

---

---

CIS 551 / TCOM 401  
**Computer and Network Security**

Spring 2007  
Lecture 3

# Announcements

---

- Email project groups to Jeff (vaughan2 AT seas.upenn.edu) by Jan. 25
- Start your projects early!

# The “Gold” Standard

---

---

- *Authentication*
  - Identify which principals take which actions
- *Audit*
  - Recording the security relevant actions
- *Authorization*
  - Determine what actions are permissible
  - This lecture is about authorization.
  - We'll get to authentication & audit later.

# Authorization

---

---

- Authorization is the process of determining whether a principal is permitted to perform a particular action.
- Access control
  - Example: Read/Write/Execute permissions for a file system.
  - Example: Java applets have restricted authorization to perform network & disk I/O.

# Policy vs. Mechanism

---

---

- Access control policy is a *specification*
  - Given in terms of a model of the system
  - Subjects: do things (i.e. a process writes to files)
  - Objects: are passive (i.e. the file itself)
  - Actions: what the subjects do (i.e. read a string from a file)
  - Rights: describe authority (i.e. read or write permission)
- Mechanisms are used to *implement* a policy
  - Example: access control bits in Unix file system & OS checks
  - Mechanism should be general; ideally should not constrain the possible policies.
  - Complete mediation: every access must be checked

# Access Control Matrices

A[s][o]	Obj <sub>1</sub>	Obj <sub>2</sub>	...	Obj <sub>N</sub>
Subj <sub>1</sub>	{r,w,x}	{r,w}	...	{}
Subj <sub>2</sub>	{w,x}	{}	...	
...	...	...	...	
Subj <sub>M</sub>	{x}	{r,w,x}	...	{r,w,x}

Each entry contains a set of rights.

# Access Control Checks

---

---

- Suppose subject  $s$  wants to perform action that requires right  $r$  on object  $o$ :
- If  $(r \in A[s][o])$  then *perform action*  
else *access is denied*

# Rights and Actions

---

- Besides read, write, execute actions there are many others:
- Ownership
- Creation
  - New subjects (i.e. in Unix add a user)
  - New objects (i.e. create a new file)
  - New rights: Grant right  $r$  to subject  $s$  with respect to object  $o$  (sometimes called delegation)
- Deletion of
  - Subjects
  - Objects
  - Rights (sometimes called revocation)



# Access Control Examples

---

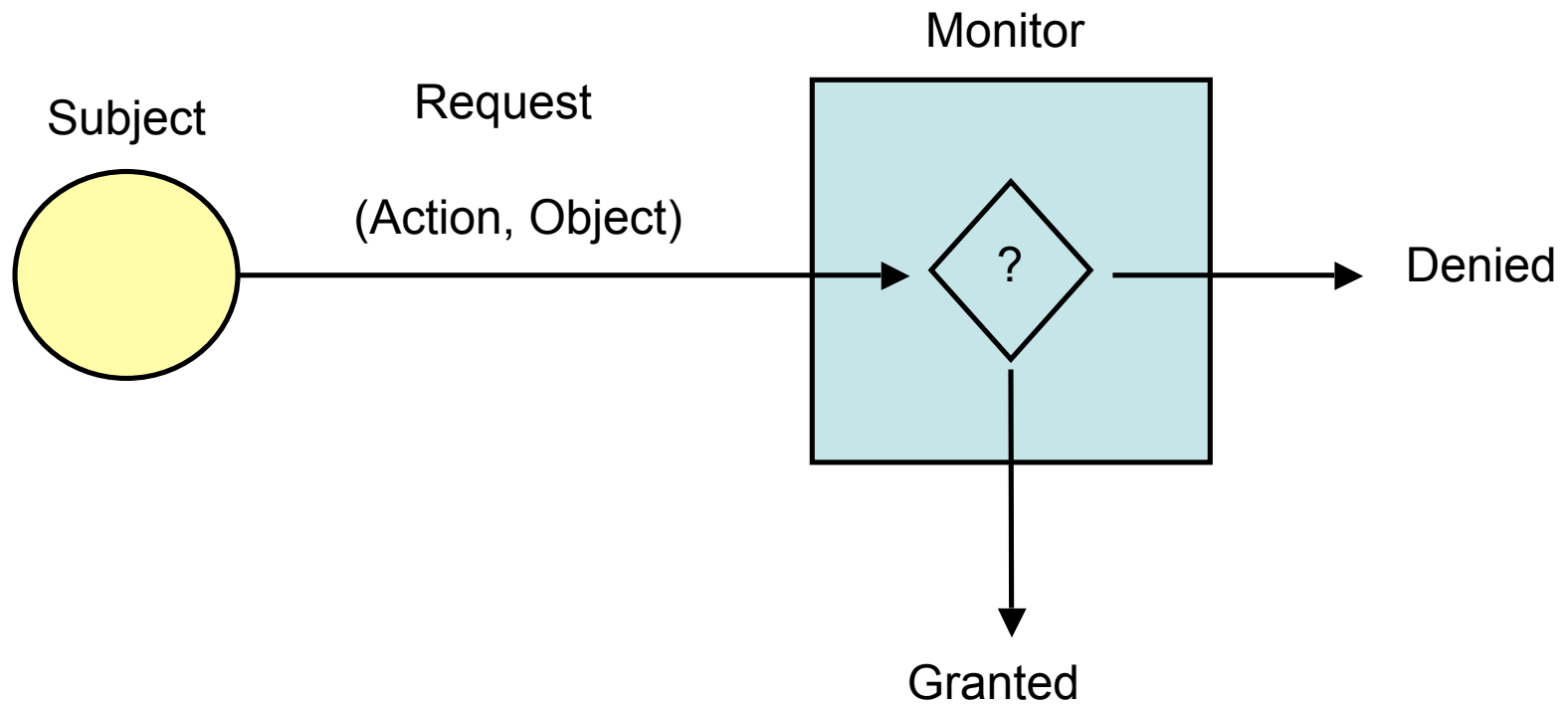
---

- Assume OS is a subject with all rights
- To create a file *f* owned by Alice:
  - Create object *f*
  - Grant own to Alice with respect to *f*
  - Grant read to Alice with respect to *f*
  - Grant write to Alice with respect to *f*
- To start a login for Alice
  - Input and check password
  - Create a shell process *p*
  - Grant own\_process to Alice with respect to *p*

# Reference Monitors

---

---



# Reference Monitors

---

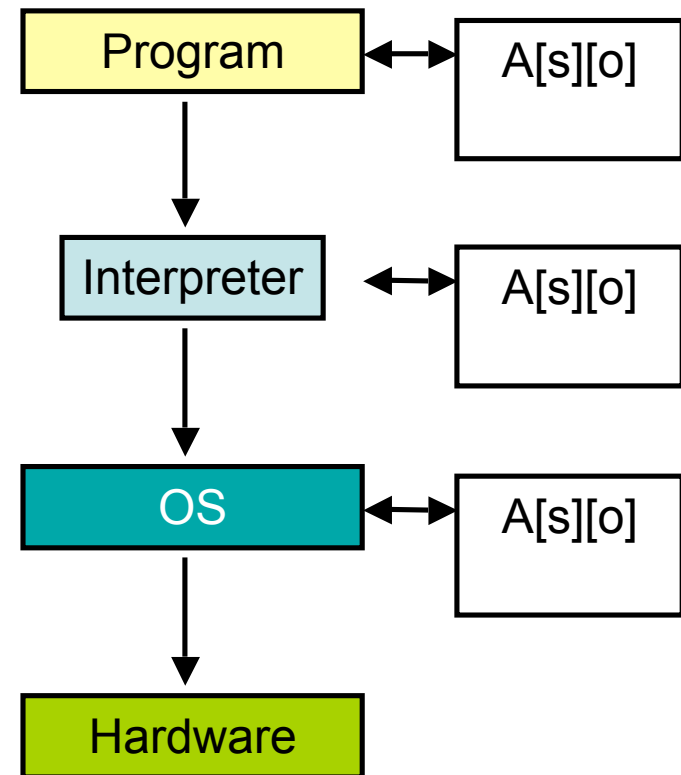
---

- Criteria
  - Correctness
  - Complete mediation (all avenues of access must be protected)
  - Expressiveness (what policies are admitted)
  - How large/complex is the mechanism?
  
- Trusted Computing Base (TCB)
  - The set of components that must be trusted to enforce a given security policy
  - Would like to simplify/minimize the TCB to improve assurance of correctness

# Software Mechanisms

---

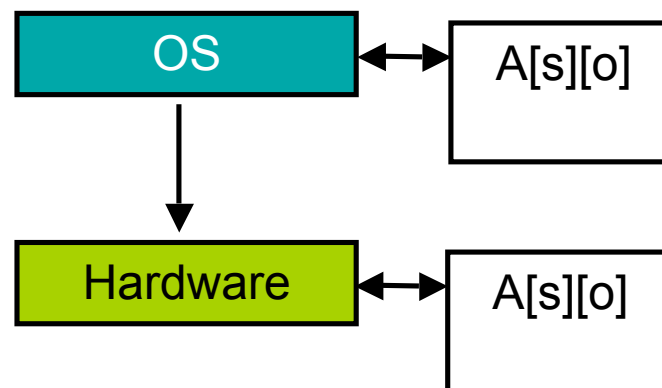
- Interpreters
  - Check the execution of every instruction
  - Hard to mediate high-level abstractions
- Wrappers
  - Only “interpret” some of the instructions
  - What do you wrap?
  - Where do you wrap? (link-time?)
- Operating Systems
  - Level of granularity?
  - Context switching overheads?
- Example
  - Java and C# runtime systems



# Hardware Mechanisms

---

- Multiple modes of operation
  - User mode (problem state)
  - Kernel mode (supervisor state)
- Specialized hardware
  - Virtual memory support (TLB's, etc.)
  - Interrupts



# Protecting Reference Monitors

---

---

- It must not be possible to circumvent the reference monitor by corrupting it
- Mechanisms
  - Type checking
  - Sandboxing: run processes in isolation
  - Software fault isolation: rewrite memory access instructions to perform bounds checking
  - User/Kernel modes
  - Segmentation of memory (OS resources aren't part of virtual memory system)

# Implementing Access Control

---

---

- Access control matrices
    - Subjects  $\gg$  #users (say 1000s)
    - Objects  $\gg$  #files (say 1,000,000s)
    - To specify “all users read f”
      - Change  $O(\text{users})$  entries
  - Matrix is typically sparse
    - Store only non-empty entries
  - Special consideration for groups of users
-

# Access Control Lists

A[s][o]	Obj <sub>1</sub>	Obj <sub>2</sub>	...	Obj <sub>N</sub>
Subj <sub>1</sub>	{r,w,x}	{r,w}	...	{}
Subj <sub>2</sub>	{w,x}	{}	...	{r}
...	...	...	...	...
Subj <sub>M</sub>	{x}	{r,w,x}	...	{r,w,x}

For each object, store a list of (Subject x Rights) pairs.



# Access Control Lists

---

---

- Resolving queries is linear in length of the list
- Revocation w.r.t. a single object is easy
- “Who can access this object?” is easy
  - Useful for auditing
- Lists could be long
  - Factor into groups (lists of subjects)
  - Give permissions based on group
  - Introduces consistency question w.r.t. groups
- Authentication critical
  - When does it take place? Every access would be expensive.

# Representational Completeness

---

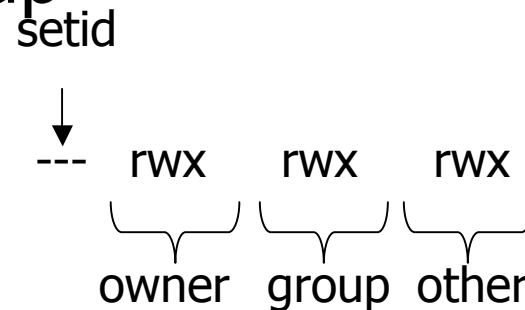
---

- Access Control Lists
  - Can represent any access control matrix
  - Potentially very large
  - Used in windows file system, NTFS
- Unix file permissions (next topic)
  - Fixed size
  - Can't naturally express some access control policies/matrices

# 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?
    - 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

# Unix Policies Interact

---

---

```
/home/jeff/          jeff  jeff  -rwx  ---  ---  
/home/jeff/.bashrc  jeff  jeff  -rwx  r--  r--
```

- stevez cannot read /home/jeff/.bashrc
  - The confidentiality/availability of an object depends on policies other than it's own.
  - Such interacts make specifying policies hard.
  - Problem is not limited to unix (or file systems).

# Setid bits on executable Unix file

---

- Three setid bits
  - 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
  - Setuid – set EUID of process to ID of file owner
    - passwd owned by root and setuid is true
    - Jeff executes passwd: “passwd runs as root”
  - Setgid – set EGID of process to GID of file

# Effective User ID (EUID)

---

---

- Each process has three user IDs (more in 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 program file, or system 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

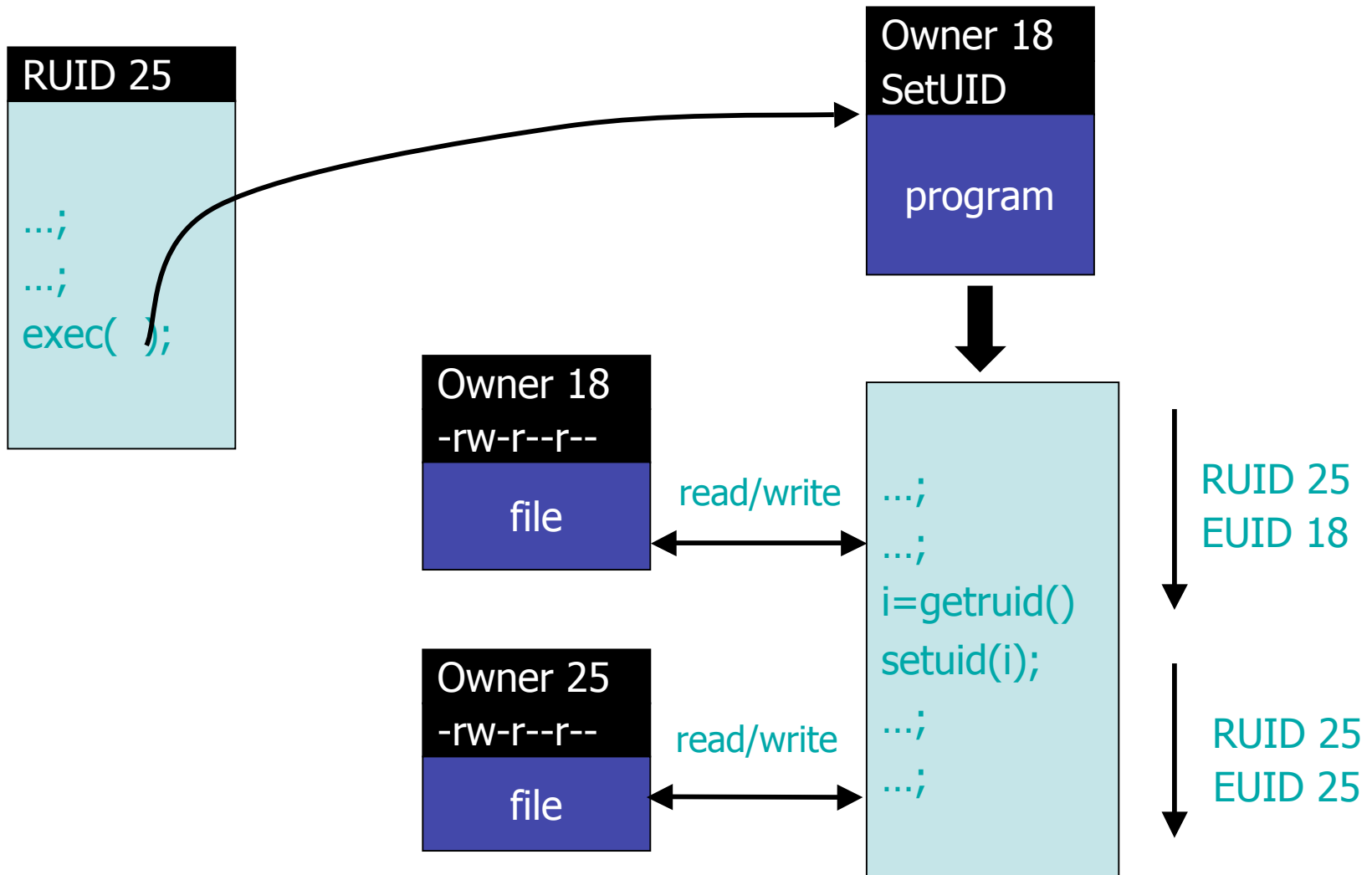
# Process Operations and IDs

---

- 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



# Example



# 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)

---

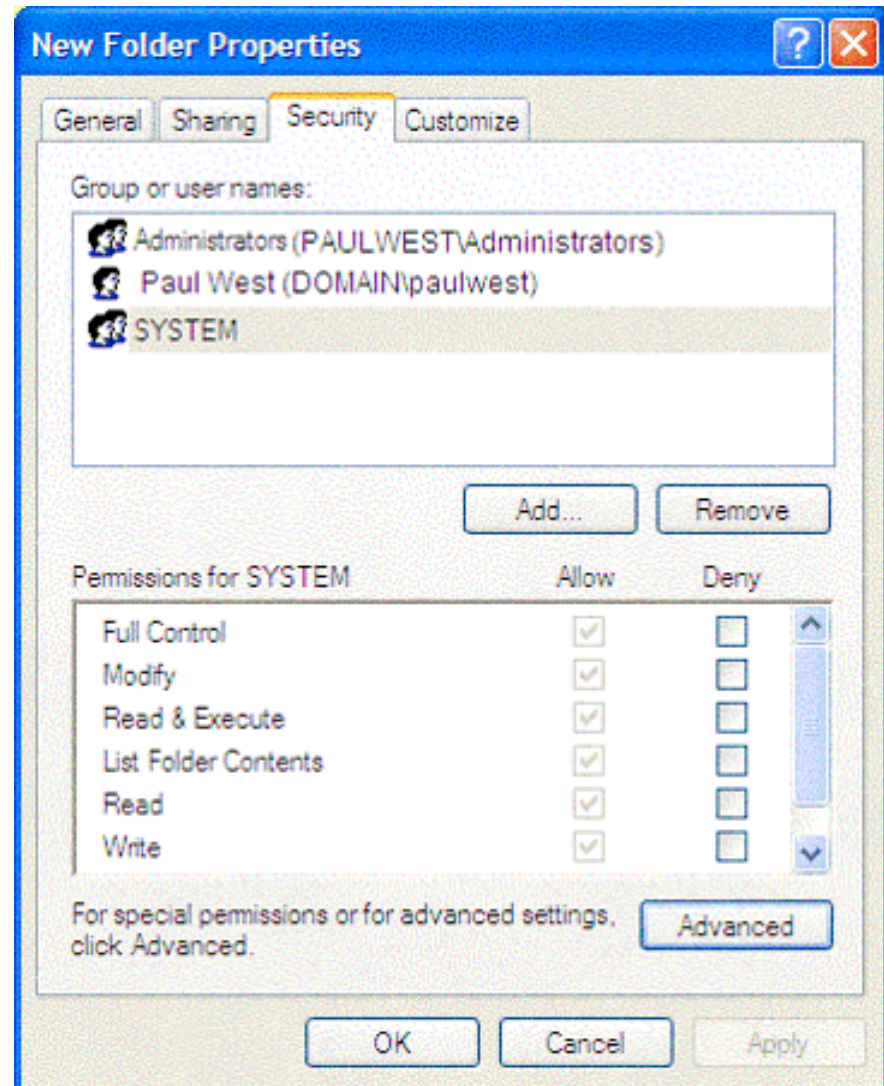
---

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
  - ACLs used for fine grained control
- 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

- “Security IDentifier”
- Identity (like Unix UID)
  - SID revision number
  - 48-bit authority value
  - Globally unique
- Describes users, groups, computers, domains, domain members



# File 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

# 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)
      - Describes access policy
    - System Access Control List (SACL)
      - Describes audit/logging policy

# 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.



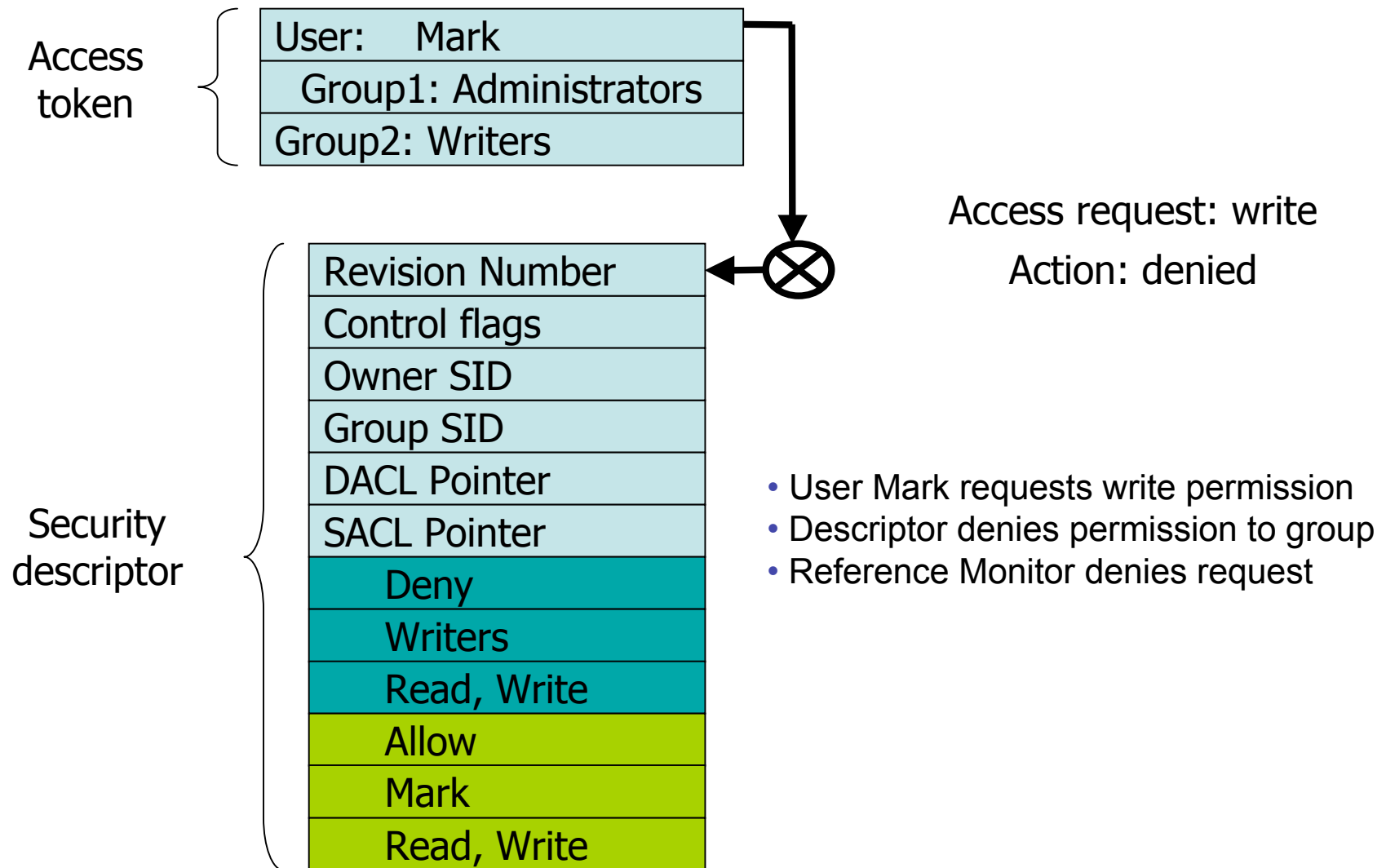
# Impersonation Tokens

---

---

- 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 (= Anonymous + Identification)
    - server identify and impersonate the client
  - Delegation (= Impersonation + Authentication)
    - lets server impersonate client on local, remote system

# Example access request



# Windows Summary

---

---

- Good things
  - Very expressive
  - Don't need full SYSTEM (e.g. root) privileges for many tasks
- Bad thing
  - More complex policies
    - Harder to implement: Larger TCB
    - Harder for users to understand

