Security

CS 4410
Operating Systems

[E. Birrell, A. Bracy, E. Sirer, R. Van Renesse]

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

References: Security Introduction and Access Control by Fred Schneider

Historical Context

1961

Compatible Time-Sharing System (CTSS) is Demonstrated

The increasing number of users needing access to computers in the early 1960s leads to experiments in timesharing computer systems. Timesharing systems can support many users – sometimes hundreds – by sharing the computer with each user. CTSS was developed by the MIT Computation Center under the direction of Fernando Corbató and was based on a modified IBM 7094 mainframe computer. Programs created for CTSS included RUNOFF, an early text formatting utility, and an early inter-user messaging system that presaged email. CTSS operated until 1973.

1969

Kenneth Thompson and Dennis Ritchie develop UNIX

AT&T Bell Labs programmers Kenneth Thompson and Dennis Ritchie develop the UNIX operating system on a spare DEC minicomputer. UNIX combined many of the timesharing and file management features offered by Multics, from which it took its name. (Multics, a project of the mid-1960s, represented one of the earliest efforts at creating a multi-user, multi-tasking operating system.) The UNIX operating system quickly secured a wide following, particularly among engineers and scientists, and today is the basis of much of our world's computing infrastructure.

1960's OSes begin to be shared. Enter:

- Communication
- Synchronization
- Security: once a small OS sub-topic. *Not anymore!*

http://www.computerhistory.org, https://en.wikipedia.org

Security Properties: CIA

Confidentiality: *keeping secrets*

- who is allowed to learn what information

Integrity: *permitting changes*

- what changes to the system and its environment are allowed

Availability: *guarantee of service*

- service should be “timely”

Security in Computer Systems

Gold (Au) Standard for Security [Lampson]

Authorization: mechanisms that govern whether actions are permitted

Authentication: mechanisms that bind *principals* to actions

Audit: mechanisms that record and review actions

Plan of Attack *(no pun intended!)*

- Protection - *This lecture*
  - Authorization: *what are you permitted to do?*
  - Access Control Matrix

- Security – *Next lecture*
  - Authentication: *how do we know who you are?*
  - Threats and Attacks

Access Control Terminology

Operations: how one learns or updates information

Principals: executors (users, processes, threads, procedures)

Objects of operations: memory, files, modules, services

Access Control Policy:

- who may perform which operations on which objects
- enforces confidentiality & integrity

Goal: each object is accessed correctly and only by those principals that are allowed to do so

Access Control Mechanisms

Reference Monitor:

- entity with the power to observe and enforce the policy
- consulted on each operation invocation
- allows operation if invoker has required privileges
- can enforce **confidentiality** and/or **integrity**

Assumptions:

- Predefined operations are the sole means by which principals can learn or update information
- All predefined operations can be monitored (complete mediation)

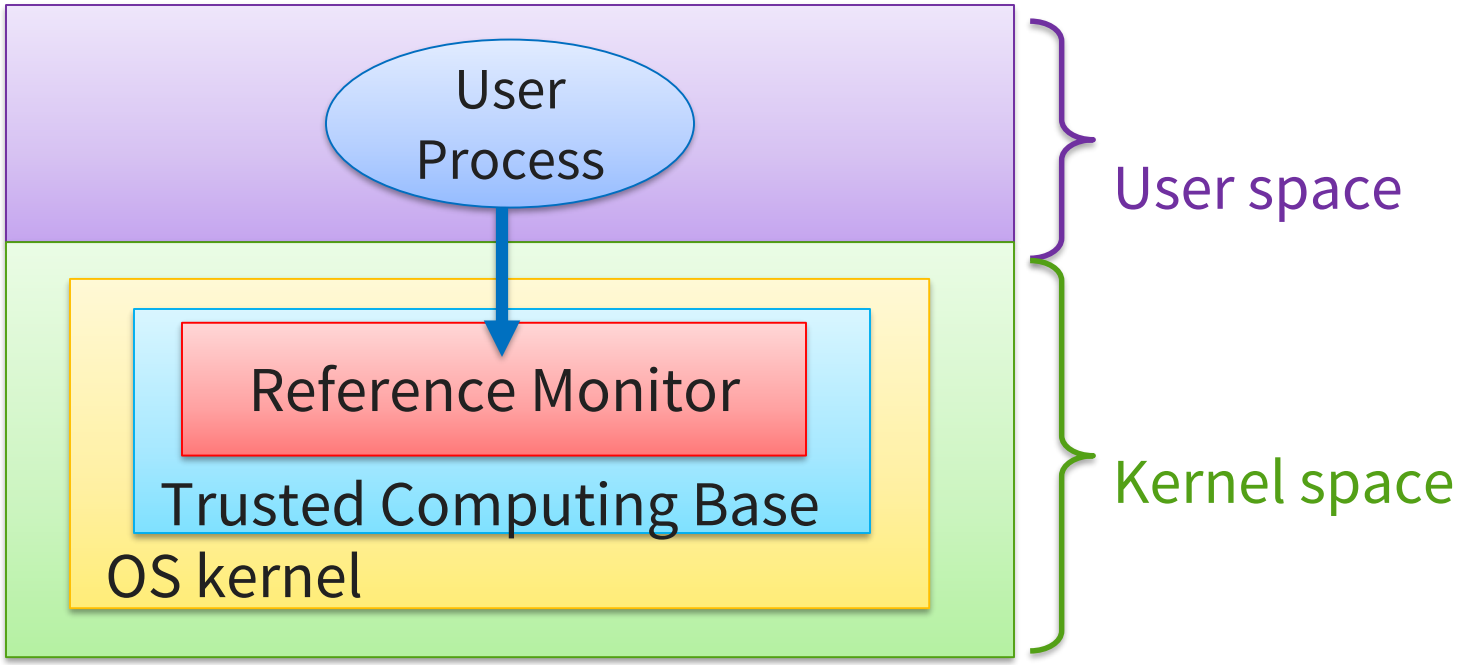
Trusted Computing Base (TCB)

Heart of every trusted system has a small TCB

- HW & SW necessary for enforcing security rules
- Typically has:
  - most hardware, firmware
  - portion of OS kernel
  - most or all programs with superuser power
- Desirable features include:
  - Should be small
  - Should be separable and well-defined
  - Easy to scrutinize independently

TCB and Reference Monitor

- All sensitive operations go through the reference monitor
- Monitor decides if operation should proceed
- Not a separable module in most OSes…



Who defines authorizations?

Discretionary Access Control:

- owner defines authorizations
- Subjects determine who has access to their objects
- Commonly used (Linux/MacOSX/Windows File Systems)
- Flawed for tighter security (program might be buggy)
- **This lecture**

Mandatory Access Control:

- System imposes access control policy that object owners cannot change
- centralized authority defines authorizations

Principle of Least Privilege

"Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."

- Jerome Saltzer
(of the end-to-end argument)

Want to minimize:

- code running inside kernel
- code running as sysadmin

Challenge: It's hard to know:

- what permissions are needed in advance
- what permissions should be granted

Access Control Matrix

- Abstract model of protection
- Rows: principals = users
- Columns: objects = files, I/O, *etc.*

Principals	**OBJECTS**		
	`prelim.pdf`	`jan-hw.tex`	`scores.xls`
rvr (prof)	r, w	r	r, w
jan (student)		r, w	

Unordered set of triples <Principal,Object,Operation>

What does Principle of Least Privilege say about this?

Need Finer-Grained Principals

Protection Domains = new set of principals

- each process belongs to a protection domain
- executing process can transition from domain to domain

Example domain: user ▷ task

- task = program, procedure, block of statements
- task = started by user or in response to user's request
- user ▷ task: holds minimum privilege to get task done for user

→ *task-specific privileges (PoLP is ☺)*

Protection Domain Implementation

Possibilities:

1. Certain system calls cause protection-domain transitions. Obvious candidates:
   - invoking a program
   - changing from user mode to supervisor mode

2. Provide explicit domain-change syscall
   - application programmer or a compiler then required to decide when to invoke this domain-change system call

Access Matrix with Protection Domains

Principals	OBJECTS		
	`prelim.pdf`	`jan-hw.tex`	`scores.xls`
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

When to transition protection-domains?

- invoking a program
- changing from user to kernel mode
- …

Need to explicitly authorize them in the matrix

Access Matrix with Domain Transitions

Principals	OBJECTS								
	prelim.pdf	jan-hw.tex	scores.xls	rvr△sh	rvr△latex	rvr△excel	jan△sh	jan△latex	jan△excel
rvr▷sh					e	e			
rvr▷latex	r, w	r							
rvr▷excel			r, w						
jan▷sh								e	e
jan▷latex		r, w							
jan▷excel									

e = enter

DAC Implementation Needs

Must support:

- Determining if <*Principal,Object,Operation*> is in matrix
- Changing the matrix
- Assigning each process a protection domain
- Transitioning between domains as needed
- Listing each principal's privileges (for each object)
- Listing each object's privileges (held by principals)

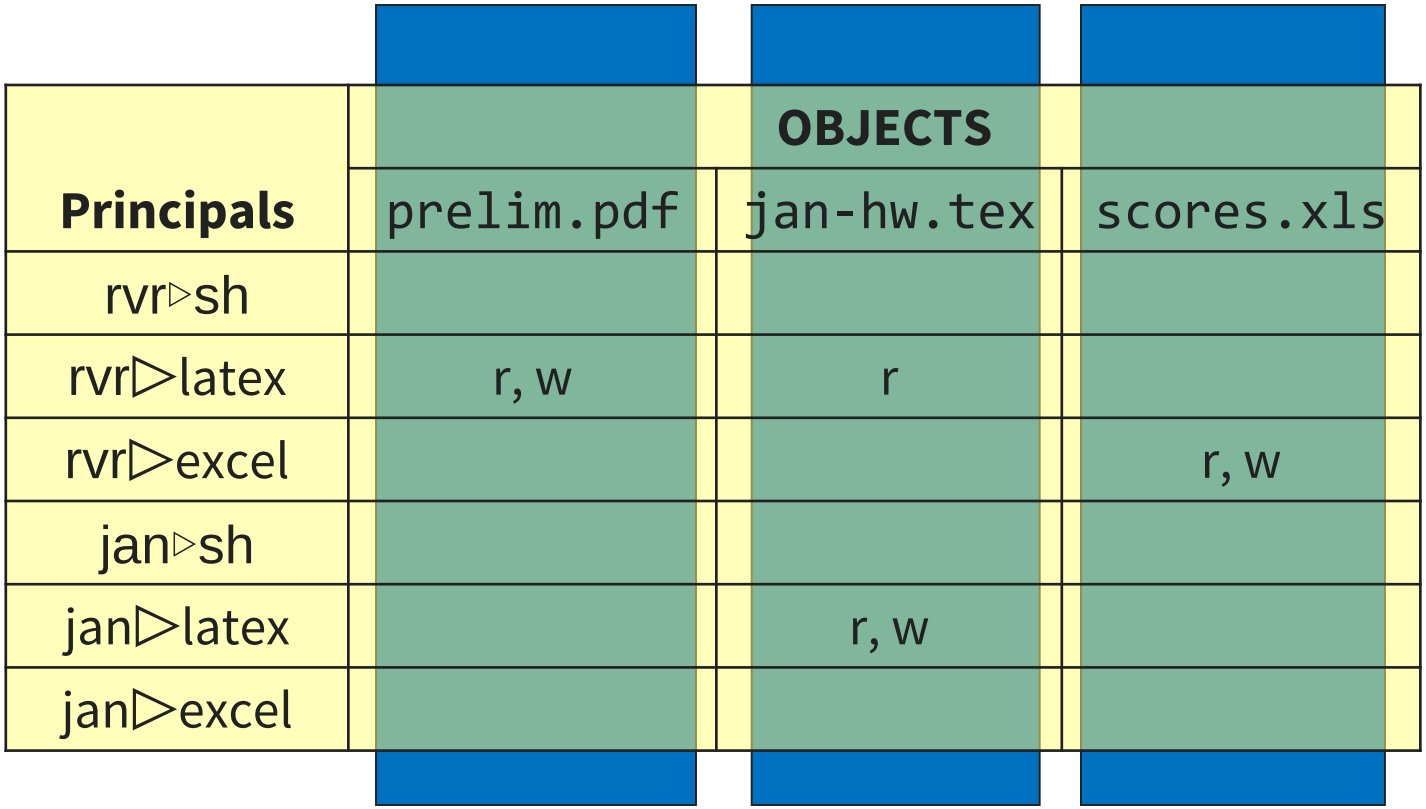
2D array?

+ looks good in powerpoint!

− sparse → store only the non-empty cells

How shall we implement this?

Access Control List (ACL): column for each object stored as a list for the object



Principals	OBJECTS		
	prelim.pdf	jan-hw.tex	scores.xls
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

How shall we implement this?

Access Control List (ACL): column for each object stored as a list for the object

Capabilities: row for each subject stored as list for the subject

Principals	OBJECTS		
	prelim.pdf	jan-hw.tex	scores.xls
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

Same in theory; different in practice!

Access Control Lists

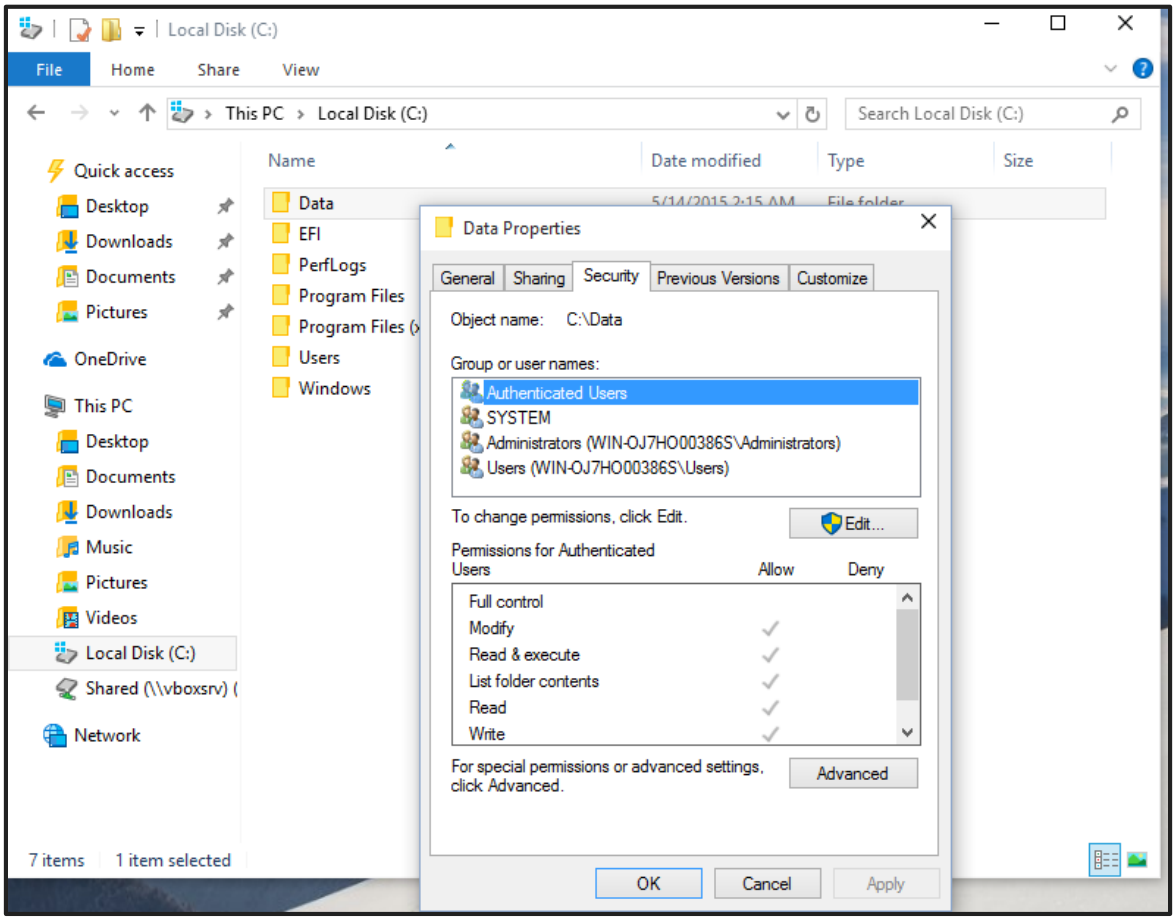
ACL for an object is a list

e.g., ⟨ebirrell, {r,w}⟩ ⟨clarkson, {r}⟩ ⟨student, {r}⟩

To check whether is allowed to perform some operation on some object,

- Look up principal in object's ACL. If not in ACL, reject
- Check whether operation is in the set for that principal. If not, reject

Access Control in Windows



In NTFS: each file has a set of properties

- Richer set than UNIX: RWX
- P(permission) O(owner) D(delete), read (RX), change (RWXO), full control (RWXOPD)

Access Control Lists Roundup

Advantages:

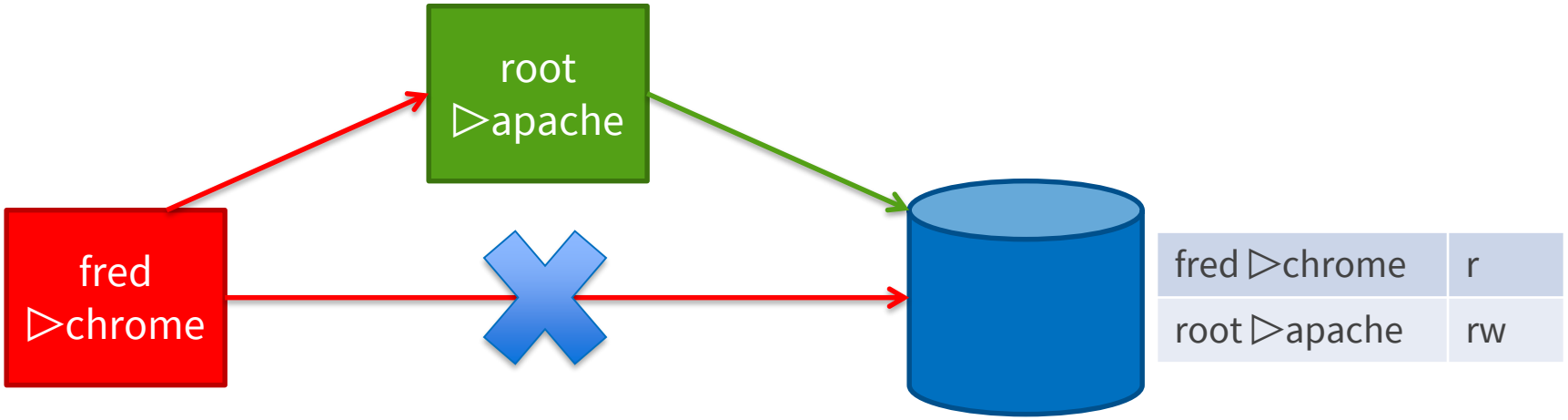
- Efficient review of permissions for an object
- Centralized enforcement is simple to deploy, verify
- Revocation is straightforward

Disadvantages:

- Inefficient review of permissions for a principal
- Large ACLs take up space in object
- Vulnerable to confused deputy attack

Confused Deputy Attack

- Process A does not have permission to write file F, but it can communicate with process B
- Process B has permission to write file F
- Process A tricks process B into writing file F with a value process A supplies
- Example: SQL injection and cross-site scripting



Capability Lists

The capability list for a *principal* is a list

e.g., ⟨dac.tex, {r,w}⟩ ⟨dac.pptx, {r,w}⟩

Capabilities carry privileges:

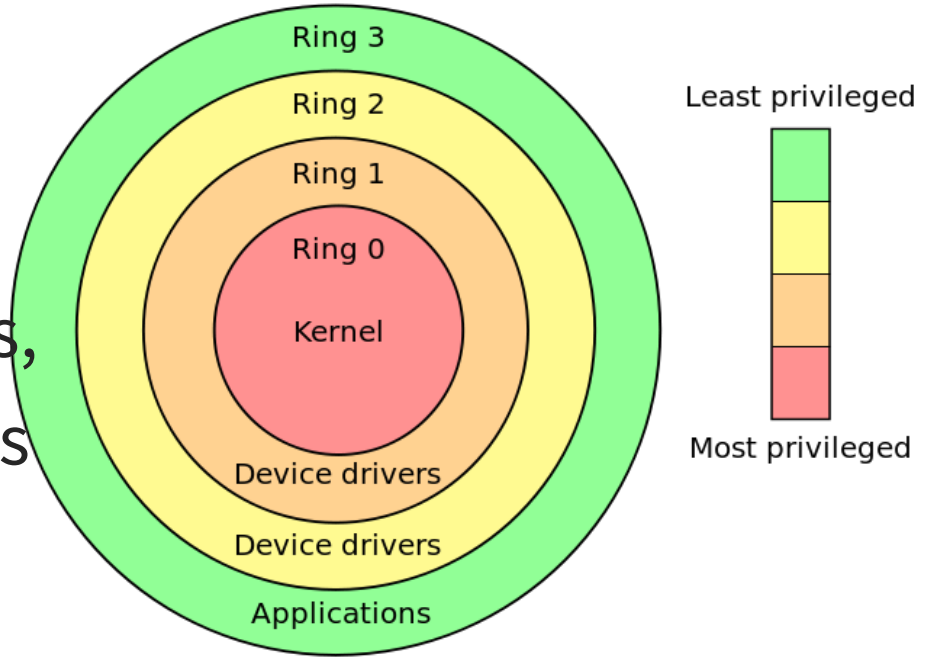
1) **Authorization:** Performing operation on object requires a principal to hold a capability such that
2) **Unforgeability:** Capabilities cannot be counterfeited or corrupted.

Note: Capabilities are (typically) transferable

C-Lists

OS maintains a list of capabilities for each principal (process)

1) **Authorization:** OS mediates access to objects, checks process capabilities
2) **Unforgeability:** capabilities are stored in protected memory region (kernel memory)



Access Control in UNIX

UNIX: has user and group identifiers: `uid` and `gid`

Per process: protection domain = `rvr|faculty▷sh`

Per file: ACL `owner|group|other` → stored in i-node

- Only owner can change these rights (using chmod)
- Each i-node has 3x3 RWX bits for user, group, others
- 2 mode bits allow process to change across domains
  - setuid, setgid bits

(Hybrid!) Approximation of access control scheme:

- Authorization (check ACL) performed at `open`
- Returns a file handle → essentially a capability
- Subsequent `read` or `write` uses the file handle

Capabilities Roundup

Advantages:

- Eliminates confused deputy problems
- Natural approach for user-defined objects

Disadvantages:

- Review of permissions?
- Delegation?
- Revocation?

ACLs vs Capabilities

	ACLs: For each Object: <P_1,$privs_1$> <P_2,$privs_2$>…	**Capabilities:** For each Principal: <O_1,$privs_1$> <O_2,$privs_2$>…
Review rights for object O	Easy! Print the list.	Hard. Need to scan all principals' lists.
Review rights for principal P across all objects	Hard. Need to scan all objects' lists.	Easy! Print the c-list.
Revocation	Easy! Delete P from O's list.	If kernel tracks capabilities, invalidates on revocation. Harder if object tracks revocation list.

History of Discretionary Access Control (DAC)

1760+	early philosophical pioneers of private property (Blackston, Bastiat,+)
1965	"access control lists" coined @ MIT describing Multics (CTSS foreshadowed ACLs) (Daley & Neumann)
1966	"capability" coined and OS supervisor outlined @ MIT (Dennis & van Horn)
1974	early computer security: "the user gives access rights at his own discretion" (Walter+)
1983	DoD's Orange book coins the term "discretionary access control"