Secure Architecture Principles

- Isolation and Least Privilege
- Access Control Concepts
- Operating Systems
- Browser Isolation and Least Privilege

Original slides were created by Prof. John Mitchel and Suman Janna
Some slides are from Prof. David Mazieres

Secure Architecture Principles

Isolation and Least Privilege

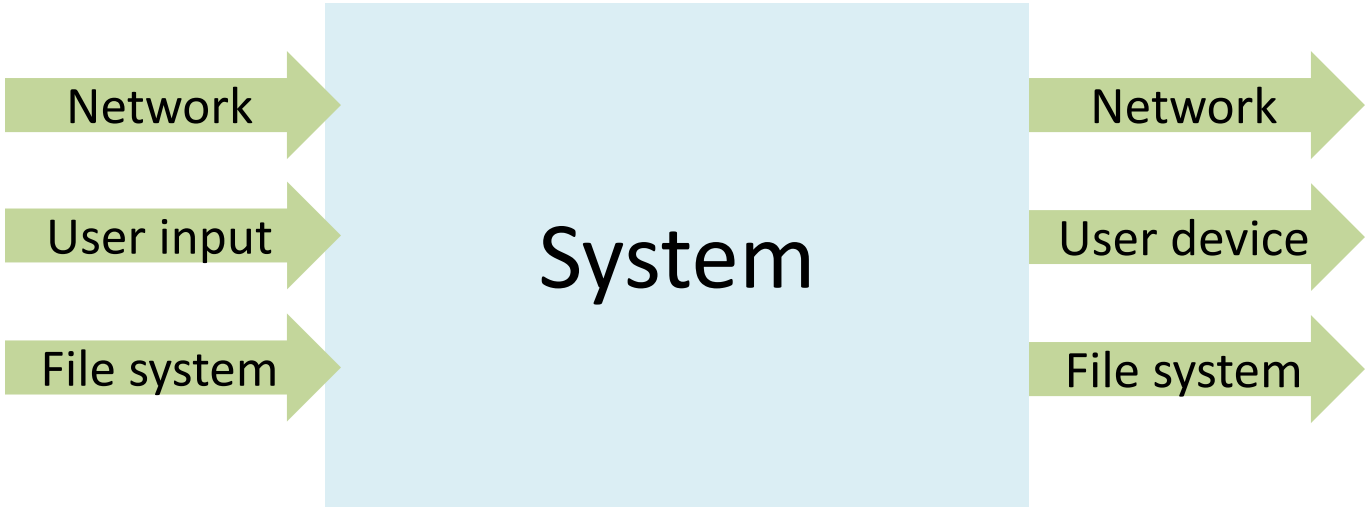
Principles of Secure Design

- Compartmentalization
  - Isolation
  - Principle of least privilege
- Defense in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Keep it simple

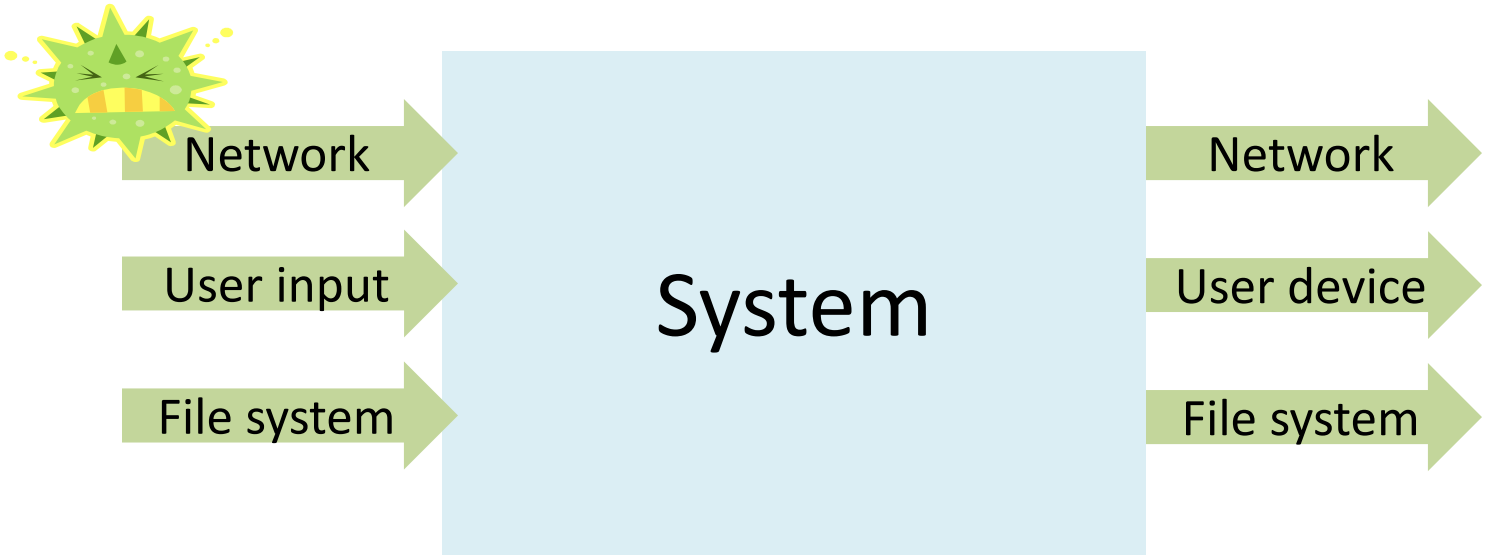
Principle of Least Privilege

- What's a privilege?
  - Ability to access or modify a resource
- Assume compartmentalization and isolation
  - Separate the system into isolated compartments
  - Limit interaction between compartments
- Principle of Least Privilege
  - A system module should only have the minimal privileges needed for its intended purposes

Monolithic design

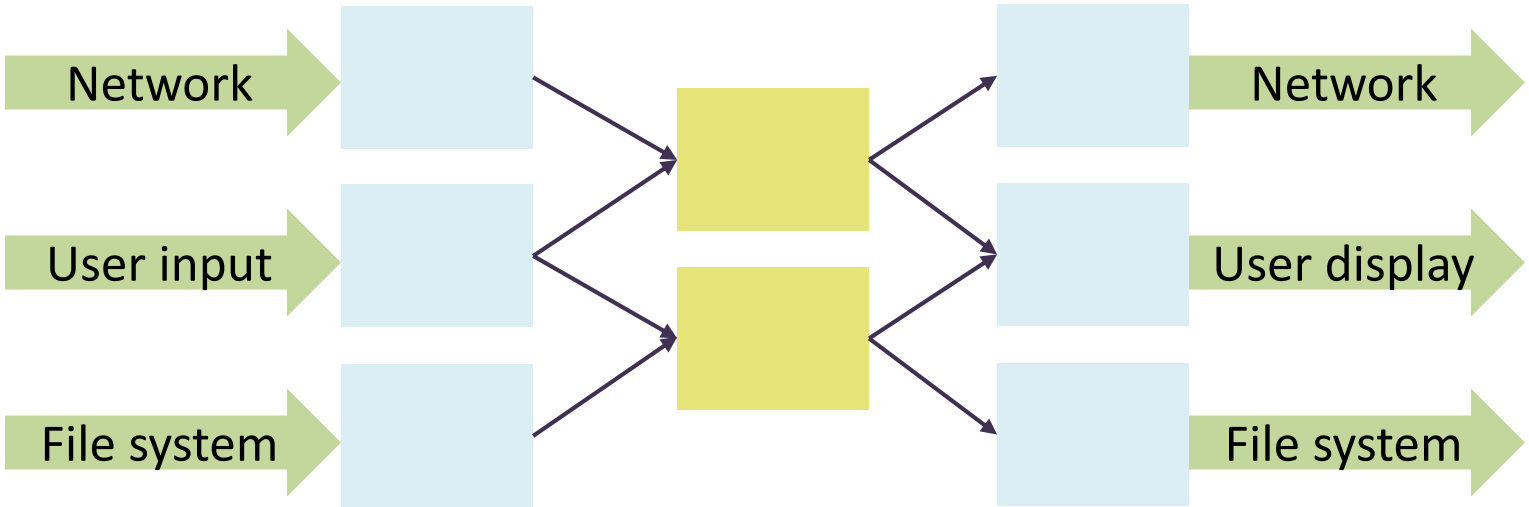


Monolithic design

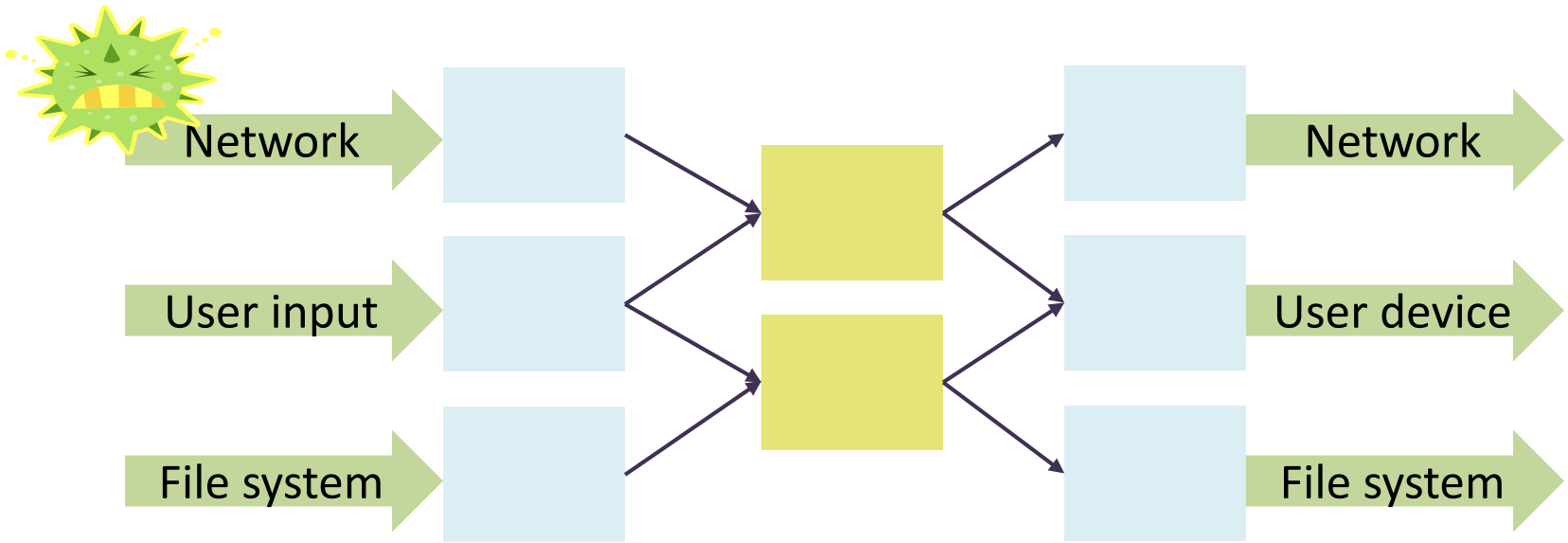


Monolithic design

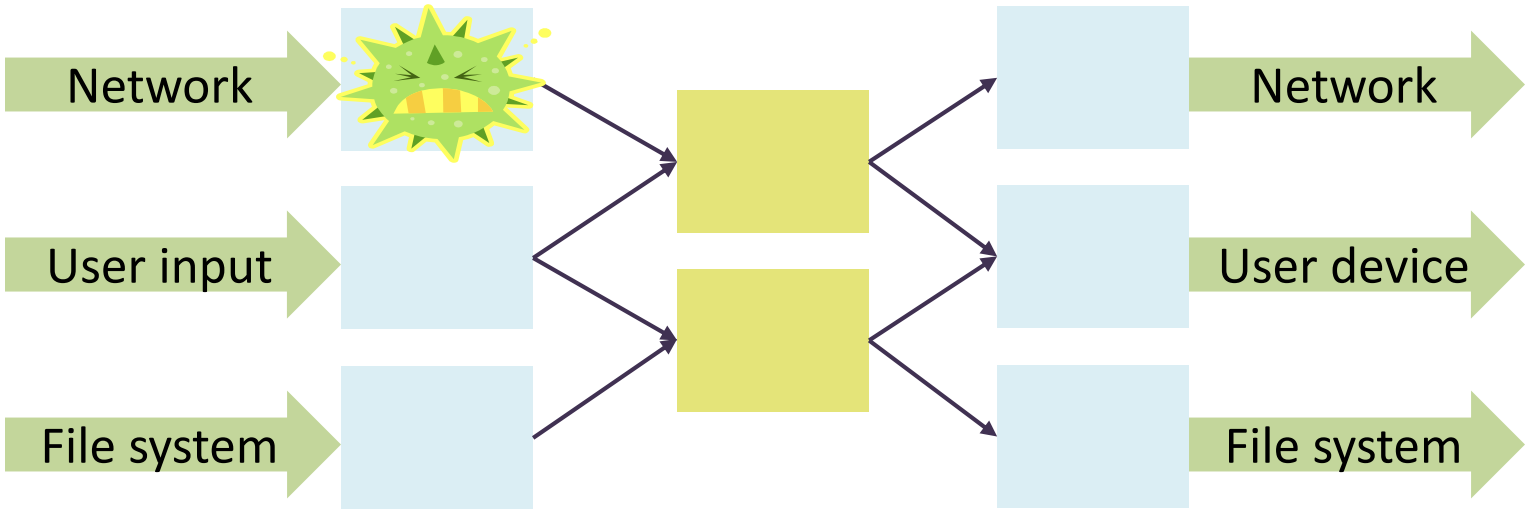
Component design



Component design



Component design



Principle of Least Privilege

- What's a privilege?
  - Ability to access or modify a resource
- Assume compartmentalization and isolation
  - Separate the system into isolated compartments
  - Limit interaction between compartments
- Principle of Least Privilege
  - A system module should only have the minimal privileges needed for its intended purposes

Example: Mail Agent

- 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

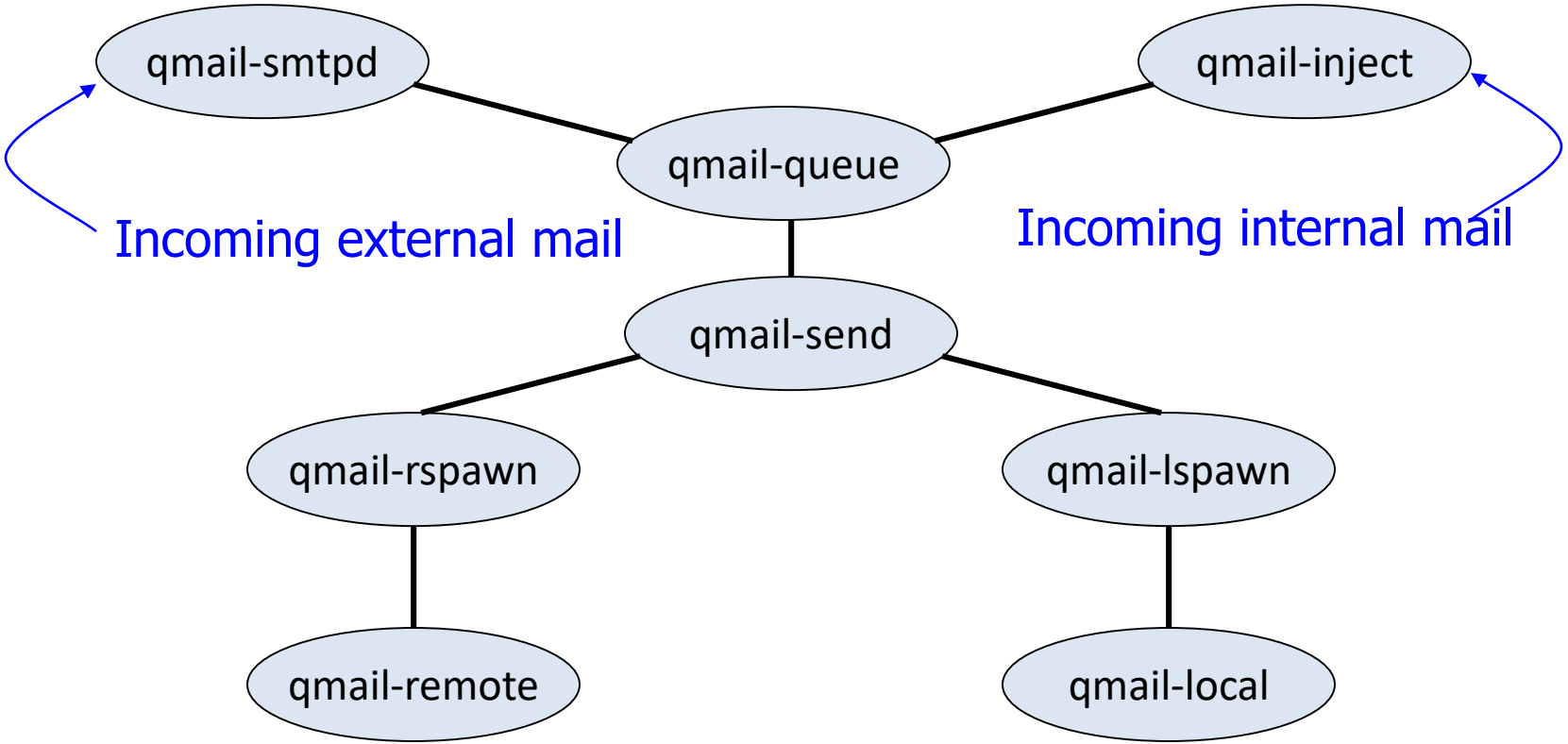
OS Basics (before examples)

- Isolation between processes
  - Each process has a UID
    - Two processes with same UID have same permissions
  - A process may access files, network sockets, ….
    - Permission granted according to UID
- Relation to previous terminology
  - Compartment defined by UID
  - Privileges defined by actions allowed on system resources

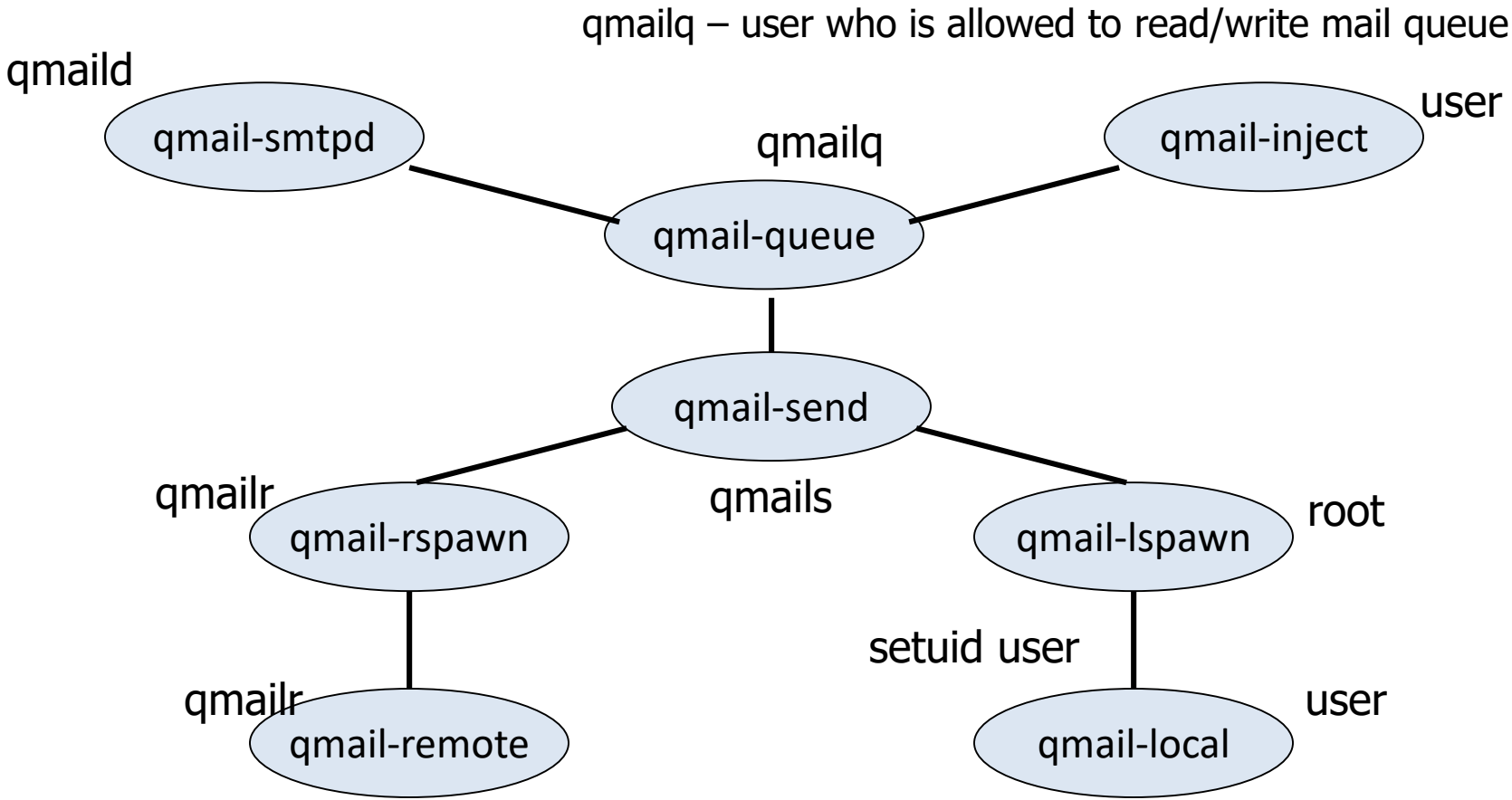
Qmail design

- 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

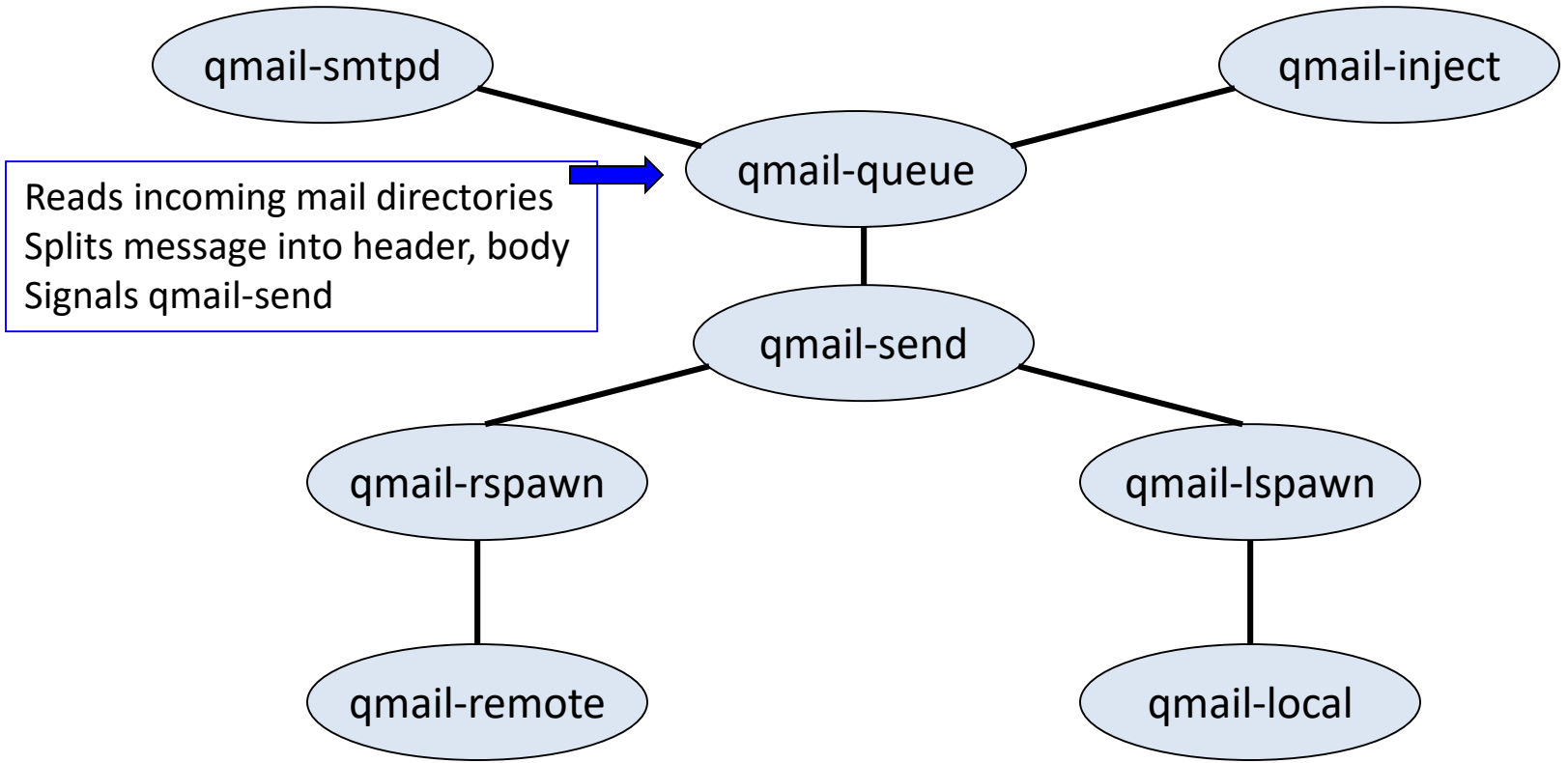
Structure of qmail



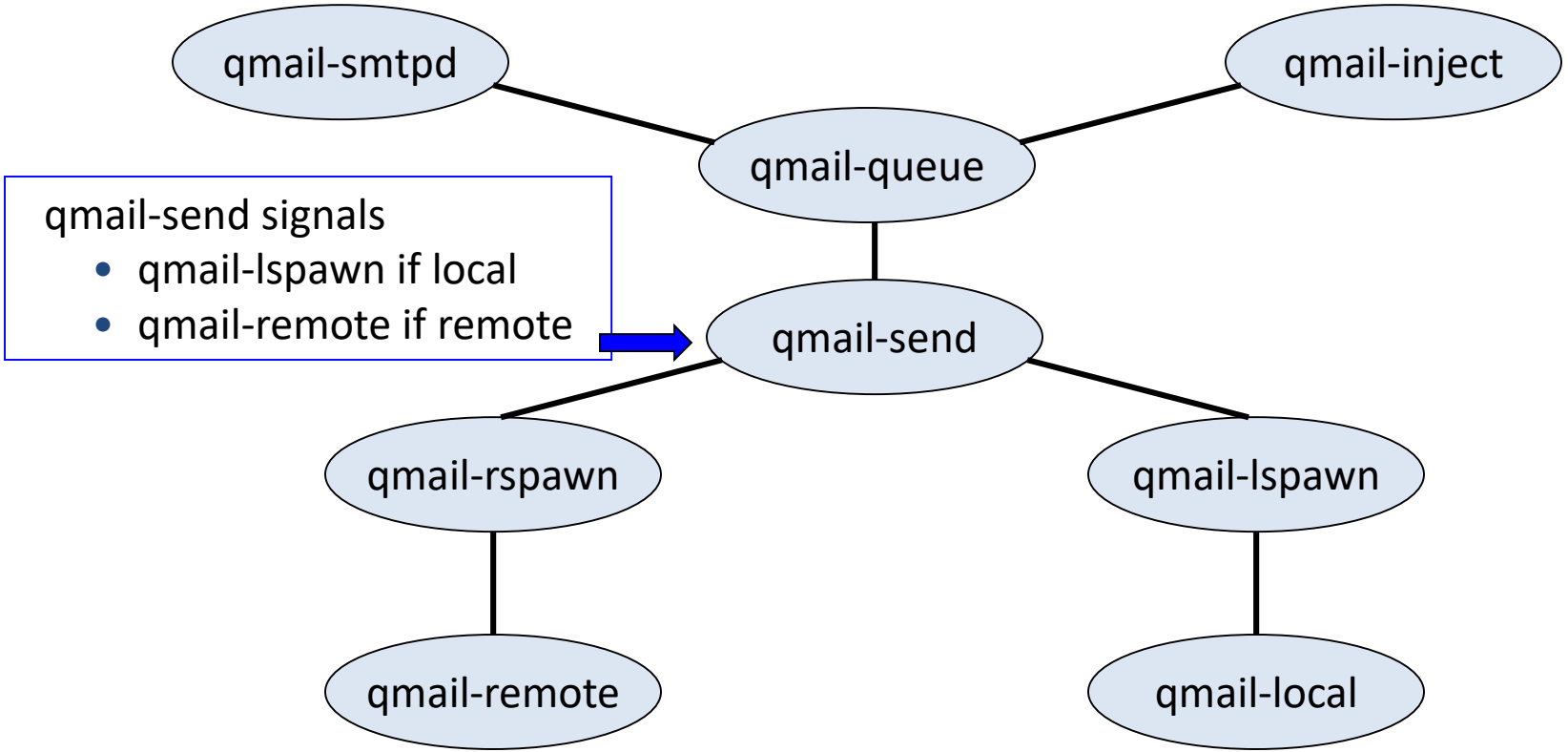
Isolation by Unix UIDs



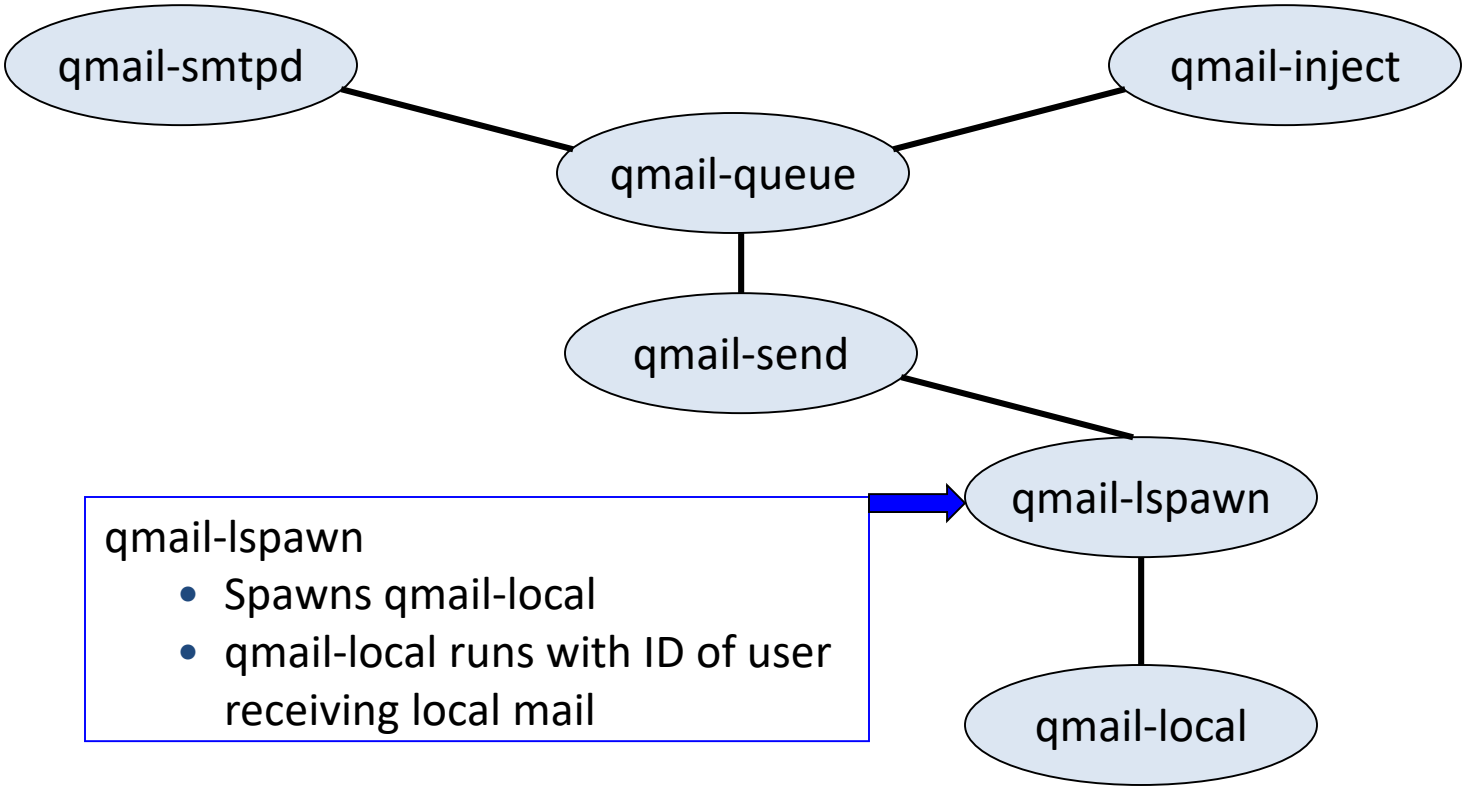
Structure of qmail



Structure of qmail



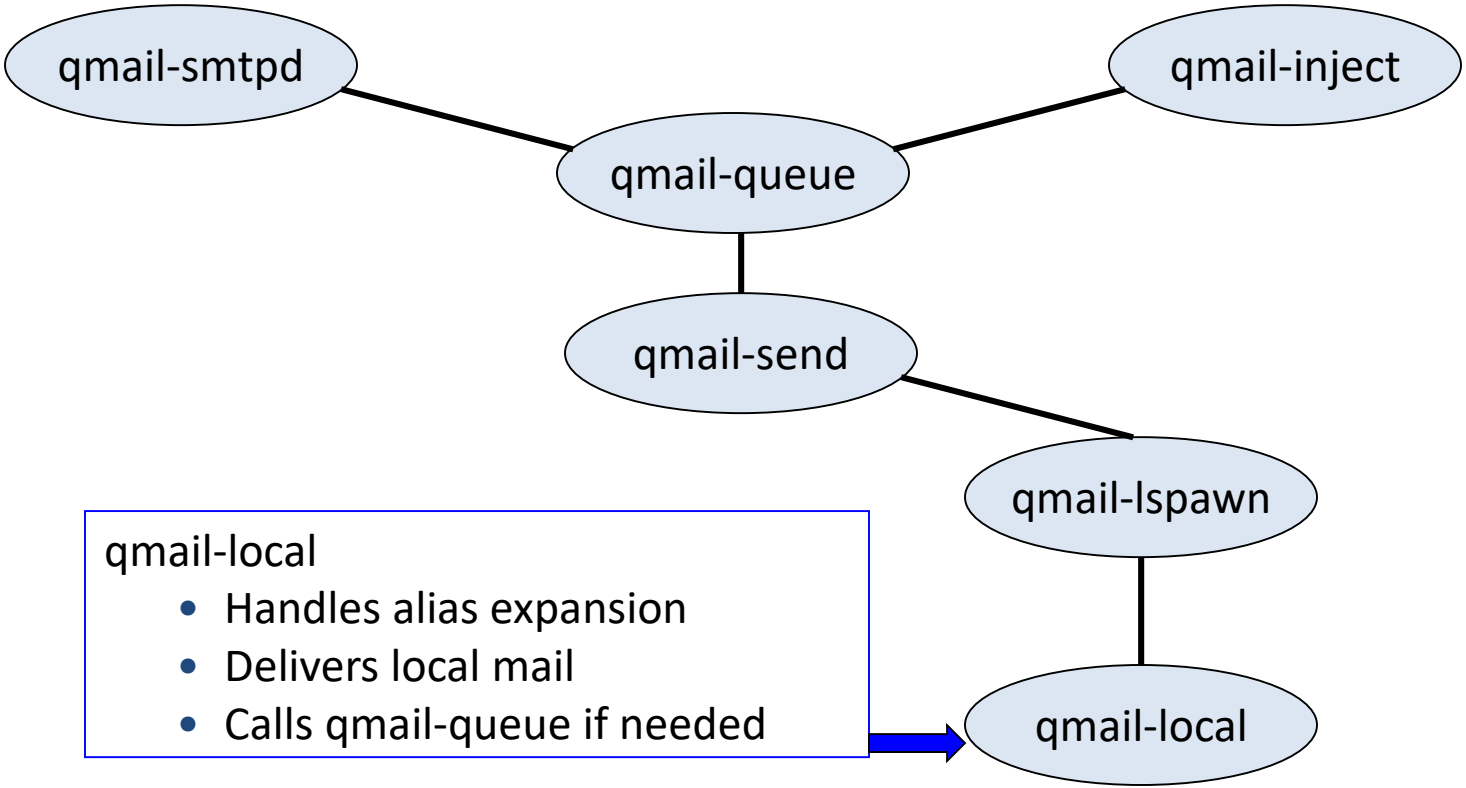
Structure of qmail



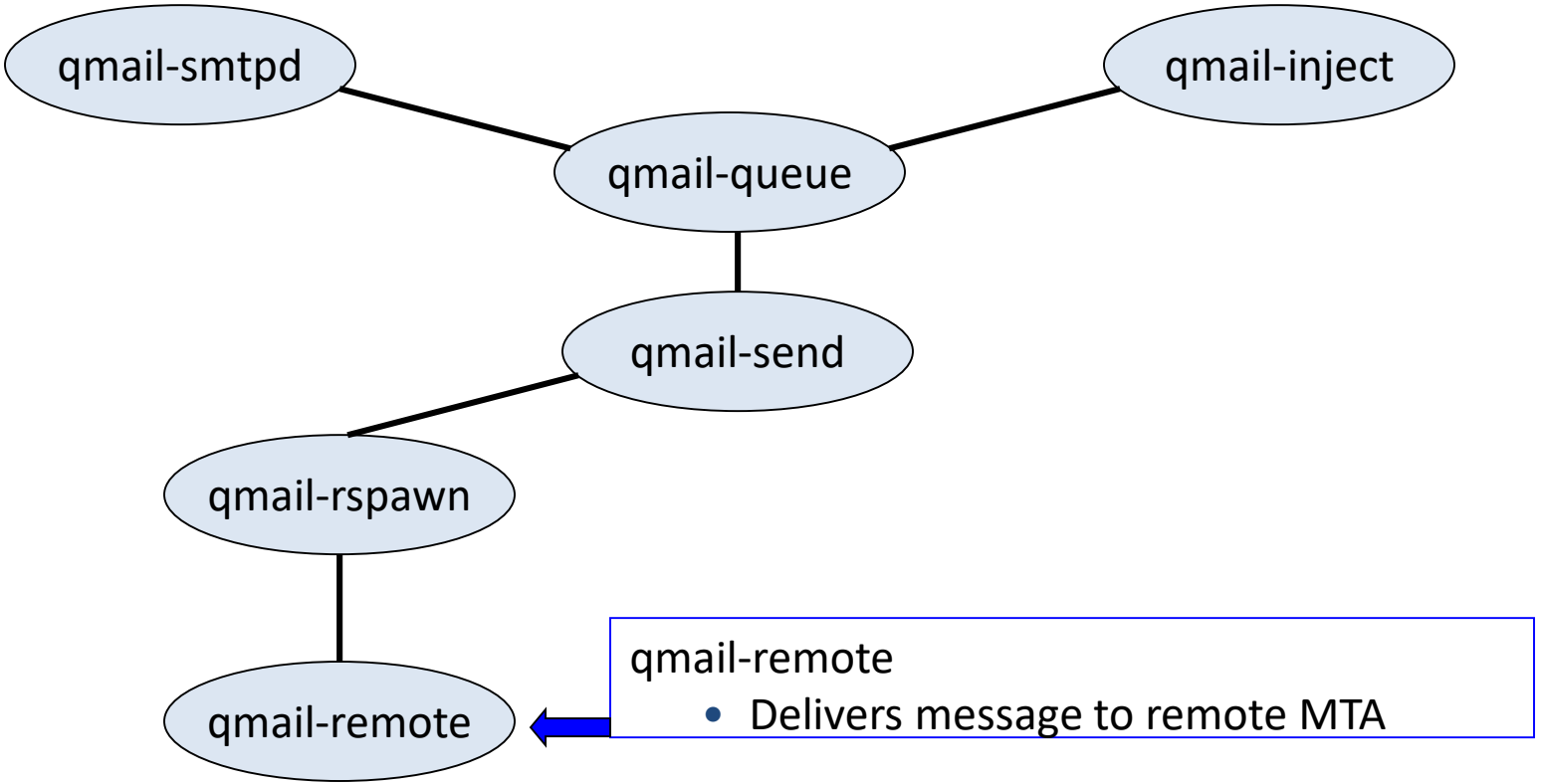
qmail-lspawn

- Spawns qmail-local
- qmail-local runs with ID of user receiving local mail

Structure of qmail

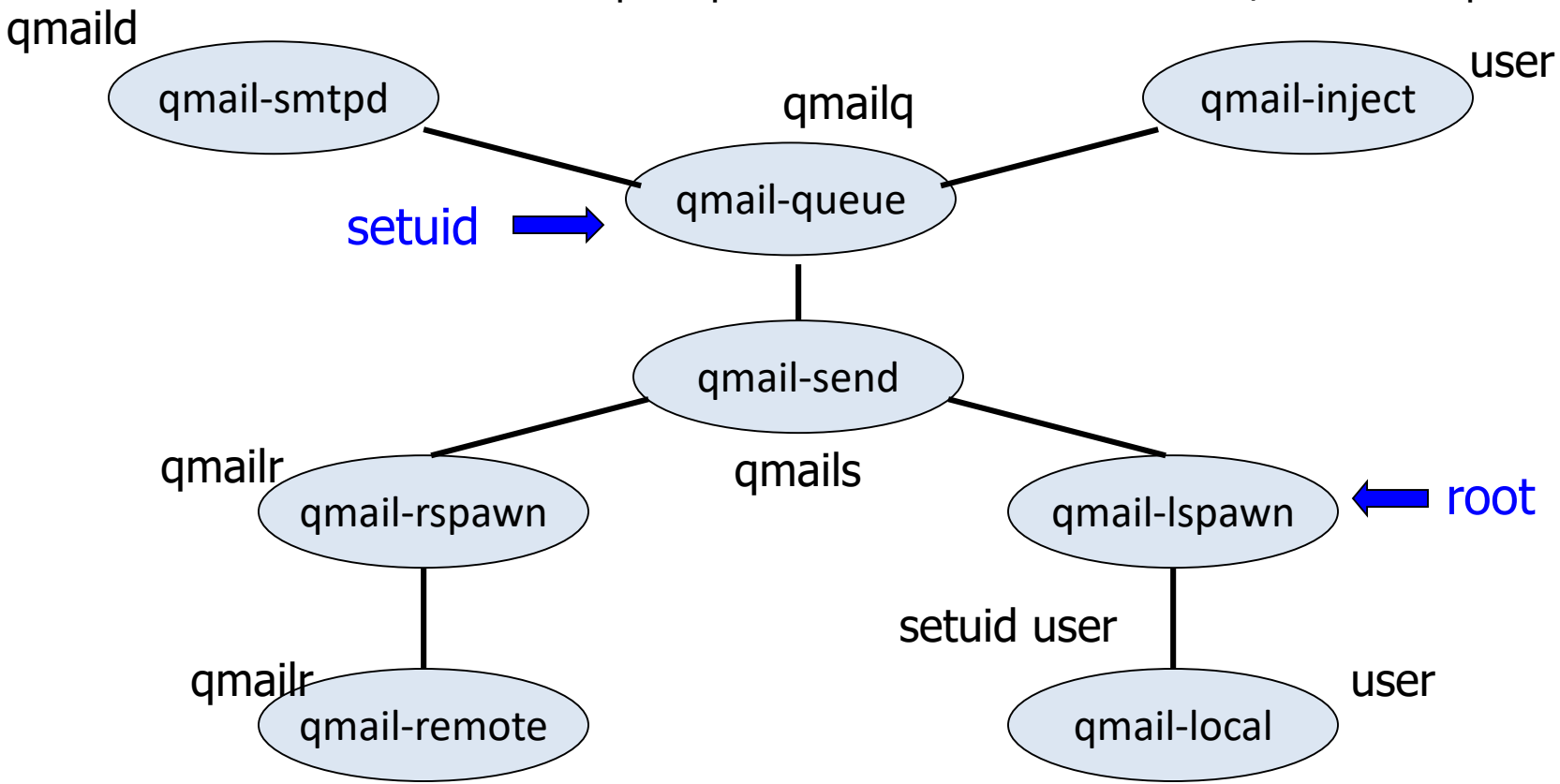


Structure of qmail

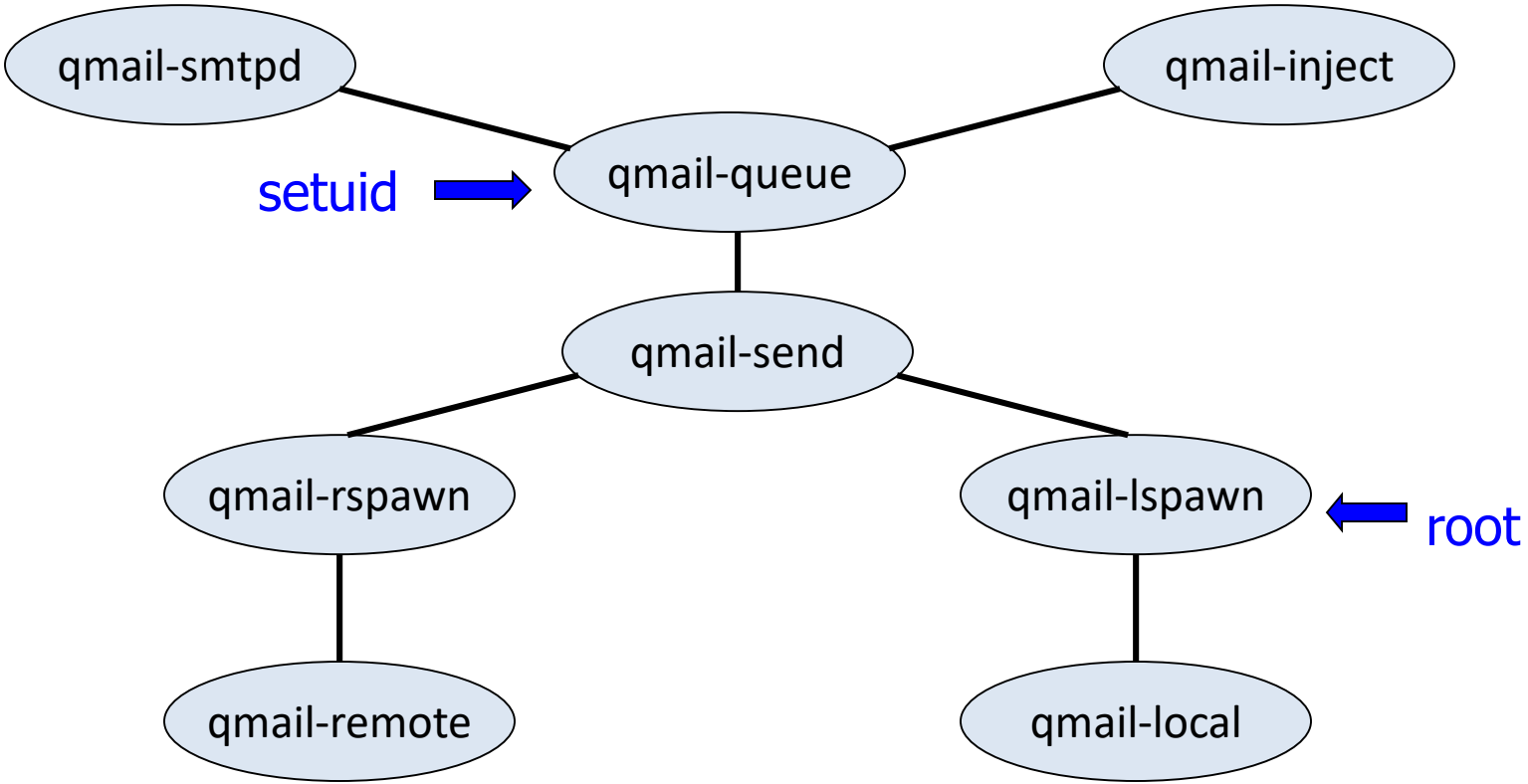


Isolation by Unix UIDs

qmailq – user who is allowed to read/write mail queue



Least privilege



Qmail summary

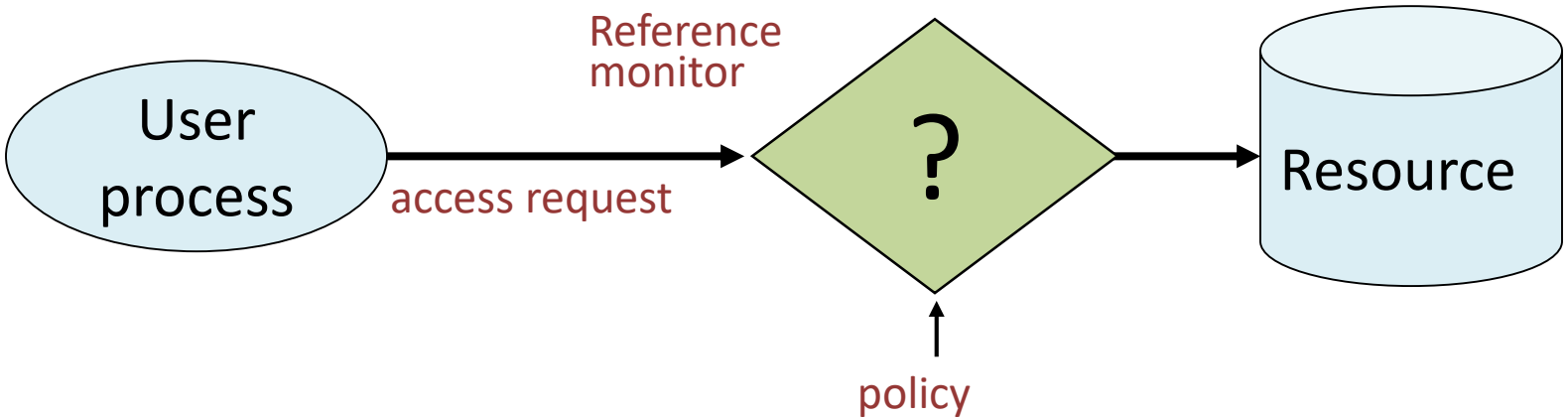
- Security goal?
- Threat model?
- Mechanisms
  - Least privilege
  - Separation

Secure Architecture Principles

Access Control Concepts

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

Subjects (Principal)

	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

Implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a “ticket” for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

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

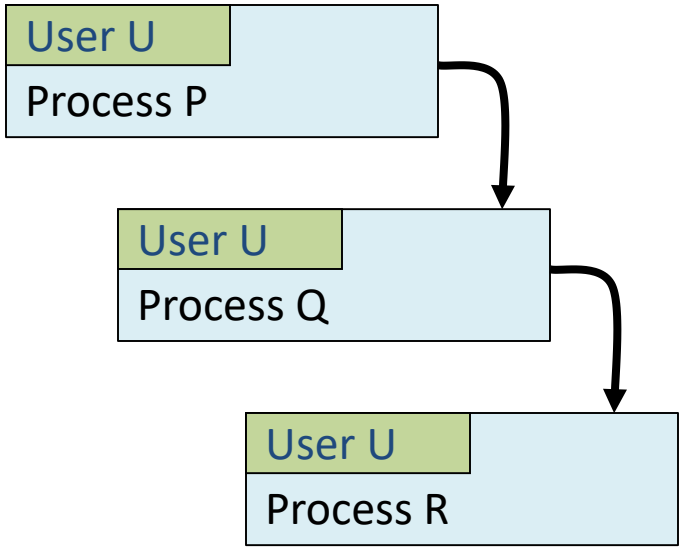
Access control lists are widely used, often with groups

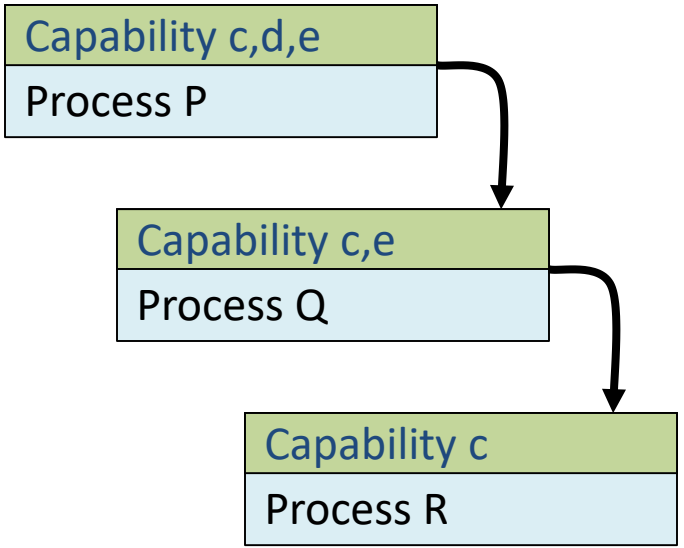
Some aspects of capability concept are used in many systems

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



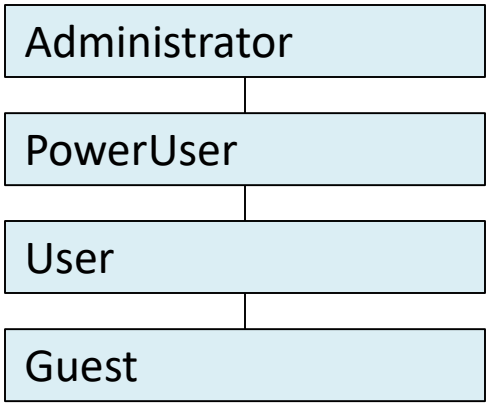


ACL vs Capabilities

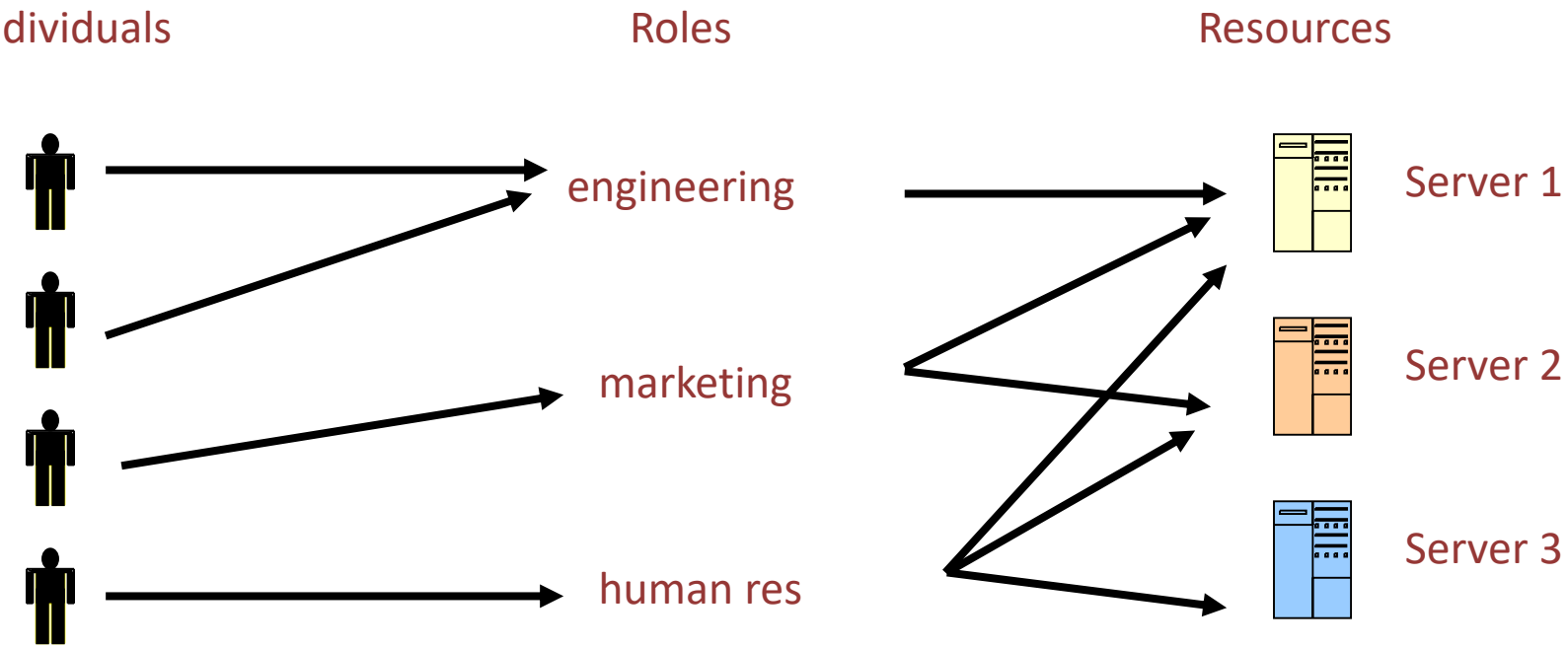
- **Delegation**
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- **Revocation**
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible 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 (aka 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: users change more frequently than roles

ACL vs Capabilities vs RBAC

- Capability? ACL? RBAC?
  - I hereby delegate to David the right to read file 4 from 9am to 1pm
  - I want to give read and write right of a file to Alice
  - I guaranteed that Charlie will have the same authority as me when accessing a file
  - A person in the financial team can perform “create a credit account transaction” in a financial application
  - a nurse shall have access to all the patients who are on her ward, or who have been there in the last 90 days

Access control summary

- Access control involves reference monitor
  - Check permissions: ⟨user info, action⟩→ yes/no
  - Important: no way around this check
- Access control matrix
  - Access control lists vs capabilities
  - Advantages and disadvantages of each
- Role-based access control
  - Use group as “user info”; use group hierarchies

Secure Architecture Principles

Access Control in UNIX

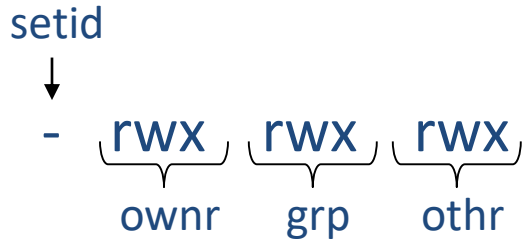
Unix access control

- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
  - Special “root” id
    - All access allowed
- File has access control list (ACL)
  - Grants permission to user ids
  - Owner, group, other

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

Unix file access control list

- 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



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)
    - 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

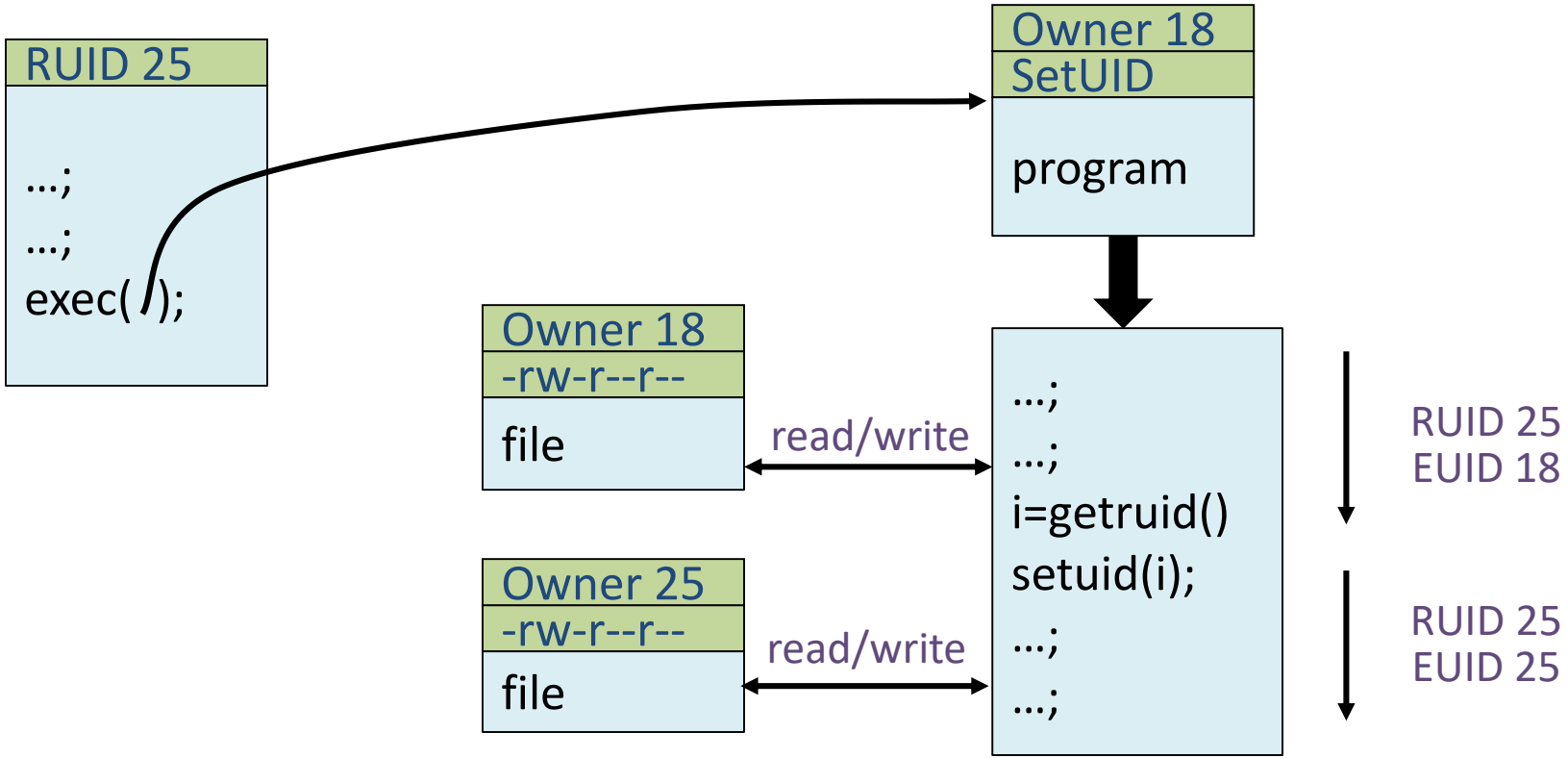
- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system call
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0

- 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


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

Runs as root

- /usr/bin/login runs as root
  - Reads username & password from terminal
  - Looks up username in /etc/passwd, etc.
  - Computes H(salt, typed password) & checks that it matches
  - If matches, sets group ID & user ID corresponding to username
  - Execute user's shell with execve system call

Another example

- Why do we need the setuid bit?
  - Some programs need to do privileged operations on behalf of unprivileged users
    - /usr/bin/ping should be able to create raw sockets (needs root)
    - An unprivileged user should be able to run ping
    - Solution: /usr/bin/ping in Linux is owned by root with setuid bit set

Unix summary

- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main limitation
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

Secure Architecture Principles

Security holes

A Security hole

- Even without root or setuid, attackers can trick root owned processes into doing things...
- Example: Want to clear unused files in /tmp
- Every night, automatically run this command as root:
  - find /tmp -atime +3 -exec rm -f -- {} \;
- find identifies files not accessed in 3 days
  - executes rm, replacing {} with file name
- rm -f -- path deletes file path
  - Note “--” prevents path from being parsed as option
- **What’s wrong here?**

An attack

find/rm	Attacker
	mkdir (“/tmp/badetc”)
	creat (“/tmp/badetc/passwd”)
readdir (“/tmp”) → “badetc”	
lstat (“/tmp/badetc”) → DIRECTORY	
readdir (“/tmp/badetc”) → “passwd”	
unlink (“/tmp/badetc/passwd”)	

An attack (cont'd)

find/rm	Attacker
	mkdir ("/tmp/badetc") creat ("/tmp/badetc/passwd")
readdir ("/tmp") → "badetc" lstat ("/tmp/badetc") → DIRECTORY readdir ("/tmp/badetc") → "passwd"	
	rename ("/tmp/badetc" → "/tmp/x") symlink ("/etc", "/tmp/badetc")
unlink ("/tmp/badetc/passwd")	

- Time-of-check-to-time-of-use [**TOCTTOU**] bug
  - find checks that /tmp/badetc is not symlink
  - But meaning of file name changes before it is used

Xterm command

- Provides a terminal window in X-windows
- Used to run with setuid root privileges
  - Requires kernel pseudo-terminal (pty) device
  - Required root privs to change ownership of pty to user
  - Also writes protected utmp/wtmp files to record users
- Had feature to log terminal session to file

```
fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666);
        /* ... */
```

- **What's wrong here?**

Xterm command

- Provides a terminal window in X-windows
- Used to run with setuid root privileges
  - Requires kernel pseudo-terminal (pty) device
  - Required root privs to change ownership of pty to user
  - Also writes protected utmp/wtmp files to record users
- Had feature to log terminal session to file

```
if (access (logfile, W_OK) < 0)
        return ERROR;
fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666);
        /* ... */
```

- xterm is root, but shouldn't log to file user can't write
- access call avoids dangerous security hole
  - Does permission check with real, not effective UID

TOCTTOU attack in xterm

xterm	Attacker
	creat (“/tmp/log”)
access (“/tmp/log”) → OK	
	unlink (“/tmp/log”)
	symlink (“/tmp/log” → “/etc/passwd”)
open (“/tmp/log”)	

- Attacker changes /tmp/log between check and use
  - xterm unwittingly overwrites /etc/passwd
  - Another TOCTTOU bug
- OpenBSD man page: “CAVEATS: access() is a potential security hole and should never be used.”

Prevent TOCTTOU

- Use new APIs that are relative to an opened directory fd
  - openat, renameat, unlinkat, symlinkat, faccessat
  - fchown, fchownat, fchmod, fchmodat, fstat, fstatat
  - O_NOFOLLOW flag to open avoids symbolic links in last component
  - But can still have TOCTTOU problems with hardlinks
- Lock resources, though most systems only lock files (and locks are typically advisory)
- Wrap groups of operations in OS transactions
  - A few research projects for POSIX [Valor] [TxOS 2009]

Secure Architecture Principles

Capability-based protection

A security problem

- Setting: A multi-user time sharing system
  - This time it's not Unix
- Wanted Fortran compiler to keep statistics
  - Modified compiler /sysx/fort to record stats in /sysx/stat
  - Gave compiler "home files license"—allows writing to anything in /sysx (kind of like Unix setuid)
- **What's wrong here?**

A confused deputy

- Attacker could overwrite any files in /sysx
  - System billing records kept in /sysx/bill got wiped
  - Probably command like fort -o /sysx/bill file.f
- Is this a bug in the compiler fort?
  - Original implementors did not anticipate extra rights
  - Can’t blame them for unchecked output file
- Compiler is a “confused deputy”
  - Inherits privileges from invoking user (e.g., read file.f)
  - Also inherits privileges from home files license
  - Which master is it serving on any given system call?
  - OS doesn’t know if it just sees open ("/sysx/bill", ...)

Recall the access control matrix

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

- Slicing matrix along rows yields capabilities
  - E.g., For each process, store a list of objects it can access
  - Process explicitly invokes particular capabilities

Capability-Based System

- Can help avoid confused deputy problem
  - E.g., Must give compiler an argument that both specifies the output file and conveys the capability to write the file (think about passing a file descriptor, not a file name)
  - So compiler uses **no ambient authority** to write file
- Three general approaches to capabilities:
  - Hardware enforced (Tagged architectures like M-machine)
  - Kernel-enforced (Hydra, KeyKOS)
  - Self-authenticating capabilities (like Amoeba)