Hybrid Policies

- Chinese Wall Model
  - Focuses on conflict of interest
  - Combines integrity and confidentiality
- RBAC
  - Base controls on job function

1

Chinese Wall Model

- Introduced by Brewer-Nash in 1989

Problem:

- Consultant advises Bank1 and Bank2 about investments
- Conflict of interest: advice for either bank would affect advice to the other bank
- Solution
  - Consultant can only access objects on his/her side of the wall
- Organization
  - Organize entities into "conflict of interest" classes
  - Control read accesses based on COI and access history
  - Control writing to all classes to ensure information is not passed along in violation of rules
  - No control over sanitized data (no conflict)

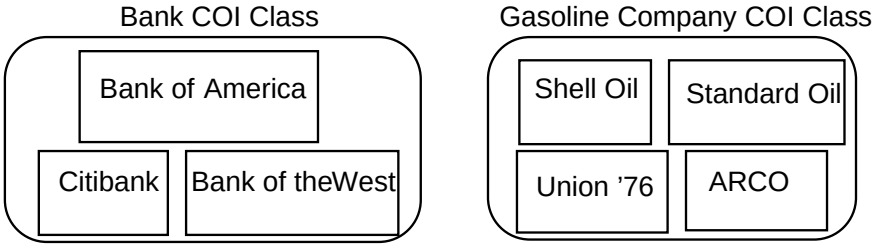
2

Definitions

- *Objects* : items of information related to a company
- *Company dataset* (CD): collection of objects related to a single company
  - Written *CD*(o)
- *Conflict of interest class* (COI): collection of datasets of companies in competition
  - Written *COI*(o)
- Assumption: each object belongs to exactly one *CD* and each CD to one *COI* class

3

Example



4

Temporal Element

- Rights depend on access history
- Initially, a subject can read any object in any CD of any COI
- If a subject reads an object in a CD in a COI, he can *never* read an object in another CD in the same COI
  - Possible that information learned earlier may allow him to make decisions later
- *PR*(*s*) denotes the set of objects that a subject *s* has already read

5

Sanitization

- Public information may belong to a CD
  - As is publicly available, no conflicts of interest arise
  - So, should not affect ability of subject to read
  - Typically, all sensitive data removed from such information before it is released publicly (called *sanitization*)

6

CW-Simple Security Condition

s can read *o* iff any of these conditions holds:

1. There is an o' such that $o' \in PR(s)$ and $CD(o') = CD(o)$
   - Meaning *s* has read something else in *o*'s dataset
2. For all $o' \in O$, $o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$
   - Meaning *s* has not read any objects in $COI(o)$
3. *o* is a sanitized object

Initially, $PR(s) = \varnothing$, so any initial read request is granted

7

What about writing

- Alice and Bob work in same trading house
- Alice can read objects in Citibank's CD and in Shell's CD
- Bob can read objects in Bank of America's CD and in Shell's CD
- If Alice could write (information from Citibank's objects) to objects in Shell's CD, then Bob can read it
  - Hence, indirectly, he can read information from Citibank's CD, a clear conflict of interest

8

CW-*-Property

s can write to *o* if and only if :

1. The CW-simple security condition permits *s* to read *o*
   - No blind writes as in BLP

   and
2. For all *unsanitized* objects *o′*, if *s* can read *o′*, then *CD*(*o′*) = *CD*(*o*)
   - Says that s can write to an object if all the objects it can read are in the same dataset or sanitized

9

Compare to Bell-LaPadula

- Fundamentally different
  - ChW has no security labels, BLP does
  - ChW has notion of past accesses, BLP does not
- BLP can capture state at any time, but cannot track changes over time
  - Each (COI, CD) pair gets security category
  - Two clearances, *S* (sanitized) and *U* (unsanitized)
    - *U dom S*
  - Subjects assigned clearance for compartments that do not have categories corresponding to CDs in the same COI class

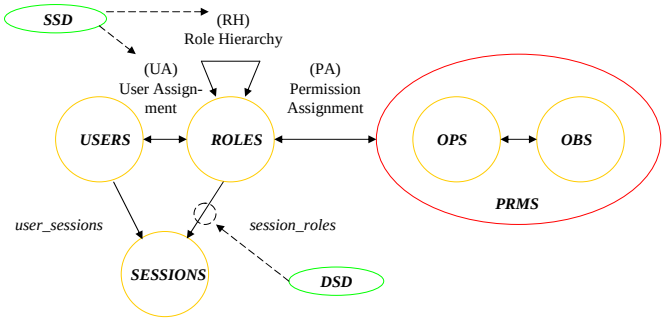
10

Summary of Chinese Wall

- The Chinese Wall policy is just another lattice-based information flow policy
- To properly understand and enforce Information Security policies we must distinguish between
  - policy applied to users, and
  - policy applied to principals and subjects

11

Role-Based Access Control



RBAC (http://csrc.nist.gov/rbac/)

- A policy-neutral model, that can express both DAC (role as identity) and MAC (role as clearance)
- Access/right often depends on role (job function), not on identity
  - Example:
    - Allison, bookkeeper, has access to financial records.
    - Bob hired to replace Allison as the new bookkeeper
    - Bob now has access automatically to those records
  - The role of "bookkeeper" determines access, not the identity of the individual, and 'connects' the subject to the permission(s).

13

Role-Based AC

- A user has access to an object based on the assigned role.
- Roles are defined based on job functions.
- Permissions are defined based on job authority and responsibilities within a job function.
- Operations on an object are invocated based on the permissions.
- The object is concerned with the user's role and not the user.

14

Privilege

- Roles are engineered based on the principle of least privilege.
- A role contains the minimum amount of permissions to instantiate an object.
- A user is assigned to a role that allows him or her to perform only what is required for that role.
- No single role is given more permission than the same role for another user.

15

Role-Based AC Framework

- Core Components
- Constraining Components
  - Hierarchical RBAC
    - General
    - Limited
  - Separation of Duty Relations
    - Static
    - Dynamic

16

Core Components

- Defines:
  - USERS {process, intelligent agent, human}
  - ROLES
  - OPERATIONS (*ops*)
  - OBJECTS (*obs*)
  - User Assignments (*ua*)
    - assigned_users

17

Core Components (cont)

- Permissions (*prms*)
  - Assigned Permissions
  - Object Permissions
  - Operation Permissions
- Sessions
  - User Sessions
  - Available Session Permissions
  - Session Roles

18

Constraint Components

- Role Hierarchies (*rh*)
  - General
  - Limited
- Separation of Duties
  - Static
  - Dynamic

19

RBAC Transition

Least Privilege
Separation of
Duties

Most
Complex

Models	Hierarchies	Constraints
$RBAC_0$	No	No
$RBAC_1$	Yes	No
$RBAC_2$	No	Yes
$RBAC_3$	Yes	Yes

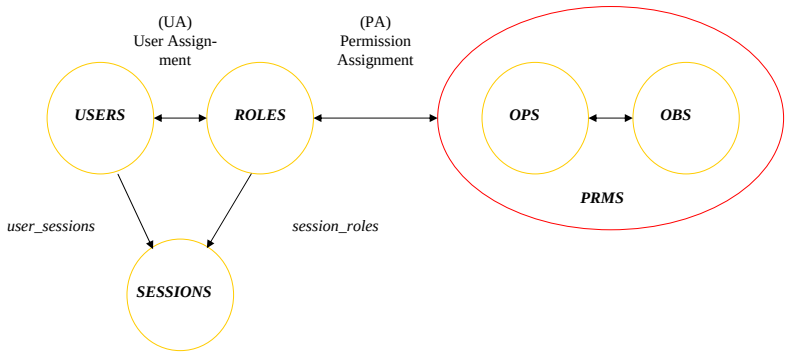
20

RBAC Functional Specification

- It defines the features required of an RBAC system.
- These features fall into three categories
  - Administrative Operations
    - Administrative operations define requirements in terms of an administrative interfaces and an associated set of semantics that provide the capability to create, delete and maintain RBAC elements and relations.
  - Administrative Reviews
    - The administrative review features define requirements in terms of an administrative interfaces and an associated set of semantics that provide the capability to perform query operations on RBAC elements and relations.
  - System level functionality
    - The System level functionality defines features for the creations of user sessions to include role activation/deactivation, the enforcement of constraints on role activation, and for calculation of an access decision.

21

Core RBAC



22

Definitions

- Role *r* : collection of job functions
  - developer, director, manager, …
  - an organizational job function with a clear definition of inherent responsibility and authority (permissions).
- M-T-M relation between USERS and PRMS (going through roles)

23

Definitions

- Role *r*
  - *trans*(*r*): set of authorized transactions for *r*
- Active role of subject *s* : the role *s* is currently in
  - *actr*(*s*)
- Authorized roles of *s* : set of roles *s* can assume
  - *authr*(*s*)
- *canexec*(*s*, *t*) is true if and only if subject *s* can execute transaction *t* at current time

24

Axioms (mandatory style)

S the set of subjects; 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, not users

- ***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)

25

UA (user assignment)

A user can be assigned to one or more roles

A role can be assigned to one or more users

$$UA \subseteq USERS x ROLES$$

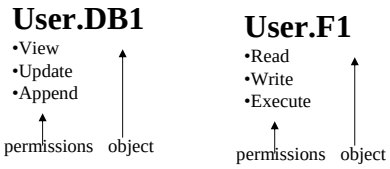
$$assigned\_user : (r : ROLES) \rightarrow 2^{users}$$

$$assigned\_user(r) = \{u \in USERS \mid (u, r) \in UA\}$$

26

PRMS (permissions)

The set of permissions that each grant the approval to perform an operation on a protected object.



$$PRMS = 2^{(OPSxOBS)}$$

27

PA (prms assignment)

- A prms can be assigned to one or more roles
- A role can be assigned to one or more prms

$$PA \subseteq PRMSxROLES$$

$$assigned\_permissions(r : ROLES) \rightarrow 2^{PRMS}$$

$$assigned\_permissions(r) = \{p \in PRMS \mid (p, r) \in PA\}$$

$$Op(p : PRMS) \rightarrow \{op \subseteq OPS\}$$

$$Ob(p : PRMS) \rightarrow \{ob \subseteq OBS\}$$

28

SESSIONS

Each session is associated to a number of roles and each user u is associated to a set of sessions.

$$session\_roles(s: SESSIONS) \rightarrow 2^{ROLES}$$

$$session\_roles(s_i) \subseteq \{r \in ROLES \mid (session\_users(s_i), r) \in UA)\}$$

$$user\_sessions(u: USERS) \rightarrow 2^{SESSIONS}$$

$$avail\_session\_persm(s: SESSIONS) \rightarrow 2^{PRMS}$$

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

29

Axiom

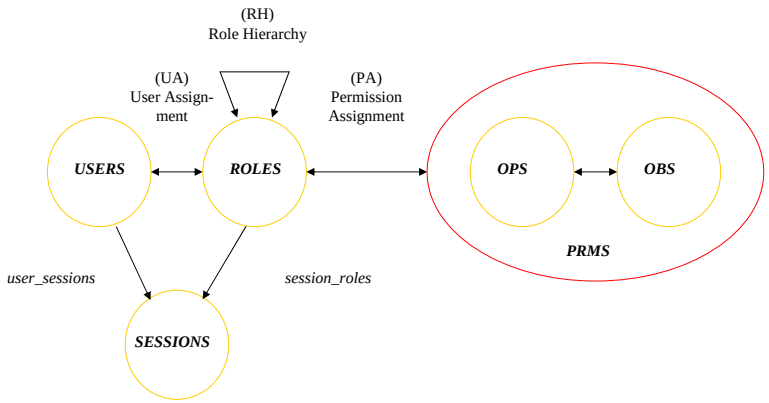
- *Rule of transaction authorization*:

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

  - A subject *s* can execute a transaction only if the transaction is authorized one for the role *s* has assumed (active)

30

Hierarchical RBAC



31

RH (Role Hierarchies)

- Natural means of structuring roles to reflect organizational lines of authority and responsibilities
- General and Limited
- Define the inheritance relation among roles

i.e. r_1 *inherits* r_2

User
r-w-h

Guest
-r-

$$RH \subseteq ROLES x ROLES$$

32

Containment of Roles

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

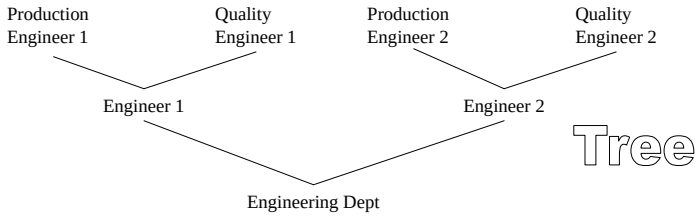
  $(\forall s \in S)[\ r' \in authr(s) \wedge r' > r \rightarrow r \in authr(s)\ ]$

  $(\forall t \in T)[\ t \in trans(r) \wedge r' > r \rightarrow t \in trans(r')\ ]$

- The set of roles is organized in a hierarchy (partial order)

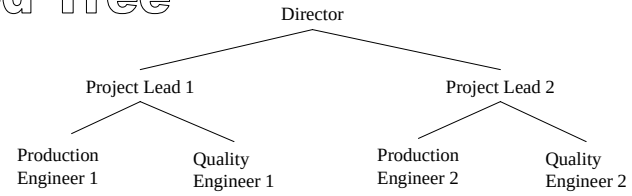
33

Tree Hierarchies



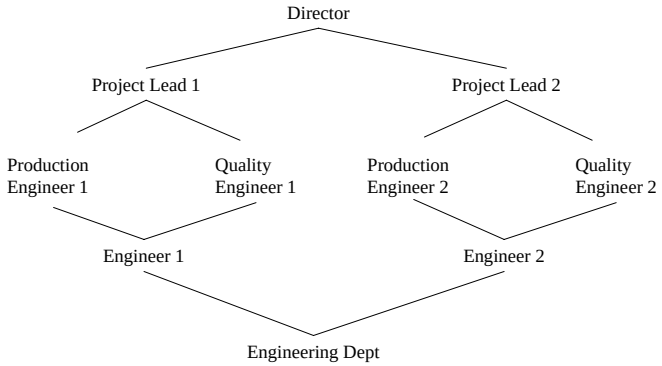
Tree

Inverted Tree



34

Lattice Hierarchy



Supports multiple inheritance

35

General RH

$$UA \subseteq USERSxROLE\ \ S$$

$$r_1 \succeq r_2 \Rightarrow authorized\ \_\ permissions(r_2) \subseteq authorized\ \_\ permissions(r_1)$$
$$\wedge\ authorized\ \_\ users(r_1) \subseteq authorized\ \_\ users(r_2)$$

$$assigned\_user(r) = \{u \in USERS \mid (u,r) \in UA\}$$
$$authorized\ \_\ users(r) = \{u \in USERS \mid r' \succeq r \wedge (u,r') \in UA\}$$

$$authorized\ \_\ permission\ s(r : ROLES\ ) \rightarrow 2^{PRMS}$$
$$authorized\ \_\ permissions(r) = \{p \in PRMS \mid r \succeq r', (p,r') \in PA\}$$

36

Limited RH

A restriction on the immediate descendants in the general role hierarchy: roles can have only one descendant, but may have one or more ascendants (single inheritance)

Role2

Role2 inherits from Role1

Role1

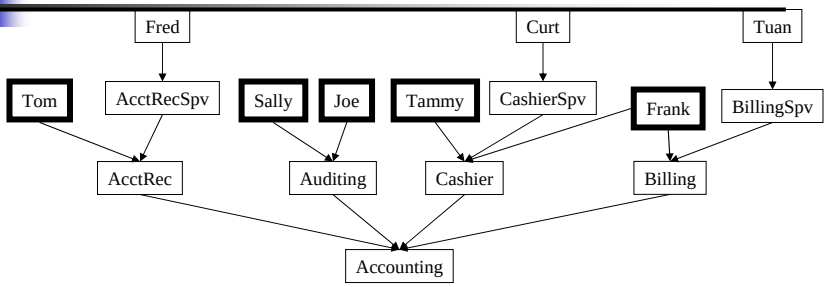
- Role3

Role3 does not inherit from Role1 or Role2

$$\forall r, r_1, r_2 \in ROLES, r \succeq r_1 \wedge r \succeq r_2 \Rightarrow r_1 = r_2$$

37

Limited RH (cont)

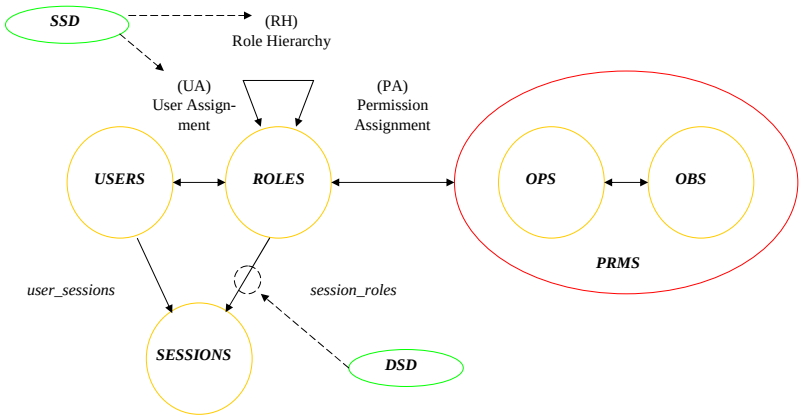


Accounting Role

Notice that Frank has two roles: Billing and Cashier
This requires the union of two distinct roles and prevents Frank from being a (role) node to others

38

Constrained RBAC



39

Separation of Duties

- 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.
- Two Types:
  - Static Separation of Duties (SSD)
  - Dynamic Separation of Duties (DSD)

40

Separation of Duty (static)

- For r a role, the predicate *meauth*(r) (for *m*utually *e*xclusive *auth*orizations) is the set of roles that a subject s , for which $r \in$ *auth*(s), cannot assume because of some separation of duty requirement.
- Separation of duty constraint:

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

41

SSD

$$SSD \subseteq (2^{ROLES} \; xN)$$

- SSD places restrictions on the set of roles and in particular on their ability to form *UA* relations.
- No user is assigned to n or more roles from the same role set, where n or more roles conflict with each other.
- A user may be in one role, but not in another—mutually exclusive.
- Prevents a person from submitting and approving their own request.

$$\forall (rs, n) \in SSD, \forall t \subseteq rs : | t | \geq n \Rightarrow \bigcap_{r \in t} assigned\_users(r) = \varnothing$$

42

SSD in Presence of RH

- A constraint on the authorized users of the roles that have an SSD relation.
- Based on the authorized users rather than assigned users.
- Ensures that inheritance does not undermine SSD policies.
- Reduce the number of potential permissions that can be made available to a user by placing constraints on the users that can be assigned to a set of roles.

$$\forall (rs, n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} authorized\_users(r) = \varnothing$$

43

DSD

$$DSD \subseteq (2^{ROLESxN})$$

- Places constraints on the users that can be assigned to a set of roles, thereby reducing the number of potential prms that can be made available to a user.
- Constraints are across or within a user's session.
- No user may activate *n* or more roles from the roles set in each user session.
- *Timely Revocation of Trust* ensures that prms do not persist beyond the time that they are required for performance of duty.

44

DSD

$$DSD \subseteq (2^{ROLESxN})$$

$$\forall rs \in 2^{ROLES}, n \in N, (rs, n) \in DSD \Rightarrow n \geq 2\text{^}|\, rs \,| \geq n, and$$

$$\forall s \in SESSIONS, \forall rs \in 2^{ROLES}, \forall role\_subset \in 2^{ROLES}, \forall n \in N, (rs, n) \in DSD,$$
$$role\_subset \subseteq rs, role\_subset \subseteq session\_role(s) \Rightarrow |\, role\_subset \,| < n$$

There can be constraints on mutually exclusive permissions as well, both static and dynamic

45