Role Based Access Control

- Role-Based Access Control Models.
- The most cited paper in access control!

RBAC96 Family of Models

- RBAC0: BASIC RBAC
- RBAC1: Role Hierarchies
- RBAC2: Constraints
- RBAC3: Role Hierarchies + Constraints
RBAC0: Formal Model

- U, R, P, S (users, roles, permissions, and sessions)
- PA \subseteq P \times R (permission assignment)
- UA \subseteq U \times R (user assignment)
- user: S \rightarrow U
- roles: S \rightarrow 2^R
  - requires roles(s) \subseteq \{ r | (user(s), r) \in UA \}
- Session s has permissions
  \neg \bigcup_{r \in roles(s)} \{ p | (p, r) \in PA \}
Why RBAC

- Fewer relationships to manage
- from $O(mn)$ to $O(m+n)$, where $m$ is the number of users and $n$ is the number of permissions
- Roles add a useful level of indirection
- RBAC? DAC? MAC?

RBAC1: RBAC0+ Role Hierarchies

Primary-Care Physician

Specialist Physician

Physician

Health-Care Provider
RBAC1: Formal Model

- $U, R, P, S, PA, UA,$ and users unchanged from RBAC0
- $RH \subseteq R \times R$: a partial order on $R$, written as $\geq$; $r_1 \geq r_2$ means that all privileges of $r_2$ are also of $r_1$ and all users of $r_1$ are also of $r_2$.
- roles: $S \rightarrow 2^R$
  requires roles $(s) \subseteq$
  \[
  \{ r | \exists r' [(r' \geq r) \& (\text{user}(s), r') \in UA] \}
  \]
  $u$ is said to be authorized for role $r$, if $u \rightarrow^+ r$

- Session $s$ has permissions
  \[
  \bigcup_{r \in \text{roles}(s)} \{ p | \exists r'' [(r \geq r'') \& (p, r'') \in PA] \}
  \]

Inheritance Relation

- $\rightarrow$ defines both the permission inheritance and membership inheritance.
- The roles toward top of the hierarchy represents more powerful roles, and the roles toward the bottom represent the more general rules.
- $r_1 \rightarrow r_2$; iff all permissions of $r_2$ are also permissions of $r_2$ and all users of $r_1$ are also users of $r_2$.
- By $\rightarrow^*$, the reflexive-transitive closure of the inheritance relation ($r_1 \rightarrow^* r_2$ iff $r_1 = q_1 \ldots q_n = r_2$) is denoted.
• RH ⊆ ROLES × ROLES is a partial order on
  ROLES, where r₁ ≥ r₂ ⇒ authorized_permissions(r₂) ⊆
  authorized_permissions (r₁) ∧ authorized_users(r₁) ⊆
  authorized_users(r₂)

• authorized_users(r: ROLES) → 2^USERS,
  authorized_users(r) = \{ u ∈ USERS | r’ ≥ r ∧ (u, r’) ∈ UA \}

• Is this correct??
• authorized-permissions (r: ROLES) → 2^PRMS
  authorized-permissions (r) = \{ p ∈ PRMS | r’ ≥ r ∧ (p, r’) ∈ PA \}

Semantics of Role Hierarchies

• User inheritance
  – r₁ ≥ r₂ means every user that is a
    member of r₁ is also a member of
    r₂

• Permission inheritance
  – r₁ ≥ r₂ means every permission
    that is authorized for r₂ is also
    authorized for r₁

• Activation inheritance
  – r₁ ≥ r₂ means that activating r₁ will
    also activate r₂
RBAC2: RBAC0 + Constraints

- No formal model specified
- A list of examples are given

Static Mutual Exclusion Constraints

- Two mutually exclusive roles: cannot both have the same user as members
- Two mutually exclusive roles: cannot both have the same permissions
  - why?
- Two mutually exclusive permissions: one role cannot have both permissions
  - why?
Cardinality Constraints

• On User-Role Assignment
  – at most $k$ users can belong to the role
  – at least $k$ users must belong to the role
  – exactly $k$ users must belong to the role

• On activation
  – at most $k$ users can activate a role
  – …

Why Using Constraints?

• For laying out higher level organization policy
  – simply a convenience when admin is centralized
  – a tool to enforce high-level policies when admin is decentralized
RBAC3

- RBAC3 = 
  RBAC0 + Role Hierarchies + Constraints

Some Issues in RBAC

- 1- Whether to Allow Multiple Roles to be Activated?
- RBAC96 allows it
- Some other does not allow!
- Observations:
  - one can define new role to achieve the effect of activating multiple roles
  - dynamic constraints are implicit when only one role can be activated in a session
2- What is a Role?

- A set of users
- A set of permissions (named protection domains)
- A set of users and permissions
- Also affects how to interpret role hierarchies
- Maybe it is useful to have both roles and groups?

Roles vs. Groups

- What are the differences?
  - Answer 1: groups are sets of users, and roles are sets of users as well as permissions
    - doesn’t seem to be true.
  - Answer 2: one can activate and deactivate roles, but cannot deactivate groups
    - seems unimportant unless there is negative authorization
  - Answer 3: one can enumerate permissions that a role has
    - seems an implementation issue
Everything as an attribute?

- Some attributes are more intrinsic about properties of a user
- Some attributes are more intrinsic about job functionalities

The NIST Standard

- ANSI Standard
Overview of the NIST Standard for RBAC

Core RBAC (1)

- USERS
- ROLES
- OBS
- OPS
- \( PRMS = 2^{(OPS \times OBS)} \)
  - \( Op : (p : PRMS) \rightarrow 2^{OPS} \)
  - \( Ob : (p : PRMS) \rightarrow 2^{OBS} \)
Core RBAC (2)

- $UA \subseteq \text{USERS} \times \text{ROLES}$
  - $\text{assigned\_users} : (r : \text{Roles}) \rightarrow 2^{\text{USERS}}$
- $PA \subseteq \text{PRMS} \times \text{ROLES}$
  - $\text{assigned\_permissions} : (r : \text{Roles}) \rightarrow 2^{\text{PRMS}}$

Core RBAC (3)

- $\text{SESSIONS}$
- $\text{session\_users} : (s : \text{SESSIONS}) \rightarrow \text{USERS}$
  - $\text{user\_sessions} : (u : \text{USERS}) \rightarrow 2^{\text{SESSIONS}}$
- $\text{session\_roles} : (s : \text{SESSIONS}) \rightarrow 2^{\text{ROLES}}$
  - $\text{avail\_session\_perms} : (s : \text{SESSIONS}) \rightarrow 2^{\text{PRMS}}$
Hierarchical RBAC: Generalized Role Hierarchies

- $RH \subseteq ROLES \times ROLES$
  - user inheritance & permission inheritance
  - we say $r_1$ inherits $r_2$ if $r_1 \geq r_2$
- $authorized\_users : (r : Roles) \rightarrow 2^{\text{USERS}}$
- $authorized\_permissions : (r : Roles) \rightarrow 2^{\text{PRMS}}$

Hierarchical RBAC: Limited Role Hierarchies

- Role Hierarchies with the limitation that each role has at most one immediate senior
- Role hierarchies form a forest
Constrained RBAC: Motivations

Example of SoD
- The following duties shall be performed by different individuals:
  1. Check request reviewer
  2. Check preparer
  3. Check issuer
  4. Check deliverer
  5. Ledger reviewer

Constrained RBAC: Static SoD

SSD $\subseteq (2^\text{ROLES} \times \text{N})$ is a collection of pairs (rs, n)
- rs: a role set
- n: $n \geq 2$ is a natural number

For each (rs, n), no user is authorized for n or more roles in rs
SoD with Role Hierarchies

- Two roles can be mutually exclusive only if neither one inherits the other
- If two roles are mutually exclusive, no role can inherit from both
- If two roles are mutually exclusive, there can be no “root” or “super user”.

Constrained RBAC: Dynamic SoD

- $\text{DSD} \subseteq (2^{\text{ROLES}} \times \mathbb{N})$ is a collection of pairs $(rs, n)$
  - $rs$: a role set
  - $n$: $n \geq 2$ is a natural number
- For each $(rs, n)$, no user is allowed to activate $n$ or more roles in $rs$ in one session
Constraints in Role Based Access Control

SoD

- If a sensitive task comprises two steps, then two different users should perform each step.

- e.g. the same user cannot order goods, and authorize payment for those goods.

- SoD is a security principle that is generally considered to be useful.
SoD (contd.)

- More elaborate example:
  (a) Order goods and record details of order
  (b) Receive invoice and check against order
  (c) Receive goods and check against invoice
  (d) Authorize payment against invoice

- A set of SoD requirements:
  (1) No user performs (a) and (d).
  (2) At least 3 users to perform all 4 steps.

Enforcement of SoD

- Static enforcement
  - the permissions to perform two steps are not assigned to a single user

- Dynamic enforcement
  - remember which user performed each step, and don’t allow a user to perform the next step if violating SoD policy
SSoD Safety

- An RBAC state is given by <UA, PA, RH>
- Definition: An RBAC state $\gamma$ is safe wrt. $\text{ssod}(\{p_1, \ldots, p_n\}, k)$ iff in $\gamma$, no k-1 users together have all permissions in $\{p_1, \ldots, p_n\}$.
- Definition: An RBAC state $\gamma$ is safe wrt. a set $E$ of SSoD policies iff $\gamma$ is safe wrt. each $e$ in $E$.
- Definition: The SCSSoD (Safety Checking problem for SSoD) problem is to determine whether an RBAC state is safe wrt. a set $E$ of SSoD policies.
SMER Constraints

- Statically mutually-exclusive role (SMER) constraints: \( \text{smer}(\{r_1, \ldots, r_m\}, t) \)
  - means that no user can be a member of \( t \) roles from \( \{r_1, \ldots, r_m\} \)
  - \( \text{smer}(\{r_1, r_2\}, 2) \) means that \( r_1 \) and \( r_2 \) are mutually exclusive, i.e., no user can be a member of both roles

- Example:
  - \( C = \{c_1, c_2, c_3\} \), where:
    - \( c_1 = \text{smer}(\{\text{Warehouse, Accounting, Finance}\}, 2) \)
    - \( c_2 = \text{smer}(\{\text{Engineering, Finance}\}, 2) \)
    - \( c_3 = \text{smer}(\{\text{Quality, Finance}\}, 2) \)

The ARBAC97 model for role-based administration of roles

- Goal: Decentralize the administration of RBAC, i.e., allowing others to change parts of (UA, PA, RH)

- Overview
  - there exist a set of administrative roles (in the RBAC terminology) that are disjoint from the regular roles
ARBAC97

- 3 components:
  - URA97 (User-Role-Assignment)
  - PRA97 (Permission-Role-Assignment)
  - RRA97 (Role-Role Assignment)
The URA97 Component

- Prerequisite condition
  - e.g., \( r_1 \lor (r_2 \land \neg r_3) \) is such a condition
- can_assign
  - e.g., can_assign \((a, \text{cond}, \{r_4,r_5,r_6\})\), \(a\) is an administrative role:: all users in the admin role can assign users who pass the condition \text{cond} into the role set.
- can_revoke
  - e.g., can_revoke \((a, \{r_4,r_5\})\)
  - weak revocation vs. strong revocation

PRA97

- Treat permission assignment as dual to user assignment
- can_assign
  - e.g., can_assign \((a, \text{cond}, \{r_4,r_5,r_6\})\)
- can_revoke
  - e.g., can_revoke \((a, \{r_4,r_5\})\)
RRA