Protection
Protection

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection
Objectives

• Discuss the goals and principles of protection in a modern computer system
• Explain how protection domains combined with an access matrix are used to specify the resources a process may access
• Examine capability and language-based protection systems
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.
Principles of Protection

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

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

- System consists of 2 domains:
  - User
  - Supervisor

- UNIX
  - Domain = user-id
  - Domain switch accomplished via file system.
    - Each file has associated with it a domain bit (setuid bit).
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset.
Domain Implementation (Multics)

- Let $D_i$ and $D_j$ be any two domain rings.
- If $j < i \Rightarrow D_i \subseteq D_j$
Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- 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>print</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td></td>
<td>read</td>
<td></td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>

**Figure A**
Use of Access Matrix

- If a process in Domain $D_i$ tries to do “op” on object $O_j$, then “op” must be in the access matrix.

- Can be expanded to dynamic protection.
  - Operations to add, delete access rights.
  - Special access rights:
    - owner of $O_i$
    - copy op from $O_i$ to $O_j$
    - control – $D_i$ can modify $D_j$ access rights
    - transfer – switch from domain $D_i$ to $D_j$
Use of Access Matrix (Cont.)

- Access matrix design separates mechanism from policy.
  - **Mechanism**
    - Operating system provides access-matrix + rules.
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
  - **Policy**
    - User dictates policy.
    - Who can access what object and in what mode.
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 a given domain, what operations are allowed on each object.
  - Object 1 – Read
  - Object 4 – Read, Write, Execute
  - Object 5 – Read, Write, Delete, Copy
## Access Matrix of Figure A With Domains as Objects

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

**Figure B**
## Access Matrix with Copy Rights

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>execute</td>
<td></td>
<td>write*</td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>execute</td>
<td></td>
<td>write*</td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>execute</td>
<td>read</td>
<td></td>
</tr>
</tbody>
</table>

(b)
### Access Matrix With Owner Rights

#### (a)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>owner</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>read*</td>
<td>read*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>owner</td>
<td>owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### (b)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td>owner</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>owner</td>
<td>read*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>read*</td>
<td>owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>write*</td>
<td>write</td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td>write</td>
<td>write</td>
</tr>
<tr>
<td>domain</td>
<td>object</td>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$F_3$</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>$D_1$</td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td>print</td>
</tr>
<tr>
<td>$D_3$</td>
<td>read</td>
<td></td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>write</td>
<td></td>
<td>write</td>
<td></td>
</tr>
</tbody>
</table>
Access Control

- Protection can be applied to non-file resources
- Solaris 10 provides **role-based access control** to implement least privilege
  - Privilege is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned roles granting access to privileges and programs
Role-based Access Control in Solaris 10

user 1

role 1

privileges 1

privileges 2

executes with role 1 privileges

process
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
Capability-Based Systems

• Hydra
  ◦ Fixed set of access rights known to and interpreted by the system.
  ◦ Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights.

• Cambridge CAP System
  ◦ Data capability - provides standard read, write, execute of individual storage segments associated with object.
  ◦ Software capability - interpretation left to the subsystem, through its protected procedures.
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.