OS access control

- Kernel reference monitor
  - Who (subject) gets to access what (object) and how (rights)
- Goal: protect the confidentiality and integrity of objects
- Older than computers, policies were created for accessing classified documents, e.g.,
  - Bell--LaPadula Model (confidentiality)
  - Biba model (integrity)
Access control in computer systems

- **Subject**: users (processes)
- **Object**: all other OS abstractions
  - Including other processes
- **Rights**
  - Unix: read, write, execute (rwx)
  - Windows: more complicated
**Principles of access control**

- **Complete mediation**: any access to any object should be checked by the access control system
- **Tamper proof**: data used by the access control system should be protected from illegal modification
- **Correctness**: the correctness of the access control system should be verifiable
Complete mediation
### Access control matrix

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objects</th>
<th>/one</th>
<th>/two</th>
<th>/three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>rw</td>
<td>-</td>
<td>rw</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>w</td>
<td>-</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>w</td>
<td>r</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

- Problem: sparse, many cells are empty
Access control lists (ACL)

- ACL: associate access rights with objects

<table>
<thead>
<tr>
<th>Subjects</th>
<th>/one</th>
<th>/two</th>
<th>/three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>rw</td>
<td>-</td>
<td>rw</td>
</tr>
<tr>
<td>Bob</td>
<td>w</td>
<td>-</td>
<td>r</td>
</tr>
<tr>
<td>Charlie</td>
<td>w</td>
<td>r</td>
<td>rw</td>
</tr>
</tbody>
</table>
**Capabilities**

- Cap: associate access rights with subjects

<table>
<thead>
<tr>
<th>Subjects</th>
<th>/one</th>
<th>/two</th>
<th>/three</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>rw</td>
<td>-</td>
<td>rw</td>
</tr>
<tr>
<td>Bob</td>
<td>w</td>
<td>-</td>
<td>r</td>
</tr>
<tr>
<td>Charlie</td>
<td>w</td>
<td>r</td>
<td>rw</td>
</tr>
</tbody>
</table>
ACLs and capabilities

- Capabilities are easier to transfer
  - They are like keys, can handoff, does not depend on subject
- In practice, ACLs are easier to manage
  - Object-centric, easy to grant, revoke
  - To revoke capabilities, have to keep track of all subjects that have the capability – a challenging problem
ACLs and capabilities (cont.)

• ACL is the de-facto model for file protection.  Why?
• Capabilities are widely used in mobile systems.  Why?
• Hint: length of the list
Unix access control model

$ ll sec1.html
-rw-r--r--  1 csong  staff   3.4K Jun 2 11:31 sec1.html

• Unix uses ACLs: rights are associated with files
• Three sets of rights: owner, group, others
Owners

• Q: who gets to assign the access rights?
• A: the owner of the object
Groups

- Motivations: ACLs have a problem when objects are heavily shared the ACLs become very large
  - Hard to embed into inode
- Solution: put subject into groups (roles)
  - Administrator, PowerUser, User, Guest
  - Assign permissions to groups/roles; each user gets permission
Role-Based Access Control (RBAC)

Advantage: user’s change more frequently than roles
Unix access control rights

- Files
  - Literal: read, write content and execute
- Directories
  - Read: read file names, but not attributes
  - Write: create, rename, and delete files
  - Execute: read file attributes, list files
DAC and Root

The access control model we have discussed so far is called **Discretionary Access Control** (DAC)

- Including both ACLs and capabilities
- Root is a special user in DAC who can **override** any existing policies e.g.,
  - Change the owner/group of any files
  - Change the access rights of any files

**This is the reason why attackers are after root**
Users and processes

- **FACT:** although ACLs use *users* as subject, the OS actually treats *processes* as subjects
  - Processes act on behalf of the users, like a proxy
- **Q:** how to decide/change the identity (user id) of a process?
  - **A1:** inherited from its parent process unless
  - **A2:** changed by the process (via `setuid()` system call) or
  - **A3:** executed a setuid program
Process tree (Unix)

- The first process of the *nix system is `init`
  - Executed as root
  - Start daemons (services) and the login process
- After a successful login
  - A new shell is spawned and it changes its `uid` to the authenticated user
setuid programs

• Each process has three uids
  • **ruid**: real user id -> who starts the process
  • **euid**: effective user id -> used for access control
  • **suid**: saved user id -> so previous euid can be restored

• setuid programs
  • Once executed, changes the euid of the process to the owner of the file
  • Why is this useful?
Problems of DAC

- Root has unrestricted privileges
- Processes may be malicious
Mandatory Access Control (MAC)

- MAC = mandatory, so even root is checked against the policies

- Examples
  - Integrity levels on Windows
  - Capabilities on Linux
    - Same name, different meanings
    - Divide root's privileges into different capabilities so as to enforce the principle of least privilege
  - SELinux
Linux capabilities

For next class ...

- OS Security III: Sandbox