PANGU TEAM

Breaking through the cage

Get Android universal root by B-PUAF

2024/11/08

About us

- Lu Yutao
- Twitter： @_snowfish
- Security researcher in Pangu Team
- Focus on Android vulnerability and exploitation technique

- Ling Hanqin
- Twitter： @0Cru5h48452
- Security researcher in Pangu Team
- Focus on Linux vulnerability and exploitation technique

Agenda

01 **Attack Surface**

Background

Security issues

Our work

02 **Vulnerability Evolution**

Bad 2022

Red 2023

Debt 2024

03 **Offensive Complexity**

Decrement primitives

Manipulate pagetable

Hypervisor challenge

04 **Demo and Advance**

Exploit videos

Patch suspicion

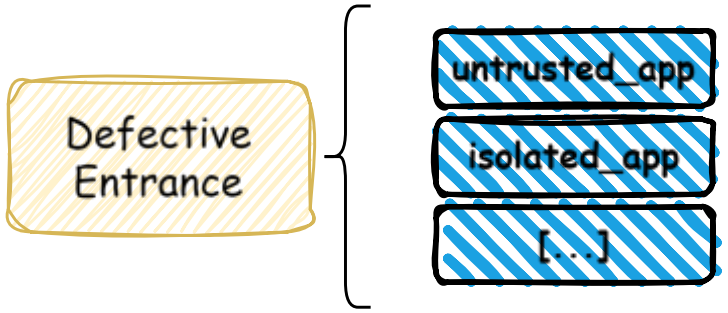
Mitigation strategies

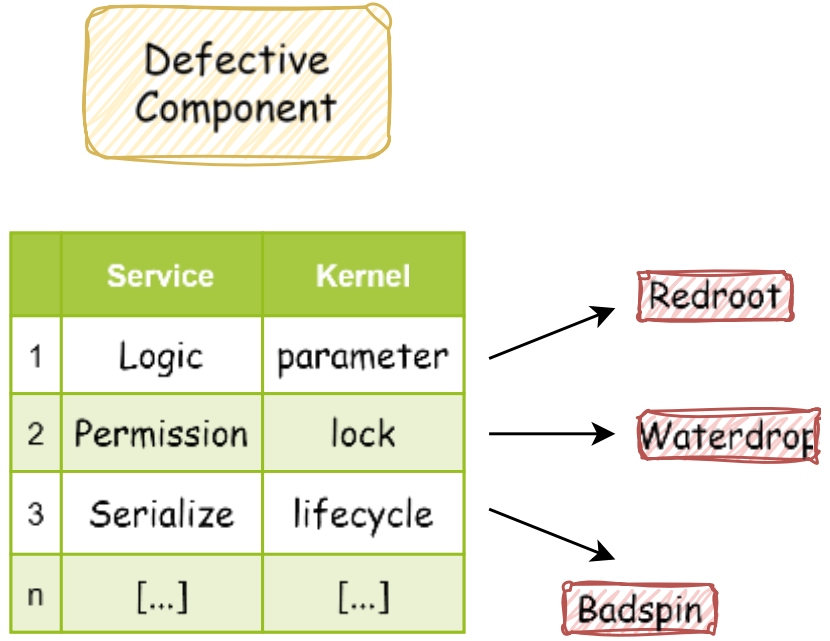
PANGU TEAM

Attack Surface

Android binder

Binder Threat Model

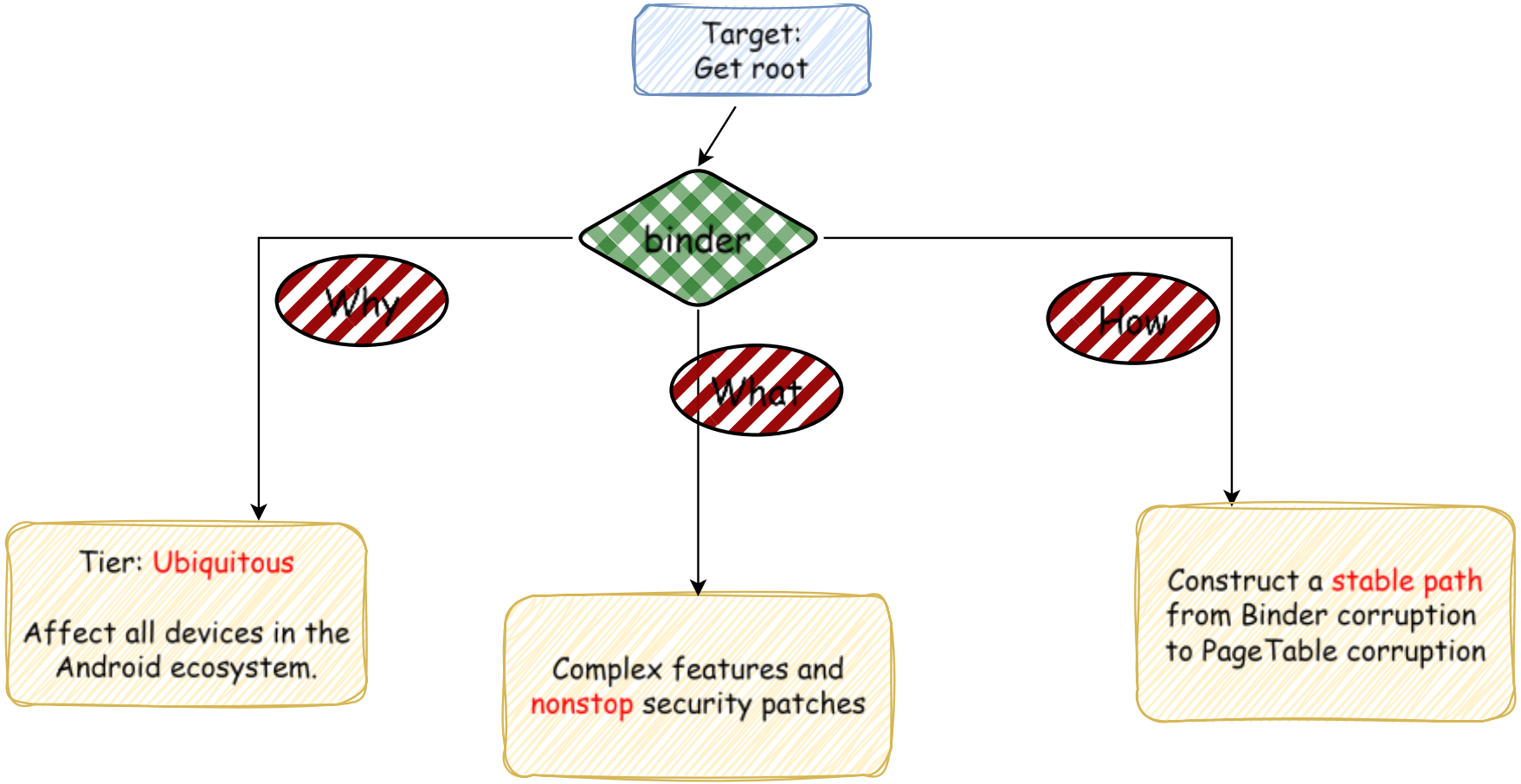




Constant exploitation

- Binder: The Bridge to Root(CVE-2019-2025) from Mosec2019
- Binder IPC and its vulnerabilities(CVE-2020-0041) from Thcon2020
- Typhoon Mangkhut(CVE-2020-0423) from Blackhat USA2021
- Exploiting Spinlock UAF in the Android Kernel(CVE-2022-20421) from Offensivecon2023
- Attacking Android Binder(CVE-2023-20938 & CVE-2023-21255) from Offensivecon2024

Our work


Target:
Get root
binder
Why
What
How
Tier: Ubiquitous
Affect all devices in the Android ecosystem.
Complex features and nonstop security patches
Construct a stable path from Binder corruption to PageTable corruption

PANGU TEAM

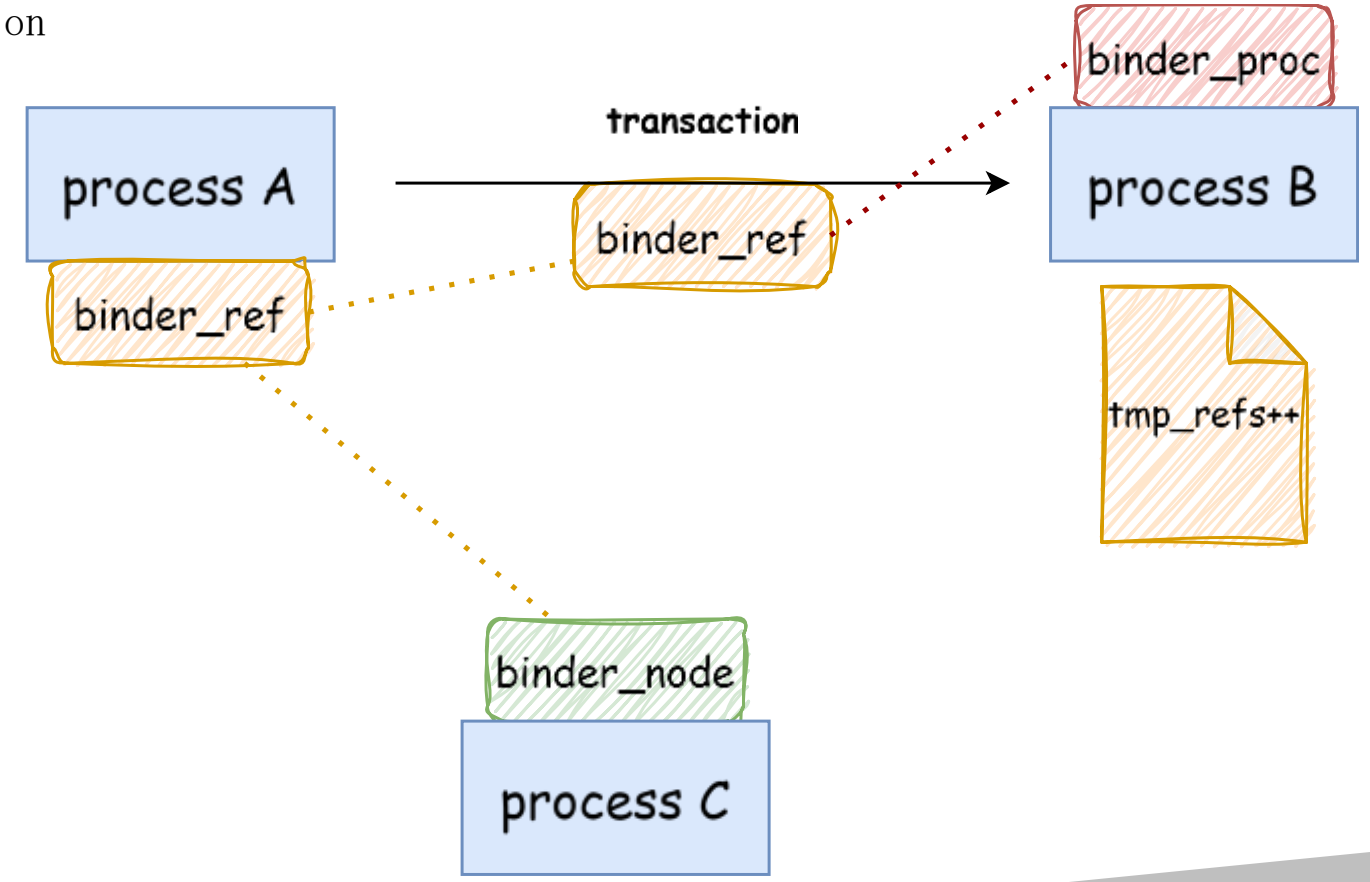
Vulnerability Evolution

Android binder

Bad spin in 2022

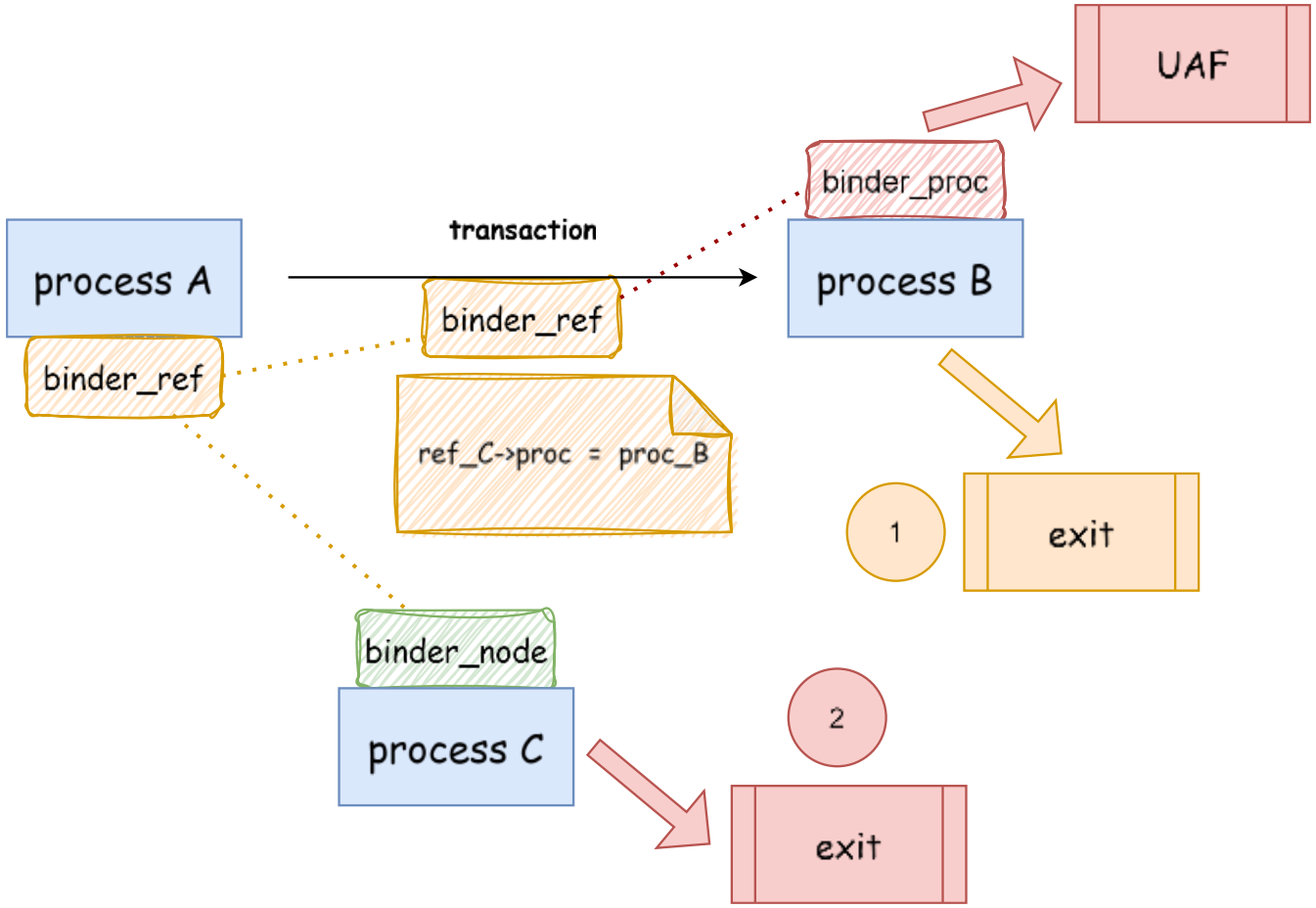
Normal

During normal binder transaction



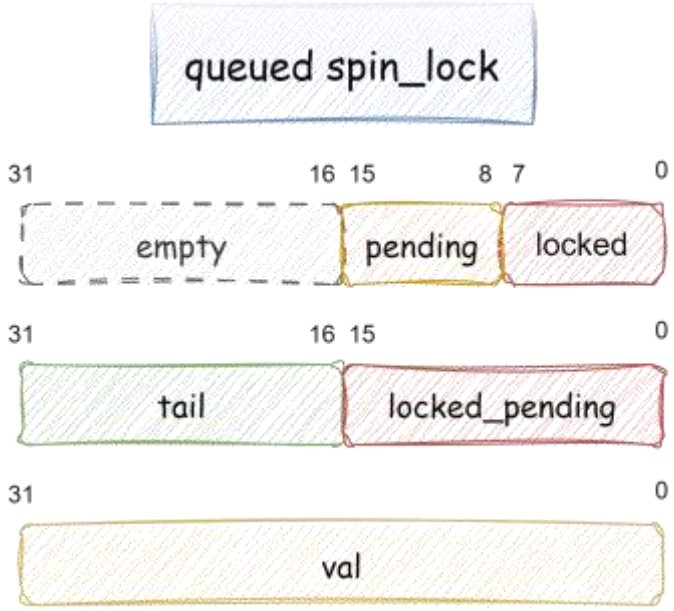
Vulnerable

Trigger CVE-2022-20421

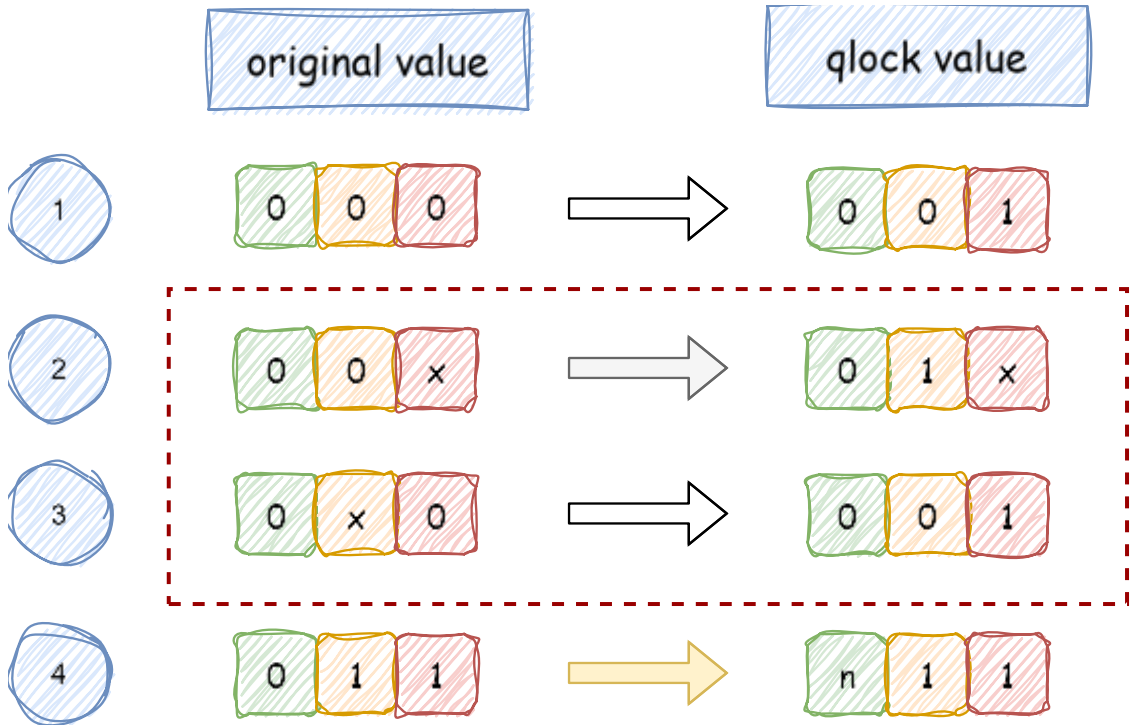


Queued spin lock

TAS -> ticket -> queued

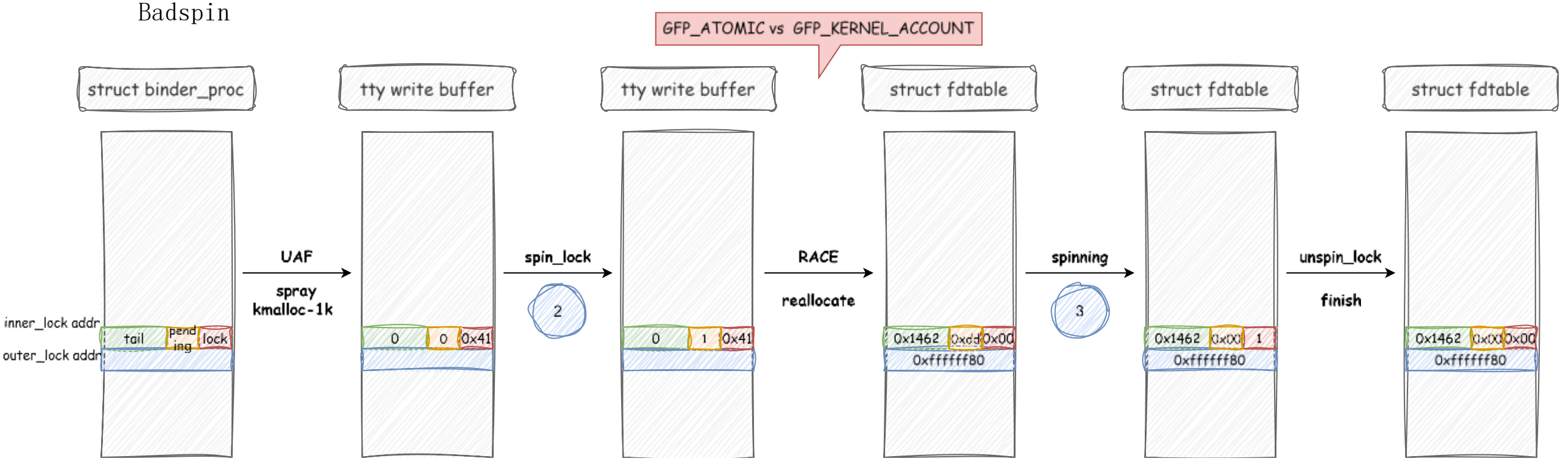


State Migration



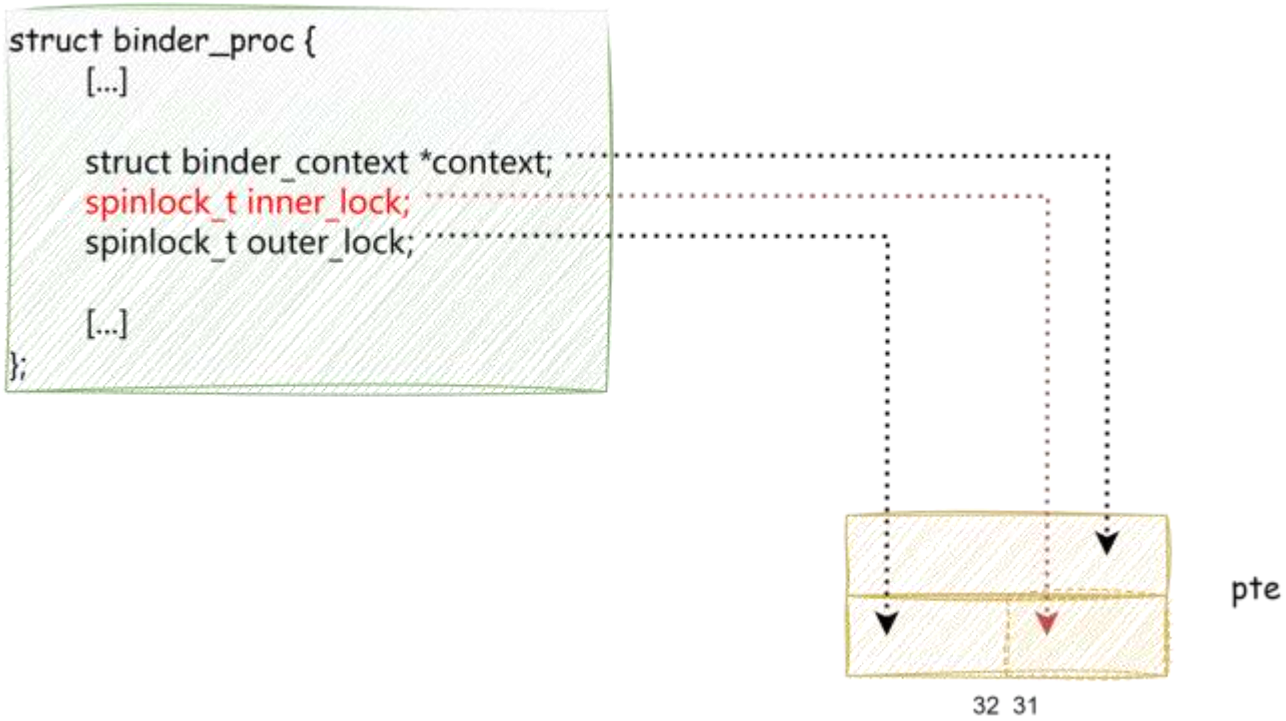
The original idea

Badspin



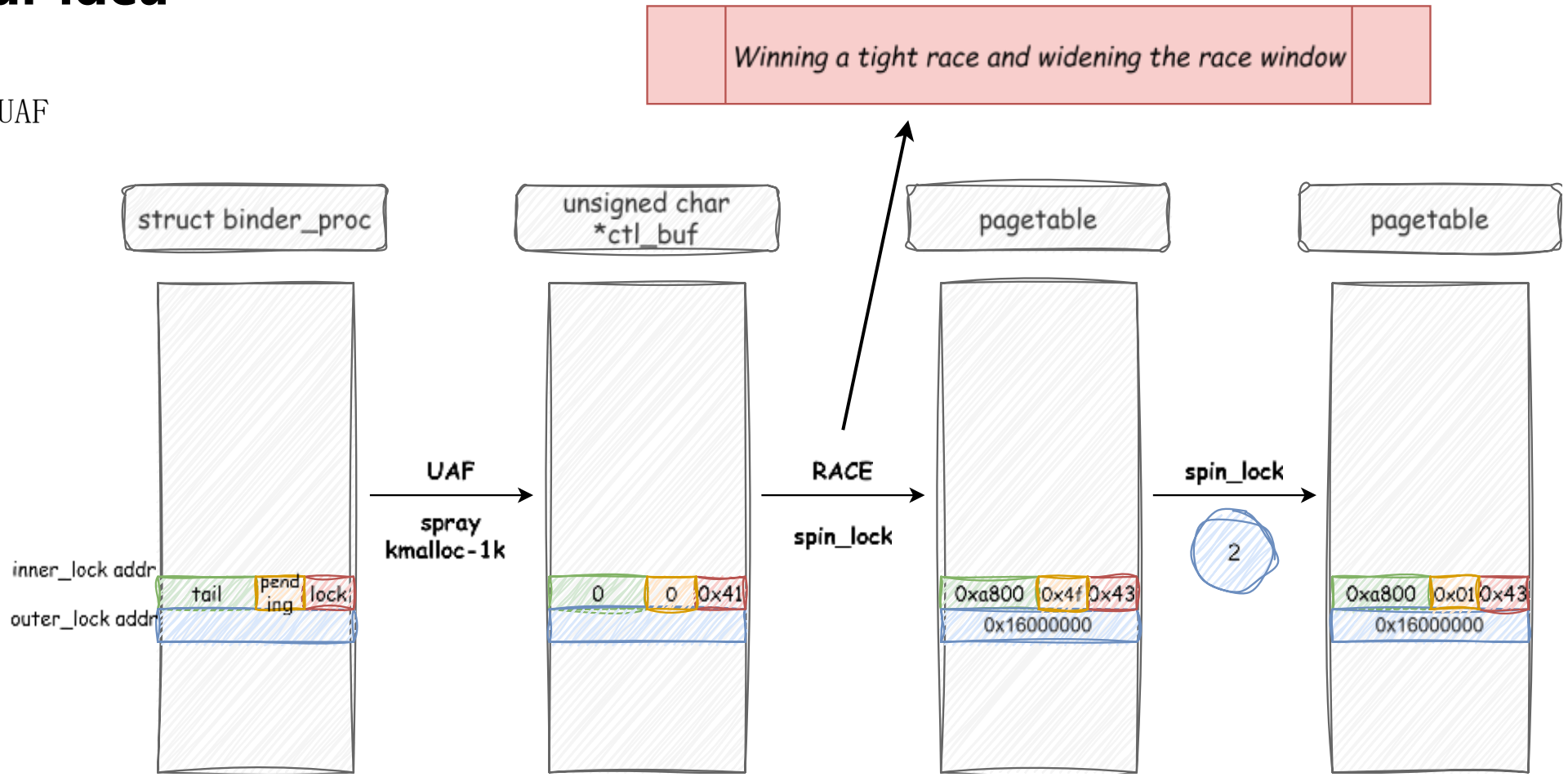
Source of insight

Key data offset mapping



Our idea

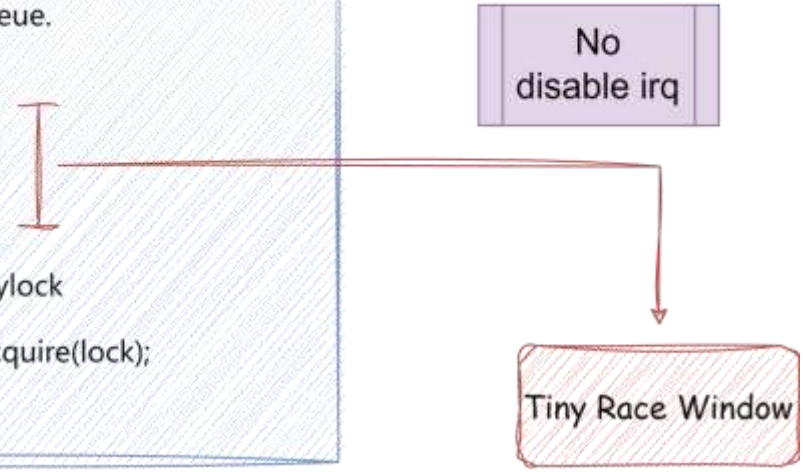
B-PUAF



Race condition

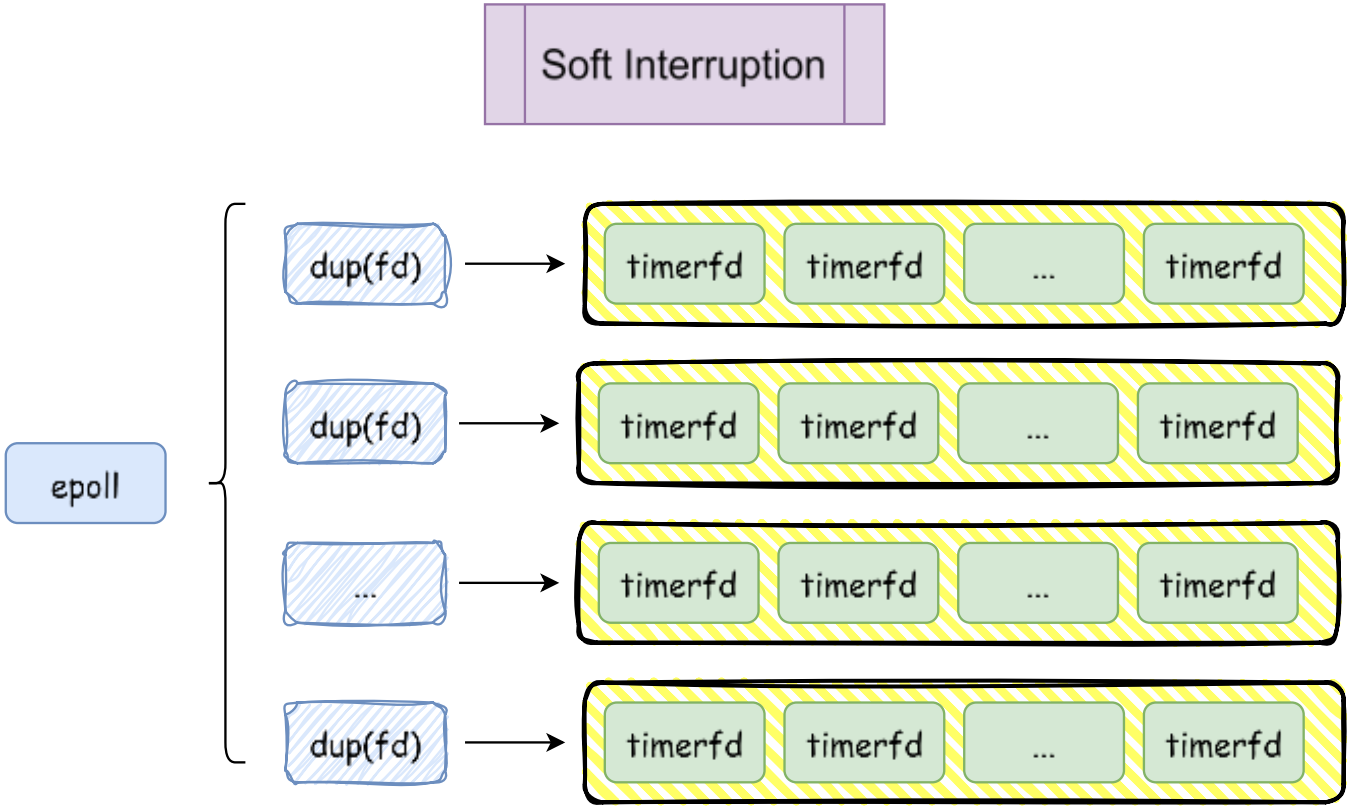
B-PUAF

```
void queued_spin_lock_slowpath(struct qspinlock *lock, u32 val)
{
        [...]
        /*
         * If we observe any contention; queue.
         */
        if (val & ~_Q_LOCKED_MASK)
                goto queue;

        /*
         * trylock || pending
         *
         * 0,0,* -> 0,1,* -> 0,0,1 pending, trylock
         */
        val = queued_fetch_set_pending_acquire(lock);
        [...]
}
```



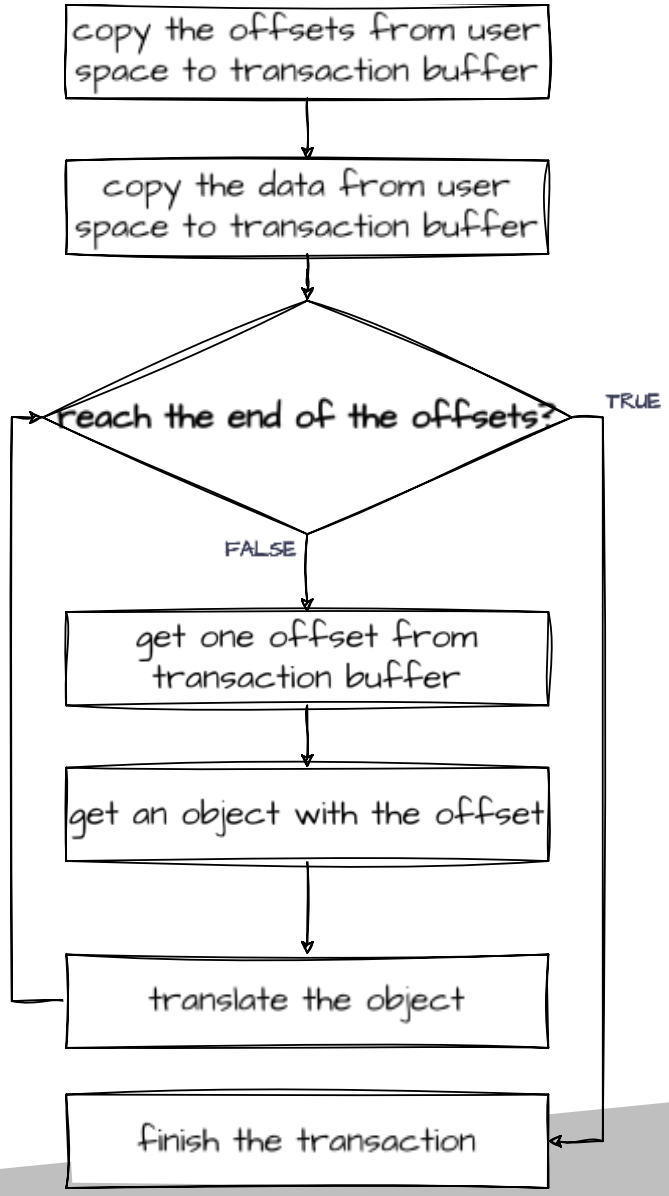
Race against time

From Jann Horn

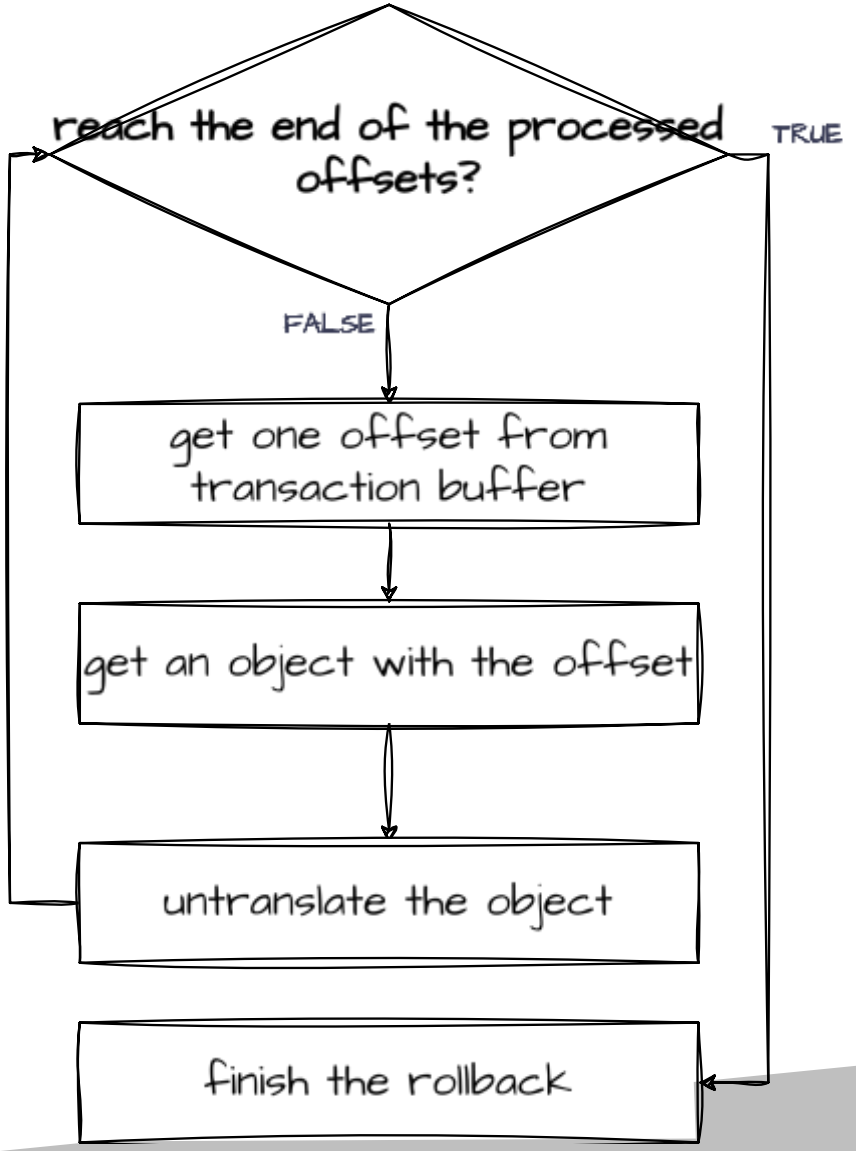


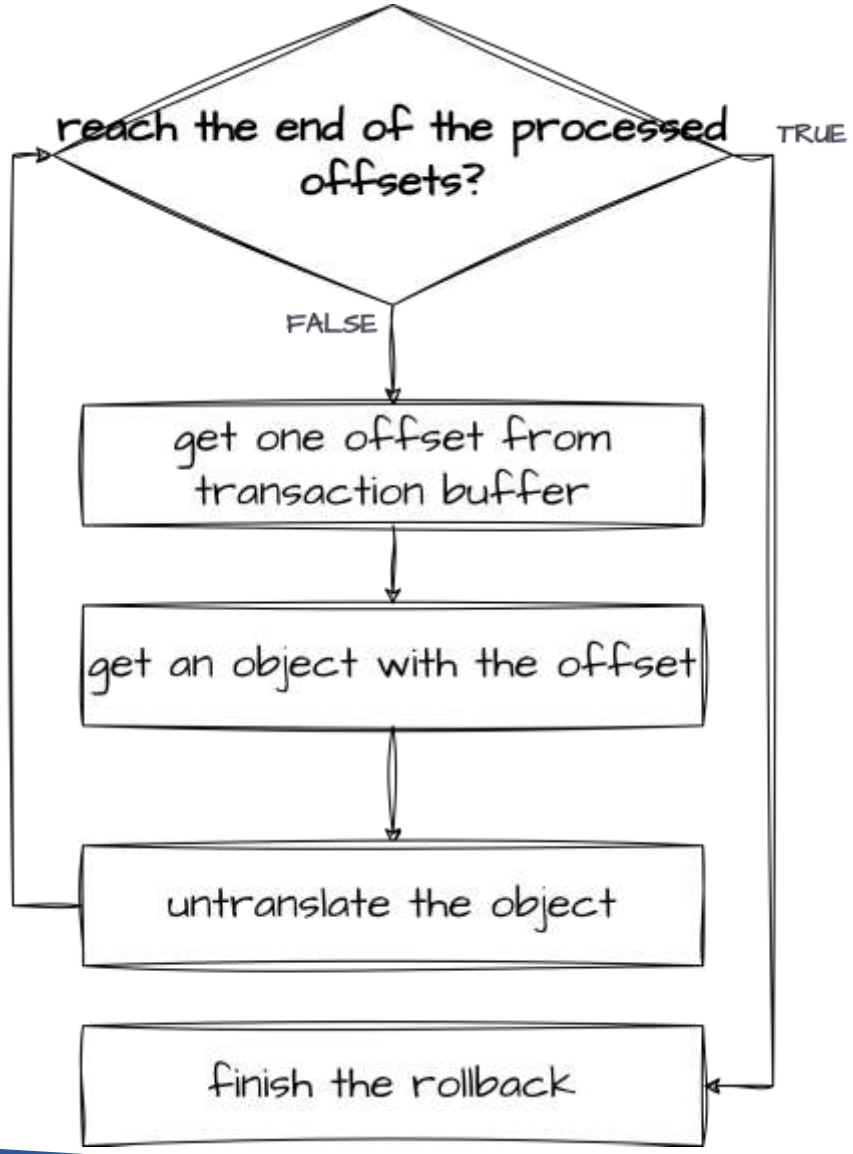
Red binder in 2023

Transaction



Transaction rollback

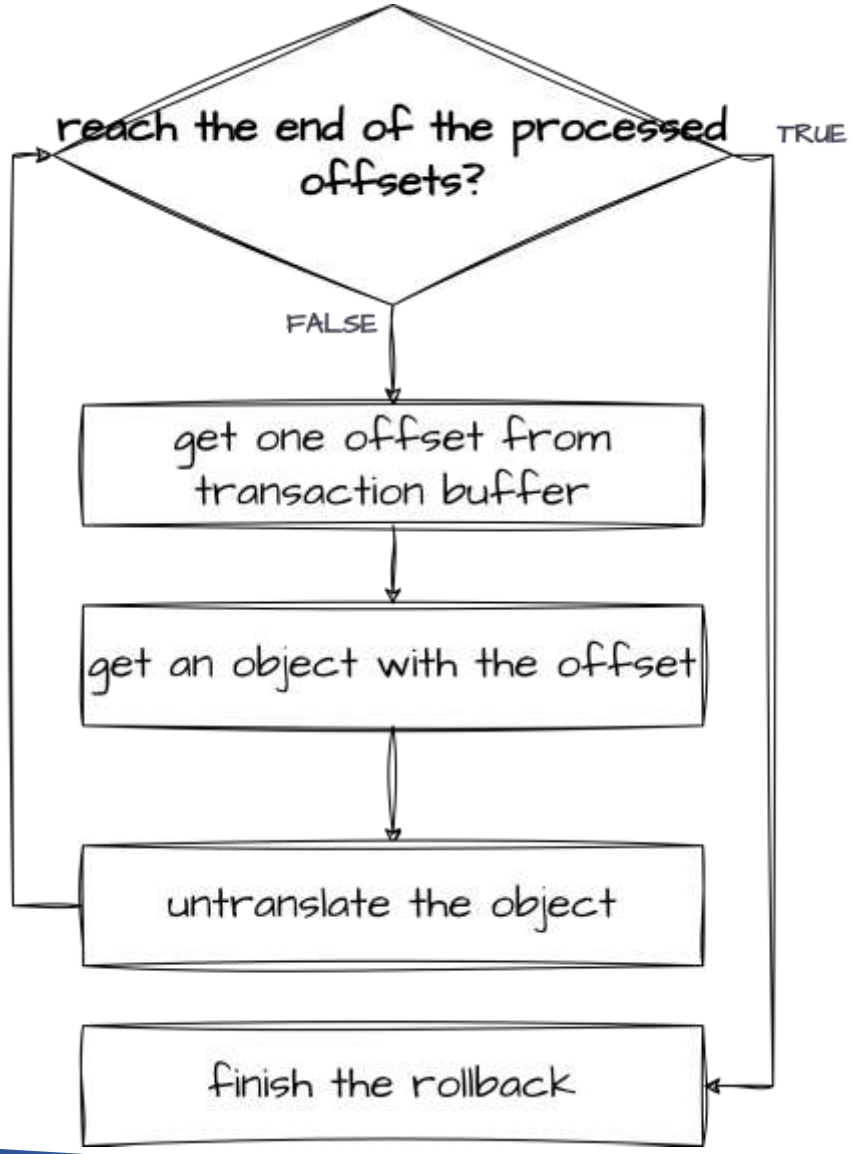




What if error occurs on the first offset during translation?

- off_end_offset should be 0
- binder should be untranslated nothing

```
// binder_transaction_buffer_release
    ...
    off_end_offset = is_failure && failed_at ? failed_at :
                off_start_offset + buffer->offsets_size; // [1] error calc
    for (buffer_offset = off_start_offset; buffer_offset < off_end_offset;
        buffer_offset += sizeof(binder_size_t)) {
        ... // [2] get an object and untranslate it
    }
    ...
```



What if error occurs on the first offset during translation?

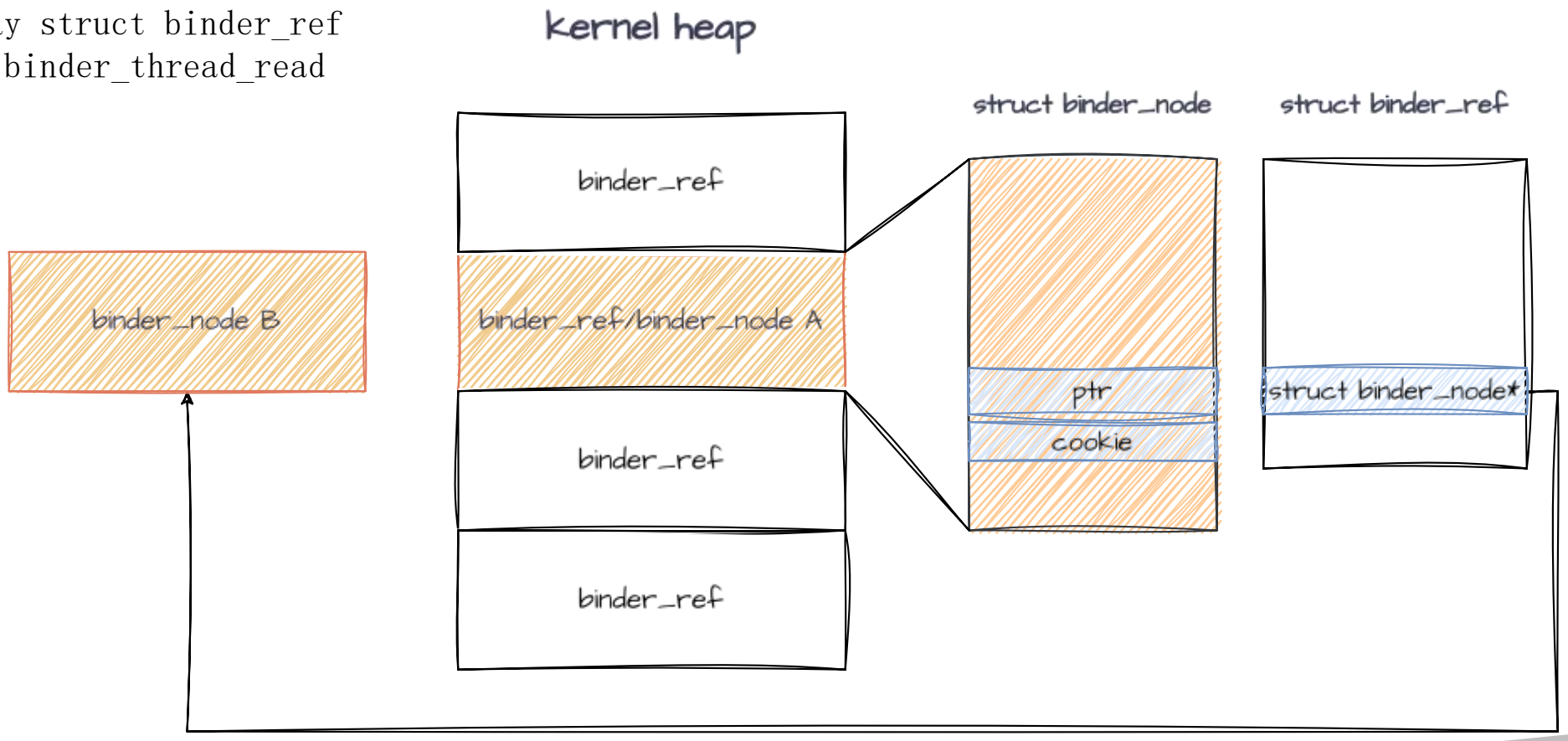
- `failed_at = 0`
- `off_end_offset = endof(offsets)`

Untranslated everything

```
// binder_transaction_buffer_release
    ...
    off_end_offset = is_failure && failed_at ? failed_at :
                off_start_offset + buffer->offsets_size; // [1] error calc
    for (buffer_offset = off_start_offset; buffer_offset < off_end_offset;
        buffer_offset += sizeof(binder_size_t)) {
        ... // [2] get an object and untranslate it
    }
    ...
```

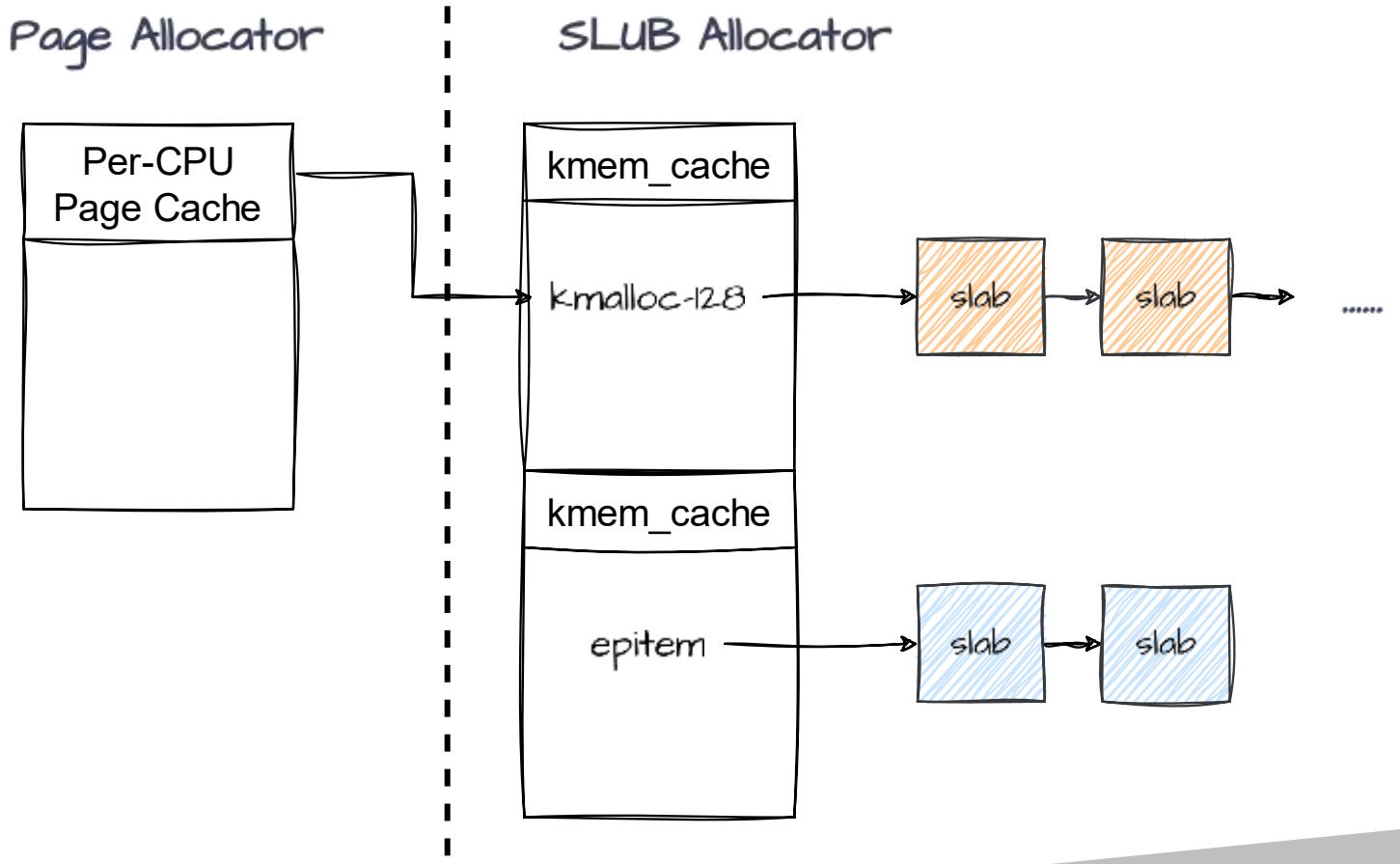
Leak the address of binder_node B

- Trigger to free binder_node A
- Spray struct binder_ref
- UAF binder_thread_read



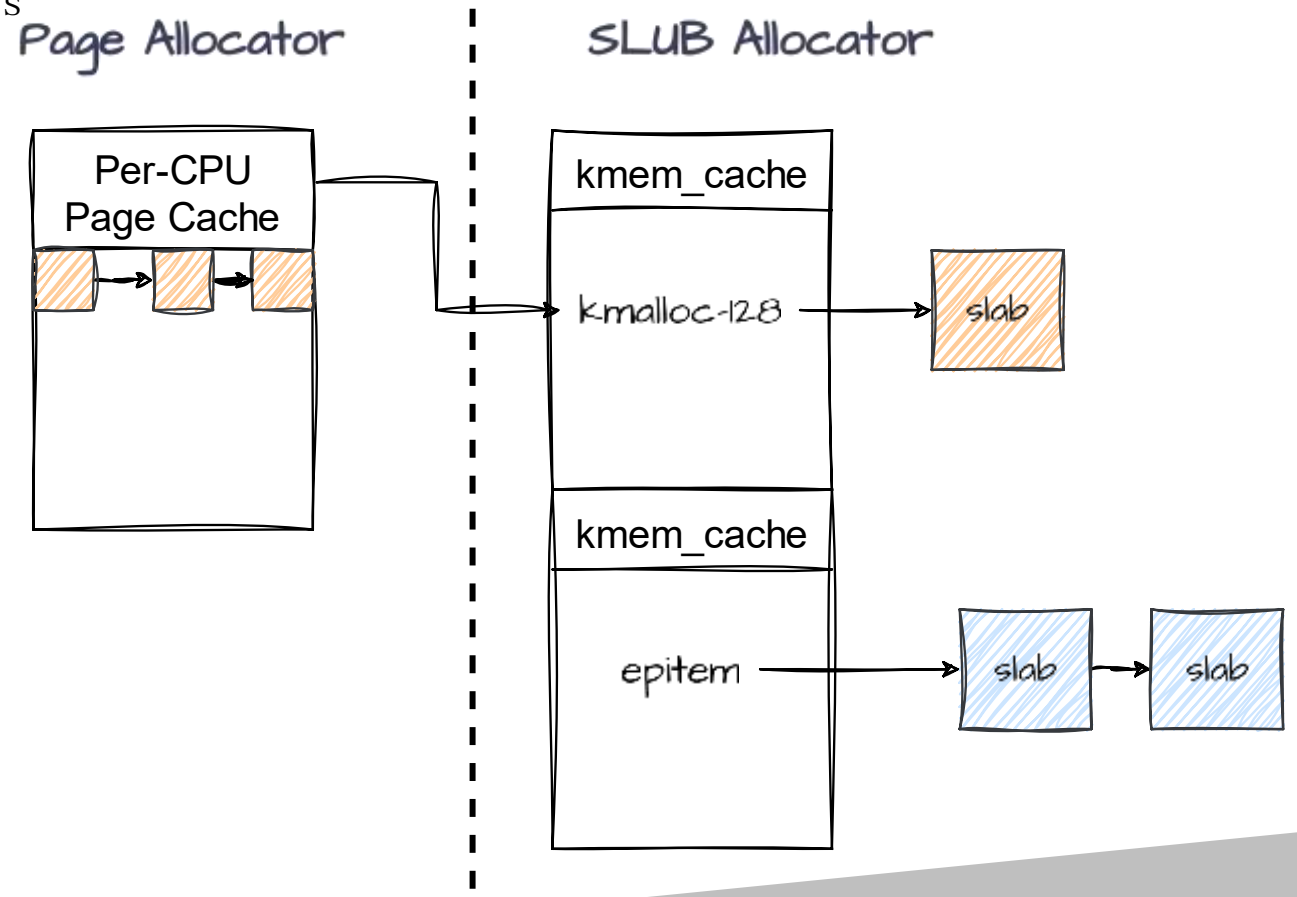
Cross cache to spray epitem over the binder_node

- Spray kmalloc-128 object



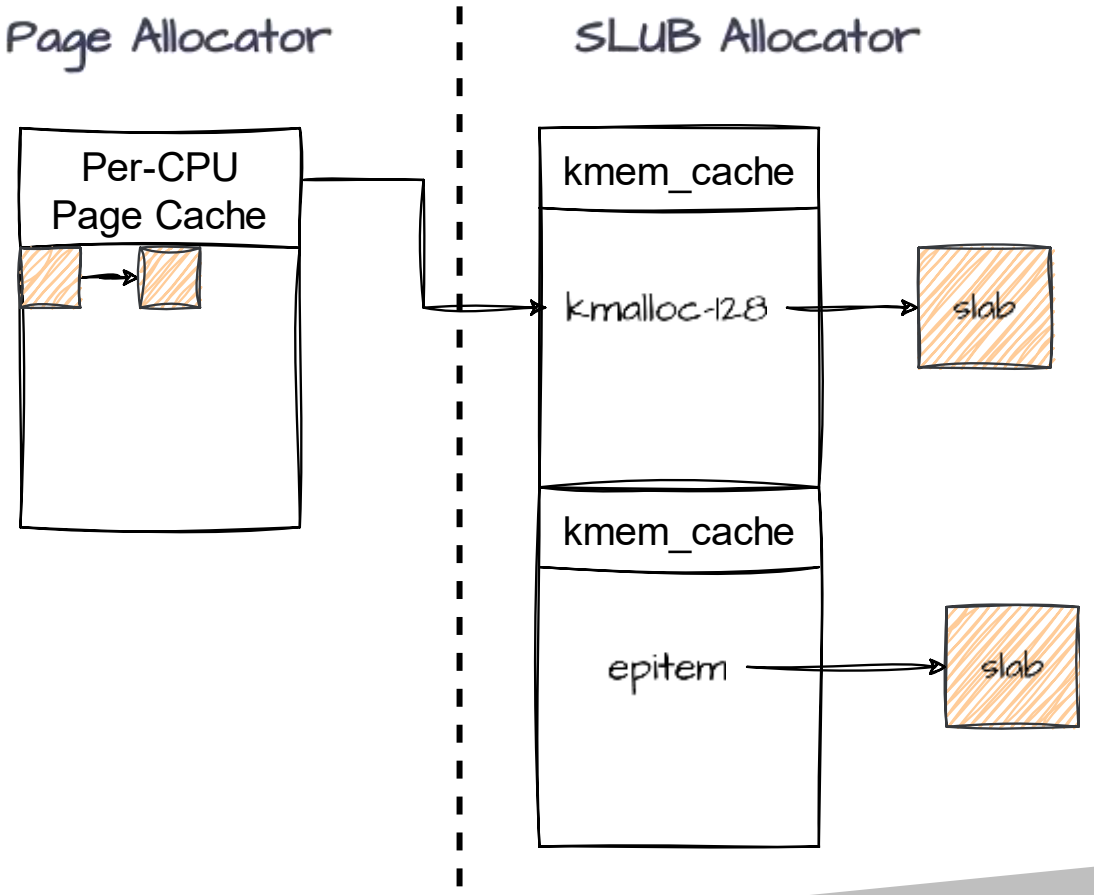
Cross cache to spray epitem over the binder_node

- Spray kmalloc-128 object
- Release object to Per-CPU Pages



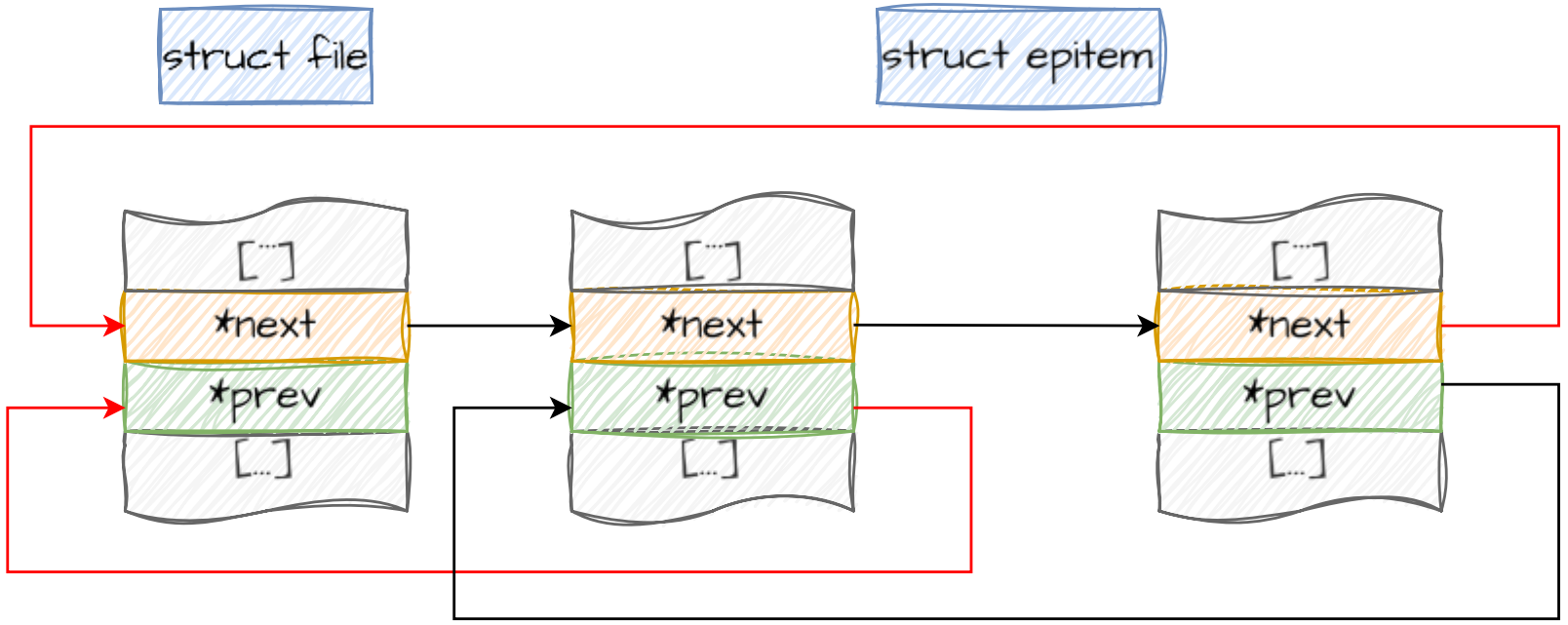
Cross cache to spray epitem over the binder_node

- Spray kmalloc-128 object
- Release object to Per-CPU Pages
- Spray struct epitem to get the UAF page



Leak the address of the file

- Overlap epitem with the UAF node
- UAF Binder_thread_read



Unlink primitive

- Spray fake nodes by calling sendmsg with user-controllable data
- Call BC_FREE_BUFFER
- Use the leaked node address

Write NULL or a kernel pointer to an address

```
static bool binder_dec_node_nilocked(struct binder_node *node, ...) {
...
// use leaked address to satisfy all checks
...
hlist_del(&node->dead_node);
...
}
```

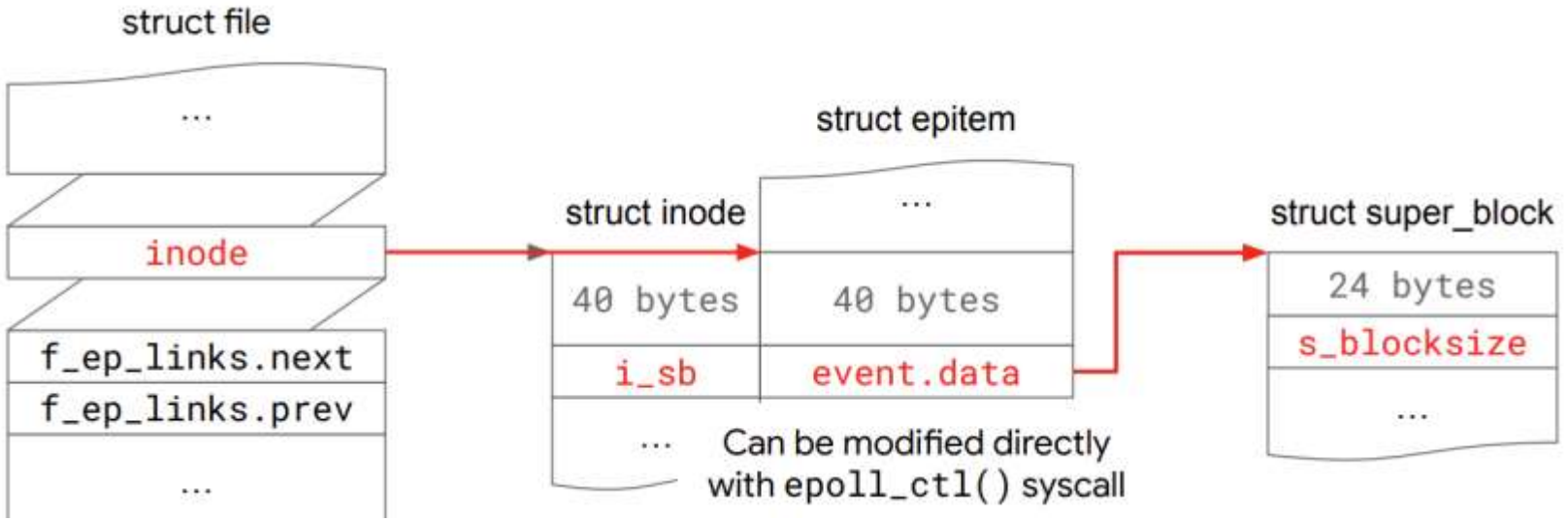
*pprev = next
*(next + 8) = pprev

Arbitrary address 4 bytes read primitive

- Write the leaked node_address to the inode using unlink primitive

```
ioctl(fd, FIGETBSZ, &value); // &value == argp
```

```
return put_user(inode->i_sb->s_blocksize, (int __user *)argp);
```



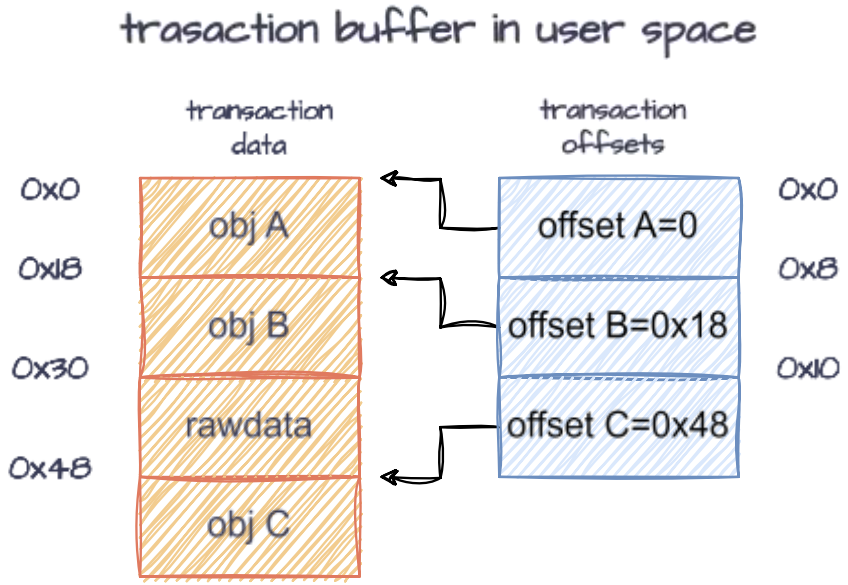
Get root

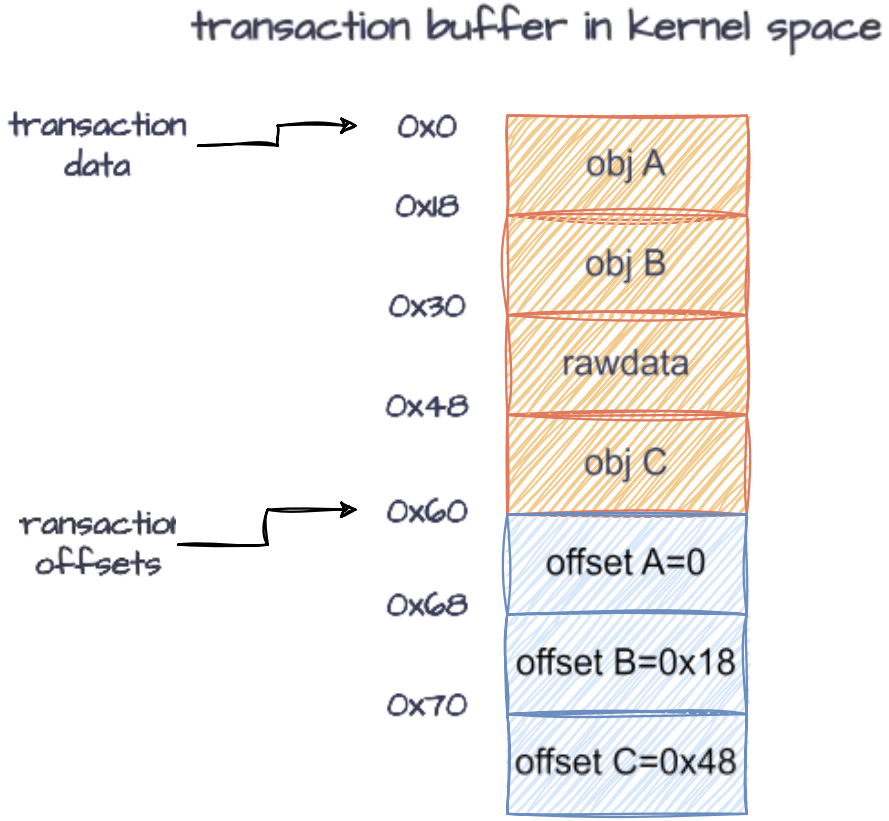
- Using the 4 bytes read primitive to find cred and selinux_state
- Using the unlink primitive to write 0 to that addresses

Debt binder in 2024

CVE-2024-46740

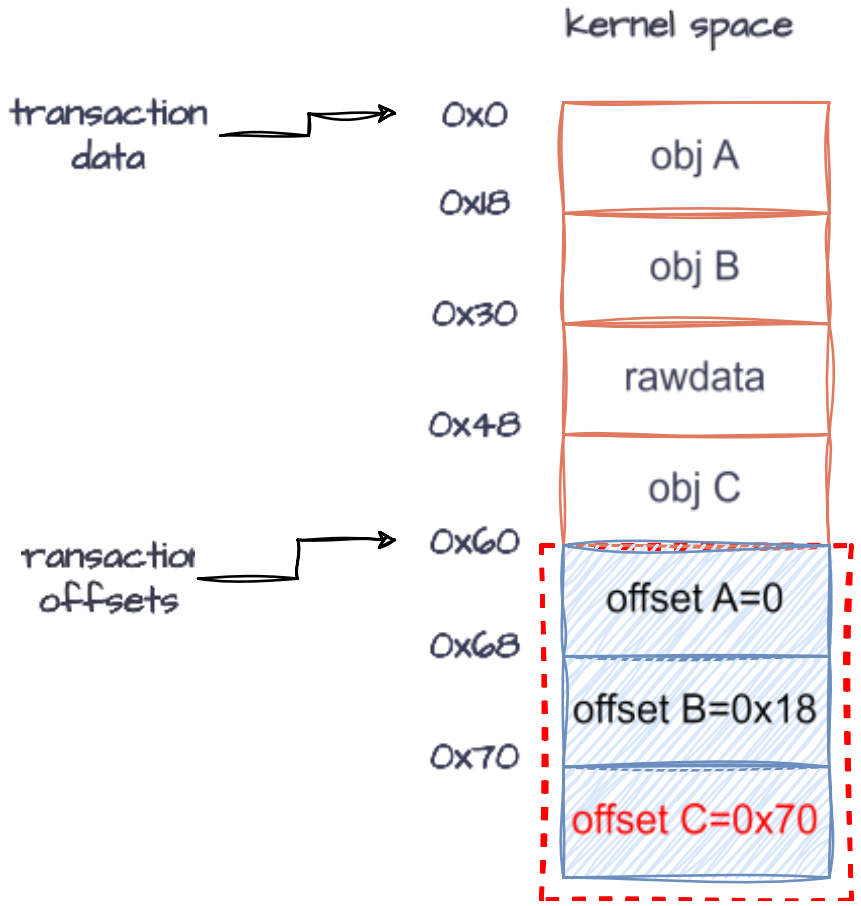
- Introduction





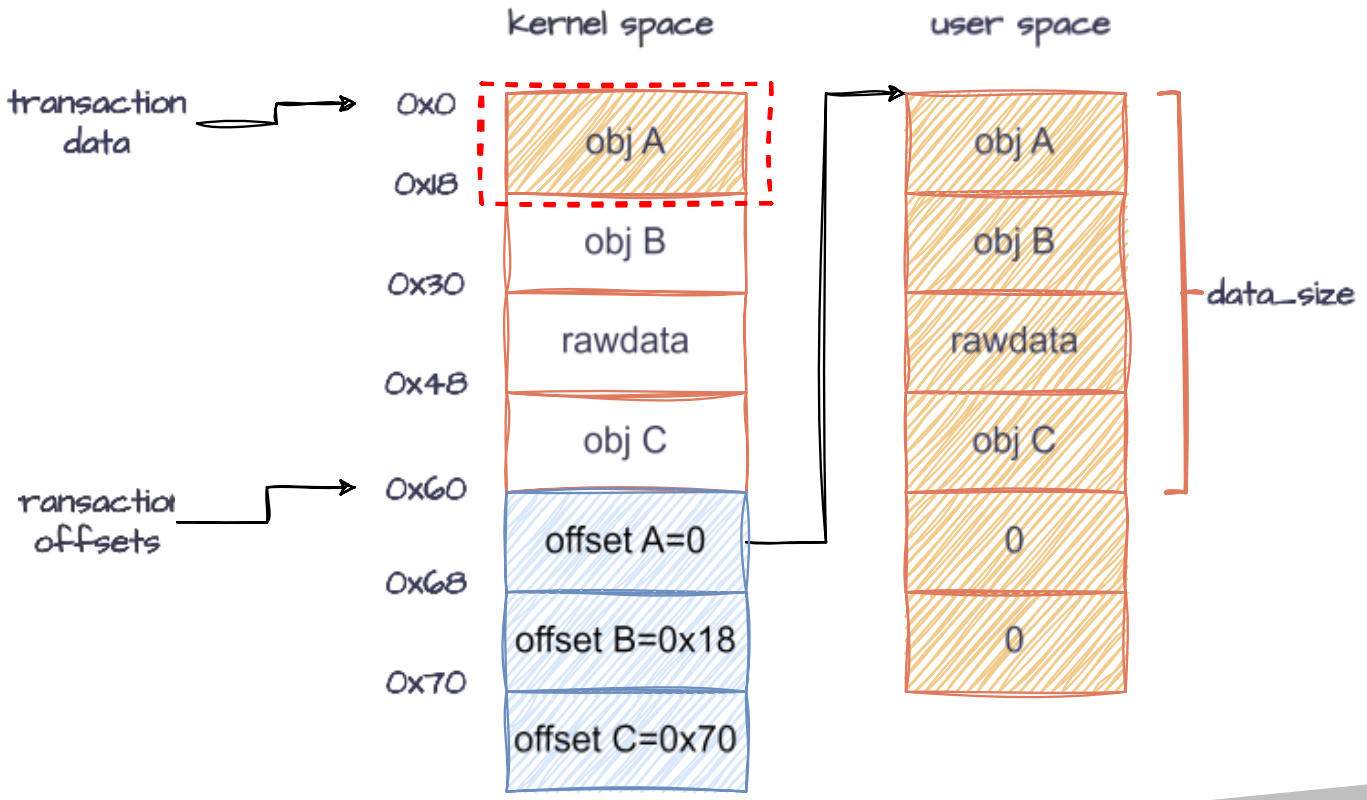
CVE-2024-46740

- copy the offsets from user space by binder_transaction



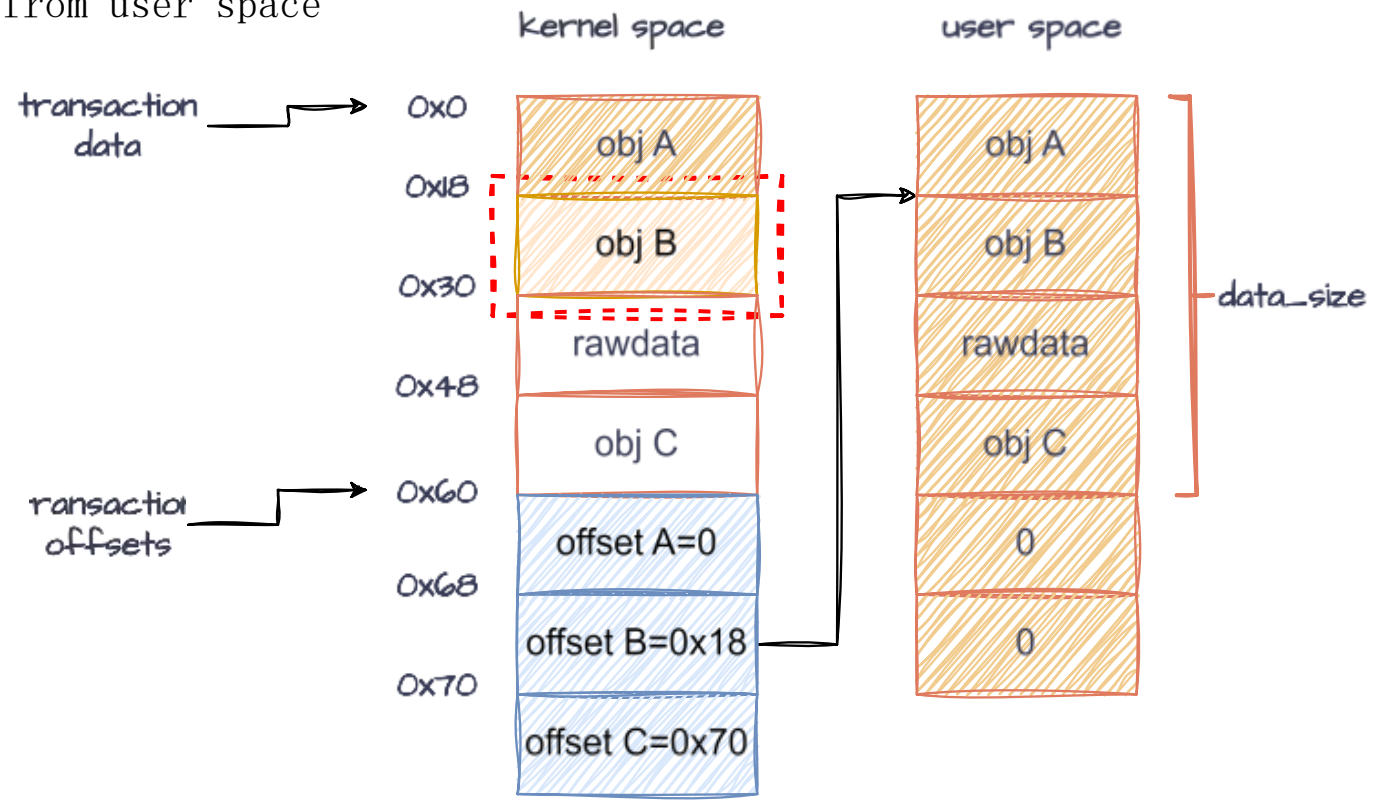
CVE-2024-46740

- Copy the offsets from user space
- Copy the objects and rawdata from user space



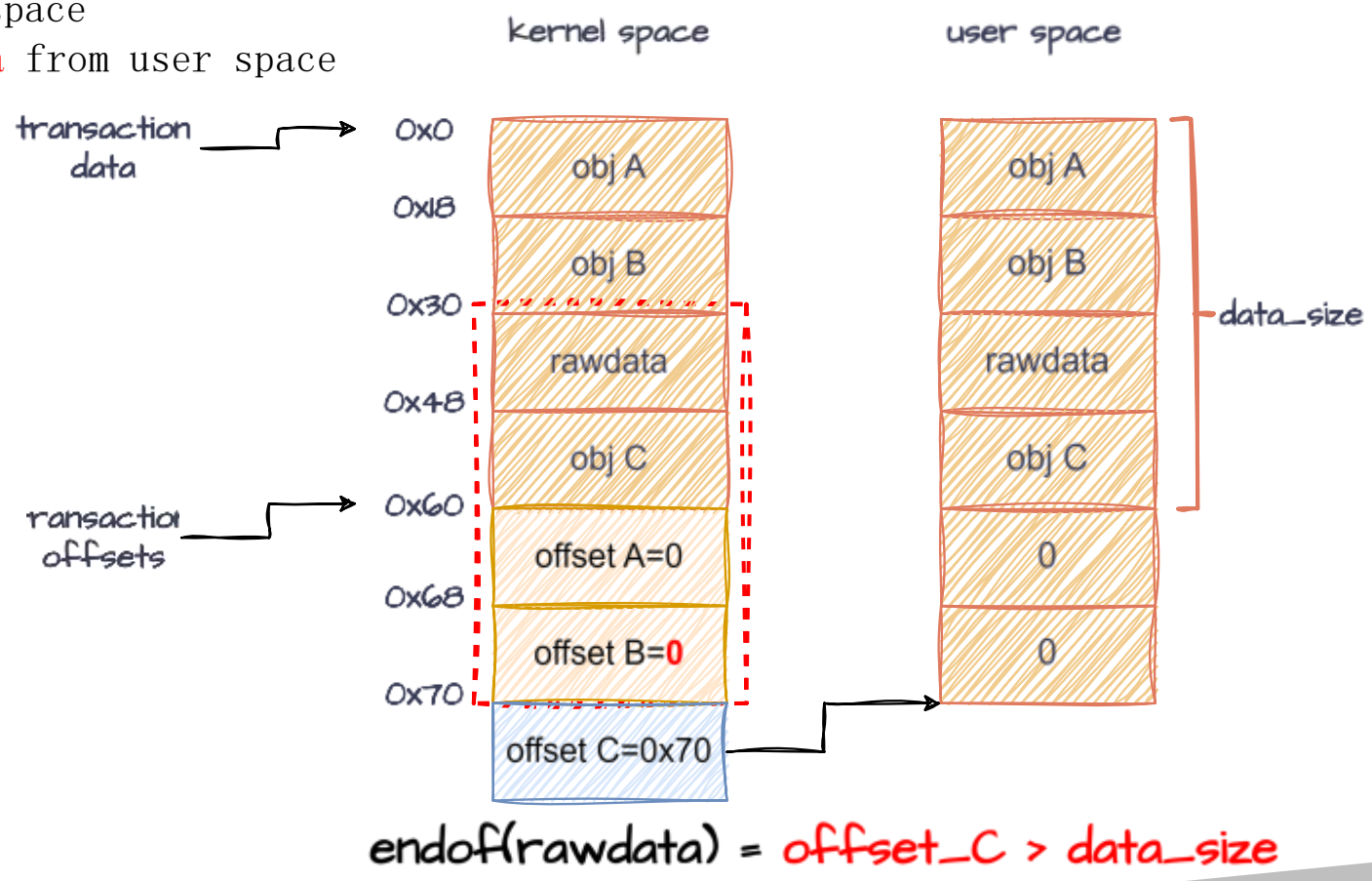
CVE-2024-46740

- Copy the offsets from user space
- Copy the objects and rawdata from user space



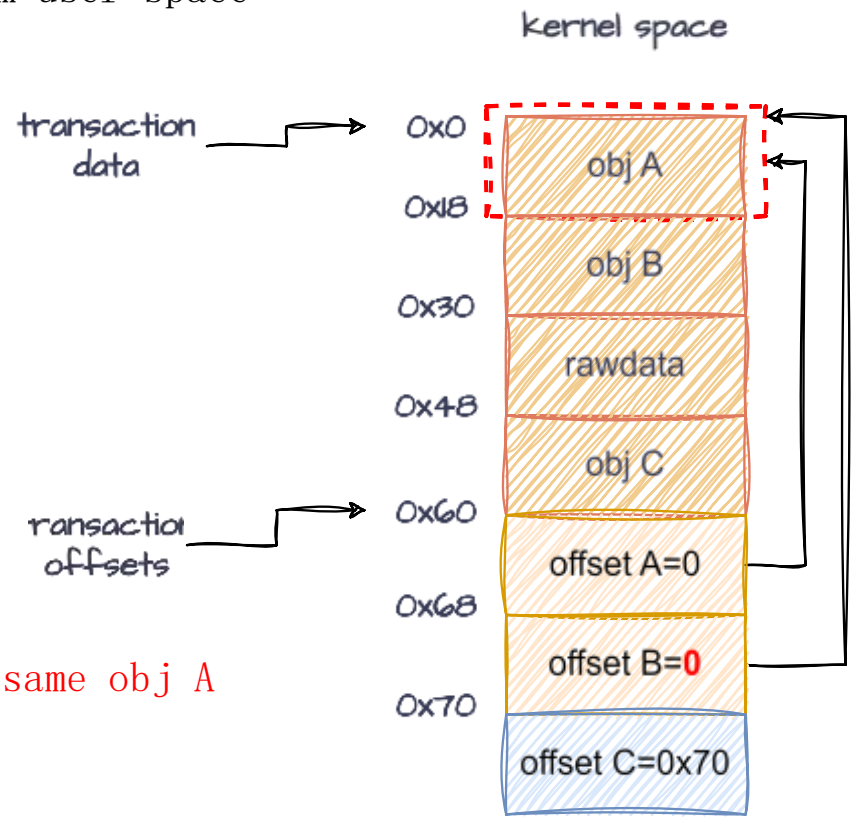
CVE-2024-46740

- Copy the offsets from user space
- Copy the objects and rawdata from user space



CVE-2024-46740

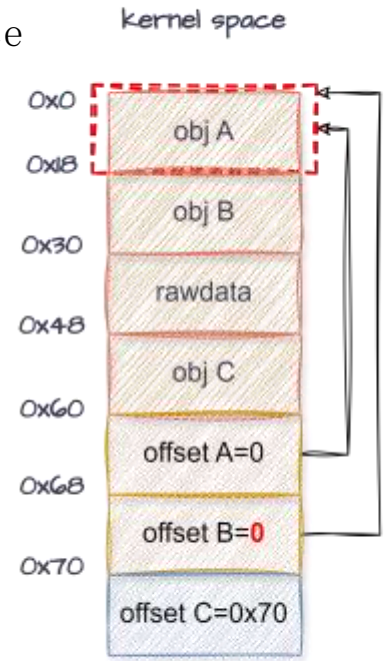
- Copy the offsets from user space
- Copy the objects and rawdata from user space

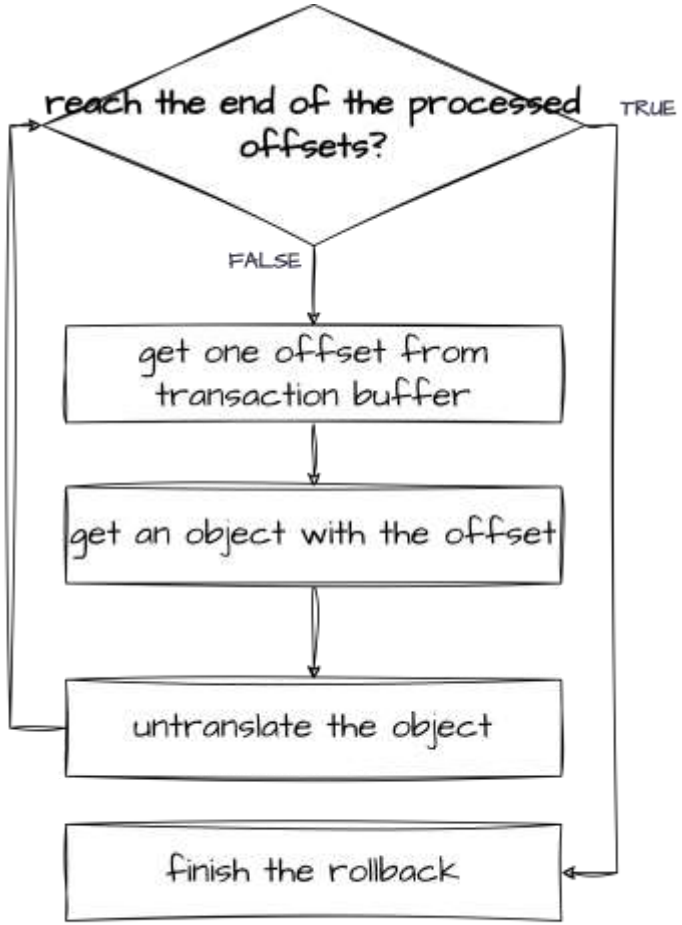


Overwrite 2 offsets, which point to the same obj A

CVE-2024-46740

- Copy the offsets from user space
- Copy the objects and rawdata from user space
- Roll back the transaction with error





untranslate obj A twice

Debt 2024

CVE-2024-46740

- Untranslate object incorrectly
- Struct binder_node UAF

PANGU TEAM

Offensive Complexity

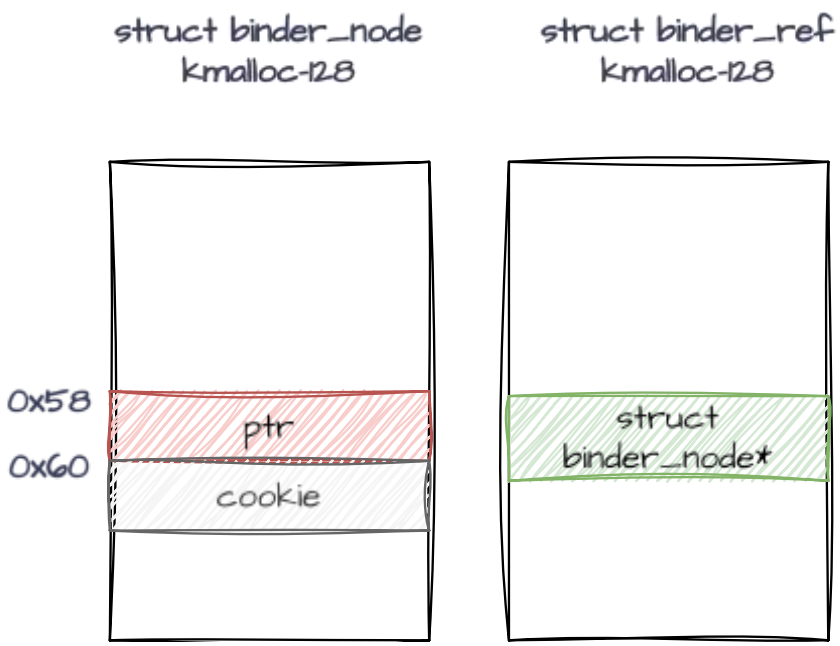
Android binder

Limitation of the old exploitation

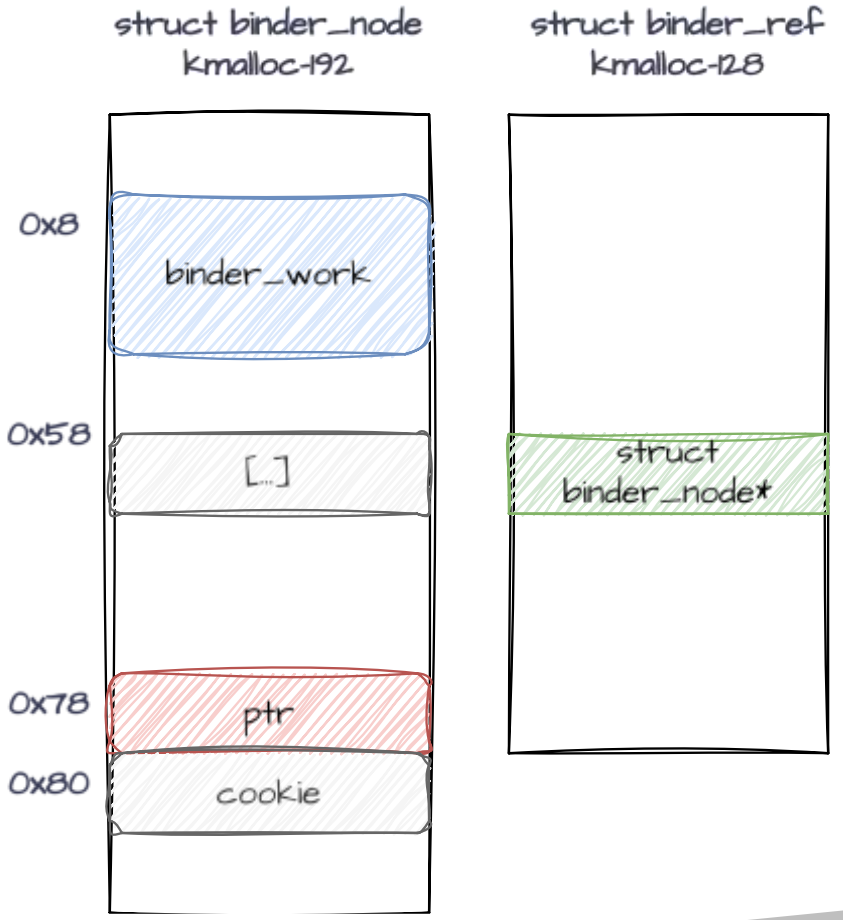
- 4 bytes read primitive would be slow
- Multiple heap spray and memory occupation
- H* series devices are not supported

Limitation of the old exploitation

P* series device

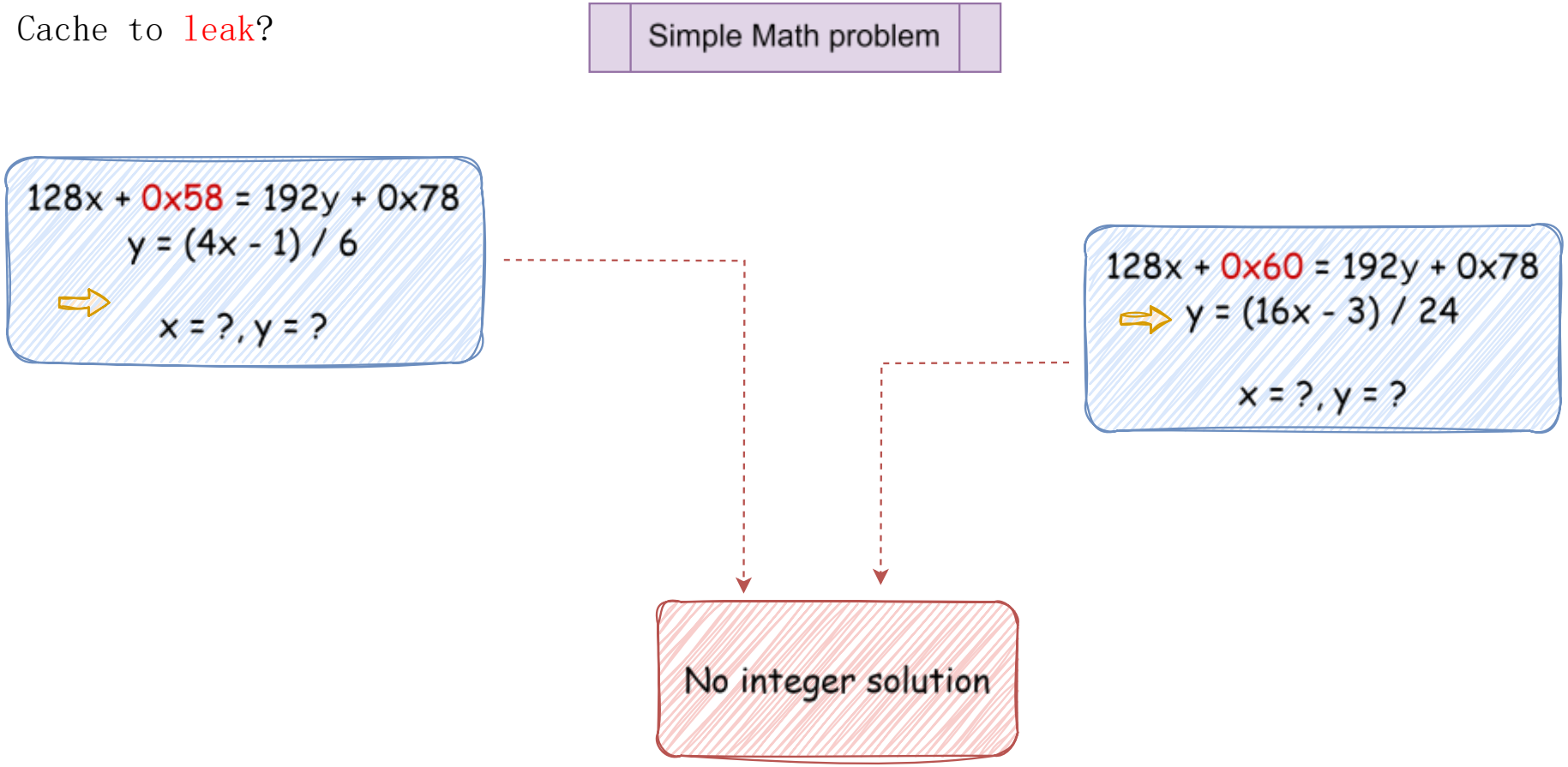


H* series device



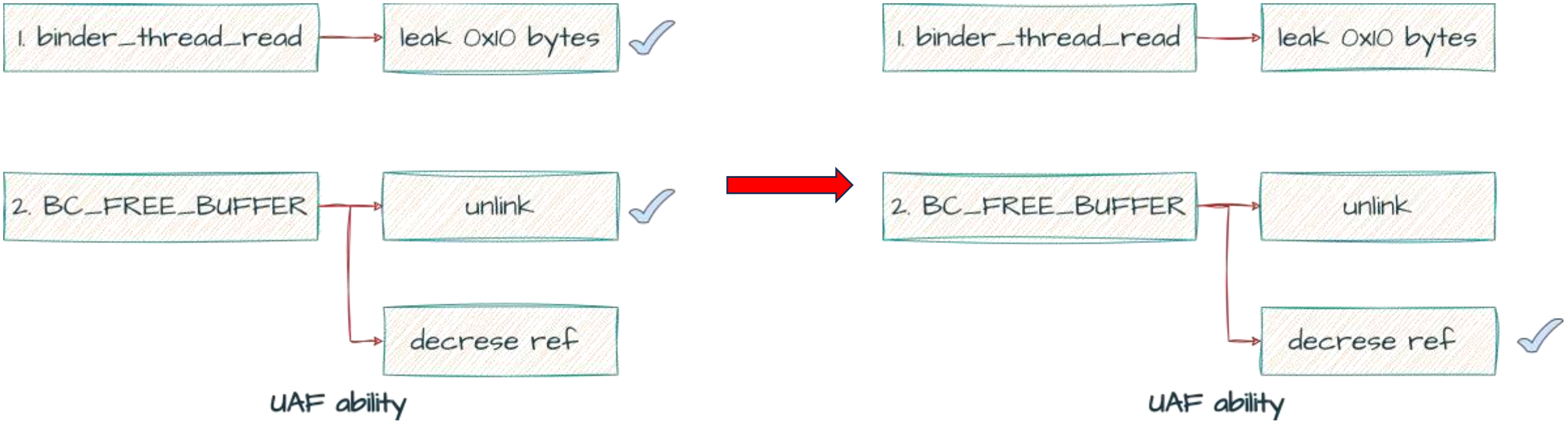
Limitation of the old exploitation

- Cross Cache to leak?



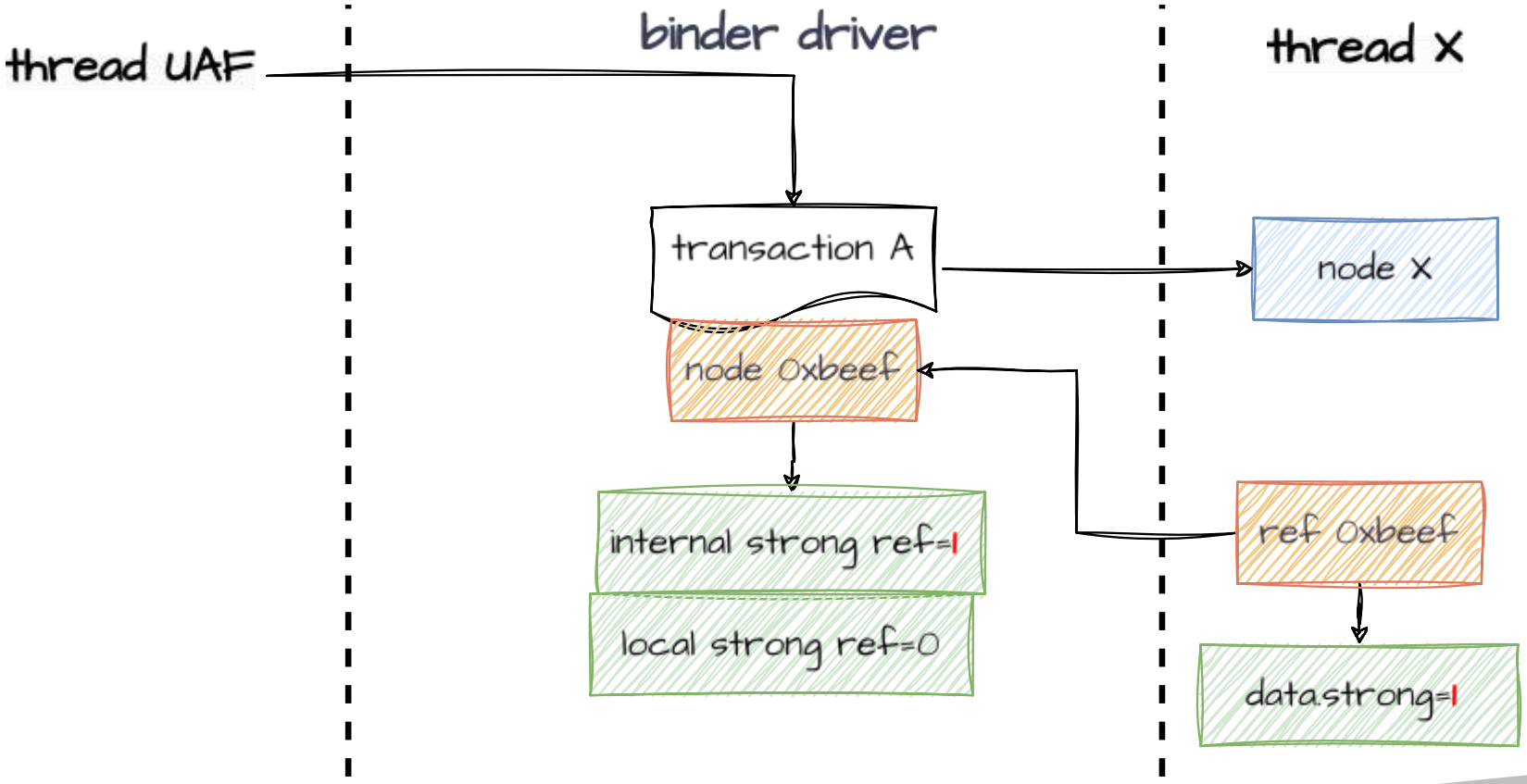
Our exploitation

- What can we do after struct binder_node is freed?



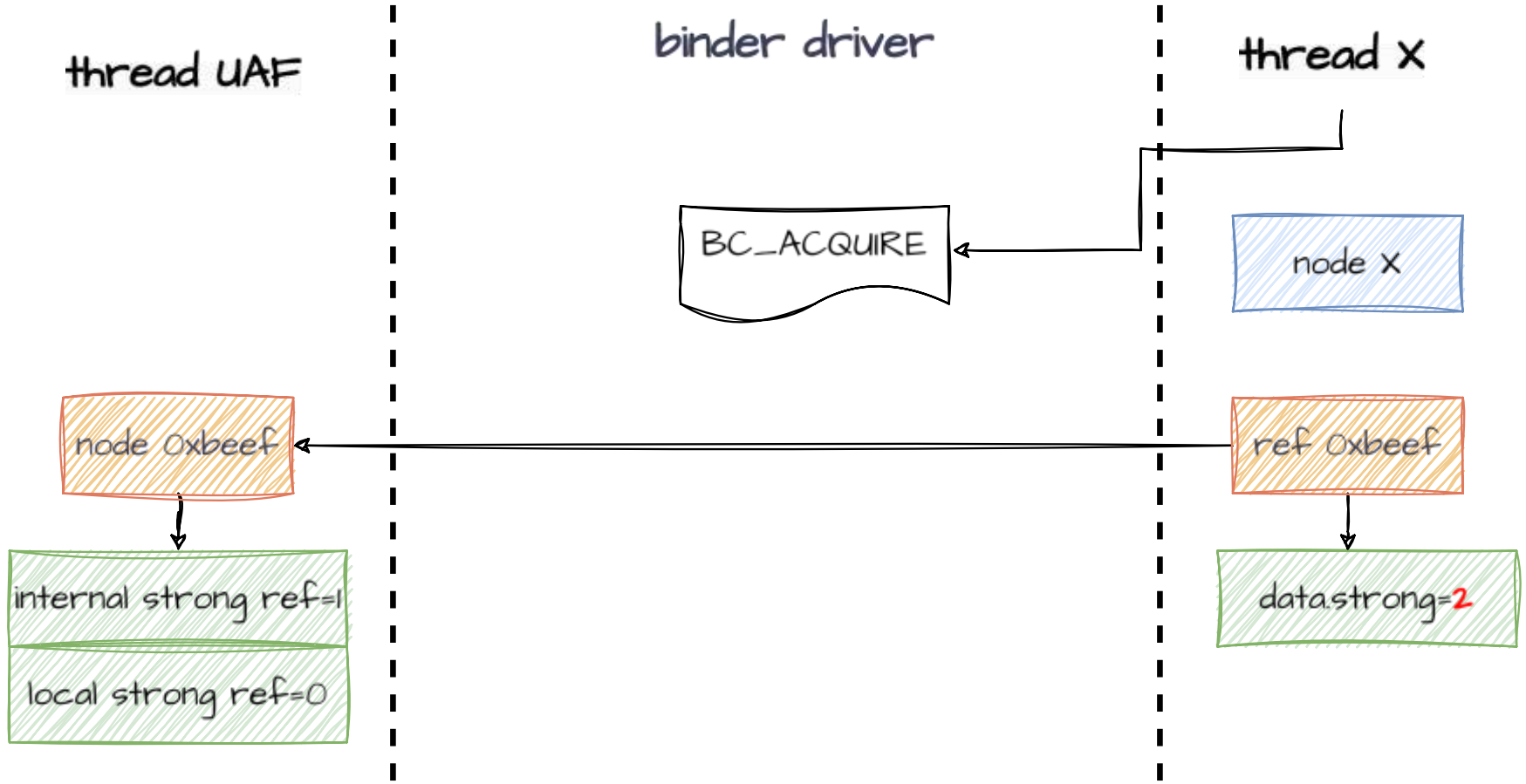
Our exploitation

- Construct a stable decrease primitive



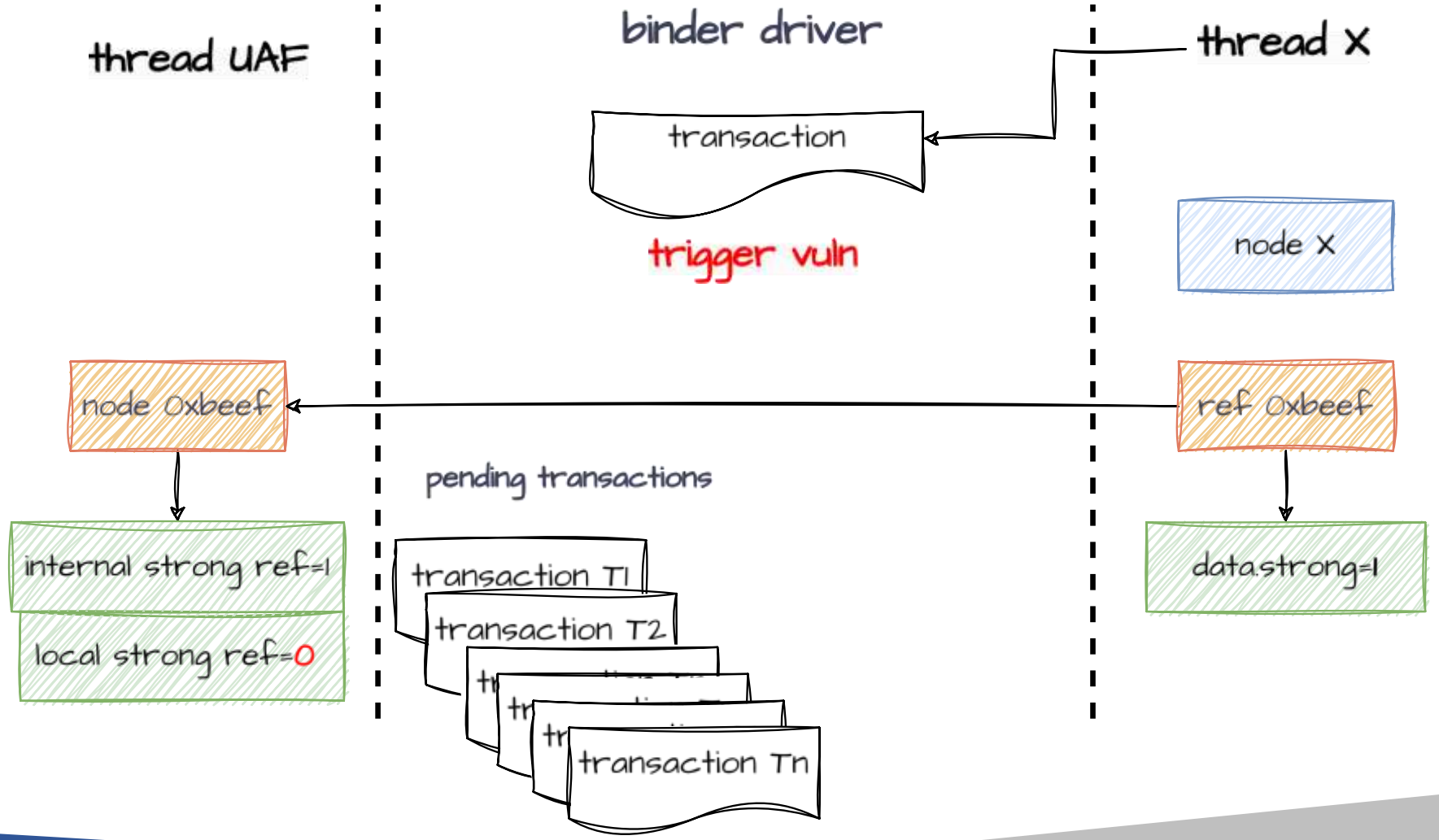
Decrement primitives

- Construct a stable decrease primitive



Decrement primitives

- Construct a stable decrease primitive

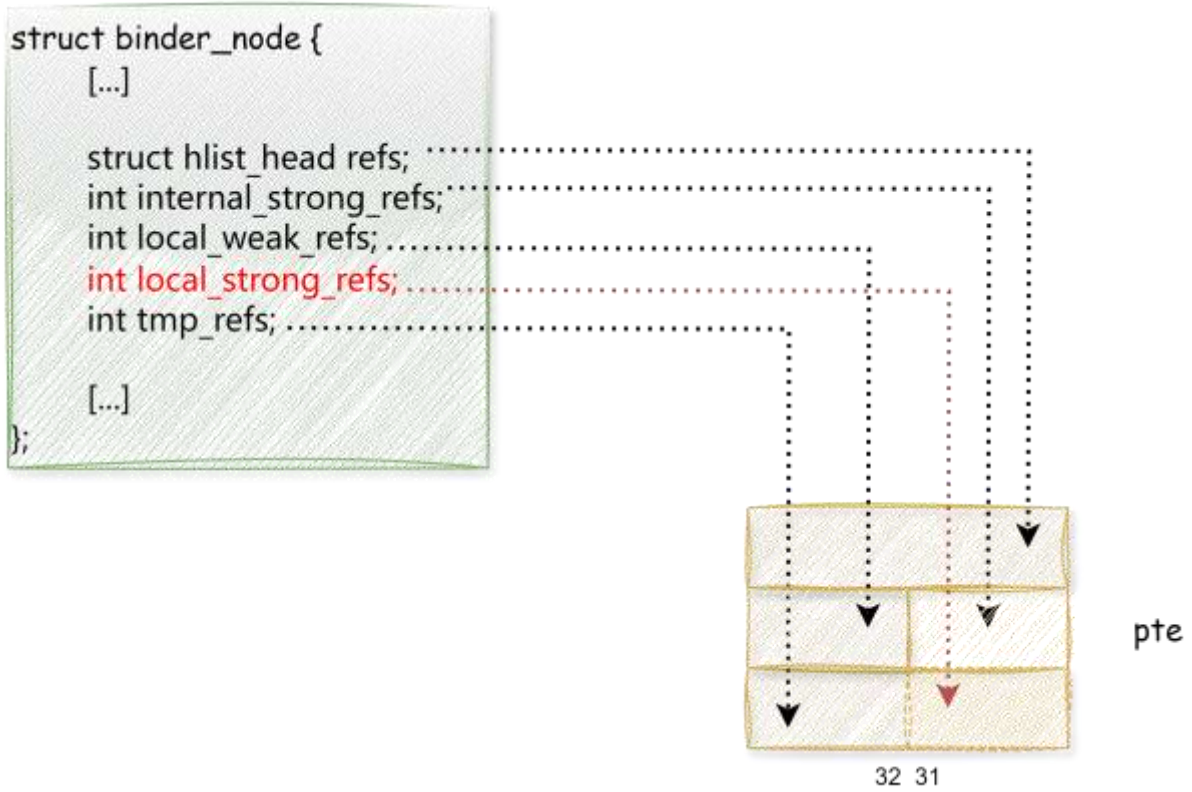


Our exploitation

- Construct a stable decrease primitive
  - ~0x1500 decrement primitives

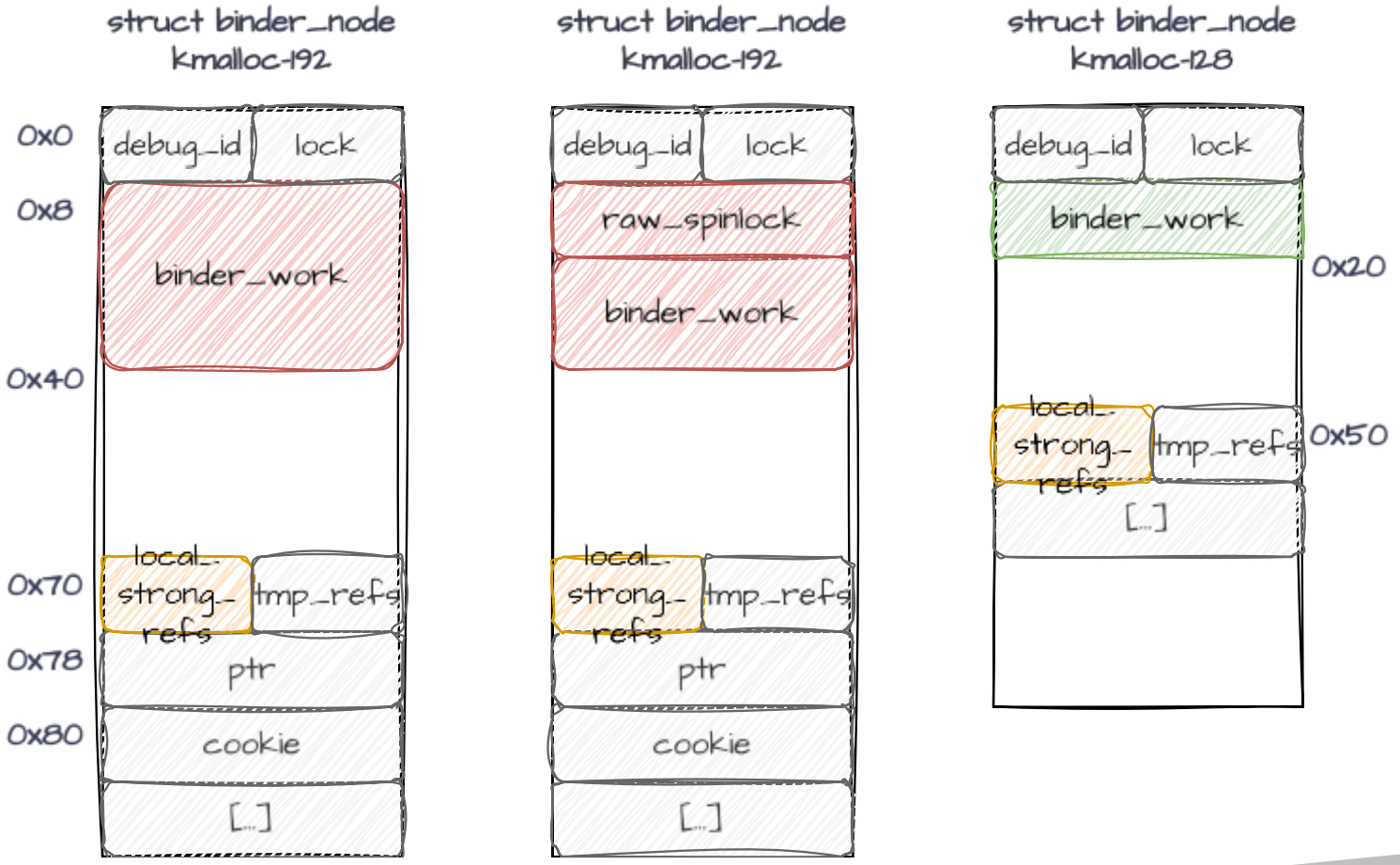
Apply dirty pagetable technique

- the local_strong_ref offset mapping



Apply dirty pagetable technique

- Support all device



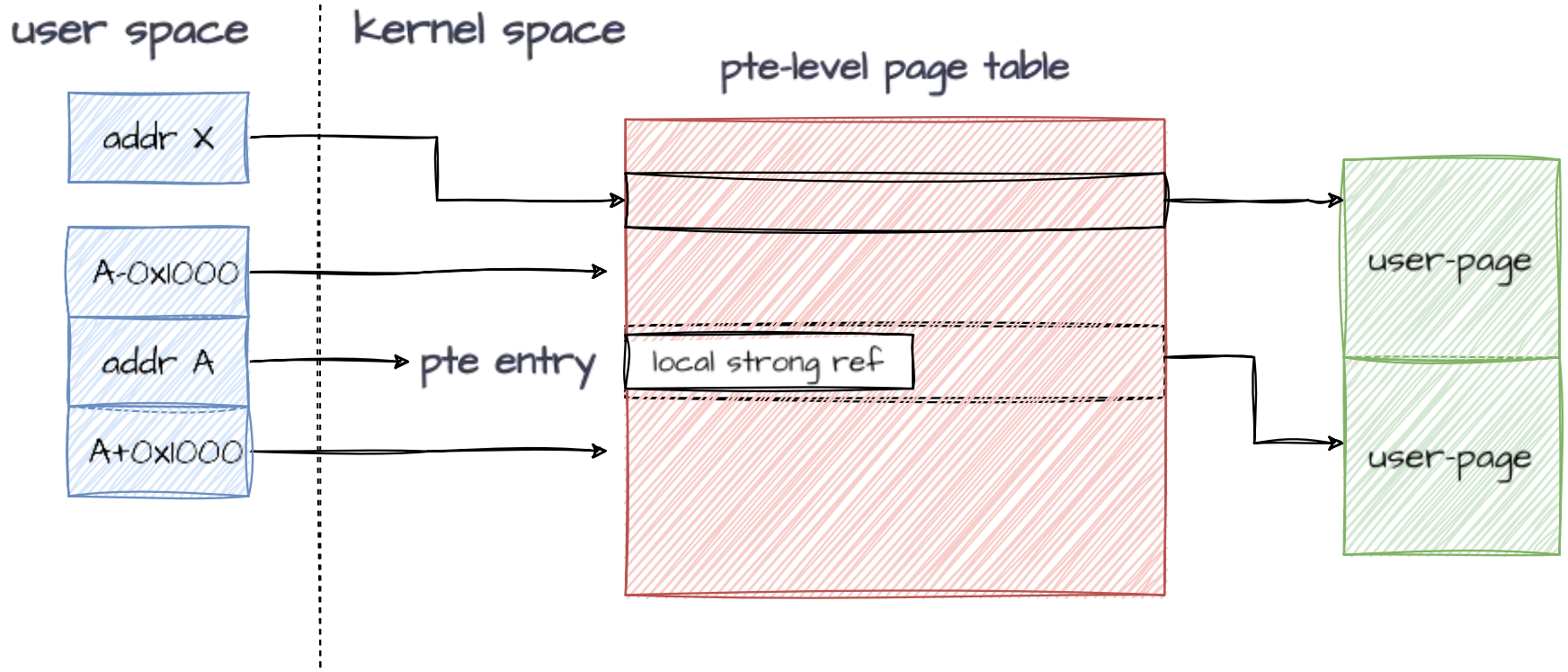
Our exploitation

- Construct a stable decrease primitive
  - ~0x1500 decrement primitives

- Apply dirty pagetable technique
  - Spray pagetable

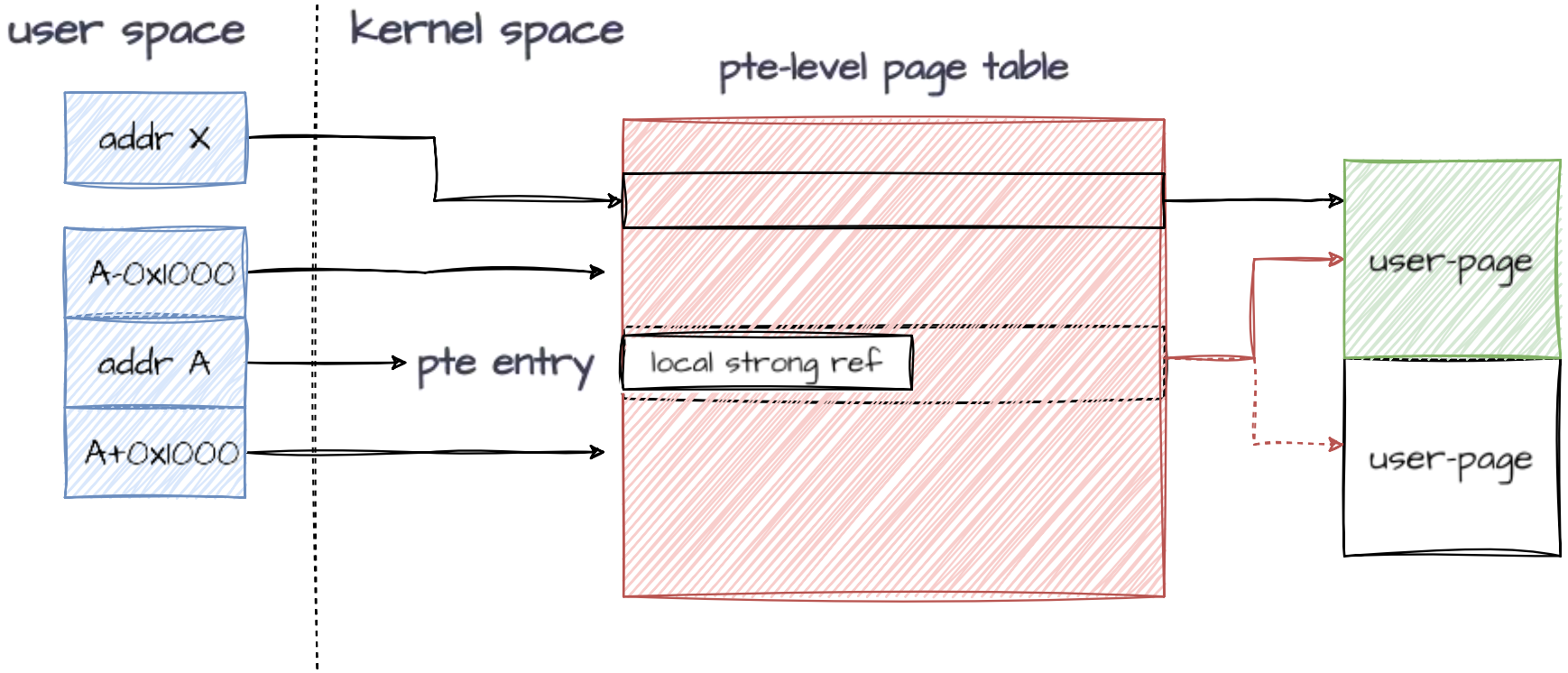
Apply dirty pagetable technique - old

- Spray crafted pagetable with limited dec primitive



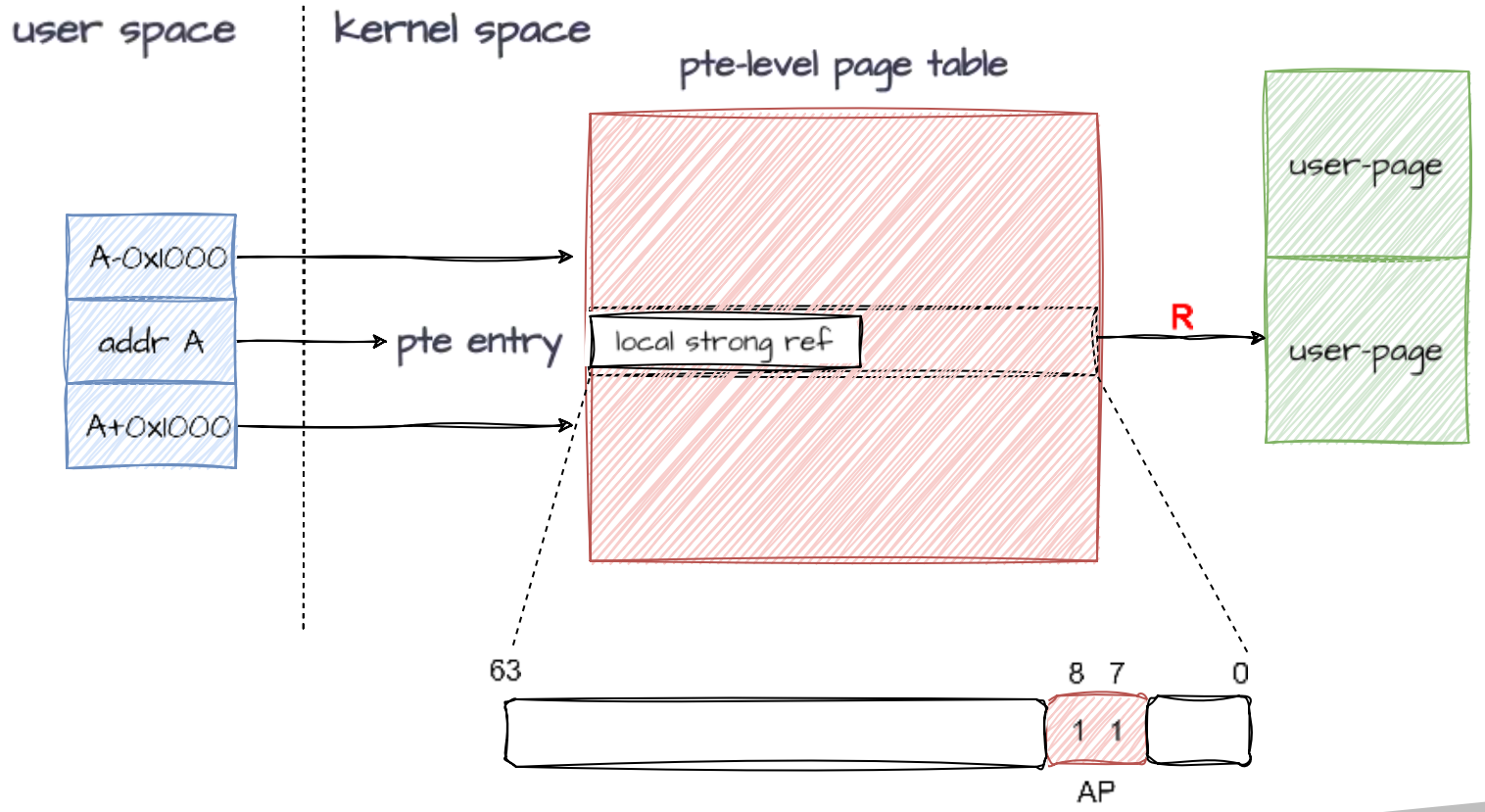
Apply dirty pagetable technique - old

- Decrease the local_strong_ref for 0x1000 times



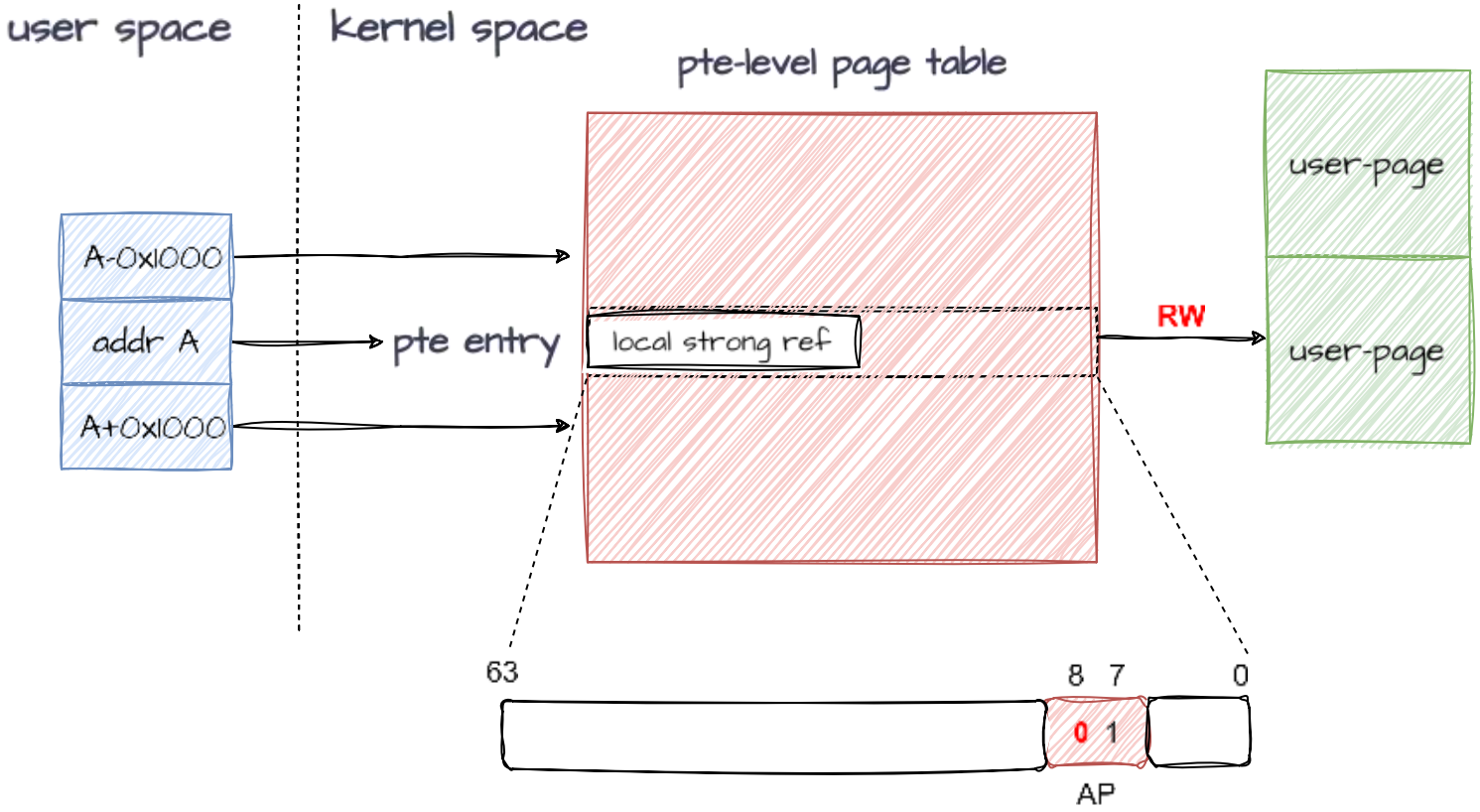
Apply dirty pagetable technique - new

- Tamper crafted pagetable with limited dec primitive



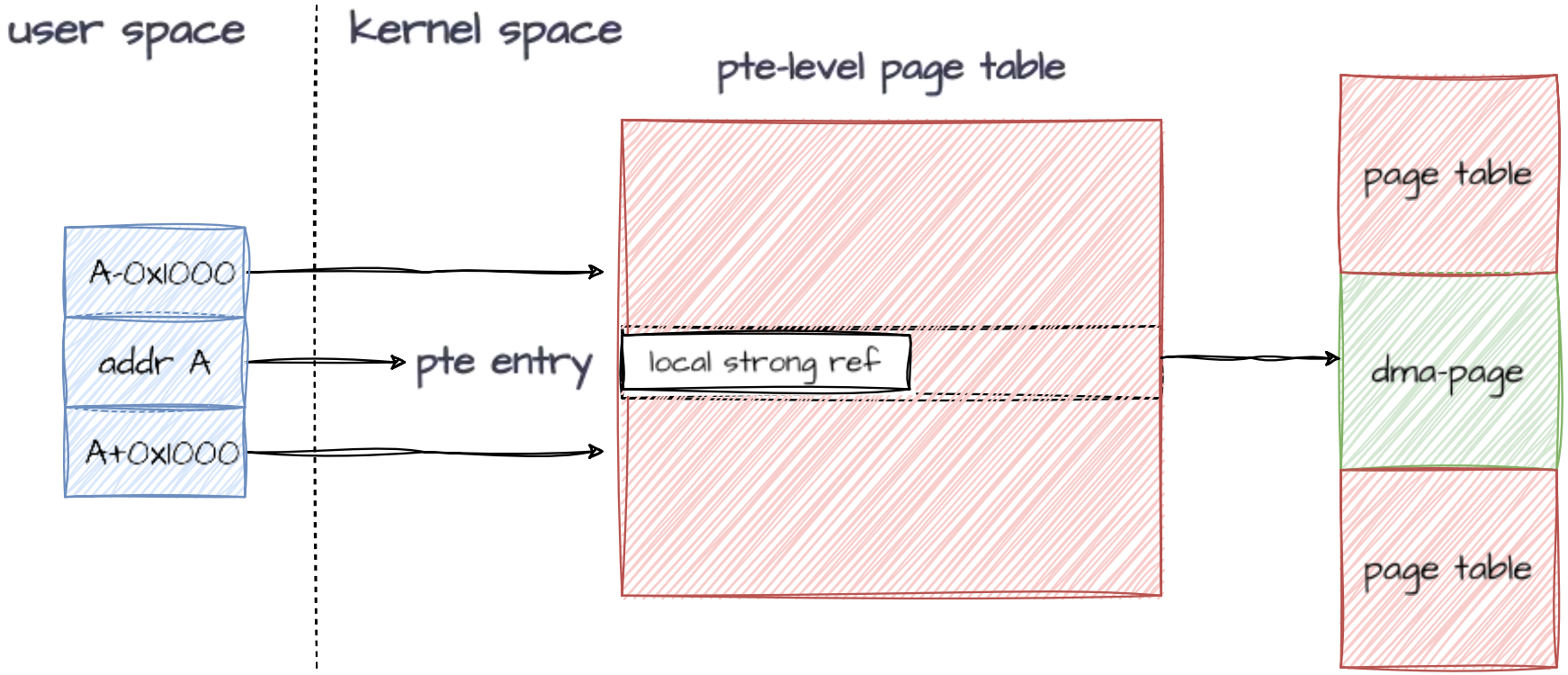
Apply dirty pagetable technique - new

- Decrease the local_strong_ref for 0x80 times



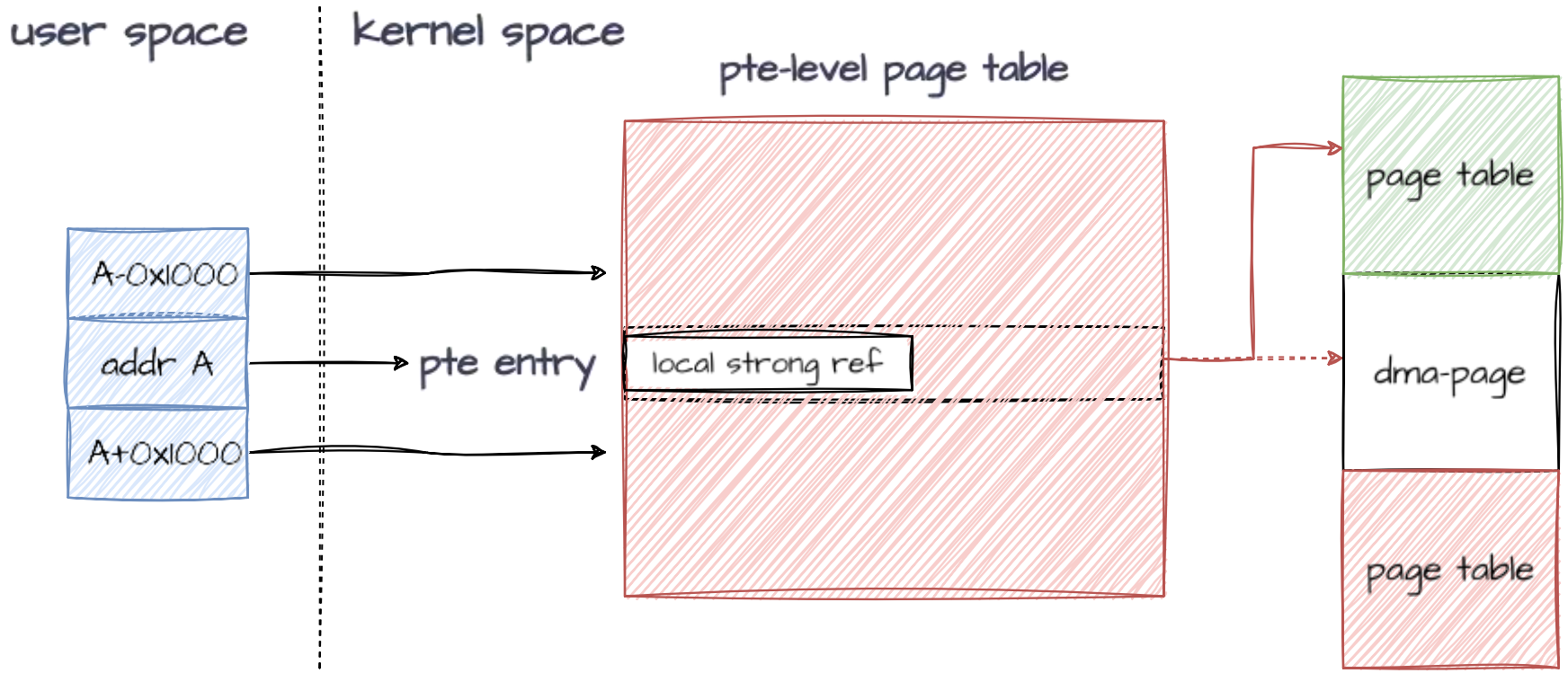
Apply dirty pagetable technique

- Remap addr A for DMA



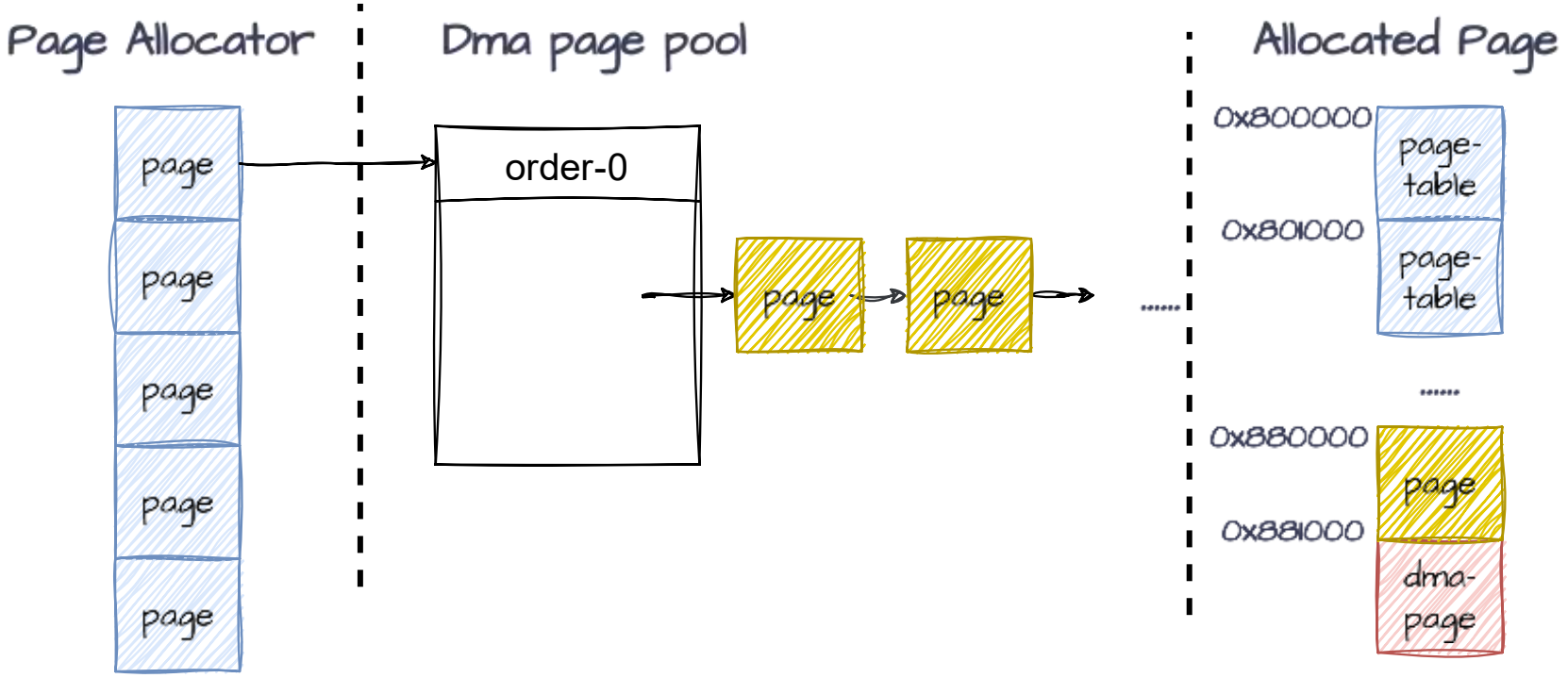
Apply dirty pagetable technique

- Decrease the local_strong_ref for 0x1000 times



Apply dirty pagetable technique

- Manipulate the page table to get physical address r/w



Our exploitation

- Construct a stable decrease primitive
  - ~0x1500 decrement primitives

- Apply dirty pagetable technique
  - Spray pagetable
  - Tamper pagetable

PANGU TEAM

Demo and Advance

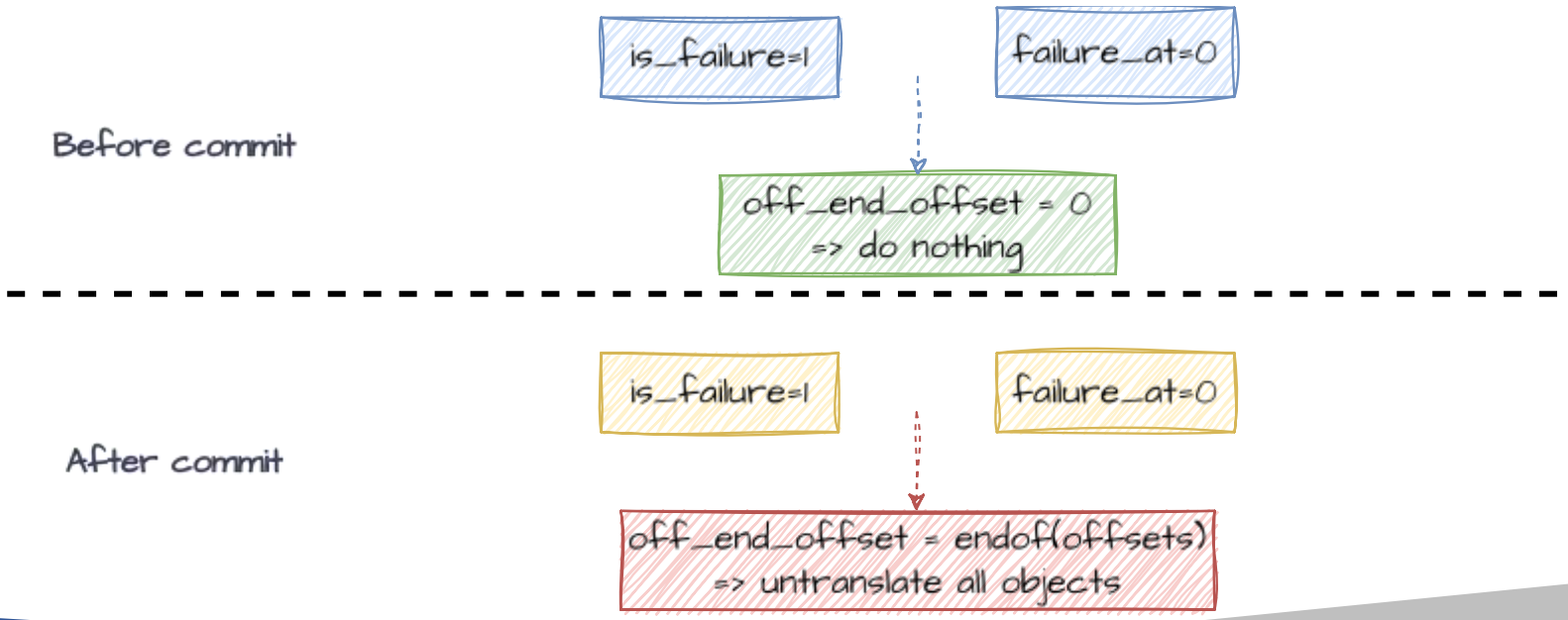
Android binder

Exploit videos

Patch suspicion

The commit introduced CVE-2023-20938

```
     off_start_offset = ALIGN(buffer->data_size, sizeof(void *));
-    off_end_offset = is_failure ? failed_at :
+    off_end_offset = is_failure && failed_at ? failed_at :
                 off_start_offset + buffer->offsets_size;
     for (buffer_offset = off_start_offset; buffer_offset < off_end_offset;
          buffer_offset += sizeof(binder_size_t)) {
```



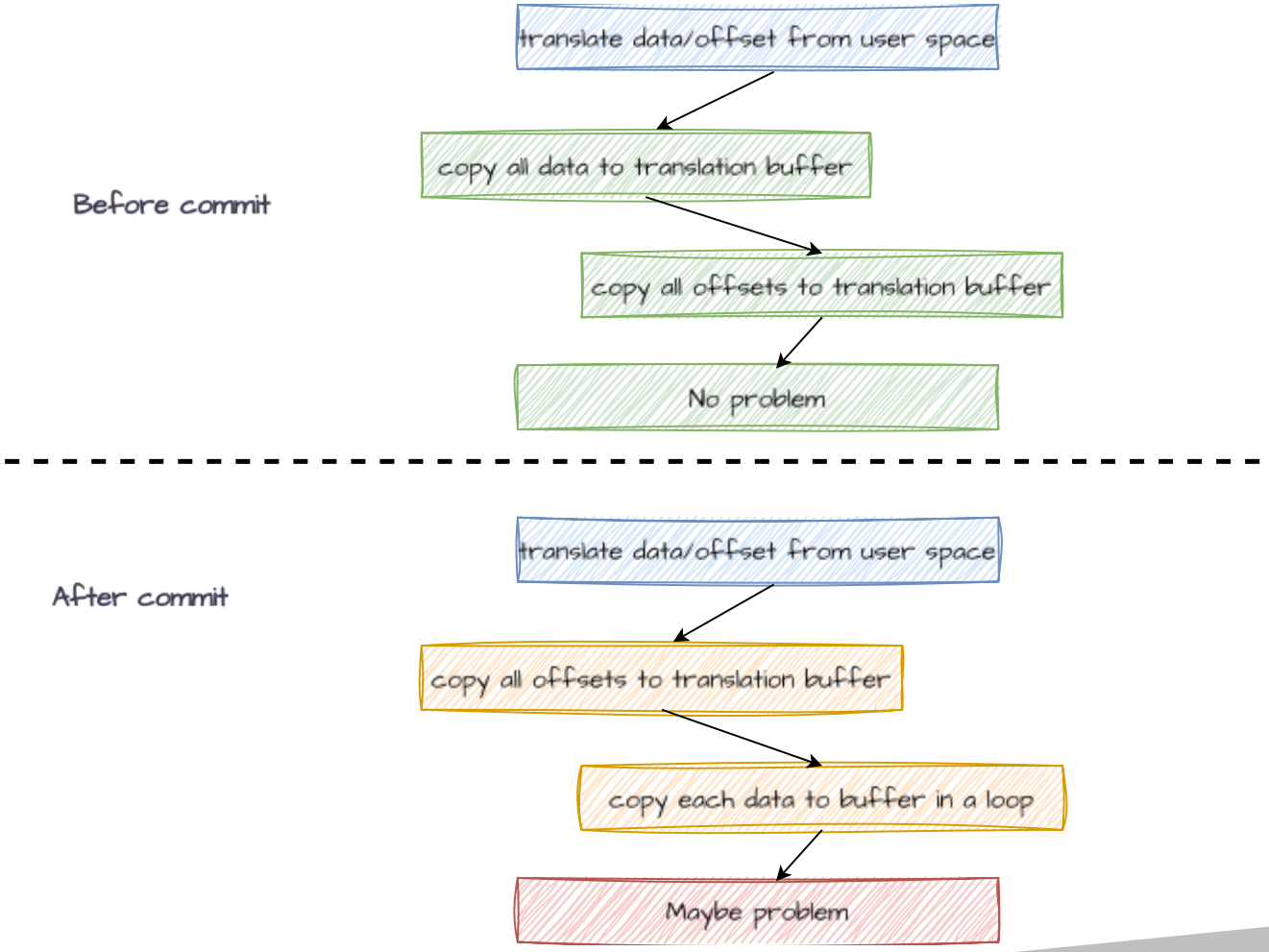
Patch suspicion

The commit patched CVE-2023-20938 and introduced CVE-2024-46740

```
- if (binder_alloc_copy_user_to_buffer(
-  &target_proc->alloc,
-  t->buffer, 0,
-  (const void __user *)
-   (uintptr_t)tr->data.ptr.buffer,
-  tr->data_size)) {                          // copy all transaction data from user space

[…]

for (buffer_offset = off_start_offset; buffer_offset < off_end_offset; buffer_offset +=
sizeof(binder_size_t)) {
+    if (copy_size && (user_offset > object_offset ||
+   binder_alloc_copy_user_to_buffer(          // copy each object of all data from user space
+    &target_proc->alloc,
+    t->buffer, user_offset,
+    user_buffer + user_offset,
+    copy_size))) {

+    object_size = binder_get_object(target_proc, user_buffer,
```

Patch suspicion

The commit patched CVE-2023-20938 and introduced CVE-2024-46740



1. CONFIG_ARM64_MTE

- Prevent memory corruption attacks

2. CONFIG_SLAB_VIRTUAL

- Prevent cross-cache attacks

3. Physical ASLR

- Prevent address guess attacks

PANGU TEAM

Thanks!

And question?