POC 2023

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 Twitter/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

Android kernel mitigations 101

- Android 13
  - 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

Memory Tagging Extension

Attackers have been strategizing for 4+ years

- Strong infoleak bug to leak tags → MTE defeated
- CPU side channel attacks to leak tags (i.e. PAC
- Exemption of tagging on certain edge-cases
- Attack surfaces that are not affected by MTE
- Logic bugs
- Some great insight on Saar Amar's presentation :
  "Security Analysis of MTE Through Examples – BlueHatIL 2022"

No doubt it's one of the strongest mitigations, but strong attackers will likely re-strategize and adapt (or shift to other domains)

DATAFLOW SECURITY

Android kernel mitigations 101

- Android 14
  - 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 LPE attack surfaces

- DAC
  - ptmx (root root 0o666) ptmx_device
  - tty (root root 0o666) owntty_device
  - system (system system 0o664) dmabuf_system_heap_device
  - ashmem (root root 0o666) ashmem_device
  - binder(root root 0o777) binder_device
  - kgsl-3d0 (system system 0o666) gpu_device / mali0 (system system 0o664) gpu_device
- SELinux policy
  - ALLOW domain-->ptmx_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
  - ALLOW domain-->owntty_device (chr_file) [map append write ioctl watch_reads getattr read watch lock open]
  - ALLOW domain-->ashmem_device (chr_file) [map append write ioctl getattr read lock]
  - ALLOW untrusted_app-->dmabuf_system_heap_device (chr_file) [map ioctl watch_reads getattr read watch lock open]
  - ALLOW untrusted_app-->binder_device (chr_file) [map ioctl watch_reads getattr read watch lock open]
  - ALLOW untrusted_app-->gpu_device (chr_file) [map ioctl watch_reads getattr read watch lock open]
- Syscall
  - io_uring disabled

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;
    }
```

Diff: r33 vs r34

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!

Fix

```
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;
```

Diff: r34 vs r35

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
```

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
  - 224(32 * 7) bytes
  - 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

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

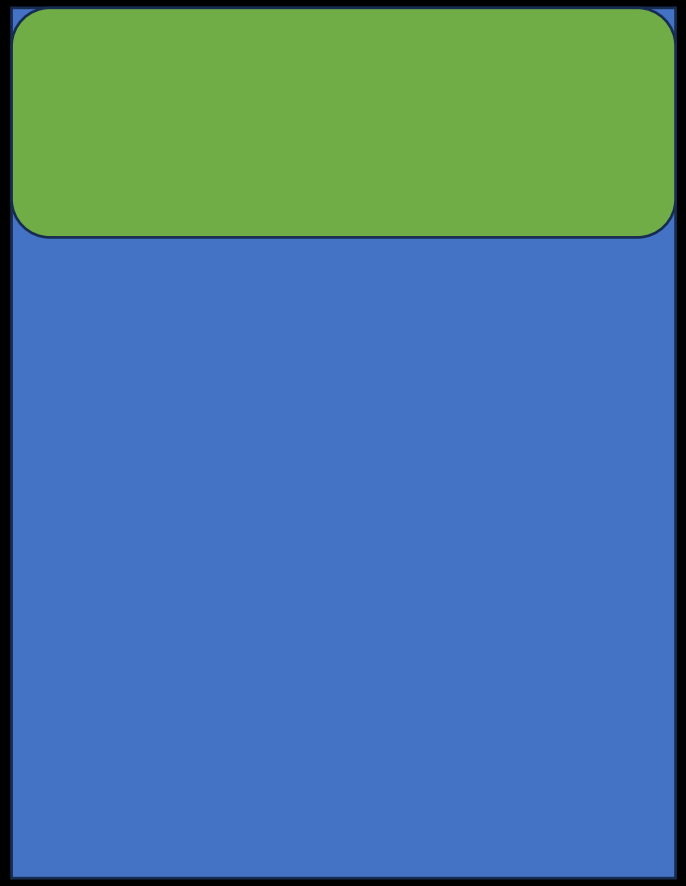
- Hardcode the addresses of kernel objects
  - Early allocated objects during booting

```
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



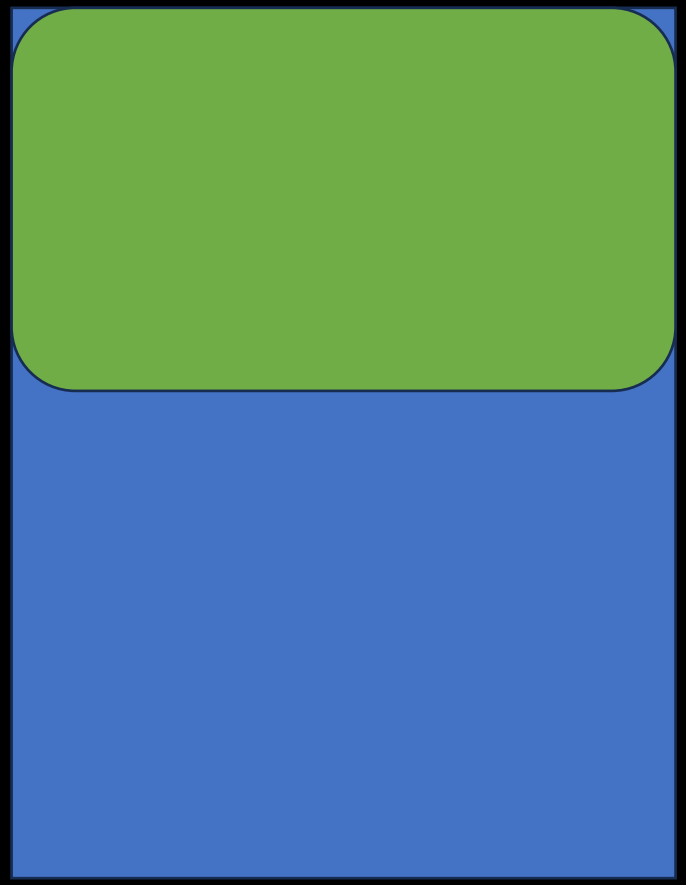
VMALLOC_END

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

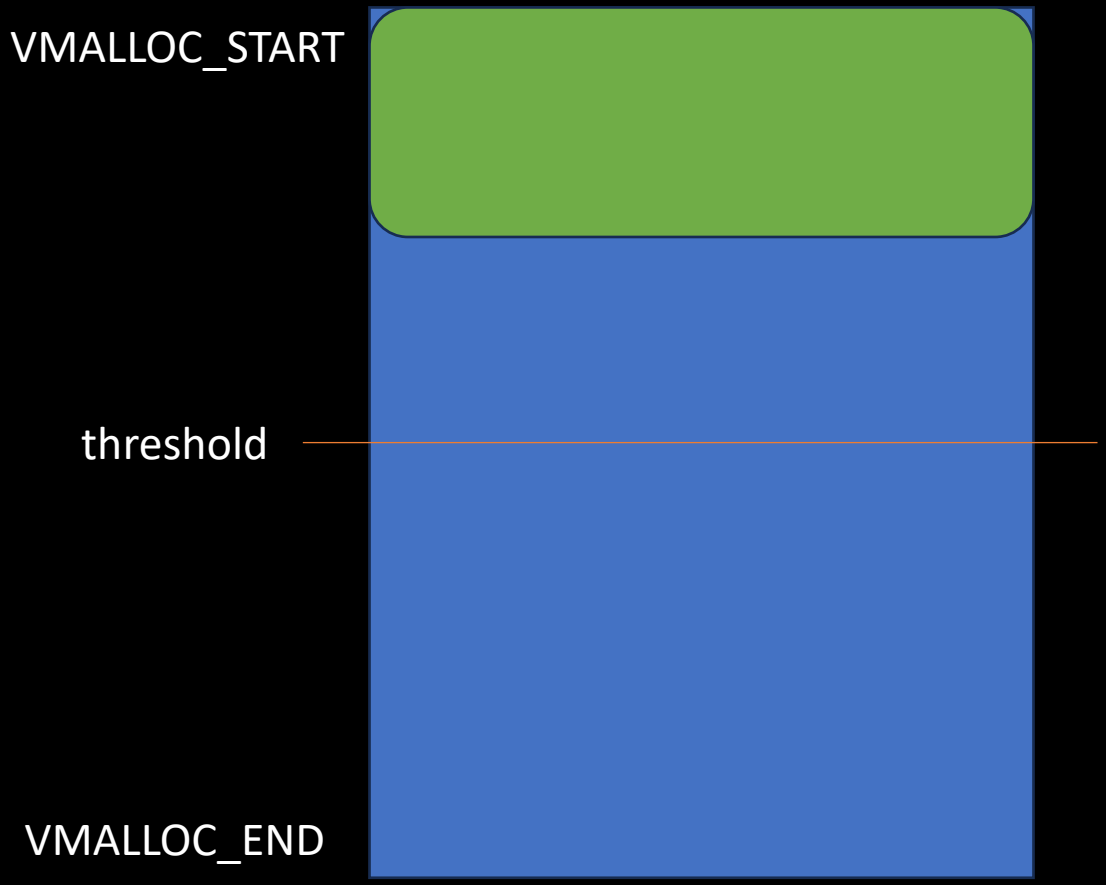


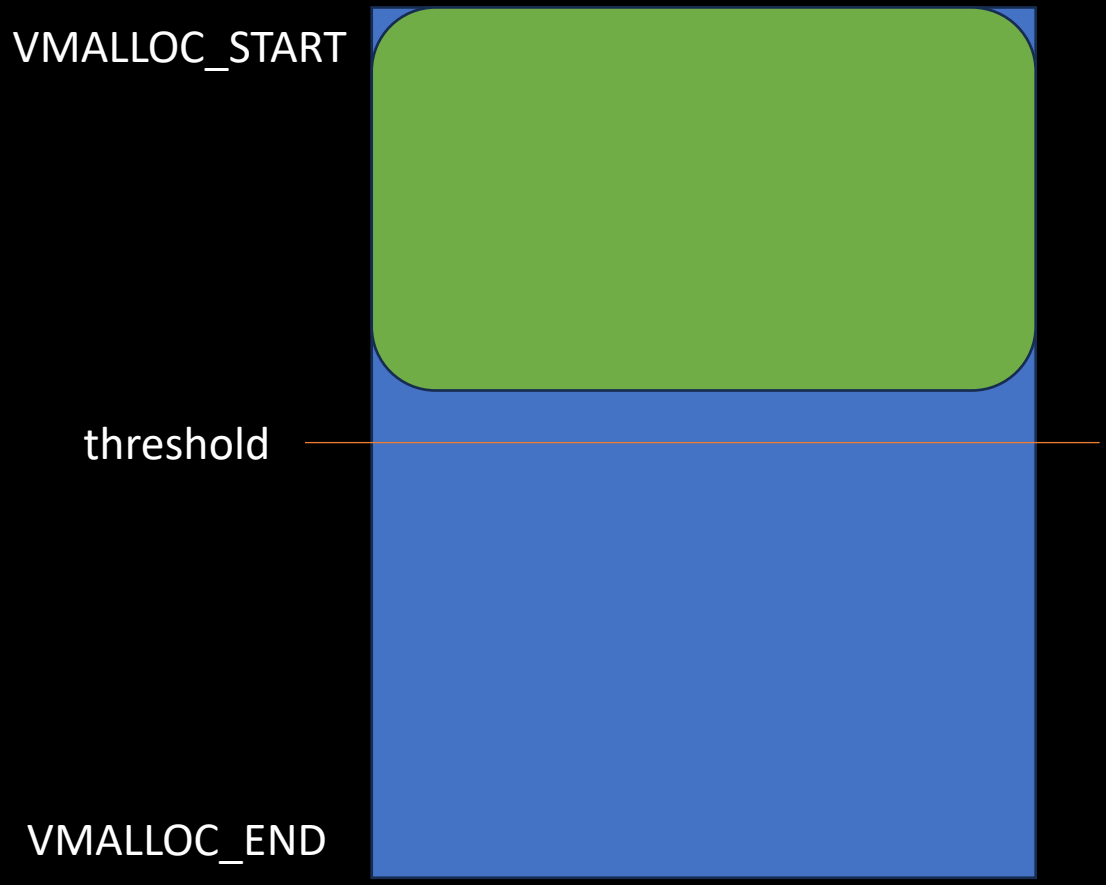
VMALLOC_END

Chrome

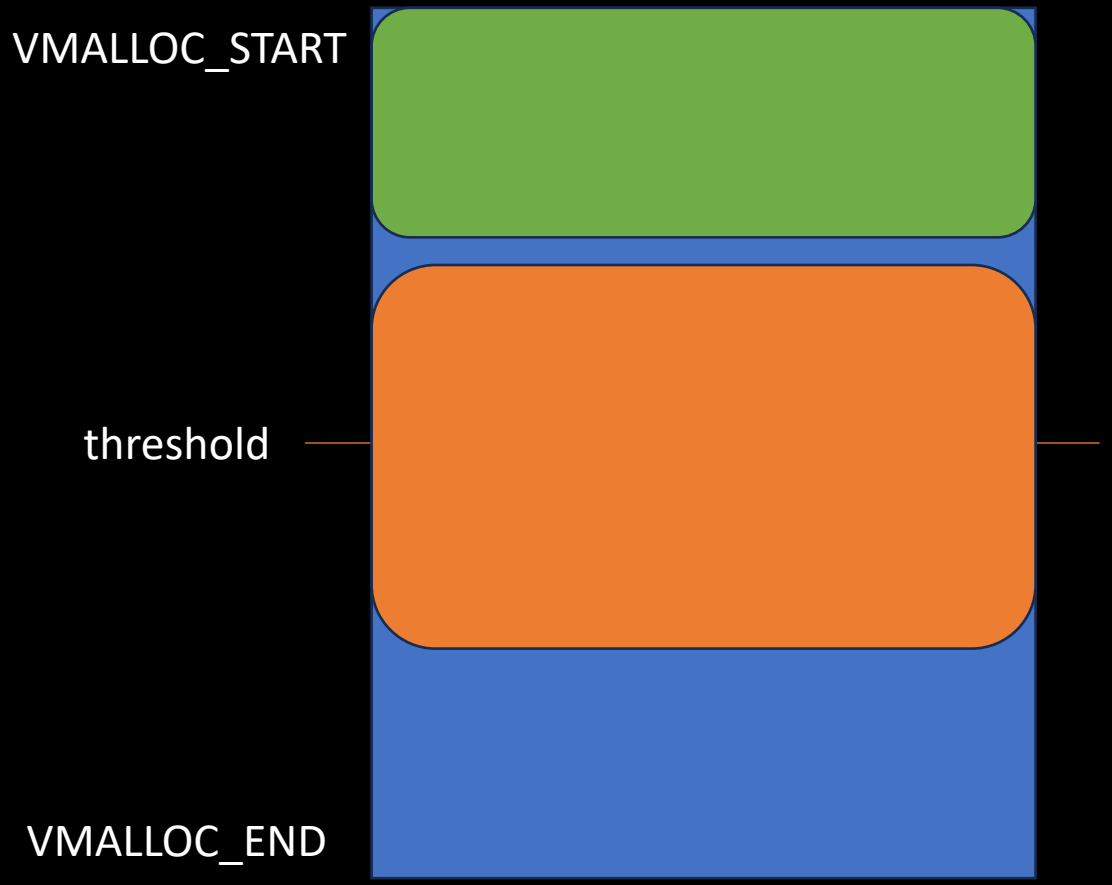
```
0xffffffe51f284000-0xffffffe51f289000   20480 load_module+0x1ecc/0x2554 pages=4 vmalloc
0xffffffe51f289000-0xffffffe51f75d000 5062656 load_module+0x1ecc/0x2554 pages=1235
vmalloc vpages
0xffffffebf498000-0xffffffebf5a0000 1081344 pcpu_get_vm_areas.cfi_jt+0x0/0x8 vmalloc
0xffffffebf5a0000-0xffffffebf6a8000 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

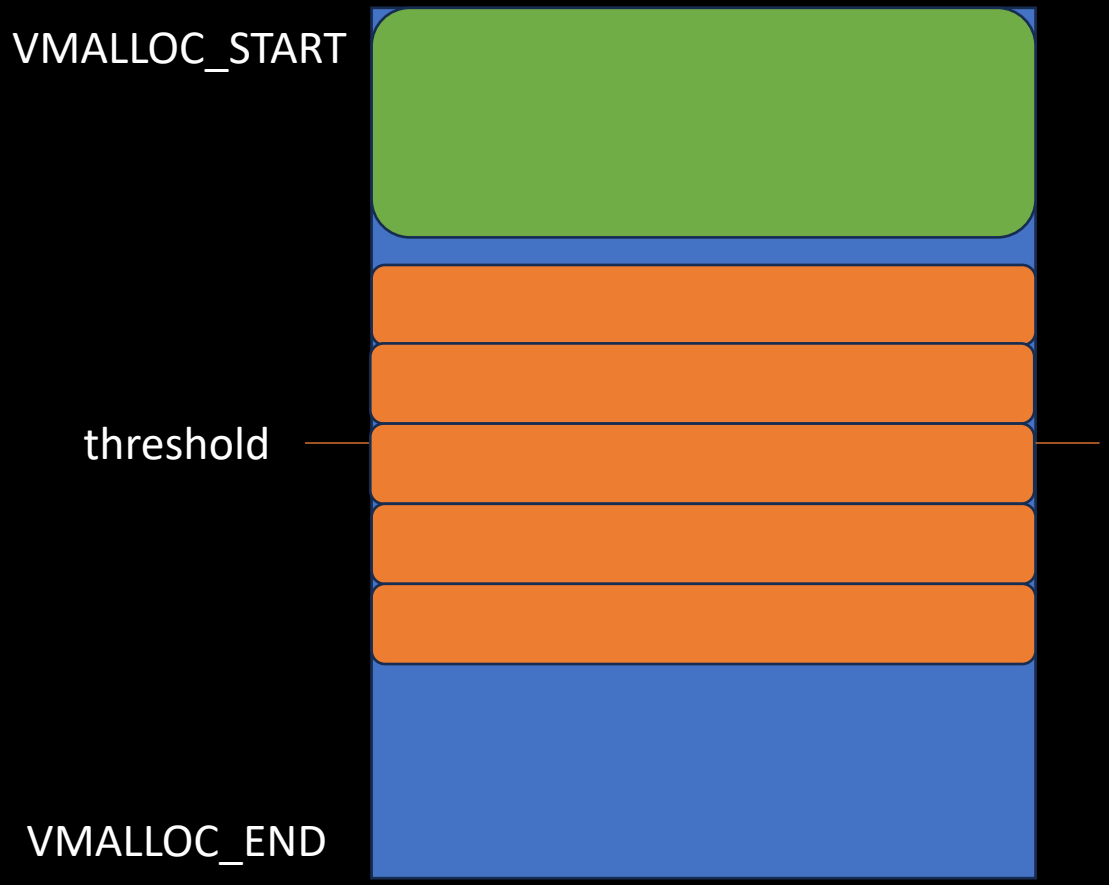




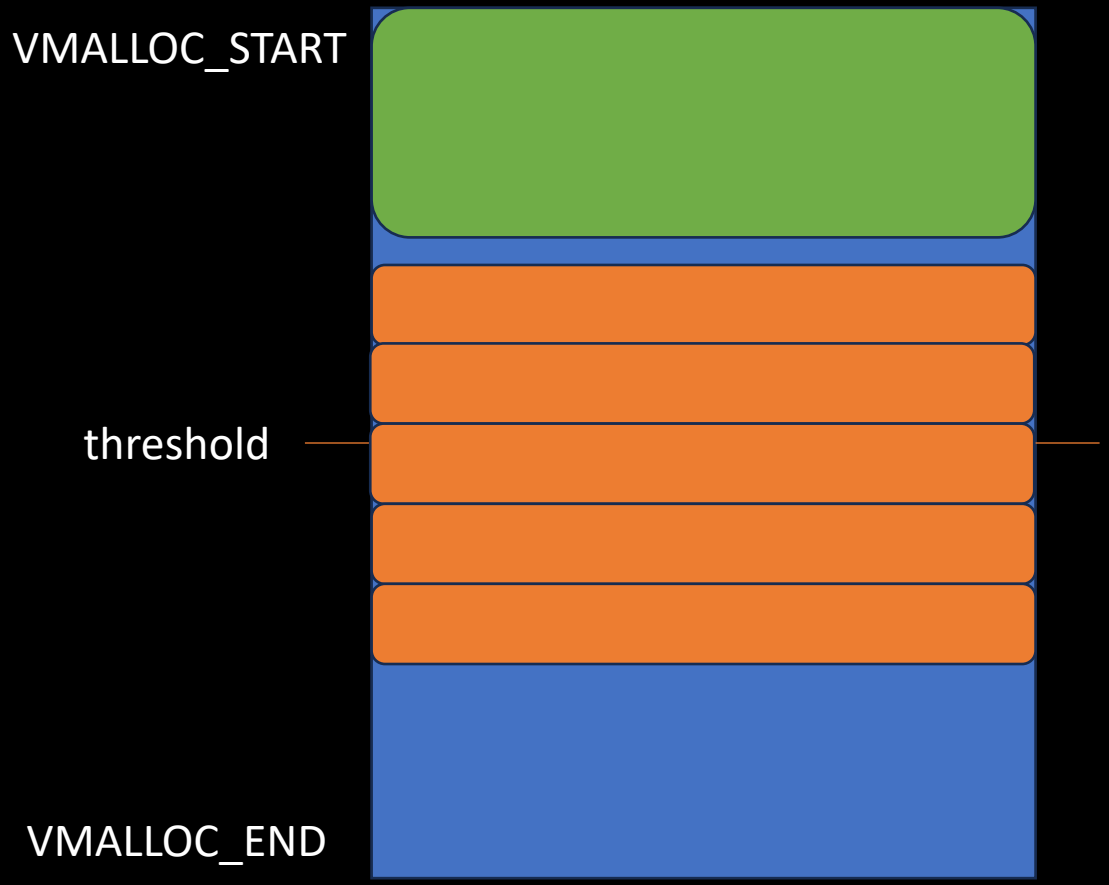
Predict the address



Predict the address



Predict the address



Is it possible to spray many 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

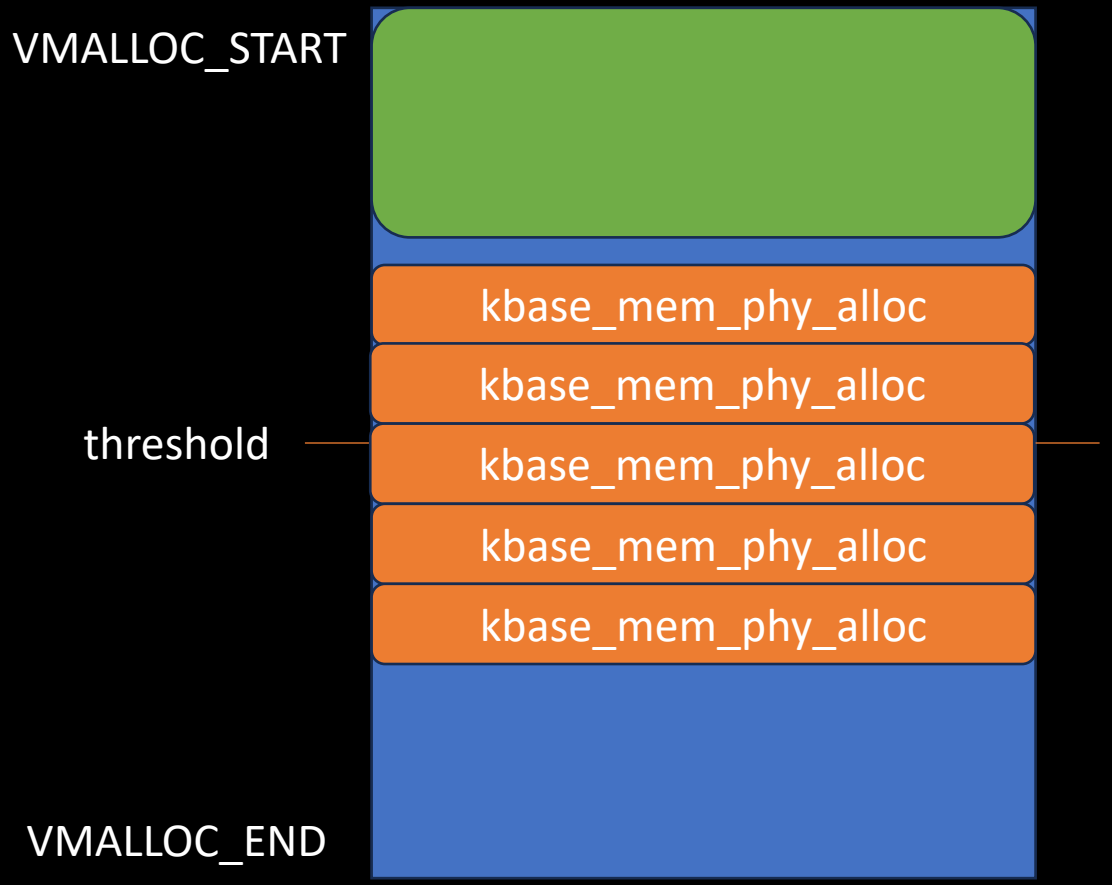
0xffffffc03c80e000-0xffffffc03d80f000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03d80f000-0xffffffc03e810000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03e810000-0xffffffc03f811000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03f811000-0xffffffc040812000 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



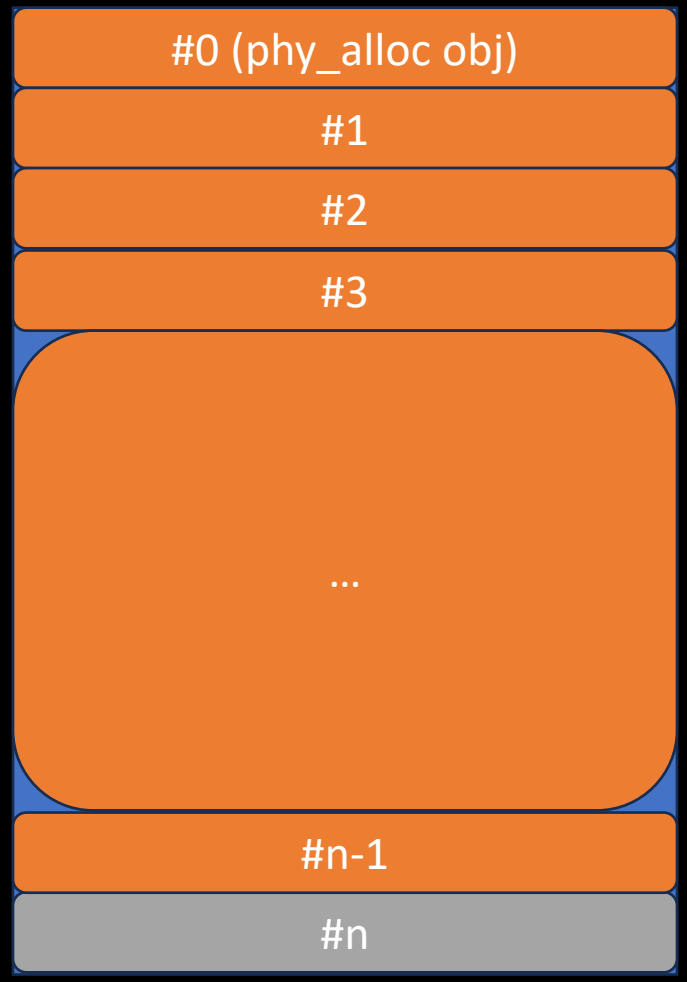
How to find the start address? 🤔

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



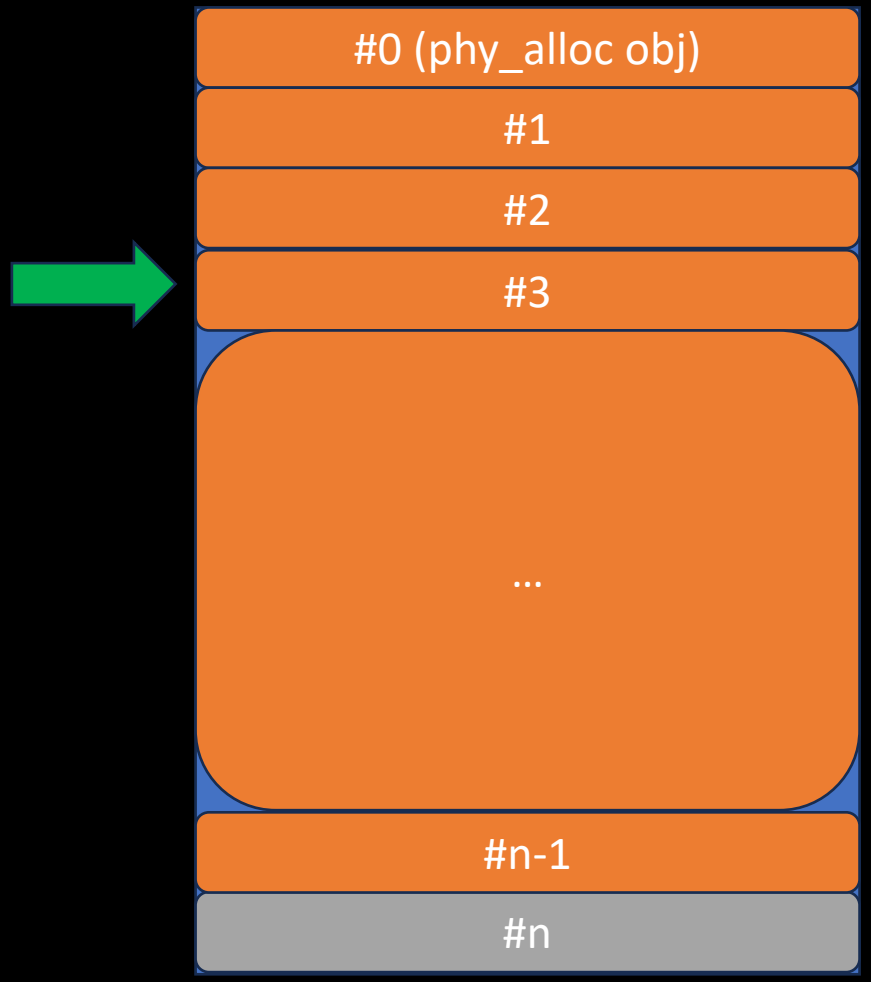
kbase_mem_phy_alloc & page_array

```
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

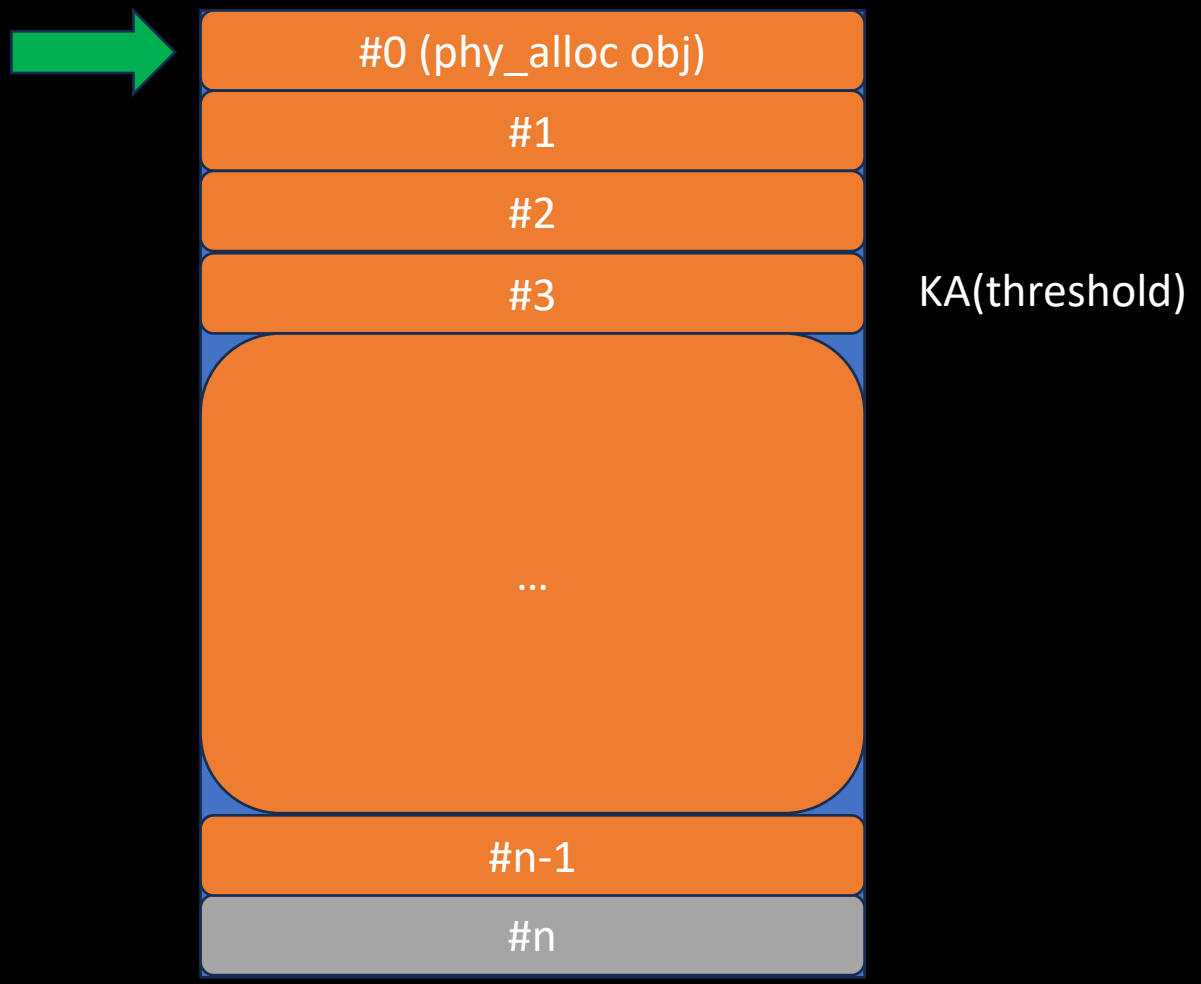
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Search the start address



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

Search the start address



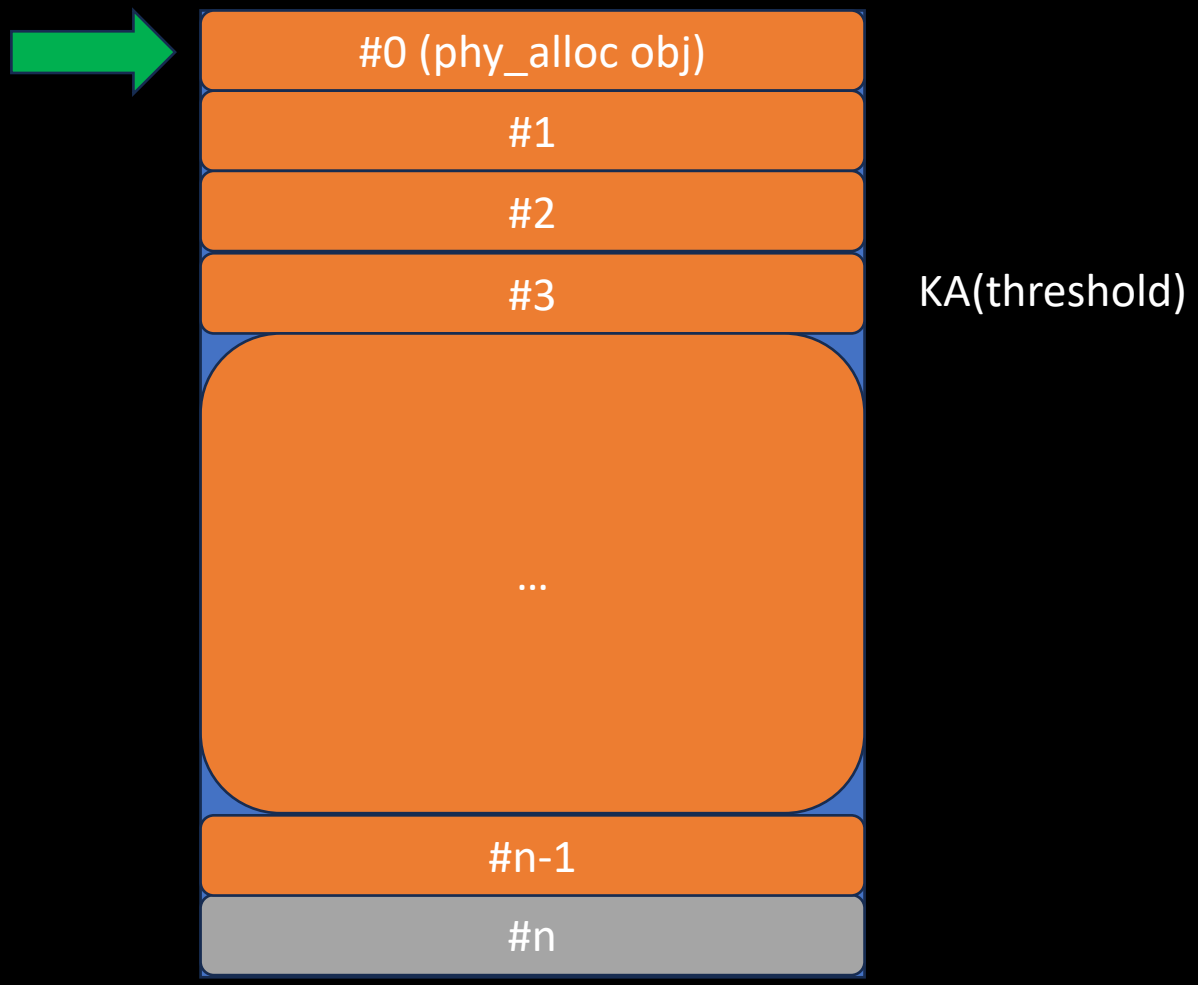
- Trigger the bug
  - KA - 0x1000 * index + nents_offset
- Query the nents
  - 1
- GPU VA Reg_n
  - Precise kernel address
  - Fields are corrupted

Predict the address

- Boot #n

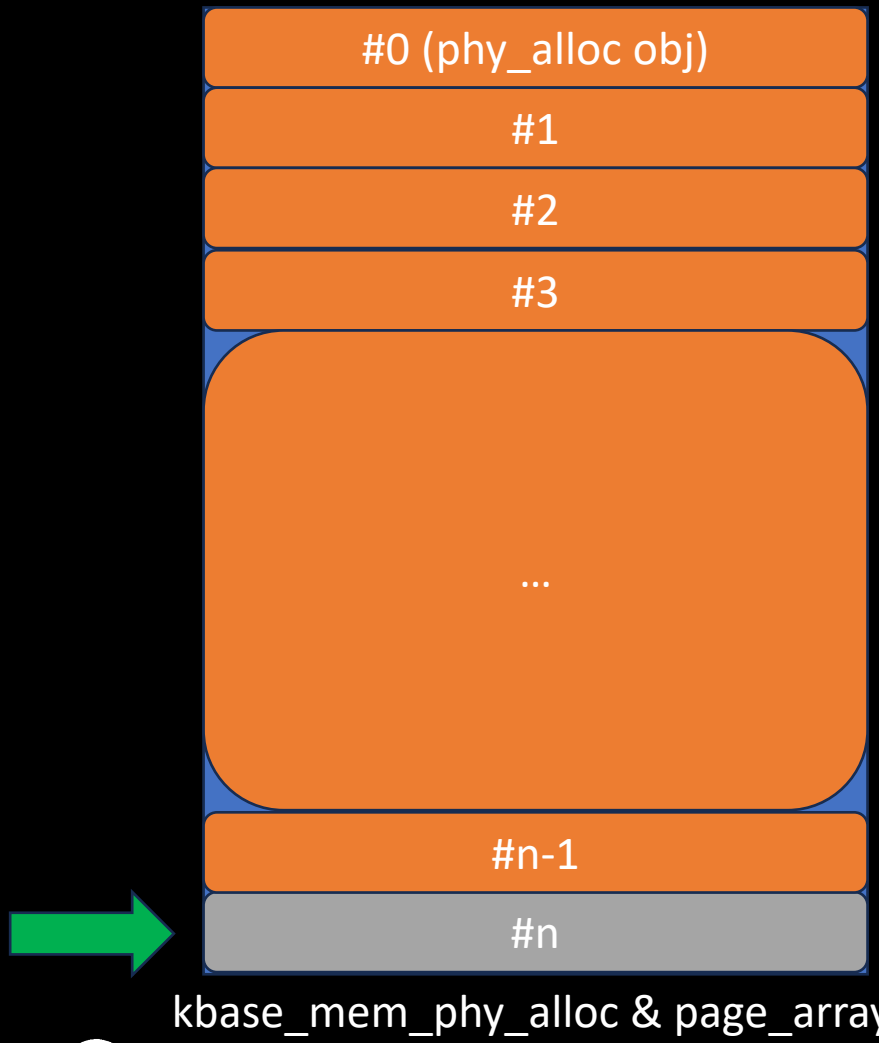
0xffffffc03cc0f000-0xffffffc03dc10000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03dc10000-0xffffffc03ec11000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03ec11000-0xffffffc03fc12000 16781312 kbase_ioctl+0x2a20/0x3430 pages=4096 vmalloc vpages
0xffffffc03fc12000-0xffffffc040c13000 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

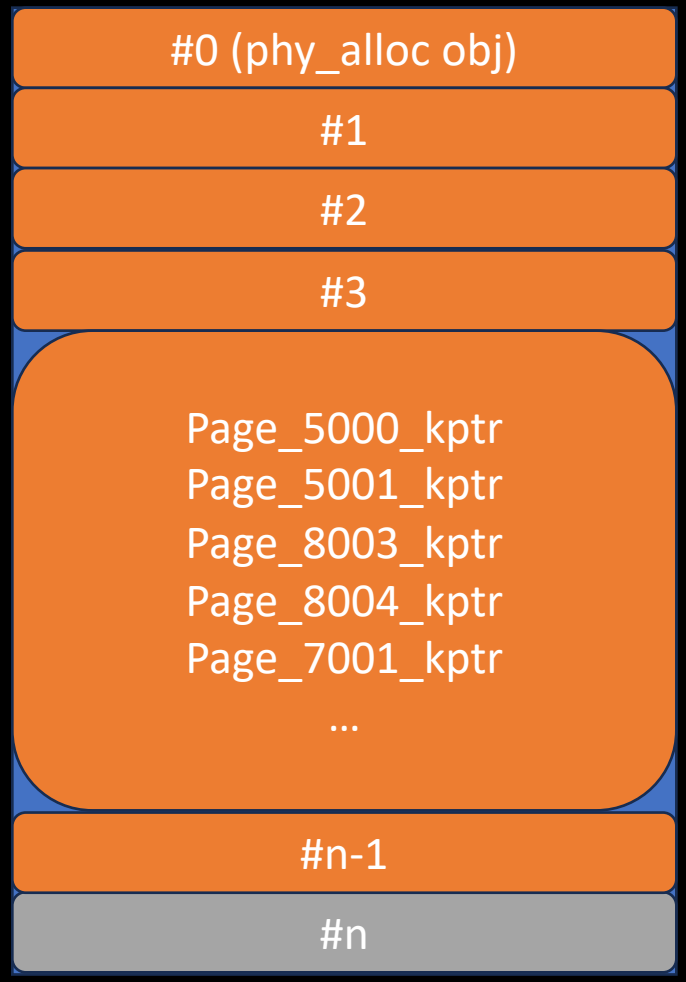
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- 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

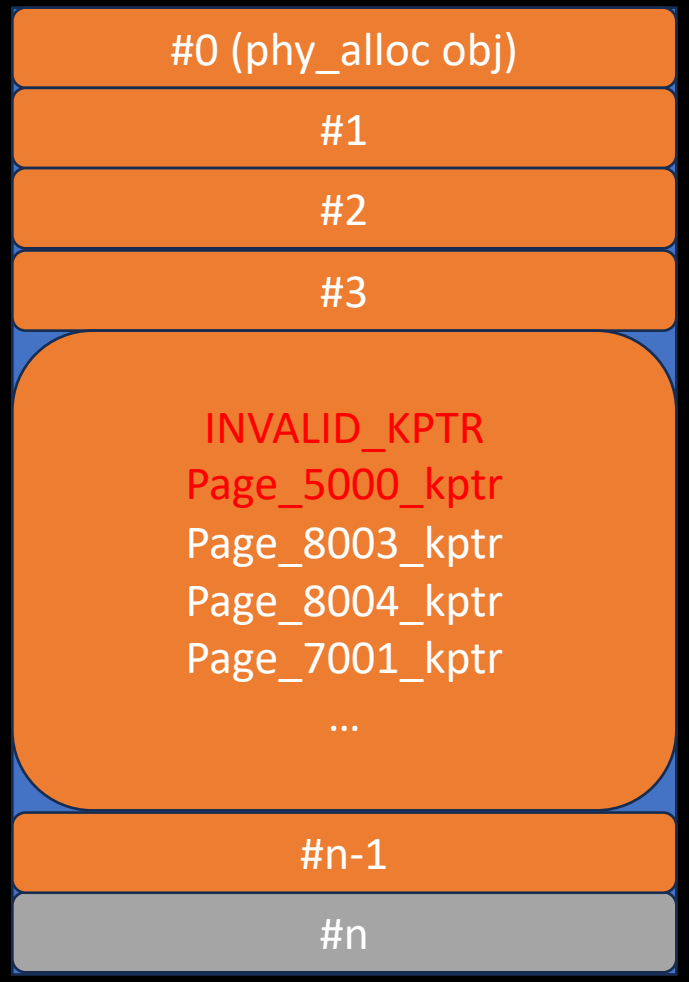
```
0000: 01 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00
0016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0032: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00
0048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

- 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
```

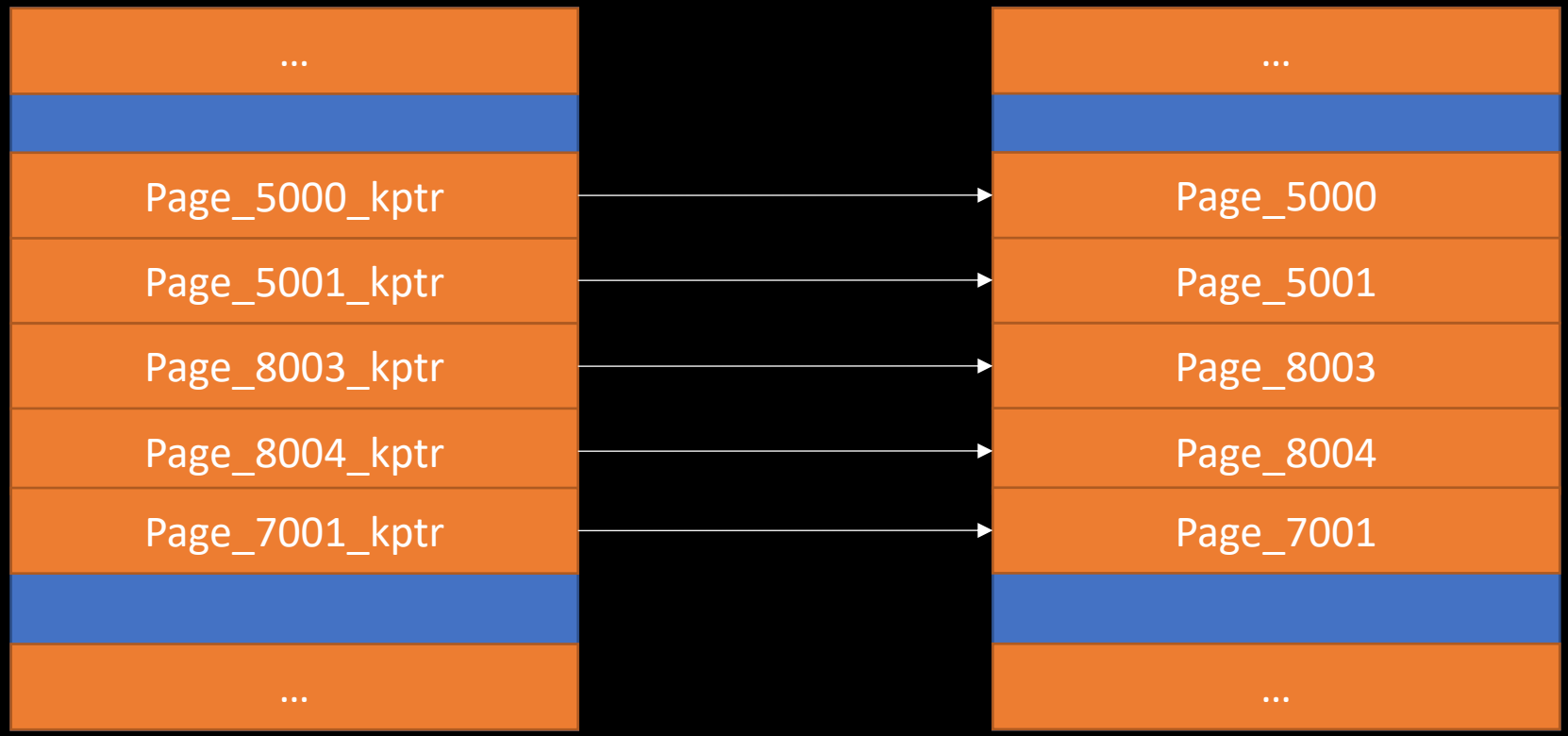
- The last sentinel item

```
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
```

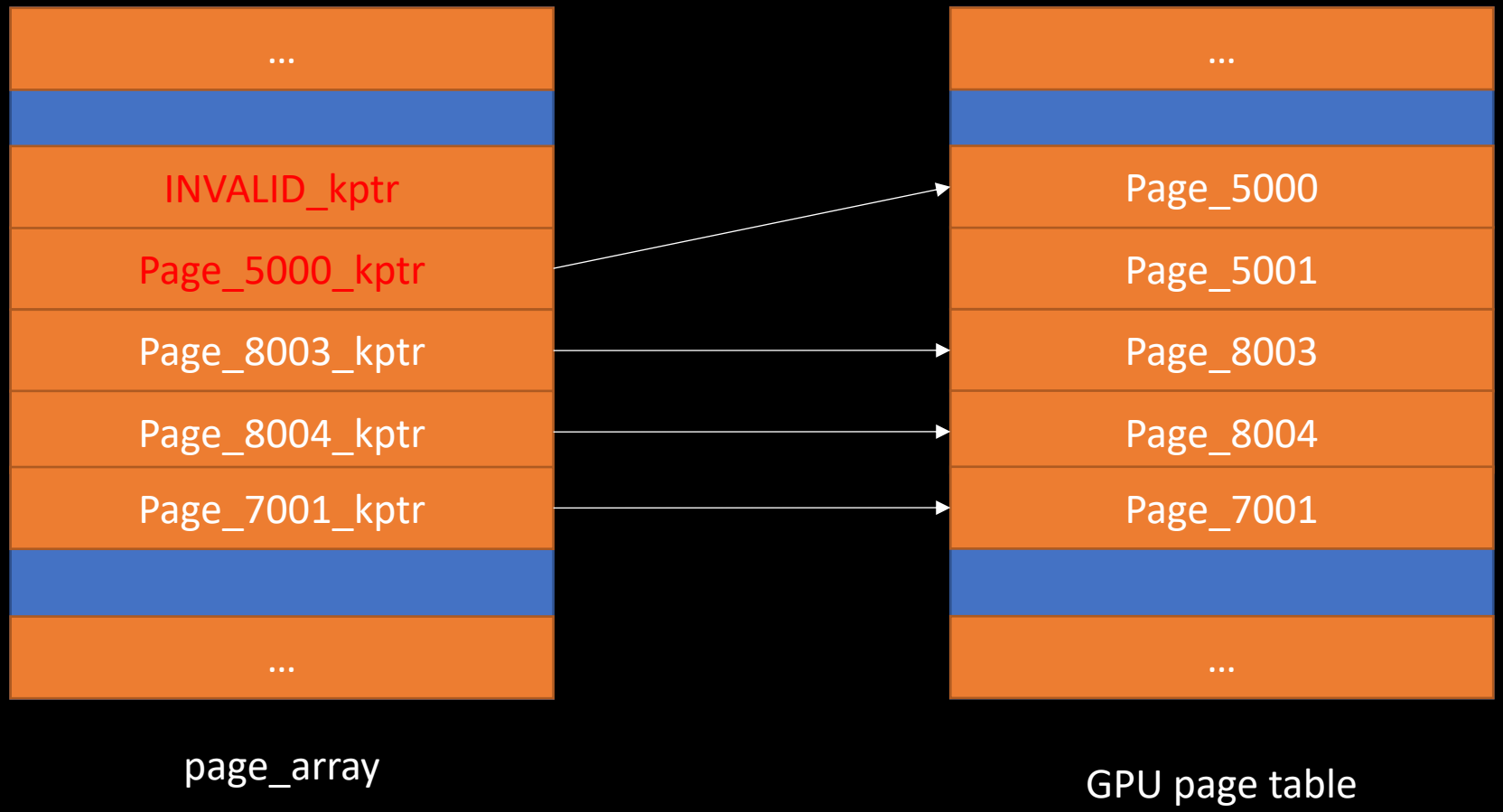


kbase_mem_phy_alloc & page_array

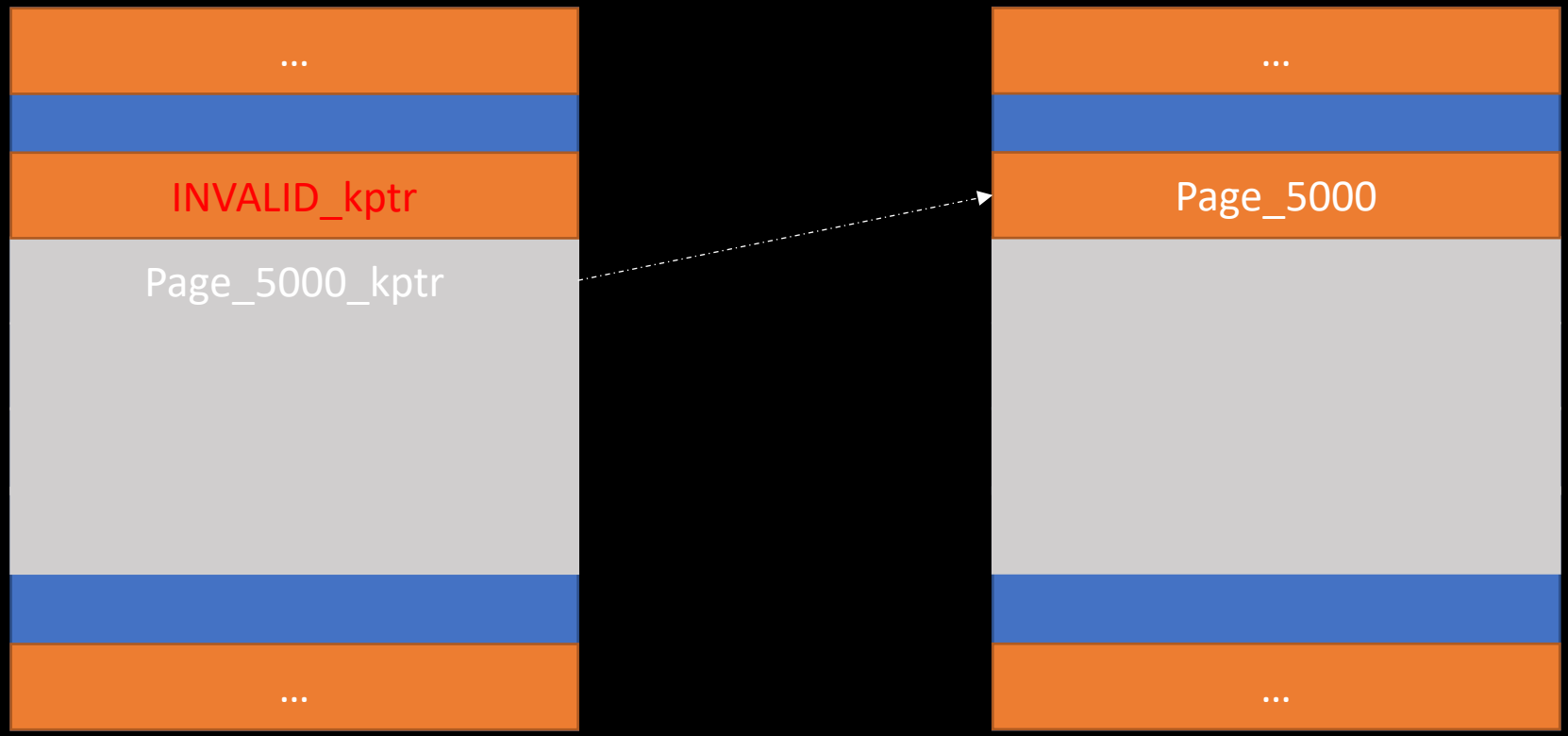
Page UAF in GPU MMU



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)
  - 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

Physical Page Management

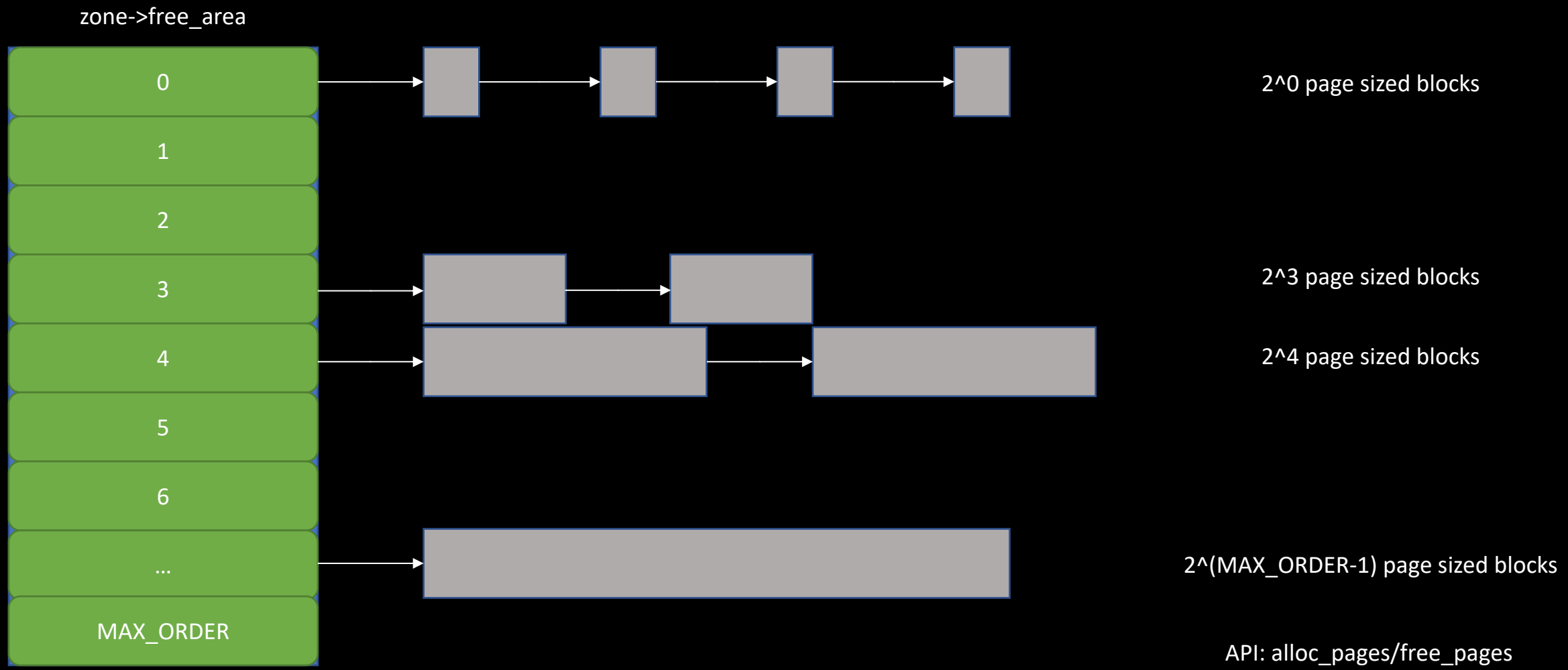
```
struct kbase_mem_pool {
        struct kbase_device *kbdev;
        size_t            cur_size;
        size_t            max_size;
        u8                order;
        u8                group_id;
        spinlock_t        pool_lock;
        struct list_head    page_list;
        struct shrinker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

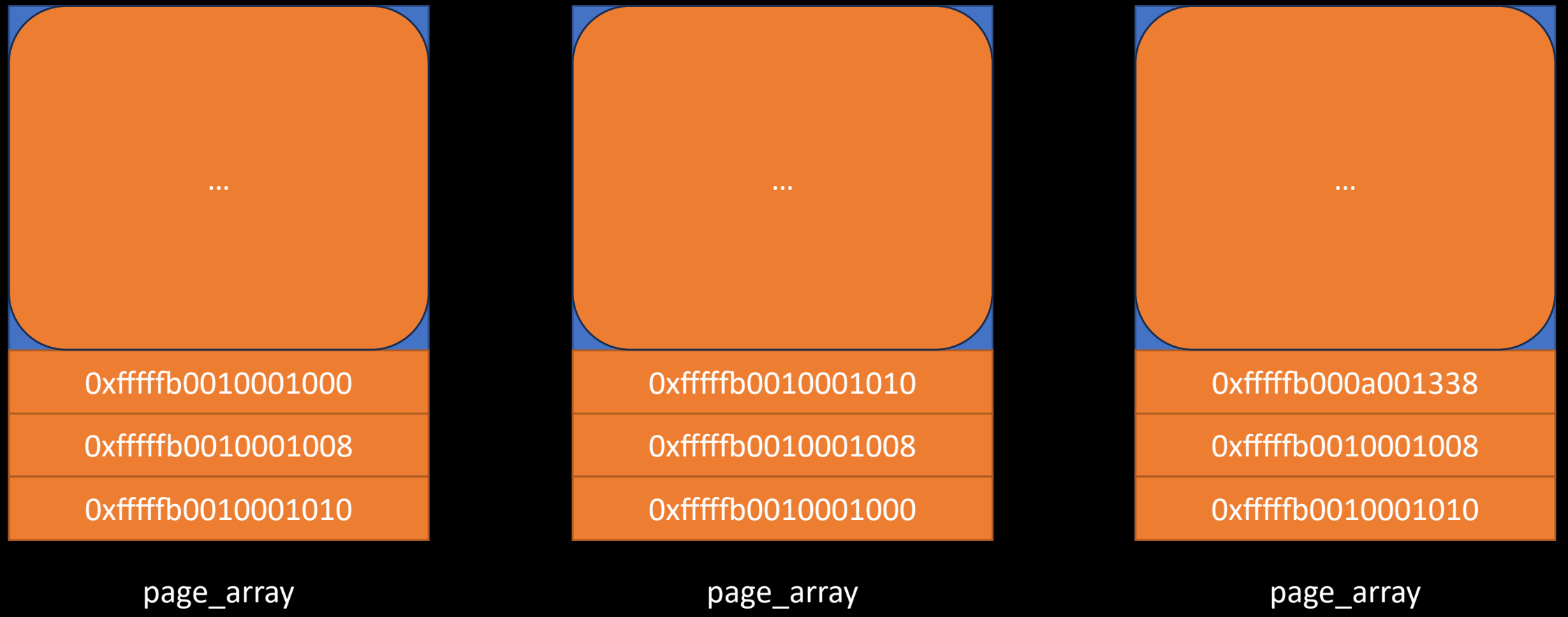
- Allocate
  - Step 1: allocate from the kctx->mem_pools. If insufficient, goto step 2
  - Step 2: allocate from the kbdev->mem_pools. If insufficient, goto step 3
  - Step 3: allocate from the kernel
- Free
  - Step 1: add the pages to kctx->mem_pools. If full, goto step 2
  - Step 2: add the pages 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);

Buddy allocator internal

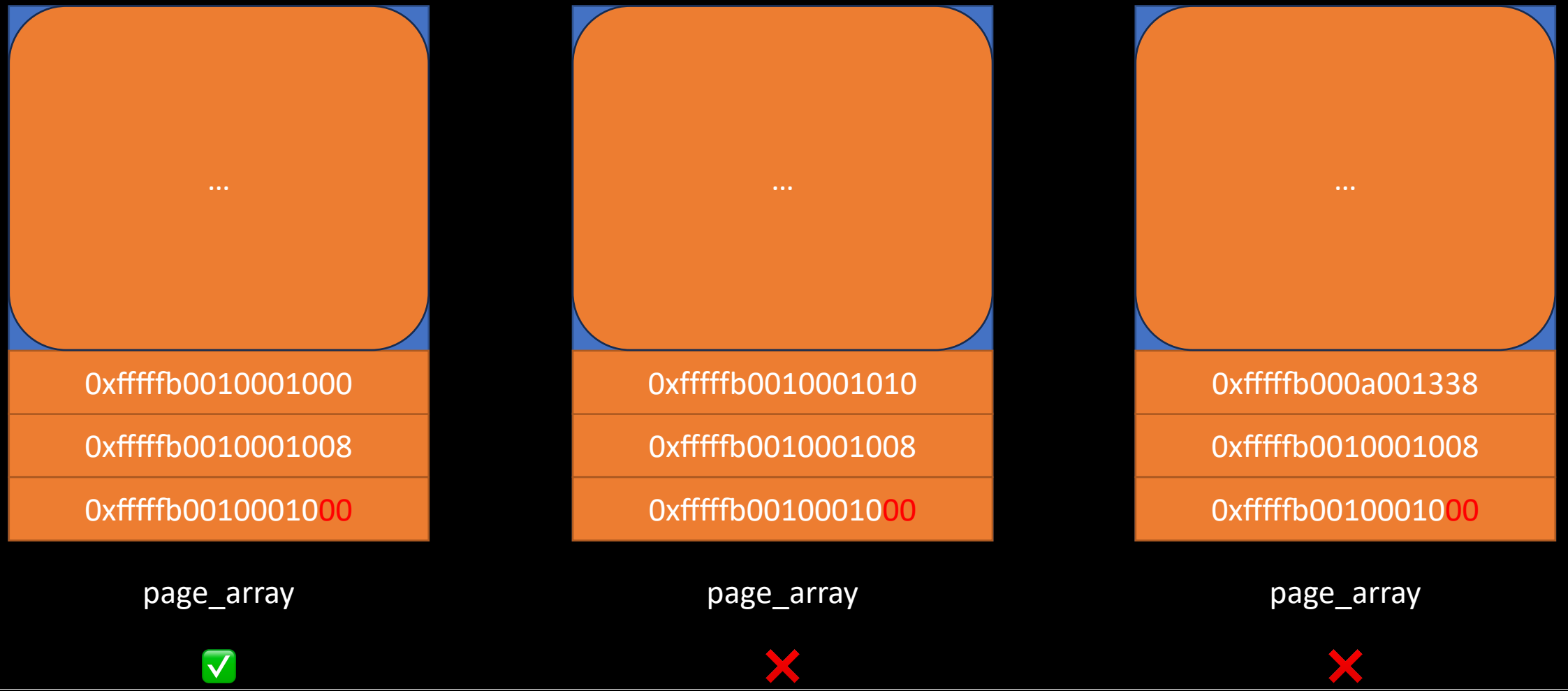


API: alloc_pages/free_pages

Page UAF in GPU MMU



Page UAF in GPU MMU



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

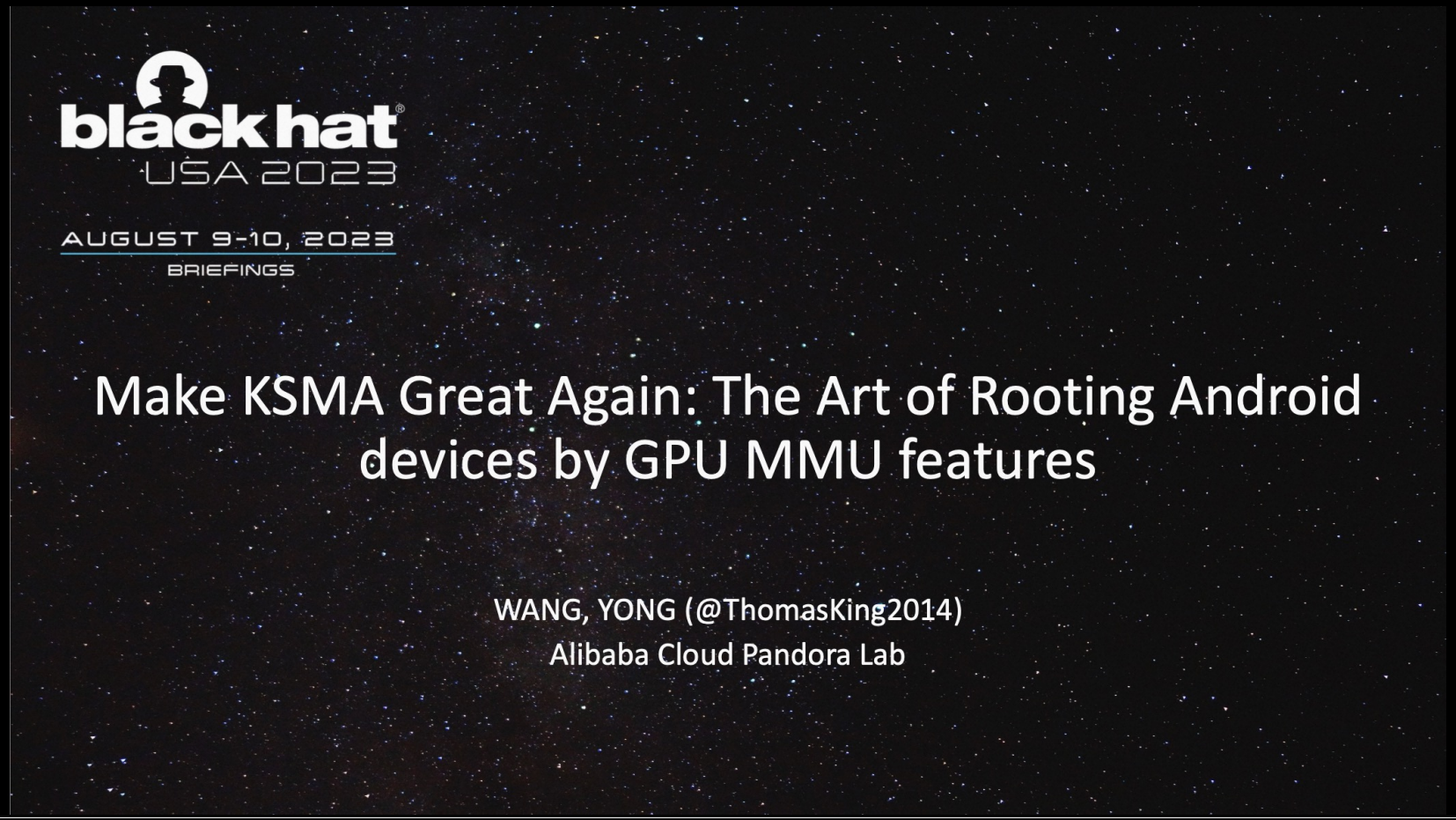
- kbase_mmu_insert_pages_no_flush
  - If invalid, allocate one page as the PGD
  - Allocate from kbdev->mem_pools, not from kctx->mem_pools
  - It’s possible to reuse the freed pages as the PGD

```
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;
```

Page UAF in GPU MMU

- How to craft the valid block entry



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: Compute 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 generic and vendor’s mitigation and gain the root 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 landing, the good quality bugs and more advanced exploitation technique becomes more valuable.

References

[1]https://github.com/externalist/presentations/blob/master/2023%20Zer0con/Mobile%20Exploitation%2C%20the%20past%2C%20present%2C%20and%20future.pdf

[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

WANG, YONG (@ThomasKing2014)
ThomasKingNew@gmail.com