Engineering &
Computer Science
Syracuse University

CIS 657 – Principles of Operating Systems

Topic: Persistence – Access Control

Endadul Hoque

Slide Acknowledgment

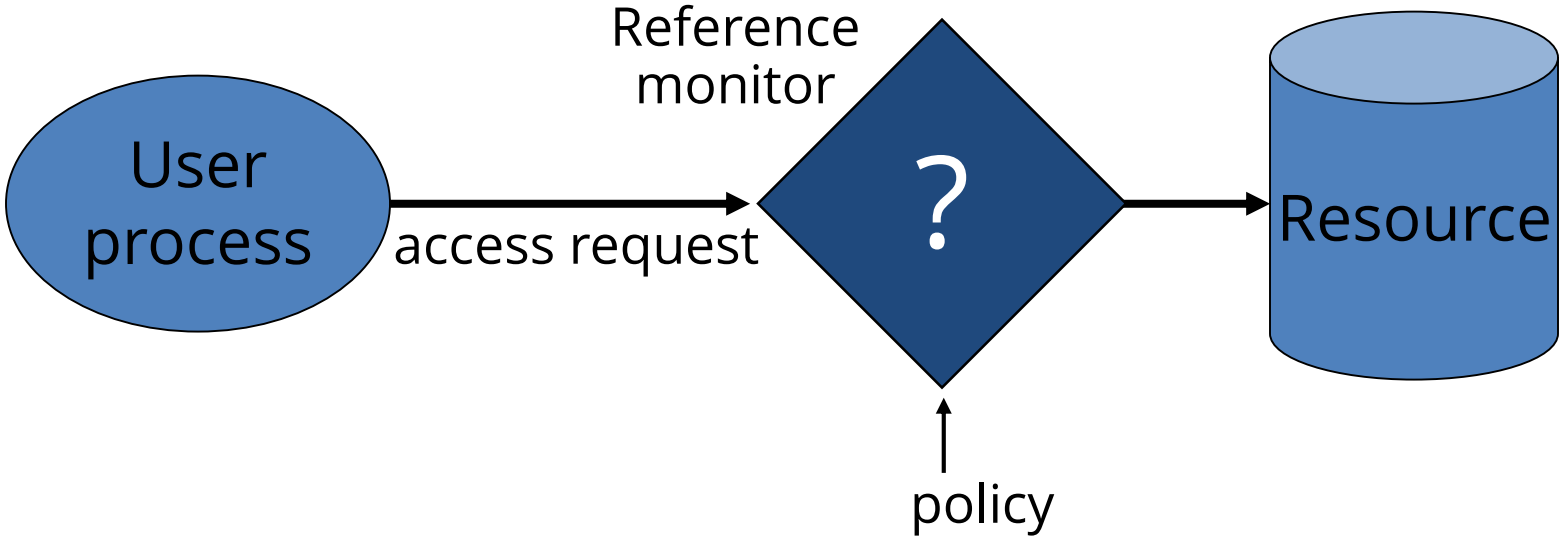
- Contents are based on slides from Ninghui Li (Purdue), John Mitchell (Stanford), Bogdan Carbunar (FIU)

OS Security

UNIX ACCESS CONTROL

Access Control

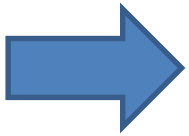
- A **reference monitor** mediates all access to resources
  - Principle: Complete mediation: control **all** accesses to resources



UNIX access control

- What access control concepts are used?
  - Truncated access control list
  - A form of role-based access control

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			



	File 1	File 2	…
Owner	read	write	-
Group	write	write	-
Other	-	-	read

UNIX access control

- Each process has a **user ID**
  - Inherit from Parent process
  - Process can change **ID**
    - Restricted set of options
  - Special "**root**" ID
    - All access allowed

	File 1	File 2	…
Owner	read	write	-
Group	write	write	-
Other	-	-	read

- File has access control list (**ACL**)
  - Grants permission to users via roles
  - Three "roles": **owner**, **group**, **other**

Unix File Access Control List

- Each file has an **owner** and a **group**
- Permissions set by **owner**
  - Permissions: Read, write, execute
  - Roles: Owner, group, other
  - Represented by vector of
    four octal values

- Only **owner** and **root** can change permissions
  - This privilege ***cannot be delegated or shared***
- Setid bits – Discuss in a few slides

Example directory listing

access	*owner*	*group*	*size*	*modification*	*name*
-rw-rw-r--	pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	pbg	staff	9423	Feb 24 2012	program.c
-rwxr-xr-x	pbg	staff	20471	Feb 24 2012	program
drwx--x--x	tag	faculty	512	Jul 31 10:31	lib/
drwx------	pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	pbg	staff	512	Jul 8 09:35	test/

Process User IDs (Motivation)

- A regular user must be able to change his/her password
- There is a helper program "**passwd**" in Linux to help users change their password
- There is a file "**/etc/shadow**" that stores the (hashed) passwords

```
prompt> ls -l /etc/shadow
-rw-r----- 1 root shadow 930 Jan 31 14:49 /etc/shadow
```

- **How would you solve this problem?**
  - Allow any user to read/write the file?

Process 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**)
  - Set based on the set-user-ID (setuid) bit on the file/program being executed, or by system call
  - determine the permissions of 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

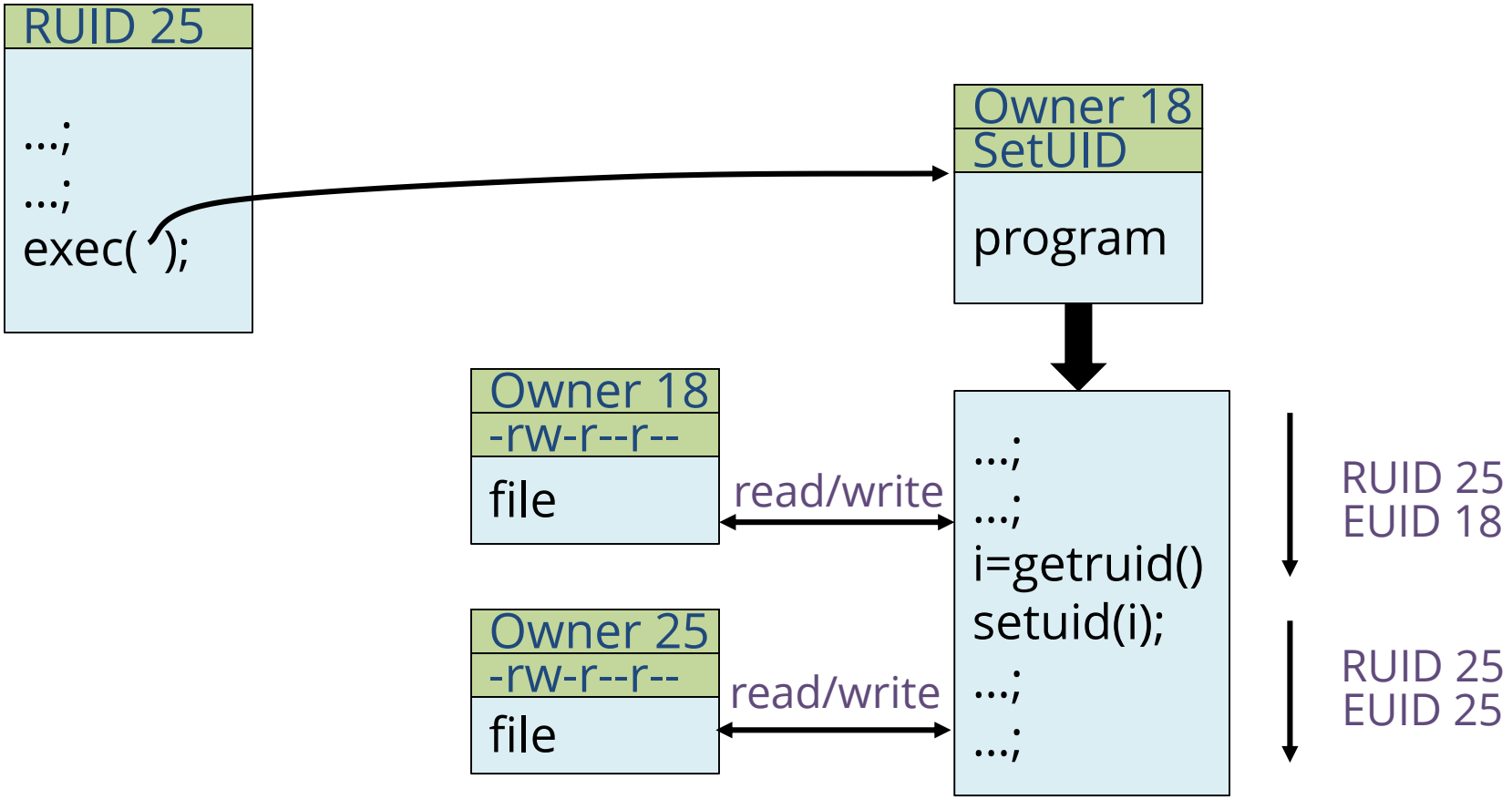
- Root user
  - **ID=0** for superuser/root; can access any file
- Fork() and Exec()
  - Inherit three IDs
- System calls
  - **seteuid(newid)** can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID is root
  - **setuid (newid)** can set EUID to
    - Real ID or saved ID, if EUID is not root
    - **Change all IDs** to an arbitrary ID, if EUID is root
- Details are more complicated
  - Several different calls: setresuid, 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

Used only for directories

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

The need for setid bits

- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on “privileged ports” (TCP/UDP ports below 1024)
  - non-root users need these privileges
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

Security issues with setid bits

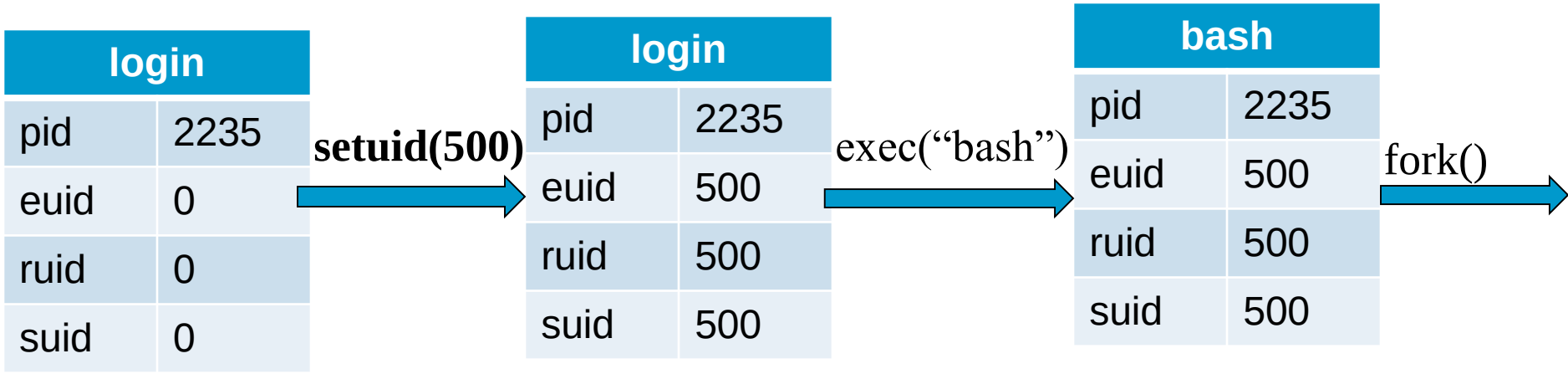
- These programs are typically **set-user-ID-root**
- Violates the **least privilege principle**
  - every program and every user should operate using the least privilege necessary to complete the job

- Why violating least privilege is bad?
- How would an attacker exploit this problem?
- How to solve this problem?

Drop privilege

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid

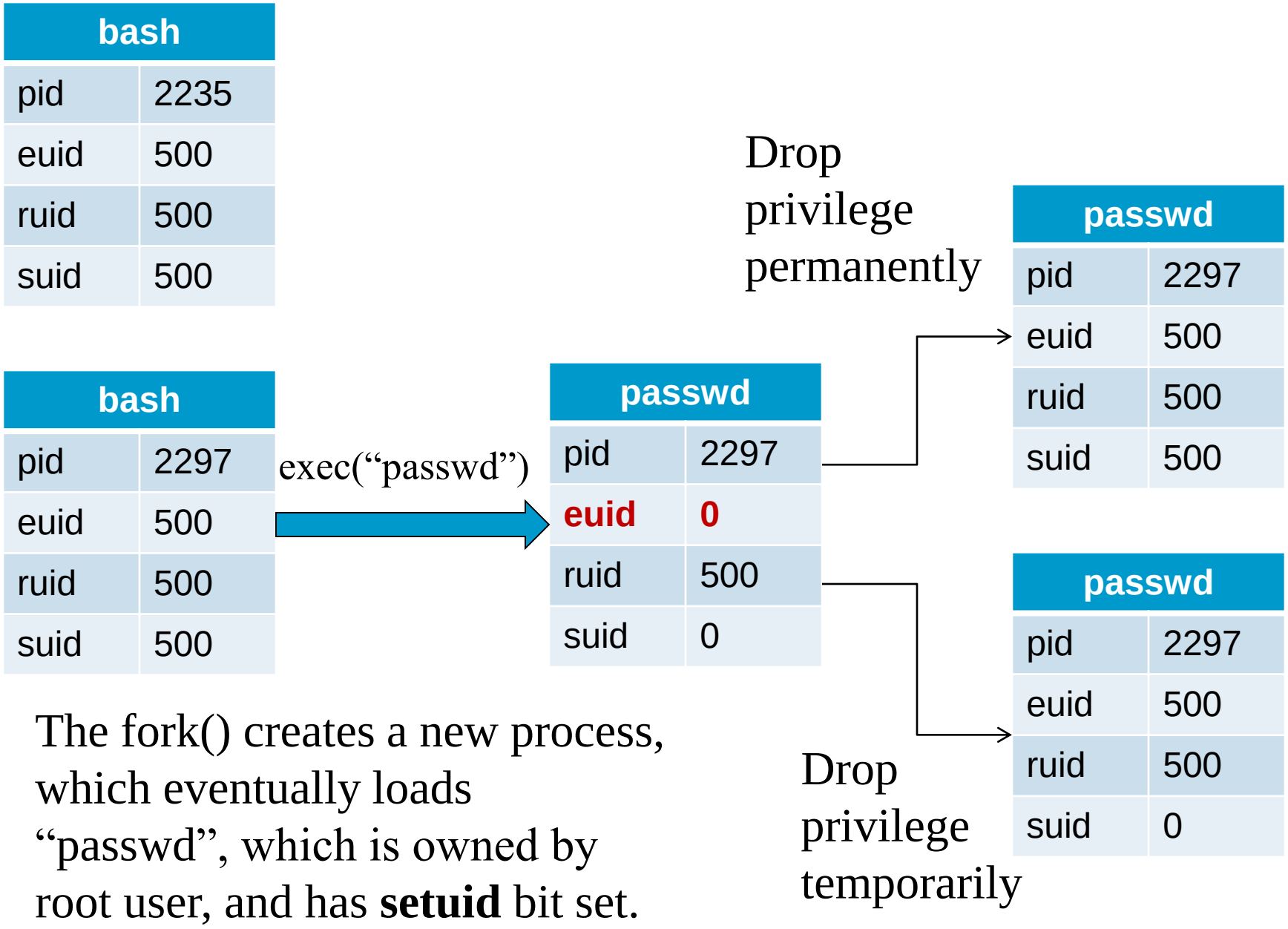
Example: User login



After the login process verifies that the entered password is correct, it issues a setuid system call.

The login process then loads the shell, giving the user a login shell.

The user types in the **`passwd`** command to change his password.



The fork() creates a new process, which eventually loads "passwd", which is owned by root user, and has **setuid** bit set.

Other Scenarios

- Accesses other than read/write/execute
  - Who can change the **permission** bits?
    - The owner or the root
  - Who can change the **owner** of a file/directory?
    - Only the superuser/root
- Rights not related to a file
  - Affecting another process
  - Operations such as shutting down the system, mounting a new file system, listening on a low port
    - traditionally reserved for the root user

Unix access control summary

- Good things
  - Some protection from most users
  - Flexible enough to make practical systems possible

- Main limitation
  - Coarse-grained ACLs – user, group, other
  - Too tempting to use root privileges
  - No way to assume some root privileges without all

Questions