Three Dark Clouds over the Android Kernel

Jun Yao @ Alpha Lab

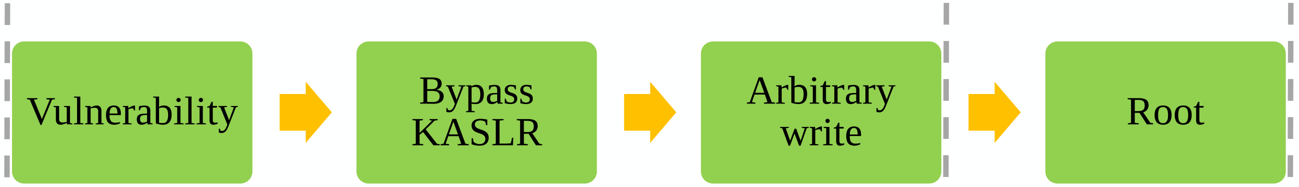
About me

- Security researcher in Alpha Lab of 360
- Focus on the Android/Linux kernel
  - ✓ Bug hunting
  - ✓ Writing exploits
  - ✓ Researching Mitigations

Contents

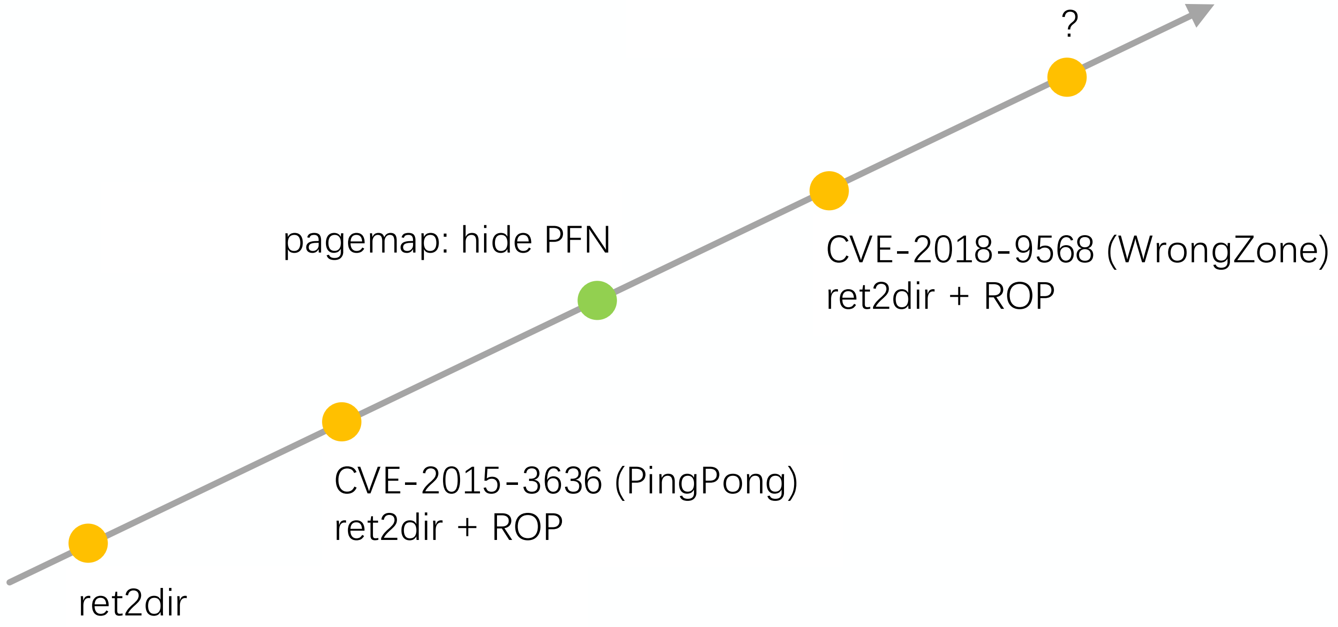
Contents

Classic attack path



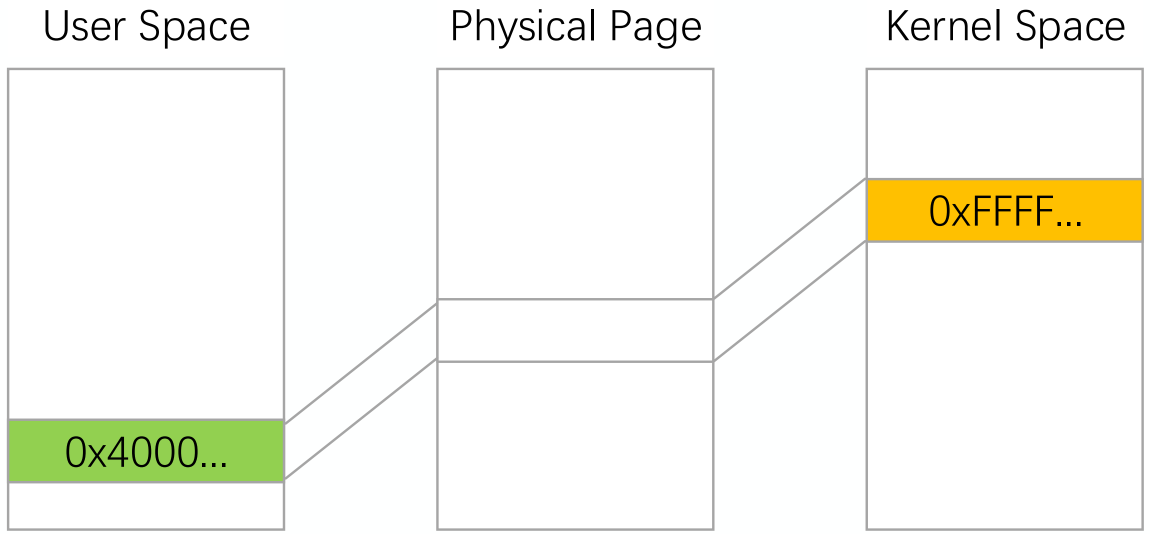
- **Methods**
  - ✓ ret2usr
  - ✓ ROP/JOP
  - ✓ …

The ret2dir


?
pagemap: hide PFN
CVE-2018-9568 (WrongZone)
ret2dir + ROP
CVE-2015-3636 (PingPong)
ret2dir + ROP
ret2dir

The ret2dir

- **Double Mapping**



https://cs.brown.edu/~vpk/papers/ret2dir.sec14.pdf

The ret2dir

- **User can calculate the kernel address**
  1. Get PFN from /proc/self/pagemap
  2. kaddr = (PFN << PAGE_SHIFT) – PHYS_OFFSET

     + PAGE_OFFSET
- **The kernel mapping attributes**
  - ✓ qcom: RW
  - ✓ mtk: RWX (CVE-2016-0820)

The ret2dir

- **Hide PFN in the /proc/self/pagemap**

```
+static int pagemap_open(struct inode *inode, struct file *file)
+{
+       /* do not disclose physical addresses: attack vector */
+       if (!capable(CAP_SYS_ADMIN))
+               return -EPERM;
+       return 0;
+}
+
 const struct file_operations proc_pagemap_operations = {
        .llseek     = mem_lseek, /* borrow this */
        .read       = pagemap_read,
+       .open       = pagemap_open,
 };
```

https://android.googlesource.com/kernel/msm/+/718c232053ec81a4a732ed50ba0224ae512bb694%5E%21/

The ret2dir

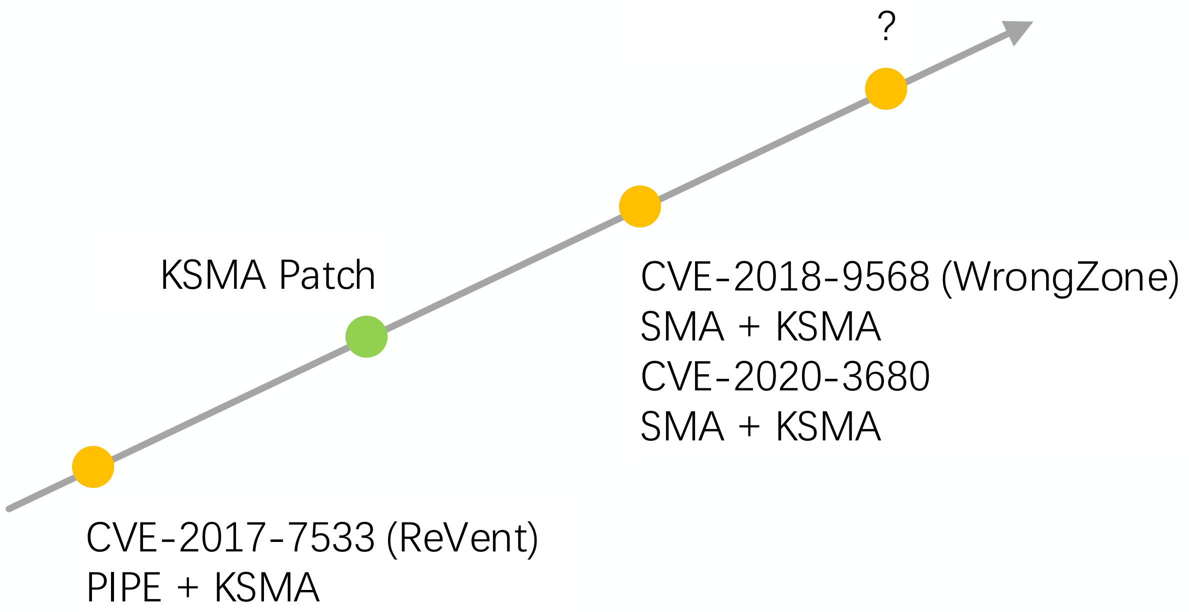
- **It still works**
  - ✓ CVE-2015-3636[1]
  - ✓ CVE-2018-9568[2]
- **The key is information leakage**

[1] https://www.blackhat.com/docs/us-15/materials/us-15-Xu-Ah-Universal-Android-Rooting-Is-Back.pdf

[2] https://github.com/ThomasKing2014/slides/blob/master/Building%20universal%20Android%1920rooting%20with%20a%20type%20confusion%20vulnerability.pdf

The KSMA

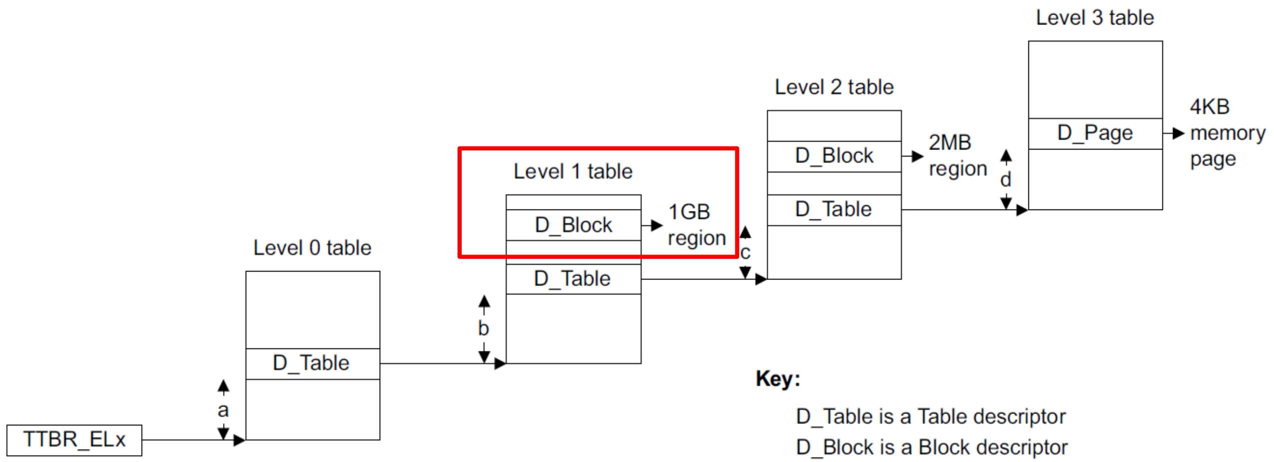
- **Kernel Space Mirror Attack**



https://i.blackhat.com/briefings/asia/2018/asia-18-WANG-KSMA-Breaking-Android-kernel-isolation-and-Rooting-with-ARM-MMU-features.pdf

The KSMA

- **The block entry**



Key:

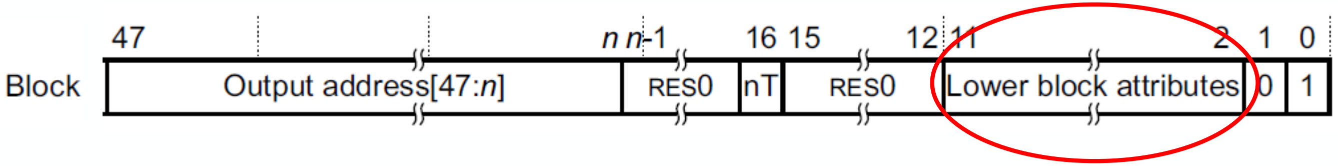
D_Table is a Table descriptor
D_Block is a Block descriptor
D_Page is a Page descriptor

a	Indexed by IA[*n*:39], where IA width is (*n*+1) bits
b	Indexed by IA[38:30]
c	Indexed by IA[29:21]
d	Indexed by IA[20:12]

https://documentation-service.arm.com/static/5f20515cbb903e39c84dc459?token=

The KSMA

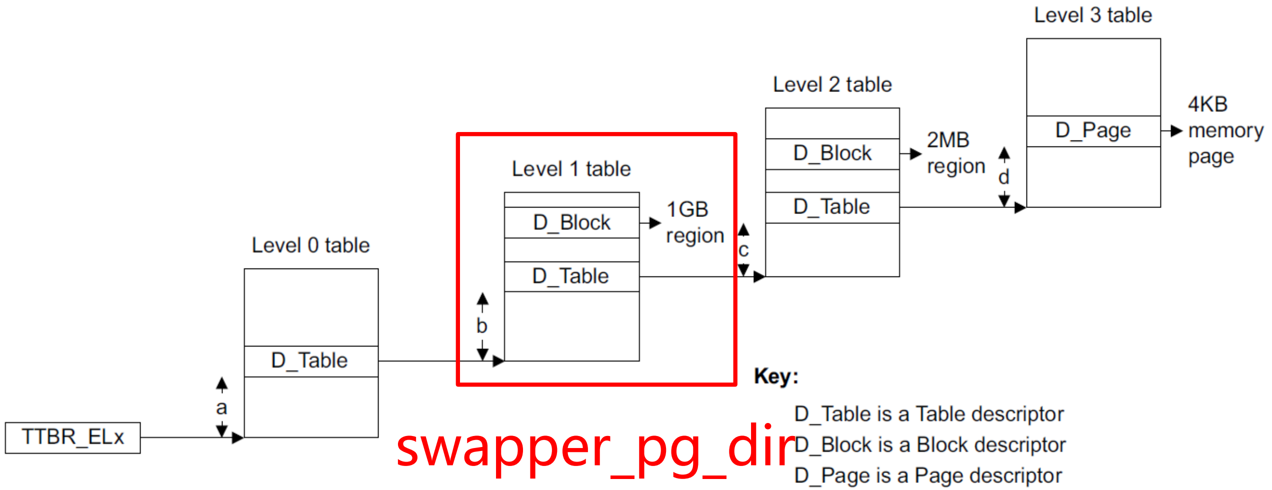
- **The AP (Access Permissions) attribute**



AP[2:1]	Access from higher Exception level	Access from EL0
00	Read/write	None
01	Read/write	Read/write
10	Read-only	None
11	Read-only	Read-only

The KSMA

- Android: 39-bits VA & 4K page



Key:

D_Table is a Table descriptor
D_Block is a Block descriptor
D_Page is a Page descriptor

a	Indexed by IA[*n*:39], where IA width is (*n*+1) bits
b	Indexed by IA[38:30]
c	Indexed by IA[29:21]
d	Indexed by IA[20:12]

The KSMA

- **The position of the swapper_pg_dir is fixed**

```
arch/arm64/kernel/vmlinux.lds.S

     BSS_SECTION(0, 0, 0)

     . = ALIGN(PAGE_SIZE);
     idmap_pg_dir = .;
     . += IDMAP_DIR_SIZE;
     swapper_pg_dir = .;
     . += SWAPPER_DIR_SIZE;
```

- **Arbitrary write is needed**

The PIPE

- Proposed in 2017
- Has the characteristics of TOCTOU
- Converts heap problems to arbitrary reading and writing

https://pacsec.jp/psj17/PSJ2017_DiShen_Pacsec_FINAL.pdf

The PIPE

- **Allocate iovec on the heap**

```
fs/read_write.c

ssize_t rw_copy_check_uvector(int type, const struct iovec __user * uvector,
                              unsigned long nr_segs, unsigned long fast_segs,
                              struct iovec *fast_pointer, ...) {
    [...]
    if (nr_segs > fast_segs) { // UIO_FASTIOV == 8
        iov = kmalloc(nr_segs*sizeof(struct iovec), GFP_KERNEL);
        [...]
    }
    if (copy_from_user(iov, uvector, nr_segs*sizeof(*uvector))) {
        ret = -EFAULT;
        goto out;
    }
}
```

The PIPE

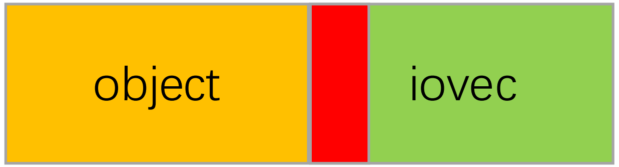
- **Check the iovec passed in by the user**

```
fs/read_write.c

ssize_t rw_copy_check_uvector(int type, const struct iovec __user * uvector,
                              unsigned long nr_segs, unsigned long fast_segs,
                              struct iovec *fast_pointer, ...) {
    [...]
    for (seg = 0; seg < nr_segs; seg++) {
        void __user *buf = iov[seg].iov_base;
        ssize_t len = (ssize_t)iov[seg].iov_len;

        if (type >= 0 && unlikely(!access_ok(vrfy_dir(type), buf, len))) {}
        ret += len;
    }
}
```

The PIPE

- **We can block the pipe**
  1. There is no contents when reading
  2. There is no space when writing

- **Overwrite the iovec**



The PIPE

- Does NOT check when copying from/to the user

```
lib/iov_iter.c

static size_t copy_page_to_iter_iovec(..) {
     [...]
     left = __copy_to_user_inatomic(buf, from, copy);
     copy -= left;
     skip += copy;
     from += copy;
     bytes -= copy;
     [...]
}
```

The PIPE

- **Two mitigations**
  1. UAO (User Access Override)
  2. uaccess_mask_ptr()

```
arch/arm64/include/asm/uaccess.h

static inline void __user *__uaccess_mask_ptr(const void __user *ptr)
{
	asm volatile(
	"	bics xzr, %1, %2\n"
	"	csel %0, %1, xzr, eq\n"
	: "=&r" (safe_ptr)
	: "r" (ptr), "r" (current_thread_info()->addr_limit)
	: "cc");
	return safe_ptr;
}
```

https://developer.arm.com/docs/ddi0595/h/aarch64-system-registers/uao

The SMA

- **SLAB Mirror Attack**
  - ✓ Two objects share the same SLAB
  - ✓ Bypass KASLR
  - ✓ Write kernel arbitrary (+KSMA = ROOT)

Contents

CVE-2020-3680

- **Timeline**

- 2019-11-10 Report to Google
- 2020-4-22 The CVE ID is CVE-2020-3680
- 2020-5-4 Public disclosure on the Qualcomm Security Bulletin

CVE-2020-3680

- Qualcomm ADSPRPC driver

```
drivers/char/adsprpc.c

int fastrpc_internal_mmap(...) {
	mutex_lock(&fl->map_mutex);
	[...]
	mutex_lock(&fl->fl_map_mutex);
	VERIFY(err, !fastrpc_mmap_create(..., &map));
	mutex_unlock(&fl->fl_map_mutex); // 1
	VERIFY(err, !fastrpc_mmap_on_dsp());
bail:
	if (err && map) {
		mutex_lock(&fl->fl_map_mutex);
		fastrpc_mmap_free(map, 0); // 4
		mutex_unlock(&fl->fl_map_mutex);
	}
	mutex_unlock(&fl->map_mutex);
	return err;
}
```

```
int fastrpc_internal_munmap_fd(...) {
	mutex_lock(&fl->fl_map_mutex);// 2
	if (fastrpc_mmap_find(..., &map))
		[...]
	if (map)
		fastrpc_mmap_free(map, 0);// 3
	mutex_unlock(&fl->fl_map_mutex);
	return err;
}
```

How to exploit

- **ROP/JOP** 😭
  - ✓ PAN (Privileged Access Never)
  - ✓ CFI (Control Flow Integrity)

- **SMA + KSMA** 😀
  1. Convert UAF to Double Free
  2. Bypass KASLR
  3. Apply KSMA

Convert UAF to Double Free

```
drivers/char/adsprpc.c

void fastrpc_mmap_free(struct fastrpc_mmap *map, uint32_t flags) {
        [...]
        if (map->flags == ADSP_MMAP_HEAP_ADDR || ...) {
                [...]
        } else {
                map->refs--;
                if (!map->refs)
                        hlist_del_init(&map->hn); // 1
                if (map->refs > 0 && !flags)
                        return;
        }
        if (map->flags == ADSP_MMAP_HEAP_ADDR || ...) {
                [...]
        } else if (map->flags == FASTRPC_DMAHANDLE_NOMAP) {
                if (!IS_ERR_OR_NULL(map->handle))
                        ion_free(fl->apps->client, map->handle); // 2
        }
        kfree(map); // 3
}
```

Convert UAF to Double Free

- **How to bypass hlist_del_init()**

```
include/linux/list.h

static inline void hlist_del_init(struct hlist_node *n) {
        if (!hlist_unhashed(n)) {
                __hlist_del(n);
                INIT_HLIST_NODE(n);
        }
}

static inline int hlist_unhashed(const struct hlist_node *h
{
        return !h->pprev;
}
```

Convert UAF to Double Free

- **The object used for heap spray**
  1. It uses the kmalloc-256 SLAB
  2. I can control the map->refs and map->flags
  3. The map->hn->pprev is zero

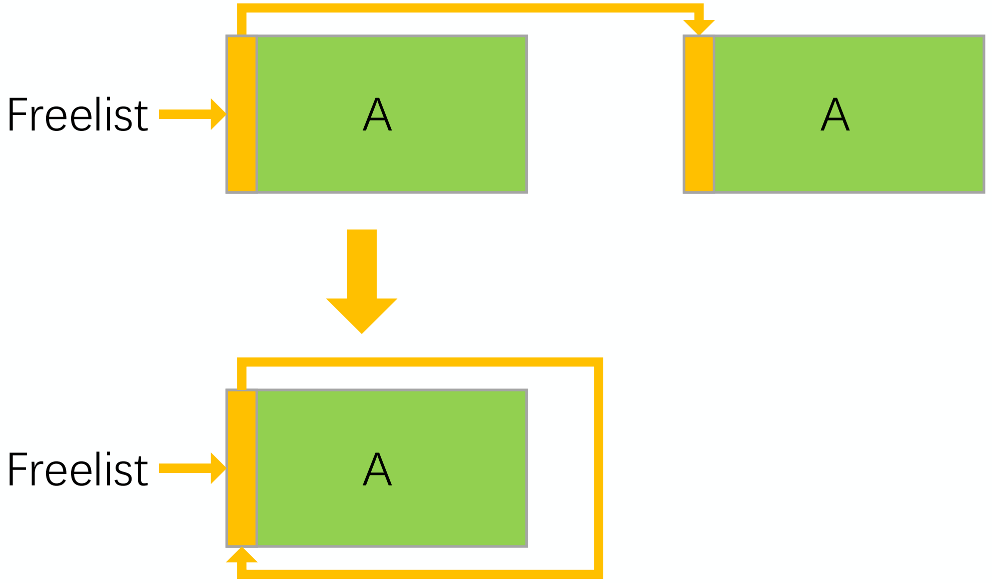
Convert UAF to Double Free

- **The object used for heap spray**

```
fs/xattr.c

setxattr(..., const void __user *value, size_t size, int flags) {
        [...]
        if (size) {
                kvalue = kmalloc(size, GFP_KERNEL | __GFP_NOWARN);
                if (copy_from_user(kvalue, value, size)) {
                        error = -EFAULT;
                        goto out;
                }
        }
}
```

Convert UAF to Double Free

- **The SLAB appears twice in the freelist**



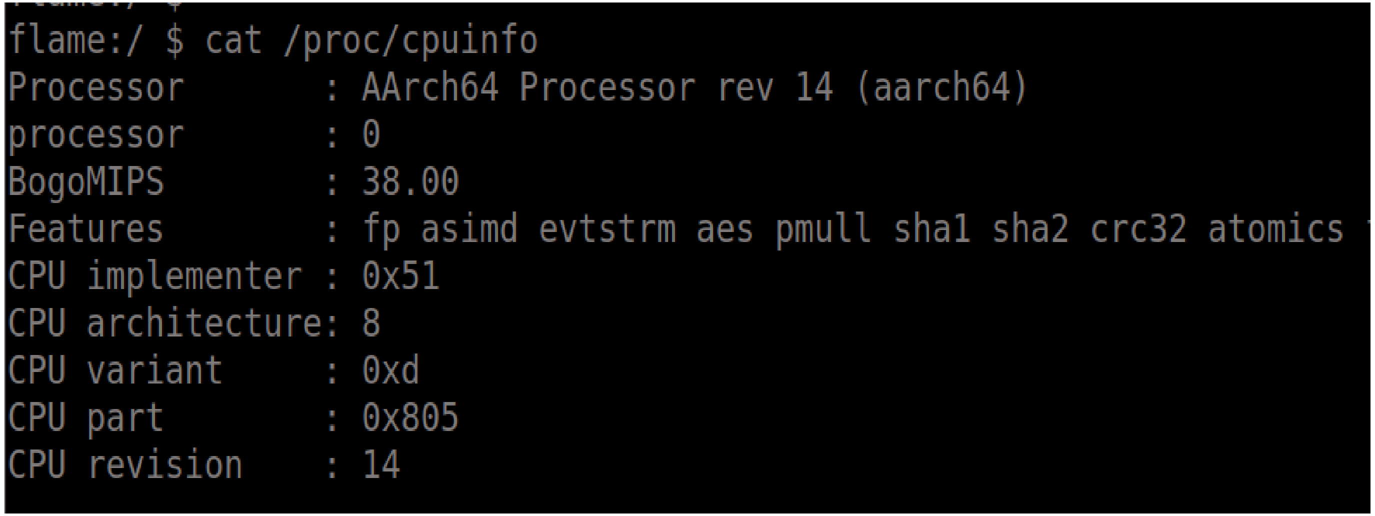
Bypass KASLR

- Use the freelist to modify the object

```
struct seq_file {
      char *buf; // is modified by the freelist
      size_t size;
      [...]
      struct mutex lock;
      const struct seq_operations *op; // is leaked
      int poll_event;
      const struct file *file;
      void *private;
}
```

Bypass KASLR

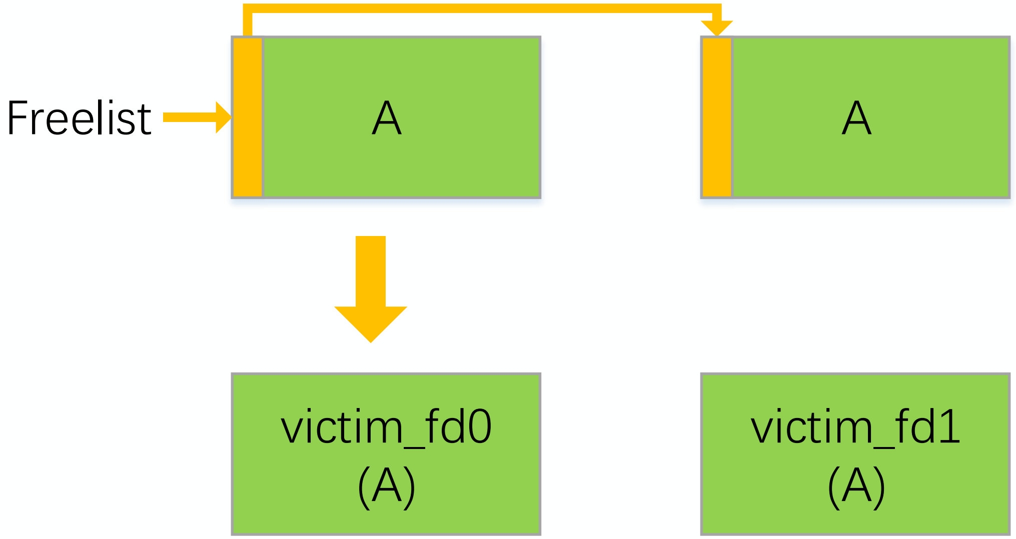
- **Users can read the seq_file->buf**



```
flame:/ $ cat /proc/cpuinfo
Processor       : AArch64 Processor rev 14 (aarch64)
processor       : 0
BogoMIPS        : 38.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics
CPU implementer : 0x51
CPU architecture: 8
CPU variant     : 0xd
CPU part        : 0x805
CPU revision    : 14
```

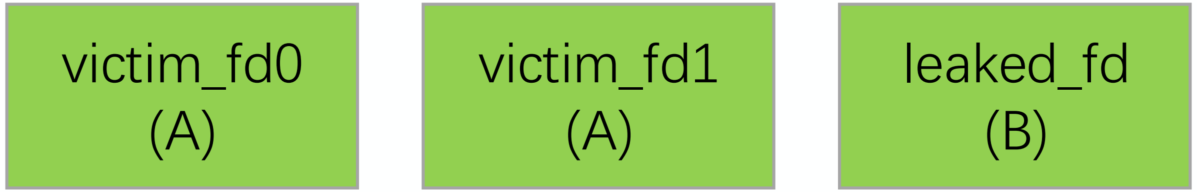
Bypass KASLR

- Allocate victim_fd0 & victim_fd1

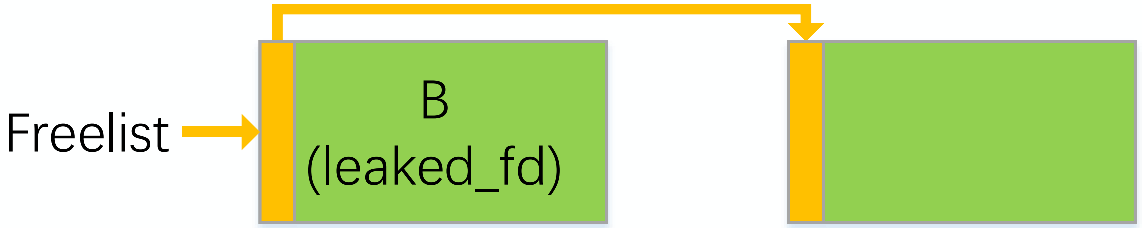


Bypass KASLR

- **Allocate the leaked_fd**

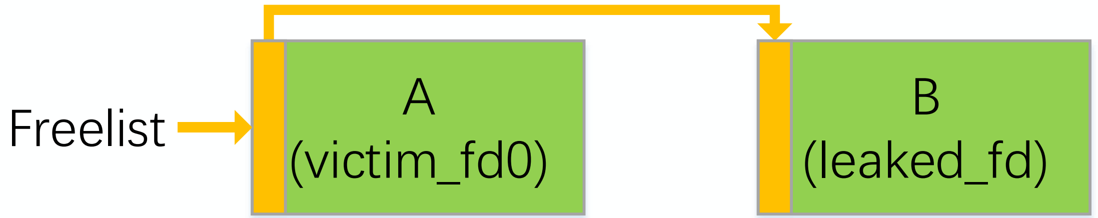


- **Free the leaked_fd**



Bypass KASLR

- Free the victim_fd0



- Read seq_file->op from victim_fd1

Apply KSMA

- **Use the object to modify the freelist**
  1. It uses the kmalloc-256 SLAB
  2. I can control the first 8 bytes

Apply KSMA

- Use the object to modify the freelist

```
struct ipv6_mc_socklist {
        struct in6_addr addr;
        int ifindex;
        struct ipv6_mc_socklist __rcu *next;
        rwlock_t sflock;
        unsigned int sfmode;
        struct ip6_sf_socklist *sflist;
        struct rcu_head rcu;
}
```

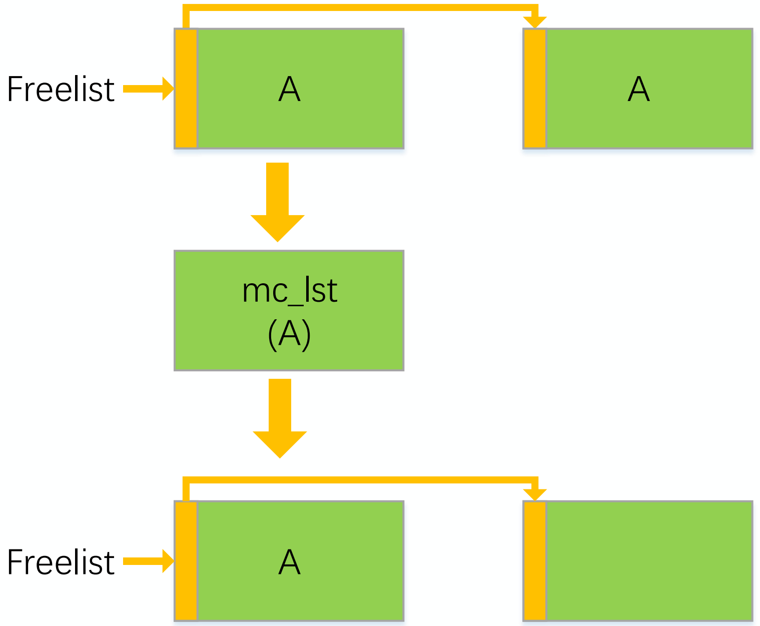
Apply KSMA

- **Use the object to modify the freelist**

```
net/ipv6/mcast.c

int ipv6_sock_mc_join(..., const struct in6_addr *addr) {
    [...]
    mc_lst = sock_kmalloc(sizeof(struct ipv6_mc_socklist));
    if (!mc_lst)
        return -ENOMEM;
    mc_lst->next = NULL;
    mc_lst->addr = *addr;
    [...]
}
```

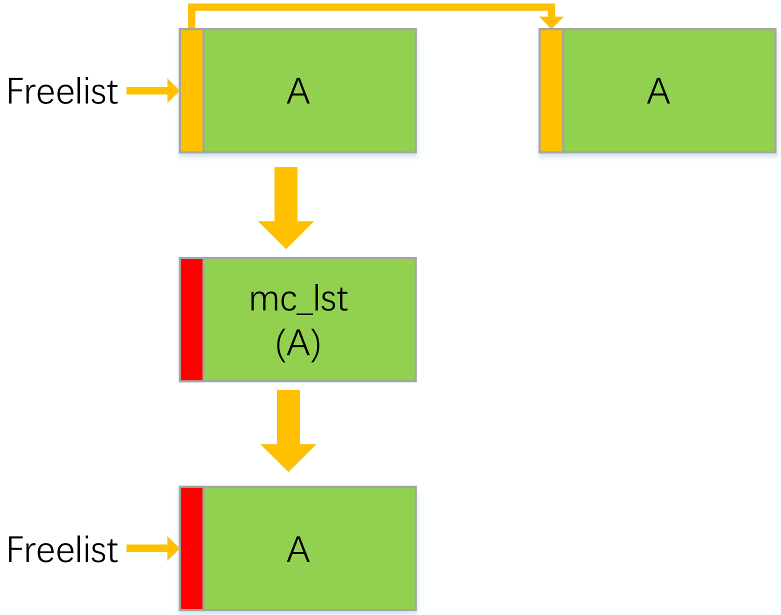
Apply KSMA

- Allocate the mc_lst

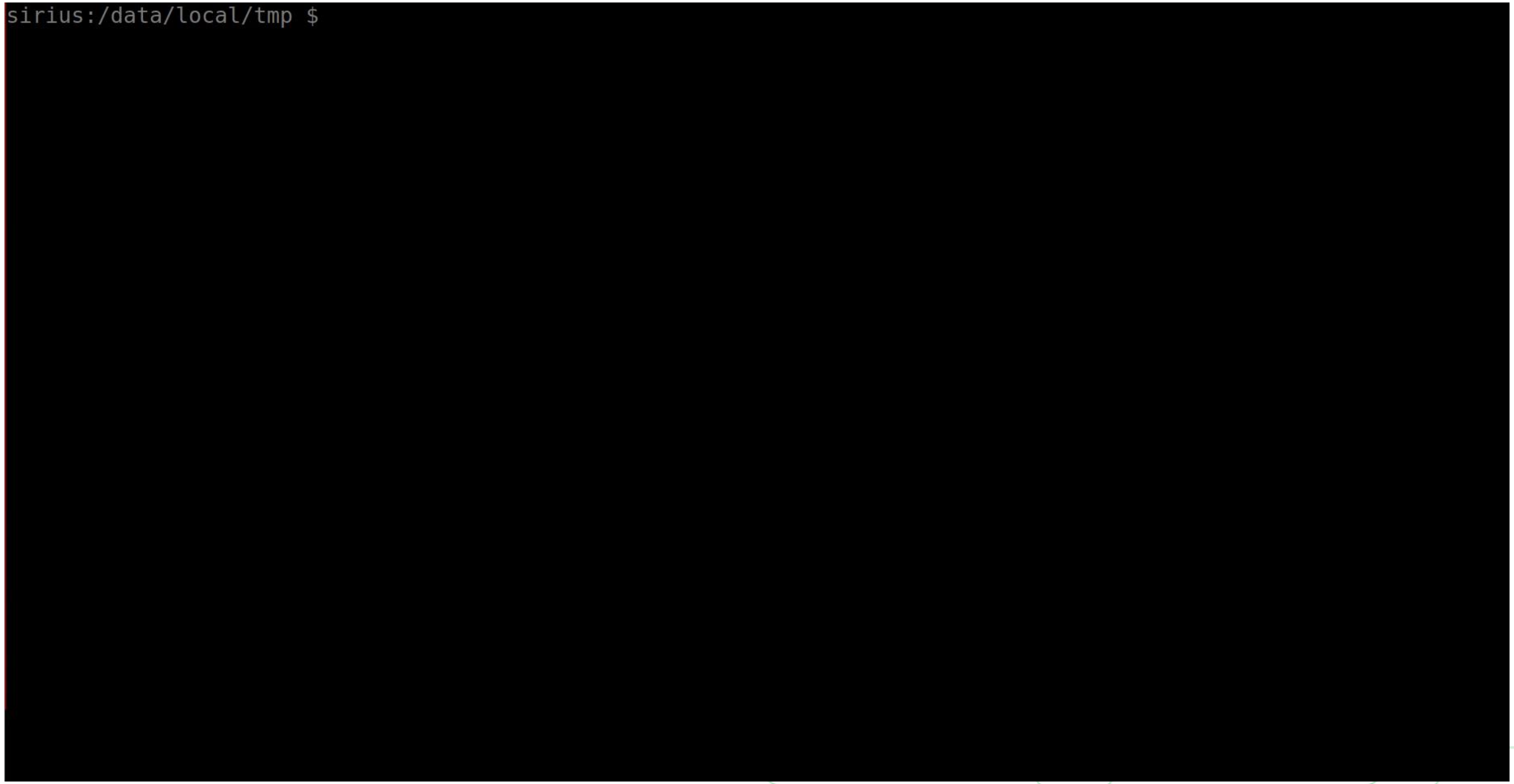


Apply KSMA

- **Update the mc_lst->addr**



Demo



```
sirius:/data/local/tmp $
```

Contents

The ret2dir mitigation

- **Exclusive Page-Frame Ownership**
  - ✓ Unmap the user page in the kernel
  - ✓ Not yet merged into the mainline

https://lwn.net/Articles/784839/

The KSMA mitigation

- **The KSMA patch**
  - ✓ I submitted it in May 2018
  - ✓ Was merged into the mainline in September 2018
  - ✓ Android has not yet been backported 😰

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/log/?h=v5.8.8&qt=author&q=yaojun

The KSMA patch

- Move the page table to the rodata section

```
arch/arm64/kernel/vmlinux.lds.S

+#define KERNEL_PG_TABLES \
+ . = ALIGN(PAGE_SIZE); \
+ idmap_pg_dir = .; \
+ . += IDMAP_DIR_SIZE; \
+ TRAMP_PG_TABLE \
+ RESERVED_PG_TABLE \
+ swapper_pg_dir = .; \
+ . += PAGE_SIZE; \
+ swapper_pg_end = .;

RO_DATA(PAGE_SIZE)
EXCEPTION_TABLE(8)
NOTES
+ KERNEL_PG_TABLES
```

The KSMA patch

- The kernel assumes the relative offset between the swapper_pg_dir and the tramp_pg_dir

```
arch/arm64/kernel/entry.S

.macro tramp_map_kernel, tmp
mrs \tmp, ttbr1_el1
sub \tmp, \tmp, #(SWAPPER_DIR_SIZE + RESERVED_TTBR0_SIZE)
bic \tmp, \tmp, #USER_ASID_FLAG
msr ttbr1_el1, \tmp
[...]
.endm
```

The KSMA patch

- Update the page table through fixmap

```
void set_swapper_pgd(pgd_t *pgdp, pgd_t pgd)
{
        pgd_t *fixmap_pgdp;
        spin_lock(&swapper_pgdir_lock);
        fixmap_pgdp = pgd_set_fixmap(pa(pgdp));
        WRITE_ONCE(*fixmap_pgdp, pgd);
        pgd_clear_fixmap();
        spin_unlock(&swapper_pgdir_lock);
}
```

The SMA mitigation

- **CONFIG_SLAB_FREELIST_HARDENED[1]**
  - ✓ Not enabled in the kernel 4.9
  - ✓ Not enabled on the Google Pixel (4.14)

- **CONFIG_INIT_ON_FREE_DEFAULT_ON[2]**
  - ✓ Not enabled in the Android kernel

[1] https://hardenedlinux.github.io/system-security/2017/12/02/linux_kernel_4.14%E7%9A%84SLAB_FREELIST_HARDENED%E7%9A%84%E7%AE%80%E8%A6%81%E5%88%86%E6%9E%90.html

[2] https://outflux.net/blog/archives/2019/11/14/security-things-in-linux-v5-3/

SLAB_FREELIST_HARDENED

- **Generate a random number**

```
mm/slub.c

int kmem_cache_open(struct kmem_cache *s, unsigned long flags)
{
        s->flags = kmem_cache_flags(s->size, flags, s->name, s->ctor);
        s->reserved = 0;
#ifdef CONFIG_SLAB_FREELIST_HARDENED
        s->random = get_random_long();
#endif
}
```

SLAB_FREELIST_HARDENED

- Obfuscate the address of SLAB

```
mm/slub.c

void *freelist_ptr(const struct kmem_cache *s, void *ptr, unsigned long ptr_addr)
{
#ifdef CONFIG_SLAB_FREELIST_HARDENED
	return (void *)((unsigned long)ptr ^ s->random ^
		(unsigned long)kasan_reset_tag((void *)ptr_addr));
#else
	return ptr;
#endif
}
```

SLAB_FREELIST_HARDENED

- **The obfuscation has weaknesses**

```
kmalloc-32 freelist walk, before:

ptr                 ptr_addr             stored value       random number
ffff90c22e019020@ffff90c22e019000 is 86528eb656b3b5bd (86528eb656b3b59d)
ffff90c22e019040@ffff90c22e019020 is 86528eb656b3b5fd (86528eb656b3b59d)
ffff90c22e019060@ffff90c22e019040 is 86528eb656b3b5bd (86528eb656b3b59d)
```

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=1ad53d9fa3f6168ebcf48a50e08b170432da2257

SLAB_FREELIST_HARDENED

- **Improve the obfuscation**

```
mm/slub.c

void *freelist_ptr(const struct kmem_cache *s, void *ptr, unsigned long ptr_addr)
{
#ifdef CONFIG_SLAB_FREELIST_HARDENED
	return (void *)((unsigned long)ptr ^ s->random ^
-			(unsigned long)kasan_reset_tag((void *)ptr_addr));
+			swab((unsigned long)kasan_reset_tag((void *)ptr_addr)));
#else
	return ptr;
#endif
}
```

SLAB_FREELIST_HARDENED

- **Check the double free**

```
mm/slub.c

void set_freepointer(struct kmem_cache *s, void *object, void *fp)
{
	unsigned long freeptr_addr = (unsigned long)object + s->offset;
#ifdef CONFIG_SLAB_FREELIST_HARDENED
	BUG_ON(object == fp); // Double free
#endif
	*(void **)freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
}
```

INIT_ON_FREE_DEFAULT_ON

- **Clear the data in the released SLAB**

```
mm/slub.c

bool slab_free_freelist_hook(struct kmem_cache *s, void **head, void **tail)
{
        do {
                object = next;
                next = get_freepointer(s, object);
                if (slab_want_init_on_free(s)) {
                        memset(object, 0, s->object_size);
                        rsize = (s->flags & SLAB_RED_ZONE) ? s->red_left_pad : 0;
                        memset((char *)object + s->inuse, 0, s->size - s->inuse - rsize);
                }
        } while (object != old_tail);
}
```

Contents

Conclusion

- **The combination of the SMA and the KSMA is powerful**

- **In order to harden the kernel, we need:**
  1. Mitigate known attacks (the ret2dir)
  2. Fill the gap between Android and the mainline (the KSMA patch)
  3. Prevent kernel fragmentation (the SMA)

Thank You

Jun Yao @ Alpha Lab

yaojun8558363@gmail.com