DirtyFree

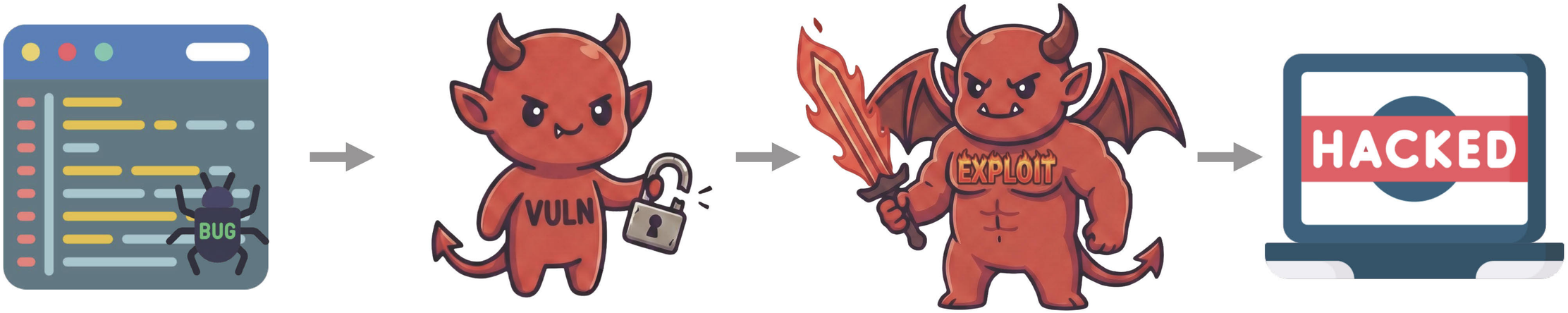
Simplified Data-Oriented Programming in the Linux Kernel

Yoochan Lee, Hyuk Kwon, and Thorsten Holz

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

theori

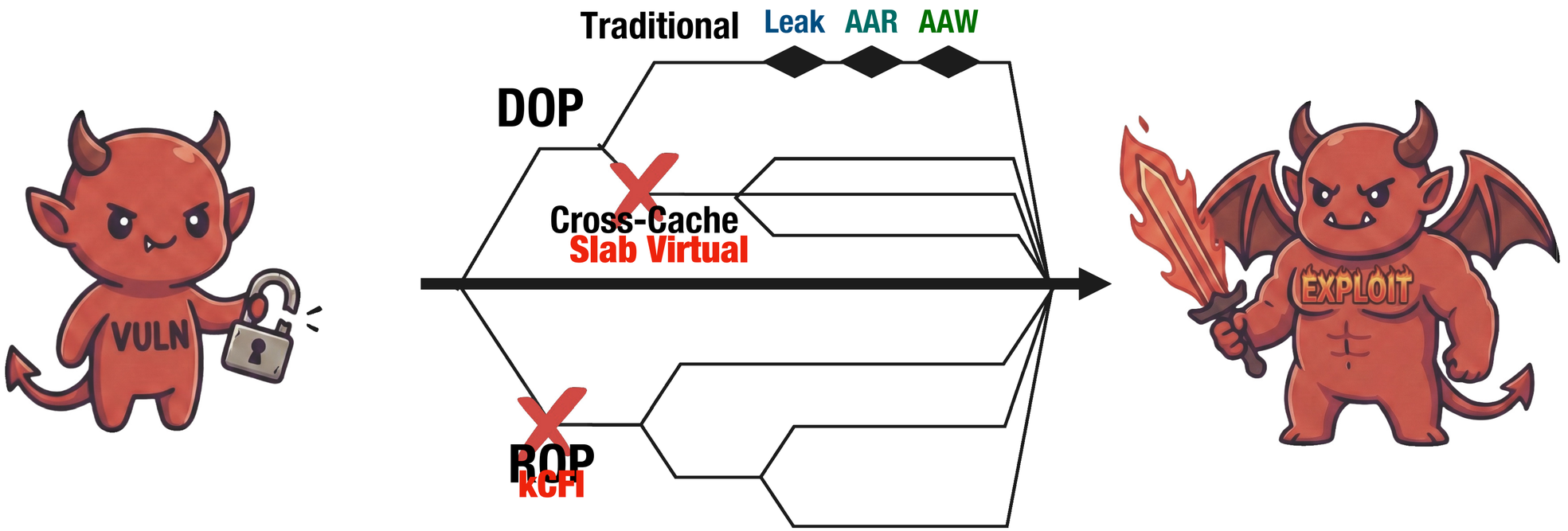
Motivation



Motivation

— How to? →

Motivation


Traditional
Leak
AAR
AAW
DOP
Cross-Cache
Slab Virtual
ROP
kCFI
VULN
EXPLOIT

Motivation

Vulnerability must have **strong exploitability** to fulfill three requirements

(for Information Leakage, Arbitrary Address Read, and Arbitrary Address Write)

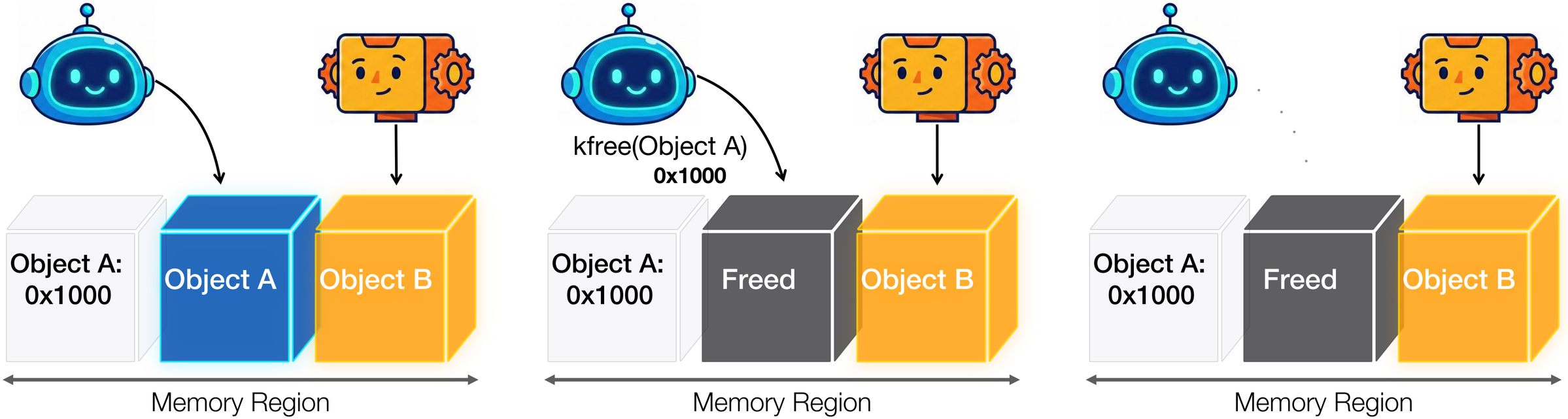
Motivation

Vulnerability must have **strong exploitability** to fulfill three requirements

DirtyFree

- A new exploitation methodology
- Simplified Data-Oriented Programming
- Using an arbitrary free primitive

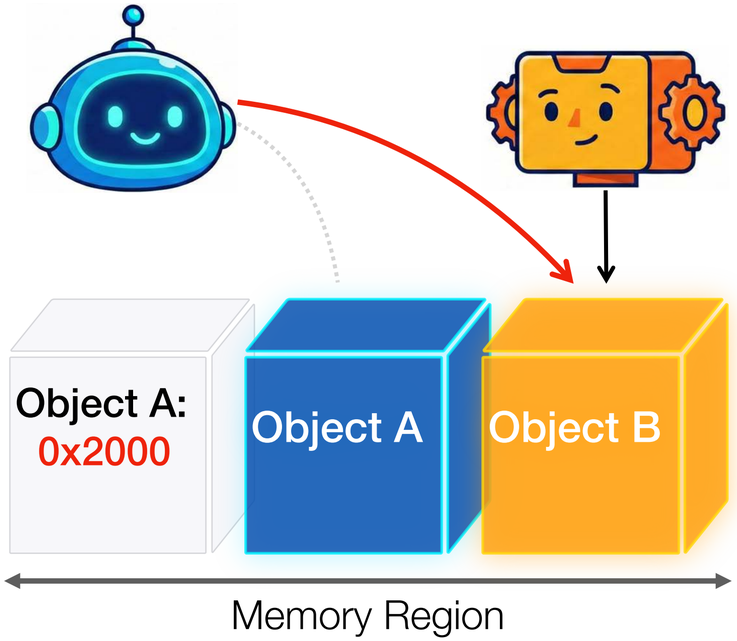
Object Deallocation Process



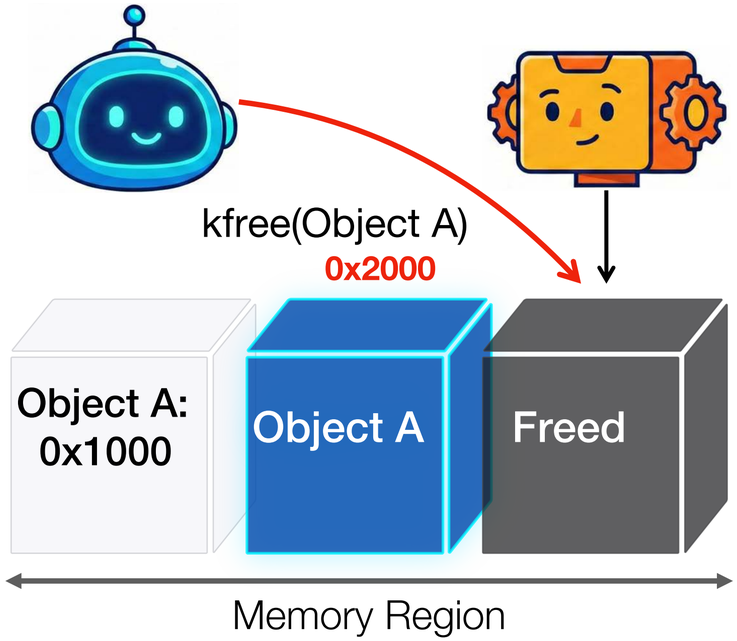
1. Freeing Object

2. Invalidate Pointer

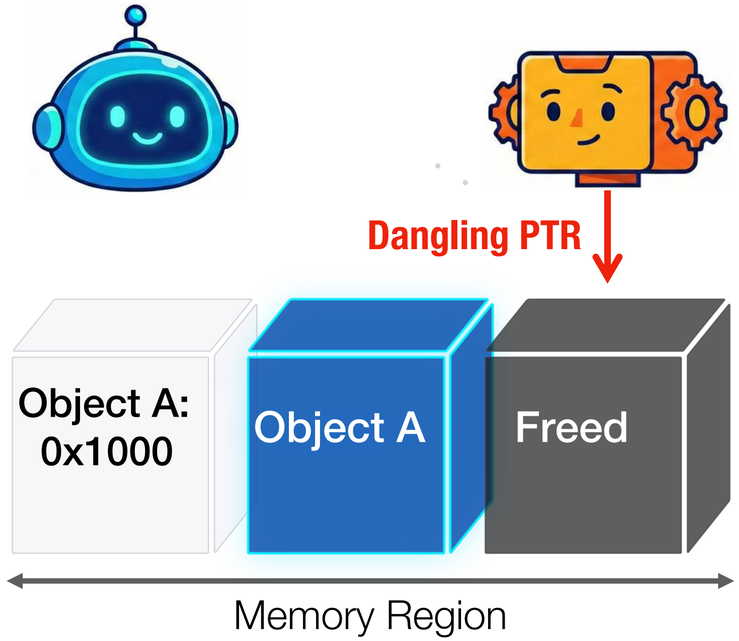
Arbitrary Free Primitive



1. Corrupting a pointer

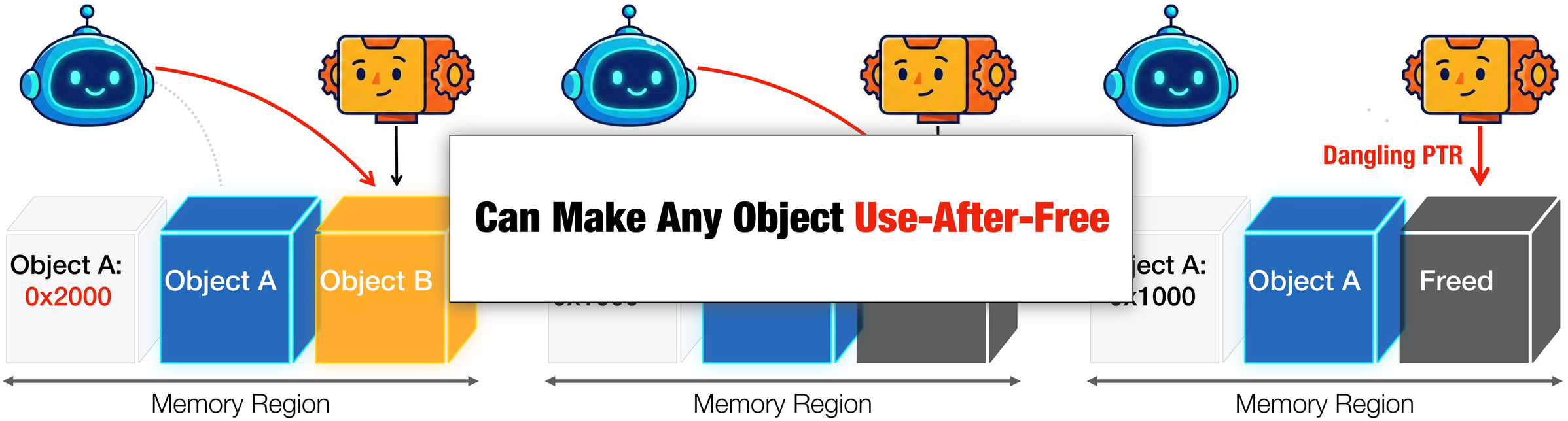


2. Freeing Object



3. Invalidate Pointer

Arbitrary Free Primitive



1. Corrupting a pointer

2. Freeing Object

3. Invalidate Pointer

Previously Considered **Impractical**

- Few arbitrary free objects are known

  i.e., Applicable only in special cases

- Weak primitive

  i.e., No write operation

→ Only used for “**Exploitability Upgrade**” in special cases

Challenges

- **Systematically** identifying arbitrary free objects (afo)

- **Exploit methodology** using an arbitrary free primitive

Challenges

- **Systematically** identifying arbitrary free objects

- **Exploit methodology** using an arbitrary free primitive

Systematic Identification Process

```
syscall {
    …
    afo = kmalloc(X);
    …
}
```

User-allocatable

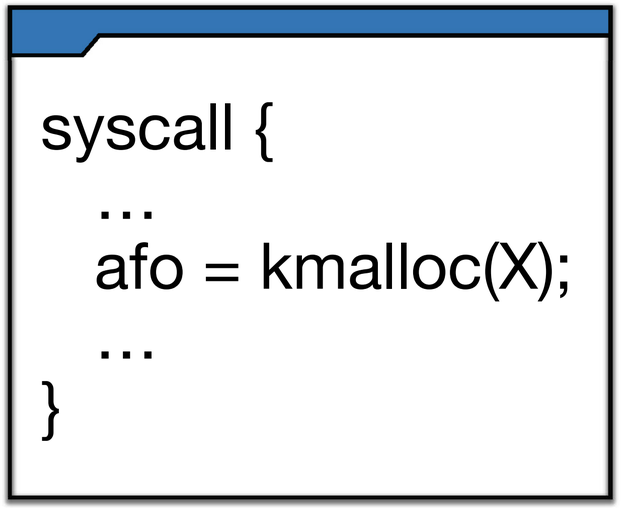
```
struct afo {
    …
    void *ptr;
    …
}
```

Pointer-containing

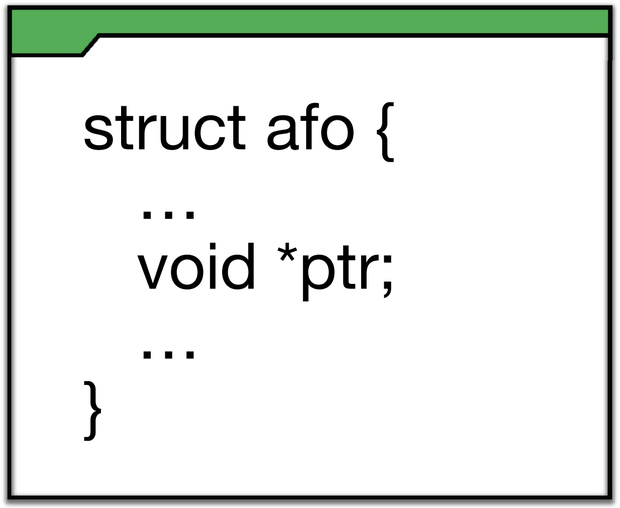
```
syscall {
    …
    OBJ = afo->ptr;
    kfree(OBJ);
}
```

User-controlled kfree

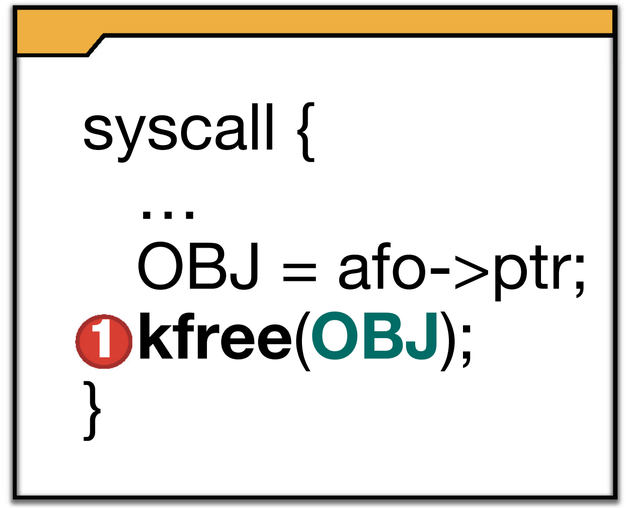
Systematic Identification Process



User-allocatable

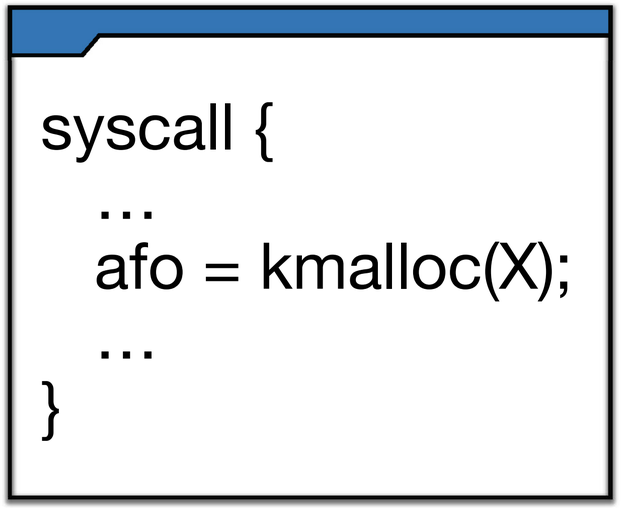


Pointer-containing

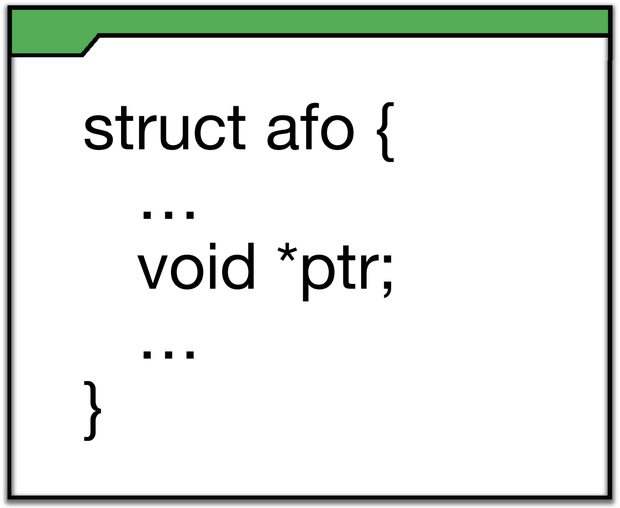


User-controlled kfree

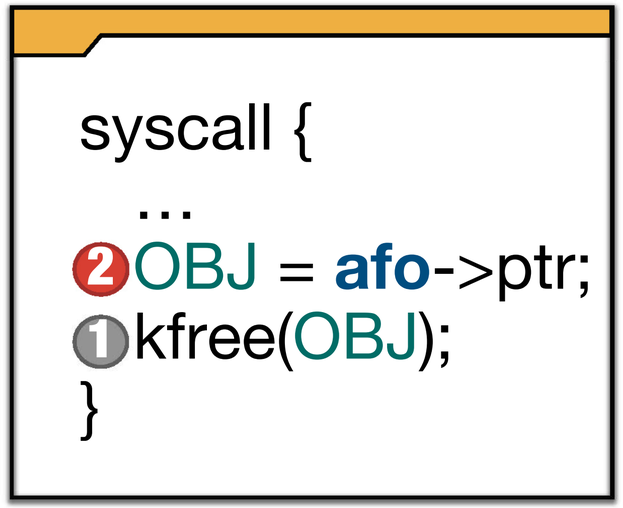
Systematic Identification Process



User-allocatable

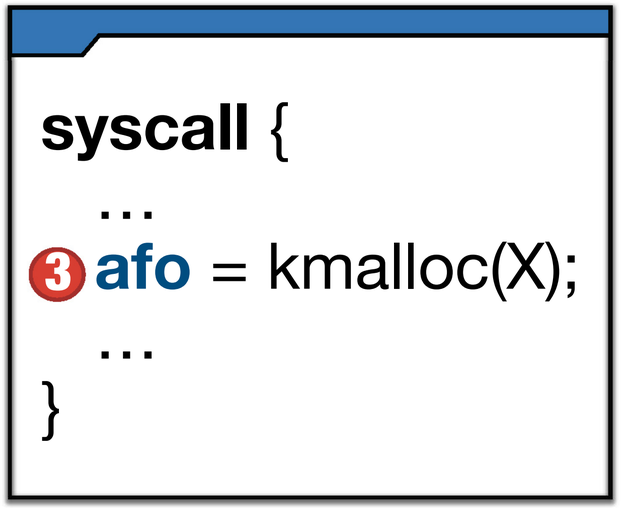


Pointer-containing

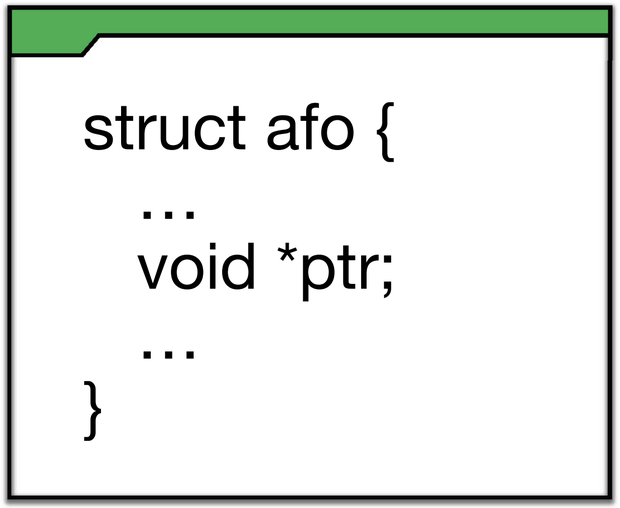


User-controlled kfree

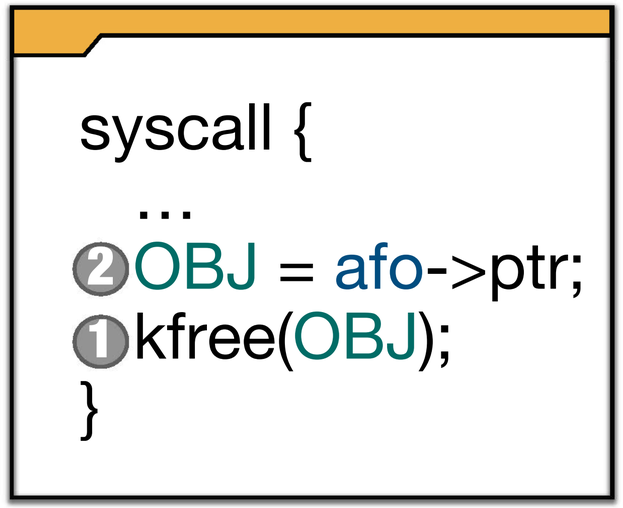
Systematic Identification Process



User-allocatable

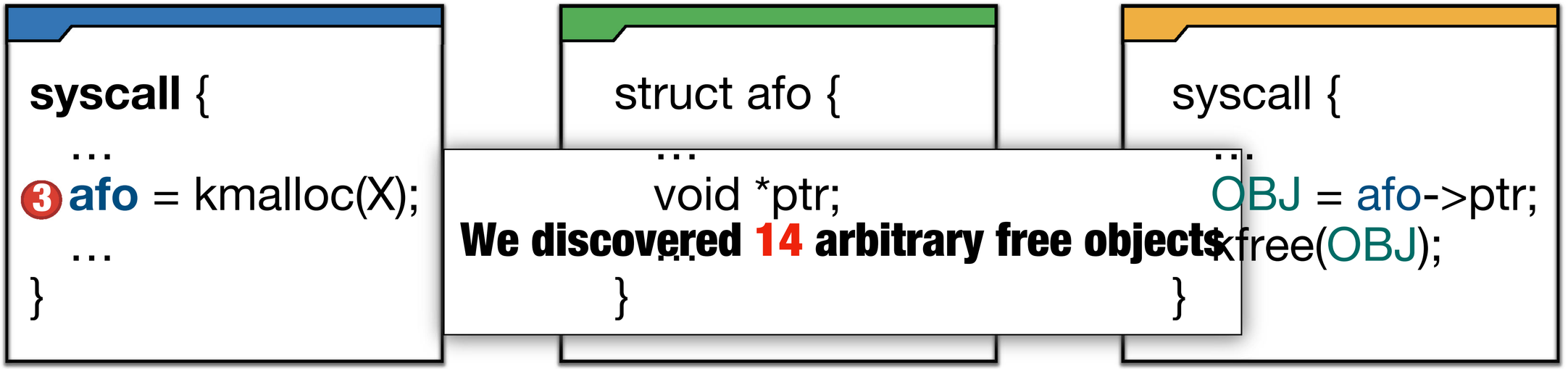


Pointer-containing



User-controlled kfree

Systematic Identification Process



User-allocatable

Pointer-containing

User-controlled kfree

Challenges

- Systematically identifying arbitrary free objects

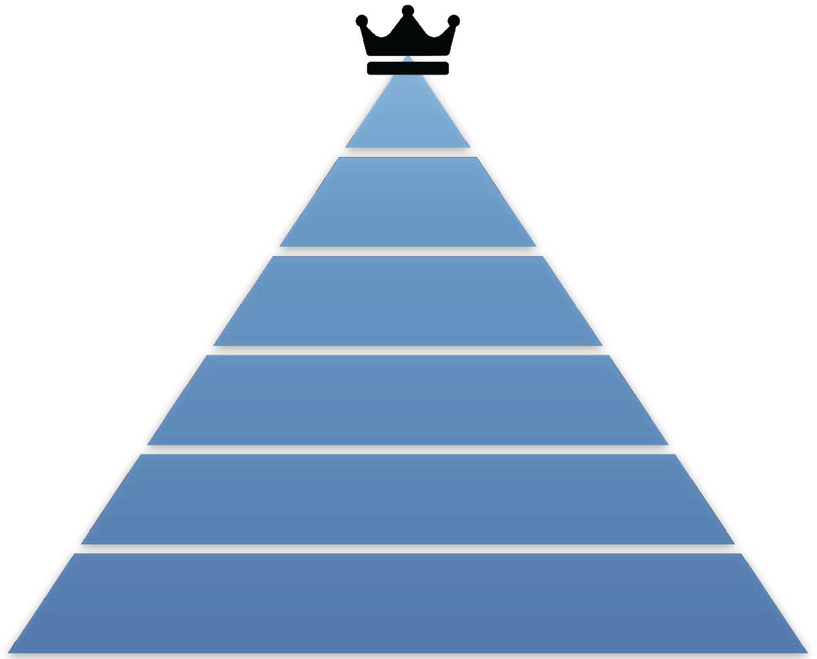
- **Exploit methodology** using an arbitrary free primitive

Target Requirements

1. Privilege-Related
2. Low-Priv Sprayable
3. High-Priv Sprayable
4. Process Rootable

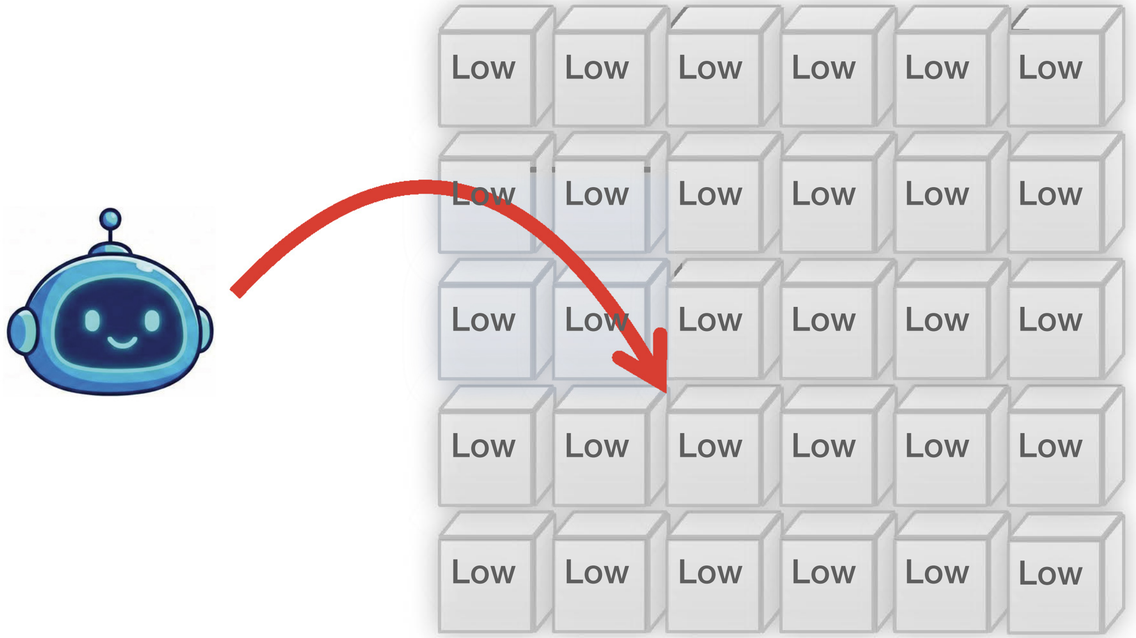
Target Requirements

1. **Privilege-Related**
2. Low-Priv Sprayable
3. High-Priv Sprayable
4. Process Rootable



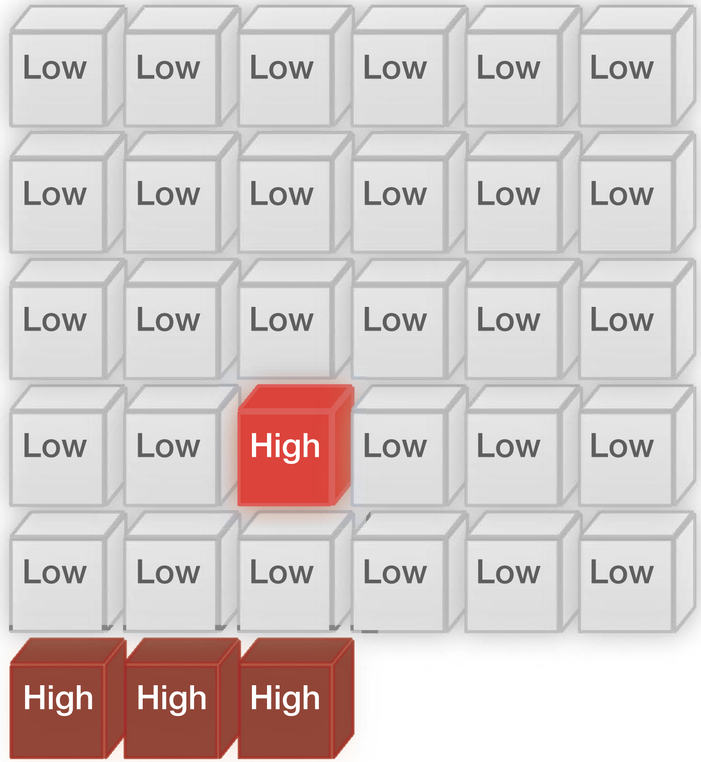
Target Requirements

1. Privilege-Related
2. **Low-Priv Sprayable**
3. High-Priv Sprayable
4. Process Rootable



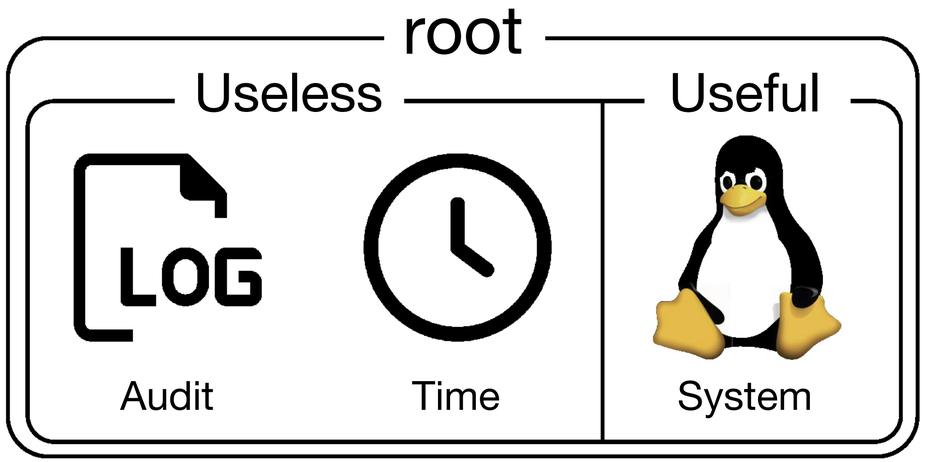
Target Requirements

1. Privilege-Related
2. Low-Priv Sprayable
3. **High-Priv Sprayable**
4. Process Rootable



Target Requirements

1. Privilege-Related
2. Low-Priv Sprayable
3. High-Priv Sprayable
4. **Process Rootable**



Struct **CRED** as a Target

```
struct cred {
    …
    uid_t uid;
    …
}
```

Privilege-Related

```
io_uring_register {
    …
    cred = alloc();
    …
}
```

Low-Priv Sprayable

A. setuid-binary

B. priv-daemon

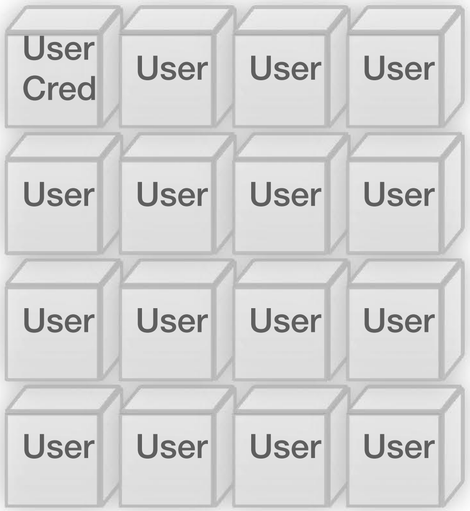
C. workqueue

High-Priv Sprayable

```
uring_write(
      uring_cred,
      “/etc/passwd”,
      “dirtyfree:x:0:0”
);
```

Process Rootable

Overview of **DirtyFree**



Step 1: User-cred spray

Step 2: AFO overwrite & Arbitrary Free

Step 3: Root-cred spray

> id
uid=0 (root)

Step 4: Post-Exploit

Effectiveness of **DirtyFree**

- **Successfully exploited 24 out of 31 publicly known CVEs.**

- **Outperformed state-of-the-art DOP techniques by 3.4x**
  - Traditional DOP: **6 CVEs** & DirtyCred: **14 CVEs**

Summary

- **DirtyFree**: A new Exploit Methodology using Arbitrary Free Primitive
  - Found 14 arbitrary free objects
  - Bypassing modern mitigations (e.g., kCFI, slab-virtual)
  - Demonstrated efficacy across 24 diverse CVEs

I will be on the faculty job market in 2026–2027

Building a research program in practical kernel exploitation & system security.

Homepage