ECS 235B Module 31
Role-Based Access Control
Role-Based Access Control

• Access depends on function, not identity
  • Example:
    • Allison, bookkeeper for Math Dept, has access to financial records.
    • She leaves.
    • Betty hired as the new bookkeeper, so she now has access to those records
  • The role of “bookkeeper” dictates access, not the identity of the individual.
Definitions

- Role $r$: collection of job functions
  - $\text{trans}(r)$: set of authorized transactions for $r$

- Active role of subject $s$: role $s$ is currently in
  - $\text{actr}(s)$

- Authorized roles of a subject $s$: set of roles $s$ is authorized to assume
  - $\text{authr}(s)$

- $\text{canexec}(s, t)$ iff subject $s$ can execute transaction $t$ at current time
Axioms

Let $S$ be the set of subjects and $T$ the set of transactions.

- **Rule of role assignment**: $(\forall s \in S)(\forall t \in T) [canexec(s, t) \rightarrow actr(s) \neq \emptyset]$.
  - If $s$ can execute a transaction, it has a role
  - This ties transactions to roles

- **Rule of role authorization**: $(\forall s \in S) [actr(s) \subseteq authr(s)]$.
  - Subject must be authorized to assume an active role (otherwise, any subject could assume any role)
Axiom

• Rule of transaction authorization:

\[ (\forall s \in S)(\forall t \in T) \ [canexec(s, t) \rightarrow t \in trans(actr(s))] \].

• If a subject \( s \) can execute a transaction, then the transaction is an authorized one for the role \( s \) has assumed
Containment of Roles

• Trainer can do all transactions that trainee can do (and then some). This means role $r$ contains role $r' (r > r')$. So:

$$\forall s \in S \left[ r \in authr(s) \land r > r' \rightarrow r' \in authr(s) \right]$$
Separation of Duty

• Let $r$ be a role, and let $s$ be a subject such that $r \in auth(s)$. Then the predicate $meauth(r)$ (for mutually exclusive authorizations) is the set of roles that $s$ cannot assume because of the separation of duty requirement.

• Separation of duty:

$$(\forall r_1, r_2 \in R) \ [ r_2 \in meauth(r_1) \rightarrow [ (\forall s \in S) \ [ r_1 \in auth_r(s) \rightarrow r_2 \notin auth_r(s) ] ] ]$$
RBAC Hierarchy

• RBAC\(_0\): basic model (you just saw it)
• RBAC\(_1\): adds role hierarchies to RBAC\(_0\)
• RBAC\(_2\): adds constraints to RBAC\(_0\)
• RBAC\(_3\): adds both role hierarchies, constraints to RBAC\(_0\)
  • It combines RBAC\(_1\) and RBAC\(_2\)
RBAC$_0$, Formally

- Set of users $U$, roles $R$, permissions $P$, sessions $S$
- Relation $PA \subseteq P \times R$ mapping permissions to roles
- Relation $UA \subseteq U \times R$ mapping users to roles
- Function $user: S \rightarrow U$ mapping each session to a user
- Function $roles: S \rightarrow 2^R$ mapping each session $s \in S$ to a set of roles $roles(s) \subseteq \{ r \in R \mid (user(s), r) \in UA \}$, where $s$ has permissions
  \[ \bigcup_{r \in \text{roles}(s)} \{ p \in P \mid (p, r) \in PA \} \]
- When a user assumes role $r$ during session, $r$ and hence the user assuming $r$ gets the set of permissions associated with $r$
RBAC$_1$, Intuitively

- Add containment of roles to RBAC$_0$ (this is the hierarchy)
  - It’s a partial ordering
- Each role less powerful than its containing role
  - Containing role contains job functions (permissions) of the contained role
- Can define *private roles* in which one role is subordinate to two others, and those two are not related
RBAC\textsubscript{1}, Formally

- Set of users $U$, roles $R$, permissions $P$, sessions $S$
- Partial order $RH \subseteq R \times R$
  - Write $(r_1, r_2) \in R$ as $r_1 \geq r_2$
- Relation $PA \subseteq P \times R$ mapping permissions to roles
- Relation $UA \subseteq U \times R$ mapping users to roles
- Function $user: S \rightarrow U$ mapping each session to a user
- Function $roles: S \rightarrow 2^R$ mapping each session $s \in S$ to a set of roles $roles(s) \subseteq \{ r \in R \mid (\exists r' \geq r)(user(s), r') \in UA \}$, where $s$ has permissions
  \[ \bigcup_{r \in \text{roles}(s)} \{ p \in P \mid (\exists r'' \geq r')(p, r'') \in PA \} \]
  - When a user assumes role $r$ with subordinate role $r'$ during session, $r$ and hence the user assuming $r$ gets the set of permissions associated with $r$, and hence with $r'$
RBAC$_2$ and RBAC$_3$

- RBAC$_2$ adds constraints on values that components can assume to RBAC$_0$
  - Example: user can be in only one role at a time
  - Example: make 2 roles mutually exclusive

- RBAC$_3$ provides both role hierarchies and constraints that determine allowable values for relations and functions
  - Combines RBAC$_1$ and RBAC$_2$

- Can be extended to manage role and privilege assignments
  - A set of administrative roles $AR$ and permissions $AP$ defined disjointly from $R$ and $P$
  - Constraints allow $ap \in AP$ to be assigned to $ar \in AR$ only, and $p \in P$ to $r \in R$ only
Role Engineering

• *Role engineering*: defining roles and determining needed permissions

• Often used when two organizations using RBAC merge
  • Roles in one organization rarely overlap with roles in other
  • Job functions often do overlap

• *Role mining*: analyzing existing roles, permission assignments to determine optimal assignment of permissions to roles
  • *NP*-complete, but in practice optimal solutions can be approximated or produced