Protection and Security

- **Protection**
  - Mechanisms supporting policies to control access to system resources
  - Protection policies essentially aim to prevent unauthorized access, modification, or destruction of data and resources.

- **Security**
  - Mechanisms to prevent external circumvention of the protection system
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.

Objects and Operations

- Objects are both hardware and software entities, at different levels of abstraction
  - CPU
  - Memory
  - File
  - Process

- Each object has type-specific operations
  - CPU can be executed
  - Memory can be read or written
  - Program files can be read, written, executed
Privileged Operations

- Even if an object supports a particular operation, we may not want to allow it for all processes
  - Only Alice can read her files
  - Only process P can read memory block M
  - Only the kernel can execute the `inb` instruction

1st Principle of Security Design

*Least Privilege* (“need to know”): each principal is given the minimum access needed to accomplish its task. [Saltzer & Schroeder ‘75]

Examples:

+ Administrators don’t run day-to-day tasks as root.
  So “`rm -rf /`” won’t wipe the disk.
+ In contrast: fingerd runs as root so it can access different users’ `.plan` files. But then it can also “`rm -rf /`.”
Policies should Support LP

- Should be possible to specify different sets of permitted operations for the same objects
  - Like a role, or domain of authority.
- Should be easy to switch between roles, to control “dangerous” operations
  - System calls
  - `su` or `newgroup` commands

Least Privilege Generally

Least Privilege shows up in almost all engineering design patterns. E.g., SE & languages:
- abstract data types,
- strong interfaces,
- Encapsulation (OO),
- black-box principle, etc.
Domains

- **Access-right** = `<object-id, rights-set>`
  where `rights-set` is a subset of all valid operations that can be performed on the object.
- **Domain** = set of access-rights

![Diagram showing access rights and domains]

Domain Use

- A process P executing “within” domain D is granted all of the access rights specified by the domain.
- For simplicity, we think of a process only ever within one domain at a time
- To change its rights, we may allow
  - A process to switch domains as it runs
  - A domain to expand its access rights
Domains in UNIX

- Two broad domains:
  - User
  - Supervisor
    - Switch from user to supervisor via system calls

- User domains further subdivided
  - Domain = user-id (+group-id)

Domains in UNIX

- User domain switch via `su` syscall
- ... or the file system
  - When file is executed, if its `setuid` bit is set, then user-id is set to owner of the file being executed. When execution completes user-id is reset.
- ... or by message passing
  - Send a message to a more privileged process to perform an operation on your behalf
Domains in UNIX

- User domain access rights expanded and contracted through the file system
  - Adding a user-id to a group permits it to access files at the group’s privileges
  - Changing the access rights of a file may allow other domains to access it

Access Matrix

- A protection policy can be viewed 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$_j$
- Policy is established by the OS, and the users. Matrix (mechanism) is enforced by the OS and the hardware.
**Use of Access Matrix**

- If a process in Domain $D_i$ tries to do $op$ on object $O_j$, then $op$ must be in $Access(i,j)$.

- Can be expanded to dynamic protection.
  - Operations to add, delete access rights.
  - Special access rights:
    - *owner of $O_i$*
    - *copy right from $O_i$ to $O_j*
    - *control - $D_i$ can modify $D_j$ access rights*
    - *transfer - switch rights from domain $D_i$ to $D_j*$

---

**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></td>
<td>read</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>read</td>
<td>write</td>
<td></td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>

*Figure A*
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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td>print</td>
<td></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</td>
<td></td>
<td>write</td>
<td></td>
<td>switch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B

Access Matrix and 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>execute</td>
<td></td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>execute</td>
<td></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>execute</td>
<td></td>
<td>write*</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>execute</td>
<td>read*</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>execute</td>
<td>read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b)
Access Matrix and Owner Rights

![Modified Access Matrix of Figure B (control rights)](image)

Modified Access Matrix of Figure B (control rights)
Problems with Access Control

- Must be enforced at every step
  - What if process $P$ opens and begins reading a file for which it has been given access, but then that access is revoked?
- Does not dictate information propagation, only initial access
  - What if process $P$ copies an authorized file $F$ to a location accessible by $Q$, a process not allowed to access $F$?

Mandatory Access Control

- The systems with which we are familiar (UNIX, Windows) employ discretionary access control, in which policies are at the users’ discretion
- Mandatory access control creates an ordering of users that the system enforces across all of its operations
  - This would forbid the illegal information flow described before
Access Control Lists

• **Access-control list** (ACL) implements each column in the matrix. Defines which domain can perform what operation on each object.

  Object O:  
  Domain 1 = Read, Write  
  Domain 2 = Read  
  Domain 3 = Read

  Object P:  
  ...

  • For each operation OP on O, find it’s ACL, ensure the current domain D has permission to perform OP.

Capability List

• **Capability List** implements each row in the matrix.

  • A **capability** is like a key that permits some set of operations on an object.
    - To perform operation OP on object O, the process must present a capability C that states it may do so. (The object and the capability may be synonymous.)

  • Each domain is granted a list of capabilities
Acquiring Capabilities

- Can be implicit, based on the domain in which a process executes
- Can be explicit, based on actions
  - For example, UNIX file descriptors are capabilities granted based on traditional access control via `open`
- Capabilities, once acquired, must be tamper-proof
  - Hardware or software-based

Revocation of Access Rights

- **ACL** - Delete access rights from list.
  - Simple
  - Immediate (almost)

- **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.

Capability Unforgeability

- In Hydra and CAP, unforgeability is implemented via (special) hardware
  - In CAP, capabilities are stored in capability segments. Their meaning is determined by a parent process (e.g., the OS kernel) to whose memory they do not have access.

- In Eros, it is implemented on commodity hardware
  - Using virtual memory protection, as with UNIX

- We can also implement this via language-based protection.
Language-Based Protection

- Implement these systems in the programming language, not the OS
  - Provides more flexibility: objects are application-specific (high-level) rather than system-specific (low-level).
  - Problem of protection: how to avoid circumventing security checks? Use type-safety and verification.

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 perform.
- Type-safety ensures enforcement cannot be circumvented.
- 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.
**2\textsuperscript{nd} Principle of Security Design**

*Keep the Trusted Computing Base small.*

Trusted Computing Base (TCB):
- the parts of a system that must work correctly to ensure the proper functioning of the system.
- e.g., the OS Kernel & Hardware.

Smaller, simpler systems tend to have fewer bugs and bad interactions.
- so keep the kernel small and simple.

“Small TCB” is a basic principle in all software.
Who do you trust?

• Do I trust a login prompt?
• Do I trust the OS that I got from the vendor?
• Do I trust the system staff?
  - should I encrypt all my files?
• Networking
  - do you trust the network provider?
  - do you trust the phone company?
• How do you bootstrap security?
  - need one “out of band” transfer to get going