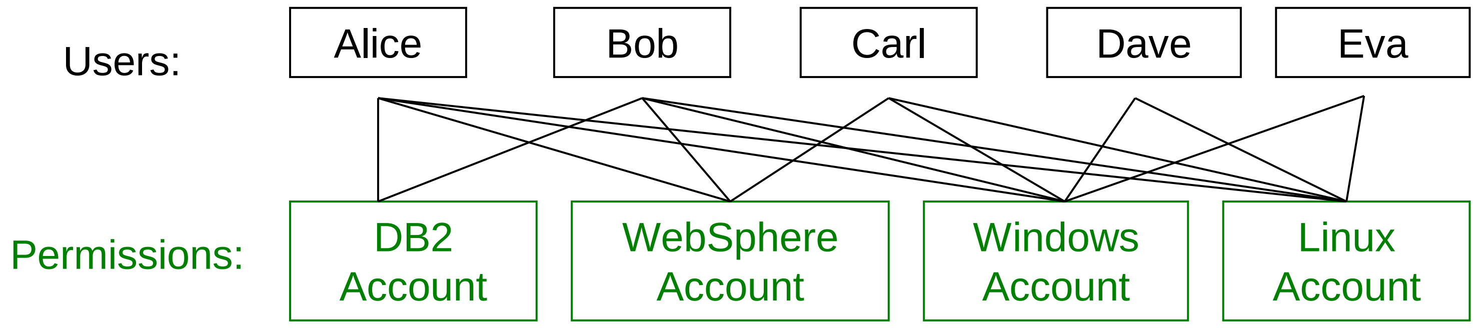
RBAC: Motivations

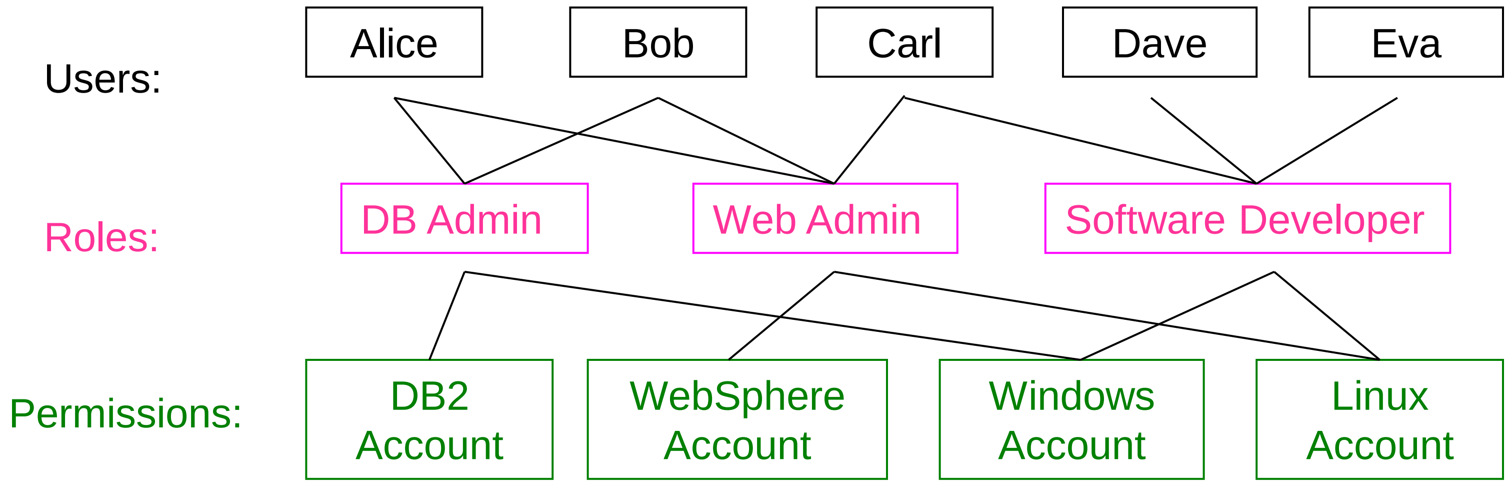
- One challenging problem in managing large systems is the complexity of security administration
- Whenever the number of subjects and objects is high, the number of authorizations can become extremely large
- Moreover, if the user population is highly dynamic, the number of grant and revoke operations to be performed can become very difficult to manage

Background: Role Based Access Control

- Non-role-based systems



- Role-Based Access Control Systems (RBAC)



Reference: CS526 Purdue University

RBAC: Motivations

- End users often do not own the information for which they are allowed access. The corporation or agency is the actual owner of data objects
- Control is often based on employee functions rather than data ownership
- RBAC has been proposed as an *alternative* approach to DAC and MAC both to simplify the task of access control management and to directly support function-based access control

RBAC: Basic Concepts

- Roles represent functions within a given organization and authorizations are granted to roles instead of to single users
- Users are thus simply authorized to "play" the appropriate roles, thereby acquiring the roles’ authorizations

RBAC: Basic Concepts

- 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.

RBAC: Benefits

- Because roles represent organizational functions, an RBAC model can directly support security policies of the organization
- Granting and revoking of user authorizations is greatly simplified
- RBAC models have been shown to be policy-neutral

RBAC

- DBMS vendors have recognized the importance and the advantages of RBAC, and today most of the commercial DBMSs support RBAC features at some extents
- There is some consensus on a standard RBAC model
- The NIST model [Sandhu,Ferraiolo,Kuhn 00] has been the first step towards the definition of a standard
- A recent definition is by ANSI. American national standard for information technology – role based access control. ANSI INCITS 359-2004, February 2004

RBAC - Definitions

- Role *r*: collection of job functions
  - *trans*(*r*): set of authorized transactions for *r*
- Active role of subject *s*: role *s* is currently in
  - *actr*(*s*)
- Authorized roles of a subject *s*: set of roles *s* is authorized to assume
  - *authr*(*s*)
- *canexec*(*s*, *t*) iff subject *s* can execute transaction *t* at current time

NIST Model

- Three main levels of increasing functional capabilities
  - Core RBAC – also called Flat RBAC
  - Hierarchical RBAC
  - Constrained RBAC

RBAC- Basic concepts

- *User* – is defined as a human being, a machine, a process, or an intelligent autonomous agent, etc.
- *Role* – is a function within the context of an organization with an associated semantics regarding its authority and responsibility

RBAC- Basic concepts

- *Permission* – is an access mode that can be exercised on objects in the system. Both objects and access modes are domain dependent. For example, in the case of databases, the object set includes tables, columns, and rows, and the access mode set includes insert, delete, and update operations.

RBAC- Basic concepts

- *Session* – it is a particular instance of a connection of a user to the system and defines the subset of activated roles. At each time instant, different sessions for the same user can be active. When a user logs in the system, he/she establishes a session and, during this session, can request to activate a subset of the roles he/she is authorized to play. The user obtains all permissions associated with the role he/she has activated in the session

Core RBAC - Permissions

- Let S be the set of subjects and T the set of transactions.
- *Rule of role assignment*: $(\forall s \in S)(\forall t \in T)\ [canexec(s, t) \rightarrow actr(s) \neq \varnothing]$.
  - 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)

- *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

Core RBAC
Sets, Functions, and Relations

- *USERS*, *ROLES*, *OPS*, and *OBS*
- $UA \subseteq USERS \times ROLES$, a many-to-many mapping user-to-role assignment relation
  - *assigned_users*: (*r*: *ROLES*), the mapping of a role *r* onto a set of users. Formally:
    - $assigned\_users(r) = \{u \in USERS \mid (u, r) \in UA\}$
- *PRMS* , the set of permissions
- $PA \subseteq PRMS \times ROLES$, a many-to-many mapping permission-to-role assignment relation
  - *assigned_permissions*: (*r*: *ROLES*), the mapping of a role *r* onto a set of permissions. Formally:
    - $assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$

Core RBAC
Sets, Functions, and Relations

- *SESSIONS* = the set of sessions
  - *session_users* (*s* : *SESSIONS*) $\longrightarrow$ *USERS*, the mapping of session *s* onto the corresponding user
  - *session_roles* (*s* : *SESSIONS*), the mapping of session *s* onto

  a set of roles. Formally:
  - *session_roles* (*s*) = {*r* ∈ *ROLES* | (*session_users* (*s*), *r*) ∈ *UA*}
- *avail_session_perms* (*s* : *SESSIONS*), the permissions available to a user in a session. Formally:
  - *avail_session_perms* (*s*) =

$$\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$$

Core RBAC - Sessions

- The notion of session is quite abstract – it is defined as "*a mapping between a user and an activated subset of roles that are assigned to the user*"
- Basic distinction:
  - *Single-role activation (SRA)* Only one role can be activated
  - *Multi-role activation (MRA)* Multiple roles can be activated in one session, and dynamic separation of duty constraints may be used to restrict concurrent activation of some roles
  - There are trade-offs between the use of these two types of session

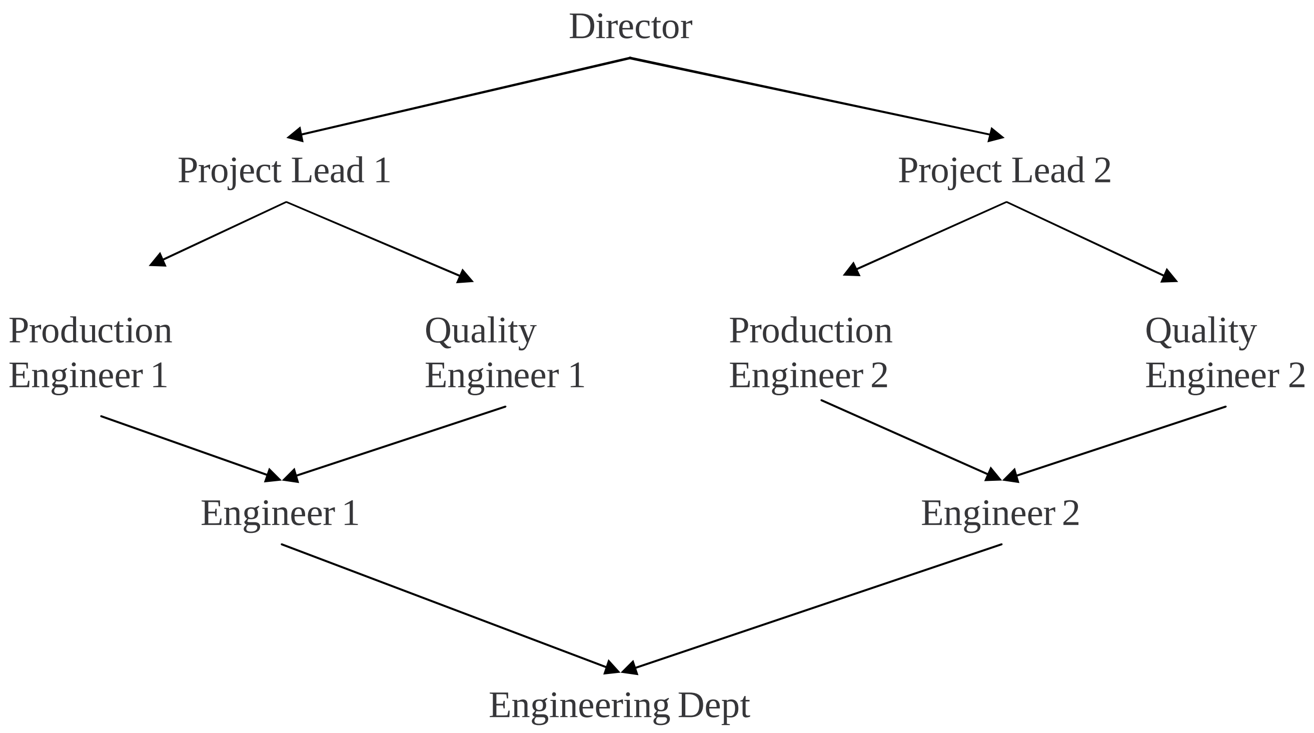
Hierarchical RBAC - Motivations

- Role hierarchies are a natural means for structuring roles to reflect an organization’s line of authority and responsibility

Hierarchical RBAC - Model

- *RH* $\subseteq$ *ROLES* $\times$ *ROLES* it is a relation defined on the set of roles in the system
- It has to be irreflexive. We refer to this relation as *dominance relation;* if $(r_i, r_j) \in RH$ we say that r_i dominates r_j
- We also define a partial order $\geq$ which is the reflexive and transitive closure of *RH*.
- An RBAC system may choose to store **≥** or to compute it when needed

Example of RH



Hierarchical RBAC - Semantics

- User Inheritance (*UI*): All users authorized for a role *r* are also authorized for any role *r'* where *r* ≥ *r'*
- Permission Inheritance (*PI*): A role *r* is authorized for all permissions for which any role *r'*, such that *r* ≥ *r'*, is authorized
- Activation Inheritance (*AI*): Activating a role *r* automatically activates all roles *r'*, such that *r* ≥ *r'*. This semantics can be used only if MRA sessions are used

Constrained RBAC

Example constraints

- Mutual exclusion: capability of supporting *Separation of Duties* policies
  - Two main categories:
    - Static SoD
    - Dynamic SoD
- Pre-condition: Must satisfy some condition to be member of some role
  - E.g., a user must be an undergrad student before being assigned the UTA role
- Cardinality

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

Mutual Exclusion Constraints

Mutually Exclusive Roles

- Static Exclusion: No user can hold both roles
  - often referred to as Static Separation of Duty constraints
  - Preventing a single user from having too much permissions
- Dynamic Exclusion: No user can activate both roles in one session
  - Often referred to as Dynamic Separation of Duty constraints
  - Interact with role hierarchy interpretation

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:

$$(\forall r_1, r_2 \in \mathrm{R})\ [\ r_2 \in meauth(r_1) \rightarrow$$
$$[\ (\forall s \in \mathrm{S})\ [\ r_1 \in authr(s) \rightarrow r_2 \notin authr(s)\ ]\ ]\ ]$$

RBAC - SoD Policies

- Enforces conflict of interest policies employed to prevent users from exceeding a reasonable level of authority for their position.
- Ensures that failures of omission or commission within an organization can be caused only as a result of collusion among individuals.

SoD Definitions

- ANSI: “Dividing responsibility for sensitive information so that no individual acting alone can compromise the security of the data processing system”
- The U.S. Office of Management and Budget’s Circular A-123: “Key duties and responsibilities in authorizing, processing, recording, and reviewing official agency transactions should be separated among individuals”.

RBAC – Static SoD Constraints

- SSoD places restrictions on the set of roles and in particular on their ability to form *RA* relations.
- No user is assigned to *t* or more roles in a set of *m*
  roles
- Prevents a person being authorized to use too many roles
- These constraints can be enforced based on the users assigned to each role

RBAC – Dynamic SoD Constraints

- These constraints limit the number of roles a user can activate in a single session
- Examples of constraints:
  - No user may activate *t* or more roles from the roles set in each user session.
  - If a user has used role *r*1 in a session, he/she cannot use role *r*2 in the same session
- Enforcement of these roles requires keeping the history of the user access to roles within a session