented here.
- RRA97 supports changes to the role hierarchy but does not permit the deletion of a role if it is the end point of a range in some ARBAC97 relation.
- In RRA97 role deletion is not permitted for roles which appear in any of the ranges of the can-predicates. Here can-predicates mean any can tables like can-assign, can-modify and can-revoke.
- Administration of separation of duty constraints is not considered, nor is the effect of changes to the role hierarchy on role constraints.

So, ARBAC97 lacks certain perspectives. Next section continues with the proposal to these problems.

5. Proposed Solution

In this section, firstly, alternative to authority range, which we name “administrative extent”, is discussed. Defining authority through range restricts the model and also introduce problem of dangling range if any roles gets deleted in the range.

Definition: Administrative extent of \( r \in R \) is denoted by \( \text{AE}(r) \) is defined to be the set \( \{ s \in R: s \leq r, \uparrow s/ \uparrow r \subseteq \downarrow r \} \cup r \).

Also there is need for another check while considering administrative extent. Let’s say \( r_j \) is subsequent role of \( r \) and child set of \( r_j \) is non empty and if every upward path starting from each child role \( r_c \) of \( r_j \), then \( r_c \) also belongs to administrative extent of \( r \) and \( r_j \) belongs to extended administrative extent of role \( r \).

In administrative extent, all those roles that are in authority of particular administrative role are defined rather than giving range of roles. An example of administrative extent is as follows:

<table>
<thead>
<tr>
<th>Administrative role</th>
<th>Administrative extent</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>PE1, QE1, E1</td>
</tr>
<tr>
<td>DSO</td>
<td>DIR, PL1, PL2, PE1,</td>
</tr>
<tr>
<td></td>
<td>QE1, PE2, QE2, E1,</td>
</tr>
<tr>
<td></td>
<td>E2, ED</td>
</tr>
</tbody>
</table>

So, administrative extent includes all the features which authority range considered, rather it provides a more flexible authority structure. One problem that is solved by introducing administrative extent is of dangling ranges. If any role is deleted from the hierarchy, there is no need to check whether that role is in can-predicate table or not.

However, every hierarchy has some roles which form the back bone of hierarchy and deletion of these roles causes instability in the hierarchy. Question arises that there has to be a check on role deletion
regarding these backbone roles. Character roles list solves this problem. This list contains all those roles which form the backbone of hierarchy. So role deletion checks in this list the character roles and if role to be deleted is a character role, then deletion is marked invalid. Only SSO i.e. senior security officer will have authority to change this list. Whenever any serious changes in the hierarchy are required, only then SSO can accordingly change these roles. This will solve our problem of role deletion which form backbone of hierarchy.

After getting through the concept of administrative extent, operations of role hierarchy administration are being considered here. As discussed earlier the conditions for role addition or deletion and edge addition and deletion are too restraining. In the following table, we specify novel operational conditions keeping in mind the administrative extent.

<table>
<thead>
<tr>
<th>Function</th>
<th>ARBAC97</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddRole</td>
<td>(a, r, Δ r, ∇ r)</td>
</tr>
<tr>
<td>(a, r)</td>
<td>(a, r) ∈ can – mod ify</td>
</tr>
<tr>
<td>Δ r, ∇ r ∈ can – mod ify(a)</td>
<td>r is the role to be added to hierarchy</td>
</tr>
<tr>
<td>DeleteRole</td>
<td>(a, r) ∈ can – mod ify</td>
</tr>
<tr>
<td>(a, r)</td>
<td>r ∉ character roles list</td>
</tr>
<tr>
<td>AddEdge</td>
<td>(a, r', r')</td>
</tr>
<tr>
<td>(a, r, r')</td>
<td>(a, r') ∈ can – mod ify</td>
</tr>
<tr>
<td>DeleteEdge</td>
<td>(a, r) ∈ can – mod ify</td>
</tr>
<tr>
<td>(a, r)</td>
<td>r, r' ∈ R'</td>
</tr>
</tbody>
</table>

According the above defined equations,

AddRole (a, r, Δ r, ∇ r), here r is the role to be added to hierarchy. Administrative role a can add role r to hierarchy iff r ∈ R' and R’ is in administrative extent of a. Δ r represents immediate children of r and ∇ r represents immediate parents of r. Both must also be in administrative extent of a, so as to validate role addition.

DeleteRole (a, r), here check is implemented whether r belongs to administrative extent of a or not. R must also not belong to list of character roles. During database hierarchy setup, create a list which contains list of backbone list of roles. Deletion of these roles leaves a dangling hierarchy.

AddEdge (a, r, r') Edge to be added between roles r and r’ and both these roles belong to administrative extent of a.

DeleteEdge (a, r, r’) Edge deletion is valid if r lies in administrative extent of a. Checking for r’ in administrative extent of a adds an unnecessary overhead, it adds no security. So, we remove this clause.

There may be other issues that will arise as we evolve this proposed solution. We wish to give administrative roles autonomy within their extent but only as far as the global consequences of the resulting actions are acceptable. In the next sub section, another component which act side by side is administration of separation of duty is presented.
5.1. Administration of Separation of Duty

We will also consider a missing piece in ARBAC97, administration of separation of duty (SoD) constraint. As discussed earlier, separation of duty provides authorization for completing sensitive tasks. Whenever administrative roles update the hierarchy, updating SoD constraint is also necessary. We maintain a table that contains roles which are part of SoD, e.g. for above example, PE1 and QE1 roles are in SoD table. Here is an example of SoD table.

<table>
<thead>
<tr>
<th>Constraint ID</th>
<th>Role list</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID1</td>
<td>PE1, QE1</td>
</tr>
<tr>
<td>CID2</td>
<td>PE2, QE2</td>
</tr>
</tbody>
</table>

Table is maintained for SoD constraints. Whenever any user activates a role, he supplies which role to activate. As, we know which role that user already belongs to, we will check for the desired combination in SoD table. Administration of SoD constraint requires operation like adding and deleting constraints in the table. Administrative extent of roles is given as follows:

<table>
<thead>
<tr>
<th>Administrative Role</th>
<th>Administrative Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSO</td>
<td>R</td>
</tr>
<tr>
<td>PSO1</td>
<td>E1, PE1, QE1</td>
</tr>
<tr>
<td>PSO2</td>
<td>E2, PE2, QE2</td>
</tr>
</tbody>
</table>

So, administrative role PSO1 can update constraints related to E1, PE1 and QE1. Now, we need how to update SoD table. Following table specify the definition of operations that administrative roles can perform of SoD table.

<table>
<thead>
<tr>
<th>Function</th>
<th>Separation of Duty</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddSOD(a,r,r')</td>
<td>(a,R') ∈ can-modify r,r' ∈ R'</td>
</tr>
<tr>
<td>DeleteSOD(a,r,r')</td>
<td>(a,R') ∈ can-modify r,r' ∈ R'</td>
</tr>
</tbody>
</table>

Role ‘a’ can add SoD constraint if r and r’ belong to set R’ and R’ is in administrative extent of a. So, whenever, there is any change in hierarchy, administrators can change SoD table accordingly using commands AddSoD and DeleteSoD. We see DSO can apply separation of duty constraints on any role and PSO2 can add or delete SoD constraints in extent E2, PE2 and QE2. This is a nascent attempt to administer separation of duty constraint, so it needs improvement to make it feasible to be applicable in any industry.

A drawback is that it does not protect SoD from administrator’s attack. As PSO1 can at any time delete constraint PE1-QE1 and log in and later add the constraint PE1-QE1. One proposed solution to this problem is to maintain a log. So, at later stage security officers can check for ambiguities.

Advantages of proposed solution:
1. Introduction of character role list, solve the problem of keeping intact the role hierarchy.
2. Changing the parameters of add and delete functions provide a better control of administrative role over its hierarchy.
3. Administrative extent helps in solving can- predicate problems.
4. Finally, administration of separation of duty constraint keeps the hierarchy safe from any fraud. Also, distributed administration of this constraint lessens the overhead of senior security officer.

This briefs the extent of proposed solution and its advantages. Another advantage of proposed solution is its simplicity and practicality.

We have concentrated on the shortcomings of ARBAC97: it lacks flexibility, coherence and robustness, interaction between sub-models is not completely determined and it is rather complex. Proposed solution tries to overcome some of these short comings. Still there are many opportunities for further work in area of administration of role based access control. Administration of Separation of duty is quite a new topic, extensive research is required in this field. Also, Administrative model still lacks perspectives like overlapping of administrative roles and general roles. So, database administration still has a long way to go.

5. Conclusion

Administration of various components of any role based access control model is an important question. In this paper, the existing ARBAC97 model is discussed. ARBAC97 is widely used in products but has certain drawbacks, which include no can- predicate deletion, no role deletion at edge of range and administration of separation of duty, a missing piece.

Solution to these problems inspired the introduction of administrative scope by Crampton in SARBAC model, but further improved for precision, another thing that is introduced is character roles list, which helps in maintaining back bone of role hierarchy and allows only valid roles to be deleted. Constraints of Operations like addrole and deleterole etc are also modified according needs. Administration of separation of duty (SoD) is also highlighted and specified how operations like AddSoD and DeleteSoD will administer separation of duty constraint. Concept of administration of separation of duty constraint provides a framework for a missing piece in ARBAC97. So, proposed extension to ARBAC97 will help in managing large organizations. Future work in the field of administration of separation of duty includes more specifications on AddSoD and DeleteSoD and constraint table. Also Administration of RBAC still has long way to go to reach new heights.

References

22. QIU Jiong, MA Chen-hua, Yin Jian-wei, Dong Jin-xiang, “Research and implementation of Role based RBAC administration model.” Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT’05)