



TEL AVIV UNIVERSITY

Information Security – Theory vs. Reality

0368-4474-01, Winter 2011

Lecture 4: Access Control

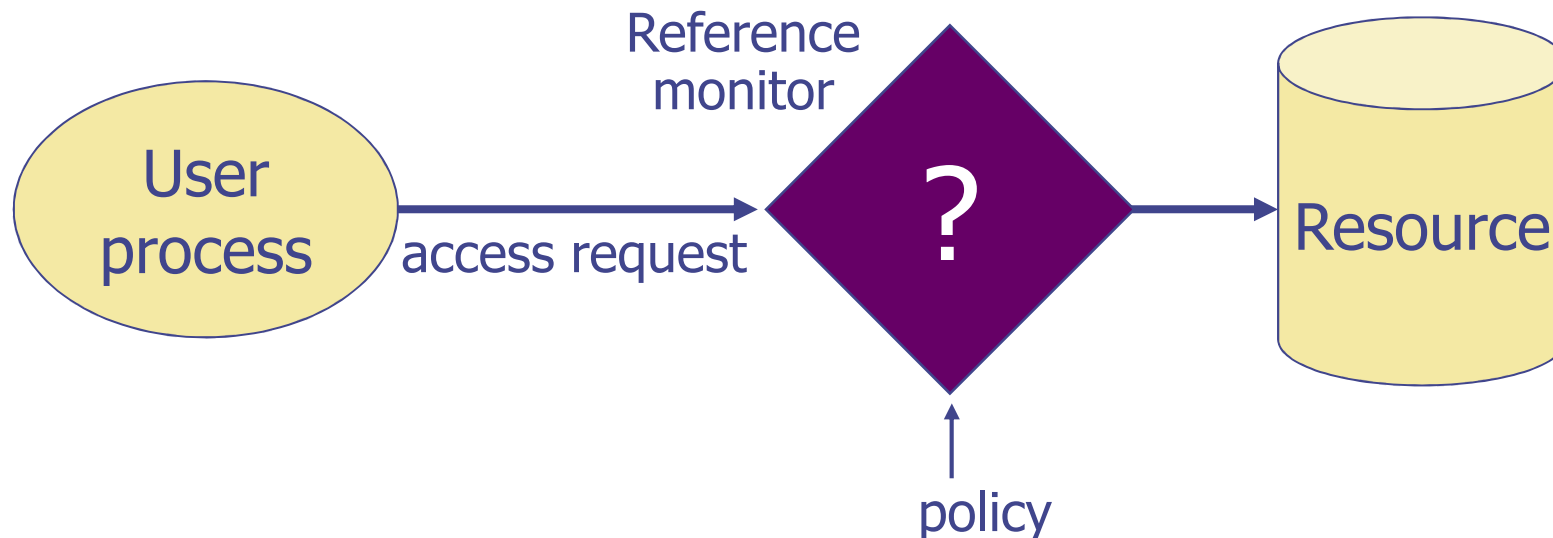
Eran Tromer

Slides credit: John Mitchell, Stanford course CS155, 2010

Access control

◆ Assumptions

- System knows who the user is
 - ◆ Authentication via name and password, other credential
- Access requests pass through gatekeeper (reference monitor)
 - ◆ System must not allow monitor to be bypassed



Access control matrix [Lampson]

Objects

	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

Subjects

Access Control Lists

- ◆ Access control list (ACL)
 - Store column of matrix with the resource

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

- ◆ Widely used in commercial systems

Capabilities

◆ Capability

- User holds a “ticket” for each resource

◆ Implementation approaches

- Operating system maintains capabilities list attached to each user/process
- Unforgeable ticket in user space (e.g., random strings, cryptographic certificates)

◆ Widely deployed examples:

- X server: “magic cookie” Xauth mechanism
- Kerberos (including Windows network file sharing)

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

ACL vs Capabilities

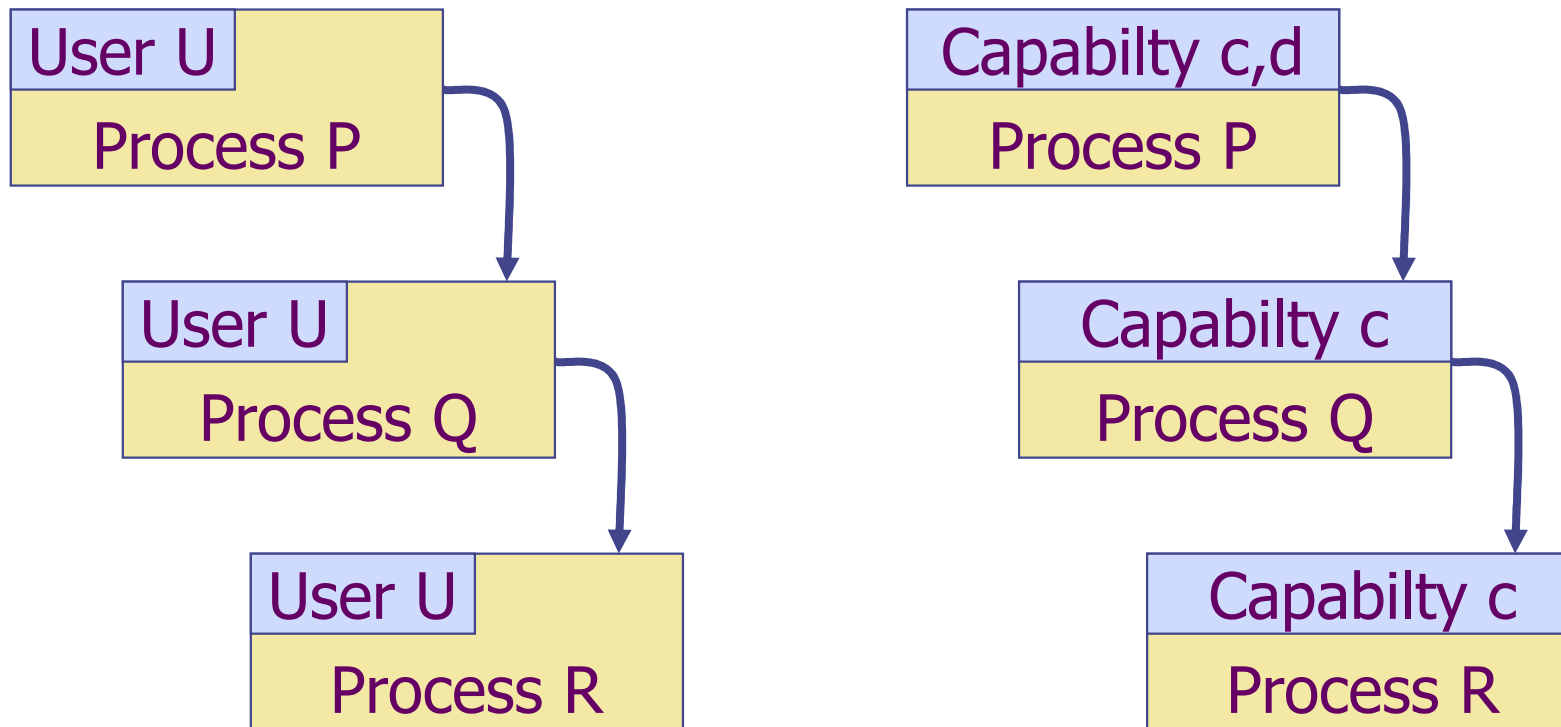
◆ Access control list

- 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?
 - ◆ More common: let other process act under current user

◆ 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 which data is capability
 - If capability is used for multiple resources, have to revoke all or none ...
 - ◆ Indirection: capability points to pointer to resource
 - If $C \rightarrow P \rightarrow R$, then revoke capability C by setting $P=0$

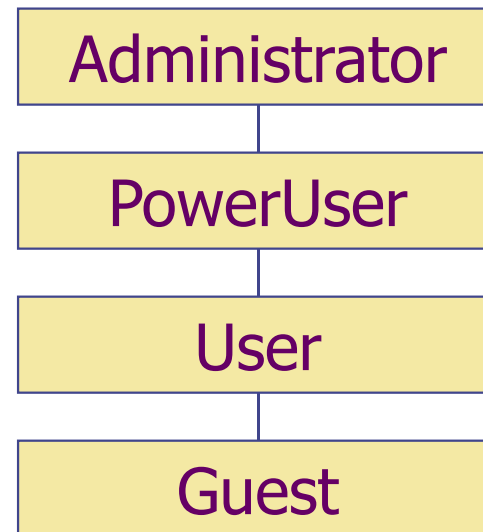
Roles (also called Groups)

◆ Role = set of users

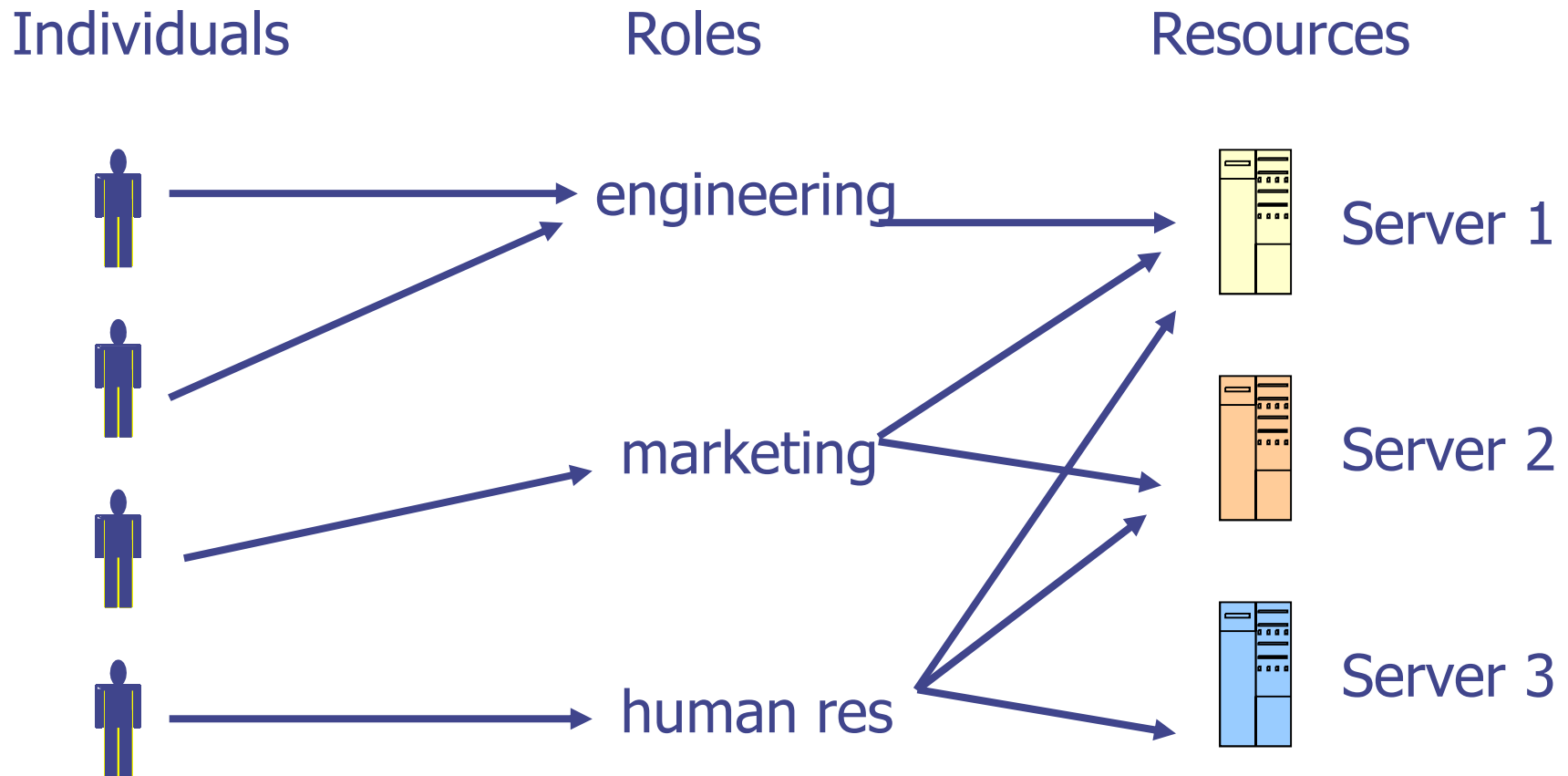
- Administrator, PowerUser, User, Guest
- Assign permissions to roles; each user gets permission

◆ Role hierarchy

- Partial order of roles
- Each role gets permissions of roles below
- List only new permissions given to each role



Role-Based Access Control



Advantage: user's change more frequently than roles

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

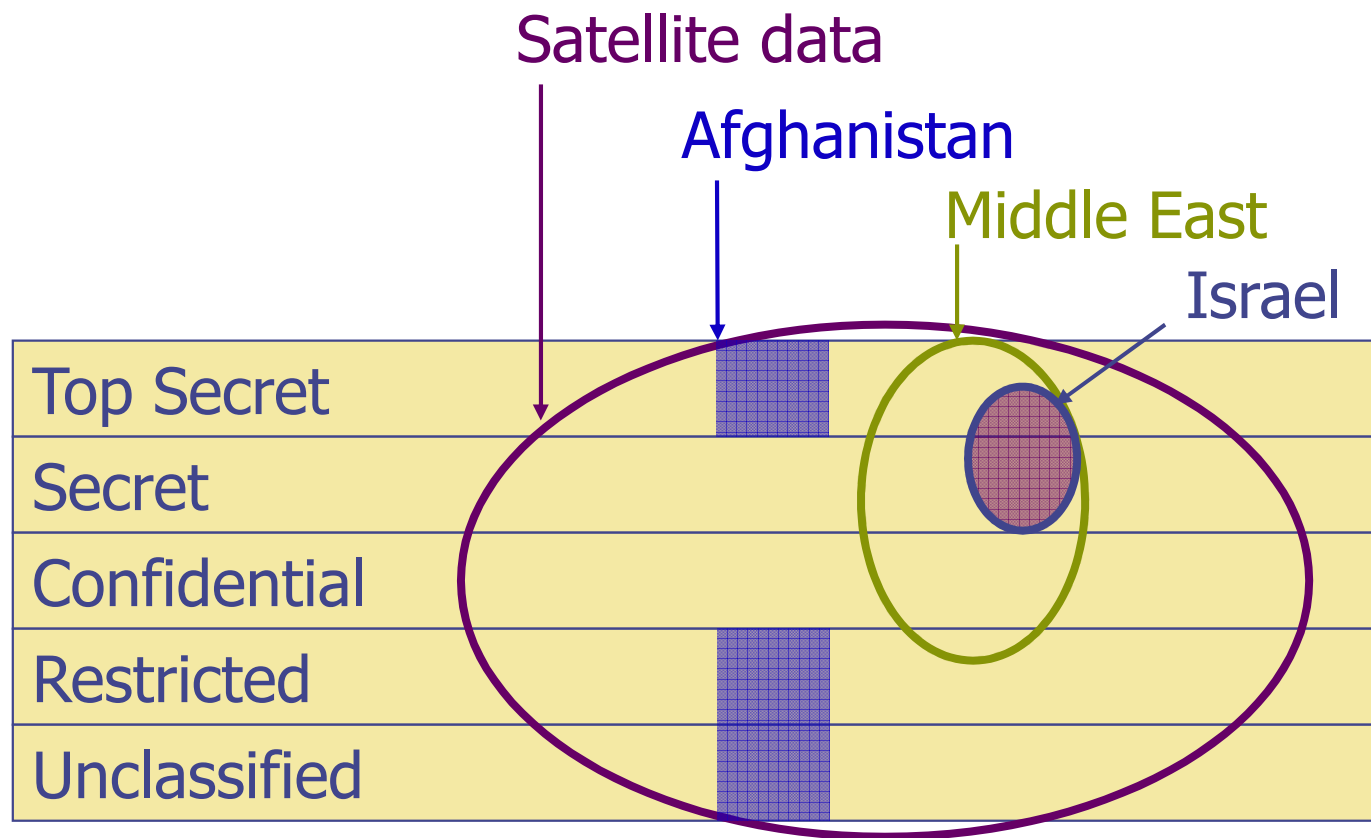
◆ Other policy concepts

- Separation of duty
- “Chinese Wall” Policy (managing conflicts of interests)

Military security policy

◆ Sensitivity levels

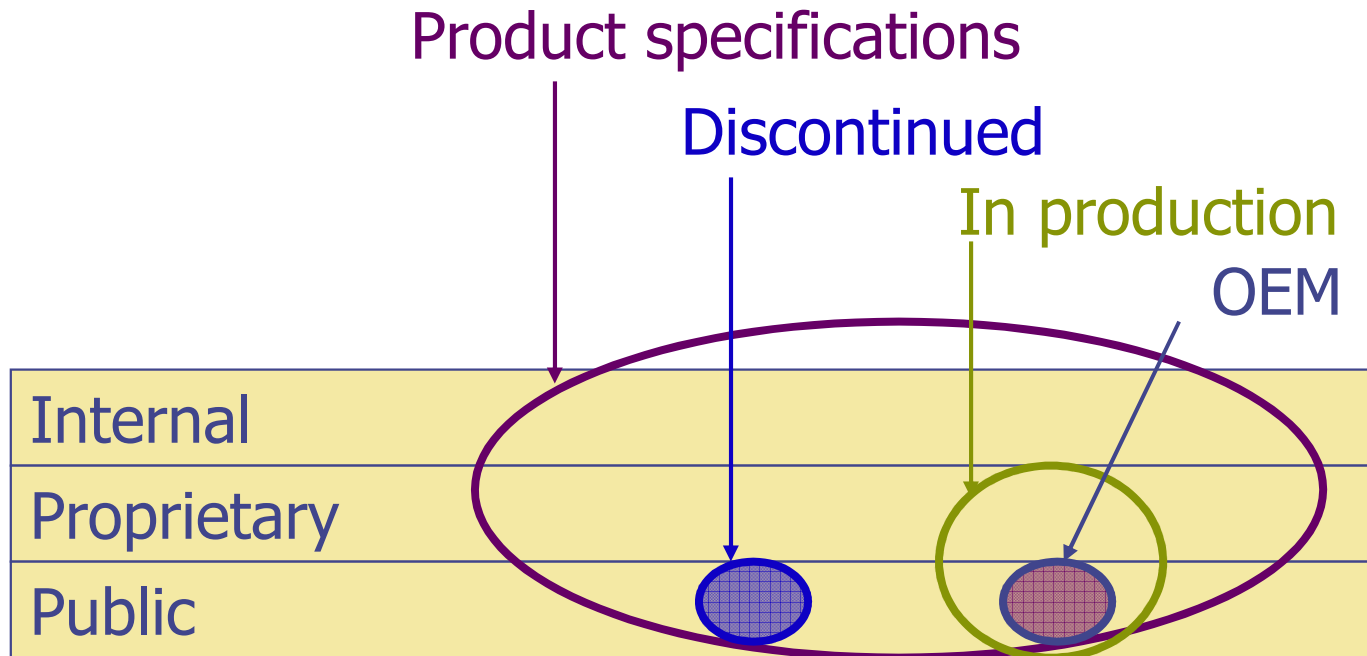
◆ Compartments



Military security policy

- ◆ Classification of personnel and data
 - Class = $\langle \text{rank}, \text{compartment} \rangle$
- ◆ Dominance relation
 - $D_1 \leq D_2$ iff $\text{rank}_1 \leq \text{rank}_2$
and $\text{compartment}_1 \subseteq \text{compartment}_2$
 - Example: $\langle \text{Restricted}, \text{Israel} \rangle \leq \langle \text{Secret}, \text{Middle East} \rangle$
- ◆ Applies to
 - Subjects – users or processes
 - Objects – documents or resources

Commercial version

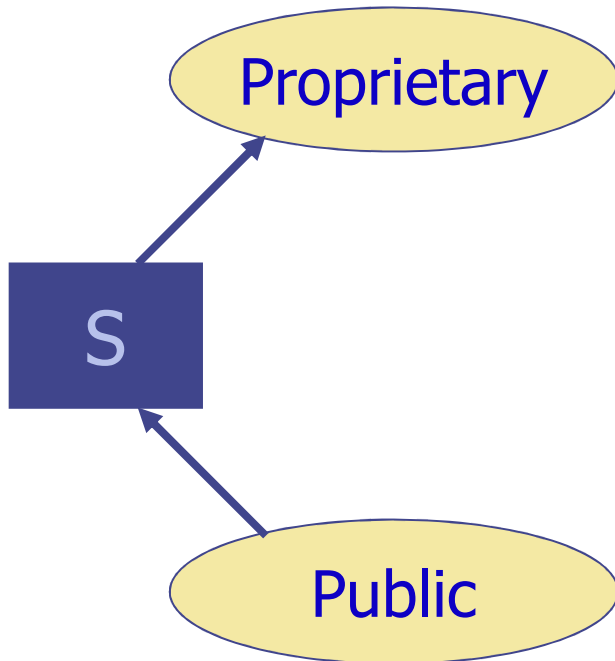


Bell-LaPadula Confidentiality Model

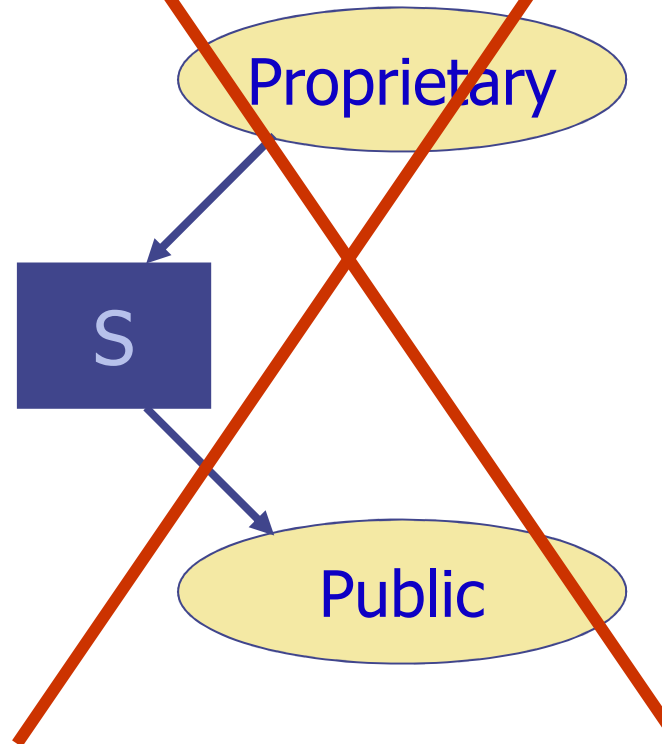
- ◆ When is it OK to release information?
- ◆ Two Properties (with silly names)
 - No read up ("Simple security property")
 - ◆ A subject S may read object O only if $C(O) \leq C(S)$
 - No write down ("*-Property")
 - ◆ A subject S with read access to O may write object P only if $C(O) \leq C(P)$
- ◆ In words,
 - You may only *read below* your classification and only *write above* your classification
- ◆ Mandatory access control

Picture: Confidentiality

Read below, write above



~~Read above, write below~~

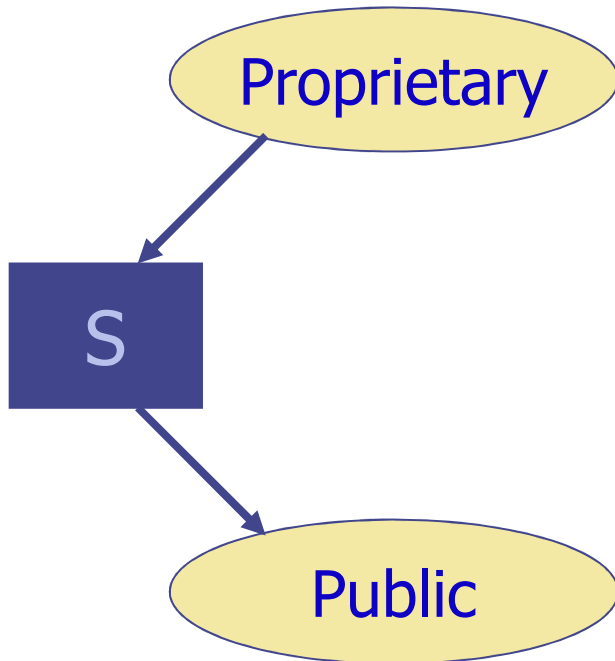


Biba Integrity Model

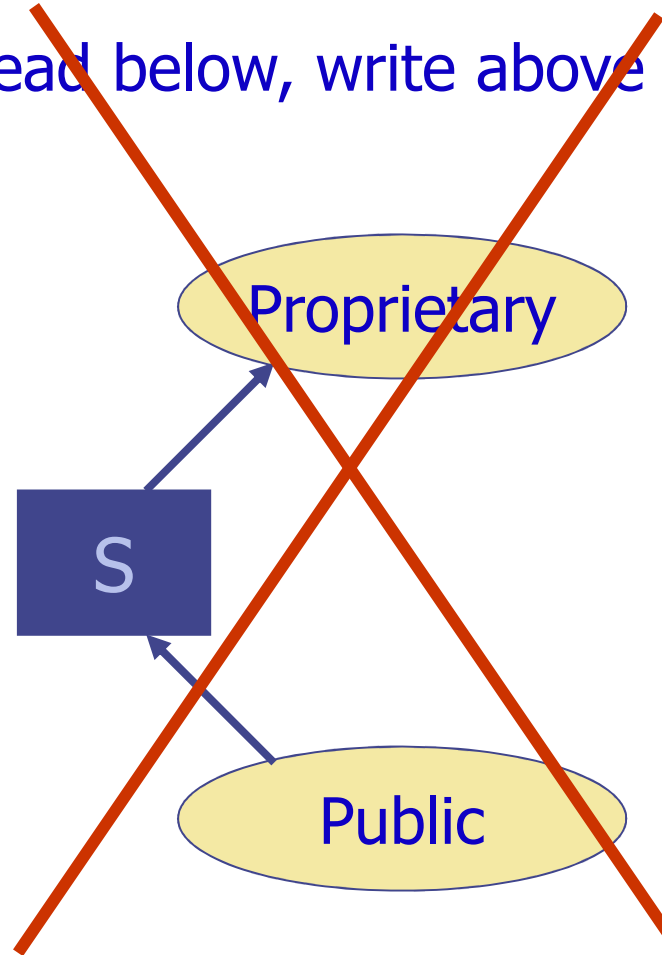
- ◆ Rules that preserve integrity of information (dual to confidentiality)
- ◆ Two Properties (with silly names)
 - No write up ("Simple security property")
 - ◆ A subject S may write object O only if $C(S) \geq C(O)$
(Only trust S to modify O if S has higher rank ...)
 - No read down ("*-Property")
 - ◆ A subject S with read access to O may write object P only if $C(O) \geq C(P)$
(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

Read above, write below



Read below, write above



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 \neq

◆ Chinese Wall Policy

- Lawyers L1, L2 in same firm
- If company C1 sues C2,
 - ◆ L1 and L2 can each work for either C1 or C2
 - ◆ 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

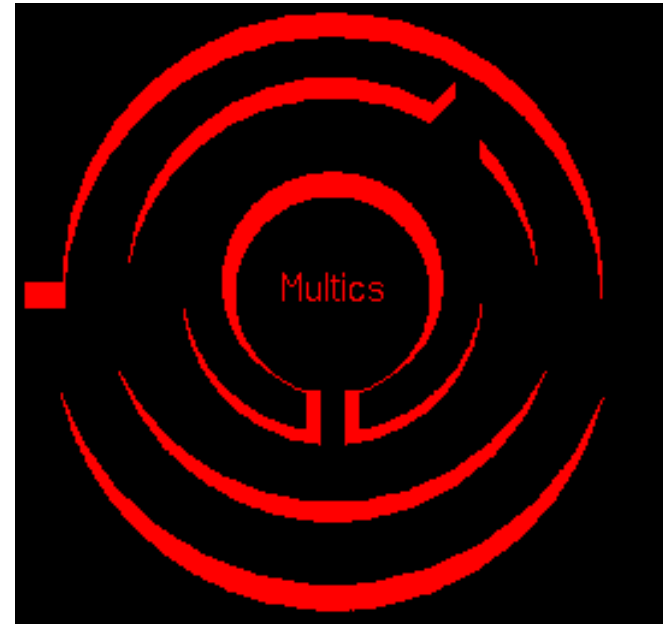
Multics

◆ 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>

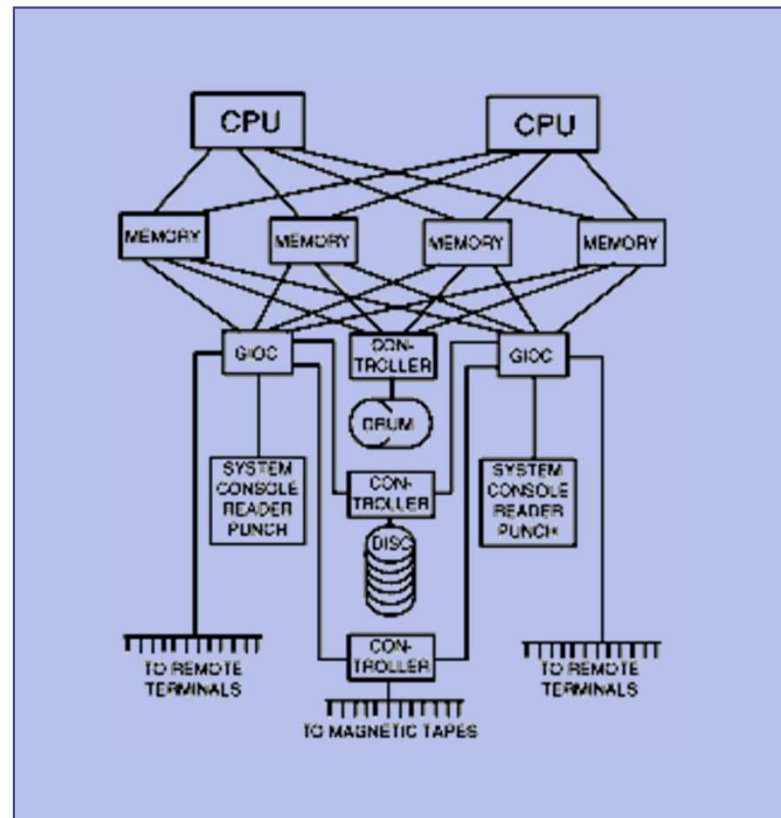
❗. Organick, The Multics System: An Examination of Its Structure, MIT Press, 1972

Multics time period

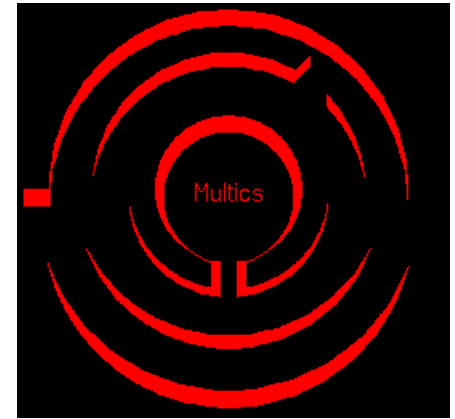


F.J. Corbato

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



Multics 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 $\langle b1, b2, b3 \rangle$ with $b1 \leq b2 \leq b3$
 - ◆ Process/data can be accessed from rings $b1 \dots b2$
 - ◆ A process from rings $b2 \dots b3$ 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

Server port	Obj #	Rights	Check field
-------------	-------	--------	-------------

◆ 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
 - ◆ Server uses object # to indentify object
 - ◆ Server checks rights field to see if operation is allowed
 - ◆ Check field prevents processes from forging capabilities

Capabilities

Server port	Obj #	Rights	Check field
-------------	-------	--------	-------------

◆ 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

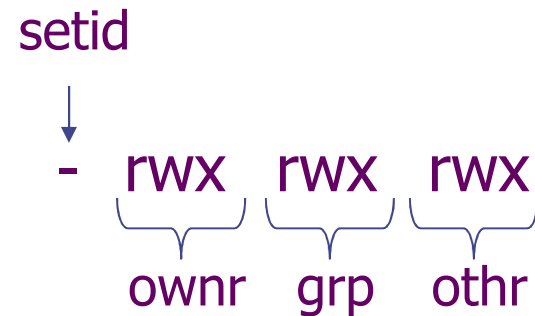
Protection by user-process at server; no special OS support needed

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

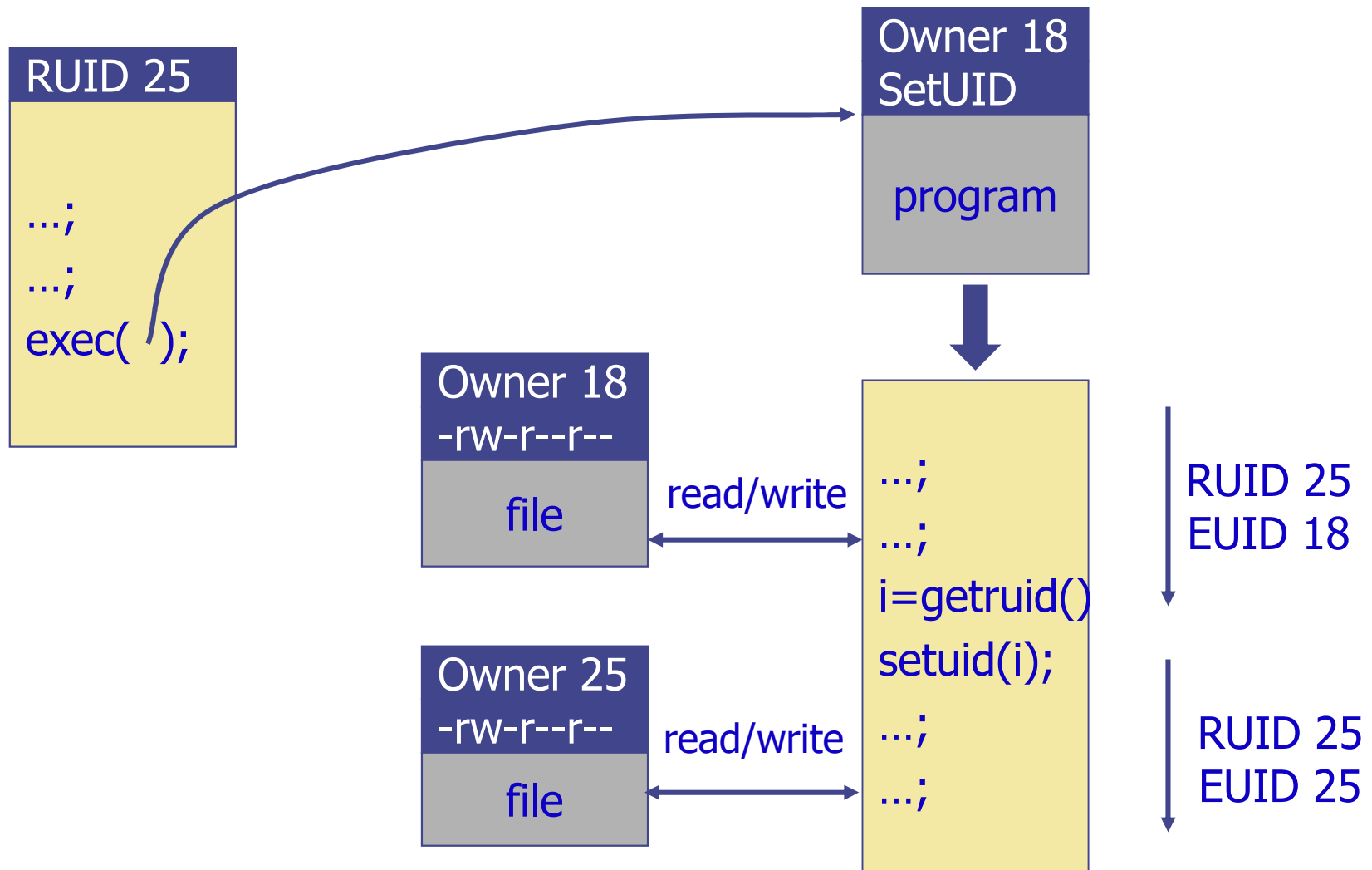
- Several different calls: setuid, seteuid, setreuid

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



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 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

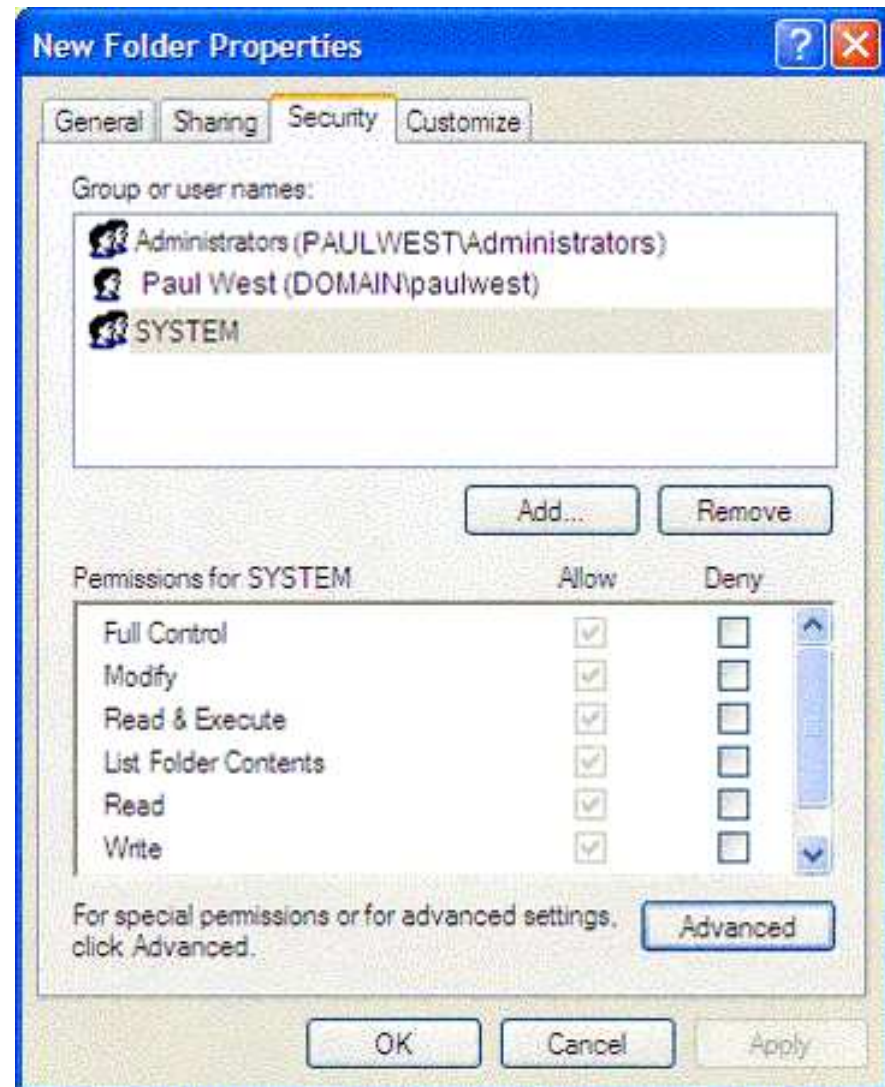
◆ New systems (e.g., Linux) add “capabilities”

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



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

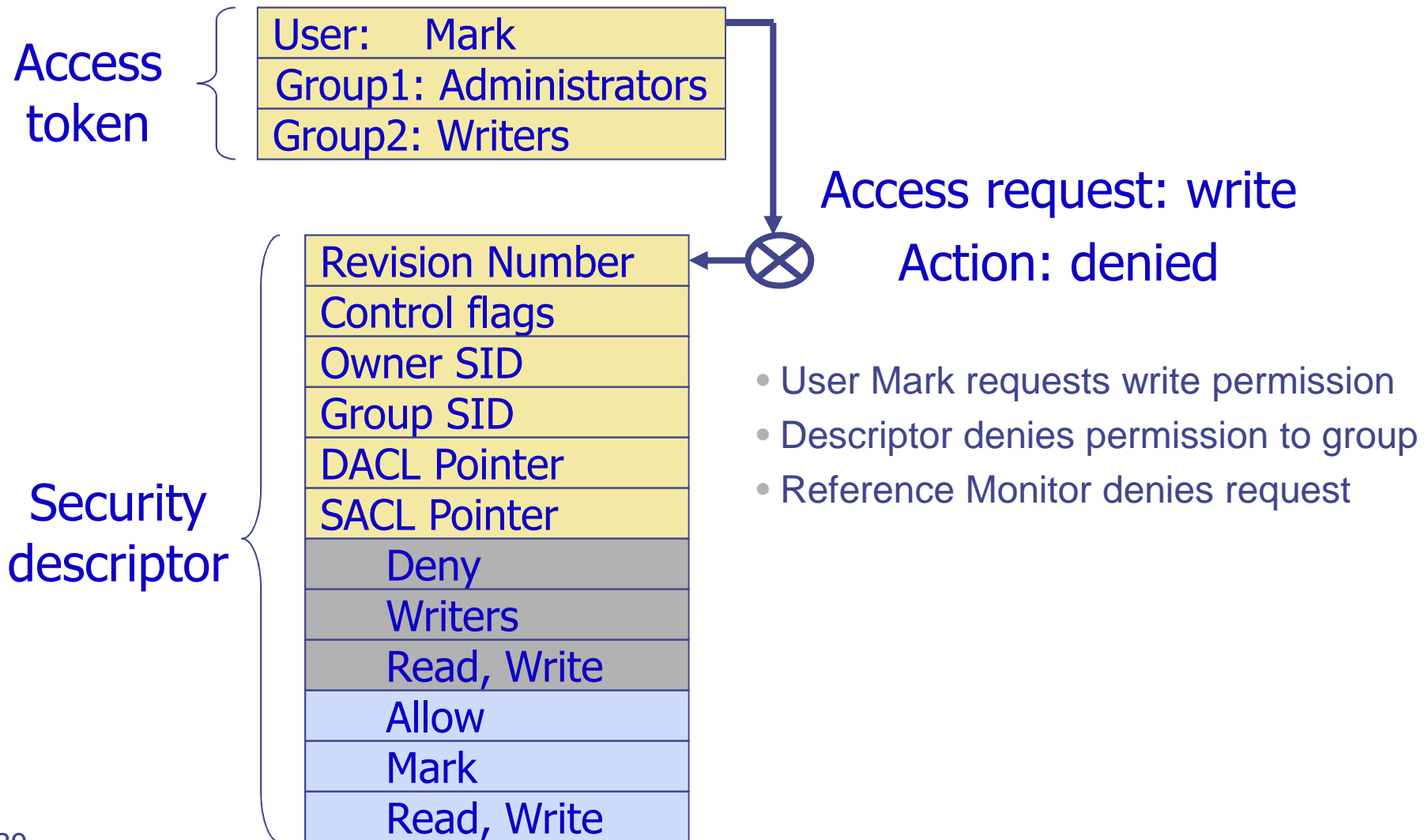
◆ Impersonation token

- 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

An Analogy

Operating system

◆ Primitives

- System calls
- Processes
- Disk

◆ Principals: Users

- Discretionary access control

◆ Vulnerabilities

- Buffer overflow
- Root exploit

Web browser

◆ Primitives

- Document object model
- Frames
- Cookies / localStorage

◆ Principals: "Origins"

- Mandatory access control

◆ Vulnerabilities

- Cross-site scripting
- Universal scripting

Components of browser security policy

◆ Frame-Frame relationships

- `canScript(A,B)`
 - ◆ Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
- `canNavigate(A,B)`
 - ◆ Can Frame A change the origin of content for Frame B?

◆ Frame-principal relationships

- `readCookie(A,S)`, `writeCookie(A,S)`
 - ◆ Can Frame A read/write cookies from site S?

Problems

- ◆ Covert channels
- ◆ Declassification
- ◆ Composability
- ◆ Overclassification
- ◆ Aggregation
- ◆ Incompatibilities (e.g., upgraded files “disappear”)
- ◆ Polyinstantiation (“cover stories” and consistency)

Further reading

- ◆ Anderson, Stajano, Lee, *Security Policies*
<https://www.cl.cam.ac.uk/~rja14/Papers/security-policies.pdf>
- ◆ Levy, *Capability-based Computer Systems*
<http://www.cs.washington.edu/homes/levy/capabook>