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

Graz University of Technology

Ernesto Martínez García, Lukas Maar, Martin Unterguggenberger, Stefan Mangard

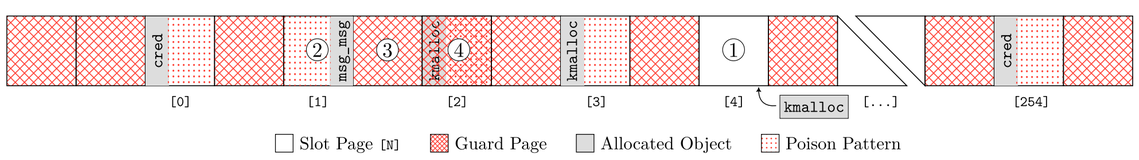
SCIENCE
PASSION
TECHNOLOGY

Linux Heap Segregation

The Linux kernel hardens its allocator to disrupt heap exploit techniques. A core defense is **heap segregation**: objects from different caches avoid reusing each other's memory, typically divided by partitioning allocations by size and type. Bypassing it involves a **cross-cache attack**, which is probabilistically mitigated in production by MTE and deterministically mitigated by `SLAB_VIRTUAL`.

Kernel Electric Fence (KFENCE)

Kernel Electric Fence (KFENCE) is a **sample-based memory allocator** designed for detecting memory-safety issues in **production environments**. Every **100 ms**, the KFENCE allocator **transparently intercepts** the next heap allocation and serves it from its own memory pool instead of the main kernel allocator.



Each allocated object is placed on an isolated page, at the start or end of the page chosen at random. Objects are surrounded by **guard pages** on both sides. On free, the page is unmapped and the slot is appended to the trail of the **freelist**, which operates in **FIFO** mode for use-after-free (UAF) and double-free (DF) safety checks.

Special Operation Mode

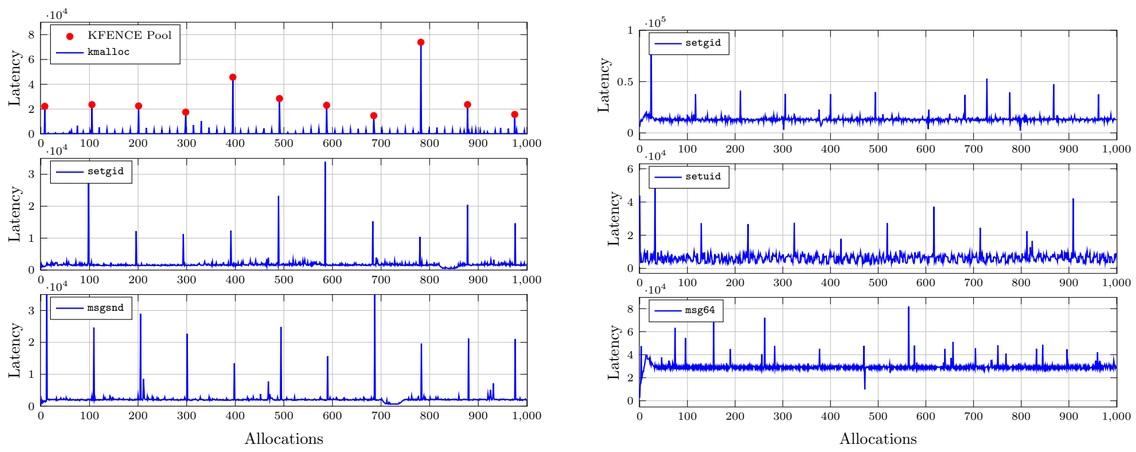
To avoid always serving the same code paths, KFENCE has a special mode once pool utilization exceeds 75%: objects from the same allocation site are no longer served.

Weaknesses

We analyzed KFENCE from a security perspective and identified five weaknesses:

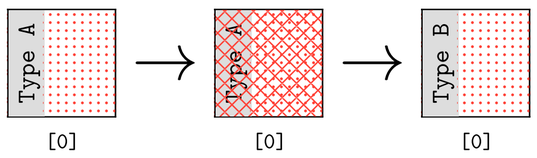
(W1) Timing Side Channel

KFENCE's alternative allocation path leaves a measurable latency trace matching its sampling periodicity, exposing a side channel on KFENCE activations.



(W2) Bypass of Heap Segregation

KFENCE's globally shared pool does not provide heap segregation: a freed slot can be reclaimed by any compatible object, regardless of type, size, or `SLAB_VIRTUAL`.



(W3) Bypass of Memory Tagging

KFENCE does not instrument objects with any KASAN mode, including MTE:

```
void *__kasan_slab_alloc(..., void *object, ...) {
    if (is_kfence_address(object))
        return (void *)object;

    tag = assign_tag(cache, object, false);
    tagged_object = set_tag(object, tag);
    return tagged_object;
}
```

(W4) Deterministic Poison & Deferred Validation

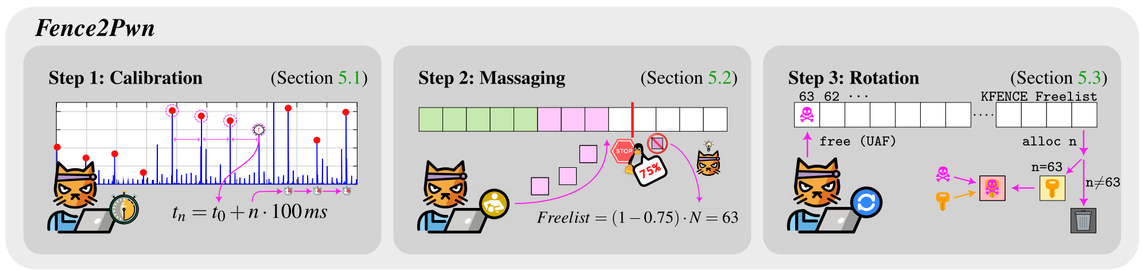
Page poison follows a deterministic pattern, predictable and restorable for an attacker. Poison is checked on free, so attackers may avoid freeing corrupted objects.

(W5) Non-fatal Error Handling

KFENCE **does not panic** on detecting a memory-safety violation, production distributions we observed default to **log-only mode**. This lets an attacker corrupt object without limitation and erase logs after succeeding.

Fence2Pwn

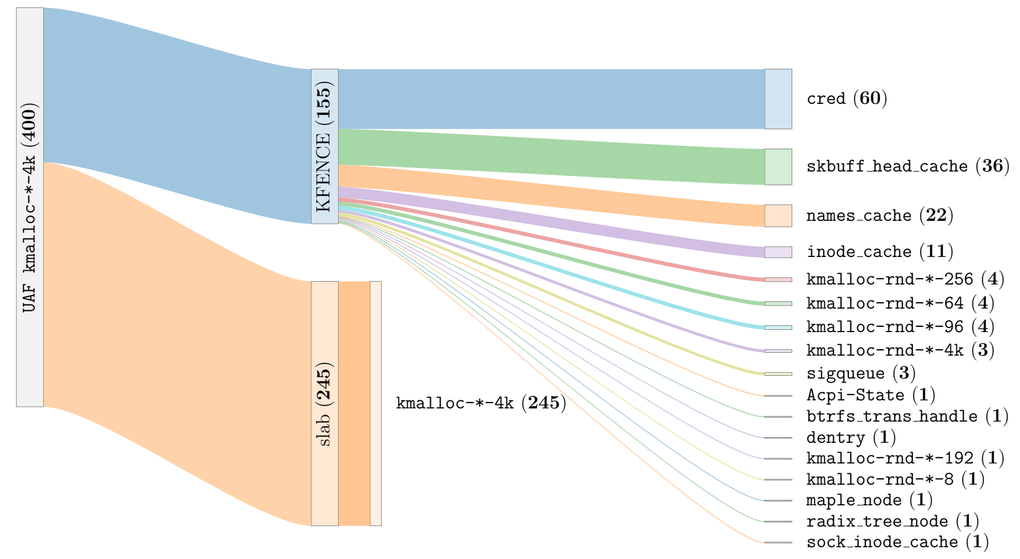
Combining found KFENCE weaknesses, we designed the Fence2Pwn technique:



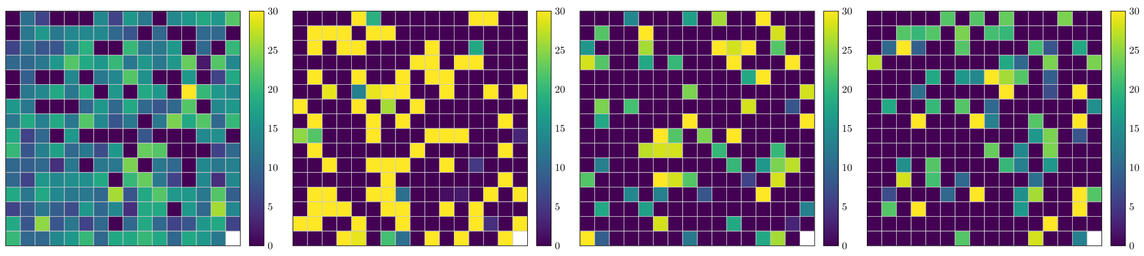
First, we exploit KFENCE's latency and periodicity as a side channel to calibrate and predict future allocations. **Second**, we estimate the size of KFENCE's freelist, e.g., by observing its special operation mode. **Third**, we free an UAF-vulnerable object through KFENCE, rotate the freelist by its estimated length (FIFO), and allocate a victim object. On success, it overlaps with 50% probability and without an MTE tag.

Experiments

Use-after-free kmalloc Object Reclaimed as Sensitive Object



High-uptime Systems Concentrate KFENCE Reclaim Activity



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

Conclusion

A system with multiple allocators where an attacker may choose between any **(W1)** must keep all at the same level of hardening **(W2-5)**, as an attacker is assumed to always choose the weakest link.

- KFENCE default on major distros
- Disabling KFENCE closes the gap
- KFENCE changed to disable on MTE now
- Full fix needs equal hardening everywhere