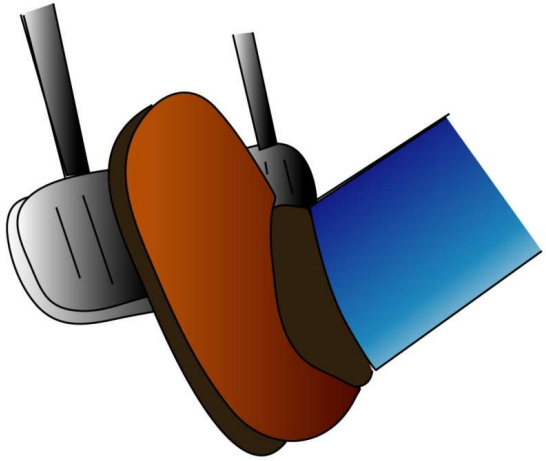
Process sandboxing & Inline Reference Monitors

Prateek Saxena

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

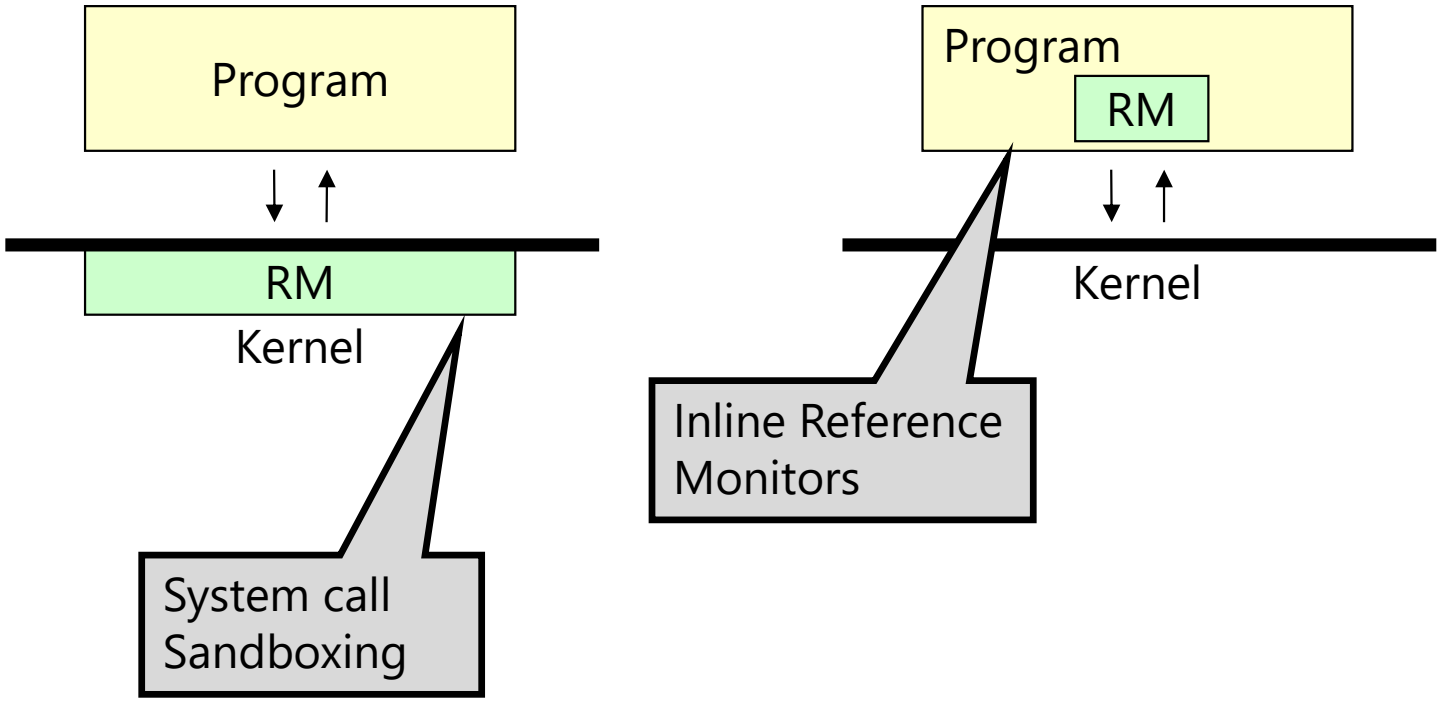


Vs.

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



Slide from Shmatikov et. al.

3 Security Principles

- Separation of Concerns:
  - Separate the **policy** from its **enforcement**

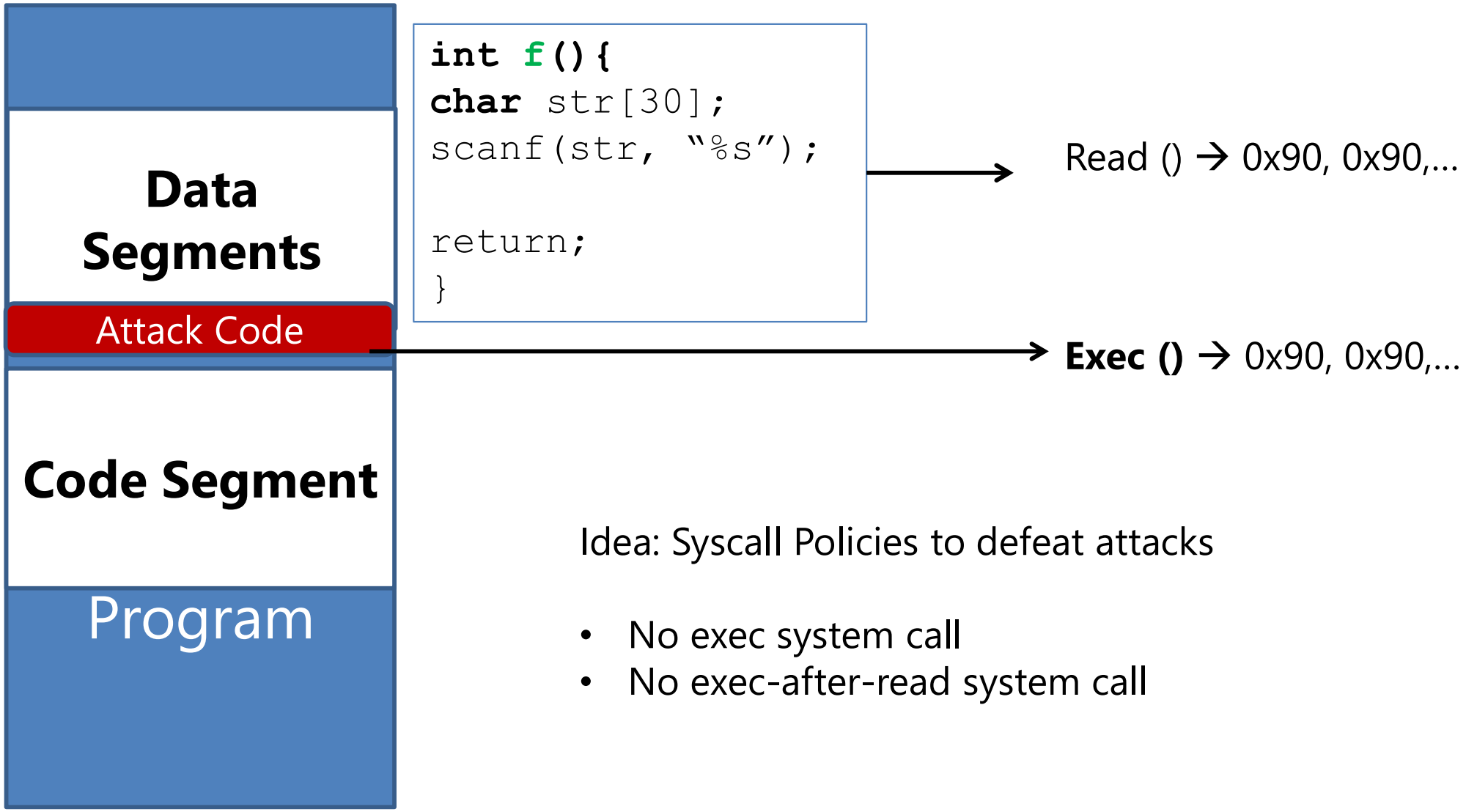
- Minimize Trusted Code Base (TCB)
  - Reduce what one needs to trust
  - Separate **verifier** from the **enforcement**

- Least Privilege
  - Give each component only the privileges necessary

Policy vs. Enforcement Mechanism

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

Process Sandboxing



Idea: Syscall Policies to defeat attacks

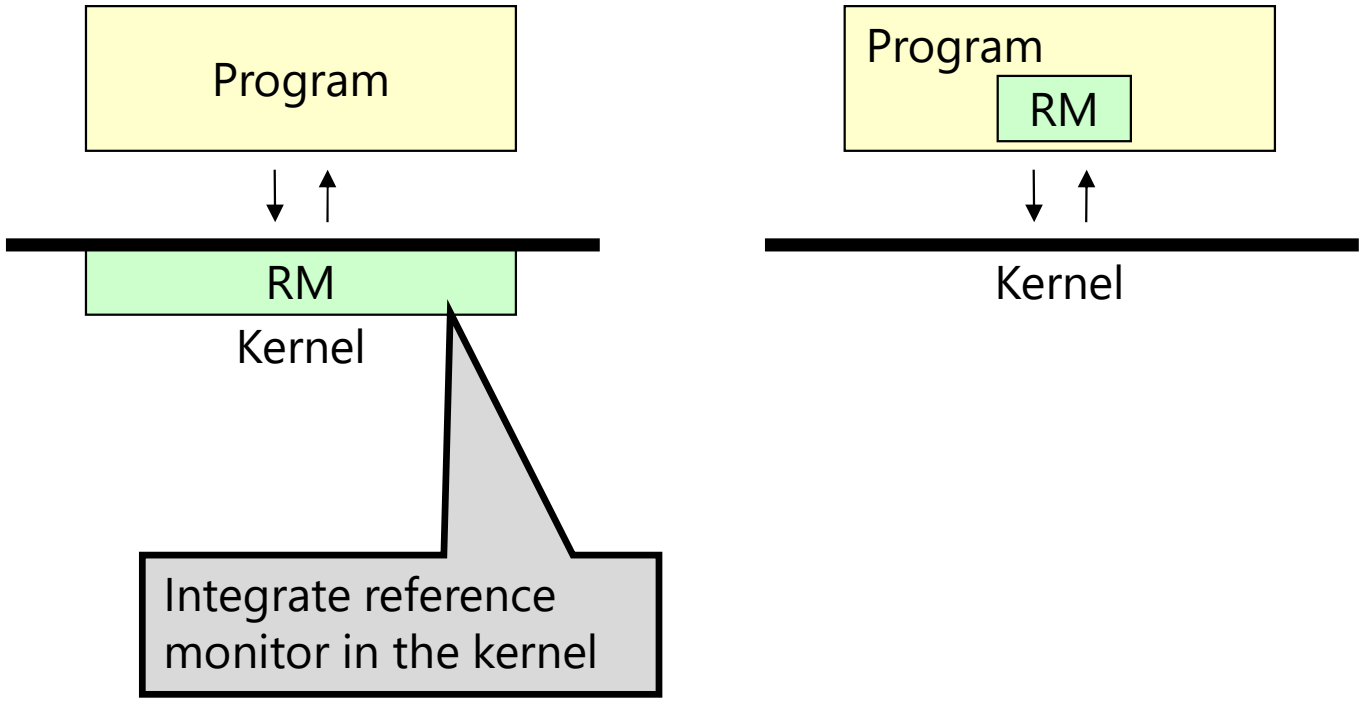
- No exec system call
- No exec-after-read system call

Enforcement Mechanisms:
Process Isolation / Sandboxing

System Call Sandboxing

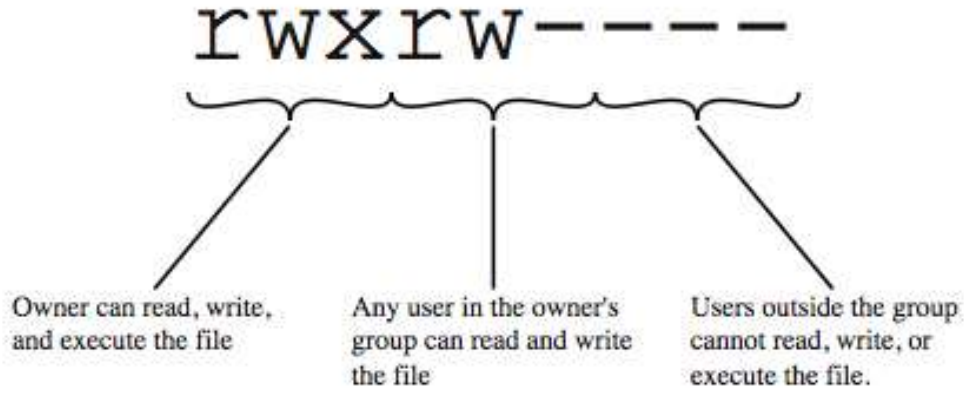
Reference Monitor: A piece of code that *checks all* **references** to an **object**

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



Slide from Shmatikov et. al.

Kernelized Syscall Sandbox (I): Access Control Lists

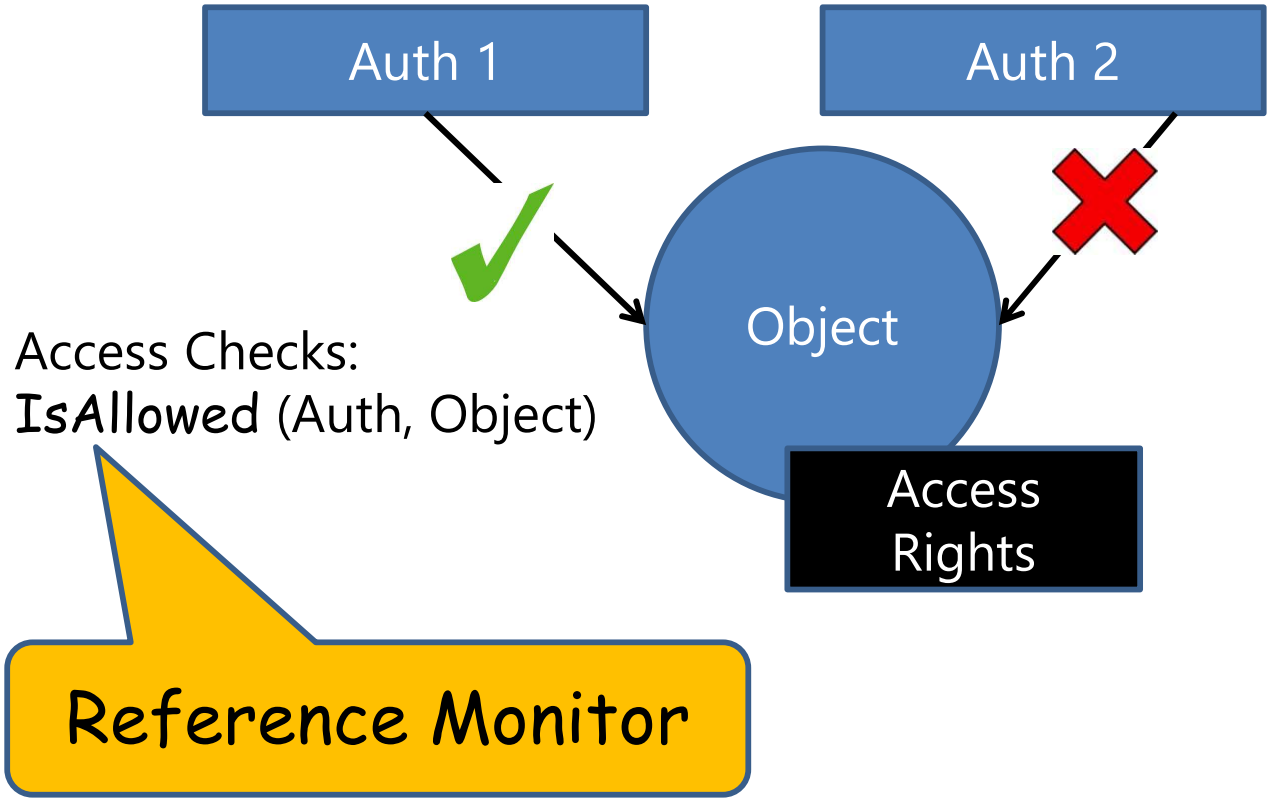


Access Control List

	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

Protection and access control in operating systems [Lampson'72]

Kernelized Syscall Sandbox (I): Access Control Lists



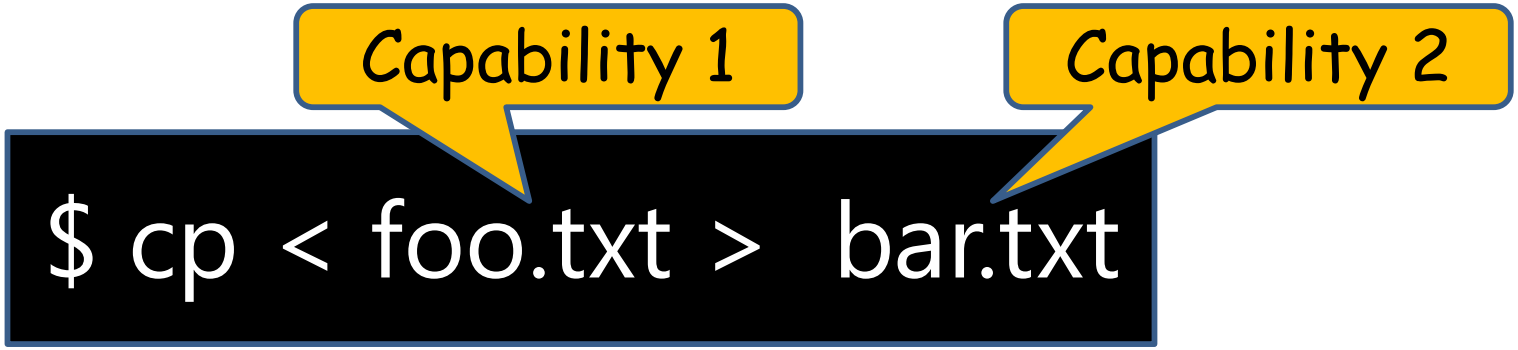
Challenge: Ambient Authority

```
$ 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



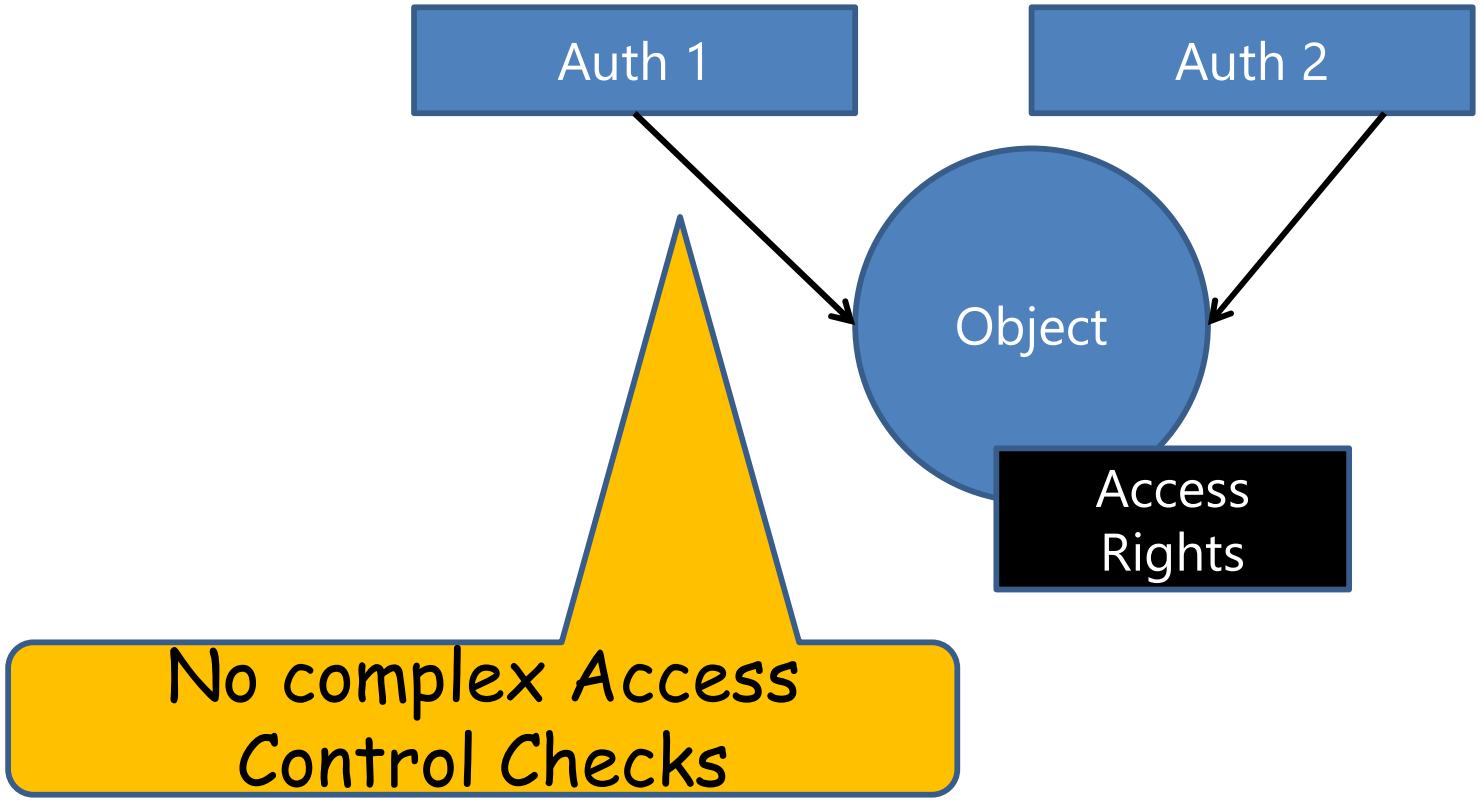
The "cp" program has no authority, by default.
It can only use "capabilities" it is given (e.g. UNIX file handles)

Definition of a **Capability**:
- An *identifier* which, when presented, provides certain access rights

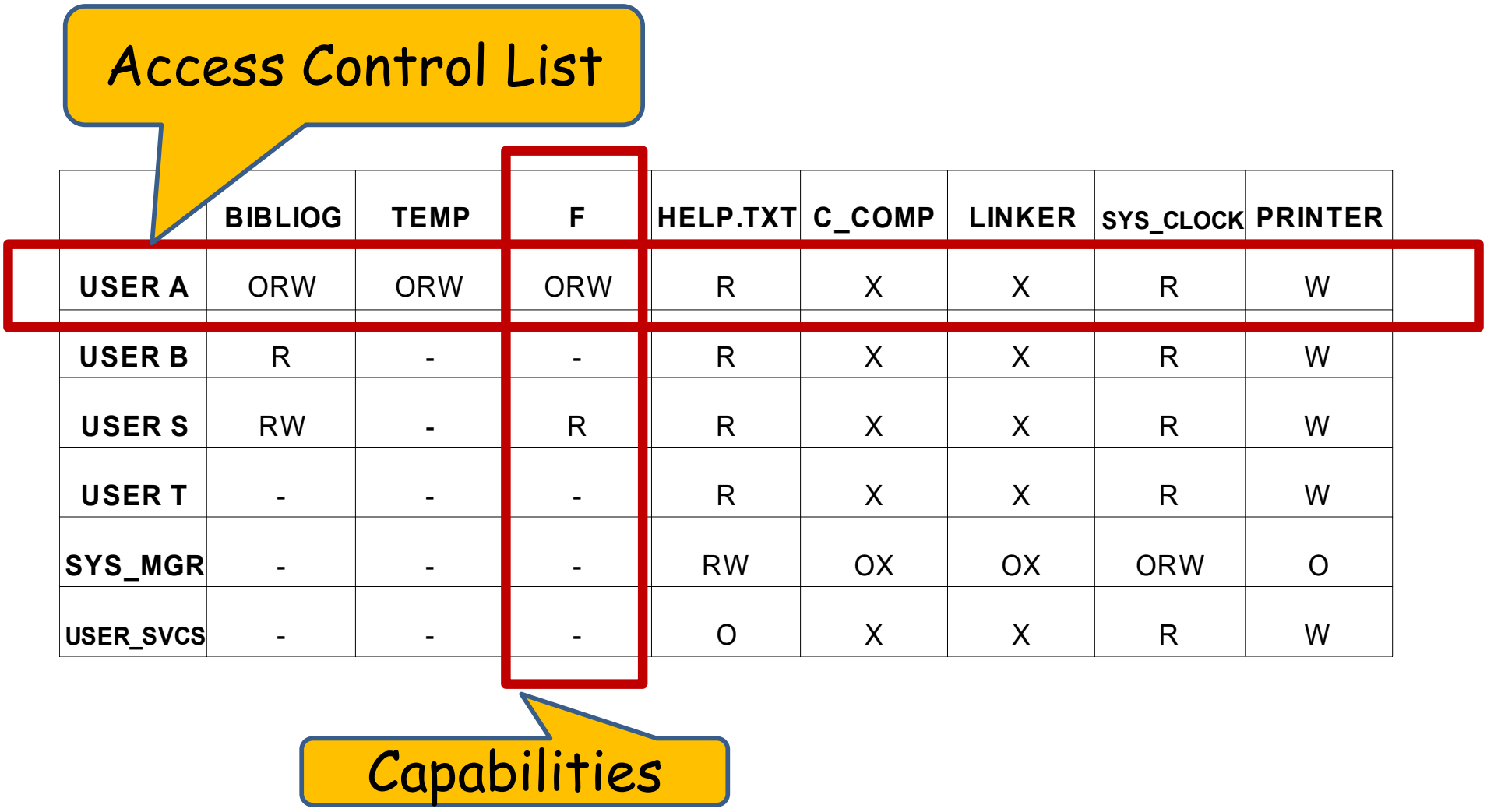
Properties of a **Capability**:
- Unforgeable: Can't manufacture without explicitly getting it.

Reference: First 20 minutes of Object Capabilities for Security

Kernelized Syscall Sandbox (II): Capabilities



Kernelized Syscall Sandbox (II): Capabilities



	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

Protection and access control in operating systems [Lampson'72]

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

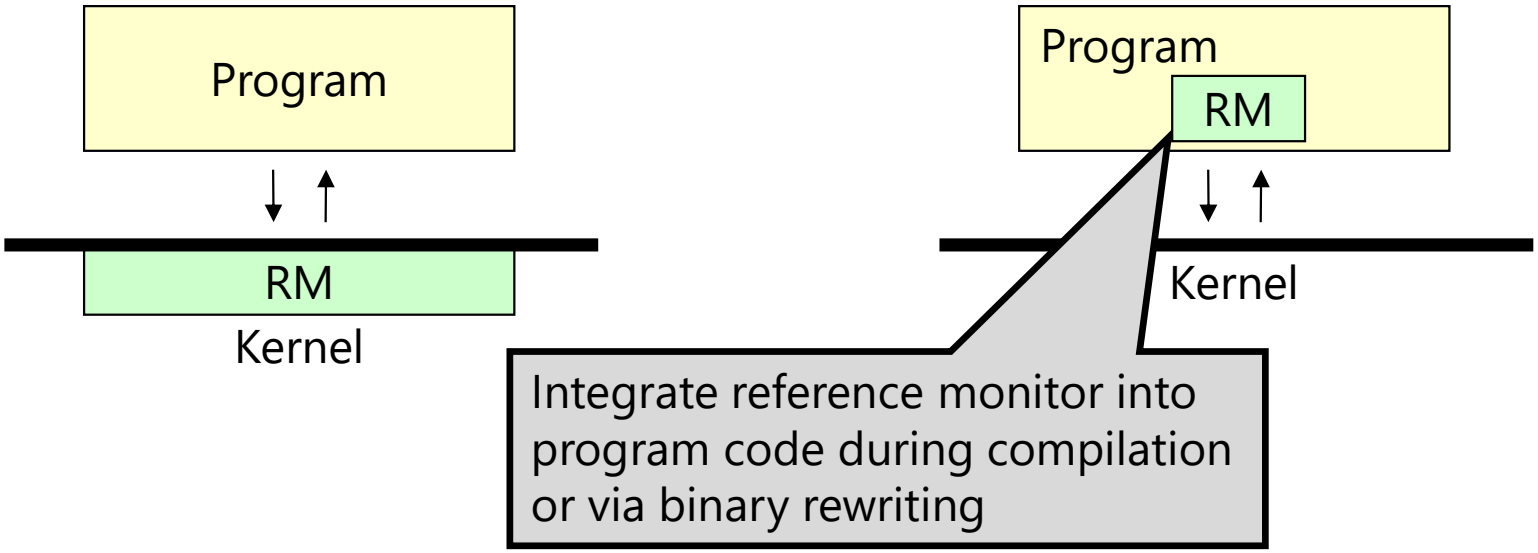
- Pros:
  - Eliminates access check logic
  - No pre-specification of who is allowed to access, i.e., can follow the natural flow of access rights
  - No ambient authority
    - Recall Least Privilege
- Cons:
  - Unsuitable when access rights change frequently
  - Capabilities can leak!

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

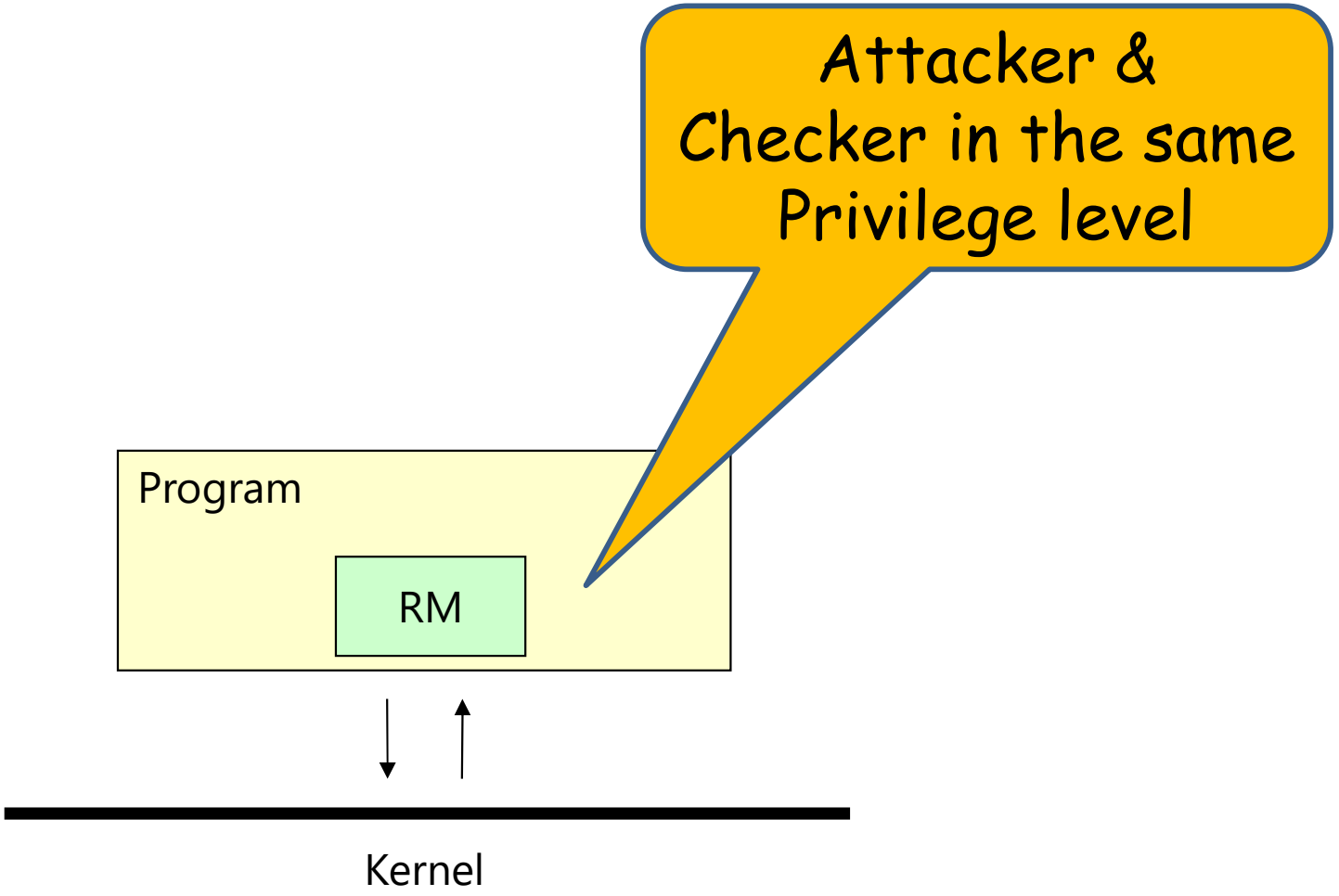


Slide from Shmatikov et. al.

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



Inline Reference Monitors:
Software Fault Isolation

Software Fault Isolation (SFI)

- Goal: Fault Isolation
  - Confine read/write to certain region M
  - This goal is also called “address sandboxing”

- Attacker controls all memory values in M
- Mechanism: Inline instrumentation of D
- Limit all memory accesses to region M
- Take an example: Let M be [0xbe00, 0xbeff]

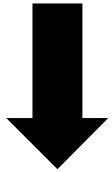
Naïve SFI Implementation

P

```
mov (ebp), eax
```

SRC DST

(REG) : Register Indirect Memory access

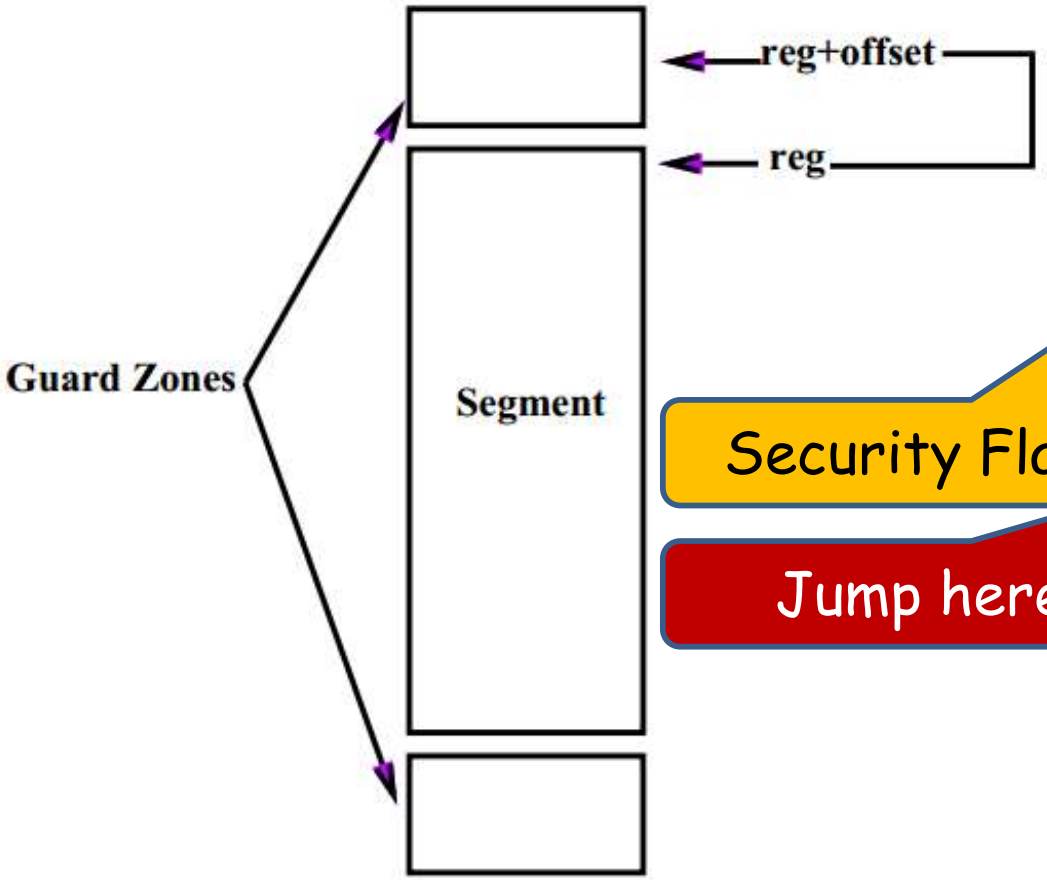


P'

```
call (InRange(ebp))
jz error_label
mov (ebp), eax
```

Runtime Overhead?

Fast SFI Implementation




```
mov (ebp), eax
```

```
and ebp, 0x00ff
or  ebp, 0xbe00
mov (ebp), eax
```

Security Flaw?

Jump here!

```
and reg1, 0x00ff
or  reg1, 0xbe00
mov (reg1), eax
```

FIX: Using Dediated Regs.

Co-erce
Rather than Check

[Efficient Software-based Fault Isolation [SOSP'93]]

Verifying Correctness of Fast-SFI

1. Check if these IRM instructions exist before memory access
2. All memory accesses use the dedicated register
3. The dedicated registers are used only in IRM instructions

```
and reg1, 0x00ff
or  reg1, 0xbe00
mov (reg1), eax
```

Efficient Software-based Fault Isolation [SOSP'93]

3 Security Principles

- Separation of Concerns:
  - Separate the **policy** from its **enforcement**

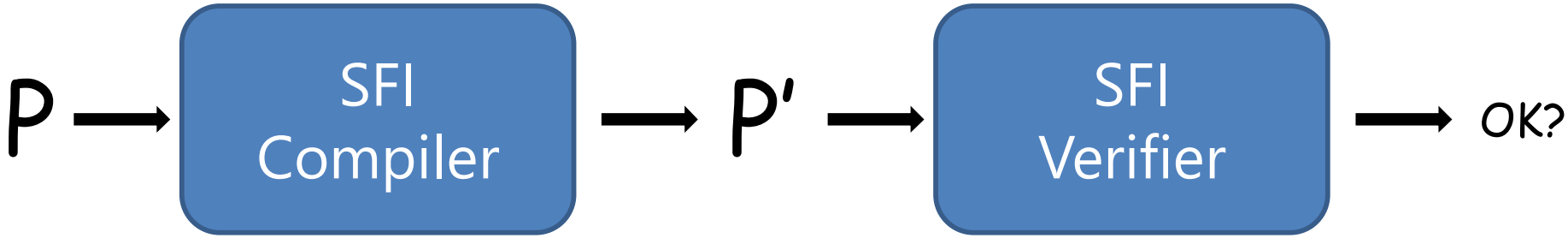
- Minimize Trusted Code Base (TCB)
  - Reduce what one needs to trust
  - Separate **verifier** from the **enforcement**

- Least Privilege
  - Give each component only the privileges necessary

SFI Has a Small TCB…

- Goal of Software Fault Isolation:
  - Address Sandboxing

    " Access memory segments statically verified"



- Trusted Computing Base (TCB):

  "The trusted codebase for ensuring security properties"

- Smaller the TCB, the better the design

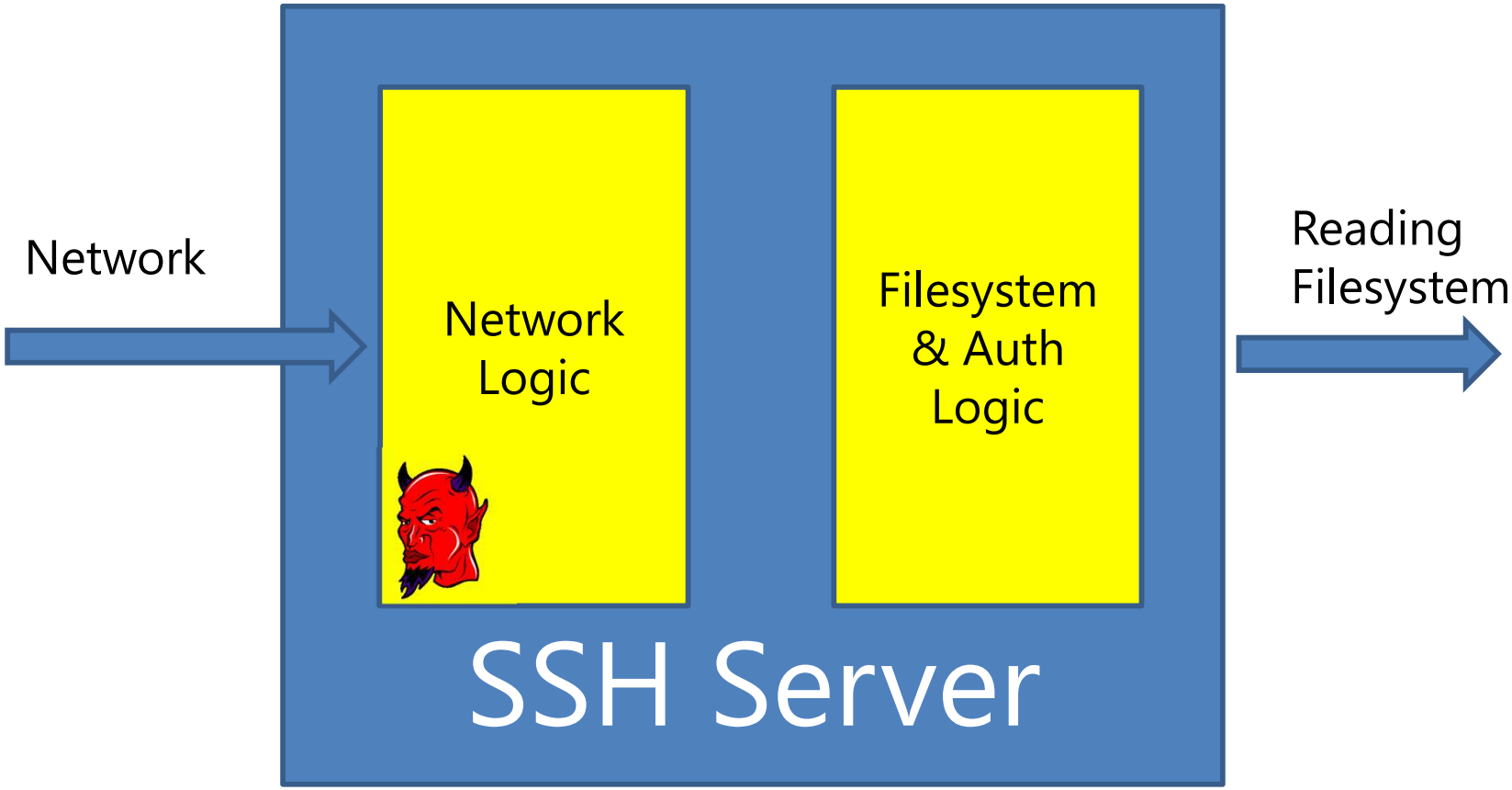
Aiding Syscall Sandboxing: Privilege Separation

Takeaways: 3 Security Principles

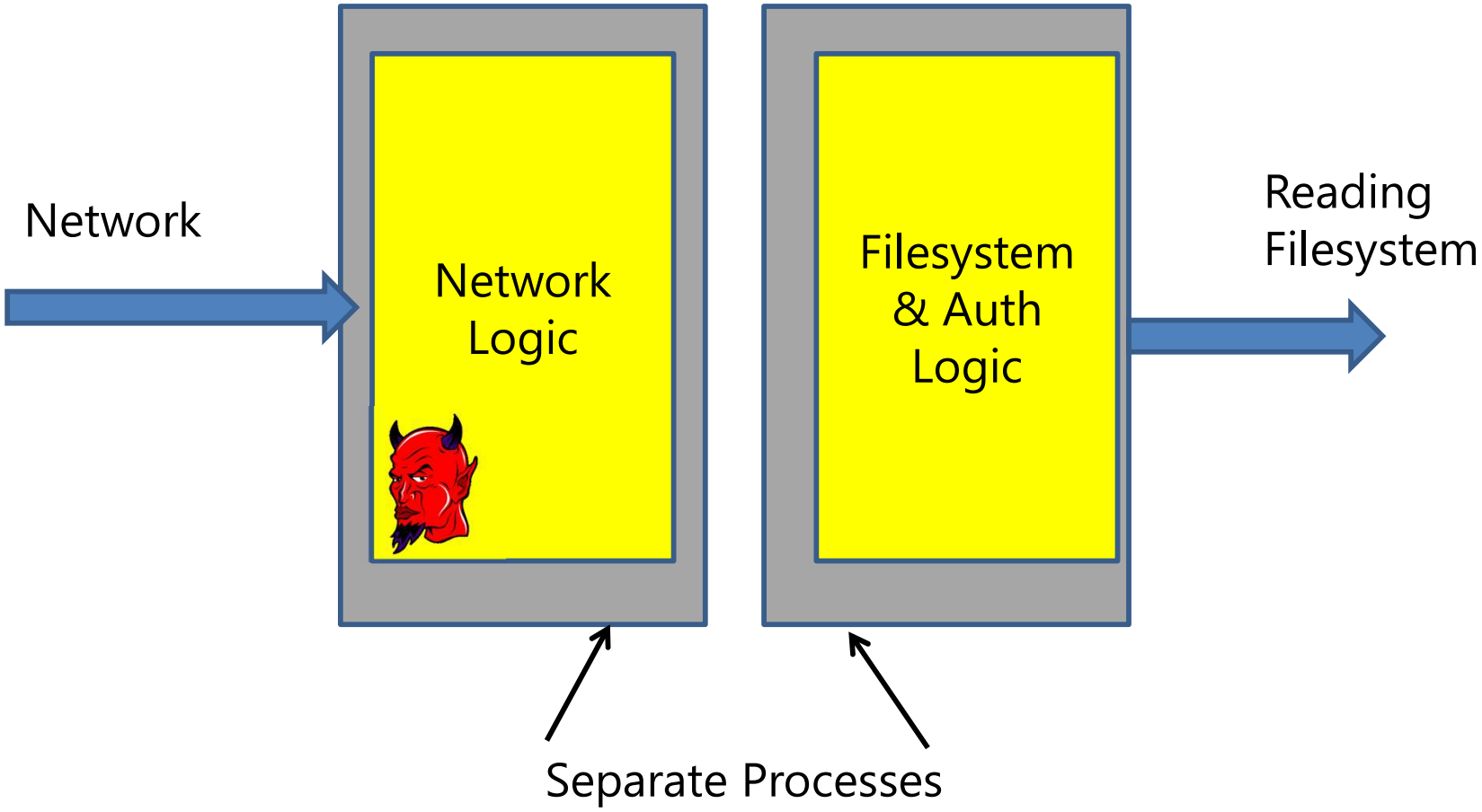
- Separation of Concerns:
  - Separate the **policy** from its **enforcement**
- Minimize Trusted Code Base (TCB)
  - Reduce what one needs to trust
  - Separate **verifier** from the **enforcement**
- Least Privilege
  - Give each component only the privileges necessary

Problem:
Bundling of Functionality

Problem:
Bundling of Functionality



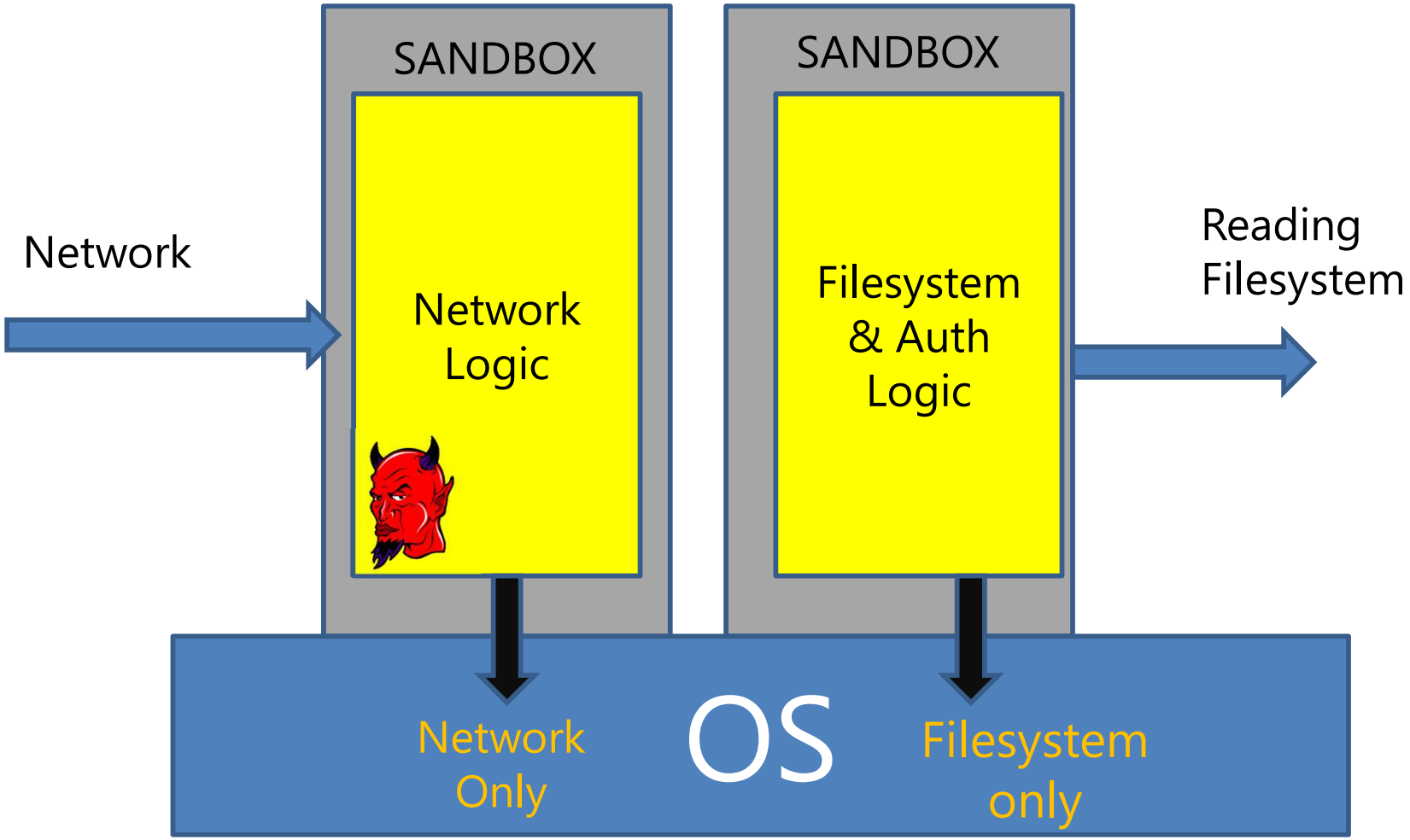
Solution: Privilege Separation



Courtesy: John Mitchell

Principle of Least Privilege

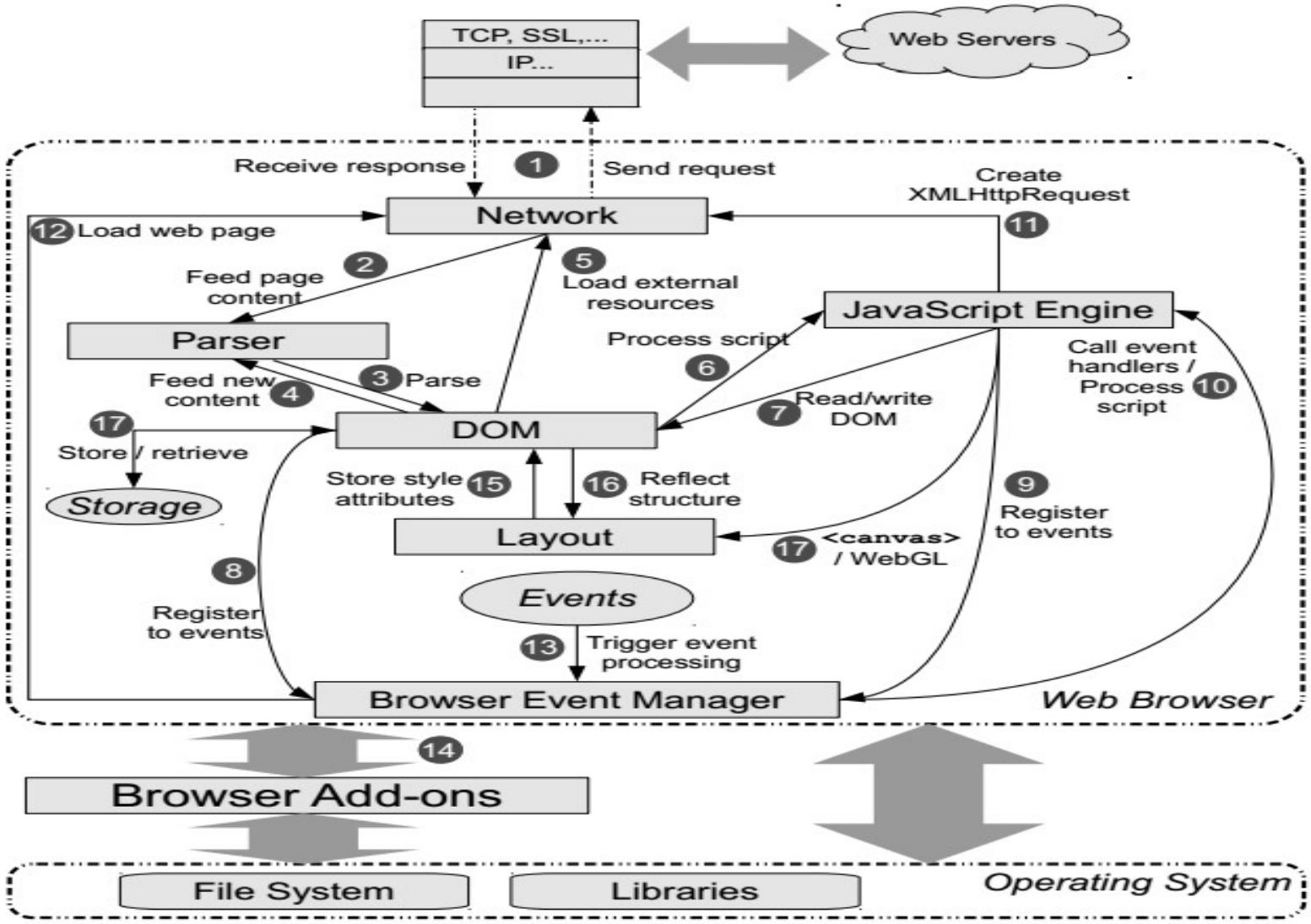
- Each compartment gets the least set of privileges it needs for its function



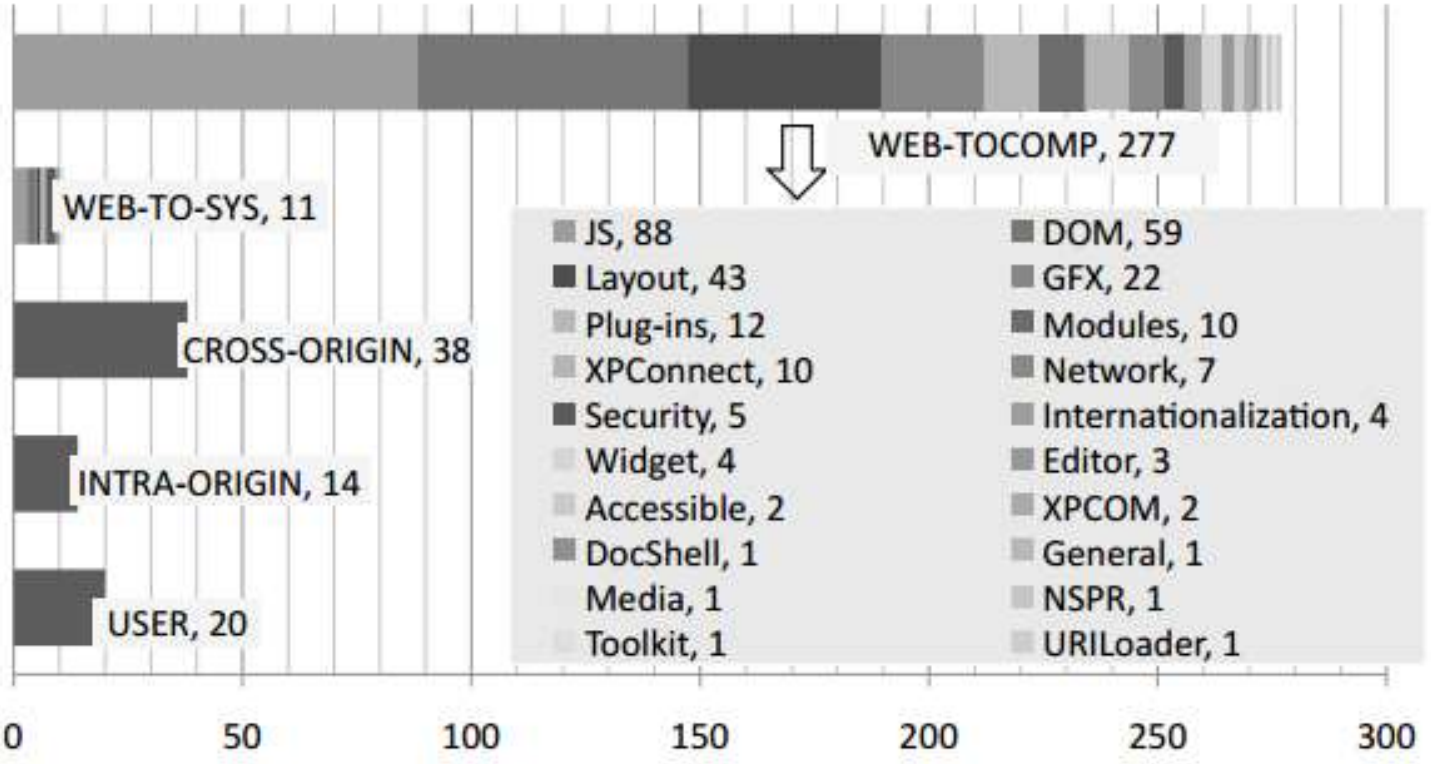
Privilege Separation Case Study (I): Design of the Web Browsers

Security Issues for Implementation

- A huge codebase (2 MLOC)
- Many languages (JS, CSS, HTML, URL, …)



Distribution of Browser Implementation Bugs



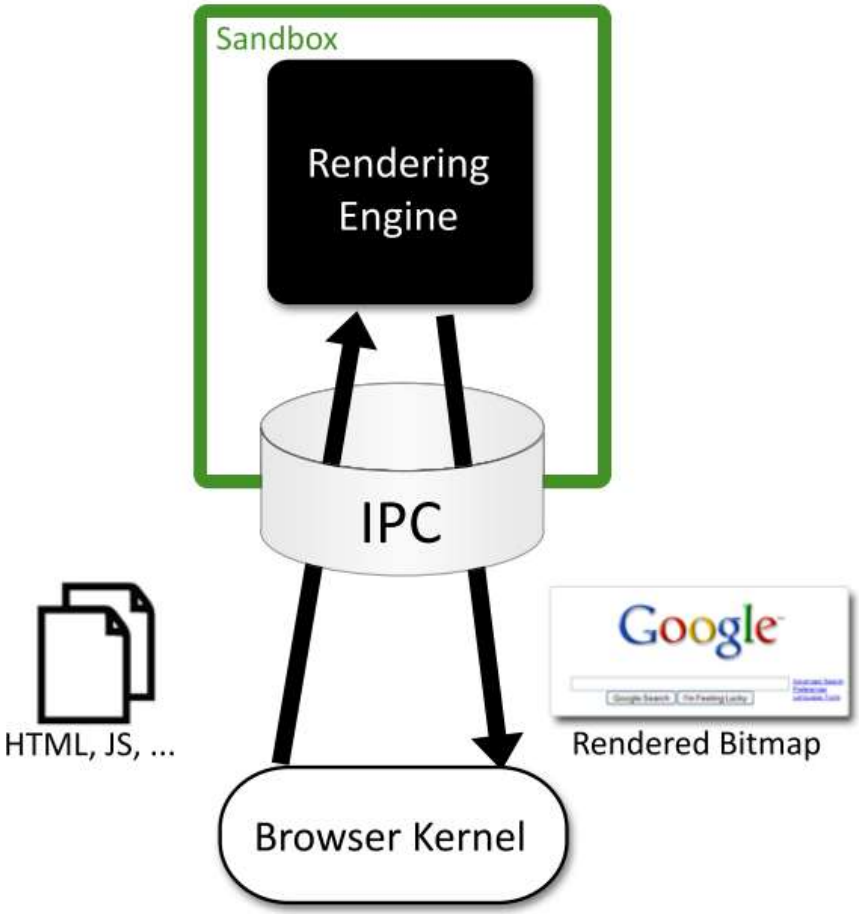
(a) Number of Historical Security Vulnerabilities in Firefox, Categorized by Severity and Firefox Components

So, what should we do?

- Auto-patching
  - E.g. Google Chrome
- Consider Firefox: Single- process
  - 1 Vulnerability leads to accessing all origins
- Solution: 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

[The Security Architecture of the Chromium Browser]

3 Security Principles

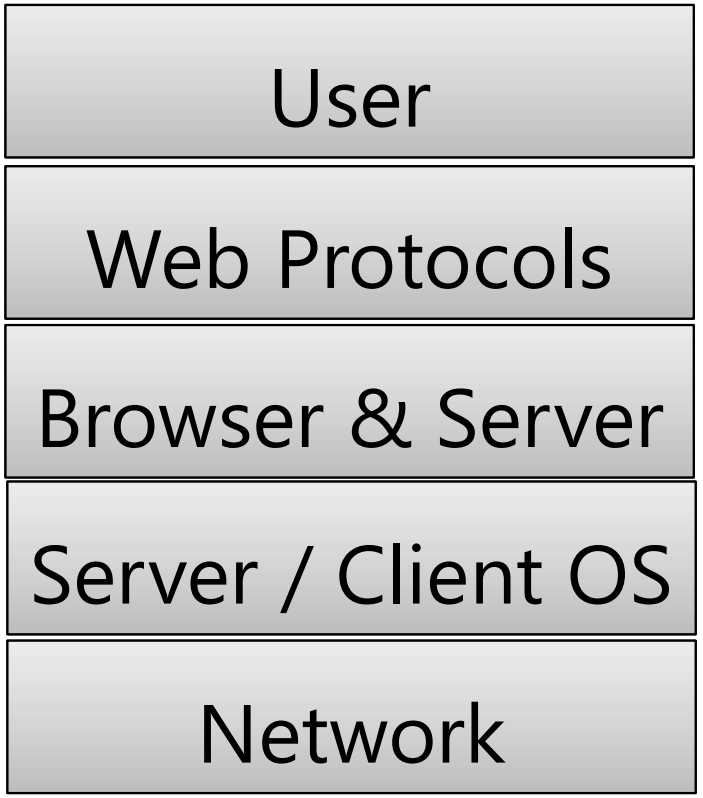
- Separation of Concerns:
  - Separate the **policy** from its **enforcement**

- 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

Wrap Up: The Key Takeaway

- Concept: A Threat Model defines:
  - Desired Security Property / Goal
  - Attacker Capabilities
  - Assumptions about the setup
- **This Module at a glance**:
  - Stack of Threats
  - Real attacks
  - Defenses
  - How to argue security!



Wrap Up…

- **The Attacker Perspective: A New Viewpoint**
- We aren't teaching you an artefact (e.g. OS, DB)
  - We're teaching you how to build it securely!

- As much about "new knowledge", as about
  - **Making sound security arguments (with threat models)**
  - **Principles and abstractions** of "secure" construction

- You learnt:
  - Basics of Crypto, web, OS, systems security
  - With 2 hands-on coding assignments!
  - How theory meets practice in computer security

Thanks!
See you next semester...

(Good luck!)