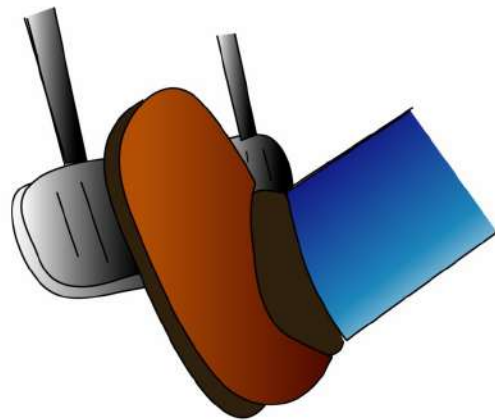


CS5231: Systems Security

Lecture 5: Isolation

Second Line of Defense

- First Line of Defense
 - Directly prevent the attack from happening
- Second Line of Defense
 - Assume that attack happens, minimize the impact



Vs.





Sandboxing: Access Control

Access Control Primitives

- Definitions:
 - Resource Objects
 - "Elements that need to be protected"
 - Authorities or Principals
 - "Subjects accessing the resources"
 - Permissions
 - "Access Rights"
 - Isolation Environment (or protection domain)
 - "A domain in which program executes. It determines what the program will do."

Access Control Matrix

Directory

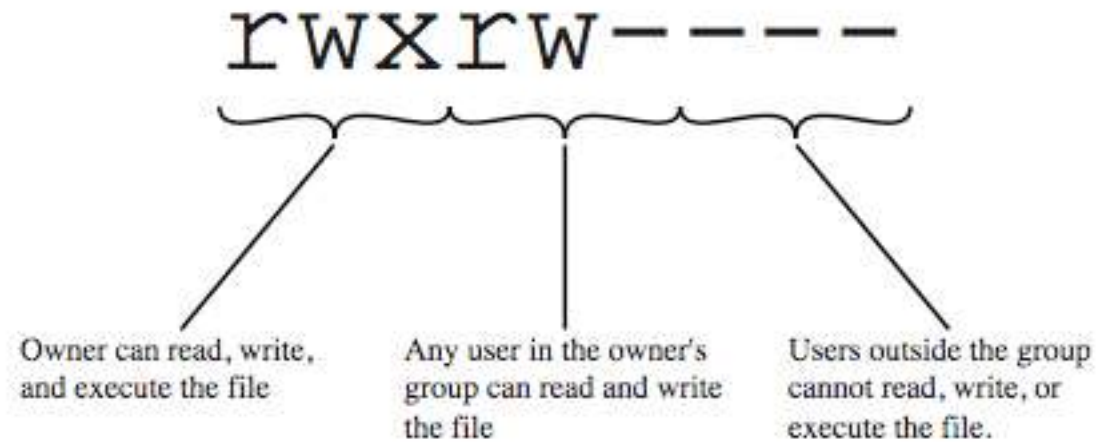


	BIBLIOG	TEMP	F	HELP.TXT	C_COMP	LINKER	SYS_CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Access
Control List

Access Rights or
Permissions

Example: UNIX File Access Control



Example of Delegation & Groups: UNIX File Access Control

- “set user ID”(SetUID) or “set group ID”(SetGID)
 - system temporarily uses rights of the file owner / group in addition to the real user’s rights when making access control decisions
 - enables privileged programs to access files / resources not generally accessible
- sticky bit
 - on directory limits rename/move/delete to owner
- superuser
 - is exempt from usual access control restrictions

Example of Delegation & Groups: UNIX Access Control Lists

- modern UNIX systems support ACLs
- can specify any number of additional users / groups and associated rwx permissions
- ACLs are optional extensions to std perms
- group perms also set max ACL perms
- when access is required
 - select most appropriate ACL
 - owner, named users, owning / named groups, others
 - check if have sufficient permissions for access

Summary of Definitions: Access Control Primitives

- Definitions:
 - **Authorities or Principals**
"Subjects accessing the resources"
 - **Resource Objects**
"Elements that need to be protected"
 - **Permissions**
"Access Rights"
 - **Isolation Environment (or protection domain)**
"A domain in which program executes. It determines what the program will do."

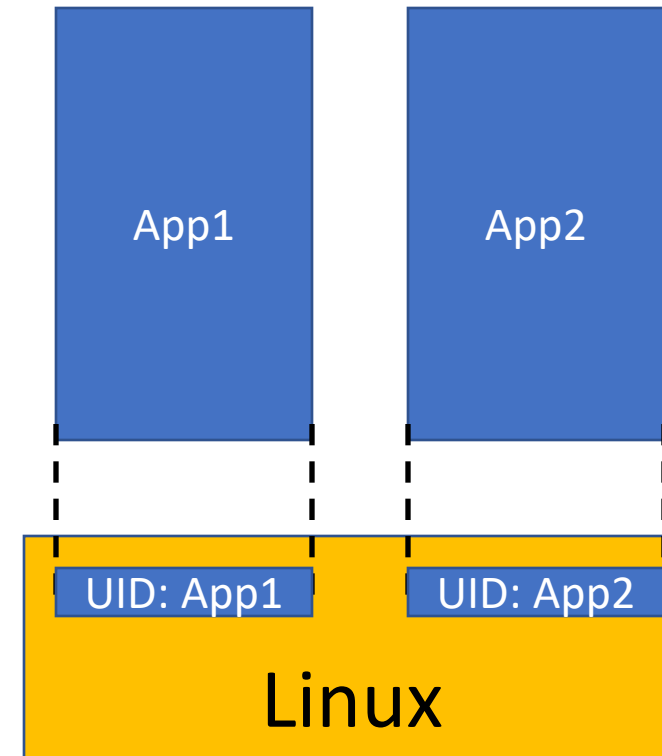
An Example: Android OS



Authorities / Principals

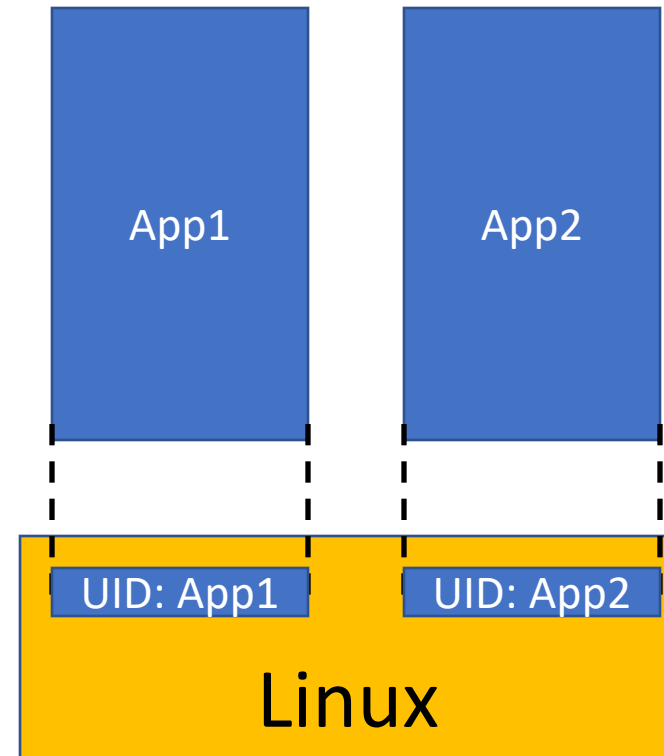
- Each applications is a unique authority (Unix user ids)
- Each app is **signed**

```
$ jarsigner -verify my_signed.apk
```



Isolation Environment

- Isolation via OS Processes
- Why is it better?
 - E.g. Apple iOS browser bug
 - Safari exploit [Miller'08]
 - Lead to compromising the Whole phone!
 - On Android, confined to browser app (UID) only!

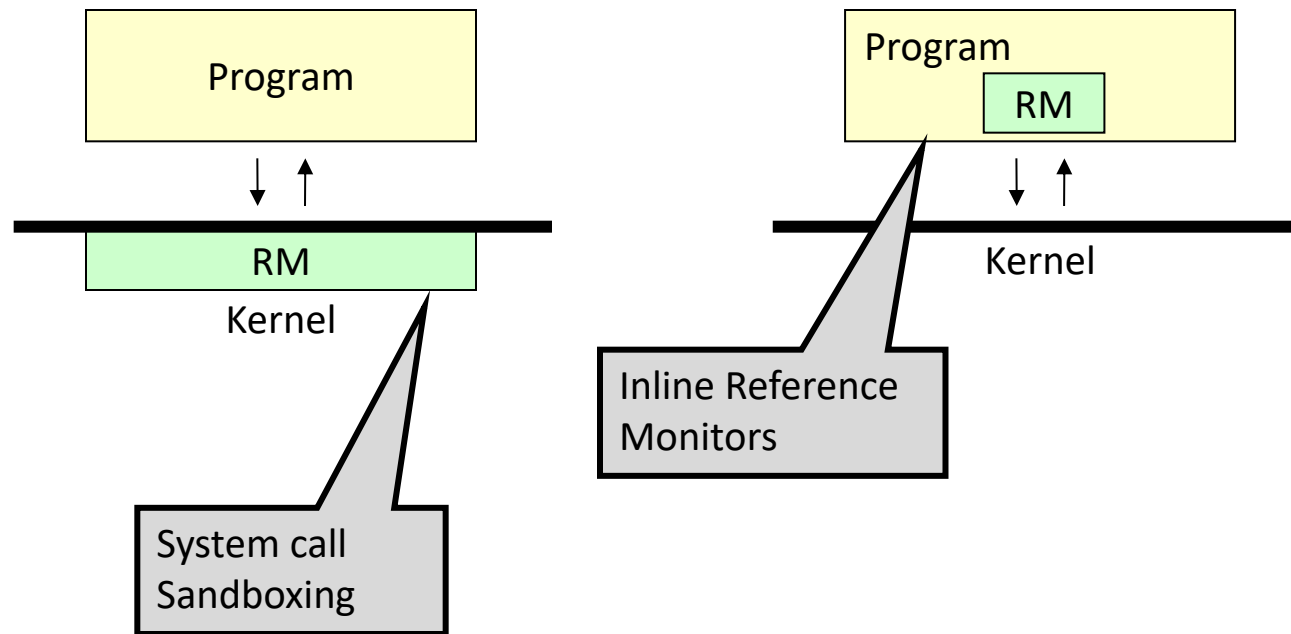


Process sandboxing & Inline Reference Monitors

Reference Monitors

Reference Monitor: A piece of code that *checks all references* to an **object**

Syscall Sandbox: A reference monitor for protecting OS resource objects from an app



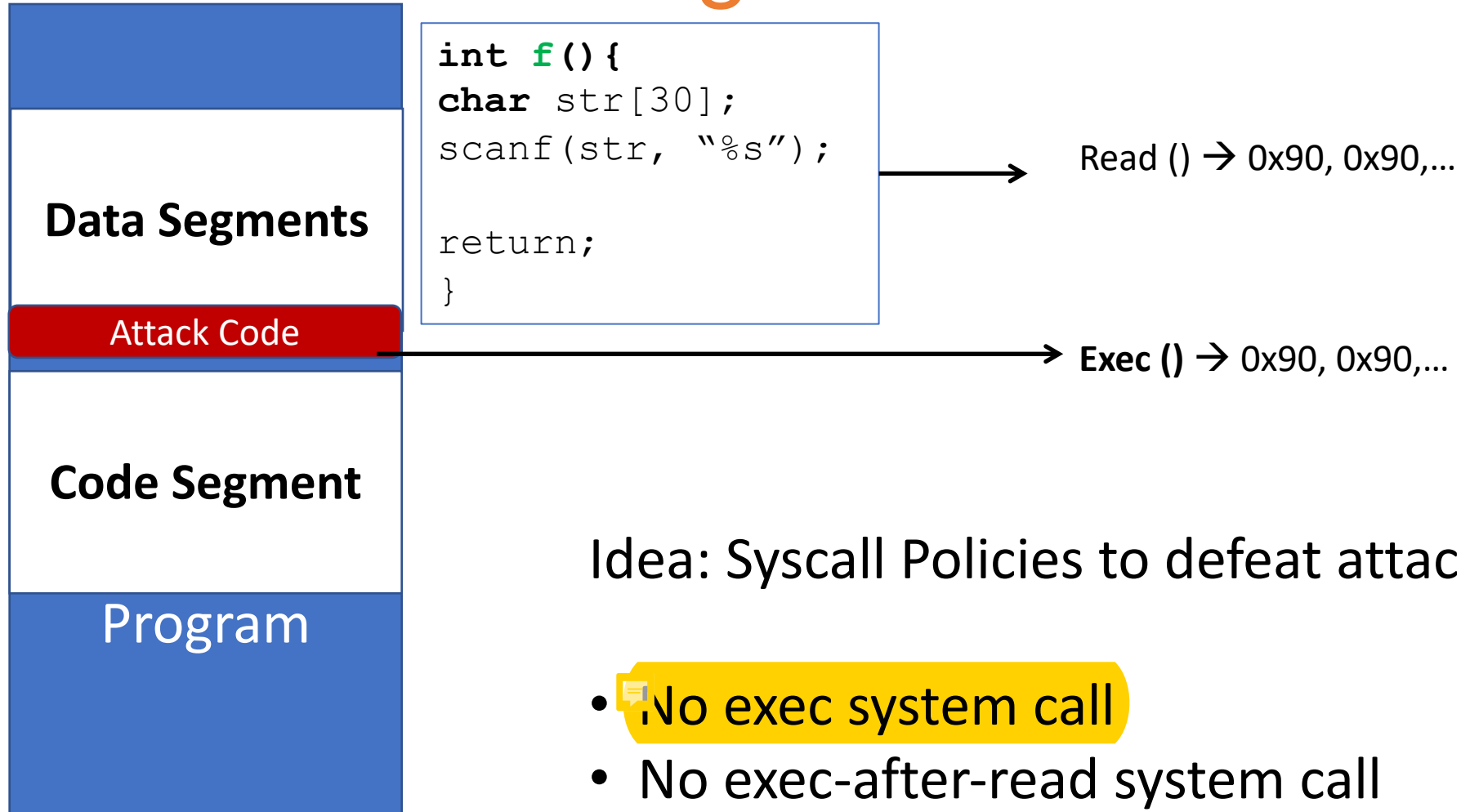
3 Security Principles

- Separation of Concerns:
 - Separate the **policy** from its **enforcement**
- Minimize Trusted Code Base (TCB)
 - Reduce what one needs to trust
 - Separate **verifier** from the **enforcement**
- Least Privilege
 - Give each component only the privileges necessary

Policy vs. Enforcement Mechanism

- Access Control Policies
- Enforcement:
 - Process sandboxing
 - Inline Reference Monitors
 - Virtualization
 - Hardware-based isolation / Trusted Execution Env.

Process Sandboxing



Idea: Syscall Policies to defeat attacks

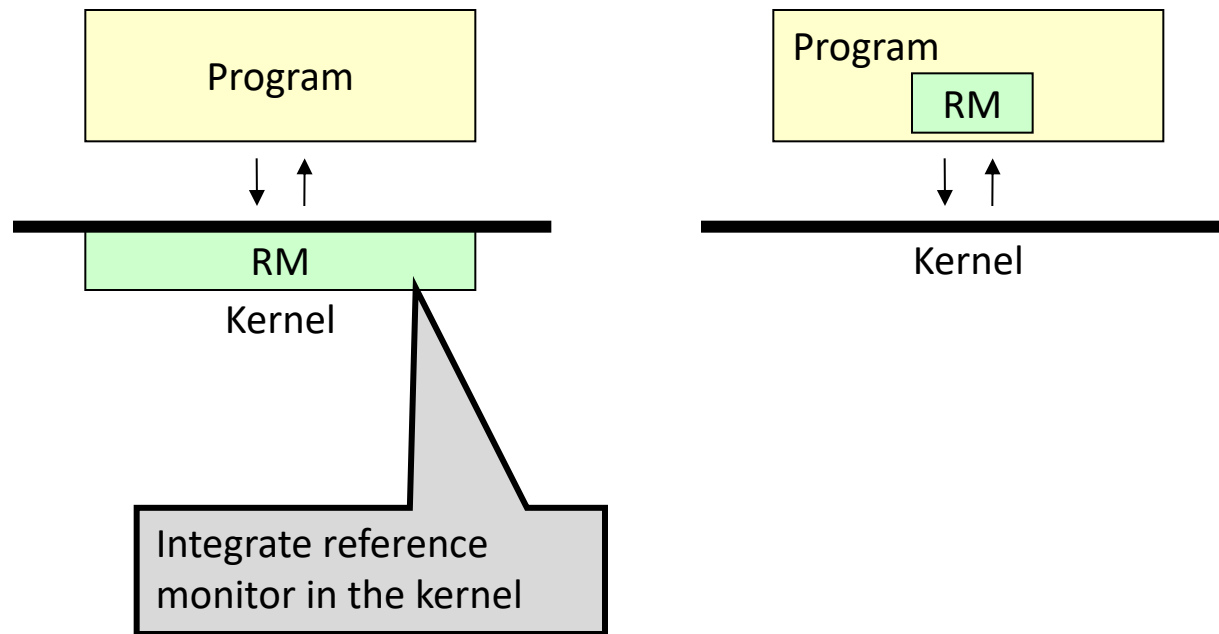
- No exec system call
- No exec-after-read system call

Enforcement Mechanisms: Process Isolation / Sandboxing

System Call Sandboxing

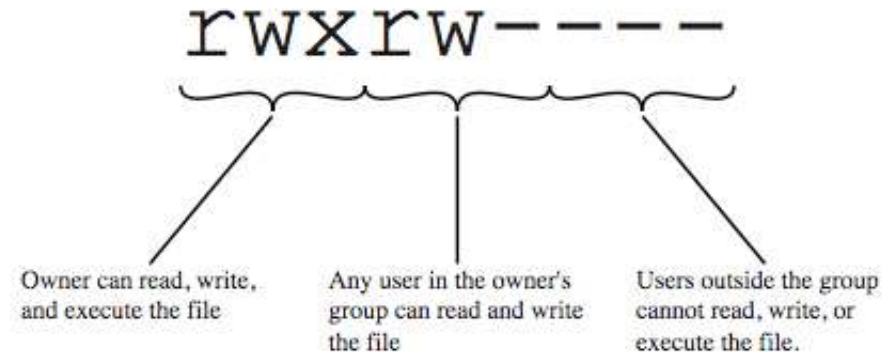
Reference Monitor: A piece of code that *checks all references* to an **object**

Syscall Sandbox: A reference monitor for protecting OS resource objects from an app



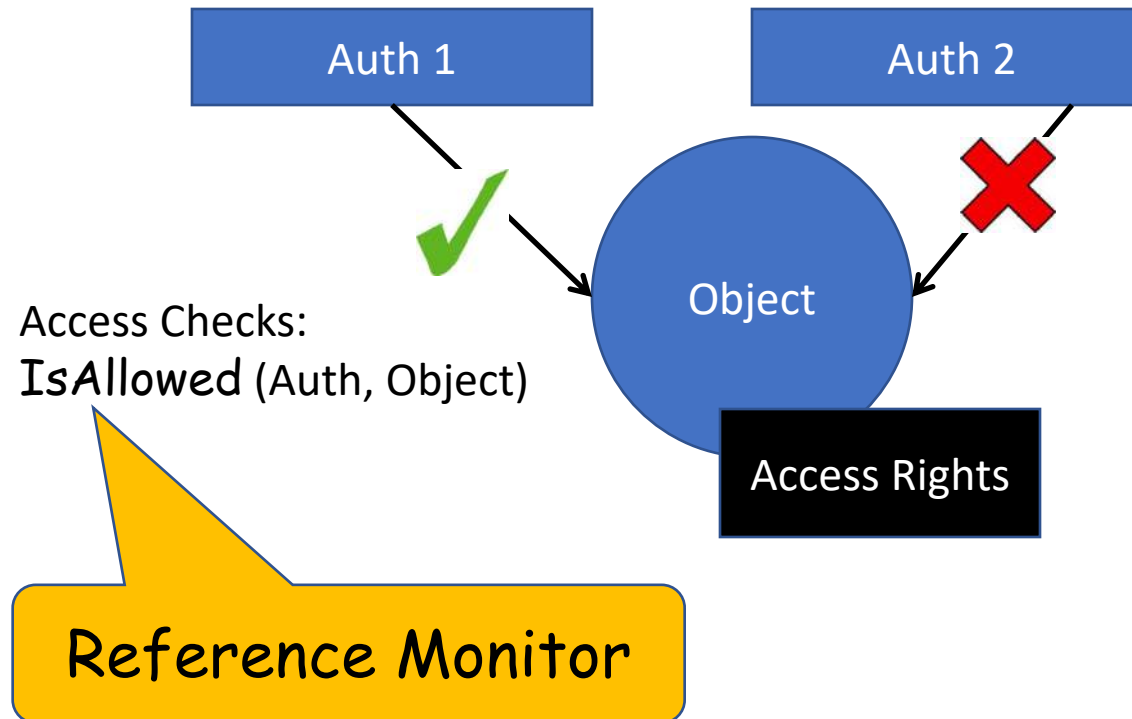
Kernelized Syscall Sandbox (I): Access Control Lists

Access Control List



	BIBLIOG	TEMP	F	HELP.TXT	C COMP	LINKER	SYS CLOCK	PRINTER
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	-	-	R	X	X	R	W
USER S	RW	-	R	R	X	X	R	W
USER T	-	-	-	R	X	X	R	W
SYS_MGR	-	-	-	RW	OX	OX	ORW	O
USER_SVCS	-	-	-	O	X	X	R	W

Kernelized Syscall Sandbox (I): Access Control Lists



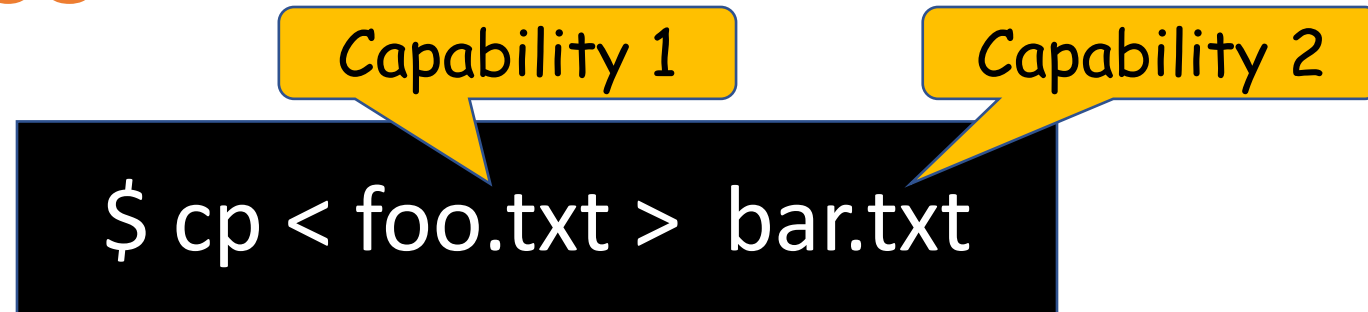
Challenge: Ambient Authority

```
$ cp foo.txt bar.txt
```

The “cp” program has authority to write to **any** file on the system.

This is not in line with “Principle of Least Privilege”

Kernelized Syscall Sandbox (II): Capabilities



The “cp” program has no authority, by default.
It can only use “capabilities” it is given (e.g. UNIX file handles)

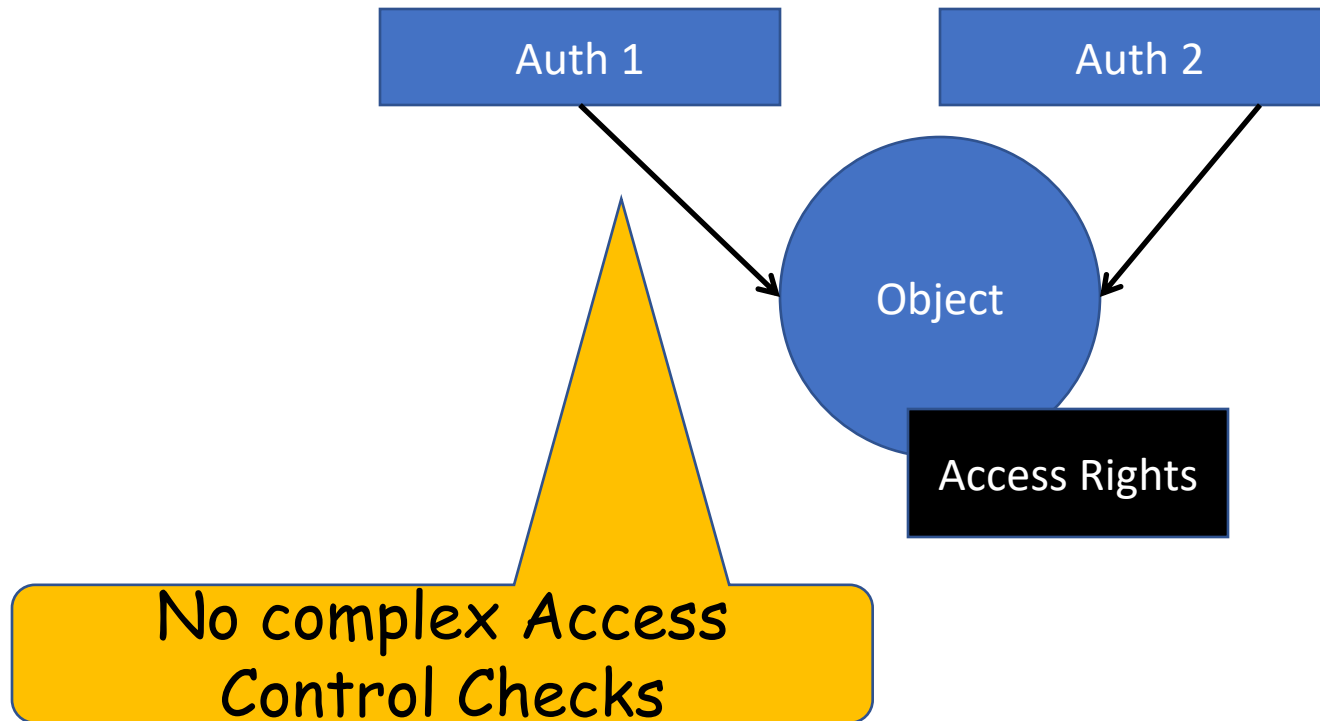
Definition of a **Capability**:

- An *identifier* which, when presented, provides certain access rights

Properties of a **Capability**:

- Unforgeable: Can’t manufacture without explicitly getting it.

Kernelized Syscall Sandbox (II): Capabilities



Access Control Lists vs. Capabilities

ACL

- Pros:
 - When the checks are simple and centralized, easier to implement ACL
 - Works well when rights change
- Cons:
 - Ambient Authority
 - Incomplete mediation:
 - Missing access control checks

Capabilities

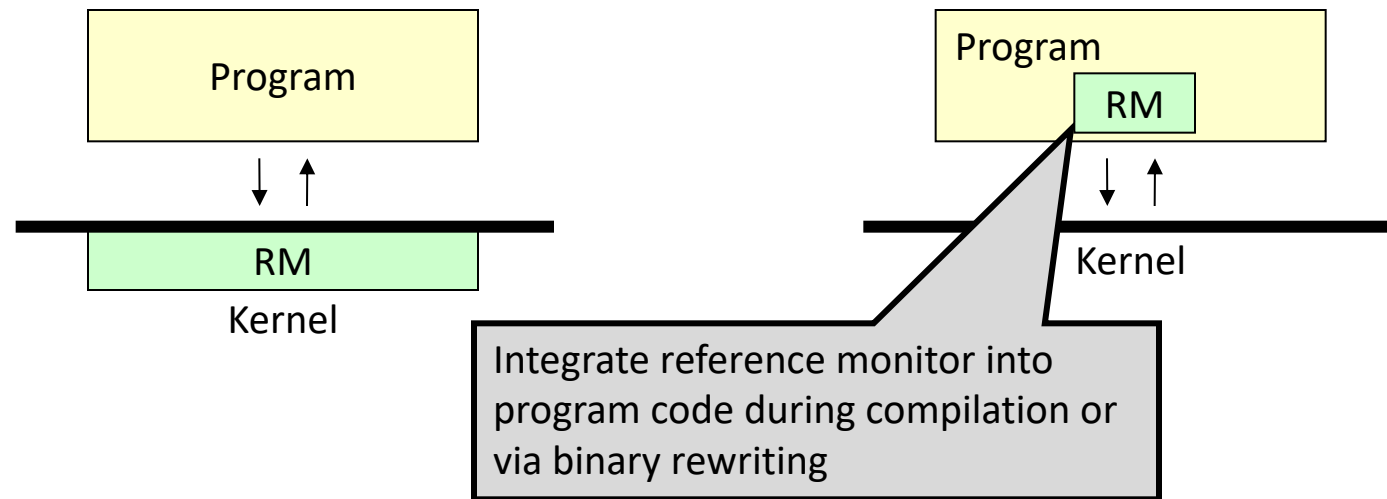
- Pros:
 - Eliminates access check logic
 - No pre-specification of who is allowed to access, i.e., can follow the natural flow of access rights
 - No ambient authority
 - Recall Least Privilege
- Cons:
 - Unsuitable when access rights change frequently
 - Capabilities can leak!

Inline Reference Monitors

Inline Reference Monitors

Reference Monitor: A piece of code that *checks all* **references** to an **object**

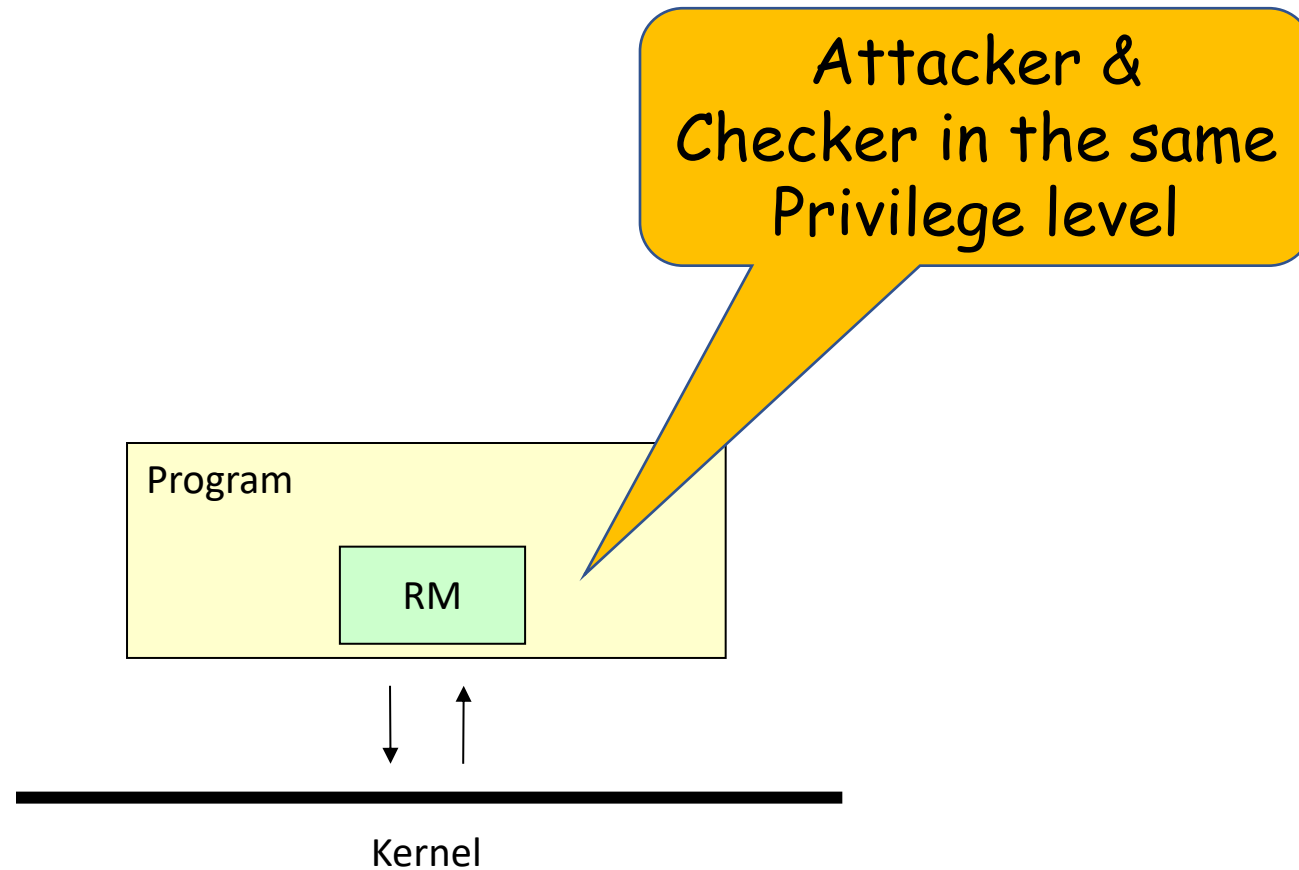
Syscall Sandbox: A reference monitor for protecting OS resource objects from an app



Inline Reference Monitors Can Check...

- **Complete Memory Safety**
"Access memory objects in an intended way"
- **Fault Isolation**
"Each module only accesses pre-determined data / code"
- **No foreign code**
"Execute only predetermined code"
- **Control Flow Integrity**
"Control transfers are to legitimate points only"
- **System Call Sandboxing**
"Access only a subset of system calls"
- **(Code) Pointers / Data Integrity**
"Ensure (code) pointers / data have valid values"
- **Data Flow Integrity...**

Challenges in Inline / Wrapper-based Enforcement



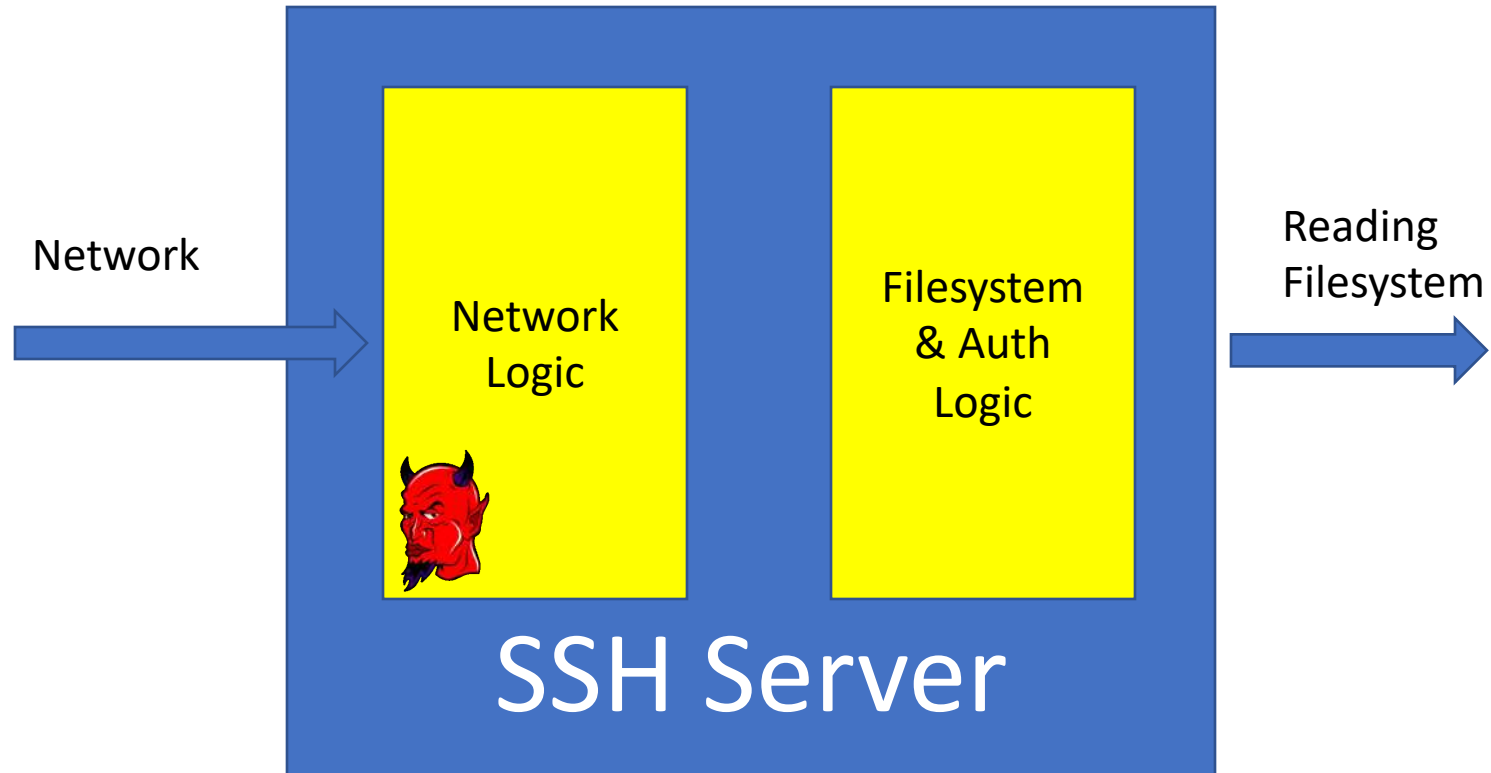
Takeaways: 3 Security Principles

- Separation of Concerns:
 - Separate the **policy** from its **enforcement**
- Minimize Trusted Code Base (TCB)
 - Reduce what one needs to trust
 - Separate **verifier** from the **enforcement**
- Least Privilege
 - Give each component only the privileges necessary

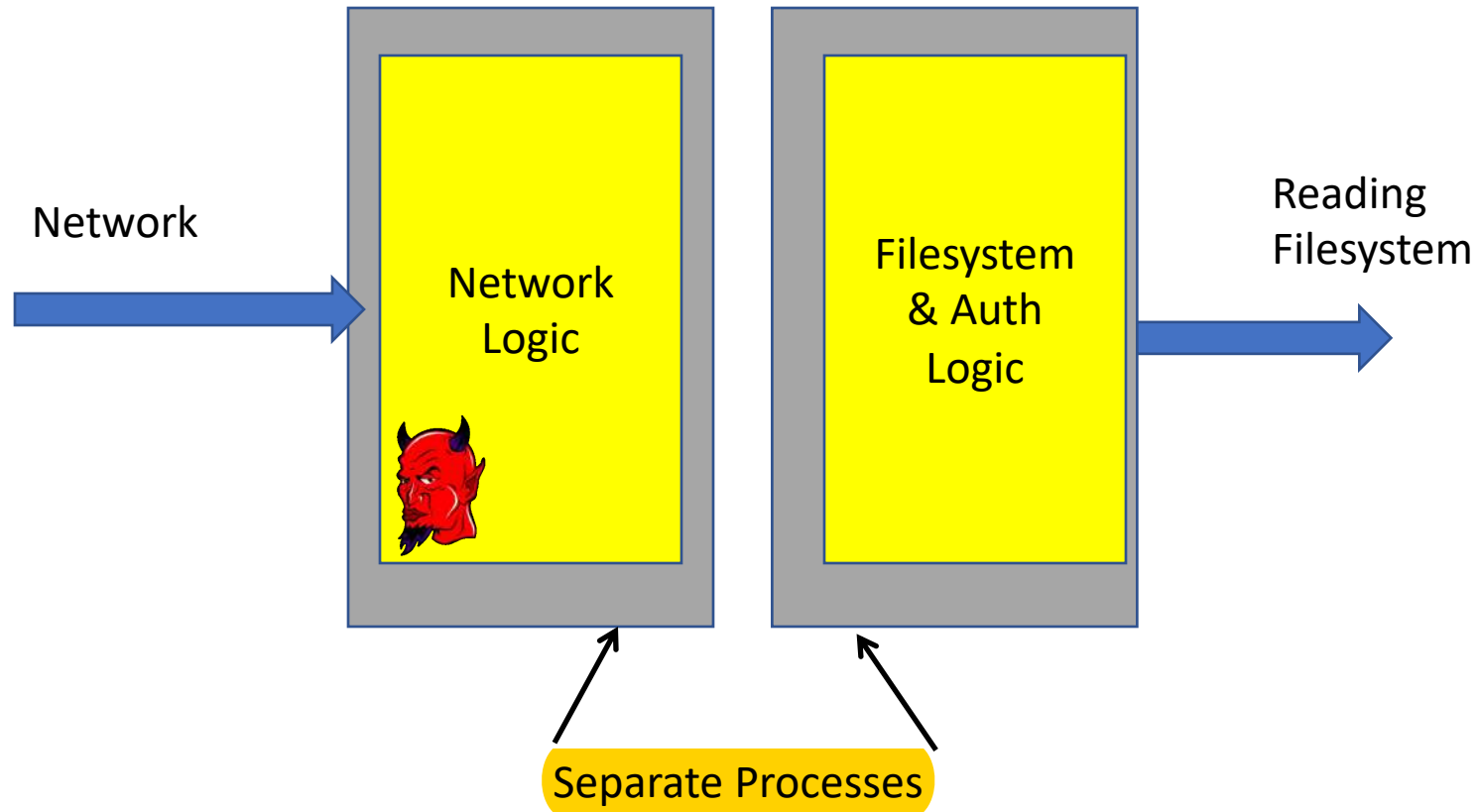
Problem: Bundling of Functionality



Problem: Bundling of Functionality



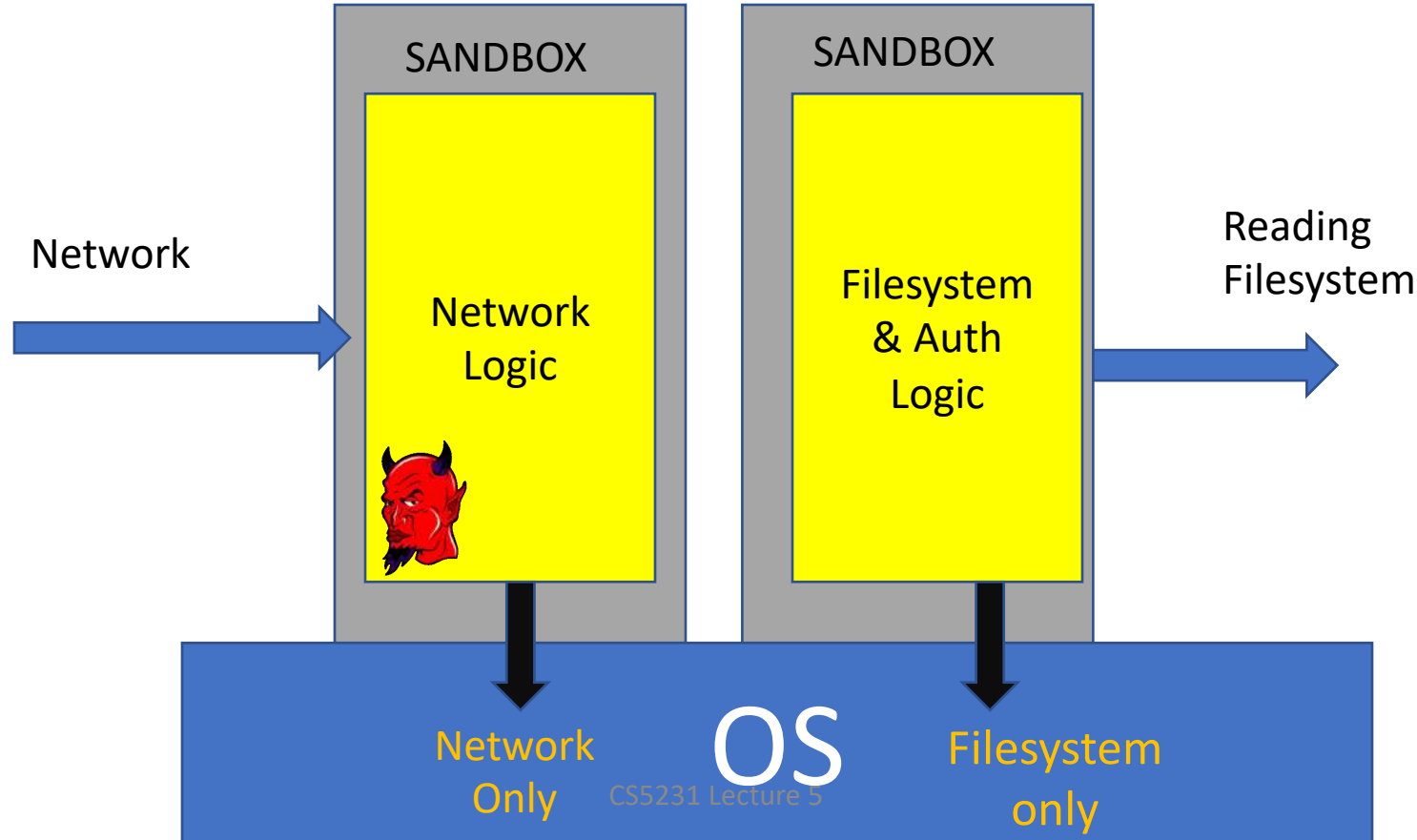
Solution: Privilege Separation





Principle of Least Privilege

- Each compartment gets the least set of privileges it needs for its function

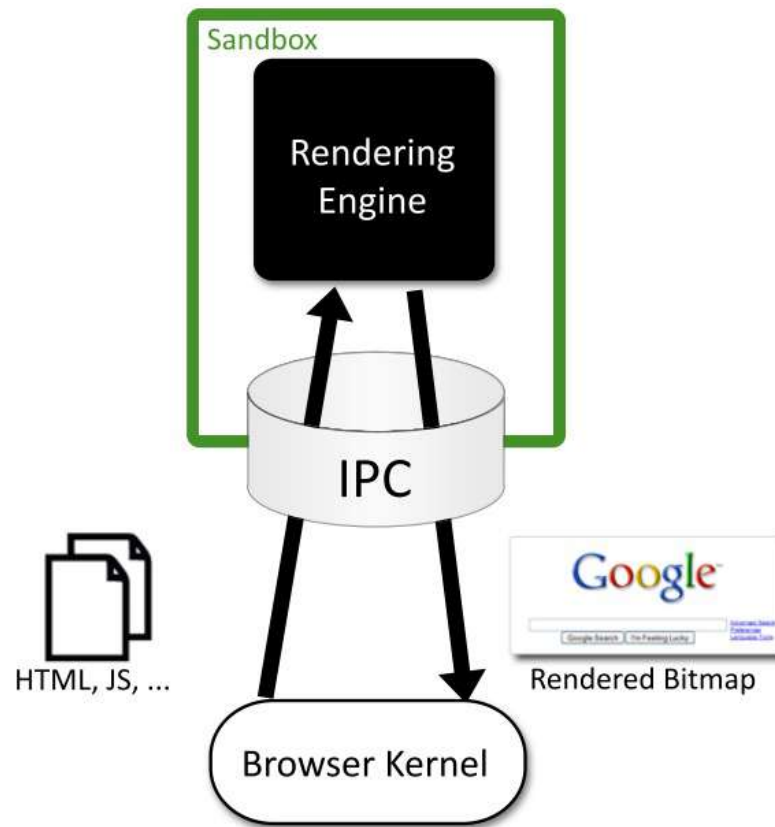


Design Browser With Isolation

- Problem with Old Browser Design (such as early Firefox): Single-process
 - Vulnerability leads to accessing all origins
- Solution: better Privilege Separation
 - Compartmentalize & assign least privilege
- Google Chrome
 - Goal: Separate filesystem from web code

Google Chrome Design

- Goal: Prevent web & network attacker from compromising OS resources (e.g. filesystem)



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	

Google Chrome

- One excellent idea: Using OS mechanism to protect resources in browser
 - Run each tab in a separate process
 - Error in one tab won't affect other tabs
- Read more:
<http://www.google.com/googlebooks/chrome/>

