TU Graz

SCIENCE
PASSION
TECHNOLOGY

Fence2Pwn

KFENCE-Enabled Kernel Exploitation Bypassing Slab Hardening and Memory Tagging

Ernesto Martínez García Lukas Maar
Martin Unterguggenberger Stefan Mangard

USENIX Security 2026

› isec.tugraz.at

$ whoami

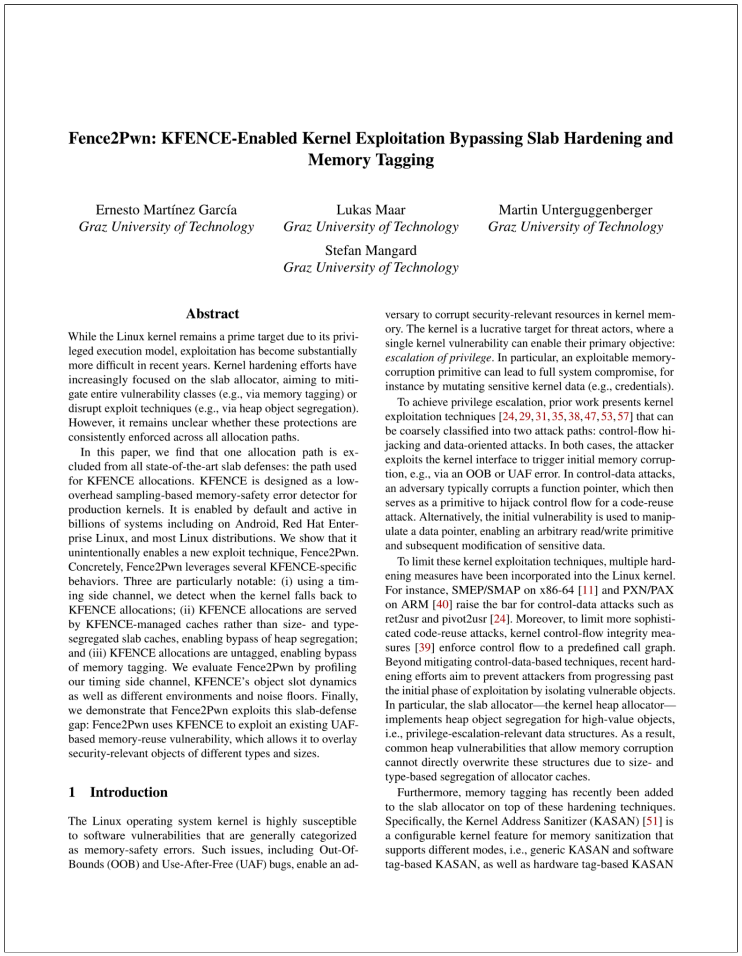
Ernesto Martínez García

PhD Student @ Graz University of Technology

ernesto.martinezgarcia@tugraz.at

ecomaikgolf.com

Overview



Fence2Pwn: KFENCE-Enabled Kernel Exploitation Bypassing Slab Hardening and Memory Tagging

Ernesto Martínez García
Graz University of Technology

Lukas Maar
Graz University of Technology

Martin Unterguggenberger
Graz University of Technology

Stefan Mangard
Graz University of Technology

Abstract

While the Linux kernel remains a prime target due to its privileged execution model, exploitation has become substantially more difficult in recent years. Kernel hardening efforts have increasingly focused on the slab allocator, aiming to mitigate entire vulnerability classes (e.g., via memory tagging) or disrupt exploit techniques (e.g., via heap object segregation). However, it remains unclear whether these protections are consistently enforced across all allocation paths.

In this paper, we find that one allocation path is excluded from all state-of-the-art slab defenses: the path used for KFENCE allocations. KFENCE is designed as a low-overhead sampling-based memory-safety error detector for production kernels. It is enabled by default and active in billions of systems including on Android, Red Hat Enterprise Linux, and most Linux distributions. We show that it unintentionally enables a new exploit technique, Fence2Pwn. Concretely, Fence2Pwn leverages several KFENCE-specific behaviors. Three are particularly notable: (i) using a timing side channel, we detect when the kernel falls back to KFENCE allocations; (ii) KFENCE allocations are served by KFENCE-managed caches rather than size- and type-segregated slab caches, enabling bypass of heap segregation; and (iii) KFENCE allocations are untagged, enabling bypass of memory tagging. We evaluate Fence2Pwn by profiling our timing side channel, KFENCE's object slot dynamics as well as different environments and noise floors. Finally, we demonstrate that Fence2Pwn exploits this slab-defense gap: Fence2Pwn uses KFENCE to exploit an existing UAF-based memory-reuse vulnerability, which allows it to overlay security-relevant objects of different types and sizes.

1 Introduction

The Linux operating system kernel is highly susceptible to software vulnerabilities that are generally categorized as memory-safety errors. Such issues, including Out-Of-Bounds (OOB) and Use-After-Free (UAF) bugs, enable an adversary to corrupt security-relevant resources in kernel memory. The kernel is a lucrative target for threat actors, where a single kernel vulnerability can enable their primary objective: *escalation of privilege*. In particular, an exploitable memory-corruption primitive can lead to full system compromise, for instance by mutating sensitive kernel data (e.g., credentials).

To achieve privilege escalation, prior work presents kernel exploitation techniques [24, 29, 31, 35, 38, 47, 53, 57] that can be coarsely classified into two attack paths: control-flow hijacking and data-oriented attacks. In both cases, the attacker exploits the kernel interface to trigger initial memory corruption, e.g., via an OOB or UAF error. In control-data attacks, an adversary typically corrupts a function pointer, which then serves as a primitive to hijack control flow for a code-reuse attack. Alternatively, the initial vulnerability is used to manipulate a data pointer, enabling an arbitrary read/write primitive and subsequent modification of sensitive data.

To limit these kernel exploitation techniques, multiple hardening measures have been incorporated into the Linux kernel. For instance, SMEP/SMAP on x86-64 [11] and PXN/PAX on ARM [40] raise the bar for control-data attacks such as ret2usr and pivot2usr [24]. Moreover, to limit more sophisticated code-reuse attacks, kernel control-flow integrity measures [39] enforce control flow to a predefined call graph. Beyond mitigating control-data-based techniques, recent hardening efforts aim to prevent attackers from progressing past the initial phase of exploitation by isolating vulnerable objects. In particular, the slab allocator—the kernel heap allocator—implements heap object segregation for high-value objects, i.e., privilege-escalation-relevant data structures. As a result, common heap vulnerabilities that allow memory corruption cannot directly overwrite these structures due to size- and type-based segregation of allocator caches.

Furthermore, memory tagging has recently been added to the slab allocator on top of these hardening techniques. Specifically, the Kernel Address Sanitizer (KASAN) [51] is a configurable kernel feature for memory sanitization that supports different modes, i.e., generic KASAN and software tag-based KASAN, as well as hardware tag-based KASAN

Fence2Pwn:

Technique to bypass heap hardening by using a lesser-known production-aimed debug allocator

Linux Kernel, Software Exploitation, Side Channels
Cross-cache Attacks, Memory Allocators

- Linux **Slab** allocator has multiple **defenses in depth**
- **KFENCE bypasses Slab**'s hardened allocation path
- Attackers can **exploit the alternative path**

ernesto.martinezgarcia@tugraz.at

Background

Linux Kernel Slab Allocator

> Slab (SLUB) is the core Linux Kernel heap **memory allocator**

Contains comprehensive defense-in-depth mechanisms against exploitation

D1: Heap Segregation

`SLAB_BUCKETS`

`SLAB_VIRTUAL`

`TYPED_KMALLOC_CACHES`

D2: Sanitization

`{KASAN,KMSAN}_*`

`KASAN_HW_TAGS`

`INIT_ON_{ALLOC,FREE}`

D3: Randomization(s)

`RANDOM_KMALLOC_CACHES`[D1]

`SLAB_FREELIST_RANDOM`

Most importantly, **objects are segregated**:

```
1 free(vuln_sketchy_object)        // -> UAF Weak Code Object
2 very_important_object = alloc() // -> Even With Same Size
3 write(vuln_sketchy_object)       // -> No UAF Memory Overlap
```

D4: Other

`SAMSUNG RKP`

`HUAWEI HYPERVISOR`

`OTHER DOWNSTREAM`

ernesto.martinezgarcia@tugraz.at

Cross-cache Attacks

> Technique to reclaim cross-domain memory via the physical allocator

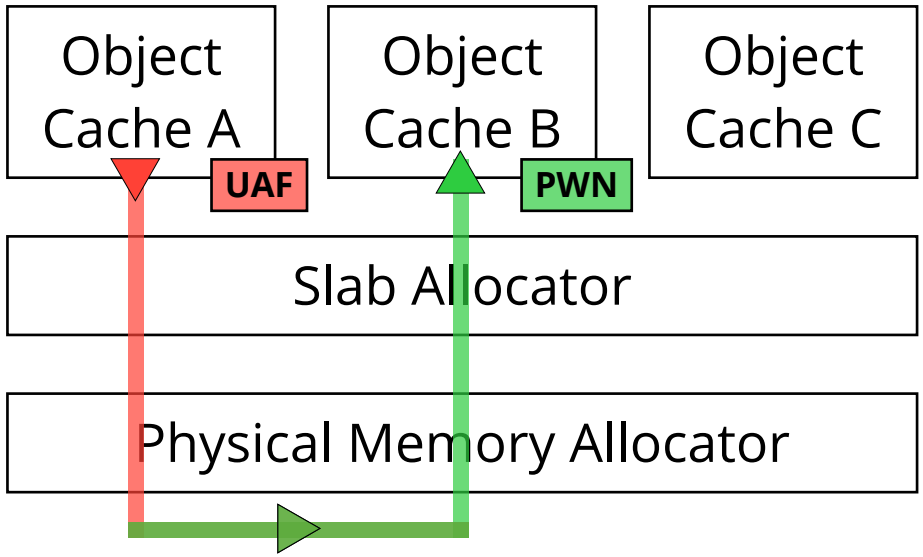
Apply **free pressure** on Cache A

↳ Page(s) containing old A are freed

Apply **alloc pressure** on Cache B

↳ Page is re-allocated for new B

↳ UAF moved from A to B



Future of Cross-cache Attacks

- Memory Tagging probabilistically defends against them — In Deployment
- `SLAB_VIRTUAL` deterministically mitigates them — Not Deployed

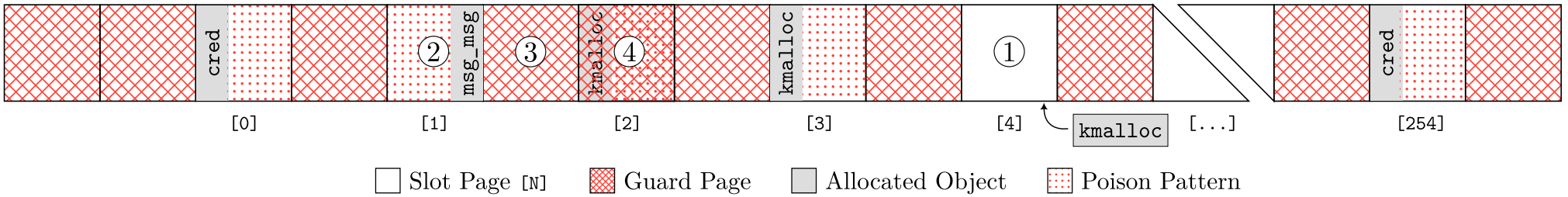
ernesto.martinezgarcia@tugraz.at

Fence2Pwn

Kernel Electric Fence (KFENCE)

> A sample-based memory allocator designed to **detect memory-safety errors**

Every **100ms**, KFENCE short-circuits the Slab allocator for a single allocation



Properties

- Placement is 1/2 randomized
- Freed pages are guarded
- Operates in **FIFO** mode

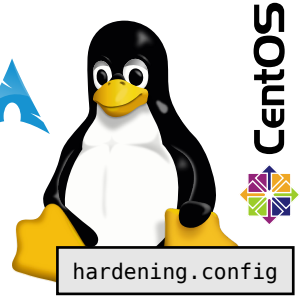
Default Enabled

Android

Red Hat Enterprise Linux

NixOS

CentOS

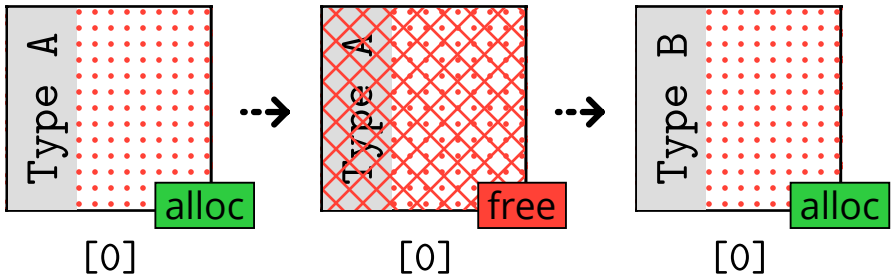


ernesto.martinezgarcia@tugraz.at

KFENCE Weaknesses

We analyzed KFENCE and found five potential weaknesses

(W2) Lack of Heap Segregation



(W3) Memory Tagging (MTE) Bypass

```
1  void * __kasan_slab_alloc( … void *object)
2  {
3    // ...
4    if (is_kfence_address(object))
5      return (void *)object;
6
7    tag = assign_tag(cache, object, false);
8    tagged_object = set_tag(object, tag);
9    // ...
10   return tagged_object;
11 }
```

(W5) Non-fatal Error Handling

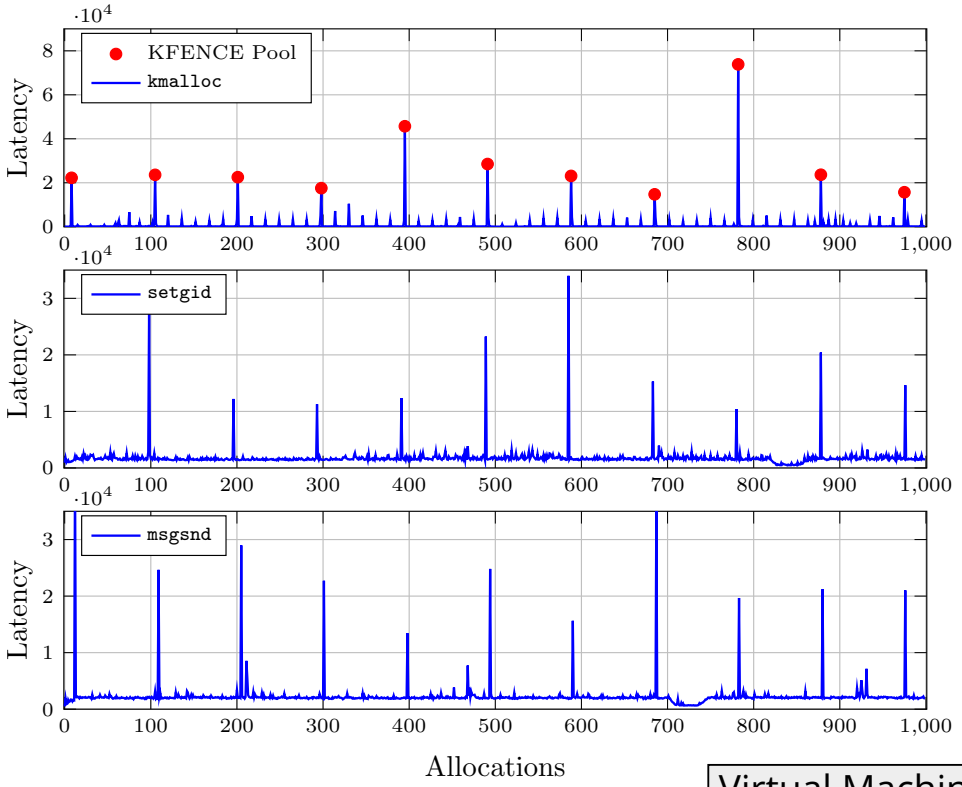
Default & common mode is log-only

↳ Retry until privilege escalation

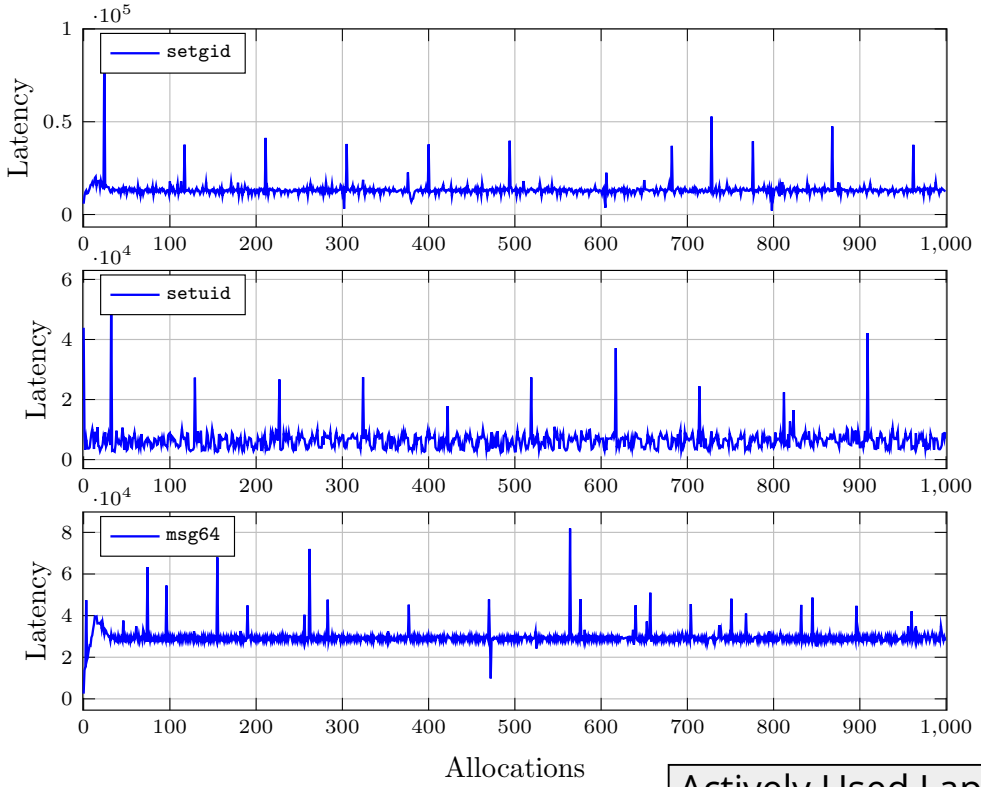
↳ Delete all logs

Side-channel on KFENCE Allocations

(W1) Latency and periodicity on allocating syscalls signal KFENCE activity



Virtual Machine

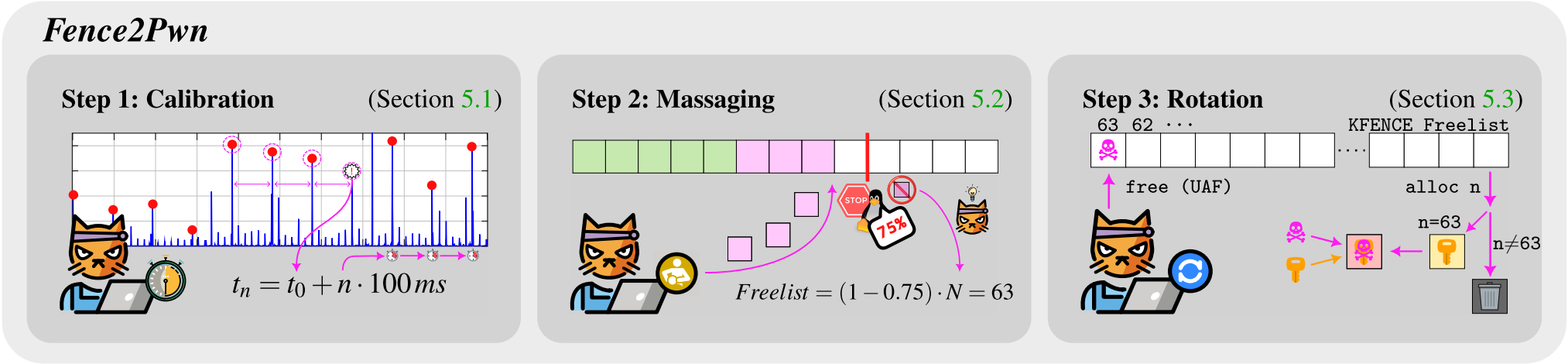


Actively Used Laptop

ernesto.martinezgarcia@tugraz.at

Fence2Pwn

A cache-agnostic object-overlapping technique that bypasses Slab



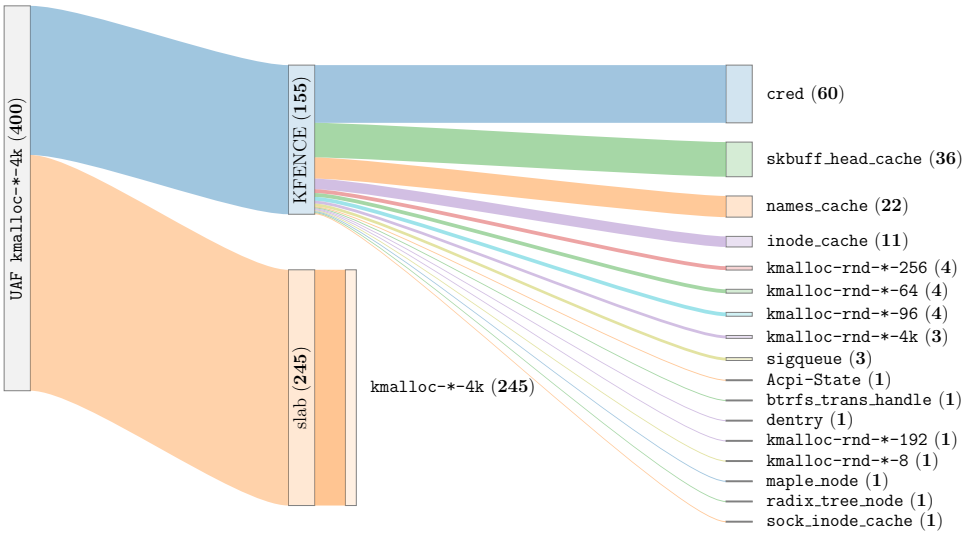
Target KFENCE **Estimate len(freelist)** **Discard N + Reclaim**

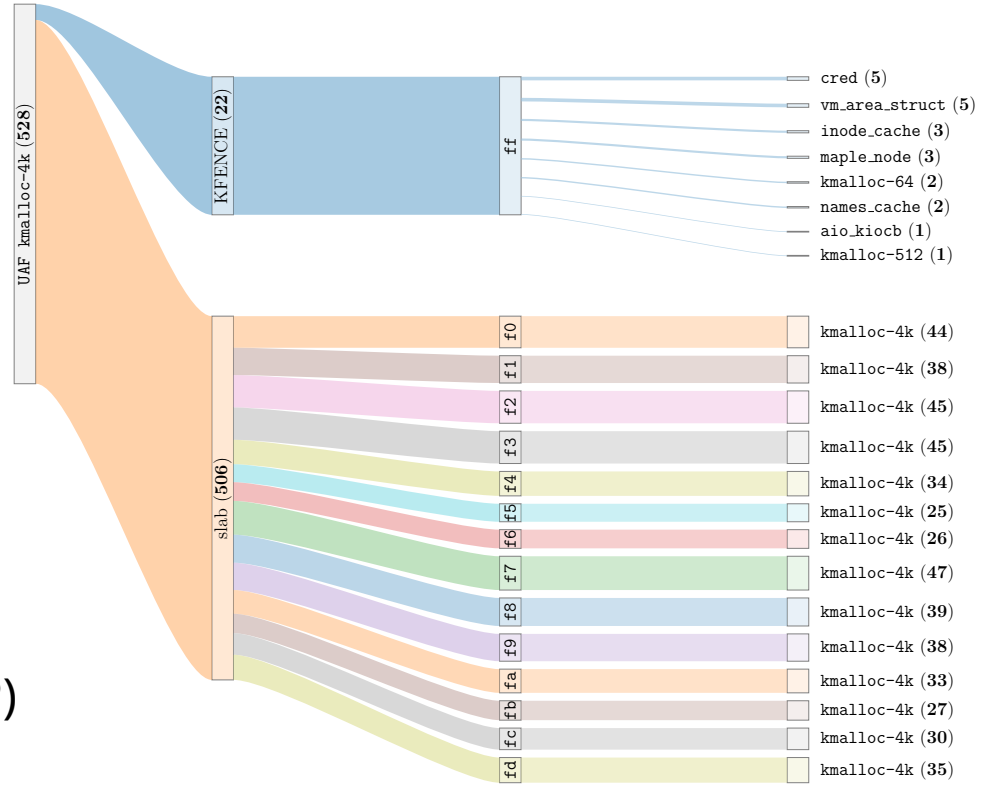
Failed attempts due to non-overlapping KFENCE objects do not crash

↳ An **attacker can retry indefinitely** until good 1/2 random placement

Kernel Object Reclaim

Object type diagram of **reclaimed memory** from previous **UAF**



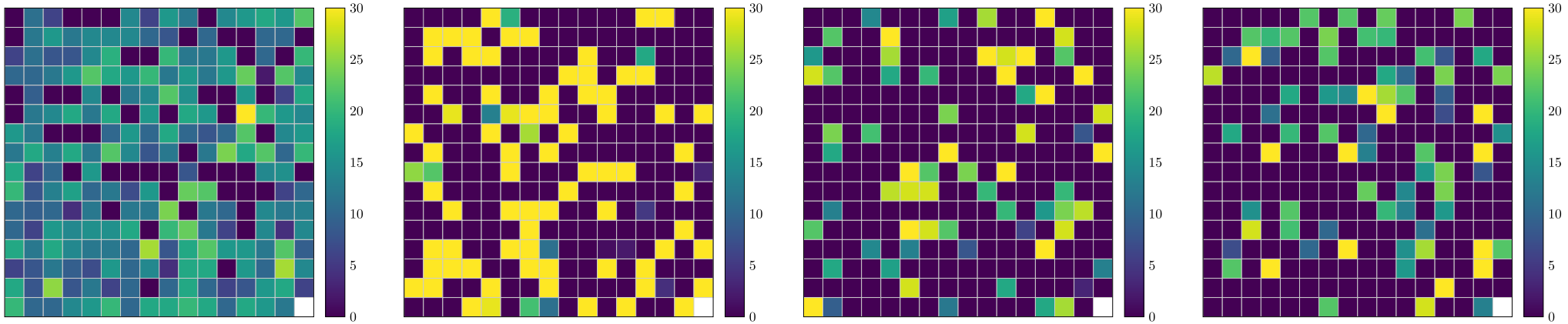


Bypass of current and future mitigations

- ↳ Heap Segregation - Protected Objects (RKP)
- ↳ Memory Tagging - Slab Virtual

Naive Spray-based Exploitation

High-uptime systems **concentrate reclaim activity** on fewer slots



(a) Server A (13h uptime) (b) Laptop (3d uptime) (c) Server B (7d uptime) (d) Laptop (9d uptime)

Long-lived objects hoard KFENCE memory slots as they do not deallocate

↳ Allocations and deallocations concentrate on few slots

↳ More likely to **reclaim** target **UAF** object slots during **sprays**

Conclusion

Defenses and Patches

> Our research motivated **three patches** upstream

1. Disabling KFENCE on MTE

↳ MTE disables KFENCE at boot time

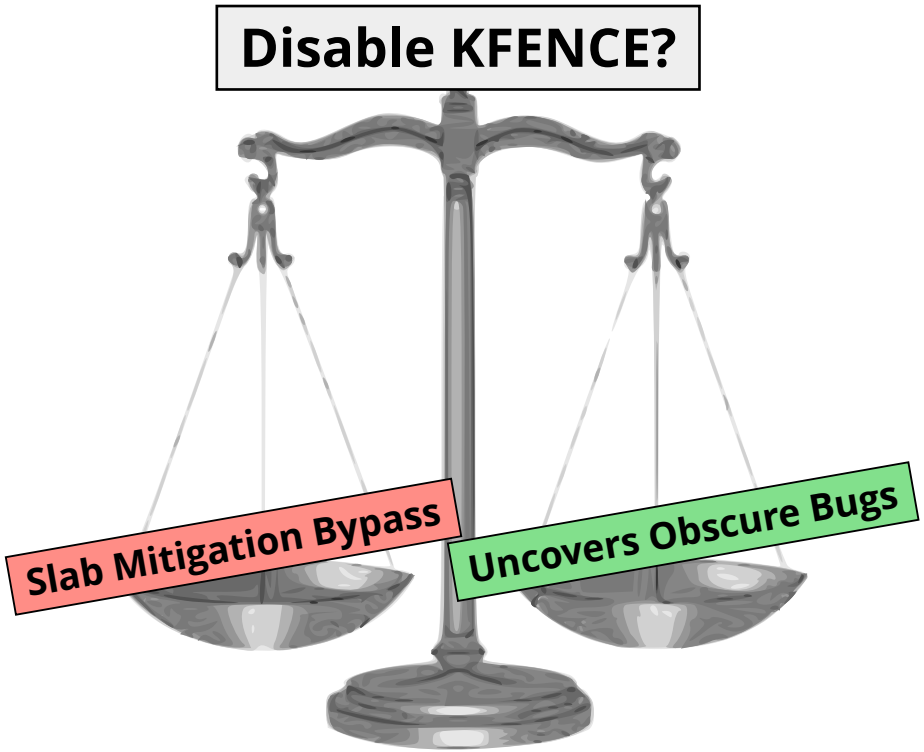
2. Dedicated crash-on-fault toggles

↳ Better KFENCE fault controls

3. Freelist Fisher-Yates Shuffle

↳ Further freelist randomization

Heap segregation remains unsolved



"[...] I don't see a way to resolve it without just disabling it [...]"

Conclusion

KFENCE can be used by an attacker to bypass state-of-the-art heap defenses

Fence2Pwn: KFENCE-Enabled Kernel Exploitation Bypassing Slab Hardening and Memory Tagging

Martínez et al. — **Graz University of Technology**

- Novel Technique on Unexplored Surface
- Major Distributions Impacted
- Unclear Mitigation

ernesto.martinezgarcia@tugraz.at