Announcements

- Essay should be done
- Next week, you'll get lab 1
Access Control

The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.

Probably the central element of computer security.
Incorporates

1) Authentication
2) Authorization
3) Audit

(Chapter 4) (last lecture)
Access Control Policies
(governs authorization)

1) Discretionary Access Control (DAC)
2) Mandatory Access Control (MAC)
3) Role-Based Access Control (RBAC)

(These aren't necessarily mutually exclusive.)
Terminology

- subject: a process (or user)
  - 3 classes
    - owner
    - group
    - world

- object: a resource
**Dir:** Access rights describe ways in which subjects may interact with objects.

**Ex:**
- read
- write
- execute
- delete
- create
- search
Discretionary Access Control (DAC)

- Most common in modern OS

- Based on subject's identity combined with access rules stating what each subject is allowed to do.

Note: An entity may be given access rights which allow it to glue another subject access rights.
Visualization: Access Control Matrix

<table>
<thead>
<tr>
<th>SUBJECTS</th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
<th>File 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>User A</td>
<td>Own Read Write</td>
<td>Own Read Write</td>
<td>Own Read Write</td>
<td></td>
</tr>
<tr>
<td>User B</td>
<td>Read</td>
<td>Own Read Write</td>
<td>Write</td>
<td>Read</td>
</tr>
<tr>
<td>User C</td>
<td>Read Write</td>
<td>Read</td>
<td>Own Read Write</td>
<td></td>
</tr>
</tbody>
</table>

(based on Lampson in '71, image taken from course text)
How to implement this matrix?

In practice, this matrix is very sparse.

(Think of the number of users and files on our Linux systems, much less on larger labs.)

How to solve this?
Windows OS: Access Control Lists

The good:
- Space efficient
- Fast if accessing a particular file

The bad:
- A particular list can get very long
- Listing files A has access to is slow
Flip this: Capability Lists

The good:

The bad:
Case Study: UNIX

- Files and directories in a tree structure
- Each user has a user id (uid) and at least one group id (gid)
- At creation, a file is set to its creator's uid and either its owner's gid or its parent directory's gid (if directory has setgid permissions)
- Each file gets 12 protection bits
Protection Bits (in UNIX)
- 9 bits specify read, write and execute privileges for:
  owner, group, world

eg: 111 101 000
  rwx r-- ----

7 5 0

chmod 777 myfile
- remaining bits specify additional behaviors
  setuid, setgid, "sticky"
  ⬆
govern executables in a directory, means only owner of any file can rename, move, or delete that file
when someone else executes file it runs with their user id
Security in UNIX

- the super user account (root) can do anything

How is this good and bad?

Good: someone has to fix it all

Bad: big, fat target
UNIX (cont)

Most UNIX file systems (including ours) also use ACLs.

- Admin can assign any # of uids & gids to a file using setfacl

- Any file may not have additional entries

  *If it does have ACL entries, an extra bit is set

(use getfacl to check)
demo - chmod
get fact
Mandatory Access Control (MAC)

Based on comparing security labels with security clearances.

(Evolved for military and government settings)

TS/SCI -> top secret, secret, unclassified

Mandatory — a subject with access to some resource may not share that access with another subject
Top Secret!

In the real world, you can’t just wave your gun & "hack" to get the information!
Example: Bell-LaPadula Model

- Every subject gets a security clearance
- Every object gets a security classification

2 principles

① No read up (simple security property)
   Don't want individuals to access into w/ a higher classification

② No write down (*-property)
   Should not be able to 'unclassify' anything
Many others exist (Ch. 10 in the text)

- Biba (military)
- Clark-Wilson (commercial)
- Chinese Wall (for conflict of interests in commercial applications)
Role-Based Access Control

Access rights are based on what roles the user assumes in the system, rather than the user's identity.

Example: Doctor's office Medical records

role division
Visualization:

<table>
<thead>
<tr>
<th>Roles</th>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
<th>R₄</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>U₁</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>U₂</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>U₃</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>U₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Roles may own or control other roles, as well as files + directories.
Next week:

- First lab
- Cryptography