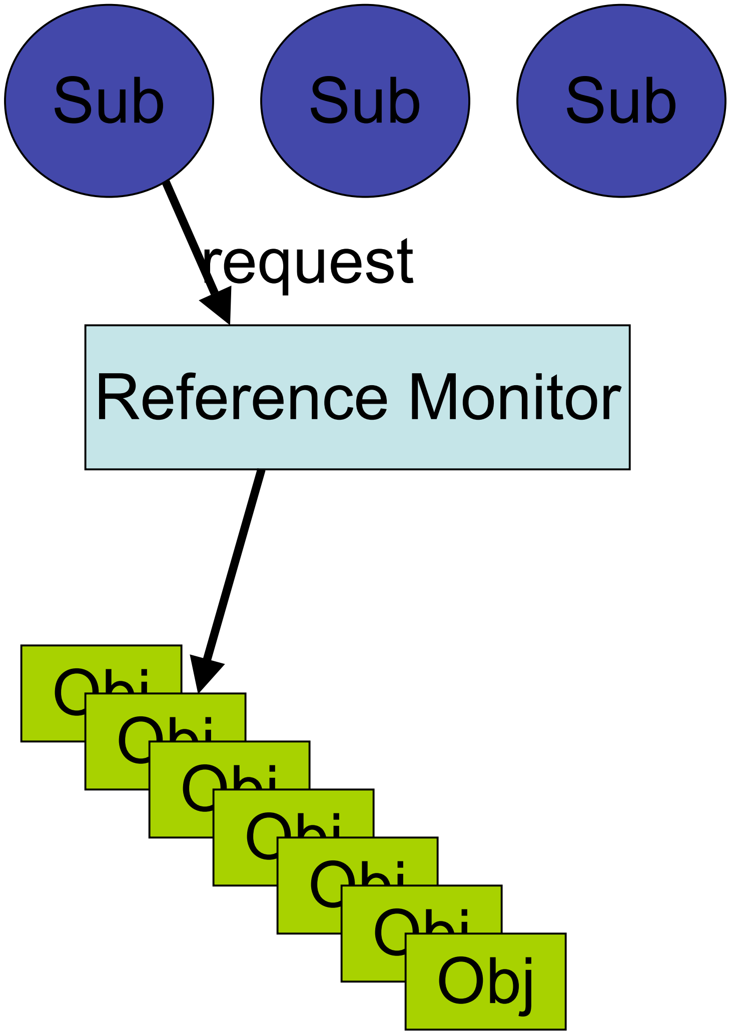
CIS 551 / TCOM 401

Computer and Network Security

Spring 2005
Lecture 4

Access Control: The Big Picture

- Objects - resources being protected
  - E.g. files, devices, etc.
- Subjects - active entities
  - E.g. processes, machines
- Permissions (or Rights) - Kinds of access requests that are monitored
  - E.g. read, write, execute
- Reference monitor - mediates requests made by subjects
  - Permits or denies access



Outline

- Access Control Concepts
  - Matrix, ACL, Capabilities
- OS Mechanisms
  - Multics
    - Ring structure
  - Amoeba
    - Distributed, capabilities
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS

- Next time:

  Java / C# access control
  - Privileges & Stack inspection

- Some slides courtesy of John Mitchell

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

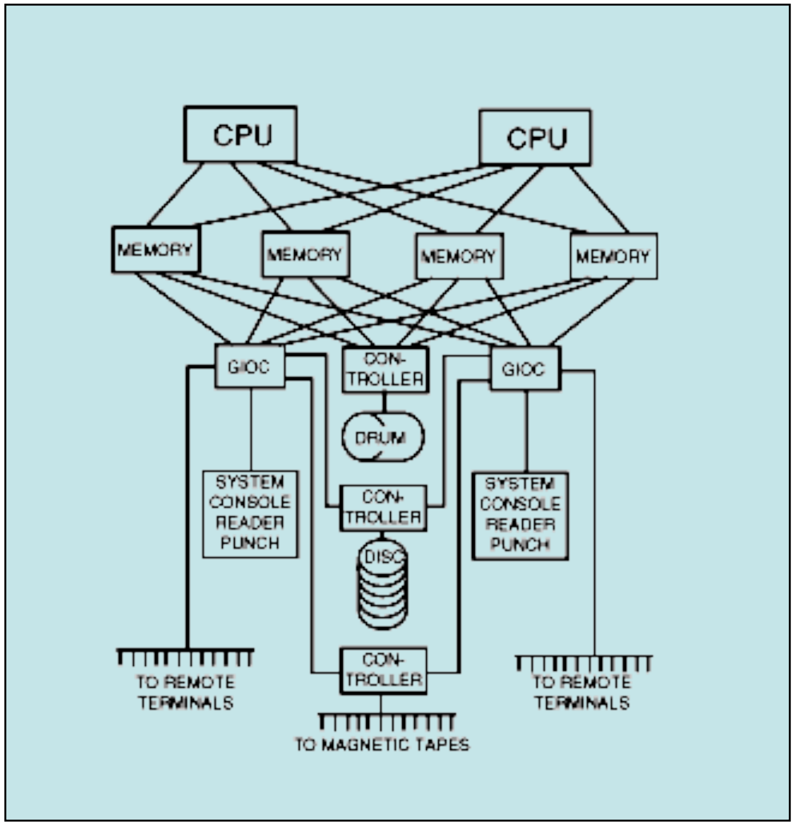
http://www.multicians.org/security.html

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

Multics time period

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

F.J. Corbato



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 language
- 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
    - More about government certification levels later

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 processes

- 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

- http://www.cs.vu.nl/pub/amoeba/amoeba.html
- 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

Server port	Obj #	Rights	Check field

Capabilities

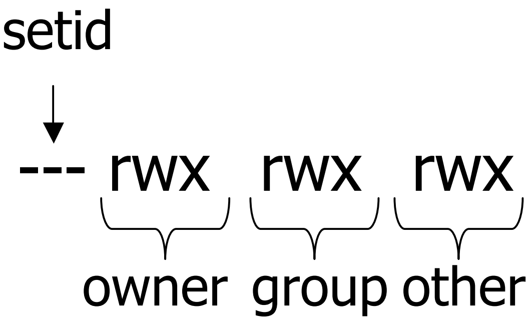
- Owner capability
  - When server creates object, returns owner cap.
    - All rights bits are set to 1 (= allow operation)
    - Check field contains 48-bit rand number stored by server
- Derived capability
  - Owner can set some rights bits to 0
  - Calculate new check field
    - XOR rights field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

Server port	Obj #	Rights	Check field

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



setid

--- rwx rwx rwx

owner group other

- 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?
    - User gets access?
    - User 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 user 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

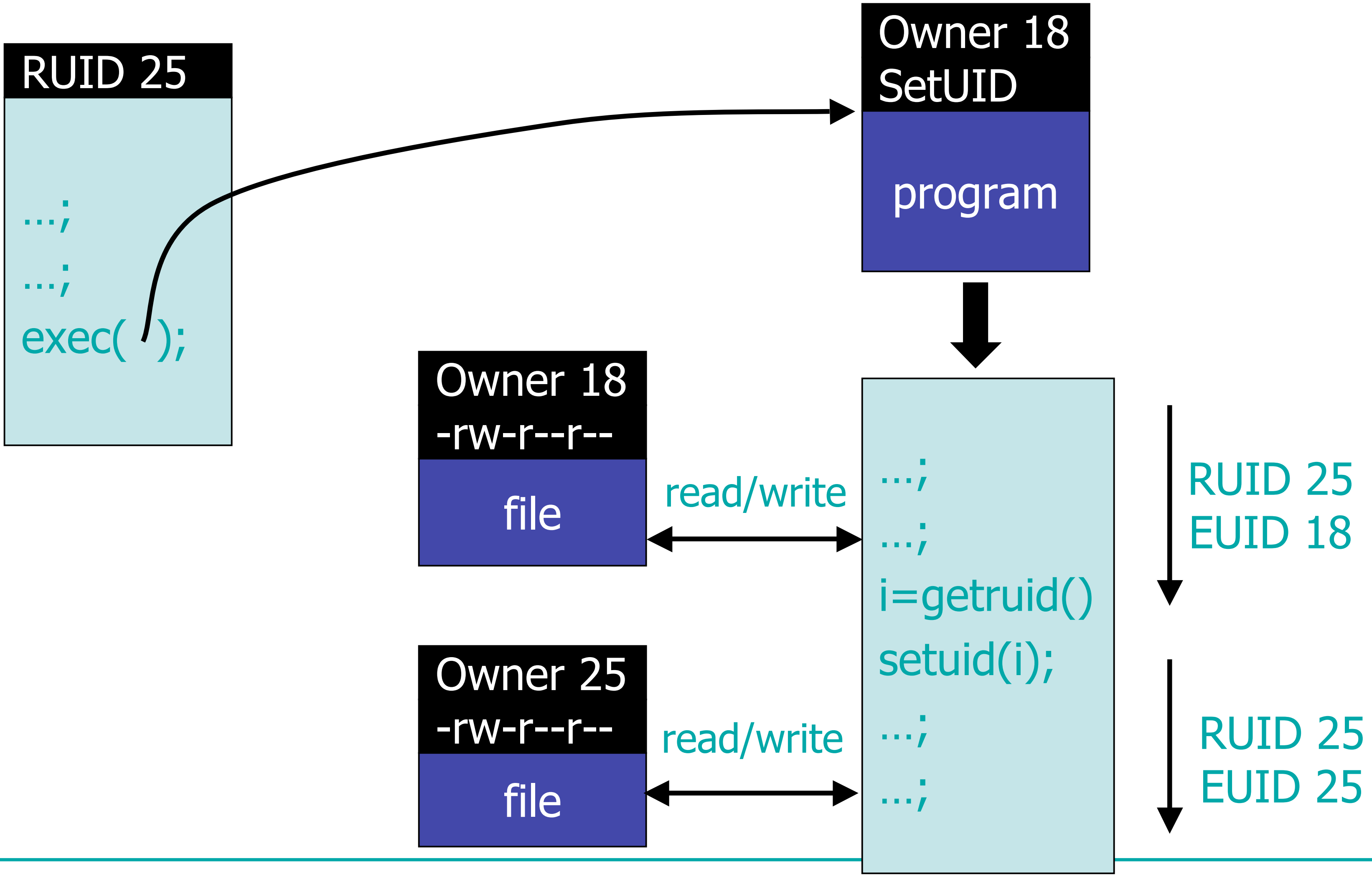
Process Operations and IDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except when executing a file with setuid bit on.
- 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, setruid

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



RUID 25

…;

…;

exec( );

Owner 18
SetUID

program

Owner 18
-rw-r--r--

file

read/write

Owner 25
-rw-r--r--

file

read/write

…;

…;

i=getruid()

setuid(i);

…;

…;

RUID 25
EUID 18

RUID 25
EUID 25

Setuid programming

- Can do anything that owner of file is allowed to do
- Be Careful!
  - Root can do anything; don’t get tricked (no middle ground)
  - Principle of least privilege – change EUID when root privileges no longer needed
  - Be sure not to
    - Take action for untrusted user
    - Return secret data to untrusted user
- 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

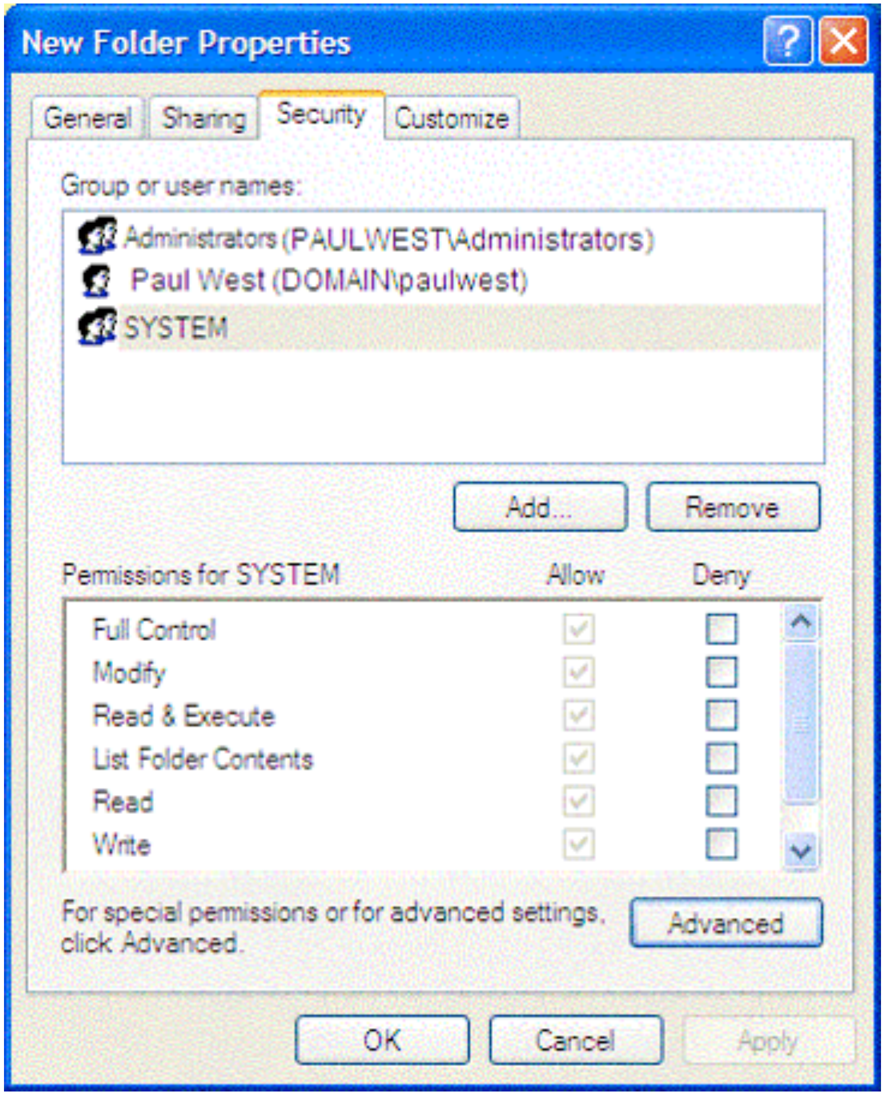
- We’re all very used to this …
  - So probably seems pretty good
  - We overlook ways it might be better
- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main bad thing
  - Too tempting to use root privileges
  - No way to assume some root privileges without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

- 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



Group or user names:

Administrators (PAULWEST\Administrators)
Paul West (DOMAIN\paulwest)
SYSTEM

Permissions for SYSTEM	Allow	Deny
Full Control	☑	□
Modify	☑	□
Read & Execute	☑	□
List Folder Contents	☑	□
Read	☑	□
Write	☑	□

For special permissions or for advanced settings, click Advanced.

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 (take union of all permissions)
      - Negative permission (deny access) takes priority

Tokens

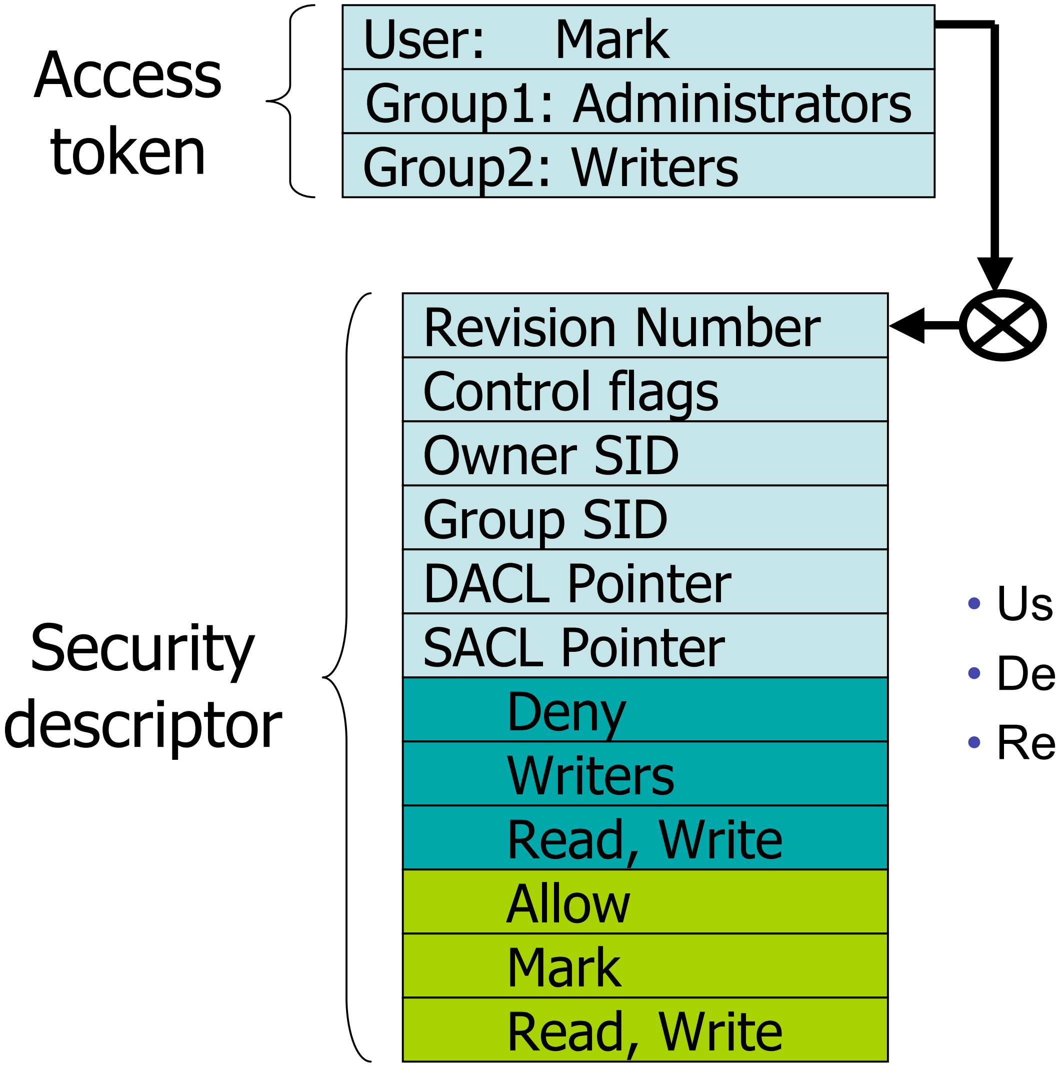
- Security Reference Monitor
  - uses tokens to identify the security context of a process or thread
- Security context
  - privileges, accounts, and groups associated with the process or thread
- Impersonation token
  - thread uses temporarily to adopt a different security context, usually of another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

- Related to the EUID used in Unix.

Security Descriptor

- Access Control List associated with an object
  - Specifies 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



Access token

User: Mark

Group1: Administrators

Group2: Writers

Security descriptor

Revision Number

Control flags

Owner SID

Group SID

DACL Pointer

SACL Pointer

Deny

Writers

Read, Write

Allow

Mark

Read, Write

Access request: write

Action: denied

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

Impersonation Tokens

- Windows equivalent of 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