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: \( \text{ssod}(\{ p_1, \ldots, p_n \}, k) \)
  - \( e_1 = \text{ssod}(\{\text{order, pay}\}, 2) \)
  - \( e_2 = \text{ssod}(\{\text{order, invoice, goods, pay}\}, 3) \)
SSoD Safety

- An RBAC state is given by $\langle UA, PA, RH \rangle$.
- **Definition**: An RBAC state $\gamma$ is safe wrt. $ssod(\{p_1, \ldots, p_n\}, k)$ 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 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 \( \gamma \) 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:
  \[ \text{smer}\{\{r_1, \ldots, r_m\}, t\} \]
  - 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:
    - \( c_1 = \text{smer}\{\{\text{WHouse, Accnt, Fin}\}, 2\} \)
    - \( c_2 = \text{smer}\{\{\text{Engg, Fin}\}, 2\} \)
    - \( c_3 = \text{smer}\{\{\text{QA, Fin}\}, 2\} \)
SMER constraints are called SSoD constraints in the literature

possible reason: given $\text{ssod}({p_1, p_2}, 2)$, if only $r_1$ has $p_1$ and only $r_2$ has $p_2$, then making $r_1$ and $r_2$ mutually exclusive enforces $\text{ssod}({p_1, p_2}, 2)$

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 $\gamma$ satisfies an SMER constraint $\text{smer}(\{r_1, \ldots, r_m\}, t)$ 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 \( \text{satisfies}_C(s) \Rightarrow \text{safe}_E(s) \) ?
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., \((p1 \lor \neg p2 \lor \neg p3) \land (p2 \lor \neg p3 \lor p4) \land\)
- 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.
Next Lecture

- Constraint Generation