

# **Security Principles and Mechanisms**

**CS155 Computer and Network Security**

**Stanford University**

# Least Privilege

Every program and user should operate using the least privileges necessary to complete their job.

“This principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Thus, if a question arises related to misuse of a privilege, the number of programs that must be audited is minimized. Put another way, if a mechanism can provide "firewalls," the principle of least privilege provides a rationale for where to install the firewalls. The military security rule of "need-to-know" is an example of this principle.”

# Security Models & Access Control

# System Security

How do we protect confidentiality, integrity, and availability of system components from attackers?

**Security Model:** System abstraction that enables us to discuss and formulate a policy.

**Security Policy:** Set of allowed actions. Who is allowed to do what?

**Security Mechanism:** Implementation of the policy, e.g., access control or encryption

# Security Model

**Subjects (Who):** acting system principals

UNIX: Users and Processes

Android: Apps

Web: Domain

**Objects (What):** protected sources

UNIX: memory, files, hardware devices, syscalls

Other: database tables/rows, cookies, DOM components, phone camera

**Access Operations:** How can subjects operate on (e.g., read) objects

# Security Policy

What actions is a subject allowed to perform on an object?

## **Examples:**

Alice and Bob can read the files in directory X but not create new files in the directory

John can create new users

# Access Control Matrix [Lampson]

	Objects
Subjects	
	<b>{ allowed actions }</b>

# Access Control Matrix [Lampson]

	File 1	File 2	File 3	File 4
Alice	read	read/write	no access	no access
Bob	read	read/write	no access	no access
Carol	read	write	read/write	read/write
Wendy	read/write	read/write	read	read



# Security Mechanisms

There are two *security mechanisms* for enforcing *security policies*:

**Access Control Lists (ACLs):** Every object has a list of who can access

**Capabilities:** User has an unforgeable ticket that grants access

# Access Control Lists (ACLs)

Object-centric approach.

Every object has an ACL that identifies what operations subjects can perform.

Each access to object is checked against object's ACL.

**Example: guest list**

File 1	
Alice	read
Bob	read
Carol	write
Wendy	read/write

# Capabilities

User-centric approach.

A capability grants a subject permission to perform a certain action.

Subject provides capability to *reference monitor* to check before allowing operation.

**Example: movie ticket.**



# ACL vs Capability

**What are the pros and cons?**

Delegation?

Revocation?

Audibility?

# Role Based Access Control

Access control matrices can grow complex as number of subjects, objects, and possible operations grow.

**Observation:** Users change more often than roles

	hr/	eng/	admin/	all/
marketing	—	—	—	read
executive	read	read	read/write	read/write
hr	read/write	—	—	read
engineering	—	read/write	—	read

# Attribute-Based Access Control

RBAC is one type of *Attribute-Based Access Control*

**Idea:** assign attributes (e.g., tags) to subjects and/or objects.

Access control matrix defines which attributes subjects need to have in order to access objects with given attributes

# UNIX Security Model

# UNIX Security Model

**Subjects (Who?)**

**Objects (What?)**

**Access Operations**



# UNIX Security Model

## **Subjects (Who?)**

- Users

## **Objects (What?)**

## **Access Operations**

# UNIX Security Model

## **Subjects (Who?)**

- Users

## **Objects (What?)**

- Files, directories
- Files: sockets, pipes, hardware devices, kernel objects, process data

## **Access Operations**

# UNIX Security Model

## **Subjects (Who?)**

- Users

## **Objects (What?)**

- Files, directories
- Files: sockets, pipes, hardware devices, kernel objects, process data

## **Access Operations**

- Read, Write, Execute

# Unix Groups

A user may belong to several groups

- Used to provide simple role-based access control

Every user belongs to

- primary group (defined in `/etc/passwd`)
- optional additional groups (defined in `/etc/group`)

```
zakhir@scratch-01:~$ groups
zakhir sudo lxd esrg esrg_
```

# Unix File System Security

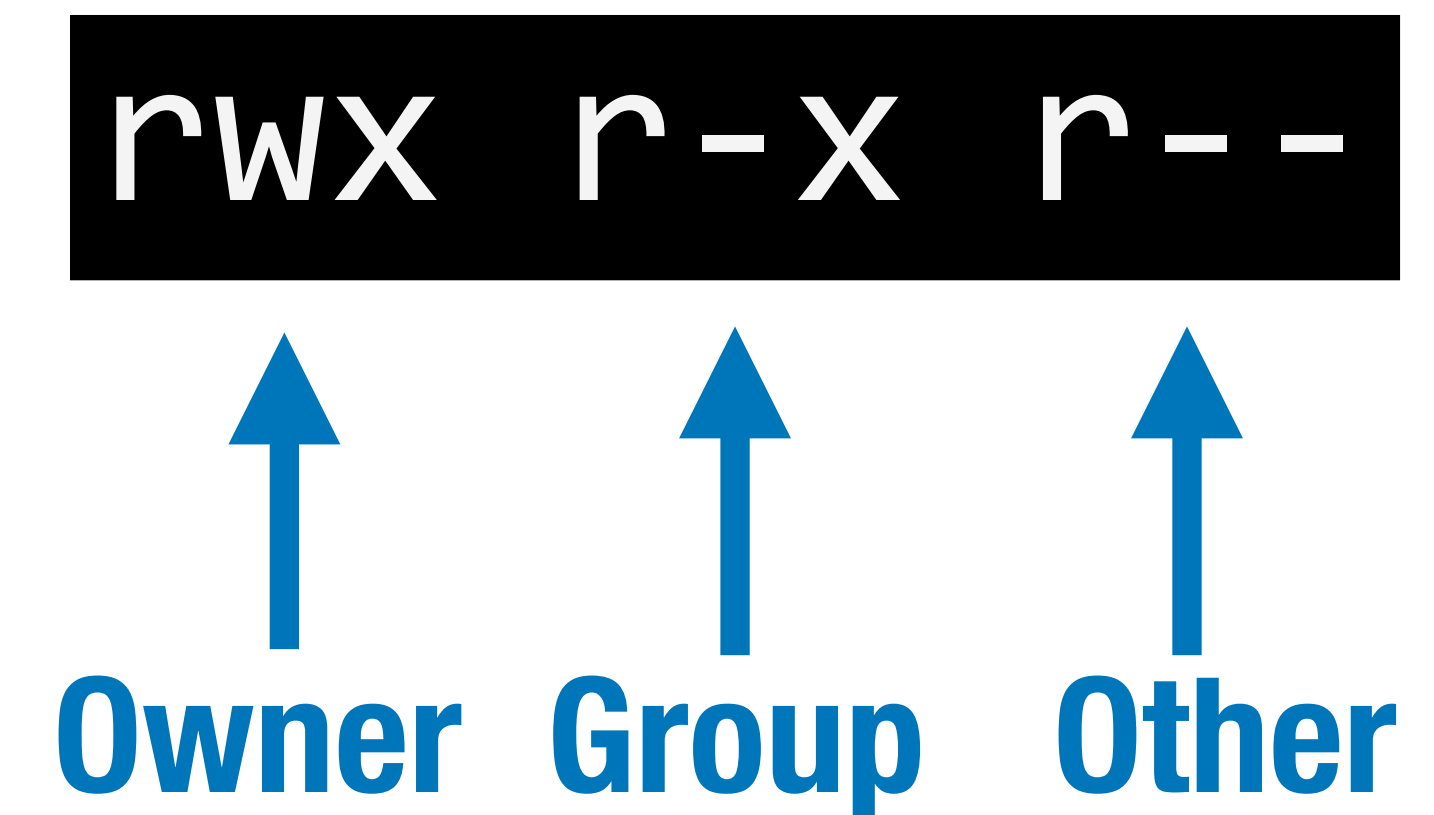
Every file and directory has an *owner* and *group* and simple ACL

File permissions specify what role can do what

- **Three Roles:** owner, group, other
- **Three Operations:** read, write, execute

Permissions are set by owner (or root)

- Cannot be delegated



# Sticky Bit

Can a user rename a file in a directory if they have access to the directory but not the file?

# Sticky Bit

Can a user rename a file in a directory if they have access to the directory but not the file?

## **Sticky Bit**

Off: if user has write permission on directory, can rename or remove files, even if not owner

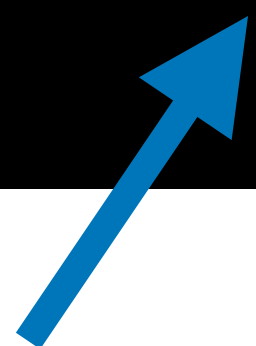
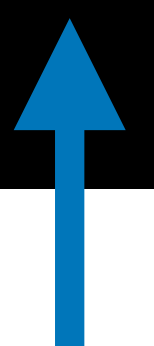

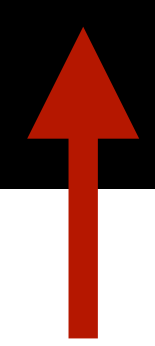
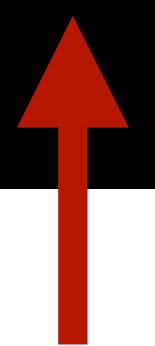


On: only file owner, directory owner, and root can rename or remove file in the directory

# Example Directory Listing

```
zakir@scratch-01:~$ ls -l
```

```
total 8
```

```
d rwx rwx --- 5 zakir zakir 4096 Apr  2 15:56 go
- rw- rw- r-- 1 zakir zakir    0 Apr 11 04:15 test.py
d rwx rwx --- 11 zakir esrg 4096 Dec 28 21:09 zmap
```

 **Owner**  **Group**  **Other**  **Owner**  **Group**  **Size**  **Modified**

**Access Operations**



# Unix Processes

Every process has three user IDs

## 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)

- Determines the permissions for process
- (Usually) Inherited from parent process
- Root user can change process id

## Saved user ID (SUID)

- Prior EUID that can be restored

# Superuser / Root

Superuser allowed to do anything that is possible

Called root and mapped to user id 0

Think about superuser as a role rather than a particular user

System administrators assume superuser role to perform privileged actions

- Good practice to assume superuser role only when necessary

# login and sshd

Login and SSHD run as root

- Authenticate the user using username/password, public key
- Changes its user id and group id to that of user
- Executes users' shell

**Critical:** dropping privileges from root to regular user

# Elevating Privilege

Typically, executables are run with user ID and group ID of the user that executed them

Executable files have a **setuid** bit

- If set, file is executed with the effective id (and associated privileges) of the file owner (i.e., EUID is file owner, RUID is executer)

The passwd command is owned by root and has setuid bit set.

# Linux Capabilities

Traditional UNIX distinguished between privileged processes (EUID == 0) and unprivileged processes (EUID != 0)

Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking

Capabilities divide the power of superuser into pieces, such that if a program that has one or more capabilities is compromised, its power to do damage to the system would be less than the same program running with root privilege.

# Capability Examples

## **CAP\_KILL**

Bypass permission checks for sending signals

## **CAP\_NET\_BIND\_SERVICE**

Bind a socket to privileged ports (port < 1024).

## **CAP\_SYS\_MODULE**

Load and unload kernel modules

## **CAP\_SYS\_PTRACE**

Trace arbitrary processes using ptrace(2)

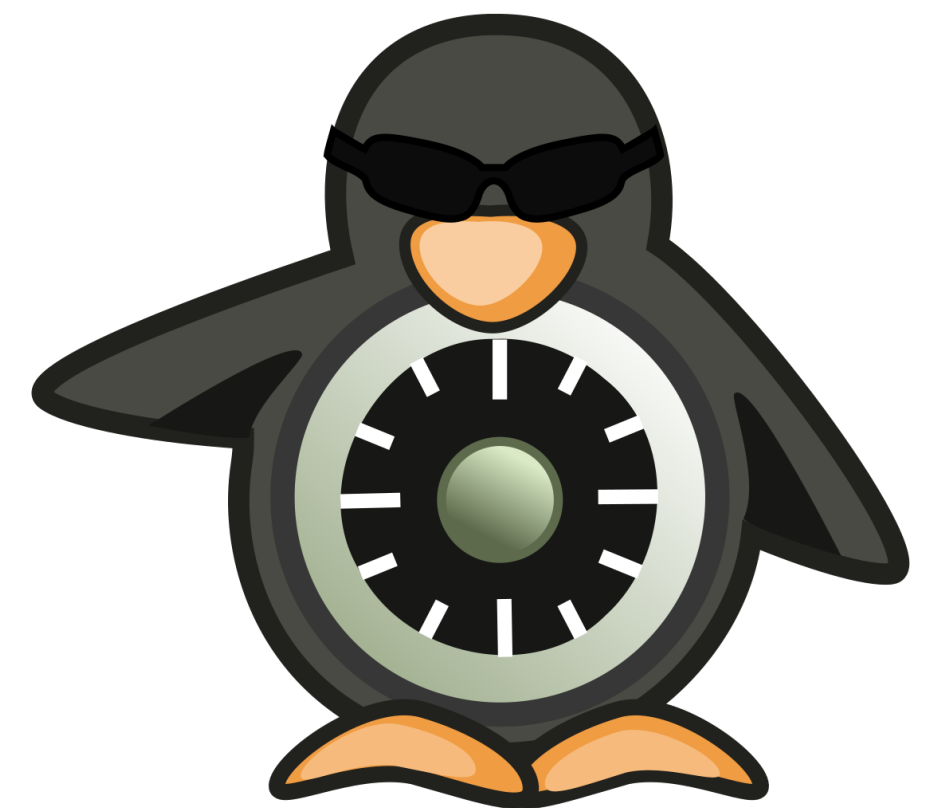
# DAC vs MAC

## **Discretionary Access Control (DAC)**

- Linux is an example of DAC.
- Resource owners can set the security policy for objects they own

## **Mandatory Access Control (MAC)**

- Centralized authority sets security policy for all resources
- Example: SELinux



# Windows Security Model

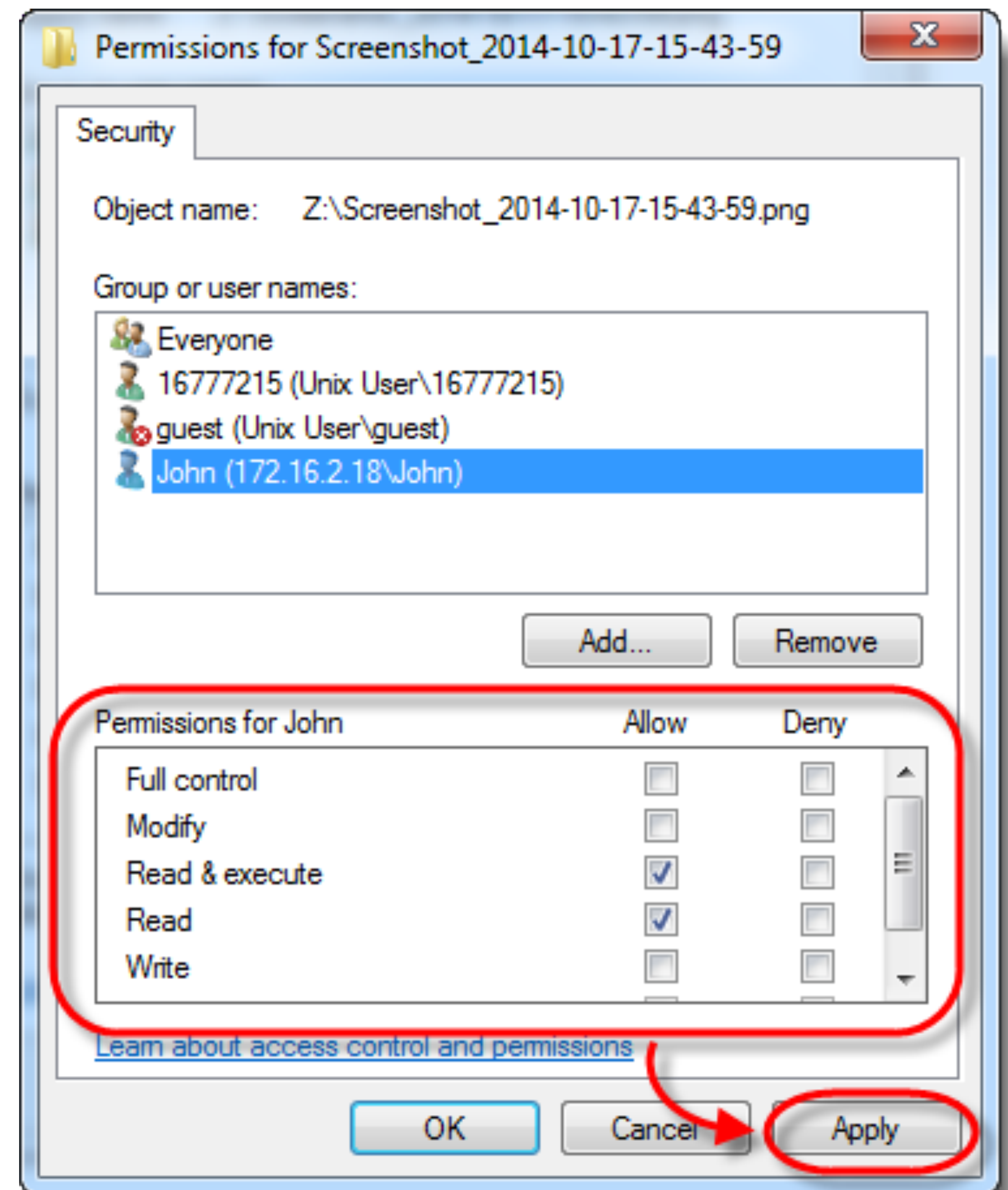


Windows has more complex access control

Objects have full ACLs — possibility for fine grained permissions

Users can be member of multiple groups, groups can be nested

ACLs support Allow and Deny rules



# Subjects and Access Tokens

Every process has an access token that describe its security context

- ID of user account
- ID of groups
- ID of login session
- List of OS privileges held by either the user or the user's group
- List of restrictions

# Windows Objects

**Every object has a security descriptor**

- Specifies who can perform what and audit rules

**Contains**

- Security identifiers (SIDs) for the owner and primary group of an object.
- Discretionary ACL (DACL): access rights allowed users or groups.
- System ACL (SACL): types of attempts that generate audit records

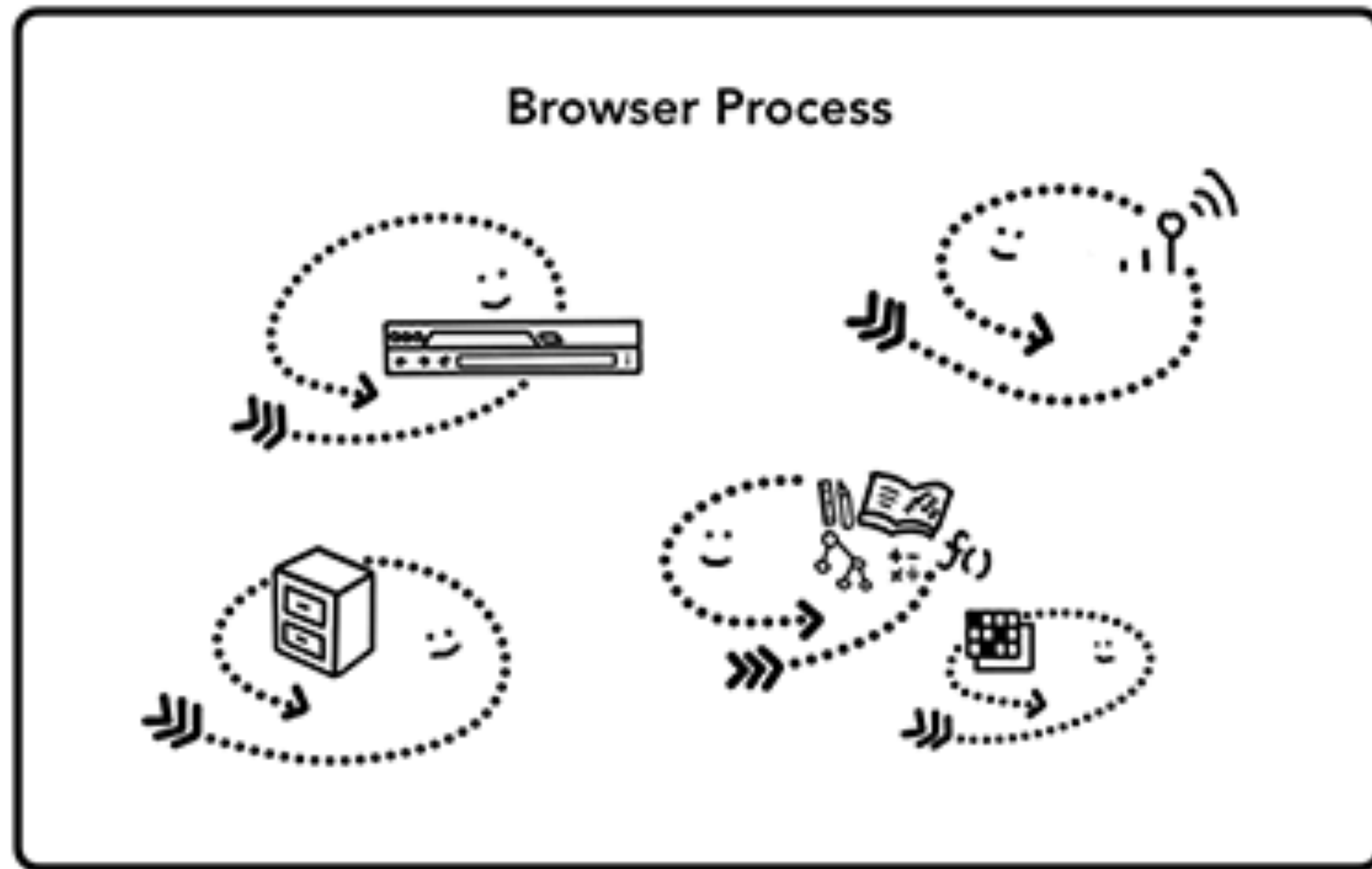
# Privilege Separation

**Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.**

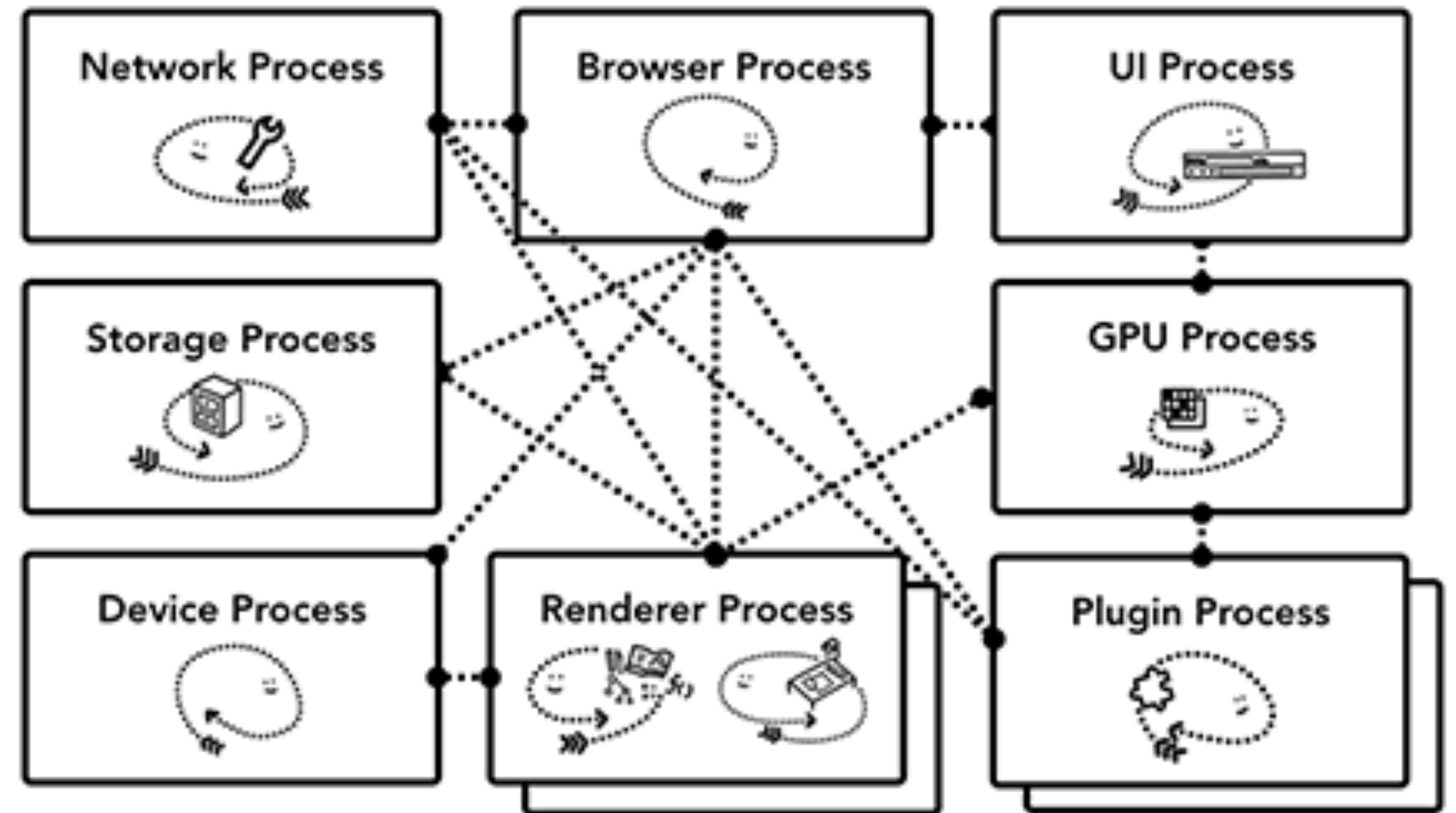
“[...] Once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information. [...] This principle is at work in the defense system that fires a nuclear weapon only if two different people both give the correct command. In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted. For example, systems providing user-extendible protected data types usually depend on separation of privilege for their implementation.”

# Chrome Security Architecture

# Browsers Pre-2006 vs Now



Old



New



# Chrome Processes

## Browser Process

Controls "chrome" part of the application like address bar and, bookmarks. Also handles the invisible, privileged parts of a web browser like network requests.

## Renderer Process

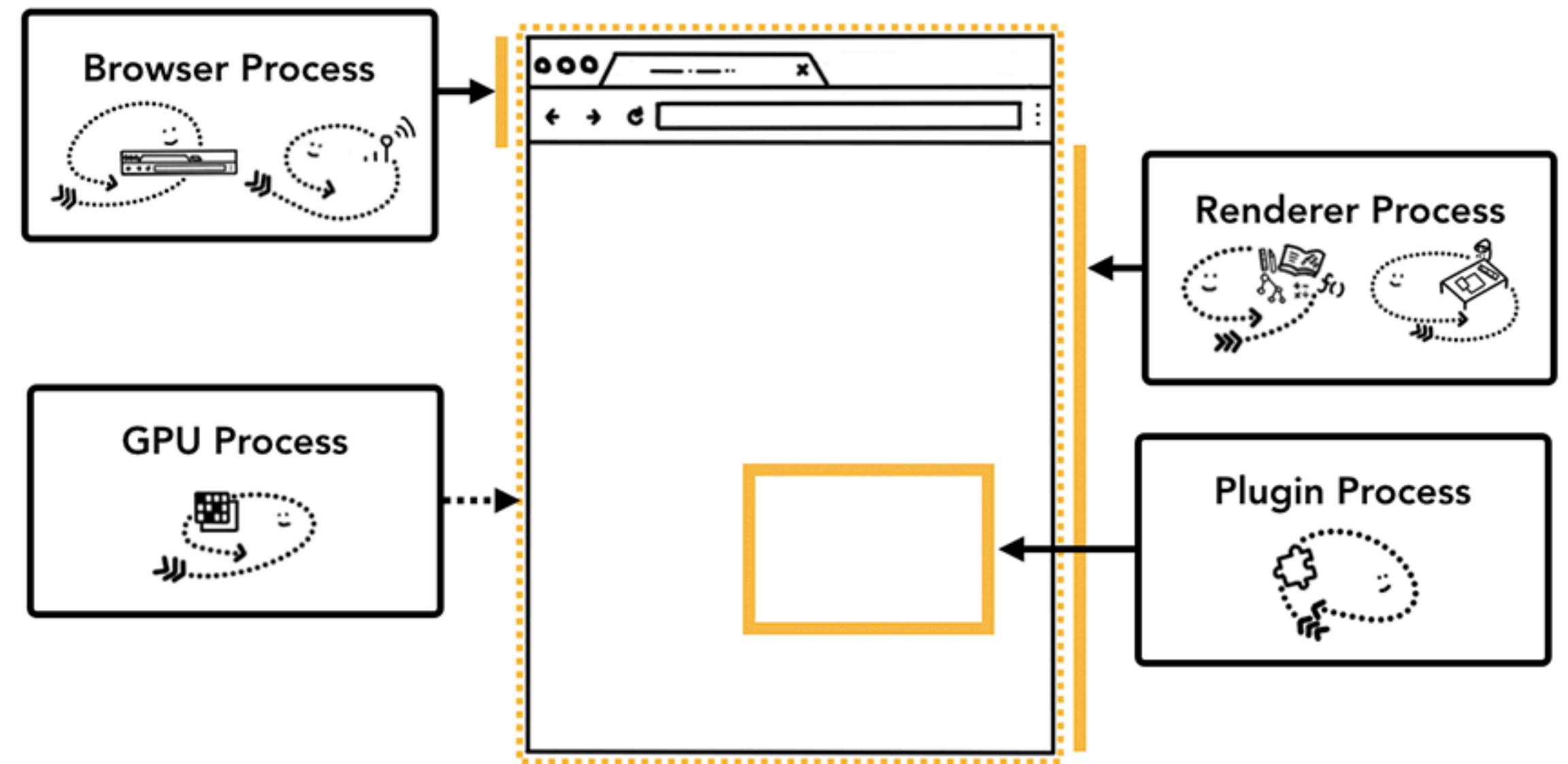
Controls anything inside of the tab where a website is displayed.

## Plugin Process

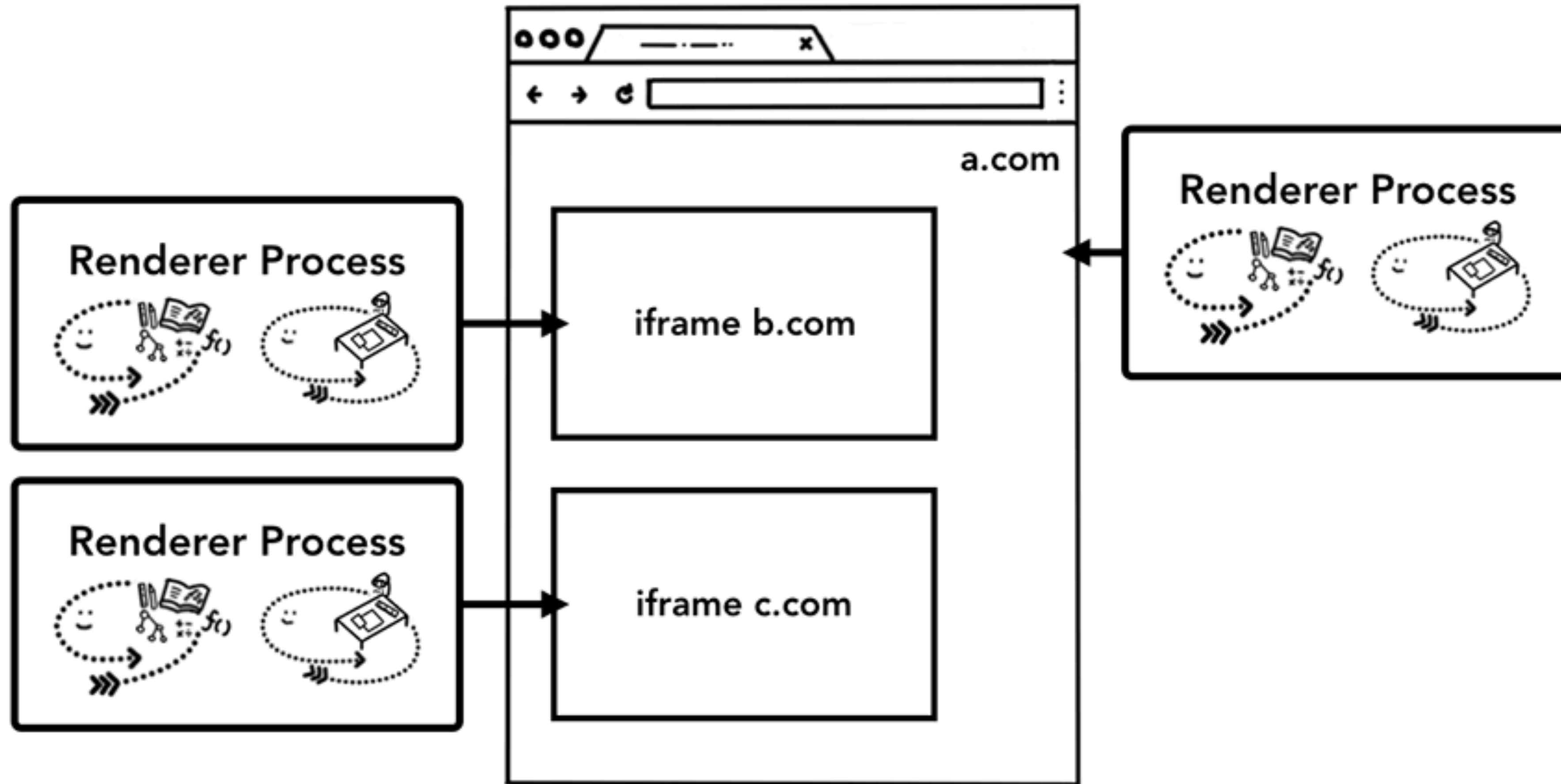
Controls any plugins used by the website, for example, flash.

## GPU Process

Handles GPU tasks in isolation from other processes. It is separated into different process because GPUs handles requests from multiple apps and draw them in the same surface



# Site Isolation



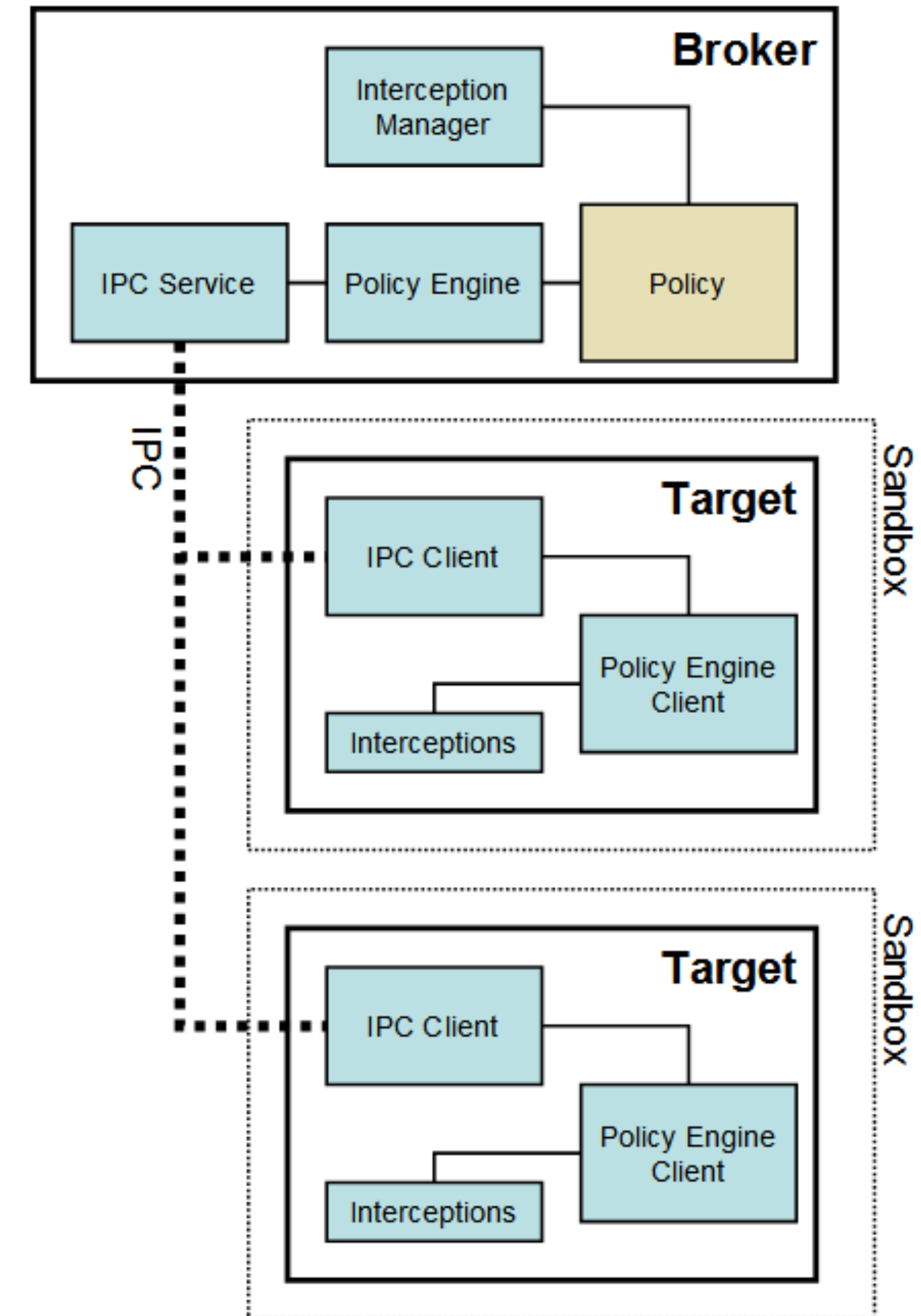


# Chrome Architecture

## Broker (Main Browser)

Privileged controller/supervisor of the activities of the sandboxed processes

Renderer's only access to the network is via its parent browser process and file system access can be restricted



# Chrome Sandbox

Chrome's sandbox depends on four Windows mechanisms:

1. a restricted token
2. Windows job object
3. Windows desktop object
4. Windows integrity levels

# Restricted Token

Chrome calls `CreateRestrictedToken` to create a token that has a subset of the user's privileges.

Assigns the token the user and group S-1-0-0 Nobody. Removes access to nearly every system resource.

As long as the disk root directories have non-null security, no files (even with null ACLs) can be accessed

No network access (on Vista and later)

# Windows Job Object

Renderer runs as a “Job” object rather than an interactive process.

Eliminates access to:

- desktop and display settings
- clipboard
- creating subprocesses
- access to global atoms table

# Alternate Windows Desktop

Windows on the same desktop are effectively in the same security context because the sending and receiving of window messages is not subject to any security checks.

Sending messages across desktops is not allowed.

Chrome creates an additional desktop for target processes

Isolates the sandboxed processes from snooping in the user's interactions

# Windows Integrity Levels

Windows Vista introduced concept of integrity levels

Five Levels: untrusted, low, medium, high, system

Most processes run at medium level

Low-integrity level has limited scope, e.g., can read but cannot write most files

Example of defense in depth.

# Android Process Isolation

Android uses Linux and its own kernel application sandbox for isolation

Each application runs with its own UID in its own VM

- Apps cannot interact with one another
- Limit access to system resources (decided at installation time)

Reference monitor checks permissions on intercomponent communication

# Design Principles of Secure Systems



# Least Privilege

Every program and user should operate using the least privileges necessary to complete their job.

“Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. Thus, if a question arises related to misuse of a privilege, the number of programs that must be audited is minimized. Put another way, if a mechanism can provide "firewalls," the principle of least privilege provides a rationale for where to install the firewalls. The military security rule of "need-to-know" is an example of this principle.”

# Privilege Separation

**Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.**

“[...] Once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information. [...] This principle is at work in the defense system that fires a nuclear weapon only if two different people both give the correct command. In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted. For example, systems providing user-extendible protected data types usually depend on separation of privilege for their implementation.”

# Economy of Mechanism

**Keep the design as simple and small as possible.**

“This well-known principle applies to any aspect of a system, but it deserves emphasis for protection mechanisms for this reason: design and implementation errors that result in unwanted access paths will not be noticed during normal use (since normal use usually does not include attempts to access improper paths).

As a result, techniques such as line-by-line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.”

# Fail-Safe Defaults (Fail Closed)

Base access decisions on permission rather than exclusion. e.g., Whitelists are better than blacklists.

“The default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design. A conservative design must be based on arguments why objects should be accessible, rather than why they should not.”

# Open Design

The design should not be secret. Do not rely on security through obscurity.

“The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. In addition, any skeptical user may be allowed to convince himself that the system he is about to use is adequate for his purpose. Finally, it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.”



# Complete Mediation

**Every access to every object must be checked for authority.**

“This principle, when systematically applied, is the primary underpinning of the protection system. It forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.”



# Complete Mediation





# Least Common Mechanism

**Minimize the amount of mechanism common to more than one user and depended on by all users.**

“Every shared mechanism represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security. Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users. For example, given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the user's own, choose the latter course.”



# Work Factor

**Compare the cost of circumventing the mechanism with the resources of a potential attacker.**

The cost of circumventing in some cases can be easily calculated. For example, the number of experiments needed to try all possible four letter alphabetic passwords is  $26^4 = 456,976$ . If the potential attacker must enter each experimental password at a terminal, one might consider a four-letter password to be adequate. On the other hand, if the attacker could use a large computer capable of trying a million passwords per second, as might be the case where industrial espionage or military security is being considered, a four-letter password would be a minor barrier for a potential intruder.

# Defense in Depth

Layer defensive mechanisms.

Defend a system against any particular attack using several independent methods

If one mechanism fails, another steps in to prevent the attack.



# **Security Principles and Mechanisms**

**CS155 Computer and Network Security**

**Stanford University**