Chapter 14: 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

Principles of Protection

- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
  - Limit damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed – domain switching, privilege escalation)
  - Need to know – a similar concept regarding access to data
- Must consider “gran” 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
- Domain can be user, process, procedure

Goals of Protection

- In one protection model, 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

Domain Structure

- Access-right = subject-name, rights-set
  - Rights-set is a subset of all valid operations that can be performed on the object
- Domain = set of access rights

- Access-right = subject-name, rights-set

- Domain = set of access rights

- Domain Structure

- Access-right = subject-name, rights-set
  - 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
  - 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 is 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 Di and Dj be any two domain rings
- If j < i ⇒ Di ⊆ Dj

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 Dj but not in Di, then j must be < i
  - But then every segment accessible in Di also accessible in Dj

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 Domaini can invoke on Objectj

Use of Access Matrix

- If a process in Domain Di tries to do "op" on object Oj, 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 Oi or Oj
    - control — switch from domain Di to Dj
    - manage — switch from user to domain
  - Copy and Owner applicable to an object
  - Control applicable to domain object
Use of Access Matrix (Cont.)

- Access matrix design separates mechanisms from policy.
  - Mechanism:
    - Operating system provides access-matrix + rules
    - Ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy:
    - User defines policy
    - Who can access what object and in what mode
  - But doesn’t solve the general confinement problem

Access Matrix of Figure A with Domains as Objects

<table>
<thead>
<tr>
<th>domain</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
<th>$D_5$</th>
<th>$D_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>read</td>
<td>read</td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td>print</td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_3$</td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_4$</td>
<td>read</td>
<td>write</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access Matrix with Copy Rights

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

Access Matrix With Owner Rights

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

Modified Access Matrix of Figure B

<table>
<thead>
<tr>
<th>domain</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
<th>$D_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>read</td>
<td>read</td>
<td>switch</td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td>print</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_3$</td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_4$</td>
<td>write</td>
<td>write</td>
<td>switch</td>
<td></td>
</tr>
</tbody>
</table>

Implementation of Access Matrix

- Generally a sparse matrix
  - Option 1: Global table
    - Store ordered triples $<$ domain, object, rights-set $>$ in table
    - A requested operation on an 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 $<$ domain, rights-set $>$ defining all domains with non-empty set of access rights for the object
    - Easily extended to contain default set $= 0$ if no default set, also allow access
Most systems use a combination of access lists and capabilities.

Many trade-offs to consider:
- Global table is simple, but can be large.
- Revocation of access rights can be inefficient.
- ACCESS MATRIX – similar to access matrix

For each domain, what operations allowed on what objects
- Each domain can only access objects it has keys for.
- Possession of capability means access is allowed.
- Use of “emulator” key – creation can be extended to applications.
- Each domain has list of unique bit patterns called "keys".
- Process in a domain can only access object if domain has key that matches one of the keys.

Access Control in Solaris 10
- Role-based access control
- Idea can be extended up to applications.
- Policy decision of who can create and modify keys – object owner or others?
- Secure pointer – like a "secure pointer".
- Possession of capability means access is allowed.
- Rather, protected object, maintained by OS and accessed indirectly.

Implementation of Access Matrix (Cont.)
- Option 3 – Capability list for domains
  - 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
- Many trade-offs to consider:
  - Global table is simple, but can be large.
  - REVOCATION of access rights can be inefficient.

Role-based Access Control in Solaris 10
- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Partial vs. total
  - Temporary vs. permanent
- Access List – Delete access lists from access list
  - Simple – search access list and remove entry
  - Immediate, general, or selective, total or partial, permanent or temporary
- Capability List – Scheme required to locate capability in the system before capability can be revoked
  - Master key associated with object, key matches master key for access.
  - Possession of capability means access is allowed.
  - Rather, protected object, maintained by OS and accessed indirectly.

Revision of Access Rights
- Various options to remove the access right of a domain to an object
  - Immediate vs. deferred
  - Partial vs. total
  - Temporary vs. permanent
### Capability-Based Systems

- **Hydra**
  - Fixed set of access rights known to and interpreted by the system
  - i.e. 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 only by user's program; system provides access protection for use of those rights
  - Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities
  - Serves the problem of mutually suspicious subsystems
- **Cambridge CAP System**
  - 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 is 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
- Interpreter protection specifications to generate calls to 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

### Stack Inspection

<table>
<thead>
<tr>
<th>protection domain</th>
<th>utility/applet</th>
<th>utility/loader</th>
<th>networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>* localhost.com (80), connect</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>get/DPL (s):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>()</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open/oth (d):</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>request/&lt;from proxy&gt;</td>
<td>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## End of Chapter 13