CS590U
Access Control: Theory and Practice

Lecture 12 (February 17)

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.

- 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:

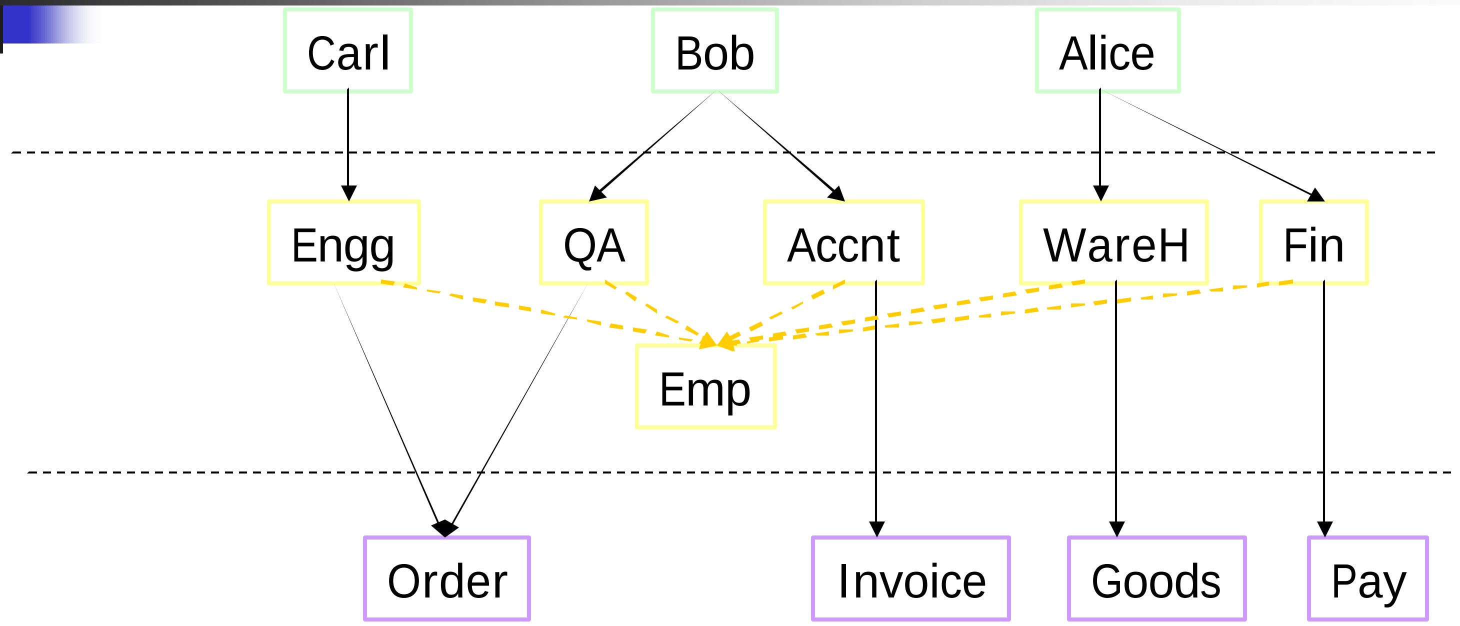
(1) No user performs (a) and (d).

(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

SoD and RBAC



- Static SoD policy: ssod($\{ p_1, \ldots, p_n \}$, k)
  - e_1 = ssod({order, pay}, 2)
  - e_2 = ssod({order, invoice, goods, pay}, 3)

SSoD Safety

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

SCSSOD is coNP-complete

Proof: Show that determining whether γ is not safe wrt. E is NP-complete.

In NP: if unsafe, then $\exists$ ssod($\{p_1, \ldots, p_n\}$,k) in E, and k-1 users such that the permissions they have contains $\{p_1, \ldots, p_n\}$. After guessing e, and k-1 users, can be verified in polynomial time.

NP-hard: The set covering problem: Given a finite set S, $F=\{S_1,\ldots,S_m\}$ (where $S_j \subseteq S$), B, determine whether exist B members of F such that their union is S.

Reduction: each element in S maps to a permission, each S_j maps to a user

SMER Constraints

- Statically mutually-exclusive role (SMER) constraints: smer($\{r_1, \dots, r_m\}$, t)
  - means that no user can be a member of t roles from $\{r_1, \dots, r_m\}$
  - 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:

Terminology Confusion in Literature

- SMER constraints are called SSoD constraints in the literature
  - possible reason: given ssod({p_1, p_2},2), if only r_1 has p_1 and only r_2 has p_2, then making r_1 and r_2 mutually exclusive enforces ssod({p_1, p_2},2)
- Why this is bad?
  - confusing objective with mechanism
  - suppose that one makes r_1 and r_2 exclusive and permission assignment changes, then it may not enforce the SSoD policy anymore

Even more Terminology Confusion

- DMER constraints, which require that certain roles cannot be activated in the same session, are called DSoD constraints in the literature
  - because they are dynamic version of “SSoD constraints”
- However, DMER constraints have nothing to do with Separation of Duty; they are motivated by the Least Privilege Principle.

SMER Constraints and SSoD Policies

- How effective is it to use SMER constraints to enforce SSoD policies?

SC-SMER

- Definition: An RBAC state γ satisfies an SMER constraint smer($\{r_1, \ldots, r_m\}$, t) iff. no user is a member of at least t roles in $\{r_1, \ldots, r_m\}$
- Firstly: can we check whether an RBAC state satisfies an SMER constraint efficiently?
- Yes: for each user
  - compute set of roles of which she is a member
  - intersect with set of roles from constraint
  - check if size $< t$

SSoD and SMER

- Enforcement Verification (EV) problem: whether a set C of SMER constraints enforces a set E of SSoD policies under a given PA and RH
  - for all possible user-role assignments, does $satisfies_C(s) => safe_E(s)$?

CEV

- CEV problem: similar to EV, except with
  - Singleton set of SSoD policies
  - Set of canonical SMER constraints

- EV and CEV are coNP-complete
  - Monotone-3-2-SAT reduces to CEV with only 2-2 SMER constraints
  - EV is in coNP

Monotone 3-2-SAT is NP-complete

- CNF-SAT is to determine whether a list of disjunctive clauses can be satisfied at the same time
  - e.g., $(p1 \vee \neg p2 \vee \neg p3) \wedge (p2 \vee \neg p3 \vee p4) \wedge$
- In a monotone 3-2-SAT instance, each clause either consists of 3 positive literals, or 2 negative literals
- Every 3-SAT instance can be transformed to an equivalent 3-2-SAT instance.

A Special Case of CEV is NP-complete

- Determining whether a set of 2-2 smer constraints does not enforce a 2-n SSoD policy is NP-complete
- Given a monotone 3-2-SAT instance,
  - for each clause, creates a permission,
  - for each role creates a propositional variable,
  - each positive clause is translated into permission-role assignments
  - each negative clause is translated into a 2-2 smer

The case in favor of SMER

- EV needs to be performed only when role-role or permission-role relationships change. These are infrequent.

- When (u,r) is added to UA, only SC-SMER needs to be checked.

- Complement of CEV reduces to SAT.

Generation of SMER

- How did SMER constraints get there in the first place (for us to consider EV)?
- Alternate approach: start with set E of SSoD policies, then generate SMER constraints. Then, EV is inconsequential.
- Naïve approach: make each role mutually exclusive from every other role. But this is too restrictive.

First Step: From SSoD to RSSoD

- SSoD policies are about permissions
- SMER constraints are about role memberships
- Need to translate requirements on permissions to those on roles
  - $ssod(\{p_1,...p_n\}, k)$ no k-1 users have all permissions
  - $rssod(\{r_1,...,r_n\}, k)$ no k-1 users have all roles
  - $smer(\{r_1,...,r_m\}, t)$ no single user has t or more roles

A Generation Algorithm

Input: rssod(R, k)

Output: SMER constraints

```
Let n = |R|, S = emptyset
If k = 2 output smer(R, n)
Else
    for all j from 2 to floor((n-1)/(k-1)) + 1
        let m = (k-1)(j-1) + 1
        for each size-m subset R' of R
            output smer(R', j)
```

Output of the Algorithm

- If k = 2, output is smer(R, n)
- If k = n, output is smer(R, 2)
- In other cases, we get multiple outputs. *Each* is sufficient to enforce the RSSoD requirement.

- How good is the algorithm?

Precise Enforcement

- A set C of SMER constraints precisely enforces a set D of RSSoD requirements when for every state s:
  - $satisfies_C(s) \Rightarrow safe_D(s)$
  - $safe_D(s)$ and $live_D(s) \Rightarrow satisfies_C(s)$

- Only two cases that precise enforcement is possible for rssod(R, k):
  - $k = 2$
  - $k = n = |R|$

Minimal Enforcement

- C is minimal if C enforces D and no other constraint that enforces D is less restrictive.
- If C is precise, then C is minimal.
- The algorithm:
  - Each constraint that is generated is minimal.
  - Every singleton set of constraints that is minimal is generated.

Next Lecture

- Administration of RBAC