EURECOM

Sophia Antipolis

Author: Paolo Monti
Supervisor: Yanick Fratantonio

SALT - A kernel heap memory reversing tool

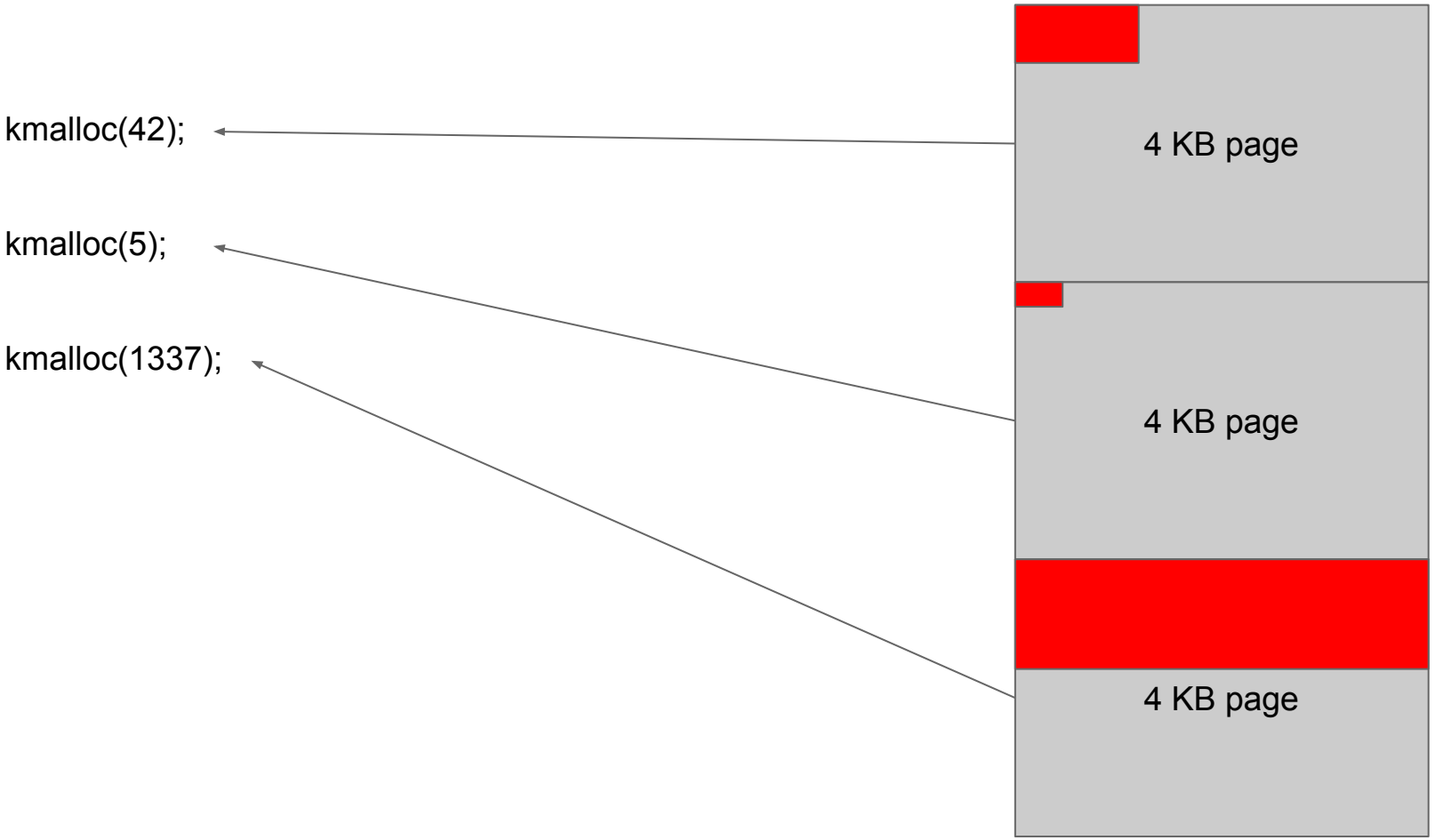
Agenda

- The SLUB allocator
- The setup
- The plugin
- The playground

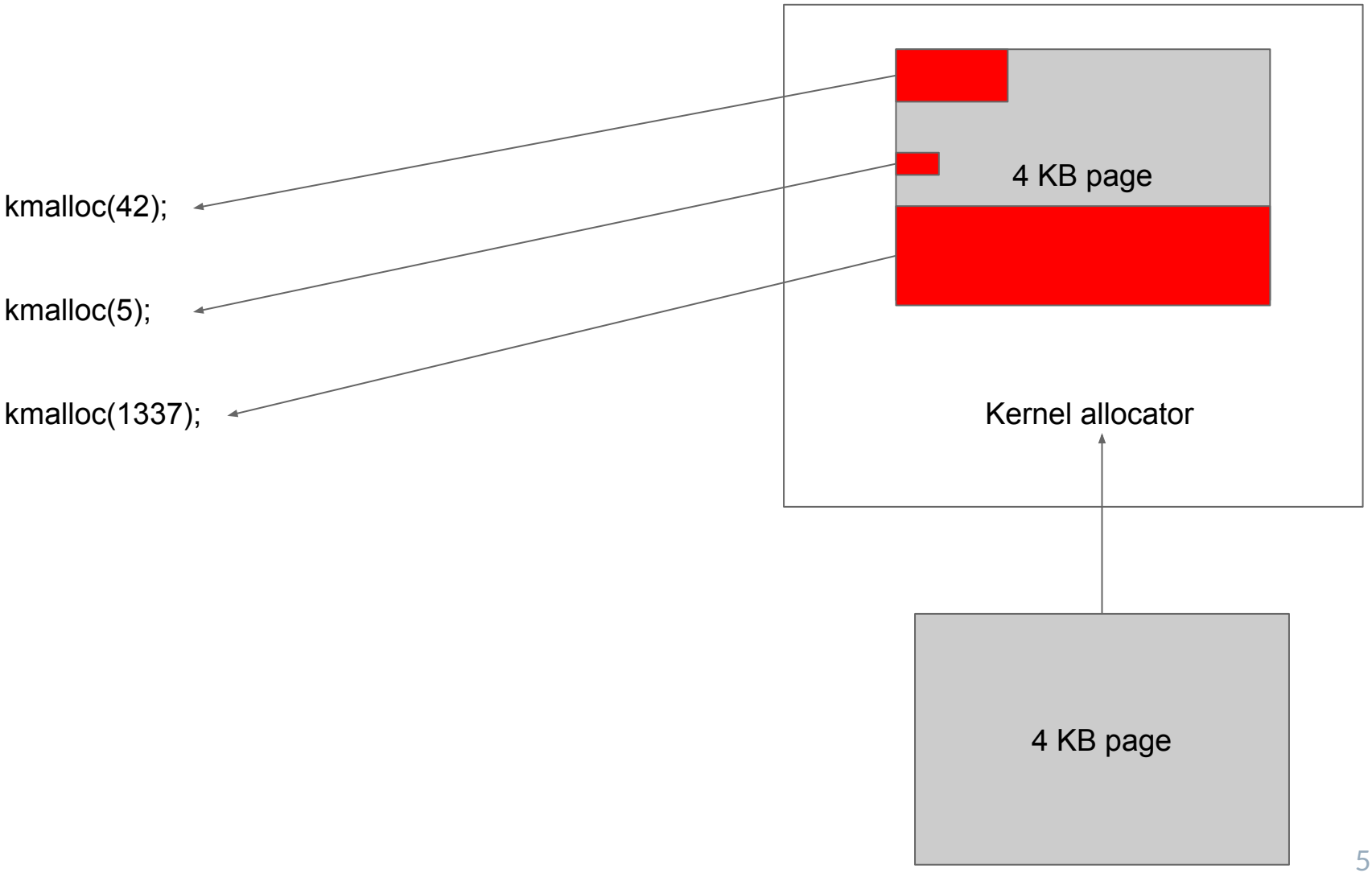
1.
SLUB internals

but first...

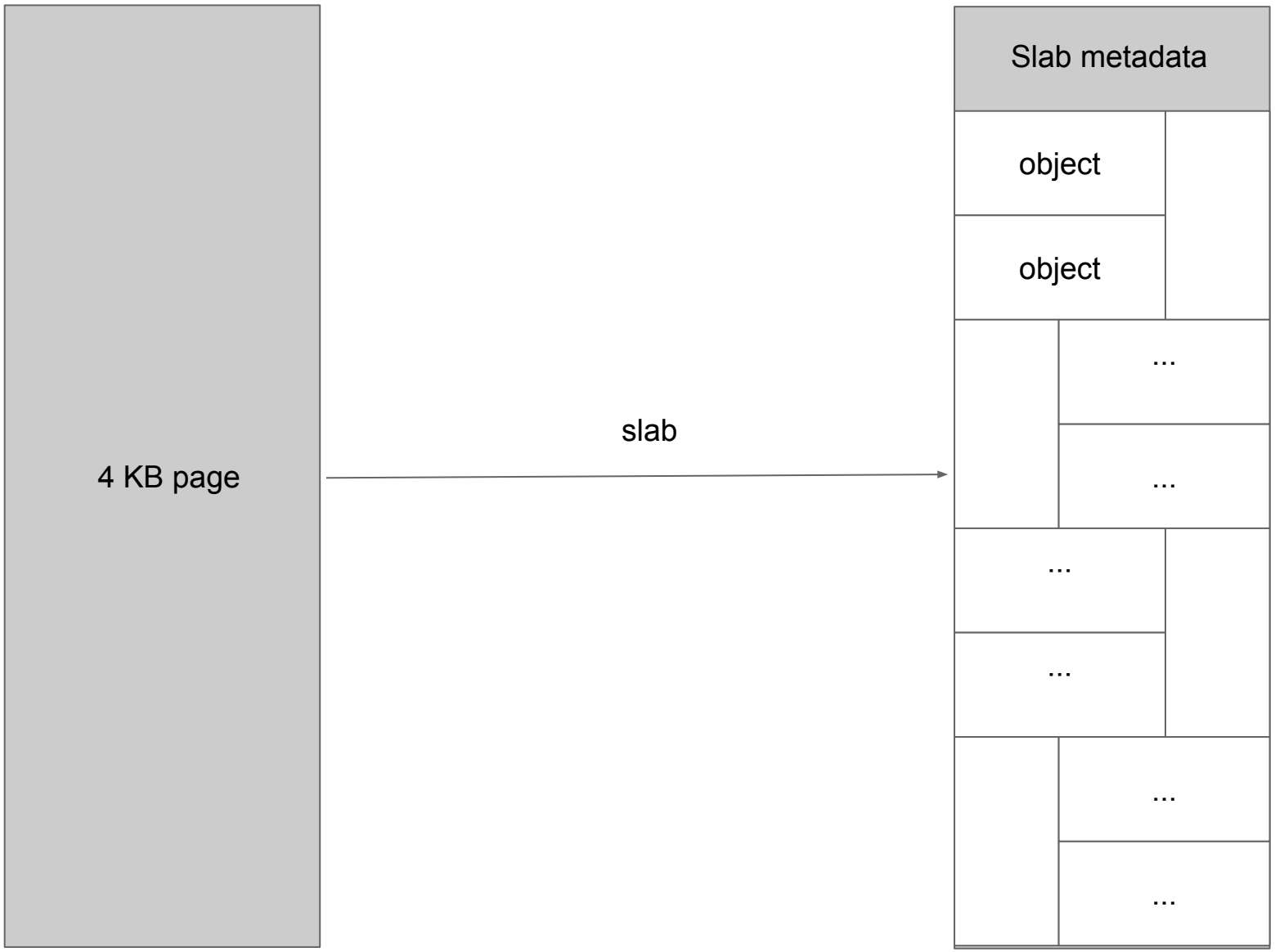
What is a kernel allocator and why do we need it?



What is a kernel allocator and why do we need it?

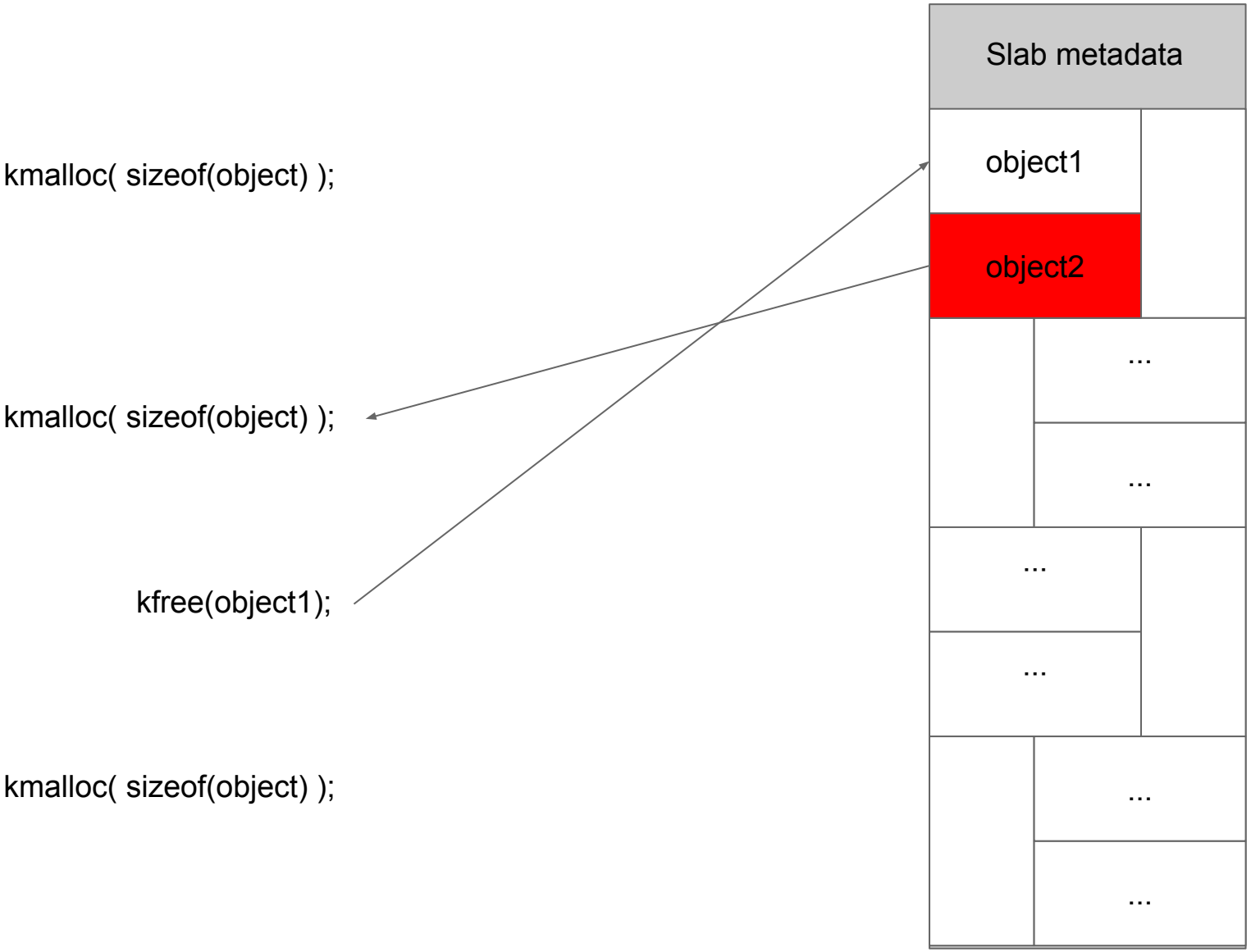


The SLAB allocator



The SLAB allocator

The SLAB allocator



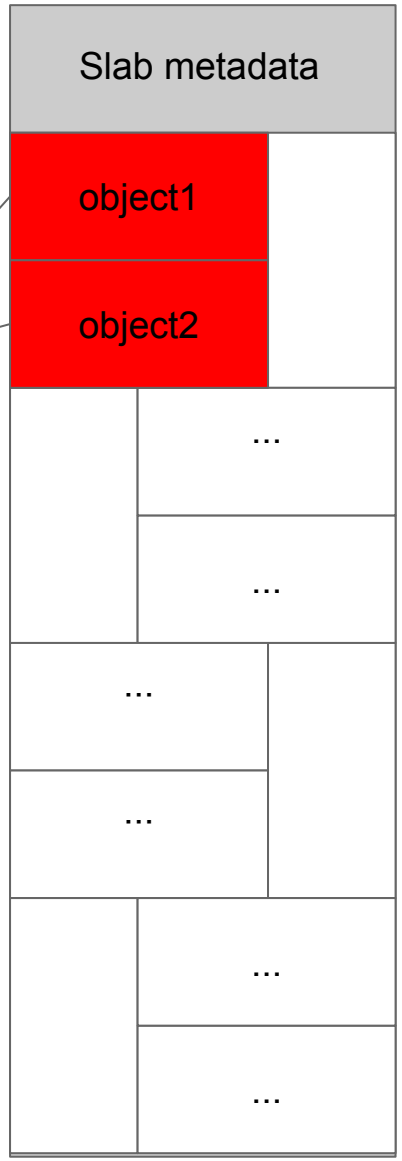
The SLAB allocator

kmalloc(sizeof(object));

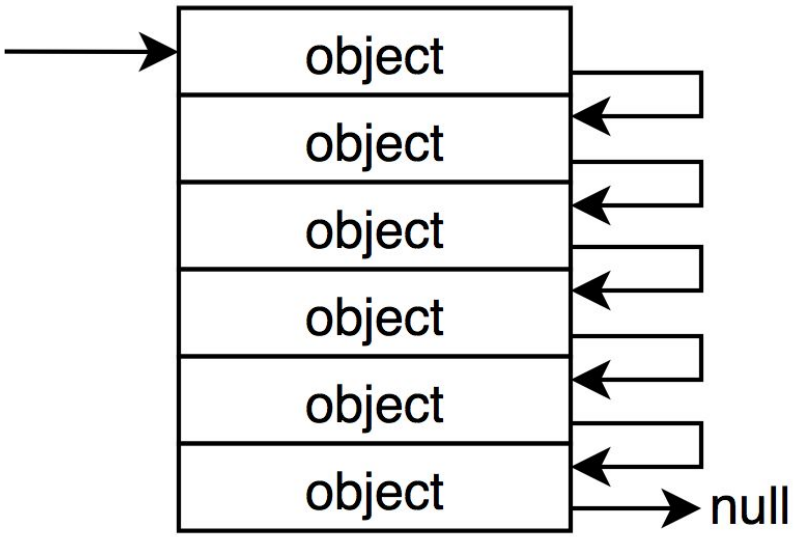
kmalloc(sizeof(object));

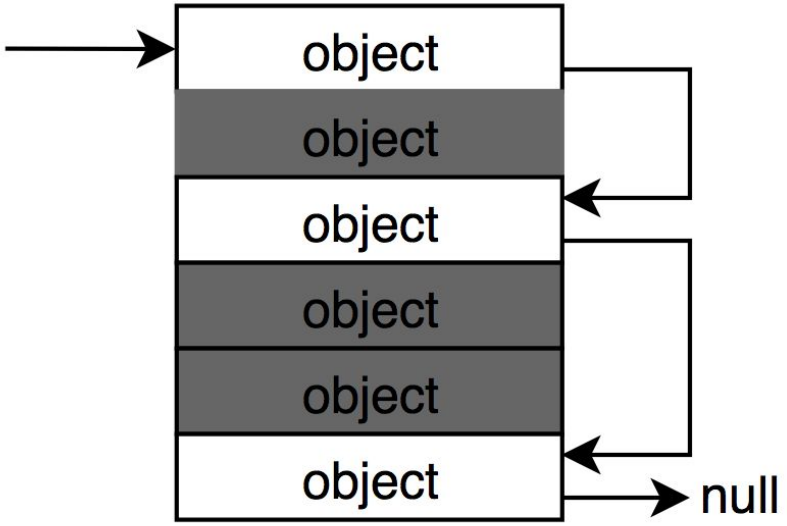
kfree(object1);

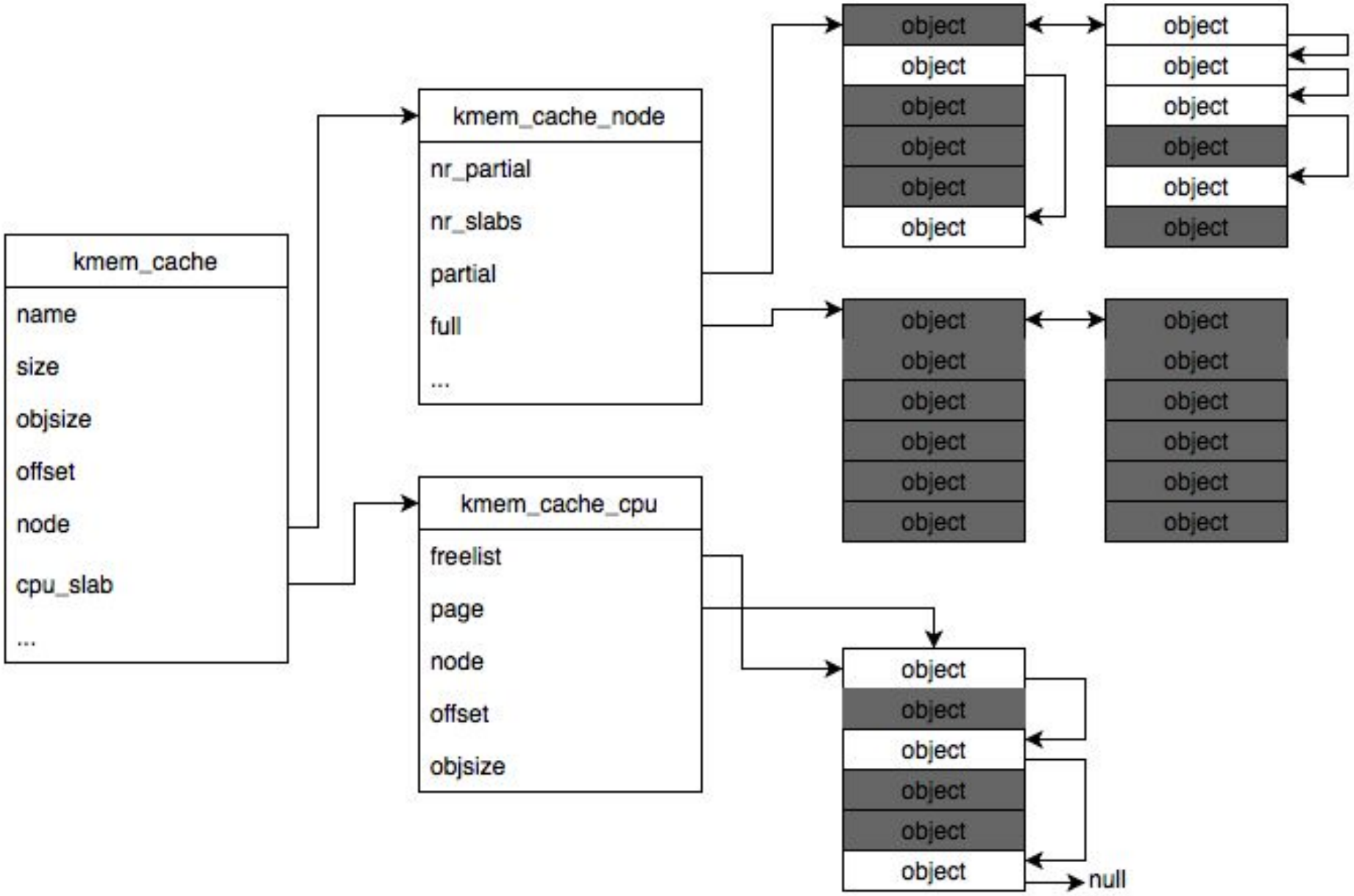
kmalloc(sizeof(object));



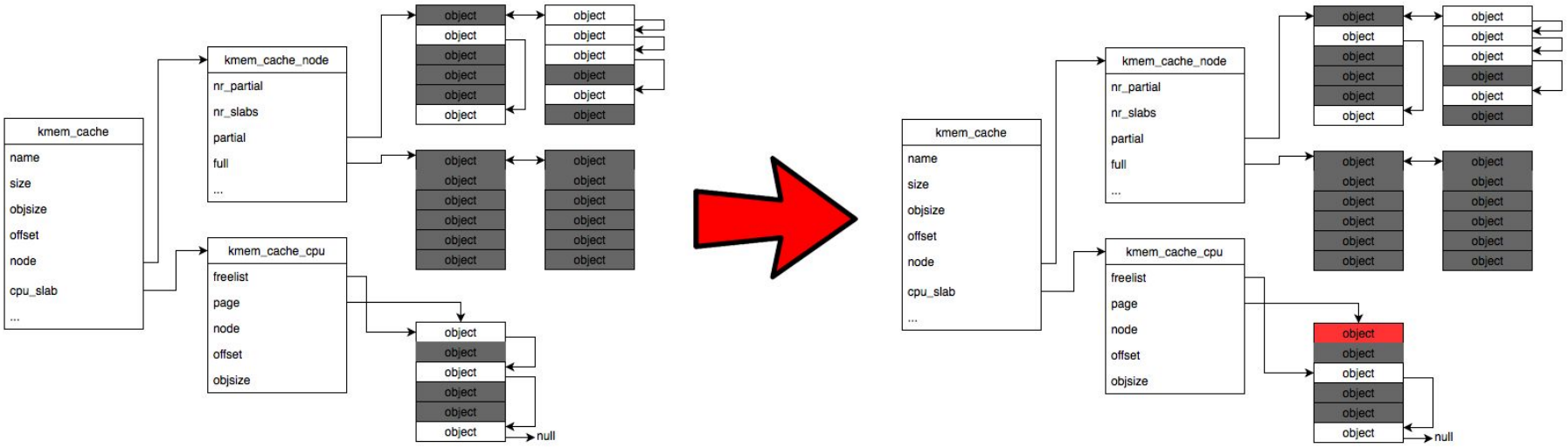
The SLUB allocator





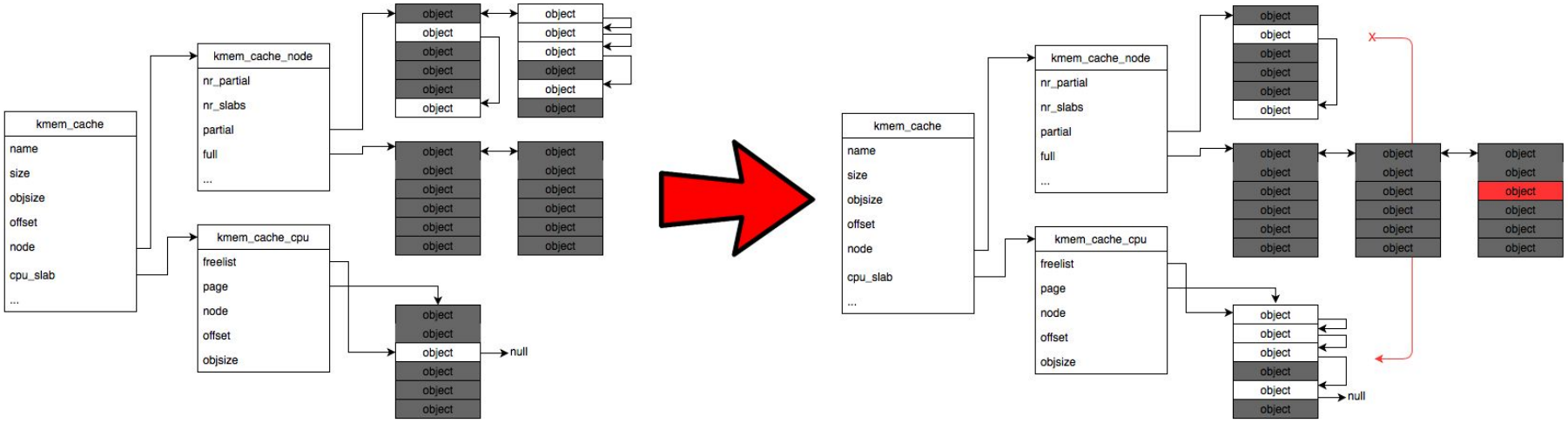

kmem_cache
name
size
objsize
offset
node
cpu_slab
...
kmem_cache_node
nr_partial
nr_slabs
partial
full
...
kmem_cache_cpu
freelist
page
node
offset
objsize
object
null

SLUB examples



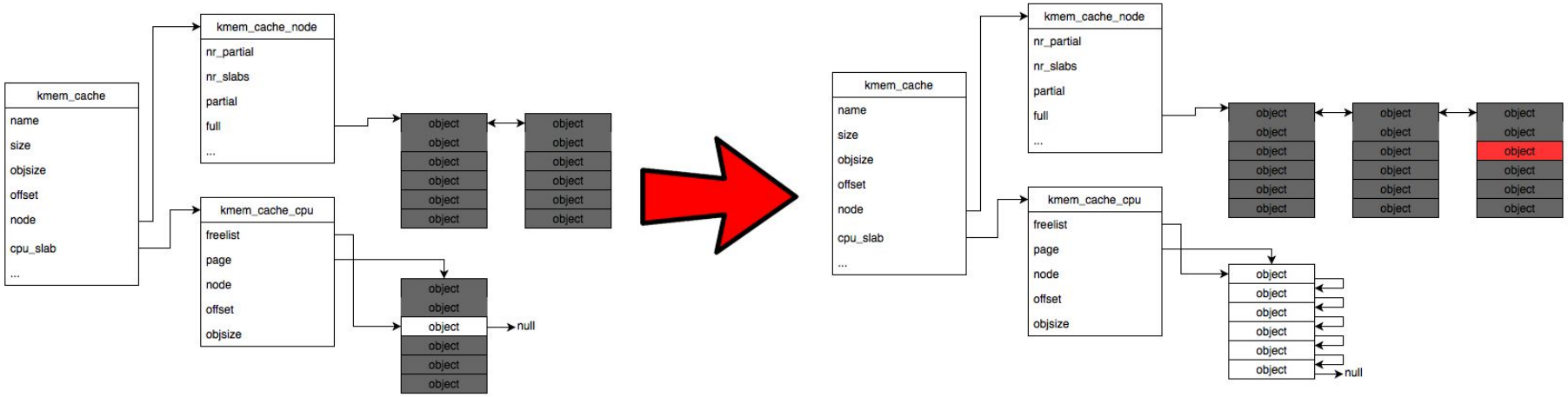
Standard allocation - the allocated object is taken from the freelist

SLUB examples



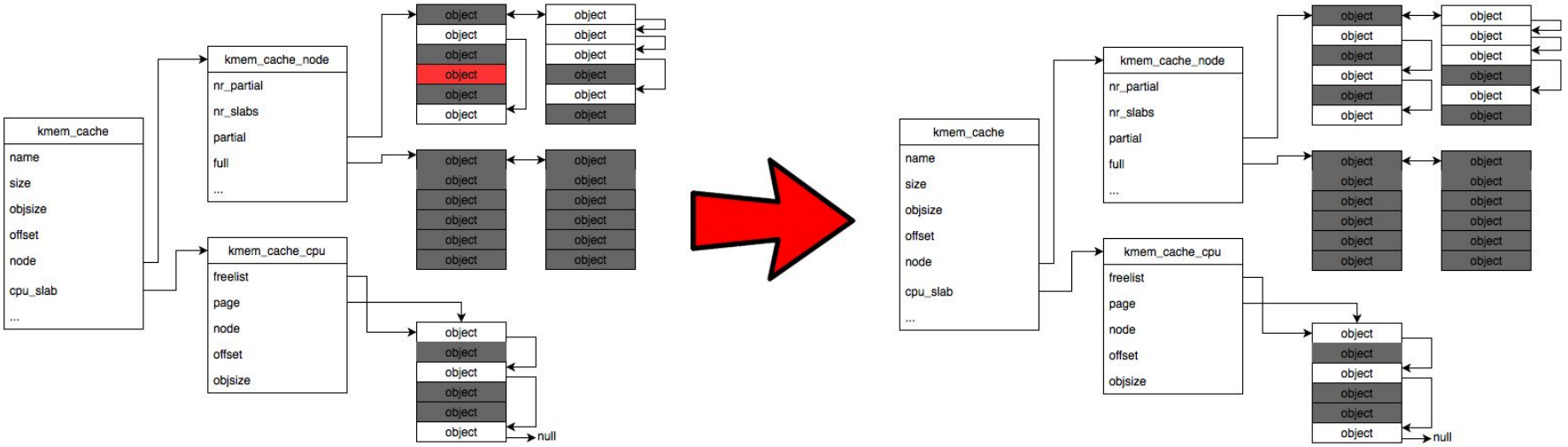
Standard allocation - the allocated object is taken from the freelist.
The active slab is now full, so it's replaced by one of the partial slabs

SLUB examples



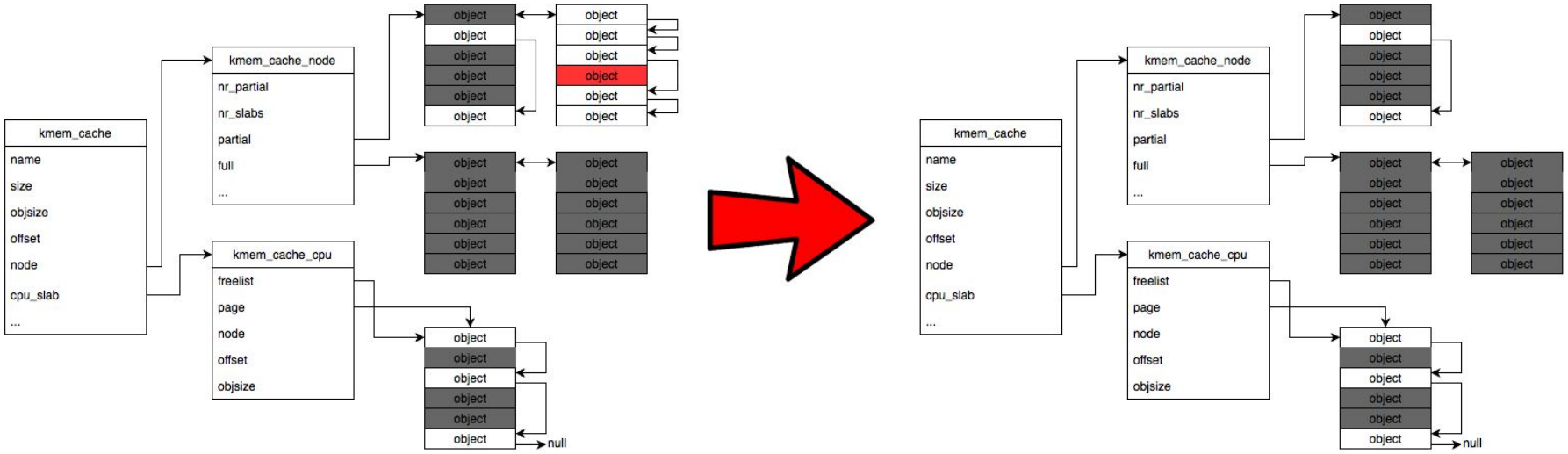
Standard allocation - the allocated object is taken from the freelist. The active slab is now full and there are no partial slabs, so a page is allocated by the kernel, passed to SLUB, then a fresh slab is created.

SLUB examples



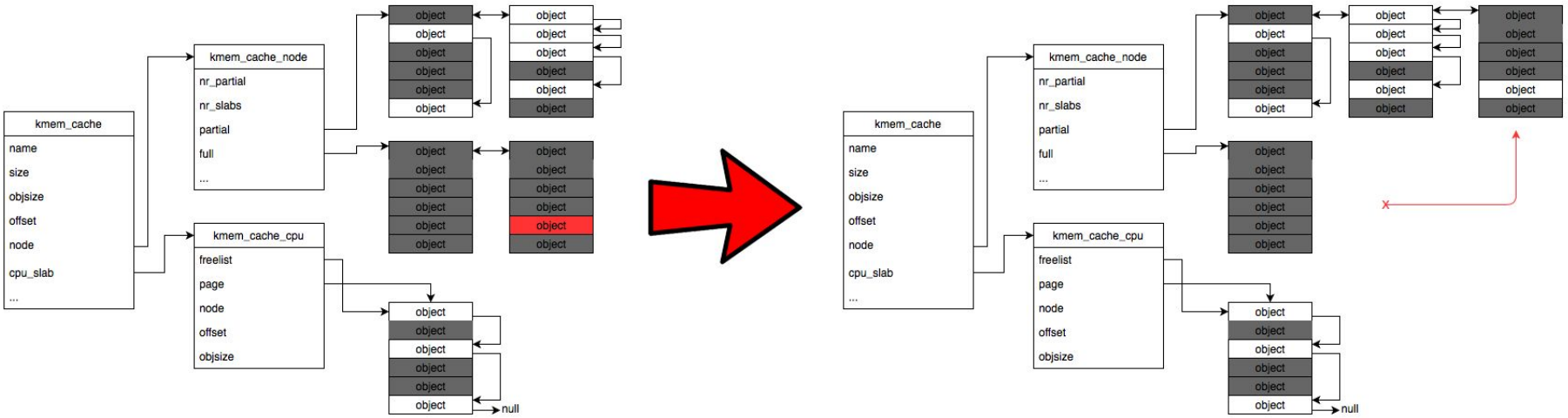
Standard deallocation - the allocated object is taken from the corresponding partial slab (or active slab).

SLUB examples



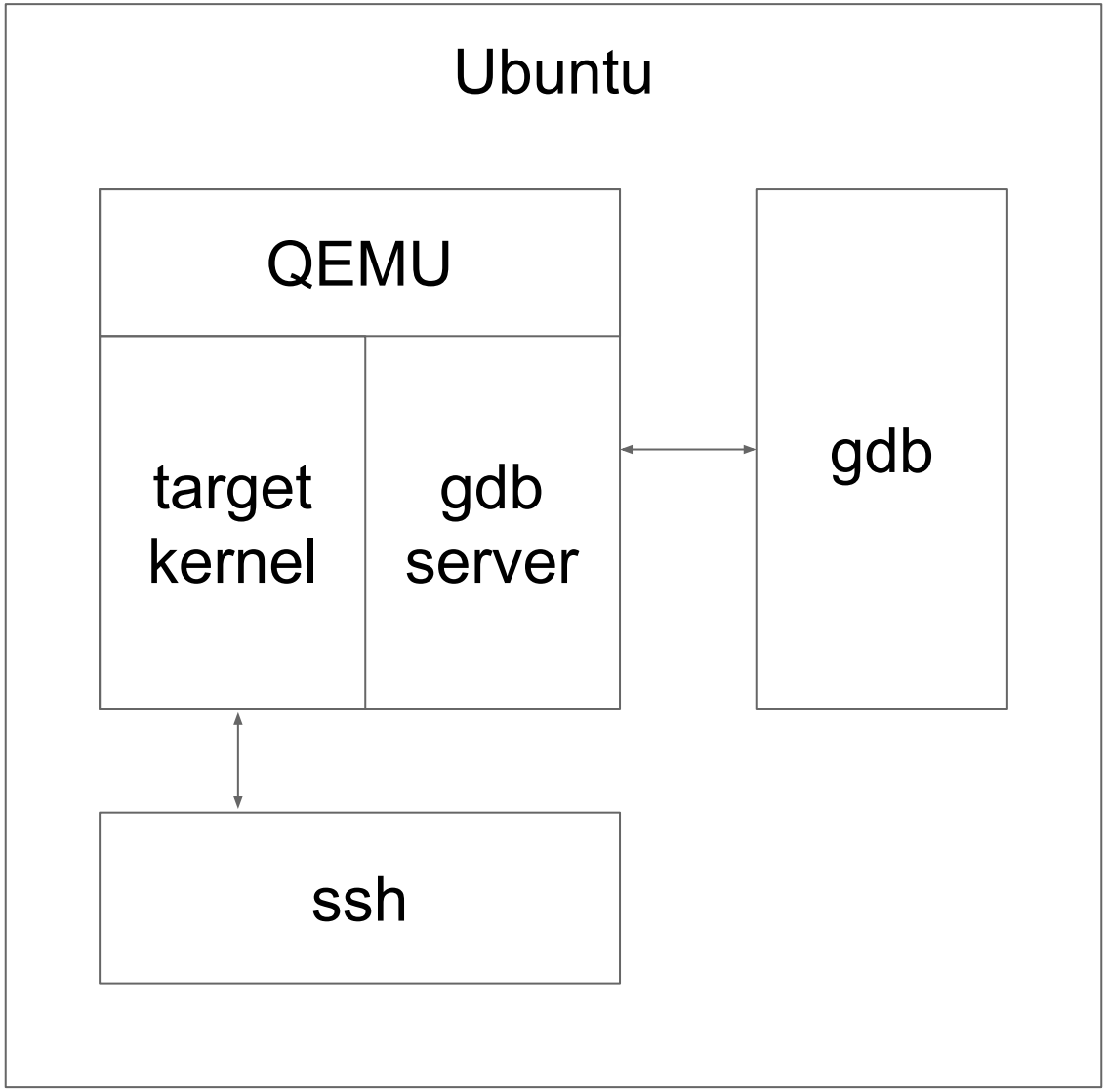
Standard deallocation - the allocated object is taken from the corresponding partial slab (or active slab) that becomes free. The page is released to the MMU.

SLUB examples

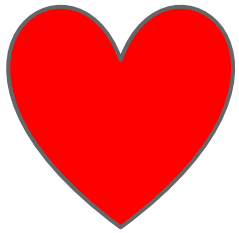


Standard deallocation - the allocated object is taken from the corresponding full slab that becomes a partial slab and is therefore moved to the partial list.

2.
My working setup


Ubuntu
QEMU
target
kernel
gdb
server
gdb
ssh

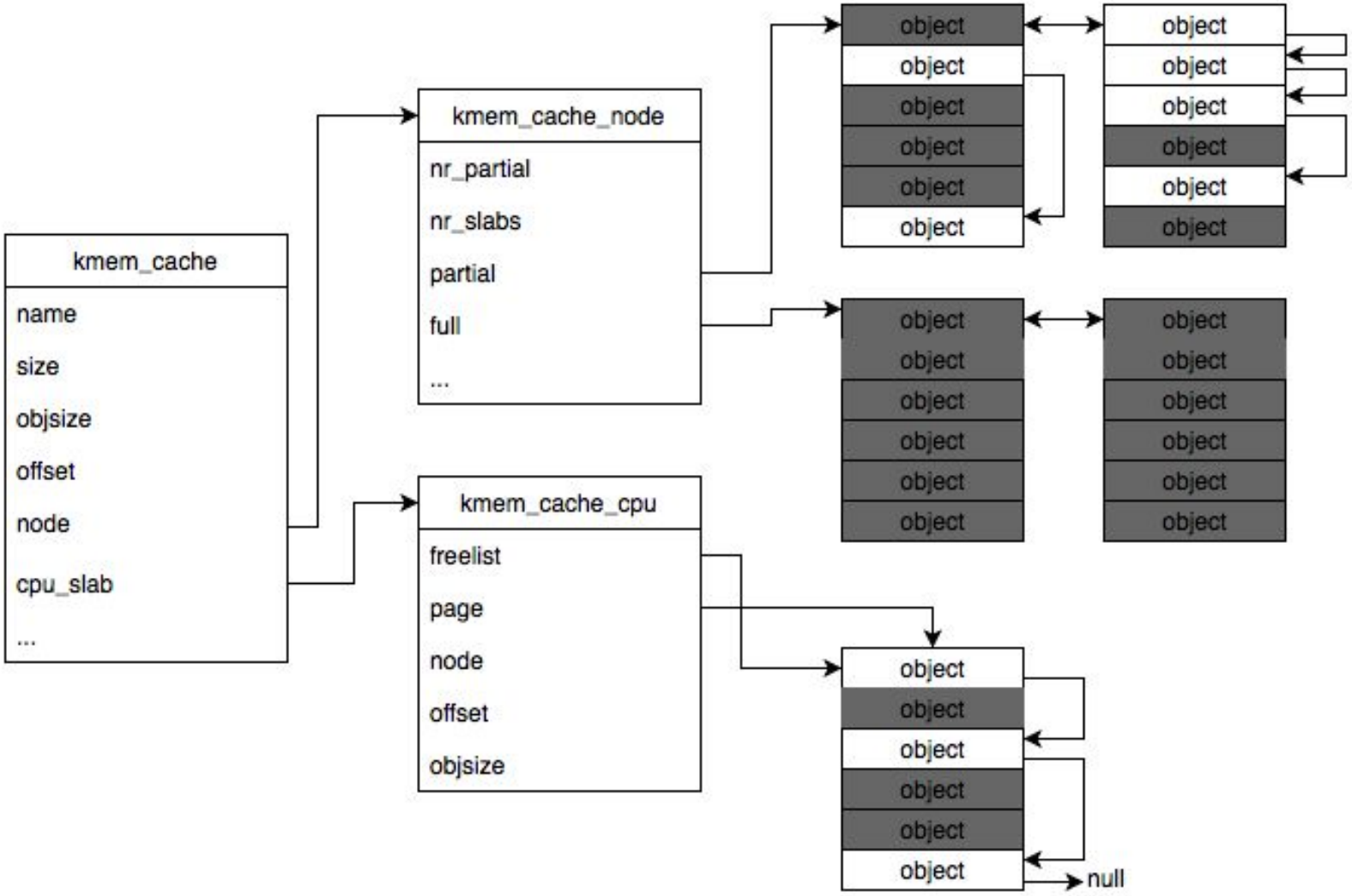
3.
The SALT tool

gdb  python

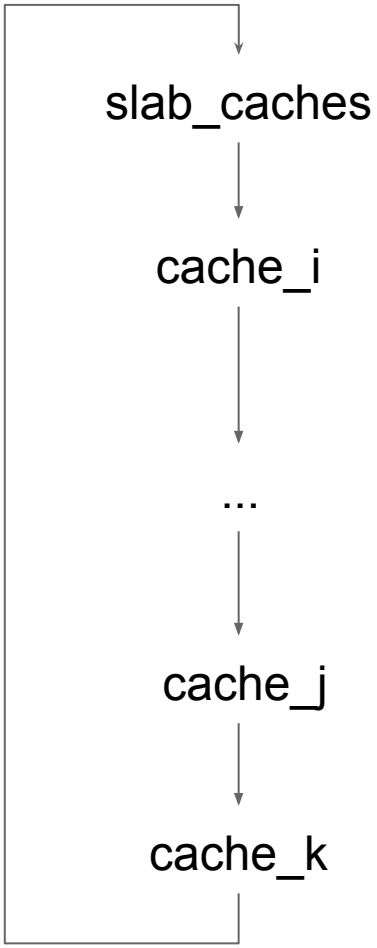
- ✓ Lookup function and structure declarations
- ✓ Read global variables
- ✓ Inspect memory and registers
- ✓ Insert and override breakpoints

- ✗ Resolve defines and macros
- ✗ Detect inline functions

Static analysis



Walking




```
pwndbg> salt walk mnt_cache
  -------------
 |             |
 |       slab_caches
 |             |
 |            ...
 |             |
 |             v
 |   name: mnt_cache
 |   first_free: 0xffff88003b79f180
 |   freelist:   0xffff88003b79f000
 |               0xffff88003b79f480
 |               0xffff88003b79f600
 |               0xffff88003b79f780
 |               0xffff88003b79f900
 |               0xffff88003b79fa80
 |               0xffff88003b79fc00
 |               0xffff88003b79fd80
 |               0x0
 |   next: 0xffff88003e090e00
 |             |
 |            ...
 |             |
 |             v
  <------------
```

Dynamic analysis

- kmalloc (int size, gfp_t flags);
- kfree (void *obj);
- kmem_cache_alloc (struct kmem_cache * cachep, gfp_t flags);
- kmem_cache_free (struct kmem_cache * cachep, void *obj);

▷ Filtering

Set rules about what events should be notified to the user.

Can filter based on process name or/and cache name.

▷ Recording

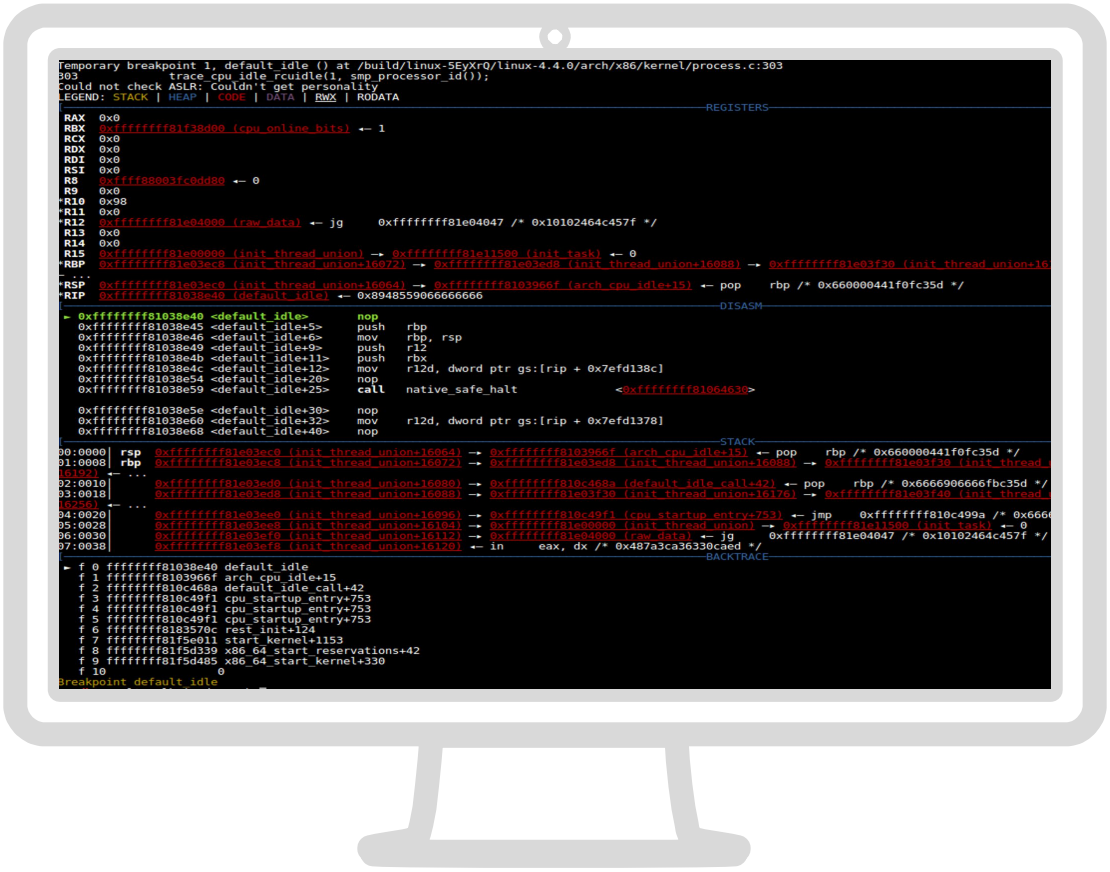
Generate a history of events. Follows filtering rules.

Can be later displayed or dumped into a file.

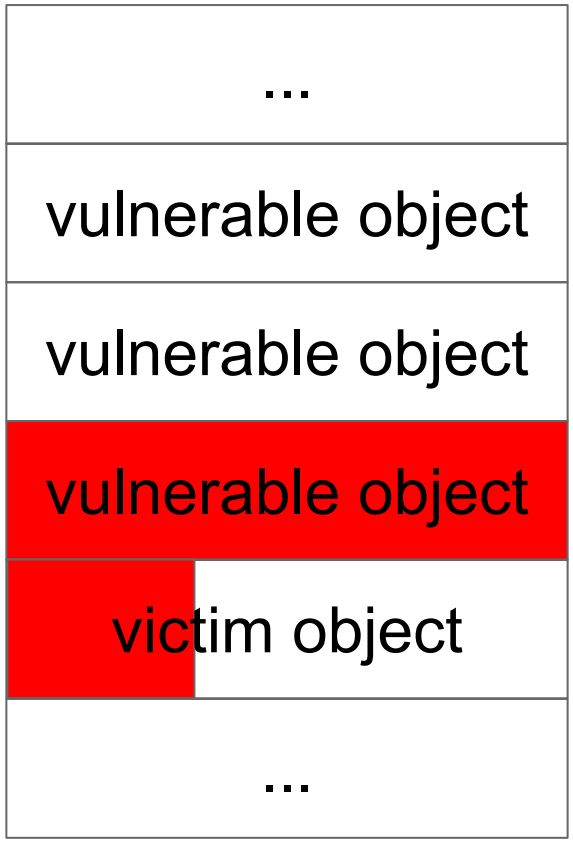
▷ Tracing

Combine filtering and recording in a single command for quick setup. Useful when only interested in understanding how a certain process behaves (e.g. sshd, cron, l33t_3xpl01t).

Demo time!


Temporary breakpoint 1, default_idle () at /build/linux-5EyXrQ/linux-4.4.0/arch/x86/kernel/process.c:303
303 trace_cpu_idle_rcuidle(1, smp_processor_id());
Could not check ASLR: Couldn't get personality
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
REGISTERS
RAX 0x0
RBX 0xffffffff81f38d00 (cpu_online_bits) ← 1
RCX 0x0
RDX 0x0
RDI 0x0
RSI 0x0
R8 0xffff88003fc0dd80 ← 0
R9 0x0
*R10 0x98
*R11 0x0
R12 0xffffffff81e04000 (raw_data) ← jg 0xffffffff81e04047 / 0x10102464c457f */
R13 0x0
R14 0x0
R15 0xffffffff81e00000 (init_thread_union) → 0xffffffff81e11500 (init_task) ← 0
*RBP 0xffffffff81e03ec8 (init_thread_union+16072) → 0xffffffff81e03ed8 (init_thread_union+16088) → 0xffffffff81e03f30 (init_thread_union+16176)
...
RSP 0xffffffff81e03ec0 (init_thread_union+16064) → 0xffffffff8103966f (arch_cpu_idle+15) ← pop rbp / 0x660000441f0fc35d */
*RIP 0xffffffff81038e40 (default_idle) ← 0x8948559066666666
DISASM
► 0xffffffff81038e40 <default_idle> nop
0xffffffff81038e45 <default_idle+5> push rbp
0xffffffff81038e46 <default_idle+6> mov rbp, rsp
0xffffffff81038e49 <default_idle+9> push r12
0xffffffff81038e4b <default_idle+11> push rbx
0xffffffff81038e4c <default_idle+12> mov r12d, dword ptr gs:[rip + 0x7efd138c]
0xffffffff81038e54 <default_idle+20> nop
0xffffffff81038e59 <default_idle+25> call native_safe_halt <0xffffffff81064630>
0xffffffff81038e5e <default_idle+30> nop
0xffffffff81038e60 <default_idle+32> mov r12d, dword ptr gs:[rip + 0x7efd1378]
0xffffffff81038e68 <default_idle+40> nop
STACK
00:0000| rsp 0xffffffff81e03ec0 (init_thread_union+16064) → 0xffffffff8103966f (arch_cpu_idle+15) ← pop rbp /* 0x660000441f0fc35d */
01:0008| rbp 0xffffffff81e03ec8 (init_thread_union+16072) → 0xffffffff81e03ed8 (init_thread_union+16088) → 0xffffffff81e03f30 (init_thread_union+16176) ← ...
02:0010| 0xffffffff81e03ed0 (init_thread_union+16080) → 0xffffffff810c468a (default_idle_call+42) ← pop rbp /* 0x6666906666fbc35d */
03:0018| 0xffffffff81e03ed8 (init_thread_union+16088) → 0xffffffff81e03f30 (init_thread_union+16176) → 0xffffffff81e05f40 (init_thread_union+24896) ← ...
04:0020| 0xffffffff81e03ee0 (init_thread_union+16096) → 0xffffffff810c49f1 (cpu_startup_entry+753) ← jmp 0xffffffff810c499a /* 0x666
05:0028| 0xffffffff81e03ee8 (init_thread_union+16104) → 0xffffffff81e00000 (init_thread_union) → 0xffffffff81e11500 (init_task) ← 0
06:0030| 0xffffffff81e03ef0 (init_thread_union+16112) → 0xffffffff81e04000 (raw_data) ← jg 0xffffffff81e04047 /* 0x10102464c457f */
07:0038| 0xffffffff81e03ef8 (init_thread_union+16120) ← in eax, dx /* 0x487a3ca36330caed */
BACKTRACE
► f 0 ffffffff81038e40 default_idle
f 1 ffffffff8103966f arch_cpu_idle+15
f 2 ffffffff810c468a default_idle_call+42
f 3 ffffffff810c49f1 cpu_startup_entry+753
f 4 ffffffff810c49f1 cpu_startup_entry+753
f 5 ffffffff810c49f1 cpu_startup_entry+753
f 6 ffffffff8183570c rest_init+124
f 7 ffffffff81f5e011 start_kernel+1153
f 8 ffffffff81f5d339 x86_64_start_reservations+42
f 9 ffffffff81f5d485 x86_64_start_kernel+330
f 10 0
Breakpoint default_idle

Common exploit scenario



4.
The playground

Idea: a way to play with your kernel's memory allocator, in order to better understand how it works

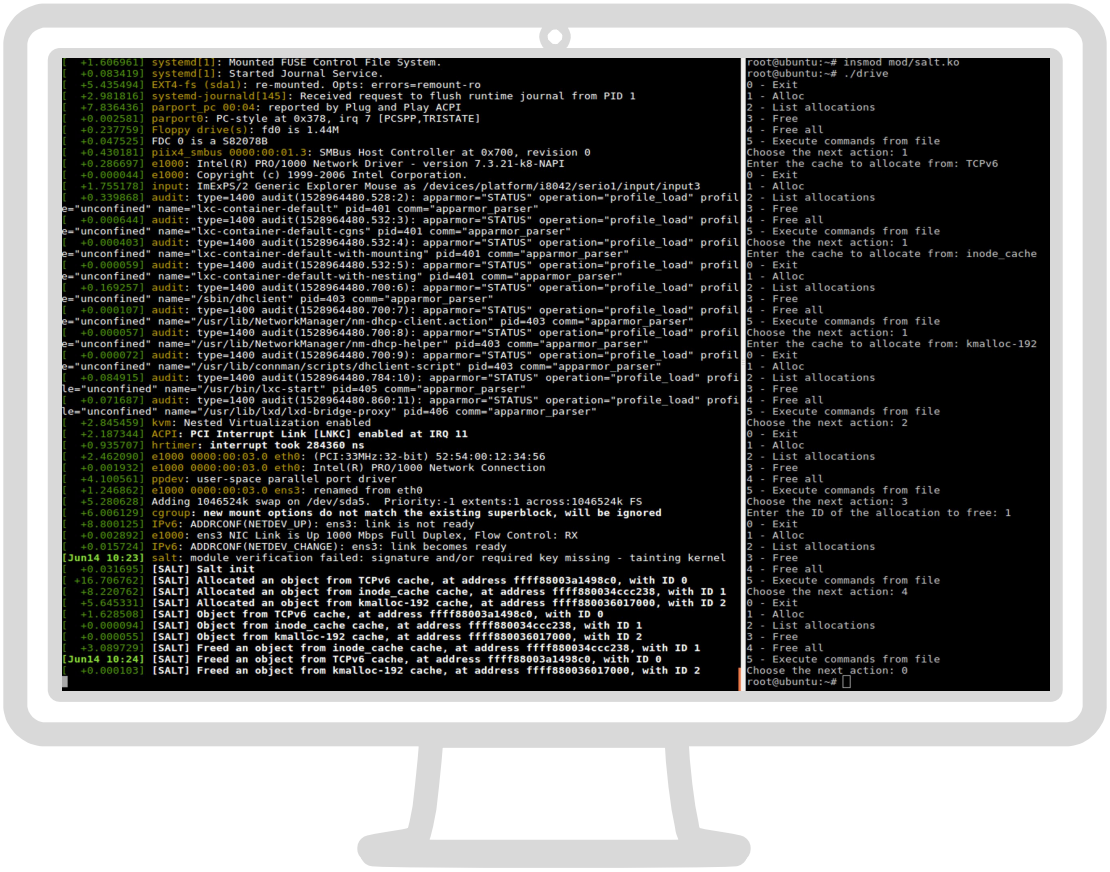
Usage: teaching purposes, debugging your own kernel code, reversing a system, testing exploits

Solution: a kernel module that is implemented as a device driver and can receive commands through the ioctl system call

My implementation:

- Can generate allocations from any cache in the system. Each allocation is identified by a unique ID.
- Can list all current allocations.
- Can free any allocated object by referring to it through its ID.
- Can free all current allocations.
- Can execute commands from a file, to repeat tests.
- Can be used for a future CTF, since it's probably buggy.

Demo time!


[+1.606961] systemd[1]: Mounted FUSE Control File System.
[+0.083419] systemd[1]: Started Journal Service.
[+5.435494] EXT4-fs (sda1): re-mounted. Opts: errors=remount-ro
[+2.981816] systemd-journald[145]: Received request to flush runtime journal from PID 1
[+7.836436] parport_pc 00:04: reported by Plug and Play ACPI
[+0.002581] parport0: PC-style at 0x378, irq 7 [PCSPP,TRISTATE]
[+0.237759] Floppy drive(s): fd0 is 1.44M
[+0.047525] FDC 0 is a S82078B
[+0.430181] piix4_smbus 0000:00:01.3: SMBus Host Controller at 0x700, revision 0
[+0.286697] e1000: Intel(R) PRO/1000 Network Driver - version 7.3.21-k8-NAPI
[+0.000044] e1000: Copyright (c) 1999-2006 Intel Corporation.
[+1.755178] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/serio1/input/input3
[+0.339868] audit: type=1400 audit(1528964480.528:2): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="lxc-container-default" pid=401 comm="apparmor_parser"
[+0.000644] audit: type=1400 audit(1528964480.532:3): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="lxc-container-default-cgns" pid=401 comm="apparmor_parser"
[+0.000403] audit: type=1400 audit(1528964480.532:4): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="lxc-container-default-with-mounting" pid=401 comm="apparmor_parser"
[+0.000059] audit: type=1400 audit(1528964480.532:5): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="lxc-container-default-with-nesting" pid=401 comm="apparmor_parser"
[+0.169257] audit: type=1400 audit(1528964480.700:6): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="/sbin/dhclient" pid=403 comm="apparmor_parser"
[+0.000107] audit: type=1400 audit(1528964480.700:7): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-client.action" pid=403 comm="apparmor_parser"
[+0.000057] audit: type=1400 audit(1528964480.700:8): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-helper" pid=403 comm="apparmor_parser"
[+0.000072] audit: type=1400 audit(1528964480.700:9): apparmor="STATUS" operation="profile_load" profil
e="unconfined" name="/usr/lib/connman/scripts/dhclient-script" pid=403 comm="apparmor_parser"
[+0.084915] audit: type=1400 audit(1528964480.784:10): apparmor="STATUS" operation="profile_load" profi
le="unconfined" name="/usr/bin/lxc-start" pid=405 comm="apparmor_parser"
[+0.071687] audit: type=1400 audit(1528964480.860:11): apparmor="STATUS" operation="profile_load" profi
le="unconfined" name="/usr/lib/lxd/lxd-bridge-proxy" pid=406 comm="apparmor_parser"
[+2.845459] kvm: Nested Virtualization enabled
[+2.187344] ACPI: PCI Interrupt Link [LNKC] enabled at IRQ 11
[+0.935707] hrtimer: interrupt took 284360 ns
[+2.462090] e1000 0000:00:03.0 eth0: (PCI:33MHz:32-bit) 52:54:00:12:34:56
[+0.001932] e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Connection
[+4.100561] ppdev: user-space parallel port driver
[+1.246862] e1000 0000:00:03.0 ens3: renamed from eth0
[+5.280628] Adding 1046524k swap on /dev/sda5. Priority:-1 extents:1 across:1046524k FS
[+6.006129] cgroup: new mount options do not match the existing superblock, will be ignored
[+8.800125] IPv6: ADDRCONF(NETDEV_UP): ens3: link is not ready
[+0.002892] e1000: ens3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[+0.015724] IPv6: ADDRCONF(NETDEV_CHANGE): ens3: link becomes ready
[Jun14 10:23] salt: module verification failed: signature and/or required key missing - tainting kernel
[+0.031695] [SALT] Salt init
[+16.706762] [SALT] Allocated an object from TCPv6 cache, at address ffff88003a1498c0, with ID 0
[+8.220762] [SALT] Allocated an object from inode_cache cache, at address ffff880034ccc238, with ID 1
[+5.645331] [SALT] Allocated an object from kmalloc-192 cache, at address ffff880036017000, with ID 2
[+1.628508] [SALT] Object from TCPv6 cache, at address ffff88003a1498c0, with ID 0
[+0.000094] [SALT] Object from inode_cache cache, at address ffff880034ccc238, with ID 1
[+0.000055] [SALT] Object from kmalloc-192 cache, at address ffff880036017000, with ID 2
[+3.089729] [SALT] Freed an object from inode_cache cache, at address ffff880034ccc238, with ID 1
[Jun14 10:24] [SALT] Freed an object from TCPv6 cache, at address ffff88003a1498c0, with ID 0
[+0.000103] [SALT] Freed an object from kmalloc-192 cache, at address ffff880036017000, with ID 2
root@ubuntu:~# insmod mod/salt.ko
root@ubuntu:~# ./drive
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 1
Enter the cache to allocate from: TCPv6
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 1
Enter the cache to allocate from: inode_cache
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 1
Enter the cache to allocate from: kmalloc-192
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 2
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 3
Enter the ID of the allocation to free: 1
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 4
0 - Exit
1 - Alloc
2 - List allocations
3 - Free
4 - Free all
5 - Execute commands from file
Choose the next action: 0
root@ubuntu:~#

Thanks!

Any questions?

You can find me at:

GitHub -> PaoloMonti42

Mattermost -> @pmonti

Email -> monti@eurecom.fr