

9/28/26

COMP 435:

Computer Security Concepts



Lecture 16: Race Conditions, Access Control



The University
of North Carolina
at Chapel Hill

<https://tinyurl.com/comp435-fa26>

Plan for Class Today



- 10:10-10:25 – Race Conditions
 - 10:35-11:00 – Access Control
-

Guiding Question 😊

How does the operating system create isolation and manage access?



Race conditions

A type of vulnerability!



Race Condition

Def'n: Concurrent access of a resource is not serializable

Occurs when:

- Two or more processes have access to **the same resource**
- The final state of the resource depends on the **relative timing** of the **two processes**



ATM



Bank



balance = 900

withdraw \$100



get_balance()



balance = 900



\$100



set_balance(800)



balance = 800



balance = 900

w/d \$100



get_bal()



bal = 900



get_bal()



bal = 900



w/d \$50



\$100



set_bal(800)



set_bal(850)



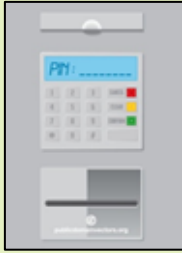
\$50



balance = ??



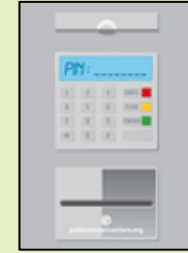
ATM



Bank



ATM



balance = 900

w/d \$100



get_bal()



bal = 900



\$100



set_bal(800)



get_bal()



bal = 900



w/d \$50



set_bal(850)



\$50



balance = ??

Problem!!



- The bug is **not**: “two users accessed the account at once”
- The bug is that the supposedly single operation **withdraw** was implemented as multiple interleavable operations.
 - Caused inconsistent behavior!!
- **Potential solution:**
 - Use locking/transactions so only one withdrawal updates the balance at a time
- Let's look at a race condition in some code...
 - You launch this attack in lab 2!

The open syscall



```
filename = "tmpname";
```

// Write to the file

```
fd = open(filename, O_CREATE | O_RDWR);
```

Symlink System Call



- creates a symbolic link (soft link) in the filesystem.
- Ex: `symlink(target, linkpath);`
 - If you open and read the linkpath, now the system is going to follow the link and read target
 - After running this you have, linkpath → target

Example Race Condition Attack



```
filename = "tmpname";
```

```
    symlink("/etc/passwd", "tmpname");
```

```
// Write to the file
```

```
fd = open(filename, O_CREATE|O_RDWR);
```

Program A: Privileged Program!

```
1. filename = "tmpname";  
   // Write to the file  
2. fd = open(filename, O_CREATE | O_RDWR);
```

Program B: Attacker!



```
1. symlink("/etc/passwd", "tmpname");
```

**Problem: TOCTTOU
Race condition!**

Time of Check to Time of Use (TOCTTOU)



Def'n: a particular type of race condition in which the gap between checking permissions and using permissions can be exploited



Under the hood of the open syscall

```
if (access (fname, W_OK) == 0) {  
  
    fd = open (fname, O_WRONLY) ;  
}
```

W_OK == 0, if the current
process has write permission!



Another version of the attack

```
if (access (fname, W_OK) == 0) {  
    → unlink (fname);  
    symlink ("\\etc\\passwd", fname);  
  
    fd = open (fname, O_WRONLY);  
}
```

Key to any type of race condition
is that two entities have access to
the same resource

Design Principles (Saltzer and Schroeder)



- economy of mechanism
- fail safe defaults
- complete mediation
- open design
- separation of privilege
- least privilege
- least common mechanism
- psychological acceptability

Answer Q4 on the WS
from last time



Making Authorization Decisions



Recall: OS Responsibilities

- manage resources
- provide interface to HW
- provide user authentication
- mediate interprocess communication
- protect itself

*All of these responsibilities are
working in service of
security policies*

Ex: Process A is authorized to
access memory region A, but not
memory region B.



Access Control

**Which users can access which resources and
with which rights**

Access Control



Who

How

Which users can access which resources and with which rights

What

Access Control



Subject

Right/Type

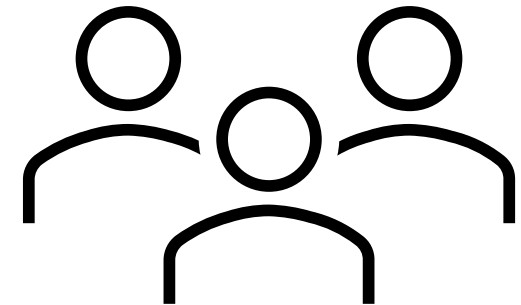
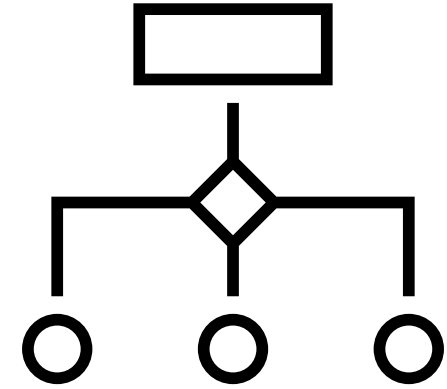
Which users can access which resources and with which rights

Object

Subjects



- users
- processes



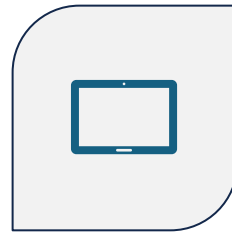
Objects



FILES



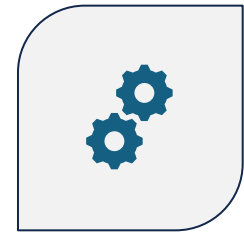
MEMORY



I/O DEVICES



USERS



PROCESSES

Objects



- files
- memory
- I/O devices
- users
- processes

} subjects

Rights



- read
- write
- execute
- create
- transfer



Best Practices

Best Practice



- universal application
- least privilege
- type checking

Universal Application



Def'n: every access of an object by a subject should be checked

Ex: In practice: using cached results is a bad idea!

Also in practice... we use cached results all the time!

Universal Application



Def'n: every access of an object by a subject should be checked

Non-Universal Application

- random checking: randomly select a subset of accesses to check
- random auditing: log all accesses, then later randomly check some. Only provides detection, not prevention.
- selective checking: if a cheap-to-implement first glance seems suspicious, do a more thorough check

Least Privilege



Def'n: every subject should be granted the least amount of access necessary to do its job

Supports the idea of Failing Safe.

Limiting the amount of privilege a subject has, limits the damage they can do if they are compromised or turn out to be malicious.

Challenge: Need to know up front exactly what access each subject will need.

Result: Designers often over-privilege an entity (ex. A process) to ensure availability.

Type Checking



Def'n: operations should be meaningful for the object accessed

A type system, broadly speaking, defines the kinds of objects that will exist and the legal operations on those objects



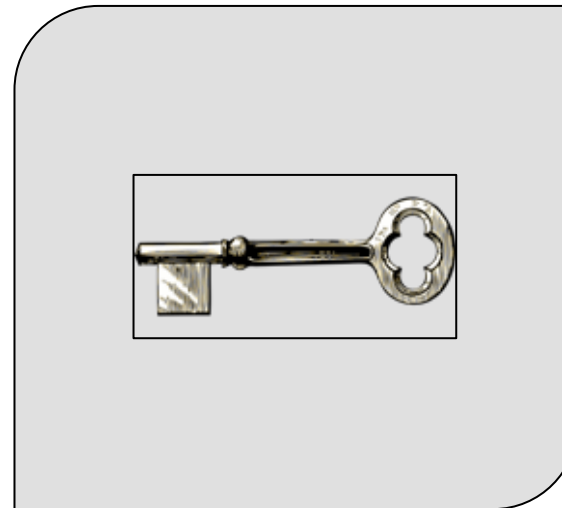
Mechanisms for Making Authorization Decisions

Mechanisms for Making Authorization Decisions



	BIBLIOG	TEMP	F	HELPTXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USERA	ORW	ORW	ORW	R	X	X	R	W
USERB	R	-	-	R	X	X	R	W
USERS	RW	-	R	R	X	X	R	W
USERT	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Access Control



Capabilities

Is this process
authorized to access
this resource?



Access Control Policies

Access Control Policies



Discretionary
Access
Control (DAC)

Role-Based
Access
Control
(RBAC)

Attribute-Based
Access
Control
(ABAC)

Mandatory
Access
Control (MAC)

Access Control Matrix

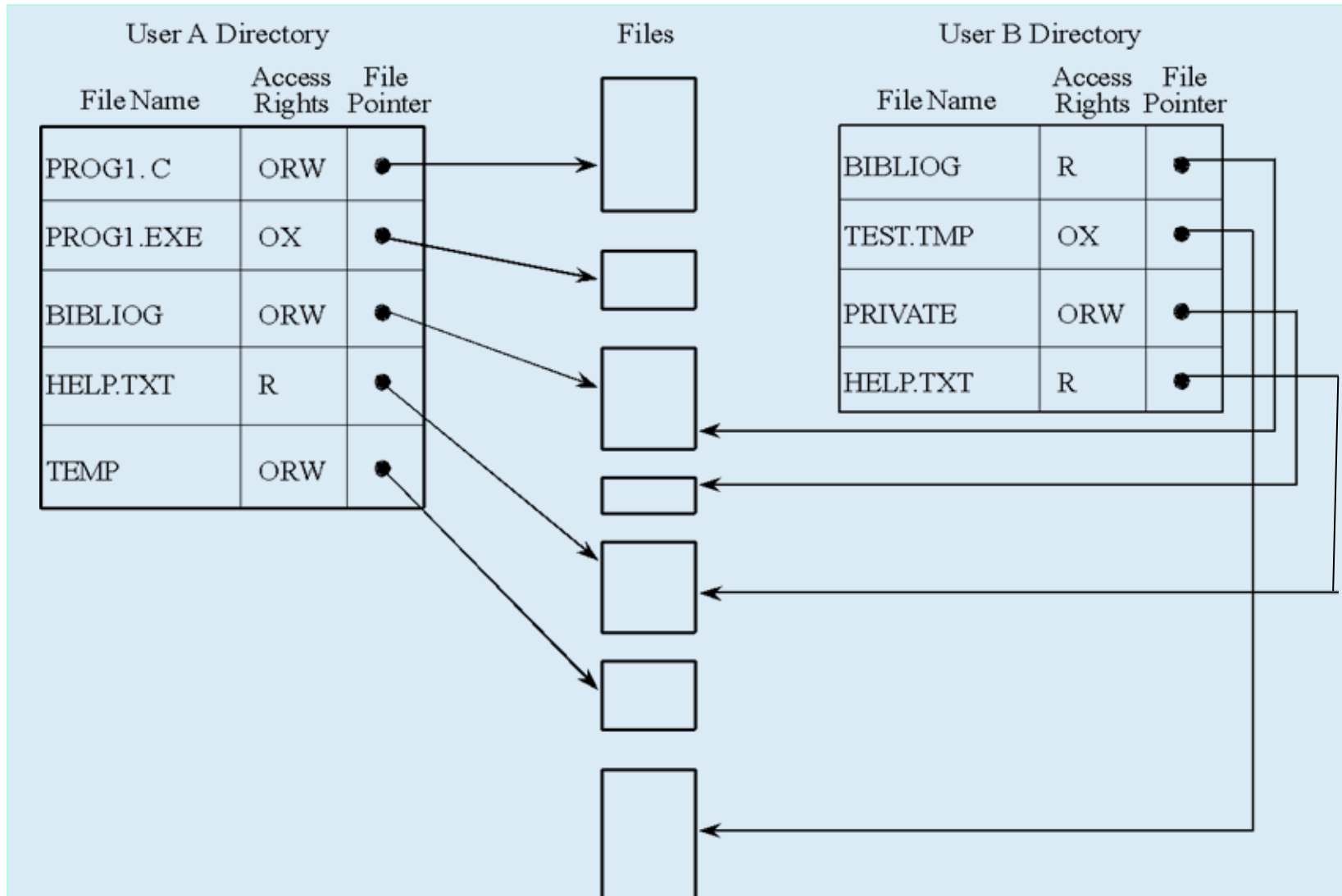


Objects

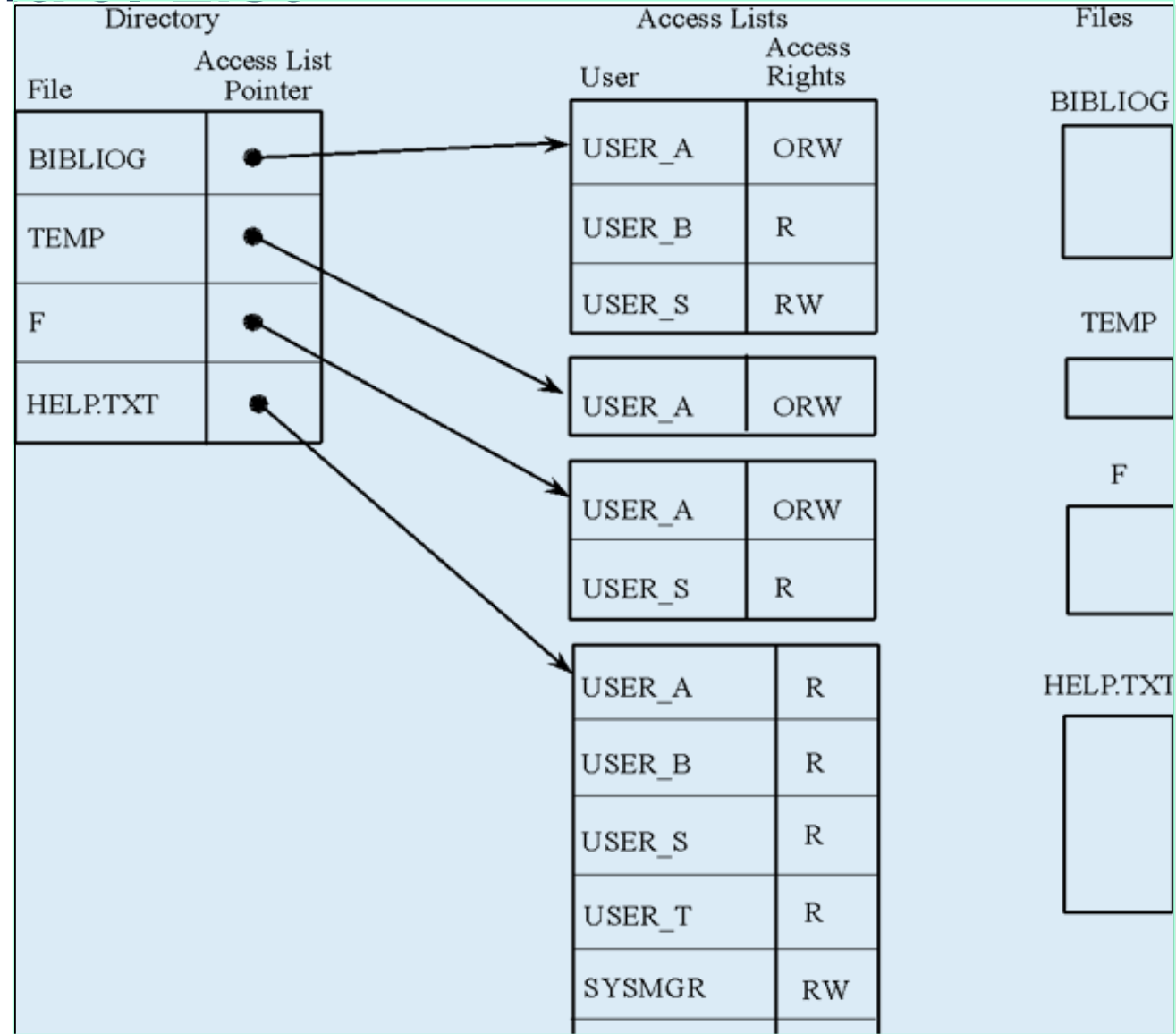
Subjects

	BIBLIOG	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Access Control Directory



Access Control List



DAC Mechanisms



- + easy
- + revocation
- + no aliasing
- sparse matrix

Access Control
Matrix

- + easy
- + fixes sparsity
- file revocation
- aliasing
- long lists

Access Control
Directory

- + file revocation
- + no aliasing
- + default rights
- user
revocation

Access Control List