POLITECNICO
MILANO 1863

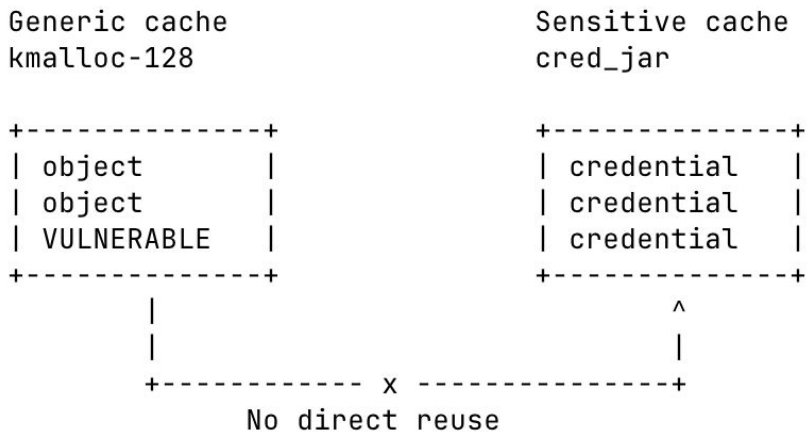
CacheMeIfYouCan

Linux Kernel Privilege escalation using a cross-cache attack

Oliver Quin, Leonardo Sinibaldi

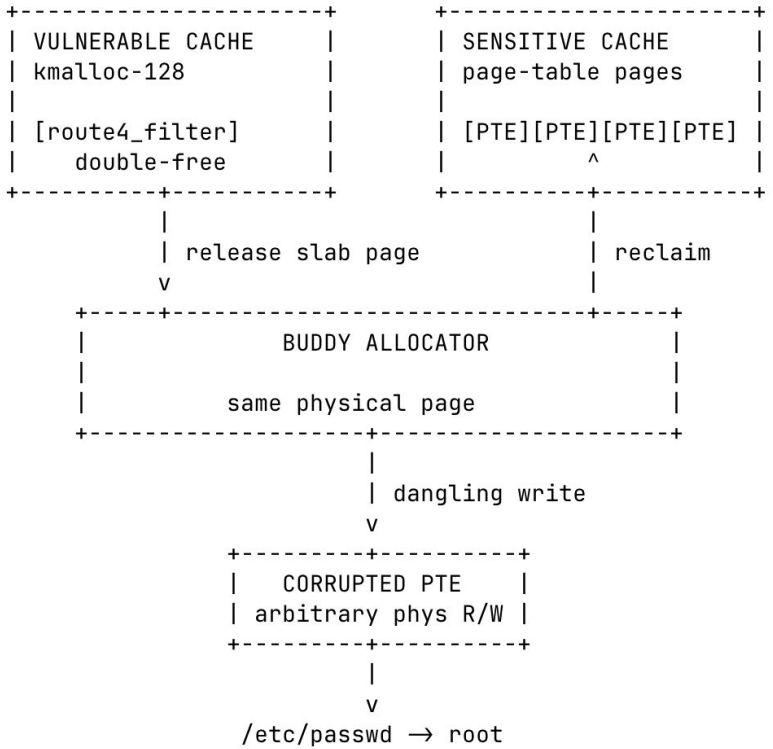
Why is kernel heap exploitation hard?

- **Limited primitives:** Kernel vulnerabilities commonly provide only a double-free, use-after-free, or constrained write.
- **Cache isolation:** Vulnerable objects and security-sensitive objects are allocated in separate SLUB caches.
- **Unreliable recycling:** Returning the correct slab page to the buddy allocator is difficult to control.
- **Our objective:** Turn a double-free in kmalloc-128 into arbitrary physical memory access and root privileges.



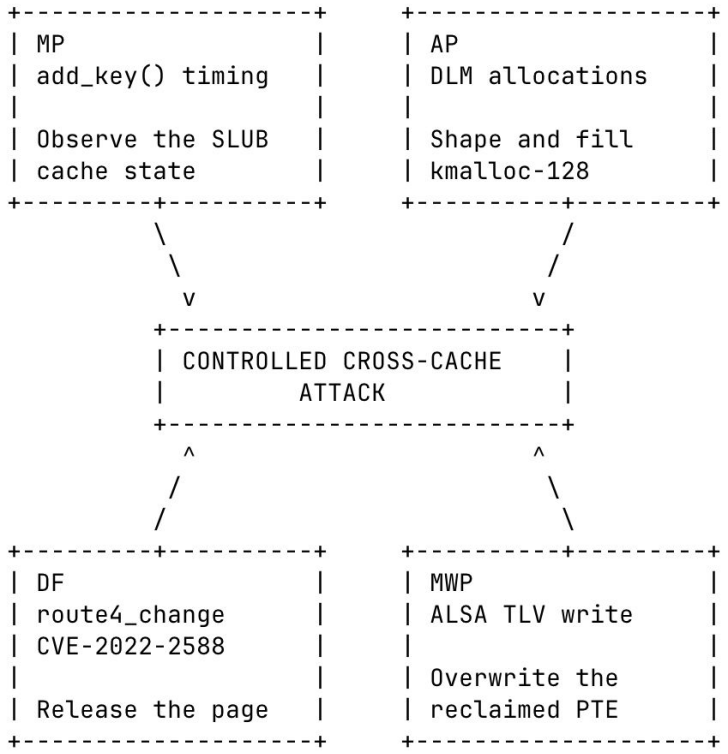
Our goal: cross the cache boundary

- **Starting point:** CVE-2022-2588 provides a double-free affecting an object allocated in kmalloc-128.
- **Target:** Recycle the vulnerable slab page and reclaim it as a page-table page.
- **Expected impact:** Convert a constrained heap primitive into arbitrary physical memory read/write.
- **Final objective:** Modify /etc/passwd and obtain root privileges from an unprivileged process.



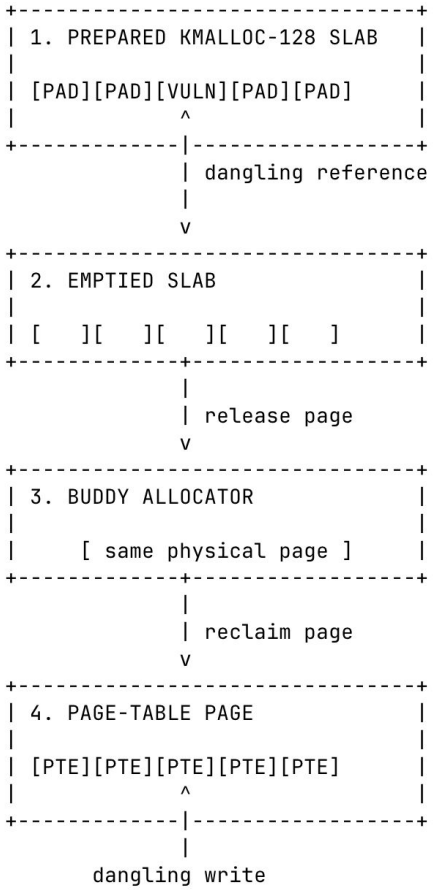
Tools of the craft: kernel primitives

- **Measurement Primitive:** Time short-lived allocations to infer the state of the SLUB cache.
- **Allocation Primitive:** Use the Distributed Lock Manager to allocate objects in kmalloc-128.
- **Double-Free Primitive:** Trigger CVE-2022-2588 through the vulnerable route4_change path.
- **Memory Write Primitive:** Use the ALSA replace_user_tlv() path for an attacker-controlled kernel write.



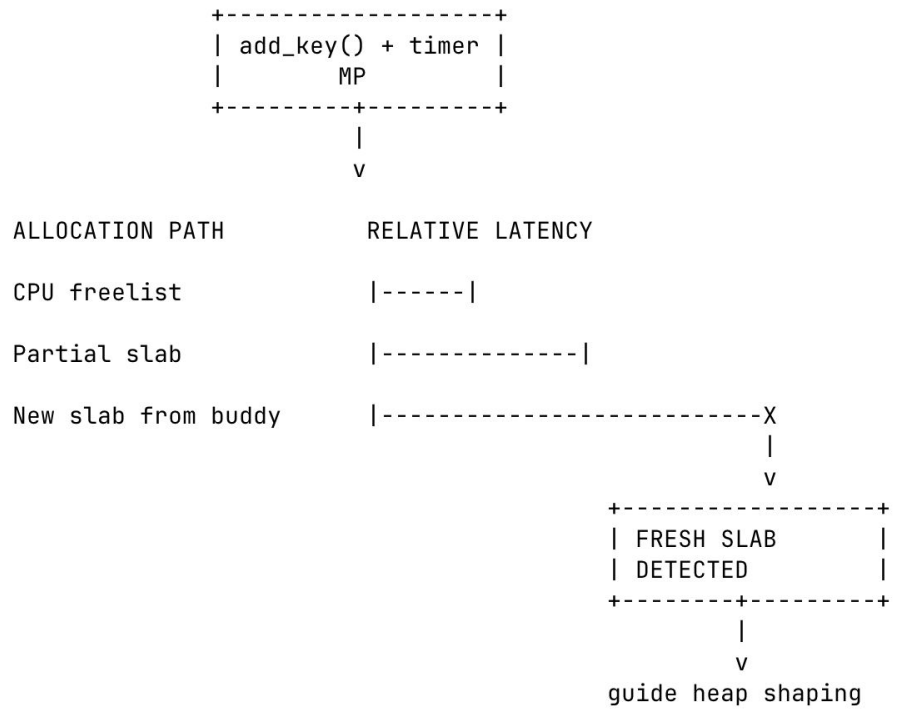
Crossing boundaries

- **Prepare the cache:** Fill a controlled kmalloc-128 slab with padding and vulnerable objects.
- **Create a dangling reference:** Free the selected object while preserving a way to trigger the write.
- **Release the slab:** Empty the slab so that its backing page returns to the buddy allocator.
- **Reclaim the page:** Force the same physical page to host page-table entries.



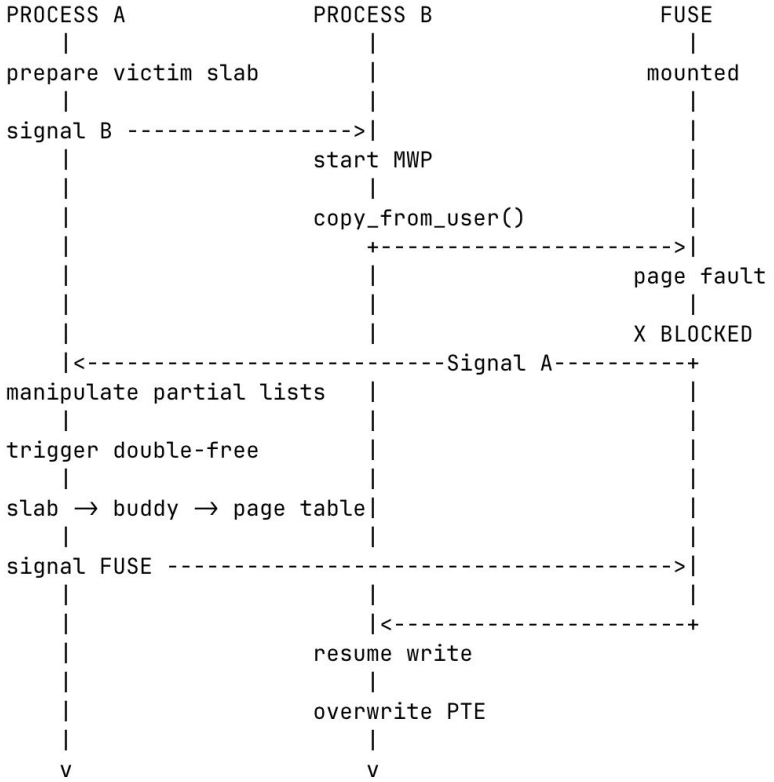
Side channel in a nutshell

- **Different allocation paths:** SLUB serves allocations through paths with different execution times.
- **Latency feedback:** Measurements reveal whether an allocation uses a freelist, a partial slab, or a new page.
- **Fresh-slab detection:** The side channel indicates when the allocator obtains a new slab page.
- **Improved control:** Heap manipulation becomes observable instead of relying entirely on blind spraying.



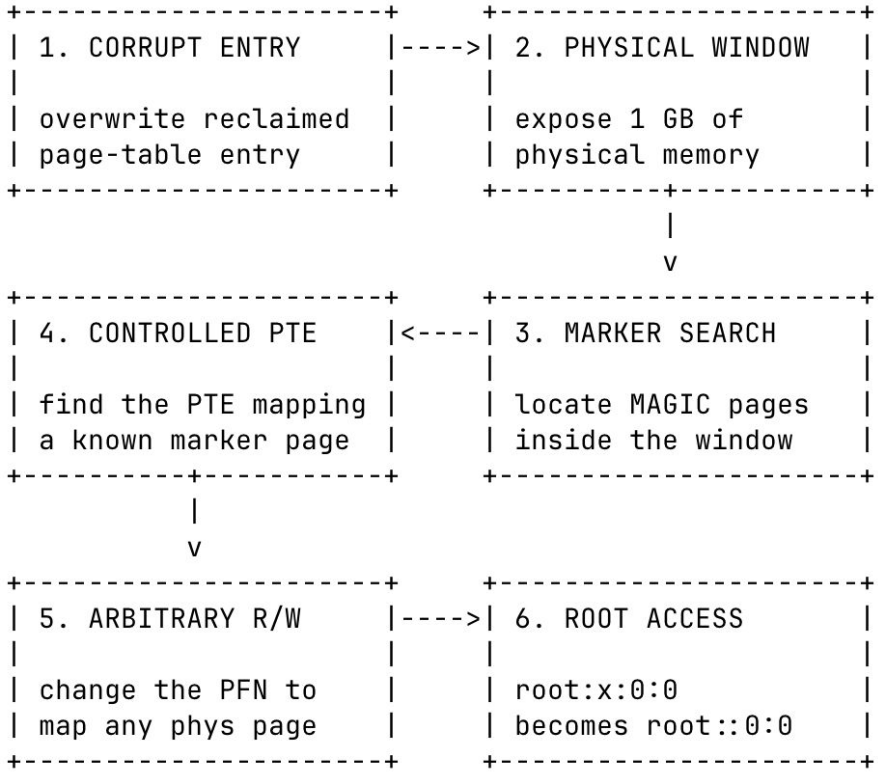
FUSE extendes recycling window

- **Process B starts the MWP:** The kernel reads data from a FUSE-backed userspace page.
- **FUSE blocks the page fault:** Execution remains suspended inside copy_from_user().
- **Process A recycles the page:** It manipulates the slab, triggers the double-free, and reclaims the page as a page table.
- **The write resumes:** FUSE releases the fault and the pending write overwrites the reclaimed page-table entry.



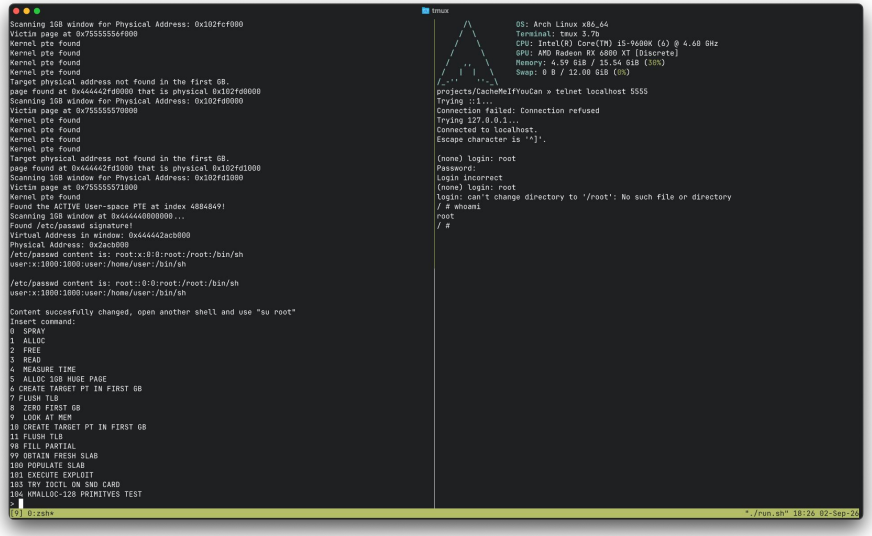
From page-table corruption to root

- **Expose physical memory:** The corrupted page-table entry creates a large accessible physical-memory window.
- **Find a controlled PTE:** Marker pages are allocated and located by scanning the exposed memory.
- **Gain arbitrary R/W:** The controlled PTE is repeatedly redirected to arbitrary physical frames.
- **Modify /etc/passwd:** The exploit locates root:x:0:0 in the page cache and removes the password marker.



Concrete outcomes and validation

- **Working exploit:** We reproduced a complete privilege-escalation chain on Linux 5.14 using CVE-2022-2588.
- **End-to-end validation:** The QEMU demonstration starts as an unprivileged user and ends with passwordless root access.
- **Reproducible environment:** Setup and execution scripts prepare the kernel source, filesystem, modules, and virtual-machine execution.
- **Debugging artifacts:** Custom GDB scripts inspect SLUB freelists, page tables, and the buddy allocator.



Challenges, learning outcomes, and takeaway

- **Problems encountered:** Allocator nondeterminism, process synchronization, repeated kernel builds, and subsystem configuration.
- **Research challenge:** The original technique had to be understood and adapted to Linux 5.14, our environment, and the selected kernel primitives.
- **Technical learning:** SLUB internals, buddy allocation, page tables, FUSE, namespaces, pipes, and kernel debugging.
- **Main takeaway:** Controlling the lifetime and reuse of a slab page can transform a constrained heap vulnerability into full privilege escalation.