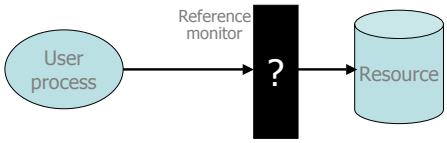
Protection: ACLs, Capabilities, and More

We've seen:

- Some cryptographic techniques
  - Encryption, hashing, types of keys, . . .
- Some kinds of attacks
  - Viruses, worms, DoS, . . .
- And a distributed authorization and authentication system
  - Kerberos
- Now lets look at access controls and other forms of protection

Access control

- Common Assumption
  - System knows who the user is
    - User has entered a name and password, or other info
  - Access requests pass through gatekeeper
    - Global property; OS must be designed so that this is true



Decide whether user can apply operation to resource

Principle of Least Privilege

- Guiding principle in security
- Programs, users and systems should be given enough privileges to perform their tasks, and no more
- Impossible to get "just right"
- Commercial systems tend to err on the side of too much privilege
  - UNIX, the Internet, . . .
  - Though this is finally changing

What makes it hard?

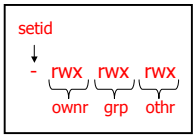
- Real (least privilege) policies are complex…a lot of work to set them
  - Here's a new file…Mary and George can see it, George may need to modify it, but Alex and Bill have no reason to see it, . . .
- Least privilege changes over time
  - A user setting his password needs to modify the password file at a brief moment in time

Separation of Policy and Mechanism

- This is another guiding principle
- Related to design and implementation of a security system:
  - Mechanisms should be simple and generic
  - And support a wide range of policies

An example: Unix file security

- Each file has a single owner and group
- Each user can belong to multiple groups
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
  - Represented by vector of four octal values



- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discuss in a few slides

Unix slides stolen from John Mitchell, Stanford

Effective user id (EUID)

- Each process has three Ids (+ more under Linux)
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

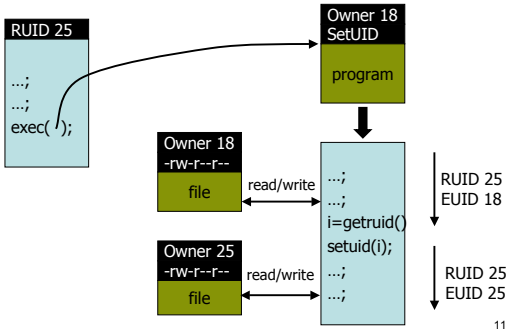
Process Operations and IDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system calls
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0
- Details are actually more complicated
  - Several different calls: setuid, seteuid, setreuid

Setid bits on executable Unix file

- Three setid bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory

Example



Setuid programming

- Be Careful!
  - Root can do anything; don't get tricked
  - Principle of least privilege – change EUID when root privileges no longer needed
- Setuid scripts
  - This is a bad idea
  - Historically, race conditions
    - Begin executing setuid program; change contents of program before it loads and is executed

Unix: separation of mechanism and policy?

- Probably not enough
- Mechanism of root forces “root-style” policy
- Mechanism of owner forces “owner-style” policy
  - A form of Discretionary Access Control (DAC)
    - User controls access at his/her discretion
  - Not Mandatory Access Control (MAC)
    - Administrator controls access
- Though certainly some flexibility (many groups, chown, etc.)

Unix summary

- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main bad thing
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

Access Matrix

- Better separation of mechanism and policy
  - Lampson
- View protection 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 $Object_j$

Access Matrix

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure A

Use of Access Matrix

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

Use of Access Matrix (Cont.)

- Access matrix design separates mechanism from policy.
  - Mechanism
    - Operating system provides access-matrix + rules.
    - If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
  - Policy
    - User dictates policy.
    - Who can access what object and in what mode.

Concept generalizes to switching domains (setuid-like)

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure B

Copy Rights

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Owner Rights

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Control Rights

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Simplify with roles, groups, and hierarchy

- Big matrix is hard to configure
- Roles/groups:
  - Domain is a role or group, rather than a user
  - Assign users to roles
  - Administrator, PowerUser, User, Guest
- Roles can be hierarchical
  - Higher role has all rights of lower roles
- Hierarchy in directory structure
  - If user has read access to directory, user has read access to every file in directory

Two implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - Allow user to hold a "ticket" for each resource
  - Roughly: store row of matrix with the user

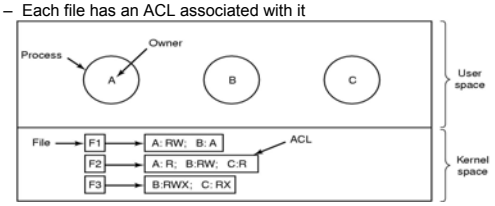
	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
User m	read	write	write

Access control lists are widely used, often with groups

Some aspects of capability concept are used in Kerberos, …

Access Control Lists

- Example: to control file access
  - Each file has an ACL associated with it



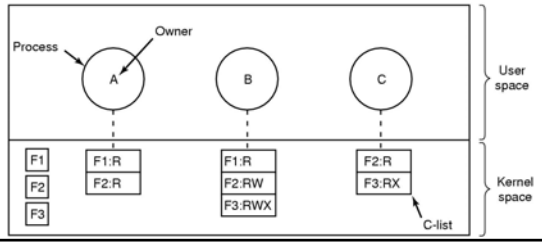
File	Access control list
Password	tana, sysadm: RW
Pigeon_data	bill, pigfan: RW; tana, pigfan: RW; ...

ACLs Discussion

- Need good data structures
- User will need to have multiple identities
- Need defaults for new objects
- Good security metaphors to users are needed!

Capabilities

- Store information by rows
  - For each subject, there is list of objects that it can access
  - Called a capability list of c-list; individual items are capabilities
    - C-lists are objects too, and may be pointed to from other c-lists



Capabilities

- To access an object, subject presents the capability
  - 'capability' word coined by Dennis and Van Horn in 1966
  - Capability is (x, r) list. x is object and r is set of rights
  - Capabilities are transferable
- How to name an object?
  - Is start address sufficient?
    - Array and first element of array have same address
  - Is start address + length of object sufficient?
    - What if start address changes?
  - Random bit string: use hash table to translate from name to bits
- Need to protect capabilities from being forged by others
  - ACLs were inherently unforgeable

Protecting Capabilities

- Prevent users from tampering with capabilities
- Tagged Architecture
  - Each memory word has extra bit indicating that it is a capability
  - These bits can only be modified in kernel mode
  - Cannot be used for arithmetic, etc.
- Sparse name space implementation
  - Kernel stores capability as *object+rights+random number*
  - Give copy of capability to the user; user can transfer rights
  - Relies on inability of user to guess the random number
    - Need a good random number generator

Capability Revocation

- Kernel based implementation
  - Kernel keeps track of all capabilities; invalidates on revocation
- Object keeps track of revocation list
  - Difficult to implement
- Timeout the capabilities
  - How long should the expiration timer be?
- Revocation by indirection
  - Grant access to object by creating alias; give capability to alias
  - Difficult to review all capabilities
- Revocation with conditional capabilities
  - Object has state called "big bag"
  - Access only if capability's little bag has sth. in object's big bag

Comparing ACLs & Capabilities

- Number of comparisons on opening a file?
  - Capability: just one ACLs: linear with number of subjects
- Implementing when no groups are supported:
  - Capabilities: easier ACLs: Need to enumerate all the subjects
- Finding out who has access to an object?
  - Capabilities: difficult
- Is it possible to control propagation of rights?
  - Capabilities: some counter can be used
- Selective revocation of rights:
  - Easy for ACLs (no immediate effect); difficult for capabilities
- Easier propagation of rights for capabilities

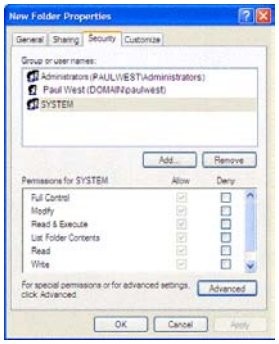
Access Control in Windows (NTFS)

- Some basic ideas similar to Unix, but:
  - Can associate many users and groups with objects
  - Richer set of operations:
    - Read, write, execute, delete, change owner, change permission
    - These come packaged as: Read, Write, Read and Execute, Modify, Full Control
- Some additional concepts
  - Tokens
  - Security attributes
  - These can be changed to "impersonate" another user (analogous to setuid)

NTFS slides stolen from John Mitchell, Stanford

Sample permission options

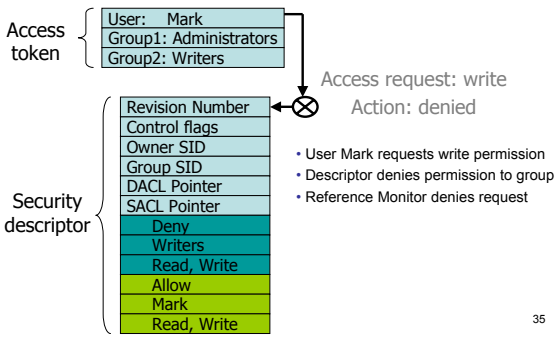
- SID
  - Identity (replaces UID)
    - SID revision number
    - 48-bit authority value
    - variable number of Relative Identifiers (RIDs), for uniqueness
  - Users, groups, computers, domains, domain members all have SIDs



Security Descriptor

- Information associated with an object
  - who can perform what actions on the object
- Several fields
  - Header
    - Descriptor revision number
    - Control flags, attributes of the descriptor
      - E.g., memory layout of the descriptor
  - SID of the object's owner
  - SID of the primary group of the object
  - Two attached optional lists:
    - Discretionary Access Control List (DACL) – users, groups, …
    - System Access Control List (SACL) – system logs, ..

Example access request



- User Mark requests write permission
- Descriptor denies permission to group
- Reference Monitor denies request

Trusted Systems

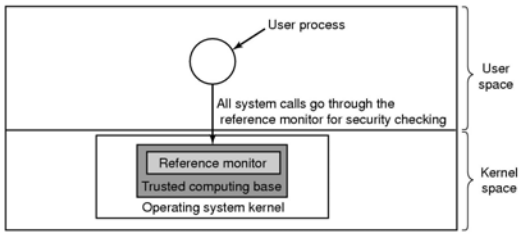
- The computer world right now is full of security problems
- Can we build a secure computer system?
  - Yes! (Well, more-or-less)
- Then why has it not been built yet?
  - Users unwilling to throw out existing systems
  - Features and performance (almost) always trumps security, so:
    - more complexity, code, bugs and security errors
- Examples: e-mail (from ASCII to Word), web (applets)
- Trusted Systems: formally stated security requirements, and how they are met

Trusted Computing Base

- Heart of every trusted system has a small TCB
  - Hardware and software necessary for enforcing all security rules
  - Typically has:
    - most hardware,
    - Portion of OS kernel, and
    - most or all programs with superuser power
- Desirable features include:
  - Should be small
  - Should be separable and well defined
  - Easy to audit independently

Reference Monitor

- Critical component of the TCB
  - All sensitive operations go through the reference monitor
  - Monitor decides if the operation should proceed
  - Some of this starting to appear in consumer machines
    - TPM chips, Security Enhanced Linux (SELinux)

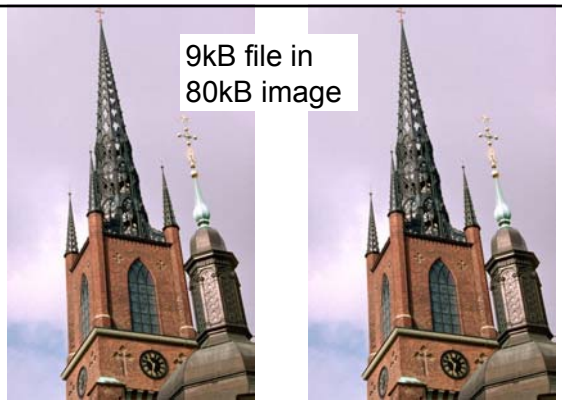


Covert Channels

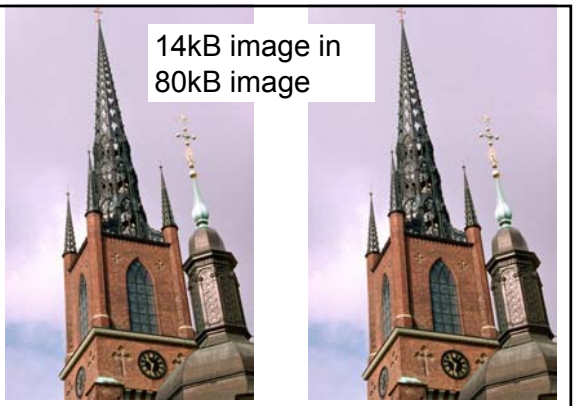
- Do these ideas make our system completely secure?
  - No. Security leaks possible even in a system proved secure mathematically. Lampson 1973
- Model: 3 processes. The client, server and collaborator
  - Server and collaborator collude
  - Goal: design a system where it is impossible for server to leak to the collaborator info received from the client (Confinement)
- Solution: Access Matrix prevents server to write to a file that collaborator has read access; no IPC either
- Covert Channel: compute hard for 1, sleep for a 0

Steganography

- Original picture 1024x768
- Using lower order RGB bits: 1024x768x3 = 294,912 bytes
- Five Shakespeare plays total 734,891 bytes:
  - Hamlet, King Lear, Julius Caesar, The Merchant of Venice, Macbeth
  - Compress to: 274 KB, and then encode



http://www.elec.reading.ac.uk/people/J.Grimbleby/Stego.htm



http://www.elec.reading.ac.uk/people/J.Grimbleby/Stego.htm

Orange Book

- Dept. of Defense Standards DoD 5200.28 in 1985
  - Known as Orange Book for the color of its cover
- Divides OSes into categories based on security property
  - **D** – Minimal security.
  - **C** – Provides discretionary protection through auditing. Divided into **C1** and **C2**. **C1** identifies cooperating users with the same level of protection (Unix). **C2** allows user-level access control (Windows NT 4.0).
  - **B** – All the properties of **C**, however each object may have unique sensitivity labels. Divided into **B1**, **B2**, and **B3**.
  - **A** – Uses formal design and verification techniques to ensure security.