Access control

- Access control determines access to files and processes in OS
- Old area of security, but not well understood

System security at the OS level

- Crypto is not the only possible mechanism
- Motivating example: secrecy of user files
  - In general, these files are not encrypted
  - Instead, access to users' files is controlled by the OS
    - This offers protection against other users...
    - ...but does not offer protection if the hard drive is physically compromised, or against a sys admin who may have the ability to read all files
  - Anyway, before closing “backdoors” let’s close “front doors”

Other issues...

- File confidentiality is not the only issue
  - Confidentiality of actions or temporary data
  - Should allow file sharing when called for
  - Integrity of system-wide files
  - Resource sharing (cycles, bandwidth, disk space, ...)
- These various requirements stem from the fact that modern OSs use multi-programming
  - One user’s actions can affect other users
Some terminology

- Protected entities: "objects" O, have things done to them
- Active objects: "subjects" S, do things (i.e., users/processes)
- Subjects/objects can be:
  - Files
  - Processes
  - Systems
  - Functions/variables (within a program)
  - Database entries
  - Etc.

Authorization vs. authentication

- The issues are orthogonal
- Authentication is a means of proving to the system that you are a particular user (or belong to a particular group, etc.)
- Authorization assumes that you have already been properly authenticated, and is concerned with controlling your access to objects
- We will discuss authentication later

Some possible approaches

- Physical separation
  - Different systems for different users
- Temporal separation
  - Users run processes at different times
  - Both of the above use resources poorly
- Logical separation
  - Cryptographic separation

Logical separation...

- Complete isolation
  - E.g., all users/processes unaware of any others
- "All or nothing"
  - Either a file is completely public or private
  - Or, users either aware of each other or not
- Access control
  - More fine-grained; determines access at the subject/object level
Continued...

• Dynamic access control (capabilities)
  • Access may depend on the context, or on more complicated conditions
  • Limited use
    • Access to object is limited: e.g., read but not modify
  • Level of granularity is important
    • Finer granularity more “secure” but more difficult to implement

Memory protection

• "Fence": restricts access to portions of memory
  • E.g., predefined memory address where OS resides; users disallowed from modifying
    • Can be enforced at the hardware level
    • More difficult if OS is supposed to be “modifiable”; e.g., if systems must support multiple OSs

Continued...

• Variation is to have a “fence register” which stores the address of the protected portion of memory
  • Protected portion can dynamically change
  • More opportunity for security breaches...

Continued...

• Fence registers have other advantages
  • Allow easy “relocation” by simple addition (in hardware)
  • Can have two such registers: base register and bounds register
  • Extends to allow separation of memory space for multiple users
  • Context switching also updates base/bounds registers
Further extensions

- Note that this only protects users from each other
  - Does not prevent error within one user's memory space
  - Can add additional base/bounds registers
    - I.e., one set for instructions and one for data
    - In theory, can extend this; in practice it is difficult to have more than two sets per user

Tagged architecture

- Base/bounds registers offer very coarse-grained protection
  - Also, have the restriction that different sections of memory space must be contiguous
  - Possible to improve this by tagging every, e.g., word of memory via protected op.
    - Can be wasteful of bits...
    - Requires changes at the hardware level

Segmentation

- Program components divided into logical segments (e.g., code of a single procedure)
  - Each segment has a unique name; items within segment addressed by (name, offset)
- Each segment stored anywhere in memory
  - OS handles mapping; transparent to user
  - Can implement protection for each segment
- OS controls which programs have which entries in their segment address tables

Drawbacks of segmentation?

- Users can guess memory locations
  - Can imagine fixing this, but this is inefficient
- Users can generate (name, offset) where the offset is larger than the segment size
- Memory fragmentation
  - Address table lookup can be slow
Paging

- Similar to segmentation, but with fixed-size segments called pages
  - Addressing via (page, offset)
- Avoids fragmentation problem...as well as "large offset" issue
- Inefficient as program grows, since pages cannot be dynamically resized
  - "Re-paging" also causes potential security problems as data is shifted from one page to another
- Also can be difficult to describe desired protection, since pages are no longer logical units

Best of both worlds?

- Can combine paging with segmentation
  - Logical units, each broken into same-size pages