Chapter 14: Protection
Goals of Protection

- Operating system consists of a collection of objects, hardware or software

- Each object has a unique name and can be accessed through a well-defined set of operations.

- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
Domain Structure

- Access-right = \(<object-name, rights-set>\)
  where \(rights-set\) is a subset of all valid operations that can be performed on the object.

- Domain = set of access-rights
Access Matrix

- View protection as a matrix (*access matrix*)

- Rows represent domains

- Columns represent objects

- \( \text{Access}(i, j) \) is the set of operations that a process executing in Domain\(_i\) can invoke on Object\(_j\)
## Access Matrix

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>print</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td></td>
<td>read</td>
<td>write</td>
<td></td>
<td>read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write</td>
</tr>
</tbody>
</table>
Implementation of Access Matrix

- Each column = Access-control list for one object
  Defines who can perform what operation.

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects.

  Object 1 – Read
  Object 4 – Read, Write, Execute
  Object 5 – Read, Write, Delete, Copy
Revocation of Access Rights

- **Access List** – Delete access rights from access list.
  - Simple
  - Immediate

- **Capability List** – Scheme required to locate capability in the system before capability can be revoked.
  - Reacquisition
  - Back-pointers
  - Indirection
  - Keys
Role-based Access Control in Solaris 10

user 1

role 1

privileges 1

privileges 2

executes with role 1 privileges

process
Role-Based Access Control (RBAC)

- **Users** are assigned to **roles**.
  - Example: $UA = \{(\text{Bob}, \text{Doctor})\}$

- **Permissions** are associated with roles.
  - Example: $PA = \{(\text{Doctor}, \text{Modify}, \text{Prescriptions})\}$

- A user has a permission if he is a member of some role with that permission.
ARBAC Syntax

- **Example:**
  - `can_assign(President, Professor ∧ ¬DeptChair, Dean)`: Administrator in role President can assign a user in role Professor and not in role DeptChair to role Dean.
    - Professor is a **positive precondition**, DeptChair is a **negative precondition** and Dean is a target.

- **Example:**
  - `can_revoke(DeptChair, TA)`: an administrator in role DeptChair can remove any user from role TA.

- Roles President and DeptChair are **administrative**: have administrative permissions, i.e. are the first components of `can_assign` or `can_revoke` rules.

- All other roles are regular.

- **Separate Administration Restriction**: administrative roles and regular roles are disjoint.
Large organizations have large and complex policies which are managed by multiple administrators.

The effects of an ARBAC policy are often hard to understand by manual inspection alone.

- Changes by one administrator may interact in unintended ways with changes by other administrators.

ARBAC Policy Analysis can help by answering questions such as:

- **User-Role Reachability Problem**: given an initial RBAC policy, an ARBAC policy, a set of administrators, a target user, and a set of roles (called the “goal”), is it possible for those administrators to modify the RBAC policy so that the target user is a member of those roles?
User-Role Reachability: can user $u$ be assigned to all roles in $goal = \{r_1...r_n\}$ by a group of administrators $A$?

Approach: express the analysis problem as a finite state machine where:

- **Initial State:** initial role assignments of users.
- **Accept State:** a state where user $u$ is assigned to all roles in $goal$.
- **Transitions:** changes allowed by ARBAC policy.
Example

ARBAC Policy:

1. `can_assign(r_a, \neg r_6, r_1)`
2. `can_assign(r_a, r_4 \land \neg r_7, r_5)`
3. `can_assign(r_a, r_1, r_2)`
4. `can_assign(r_a, r_1, r_6)`

User-role Reachability Query: Can administrator in \{r_a\} assign user initially in \{r_5\} to \{r_1,r_2\}?

Answer: Yes
ARBAC Policy:

1. $\text{can\_assign}(r_a, \neg r_1, r_2)$
2. $\text{can\_assign}(r_a, r_2, r_3)$
3. $\text{can\_assign}(r_a, r_3 \land \neg r_4, r_5)$
4. $\text{can\_assign}(r_a, r_5, r_6)$
5. $\text{can\_assign}(r_a, \neg r_2, r_7)$
6. $\text{can\_assign}(r_a, r_7, r_8)$
7. $\text{can\_revoke}(r_a, r_1)$
8. $\text{can\_revoke}(r_a, r_2)$
9. $\text{can\_revoke}(r_a, r_3)$
10. $\text{can\_revoke}(r_a, r_5)$
11. $\text{can\_revoke}(r_a, r_6)$
12. $\text{can\_revoke}(r_a, r_7)$

User-role Reachability Query: Can administrator in $\{r_a\}$ assign user initially in $\{r_1, r_4, r_7\}$ to $\{r_6\}$?
Example Contd.

- 32 states, 96 transitions:
In realistic ARBAC policies the sets of administrative and regular roles may **not always be disjoint**.

- They violate the separate administration restriction

Administrators may assign themselves to new roles:

- **Example:**
  
  1. `can_assign(DeptChair, DeptChair, HonorsPgmDir)`
  2. `can_assign(HonorsPgmDir, Student, HonorsStudent)`

**Query**: Can users in `{(u₁,DeptChair),(uₜ,Student)}` assign $uₜ$ to role `{HonorsStudent}`?

```latex
ua(u₁,HonorsPgmDir) ua(uₜ,HonorsStudent)
```
ARBAC Policy:

- Role Hierarchy: \( r_3 \geq r_2; \ r_3 \geq r_8 \)
- 1. can assign\( (r_8, r_9 \land \neg r_1, r_4) \)
- 2. can assign\( (r_1, r_1 \land r_10 \land r_5, r_9) \)
- 3. can assign\( (r_1, r_3 \land \neg r_5, r_10) \)
- 4. can assign\( (r_3, r_1 \land \neg r_6 \land \neg r_4, r_2) \)
- 5. can revoke\( (r_3, r_5) \)
- 6. can assign\( (r_3, r_9 \land \neg r_1, r_4) \)
- 7. can assign\( (r_2, r_3, r_4) \)
- 8. can assign\( (r_5, \text{true}, r_6) \)
- 9. can assign\( (r_3, r_3, r_4) \)
- 10. can assign\( (r_6, \text{true}, r_7) \)
- 11. can revoke\( (r_3, r_1) \)

Query: Can users in \{ \((u_1, r_1), (u_1, r_3), (u_1, r_5), (u_2, r_1), (u_2, r_3), (u_2, r_5), (u_t, r_1), (u_t, r_3), (u_t, r_5)\) \} together assign user \( u_t \) to roles \{ \( r_4, r_7 \) \}?

The graph contains 5,312 states and 27,216 transitions!
Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.

- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.

- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.
End of Chapter 14