Logistics

- Keep working hard on Project 2
  - Help desk: Monday 12-2pm, Tuesday 12-2pm
  - Help desk: Wednesday 1-3pm
- Office hours on Wednesday from 12-1pm
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 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

- How do you ensure that each object is accessed correctly?
- How do you ensure that only those processes that are allowed to access an object are actually the only ones to do so?
Principles of Protection (1)

- Guiding principle

  **Principle of Least Privilege**

- Programs, users and systems should be given just enough privilege to perform their tasks
  - Limits damage if entity has a bug or gets abused
  - Can be static
    - during lifetime of system or life of process
  - Can be dynamic
    - changed by process as needed
    - domain switching, privilege escalation
- “Need to know” is a similar concept regarding access to data
Principles of Protection (2)

- Must consider “grain” aspect
  - *Rough-grained* privilege management easier, simpler, but least privilege now done in large chunks
    - for example, traditional Unix processes either have abilities of the associated user, or of root
  - *Fine-grained* management more complex, more overhead, but more protective
    - file ACL lists, RBAC

- Domain can be user, process, procedure
Domain Structure

- **Access-right**
  
  \(<object-name, \text{rights-set}>\)
  
  - 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) (1)

- 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 (Unix) (2)

- 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)
Let $D_i$ and $D_j$ be any two domain rings

If $j < i \Rightarrow D_i \subseteq D_j$

Inner domain rings have more privileges than outer rings
Multics Benefits and Limits

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex
  - More overhead involved in management
- 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 (1)

- 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 (2)

<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></td>
<td>$D_1$</td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>$D_3$</td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_4$</td>
<td>read</td>
<td>write</td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>
Use of Access Matrix (1)

- 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 $O_i$
    - copy op from $O_i$ to $O_j$ (denoted by “*”)  
    - control – $D_i$ can modify $D_j$ access rights
    - transfer – switch from domain $D_i$ to $D_j$
  - Copy and Owner applicable to an object
  - Control applicable to domain object
Use of Access Matrix (2)

- 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
### Access Matrix 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>printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>read</td>
<td></td>
<td>read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>print</td>
<td></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>read write</td>
<td></td>
<td></td>
<td>read write</td>
<td></td>
</tr>
</tbody>
</table>

<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>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>print</td>
<td></td>
<td>switch</td>
<td>switch</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>read write</td>
<td></td>
<td></td>
<td>read write</td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Access Matrix with Copy Rights

#### (a)

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

#### (b)

<table>
<thead>
<tr>
<th>domain</th>
<th>object</th>
<th>F₁</th>
<th>F₂</th>
<th>F₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₁</td>
<td>execute</td>
<td></td>
<td></td>
<td>write*</td>
</tr>
<tr>
<td>D₂</td>
<td>execute</td>
<td></td>
<td>read*</td>
<td>execute</td>
</tr>
<tr>
<td>D₃</td>
<td>execute</td>
<td></td>
<td>read</td>
<td></td>
</tr>
</tbody>
</table>
## Access Matrix with Owner Rights

<table>
<thead>
<tr>
<th></th>
<th>Domain 1</th>
<th>Domain 2</th>
<th>Domain 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><strong>F1</strong></td>
<td><strong>F2</strong></td>
<td><strong>F3</strong></td>
</tr>
<tr>
<td>Domain 1 (D1)</td>
<td>owner execute</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>Domain 2 (D2)</td>
<td>read* owner</td>
<td>read* owner</td>
<td>read* owner write</td>
</tr>
<tr>
<td>Domain 3 (D3)</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>Domain 1</th>
<th>Domain 2</th>
<th>Domain 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><strong>F1</strong></td>
<td><strong>F2</strong></td>
<td><strong>F3</strong></td>
</tr>
<tr>
<td>Domain 1 (D1)</td>
<td>owner execute</td>
<td></td>
<td>write</td>
</tr>
<tr>
<td>Domain 2 (D2)</td>
<td>owner read* write*</td>
<td>read* owner write</td>
<td></td>
</tr>
<tr>
<td>Domain 3 (D3)</td>
<td>write</td>
<td></td>
<td>write</td>
</tr>
</tbody>
</table>

(b)
# Modified Access Matrix

<table>
<thead>
<tr>
<th>Domain</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>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td>print</td>
<td>switch</td>
<td>switch</td>
<td>switch</td>
<td>control</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>
</tr>
<tr>
<td>$D_4$</td>
<td>write</td>
<td>write</td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation of Access Matrix (1)

- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples in a table
    - \(<\text{domain, object, rights-set}>\)
  - 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
    - might not 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 
\(<domain, rights-set>\)

defining all domains with non-empty set of access rights for the object

Easily extended to contain default set

◆ if \( M \in default set \), also allow access
Implementation of Access Matrix (3)

- Each column is an *access-control list* for one object
  - Defines who can perform what operation
    - Domain 1 = Read, Write
    - Domain 2 = Read
    - Domain 3 = Read
  
- Each row represents a *capability list* (like a key)
  - For a given domain, the capability list gives what operations are allowed on each objects
    - Object 1 – Read
    - Object 4 – Read, Write, Execute
    - Object 5 – Read, Write, Delete, Copy
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
- Capability list associated with domain but never directly accessible by domain
  - rather, protected object, maintained by OS and accessed indirectly
  - like a “secure pointer”
  - idea can be extended up to applications
Implementation of Access Matrix (5)

- Option 4 – *Lock-key*
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called *locks*
  - Each domain as list of unique bit patterns called *keys*
  - Process in a domain can only access object if domain has key that matches one of the locks
Comparison of Implementations (1)

- 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 so difficult
    - every access to an object must be checked
      - many objects and access rights
      - could be 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
Comparison of Implementations (2)

- 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
    - consider file system with ACLs per file
Role-based 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
Revocation of Access Rights (1)

- 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

- Access List
  - Delete access rights from access list
  - Simple – search access list and remove entry
    - Immediate, general or selective, total or partial, permanent or temporary, is more involved
Revocation of Access Rights (2)

- Capability List
  - Scheme required to locate capability in the system before capability can be revoked
  - Reacquisition – periodic delete, with require and denial if revoked
  - Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
  - Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - Keys – unique bits associated with capability, generated when capability created
    - Master key associated with object, key matches master key for access
    - Revocation – create new master key
    - Policy decision of who can create and modify keys – object owner or others?
Capability-based Systems – Hydra

  - Fixed set of access rights known to and interpreted by the system
    - Such as read, write, or execute each memory segment
    - User can declare other auxiliary rights and register those with protection system
    - Accessing process must hold capability and know name of operation
    - Rights amplification allowed by trustworthy procedures for a specific type
  - Interpretation of user-defined rights performed solely by user's program
  - System provides access protection for use of these rights
  - Operations on objects defined procedurally
    - procedures are objects accessed indirectly by capabilities
  - Solves the problem of mutually suspicious subsystems
  - Includes library of prewritten security routines
CAP (http://research.microsoft.com/en-us/um/cambridge/events/needhambook/cap.pdf)

- Simpler but powerful
- Data capability
  - provides standard read, write, execute of individual storage segments associated with object – implemented in microcode
- Software capability
  - interpretation left to the subsystem, through its protected procedures
  - only has access to its own subsystem
  - programmers must learn principles and techniques of protection
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.
Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM
- The protection domain indicates what operations the class can (and cannot) perform
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library
- Generally, Java’s load-time and run-time checks enforce type safety
- Classes effectively encapsulate and protect data and methods from other classes