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

Secure Architecture Principles

Isolation and Least Privilege

Principles of Secure Design

"Every battle is won BEFORE it is fought."

Sun Tzu

"Victorious warriors win first and then go to war, while defeated warriors go to war first and then seek to win."

- Sun Tzu

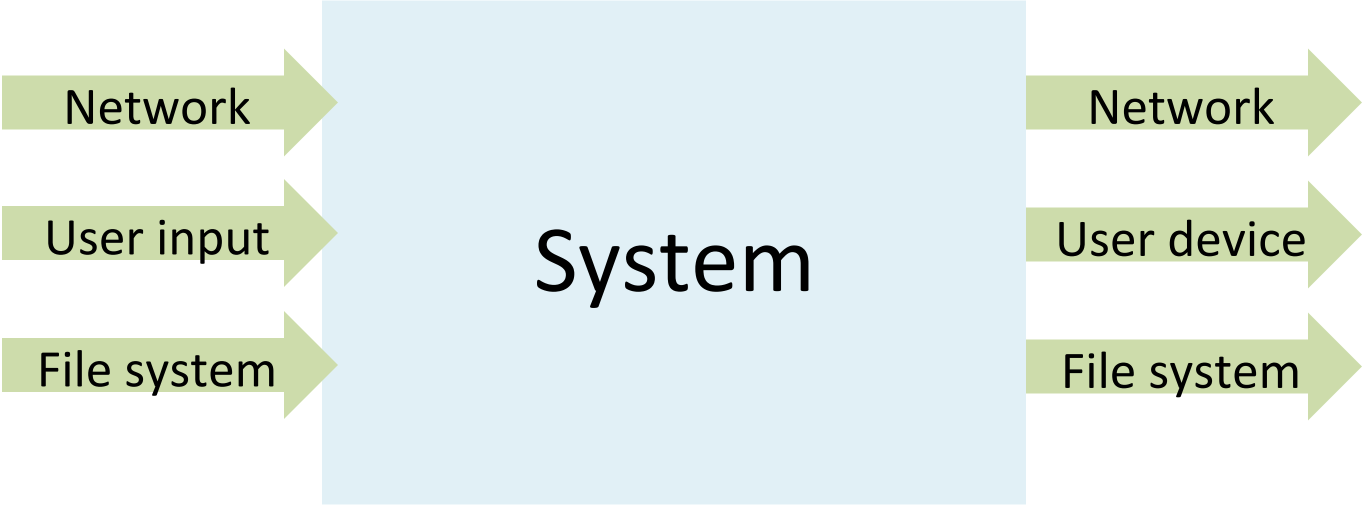
Principles of Secure Design

- 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


Network

User input

File system

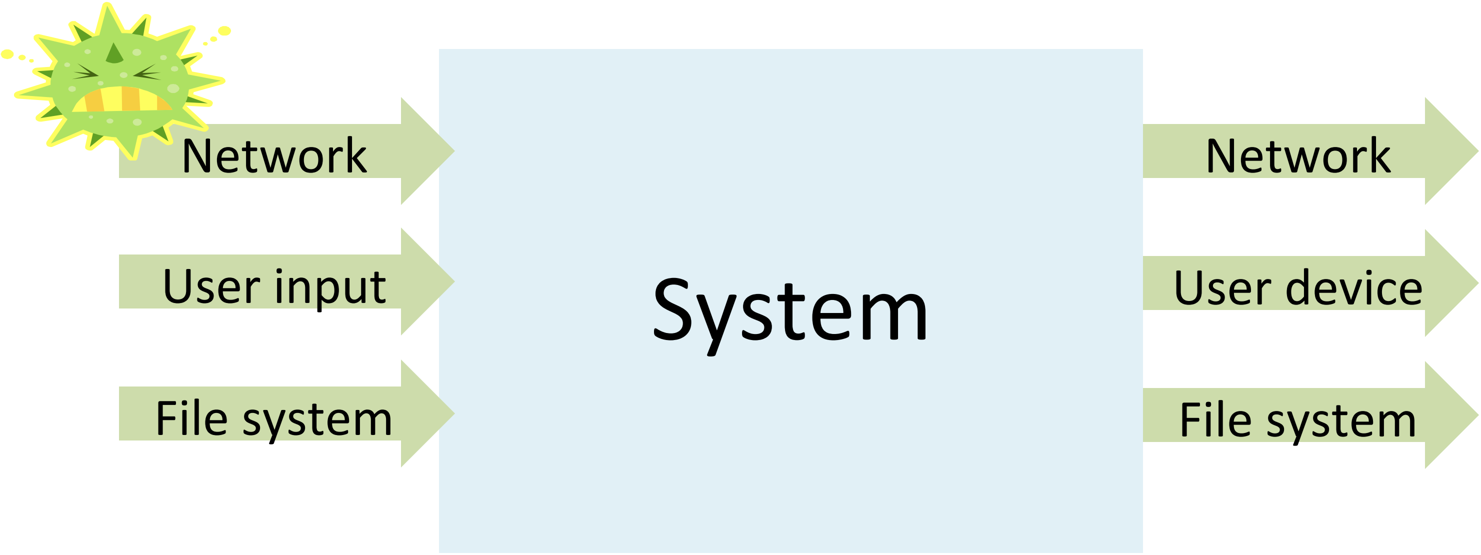
System

Network

User device

File system

Monolithic design



Network

User input

File system

System

Network

User device

File system

Monolithic design

Network

User input

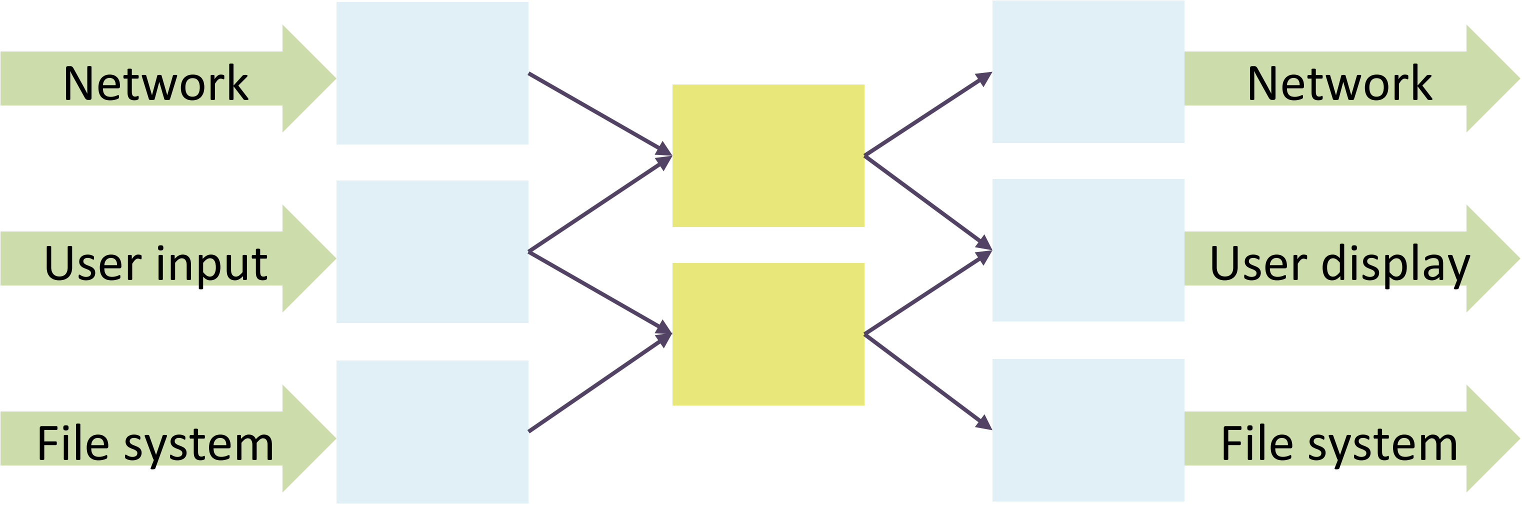
File system

Network

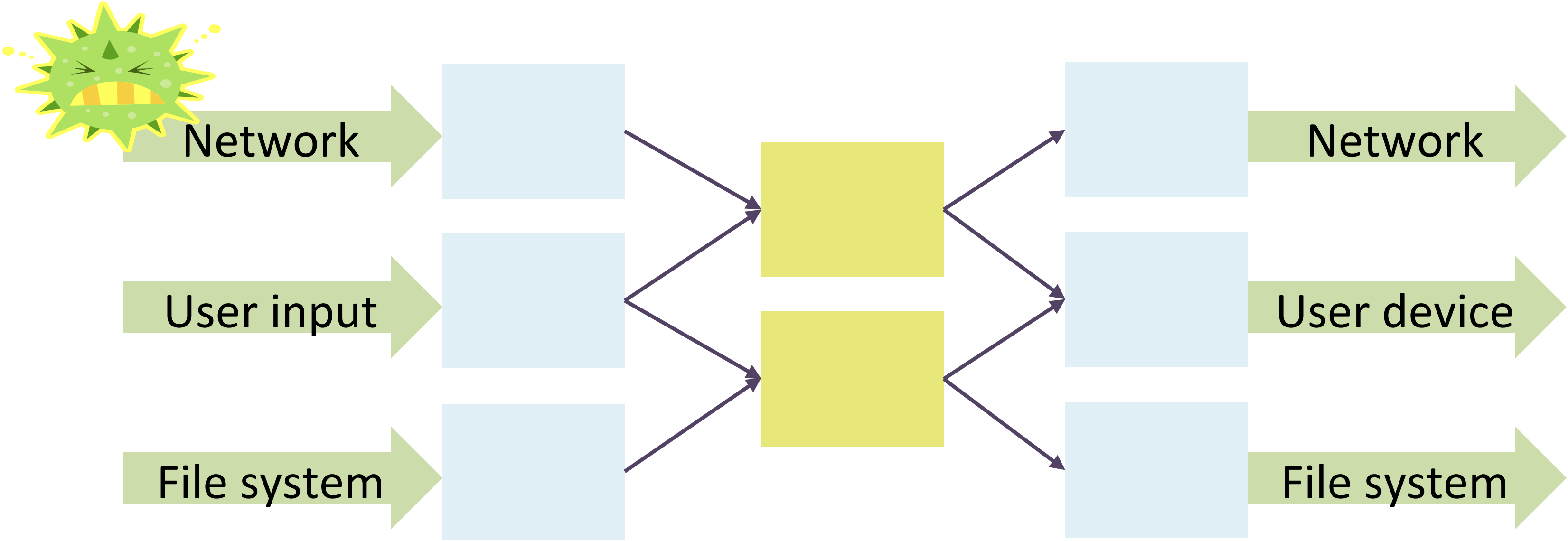
User display

File system

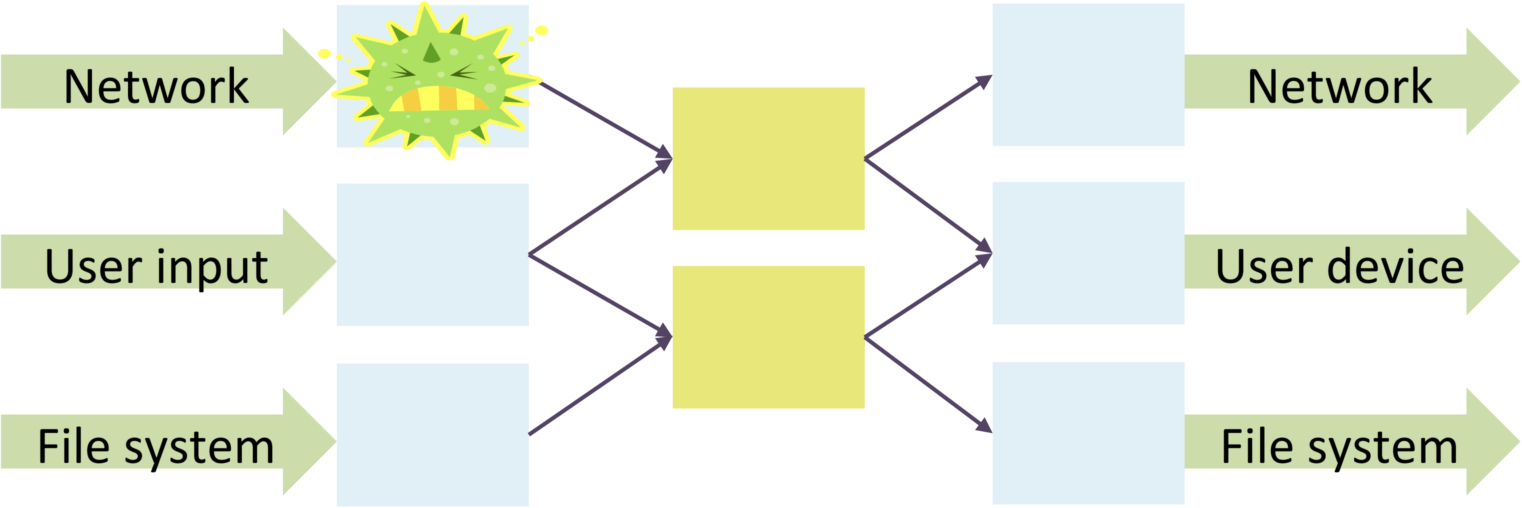
Component design



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

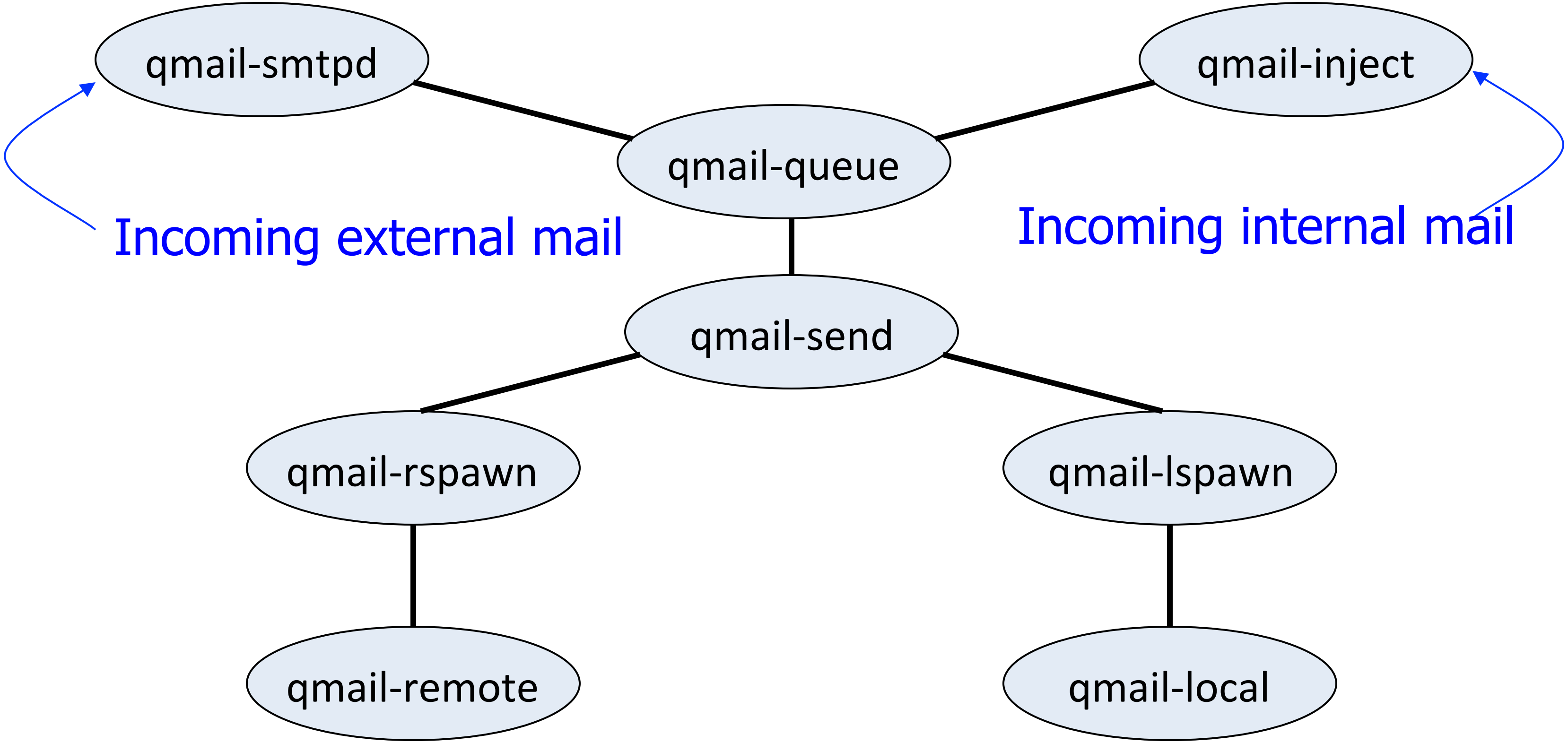
OS Basics (before examples)

- 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

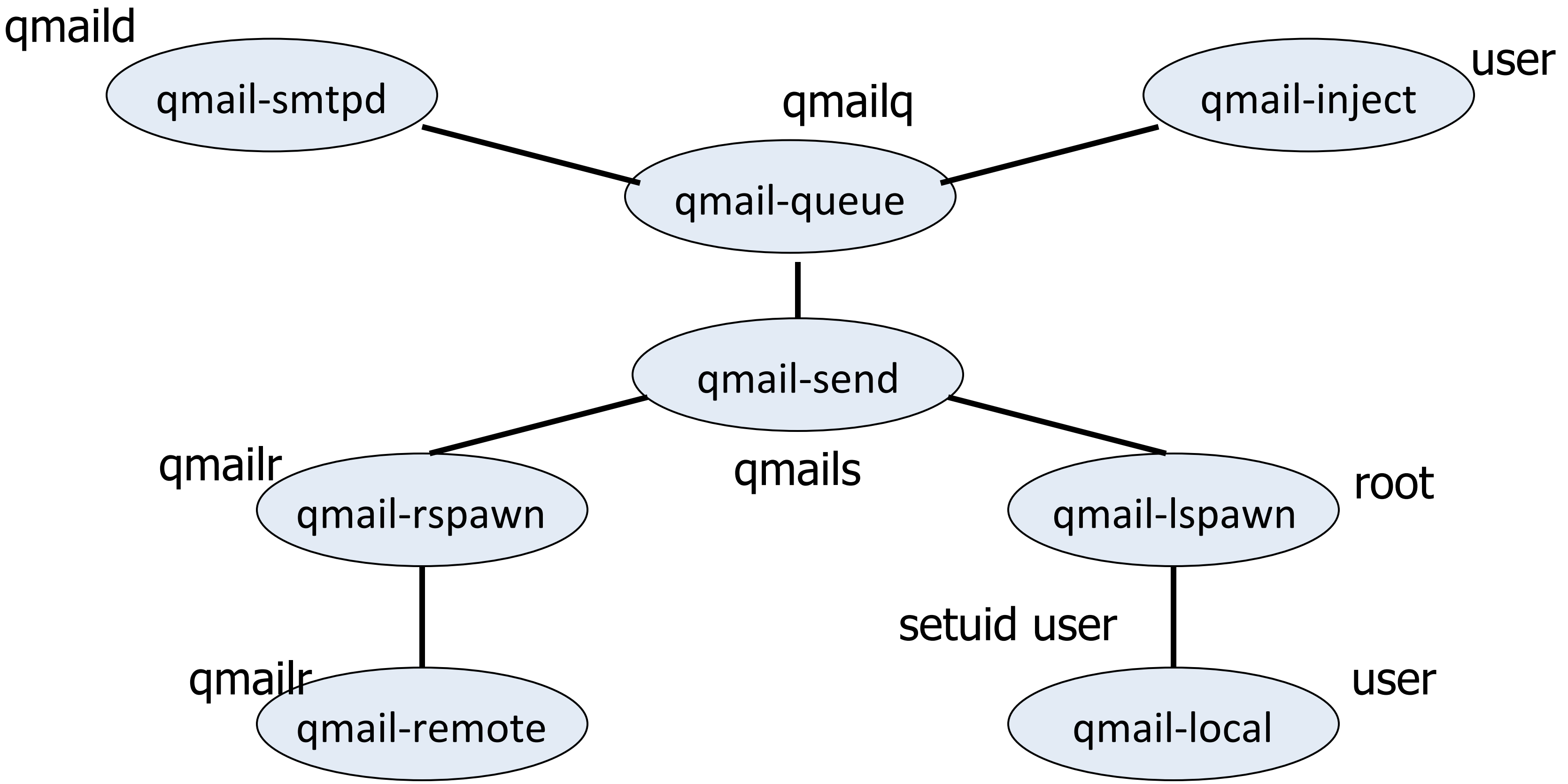
- Isolation based on OS isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Minimal privileges for each UID
  - Only one “setuid” program
    - setuid allows a program to run as different users
  - Only one “root” program
 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

Structure of qmail



Isolation by Unix UIDs

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



Android process isolation

- Android application sandbox
  - Isolation: Each application runs with its own UID in own VM
    - Provides memory protection
    - Communication limited to using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Interaction: reference monitor checks permissions on inter-component communication
  - Least Privilege: Applications announces permission
    - User grants access at install time

APPLICATIONS

Home
Contacts
Phone
Browser
...

APPLICATION FRAMEWORK

Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service

LIBRARIES

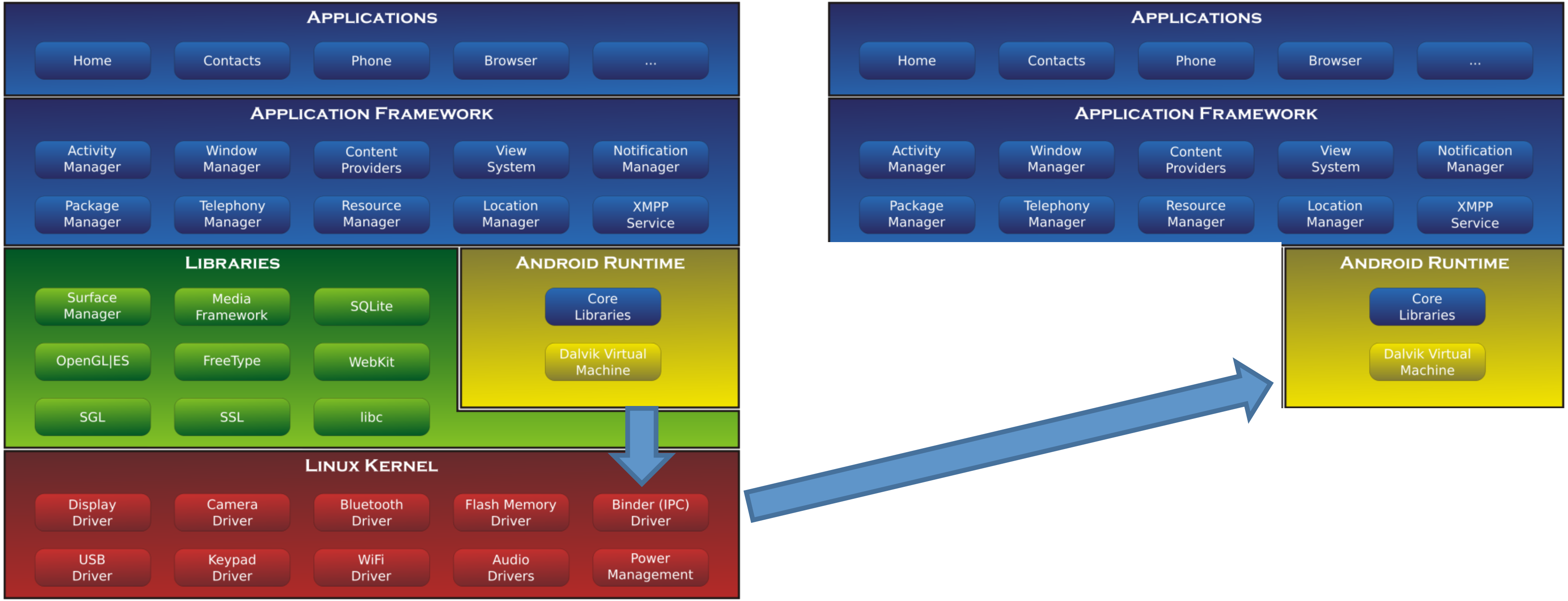
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc

ANDROID RUNTIME

Core Libraries
Dalvik Virtual Machine

LINUX KERNEL

Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management



APPLICATIONS

Home | Contacts | Phone | Browser | …

APPLICATION FRAMEWORK

Activity Manager | Window Manager | Content Providers | View System | Notification Manager

Package Manager | Telephony Manager | Resource Manager | Location Manager | XMPP Service

LIBRARIES

Surface Manager | Media Framework | SQLite

OpenGL|ES | FreeType | WebKit

SGL | SSL | libc

ANDROID RUNTIME

Core Libraries

Dalvik Virtual Machine

LINUX KERNEL

Display Driver | Camera Driver | Bluetooth Driver | Flash Memory Driver | Binder (IPC) Driver

USB Driver | Keypad Driver | WiFi Driver | Audio Drivers | Power Management

APPLICATIONS

Home | Contacts | Phone | Browser | …

APPLICATION FRAMEWORK

Activity Manager | Window Manager | Content Providers | View System | Notification Manager

Package Manager | Telephony Manager | Resource Manager | Location Manager | XMPP Service

ANDROID RUNTIME

Core Libraries

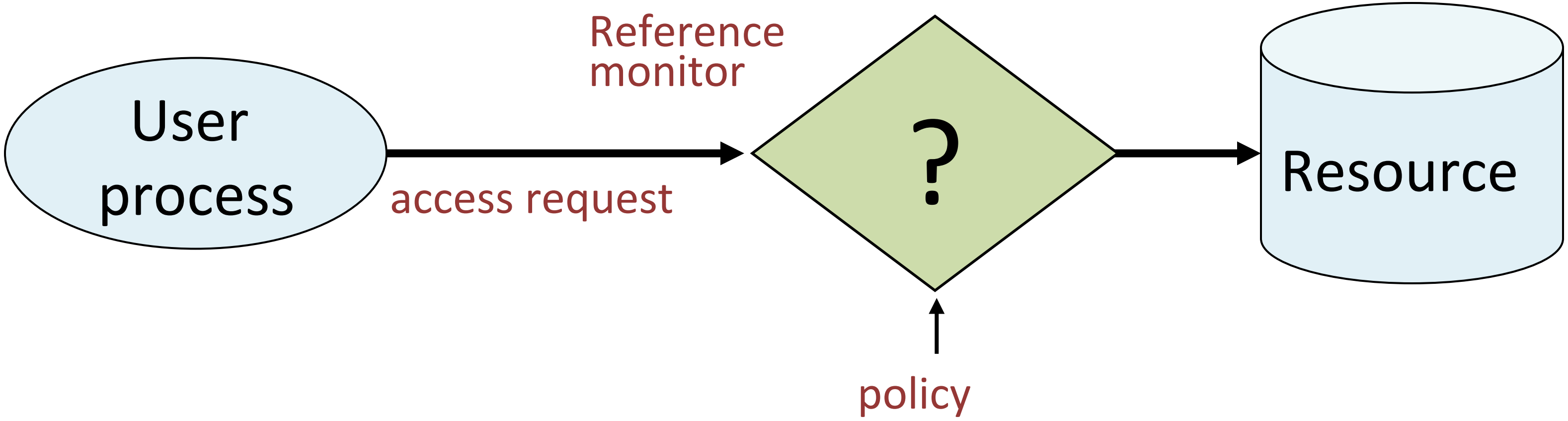
Dalvik Virtual Machine

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

	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

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

- 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

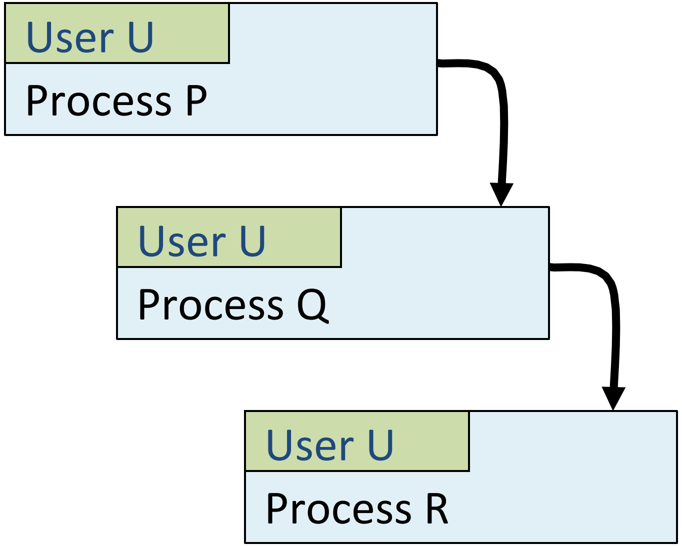
Access control lists are widely used, often with groups

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



User U

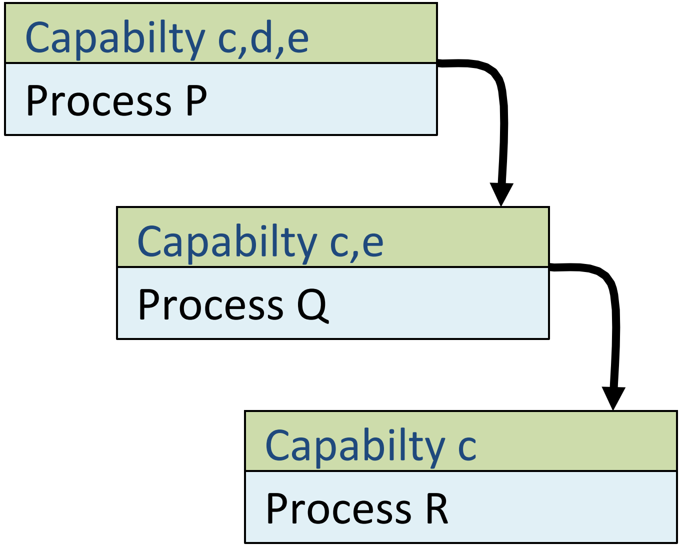
Process P

User U

Process Q

User U

Process R



Capabilty c,d,e

Process P

Capabilty c,e

Process Q

Capabilty c

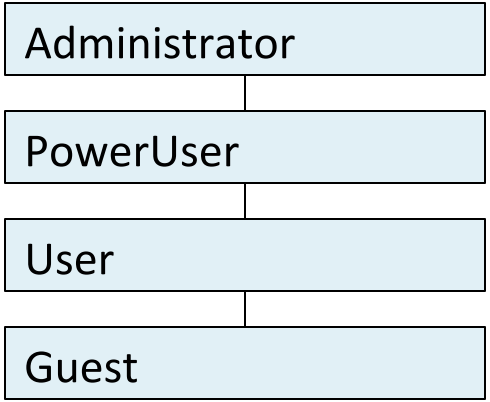
Process R

ACL vs Capabilities

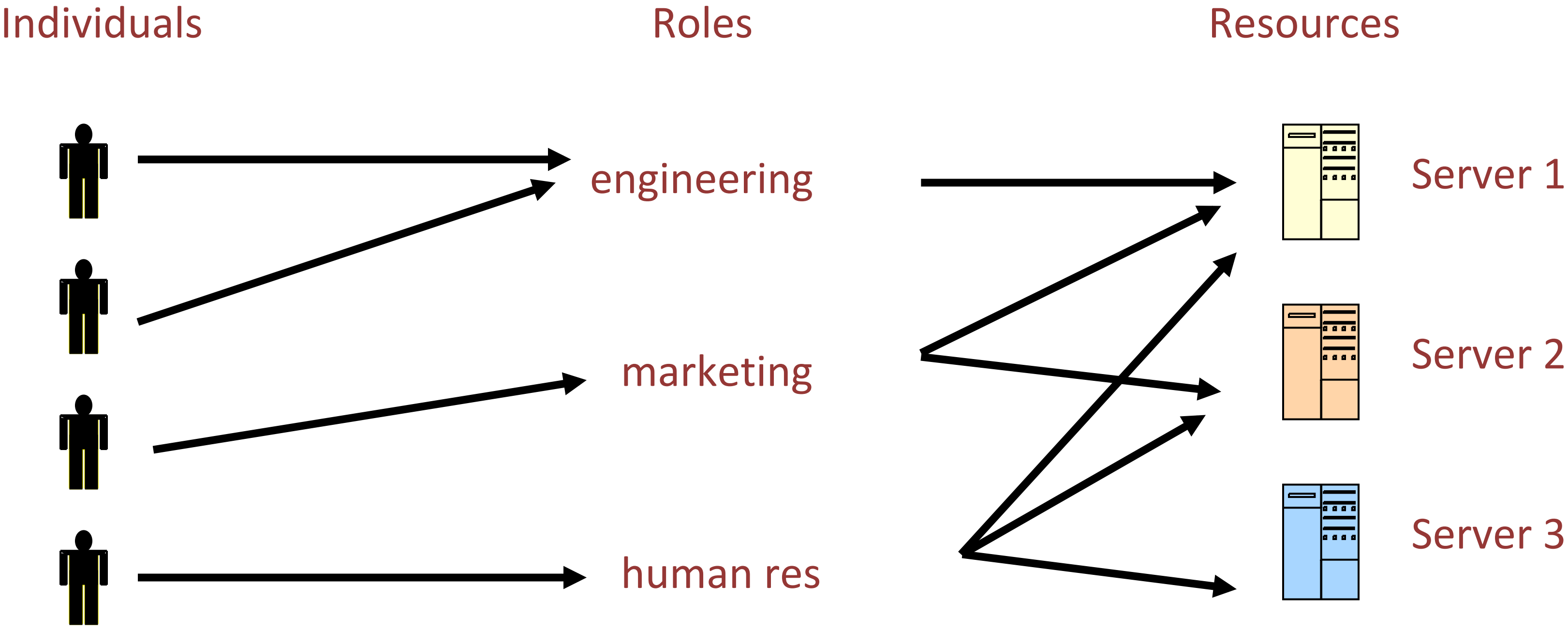
- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some 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 → P → 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

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

Operating Systems

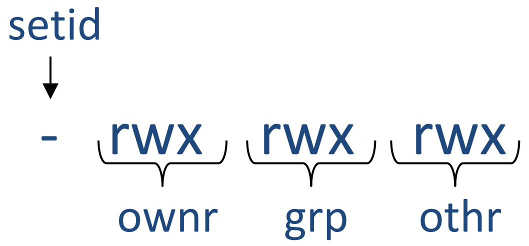
Unix access control

- 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



setid

- rwx rwx rwx

ownr grp othr

]

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

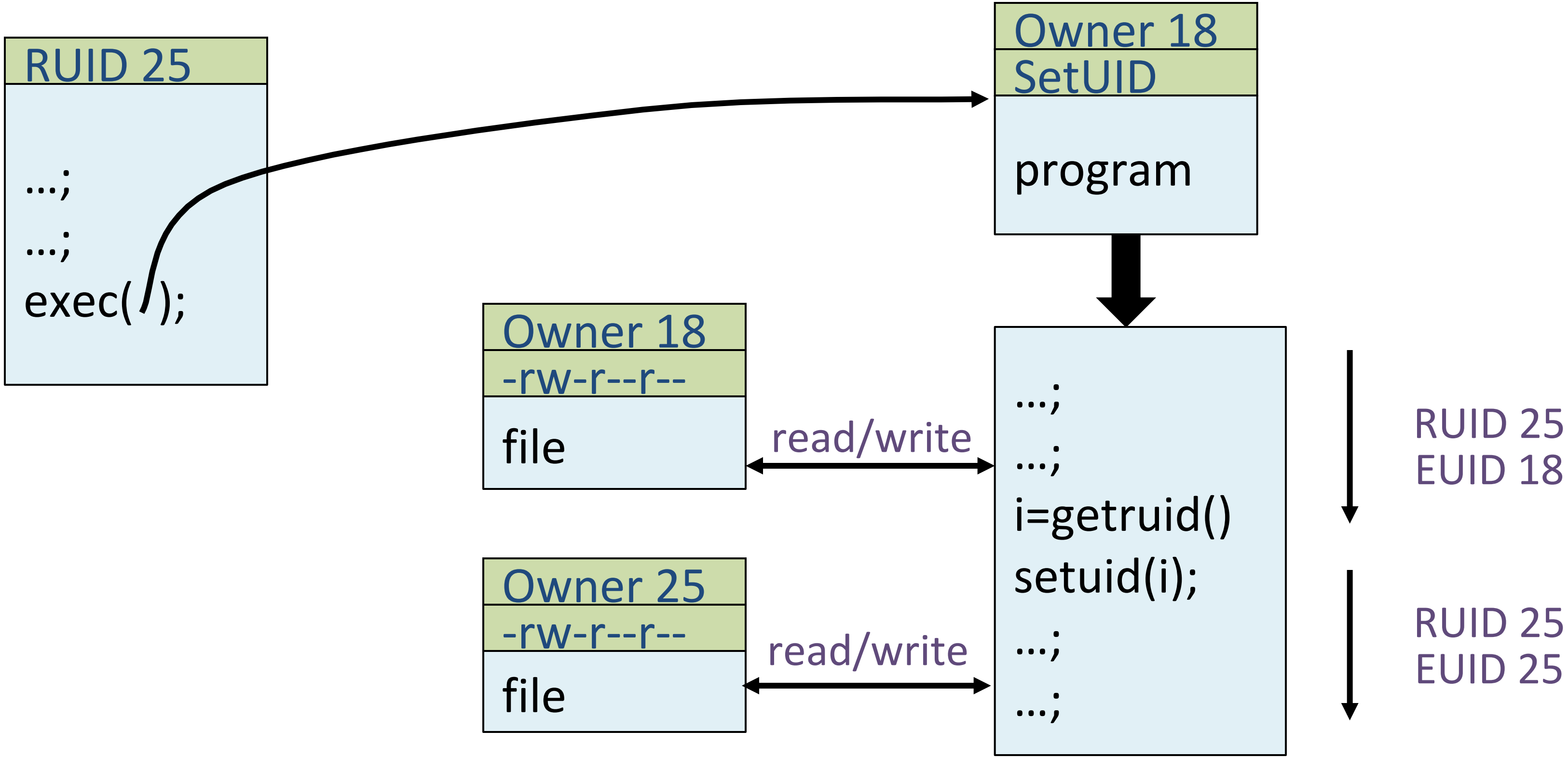
- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system call
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0

- 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);

…;

…;

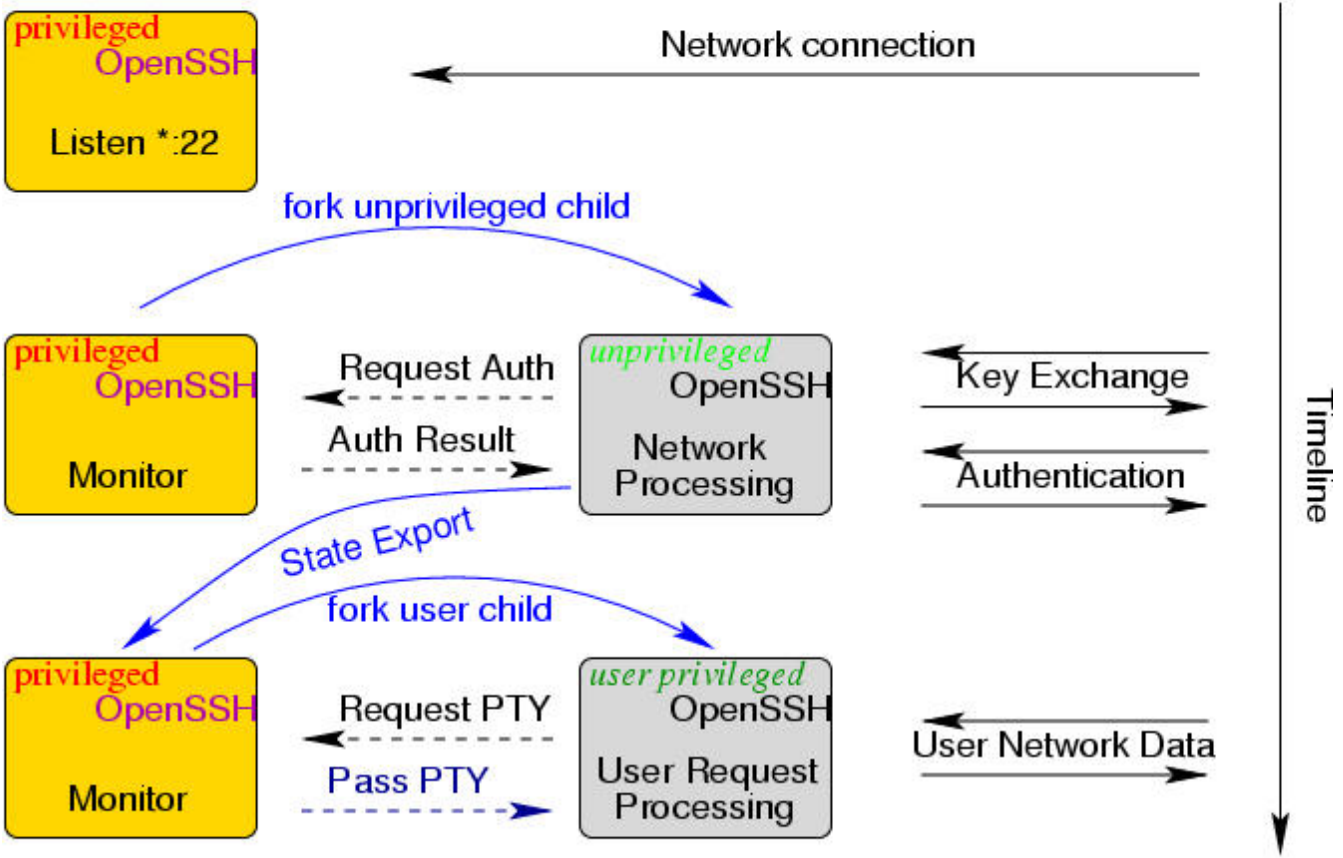
RUID 25
EUID 18

RUID 25
EUID 25

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

SetUID for least privilege: OpenSSH



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

Weakness in isolation, privileges

- Network-facing Daemons
  - Root processes with network ports open to all remote parties, e.g., sshd, ftpd, sendmail, …
    - How can you solve this?
- Rootkits
  - System extension via dynamically loaded kernel modules
- Environment Variables
  - System variables such as LD_LIBRARY_PATH that are shared state across applications. An attacker can change LD_LIBRARY_PATH to load an attacker-provided file as a dynamic library

Weakness in isolation, privileges

- Shared Resources
  - Since any process can create files in /tmp directory, an untrusted process may create files that are used by arbitrary system processes
- Time-of-Check-to-Time-of-Use (TOCTTOU)
  - Typically, a root process uses system call to determine if initiating user has permission to a particular file, e.g. /tmp/X.
  - After access is authorized and before the file open, user may change the file /tmp/X to a symbolic link to a target file /etc/shadow.

Secure Architecture Principles

Browser Isolation and Least Privilege

Web browser: an analogy

Operating system

- Subject: Processes
  - Has User ID (UID, SID)
  - Discretionary access control
- Objects
  - File
  - Network
  - …
- Vulnerabilities
  - Untrusted programs
  - Buffer overflow
  - …

Web browser

- Subject: web content (JavaScript)
  - Has "Origin"
  - Mandatory access control
- Objects
  - Document object model
  - Frames
  - Cookies / localStorage
- Vulnerabilities
  - Cross-site scripting
  - Implementation bugs
  - …

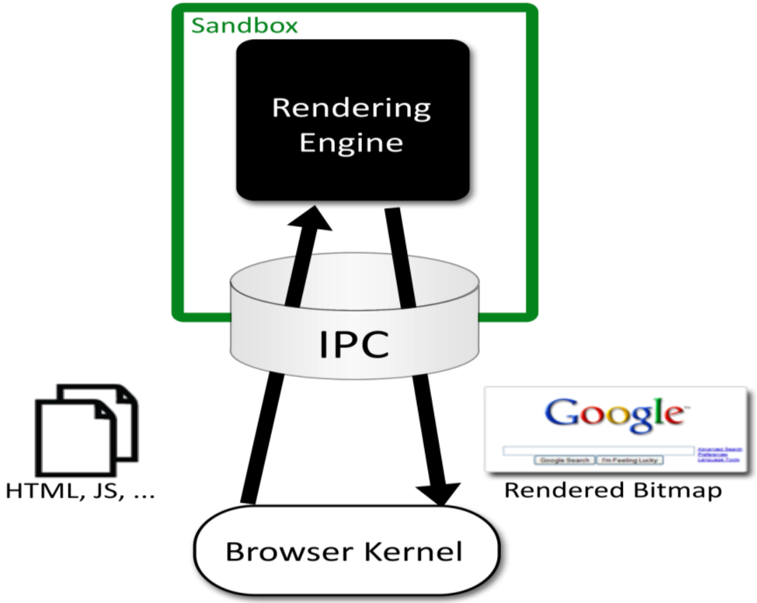
The web browser enforces its own internal policy. If the browser implementation is corrupted, this mechanism becomes unreliable.

Components of security policy

- Frame-Frame relationships
  - canScript(A,B)
    - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - canNavigate(A,B)
    - Can Frame A change the origin of content for Frame B?
- Frame-principal relationships
  - readCookie(A,S), writeCookie(A,S)
    - Can Frame A read/write cookies from site S?

Chromium Security Architecture

- Browser ("kernel")
  - Full privileges (file system, networking)
- Rendering engine
  - Up to 20 processes
  - Sandboxed
- One process per plugin
  - Full privileges of browser



Sandbox

Rendering Engine

IPC

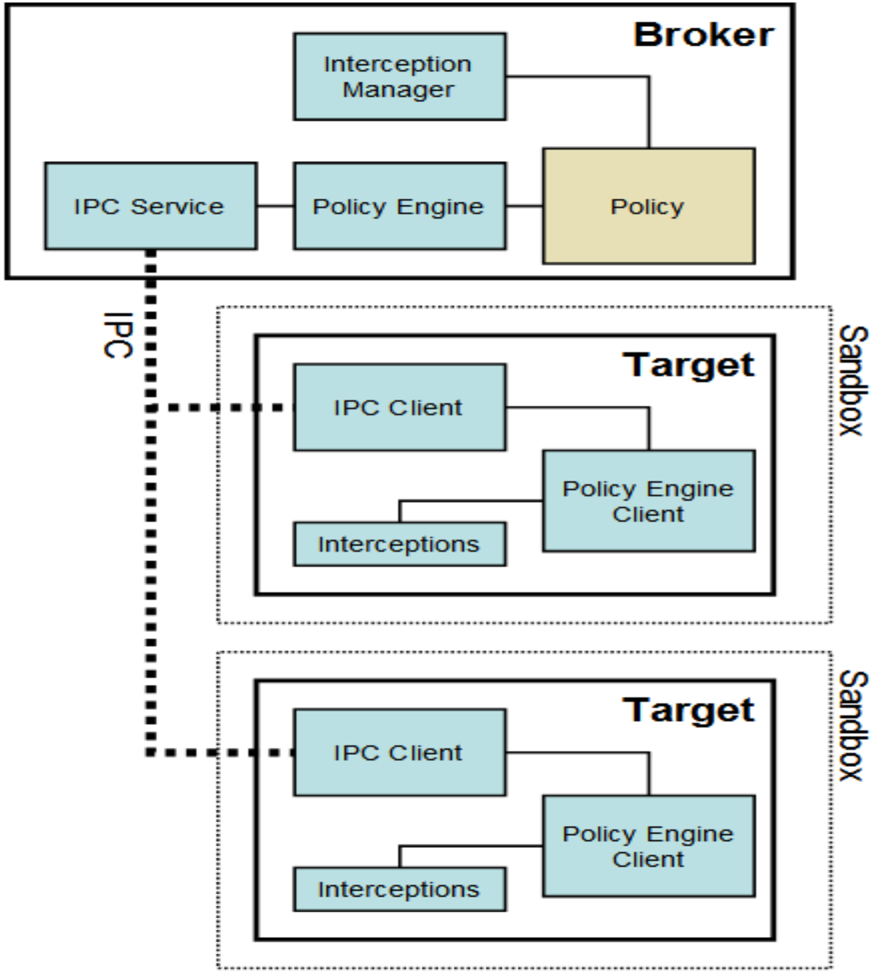
HTML, JS, ...

Rendered Bitmap

Browser Kernel

Chromium

Communicating sandbox components

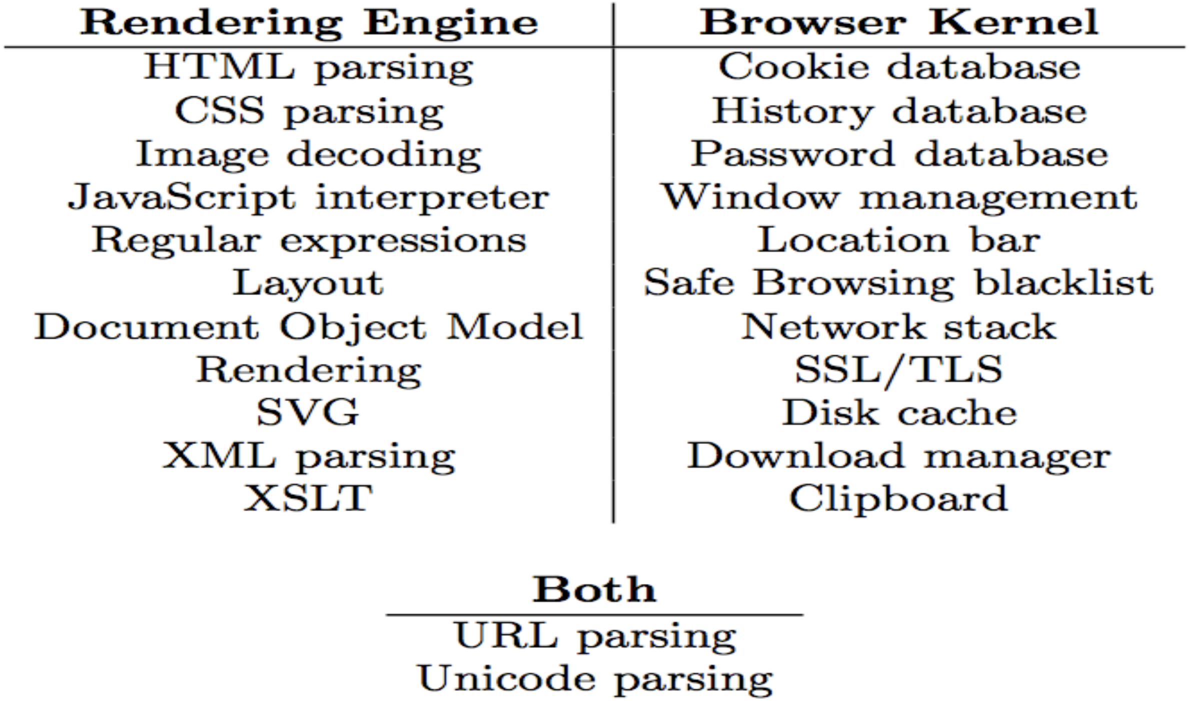


See: http://dev.chromium.org/developers/design-documents/sandbox/

Design Decisions

- Compatibility
  - Sites rely on the existing browser security policy
  - Browser is only as useful as the sites it can render
  - Rules out more “clean slate” approaches
- Black Box
  - Only renderer may parse HTML, JavaScript, etc.
  - Kernel enforces coarse-grained security policy
  - Renderer to enforces finer-grained policy decisions
- Minimize User Decisions

Task Allocation



Rendering Engine	**Browser Kernel**
HTML parsing	Cookie database
CSS parsing	History database
Image decoding	Password database
JavaScript interpreter	Window management
Regular expressions	Location bar
Layout	Safe Browsing blacklist
Document Object Model	Network stack
Rendering	SSL/TLS
SVG	Disk cache
XML parsing	Download manager
XSLT	Clipboard

Both
URL parsing
Unicode parsing

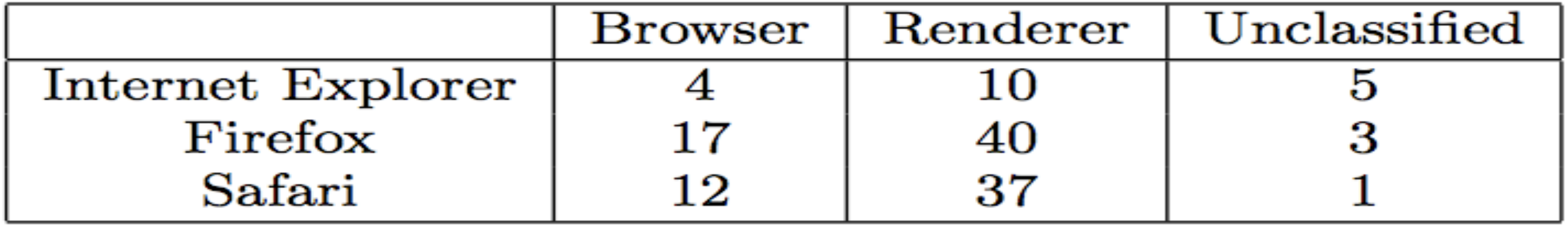
Leverage OS Isolation

- Sandbox based on four OS mechanisms
  - A restricted token
  - The Windows *job* object
  - The Windows *desktop* object
  - Windows Vista only: integrity levels
- Specifically, the rendering engine
  - adjusts security token by converting SIDS to DENY_ONLY, adding restricted SID, and calling AdjustTokenPrivileges
  - runs in a Windows Job Object, restricting ability to create new processes, read or write clipboard, ..
  - runs on a separate desktop, mitigating lax security checking of some Windows APIs

See: http://dev.chromium.org/developers/design-documents/sandbox/

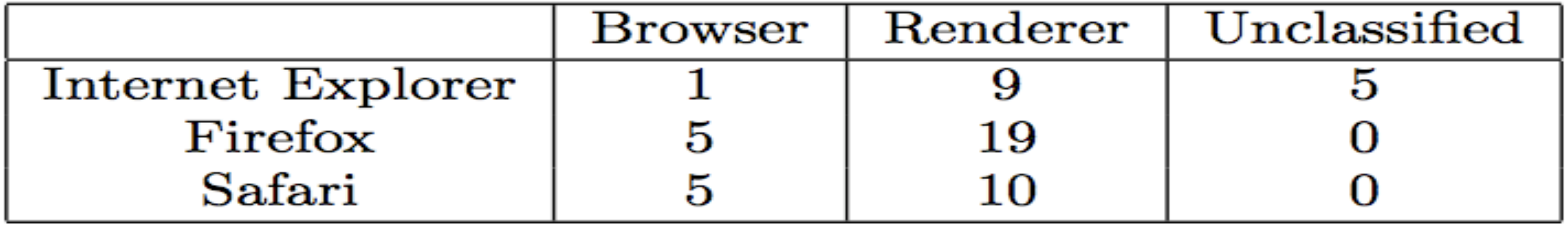
Evaluation: CVE count

- Total CVEs:



	Browser	Renderer	Unclassified
Internet Explorer	4	10	5
Firefox	17	40	3
Safari	12	37	1

- Arbitrary code execution vulnerabilities:



	Browser	Renderer	Unclassified
Internet Explorer	1	9	5
Firefox	5	19	0
Safari	5	10	0

Summary

- Security principles
  - Isolation
  - Principle of Least Privilege
  - Qmail example
- Access Control Concepts
  - Matrix, ACL, Capabilities
- OS Mechanisms
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS
- Browser security architecture
  - Isolation and least privilege example