Operating System Security

John Mitchell

Operating System Functions

- OS is a resource allocator
  - Manages resources, decides between conflicting requests
- OS is a control program
  - Controls execution of programs to prevent errors and improper use of the system

Security issues

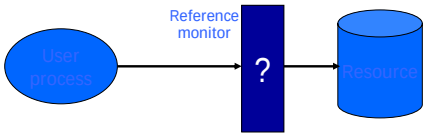
- Isolation
  - Separate processes execute in separate memory space
  - Process can only manipulate allocated pages
- Access control
  - When can process create or access a file?
  - Create or read/write to socket?
  - Make a specific system call?
- Protection problem
  - Ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Comparison between different operating systems
  - Compare protection models: which model supports least privilege most effectively?
  - Which system best enforces its protection model?

Outline

- Access Control Concepts
  - Matrix, ACL, Capabilities
  - Multi-level security (MLS)
- OS Mechanisms
  - Multics
    - Ring structure
  - Amoeba
    - Distributed, capabilities
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS
  - SE Linux
    - Role-based, Domain type enforcement
- Topics for next lecture
  - Secure OS
    - Methods for resisting stronger attacks
  - Cryptographic file systems
  - Assurance
    - Orange Book, TCSEC
    - Common Criteria
    - Windows 2000 certification
  - Some Limitations
    - Information flow
    - Covert channels

Access control

- Context
  - System knows who the user is
    - User has entered a name and password, or other info
  - Access requests pass through gatekeeper
    - OS must be designed so monitor cannot be bypassed



Decide whether user can apply operation to resource

Access control matrix [Lampson]

Objects

Subjects

	File 1	File 2	File 3	...	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
...					
User m	read	write	read	write	read

Two implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a "ticket" for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

	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, ...

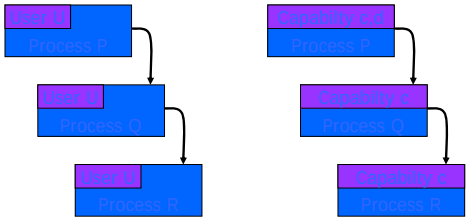
Capabilities

- Operating system concept
  - "... of the future and always will be ..."
- Examples
  - Dennis and van Horn, MIT PDP-1 Timesharing
  - Hydra, StarOS, Intel iAPX 432, Eros, ...
  - Amoeba: distributed, unforgeable tickets
- References
  - Henry Levy, Capability-based Computer Systems
    http://www.cs.washington.edu/homes/levy/capabook/
  - Tanenbaum, Amoeba papers

ACL vs Capabilities

- Access control list
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user
- Capabilities
  - Capability is unforgeable ticket
    - Random bit sequence, or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identify of user/process

ACL vs Capabilities

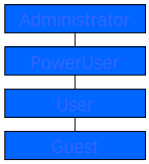


ACL vs Capabilities

- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some systems if appropriate bookkeeping
      - OS knows what data is capability
      - If capability is used for multiple resources, have to revoke all or none ...
      - Other details ...

Roles (also called Groups)

- Role = set of users
  - Administrator, PowerUser, User, Guest
  - Assign permissions to roles; each user gets permission
- Role hierarchy
  - Partial order of roles
  - Each role gets permissions of roles below
  - List only new permissions given to each role



Groups for resources, rights

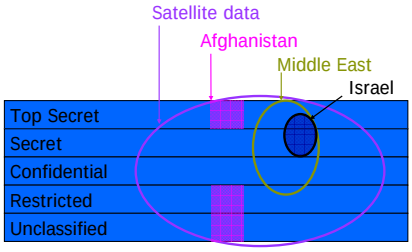
- Permission = ⟨right, resource⟩
- Permission hierarchies
  - If user has right r, and r>s, then user has right s
  - If user has read access to directory, user has read access to every file in directory
- Big problem in access control
  - Complex mechanisms require complex input
  - Difficult to configure and maintain
  - Roles, other organizing ideas try to simplify problem

Multi-Level Security (MLS) Concepts

- Military security policy
  - Classification involves sensitivity levels, compartments
  - Do not let classified information leak to unclassified files
- Group individuals and resources
  - Use some form of hierarchy to organize policy
- Other policy concepts
  - Separation of duty
  - "Chinese Wall" Policy

Military security policy

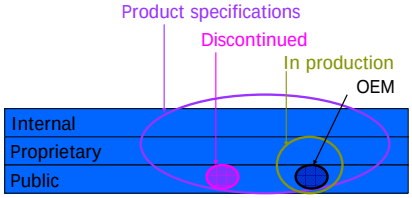
- Sensitivity levels
- Compartments



Military security policy

- Classification of personnel and data
  - Class = ⟨rank, compartment⟩
- Dominance relation
  - $D_1 \leq D_2$ iff $rank_1 \leq rank_2$ and $compartment_1 \subseteq compartment_2$
  - Example: ⟨Restricted, Israel⟩ ≤ ⟨Secret, Middle East⟩
- Applies to
  - Subjects – users or processes
  - Objects – documents or resources

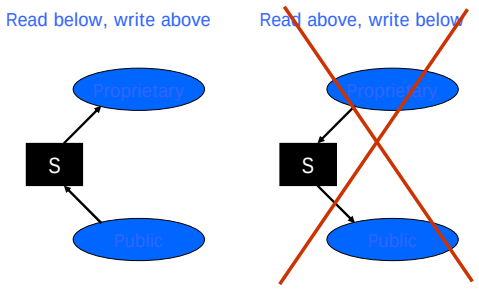
Commercial version



Bell-LaPadula Confidentiality Model

- When is it OK to release information?
- Two Properties (with silly names)
  - Simple security property
    - A subject S may read object O only if C(O) ≤ C(S)
  - *-Property
    - A subject S with read access to O may write object P only if C(O) ≤ C(P)
- In words,
  - You may only *read below* your classification and only *write above* your classification

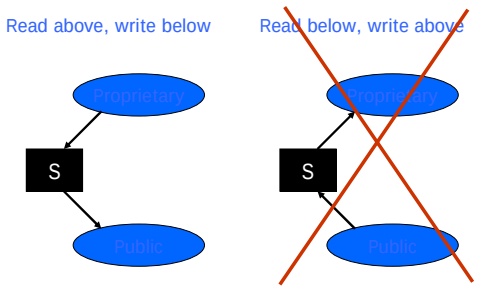
Picture: Confidentiality



Biba Integrity Model

- Rules that preserve integrity of information
- Two Properties (with silly names)
  - Simple integrity property
    - A subject S may write object O only if C(S) ≥ C(O)
      (Only trust S to modify O if S has higher rank ...)
  - *-Property
    - A subject S with read access to O may write object P only if C(O) ≥ C(P)
      (Only move info from O to P if O is more trusted than P)
- In words,
  - You may only *write below* your classification and only *read above* your classification

Picture: Integrity



Problem: Models appear contradictory

- Bell-LaPadula Confidentiality
  - Read down, write up
- Biba Integrity
  - Read up, write down
- Want both confidentiality and integrity
  - Contradiction is partly an illusion
  - May use Bell-LaPadula for some classification of personnel and data, Biba for another
    - Otherwise, only way to satisfy both models is only allow read and write at same classification

In reality: Bell-LaPadula used more than Biba model, e.g., Common Criteria

Other policy concepts

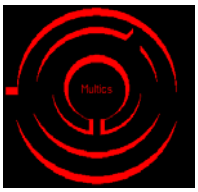
- Separation of duty
  - If amount is over $10,000, check is only valid if signed by two authorized people
  - Two people must be *different*
  - Policy involves role membership and ≠
- Chinese Wall Policy
  - Lawyers L1, L2 in Firm F are experts in banking
  - If bank B1 sues bank B2,
    - L1 and L2 can each work for either B1 or B2
    - No lawyer can work for opposite sides in any case
  - Permission depends on use of other permissions

These policies cannot be represented using access matrix

Example OS Mechanisms

- Multics
- Amoeba
- Unix
- Windows
- SE Linux (briefly)

Multics



- Operating System
  - Designed 1964-1967
    - MIT Project MAC, Bell Labs, GE
  - At peak, ~100 Multics sites
  - Last system, Canadian Department of Defense, Nova Scotia, shut down October, 2000
- Extensive Security Mechanisms
  - Influenced many subsequent systems

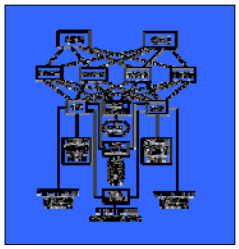
http://www.multicians.org/security.html

E.I. Organick, The Multics System: An Examination of Its Structure, MIT Press, 1972

Multics time period

F.J. Corbato

- Timesharing was new concept
  - Serve Boston area with one 386-based PC



Multics Innovations

- Segmented, Virtual memory
  - Hardware translates virtual address to real address
- High-level language implementation
  - Written in PL/1, only small part in assembly lang
- Shared memory multiprocessor
  - Multiple CPUs share same physical memory
- Relational database
  - Multics Relational Data Store (MRDS) in 1978
- Security
  - Designed to be secure from the beginning
  - First B2 security rating (1980s), only one for years

Multics Access Model

- Ring structure
  - A ring is a domain in which a process executes
  - Numbered 0, 1, 2, ... ; Kernel is ring 0
  - Graduated privileges
    - Processes at ring i have privileges of every ring j > i
- Segments
  - Each data area or procedure is called a segment
  - Segment protection ⟨b1, b2, b3⟩ with b1 ≤ b2 ≤ b3
    - Process/data can be accessed from rings b1 ... b2
    - A process from rings b2 ... b3 can only call segment at restricted entry points

Multics process

- Multiple segments
  - Segments are dynamically linked
  - Linking process uses file system to find segment
  - A segment may be shared by several processes
- Multiple rings
  - Procedure, data segments each in specific ring
  - Access depends on two mechanisms
    - Per-Segment Access Control
      - File author specifies the users that have access to it
    - Concentric Rings of Protection
      - Call or read/write segments in outer rings
      - To access inner ring, go through a "gatekeeper"
- Interprocess communication through "channels"

Amoeba

- Distributed system
  - Multiple processors, connected by network
  - Process on A can start a new process on B
  - Location of processes designed to be transparent
- Capability-based system
  - Each object resides on server
  - Invoke operation through message to server
    - Send message with capability and parameters
    - Sever uses object # to indentify object
    - Sever checks rights field to see if operation is allowed
    - Check field prevents processes from forging capabilities

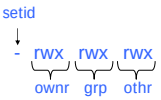
Capabilities

- Owner capability
  - When server creates object, returns owner cap.
    - All rights bits are set to 1 (= allow operation)
    - Check field contains 48-bit rand number stored by server
- Derived capability
  - Owner can set some rights bits to 0
  - Calculate new check field
    - XOR rights field with random number from check field
    - Apply one-way function to calculate new check field
  - Server can verify rights and check field
    - Without owner capability, cannot forge derived capability

Protection by user-process at server; no special OS support needed

Unix file security

- Each file has owner and group
- 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

Question

- Owner can have fewer privileges than other
  - What happens?
    - Owner gets access?
    - Owner does not?
- Prioritized resolution of differences

  if user = owner then *owner* permission
  else if user in group then *group* permission
  else *other* permission

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

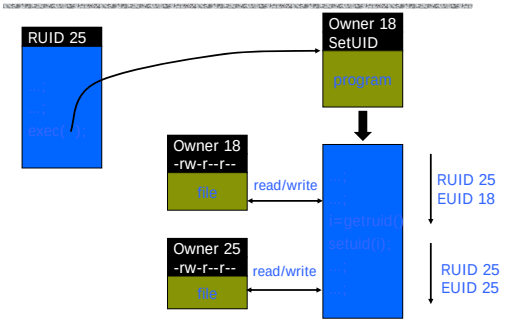
Process Operations and IDs

- 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

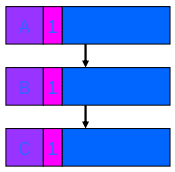
- Three setid bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove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

Example



Compare to stack inspection

- Careful with Setuid !
  - Can do anything that owner of file is allowed to do
  - Be sure not to
    - Take action for untrusted user
    - Return secret data to untrusted user



Note: anything possible if root; no middle ground between user and root

Setuid programming

- We talked about this before …
- 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 summary

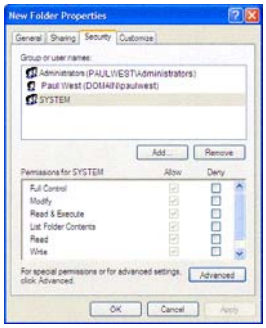
- Many of you may be used to this …
  - So probably seems pretty good
  - We overlook ways it might be better
- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main bad thing
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

Access control in Windows (NTFS)

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
- Some additional concepts
  - Tokens
  - Security attributes
- Generally
  - More flexibility than Unix
    - Can define new permissions
    - Can give some but not all administrator privileges

Sample permission options

- Security ID (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



Permission Inheritance

- Static permission inheritance (Win NT)
  - Initially, subfolders inherit permissions of folder
  - Folder, subfolder changed independently
  - *Replace Permissions on Subdirectories* command
    - Eliminates any differences in permissions
- Dynamic permission inheritance (Win 2000)
  - Child inherits parent permission, remains linked
  - Parent changes are inherited, except explicit settings
  - Inherited and explicitly-set permissions may conflict
    - Resolution rules
      - Positive permissions are additive
      - Negative permission (deny access) takes priority

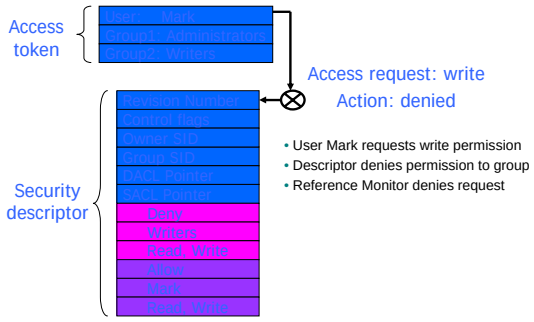
Tokens

- Security Reference Monitor
  - uses tokens to identify the security context of a process or thread
- Security context
  - privileges, accounts, and groups associated with the process or thread
- Impersonation token
  - thread uses temporarily to adopt a different security context, usually of another user

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



Impersonation Tokens (setuid?)

- Process uses security attributes of another
  - Client passes impersonation token to server
- Client specifies impersonation level of server
  - Anonymous
    - Token has no information about the client
  - Identification
    - server obtain the SIDs of client and client's privileges, but server cannot impersonate the client
  - Impersonation
    - server identify and impersonate the client
  - Delegation
    - lets server impersonate client on local, remote systems

SELinux Security Policy Abstractions

- Type enforcement
  - Each process has an associated domain
  - Each object has an associated type
  - Configuration files specify
    - How domains are allowed to access types
    - Allowable interactions and transitions between domains
- Role-based access control
  - Each process has an associated role
    - Separate system and user processes
  - configuration files specify
    - Set of domains that may be entered by each role

Outline

- Access Control Concepts
  - Matrix, ACL, Capabilities
  - Multi-level security (MLS)
- OS Mechanisms
  - Multics
    - Ring structure
  - Amoeba
    - Distributed, capabilities
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS
  - SE Linux
    - Role-based, Domain type enforcement
- Topics for next lecture
  - Secure OS
    - Methods for resisting stronger attacks
  - Cryptographic file systems
  - Assurance
    - Orange Book, TCSEC
    - Common Criteria
    - Windows 2000 certification
  - Some Limitations
    - Information flow
    - Covert channels