PeTAL: Ensuring Access Control Integrity against Data-only Attacks on Linux

Juhee Kim, Jinbum Park, Yoochan Lee,
Chengyu Song, Taesoo Kim, Byoungyoung Lee

Samsung Research

UC RIVERSIDE

Georgia Institute of Technology

Linux kernel is an attractive attack target

- Widely used
  - Mobile devices, Servers, and IoT devices
- Increasing number of vulnerabilities and exploit techniques

Vulnerabilities by types/categories

Year	Overflow	Memory Corruption	Sql Injection
2014	18	31	0
2015	13	17	0
2016	36	76	0
2017	62	86	0
2018	32	70	0
2019	30	124	0
2020	10	40	0
2021	18	54	0
2022	41	149	0
2023	19	166	0
2024	31	630	0
Total	310	1443	

CVEdetails.com

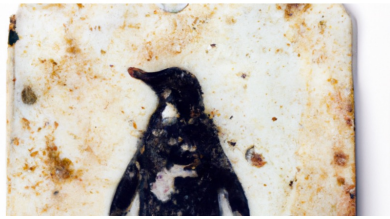
Thursday, November 21, 2019

Bad Binder: Android In-The-Wild Exploit

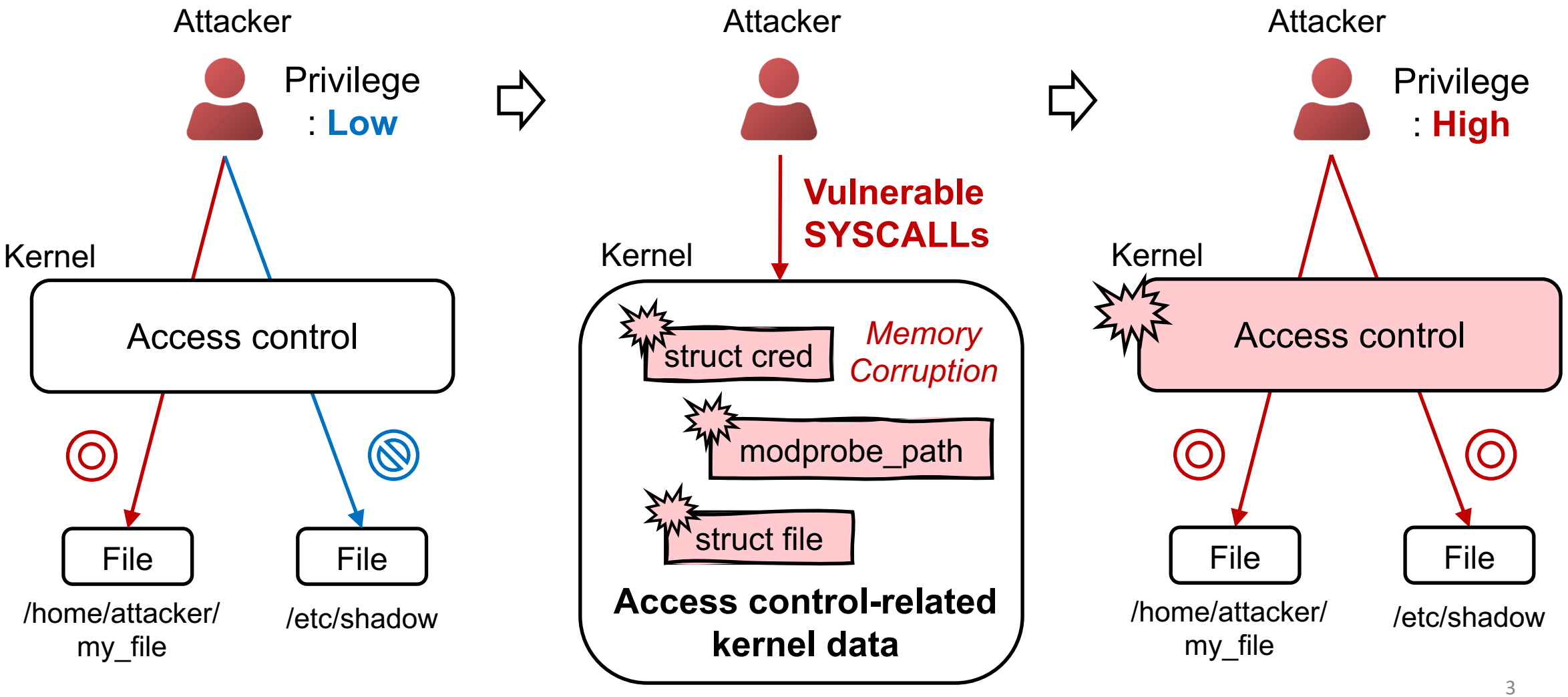
Posted by Maddie Stone, Project Zero

DirtyCred: Escalating Privilege in Linux Kernel

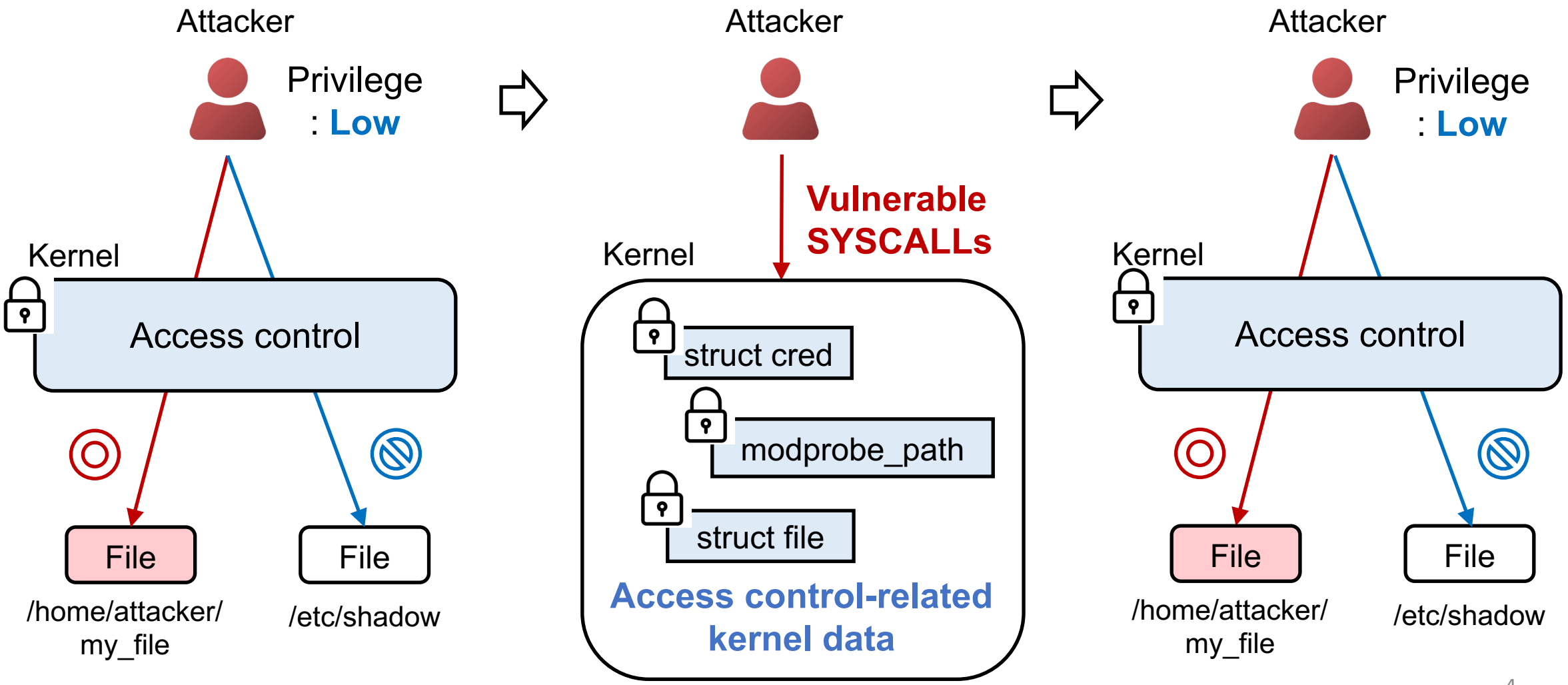
DirtyCred is a kernel exploitation concept that swaps unprivileged kernel credentials with privileged ones to escalate privilege. Instead of overwriting any critical data fields on kernel heap, DirtyCred abuses the heap memory reuse mechanism to get privileged. Although the concept is simple, it is effective. See the Blackhat presentation or CCS paper for more details.



Kernel Privilege Escalation Attacks



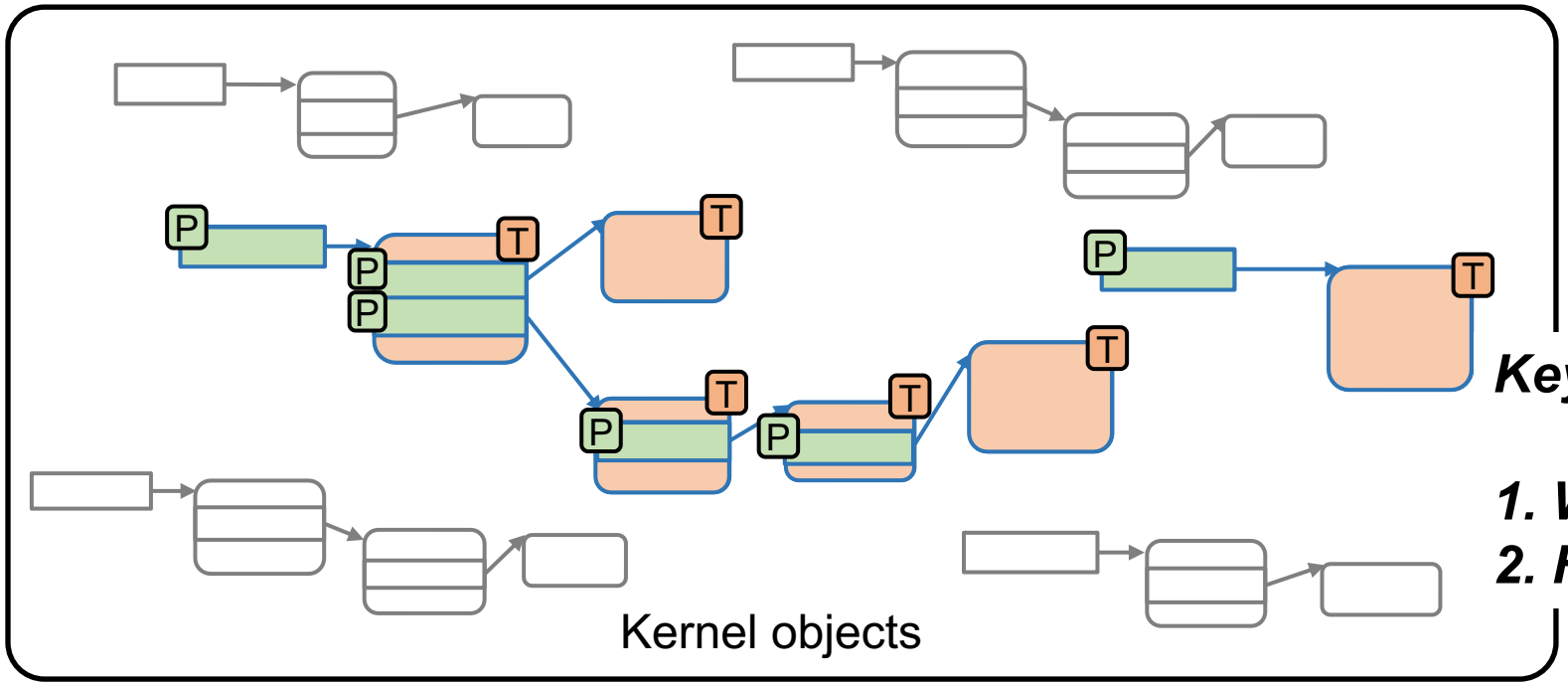
Protecting Access Control to Prevent Attacks



PeTAL's Data Flow Integrity

DFI is known to be performance-heavy

- **Selectively** protect ***access control-related data***
- Leverage **hardware extensions** : **ARM MTE** for objects, **PAC** for pointers



Key questions:

1. What should be protected?
2. How should they be protected?

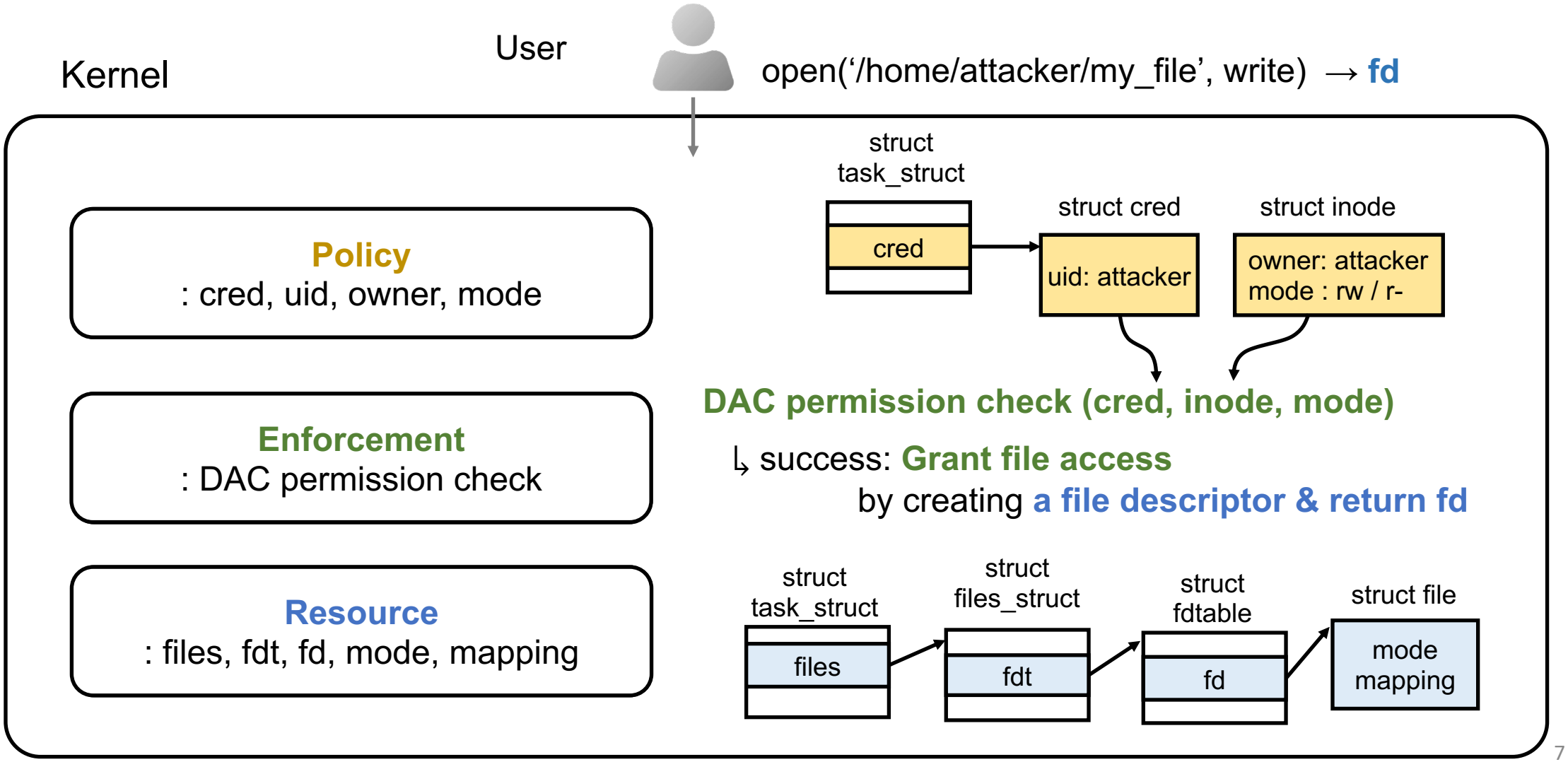
Access Control System

Policy
: Data defining the allowed access

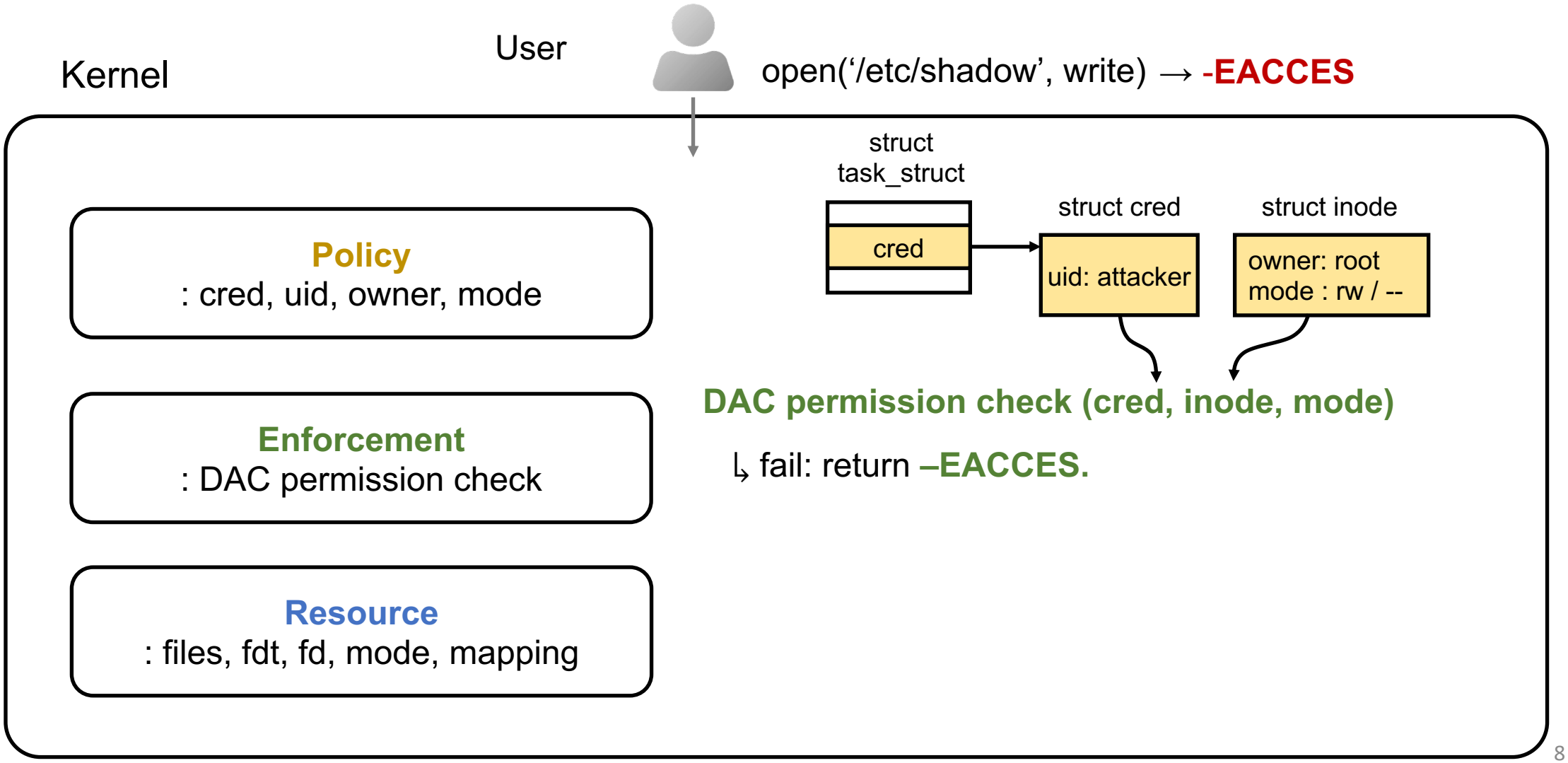
Enforcement
: Code enforcing the access control

Resource
: Data being protected

Access Control System in the Linux Kernel



Access Control System in the Linux Kernel



Attack 1: Corrupting Policy

Kernel

Attacker

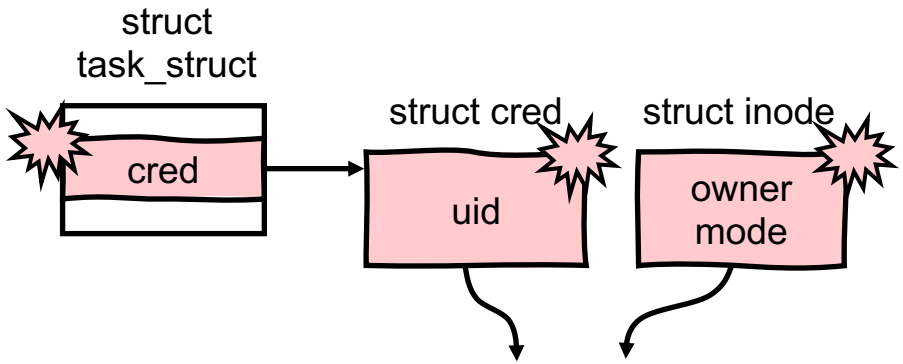
Vulnerable SYSCALLs

open('/etc/shadow' , write) → **fd**

Policy
: cred, uid, owner, mode

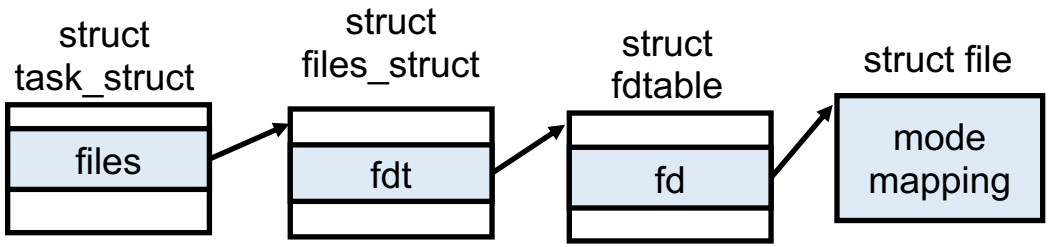
Enforcement
: DAC permission check

Resource
: files, fdt, fd, mode, mapping

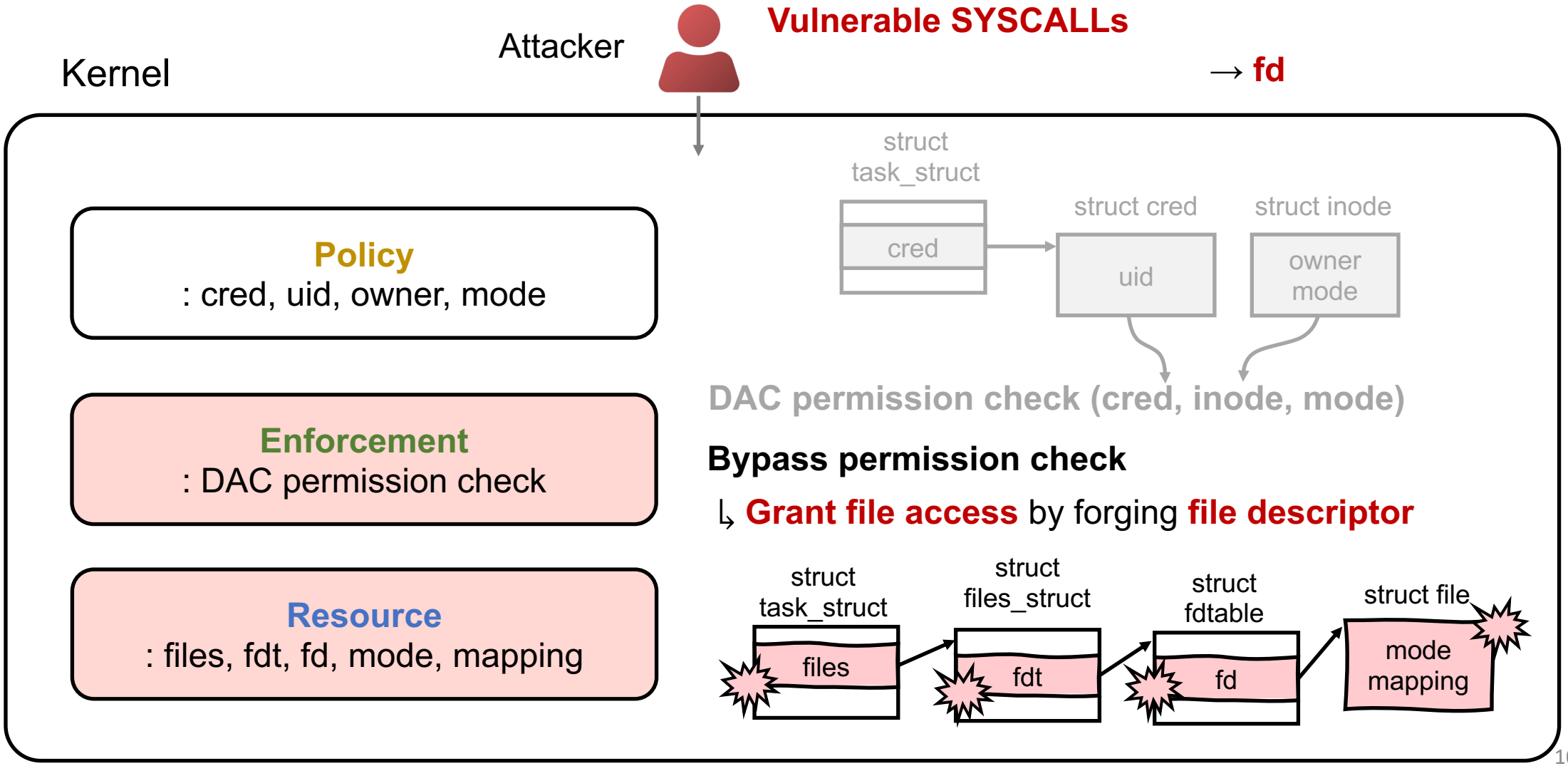


DAC permission check (cred, inode, mode)

↳ success: **Grant file access**
by creating **a file descriptor & return fd**



Attack 2: Bypassing Enforcement & Corrupting Resource



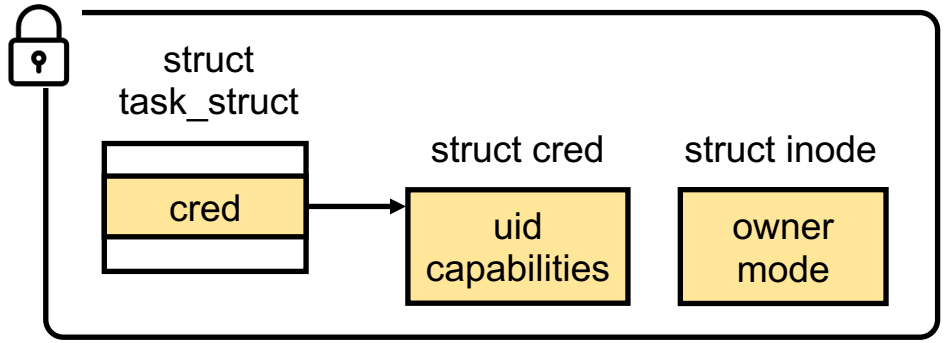
PeTAL's Access Control Integrity

1. What should be protected?

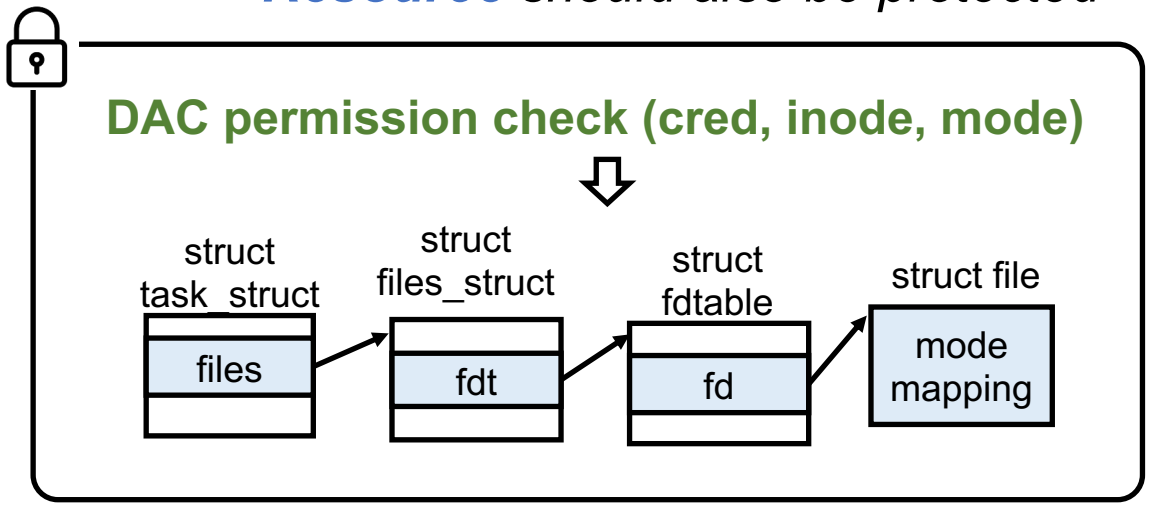
Policy Integrity
: Ensure **policy** is not corrupted

Complete Enforcement
: Ensure **enforcement** is always enforced when **resource** is accessed

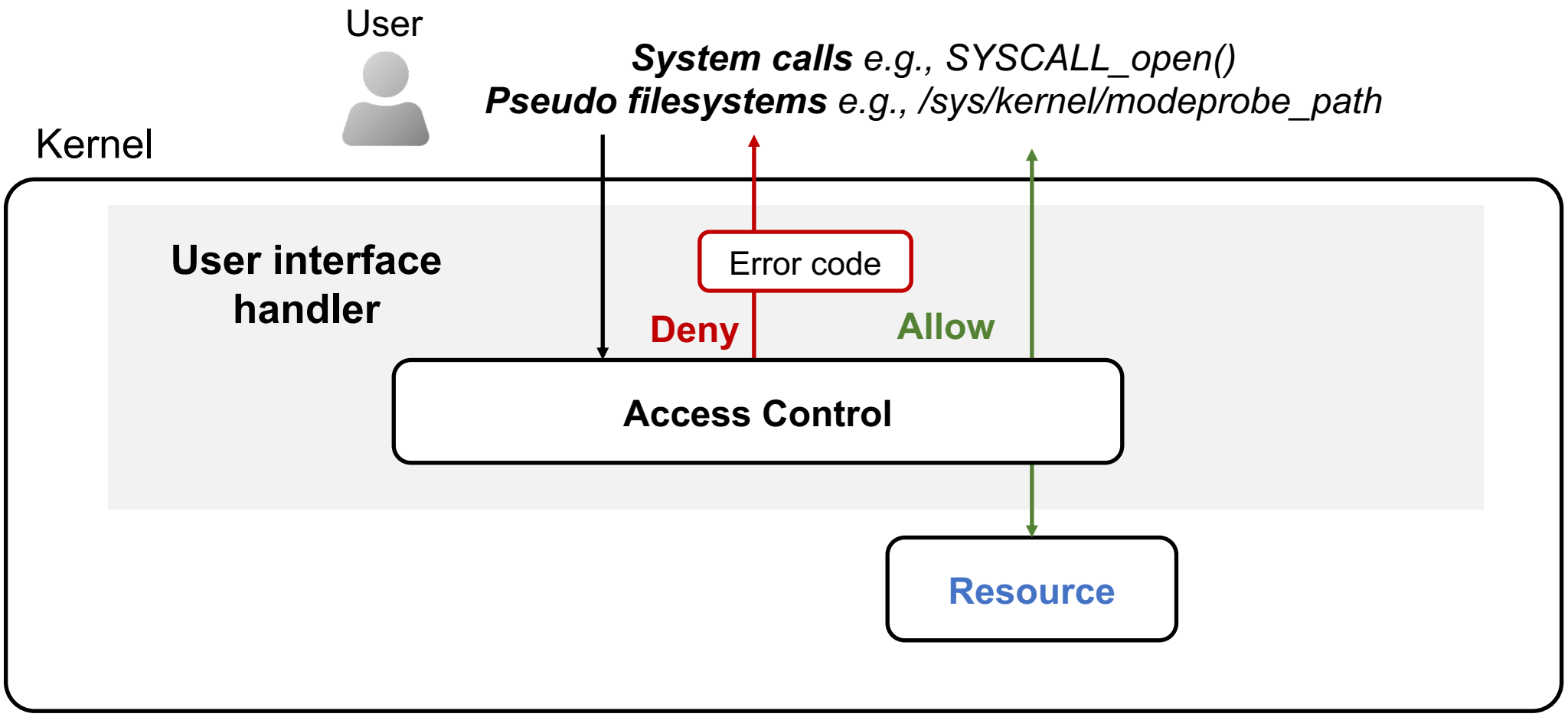
Policy *should be protected*



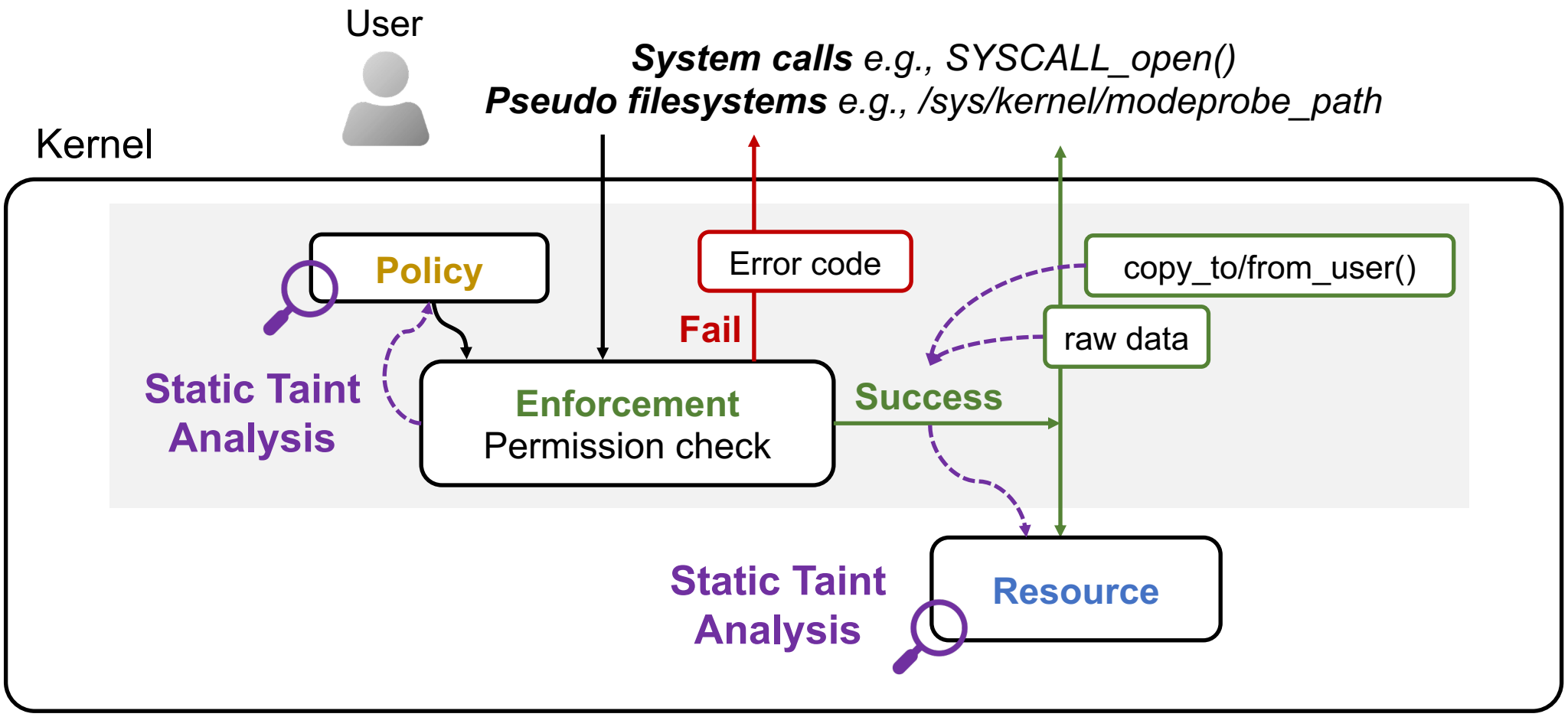
Resource *should also be protected*



Collecting Policy and Resource
from **user interfaces**



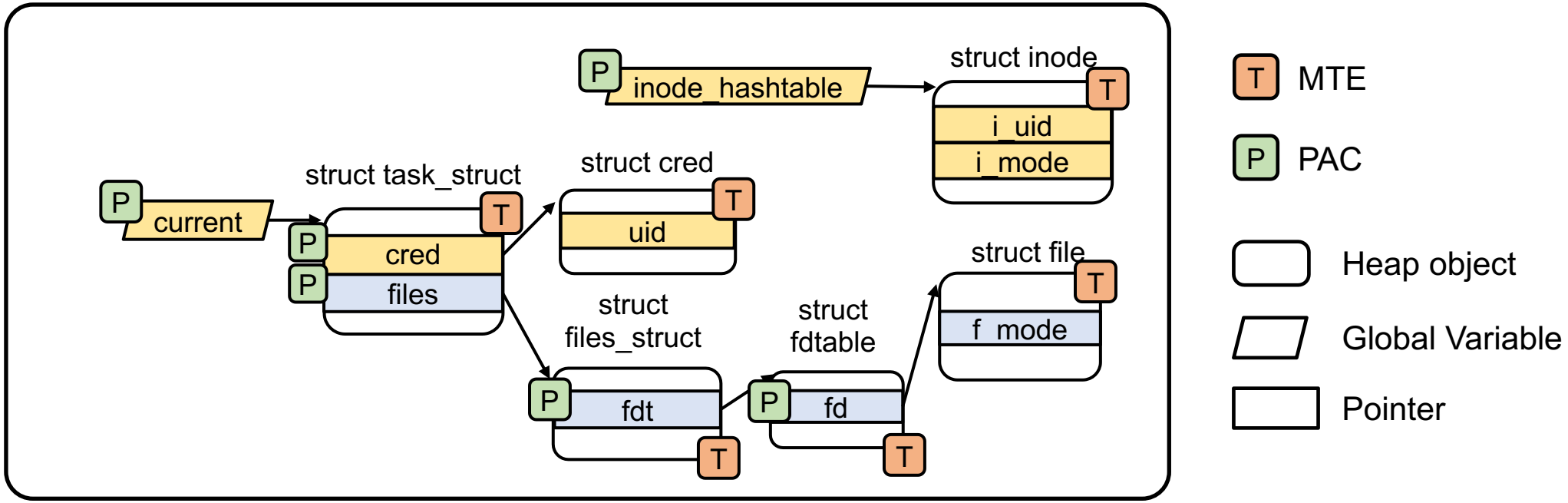
Collecting Policy and Resource
from **user interfaces**



PeTAL's Data Flow Integrity

2. How should they be protected?

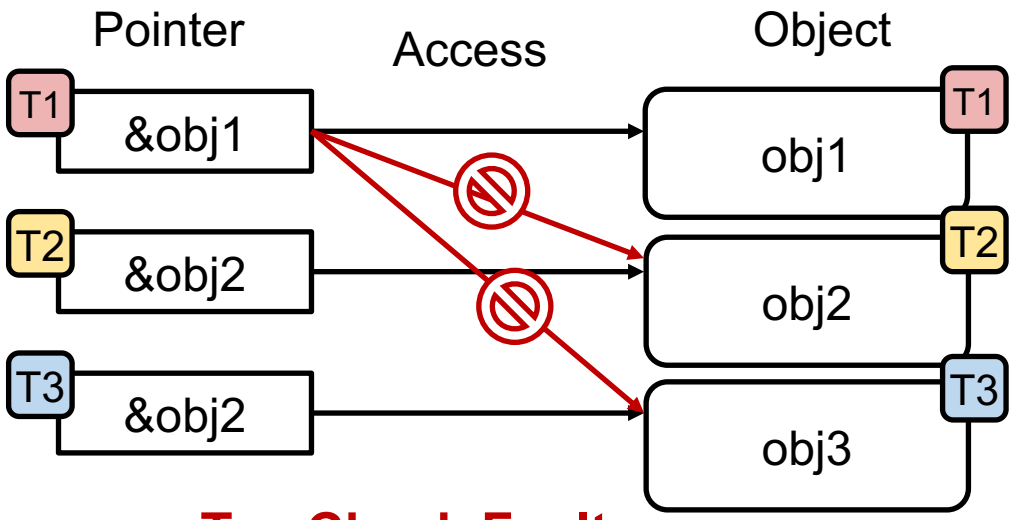
- **Selectively** protect ***access control-related data***
- Leverage **hardware extensions** : **ARM MTE** for objects, **PAC** for pointers



ARM MTE and PAC

Memory Tagging Extensions (MTE)

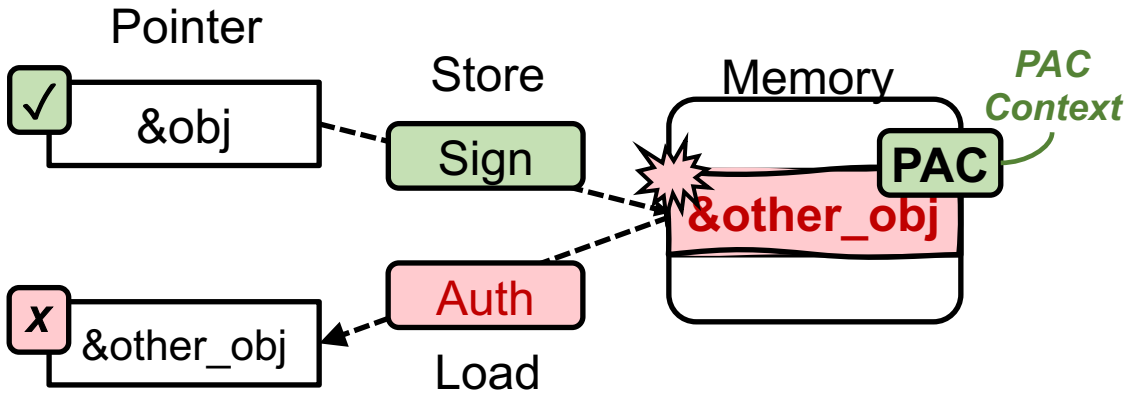
- **Memory object protection**
- Hardware **memory tagging/tag checking**
- Dedicated tag storage in physical memory



Tag Check Fault

Pointer Authentication Code (PAC)

- **Pointer protection**
- Hardware **pointer signing/authentication**
- PAC key + PAC context + Pointer value
  → PAC Signiture



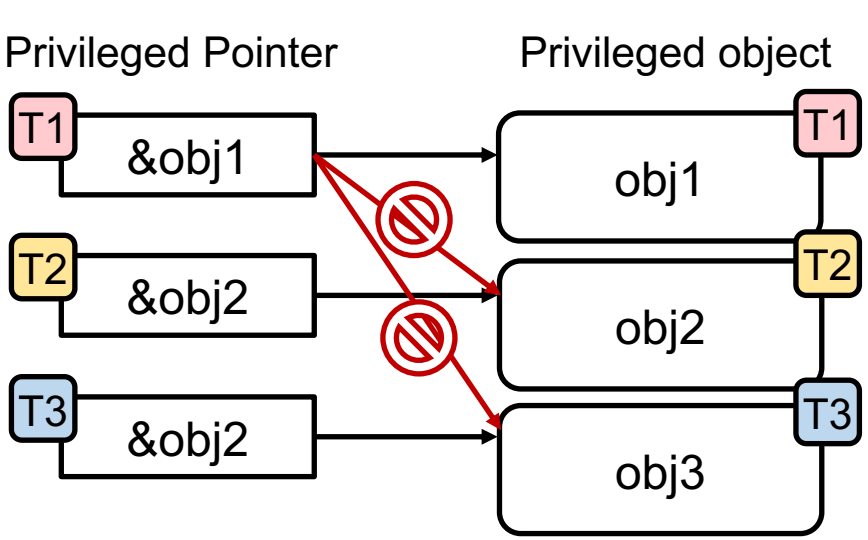
Authentication Failure

Protecting Objects with ARM MTE

Privileged Objects:
Objects that contain **policy**, **resource**, or their **pointer**

Random Tag (Tag 1-14)

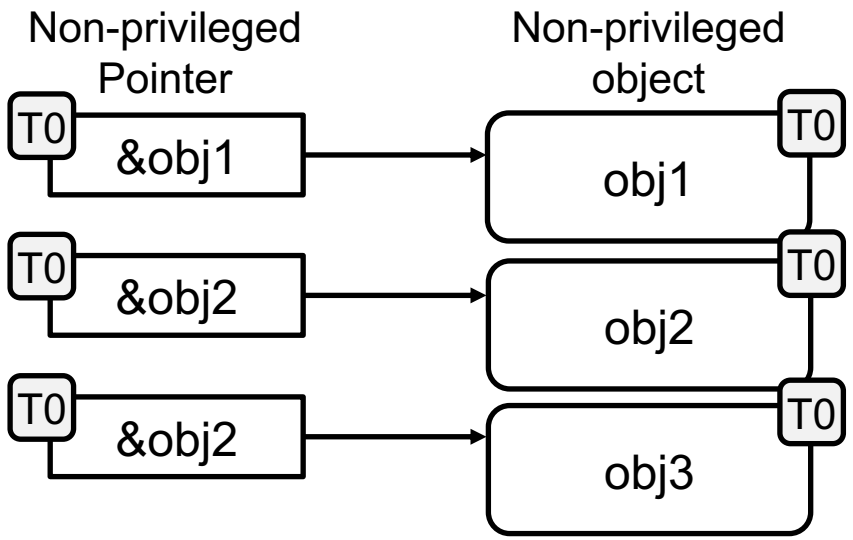
Enforce Pointer's tag on access



Non-privileged Objects:
Other objects

Fixed Tag (Tag 0)

Enforce tag 0 on access

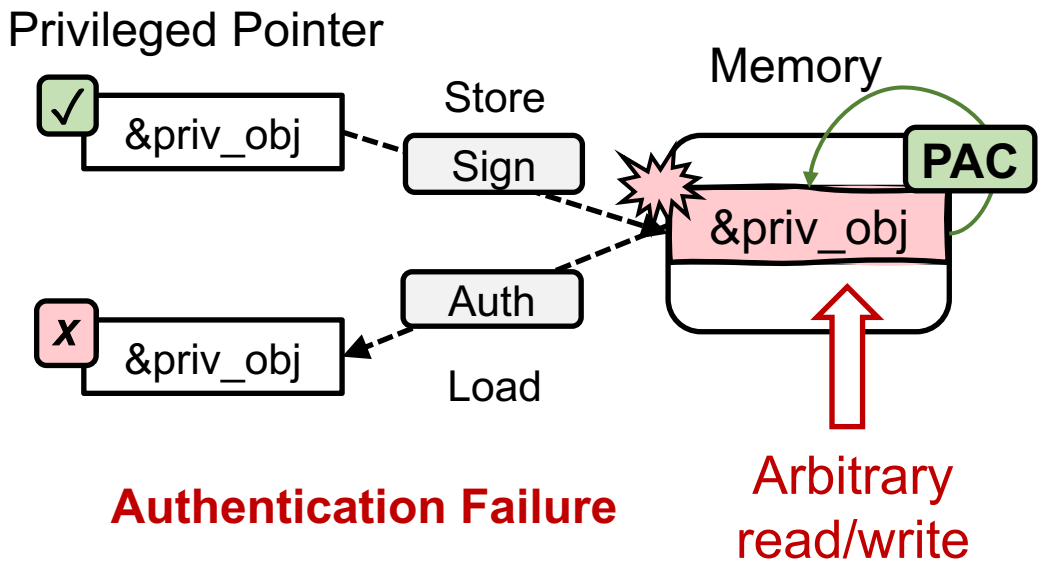


Protecting Pointers with ARM PAC

Privileged Pointers:
Pointers to privileged objects

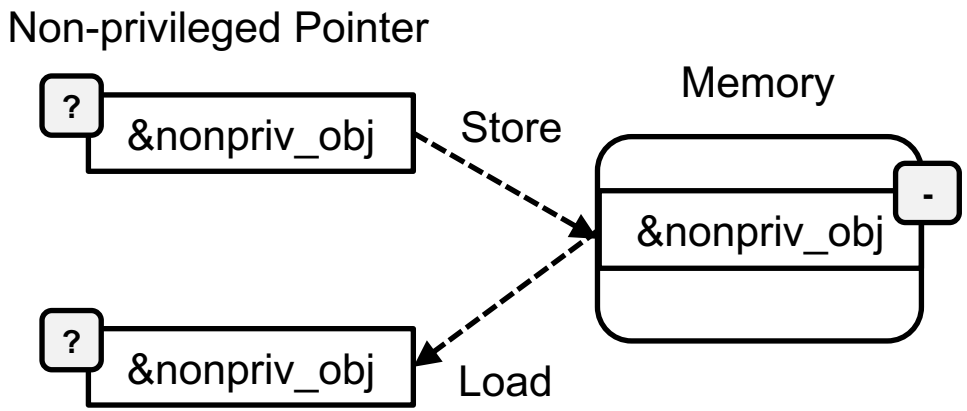
PAC Sign/Authentication

Pointer storage address as PAC Context
→ Bind PAC to the stored address



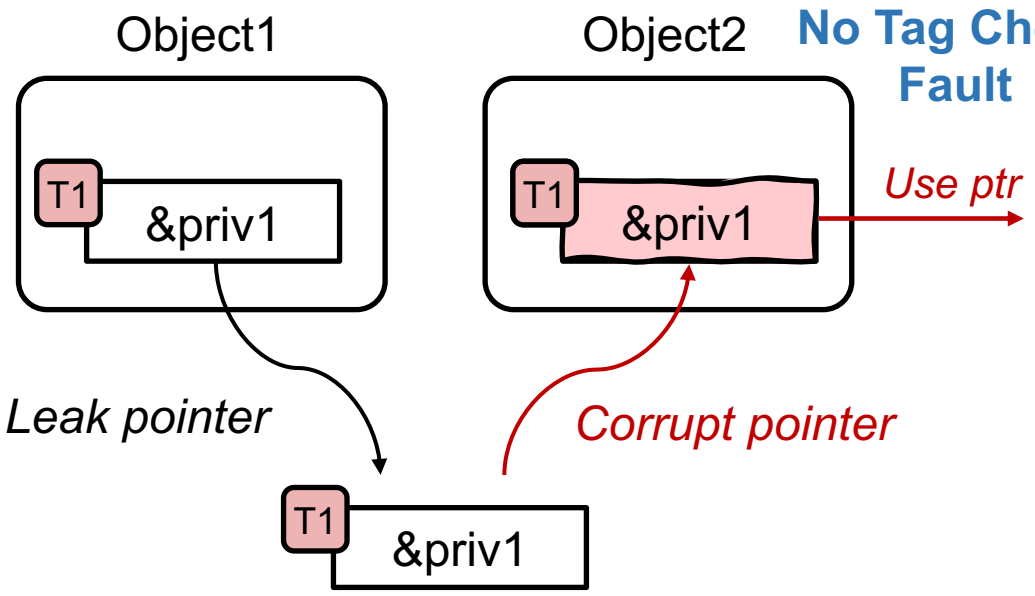
Non-privileged Pointers:
Other pointers

No PAC Sign/Authentication



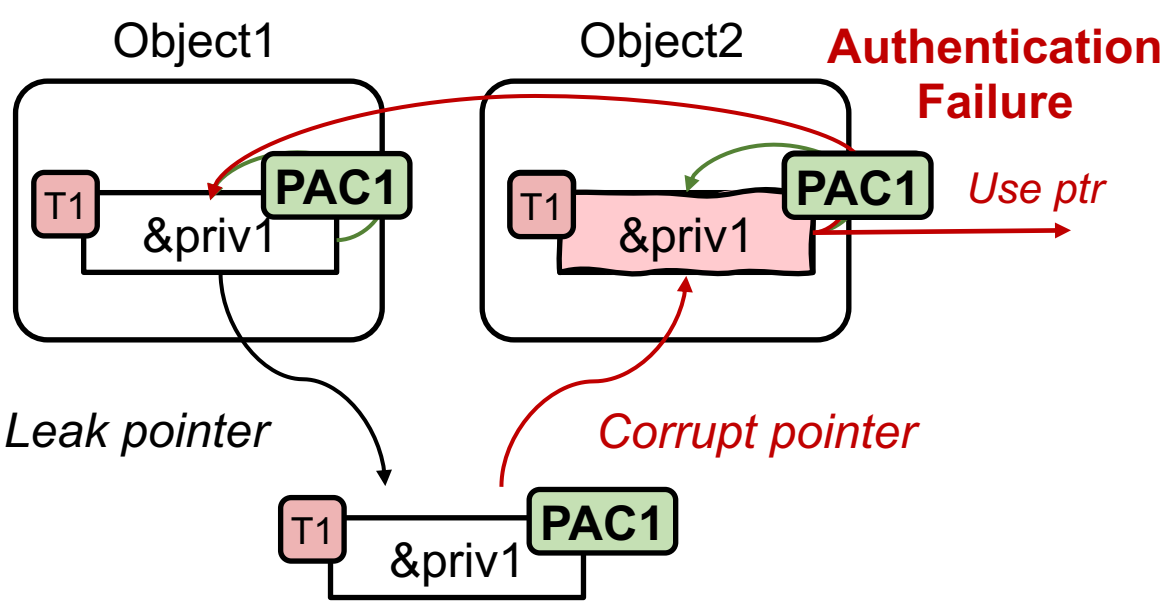
Complementary Relationship of MTE and PAC

MTE Tag Reuse

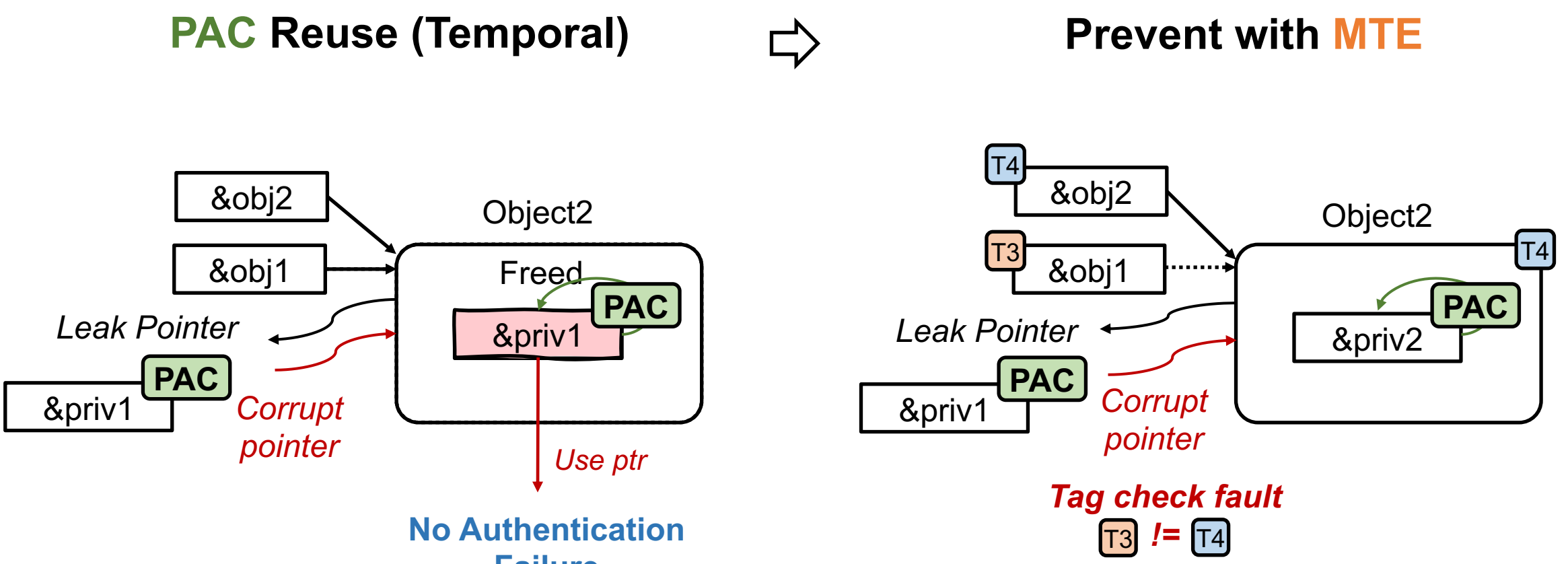


Prevent with PAC

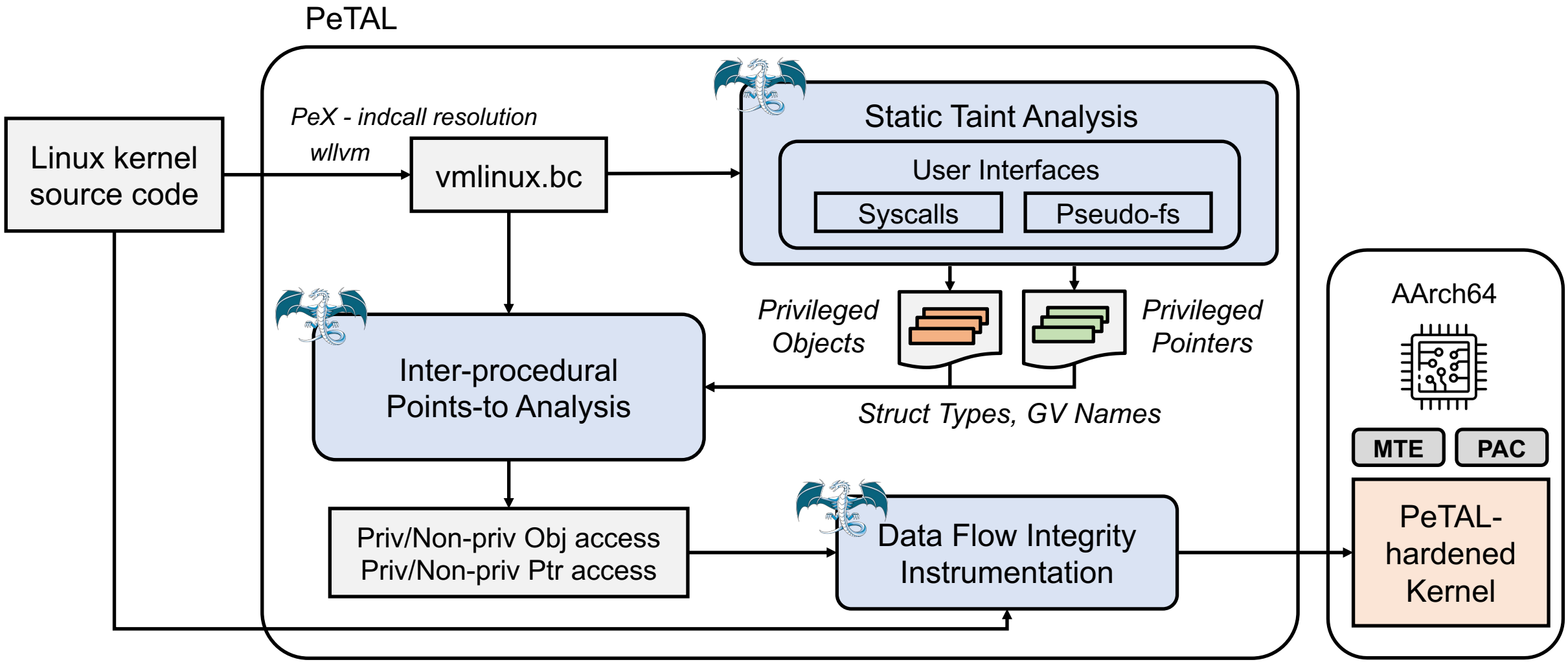
: PAC is bound to the pointer stored address



Complementary Relationship of MTE and PAC



PeTAL Implementation



Performance Evaluation

- Evaluation setup
  - Samsung Galaxy S22 – supports PAC and MTE*

    * MTE was enabled with the assistance of Samsung Electronics
  - Android kernel 5.10.136

- Kernel workloads
  - LMBench 1.18x (MTE async) / 1.32x (MTE sync)
- User workloads
  - Nbench: 1.00x / LevelDB 1.03x / Apache httpd: 1.04x (MTE sync)
- Security evaluations in the paper

Summary

- PeTAL defines **Access Control integrity** for the Linux kernel.
- PeTAL proposes a novel way to **identify protection targets** leveraging the kernel's **user interfaces**.
- PeTAL's DFI solution based on ARM **MTE and PAC** demonstrates acceptable performance overhead.

Thank you!

Samsung Research

Georgia Institute of Technology

Threat Model and Assumptions

- Hardware

  AArch64, ARM MTE PAC

- Kernel
  - State-of-the-art self-protections (e.g., ASLR, NX/DEP, SMAP, CFI)
  - 1+ Memory coruption vulnerabilities
- Attack vector
  - Memory corruption attack through vulnerable system calls
  - Corrupting access control policies/resources
- Out of scope
  - Access control system implementation error
  - Page allocator error (e.g., GPU driver vulnerabilities)
  - In-kernel executions (e.g., eBPF)
  - Hardware side-channel attacks (e.g., Spectre, PACMAN, TikTag)

Correctness of the Static Analyses

- Static Taint Analysis
  - Goal: Collect kernel objects/pointers used as policy or resource from the user intefaces
  - Evaluation: Manual inspection
    - 3 false positives due to complex data flows
    - No false negatives
- Coarse-grained Points-to Analysis
  - Goal: Classify instructions to enforce the DFI
    - Privileged / Non-privileged / Mixed
  - Evaluation: Emperical verification
    - The PeTAL-hardened kernel worked on QEMU and the Galaxy device