Simple bug but not easy exploit: Rooting Android devices in one shot

WANG, YONG (@ThomasKing2014)
Alibaba Cloud Pandora Lab

Whoami

- WANG, YONG @ThomasKing2014@infosec.exchange
  - @ThomasKing2014 on X/Weibo
- Security Engineer of Alibaba Cloud
- BlackHat{ASIA/EU/USA}/HITBAMS/Zer0Con/POC/CanSecWest/QPSS/MOSEC
- Nominated at Pwnie Award 2019(Best Privilege Escalation)

Agenda

- Introduction
- Bug analysis and exploitation
- Conclusion

MTE on the market

Friday, November 3, 2023

First handset with MTE on the market

By Mark Brand, Google Project Zero

Introduction

It's finally time for me to fulfill a long-standing promise. Since I first heard about ARM's Memory Tagging Extensions, I've said (to far too many people at this point to be able to back out...) that I'd immediately switch to the first available device that supported this feature. It's been a long wait (since late 2017) but with the release of the new Pixel 8 / Pixel 8 Pro handsets, there's finally a production handset that allows you to enable MTE!

The ability of MTE to detect memory corruption exploitation at the first dangerous access is a significant improvement in diagnostic and potential security effectiveness. The availability of MTE on a production handset for the first time is a big step forward, and I think there's real potential to use this technology to make 0-day harder.

I've been running my Pixel 8 with MTE enabled since release day, and so far I haven't found any issues with any of the applications I use on a daily basis[1], or any noticeable performance issues.

Currently, MTE is only available on the Pixel as a developer option, intended for app developers to test their apps using MTE, but we can configure it to default to synchronous mode for all[2] apps and native user mode binaries. This can be done on a stock image, without bootloader unlocking or rooting required - just a couple of debugger commands. We'll do that now, but first:

Disclaimer

This is absolutely not a supported device configuration; and it's highly likely that you'll encounter issues with at least some applications crashing or failing to run correctly with MTE if you set your device up in this way.

Android kernel mitigations 101

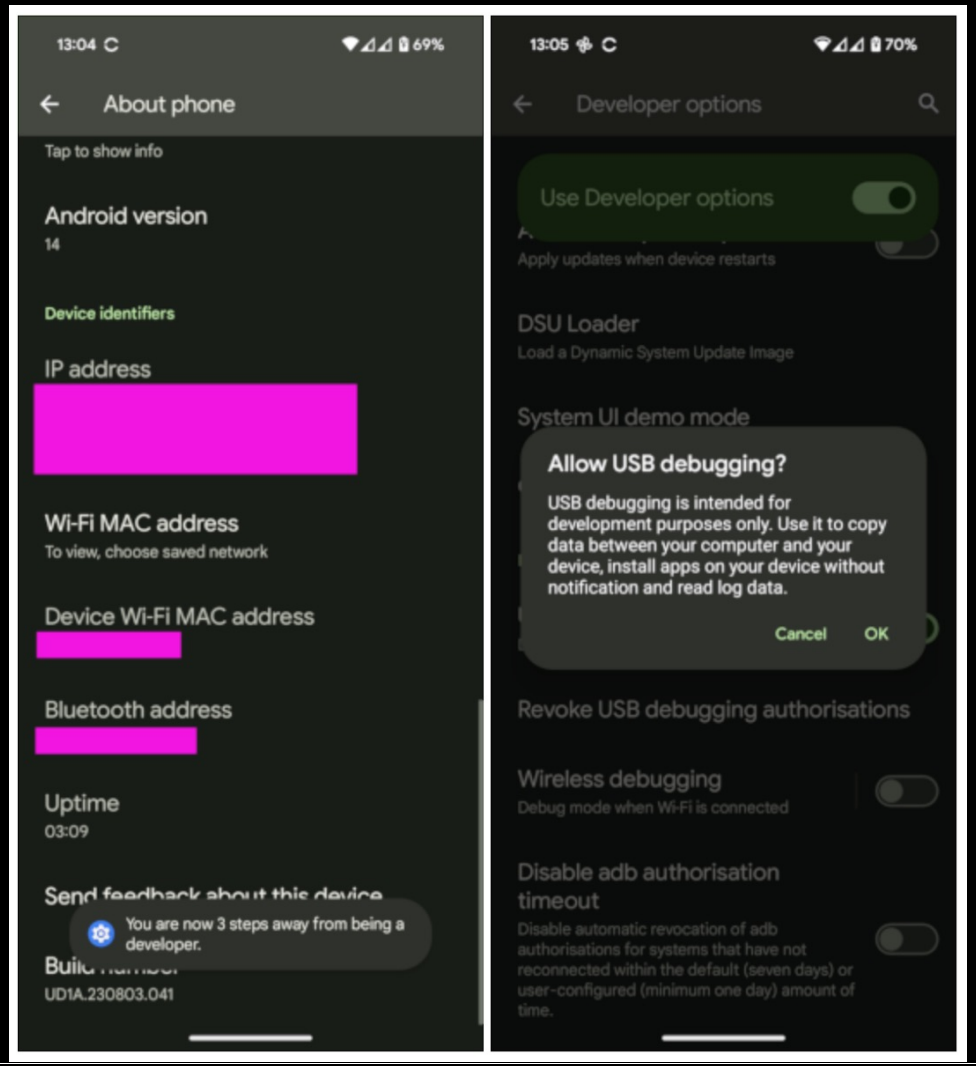
- PXN - Privileged eXecute Never
- PAN - Privileged Access Never
- UAO - User Access Override
- PAC - Pointer Authentication Code
- MTE - Memory Tagging Extension
- KASLR - Kernel Address Space Layout Randomization
- CONFIG_DEBUG_LIST
- CONFIG_SLAB_FREELIST_RANDOM/HARDENED
- # CONFIG_SLAB_MERGE_DEFAULT is not set
- CONFIG_BPF_JIT_ALWAYS_ON

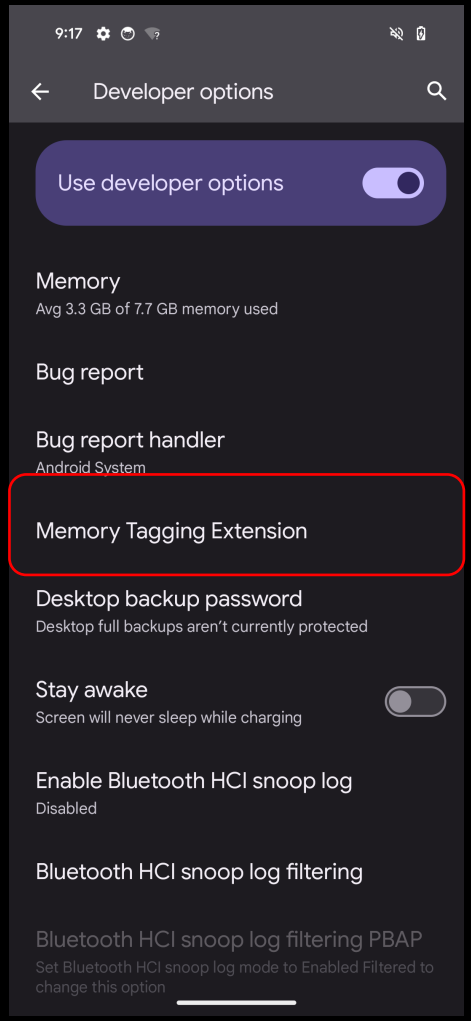
Android kernel mitigations 101

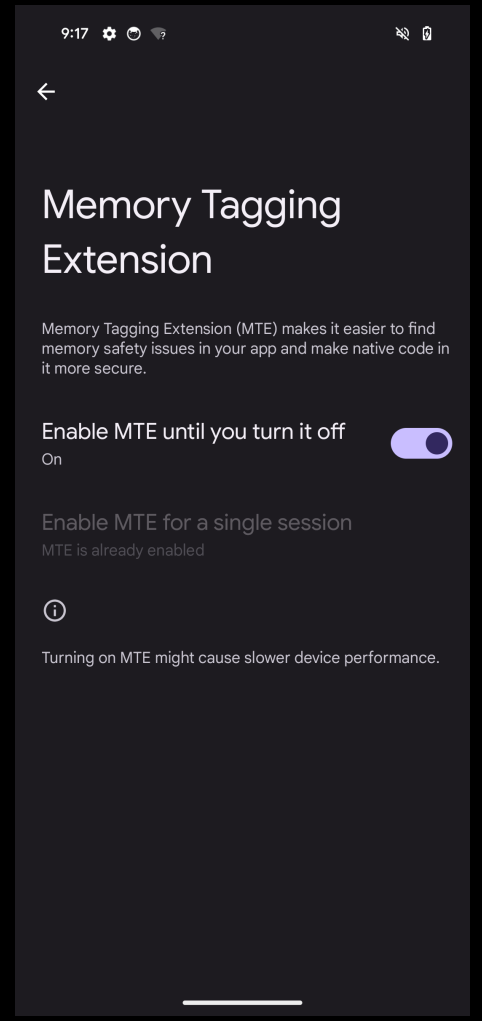
- PXN - Privileged eXecute Never
- PAN - Privileged Access Never
- UAO - User Access Override
- PAC - Pointer Authentication Code
- MTE - Memory Tagging Extension
- KASLR - Kernel Address Space Layou
- CONFIG_DEBUG_LIST
- CONFIG_SLAB_FREELIST_RANDOM
- # CONFIG_SLAB_MERGE_DEFAULT
- CONFIG_BPF_JIT_ALWAYS_ON

```
~ $ uname -a
Linux localhost 5.15.110-android14-11-ga6d7915820a0-ab10726252
 #1 SMP PREEMPT Mon Aug 28 18:42:09 UTC 2023 aarch64 Android
~ $ lscpu
Architecture:          aarch64
  CPU op-mode(s):      64-bit
  Byte Order:          Little Endian
CPU(s):                9
  On-line CPU(s) list: 0-8
Vendor ID:             ARM
  Model:               1
  Thread(s) per core:  1
  Core(s) per socket:  4
  Socket(s):           1
  Stepping:            r1p1
  CPU max MHz:         1704.0000
  CPU min MHz:         324.0000
  BogoMIPS:            49.15
  Flags:               fp asimd evtstrm aes pmull sha1 sha2 cr
                       c32 atomics fphp asimdhp cpuid asimdrdm
                        jscvt fcma lrcpc dcpop sha3 sm3 sm4 as
                       imddp sha512 sve asimdfhm dit uscat ilr
                       cpc flagm ssbs sb paca pacg dcpodp sve2
                        sveaes svepmull svebitperm svesha3 sve
                       sm4 flagm2 frint svei8mm svebf16 i8mm b
                       ti mte mte3
```

Memory Tagging Extension







Memory Tagging Extension

Synchronous mode (SYNC)

This mode is optimized for correctness of bug detection over performance and can be used as a precise bug detection tool, when higher performance overhead is acceptable. When enabled, MTE SYNC acts as a security mitigation. On a tag mismatch, the processor aborts execution immediately and terminates the process with `SIGSEGV` (code `SEGV_MTESERR`) and full information about the memory access and the faulting address.

We recommend using this mode during testing as an alternative to HWASan/KASAN or in production when the target process represents a vulnerable attack surface. In addition, when ASYNC mode has indicated the presence of a bug, an accurate bug report can be obtained by using the runtime APIs to switch execution to SYNC mode.

When running in SYNC mode, the Android allocator records stack traces for all allocations and deallocations and uses them to provide better error reports that include an explanation of a memory error, such as use-after-free, or buffer-overflow, and the stack traces of the relevant memory events. Such reports provide more contextual information and make bugs easier to trace and fix.

Asynchronous mode (ASYNC)

This mode is optimized for performance over accuracy of bug reports and can be used as low-overhead detection for memory safety bugs.
On a tag mismatch, the processor continues execution until the nearest kernel entry (for example, a syscall or timer interrupt), where it terminates the process with `SIGSEGV` (code `SEGV_MTEAERR`) without recording the faulting address or memory access.
We recommend using this mode in production on well tested codebases where the density of memory safety bugs is known to be low, which is achieved by using the SYNC mode during testing.

Asymmetric mode (ASYMM)

An additional feature in Arm v8.7-A, Asymmetric MTE mode provides synchronous checking on memory reads, and asynchronous checking of memory writes, with performance similar to that of the ASYNC mode. In most situations, this mode is an improvement over the ASYNC mode, and we recommend using it instead of ASYNC whenever it is available.

For this reason, none of the APIs described below mention the Asymmetric mode. Instead, the OS can be configured to always use Asymmetric mode when Asynchronous is requested. Please refer to the "Configuring the CPU-specific preferred MTE level" section for more information.

Memory Tagging Extension

- User space
  - setprop arm64.memtag.bootctl memtag
  - setprop persist.arm64.memtag.default sync
  - setprop persist.arm64.memtag.app_default sync
- Kernel
  - arm64.memtag.bootctl memtag,memtag-kernel
- Reboot the phone

MTE- kernel

[0.000000] Kernel command line: console=ttynull stack_depot_disable=on cgroup_disable=pressure kasan.page_alloc.sample=10 kasan.stacktrace=off bootconfig ioremap_guard kvm-arm.mode=protected root=/dev/ram0 rw clocksource=arch_sys_counter clk_ignore_unused loop.max_part=7 loop.hw_queue_depth=31 coherent_pool=4M firmware_class.path=/vendor/firmware irqaffinity=0 swiotlb=noforce sysrq_always_enabled no_console_suspend softlockup_panic=1 kasan_multi_shot kvm-arm.protected_modules=exynos-pd,pkvm_s2mpu-v9 exynos_drm.load_sequential=1 g2d.load_sequential=1 samsung_iommu_v9.load_sequential=1 swiotlb=noforce disable_dma32=on printk.devkmsg=on cma_sysfs.experimental=Y cgroup_disable=memory rcupdate.rcu_expedited=1 rcu_nocbs=all swiotlb=1024 cgroup.memory=nokmem sysctl.kernel.sched_pelt_multiplier=4 kasan=off at24.write_timeout=100 log_buf_len=1024K bootconfig console=null exynos_drm.panel_name=google-bigsurf.04a050 tcpci_max77759.conf_sbu=0 arm64.nomte ufs_pixel_fips140.fips_firs

MTE- kernel

[0.000000] Kernel command line: console=ttynull stack_depot_disable=on cgroup_disable=pressure kasan.page_alloc.sample=10 kasan.stacktrace=off bootconfig ioremap_guard kvm-arm.mode=protected root=/dev/ram0 rw clocksource=arch_sys_counter clk_ignore_unused loop.max_part=7 loop.hw_queue_depth=31 coherent_pool=4M firmware_class.path=/vendor/firmware irqaffinity=0 swiotlb=noforce sysrq_always_enabled no_console_suspend softlockup_panic=1 kasan_multi_shot kvm-arm.protected_modules=exynos-pd,pkvm_s2mpu-v9 exynos_drm.load_sequential=1 g2d.load_sequential=1 samsung_iommu_v9.load_sequential=1 swiotlb=noforce disable_dma32=on printk.devkmsg=on cma_sysfs.experimental=Y cgroup_disable=memory rcupdate.rcu_expedited=1 rcu_nocbs=all swiotlb=1024 cgroup.memory=nokmem sysctl.kernel.sched_pelt_multiplier=4 kasan=off at24.write_timeout=100 log_buf_len=1024K bootconfig console=null exynos_drm.panel_name=google-bigsurf.04a050 tcpci_max77759.conf_sbu=0 kasan=on ufs_pixel_fips140.fips_first_l

[0.000000] kasan: KernelAddressSanitizer initialized (hw-tags, mode=sync, vmalloc=on, stacktrace=off)

MTE- kernel

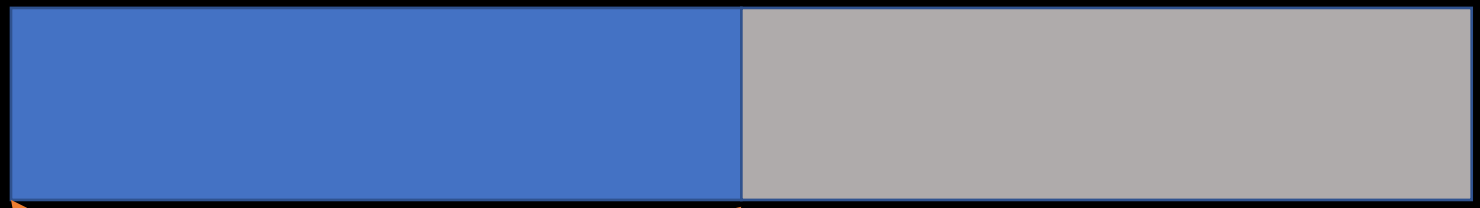
```
// ffffff8850f82b00 --> f1ffff8850f82b00
#define KASAN_TAG_KERNEL    0xFF /* native kernel pointers tag */
#define KASAN_TAG_INVALID   0xFE /* inaccessible memory tag */
#define KASAN_TAG_MAX       0xFD /* maximum value for random tags */

#ifdef CONFIG_KASAN_HW_TAGS
#define KASAN_TAG_MIN       0xF0 /* minimum value for random tags */
#else
#define KASAN_TAG_MIN       0x00 /* minimum value for random tags */
#endif

static inline u8 mte_get_ptr_tag(void *ptr)
{
    /* Note: The format of KASAN tags is 0xF<x> */
    u8 tag = 0xF0 | (u8)(((u64)(ptr)) >> MTE_TAG_SHIFT);

    return tag;
}
```

MTE- kernel



```
char *p = kmalloc(32, GFP_kernel);
p[32] = ‘A’;
```

MTE- kernel

OOB access

```
[ 4407.612749] BUG: KASAN: invalid-access in __memcpy+0x194/0x250
[ 4407.612852] Write at addr f6ffff89163e2a00 by task checker/10632
[ 4407.612864] Pointer tag: [f6], memory tag: [fe]
[ 4407.612875]
[ 4407.612886] CPU: 6 PID: 10632 Comm: checker Tainted: G S  B   W  OE     5.15.110-
android14-11-gcc48824eebe8-ab10865596 #1
[ 4407.612898] Hardware name: ZUMA SHIBA MP based on ZUMA (DT)
[ 4407.612908] Call trace:
[ 4407.612917] dump_backtrace+0xf8/0x1e8
[ 4407.612970] dump_stack_lvl+0x74/0xa4
[ 4407.613025] print_report+0x344/0x958
[ 4407.613092] kasan_report+0x90/0xe4
[ 4407.613102] __do_kernel_fault+0xc4/0x2ac
[ 4407.613129] do_bad_area+0x3c/0x154
[ 4407.613138] do_tag_check_fault+0x18/0x24
[ 4407.613145] do_mem_abort+0x60/0x134
[ 4407.613159] el1_abort+0x38/0x54
[ 4407.613262] el1h_64_sync_handler+0x54/0x88
[ 4407.613269] el1h_64_sync+0x78/0x7c
[ 4407.613282] __memcpy+0x194/0x250
```

MTE- kernel

```
char *p = kmalloc(32, GFP_kernel);
kfree(p);
p[0] = ‘A’;
```

Use-After-Free(in free state)

```
char *p = kmalloc(32, GFP_kernel);
kfree(p);
char *q = kmalloc(32, GFP_kernel);
p[0] = ‘A’;
```

Use-After-Free

MTE- kernel

Use-After-Free(in free state)

```
[   95.707717] BUG: KASAN: invalid-access in __memcpy+0x110/0x250
[   95.707811] Read at addr f9ffff88c658db00 by task checker/7490
[   95.707824] Pointer tag: [f9], memory tag: [fe]
[   95.707835]
[   95.707874] CPU: 4 PID: 7490 Comm: checker Tainted: G S      W  OE     5.15.110-
android14-11-gcc48824eebe8-ab10865596 #1
[   95.707894] Hardware name: ZUMA SHIBA MP based on ZUMA (DT)
[   95.707906] Call trace:
[   95.707916] dump_backtrace+0xf8/0x1e8
[   95.707968] dump_stack_lvl+0x74/0xa4
[   95.708064] print_report+0x344/0x958
[   95.708134] kasan_report+0x90/0xe4
[   95.708141] __do_kernel_fault+0xc4/0x2ac
[   95.708167] do_bad_area+0x3c/0x154
[   95.708173] do_tag_check_fault+0x18/0x24
[   95.708180] do_mem_abort+0x60/0x134
[   95.708192] el1_abort+0x38/0x54
[   95.708297] el1h_64_sync_handler+0x54/0x88
[   95.708305] el1h_64_sync+0x78/0x7c
[   95.708316] __memcpy+0x110/0x250
```

MTE- kernel

Use-After-Free

```
[ 1233.203312] BUG: KASAN: invalid-access in __memcpy+0x110/0x250
[ 1233.203407] Read at addr f4ffff8850f82b00 by task checker/7685
[ 1233.203420] Pointer tag: [f4], memory tag: [fd]
[ 1233.203432]
[ 1233.203445] CPU: 6 PID: 7685 Comm: checker Tainted: G S      W  OE     5.15.110-
android14-11-gcc48824eebe8-ab10865596 #1
[ 1233.203460] Hardware name: ZUMA SHIBA MP based on ZUMA (DT)
[ 1233.203469] Call trace:
[ 1233.203478] dump_backtrace+0xf8/0x1e8
[ 1233.203529] dump_stack_lvl+0x74/0xa4
[ 1233.203598] print_report+0x344/0x958
[ 1233.203662] kasan_report+0x90/0xe4
[ 1233.203670] __do_kernel_fault+0xc4/0x2ac
[ 1233.203699] do_bad_area+0x3c/0x154
[ 1233.203705] do_tag_check_fault+0x18/0x24
[ 1233.203711] do_mem_abort+0x60/0x134
[ 1233.203724] el1_abort+0x38/0x54
[ 1233.203828] el1h_64_sync_handler+0x54/0x88
[ 1233.203837] el1h_64_sync+0x78/0x7c
[ 1233.203847] __memcpy+0x110/0x250
```

MTE- kernel

- Small chunks
  - kmalloc – generic cache
  - kmem_cache_alloc – dedicated cache
- Large virtually contiguous areas
  - vmalloc
  - vmap
- Physical pages
  - alloc_pages

MTE- kernel

- Small chunks (tagged)
  - kmalloc – generic cache
  - kmem_cache_alloc – dedicated cache
- Large virtually contiguous areas
  - Vmalloc (tagged)
  - Vmap (Not tagged)
- Physical pages
  - alloc_pages (tagged, native kernel pointers tag)

MTE- kernel

- CVE-2022-22706
  - Elevates CPU RO pages to writable
- CVE-2022-38181
  - Kmalloc UAF
- CVE-2022-20186/CVE-2022-36449/CVE-2023-????/CVE-2023-6241
  - Page UAF
- CVE-2022-46395
  - Vmalloc UAF
- CVE-2023-32832
  - kmem_cache_alloc UAF

MTE- kernel

- CVE-2022-22706

```
+       write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);
+
[...]
        pinned_pages = pin_user_pages_remote(
                mm, address, alloc->imported.user_buf.nr_pages,
-               reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
-               NULL);
+               write ? FOLL_WRITE : 0, pages, NULL, NULL);
[...]
```

MTE- kernel

- CVE-2022-22706
  - Elevates CPU RO pages to writable
- CVE-2022-38181
  - Kmalloc UAF
- CVE-2022-20186/CVE-2022-36449/CVE-2023-????/CVE-2023-6241
  - Page UAF
- CVE-2022-46395
  - Vmalloc UAF
- CVE-2023-32832
  - kmem_cache_alloc UAF

MTE- kernel

- CVE-2022-22706
  - Elevates CPU RO pages to writable
- CVE-2022-38181
  - Kmalloc UAF
- CVE-2022-20186/CVE-2022-36449/CVE-2023-????/CVE-2023-6241
  - Page UAF
- CVE-2022-46395
  - Vmalloc UAF
- CVE-2023-32832
  - kmem_cache_alloc UAF

CVE-2023-6241

Security

Gaining kernel code execution on an MTE-enabled Pixel 8

In this post, I'll look at CVE-2023-6241, a vulnerability in the Arm Mali GPU that allows a malicious app to gain arbitrary kernel code execution and root on an Android phone. I'll show how this vulnerability can be exploited even when Memory Tagging Extension (MTE), a powerful mitigation, is enabled on the device.

Agenda

- Introduction
- *Bug analysis and exploitation*
- Conclusion

CVE-????

- New commands are added
  - New bugs

```
        break;
    }
```

```
        break;

    case KBASE_IOCTL_KINSTR_PRFCNT_ENUM_INFO:
        KBASE_HANDLE_IOCTL_INOUT(
            KBASE_IOCTL_KINSTR_PRFCNT_ENUM_INFO,
            kbase_api_kinstr_prfcnt_enum_info,
            struct kbase_ioctl_kinstr_prfcnt_enum_info, kfile);
        break;

    case KBASE_IOCTL_KINSTR_PRFCNT_SETUP:
        KBASE_HANDLE_IOCTL_INOUT(KBASE_IOCTL_KINSTR_PRFCNT_SETUP,
                    kbase_api_kinstr_prfcnt_setup,
                    union kbase_ioctl_kinstr_prfcnt_setup,
                    kfile);
        break;
    }
```

CVE-????

```c
static int kbase_api_kinstr_prfcnt_enum_info(
    struct kbase_file *kfile,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *prfcnt_enum_info)
{
    return kbase_kinstr_prfcnt_enum_info(kfile->kbdev->kinstr_prfcnt_ctx,
                         prfcnt_enum_info);
}
```

```c
/**
 * struct kbase_ioctl_kinstr_prfcnt_enum_info - Enum Performance counter
 *                                              information
 * @info_item_size:  Performance counter item size in bytes.
 * @info_item_count: Performance counter item count in the info_list_ptr.
 * @info_list_ptr:   Performance counter item list pointer which points to a
 *                   list with info_item_count of items.
 *
 * On success: returns info_item_size and info_item_count if info_list_ptr is
 * NULL, returns performance counter information if info_list_ptr is not NULL.
 * On error: returns a negative error code.
 */
struct kbase_ioctl_kinstr_prfcnt_enum_info {
    __u32 info_item_size;
    __u32 info_item_count;
    __u64 info_list_ptr;
};
```

```c
int kbase_kinstr_prfcnt_enum_info(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    int err;

    if (!kinstr_ctx || !enum_info)
        return -EINVAL;

    if (!enum_info->info_list_ptr)
        err = kbasep_kinstr_prfcnt_enum_info_count(kinstr_ctx,
                                   enum_info);
    else
        err = kbasep_kinstr_prfcnt_enum_info_list(kinstr_ctx,
                                  enum_info);

    return err;
}
```

CVE-????

```c
static int kbase_api_kinstr_prfcnt_enum_info(
    struct kbase_file *kfile,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *prfcnt_enum_info)
{
    return kbase_kinstr_prfcnt_enum_info(kfile->kbdev->kinstr_prfcnt_ctx,
                         prfcnt_enum_info);
}
```

```c
/**
 * struct kbase_ioctl_kinstr_prfcnt_enum_info - Enum Performance counter
 *                                              information
 * @info_item_size:  Performance counter item size in bytes.
 * @info_item_count: Performance counter item count in the info_list_ptr.
 * @info_list_ptr:   Performance counter item list pointer which points to a
 *                   list with info_item_count of items.
 *
 * On success: returns info_item_size and info_item_count if info_list_ptr is
 * NULL, returns performance counter information if info_list_ptr is not NULL.
 * On error: returns a negative error code.
 */
struct kbase_ioctl_kinstr_prfcnt_enum_info {
    __u32 info_item_size;
    __u32 info_item_count;
    __u64 info_list_ptr;
};
```

```c
static int kbasep_kinstr_prfcnt_enum_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    struct prfcnt_enum_item *prfcnt_item_arr;
    size_t arr_idx = 0;
    int err = 0;
    size_t block_info_count = 0;
    const struct kbase_hwcnt_metadata *metadata;

    if ((enum_info->info_item_size == 0) ||
        (enum_info->info_item_count == 0) || !enum_info->info_list_ptr)
        return -EINVAL;

    if (enum_info->info_item_count != kinstr_ctx->info_item_count)
        return -EINVAL;

    prfcnt_item_arr =
        (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
    kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                               &arr_idx);
    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

    if (arr_idx + block_info_count >= enum_info->info_item_count)
        err = -EINVAL;

    if (!err) {
        size_t counter_set;
```

CVE-????

```
static int kbasep_kinstr_prfcnt_enum_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    struct prfcnt_enum_item *prfcnt_item_arr;
    size_t arr_idx = 0;
    int err = 0;
    size_t block_info_count = 0;
    const struct kbase_hwcnt_metadata *metadata;

    if ((enum_info->info_item_size == 0) ||
        (enum_info->info_item_count == 0) || !enum_info->info_list_ptr)
        return -EINVAL;

    if (enum_info->info_item_count != kinstr_ctx->info_item_count)
        return -EINVAL;

    prfcnt_item_arr =
        (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
    kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                                               &arr_idx);
    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

    if (arr_idx + block_info_count >= enum_info->info_item_count)
        err = -EINVAL;

    if (!err) {
        size_t counter_set;
```

Arbitrary kernel address write!

CVE-????

```
static int kbasep_kinstr_prfcnt_parse_setup(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    union kbase_ioctl_kinstr_prfcnt_setup *setup,
    struct kbase_kinstr_prfcnt_client_config *config)
{
    uint32_t i;
    struct prfcnt_request_item *req_arr;
    int err = 0;

    if (!setup->in.requests_ptr || (setup->in.request_item_count == 0) ||
        (setup->in.request_item_size == 0)) {
        return -EINVAL;
    }

    req_arr =
        (struct prfcnt_request_item *)(uintptr_t)setup->in.requests_ptr;

    if (req_arr[setup->in.request_item_count - 1].hdr.item_type !=
        FLEX_LIST_TYPE_NONE) {
        return -EINVAL;
    }

    if (req_arr[setup->in.request_item_count - 1].hdr.item_version != 0)
        return -EINVAL;

    /* The session configuration can only feature one value for some
     * properties (like capture mode and block counter set), but the client
     * may potential issue multiple requests and try to set more than one
     * value for those properties. While issuing multiple requests for the
     * same property is allowed by the protocol, asking for different values
     * is illegal. Leaving these properties as undefined is illegal, too.
     */
    config->prfcnt_mode = PRFCNT_MODE_RESERVED;
    config->counter_set = KBASE_HWCNT_SET_UNDEFINED;

    for (i = 0; i < setup->in.request_item_count - 1; i++) {
        if (req_arr[i].hdr.item_version > PRFCNT_READER_API_VERSION) {
            err = -EINVAL;
            break;
        }
```

Arbitrary kernel address parse!

```
if (enum_info->info_item_count != kinstr_ctx->info_item_count)
    return -EINVAL;


prfcnt_item_arr =
    (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;

kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                        &arr_idx);
metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);



block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

if (arr_idx + block_info_count >= enum_info->info_item_count)
    err = -EINVAL;

if (!err) {
    size_t counter_set;

defined(CONFIG_MALI_PRFCNT_SET_SECONDARY)
    counter_set = KBASE_HWCNT_SET_SECONDARY;
if defined(CONFIG_MALI_PRFCNT_SET_TERTIARY)
    counter_set = KBASE_HWCNT_SET_TERTIARY;
se
    /* Default to primary */
    counter_set = KBASE_HWCNT_SET_PRIMARY;
dif
    kbasep_kinstr_prfcnt_get_block_info_list(
        metadata, counter_set, prfcnt_item_arr, &arr_idx);
    if (arr_idx != enum_info->info_item_count - 1)
        err = -EINVAL;
}

/* The last sentinel item. */
prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_type =
    FLEX_LIST_TYPE_NONE;
prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_version = 0;









return err;
```

```
    if (enum_info->info_item_count != kinstr_ctx->info_item_count)
        return -EINVAL;

    prfcnt_item_arr = kcalloc(enum_info->info_item_count,
                  sizeof(*prfcnt_item_arr), GFP_KERNEL);
    if (!prfcnt_item_arr)
        return -ENOMEM;

    kbasep_kinstr_prfcnt_get_request_info_list(prfcnt_item_arr, &arr_idx);

    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    /* Place the sample_info item */
    kbasep_kinstr_prfcnt_get_sample_info_item(metadata, prfcnt_item_arr, &arr_idx);

    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

    if (arr_idx + block_info_count >= enum_info->info_item_count)
        err = -EINVAL;

    if (!err) {
        size_t counter_set;

#if defined(CONFIG_MALI_PRFCNT_SET_SECONDARY)
        counter_set = KBASE_HWCNT_SET_SECONDARY;
#elif defined(CONFIG_MALI_PRFCNT_SET_TERTIARY)
        counter_set = KBASE_HWCNT_SET_TERTIARY;
#else
        /* Default to primary */
        counter_set = KBASE_HWCNT_SET_PRIMARY;
#endif
        kbasep_kinstr_prfcnt_get_block_info_list(
            metadata, counter_set, prfcnt_item_arr, &arr_idx);
        if (arr_idx != enum_info->info_item_count - 1)
            err = -EINVAL;
    }

    /* The last sentinel item. */
    prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_type =
        FLEX_LIST_TYPE_NONE;
    prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_version = 0;

    if (!err) {
        unsigned long bytes =
            enum_info->info_item_count * sizeof(*prfcnt_item_arr);

        if (copy_to_user(u64_to_user_ptr(enum_info->info_list_ptr),
                 prfcnt_item_arr, bytes))
            err = -EFAULT;
    }

    kfree(prfcnt_item_arr);
    return err;
```

Fix

```
static int kbasep_kinstr_prfcnt_parse_setup(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    union kbase_ioctl_kinstr_prfcnt_setup *setup,
    struct kbase_kinstr_prfcnt_client_config *config)
{
    uint32_t i;
    struct prfcnt_request_item *req_arr;


    int err = 0;




    if (!setup->in.requests_ptr || (setup->in.request_item_count == 0) ||
        (setup->in.request_item_size == 0)) {

        return -EINVAL;
    }


    req_arr =
        (struct prfcnt_request_item *)(uintptr_t)setup->in.requests_ptr;

    if (req_arr[setup->in.request_item_count - 1].hdr.item_type !=
        FLEX_LIST_TYPE_NONE) {
        return -EINVAL;

    }


    if (req_arr[setup->in.request_item_count - 1].hdr.item_version != 0)
        return -EINVAL;
```

```
static int kbasep_kinstr_prfcnt_parse_setup(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    union kbase_ioctl_kinstr_prfcnt_setup *setup,
    struct kbase_kinstr_prfcnt_client_config *config)
{
    uint32_t i;
    struct prfcnt_request_item *req_arr;
    unsigned int item_count = setup->in.request_item_count;
    unsigned long bytes;
    int err = 0;

    /* Limiting the request items to 2x of the expected: acommodating
     * moderate duplications but rejecting excessive abuses.
     */
    if (!setup->in.requests_ptr || (item_count < 2) ||
        (setup->in.request_item_size == 0) ||
        item_count > 2 * kinstr_ctx->info_item_count) {
        return -EINVAL;
    }

    bytes = item_count * sizeof(*req_arr);
    req_arr = kmalloc(bytes, GFP_KERNEL);
    if (!req_arr)
        return -ENOMEM;

    if (copy_from_user(req_arr, u64_to_user_ptr(setup->in.requests_ptr),
                bytes)) {
        err = -EFAULT;
        goto free_buf;
    }

    if (req_arr[item_count - 1].hdr.item_type != FLEX_LIST_TYPE_NONE ||
        req_arr[item_count - 1].hdr.item_version != 0) {
        err = -EINVAL;
        goto free_buf;
    }
```

PoC

```
static int kbasep_kinstr_prfcnt_enum_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    struct prfcnt_enum_item *prfcnt_item_arr;
    size_t arr_idx = 0;
    int err = 0;
    size_t block_info_count = 0;
    const struct kbase_hwcnt_metadata *metadata;

    if ((enum_info->info_item_size == 0) ||
        (enum_info->info_item_count == 0) || !enum_info->info_list_ptr)
        return -EINVAL;

    if (enum_info->info_item_count != kinstr_ctx->info_item_count)
        return -EINVAL;

    prfcnt_item_arr =
        (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
    kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                           &arr_idx);
    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

    if (arr_idx + block_info_count >= enum_info->info_item_count)
        err = -EINVAL;

    if (!err) {
        size_t counter_set;
```

```
int kbase_kinstr_prfcnt_enum_info(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    int err;

    if (!kinstr_ctx || !enum_info)
        return -EINVAL;

    if (!enum_info->info_list_ptr)
        err = kbasep_kinstr_prfcnt_enum_info_count(kinstr_ctx,
                                   enum_info);
    else
        err = kbasep_kinstr_prfcnt_enum_info_list(kinstr_ctx,
                                  enum_info);

    return err;
}
```

PoC

```
static int kbasep_kinstr_prfcnt_enum_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    struct prfcnt_enum_item *prfcnt_item_arr;
    size_t arr_idx = 0;
    int err = 0;
    size_t block_info_count = 0;
    const struct kbase_hwcnt_metadata *metadata;

    if ((enum_info->info_item_size == 0) ||
        (enum_info->info_item_count == 0) || !enum_info->info_list_ptr)
        return -EINVAL;

    if (enum_info->info_item_count != kinstr_ctx->info_item_count)
        return -EINVAL;

    prfcnt_item_arr =
        (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
    kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                                               &arr_idx);
    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

    if (arr_idx + block_info_count >= enum_info->info_item_count)
        err = -EINVAL;

    if (!err) {
        size_t counter_set;
```

```
static int kbasep_kinstr_prfcnt_enum_info_count(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    int err = 0;
    uint32_t count = 0;
    size_t block_info_count = 0;
    const struct kbase_hwcnt_metadata *metadata;

    count = ARRAY_SIZE(kinstr_prfcnt_supported_requests);
    metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
    block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);
    count += block_info_count;

    /* Reserve one for the last sentinel item. */
    count++;
    enum_info->info_item_count = count;
    enum_info->info_item_size = sizeof(struct prfcnt_enum_item);
    kinstr_ctx->info_item_count = count;

    return err;
}
```

PoC

```
[pid:12999,cpu4,PoC2023,3]Unable to handle kernel paging request at virtual address 0000000041414151
[pid:12999,cpu4,PoC2023,4]Mem abort info:
[pid:12999,cpu4,PoC2023,5]  ESR = 0x96000045
[pid:12999,cpu4,PoC2023,6]  EC = 0x25: DABT (current EL), IL = 32 bits
[pid:12999,cpu4,PoC2023,7]  SET = 0, FnV = 0
[pid:12999,cpu4,PoC2023,8]  EA = 0, S1PTW = 0
[pid:12999,cpu4,PoC2023,9]Data abort info:
[pid:12999,cpu4,PoC2023,0]  ISV = 0, ISS = 0x00000045
[pid:12999,cpu4,PoC2023,1]  CM = 0, WnR = 1
[pid:12999,cpu4,PoC2023,2]user pgtable: 4k pages, 39-bit VAs, pgdp=0000000085646000
[pid:12999,cpu4,PoC2023,3][0000000041414151] pgd=0000000000000000, p4d=0000000000000000, pud=0000000000000000
[pid:12999,cpu4,PoC2023,4]Internal error: Oops: 96000045 [#1] PREEMPT SMP
[pid:12999,cpu4,PoC2023,5]Modules linked in: [redacted]
[pid:12999,cpu4,PoC2023,6]CPU: 4 PID: 12999 Comm: PoC2023 VIP: 00 Tainted: G        W         5.10.43 #1
[pid:12999,cpu4,PoC2023,7]TGID: 12999 Comm: PoC2023
[pid:12999,cpu4,PoC2023,8][redacted]
[pid:12999,cpu4,PoC2023,9]pstate: 60400005 (nZCv daif +PAN -UAO -TCO BTYPE=--)
[pid:12999,cpu4,PoC2023,0]pc : kbase_kinstr_prfcnt_enum_info+0x44/0x350
[pid:12999,cpu4,PoC2023,1]lr : kbase_ioctl+0x494/0x3430
[pid:12999,cpu4,PoC2023,2]pc : [<ffffffe946d669ec>] lr : [<ffffffe946d83634>] pstate: 60400005
[pid:12999,cpu4,PoC2023,3]sp : ffffff8110b97cd0
[pid:12999,cpu4,PoC2023,4]x29: ffffff8110b97da0 x28: ffffff816e4b1180
[pid:12999,cpu4,PoC2023,5]x27: 0000000000000000 x26: ffffffe94829f21c
[pid:12999,cpu4,PoC2023,6]x25: 0000000000000000 x24: ffffff8110b97cf0
[pid:12999,cpu4,PoC2023,7]x23: 0000007fce5131c8 x22: ffffff816e4b1180
[pid:12999,cpu4,PoC2023,8]x21: ffffff81b6429500 x20: 00000000c0108038
[pid:12999,cpu4,PoC2023,9]x19: 0000007fce5131c8 x18: 0000000000000000
[pid:12999,cpu4,PoC2023,0]x17: 0000000000000000 x16: 0000000000000000
[pid:12999,cpu4,PoC2023,1]x15: 0000000000000000 x14: 0000000000000000
[pid:12999,cpu4,PoC2023,2]x13: 0000000100000000 x12: 0000000000000001
[pid:12999,cpu4,PoC2023,3]x11: 0000000000000000 x10: 0000000000000000
[pid:12999,cpu4,PoC2023,4]x9 : 0000000041414141 x8 : ffffffe948cc6360
[pid:12999,cpu4,PoC2023,5]x7 : ffffff8110b97d90 x6 : ffffff8110b97d00
[pid:12999,cpu4,PoC2023,6]x5 : ffffff8110b97d00 x4 : 0000000000000008
[pid:12999,cpu4,PoC2023,7]x3 : 0000000041414141 x2 : 0000000000000008
[pid:12999,cpu4,PoC2023,8]x1 : ffffff8110b97cf0 x0 : ffffff800951f100
```

MTE- kernel

```
// ffffff8850f82b00 --> f1ffff8850f82b00
#define KASAN_TAG_KERNEL    0xFF /* native kernel pointers tag */
#define KASAN_TAG_INVALID   0xFE /* inaccessible memory tag */
#define KASAN_TAG_MAX       0xFD /* maximum value for random tags */

#ifdef CONFIG_KASAN_HW_TAGS
#define KASAN_TAG_MIN       0xF0 /* minimum value for random tags */
#else
#define KASAN_TAG_MIN       0x00 /* minimum value for random tags */
#endif

static inline u8 mte_get_ptr_tag(void *ptr)
{
    /* Note: The format of KASAN tags is 0xF<x> */
    u8 tag = 0xF0 | (u8)(((u64)(ptr)) >> MTE_TAG_SHIFT);

    return tag;
}
```

Exploitation

```
  prfcnt_item_arr =
      (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
  kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                                             &arr_idx);
  metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
  block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

  if (arr_idx + block_info_count >= enum_info->info_item_count)
      err = -EINVAL;

  if (!err) {
      size_t counter_set;

f defined(CONFIG_MALI_PRFCNT_SET_SECONDARY)
      counter_set = KBASE_HWCNT_SET_SECONDARY;
lif defined(CONFIG_MALI_PRFCNT_SET_TERTIARY)
      counter_set = KBASE_HWCNT_SET_TERTIARY;
lse
      /* Default to primary */
      counter_set = KBASE_HWCNT_SET_PRIMARY;
ndif
      kbasep_kinstr_prfcnt_get_block_info_list(
          metadata, counter_set, prfcnt_item_arr, &arr_idx);
      if (arr_idx != enum_info->info_item_count - 1)
          err = -EINVAL;
  }

  /* The last sentinel item. */
  prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_type =
      FLEX_LIST_TYPE_NONE;
  prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_version = 0;
```

Exploitation

```
 prfcnt_item_arr =
     (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
 kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                                            &arr_idx);
 metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
 block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

 if (arr_idx + block_info_count >= enum_info->info_item_count)
     err = -EINVAL;

 if (!err) {
     size_t counter_set;

f defined(CONFIG_MALI_PRFCNT_SET_SECONDARY)
     counter_set = KBASE_HWCNT_SET_SECONDARY;
lif defined(CONFIG_MALI_PRFCNT_SET_TERTIARY)
     counter_set = KBASE_HWCNT_SET_TERTIARY;
lse
     /* Default to primary */
     counter_set = KBASE_HWCNT_SET_PRIMARY;
ndif
     kbasep_kinstr_prfcnt_get_block_info_list(
         metadata, counter_set, prfcnt_item_arr, &arr_idx);
     if (arr_idx != enum_info->info_item_count - 1)
         err = -EINVAL;
 }

 /* The last sentinel item. */
 prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_type =
     FLEX_LIST_TYPE_NONE;
 prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_version = 0;
```

```
static void kbasep_kinstr_prfcnt_get_request_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct prfcnt_enum_item *item_arr, size_t *arr_idx)
{
    memcpy(&item_arr[*arr_idx], kinstr_prfcnt_supported_requests,
           sizeof(kinstr_prfcnt_supported_requests));
    *arr_idx += ARRAY_SIZE(kinstr_prfcnt_supported_requests);
}
```

```
static struct prfcnt_enum_item kinstr_prfcnt_supported_requests[] = {
    {
        /* Request description for MODE request */
        .hdr = {
                .item_type = PRFCNT_ENUM_TYPE_REQUEST,
                .item_version = PRFCNT_READER_API_VERSION,
        },
        .u.request = {
                .request_item_type = PRFCNT_REQUEST_MODE,
                .versions_mask = 0x1,
        },
    },
    {
        /* Request description for ENABLE request */
        .hdr = {
                .item_type = PRFCNT_ENUM_TYPE_REQUEST,
                .item_version = PRFCNT_READER_API_VERSION,
        },
        .u.request = {
                .request_item_type = PRFCNT_REQUEST_ENABLE,
                .versions_mask = 0x1,
        },
    },
};
```

Exploitation

```
  prfcnt_item_arr =
      (struct prfcnt_enum_item *)(uintptr_t)enum_info->info_list_ptr;
  kbasep_kinstr_prfcnt_get_request_info_list(kinstr_ctx, prfcnt_item_arr,
                         &arr_idx);
  metadata = kbase_hwcnt_virtualizer_metadata(kinstr_ctx->hvirt);
  block_info_count = kbasep_kinstr_prfcnt_get_block_info_count(metadata);

  if (arr_idx + block_info_count >= enum_info->info_item_count)
      err = -EINVAL;

  if (!err) {
      size_t counter_set;

f defined(CONFIG_MALI_PRFCNT_SET_SECONDARY)
      counter_set = KBASE_HWCNT_SET_SECONDARY;
lif defined(CONFIG_MALI_PRFCNT_SET_TERTIARY)
      counter_set = KBASE_HWCNT_SET_TERTIARY;
lse
      /* Default to primary */
      counter_set = KBASE_HWCNT_SET_PRIMARY;
ndif
      kbasep_kinstr_prfcnt_get_block_info_list(
          metadata, counter_set, prfcnt_item_arr, &arr_idx);
      if (arr_idx != enum_info->info_item_count - 1)
          err = -EINVAL;
  }

  /* The last sentinel item. */
  prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_type =
      FLEX_LIST_TYPE_NONE;
  prfcnt_item_arr[enum_info->info_item_count - 1].hdr.item_version = 0;
```

```
static int kbasep_kinstr_prfcnt_get_block_info_list(
    const struct kbase_hwcnt_metadata *metadata, size_t block_set,
    struct prfcnt_enum_item *item_arr, size_t *arr_idx)
{
    size_t grp;
    size_t blk;

    if (!metadata || !item_arr || !arr_idx)
        return -EINVAL;

    for (grp = 0; grp < kbase_hwcnt_metadata_group_count(metadata); grp++) {
        for (blk = 0;
             blk < kbase_hwcnt_metadata_block_count(metadata, grp);
             blk++, (*arr_idx)++) {
            item_arr[*arr_idx].hdr.item_type =
                PRFCNT_ENUM_TYPE_BLOCK;
            item_arr[*arr_idx].hdr.item_version =
                PRFCNT_READER_API_VERSION;
            item_arr[*arr_idx].u.block_counter.set = block_set;

            item_arr[*arr_idx].u.block_counter.block_type =
                kbase_hwcnt_metadata_block_type_to_prfcnt_block_type(
                    kbase_hwcnt_metadata_block_type(
                        metadata, grp, blk));
            item_arr[*arr_idx].u.block_counter.num_instances =
                kbase_hwcnt_metadata_block_instance_count(
                    metadata, grp, blk);
            item_arr[*arr_idx].u.block_counter.num_values =
                kbase_hwcnt_metadata_block_values_count(
                    metadata, grp, blk);

            /* The bitmask of available counters should be dynamic.
             * Temporarily, it is set to U64_MAX, waiting for the
             * required functionality to be available in the future.
             */
            item_arr[*arr_idx].u.block_counter.counter_mask[0] =
                U64_MAX;
            item_arr[*arr_idx].u.block_counter.counter_mask[1] =
                U64_MAX;
        }
    }

    return 0;
}
```

Exploitation

```
struct prfcnt_enum_item {
  struct prfcnt_item_header hdr;
  union {
     struct prfcnt_enum_block_counter
block_counter;
     struct prfcnt_enum_request request;
   } u;
}; // 32 bytes

struct prfcnt_item_header {
  __u16 item_type;
  __u16 item_version;
};
```

```
struct prfcnt_enum_block_counter {
  __u8 block_type;
  __u8 set;
  __u8 num_instances;
  __u8 num_values;
  __u8 pad[4];
  __u64 counter_mask[2];
};

struct prfcnt_enum_request {
  __u16 request_item_type;
  __u16 pad;
  __u32 versions_mask;
};
```

Exploitation

- kbasep_kinstr_prfcnt_get_request_info_list

```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- kbasep_kinstr_prfcnt_get_block_info_list

```
0000: 00 00 00 00 00 00 00 00 ?? ?? ?? ?? 00 00 00 00
0016: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- The last sentinel item

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Exploitation

- kbasep_kinstr_prfcnt_get_request_info_list

```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- kbasep_kinstr_prfcnt_get_block_info_list

```
0000: 00 00 00 00 00 00 00 00 ?? ?? ?? ?? 00 00 00 00
0016: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- The last sentinel item

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- info_item_count is 7
  - 196(32 * 6 + 4) bytes

Exploitation

- Good news
  - Arbitrary kernel address write
- Bad news
  - Length is fixed
  - Content is almost fixed

Exploitation

- Good news
  - Arbitrary kernel address write
- Bad news
  - Length is fixed
  - Content is almost fixed
- Where to overwrite?🤔
  - Brute force(KASLR) ❌
  - Kernel information disclosure bug ✅

Exploitation

- Good news
  - Arbitrary kernel address write
- Bad news
  - Length is fixed
  - Content is almost fixed
- Where to overwrite?🤔
  - Brute force(KASLR) ❌
  - Kernel information disclosure bug ✅
- Use only one bug to exploit
  - Guess or predict the addresses of kernel objects

Exploitation

- kmalloc/kmem_cache_alloc
  - Small object(size < 1 page) - fast
  - Physically Contiguous pages
  - Linear mapping(PAGE_OFFSET)

- vmalloc
  - Large object - slow
  - Physically Non-contiguous pages
  - [VMALLOC_START, VMALLOC_END]

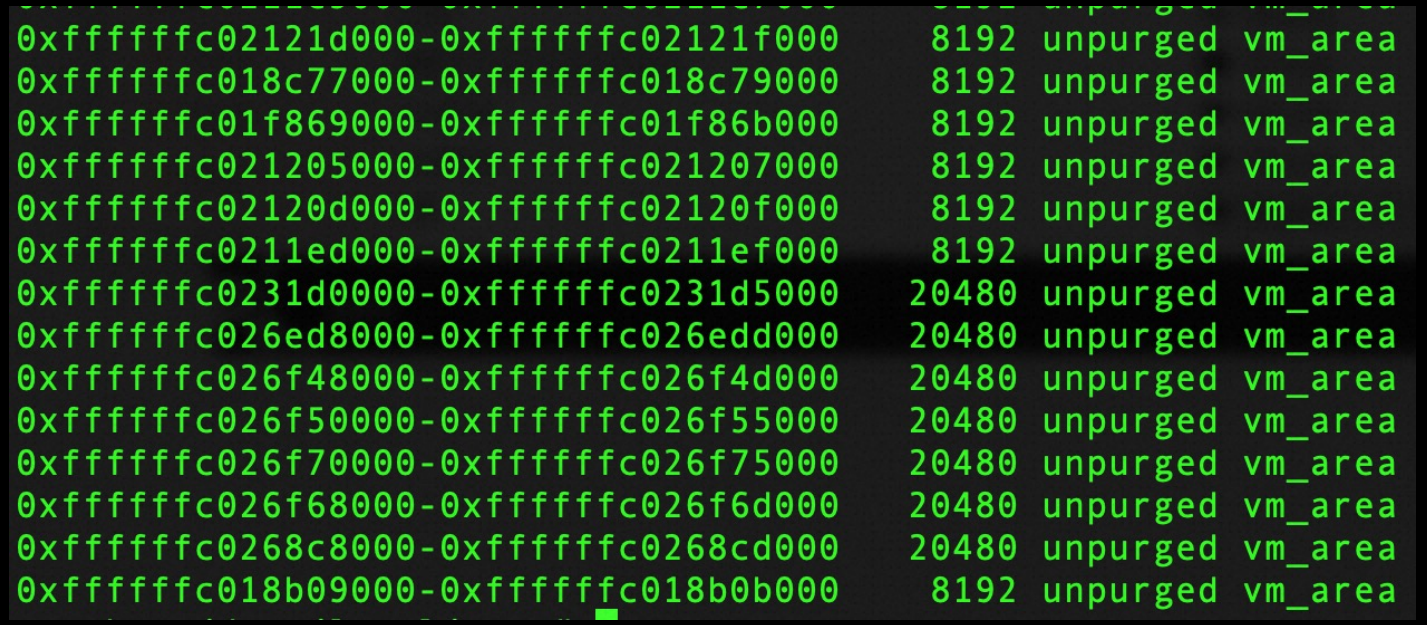
Exploitation

- kmalloc/kmem_cache_alloc
  - Small object(size < 1 page) - fast
  - Physically Contiguous pages
  - Linear mapping(PAGE_OFFSET)

- vmalloc ✅
  - Large object - slow
  - Physically Non-contiguous pages
  - [VMALLOC_START, VMALLOC_END]

VMALLOC

- Allocate
  - Find the first free virtual memory area
- Free
  - Mark the area as “unpurged”
- Purge
  - Reclaim the unpurged area



VMALLOC

- Allocate
  - Find the first free virtual memory area
- Free
  - Mark the area as “unpurged”
- Purge
  - Reclaim the unpurged area

```
0xffffffebfde0000-0xffffffebfee8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfee8000-0xffffffebfff0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffc019c45000-0xffffffc019c47000    8192 unpurged vm_area
0xffffffc0234f8000-0xffffffc0234fd000   20480 unpurged vm_area
0xffffffc01adf7000-0xffffffc01adfd000   24576 unpurged vm_area
0xffffffc01adff000-0xffffffc01ae05000   24576 unpurged vm_area
0xffffffc01ade7000-0xffffffc01aded000   24576 unpurged vm_area
0xffffffc01ac0f000-0xffffffc01ac15000   24576 unpurged vm_area
0xffffffc01adc5000-0xffffffc01adcb000   24576 unpurged vm_area
```

Exploitation

- Hardcode the addresses of kernel objects
  - Early allocated objects during booting

```
struct foo_t {
    void *a;
    void *b;
};

void init_xx() {
    foo_t *f = kmalloc(sizeof(foo_t), GFP_KERNEL);
    f->a = vmalloc(0x8000);
    f->b = vmalloc(0x4000);
    // ...
}
```

Exploitation

- Hardcode the addresses of kernel objects
  - Early allocated objects during booting

```
panther:/data/local/tmp # cat /proc/vmallocinfo | more
0xffffffc008000000-0xffffffc008005000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008005000-0xffffffc008007000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008007000-0xffffffc008008000    4096 debug_snapshot_debug_kinfo_probe+0x198/0x2d8 [debug_sn
apshot_debug_kinfo] vmap
0xffffffc008008000-0xffffffc00800d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00800d000-0xffffffc00800f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008010000-0xffffffc008015000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008015000-0xffffffc008017000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008018000-0xffffffc00801d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00801d000-0xffffffc00801f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008020000-0xffffffc008025000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008025000-0xffffffc008027000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008028000-0xffffffc00802d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00802d000-0xffffffc00802f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008030000-0xffffffc008035000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008035000-0xffffffc008037000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008038000-0xffffffc00803d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00803d000-0xffffffc00803f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008040000-0xffffffc008051000   69632 gic_of_init.31948+0x44/0x254 phys=0x000
oremap
```

```
panther:/ # reboot
thomasking@test-2 ~ % adb shell
panther:/ $ su
panther:/ # echo "1">/proc/sys/kernel/kptr_restrict
panther:/ # cat /proc/vmallocinfo |more
0xffffffc008000000-0xffffffc008005000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008005000-0xffffffc008007000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008007000-0xffffffc008008000    4096 debug_snapshot_debug_kinfo_probe+0x198/0x2d8 [debug_sn
apshot_debug_kinfo] vmap
0xffffffc008008000-0xffffffc00800d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00800d000-0xffffffc00800f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008010000-0xffffffc008015000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008015000-0xffffffc008017000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008018000-0xffffffc00801d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00801d000-0xffffffc00801f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008020000-0xffffffc008025000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008025000-0xffffffc008027000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008028000-0xffffffc00802d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00802d000-0xffffffc00802f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008030000-0xffffffc008035000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc008035000-0xffffffc008037000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008038000-0xffffffc00803d000   20480 start_kernel+0x21c/0x5f8 pages=4 vmalloc
0xffffffc00803d000-0xffffffc00803f000    8192 scs_alloc+0x18/0xc8 pages=1 vmalloc
0xffffffc008040000-0xffffffc008051000   69632 gic_of_init.31948+0x44/0x254 phys=0x0000000010400000 i
oremap
```

Exploitation

- Hardcode the addresses of kernel objects
  - Early allocated objects during booting

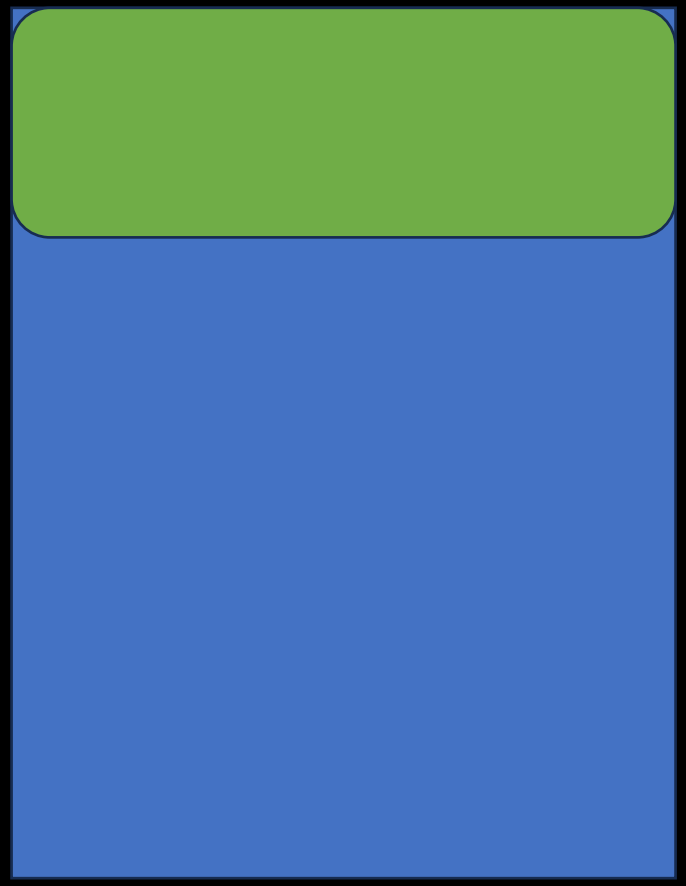
```
struct foo_t {
    void *a;
    void *b;
};

void init_xx() {
    foo_t *f = kmalloc(sizeof(foo_t), GFP_KERNEL);
    f->a = vmalloc(0x8000);
    f->b = vmalloc(0x4000);
    // ...
}
```

  - No user handle associated with those objects

Predict the address

VMALLOC_START



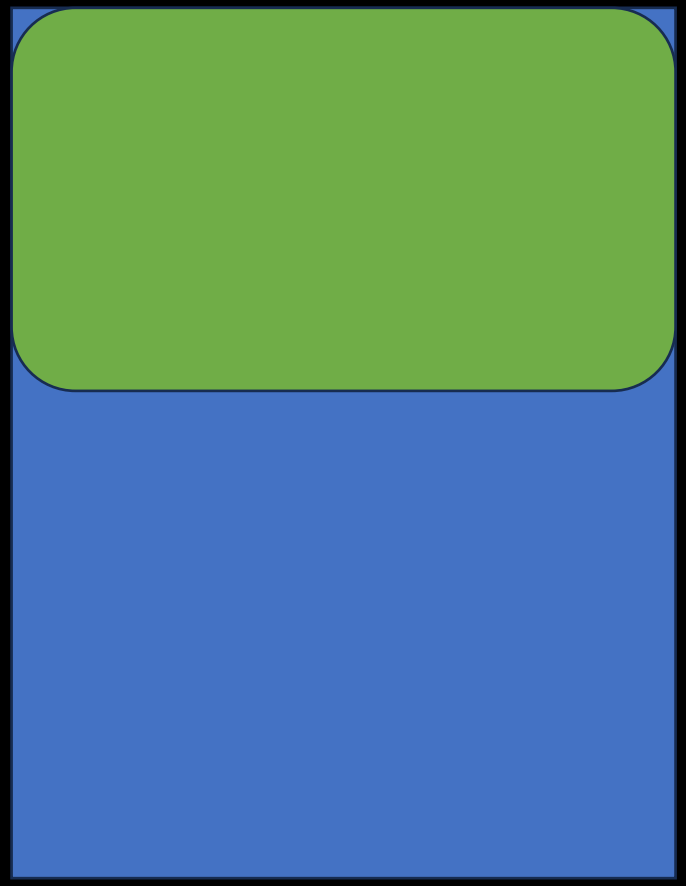
VMALLOC_END

Music

0xffffffe51f284000-0xffffffe51f289000 20480 load_module+0x1ecc/0x2554 pages=4 vmalloc
0xffffffe51f289000-0xffffffe51f75d000 5062656 load_module+0x1ecc/0x2554 pages=1235 vmalloc vpages
0xffffffebf498000-0xffffffebf5a0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf5a0000-0xffffffebf6a8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf6a8000-0xffffffebf7b0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf7b0000-0xffffffebf8b8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf8b8000-0xffffffebf9c0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf9c0000-0xffffffebfac8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfac8000-0xffffffebfbd0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfbd0000-0xffffffebfcd8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfcd8000-0xffffffebfde0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfde0000-0xffffffebfee8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfee8000-0xffffffebfff0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffc021c9b000-0xffffffc021c9d000 8192 unpurged vm_area
0xffffffc021197000-0xffffffc021199000 8192 unpurged vm_area
0xffffffc027a18000-0xffffffc027a1d000 20480 unpurged vm_area
0xffffffc021be9000-0xffffffc021beb000 8192 unpurged vm_area
0xffffffc021beb000-0xffffffc021bed000 8192 unpurged vm_area
0xffffffc02122d000-0xffffffc02122f000 8192 unpurged vm_area

Predict the address

VMALLOC_START

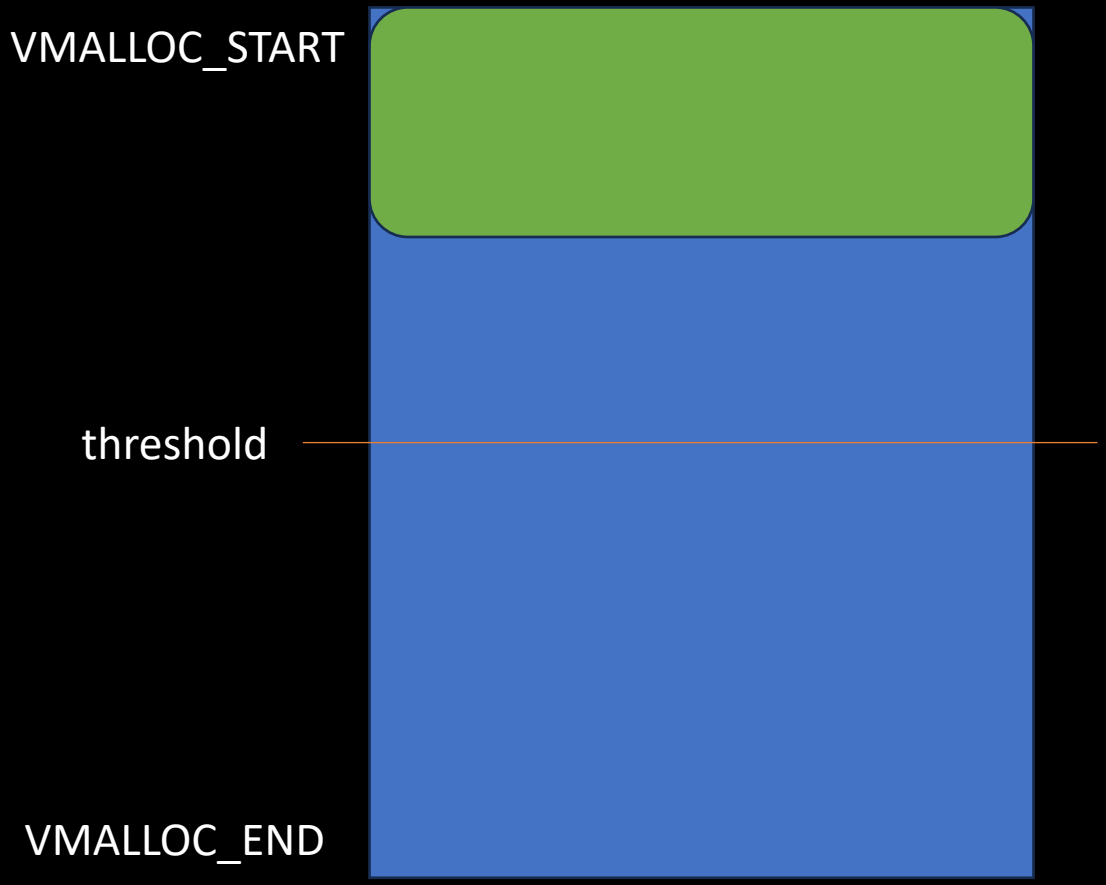


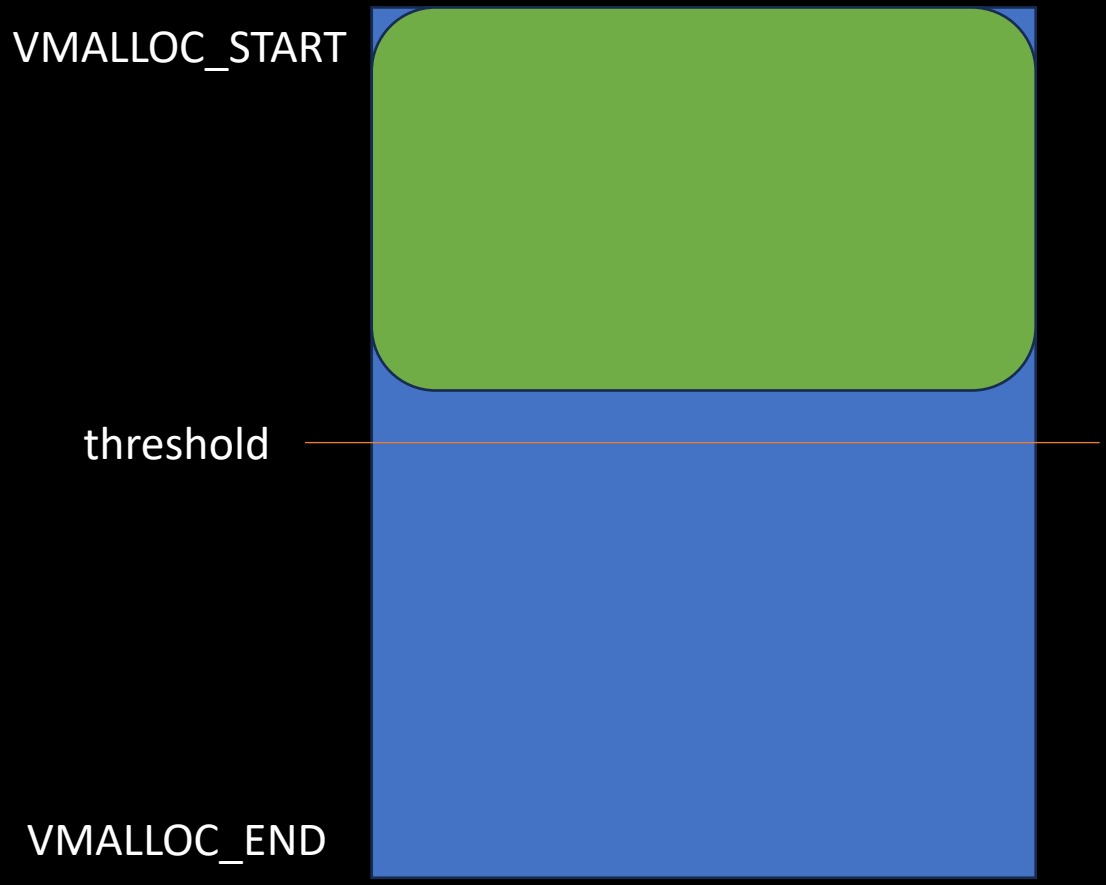
VMALLOC_END

Chrome

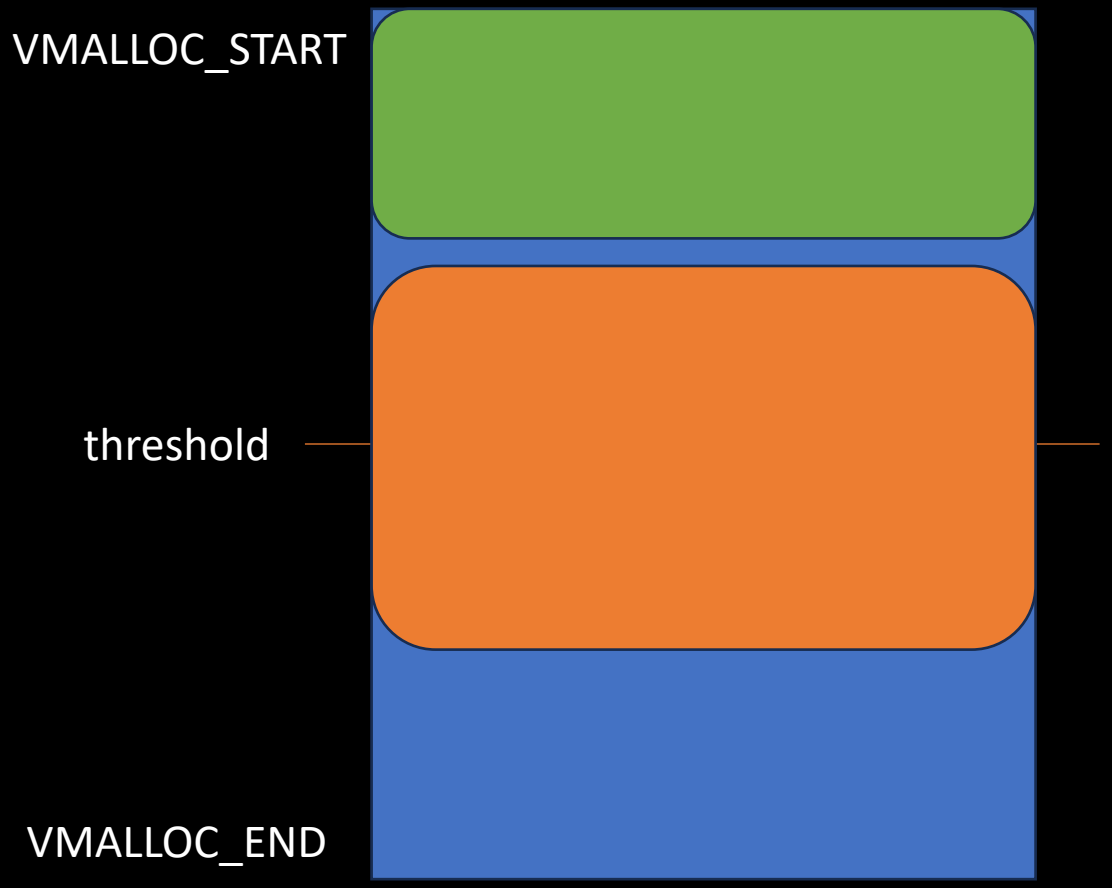
0xffffffe51f284000-0xffffffe51f289000 20480 load_module+0x1ecc/0x2554 pages=4 vmalloc
0xffffffe51f289000-0xffffffe51f75d000 5062656 load_module+0x1ecc/0x2554 pages=1235 vmalloc vpages
0xffffffebf498000-0xffffffebf5a0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf5a0000-0xffffffebf6a8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf6a8000-0xffffffebf7b0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf7b0000-0xffffffebf8b8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf8b8000-0xffffffebf9c0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf9c0000-0xffffffebfac8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfac8000-0xffffffebfbd0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfbd0000-0xffffffebfcd8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfcd8000-0xffffffebfde0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfde0000-0xffffffebfee8000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebfee8000-0xffffffebfff0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffc024e45000-0xffffffc024e47000 8192 unpurged vm_area
0xffffffc024e3d000-0xffffffc024e3f000 8192 unpurged vm_area
0xffffffc0254af000-0xffffffc0254b5000 24576 unpurged vm_area
0xffffffc024d8b000-0xffffffc024d8d000 8192 unpurged vm_area
0xffffffc025f30000-0xffffffc025f35000 20480 unpurged vm_area
0xffffffc024e35000-0xffffffc024e37000 8192 unpurged vm_area
0xffffffc024e2d000-0xffffffc024e2f000 8192 unpurged vm_area
0xffffffc023fad000-0xffffffc023faf000 8192 unpurged vm_area
0xffffffc024e1d000-0xffffffc024e1f000 8192 unpurged vm_area
0xffffffc027f90000-0xffffffc027f95000 20480 unpurged vm_area
…

Predict the address

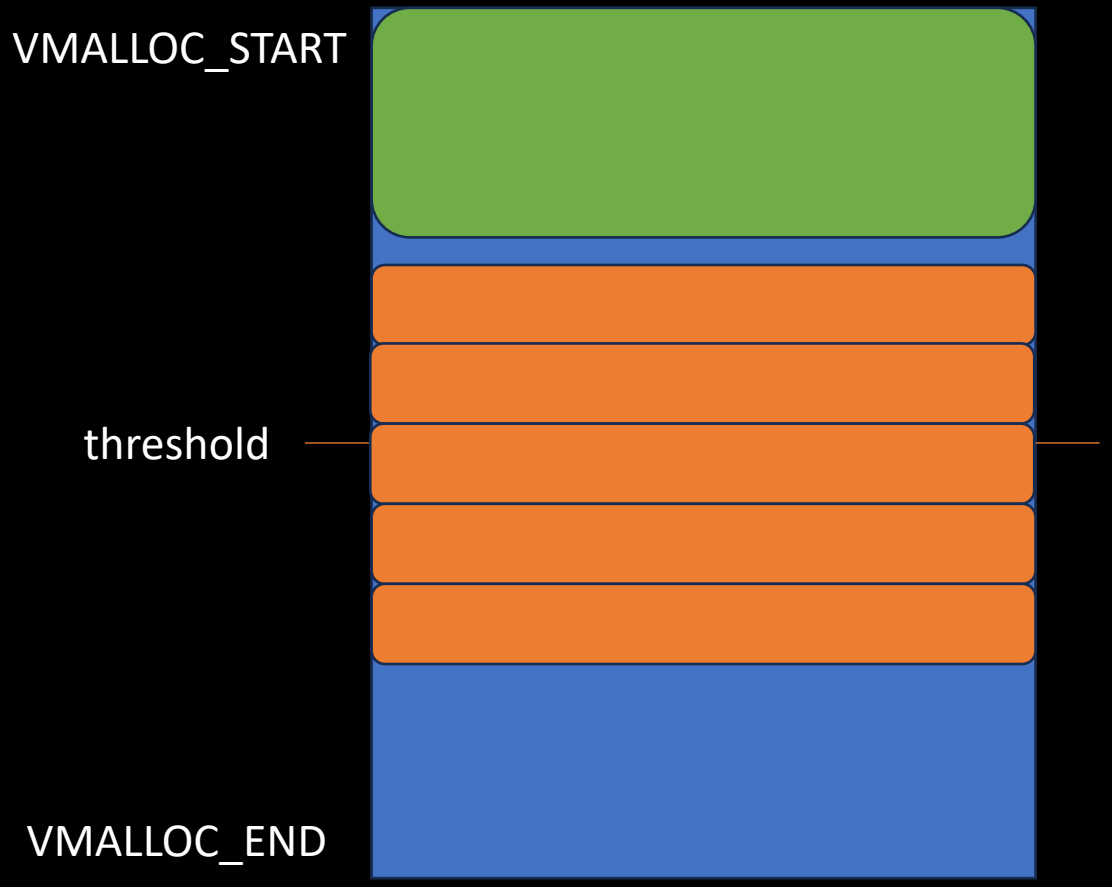




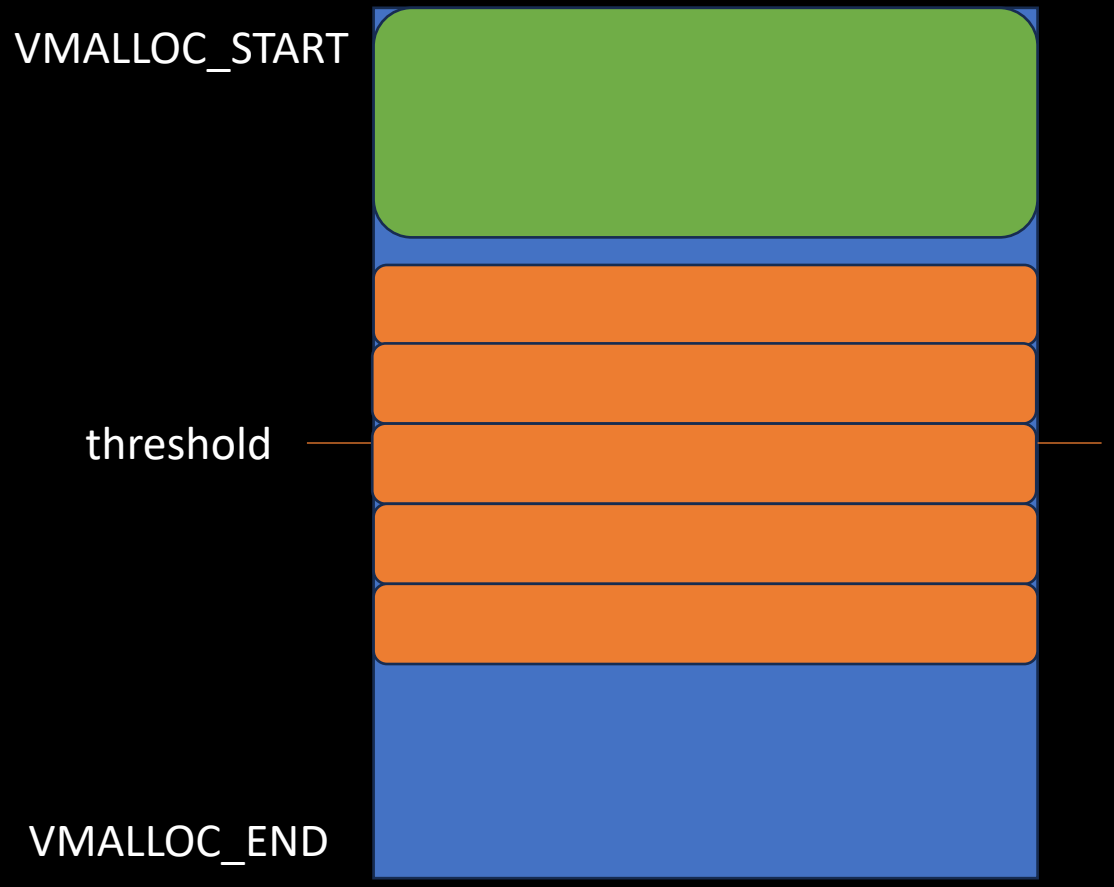
Predict the address



Predict the address



Predict the address



Is it possible to spray enough vm_areas at a low cost? 🤔

Virtual Address Management

```
struct kbase_va_region {
        struct rb_node rblink;
        struct list_head link;
        u64 start_pfn; // virtual address
        size_t nr_pages;
        size_t initial_commit;
        unsigned long flags; // {KBASE_REG_CPU_WR, KBASE_REG_FREE, …}
        struct kbase_mem_phy_alloc *cpu_alloc;
        struct kbase_mem_phy_alloc *gpu_alloc;
        struct list_head jit_node;
        u16 jit_usage_id;
        u8 jit_bin_id;
        int va_refcnt;
```

Virtual Address Management

```
struct kbase_mem_phy_alloc {
        struct kref         kref;
        atomic_t            gpu_mappings;
        atomic_t            kernel_mappings;
        size_t              nents;
        struct tagged_addr  *pages;
        struct list_head    mappings;
        struct list_head    evict_node;
        size_t              evicted;
        struct kbase_va_region *reg;
        enum kbase_memory_type type;
        struct kbase_vmap_struct *permanent_map;
        u8 properties;
        u8 group_id;
        union {umm, alias, native, user_buf} imported;
```

Predict the address

```
#define KBASE_MEM_PHY_ALLOC_LARGE_THRESHOLD ((size_t)(4*1024)) /* size above whi

static inline struct kbase_mem_phy_alloc *kbase_alloc_create(
        struct kbase_context *kctx, size_t nr_pages,
        enum kbase_memory_type type, int group_id)
{
    struct kbase_mem_phy_alloc *alloc;
    size_t alloc_size = sizeof(*alloc) + sizeof(*alloc->pages) * nr_pages;
    size_t per_page_size = sizeof(*alloc->pages);

    /* Imported pages may have page private data already in use */
    if (type == KBASE_MEM_TYPE_IMPORTED_USER_BUF) {
        alloc_size += nr_pages *
                sizeof(*alloc->imported.user_buf.dma_addrs);
        per_page_size += sizeof(*alloc->imported.user_buf.dma_addrs);
    }

    /*
     * Prevent nr_pages*per_page_size + sizeof(*alloc) from
     * wrapping around.
     */
    if (nr_pages > ((((size_t) -1) - sizeof(*alloc))
            / per_page_size))
        return ERR_PTR(-ENOMEM);

    /* Allocate based on the size to reduce internal fragmentation of vmem */
    if (alloc_size > KBASE_MEM_PHY_ALLOC_LARGE_THRESHOLD)
        alloc = vzalloc(alloc_size);
    else
        alloc = kzalloc(alloc_size, GFP_KERNEL);

    if (!alloc)
        return ERR_PTR(-ENOMEM);
```

Predict the address

```
union kbase_ioctl_mem_alloc {
        struct {
                __u64 va_pages; // virtual address
                __u64 commit_pages; // physical address
                __u64 extension;
                __u64 flags;
        } in;
        struct {
                __u64 flags;
                __u64 gpu_va;
        } out;
};
```

Predict the address

- Boot #1

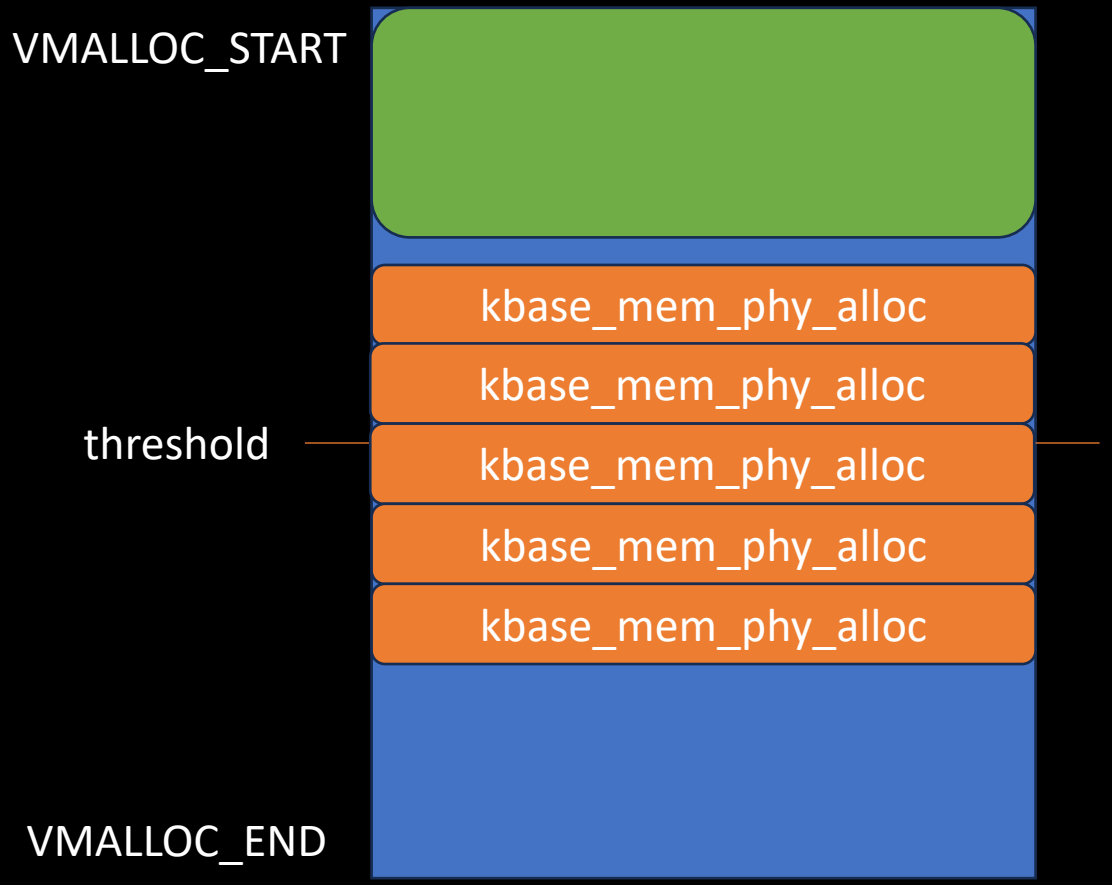
0xffffffc03c80e000-0xffffffc03d80f000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03d80f000-0xffffffc03e810000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03e810000-0xffffffc03f811000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03f811000-0xffffffc040812000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc040812000-0xffffffc041813000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc041813000-0xffffffc042814000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc042814000-0xffffffc043815000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc043815000-0xffffffc044816000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc044816000-0xffffffc045817000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc045817000-0xffffffc046818000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc046818000-0xffffffc047819000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc047819000-0xffffffc04881a000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc04881a000-0xffffffc04981b000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages

Predict the address

- Boot #n

0xffffffc03cc0f000-0xffffffc03dc10000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03dc10000-0xffffffc03ec11000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03ec11000-0xffffffc03fc12000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03fc12000-0xffffffc040c13000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc040c13000-0xffffffc041c14000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc041c14000-0xffffffc042c15000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc042c15000-0xffffffc043c16000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc043c16000-0xffffffc044c17000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc044c17000-0xffffffc045c18000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc045c18000-0xffffffc046c19000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc046c19000-0xffffffc047c1a000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc047c1a000-0xffffffc048c1b000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc048c1b000-0xffffffc049c1c000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages

Predict the address



How to find the start address? 🤔

Search the start address

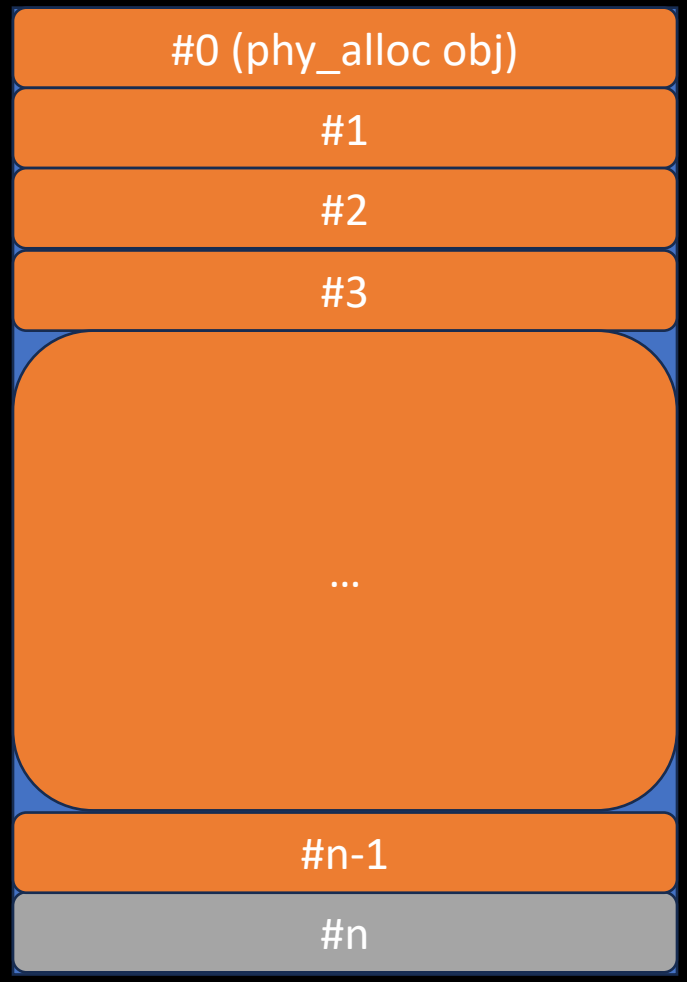
```
    switch (query) {
    case KBASE_MEM_QUERY_COMMIT_SIZE:
        if (reg->cpu_alloc->type != KBASE_MEM_TYPE_ALIAS) {
            *out = kbase_reg_current_backed_size(reg);
        } else {
            size_t i;
            struct kbase_aliased *aliased;
            *out = 0;
            aliased = reg->cpu_alloc->imported.alias.aliased;
            for (i = 0; i < reg->cpu_alloc->imported.alias.nents; i++)
                *out += aliased[i].length;
        }
        break;
```

```
static inline size_t kbase_reg_current_backed_size(struct kbase_va_region *reg)
{
    KBASE_DEBUG_ASSERT(reg);
    /* if no alloc object the backed size naturally is 0 */
    if (!reg->cpu_alloc)
        return 0;

    KBASE_DEBUG_ASSERT(reg->cpu_alloc);
    KBASE_DEBUG_ASSERT(reg->gpu_alloc);
    KBASE_DEBUG_ASSERT(reg->cpu_alloc->nents == reg->gpu_alloc->nents);

    return reg->cpu_alloc->nents;
}
```

Search the start address



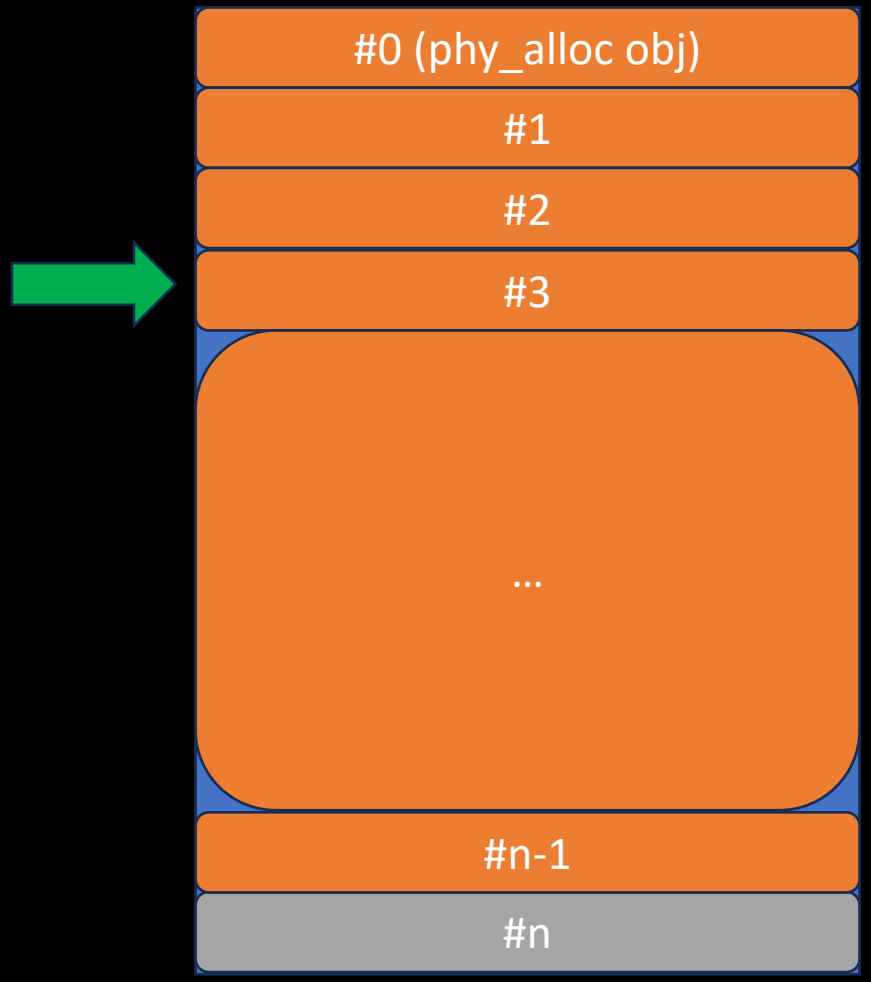
kbase_mem_phy_alloc & page_array

```
struct kbase_mem_phy_alloc {
        struct kref         kref;
        atomic_t            gpu_mappings;
        atomic_t            kernel_mappings;
        size_t              nents;
        struct tagged_addr  *pages;
```

kbasep_kinstr_prfcnt_get_request_info_list

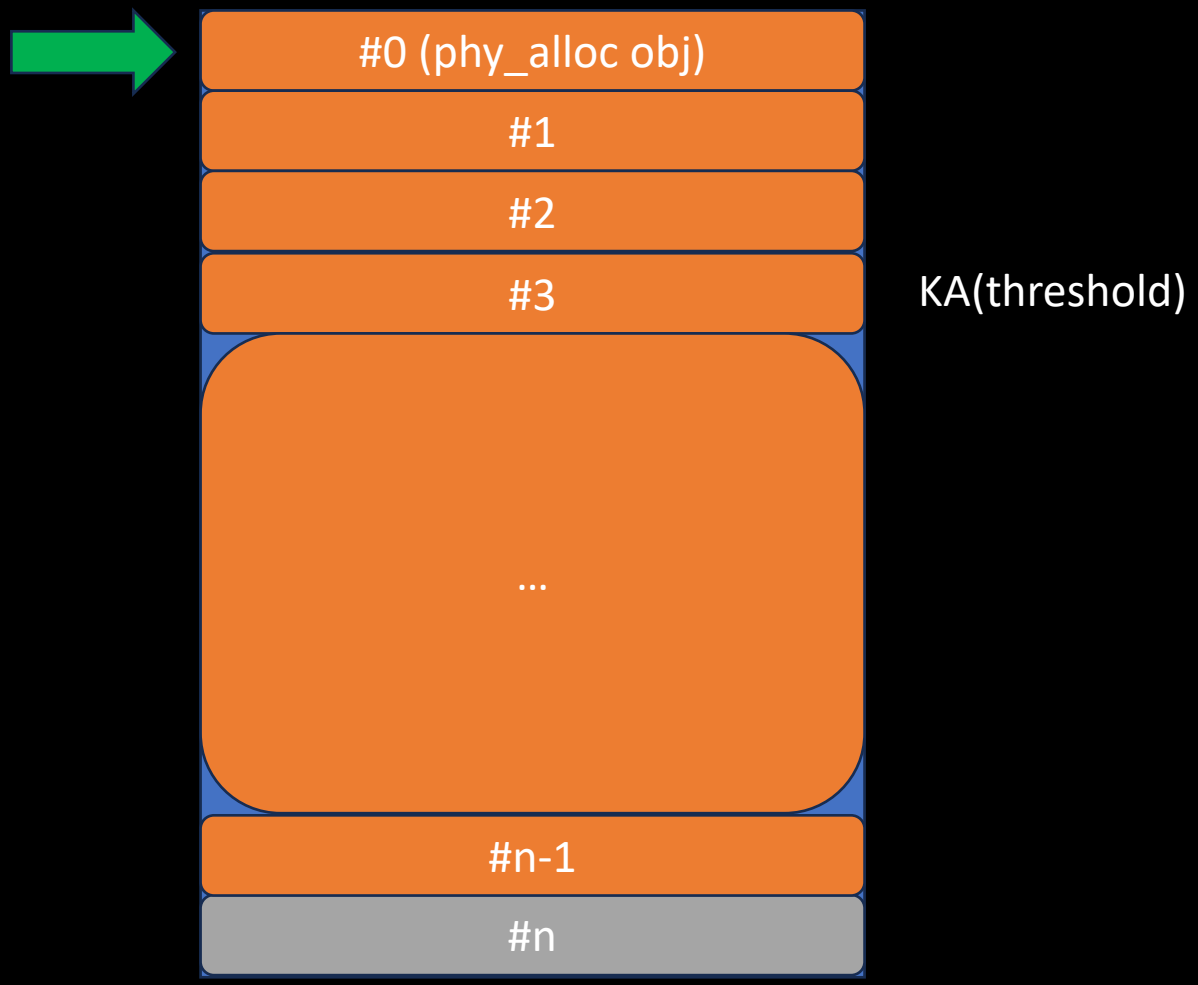
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Search the start address



- Trigger the bug
  - KA - 0x1000 * index + nents_offset
- Query the nents
  - 0

Search the start address



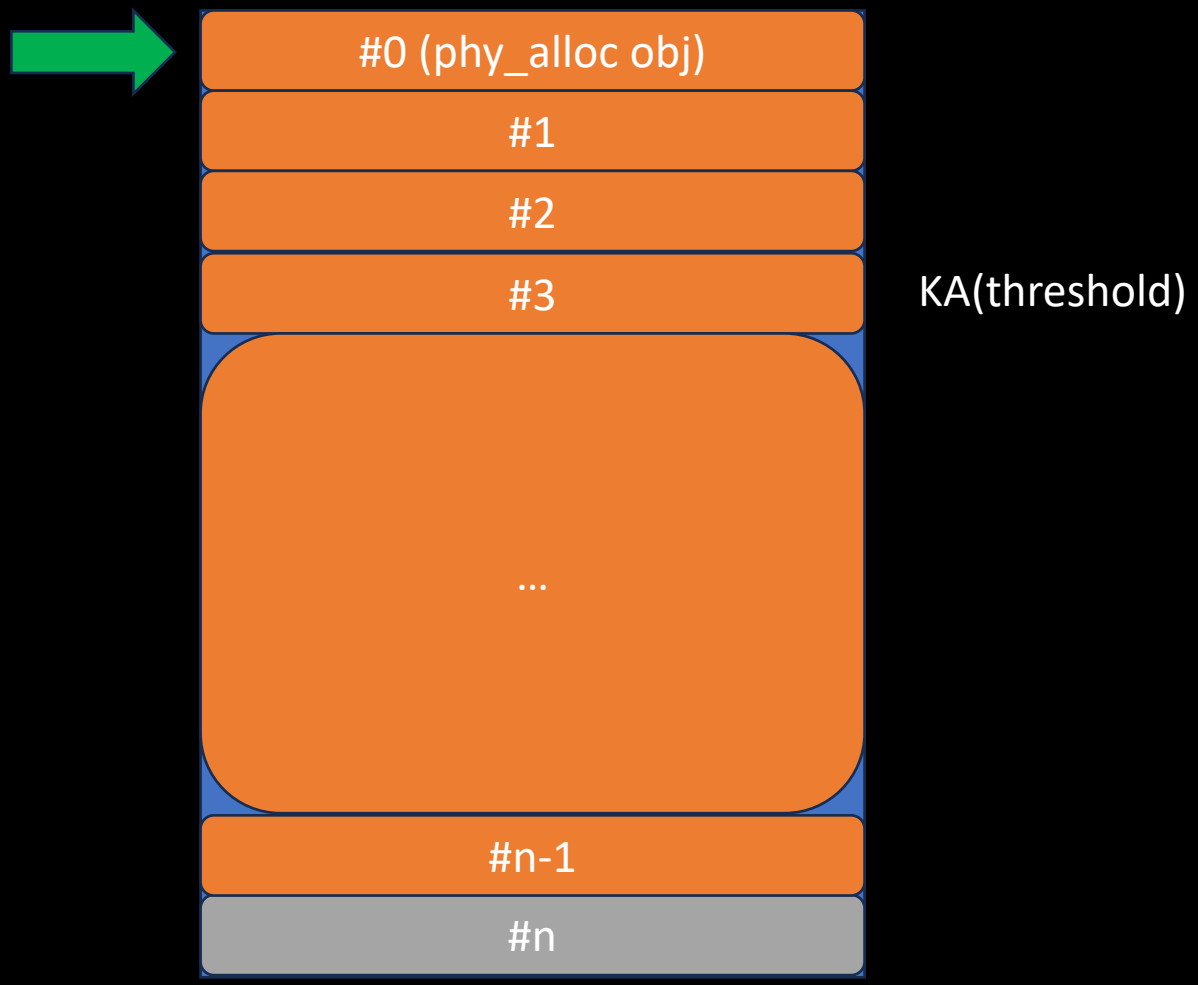
- Trigger the bug
  - KA - 0x1000 * index + nents_offset
- Query the nents
  - 1
- GPU VA Reg_n
  - Precise kernel address
  - Fields are corrupted

Predict the address

- Boot #n

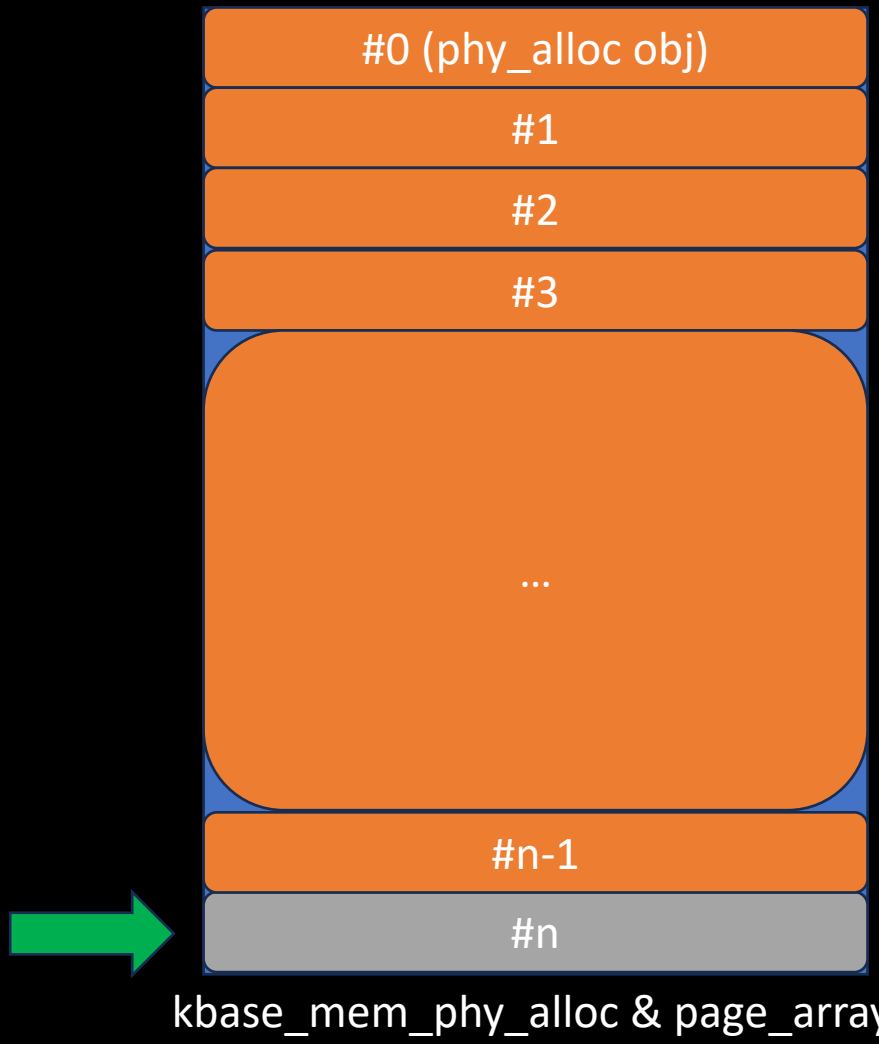
0xffffffc03cc0f000-0xffffffc03dc10000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03dc10000-0xffffffc03ec11000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03ec11000-0xffffffc03fc12000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03fc12000-0xffffffc040c13000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc040c13000-0xffffffc041c14000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages // Reg_n
0xffffffc041c14000-0xffffffc042c15000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages // Reg_n+1
0xffffffc042c15000-0xffffffc043c16000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc043c16000-0xffffffc044c17000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc044c17000-0xffffffc045c18000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc045c18000-0xffffffc046c19000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc046c19000-0xffffffc047c1a000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc047c1a000-0xffffffc048c1b000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc048c1b000-0xffffffc049c1c000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages

Search the start address



- Trigger the bug
  - KA - 0x1000 * index + nents_offset
- Query the nents
  - 1
- GPU VA Reg_n
  - Precise kernel address
  - Fields are corrupted
- GPU VA Reg_n+1
  - Precise kernel address

Search the start address



- Trigger the bug
  - KA - 0x1000 * index + nents_offset
- Query the nents
  - 1
- GPU VA Reg_n
  - Precise kernel address
  - Fields are corrupted
- GPU VA Reg_n+1
  - Precise kernel address
- Crash rate (1/n)
  - 16MB(<0.025%)

Write primitive

- kbasep_kinstr_prfcnt_get_request_info_list

```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- kbasep_kinstr_prfcnt_get_block_info_list

```
0000: 00 00 00 00 00 00 00 00 ?? ?? ?? ?? 00 00 00 00
0016: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- The last sentinel item

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
struct kbase_mem_phy_alloc {
        struct kref         kref;
        atomic_t            gpu_mappings;
        atomic_t            kernel_mappings;
        size_t              nents;
        struct tagged_addr  *pages;
        struct list_head    mappings;
        struct list_head    evict_node;
        size_t              evicted;
        struct kbase_va_region *reg;
        enum kbase_memory_type type;
        struct kbase_vmap_struct *permanent_map;
        u8 properties;
        u8 group_id;
        union {umm, alias, native, user_buf} imported;
```

Write primitive

- kbasep_kinstr_prfcnt_get_request_info_list

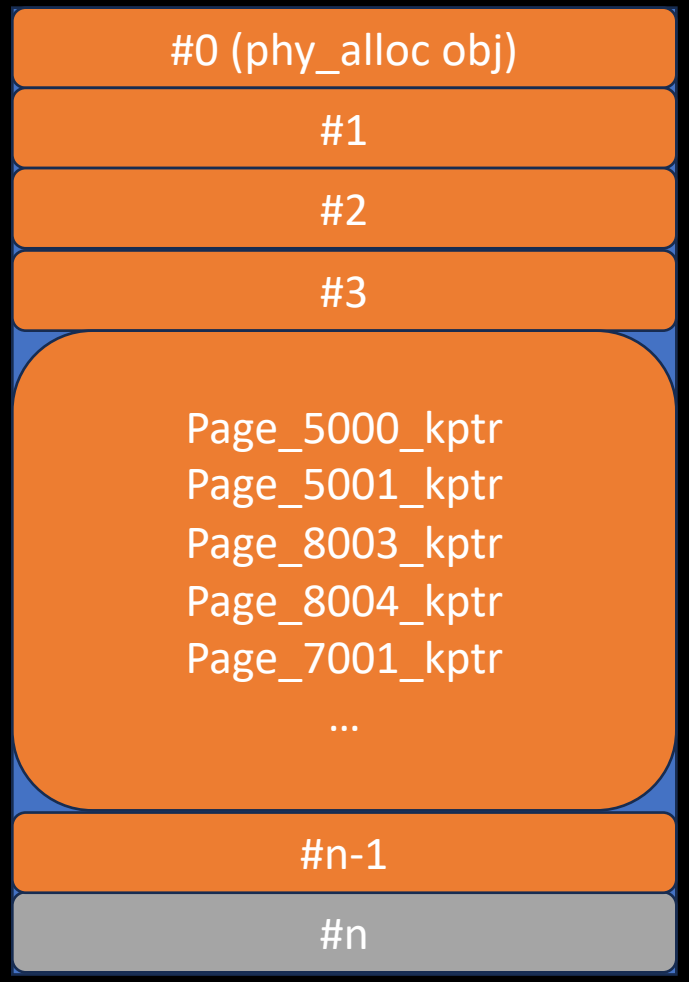
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- kbasep_kinstr_prfcnt_get_block_info_list

```
0000: 00 00 00 00 00 00 00 00 ?? ?? ?? ?? 00 00 00 00
0016: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

- The last sentinel item

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



kbase_mem_phy_alloc & page_array

Write primitive

- kbasep_kinstr_prfcnt_get_request_info_list

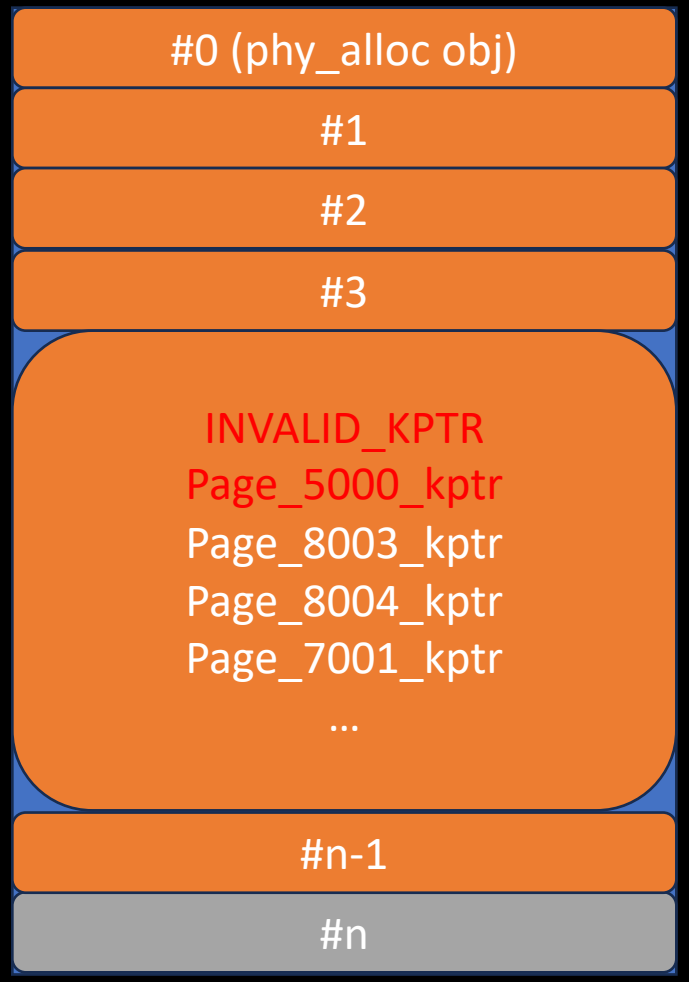
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- kbasep_kinstr_prfcnt_get_block_info_list

```
0000: 00 00 00 00 00 00 00 00 ?? ?? ?? ?? 00 00 00 00
0016: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

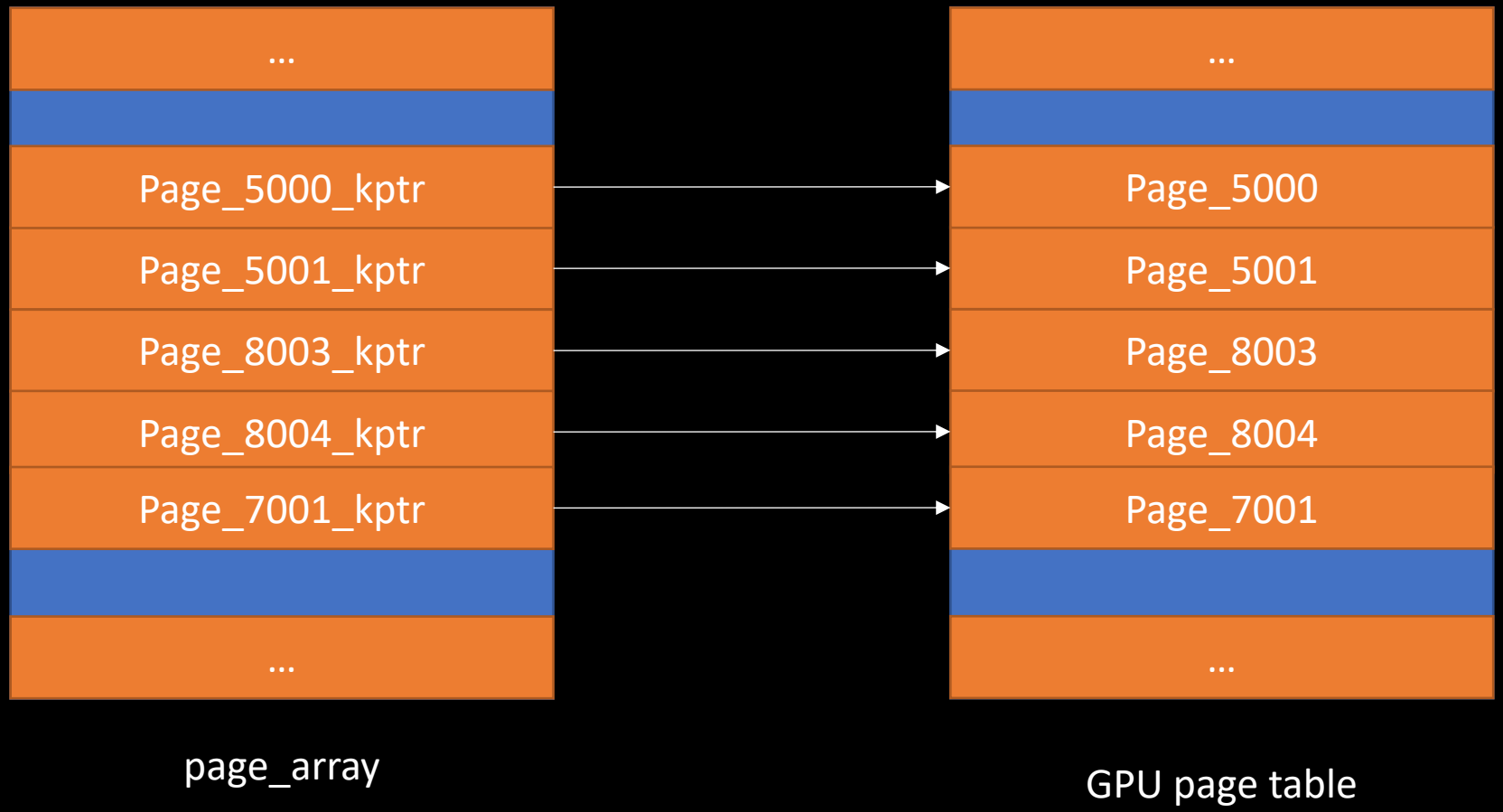
- The last sentinel item

```
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

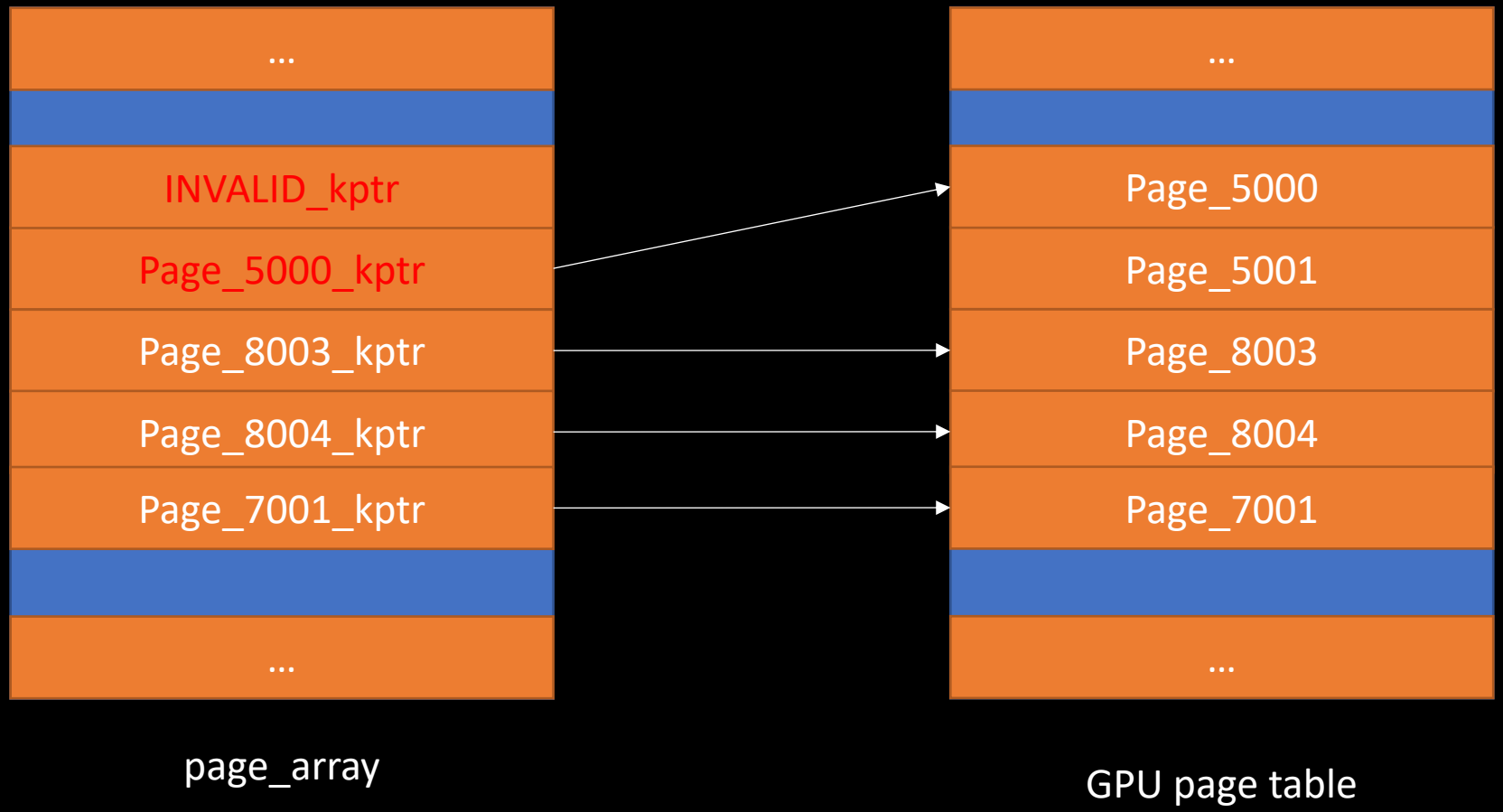


kbase_mem_phy_alloc & page_array

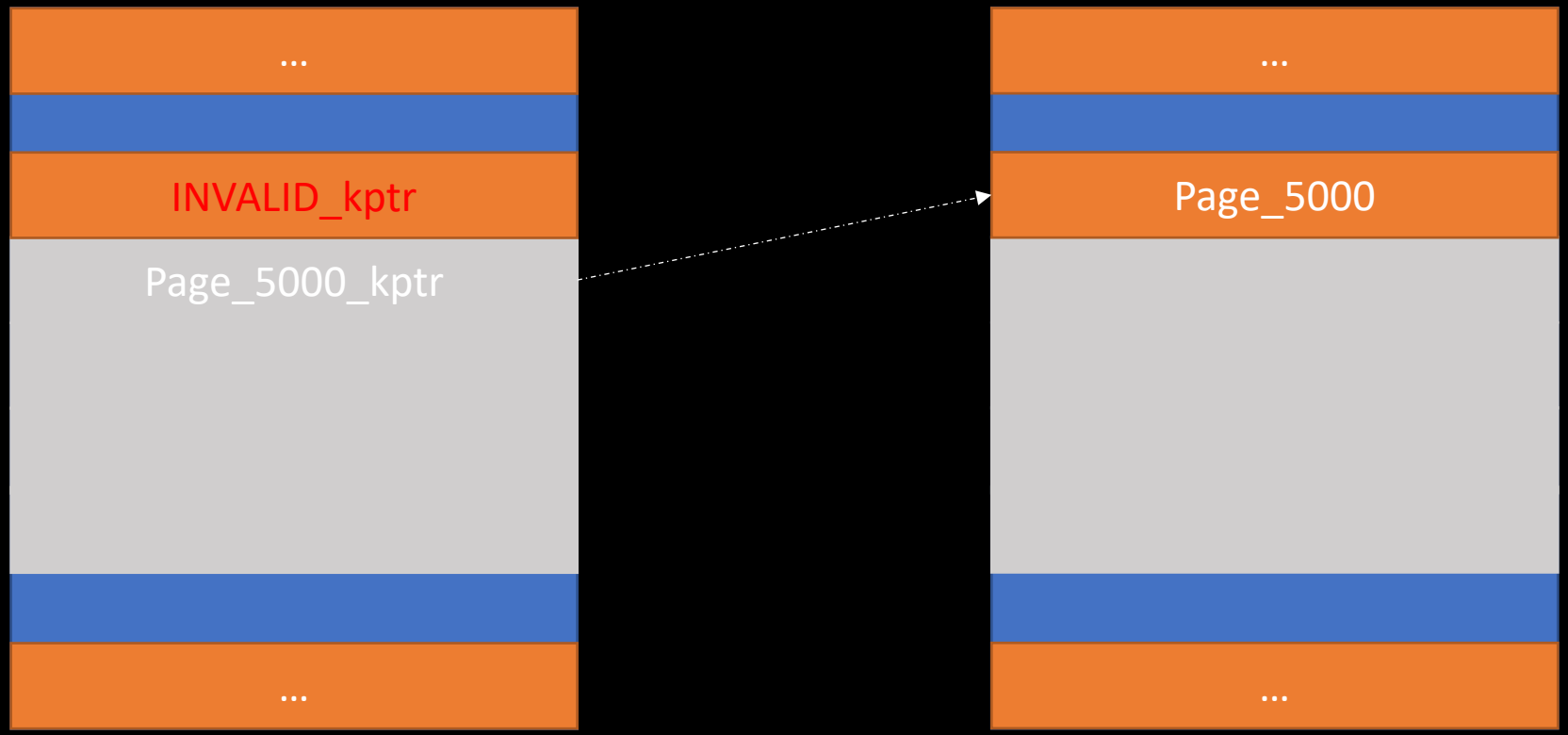
Page UAF in GPU MMU



Page UAF in GPU MMU



Page UAF in GPU MMU



How to overwrite the right page kernel pointer? 🤔

CVE-2022-36449

- MMU entries can be dumped
  - Leak the physical page frames(including zero page)

```
PGD(level: 0): 0(next PGD: 8da80a000)
 PGD(level: 1): 7f00000000(next PGD: 8da686000)
  PGD(level: 2): 7f00000000(next PGD: 8da685000)
     Range: 7f00000000-7f00020000: _*******************************
     Range: 7f00020000-7f00040000: ***********************_________
     Range: 7f00040000-7f00060000: ____________________________________
     Range: 7f00060000-7f00080000: ____________________________________
     Range: 7f00080000-7f000a0000: ____________________________________
     Range: 7f000a0000-7f000c0000: ____________________________________
     Range: 7f000c0000-7f000e0000: ____________________________________
     Range: 7f000e0000-7f00100000: ____________________________________
     Range: 7f00100000-7f00120000: ____________________________________
     Range: 7f00120000-7f00140000: ____________________________________
     Range: 7f00140000-7f00160000: ____________________________________
     Range: 7f00160000-7f00180000: ____________________________________
     Range: 7f00180000-7f001a0000: ____________________________________
     Range: 7f001a0000-7f001c0000: ____________________________________
     Range: 7f001c0000-7f001e0000: ____________________________________
     Range: 7f001e0000-7f00200000: ____________________________________
```

“-” denotes Invalid descriptor

“*” denotes Page descriptor

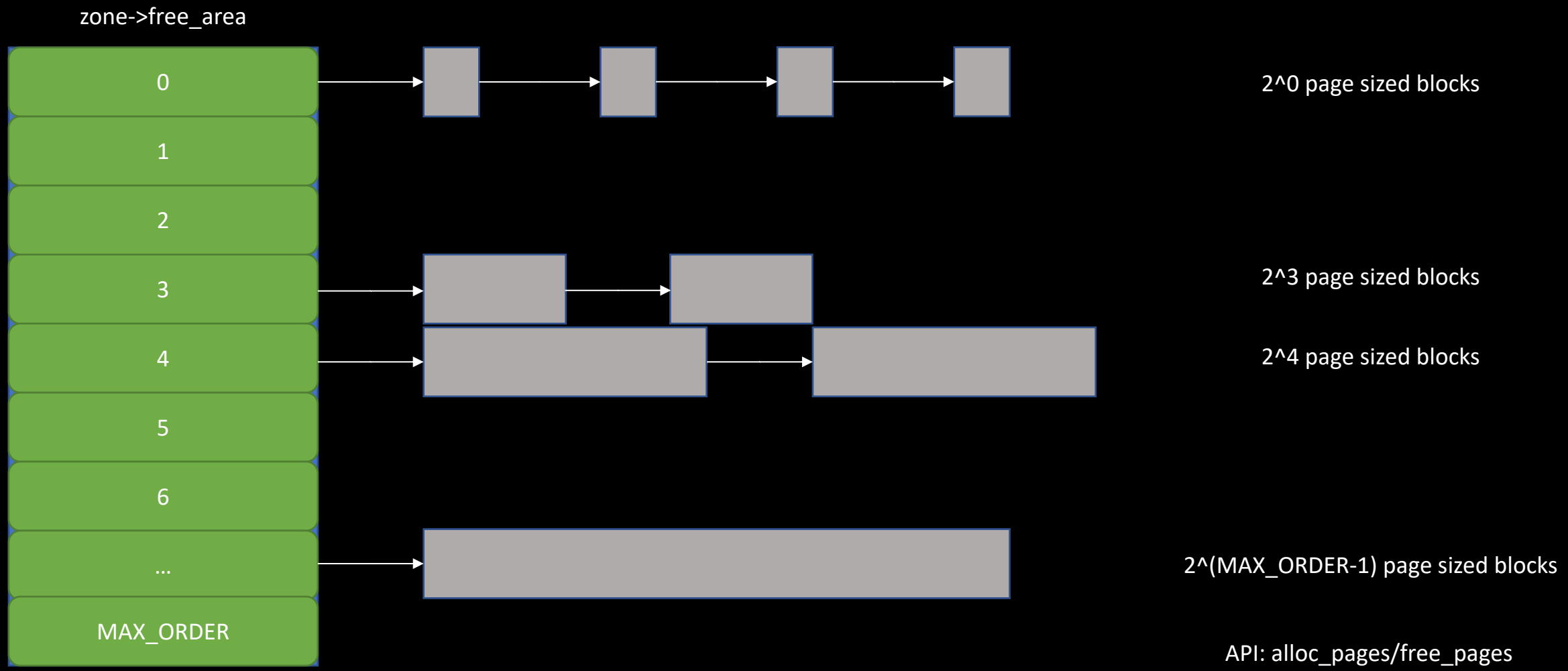
CVE-2022-36449

- MMU entries can be dumped
  - Leak the physical page frames(including zero page)
  - Fixed and guarded by non-default config

```
2850        case PFN_DOWN(BASE_MEM_MMU_DUMP_HANDLE):
2851    #if defined(CONFIG_MALI_VECTOR_DUMP)
2852            /* MMU dump */
2853            err = kbase_mmu_dump_mmap(kctx, vma, &reg, &kaddr);
2854            if (err != 0)
2855                goto out_unlock;
2856            /* free the region on munmap */
2857            free_on_close = 1;
2858            break;
2859    #else
```

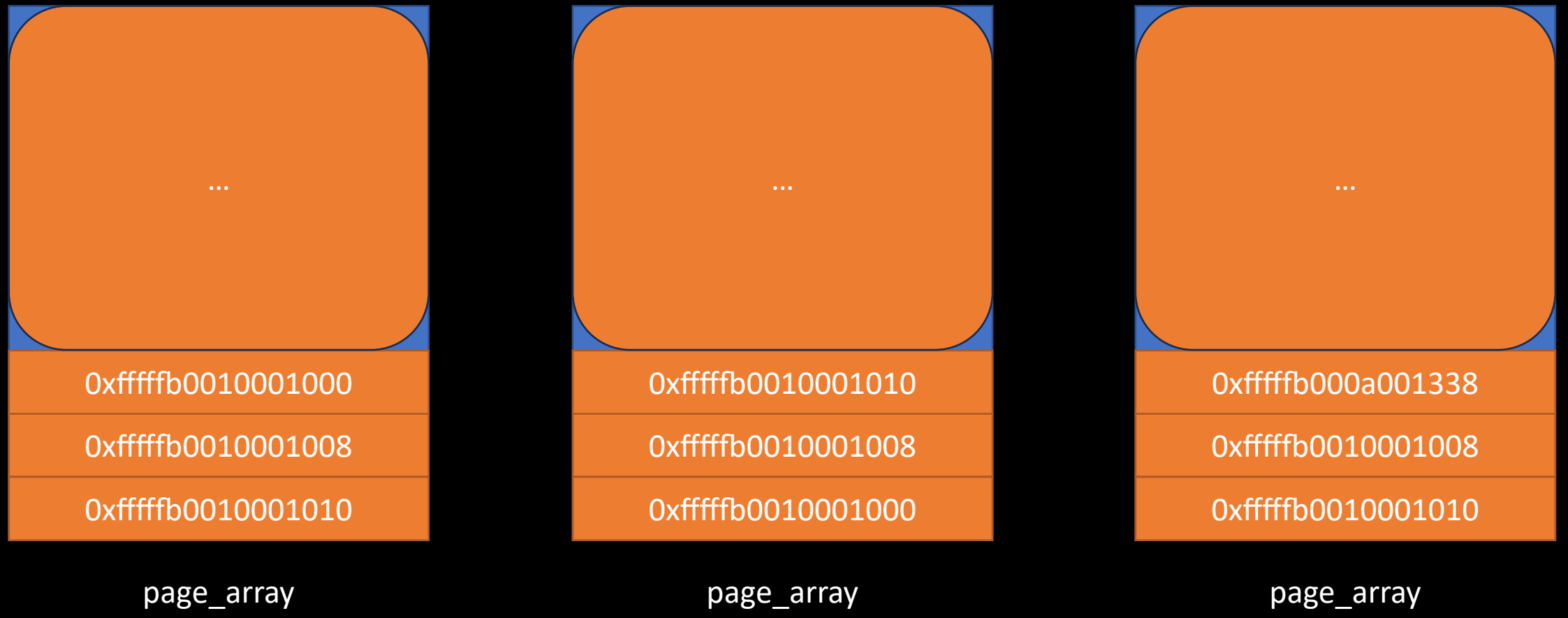
- Use only one bug to exploit

Buddy allocator internal

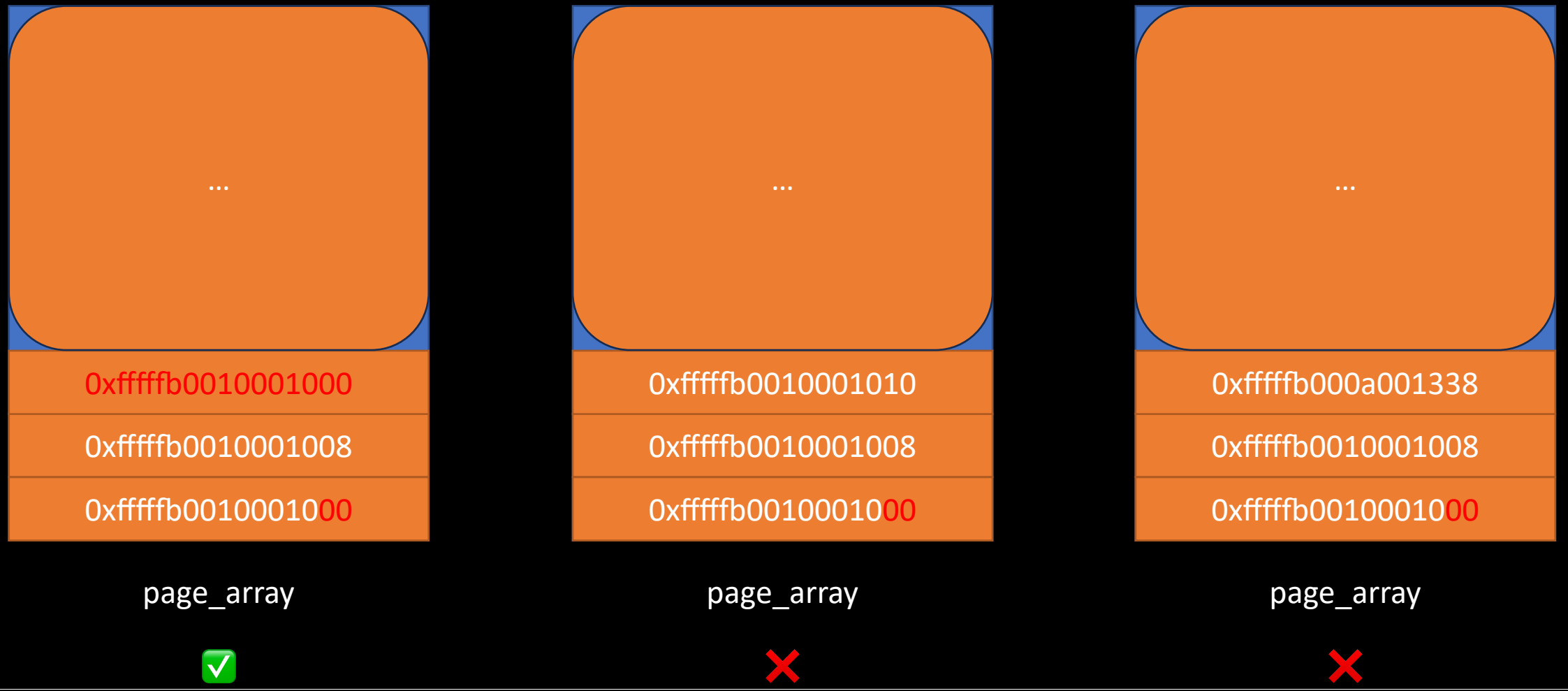


API: alloc_pages/free_pages

Page UAF in GPU MMU



Page UAF in GPU MMU



Physical Page Management

```
struct kbase_mem_pool {
        struct kbase_device *kbdev;
        size_t            cur_size;
        size_t            max_size;
        u8                order;
        u8                group_id;
        spinlock_t        pool_lock;
        struct list_head    page_list;
        struct shrinker     reclaim;

        struct kbase_mem_pool *next_pool;

        bool dying;
        bool dont_reclaim;
};
```

```
struct kbase_mem_pool_group {
        struct kbase_mem_pool small[16];
        struct kbase_mem_pool large[16];
};

int kbase_context_mem_pool_group_init(struct kbase_context
*kctx)
{
        return kbase_mem_pool_group_init(
        &kctx->mem_pools,
        kctx->kbdev,
        &kctx->kbdev->mem_pool_defaults,
        &kctx->kbdev->mem_pools);
}
```

Physical Page Management

- Allocate
  - Step 1: allocate from the kctx->mem_pools. If insufficient, goto step 2
  - Step 2: allocate from the kbdev->mem_pools. If insufficient, goto step 3
  - Step 3: allocate from the kernel
- Free
  - Step 1: add the pages to kctx->mem_pools. If full, goto step 2
  - Step 2: add the pages to kbdev->mem_pools. If full, goto step 3
  - Step 3: free the remaining pages to the kernel
- Shrinker
  - register_shrinker(&kctx->reclaim);
  - register_shrinker(&pool->reclaim);

Page UAF in GPU MMU

- kbase_mmu_insert_pages_no_flush
  - If invalid, allocate one page as the PGD
  - Allocate from kbdev->mem_pools, not from kctx->mem_pools

```
if (!kbdev->mmu_mode->pte_is_valid(page[vpfn], level)) {
    enum kbase_mmu_op_type flush_op = KBASE_MMU_OP_NONE;
    unsigned int current_valid_entries;
    u64 managed_pte;

    target_pgd = kbase_mmu_alloc_pgd(kbdev, mmut);
    if (target_pgd == KBASE_MMU_INVALID_PGD_ADDRESS) {
        dev_dbg(kbdev->dev, "%s: kbase_mmu_alloc_pgd failure\n",
                __func__);
        kunmap(p);
        return -ENOMEM;
    }
```

```
static phys_addr_t kbase_mmu_alloc_pgd(struct kbase_device *kbdev,
        struct kbase_mmu_table *mmut)
{
    u64 *page;
    struct page *p;
    phys_addr_t pgd;

    p = kbase_mem_pool_alloc(&kbdev->mem_pools.small[mmut->group_id]);
    if (!p)
        return KBASE_MMU_INVALID_PGD_ADDRESS;
```

Page UAF in GPU MMU

- kbase_mmu_insert_pages_no_flush
  - If invalid, allocate one page as the PGD
  - Allocate from kbdev->mem_pools, not from kctx->mem_pools
  - It’s possible to reuse the freed pages as the PGD

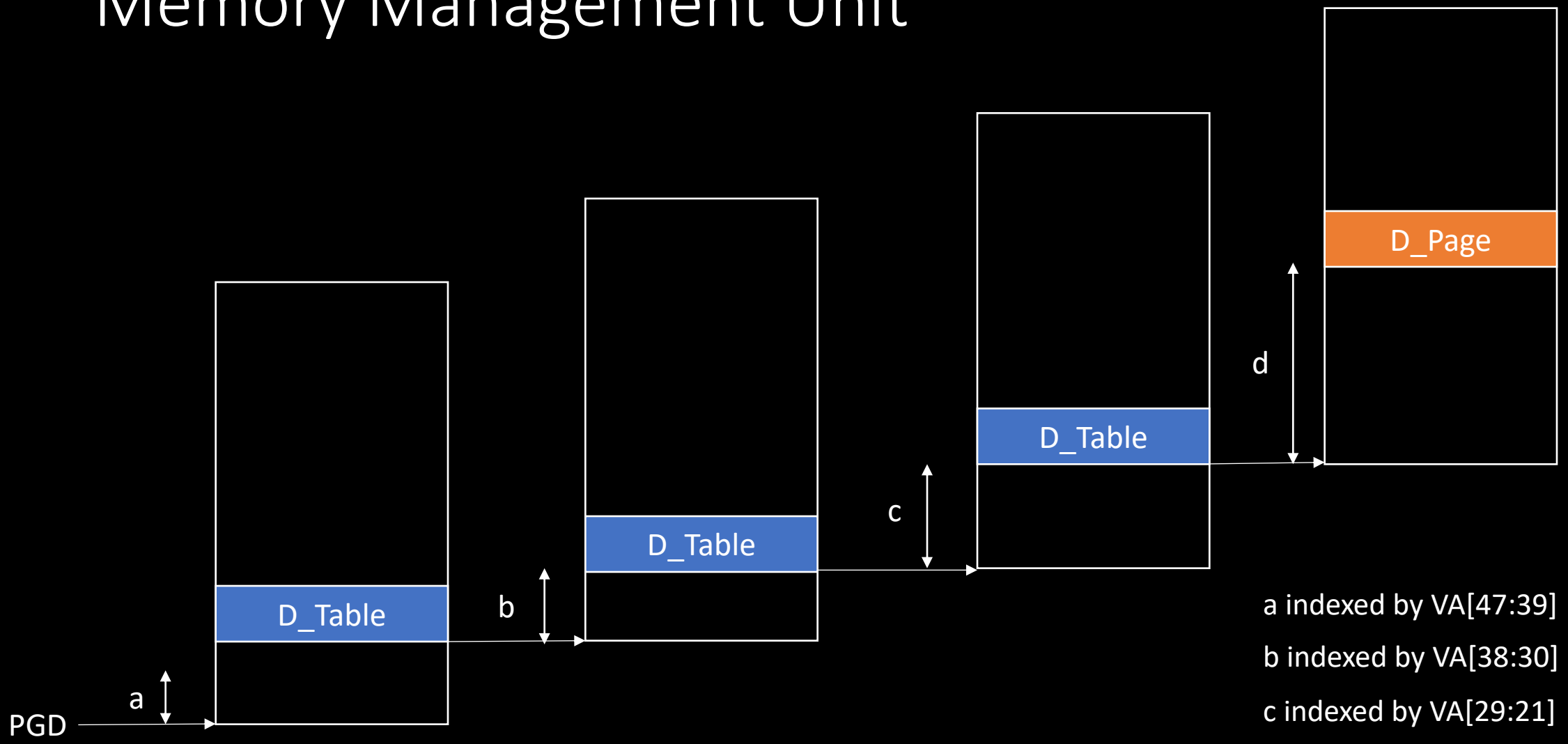
```
if (!kbdev->mmu_mode->pte_is_valid(page[vpfn], level)) {
    enum kbase_mmu_op_type flush_op = KBASE_MMU_OP_NONE;
    unsigned int current_valid_entries;
    u64 managed_pte;

    target_pgd = kbase_mmu_alloc_pgd(kbdev, mmut);
    if (target_pgd == KBASE_MMU_INVALID_PGD_ADDRESS) {
        dev_dbg(kbdev->dev, "%s: kbase_mmu_alloc_pgd failure\n",
                __func__);
        kunmap(p);
        return -ENOMEM;
    }
```

```
static phys_addr_t kbase_mmu_alloc_pgd(struct kbase_device *kbdev,
        struct kbase_mmu_table *mmut)
{
    u64 *page;
    struct page *p;
    phys_addr_t pgd;

    p = kbase_mem_pool_alloc(&kbdev->mem_pools.small[mmut->group_id]);
    if (!p)
        return KBASE_MMU_INVALID_PGD_ADDRESS;
```

Memory Management Unit



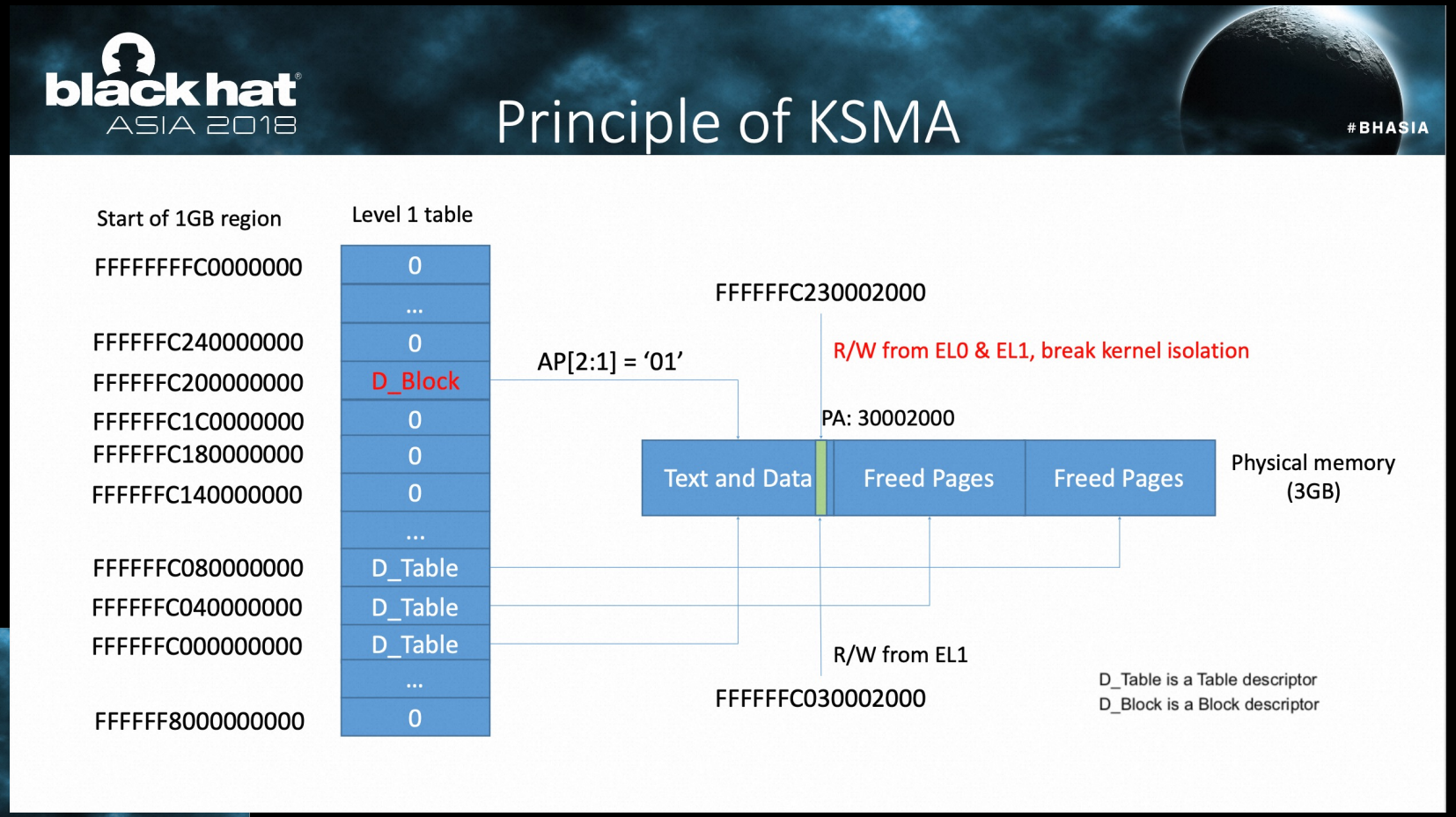
a indexed by VA[47:39]

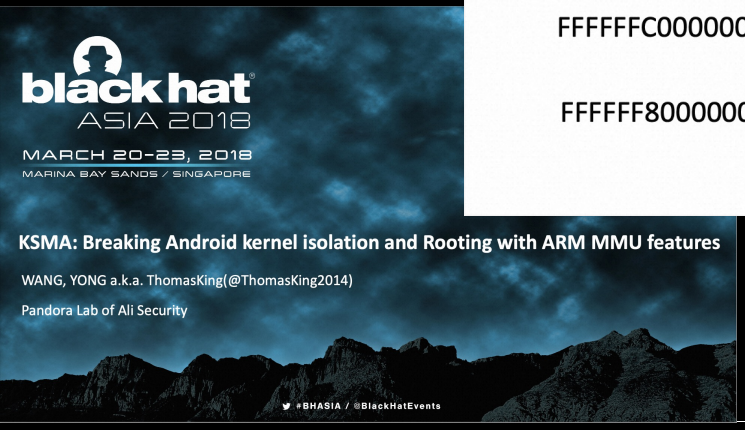
b indexed by VA[38:30]

c indexed by VA[29:21]

d indexed by VA[20:12]

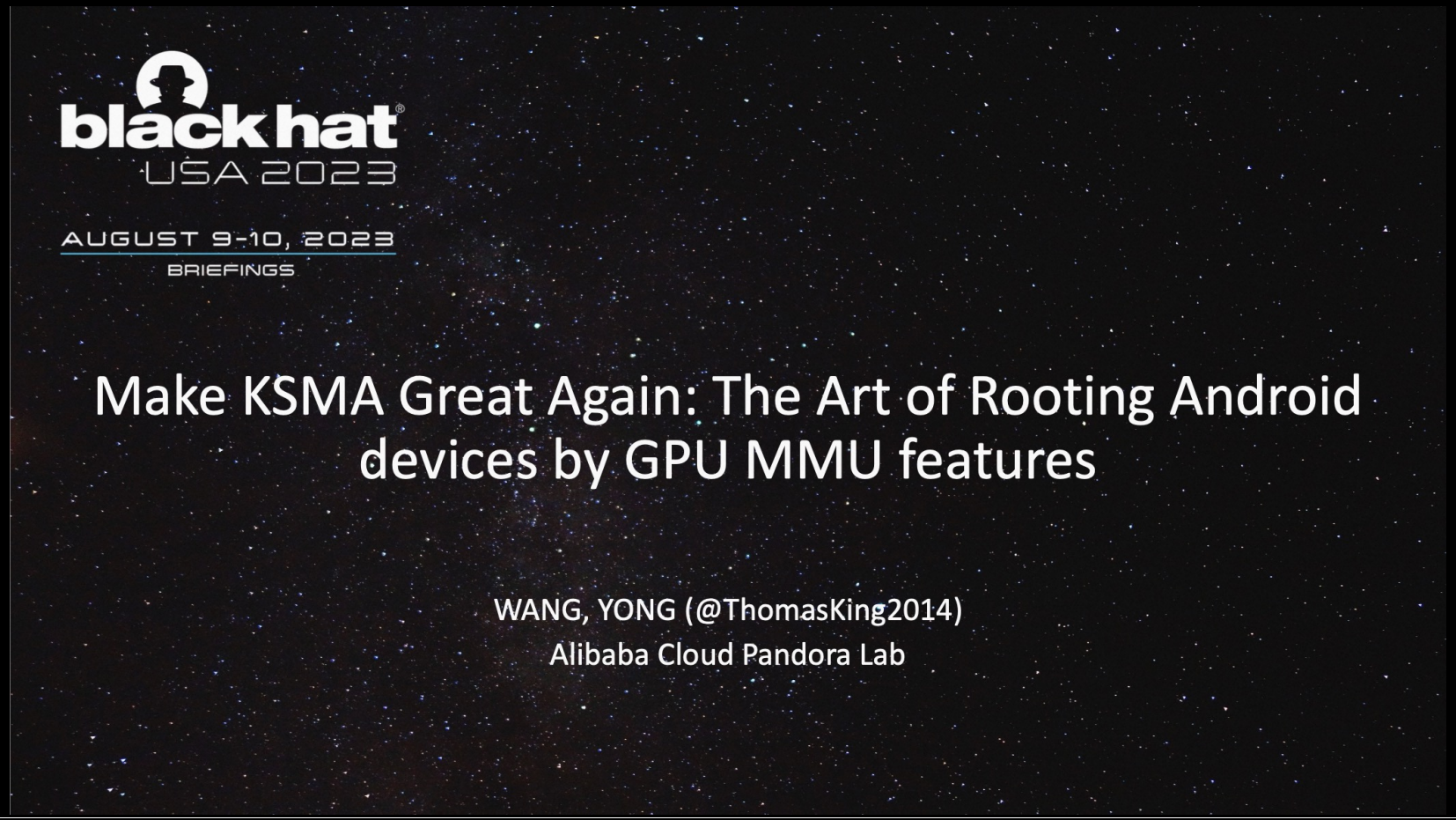
Kernel Space Mirroring Attack





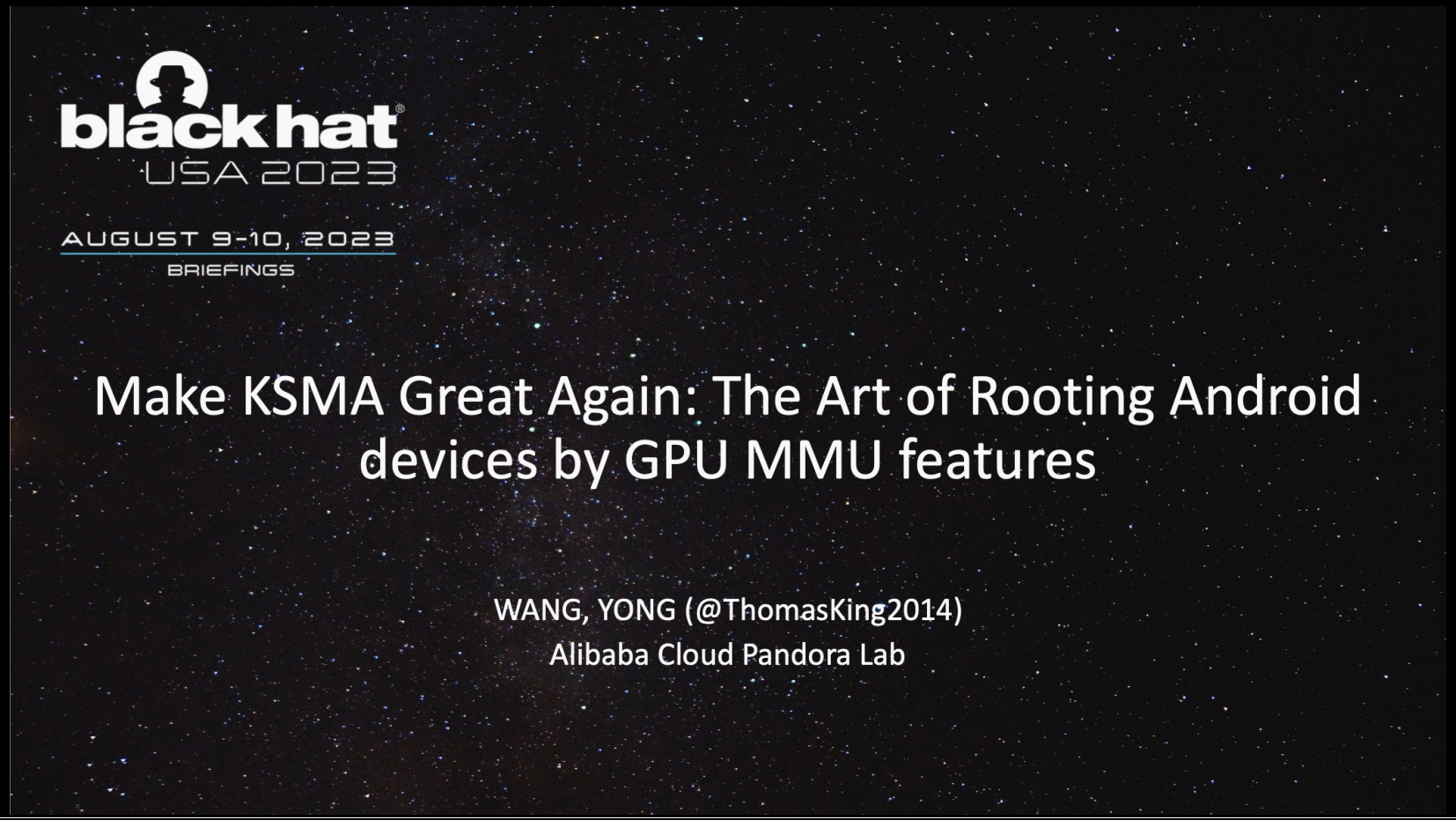
Page UAF in GPU MMU

- How to craft the valid block entry

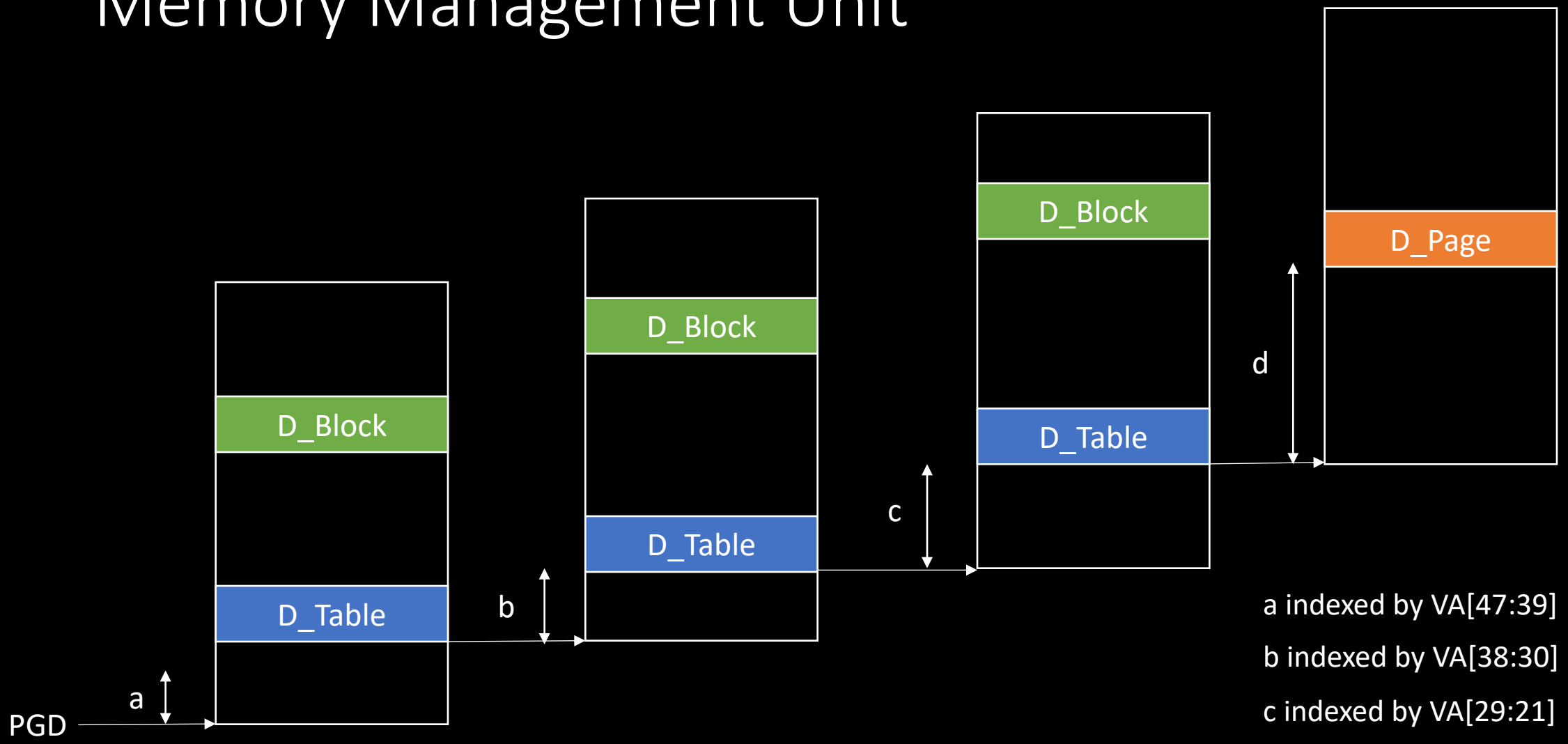


Page UAF in GPU MMU

- MTE(?)



Memory Management Unit



a indexed by VA[47:39]

b indexed by VA[38:30]

c indexed by VA[29:21]

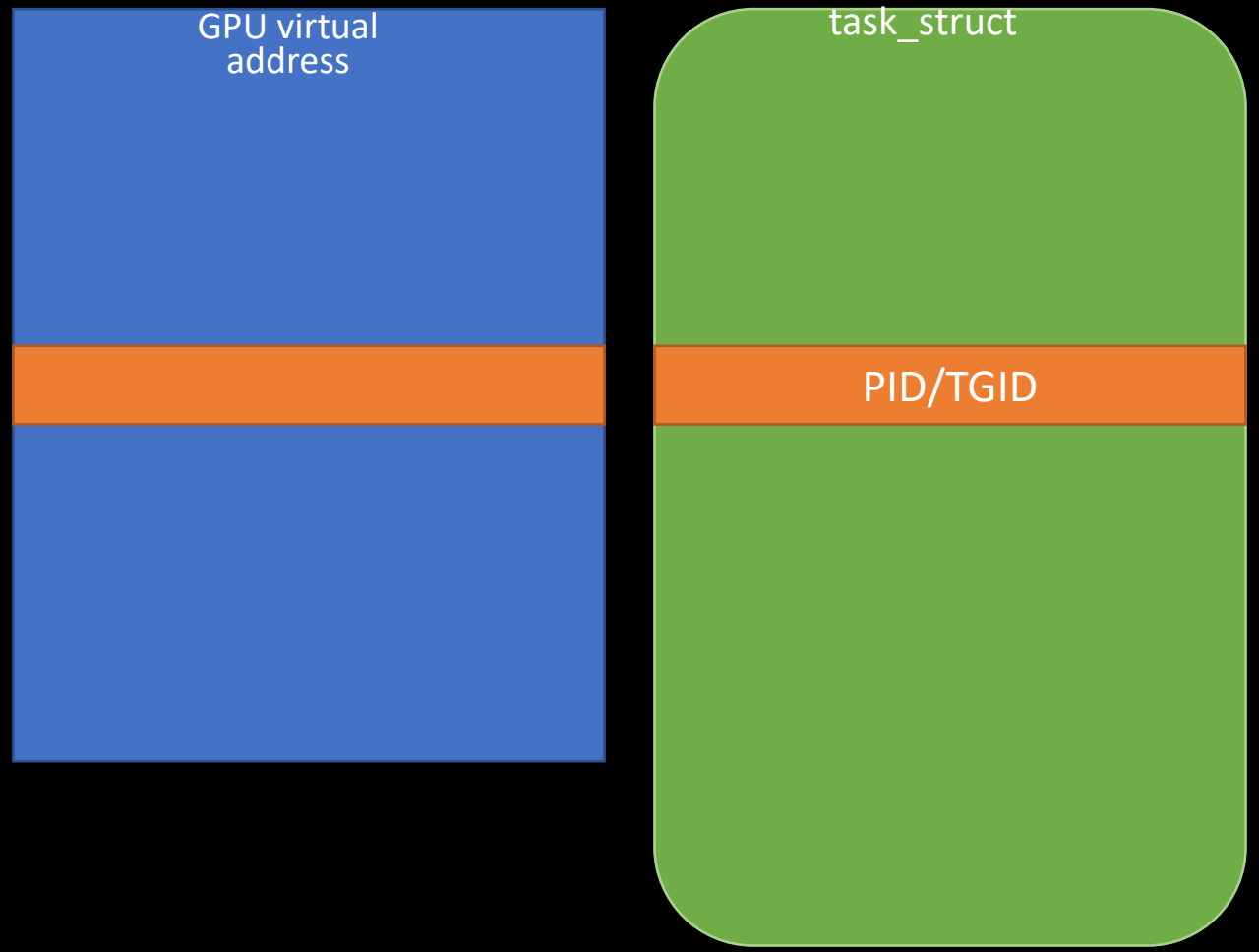
d indexed by VA[20:12]

Exploitation

```
shell:/ # cat /proc/iomem |grep "System RAM"
80000000-8affffff : System RAM
8b06e000-901fffff : System RAM
90600000-91dfffff : System RAM
97400000-97ffffff : System RAM
98500000-b71fffff : System RAM
c0000000-d80fffff : System RAM
da710000-dc20ffff : System RAM
dc620000-dfffffff : System RAM
e3800000-f87fffff : System RAM
f9800000-fd3effff : System RAM
fd3ff000-fd7fffff : System RAM
fd900000-fd90bfff : System RAM
fd90d000-ffffffff : System RAM
880000000-9fffffff : System RAM
```

Exploitation

- Search the task_struct objects
  - PID/TID
  - Comm
  - …
- Leak kernel pointers
  - Cred - *(u64*)(A + OFF_CRED – OFF_PID)
- Compute the PFN slide
  - PA <-> VA



Share the same physical page(start address aligned)

Arbitrary Physical Page Read/Write

- Put it together
  - Step 1: Spray the GPU VA regions without allocating physical pages
  - Step 2: Search the target kbase_mem_phy_alloc obj starting from the predicted kernel address
  - Step 3: calculate the kernel address of the next kbase_mem_phy_alloc obj
  - Step 4: Commit the large number of pages
  - Step 5: Trigger the bug and overwrite the last page pointer
  - Step 6: Shrink the related region and free the last page
  - Step 7: Reuse the page as the PGD. If it fails, goto step 3
  - Step 8: Apply the KSMA exploitation technique and access the whole physical pages
  - Step 9: Bypass the vendor’s mitigation and gain the root shell

```
shell:/ $
```

Agenda

- Introduction
- Bug analysis and exploitation
- *Conclusion*

Takeaways

- It’s possible to reliably predict the kernel addresses of attacker-controlled objects.
- Using only one bug to exploit now needs more advanced exploitation technique.
- With MTE mitigation landed, the high quality bugs and more advanced exploitation technique becomes more valuable.

References

[1] https://googleprojectzero.blogspot.com/2023/11/first-handset-with-mte-on-market.html

[2] https://blackhat.com/us-23/briefings/schedule/index.html#make-ksma-great-again-the-art-of-rooting-android-devices-by-gpu-mmu-features-32132

[3] https://googleprojectzero.blogspot.com/2020/02/mitigations-are-attack-surface-too.html

[4] https://github.blog/2024-03-18-gaining-kernel-code-execution-on-an-mte-enabled-pixel-8/


Thank you!
WANG, YONG (@ThomasKing2014)
ThomasKingNew@gmail.com