Module 3: Secure System Architecture

Acknowledgements

- Dan Boneh (Stanford University)
- John C. Mitchell (Stanford University)
- Nicolai Zeldovich (MIT)
- Jungmin Park (Virginia Tech)
- Patrick Schaumont (Virginia Tech)
- C. Edward Chow
- Arun Hodigere
- Sandeep Shukla
- Web Resources

Secure Architecture Principles

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

Isolation and Least Privilege

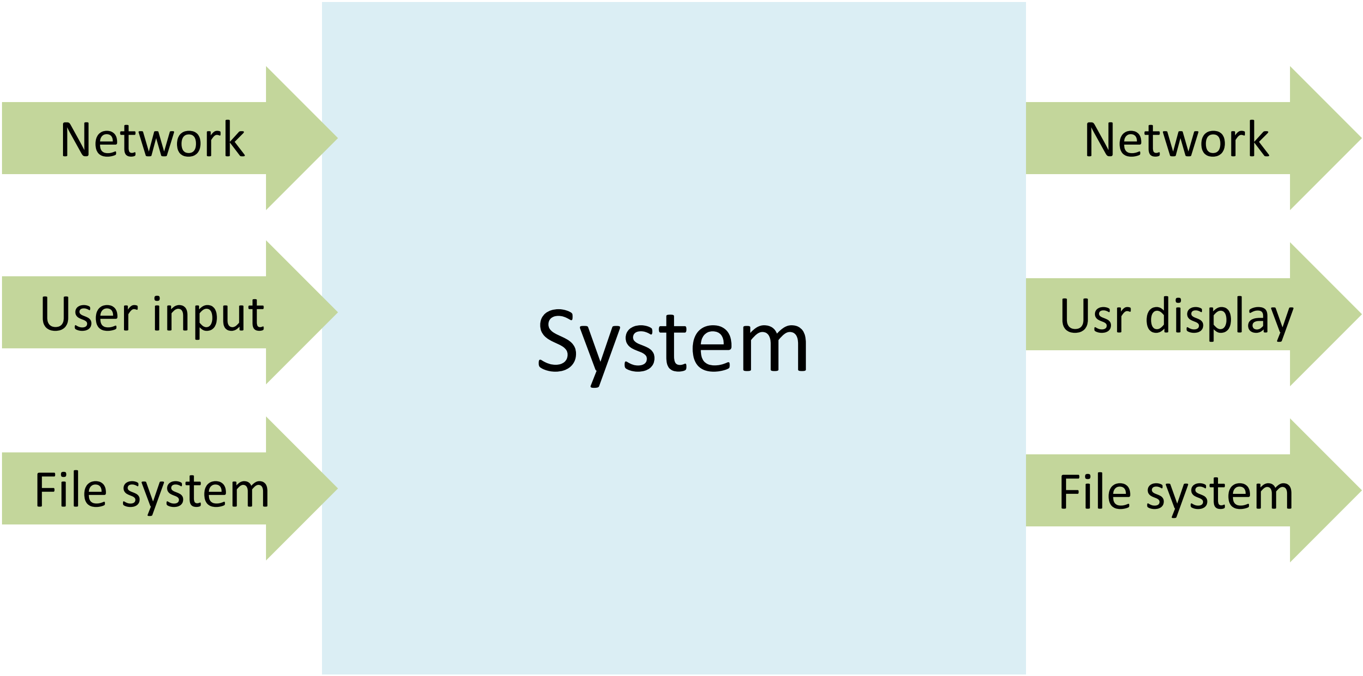
Principles of Secure Design

- 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



Network

User input

File system

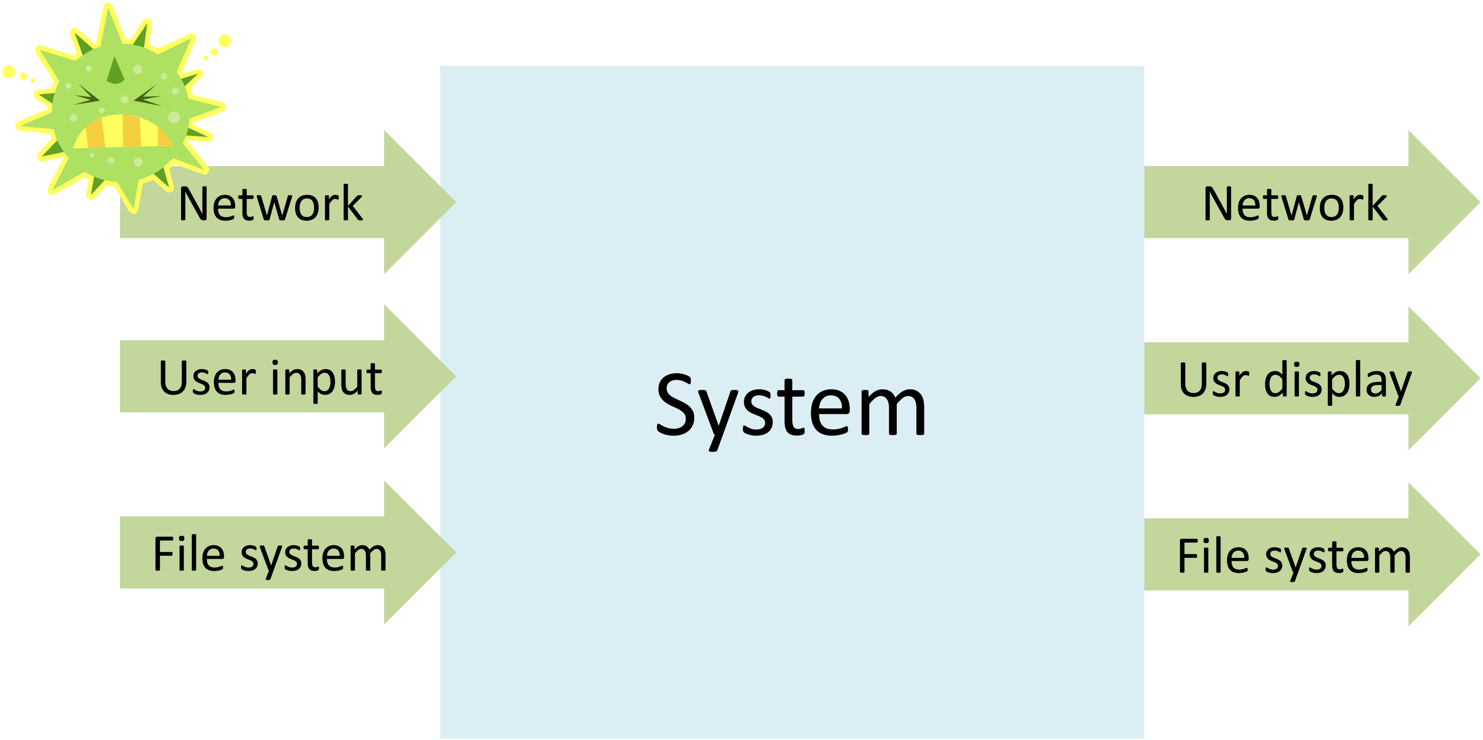
System

Network

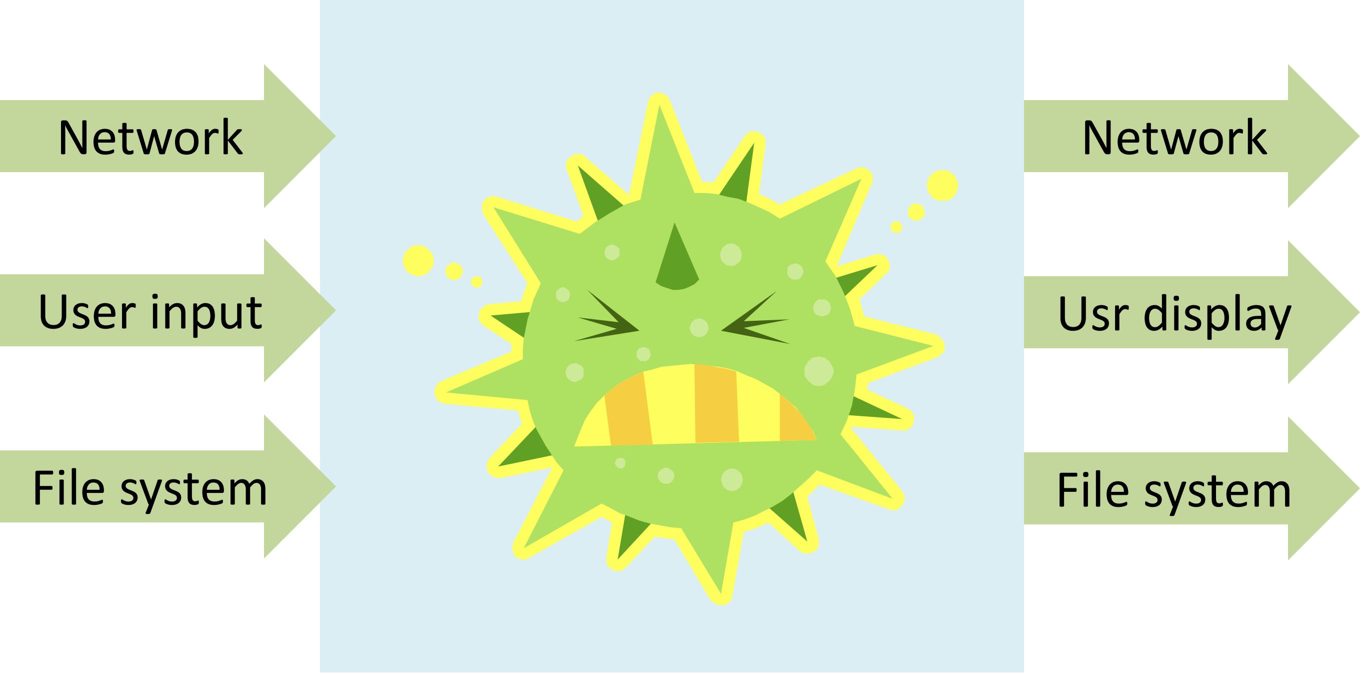
Usr display

File system

Monolithic design



Monolithic design



Network

User input

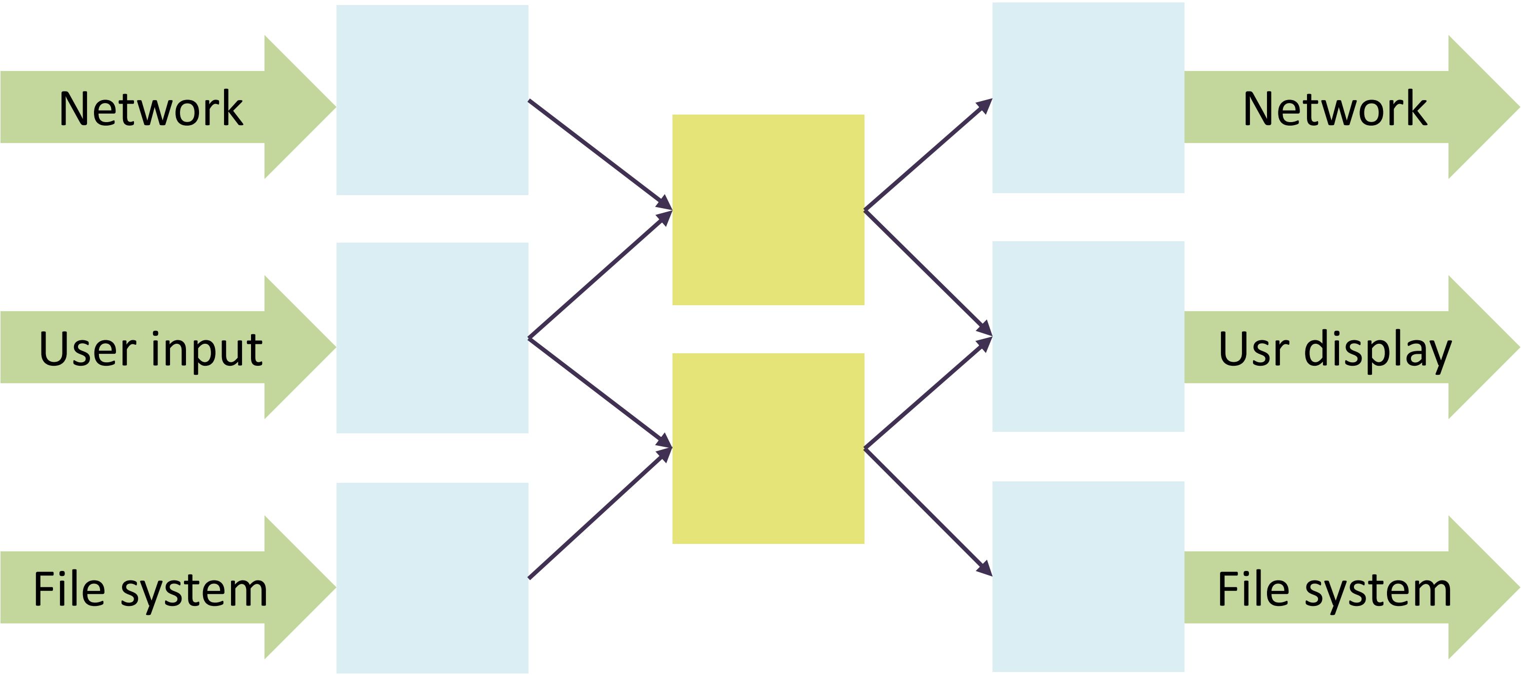
File system

Network

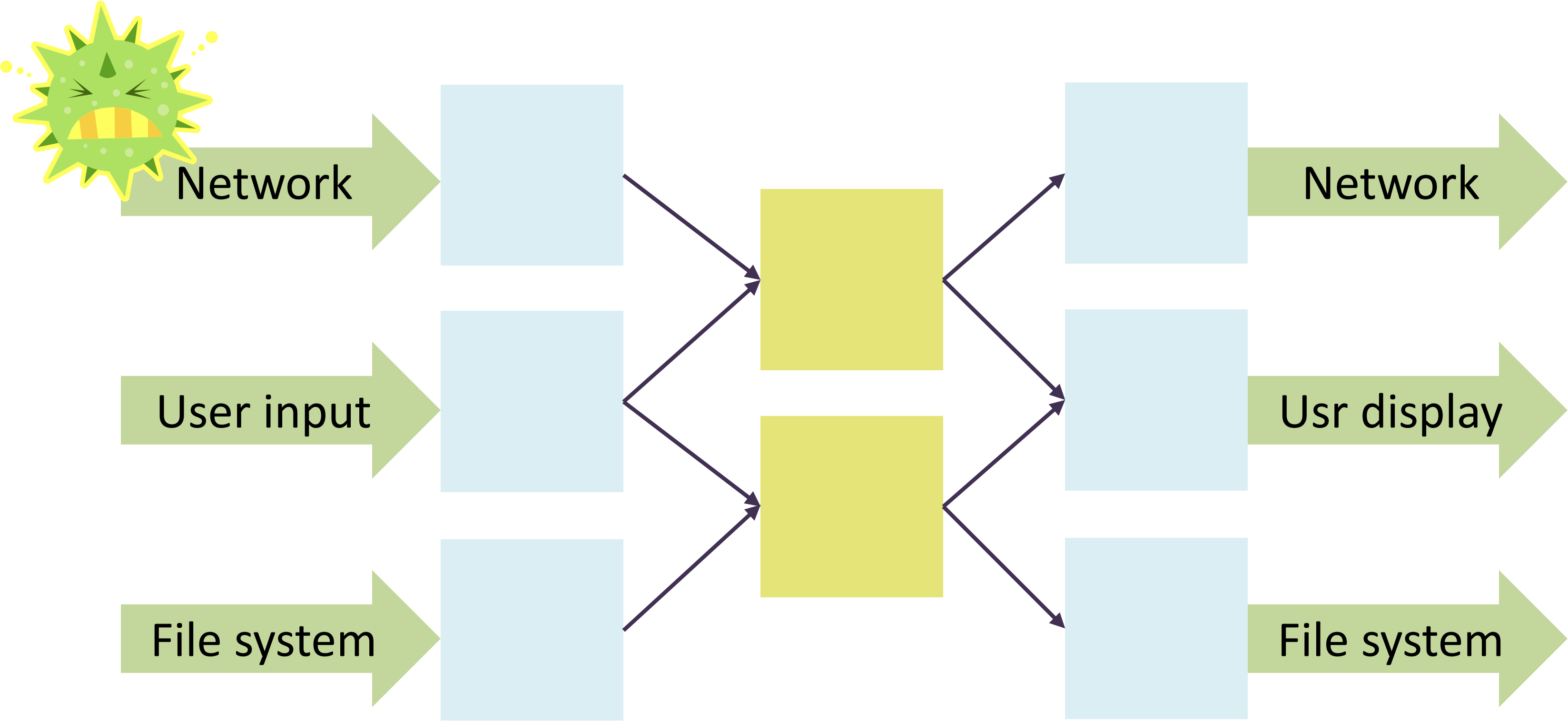
Usr display

File system

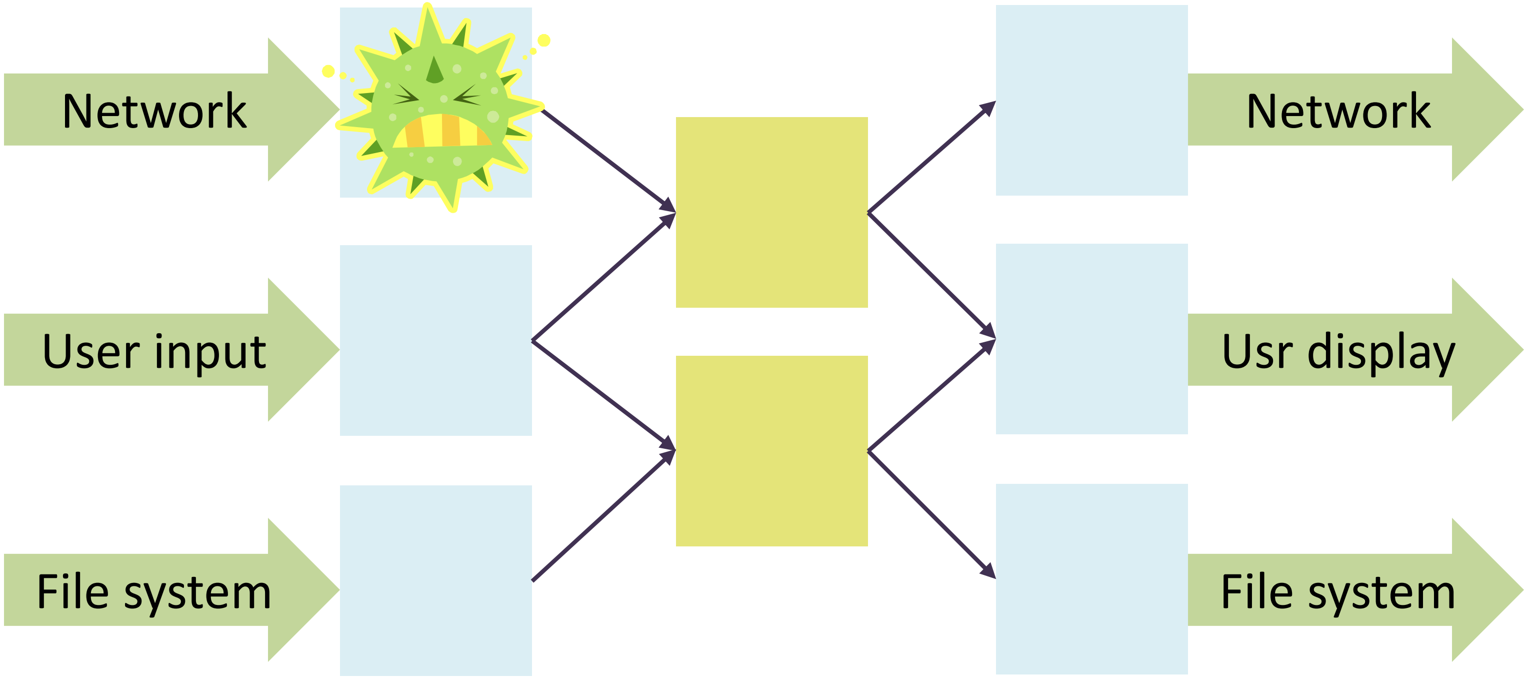
Component design



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

Qmail Security Guarantee

- Qmail written by D. J. Bernstein
  - Prof at UIC, TU Eindhoven
  - Famous for a lot of things [e.g., suing US Govt]
- Written when sendmail was full of bugs
  - 14 vulnerabilities in two years (’96, ’97)
- Wrote qmail and offered a $500 prize [‘97]
  - To find any exploitable vulnerability
  - Prize never given out

https://cr.yp.to/qmail/guarantee.html

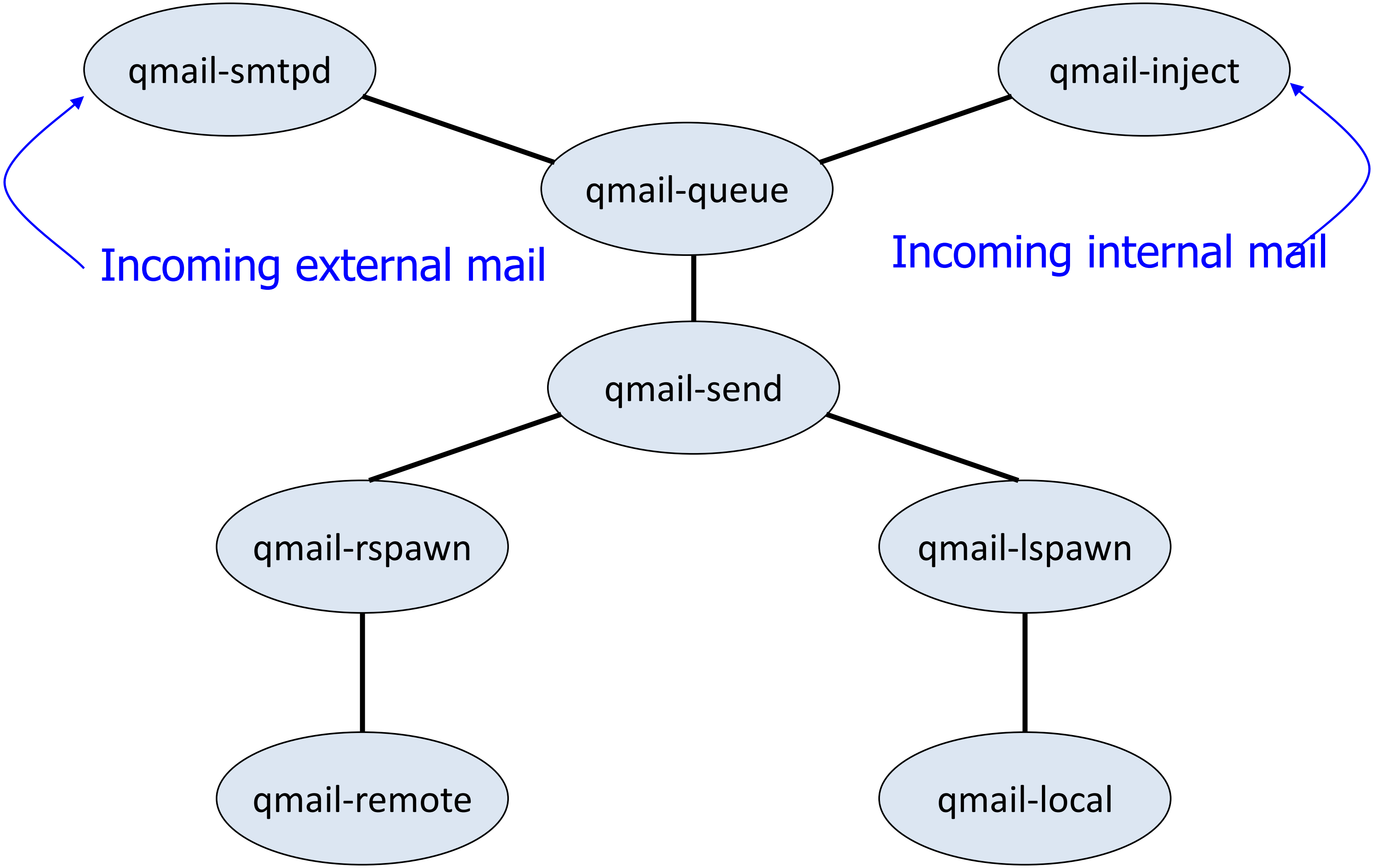
OS Basics (before examples)

- 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

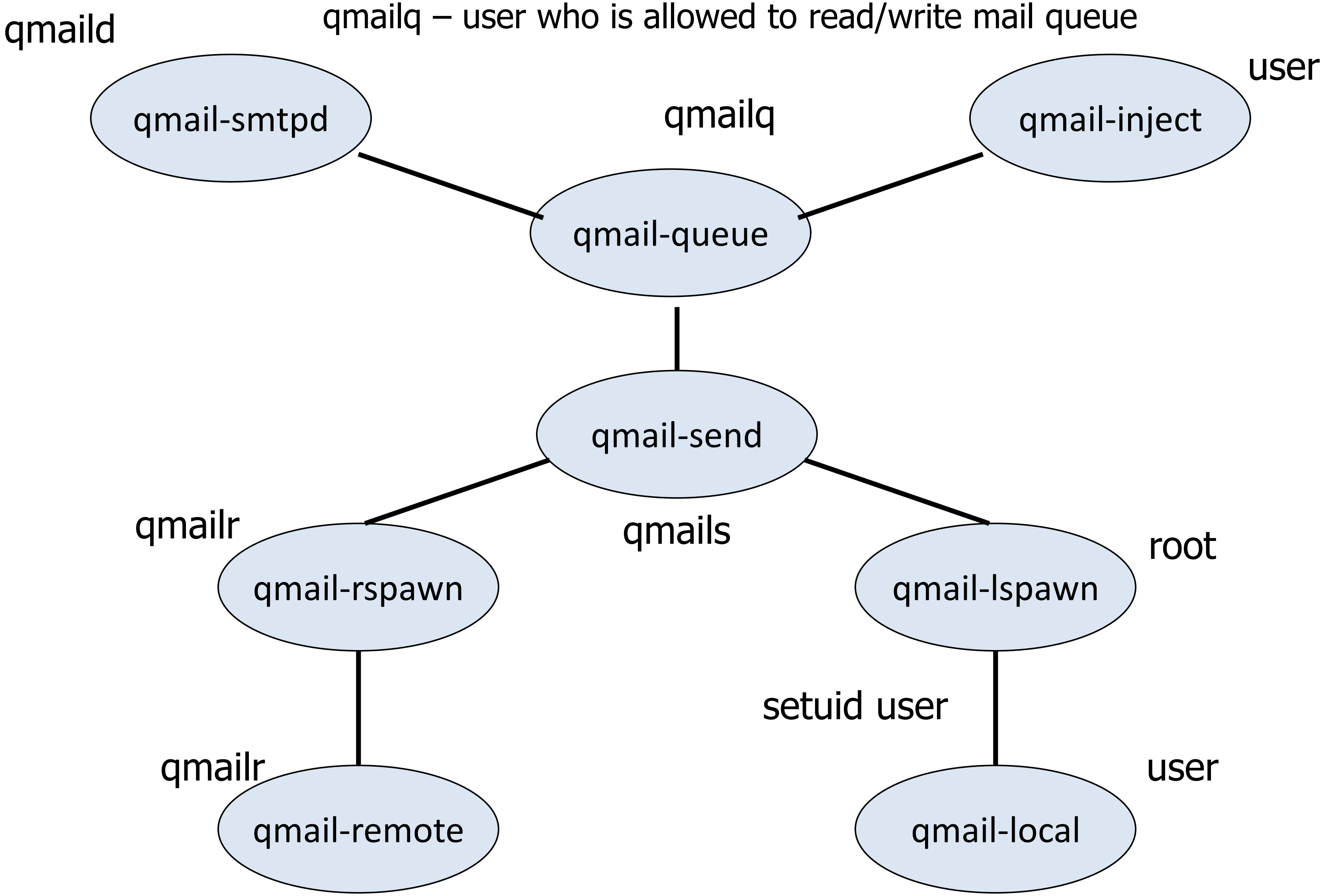
- Isolation based on OS isolation
  - Separate modules run as separate “users”
  - Each user only has access to specific resources
- Least privilege
  - Minimal privileges for each UID
  - Only one “setuid” program
    - setuid allows a program to run as different users
  - Only one “root” program
 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

Structure of qmail

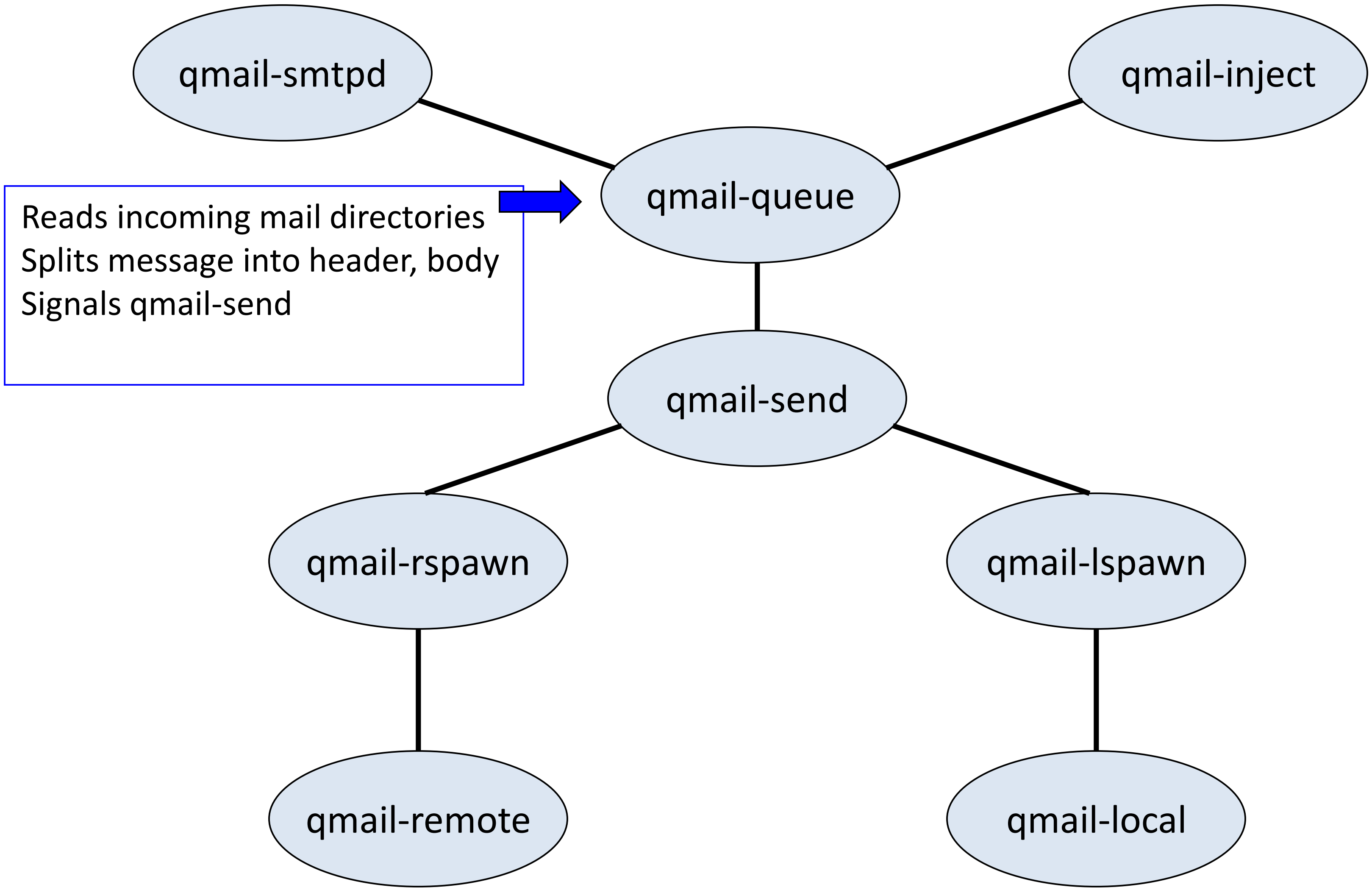


Isolation by Unix UIDs

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



Structure of qmail



qmail-smtpd

qmail-inject

qmail-queue

Reads incoming mail directories
Splits message into header, body
Signals qmail-send

qmail-send

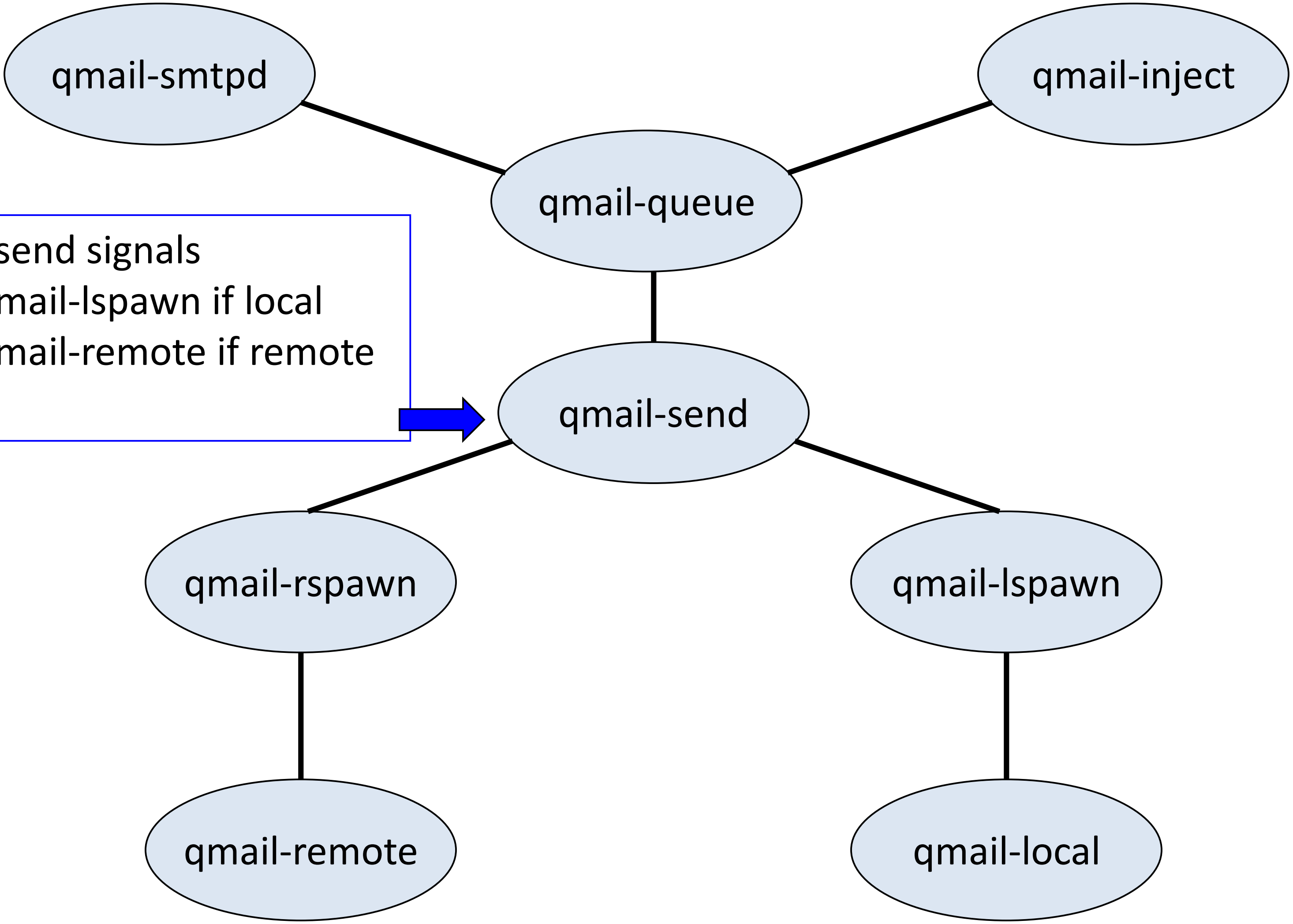
qmail-rspawn

qmail-lspawn

qmail-remote

qmail-local

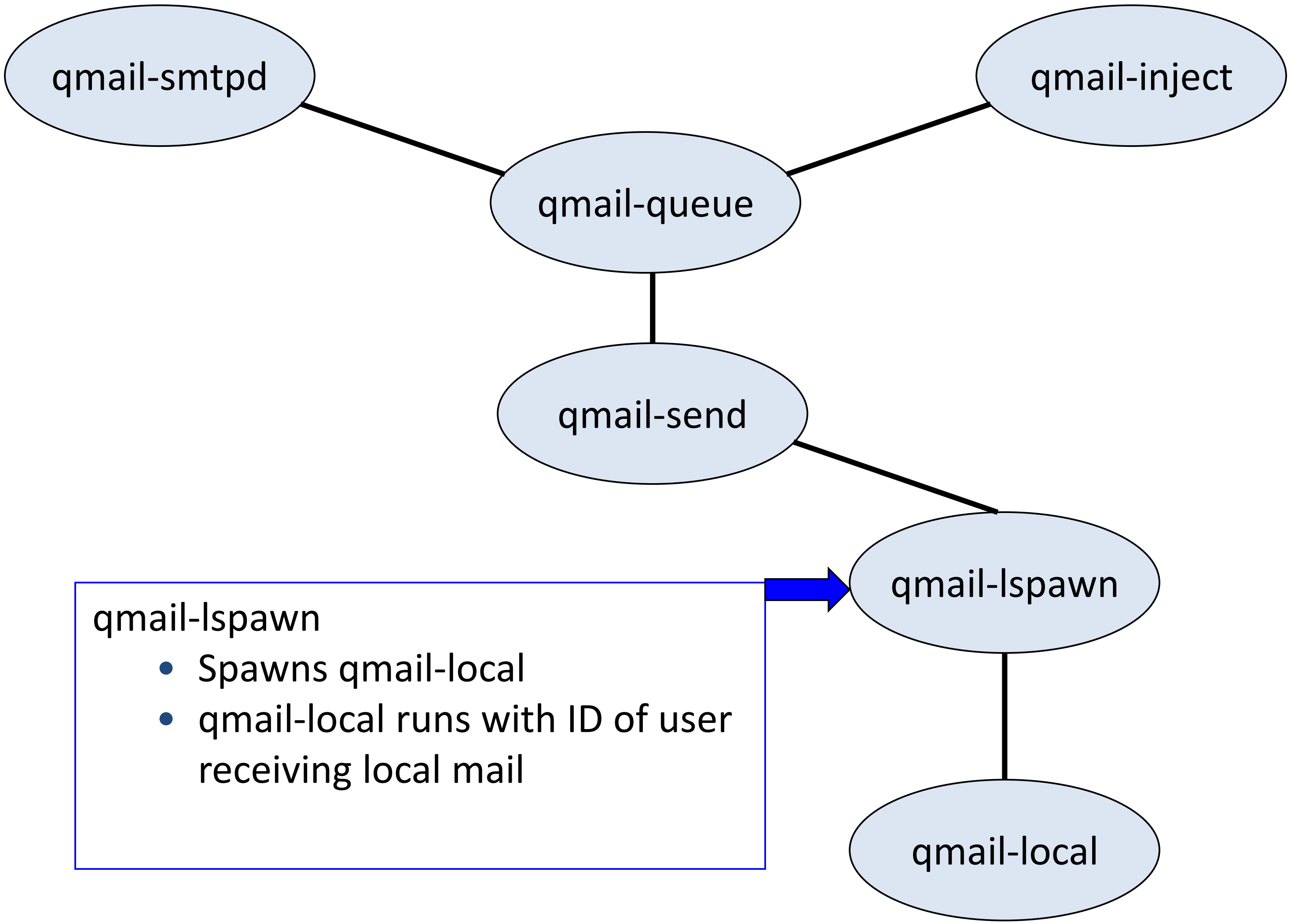
Structure of qmail



qmail-send signals

- qmail-lspawn if local
- qmail-remote if remote

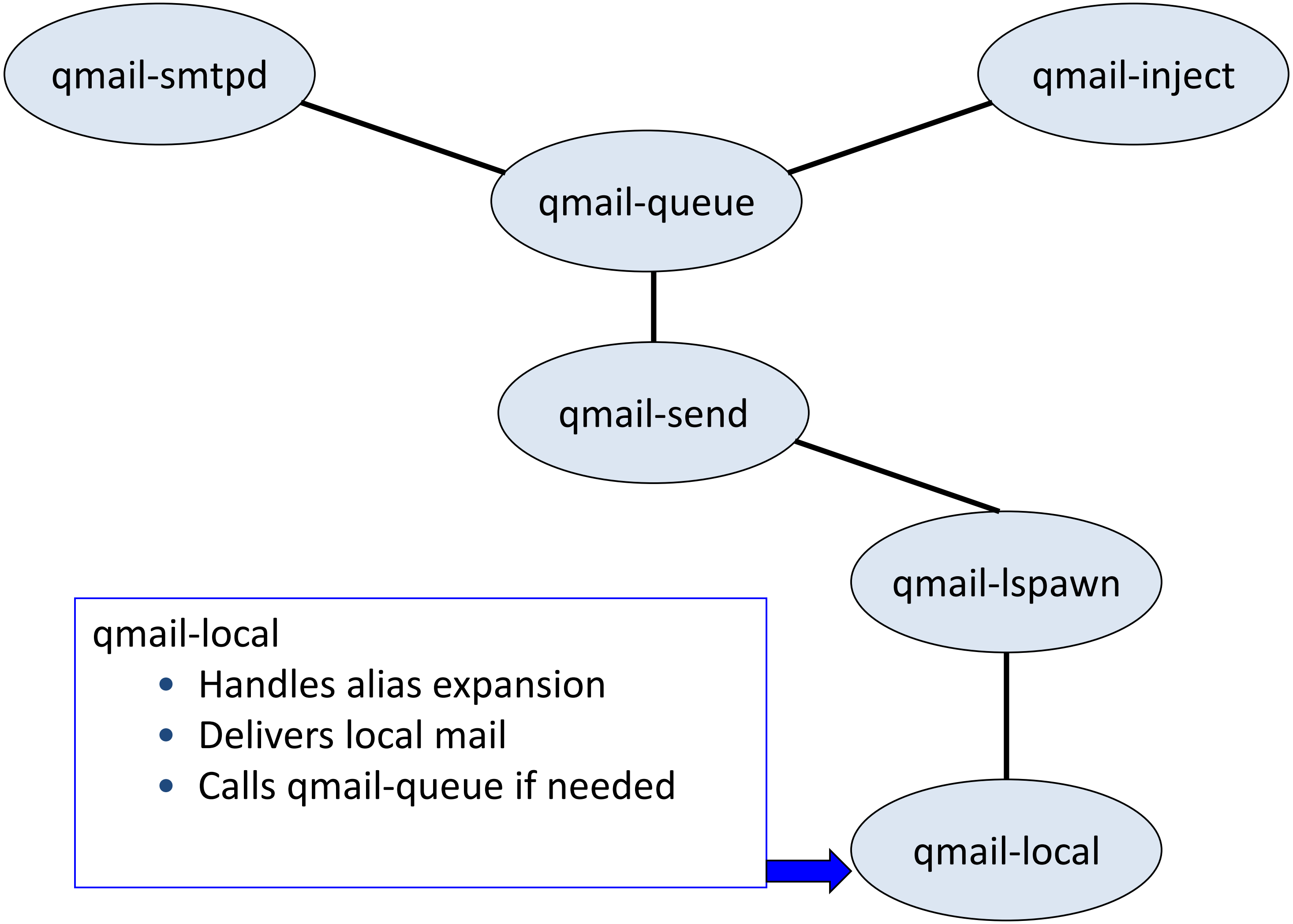
Structure of qmail



qmail-lspawn

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

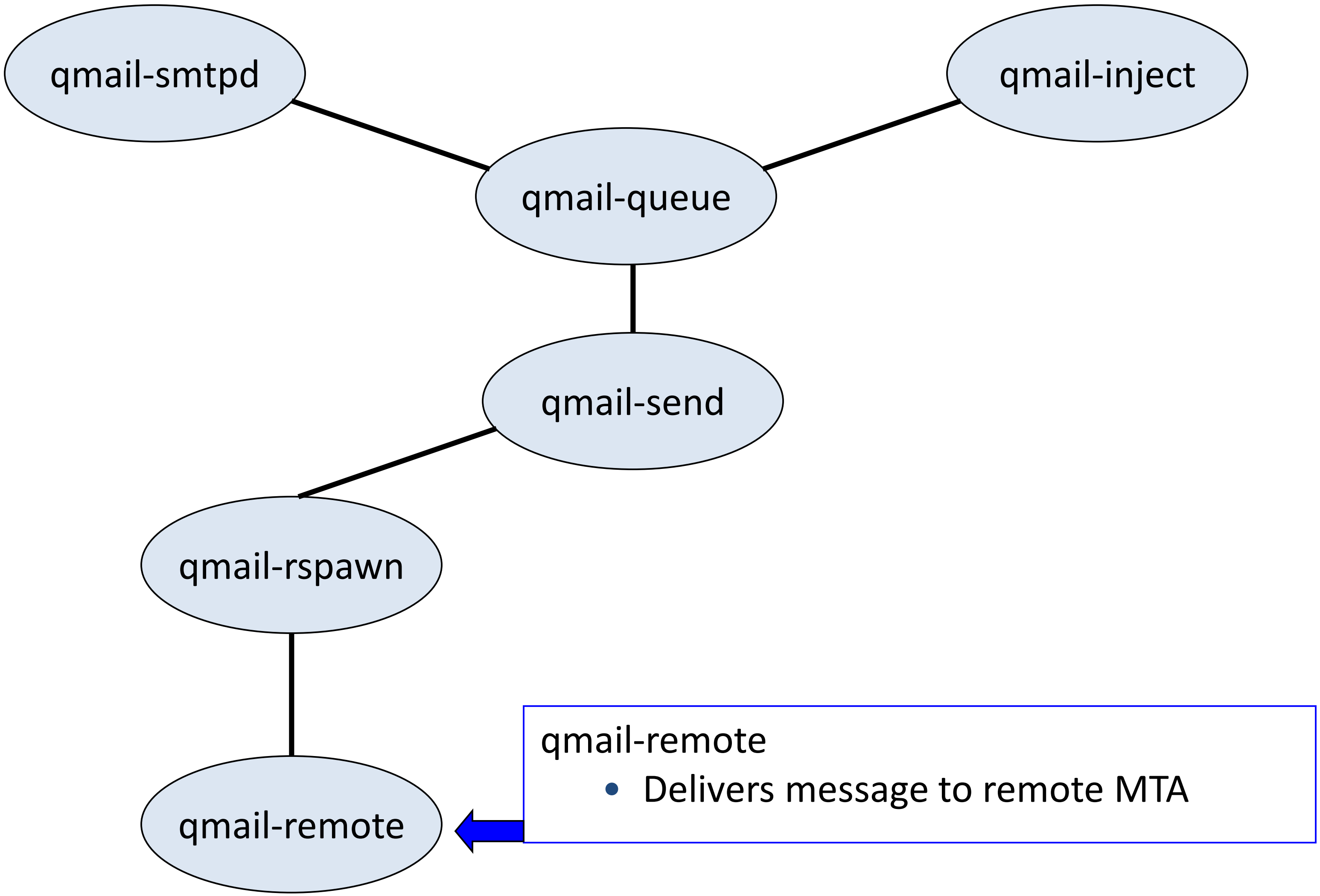
Structure of qmail



qmail-local

- Handles alias expansion
- Delivers local mail
- Calls qmail-queue if needed

Structure of qmail

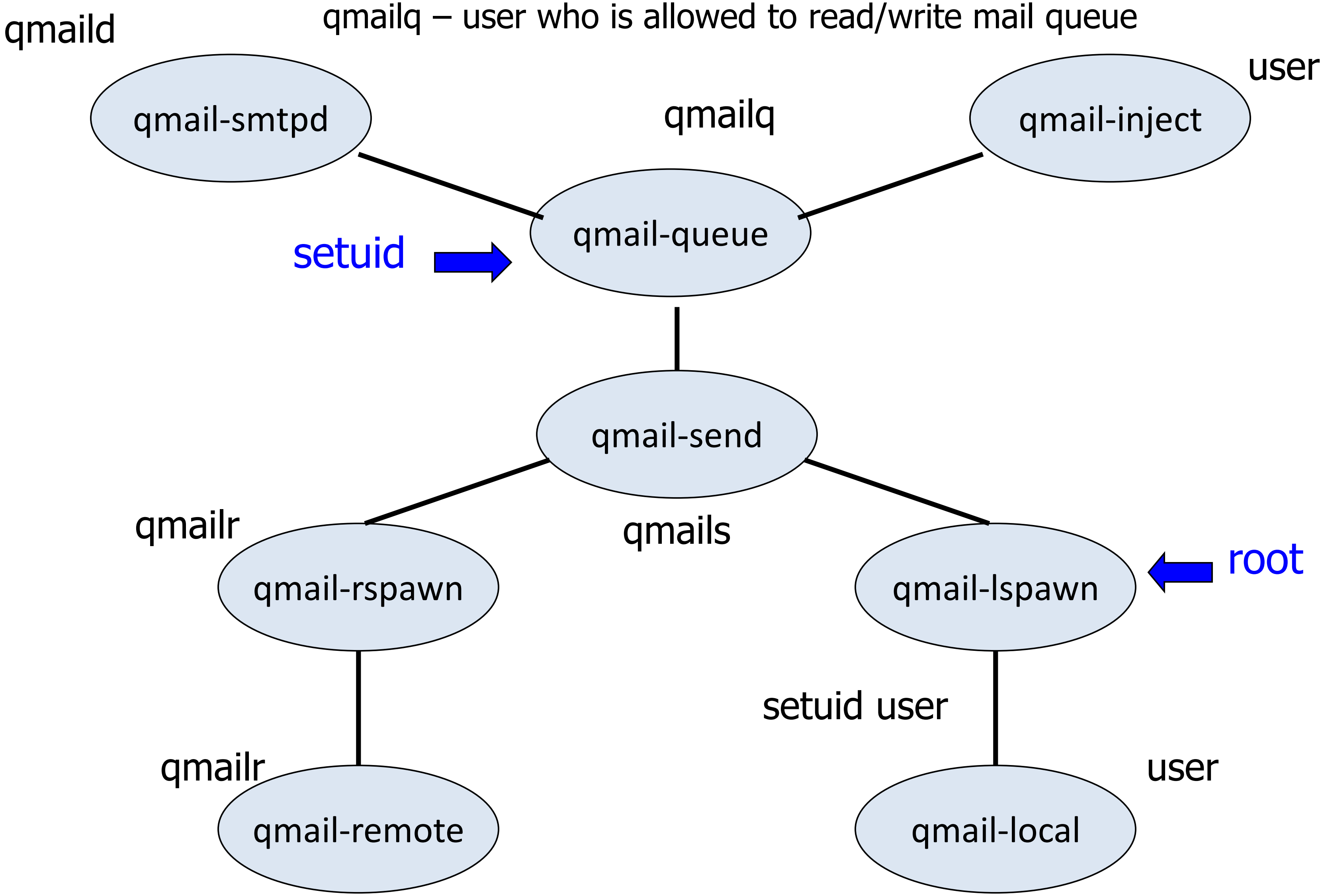


qmail-remote

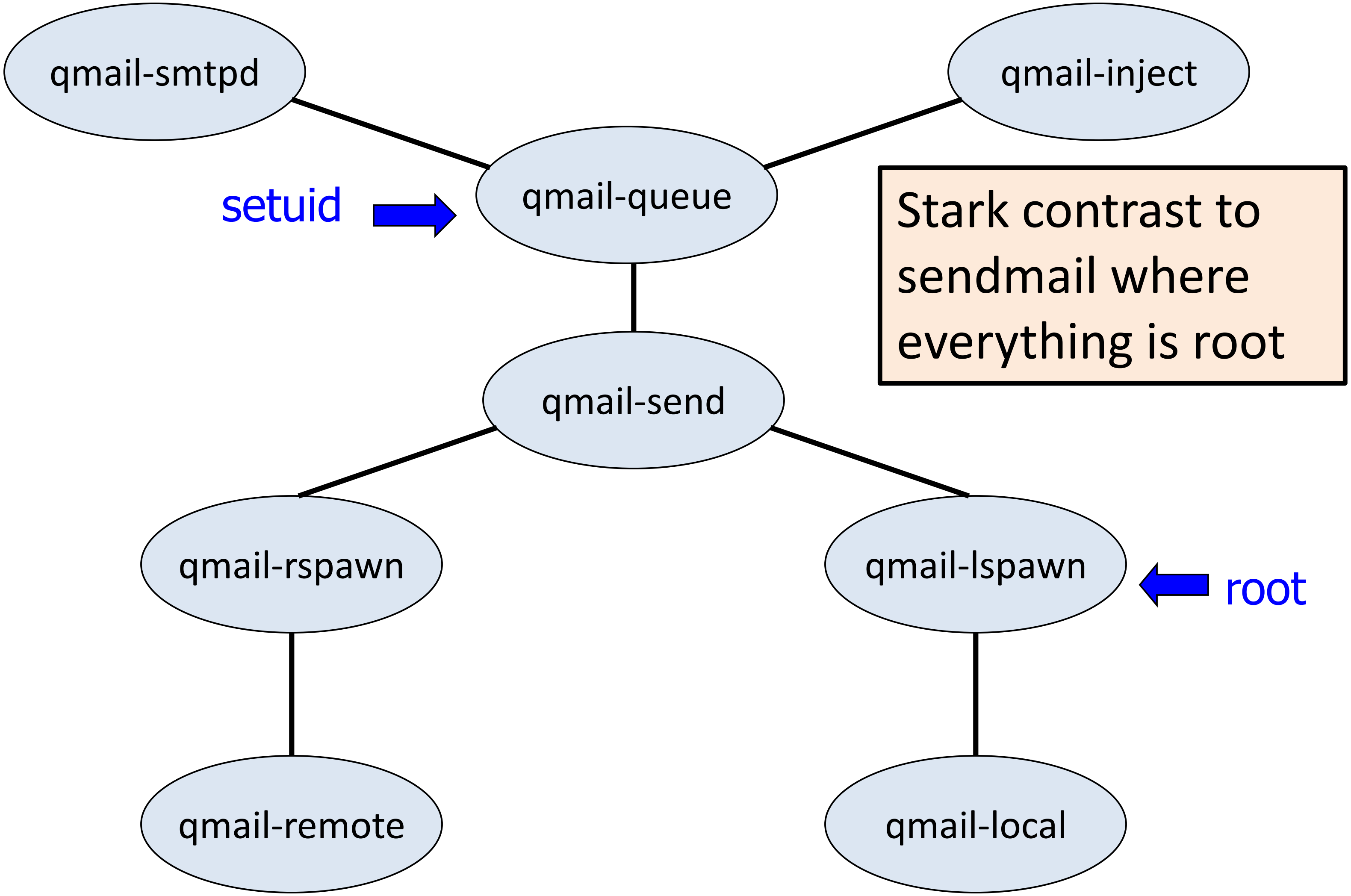
- Delivers message to remote MTA

Isolation by Unix UIDs

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



Least privilege



Stark contrast to sendmail where everything is root

DJB’s Seven Rules of Secure Design

- Execute untrusted input in a sandobox (.qmail)
- Do as little as possible w/ setuid (qmail-queue)
- Do as little as possible as root (qmail-lspawn)
- Compartmentalization and mutual distrust
- Parsing is just a bug factory, so avoid
- Keep it simple, stupid (no mailing lists, etc.)
- Write bug-free code (stralloc, getln)

https://cr.yp.to/qmail/guarantee.html

Qmail Paper

- The Security Architecture of Qmail

https://hillside.net/plop/2004/papers/mhafiz1/PLoP2004_mhafiz1_0.pdf

Android process isolation

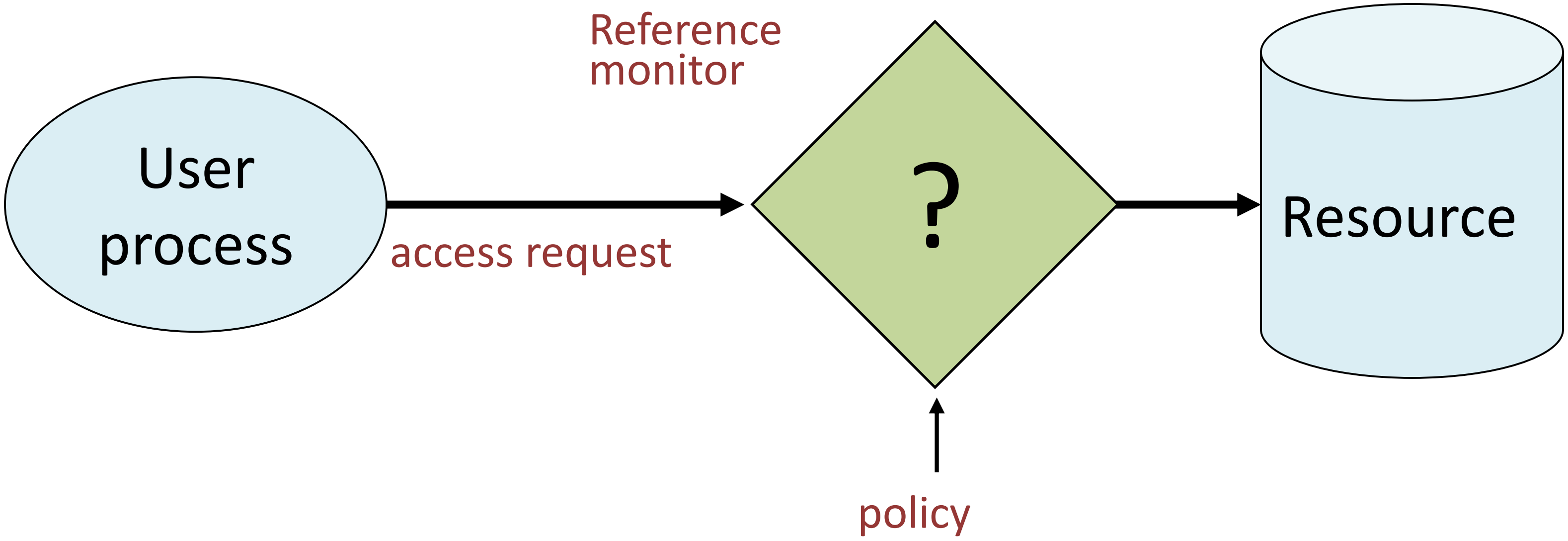
- Android application sandbox
  - Isolation: Each app runs w/ its own UID in own VM
    - Provides memory protection
    - Communication limited to using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Interaction: reference monitor checks permissions on inter-component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

Secure Architecture Principles

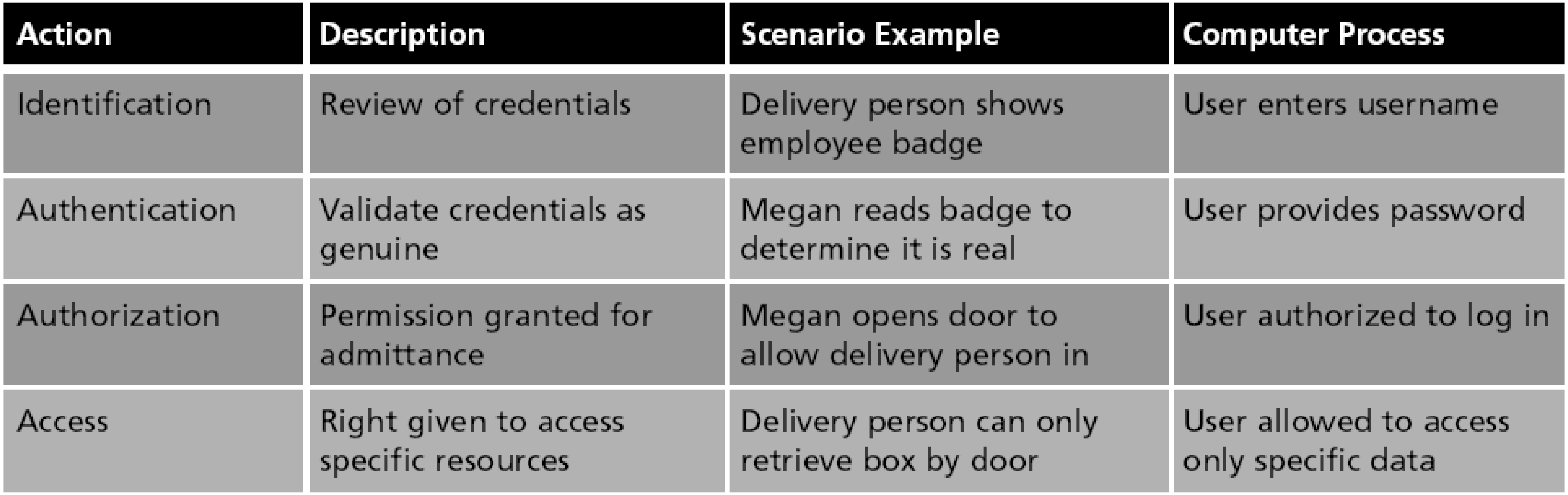
Access Control Concepts

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



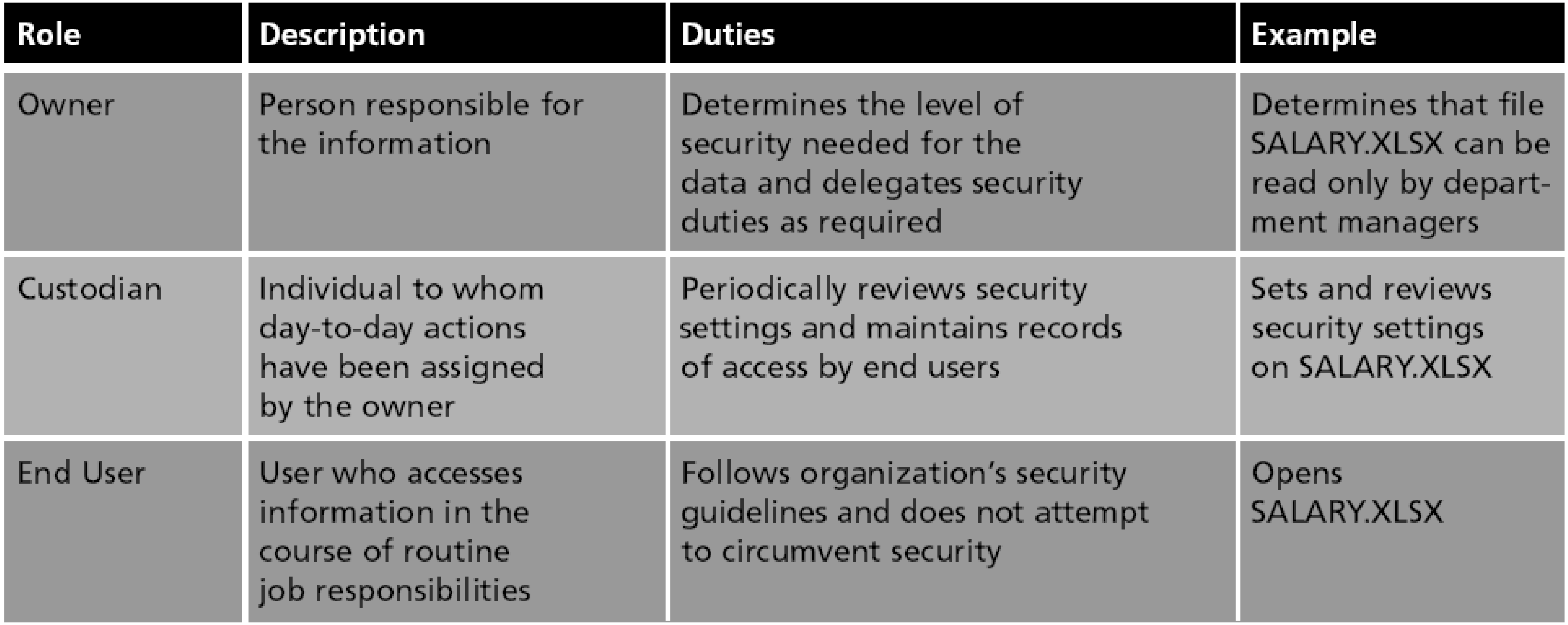
Access Control Terminology (continued)



Action	Description	Scenario Example	Computer Process
Identification	Review of credentials	Delivery person shows employee badge	User enters username
Authentication	Validate credentials as genuine	Megan reads badge to determine it is real	User provides password
Authorization	Permission granted for admittance	Megan opens door to allow delivery person in	User authorized to log in
Access	Right given to access specific resources	Delivery person can only retrieve box by door	User allowed to access only specific data

Table 7-1 Basic steps in access control

Access Control Terminology



Role	Description	Duties	Example
Owner	Person responsible for the information	Determines the level of security needed for the data and delegates security duties as required	Determines that file SALARY.XLSX can be read only by department managers
Custodian	Individual to whom day-to-day actions have been assigned by the owner	Periodically reviews security settings and maintains records of access by end users	Sets and reviews security settings on SALARY.XLSX
End User	User who accesses information in the course of routine job responsibilities	Follows organization's security guidelines and does not attempt to circumvent security	Opens SALARY.XLSX

Table 7-2 Roles in access control

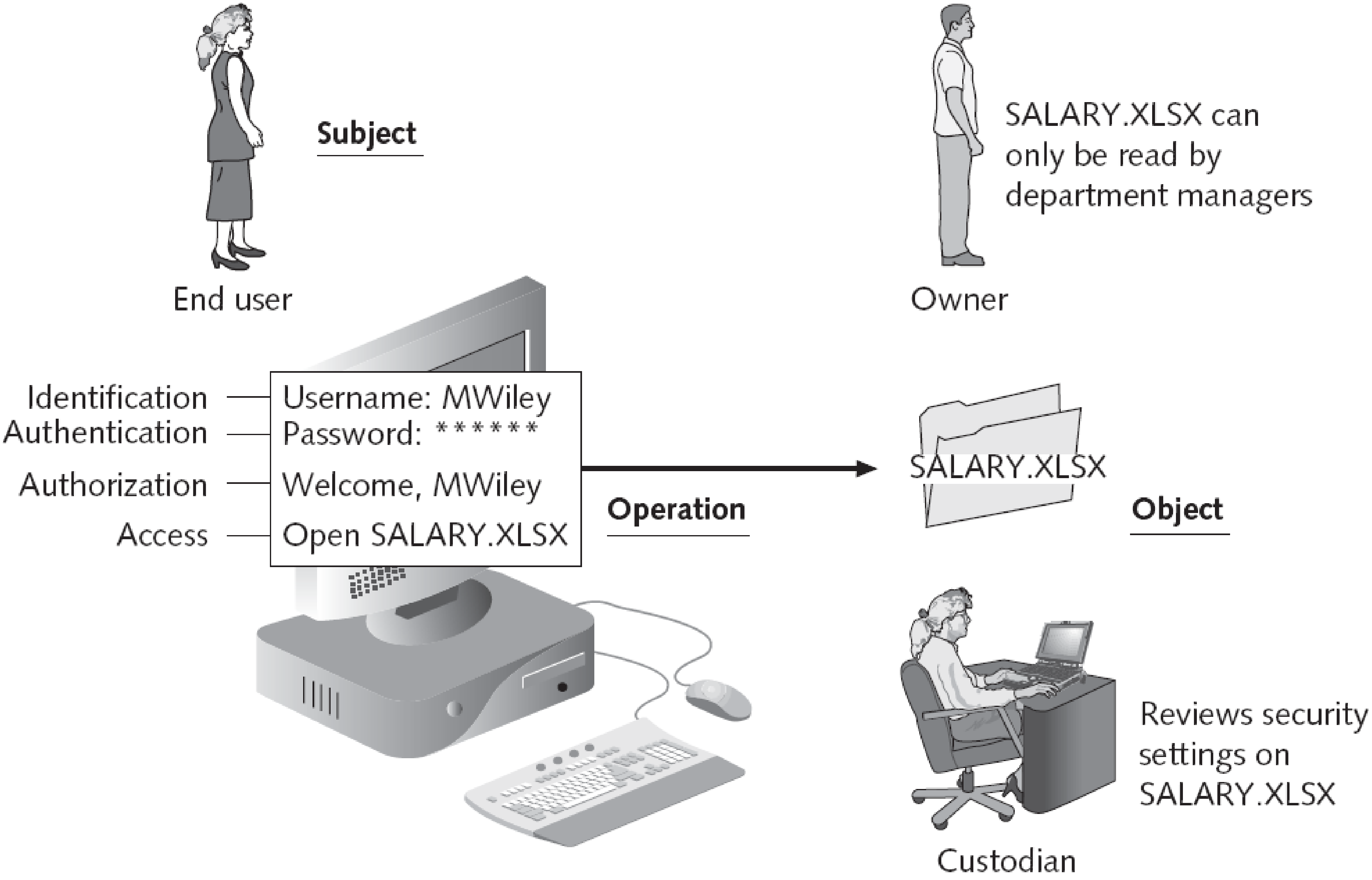
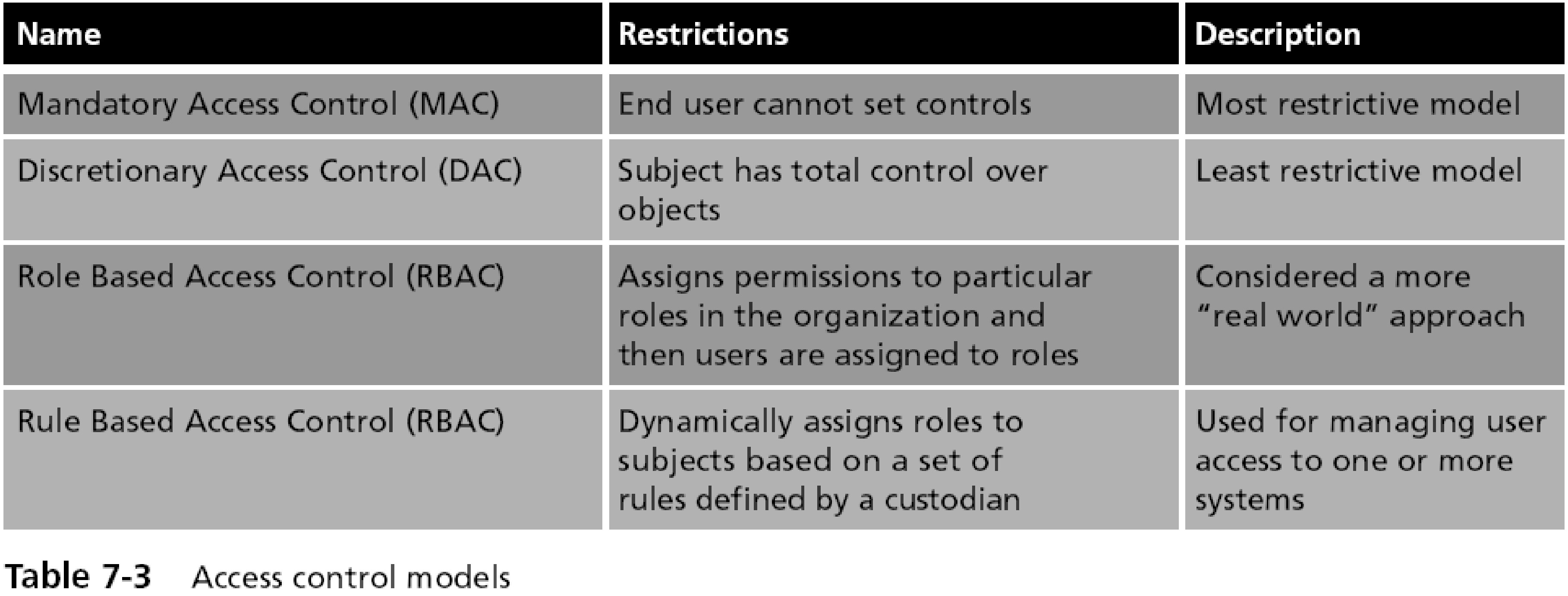


Figure 7-1 Access control process and terminology

Access Control Models (continued)



Name	Restrictions	Description
Mandatory Access Control (MAC)	End user cannot set controls	Most restrictive model
Discretionary Access Control (DAC)	Subject has total control over objects	Least restrictive model
Role Based Access Control (RBAC)	Assigns permissions to particular roles in the organization and then users are assigned to roles	Considered a more “real world” approach
Rule Based Access Control (RBAC)	Dynamically assigns roles to subjects based on a set of rules defined by a custodian	Used for managing user access to one or more systems

Table 7-3 Access control models

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

- 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

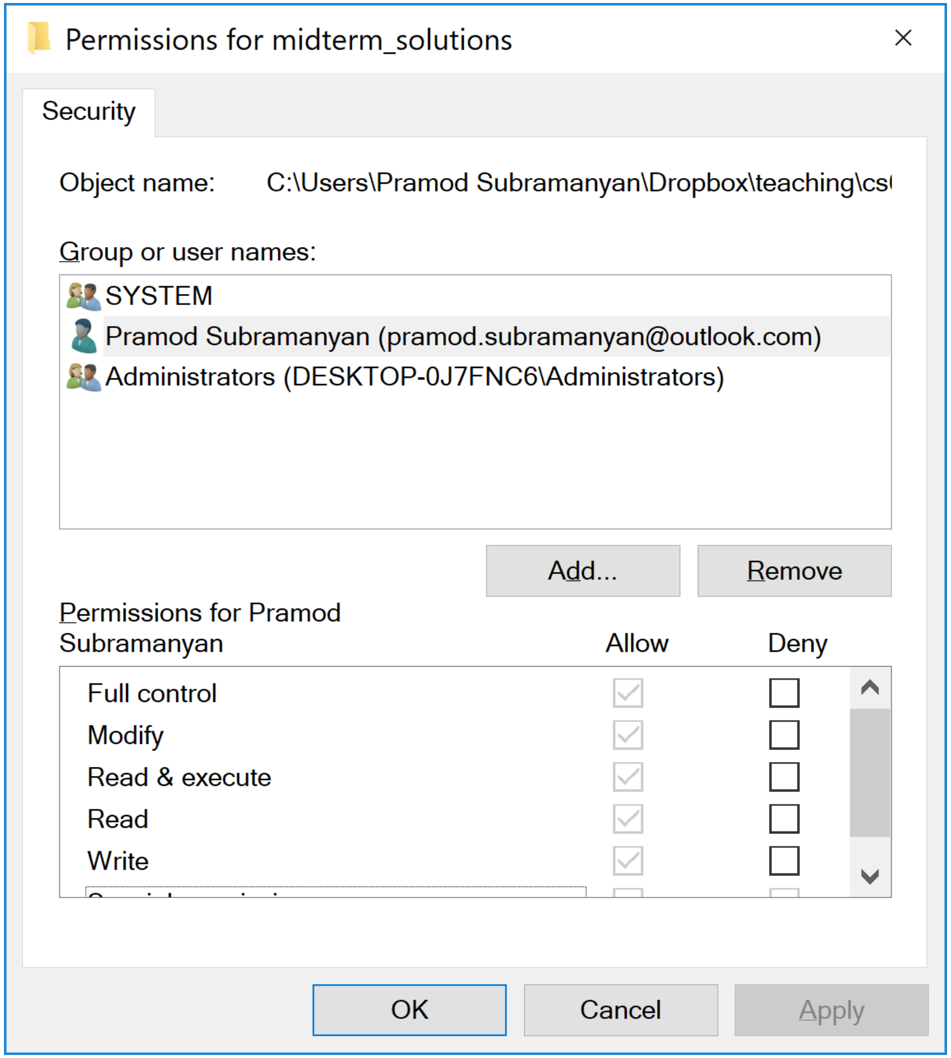
Access control lists are widely used, often with groups

Some aspects of capability concept are used in many systems

Access Control List (ACL)

- Set of permissions attached to object
- Specifies
  - which subjects allowed to access object
  - what operations they can perform on it
- Every file and folder has an ACL
- **Access control entry (ACE)**
  - Each entry in the ACL table in the Microsoft Windows, Linux, and Mac OS X operating systems

Access Control List on Windows



Permissions for midterm_solutions

Security

Object name: C:\Users\Pramod Subramanyan\Dropbox\teaching\cs

Group or user names:

- SYSTEM
- Pramod Subramanyan (pramod.subramanyan@outlook.com)
- Administrators (DESKTOP-0J7FNC6\Administrators)

Permissions for Pramod Subramanyan

	Allow	Deny
Full control	☑	□
Modify	☑	□
Read & execute	☑	□
Read	☑	□
Write	☑	□

Examples of Capabilities

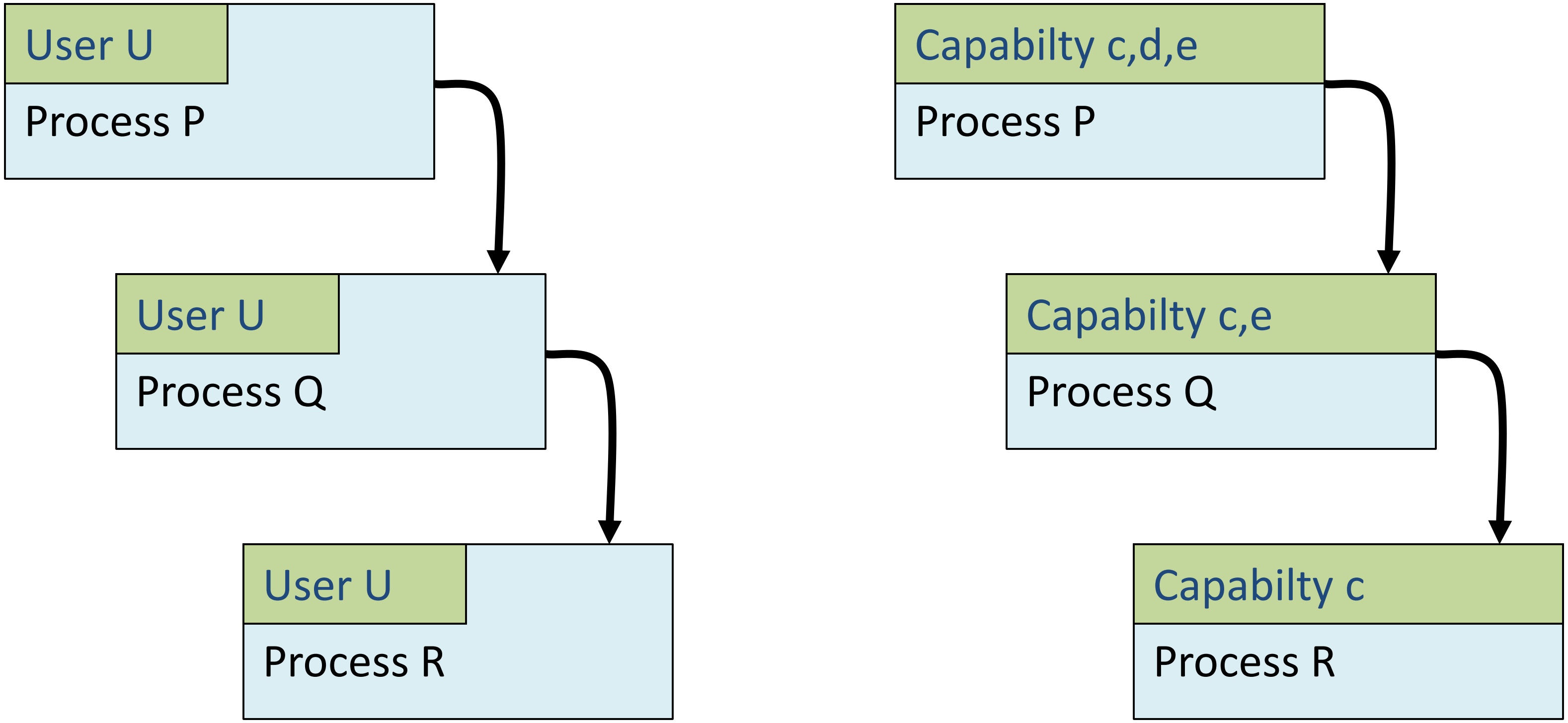
If you’ve used GDocs, you’ve used capabilities

- Unforgeable ticket is the sharing URL:
  - https://docs.google.com/document/d/ 1mxu7DZzw4Ekt7hoQ0__ EJjxAjZgKc1G2WTriz1y-Qfj /edit
- Can pass this capability to others
  - Which is a blessing and a curse
- Can revoke capability if needed
- Different capabilities may have diff privileges

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

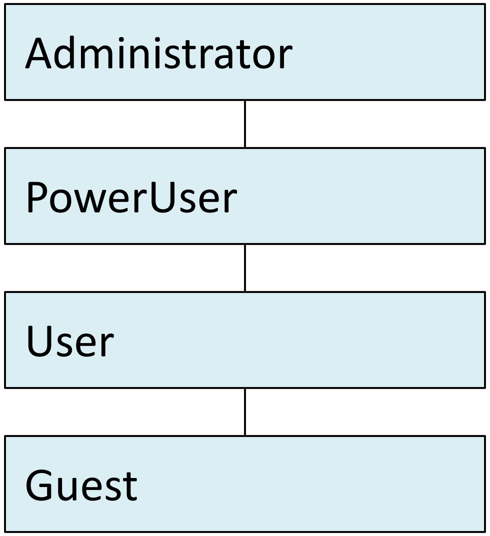
Process R

ACL vs Capabilities

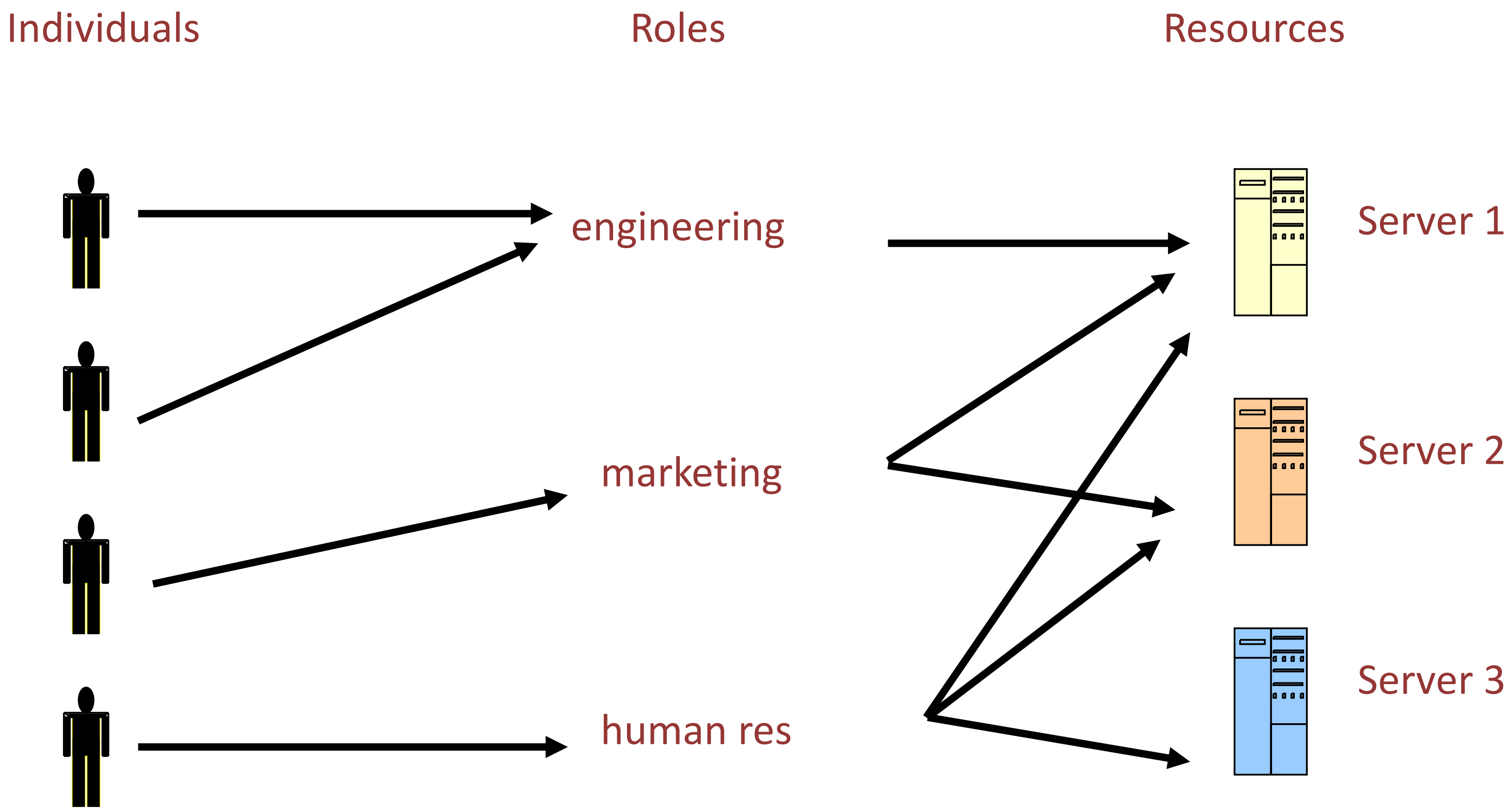
- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some systems if appropriate bookkeeping
      - OS knows which data is capability
      - If capability used 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

Where have you seen roles in action?

- Piazza/Moodle
  - Student vs. TA vs. Instructor
- Pingala
  - Staff vs. Faculty vs. Student
- Each role has different privileges
  - Examples?
  - Instructors can enter grades but students can’t

Access control summary

- Access control involves reference monitor
  - Check permissions: $\langle$user info, action$\rangle \rightarrow$ 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

Implementations of Access Control

- Unix processes and permissions
  - Controls files, sockets, etc.
- Android/iOS applications
  - Access to phone, contacts, GPS, etc.
- Role-based Access Control
  - Moodle/Piazza: Instructor vs student vs TA
  - Pingala: Staff vs Faculty vs Student
  - GitLab: Developer, Master, Reporter etc.

How to build secure applications?

What techniques have we learned so far?

- First, isolate untrusted processes
  - UID based, System call interposition, SFI, VMMs
- Architect system using principle of least privilege
  - System only as strong as the weakest link
  - Any single compromise shouldn’t cause much damage
- Use access control within and for app
  - Distinguish users/roles; use capabilities
  - Logging and accountability is important

None of this refers to writing code!

What techniques have we learned so far?

- First, isolate untrusted processes
  - UID based, System call interposition, SFI, VMMs
- Architect system using principle of least privilege
  - System only as strong as the weakest link
  - Any single compromise shouldn’t cause much damage
- Use access control within and for app
  - Distinguish users/roles; use capabilities
  - Logging and accountability is important

Extra Content Follows

END OF MODULE 3

Let’s talk about the code

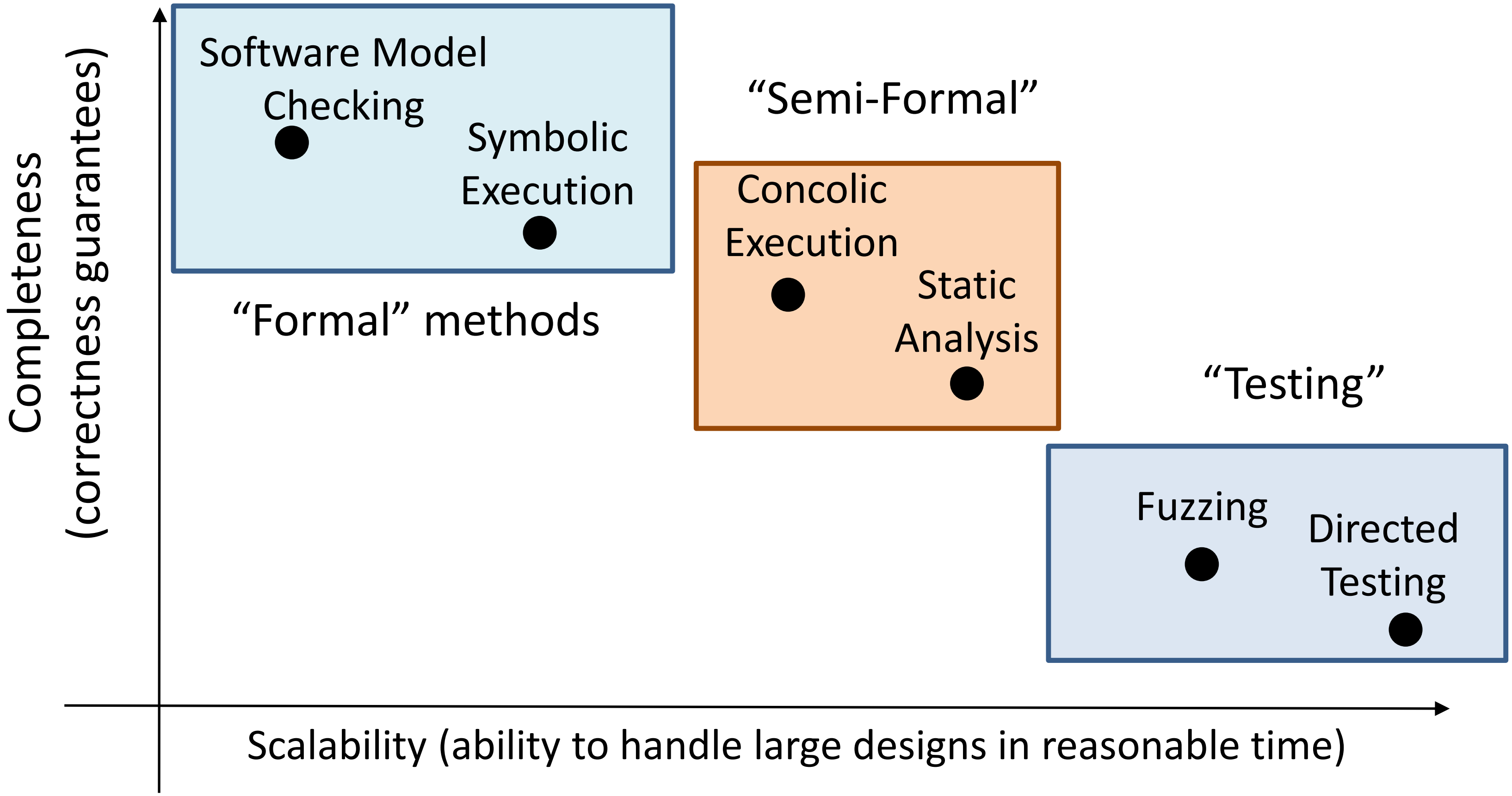
How to write secure code?

- Use memory safe languages
  - Java, ML, Scala, C#, Haskell, Python, Clojure, F#, Rust, … literally anything other than C/C++
- Follow secure coding guidelines
  - https://www.oracle.com/technetwork/java/seccodeguide-139067.html
  - https://insights.sei.cmu.edu/sei_blog/2017/04/cert-c-secure-coding-guidelines.html
  - …
- Testing/Validation/Verification of code
  - Range of methods from static analysis to fuzzing to dynamic analysis to model checking

Secure Architecture Principles

Testing and Validation

Tradeoff Space of Verification Methods



Functional vs Security Validation

- **Functionality Testing: Test plan**
  - The product specifications
  - User documentation
  - Use cases
- **Security Testing: Security plan**
  - Document security target or weakness in defenses
  - Perform Threat Modeling and Analysis
  - Derive security test cases based on the threat model

Testing Security Feature

vs

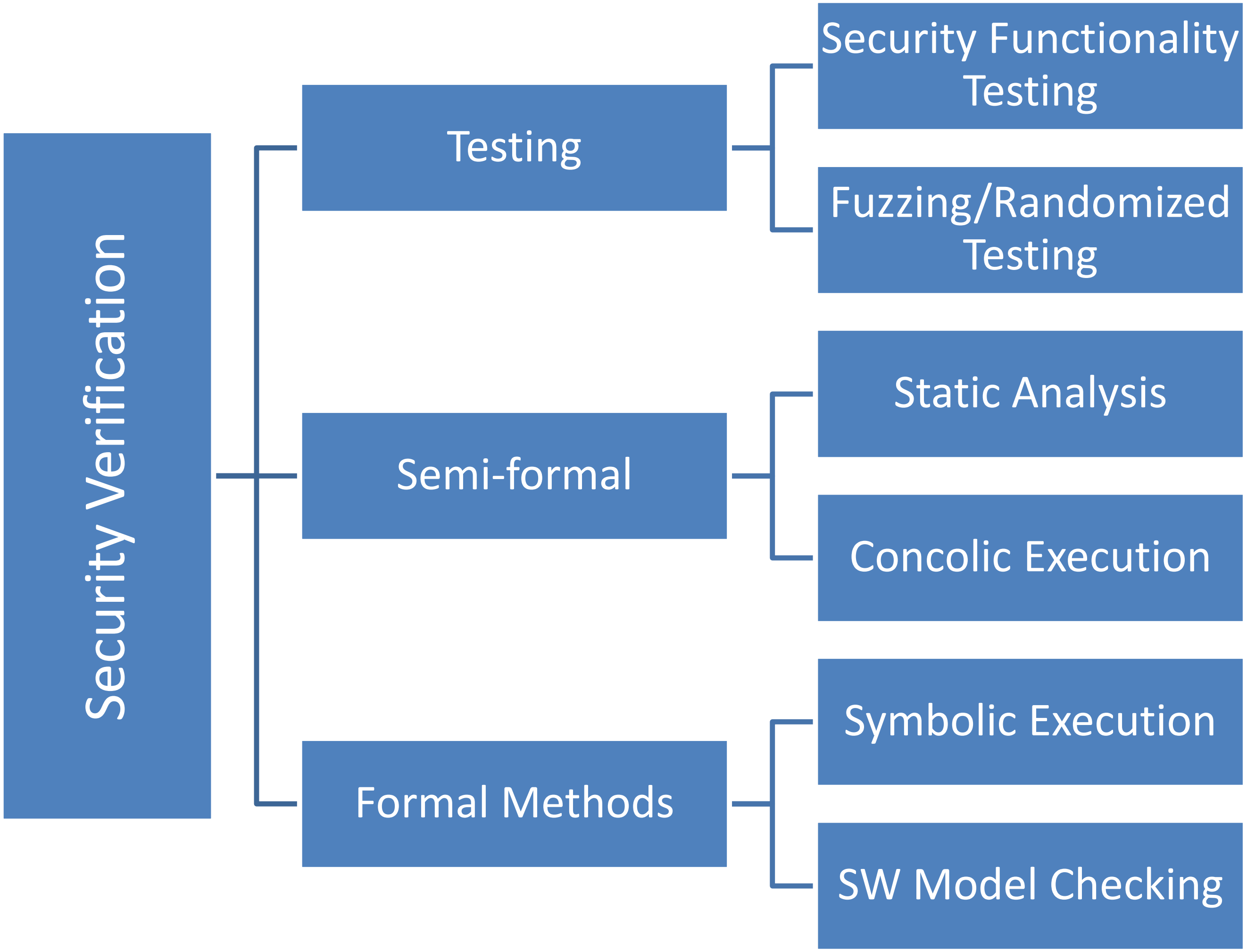
Testing How Secure the Feature is

Writing a Security Test Plan

- Develop deep understanding of target system
- Review threat model and risk analysis
- Investigate known vulnerabilities and attacks
- Think out of the box, like an attacker would
- Identify weak areas for attack
- Ask what if, how would, when does, etc
- Create test cases to break into the system
- Verify defenses against known attack techniques

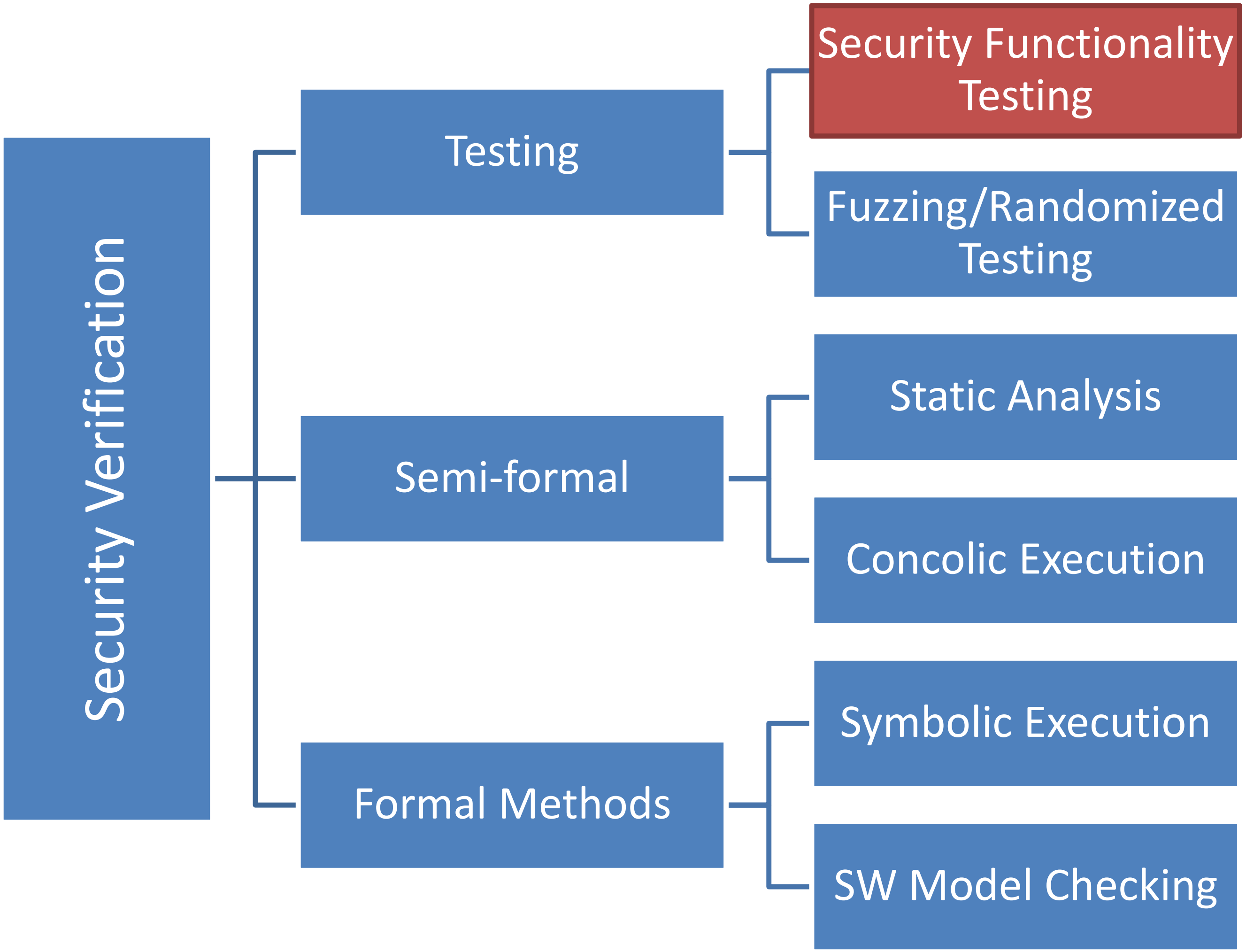
END OF MODULE 3

Security Verification Techniques



- Security Verification
  - Testing
    - Security Functionality Testing
    - Fuzzing/Randomized Testing
  - Semi-formal
    - Static Analysis
    - Concolic Execution
  - Formal Methods
    - Symbolic Execution
    - SW Model Checking

Security Verification Techniques



Testing Security Functionality

- Ensure security use cases working as specified
- Explore invalid/incorrect untrusted inputs
- Check for negative functionality
  - Unauthorized accesses are denied
  - File write fails
- Stress test
  - Out-of-memory conditions
  - Processes in multiprocess app killed randomly

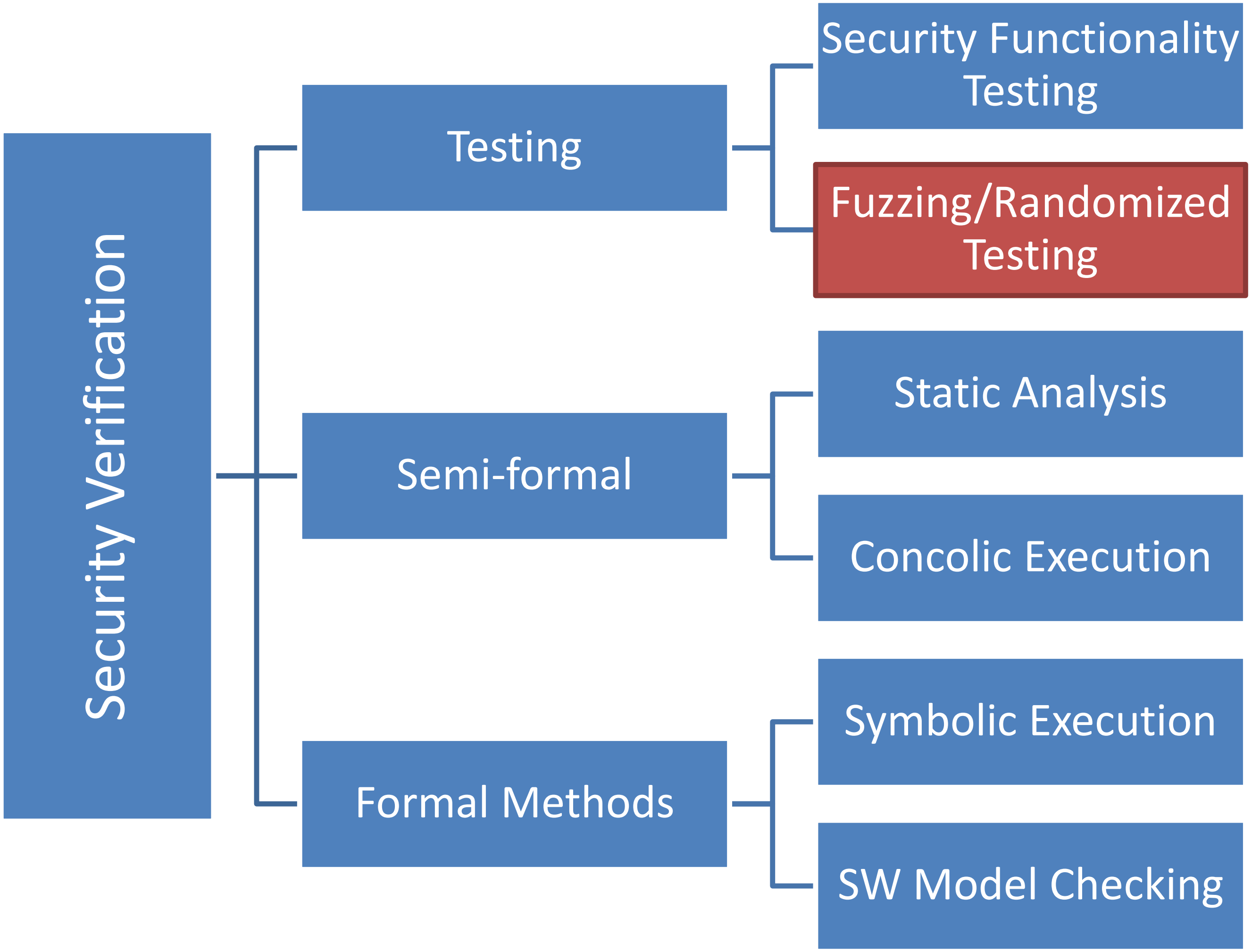
What are good tests for this code?

```
#define N 3

uint8_t tbl[] = { 1, 1 }; // &tbl = 0x100
uint8_t data = 3;         // &data=0x102
uint8_t IO_REG = 1;       // &IO_REG=0x200

void foo(int x) {
  if (x < 0 || x >= N) return;
  IO_REG = tbl[x];
}
```

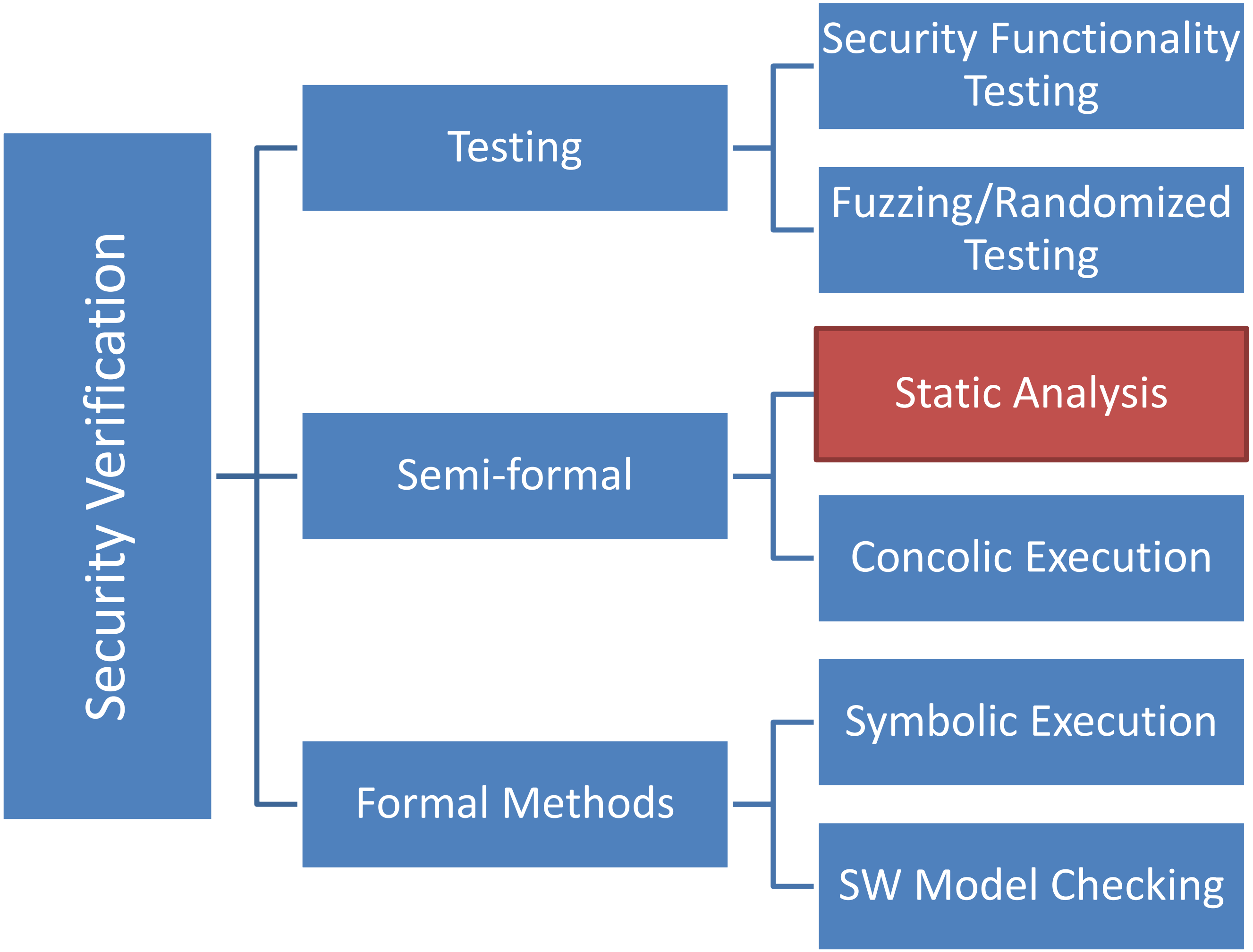
Security Verification Techniques



Fuzzing and Randomized Testing

- Fuzzing
  - Generate malformed inputs to test untrusted interfaces
    - Protocol Fuzzers: WiFi, CAN, PCI, USB, …
    - File parser: Boot image, system configuration, user data, …
    - API: Crypto engine, Graphics engine, …
  - Smart fuzzers understand protocol handshakes/FSMs and generate meaningful responses based on current context
- Directed Randomized Testing
  - Exercise internal implementations to ensure corner cases, exceptions, and errors are handled securely
  - Need special debug hooks to inject malformed traffics and monitor behaviors

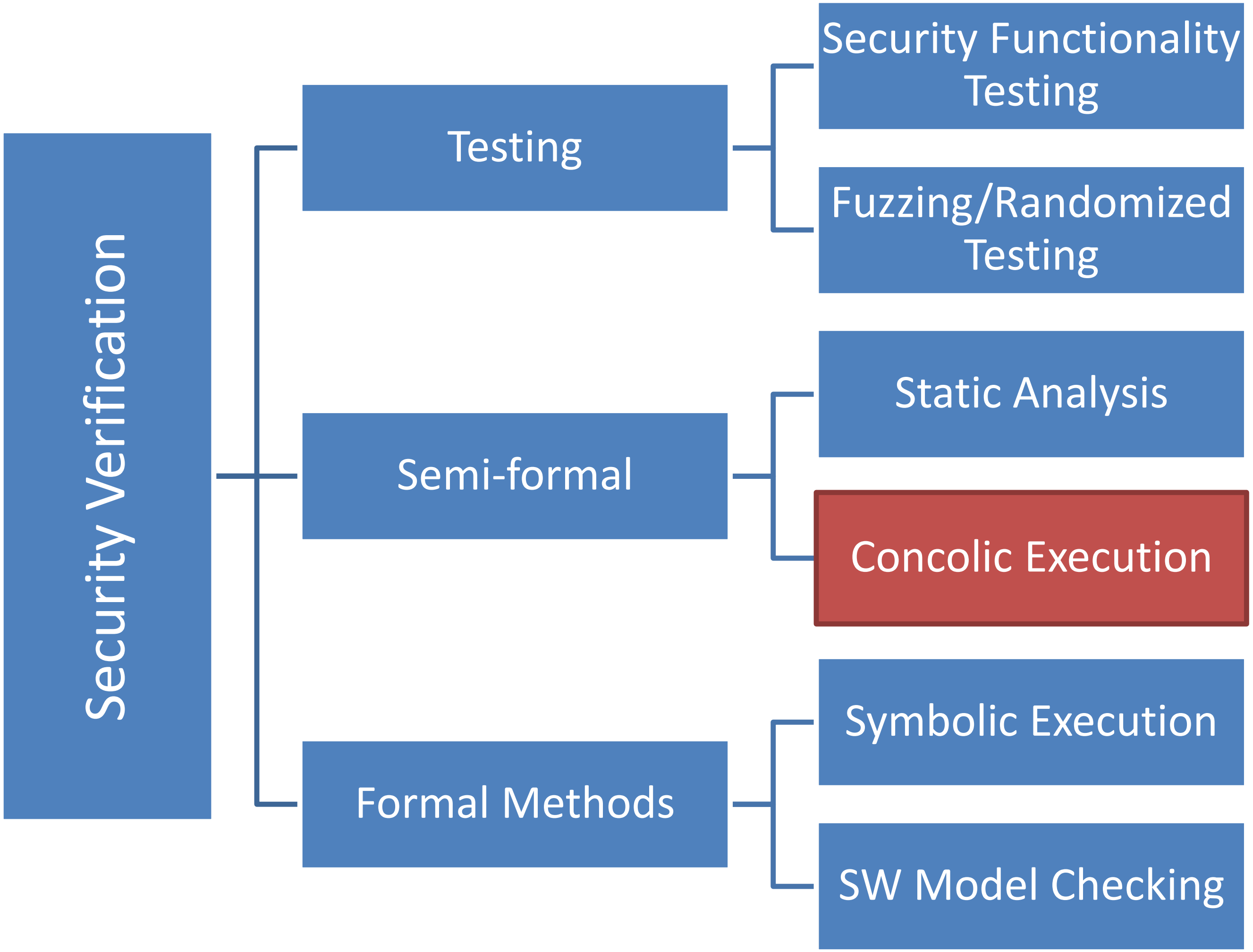
Security Verification Techniques



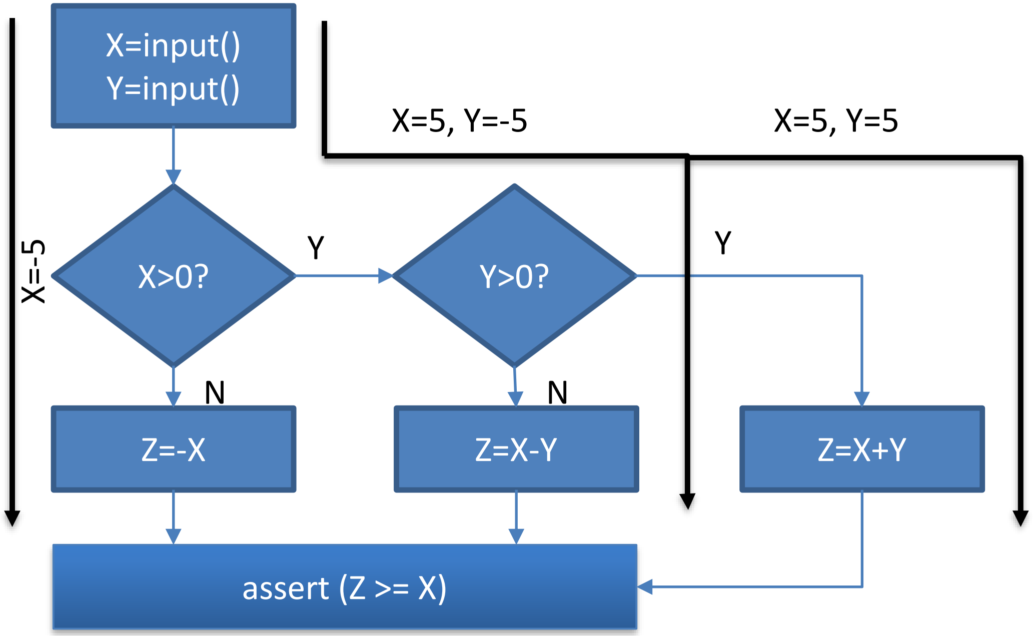
Static Analysis

- Perform heuristic/rule-based/analysis-based verification on the source code
- Software Static Analysis
  - Many commercial tools available (e.g., Coverity)
  - Mature tools typically detect many common vulnerabilities quite well (e.g., buffer overflow, use-after-free, off-by-one)
- Main disadvantages
  - Unsoundness, false positives

Security Verification Techniques



Concolic Execution for Test Generation

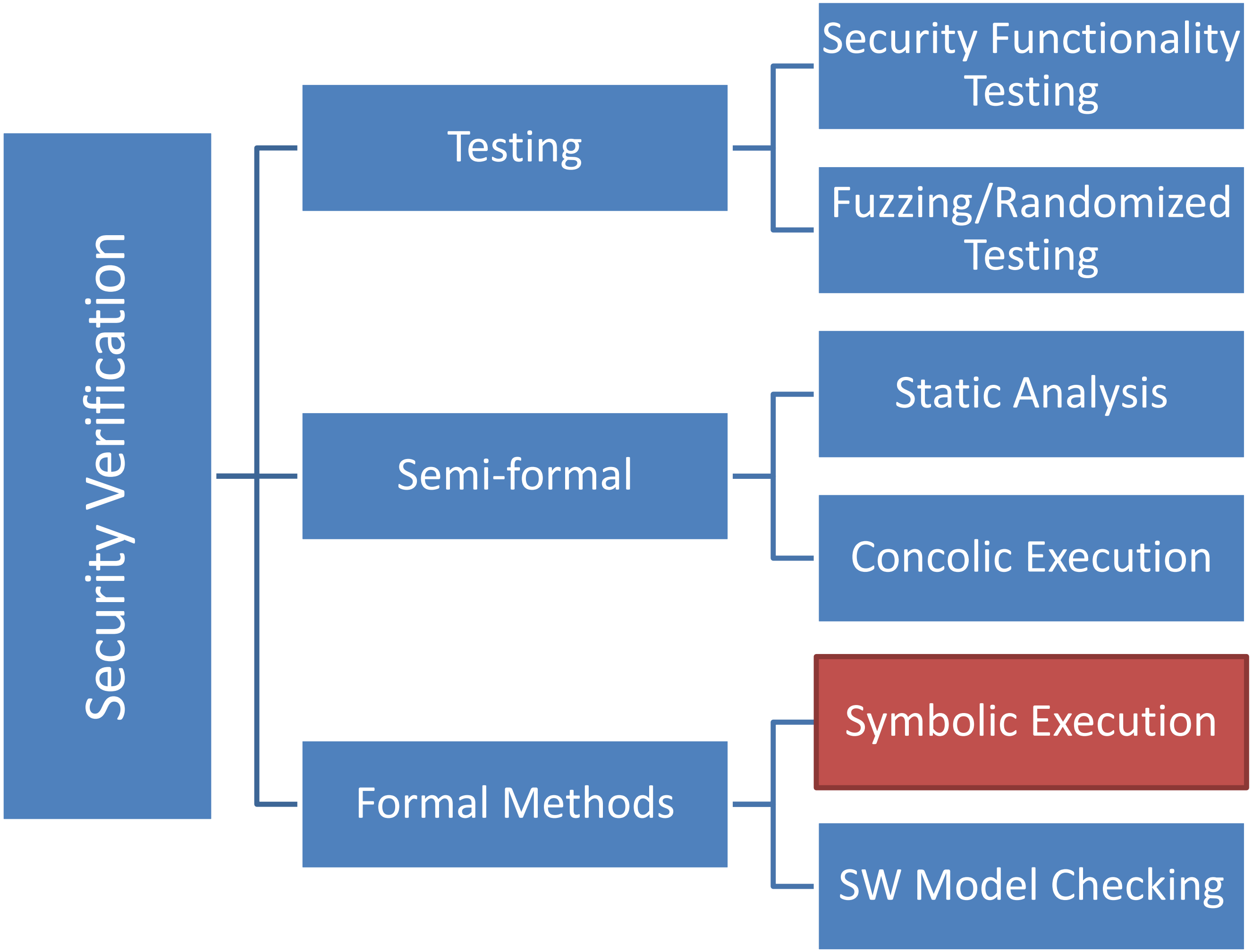


Can this prove properties?

Simply running tests is not enough; e.g., X+Y may overflow.

- Systematically generate tests to cover all program paths
- Open source tools like KLEE do this for C programs

Security Verification Techniques



Security Verification

Testing

Security Functionality Testing

Fuzzing/Randomized Testing

Semi-formal

Static Analysis

Concolic Execution

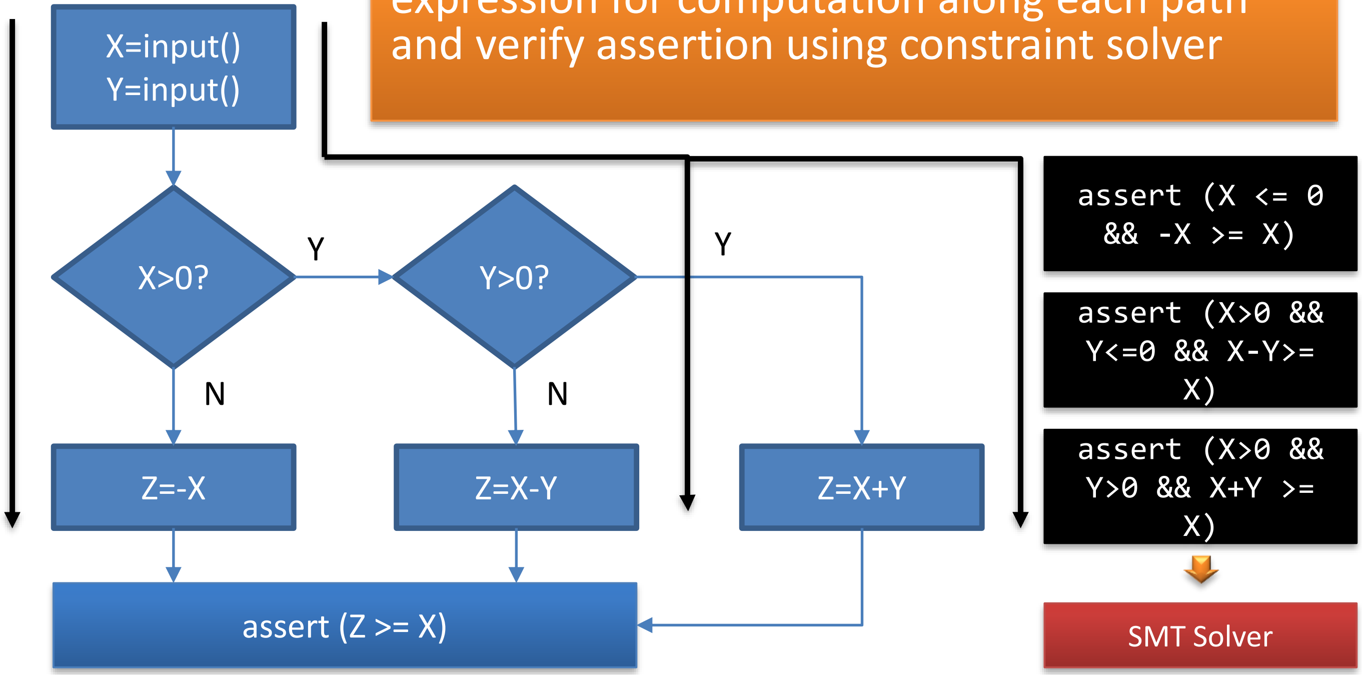
Formal Methods

Symbolic Execution

SW Model Checking

Symbolic Execution

Active Property Checking: Create symbolic expression for computation along each path and verify assertion using constraint solver



X=input()
Y=input()

X>0?

Y

N

Y>0?

Y

N

Z=-X

Z=X-Y

Z=X+Y

assert (Z >= X)

```
assert (X <= 0
&& -X >= X)
```

```
assert (X>0 &&
Y<=0 && X-Y>=
X)
```

```
assert (X>0 &&
Y>0 && X+Y >=
X)
```

SMT Solver

In Summary

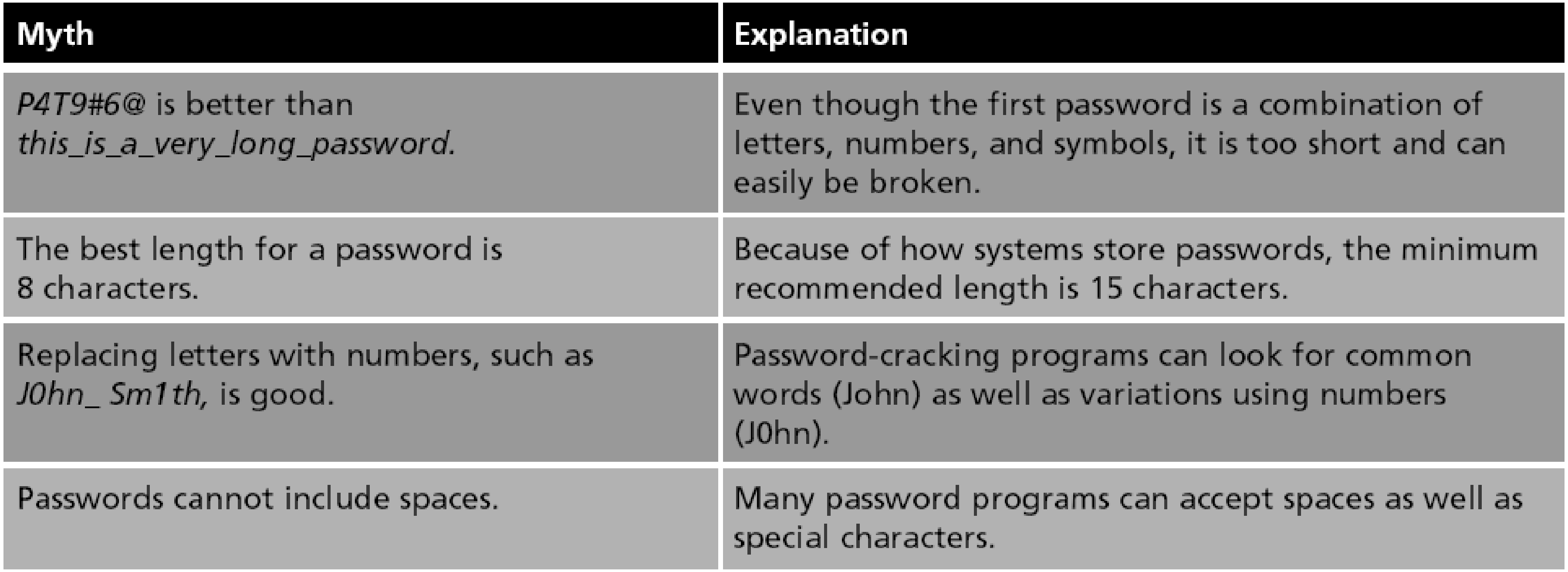
- There are a lot of tools out there, use them!
  - [IMO] #1 cause of security vulnerabilities is clueless/overconfident developers
- This is a good space to make money
  - People don’t pay for compilers or operating systems today! Not even for processor designs
  - But they do pay for security verification tools
  - Verification may be mandated by regulations

ACCESS CONTROL USING PASSWORDS

Passwords

- The most common logical access control
- Sometimes referred to as a logical token
- A secret combination of letters and numbers that only the user knows
- A password must be of a sufficient length and complexity so that an attacker cannot easily guess it (password paradox)

Passwords Myths



Myth	Explanation
P4T9#6@ is better than *this_is_a_very_long_password.*	Even though the first password is a combination of letters, numbers, and symbols, it is too short and can easily be broken.
The best length for a password is 8 characters.	Because of how systems store passwords, the minimum recommended length is 15 characters.
Replacing letters with numbers, such as *J0hn_ Sm1th,* is good.	Password-cracking programs can look for common words (John) as well as variations using numbers (J0hn).
Passwords cannot include spaces.	Many password programs can accept spaces as well as special characters.

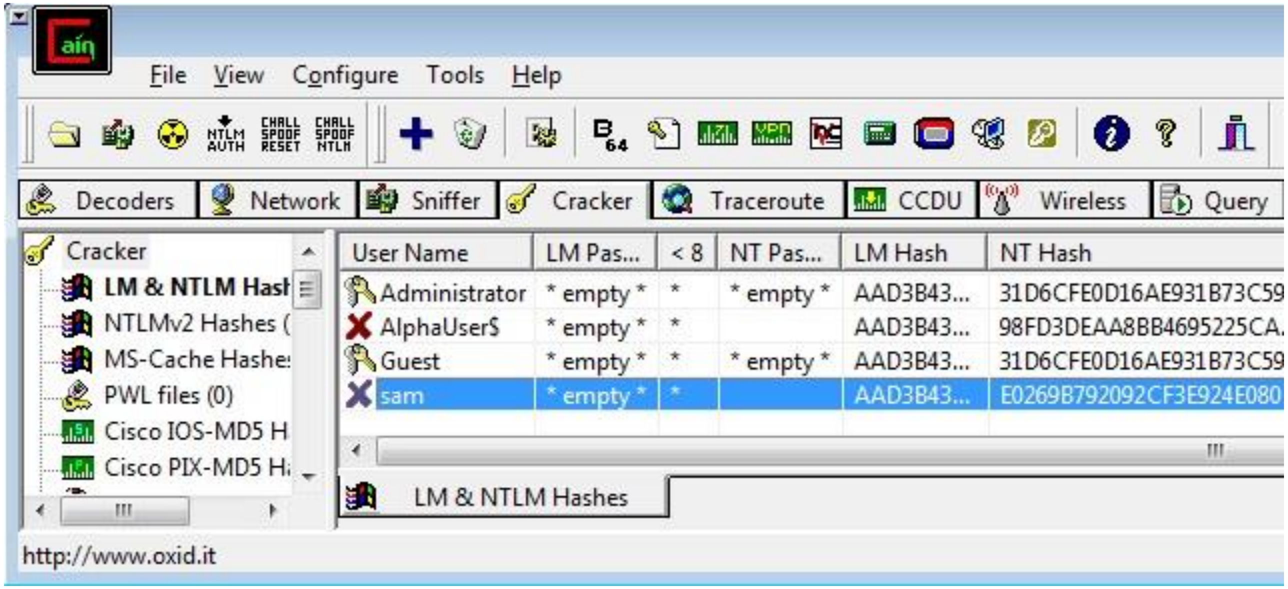
Table 7-4 Common password myths

Attacks on Passwords

- **Brute force attack**
  - Simply trying to guess a password through combining a random combination of characters
- Passwords typically are stored in an encrypted form called a “hash”
  - Attackers try to steal the file of hashed passwords and then break the hashed passwords offline

How to Get the Hashes

- Easy way: Just use Cain
- Cracker tab, right-click, "Add to List"



User Name	LM Pas...	< 8	NT Pas...	LM Hash	NT Hash
Administrator	* empty *	*	* empty *	AAD3B43...	31D6CFE0D16AE931B73C59
AlphaUser$	* empty *	*		AAD3B43...	98FD3DEAA8BB4695225CA.
Guest	* empty *	*	* empty *	AAD3B43...	31D6CFE0D16AE931B73C59
sam	* empty *	*		AAD3B43...	E0269B792092CF3E924E080

Attacks on Passwords

- **Dictionary attack**
  - Guess passwords from a dictionary
  - Works if the password is a known common password
- **Rainbow tables**
  - Make password attacks faster by creating a large pregenerated data set of hashes from nearly every possible password combination
  - Works well against Windows passwords because Microsoft doesn't use the **salting** technique when computing hashes

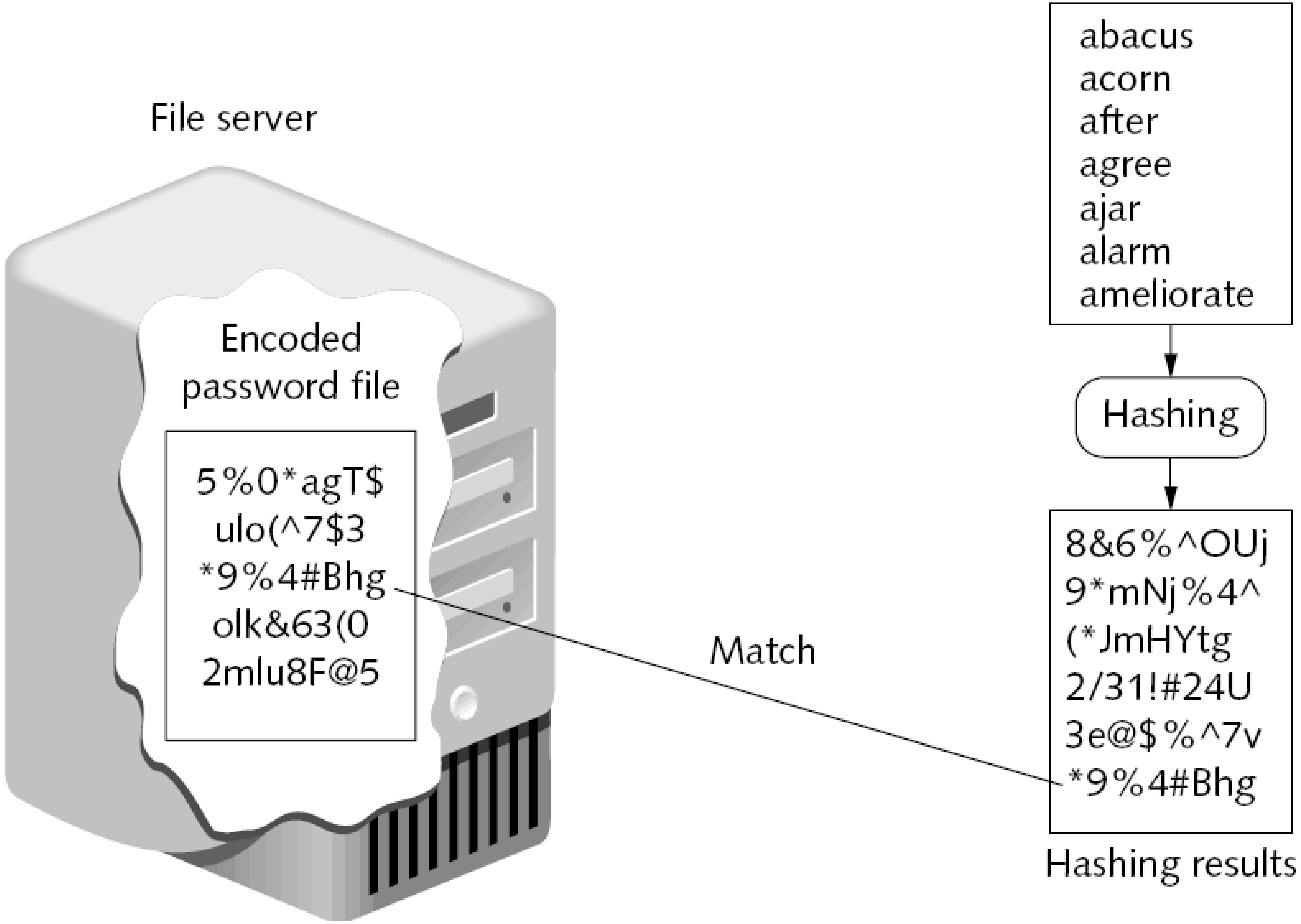
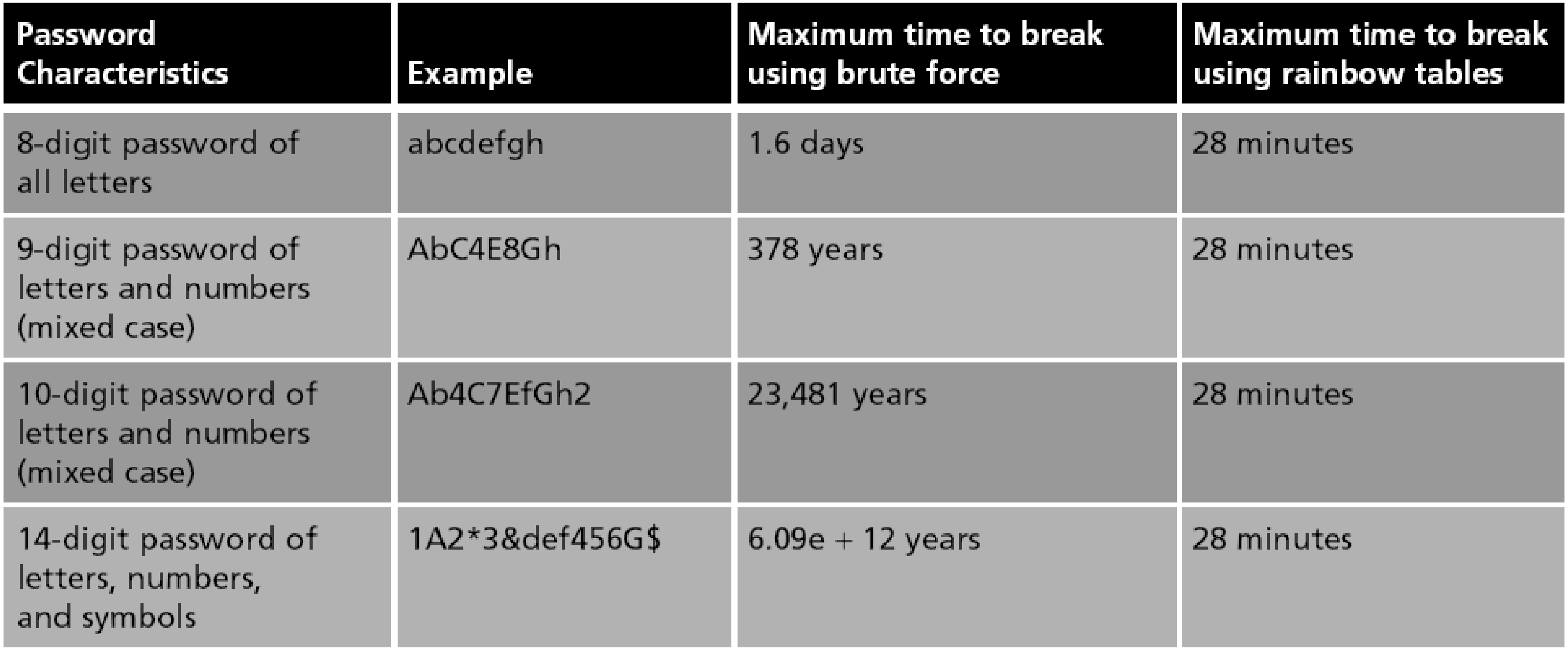


Figure 7-7 Dictionary attack

Rainbow Tables

- Generating a rainbow table requires a significant amount of time
- Rainbow table advantages
  - Can be used repeatedly for attacks on other passwords
  - Rainbow tables are much faster than dictionary attacks
  - The amount of time needed on the attacking machine is greatly reduced

Rainbow Table Attack



Password Characteristics	Example	Maximum time to break using brute force	Maximum time to break using rainbow tables
8-digit password of all letters	abcdefgh	1.6 days	28 minutes
9-digit password of letters and numbers (mixed case)	AbC4E8Gh	378 years	28 minutes
10-digit password of letters and numbers (mixed case)	Ab4C7EfGh2	23,481 years	28 minutes
14-digit password of letters, numbers, and symbols	1A2*3&def456G$	6.09e + 12 years	28 minutes

Table 7-5 Times to break a hash

Passwords (continued)

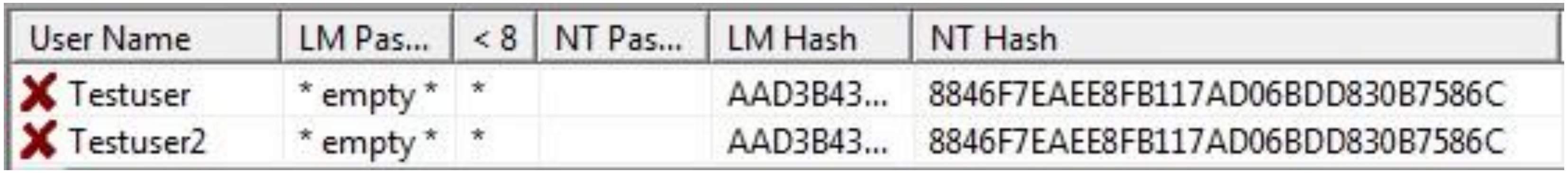
- One reason for the success of rainbow tables is how older Microsoft Windows operating systems hash passwords
- A defense against breaking encrypted passwords with rainbow tables
  - Hashing algorithm should include a random sequence of bits as input along with the user-created password
- These random bits are known as a **salt**
  - Make brute force, dictionary, and rainbow table attacks much more difficult

No Salt!

- To make hashing stronger, add a random "Salt" to a password before hashing it
- Two accounts with the same password hash to the same result
- This makes it possible to speed up password cracking with precomputed Rainbow Tables

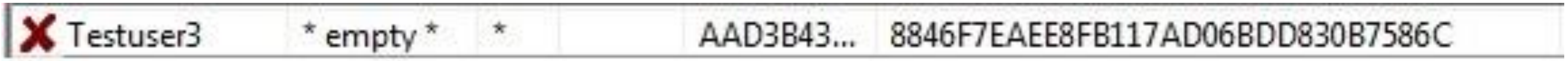
Demonstration

- Here are two accounts on a Windows 7 Beta machine with the password 'password'



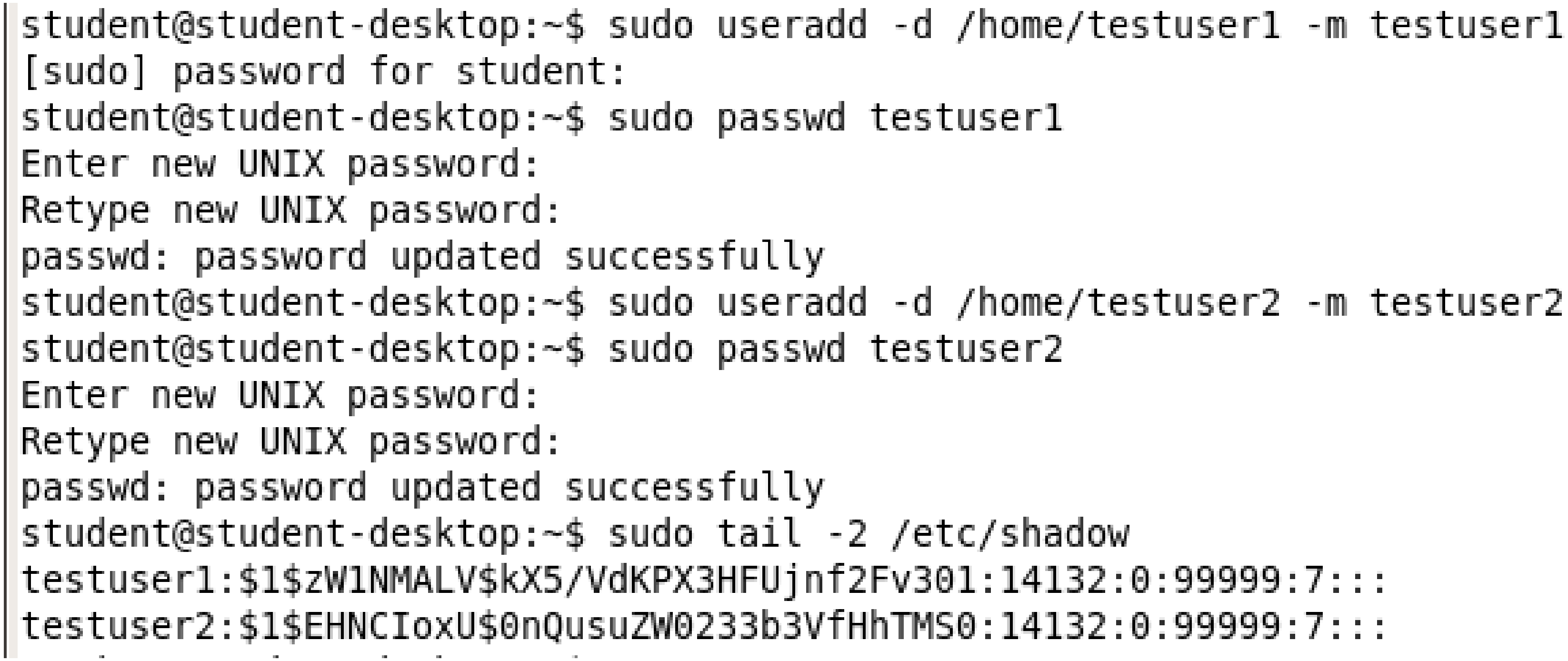
User Name	LM Pas...	< 8	NT Pas...	LM Hash	NT Hash
✗ Testuser	* empty *	*		AAD3B43...	8846F7EAEE8FB117AD06BDD830B7586C
✗ Testuser2	* empty *	*		AAD3B43...	8846F7EAEE8FB117AD06BDD830B7586C

- This hash is from a different Windows 7 Beta machine



✗ Testuser3	* empty *	*		AAD3B43...	8846F7EAEE8FB117AD06BDD830B7586C

Linux Salts its Hashes



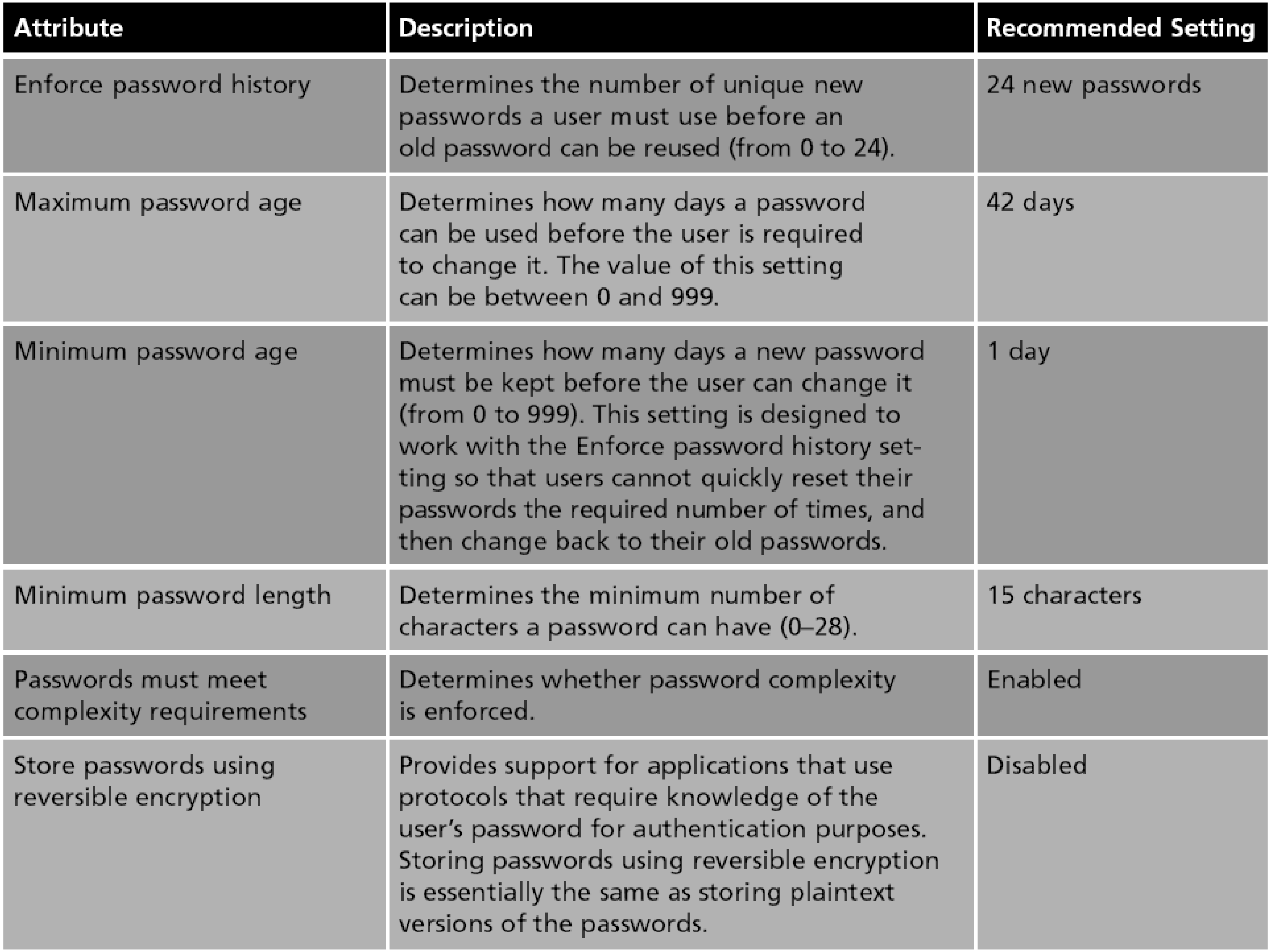
```
student@student-desktop:~$ sudo useradd -d /home/testuser1 -m testuser1
[sudo] password for student:
student@student-desktop:~$ sudo passwd testuser1
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
student@student-desktop:~$ sudo useradd -d /home/testuser2 -m testuser2
student@student-desktop:~$ sudo passwd testuser2
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
student@student-desktop:~$ sudo tail -2 /etc/shadow
testuser1:$1$zW1NMALV$kX5/VdKPX3HFUjnf2Fv301:14132:0:99999:7:::
testuser2:$1$EHNCIoxU$0nQusuZW0233b3VfHhTMS0:14132:0:99999:7:::
```

Password Policy

- A strong password policy can provide several defenses against password attacks
- The first password policy is to create and use strong passwords
- One of the best defenses against rainbow tables is to prevent the attacker from capturing the password hashes
- A final defense is to use another program to help keep track of passwords

Domain Password Policy

- Setting password restrictions for a Windows domain can be accomplished through the Windows Domain password policy
- There are six common domain password policy settings, called password setting objects
  - Used to build a domain password policy



Attribute	Description	Recommended Setting
Enforce password history	Determines the number of unique new passwords a user must use before an old password can be reused (from 0 to 24).	24 new passwords
Maximum password age	Determines how many days a password can be used before the user is required to change it. The value of this setting can be between 0 and 999.	42 days
Minimum password age	Determines how many days a new password must be kept before the user can change it (from 0 to 999). This setting is designed to work with the Enforce password history setting so that users cannot quickly reset their passwords the required number of times, and then change back to their old passwords.	1 day
Minimum password length	Determines the minimum number of characters a password can have (0–28).	15 characters
Passwords must meet complexity requirements	Determines whether password complexity is enforced.	Enabled
Store passwords using reversible encryption	Provides support for applications that use protocols that require knowledge of the user's password for authentication purposes. Storing passwords using reversible encryption is essentially the same as storing plaintext versions of the passwords.	Disabled

Table 7-6 Password objects

Passwords vs. Keys

Bad idea to use password bytes directly as keys for AES/DES/HMAC etc. Why?

- Passwords are usually English words
  - Have predictable statistical properties
  - Not uniformly distributed bytes
- Dictionary attacks become easy
- Rainbow table-like attacks become possible

Solution: Key Derivation Functions

- Examples: PBKDF2, Scrypt, Argon2
- Key feature:
  - Takes a long time to generate hash from password
  - Why is this important?
  - Makes brute force attacks slow (hence infeasible)
- Have a configurable time to generate hash
  - Why this is useful?
  - Can increase time when HW becomes faster

Secure Architecture Principles

Operating Systems

(Optional Material)

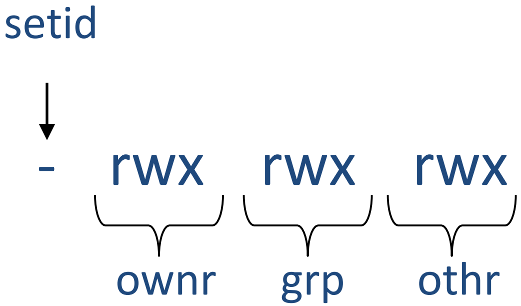
Unix access control

- 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

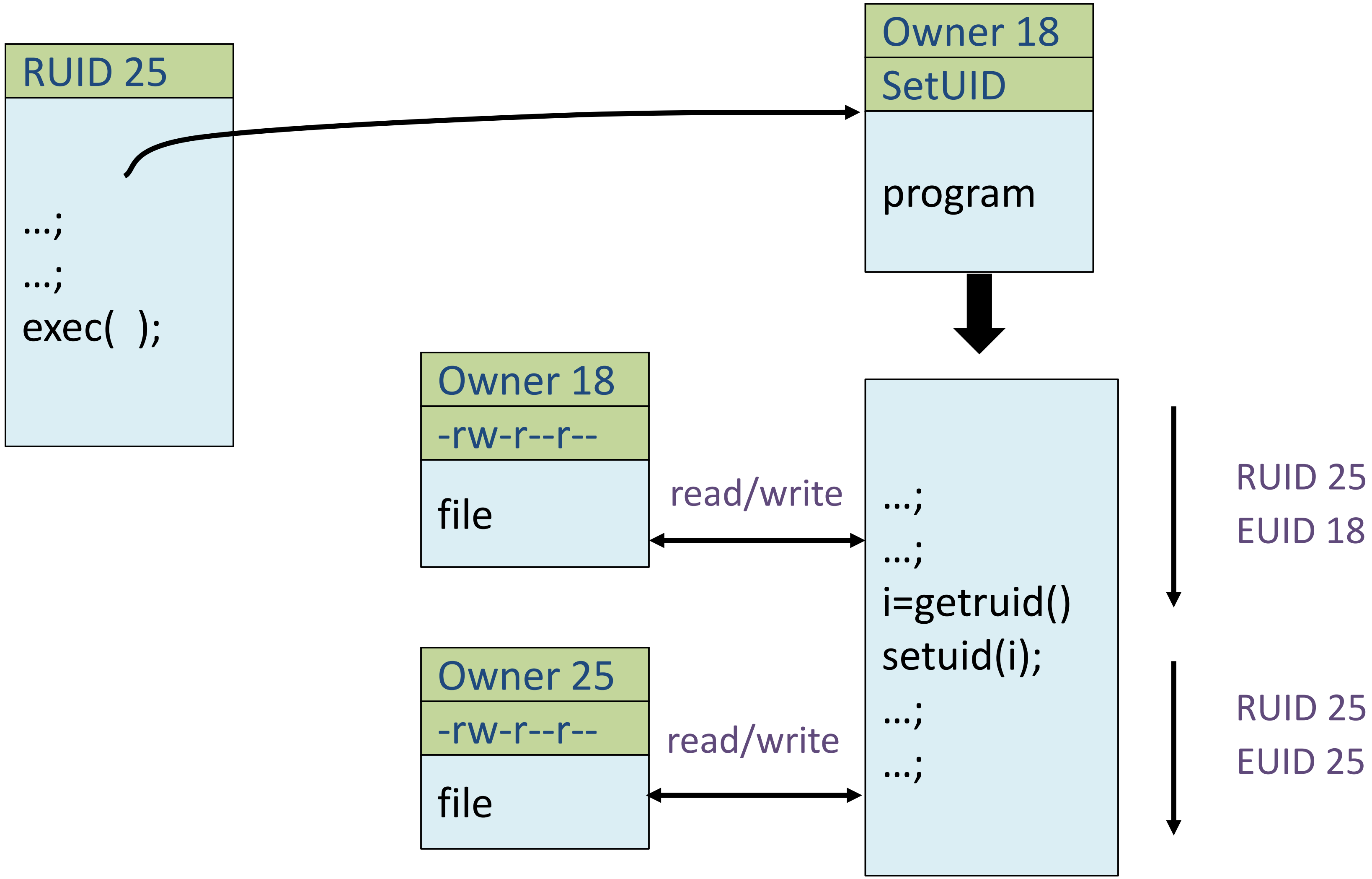
Process Operations and IDs

- 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

```
…;
…;
i=getruid()
setuid(i);
…;
…;
```

RUID 25

EUID 18

Owner 25

-rw-r--r--

file

read/write

RUID 25

EUID 25

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

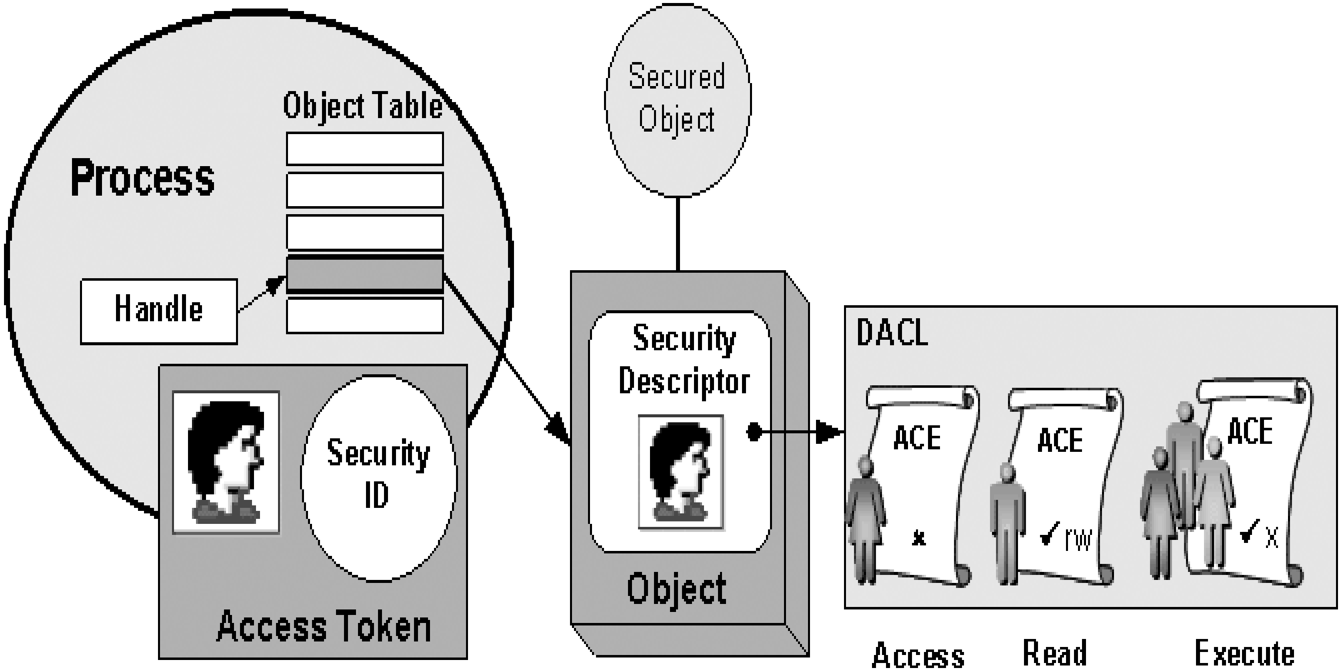
- Network-facing Daemons
  - Root processes with network ports open to all remote parties, e.g., sshd, ftpd, sendmail, …
- Rootkits
  - System extension via dynamically loaded kernel modules
- Environment Variables
  - System variables such as LIBPATH that are shared state across applications. An attacker can change LIBPATH 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.

Access control in Windows

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
- Some additional concepts
  - Tokens
  - Security attributes
- Generally
  - More flexible than Unix
    - Can define new permissions
    - Can give some but not all administrator privileges



Assigned when user logs on

Assigned when object is created

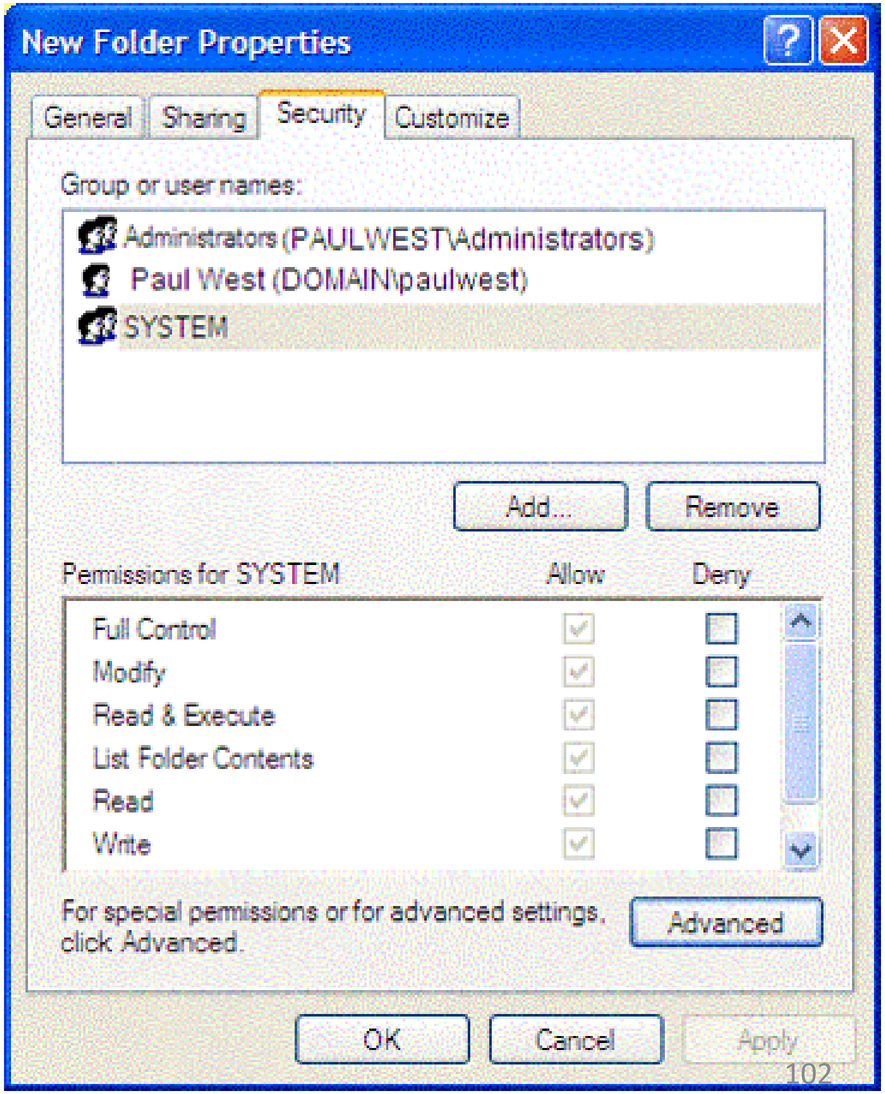
Access Denied

Read Write Access Rights

Execute Access Rights

Identify subject using SID

- Security ID (SID)
  - Identity (replaces UID)
    - SID revision number
    - 48-bit authority value
    - variable number of Relative Identifiers (RIDs), for uniqueness
  - Users, groups, computers, domains, domain members all have SIDs



Group or user names:

Administrators (PAULWEST\Administrators)
Paul West (DOMAIN\paulwest)
SYSTEM

Permissions for SYSTEM	Allow	Deny
Full Control	☑	□
Modify	☑	□
Read & Execute	☑	□
List Folder Contents	☑	□
Read	☑	□
Write	☑	□

For special permissions or for advanced settings, click Advanced.

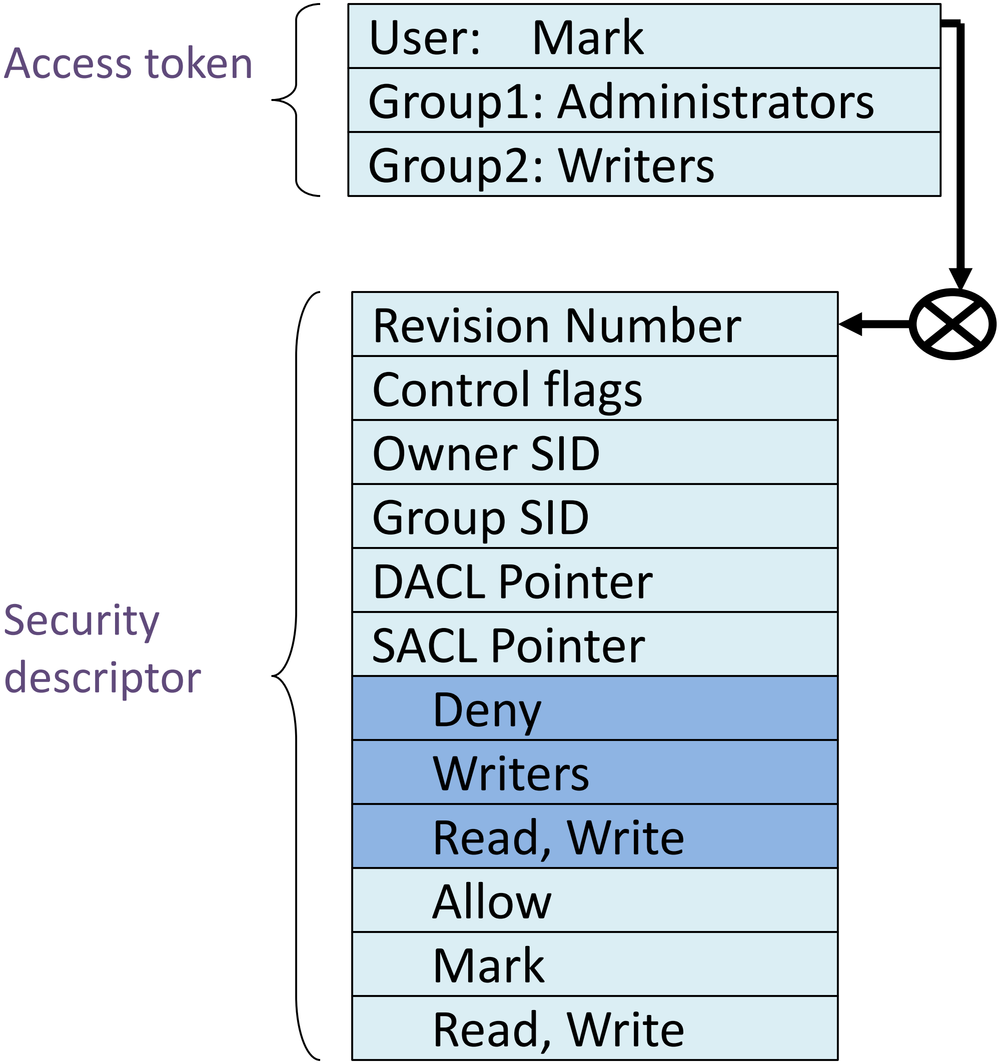
Process has set of tokens

- Security context
  - Privileges, accounts, and groups associated with the process or thread
  - Presented as set of tokens
- Impersonation token
  - Used temporarily to adopt a different security context, usually of another user
- Security Reference Monitor
  - Uses tokens to identify the security context of a process or thread

Object has security descriptor

- Security descriptor associated with an object
  - Specifies who can perform what actions on the object
- Several fields
  - Header
    - Descriptor revision number
    - Control flags, attributes of the descriptor
      - E.g., memory layout of the descriptor
  - SID of the object's owner
  - SID of the primary group of the object
  - Two attached optional lists:
    - Discretionary Access Control List (DACL) – users, groups, …
    - System Access Control List (SACL) – system logs, ..

Example access request



Access token

User: Mark

Group1: Administrators

Group2: Writers

Security descriptor

Revision Number

Control flags

Owner SID

Group SID

DACL Pointer

SACL Pointer

Deny

Writers

Read, Write

Allow

Mark

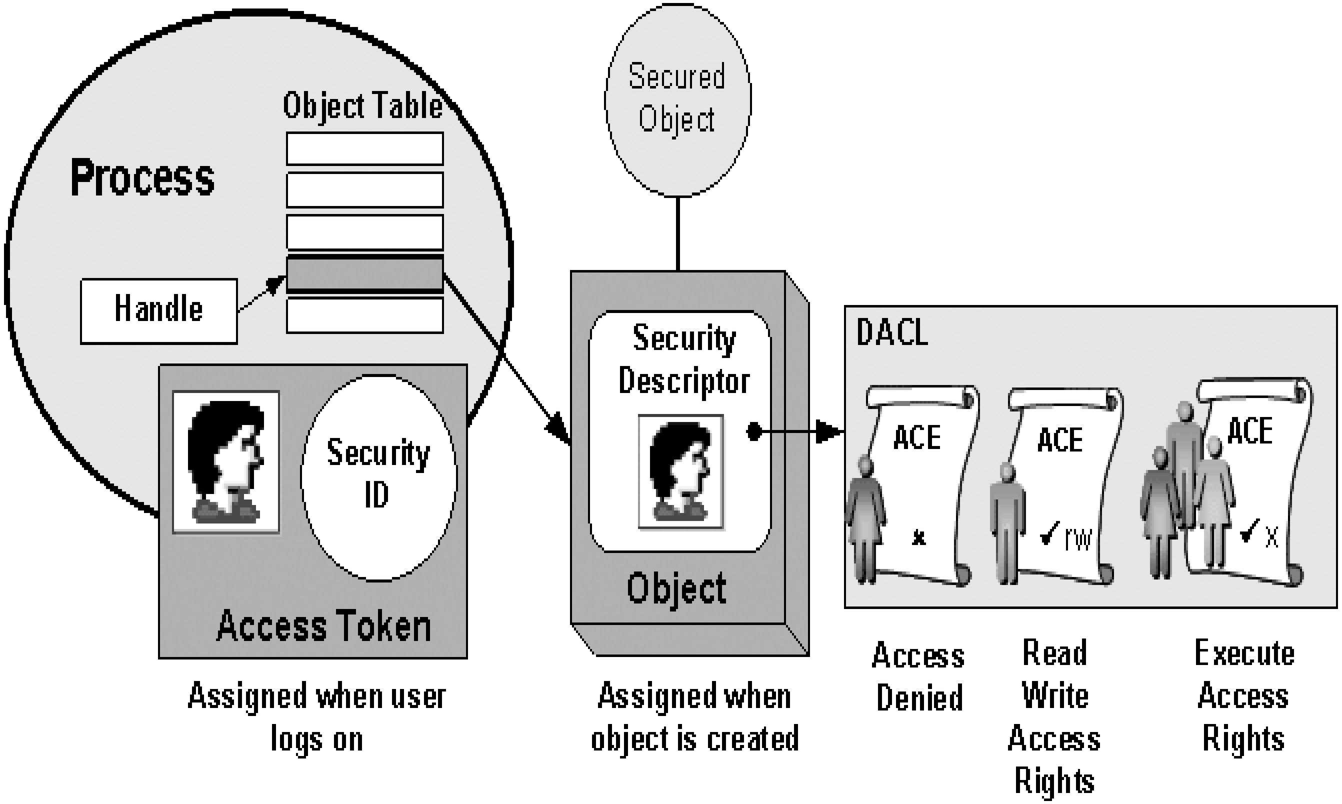
Read, Write

Access request: write
Action: denied

- User Mark requests write permission
- Descriptor denies permission to group
- Reference Monitor denies request

(DACL for access, SACL for audit and logging)

Priority:
Explicit Deny
Explicit Allow
Inherited Deny
Inherited Allow


Secured Object
Object Table
Process
Handle
Security ID
Access Token
Assigned when user logs on
Security Descriptor
Object
Assigned when object is created
DACL
ACE
x
ACE
✓ rw
ACE
✓ x
Access Denied
Read Write Access Rights
Execute Access Rights

Impersonation Tokens (compare to setuid)

- Process adopts security attributes of another
  - Client passes impersonation token to server
- Client specifies impersonation level of server
  - Anonymous
    - Token has no information about the client
  - Identification
    - Obtain the SIDs of client and client's privileges, but server cannot impersonate the client
  - Impersonation
    - Impersonate the client
  - Delegation
    - Lets server impersonate client on local, remote systems

Weakness in isolation, privileges

- Similar problems to Unix
  - E.g., Rootkits leveraging dynamically loaded kernel modules
- Windows Registry
  - Global hierarchical database to store data for all programs
  - Registry entry can be associated with a security context that limits access; common to be able to write sensitive entry
- Enabled By Default
  - Historically, many Windows deployments also came with full permissions and functionality enabled

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
  - …

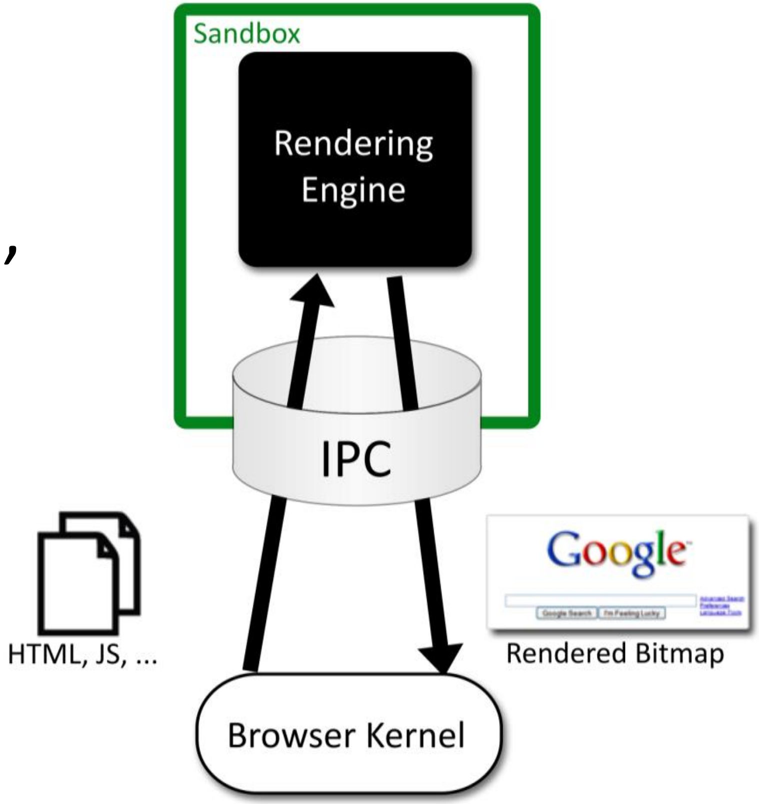
The web browser enforces its own internal policy. If the browser implementation is corrupted, this mechanism becomes unreliable.

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?

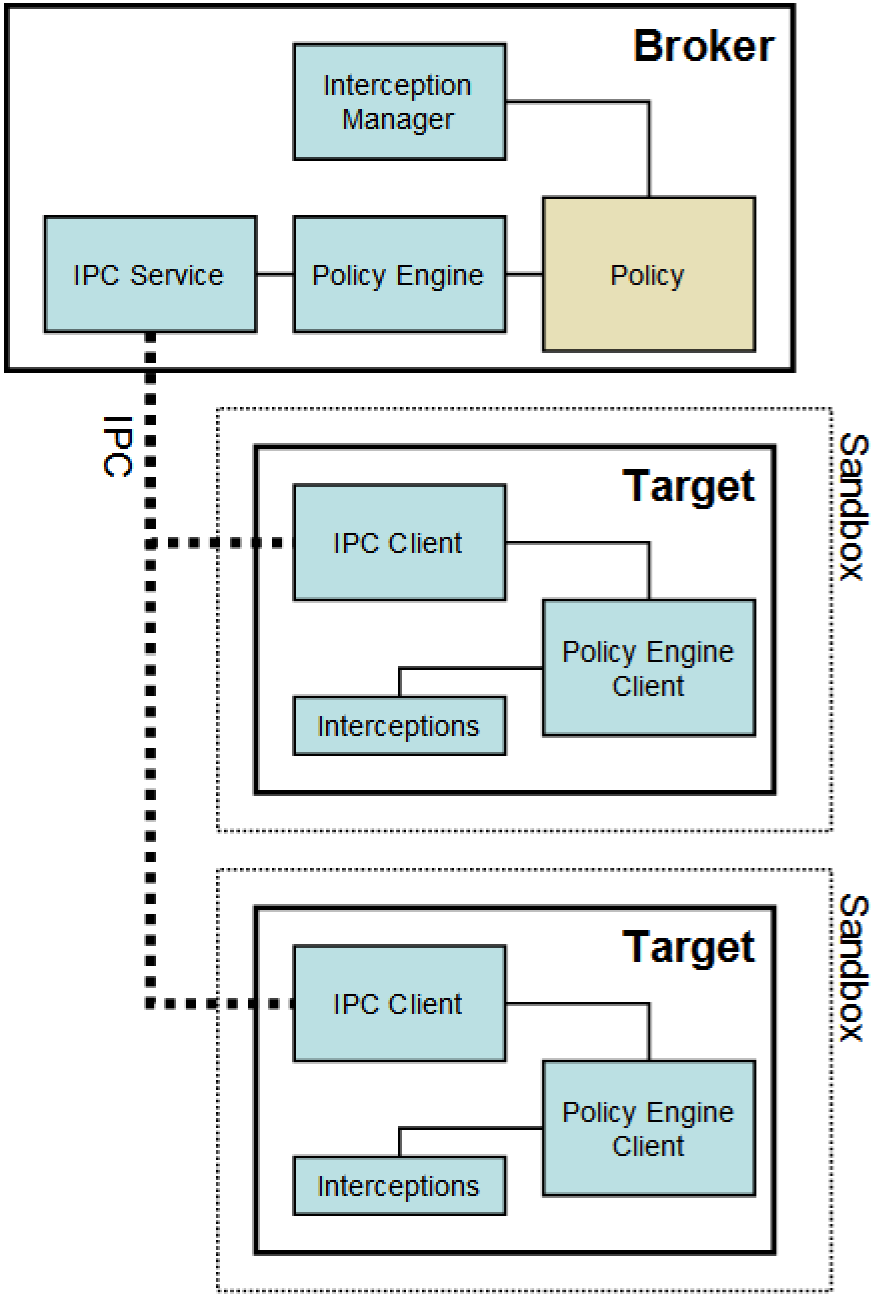
Chromium Security Architecture

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



Chromium

Communicating sandboxed components

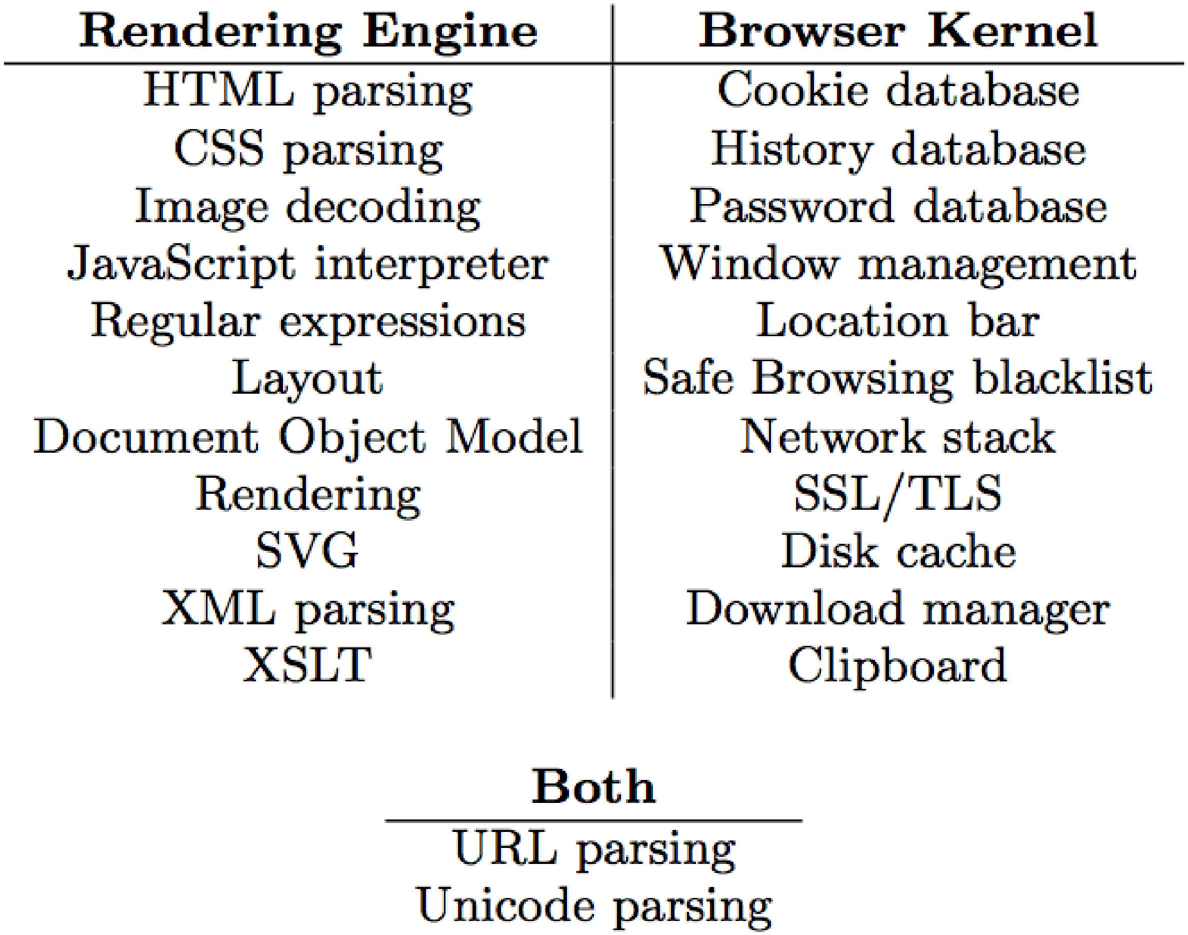


See: http://dev.chromium.org/developers/design-documents/sandbox/

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

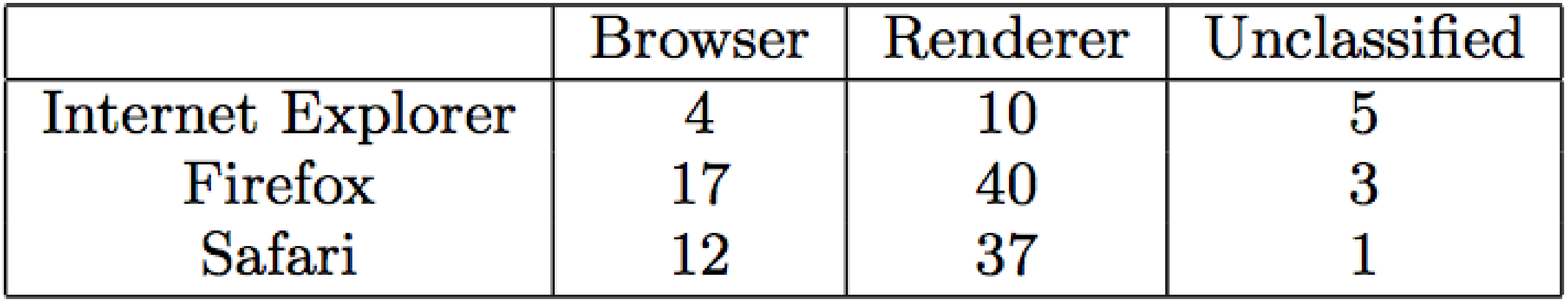
Leverage OS Isolation

- 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/

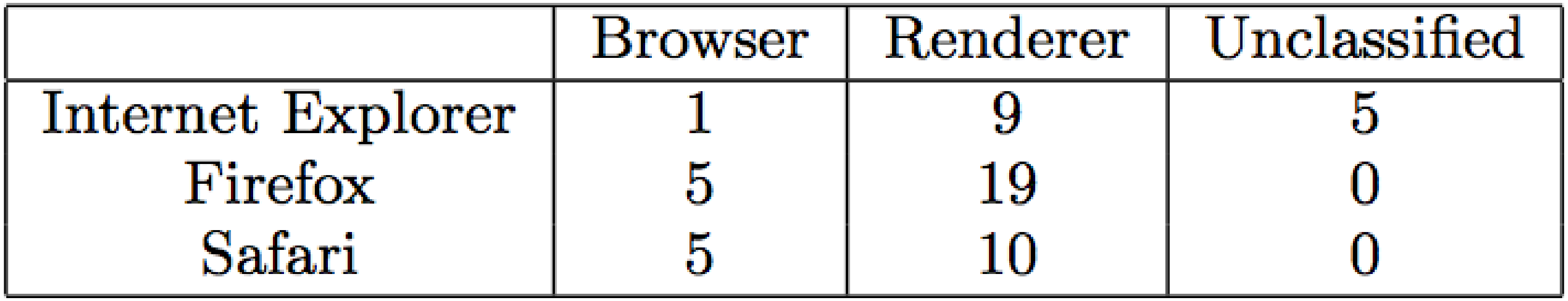
Evaluation: CVE count

- 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