FreeBSD

Enhancing Casper Library Security

with Mandatory Access Control (MAC)

Yan-Hao Wang

National Central University, Taiwan

Who am I?

Yan-Hao Wang

- Second-year Master's Student
- FreeBSD Foundation Intern (2023): Web-based Document Editor and Command Testing
- GSoC with FreeBSD (2024): IPv6 Modernization and Cleanup on Userland Command

Sandbox

Sandbox

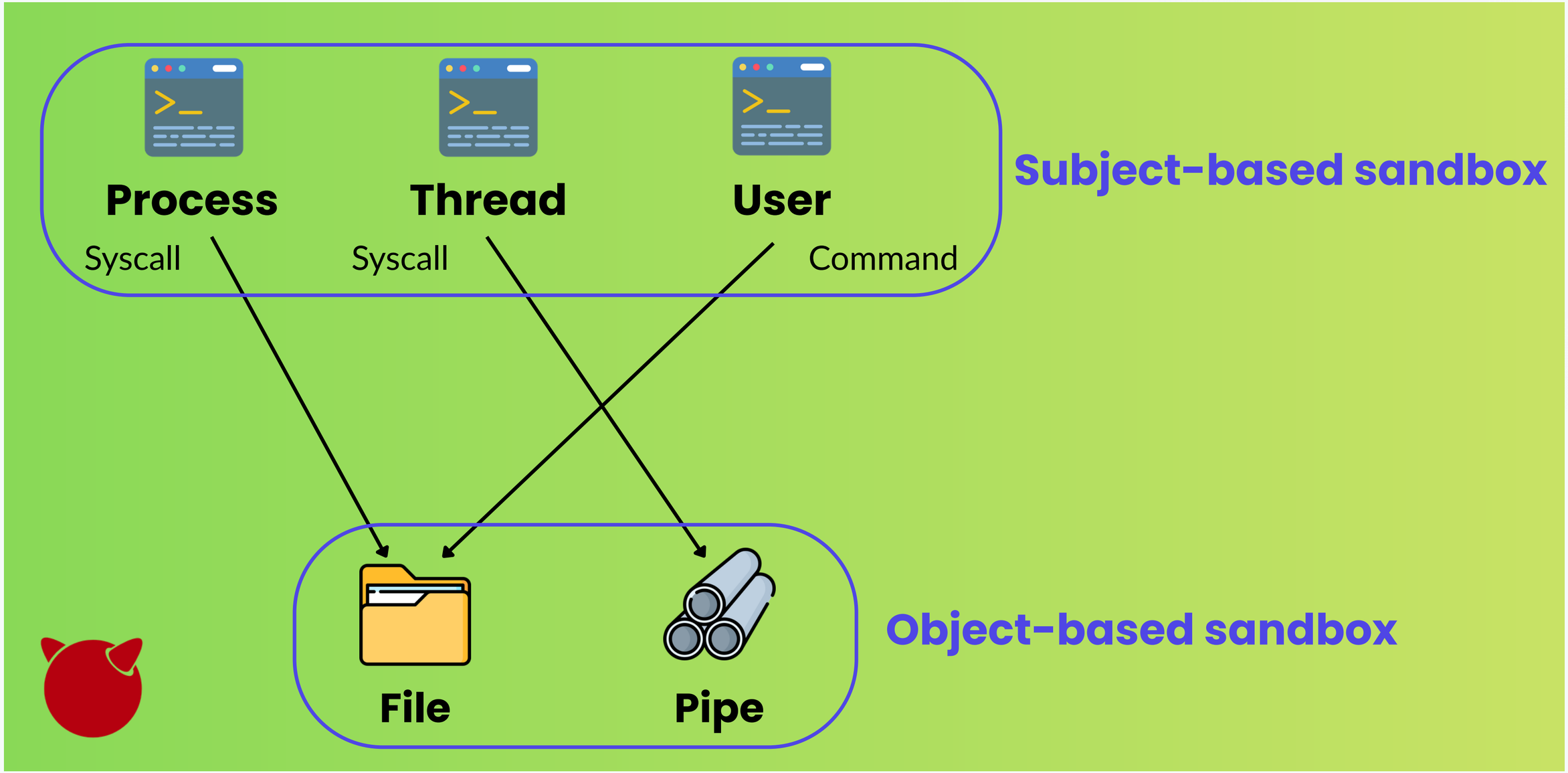
A Sandbox is an isolated environment that restricts **one’s** ability to interact the system.

Sandbox

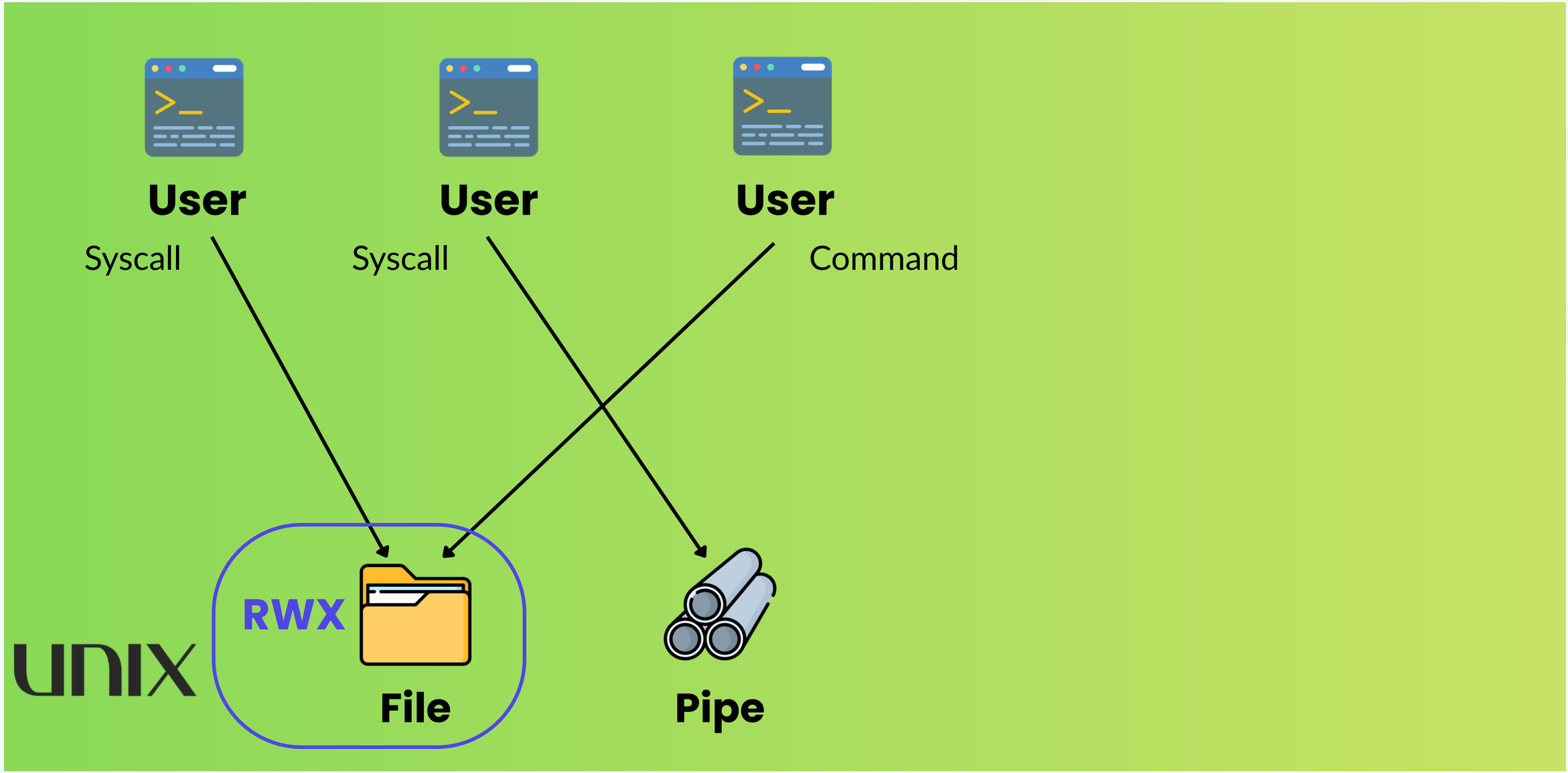
A Sandbox is an isolated environment that restricts **one's** ability to interact the system.

Based on the **Principle of Least Privilege** (Saltzer & Schroeder, 1975), ensuring each entity accesses only the resources essential for its operation.

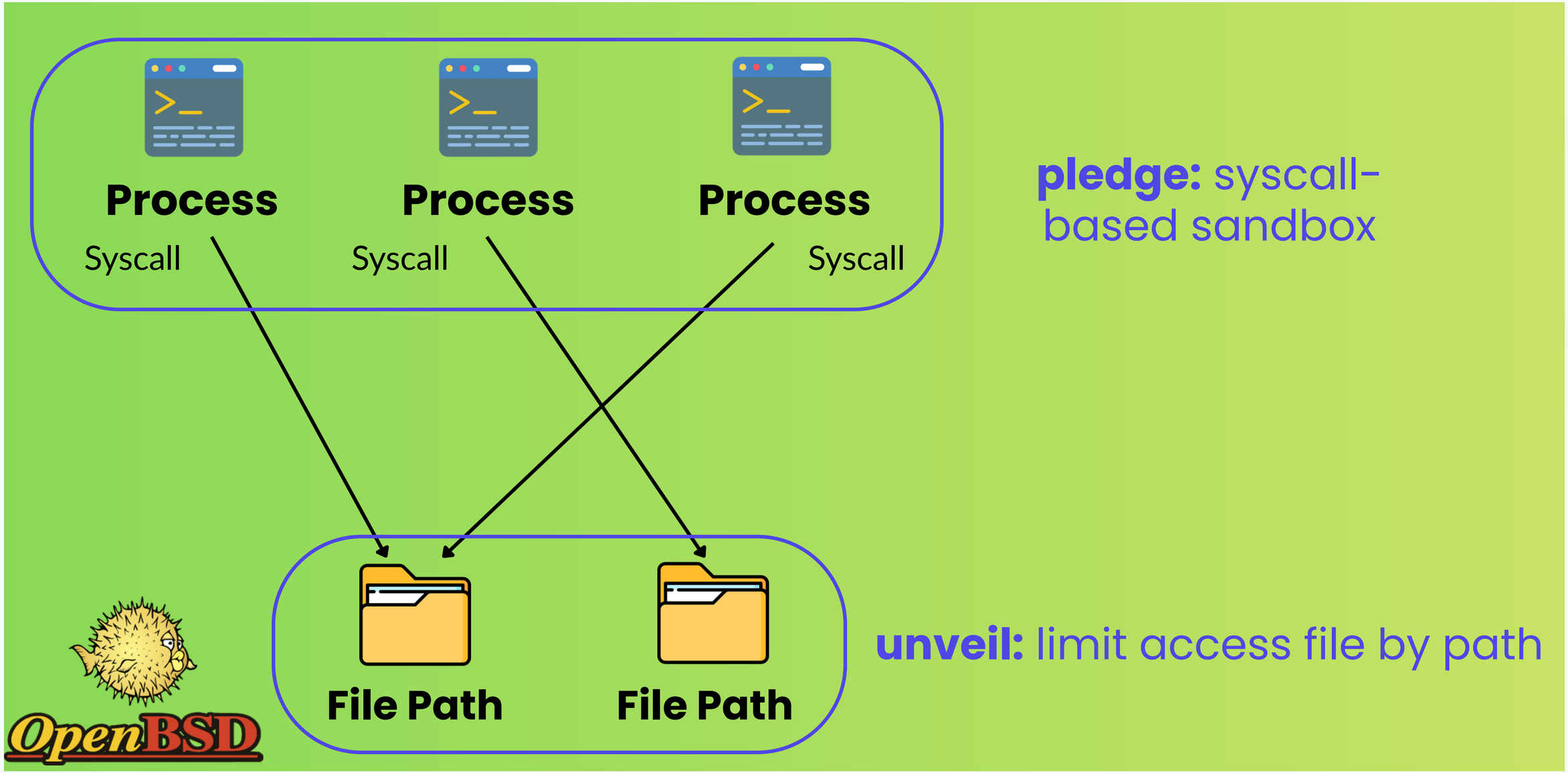
Sandbox Implement



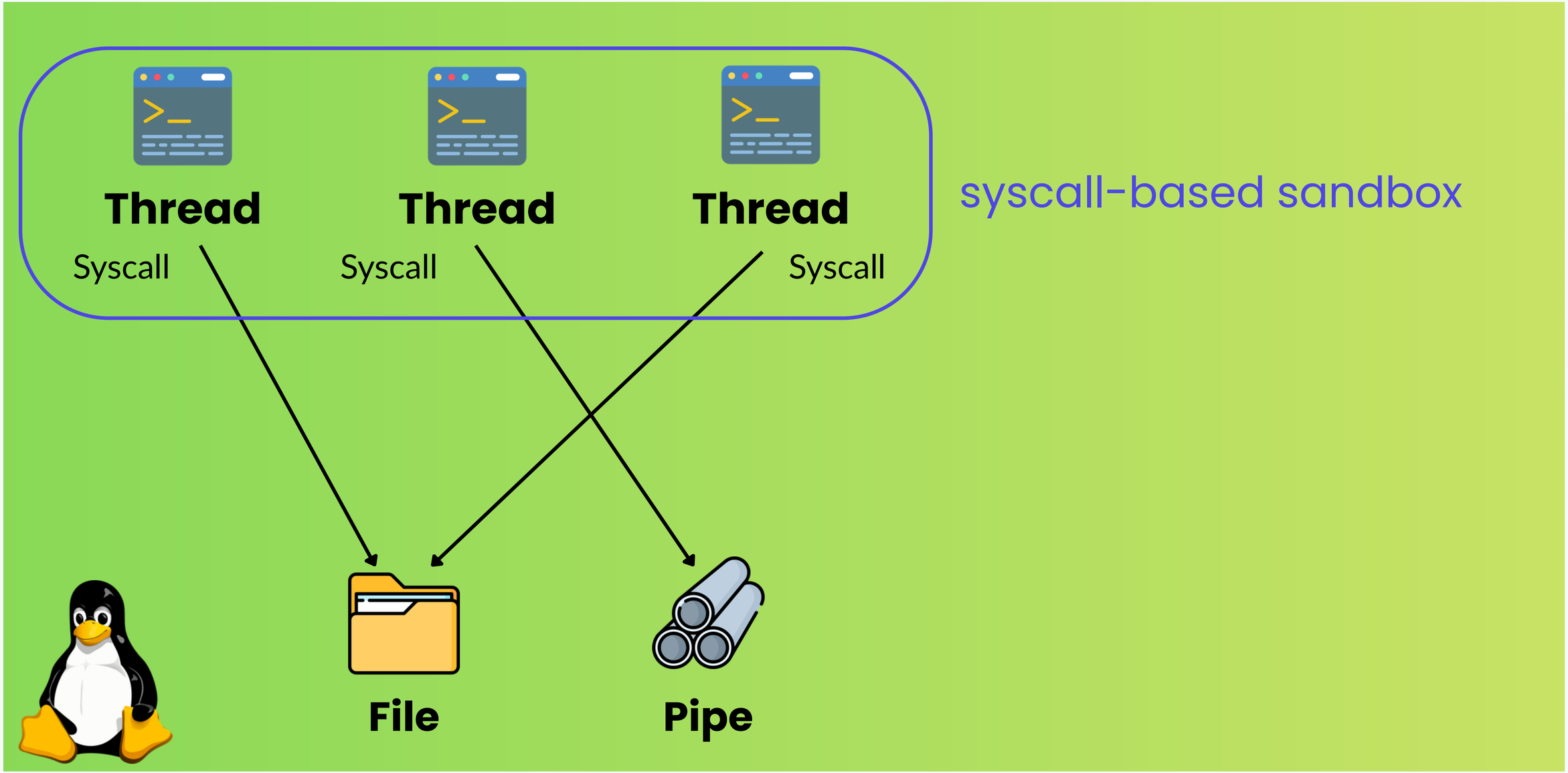
Unix DAC



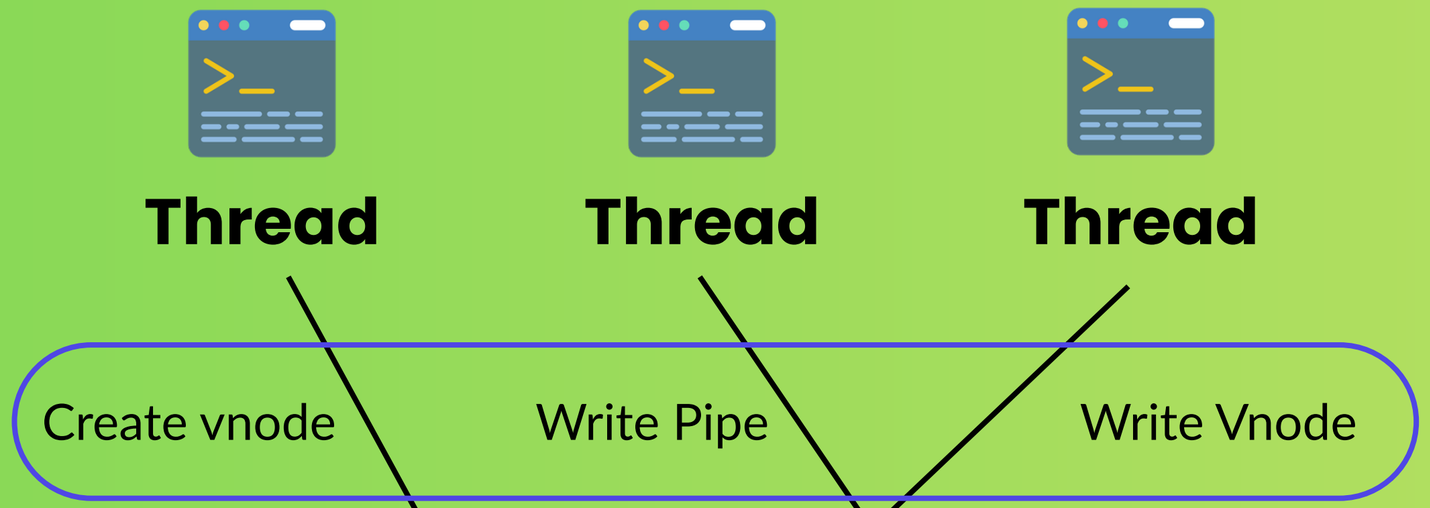
OpenBSD pledge/unveil



Linux seccomp



FreeBSD MAC



vnode

Pipe

Granular Behavior Control: Controls individual operations within a syscall.

For example: A single write syscall involves:

- Open vnode
- Write vnode

FreeBSD MAC

sys/kern/vfs_vnops.c, vn_open_vnode()

6.7.4.45. mpo_check_vnode_open

```
int mpo_check_vnode_open(struct ucred *cred, struct vn
    struct label *label, int acc_mode);
```

Parameter	Description	Locki
cred	Subject credential	
vp	Object; vnoc	
label	Policy label f	
acc_mode	open(2) acc	

```
#ifdef MAC
        if ((fmode & O_VERIFY) != 0)
                accmode |= VVERIFY;
        error = mac_vnode_check_open(cred, vp, accmode);
        if (error != 0)
                return (error);

        accmode &= ~(VCREAT | VVERIFY);
#endif
```

```
struct thread {
    LIST_HEAD(, turnstile) td_contested;    /* (q) Contested locks. */
    struct lock_list_entry *td_sleeplocks; /* (k) Held sleep locks. */
    int     td_intr_nesting_level; /* (k) Interrupt recursion. */
    int     td_pinned;  /* (k) Temporary cpu pin count. */
    struct ucred    *td_realucred;  /* (k) Reference to credentials. */
    struct ucred    *td_ucred;  /* (k) Used credentials, temporarily switchable. */
```

Sandbox Compariosn

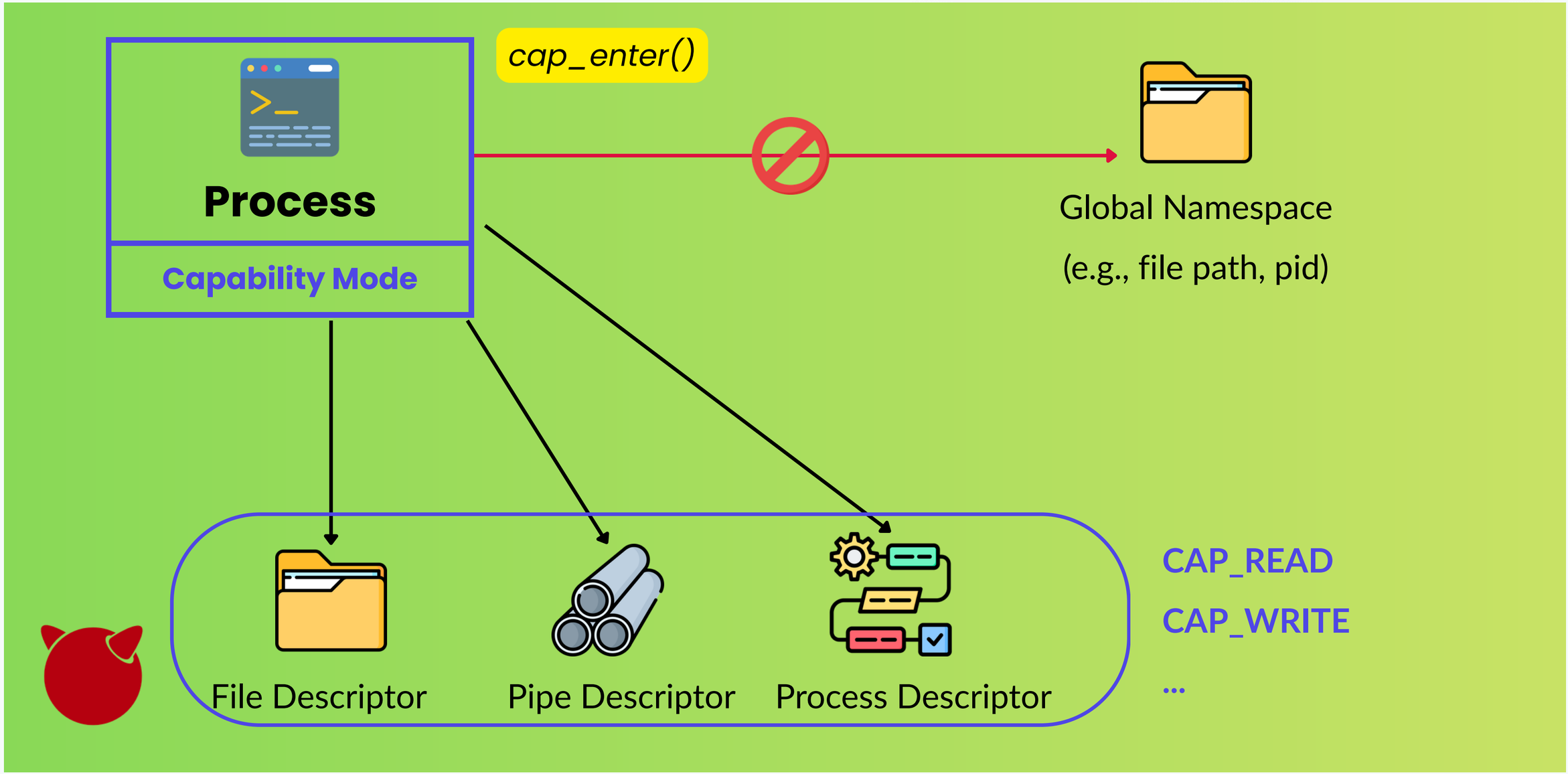
Mechanism	Focus (Subject/Object)	Enforcement Basis	Advantages & Disadvantages
Unix DAC	Object-based	File Permissions (RWX)	Pros: Extremely simple. Cons: Too coarse-grained; lacks process-level isolation.
OpenBSD pledge/unveil	Both	Syscalls & File Paths	Pros: Highly developer-friendly. Cons: Operates at a high level; limited internal granularity.
Linux seccomp	Subject-based	Syscalls	Pros: Strong syscall filtering.Cons: "All-or-nothing" approach; lacks awareness of the target object.
FreeBSD MAC	Both	Kernel Object Operations	Pros: Deepest granularity; highly powerful. Cons: Complex policy writing; typically managed by system admins.

Capsicum and Casper

Capsicum

- **Entering Capability Mode**: Programs must explicitly enter a restricted state called Capability Mode.
- **Default Deny:** Once in Capability Mode, all global operations are denied by default. This includes:
  - No new file descriptors can be opened (via path).
  - No new network connections can be created.
- Only can acccess the descriptor you have already have and the capability you have.

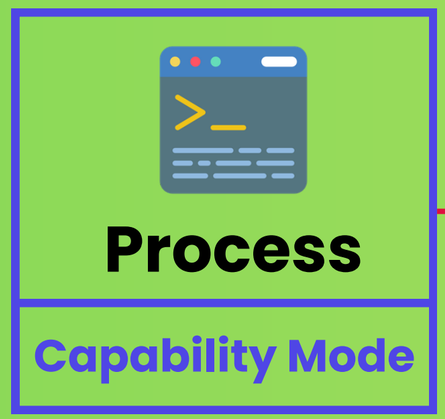
FreeBSD Capsicum



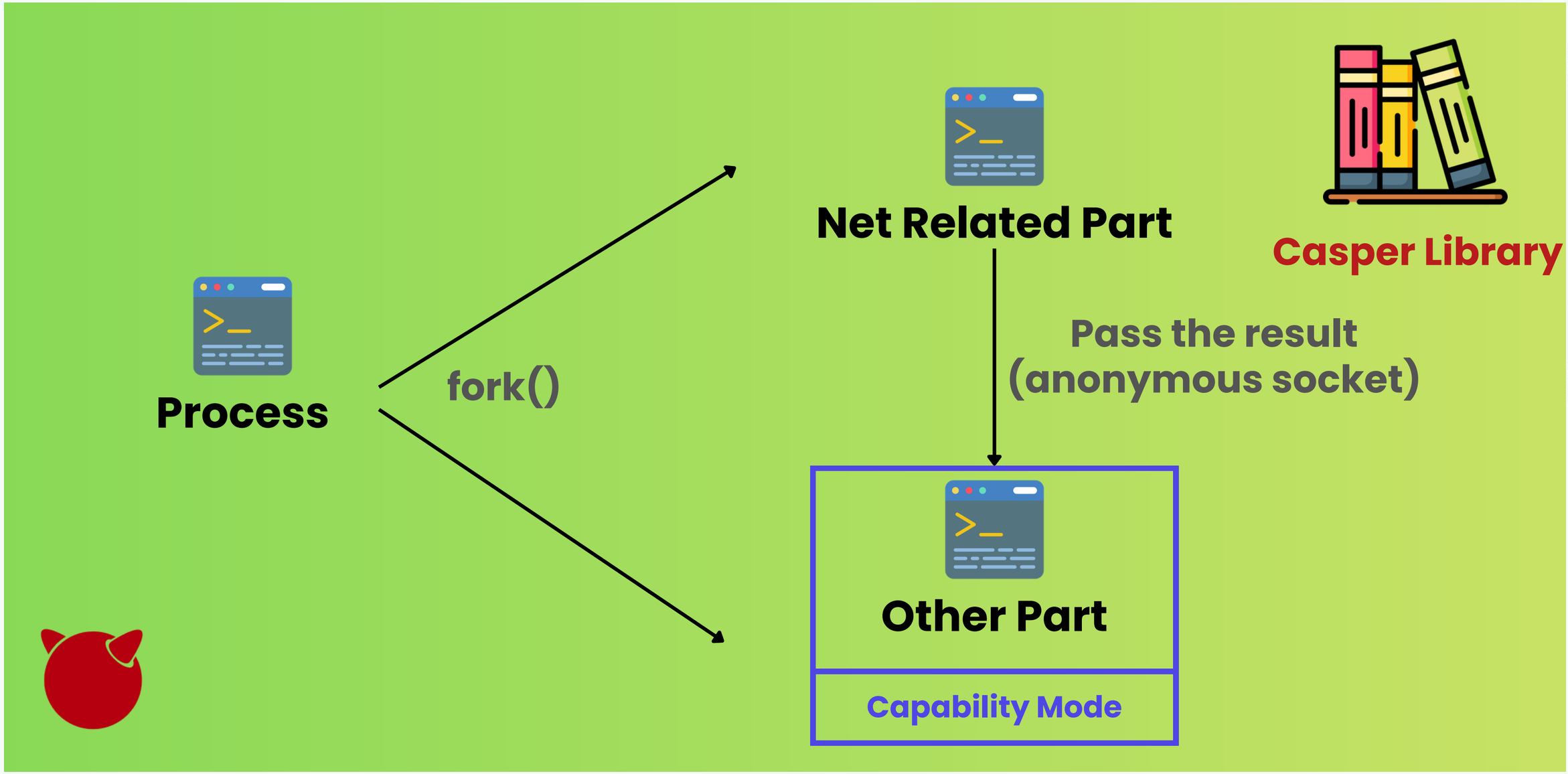
Capsicum Design

Network-related Operation

For example, connect(), because it introduces an uncontrollable state into the sandbox.



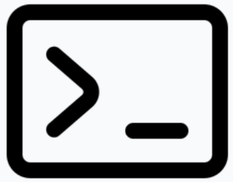
Capsicum Design



The Casper Library

Architectural Security Gaps

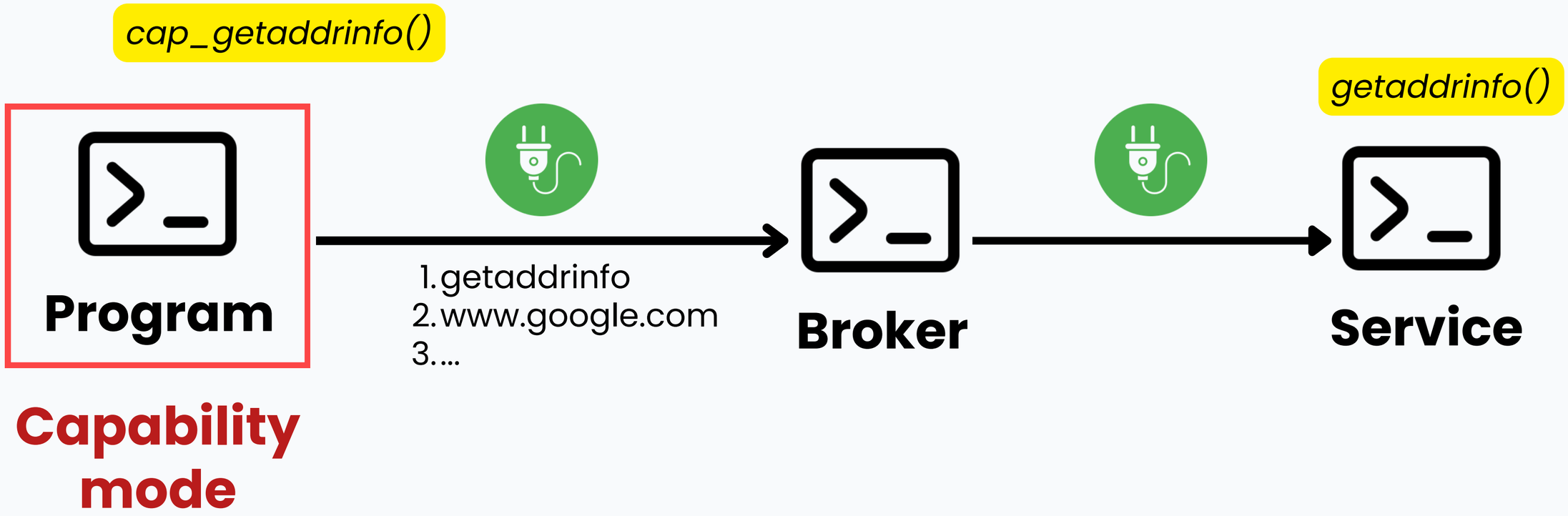
Casper Arch



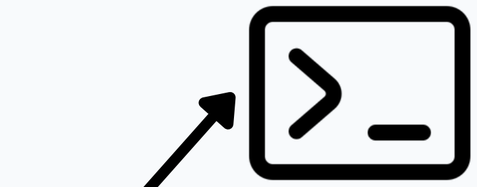
cap_init()

Program

Casper Arch


cap_getaddrinfo()
getaddrinfo()
Program
1.getaddrinfo
2.www.google.com
3....
Broker
Service
Capability mode

Casper Arch



DNS Service

```
system.dns      provides libc compatible DNS API
system.fileargs
                provides an API for opening files specified  on  a
                command line
system.grp      provides a getgrent(3) compatible API
system.net      provides a libc compatible network API
system.netdb    provides libc compatible network proto API
system.pwd      provides a getpwent(3) compatible API
system.sysctl   provides a sysctlbyname(3) compatible API
system.syslog   provides a syslog(3) compatible API
```

vice

Pro

Capability mode

Fileargs Service

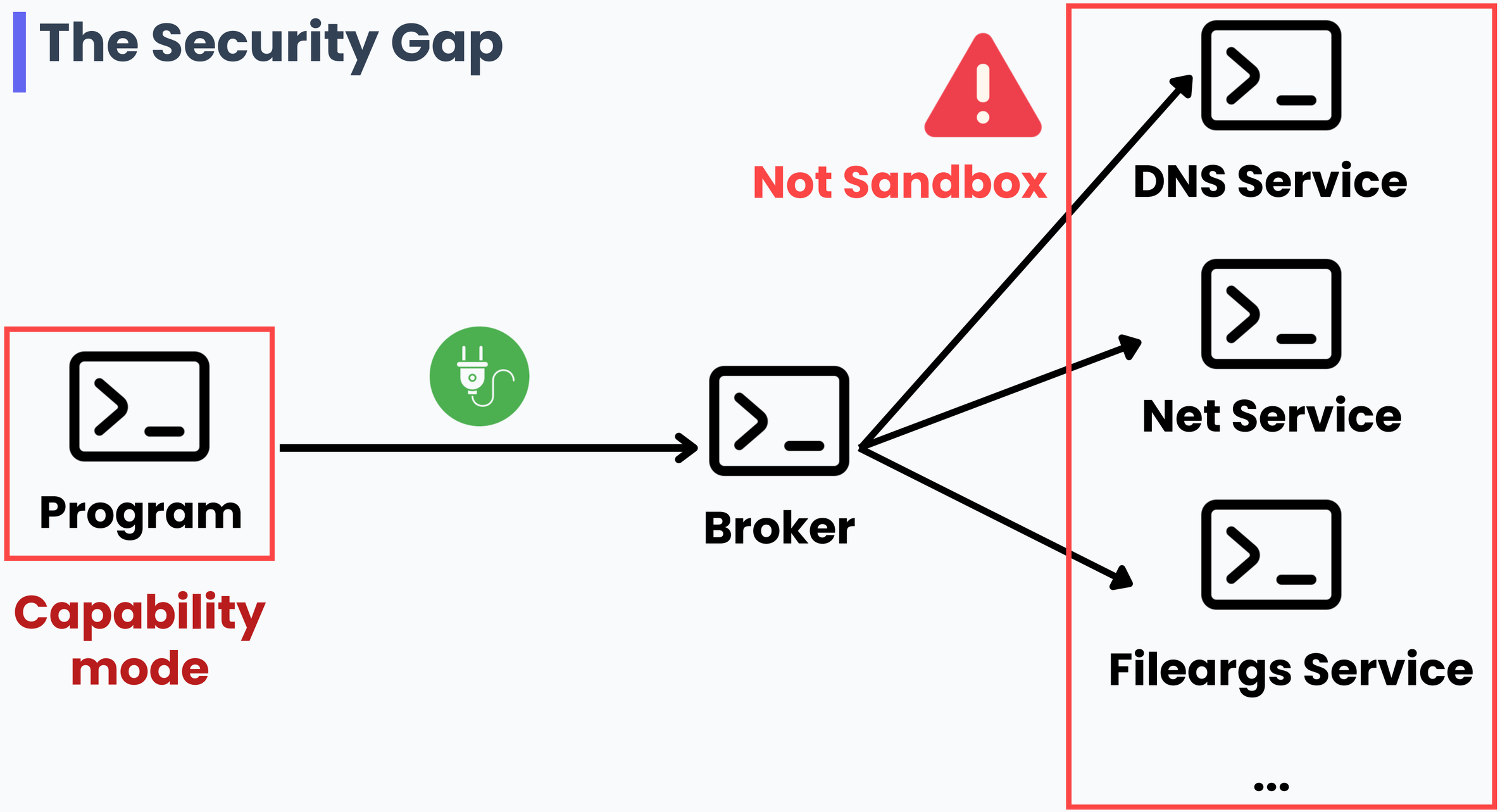
...

Casper

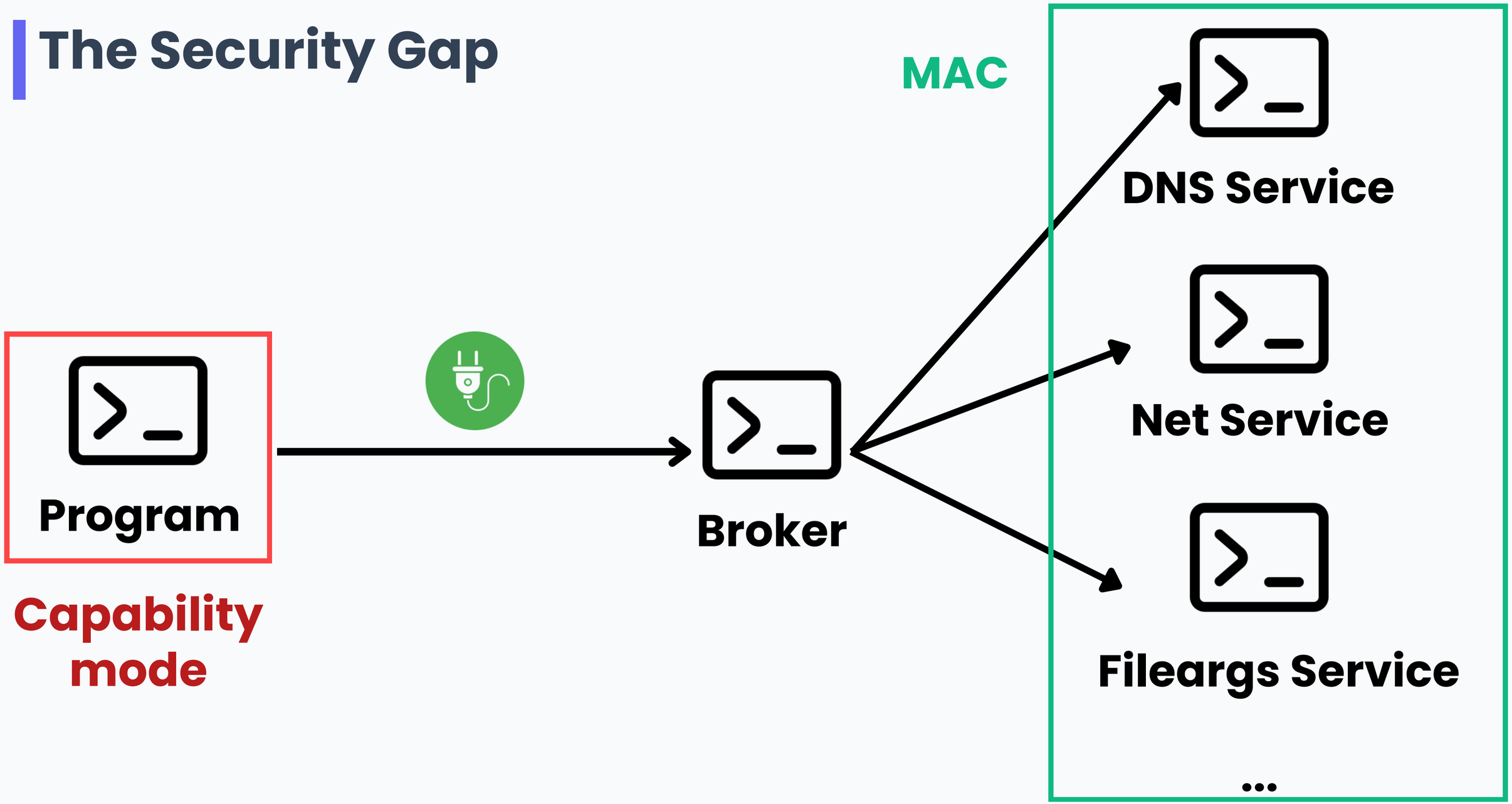
Casper help:

- **Automates Process Delegation**: Handles the complexity of forking and managing service processes for sandboxed applications.
- **Capability-Oriented Services**: Provides a set of standardized services (like DNS, system sysctl, or file access) designed specifically for Capsicum.
- **Reduces Attack Surface**: By standardizing the implementation, it prevents common security pitfalls that occur when developers write their own "helper" processes.

The Security Gap



The Security Gap



Methodology

Designing the MAC Sandbox for Casper

Why MAC for Casper?

Why MAC

The Problem with Application-Level MAC:

- General applications have diverse and unpredictable execution paths.
- Writing and maintaining a global policy for every unique application is complex and unscalable.

Why MAC

The Casper:

- Deterministic Behavior: As an OS system library, Casper has a strictly defined and predictable set of operations.
- Fixed Scope: Casper services (like DNS or sysctl) perform specific tasks that do not change based on user input.

Scope of Casper

Tracing with DTrace

We used DTrace to accurately identify the minimal privilege set needed by each Casper helper.

- Dynamic instrumentation of syscalls
- Kernel interaction monitoring
- Mapping behaviors to MAC hooks

MAC Support Operation

1. socket: socket operations
2. vnode: vnode operation
3. sysctl: sysctl operation
4. cred: credential and authorization operations
5. pipe: pipe operations
6. posixshm: POSIX shared memory
7. ....

MAC Support Operation

1. socket: socket operations
2. vnode: vnode operation
3. ~~sysctl: sysctl operation~~
4. ~~cred: credential and authorization operations~~
5. ~~pipe: pipe operations~~
6. ~~posixshm: POSIX shared memory~~
7. ....

DNS Service

MAC Support Operation

1. ~~socket: socket operations~~
2. vnode: vnode operation
3. sysctl: sysctl operation
4. ~~cred: credential and authorization operations~~
5. ~~pipe: pipe operations~~
6. ~~posixshm: POSIX shared memory~~
7. ....

Sysctl Sercice

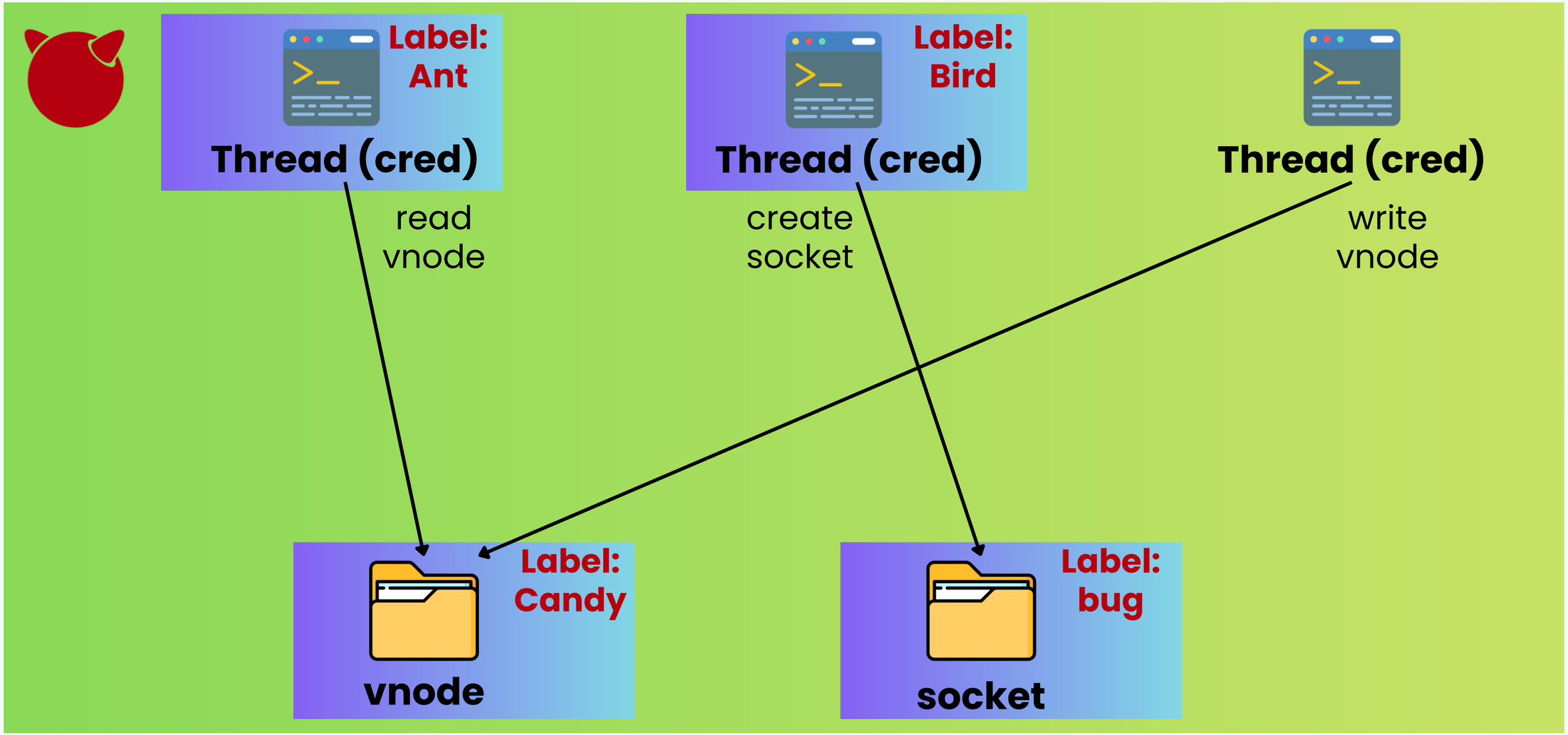
MAC Support Operation

1. socket: socket operations
2. vnode: vnode operation
3. ~~sysctl: sysctl operation~~
4. ~~cred: credential and authorization operations~~
5. ~~pipe: pipe operations~~
6. ~~posixshm: POSIX shared memory~~
7. ....

DNS Service can only open vnode realted to:

1. /etc/nsswitch.conf
2. /etc/hosts
3. /etc/resolv.conf
4. /etc/services

MAC Label


Label:
Ant
Thread (cred)
Label:
Bird
Thread (cred)
Thread (cred)
read
vnode
create
socket
write
vnode
Label:
Candy
vnode
Label:
bug
socket

ENFORCEMENT VIA MAC LABELS

Process Labels

Dynamic labels assigned by the Broker when forking specific service helpers.

Vnode Labels

Persistent filesystem labels assigned via rc.d scripts at boot time.

Deny-by-Default

All privileges not explicitly allowlisted are rejected at the kernel layer.

How to Use

Kernel Modules & Library Patches

DEPLOYMENT & REPRODUCIBILITY

- Enable **MAC Multilabel** on target filesystem
- Build and load the **kernel MAC module**
- Patch **libcasper** do process label
- Initialize persistent labels in /etc/mac.conf

```
# Registering Casper Label
default_labels {
  ?casper
}
```

Effectiveness Evaluation

Security Robustness

Simulate a Compromise

Category	Test Case	Action	Targeted Kernel Object
Process control	ATTACK_EXEC	execve	proc / vnode
File system abuse	ATTACK_FILE_READ	open (read-only)	vnode
File system abuse	ATTACK_FILE_WRITE	open (write)	vnode
Credential access	ATTACK_CRED	getuid / setuid	cred
Network misuse	ATTACK_NET	socket / connect	socket
IPC abuse	ATTACK_IPC	ptrace / shm_open	proc / ipc
Kernel primitives	ATTACK_KLD	kldload	kernel
Kernel primitives	ATTACK_SYSCTL	sysctlbyname	kernel

Result: All unauthorized operations were successfully rejected.

Performance Evaluation

Performance Benchmarks

Evaluation Setup

ARM64 Platform

Raspberry Pi 4B

Cortex-A72 CPU

8 GB RAM

AMD64 Platform

AMD Ryzen 5 5600G

12 Logical Cores

16 GB RAM

Evaluation Setup

Enviroment

FreeBSD 15 (NO-DEBUG)

Turn off SMP

Methodology

KDE-based modes

clock_gettime()

Syscall Overhead

Three Configurations:

- Baseline: Standard FreeBSD (No MAC).
- No-Label: MAC module loaded, but no security labels applied.
- Labeled: Full MAC enforcement with specific service policies.

Syscall Overhead

- Open/Close: ~5.5% additional latency.
- Socket Create: ~3% (ARM) to 6.5% (AMD) overhead.
- Sysctl: ~12% to 15% overhead. The special one may

Why sysctl is larger?

- Originally `sysctl` is so fast
- Fixed MAC check result in a large relative percentage increase

Function Ovehread

- **Function Measurement**: We evaluated the latency of individual Casper service functions.
- **Overall Trend**: Most functions show less than **6%** overhead compared to the baseline.
- The maximum overhead observed was approximately **8%** for a single specific function.
- **Consistency**: Results are consistent across both ARM64 and AMD64 platforms.

Network Throughput (QPS)

Metric: Queries Per Second (QPS) — measuring the maximum load the service can handle.

- Test Duration: 60-second sustained load.
- Impact on Throughput:
  - ARM64: Minimal decrease of 2–4%.
  - AMD64: Even lower impact, less than 3%.

Throughput Impact

<3%

Total QPS Reduction

Conclusion

Strengthening the FreeBSD Ecosystem

Conclusion

Sandboxed Casper

Addresses security gaps in Casper's normal-mode execution.

Least Privilege

Successfully applies MAC-enforced domains to system library.

Efficiency

Minimal overhead (< 8% typically) across dual architectures.

Thank You!

Questions and Feedback are Welcome

bses30074@gmail.com | lwhsu@FreeBSD.org

Q & A

Open Discussion
https://github.com/Wang-Yan-Hao/small_thing