CSE 30341
Operating System Principles

Protection

Overview
- 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-based protection systems

Goals of Protection

• In one common protection model, a computer 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**
  - Static
  - Dynamic - **domain switching, privilege escalation**
  - “Need to know” a similar concept regarding access to data
  - “Containment of failure”

- Must consider “grain” aspect
  - Rough-grained
  - Fine-grained

- Domain can be user, process, procedure

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
Domain Implementation (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 (similarly "setgid")
  - When execution completes user-id is reset

- Domain switch accomplished via passwords
  - `su` command temporarily switches to another user’s domain when other domain’s password provided

- Domain switching via commands
  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (MULTICS)

- Let $D_i$ and $D_j$ be any two domain rings
- If $j < i \Rightarrow D_i \subseteq D_j$
Multics Benefits and Limits

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex -> more overhead
- But does not allow strict need-to-know
  - Object accessible in $D_j$ but not in $D_i$, then $j$ must be $< i$
  - But then every segment accessible in $D_i$ also accessible in $D_j$

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>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>printer</th>
</tr>
</thead>
<tbody>
<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></td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>read write</td>
<td>read write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## 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.
- User who creates object can define access column for that object.
- Can be expanded to dynamic protection:
  - Operations to add, delete access rights
  - Special access rights:
    - owner of $O_j$
    - copy op from $D_i$ to $D_j$ (denoted by “*”)
    - control – $D_i$ can modify $D_j$ access rights
    - transfer (switch) – switch from domain $D_i$ to $D_j$
  - Copy and Owner applicable to an object
  - Control applicable to domain
Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - It 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
    - Good policy supported by good default values

Access Matrix Example

<table>
<thead>
<tr>
<th>object domain</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
<th>laser printer</th>
<th>D₁</th>
<th>D₂</th>
<th>D₃</th>
<th>D₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₁</td>
<td>read</td>
<td>read</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td></td>
<td></td>
<td></td>
<td>print</td>
<td>switch</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₃</td>
<td></td>
<td></td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₄</td>
<td>read</td>
<td>write</td>
<td></td>
<td>read</td>
<td>write</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access Matrix Example

(a)

<table>
<thead>
<tr>
<th>object domain</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>execute</td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
</tr>
<tr>
<td>D3</td>
<td>execute</td>
<td>read</td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>object domain</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>execute</td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
</tr>
<tr>
<td>D3</td>
<td>execute</td>
<td>read</td>
<td></td>
</tr>
</tbody>
</table>

Access Matrix Example

(a)

<table>
<thead>
<tr>
<th>object domain</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>owner</td>
<td>execute</td>
<td>write</td>
</tr>
<tr>
<td>D2</td>
<td>read</td>
<td>owner</td>
<td>read*</td>
</tr>
<tr>
<td>D3</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>object domain</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>owner</td>
<td>execute</td>
<td>write</td>
</tr>
<tr>
<td>D2</td>
<td>owner</td>
<td>read*</td>
<td>write</td>
</tr>
<tr>
<td>D3</td>
<td>write</td>
<td></td>
<td>write</td>
</tr>
</tbody>
</table>
Access Matrix Example

<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>read</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( D_2 )</td>
<td></td>
<td>print</td>
<td></td>
<td></td>
<td></td>
<td>switch</td>
<td>switch control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( D_3 )</td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( D_4 )</td>
<td>write</td>
<td>write</td>
<td></td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation of Access Matrix

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples \(<\text{domain, object, rights-set}>\) in table
  - A requested operation \( M \) on object \( O_j \) within domain \( D_i \) -> search table for \(<D_i, O_j, R_k>\)
    - with \( M \in R_k \)
  - But table could be large -> won’t fit in main memory
  - Difficult to group objects (consider an object that all domains can read)
- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs \(<\text{domain, rights-set}>\) defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If \( M \in \text{default set} \), also allow access
Implementation of Access Matrix

• Option 3 - Capability list for domains
  -- Instead of object-based, list is domain based
  -- **Capability list** for domain is list of objects together with operations allows on them
  -- Object represented by its name or address, called a **capability**
  -- Execute operation M on object O_j, process requests operation and specifies capability as parameter
    • Possession of capability means access is allowed

Comparison of Implementations

• Many trade-offs to consider
  -- Global table is simple, but can be large
  -- Access lists correspond to needs of users
    • Determining set of access rights for domain non-localized difficult
    • Every access to an object must be checked
      -- Many objects and access rights -> slow
  -- Capability lists useful for localizing information for a given process
    • But revocation capabilities can be inefficient
  -- Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

• Most systems use combination of access lists and capabilities
  -- First access to an object -> access list searched
    • If allowed, capability created and attached to process
      -- Additional accesses need not be checked
    • After last access, capability destroyed
Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent