Lecture 10
General Object protection
Its more than Memory now...

- Since computers have evolved to provide sophisticated multiuser-multiprocess capabilities the number and types of things that are shared has increased
  - Memory
  - Files
  - Executing code
  - Hardware devices
  - Authentication information
Memory is Easy

• Generally speaking memory is rather easy to control and protect
  – All requests for it come through a single point of access
  – Access point is controlled by a central authority (Operating system)
General Object may not be...

- If we talk about protecting “objects” in general the problem can be much more difficult.
  - Access might be available through a much larger number of access points
  - Centralized control may be lacking (or impossible?)
  - Control may require finer granularity or difference concepts than simple read/write and execute.
Goals for Protecting Objects

• Check Every Access
  – We may want to override previous authorizations
  – So every access by a user to an object should be checked

• Enforce Least Privilege
  – Grant access to ONLY those objects need to accomplish the authorized task.
• Verify Acceptable Usage
  – Authorization is a “yes” or “no” decision
  – Also check that the activity requested is allowed
  – Example: A user is granted authorized access to a stack. They should only be allowed perform push and pop operations.
Directories

- A simple method of protecting objects is with directories
  - Objects are owned by users.
  - A list of rights is maintained for each user.
Benefits of Directories

- Simple to implement
- Can become too large when many objects need to be shared.
- Revocation of rights can be difficult.
  - If owner A passes on rights to B; then B may continue to have rights even if A's rights are removed.
  - May require complicated dependency information
  - May require inspection/modification of all directories entries
• Namespaces and scope may be difficult.
• If A and B both have a file named “file” and want to give access rights to C then C's directory entries have to account for complications involving namespace. A.file and B.file have to be possible and distinguishable.
• Too many problems; simple directories are probably not sufficient for most objects.
Access Control Lists

• Every object is associated with a list of permissable users/usage.

• Usage of wildcards can yield very small, yet effective, lists of rights.
Directories function by maintaining a list of objects accessible by each subject.

ACLs function by maintaining a list of authorized subjects for each object.

Which is better depends on the circumstances you are dealing with. One is more efficient than the other in certain cases.
Access Control Matrix

• An alternative to Directories and ACLs is to combine the two concepts into an Access Control Matrix.
  – Rows represent a subject and the objects accessible to them.
  – Columns represent Objects and the permissions granted to each user on that object.

• Seldom used. Advantages of both directories and ACLs but horrible in terms of space efficiency.
Capability

- If Directories and ACLs are difficult to maintain then maybe it would be easier to place the burden of authentication on the user instead of the operating system.

- A capability is unforgeable token that gives the possessor some sort of rights to an object.
  - Randomized/timestamped cookies

- Umm... Unforgeable??...
Forgeability

- It is difficult to prevent tokens from being forged.
- A solution is to have the operating system hold all tokens instead of allowing the user to hold the token.
- Tokens can be encrypted using a secret known only to the operating system.
- Usually capabilities are backed up by ACLs or Directories and used as a sort of rights “cache”
Procedure Oriented Control

• So far so good as far as access rights are concerned.

• But we also want to be able to control “what” is done not just by whom.

• Procedure Oriented Access Control can be used to provide this concept.

• But you already know this... Its called Abstraction in other areas...
• We may want to grant access to users to insert and delete things from a balanced binary tree but we want to limit what the user does to the tree (so it remains balanced)

• Procedure-Oriented control specifies a restricted set of interfaces (public methods) that can be used.

• So we can basically wrap our object with a minimal set of access procedures and control what is done to them, not just by who.
Different types of objects...

- Different object types may require different, specialized access controls...
- File protection mechanism for examples
  - Used frequently
  - Heavily shared
  - Primary source of sensitive data
- Many different solutions have been specifically designed for file protection...
All or None protection

- Original IBM OS: “System Designers supposed that users could be trusted not to read or modify others' files, because the users would expect the same respect from others.”
- Users could only access files by knowing name
- But some files were recognized as sensitive and so could be protected by a password. (read/write or delete individually.)
- Had problems... (of course)
- Lack of Trust (assumption of trustworthy users)
- All or nothing (subsets of users could not be made)
- Rise of timesharing: frequent usage was impossible because each usage of protected files required operator intervention
- Complexity (efficiency) because the human intervention is so time consuming this type of protection is reserved for only the most sensitive data.
- File Listings: Users aren't ignorant of file presence.
Group Protection

- Authorized users are separated into groups.
- Overcomes many of the problems of All-None
- But introduces more problems
  - Group affiliation: users cannot belong to more than one group (or group overlaps have to be dealt with)
  - Multiple personalities: users can obtain more than one account in order to belong to more than one group
- All groups: It's the user's burden to decide what group an object/file belongs to. Users make mistakes.

- Limited Sharing: Only provides a course granularity of access control.

• Despite these problems, this system is still in use today in most Unix-like operating systems.

- They are easy to implement.

- Require few resources.

- Easy to maintain and understand.
• Single Permissions (Passwords or other tokens)

• Objects a protected by a single token.

• Sharing is done by sharing the token.
  – Loss: oops.
  – Use: ja—eeeeeze every single time I want to use it...
  – Disclosure: Now we have to kill you.
  – Revocation: Getting rid of the unwanted party guest means we have to move the whole party without telling them.
• SetUID: A way of granting temporary rights to an object

• Per-Object and Per-User protection
  – Similar to ACLs
  – Windows 2000 are a good implementation of these. (Did we just use good with a Micro$oft product???)
User Authentication

• Another common protection area

• User Authentication can be based on:
  – Something the user knows: passwords
  – Something the user has: keys, security cards
  – Something the user is: biometrics
Passwords (knows)

- We are familiar with passwords
  - Can be learned by unauthorized people.
  - Possible Insecure for a variety of reasons
    - Obsolete systems (Unix passwords < 8 characters)
    - Probability: spot, fido, ...
      - (trying all english words takes about 80 seconds)
    - If we check all combinations of six letters or less and all english words we typically account for 86% of all passwords. (Yikes!)
• Plaintext system password lists
  – Need to be heavily protected
  – Would generally require very special
    implementation in the operating system

• Encrypted system password lists
  – Can be in plain view
  – Need to prevent 1 to 1 mapping so that one user
    cannot accidentally discover another user's
    password.
• One time passwords:
  – Change everytime they are used or on a time rotation basis.
  – Usually presented as credit card like devices
  – Similar to authentication based on something a user “has”.
• Keys, security cards and biometrics, oh my.
  – Either have problems in terms of loss or duplication, or
  – Difficult to measure accurately. Do not have rigorous basis in math or accuracy.