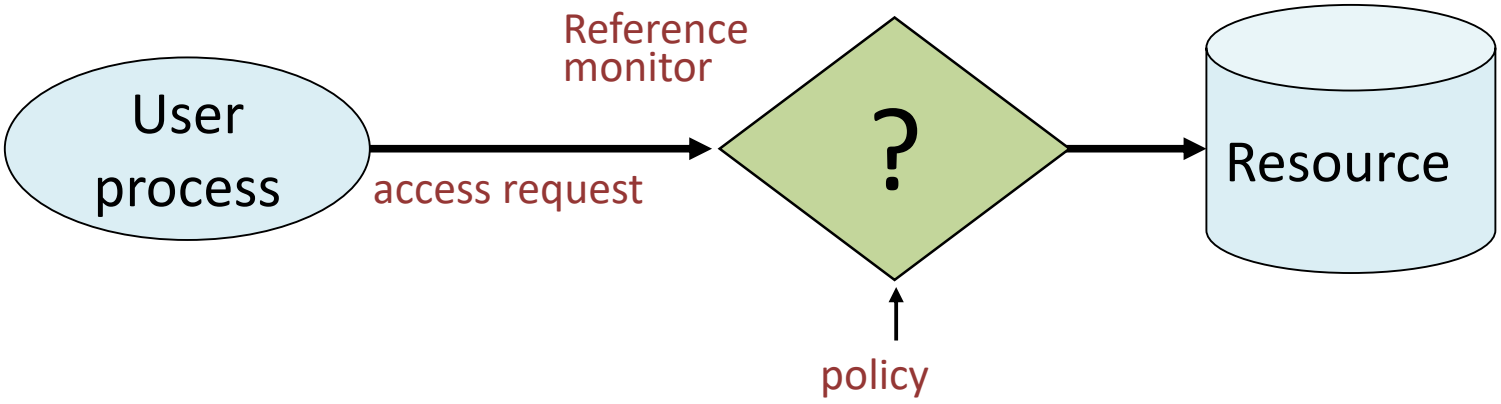
ΥΣ13 - Computer Security

Access Control

Κώστας Χατζηκοκολάκης

Access control

- **Goal:** allow access to resources only to authorized users
- Assumptions
  - Resource access only via a reference monitor
  - System knows who the user is (authentication)



Access control matrix

Objects

Subjects

	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

Access control matrix

	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	rw	r
Alice	x	x	rw	–
Bob	rx	r	r	r

User	Operating System	Accounts Program	Accounting Data	Audit Trail
Sam	rwx	rwx	r	r
Alice	rx	x	–	–
Accounts program	rx	r	rw	w
Bob	rx	r	r	r

Access control matrix

- Access control list (ACL)
  - Associate list with each object (matrix column)
  - Check user/group against list
  - Authentication is required
  - eg. Unix
- Capability
  - Unforgeable “ticket” to a resource
  - Random bit sequence
    - Can be passed from one process to another
    - Authentication is not necessary
  - eg. Sharing via a link

ACL : my name is on the list

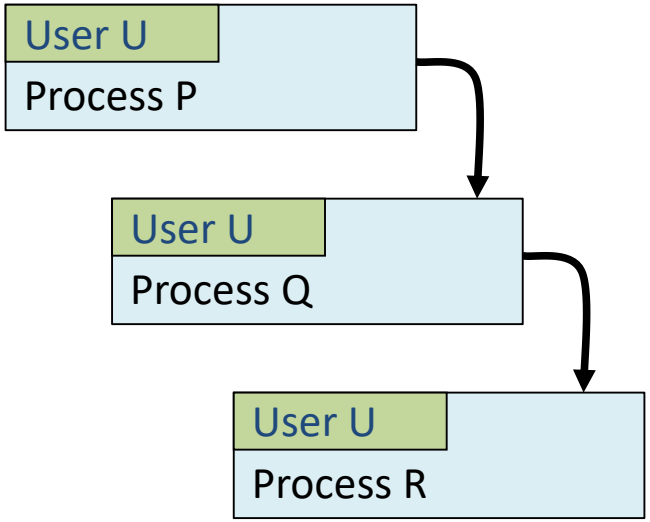
Capability : I have a ticket

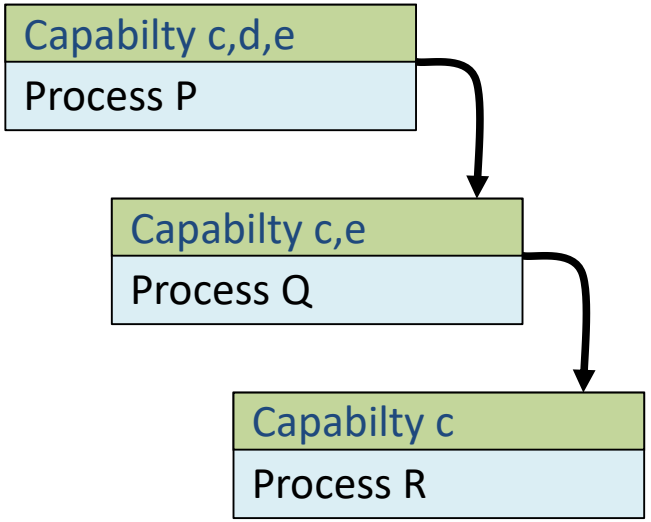
ACL vs capabilities

- 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 (unix?)
- Revocation
  - ACL: Remove user or group from list
  - Cap: unlink ticket from resource
    - revokes all access

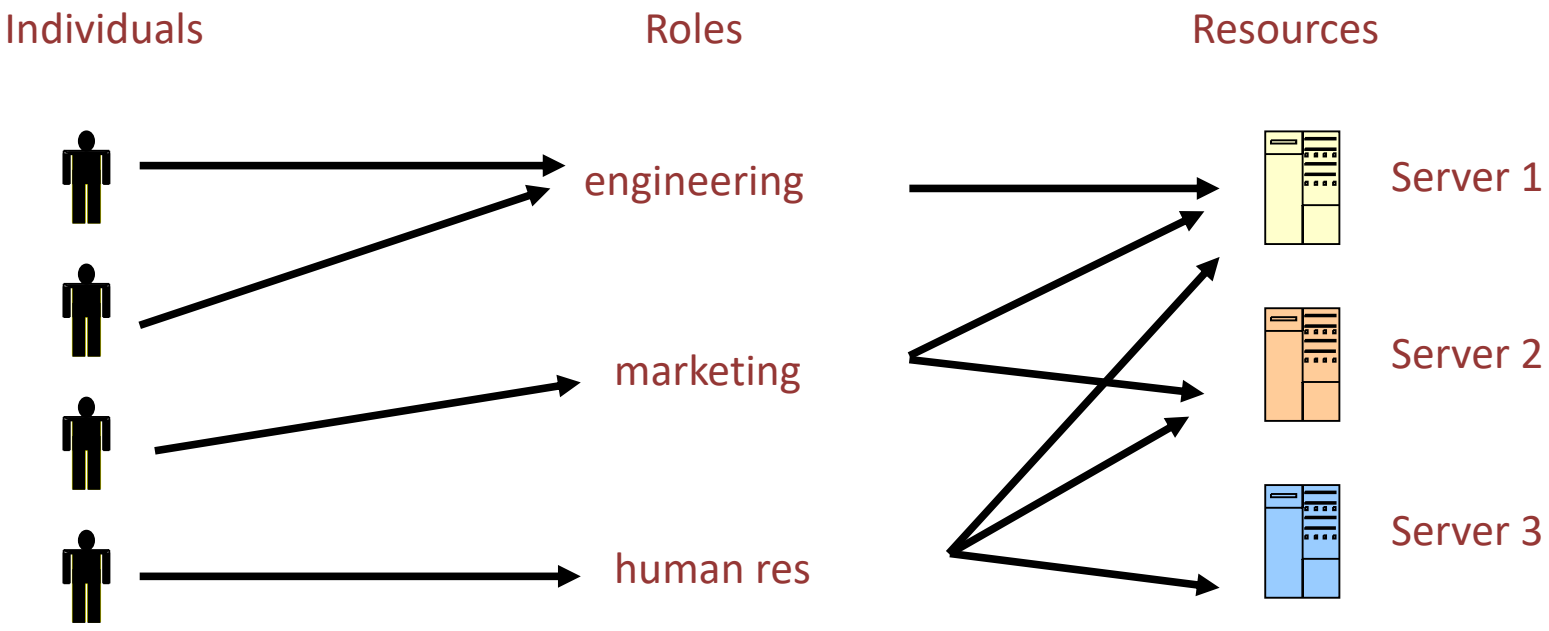
ACL vs capabilities : process creation

- ACL: inherit parent UID
- Cap: no UID concept, capabilities transferred





Roles and groups



Advantage: users change more frequently than roles

Unix

- ACL (limited to 3 permissions per file)
- A form of role-based access control

	File 1	File 2	…
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
…			
Role r	Read	write	write

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

Unix

- Process runs under UID
  - Inherit from process
  - Process can change id
- Special “root” id
  - All access allowed
- ACL associated to each file
  - Three “roles”: owner, group, other

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

Unix ACL

- Each file has owner and group
- Permissions
  - Read, write, execute
- Give to
  - Owner, group, other
- Only owner, root can change permissions
  - This privilege cannot be directly delegated

$$\underbrace{\text{rwx}}_{\text{ownr}}\ \underbrace{\text{rwx}}_{\text{grp}}\ \underbrace{\text{rwx}}_{\text{othr}}$$

Unix ACL

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/

Unix ACL problems

- Auditing is hard
- Gives access to user, not program
- Permissions for shared directory (eg `/tmp`)?
- Cannot express state

Solutions?

Give permission to a program

- Goal
  - prevent Alice from directly accessing `/var/lib/database`
  - but allow to run `/bin/mysql`
  - and allow `/bin/mysql` to access `/var/lib/database`
- Idea
  - `/bin/mysql` : owner db-user, permisisons `rwxr-xr-x`
  - `/var/lib/database` : owner db-user, permisisons `rw-r--r--`
- Does this work?

setuid/setgid bits

- setuid bit
  - run process with the UID of the file owner
- setgid bit
  - run process with the GID of the file owner
- Solves the mysql problem
  - set setuid for `/bin/mysql`
  - Alice can execute it
  - It runs as db-user, so it can access `/var/lib/database`

Sticky bit

- Anyone with write access to dir can delete files (even if not owner)
- Problem
  - Shared directories (eg. `/tmp`)
- Solution: sticky bit
  - 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 delete files in the directory

chmod

- setuid
  - `chmod u+s file`
  - `chmod u-s file`
- setgid
  - `chmod g+s file`
  - `chmod g-s file`
- Sticky
  - `chmod +t dir`
  - `chmod -t dir`

Is this ok?

- The program has root uid (via setuid)
- It wants to check whether a user has access to the file

```
if (access("file", W_OK) == 0) {
    fd = open("file", O_WRONLY);
    write(fd, buffer, sizeof(buffer));
}
```

Is this ok?

- The program has root uid (via setuid)
- It wants to check whether a user has access to the file

```
if (access("file", W_OK) == 0) {
    fd = open("file", O_WRONLY);
    write(fd, buffer, sizeof(buffer));
}
```

Time-of-Check-to-Time-of-Use! (TOCTTOU)

User id of process

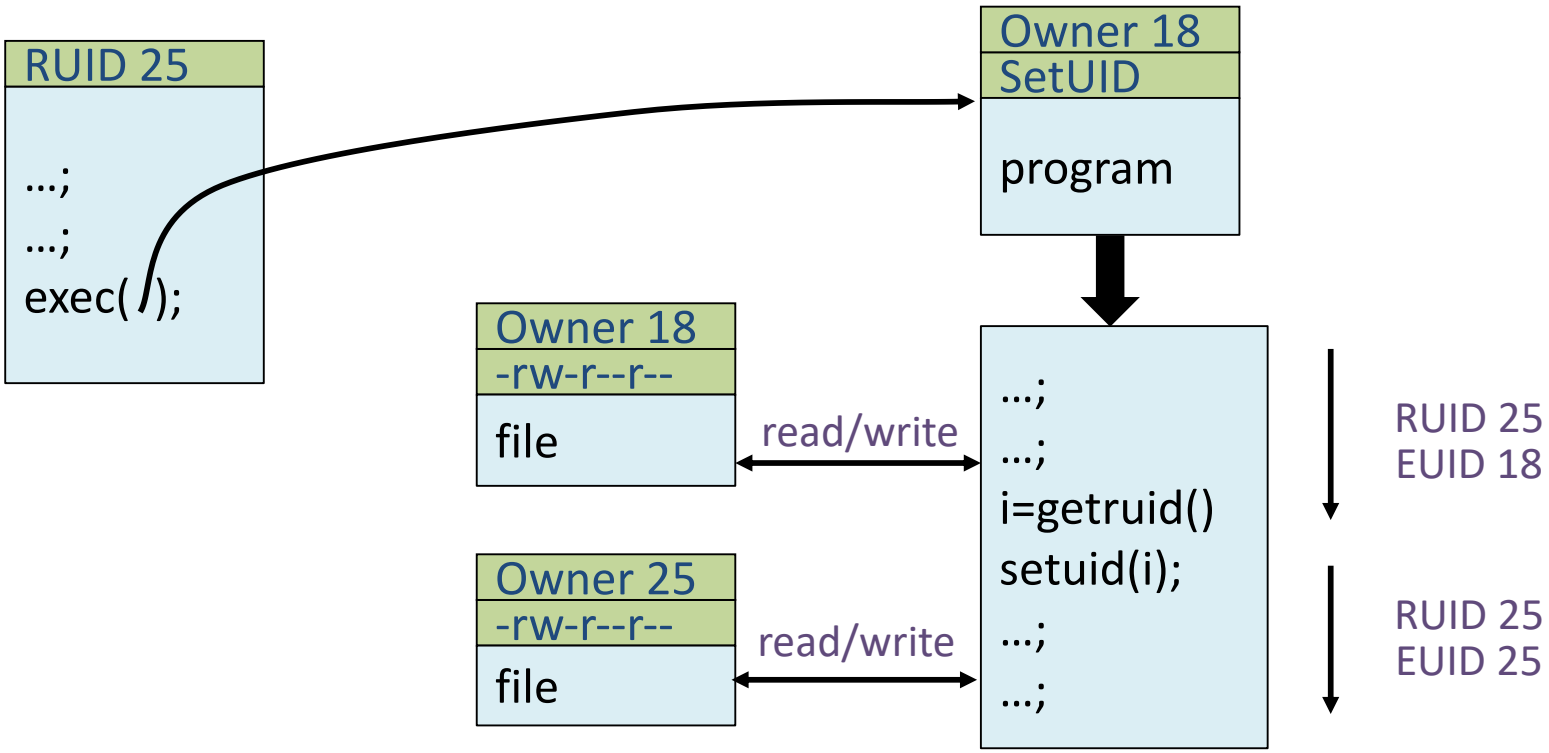
Each process has three Ids:

- Real user ID (RUID)
  - inherited from parent
- Effective user ID (EUID)
  - from `setuid` bit on the file being executed, or sys call
  - determines the permissions for process
- Saved user ID (SUID)
  - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

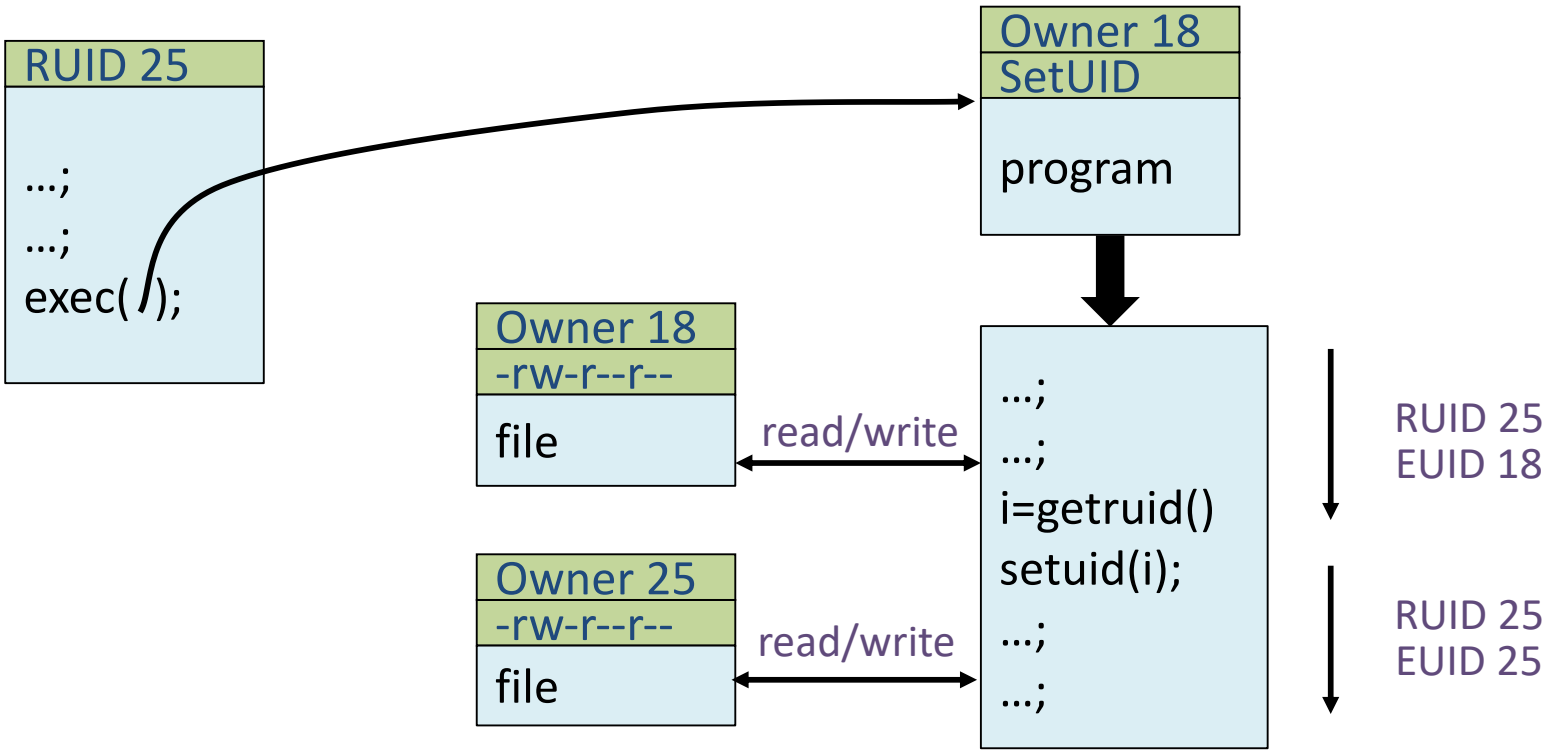
User id of process

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except when executing a file with `setuid` bit
- seteuid(newid) system call can set EUID to
  - Real ID or saved ID, regardless of current EUID
  - Any ID, if EUID is root

Avoid TOCTTOU



Avoid TOCTTOU



Also remember : permissions are checked only on open

Othe topics

- Containing a process
  - chroot (not safe for root processes!)
  - Sandbox
  - Virtualization
  - . . .

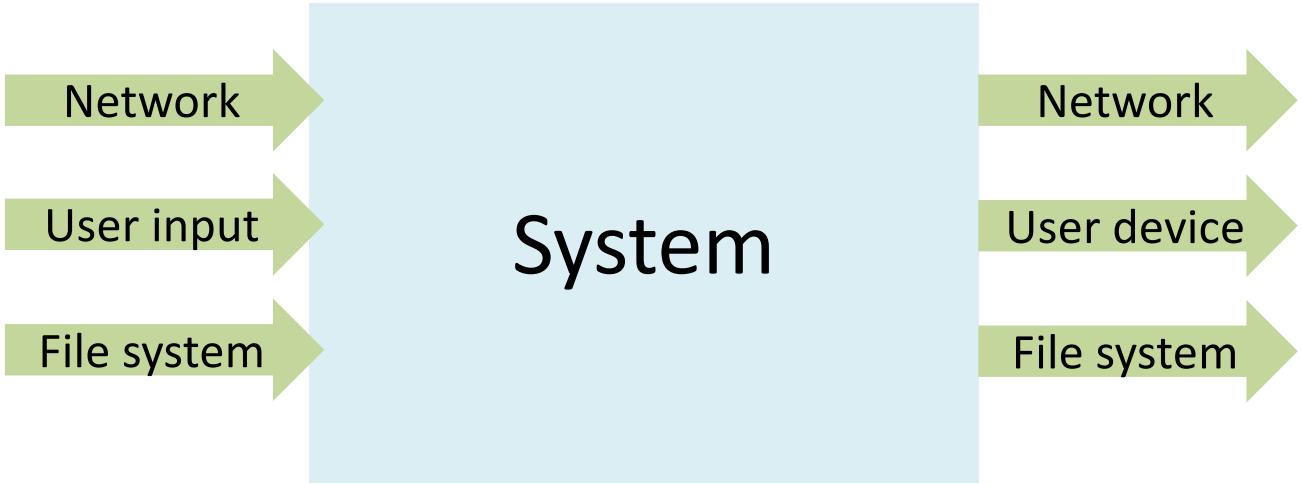
Othe topics

- Containing a process
  - chroot (not safe for root processes!)
  - Sandbox
  - Virtualization
  - . . .
- POSIX ACLs
  - Individual users, groups
  - `setfacl`
  - Backward compatibility: mask

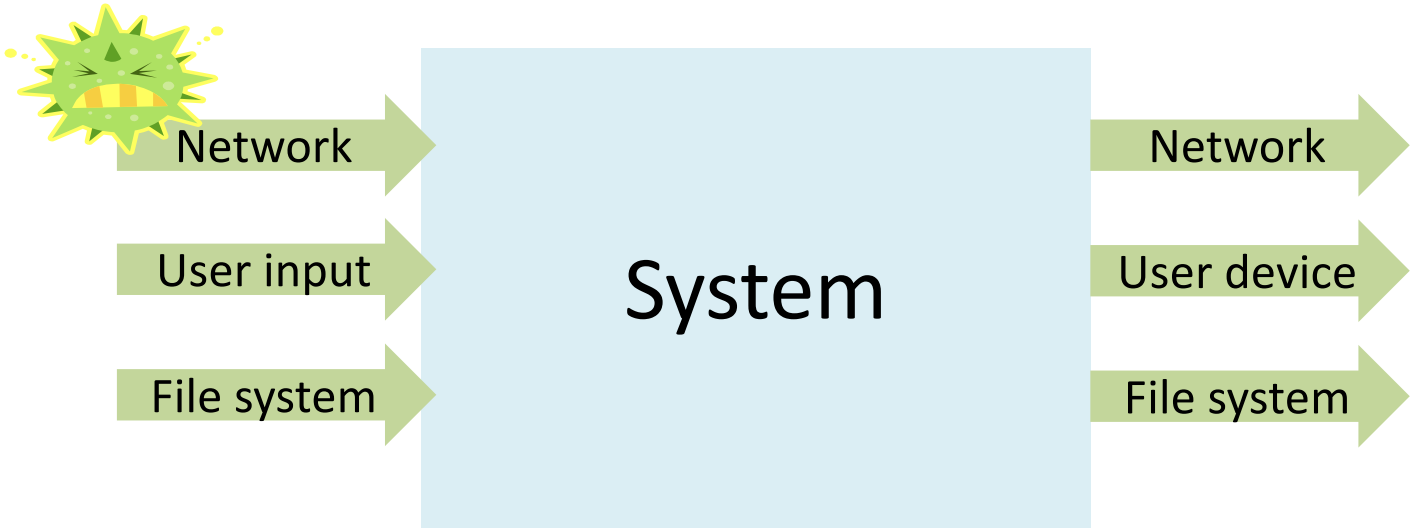
Principle of least priviledged

- A system module should only have the minimal privileges needed for its intended purposes
  - Ability to access or modify a resource
- Compartmentalization / isolation
  - Separate the system into isolated compartments
  - Limit interaction between compartments

Monolithic design

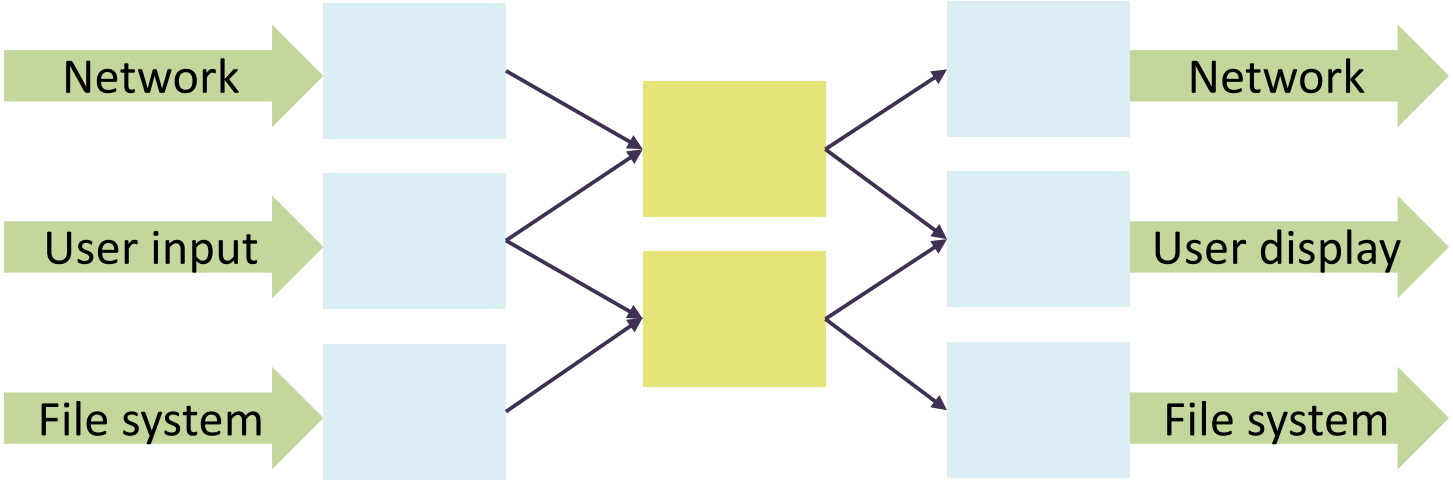


Monolithic design

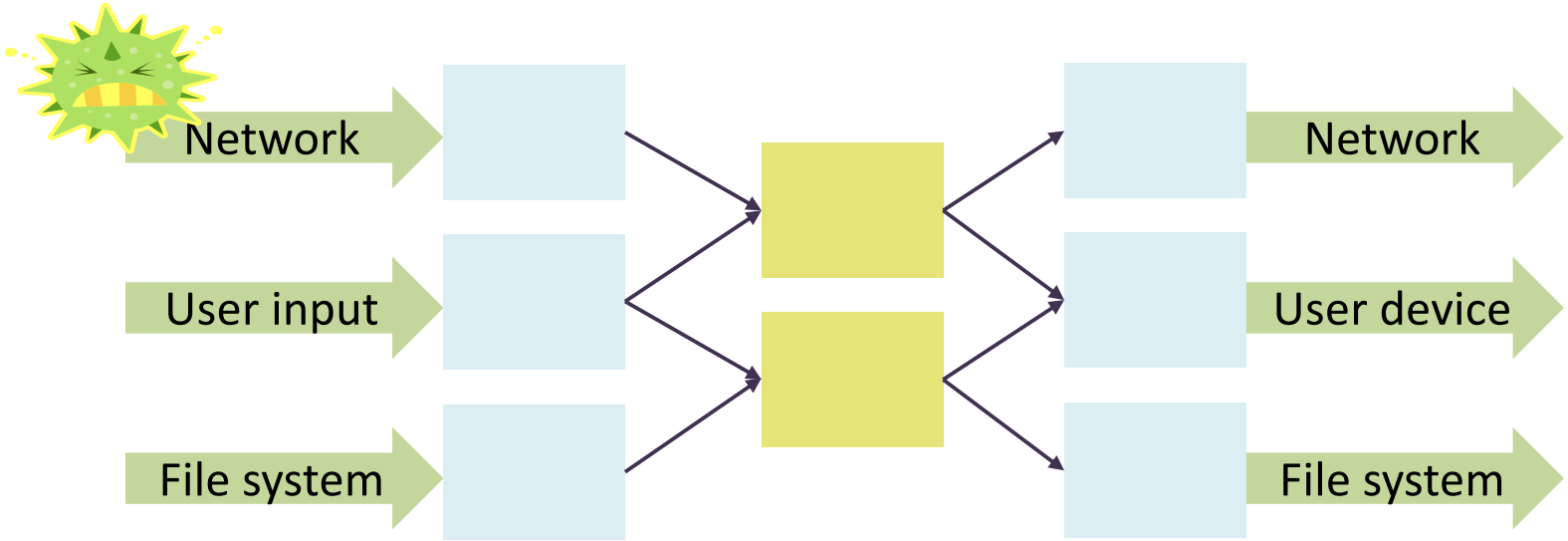


Monolithic design

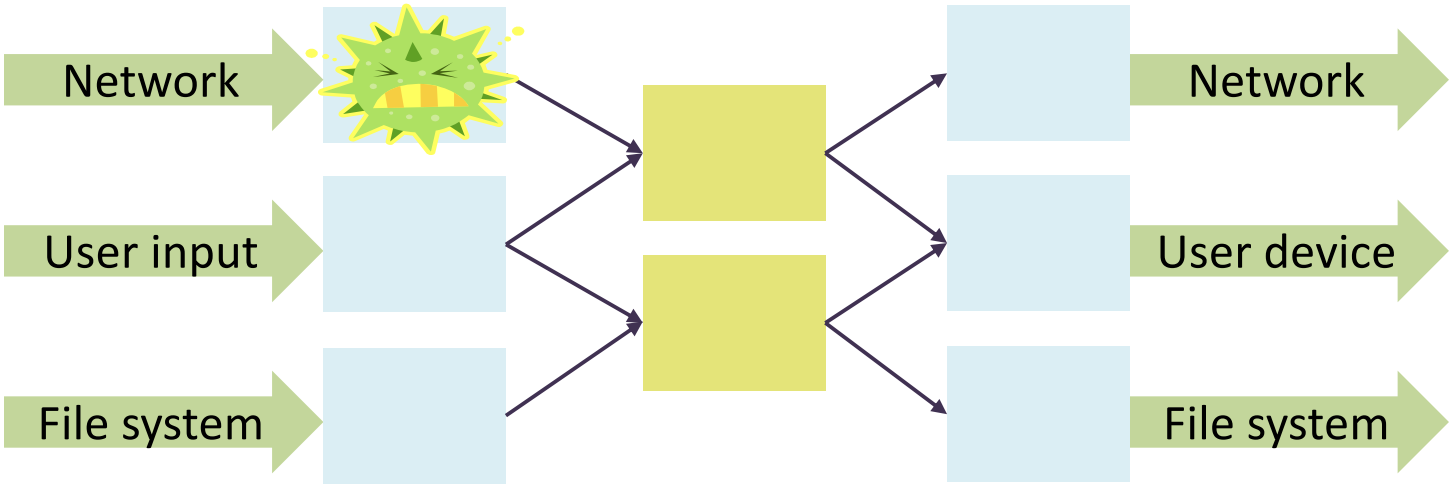
Component design



Component design



Component design



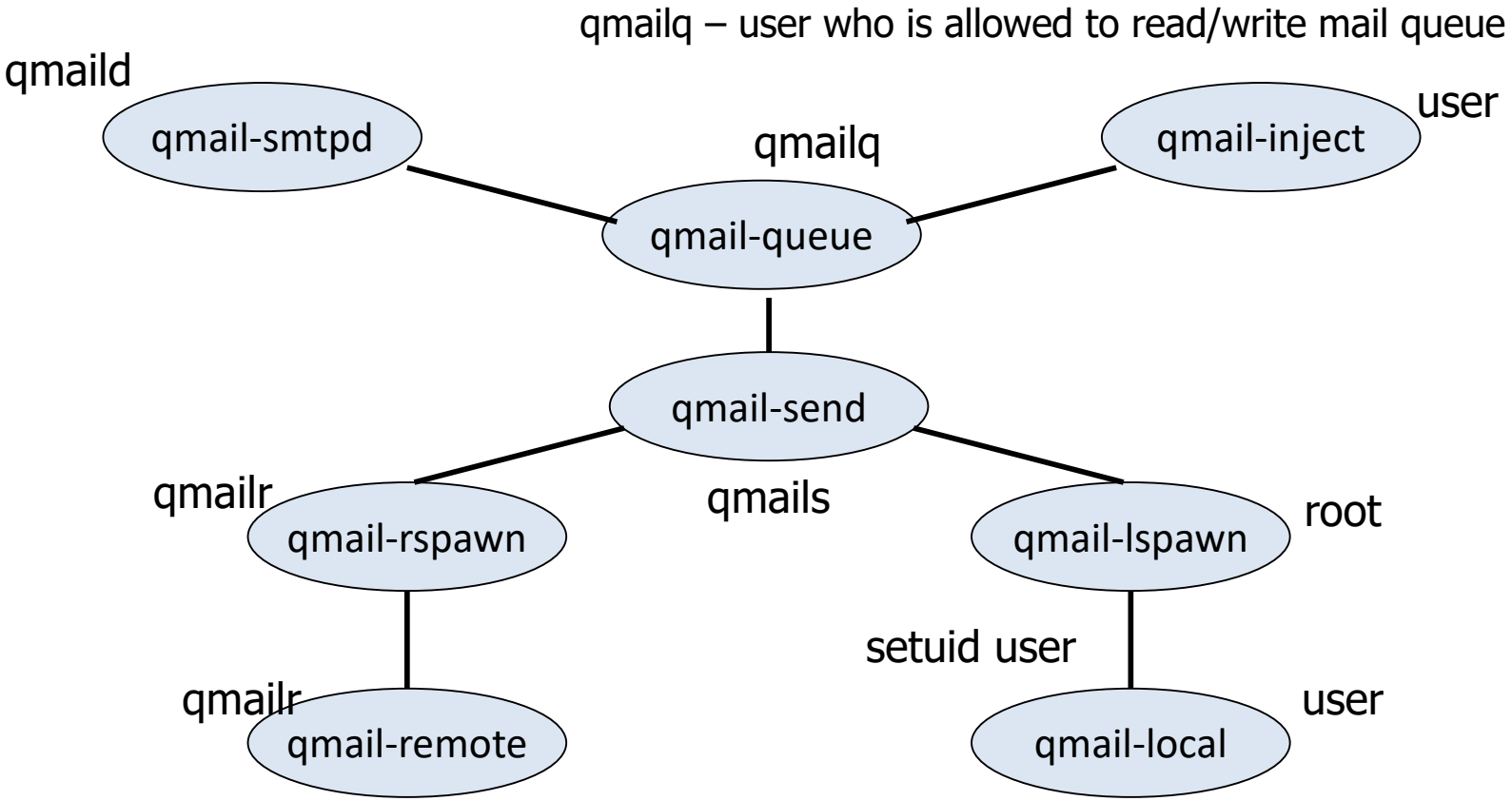
Example : email client

- Requirements
  - Receive and send email over external network
  - Place incoming email into local user inbox files
- Sendmail
  - Traditional Unix
  - Monolithic design
  - Historical source of many vulnerabilities
- Qmail
  - Compartmentalized design

Example : qmail

- Isolation based on OS isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Minimal privileges for each UID
  - Only one “setuid” program
    - setuid allows a program to run as different users
  - Only one “root” program
    - root program has all privileges

Example : qmail



References

- Ross Anderson, Security Engineering, Chapter 4
- Setuid Demystified
- POSIX Access Control Lists on Linux
- Fixing Races for Fun and Profit: How to use access (2)
- How to break out from various chroot solutions