Administration of RBAC

ISA 767, Secure Electronic Commerce
Xinwen Zhang, xzhang6@gmu.edu
George Mason University
Fall 2005

RBAC

Concept:
- Role: collection of users and permissions
  - Possible other roles if RH exists
  - Group: collection of users and other groups

Size:
- Role: can be no users/no permissions assigned
  - Abilities and groups
  - Group: more than two members in general

Activations:
- Roles: a user can active subset of assigned roles and get partial permissions
  - Least of privilege
- Group: a user get all permissions of the groups that he belongs to

RBAC

- Least of privilege:
  - Only assign necessary permissions to roles
  - Only active necessary permissions in a session

- Separation of duties:
  - Exclusive roles and permissions

- Data abstraction
  - Abstract permissions (rights and objects)

SCALE AND RATE OF CHANGE

- roles: 100s or 1000s
- users: 1000s or 10,000s or more

Simplify management:
- Frequent changes to
  - user-role assignment (UA)
  - permission-role assignment (PA)
  - More static than UA

- Less frequent changes for
  - role hierarchy

Administration of RBAC

- Administrative permissions
  - Create/destroy users
  - Create/destroy roles
  - Create/destroy permissions
  - User-role assignment (UA) and removal
  - Permission-role assignment (PA) and removal
  - Role-role assignment (RA) and removal
  - Define and maintain constraints
Administration of RBAC

- Centralized approaches:
  - Single chief security officer
  - Centralized admin tool and database
  - Role graph
- Decentralized approach:
  - Autonomous administration with decentralized authorities
  - For Scalability
  - Control of autonomous permissions/ranges by higher roles
  - Tradeoff between centralized control and autonomous management

Administrative RBAC

- Using RBAC to manage RBAC
  - Called administrative RBAC (ARBAC)
- Permissions: administrative permissions
  - U, R, P, UA, PA, RH, etc
  - Distinguished from P
- Roles: administrative roles (AR)
  - Distinguished from R
- Users: administrators
  - Generally distinguished from U, but not necessary
  - Decentralized
- User-role assignment: UAU
- Permission-role assignment: APA
- Permission role hierarchy: ARH

ARBAC

- Using RBAC to facilitate administration of RBAC.
  - URA97: Managing user-role assignment
  - PRA97: managing permission-role assignment
  - RRA97: managing role-role assignment to define role hierarchy
    - Include role creation/deletion
  - Creation of users and permissions are not included.
  - Personal management department
  - System/Application administrators

ARBAC97

- Decentralizes user-role assignment (URA97)
- permission-role assignment (PRA97)
- role-role hierarchy (RRA97)
### Simple ARBAC Model
- Use one administrative role to manage a set of regular roles
- Actually one user (say Alice) with administrative role does the management
- Alice can
  - Add/delete users to/from these roles
  - Add/delete permissions to/from these roles
  - Assign these roles to other roles (make arbitrary architecture)
- Problem: Kind of DAC (owner of the roles)
  - no control of what kind of users can be assigned to these roles
  - No control of what kind of permissions can be assigned to these roles
  - No control of what kind of roles can inherit these roles

### URA97 Grant Model
- Use negative prerequisite role for conditions
  - For mutual exclusive roles

<table>
<thead>
<tr>
<th>Admin Role</th>
<th>Prereq Cond</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>ED</td>
<td>[E1,E1]</td>
</tr>
<tr>
<td>PSO1</td>
<td>ED ∧¬PE1</td>
<td>[QE1,QE1]</td>
</tr>
<tr>
<td>PSO1</td>
<td>ED ∧¬QE1</td>
<td>[PE1,PE1]</td>
</tr>
<tr>
<td>PSO2</td>
<td>ED</td>
<td>[E2,E2]</td>
</tr>
<tr>
<td>PSO2</td>
<td>ED ∧¬PE2</td>
<td>[QE2,QE2]</td>
</tr>
<tr>
<td>PSO2</td>
<td>ED ∧¬QE2</td>
<td>[PE2,PE2]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin Role</th>
<th>Prereq Cond</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</td>
<td>(ED,DIR)</td>
</tr>
<tr>
<td>SSO</td>
<td>E</td>
<td>[ED,ED]</td>
</tr>
<tr>
<td>SSO</td>
<td>ED</td>
<td>(ED,DIR)</td>
</tr>
</tbody>
</table>

### URA97 Grant
- Administrator independent
  - A user is revoked from the administrator role, but the assignments are still effective.
- Condition independent
  - If a prerequisite is no longer valid, the previous assignments are still effective.
  - Different from DAC
URA97 REVOKE MODEL

- $can_{\text{revoke}} \subseteq AR \times \mathcal{R}$
  - $AR$: administrative role
  - $\mathcal{R}$: A set of roles (role range)

<table>
<thead>
<tr>
<th>Admin Role</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>[E1,PL1]</td>
</tr>
<tr>
<td>PSO2</td>
<td>[E2,PL2]</td>
</tr>
<tr>
<td>DSO</td>
<td>(ED,DIR)</td>
</tr>
<tr>
<td>SSO</td>
<td>[ED,DIR]</td>
</tr>
</tbody>
</table>

URA97 Revocation

**Definition 5.** Let us say a user $U$ is an explicit member of role $x$ if $(U, x) \in UA$, and that $U$ is an implicit member of role $x$ if for some $x' > x$, $(U, x') \in UA$.

URA97 Revocation

**Weak Revocation**
- revokes explicit membership in a role
- independent of who did the assignment
- Different from the revocation from DAC

**Strong Revocation**
- revokes explicit membership in a role and its seniors
- authorized only if corresponding weak revokes are authorized
- alternatives
  - all-or-nothing
  - revoke within range

ARA97 Revocation

Property 1. Implicit membership in a role $a$ is dependent on explicit membership in some senior role $b > a$. Therefore, when explicit membership of a user is revoked from $b$, implicit membership is also automatically revoked on junior role $a$, unless there is some other senior role $c > a$ in which the user continues to be an explicit member.

ARBAC97 DECENTRALIZES

- user-role assignment (URA97)
- permission-role assignment (PRA97)
- role-role hierarchy
PERMISSION-ROLE ASSIGNMENT

- dual of user-role assignment
- can_assignp \subseteq AR \times CR \times 2^R
- can_revokep \subseteq AR \times 2^R
- weak revoke
- strong revoke (propagates down)

PERMISSION-ROLE ASSIGNMENT

<table>
<thead>
<tr>
<th>ARole</th>
<th>Prereq Cond</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>PL1</td>
<td>[E1,PL1]</td>
</tr>
<tr>
<td>PSO2</td>
<td>PL2</td>
<td>[E2,PL2]</td>
</tr>
<tr>
<td>DSO</td>
<td>E1 ⊕ E2</td>
<td>[ED,ED]</td>
</tr>
<tr>
<td>SSO</td>
<td>PL1 ⊕ PL2</td>
<td>[ED,ED]</td>
</tr>
<tr>
<td>SSO</td>
<td>ED</td>
<td>[E,E]</td>
</tr>
</tbody>
</table>

PERMISSION-ROLE ASSIGNMENT

<table>
<thead>
<tr>
<th>ARole</th>
<th>Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO1</td>
<td>[E1,PL1]</td>
</tr>
<tr>
<td>PSO2</td>
<td>[E2,PL2]</td>
</tr>
<tr>
<td>DSO</td>
<td>(ED,DIR)</td>
</tr>
<tr>
<td>SSO</td>
<td>[ED,DIR]</td>
</tr>
</tbody>
</table>

ARBAC97 DECENTRALIZES

- user-role assignment (URA97)
- permission-role assignment (PRA97)
- role-role assignment (RRA97)

RRA97

- Three types of roles:
  - Group roles:
    - Only have members of users and other groups
    - Use URA97 for role-role assignments
  - Abilities:
    - Only have members of permissions or other abilities
    - PRA97  
  - UP-roles
    - user-and-permission roles
    - RRA97

RRA97

- Four operations
  - Create role
  - Delete role
  - Insert edge
  - Delete edge
can-modify

**Definition 10.** In ERA90, role creation, role deletion, edge insertion, and edge deletion are all authorized by the following relation

\[ \text{can modify} \subseteq AR \times 2^{R} \]

In this context, subsets of \( R \) are identified by the range notation, but limited to open ranges that do not include the endpoints.

Table VI. Example of can modify

<table>
<thead>
<tr>
<th>Admin Role</th>
<th>UP-Role Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSO</td>
<td>(EI, ER)</td>
</tr>
<tr>
<td>PSO1</td>
<td>(EL, PL)</td>
</tr>
<tr>
<td>PSO1</td>
<td>(ER, PLD)</td>
</tr>
</tbody>
</table>

- Authority range must be encapsulated
- To be discussed later

**Range Definitions**

- Range:
  \( R = \{ \text{Roles} | x < r < y \} \)
- Authority Range:
  - A range referenced in can-modify relation
- Partial Overlap of Ranges:
  - The ranges \( Y \) and \( Y' \) partially overlap if
    \( Y \cap Y' \neq \emptyset \) and \( Y \notin Y' \) and \( Y' \notin Y \)
  - Partial Overlap of Authority Ranges is forbidden

**Authority Range**

- Encapsulated Authority Range:
  - The authority range \( (x, y) \) is said to be encapsulated if
    \( \forall r1 \in (x, y) \) and \( \forall r2 \notin (x, y) \)
    - \( r2 > r1 \iff r2 > y \)
    - \( r2 < r1 \iff r2 < x \)

**Non-encapsulated Range**

\( (x, y) \)
ROLE CREATION

- New roles are created one at a time
- Creation of a role requires specification of immediate parent and child
  - These must be within the can-modify range or be one of the endpoints of the range
  - Immediate parent must be senior to immediate child
    - If junior will introduce cycle
    - If incomparable will introduce a new edge (so introduce the new edge first and then create the new role)
  - Immediate parent and immediate child must constitute a create range (prior to creation)
Create Ranges

A is end point of AR\_inaccessible(x)

A is end point of AR\_inaccessible(y)

B is end point of AR\_inaccessible(y)

Create ranges

• (B,A)
• (x,y)

these are not create ranges

Inactivate Roles

• End points of authority ranges can be made inactive.
• Inactive Roles:
  - A user associated to it cannot use it.
  - Inheritance of permissions is not affected.
  - Permissions and users can be revoked.

Semantics of delete role

• End points of authority ranges cannot be deleted.
• Deletion of a role preserves all transitive edges.
• Deletion that causes dangling references is prohibited
  - Prohibit deletion of roles used in can assign, can revoke, can modify OR
  - Deactivate these roles when they are deleted. Inactive roles cannot be activated in a session and new users and permissions cannot be added.
• Preserve permissions and users in a deleted role.
  - Only empty roles can be deleted OR
  - Users pushed down to immediately junior roles and permissions are pushed up to immediately senior roles.

Inactive Roles

• End points of authority ranges can be made inactive.
• Inactive Roles:
  - A user associated to it cannot use it.
  - Inheritance of permissions is not affected.
  - Permissions and users can be revoked.

Preserving encapsulation on edge insertion

• Insertion of (y,z) is ok but will prevent future insertion of (z,x)
• Insertion of (x,y) is ok but will prevent future insertion of (y,z)

Deletion of an edge

• Edges can be deleted only if they are not transitively implied.
• The edges in transitive reduction are candidates for deletion.
• Deleted one at a time.
• Deleting an edge preserves transitive edges.
• Cannot delete an edge between the endpoints of an authority range.

INSERTION OF AN EDGE

• Inserted only between incomparable roles (No Cycles).
• Inserted one at a time.
• Edge insertion must preserve encapsulation of authority ranges.

DELETION OF AN EDGE
Delete Edge
- Delete edge \((r', r'')\)

Example: Before deletion \((SQE1, JQE1)\)

Example: After deletion \((SQE1, JQE1)\)

Conclusion
- ARBAC97 provides decentralized administration of role hierarchies.
- Gives administrative role autonomy within a range but only so far as the side effects of the resulting actions are acceptable.

Other approaches
- ARBAC99
  - Mobile and immobile membership
  - Improves URA97 and PRA97
- ARBAC02
  - Try to solve some problems in URA97 and PRA97
  - Administrative scope:
    - Try to solve problems in RRA97
Contents
- Problems of URA97
- Problems of PRA97
- Solution: ARBAC02 model
- Conclusion

ARBAC97 model
- Main point of decentralized RBAC administration
  - How to control proper administration range (or boundary) of each administrative role

  ARBAC97 model: use role range and prerequisite condition
  - URA97, PRA97

RH and ARH

ARBAC97 model
- Example of can-assign and can-assignp

Problems of URA97
- Characteristics of user-role assignment
  - Security officer SO1 can assign user U1 to role R2 when U1 is already member of prerequisite role R1 that is for SO1.
  - Assigned users in R1 is restricted range (or boundary) to user-role assignment for SO1. (Assigned users in R1 is a user pool for SO1).
  - R2 can be prerequisite role for other security officers.
Problems of URA97

Characteristics of user-role assignment

Consequently users should be assigned from lowest prerequisite role to higher prerequisite role in the role hierarchy.

From can-assign table, we can ratiocinate the first URA step is as follows:

E

ED

E1

E2

: User pool

Problems of URA97

URA97 brings about:

UA1. Multi step assigns

Suppose that new employed engineer 'John' will be assigned to QE1 role.

Assign step:

assign John to E → assign John to ED

→ assign John to E1 → assign John to QE1

The higher role hierarchy may requires the more assign step. It may lead to work of two or more security officers.

Tom

QEI

E1

ED

E

E1-Tom

Tom

E-Tom

EA-

Role

Assigned user

E

Tom

ED

Tom

E1

Tom

QE1

Tom

1

2

3

4

UA table

Problems of URA97

URA97 brings about:

UA2. Duplicated UA information

Suppose that 'Tom' is a member of QE1 role. It means that 'Tom' is a explicit member of 'E', 'ED', 'E1', and 'QE1'. Removing tuple 1, 2, and 3 makes no effect to 'Tom's access rights. They are need only for administrative purpose.

Problems of URA97

URA97 brings about:

UA3. Restricted composition of user pool

Suppose the company in the example wants to maintain human resource pool H1, H2, and H3. And new policy requires that 'Production Engineer' should be picked from H1 and 'Quality Engineer' should be picked from H2.

It is impossible to realize new policy without changing of Role Hierarchy.

Problems of URA97

URA97 brings about:

UA3. Restricted composition of user pool (cont.)

In the URA97 model, composing user pool is based on the prerequisite roles, and prerequisite roles are belongs to role hierarchy. Consequently composing user pool is restricted by role hierarchy. Sometimes real world needs more flexible user pool, and it brings about more complicated Role Hierarchy.

Problems of PRA97

Characteristics of permission-role assignment

Permission-role assignment step is similar to delegation.

The permissions of highest role on the role hierarchy spread down to lower roles by security officer.

Security officer SO1 can assign permission P1 to role R1 when P1 is already member of prerequisite role R2 that is for SO1.

Assigned permissions in R2 is restricted range (or boundary) to permission-role assignment for SO1. (Assigned permissions in R2 is a permission pool for SO1).
Problems of PRA97

- Characteristics of permission-role assignment
  - From can-assignp table, we can obtain the first PRA steps as follows:

  Permission pool:
  - DIR
  - PL1
  - PL2
  - PE1
  - QE1
  - PE2
  - QE2

Problems of PRA97

- PRA97 brings about:
  1. PA1. Multi step assign
  2. PA2. Duplicated PA information
  3. PA3. Restricted composition of permission pool
  Similar to UA1, UA2, and UA3

Problems of PRA97

- PRA97 brings about:
  4. PA4. No restriction for permission pool
  - Suppose there exist can-assignp(SO1, R2, [R1,R1]). Then SO1 can assign in R2's any permissions to R1. There is no restriction. How to specify some of permissions are only for R2? cannot solve in PRA97
  - In PRA99 model, it can be solved by immobile membership concept. But it requires additional information about permission pool.

Problems of PRA97

- PRA97 brings about:
  5. PA5. Lead to undesirable side effect
  - PSO1 can move some permissions of PL1 to QL. But QL is out of range of PSO1.

Solution: ARBAC02

- Direction:
  - Choosing new base for user pool and permission pool
    (role hierarchy independent organization structure)

  - Organization unit is a good container for user pool and permission pool
  - Organization unit: A group of people and functions (permissions) for achieving given missions.
Solution: ARBAC02

Organization structure as a permission pool
- Permissions are pre-assigned to basic organization structure.
  (by IT officer)

Solution: ARBAC02

Modification of prerequisite condition
- Suppose can-assign(PSO1, E1 ∨ QE1, [PE1, PE1])
- It can be redefined by org. unit:
  can-assign'(PSO1, @PJ1 ∨ QE1, [PE1, PE1])
  → "PSO1 can assign users, who are in org. unit PJ1 and not in role QE1, to PE1"

To distinguish role and Org. unit name, we use '@' in the front of Org. unit name

Solution: ARBAC02

Proposed model solves problems UA1 and UA2
- Avoid multi-step user assignment
- Avoid duplicated user assignment information

Solution: ARBAC02

Proposed model solves problems UA3
- Suppose the company in the example want to maintain human resource pool H1, H2, and H3. And new policy requires that "Production Engineer" should be picked from H1 and "Quality Engineer" should be picked from H2.
- In the proposed model, new org. Unit H1, H2, and H3 can be added to proper position of org. structure. Then change prerequisite condition such like:
  can-assign(PSO1, @H1 ∧ QE1, [PE1, PE1])
  → can-assign(PSO1, @H1 ∧ QE1, [PE1, PE1])
- It requires no change of role hierarchy!
Solution: ARBAC02

- Proposed model solves problems PA1~PA4
- Proposed model solves problems PA5
  - In the proposed model, common permissions are assigned to lower roles in the role hierarchy, and higher roles get their special permissions. (bottom-up)
  - These bottom-up style permission-role assignment prevent undesirable side effect in PA5.

DIR
PL1
PL2
PL3
Role Range Of DSO
PE1
QE1
QE2
PE2
PS01 cannot assign PL1's any explicitly assigned permissions to QL

Conclusion

- ARBAC02 overcomes shortcomings of ARBAC97
- ARBAC02 supports flexible user pool and permission pool structure independent from role hierarchy.
  - In the ARBAC97 model, user pool and permission pool are tightly coupled with role hierarchy. It leads to some problems.
- ARBAC02 supports bottom-up oriented permission-role assignment
  - PRA97 model follows top-down approach. It leads to undesirable side effect.

Administration Scope

- For RRA
- Dynamic change of an admin range