MOSEC

2023 MOBILE SECURITY CONFERENCE

GPU Accelerated Android rooting

WANG, YONG (@ThomasKing2014)
Alibaba Cloud Pandora Lab

Whoami

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

Agenda

- Introduction
- Case study
- Exploitation
- Conclusion

Android kernel mitigations 101

- Android 13 – kernel 5.10(5.15)
  - 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]

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]

Android LPE attack surfaces

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]

Back to 2021

- Why Mali

en.wikipedia.org/wiki/Google_Tensor

Contents [hide]

(Top)

Development

Background

Design

Models

Original

G2

Future

Reception

Beginning in 2017, Google began to include custom-designed co-processors in its Pixel smartphones, namely the Pixel Visual Core on the Pixel 2 and Pixel 3 series and the Pixel Neural Core on the Pixel 4 series.[3][4]

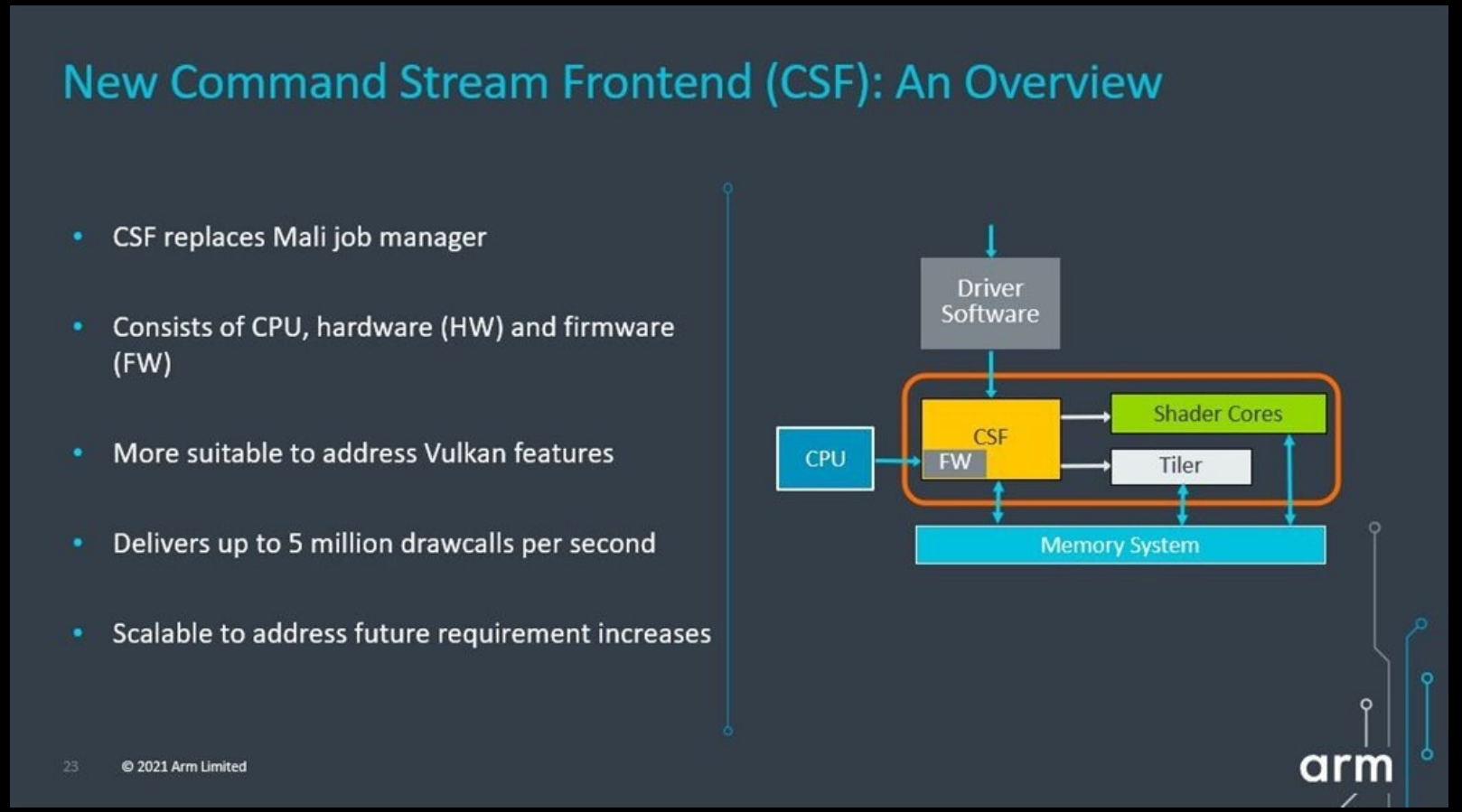
GPU(s)	Mali
Co-processor	Titan

By April 2020, the company had made "significant progress" toward a custom ARM-based processor for its Pixel and Chromebook devices, codenamed "Whitechapel".[5] At Google parent company Alphabet Inc.'s quarterly earnings investor call that October, Pichai expressed excitement at the company's "deeper investments" in hardware, which some interpreted as an allusion to Whitechapel.[6] The Neural Core was not included on the Pixel 5, which was released in 2020; Google explained that the phone's Snapdragon 765G SoC already achieved the camera performance the company had been aiming for.[7] In April 2021, it was reported that Whitechapel would power Google's next Pixel smartphones.[8]

Google officially unveiled the chip, named Tensor, in August, as part of a preview of its Pixel 6 and Pixel 6 Pro smartphones.[9][10] Previous Pixel smartphones had used Qualcomm Snapdragon chips,[11] with 2021's Pixel 5a being the final Pixel phone to do so.[12] Pichai later obliquely noted that the development of Tensor and the Pixel 6 resulted in more off-the-shelf solutions for Pixel phones released in 2020 and early 2021.[1] In September 2022, *The Verge* reported that a Tensor-powered successor to the Pixelbook laptop with a planned 2023 release had been canceled due to cost-cutting measures.[13]

Back to 2021

- Why Mali



Back to 2021

- Why Mali

Android has updated the May security with notes that 4 vulns were exploited in-the-wild.

Qualcomm GPU: CVE-2021-1905, CVE-2021-1906
ARM Mali GPU: CVE-2021-28663, CVE-2021-28664

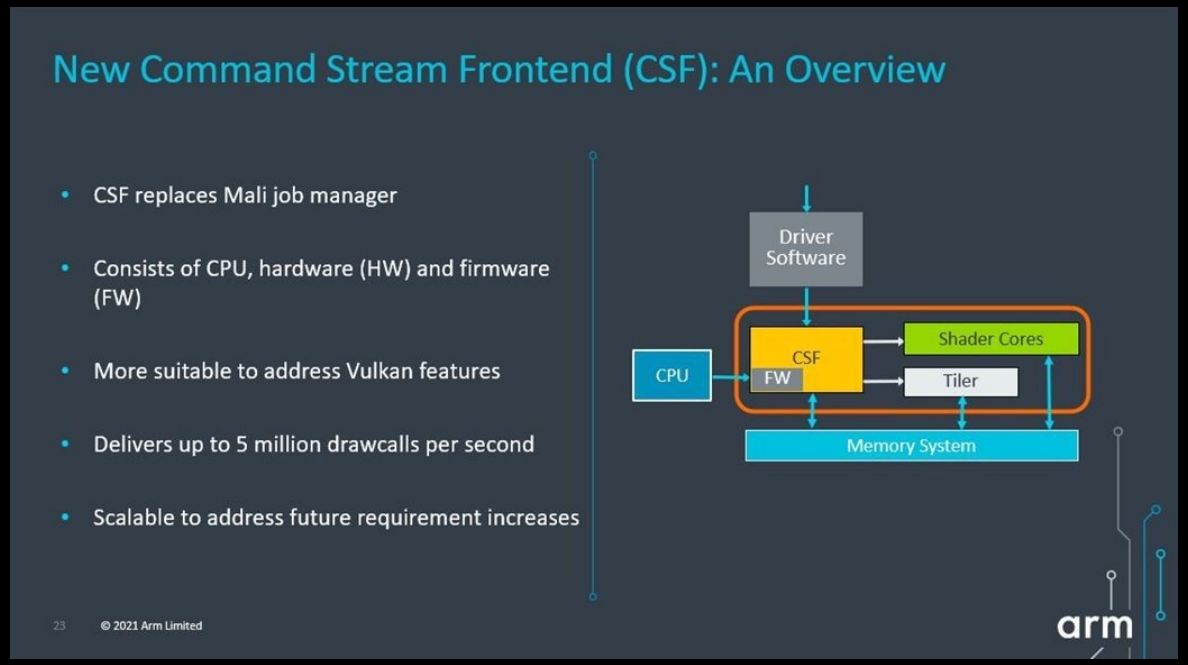
source.android.com/security/bulle...
翻译推文

下午9:12 · 2021年5月19日

Agenda

- Introduction
- *Case study*
- Exploitation
- Conclusion

Non-CSF vs CSF




```
1653        kctx = kbase_file_get_kctx_if_setup_complete(kfile);
1654        if (unlikely(!kctx))
1655            return -EPERM;
1656
1657        /* Normal ioctls */
1658        switch (cmd) {
1659    #if !MALI_USE_CSF
1660        case KBASE_IOCTL_JOB_SUBMIT:
1661            KBASE_HANDLE_IOCTL_IN(KBASE_IOCTL_JOB_SUBMIT,
1662                    kbase_api_job_submit,
1663                    struct kbase_ioctl_job_submit,
1664                    kctx);
1665            break;
1666    #endif /* !MALI_USE_CSF */
1667        case KBASE_IOCTL_GET_GPUPROPS:
1668            KBASE_HANDLE_IOCTL_IN(KBASE_IOCTL_GET_GPUPROPS,
1669                    kbase_api_get_gpuprops,
1670                    struct kbase_ioctl_get_gpuprops,
1671                    kctx);
1672            break;
1673    #if !MALI_USE_CSF
1674        case KBASE_IOCTL_POST_TERM:
1675            KBASE_HANDLE_IOCTL(KBASE_IOCTL_POST_TERM,
1676                    kbase_api_post_term,
1677                    kctx);
```

MTK dimensity 9000(Redmi K50P)

CVE-2021-28664

Title	**Mali GPU Kernel Driver elevates CPU RO pages to writable**
CVE	CVE-2021-28664
Date of issue	18th March 2021
Affects	• Midgard GPU Kernel Driver: All versions from r8p0 – r30p0 • Bifrost GPU Kernel Driver: All versions from r0p0 – r29p0 • Valhall GPU Kernel Driver: All versions from r19p0 - r29p0
Impact	A non-privileged user can get a write access to read-only memory, and may be able to gain root privilege, corrupt memory and modify the memory of other processes.
Resolution	This issue is fixed in Bifrost and Valhall GPU Kernel Driver r30p0 and in Midgard GPU Kernel Driver r31p0 release. Users are recommended to upgrade if they are impacted by this issue.
Credit	n/a

https://developer.arm.com/Arm%20Security%20Center/Mali%20GPU%20Driver%20Vulnerabilities

CVE-2021-28664

- kbase_mem_from_user_buffer diff(Bifrost r28 vs r29)

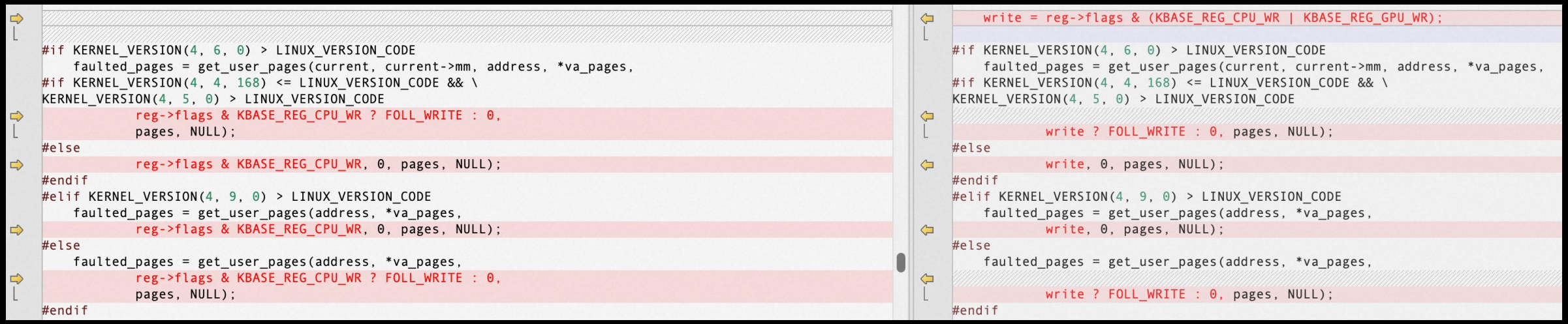
```
#if LINUX_VERSION_CODE < KERNEL_VERSION(4, 6, 0)
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_GPU_WR, 0, pages, NULL);
#endif
#elif LINUX_VERSION_CODE < KERNEL_VERSION(4, 9, 0)
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_GPU_WR, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#endif
```

```
#if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#endif
#elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#endif
```

- Only GPU_WR permission check
  - CPU_WR instead of GPU_WR

CVE-2021-28664

- kbase_mem_from_user_buffer diff(Bifrost r29 vs r30)



```
#if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#endif
#elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#endif
```

```
    write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);

#if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            write ? FOLL_WRITE : 0, pages, NULL);
#else
            write, 0, pages, NULL);
#endif
#elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(address, *va_pages,
            write, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            write ? FOLL_WRITE : 0, pages, NULL);
#endif
```

- Check both CPU_WR and GPU_WR

CVE-2021-28664

```
static int kbase_jd_user_buf_map(struct kbase_context *kctx,
        struct kbase_va_region *reg)
{
    long pinned_pages;
    struct kbase_mem_phy_alloc *alloc;
    struct page **pages;
    struct tagged_addr *pa;
    long i;
    unsigned long address;
    struct device *dev;
    unsigned long offset;
    unsigned long local_size;
    unsigned long gwt_mask = ~0;
    int err = kbase_jd_user_buf_pin_pages(kctx, reg);

    if (err)
        return err;

    alloc = reg->gpu_alloc;
    pa = kbase_get_gpu_phy_pages(reg);
    address = alloc->imported.user_buf.address;
    pinned_pages = alloc->nents;
    pages = alloc->imported.user_buf.pages;
    dev = kctx->kbdev->dev;
    offset = address & ~PAGE_MASK;
    local_size = alloc->imported.user_buf.size;
```

```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
        struct kbase_va_region *reg)
{
    struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
    struct page **pages = alloc->imported.user_buf.pages;
    unsigned long address = alloc->imported.user_buf.address;
    struct mm_struct *mm = alloc->imported.user_buf.mm;
    long pinned_pages;
    long i;

    if (WARN_ON(alloc->type != KBASE_MEM_TYPE_IMPORTED_USER_BUF))
        return -EINVAL;

    if (alloc->nents) {
        if (WARN_ON(alloc->nents != alloc->imported.user_buf.nr_pages))
            return -EINVAL;
        else
            return 0;
    }

    if (WARN_ON(reg->gpu_alloc->imported.user_buf.mm != current->mm))
        return -EINVAL;

#if LINUX_VERSION_CODE < KERNEL_VERSION(4, 6, 0)
    pinned_pages = get_user_pages(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_GPU_WR,
            0, pages, NULL);
#endif
#elif LINUX_VERSION_CODE < KERNEL_VERSION(4, 9, 0)
    pinned_pages = get_user_pages_remote(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
            reg->flags & KBASE_REG_GPU_WR,
            0, pages, NULL);
#elif LINUX_VERSION_CODE < KERNEL_VERSION(4, 10, 0)
    pinned_pages = get_user_pages_remote(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
```

Insufficient fix

```
int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
        struct kbase_va_region *reg)
{
    struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
    struct page **pages = alloc->imported.user_buf.pages;
    unsigned long address = alloc->imported.user_buf.address;
    struct mm_struct *mm = alloc->imported.user_buf.mm;
    long pinned_pages;
    long i;

    if (WARN_ON(alloc->type != KBASE_MEM_TYPE_IMPORTED_USER_BUF))
        return -EINVAL;

    if (alloc->nents) {
        if (WARN_ON(alloc->nents != alloc->imported.user_buf.nr_pages))
            return -EINVAL;
        else
            return 0;
    }

    if (WARN_ON(reg->gpu_alloc->imported.user_buf.mm != current->mm))
        return -EINVAL;

#if LINUX_VERSION_CODE < KERNEL_VERSION(4, 6, 0)
    pinned_pages = get_user_pages(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_GPU_WR,
            0, pages, NULL);
#endif
#elif LINUX_VERSION_CODE < KERNEL_VERSION(4, 9, 0)
    pinned_pages = get_user_pages_remote(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
            reg->flags & KBASE_REG_GPU_WR,
            0, pages, NULL);
#elif LINUX_VERSION_CODE < KERNEL_VERSION(4, 10, 0)
    pinned_pages = get_user_pages_remote(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
```

Import again

```
#if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#else
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#endif
#elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            reg->flags & KBASE_REG_CPU_WR ? FOLL_WRITE : 0,
            pages, NULL);
#endif
```

```
    write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);

#if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(current, current->mm, address, *va_pages,
#if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
            write ? FOLL_WRITE : 0, pages, NULL);
#else
            write, 0, pages, NULL);
#endif
#elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
    faulted_pages = get_user_pages(address, *va_pages,
            write, 0, pages, NULL);
#else
    faulted_pages = get_user_pages(address, *va_pages,
            write ? FOLL_WRITE : 0, pages, NULL);
#endif
```

- It sounds like double fetch
  - KBASE_IOCTL_MEM_IMPORT: just touch the user memory
  - KBASE_IOCTL_JOB_SUBMIT: import the physical pages

PoC

- 1. Mmap the Read/Write anonymous memory (CPU_VA1)
- 2. Import the CPU memory with BASE_MEM_PROT_CPU_WR
- 3. Mmap the GPU memory (CPU_VA2)
- 4. Munmap the CPU_VA1
- 5. Fixedly mmap the Read-Only memory (CPU_VA1)
- 6. Submit a JOB with BASE_JD_REQ_EXTERNAL_RESOURCES (CPU_VA2, same VA)
- 7. Write the CPU_VA1 via CPU_VA2
  - Wait a moment for kbase_jd_user_buf_pin_pages to be called

Fix

CVE-2022-22706 / CVE-2021-39793: Mali GPU driver makes read-only imported pages host-writable

Jann Horn

The Basics

Disclosure or Patch Date: March 7, 2022

Product: Arm Mali GPU driver for Linux/Android

Advisory:

- from Arm (upstream): https://developer.arm.com/Arm%20Security%20Center/Mali%20GPU%20Driver%20Vulnerabilities
- from Google Pixel: https://source.android.com/security/bulletin/pixel/2022-03-01#pixel

Affected Versions: see Arm advisory (note that the affected version range for the Bifrost version of the related CVE-2021-28664 seems to be off-by-one)

First Patched Version:

- for Arm: see Arm advisory
- for Pixel: patch level 2022-03-05

Issue/Bug Report: N/A

Patch CL: https://android.googlesource.com/kernel/google-modules/gpu/+/5381ff7b4106b277ff207396e293ede2bf959f0c%5E%21/

https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2021/CVE-2021-39793.html

Story continues

- 1. Mmap the Read/Write anonymous memory (CPU_VA1)
- 2. Import the CPU memory with BASE_MEM_PROT_CPU_WR
- 3. Mmap the GPU memory (CPU_VA2)
- 4. Munmap the CPU_VA1
- 5. Fixedly mmap the Read-Only memory (CPU_VA1)
- 6. Submit a JOB with BASE_JD_REQ_EXTERNAL_RESOURCES (CPU_VA2, same VA)
- 7. Write the CPU_VA1 via CPU_VA2
  - Sleep(5) == Always SIGBUS

Story continues

- No physical pages, why?🤔

```
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 vm_fault_t kbase_cpu_vm_fault(struct vm_fault *vmf)
{
    struct vm_area_struct *vma = vmf->vma;
    struct kbase_cpu_mapping *map = vma->vm_private_data;
    pgoff_t map_start_pgoff;
    pgoff_t fault_pgoff;
    size_t i;
    pgoff_t addr;
    size_t nents;
    struct tagged_addr *pages;
    vm_fault_t ret = VM_FAULT_SIGBUS;
    struct memory_group_manager_device *mgm_dev;

    KBASE_DEBUG_ASSERT(map);
    KBASE_DEBUG_ASSERT(map->count > 0);
    KBASE_DEBUG_ASSERT(map->kctx);
    KBASE_DEBUG_ASSERT(map->alloc);

    map_start_pgoff = vma->vm_pgoff - map->region->start_pfn;

    kbase_gpu_vm_lock(map->kctx);
    if (unlikely(map->region->cpu_alloc->type == KBASE_MEM_TYPE_ALIAS)) {
        struct kbase_aliased *aliased =
                get_aliased_alloc(vma, map->region, &map_start_pgoff, 1);

        if (!aliased)
            goto exit;

        nents = aliased->length;
        pages = aliased->alloc->pages + aliased->offset;
    } else  {
        nents = map->alloc->nents;
        pages = map->alloc->pages;
    }

    fault_pgoff = map_start_pgoff + (vmf->pgoff - vma->vm_pgoff);

    if (fault_pgoff >= nents)
        goto exit;
```

Story continues

```
static void kbase_jd_post_external_resources(struct kbase_jd_atom *katom)
{
    KBASE_DEBUG_ASSERT(katom);
    KBASE_DEBUG_ASSERT(katom->core_req & BASE_JD_REQ_EXTERNAL_RESOURCES);

#ifdef CONFIG_MALI_DMA_FENCE
    kbase_dma_fence_signal(katom);
#endif /* CONFIG_MALI_DMA_FENCE */

    kbase_gpu_vm_lock(katom->kctx);
    /* only roll back if extres is non-NULL */
    if (katom->extres) {
        u32 res_no;

        res_no = katom->nr_extres;
        while (res_no-- > 0) {
            struct kbase_mem_phy_alloc *alloc = katom->extres[res_no].alloc;
            struct kbase_va_region *reg;

            reg = kbase_region_tracker_find_region_base_address(
                    katom->kctx,
                    katom->extres[res_no].gpu_address);
            kbase_unmap_external_resource(katom->kctx, reg, alloc);
        }
        kfree(katom->extres);
        katom->extres = NULL;
    }
    kbase_gpu_vm_unlock(katom->kctx);
}
```

```
case KBASE_MEM_TYPE_IMPORTED_USER_BUF: {
    alloc->imported.user_buf.current_mapping_usage_count--;

    if (0 == alloc->imported.user_buf.current_mapping_usage_count)
        bool writeable = true;

        if (!kbase_is_region_invalid_or_free(reg) &&
                reg->gpu_alloc == alloc)
            kbase_mmu_teardown_pages(
                    kctx->kbdev,
                    &kctx->mmu,
                    reg->start_pfn,
                    kbase_reg_current_backed_size(reg),
                    kctx->as_nr);

        if (reg && ((reg->flags & KBASE_REG_GPU_WR) == 0))
            writeable = false;

        kbase_jd_user_buf_unmap(kctx, alloc, writeable);
    }
}
```

```
static void kbase_jd_user_buf_unmap(struct kbase_context *kctx,
        struct kbase_mem_phy_alloc *alloc, bool writeable)
{
    long i;
    struct page **pages;
    unsigned long size = alloc->imported.user_buf.size;

    KBASE_DEBUG_ASSERT(alloc->type == KBASE_MEM_TYPE_IMPORTED_USER_BUF);
    pages = alloc->imported.user_buf.pages;
    for (i = 0; i < alloc->imported.user_buf.nr_pages; i++) {
        unsigned long local_size;
        dma_addr_t dma_addr = alloc->imported.user_buf.dma_addrs[i];

        local_size = MIN(size, PAGE_SIZE - (dma_addr & ~PAGE_MASK));
        dma_unmap_page(kctx->kbdev->dev, dma_addr, local_size,
                DMA_BIDIRECTIONAL);
        if (writeable)
            set_page_dirty_lock(pages[i]);
#if !MALI_USE_CSF
        put_page(pages[i]);
        pages[i] = NULL;
#endif

        size -= local_size;
    }
#if !MALI_USE_CSF
    alloc->nents = 0;
#endif
}
```

- Physical pages will be released when the JOB is finished
- Trigger the VM_FAULT before the pages are released

Story continues

```c
static void kbase_jd_user_buf_unmap(struct kbase_context *kctx,
        struct kbase_mem_phy_alloc *alloc, bool writeable)
{
    long i;
    struct page **pages;
    unsigned long size = alloc->imported.user_buf.size;

    KBASE_DEBUG_ASSERT(alloc->type == KBASE_MEM_TYPE_IMPORTED_USER_BUF);
    pages = alloc->imported.user_buf.pages;
    for (i = 0; i < alloc->imported.user_buf.nr_pages; i++) {
        unsigned long local_size;
        dma_addr_t dma_addr = alloc->imported.user_buf.dma_addrs[i];

        local_size = MIN(size, PAGE_SIZE - (dma_addr & ~PAGE_MASK));
        dma_unmap_page(kctx->kbdev->dev, dma_addr, local_size,
                DMA_BIDIRECTIONAL);
        if (writeable)
            set_page_dirty_lock(pages[i]);
#if !MALI_USE_CSF
        put_page(pages[i]);
        pages[i] = NULL;
#endif

        size -= local_size;
    }
#if !MALI_USE_CSF
    alloc->nents = 0;
#endif
}
```

```c
int kbase_mem_shrink(struct kbase_context *const kctx,
        struct kbase_va_region *const reg, u64 const new_pages)
{
    u64 delta, old_pages;
    int err;

    lockdep_assert_held(&kctx->reg_lock);

    if (WARN_ON(!kctx))
        return -EINVAL;

    if (WARN_ON(!reg))
        return -EINVAL;

    old_pages = kbase_reg_current_backed_size(reg);
    if (WARN_ON(old_pages < new_pages))
        return -EINVAL;

    delta = old_pages - new_pages;

    /* Update the GPU mapping */
    err = kbase_mem_shrink_gpu_mapping(kctx, reg,
            new_pages, old_pages);
    if (err >= 0) {
        /* Update all CPU mapping(s) */
        kbase_mem_shrink_cpu_mapping(kctx, reg,
                new_pages, old_pages);

        kbase_free_phy_pages_helper(reg->cpu_alloc, delta);
        if (reg->cpu_alloc != reg->gpu_alloc)
            kbase_free_phy_pages_helper(reg->gpu_alloc, delta);
    }

    return err;
}
```

- The CPU mapping has not been handled 😊
- The imported pages can be freed and reclaimed

Fix

```
 */
static void kbase_jd_user_buf_unmap(struct kbase_context *kctx,
        struct kbase_mem_phy_alloc *alloc, bool writeable)
{
    long i;
    struct page **pages;

    unsigned long size = alloc->imported.user_buf.size;



    KBASE_DEBUG_ASSERT(alloc->type == KBASE_MEM_TYPE_IMPORTED_USER_BUF);
    pages = alloc->imported.user_buf.pages;









    for (i = 0; i < alloc->imported.user_buf.nr_pages; i++) {
        unsigned long local_size;

        dma_addr_t dma_addr = alloc->imported.user_buf.dma_addrs[i];

        local_size = MIN(size, PAGE_SIZE - (dma_addr & ~PAGE_MASK));
        dma_unmap_page(kctx->kbdev->dev, dma_addr, local_size,
                DMA_BIDIRECTIONAL);
        if (writeable)
            set_page_dirty_lock(pages[i]);
#if !MALI_USE_CSF
        kbase_unpin_user_buf_page(pages[i]);
        pages[i] = NULL;
#endif

        size -= local_size;

    }
#if !MALI_USE_CSF
    alloc->nents = 0;
#endif
}
```

```
 */
static void kbase_jd_user_buf_unmap(struct kbase_context *kctx, struct kbase_mem_phy_alloc *alloc,
                    struct kbase_va_region *reg, bool writeable)
{
    long i;
    struct page **pages;
    unsigned long offset_within_page = alloc->imported.user_buf.address & ~PAGE_MASK;
    unsigned long remaining_size = alloc->imported.user_buf.size;

    lockdep_assert_held(&kctx->reg_lock);

    KBASE_DEBUG_ASSERT(alloc->type == KBASE_MEM_TYPE_IMPORTED_USER_BUF);
    pages = alloc->imported.user_buf.pages;

#if !MALI_USE_CSF
    kbase_mem_shrink_cpu_mapping(kctx, reg, 0, alloc->nents);
#else
    CSTD_UNUSED(reg);
#endif

    for (i = 0; i < alloc->imported.user_buf.nr_pages; i++) {
        unsigned long unmap_size =
            MIN(remaining_size, PAGE_SIZE - offset_within_page);
        dma_addr_t dma_addr = alloc->imported.user_buf.dma_addrs[i];


        dma_unmap_page(kctx->kbdev->dev, dma_addr, unmap_size,
                DMA_BIDIRECTIONAL);
        if (writeable)
            set_page_dirty_lock(pages[i]);
#if !MALI_USE_CSF
        kbase_unpin_user_buf_page(pages[i]);
        pages[i] = NULL;
#endif

        remaining_size -= unmap_size;
        offset_within_page = 0;
    }
#if !MALI_USE_CSF
    alloc->nents = 0;
#endif
}
```

Fixed in the branch r38p1

CVE-2022-20186

- kbase_api_mem_alias

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > 2048)
        return -EINVAL;

    if (alias->in.stride > (U64_MAX / 2048))
        return -EINVAL;
```

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > BASE_MEM_ALIAS_MAX_ENTS)

        return -EINVAL;
```

Diff: r30 vs r31

CVE-2022-20186

```
union kbase_ioctl_mem_alias {
    struct {
        __u64 flags;
        __u64 stride;
        __u64 nents;
        __u64 aliasing_info;
    } in;
    struct {
        __u64 flags;
        __u64 gpu_va;
        __u64 va_pages;
    } out;
};
```

```
struct base_mem_aliasing_info {
    struct base_mem_handle
handle;
    __u64 offset;
    __u64 length;
};
```

eg:

stride = 1, nents = 3

VA0, offset = 1, length = 1

VA1, offset = 0, length = 1

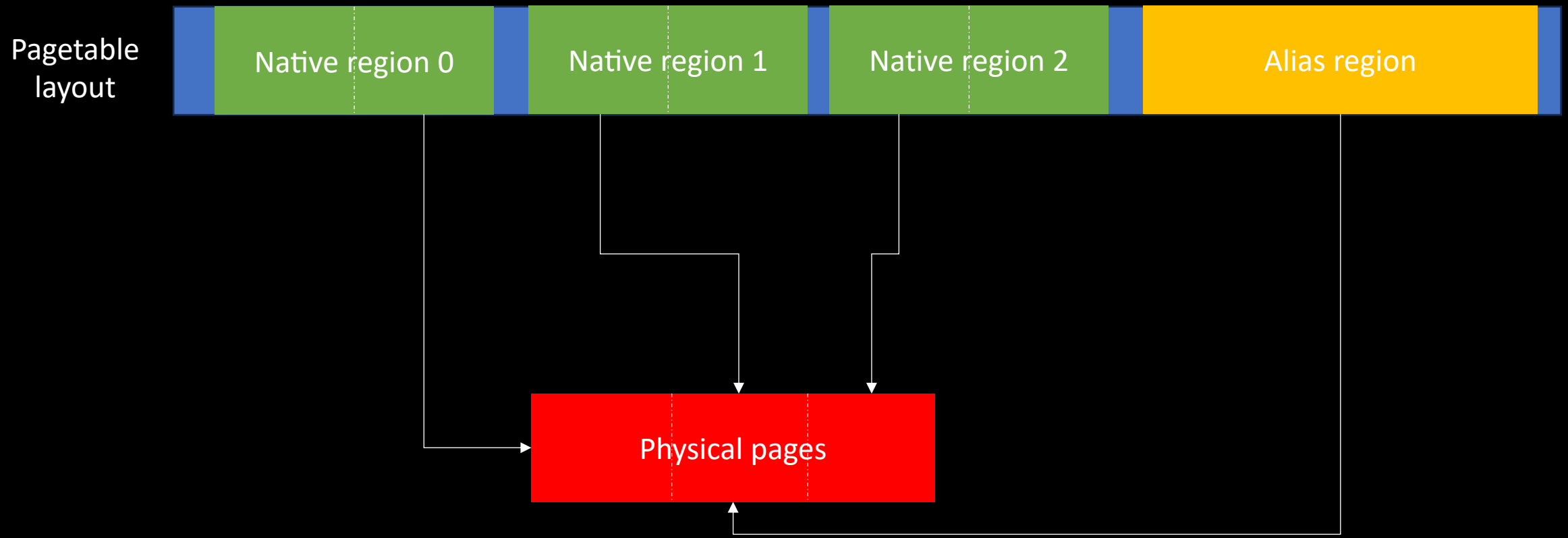
VA2, offset = 0, length = 1

CVE-2022-20186

- kbase_gpu_mmap

```
1494        KBASE_DEBUG_ASSERT(alloc->imported.alias.aliased);
1495        for (i = 0; i < alloc->imported.alias.nents; i++) {
1496            if (alloc->imported.alias.aliased[i].alloc) {
1497                err = kbase_mmu_insert_pages(kctx->kbdev,
1498                        &kctx->mmu,
1499                        reg->start_pfn + (i * stride),
1500                        alloc->imported.alias.aliased[i].alloc->pages + alloc->imported.alias.aliased[i].offset,
1501                        alloc->imported.alias.aliased[i].length,
1502                        reg->flags & gwt_mask,
1503                        kctx->as_nr,
1504                        group_id);
1505                if (err)
1506                    goto bad_insert;
```

CVE-2022-20186



CVE-2022-20186

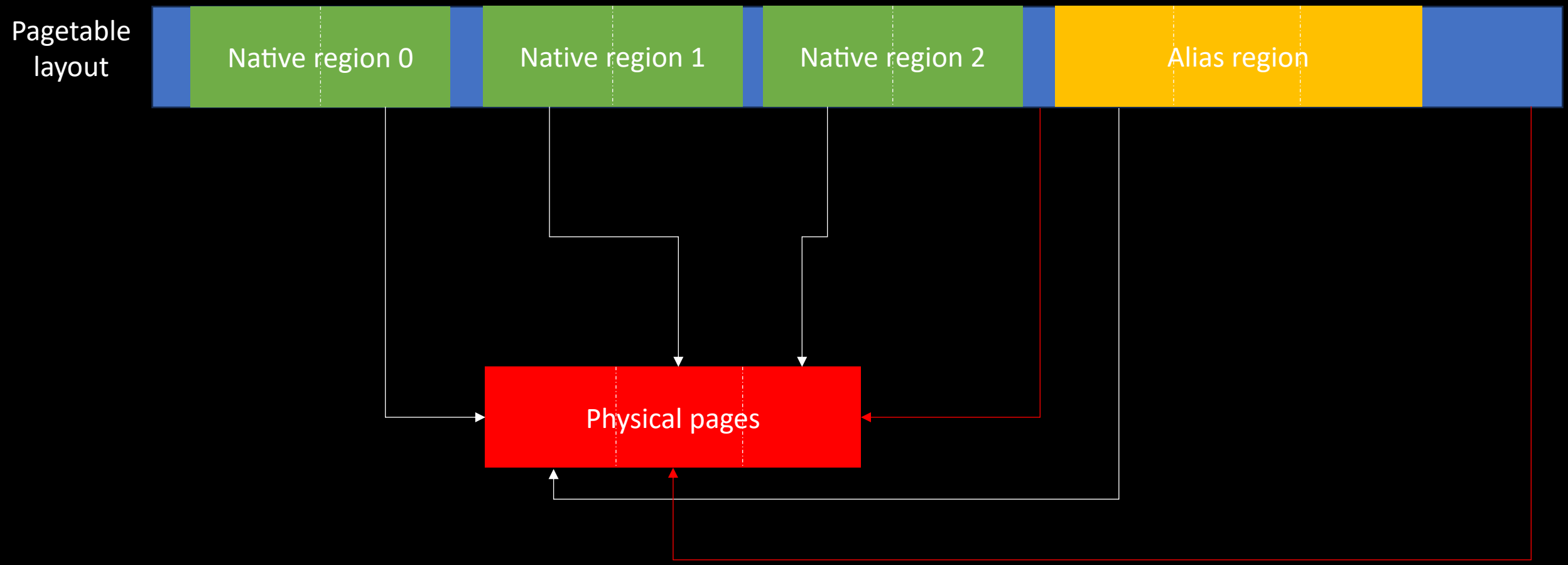
- stride = 0x8000 0000 0000 0001, nents = 3
  - VA0, offset = 1, length = 1
  - VA1, offset = 0, length = 1
  - VA2, offset = 0, length = 1

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > 2048)
        return -EINVAL;

    if (alias->in.stride > (U64_MAX / 2048))
        return -EINVAL;
```

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > BASE_MEM_ALIAS_MAX_ENTS)



        return -EINVAL;
```

Diff: r30 vs r31

CVE-2022-20186

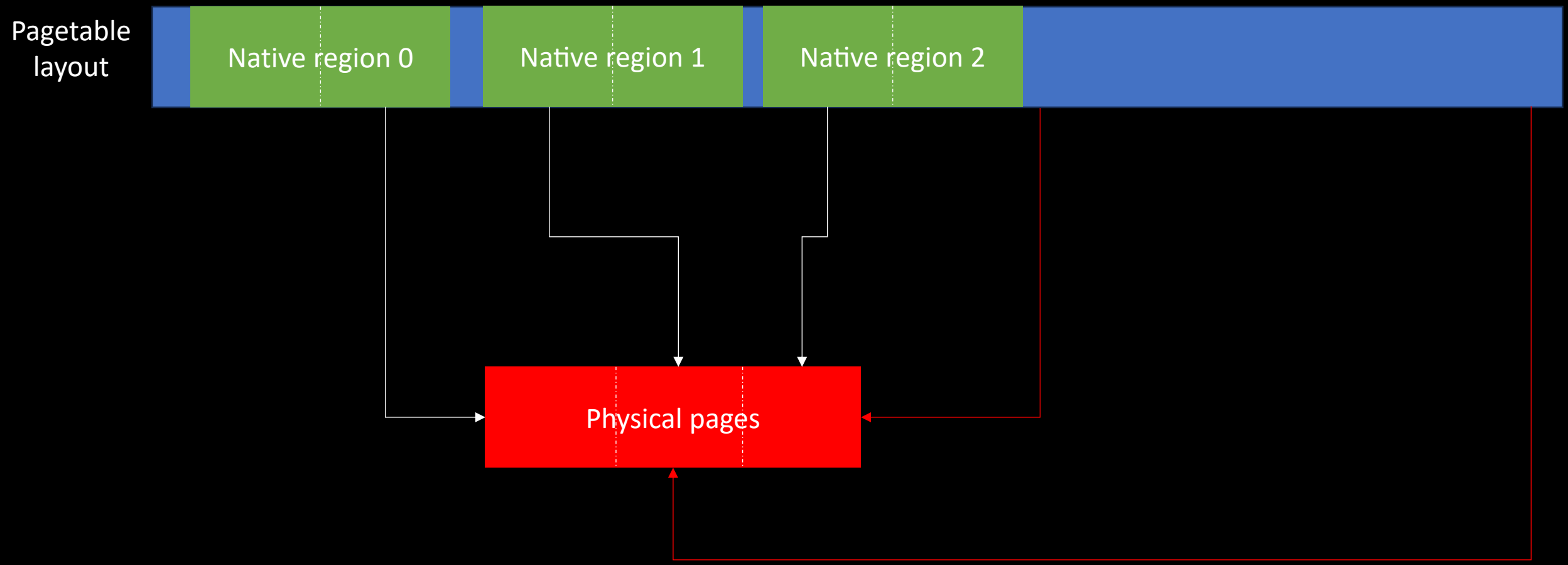


CVE-2022-20186

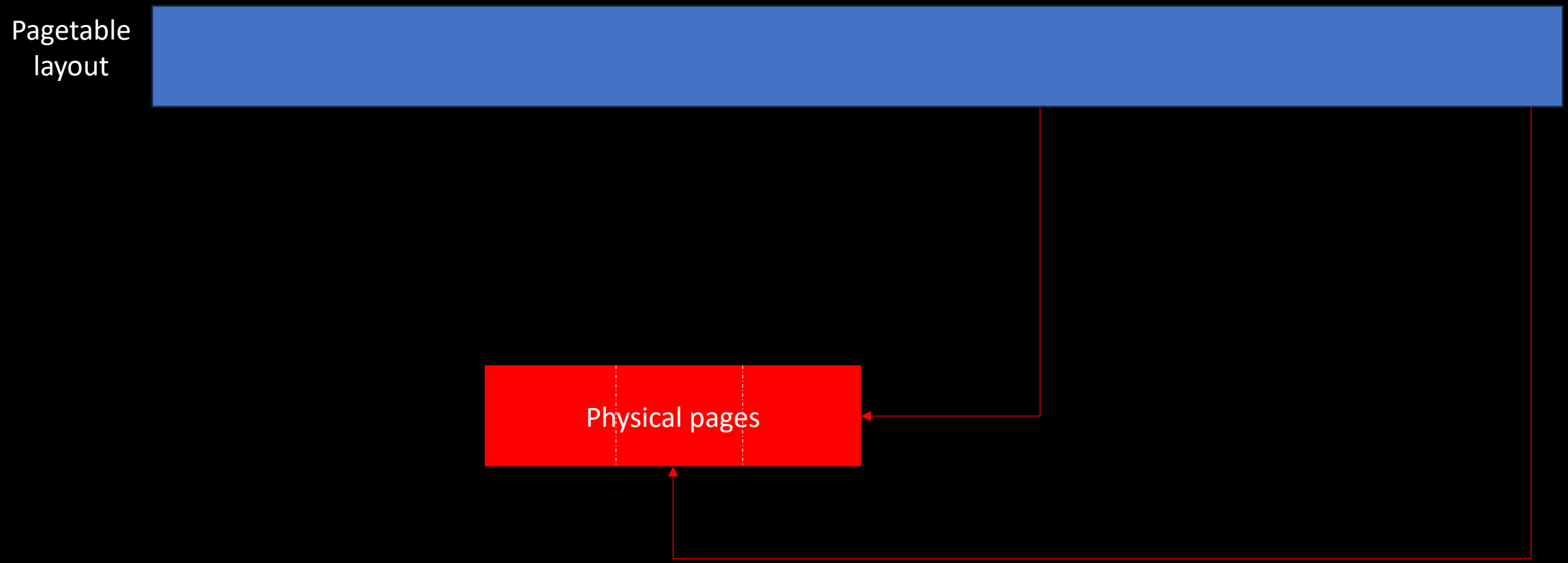
- kbase_gpu_munmap

```
int kbase_gpu_munmap(struct kbase_context *kctx, struct kbase_va_region *reg)
{
    int err = 0;

    if (reg->start_pfn == 0)
        return 0;

    if (!reg->gpu_alloc)
        return -EINVAL;

    /* Tear down down GPU page tables, depending on memory type. */
    switch (reg->gpu_alloc->type) {
    case KBASE_MEM_TYPE_ALIAS: /* Fall-through */
    case KBASE_MEM_TYPE_IMPORTED_UMM:
        err = kbase_mmu_teardown_pages(kctx->kbdev, &kctx->mmu,
                reg->start_pfn, reg->nr_pages, kctx->as_nr);
        break;
    default:
        err = kbase_mmu_teardown_pages(kctx->kbdev, &kctx->mmu,
            reg->start_pfn, kbase_reg_current_backed_size(reg),
            kctx->as_nr);
        break;
    }
```

CVE-2022-20186

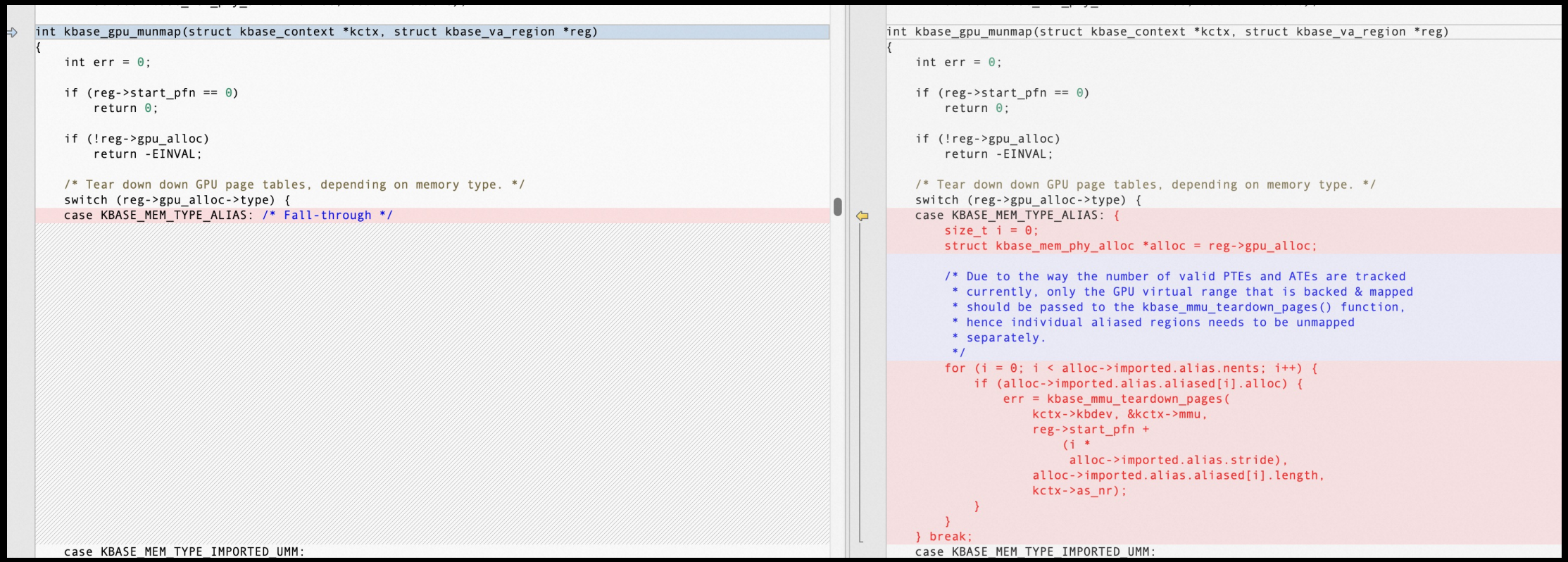


CVE-2022-20186



CVE-2022-20186

- kbase_gpu_munmap

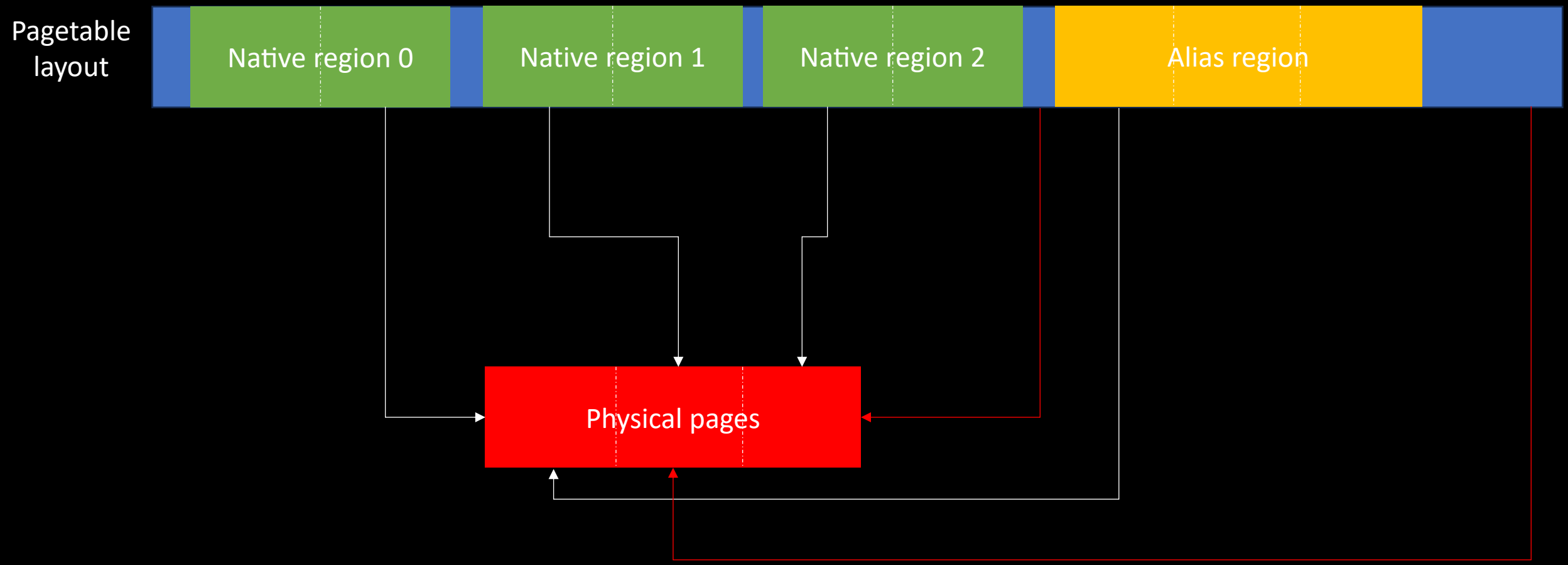


```
int kbase_gpu_munmap(struct kbase_context *kctx, struct kbase_va_region *reg)
{
    int err = 0;

    if (reg->start_pfn == 0)
        return 0;

    if (!reg->gpu_alloc)
        return -EINVAL;

    /* Tear down down GPU page tables, depending on memory type. */
    switch (reg->gpu_alloc->type) {
    case KBASE_MEM_TYPE_ALIAS: /* Fall-through */
```

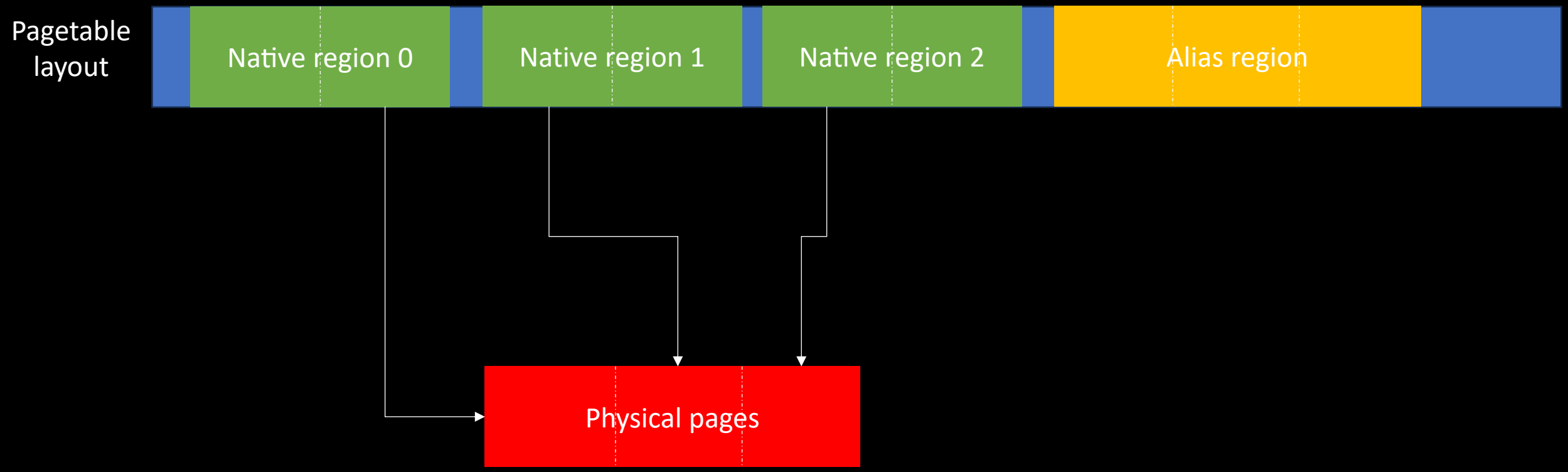
```
int kbase_gpu_munmap(struct kbase_context *kctx, struct kbase_va_region *reg)
{
    int err = 0;

    if (reg->start_pfn == 0)
        return 0;

    if (!reg->gpu_alloc)
        return -EINVAL;

    /* Tear down down GPU page tables, depending on memory type. */
    switch (reg->gpu_alloc->type) {
    case KBASE_MEM_TYPE_ALIAS: {
        size_t i = 0;
        struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;

        /* Due to the way the number of valid PTEs and ATEs are tracked
         * currently, only the GPU virtual range that is backed & mapped
         * should be passed to the kbase_mmu_teardown_pages() function,
         * hence individual aliased regions needs to be unmapped
         * separately.
         */
        for (i = 0; i < alloc->imported.alias.nents; i++) {
            if (alloc->imported.alias.aliased[i].alloc) {
                err = kbase_mmu_teardown_pages(
                    kctx->kbdev, &kctx->mmu,
                    reg->start_pfn +
                        (i *
                         alloc->imported.alias.stride),
                    alloc->imported.alias.aliased[i].length,
                    kctx->as_nr);
            }
        }
    } break;
    case KBASE_MEM_TYPE_IMPORTED_UMM:
```

Diff: r33 vs r34

CVE-2022-20186



CVE-2022-20186



CVE-2022-20186

- Pixel 6 is shipped with r32 (Non-CSF)
- MTK dimensity 9000 is shipped with r32 (CSF)

CVE-2022-20186

- Pixel 6 is shipped with r32 (Non-CSF)
- MTK dimensity 9000 is shipped with r32 (CSF)
- It’s exploitable against Android 12 devices

```
struct MALI_JOB_HEADER jh = {0};
jh.is_64b = true;
jh.type = MALI_JOB_TYPE_WRITE_VALUE;

struct MALI_WRITE_VALUE_JOB_PAYLOAD payload = {0};
payload.type = type;
payload.immediate_value = value;
payload.address = gpu_addr;

MALI_JOB_HEADER_pack((uint32_t*)jc_region, &jh);
MALI_WRITE_VALUE_JOB_PAYLOAD_pack((uint32_t*)jc_region + 8, &payload);
uint32_t* section = (uint32_t*)jc_region;
struct base_jd_atom_v2 atom = {0};
atom.jc = (uint64_t)jc_region;
atom.atom_number = atom_number;
atom.core_req = BASE_JD_REQ_CS;
struct kbase_ioctl_job_submit submit = {0};
submit.addr = (uint64_t)(&atom);
submit.nr_atoms = 1;
submit.stride = sizeof(struct base_jd_atom_v2);
if (ioctl(mali_fd, KBASE_IOCTL_JOB_SUBMIT, &submit) < 0) {
  err(1, "submit job failed\n");
}
usleep(10000);
```

CVE-2022-20186

- Pixel 6 is shipped with r32 (Non-CSF)
- MTK dimensity 9000 is shipped with r32 (CSF)
- It’s exploitable against Android 12 devices
- Is it possible to build the universal exploit against Non-CSF and CSF devices? 🤔

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
```

CVE-2023-????

```c
int kbase_timeline_io_acquire(struct kbase_device *kbdev, u32 flags)
{
    int ret = 0;
    u32 timeline_flags = TLSTREAM_ENABLED | flags;
    struct kbase_timeline *timeline = kbdev->timeline;

    if (!atomic_cmpxchg(timeline->timeline_flags, 0, timeline_flags)) {
        int rcode;

#if MALI_USE_CSF
        if (flags & BASE_TLSTREAM_ENABLE_CSFFW_TRACEPOINTS) {
            ret = kbase_csf_tl_reader_start(
                &timeline->csf_tl_reader, kbdev);
            if (ret)
            {
                atomic_set(timeline->timeline_flags, 0);
                return ret;
            }
        }
#endif
        ret = anon_inode_getfd(
                "[mali_tlstream]",
                &kbasep_tlstream_fops,
                timeline,
                O_RDONLY | O_CLOEXEC);
        if (ret < 0) {
            atomic_set(timeline->timeline_flags, 0);
#if MALI_USE_CSF
            kbase_csf_tl_reader_stop(&timeline->csf_tl_reader);
#endif
            return ret;
        }
```

```c
 */
#define KBASE_TLSTREAM_TL_NEW_CTX(  \
    kbdev,  \
    ctx,    \
    ctx_nr, \
    tgid    \
    )   \
    do {    \
        int enabled = atomic_read(&kbdev->timeline_flags);  \
        if (enabled & TLSTREAM_ENABLED) \
            __kbase_tlstream_tl_new_ctx(    \
                __TL_DISPATCH_STREAM(kbdev, obj),   \
                ctx, ctx_nr, tgid); \
    } while (0)
```

CVE-2023-????

```
void __kbase_tlstream_tl_new_ctx(
    struct kbase_tlstream *stream,
    const void *ctx,
    u32 ctx_nr,
    u32 tgid)
{
    const u32 msg_id = KBASE_TL_NEW_CTX;
    const size_t msg_size = sizeof(msg_id) + sizeof(u64)
        + sizeof(ctx)
        + sizeof(ctx_nr)
        + sizeof(tgid)
        ;
    char *buffer;
    unsigned long acq_flags;
    size_t pos = 0;

    buffer = kbase_tlstream_msgbuf_acquire(stream, msg_size, &acq_flags);

    pos = kbasep_serialize_bytes(buffer, pos, &msg_id, sizeof(msg_id));
    pos = kbasep_serialize_timestamp(buffer, pos);
    pos = kbasep_serialize_bytes(buffer,
        pos, &ctx, sizeof(ctx));
    pos = kbasep_serialize_bytes(buffer,
        pos, &ctx_nr, sizeof(ctx_nr));
    pos = kbasep_serialize_bytes(buffer,
        pos, &tgid, sizeof(tgid));

    kbase_tlstream_msgbuf_release(stream, acq_flags);
}
```

CVE-2023-????

```c
int kbase_timeline_io_acquire(struct kbase_device *kbdev, u32 flags)
{
    /* The timeline stream file operations structure. */
    static const struct file_operations kbasep_tlstream_fops = {
        .owner = THIS_MODULE,
        .release = kbasep_timeline_io_release,
        .read = kbasep_timeline_io_read,
        .poll = kbasep_timeline_io_poll,
        .fsync = kbasep_timeline_io_fsync,
    };
    int err;

    if (!timeline_is_permitted())
        return -EPERM;

    if (WARN_ON(!kbdev) || (flags & ~BASE_TLSTREAM_FLAGS_MASK))
        return -EINVAL;
```

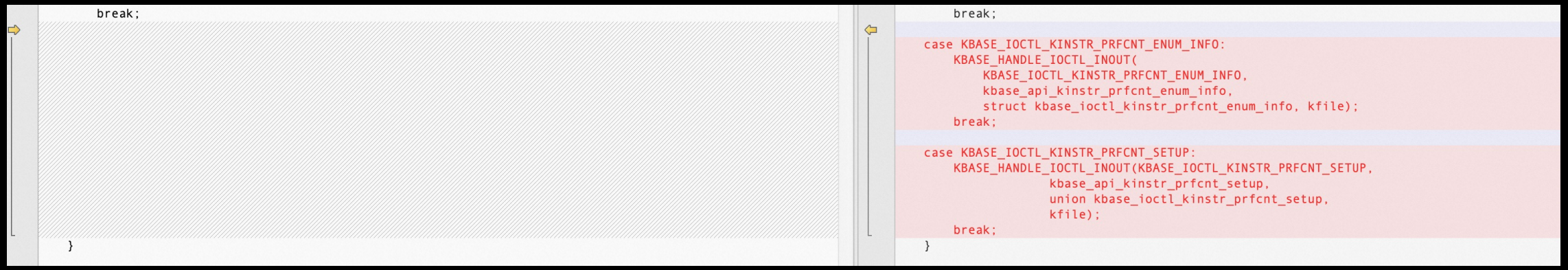
```c
static bool timeline_is_permitted(void)
{
#if KERNEL_VERSION(5, 8, 0) <= LINUX_VERSION_CODE
    return kbase_unprivileged_global_profiling || perfmon_capable();
#else
    return kbase_unprivileged_global_profiling || capable(CAP_SYS_ADMIN);
#endif
}
```

CVE-????

- The aforementioned issues are all logic bugs.
  - They do not cause kernel panic. Only error messages are logged into dmesg.

CVE-????

- The aforementioned issues are all logic bugs.
  - They do not cause kernel panic. Only error messages are logged into dmesg.
- Of course, there are some memory corruption 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-????

```
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);
}
```

```
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;
}
```

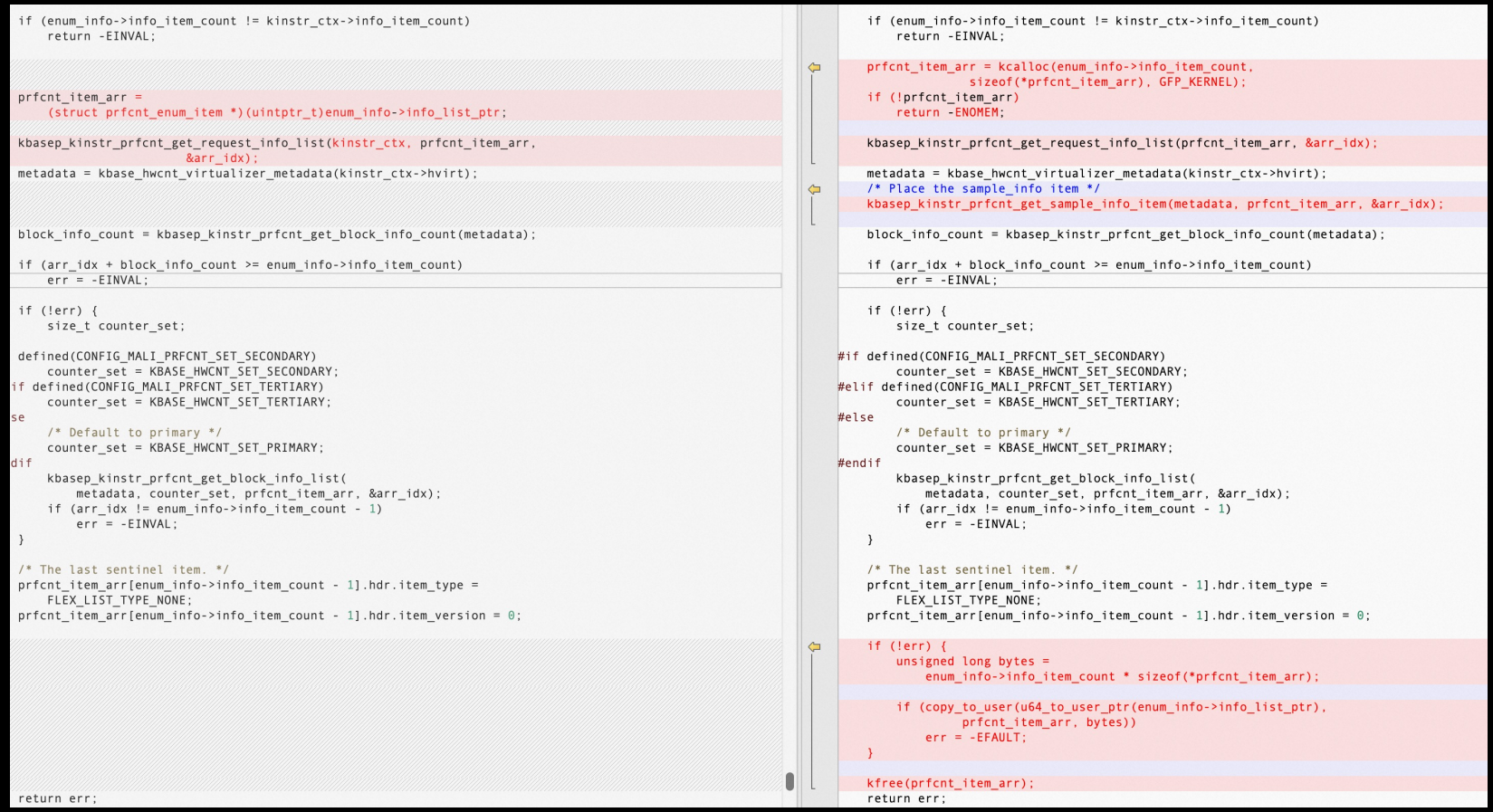
```
static int kbasep_kinstr_prfcnt_enum_info_list(
    struct kbase_kinstr_prfcnt_context *kinstr_ctx,
    struct kbase_ioctl_kinstr_prfcnt_enum_info *enum_info)
{
    struct prfcnt_enum_item *prfcnt_item_arr;
    size_t arr_idx = 0;
    int err = 0;
    size_t block_info_count = 0;
    const struct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

Arbitray 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

Case study- CSF

- What experience and history teach is this: that *** and *** have never learned anything from history or acted upon any lessons they might have drawn from it. -- G. W. F. Hegel

Case study- CSF

- What experience and history teach is this: that *** and *** have never learned anything from history or acted upon any lessons they might have drawn from it. -- G. W. F. Hegel

- What experience and history teach is this: developers have never learned anything from history or acted upon any lessons they might have drawn from it.

CVE-????

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > 2048)
        return -EINVAL;

    if (alias->in.stride > (U64_MAX / 2048))
        return -EINVAL;
```

```
static int kbase_api_mem_alias(struct kbase_context *kctx,
        union kbase_ioctl_mem_alias *alias)
{
    struct base_mem_aliasing_info *ai;
    u64 flags;
    int err;

    if (alias->in.nents == 0 || alias->in.nents > BASE_MEM_ALIAS_MAX_ENTS)



        return -EINVAL;
```

```
static int kbasep_cs_tiler_heap_init(struct kbase_context *kctx,
        union kbase_ioctl_cs_tiler_heap_init *heap_init)
{
    kctx->jit_group_id = heap_init->in.group_id;

    return kbase_csf_tiler_heap_init(kctx, heap_init->in.chunk_size,
        heap_init->in.initial_chunks, heap_init->in.max_chunks,
        heap_init->in.target_in_flight,
        &heap_init->out.gpu_heap_va, &heap_init->out.first_chunk_va);
}
```

CVE-????

```
#ifdef CONFIG_MALI_2MB_ALLOC
    if (pages_required >= (SZ_2M / SZ_4K)) {
        pool = &kctx->mem_pools.large[kctx->jit_group_id];
        /* Round up to number of 2 MB pages required */
        pages_required += ((SZ_2M / SZ_4K) - 1);
        pages_required /= (SZ_2M / SZ_4K);
    } else {
#endif
        pool = &kctx->mem_pools.small[kctx->jit_group_id];
#ifdef CONFIG_MALI_2MB_ALLOC
    }
#endif
```

```
struct kbase_mem_pool_group {
    struct kbase_mem_pool small[MEMORY_GROUP_MANAGER_NR_G
    struct kbase_mem_pool large[MEMORY_GROUP_MANAGER_NR_G
};
```

```
*/
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;
};
```

- Kbdev can be obtained from the forementioned information leak bug

Fix

```
static int kbasep_cs_tiler_heap_init(struct kbase_context *kctx,
        union kbase_ioctl_cs_tiler_heap_init *heap_init)
{
    if (heap_init->in.group_id >= MEMORY_GROUP_MANAGER_NR_GROUPS)
        return -EINVAL;

    kctx->jit_group_id = heap_init->in.group_id;

    return kbase_csf_tiler_heap_init(kctx, heap_init->in.chunk_size,
                    heap_init->in.initial_chunks, heap_init->in.max_chunks,
                    heap_init->in.target_in_flight, heap_init->in.buf_desc_va,
                    &heap_init->out.gpu_heap_va,
                    &heap_init->out.first_chunk_va);
}
```

CVE-????

```
} else {
    struct tagged_addr *page_array;
    u64 start, end, i;

    if (!(reg->flags & BASE_MEM_SAME_VA) ||
            reg->nr_pages < nr_pages ||
            kbase_reg_current_backed_size(reg) !=
                reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    start = PFN_DOWN(page_addr) - reg->start_pfn;
    end = start + nr_pages;

    if (end > reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    sus_buf->cpu_alloc = kbase_mem_phy_alloc_get(reg->cpu_alloc);
    kbase_mem_phy_alloc_kernel_mapped(reg->cpu_alloc);
    page_array = kbase_get_cpu_phy_pages(reg);
    page_array += start;

    for (i = 0; i < nr_pages; i++, page_array++)
        sus_buf->pages[i] = as_page(*page_array);
}
```

kbase_csf_queue_group_suspend_prepare

```
for (i = 0; i < PFN_UP(sus_buf->size) &&
        target_page_nr < sus_buf->nr_pages; i++) {
    struct page *pg =
        as_page(group->normal_suspend_buf.phy[i]);
    void *sus_page = kmap(pg);

    if (sus_page) {
        kbase_sync_single_for_cpu(kbdev,
            kbase_dma_addr(pg),
            PAGE_SIZE, DMA_BIDIRECTIONAL);

        err = kbase_mem_copy_to_pinned_user_pages(
                sus_buf->pages, sus_page,
                &to_copy, sus_buf->nr_pages,
                &target_page_nr, offset);
        kunmap(pg);
        if (err)
            break;
    } else {
        err = -ENOMEM;
        break;
    }
}
```

kbase_csf_scheduler_group_copy_suspend_buf

CVE-????

```
} else {
    struct tagged_addr *page_array;
    u64 start, end, i;

    if (!(reg->flags & BASE_MEM_SAME_VA) ||
            reg->nr_pages < nr_pages ||
            kbase_reg_current_backed_size(reg) !=
                reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    start = PFN_DOWN(page_addr) - reg->start_pfn;
    end = start + nr_pages;

    if (end > reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    sus_buf->cpu_alloc = kbase_mem_phy_alloc_get(reg->cpu_alloc);
    kbase_mem_phy_alloc_kernel_mapped(reg->cpu_alloc);
    page_array = kbase_get_cpu_phy_pages(reg);
    page_array += start;

    for (i = 0; i < nr_pages; i++, page_array++)
        sus_buf->pages[i] = as_page(*page_array);
}
```

kbase_csf_queue_group_suspend_prepare

- KBASE_MEM_TYPE_IMPORTED_USER_BUF
  - Read only -> Read Write

CVE-????

```
} else {
    struct tagged_addr *page_array;
    u64 start, end, i;

    if (!(reg->flags & BASE_MEM_SAME_VA) ||
            reg->nr_pages < nr_pages ||
            kbase_reg_current_backed_size(reg) !=
                reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    start = PFN_DOWN(page_addr) - reg->start_pfn;
    end = start + nr_pages;

    if (end > reg->nr_pages) {
        ret = -EINVAL;
        goto out_clean_pages;
    }

    sus_buf->cpu_alloc = kbase_mem_phy_alloc_get(reg->cpu_alloc);
    kbase_mem_phy_alloc_kernel_mapped(reg->cpu_alloc);
    page_array = kbase_get_cpu_phy_pages(reg);
    page_array += start;

    for (i = 0; i < nr_pages; i++, page_array++)
        sus_buf->pages[i] = as_page(*page_array);
}
```

kbase_csf_queue_group_suspend_prepare

- KBASE_MEM_TYPE_IMPORTED_USER_BUF
  - Read only -> Read Write
- JIT region
  - Physical pages can be reclaimed
  - Page Use-After-Free

CVE-????

```
for (i = 0; i < PFN_UP(sus_buf->size) &&
        target_page_nr < sus_buf->nr_pages; i++) {
    struct page *pg =
        as_page(group->normal_suspend_buf.phy[i]);
    void *sus_page = kmap(pg);

    if (sus_page) {
        kbase_sync_single_for_cpu(kbdev,
            kbase_dma_addr(pg),
            PAGE_SIZE, DMA_BIDIRECTIONAL);

        err = kbase_mem_copy_to_pinned_user_pages(
                sus_buf->pages, sus_page,
                &to_copy, sus_buf->nr_pages,
                &target_page_nr, offset);
        kunmap(pg);
        if (err)
            break;
    } else {
        err = -ENOMEM;
        break;
    }
}
```

OOB leads to kernel panic!

Fix

```
struct tagged_addr *page_array;
u64 start, end, i;

if (((reg->flags & KBASE_REG_ZONE_MASK) != KBASE_REG_ZONE_SAME_VA) ||
    (kbase_reg_current_backed_size(reg) < nr_pages) ||
    !(reg->flags & KBASE_REG_CPU_WR) ||
    (reg->gpu_alloc->type != KBASE_MEM_TYPE_NATIVE) ||
    (kbase_is_region_shrinkable(reg)) || (kbase_va_region_is_no_user_free(reg))) {
    ret = -EINVAL;
    goto out_clean_pages;
}

start = PFN_DOWN(page_addr) - reg->start_pfn;
end = start + nr_pages;

if (end > reg->nr_pages) {
    ret = -EINVAL;
    goto out_clean_pages;
}

sus_buf->cpu_alloc = kbase_mem_phy_alloc_get(reg->cpu_alloc);
kbase_mem_phy_alloc_kernel_mapped(reg->cpu_alloc);
page_array = kbase_get_cpu_phy_pages(reg);
page_array += start;

for (i = 0; i < nr_pages; i++, page_array++)
    sus_buf->pages[i] = as_page(*page_array);
```

```
#if IS_ENABLED(CONFIG_MALI_VECTOR_DUMP) || MALI_UNIT_TEST
static int kbase_csf_queue_group_suspend_prepare(
        struct kbase_kcpu_command_queue *kcpu_queue,
        struct base_kcpu_command_group_suspend_info *suspend_buf,
        struct kbase_kcpu_command *current_command)
{
    struct kbase_context *const kctx = kcpu_queue->kctx;
    struct kbase_suspend_copy_buffer *sus_buf = NULL;
    const u32 csg_suspend_buf_size =
        kctx->kbdev->csf.global_iface.groups[0].suspend_size;
    u64 addr = suspend_buf->buffer;
    u64 page_addr = addr & PAGE_MASK;
    u64 end_addr = addr + csg_suspend_buf_size - 1;
    u64 last_page_addr = end_addr & PAGE_MASK;
    int nr_pages = (last_page_addr - page_addr) / PAGE_SIZE + 1;
    int pinned_pages = 0, ret = 0;
    struct kbase_va_region *reg;

    lockdep_assert_held(&kcpu_queue->lock);

    if (suspend_buf->size < csg_suspend_buf_size)
        return -EINVAL;

    ret = kbase_csf_queue_group_handle_is_valid(kctx,
            suspend_buf->group_handle);
    if (ret)
        return ret;
```

CVE-????

- KBASE_IOCTL_CS_QUEUE_REGISTER

```
queue_addr = reg->buffer_gpu_addr;
queue_size = reg->buffer_size >> PAGE_SHIFT;
/* Check if queue is already registered */
if (find_queue(kctx, queue_addr) != NULL) {
        ret = -EINVAL;
        goto out;
}
region = kbase_region_tracker_find_region_enclosing_address(kctx,    queue_addr);

region->flags |= KBASE_REG_NO_USER_FREE;
```

CVE-????

- KBASE_IOCTL_CS_QUEUE_TERMINATE

```
queue = find_queue(kctx, term->buffer_gpu_addr);
if (queue) {
        unbind_queue(kctx, queue);
        if (!WARN_ON(!queue->queue_reg)) {
                /* After this the Userspace would be able to free the
                 * memory for GPU queue. In case the Userspace missed
                 * terminating the queue, the cleanup will happen on
                 * context termination where tear down of region tracker
                 * would free up the GPU queue memory.
                 */
                queue->queue_reg->flags &= ~KBASE_REG_NO_USER_FREE;
```

CVE-????

```
int kbase_mem_free_region(struct kbase_context *kctx, struct kbase_va_region *reg)
{
        int err;

        KBASE_DEBUG_ASSERT(kctx != NULL);
        KBASE_DEBUG_ASSERT(reg != NULL);
        dev_dbg(kctx->kbdev->dev, "%s %pK in kctx %pK\n",
                __func__, (void *)reg, (void *)kctx);
        lockdep_assert_held(&kctx->reg_lock);

        if (reg->flags & KBASE_REG_NO_USER_FREE) {
                dev_warn(kctx->kbdev->dev, "Attempt to free GPU memory whose freeing by user space is
forbidden!\n");
                return -EINVAL;
        }
```

CVE-????

- Unconditionally clear the KBASE_REG_NO_USER_FREE flag
  - KBASE_IOCTL_CS_QUEUE_REGISTER
  - KBASE_IOCTL_CS_QUEUE_TERMINATE

CVE-????

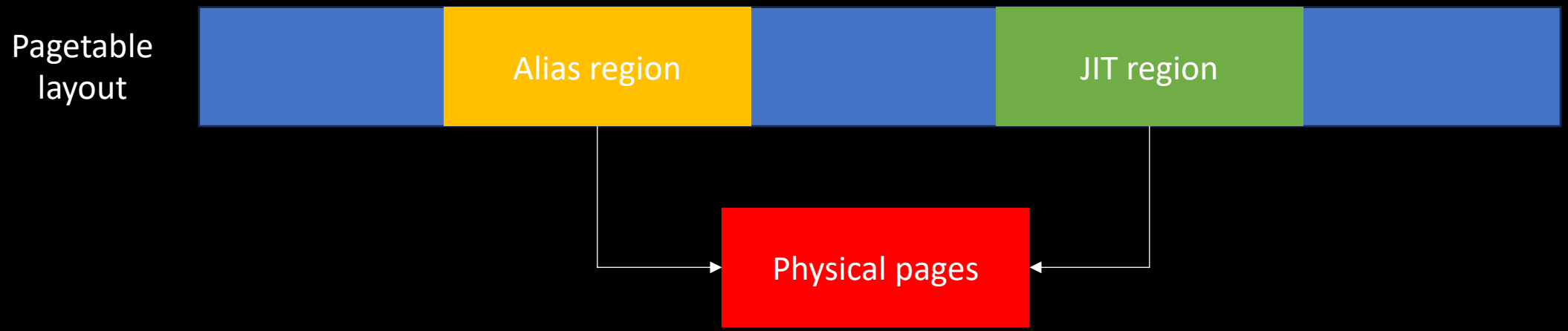
- Unconditionally clear the KBASE_REG_NO_USER_FREE flag
  - KBASE_IOCTL_CS_QUEUE_REGISTER
  - KBASE_IOCTL_CS_QUEUE_TERMINATE
- Impact
  - A region with KBASE_REG_NO_USER_FREE can be freed by user space
  - A region with KBASE_REG_NO_USER_FREE can be aliased(KBASE_IOCTL_MEM_ALIAS)

```
if (aliasing_reg->flags & KBASE_REG_NO_USER_FREE)
    goto bad_handle; /* JIT regions can't be
                      * aliased. NO_USER_FREE flag
                      * covers the entire lifetime
                      * of JIT regions. The other
                      * types of regions covered
                      * by this flag also shall
                      * not be aliased.
                      */
```

CVE-????

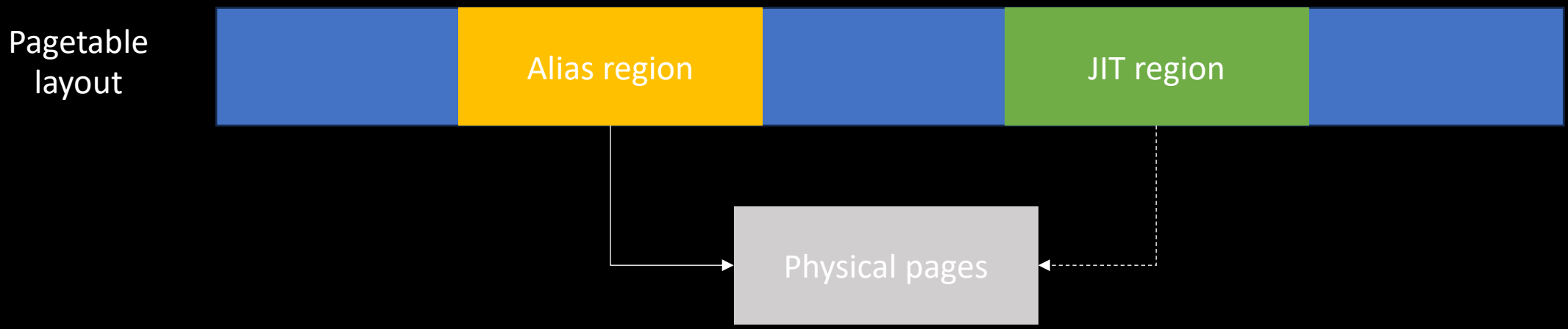
- JIT region can be aliased

  BASE_MEM_PROT_CPU_RD | BASE_MEM_PROT_GPU_RD |
  BASE_MEM_PROT_GPU_WR | BASE_MEM_GROW_ON_GPF |
  BASE_MEM_COHERENT_LOCAL | BASEP_MEM_NO_USER_FREE;



CVE-????

- JIT region can be freed via BASE_KCPU_COMMAND_TYPE_JIT_FREE
  - The associated physical pages can be reclaimed



Fix

```
-       if (kbase_is_region_invalid_or_free(region)) {
+       if (kbase_is_region_invalid_or_free(region) || kbase_is_region_shrinkable(region) ||
+           region->gpu_alloc->type != KBASE_MEM_TYPE_NATIVE) {
                ret = -ENOENT;
                goto out_unlock_vm;
        }
@@ -555,7 +567,7 @@

        queue->kctx = kctx;
        queue->base_addr = queue_addr;
-       queue->queue_reg = region;
+       queue->queue_reg = kbase_va_region_no_user_free_get(kctx, region);
        queue->size = (queue_size << PAGE_SHIFT);
        queue->csi_index = KBASEP_IF_NR_INVALID;
        queue->enabled = false;
@@ -589,7 +601,6 @@

        queue->extract_ofs = 0;

-       region->flags |= KBASE_REG_NO_USER_FREE;
        region->user_data = queue;

        /* Initialize the cs_trace configuration parameters, When buffer_size
@@ -683,16 +694,8 @@
                unbind_queue(kctx, queue);

                kbase_gpu_vm_lock(kctx);
-               if (!WARN_ON(!queue->queue_reg)) {
-                       /* After this the Userspace would be able to free the
-                        * memory for GPU queue. In case the Userspace missed
-                        * terminating the queue, the cleanup will happen on
-                        * context termination where tear down of region tracker
-                        * would free up the GPU queue memory.
-                        */
-                       queue->queue_reg->flags &= ~KBASE_REG_NO_USER_FREE;
+               if (!WARN_ON(!queue->queue_reg))
                        queue->queue_reg->user_data = NULL;
-               }
                kbase_gpu_vm_unlock(kctx);
```

- https://android.googlesource.com/kernel/google-modules/gpu/+/422aa1fad7e63f16000ffb9303e816b54ef3d8ca%5E%21/#F0

CVE-????- variable analysis

```
/* Check on the buffer descriptor virtual Address */
if (buf_desc_va) {
    kbase_gpu_vm_lock(kctx);
    reg = kbase_region_tracker_find_region_enclosing_address(kctx, buf_desc_va);
    if (kbase_is_region_invalid_or_free(reg) || !(reg->flags & KBASE_REG_CPU_RD) ||
        (reg->gpu_alloc->type != KBASE_MEM_TYPE_NATIVE)) {
        kbase_gpu_vm_unlock(kctx);
        return -EINVAL;
    }

    reg->flags |= KBASE_REG_NO_USER_FREE;
    kbase_gpu_vm_unlock(kctx);
}
```

- kbase_csf_tiler_heap_init

```
static void delete_chunk(struct kbase_csf_tiler_heap *const heap,
            struct kbase_csf_tiler_heap_chunk *const chunk, bool reclaim)
{
    struct kbase_context *const kctx = heap->kctx;

    kbase_gpu_vm_lock(kctx);
    chunk->region->flags &= ~KBASE_REG_NO_USER_FREE;
    if (reclaim)
        mark_free_mem_bypassing_pool(chunk->region);
    kbase_mem_free_region(kctx, chunk->region);
    kbase_gpu_vm_unlock(kctx);
    list_del(&chunk->link);
    heap->chunk_count--;
    kfree(chunk);
}
```

- kbase_csf_tiler_heap_term

Fix

```
static bool kbasep_is_buffer_descriptor_region_suitable(struct kbase_context *const kctx,
                                                        struct kbase_va_region *const reg)
{
    if (kbase_is_region_invalid_or_free(reg)) {
        dev_err(kctx->kbdev->dev, "Region is either invalid or free!\n");
        return false;
    }

    if (!(reg->flags & KBASE_REG_CPU_RD) || kbase_is_region_shrinkable(reg) ||
        (reg->flags & KBASE_REG_PF_GROW)) {
        dev_err(kctx->kbdev->dev, "Region has invalid flags: 0x%lX!\n", reg->flags);
        return false;
    }

    if (reg->gpu_alloc->type != KBASE_MEM_TYPE_NATIVE) {
        dev_err(kctx->kbdev->dev, "Region has invalid type!\n");
        return false;
    }

    if ((reg->nr_pages != kbase_reg_current_backed_size(reg)) ||
        (reg->nr_pages < PFN_UP(sizeof(struct kbase_csf_gpu_buffer_heap)))) {
        dev_err(kctx->kbdev->dev, "Region has invalid backing!\n");
        return false;
    }

    return true;
}
```

Agenda

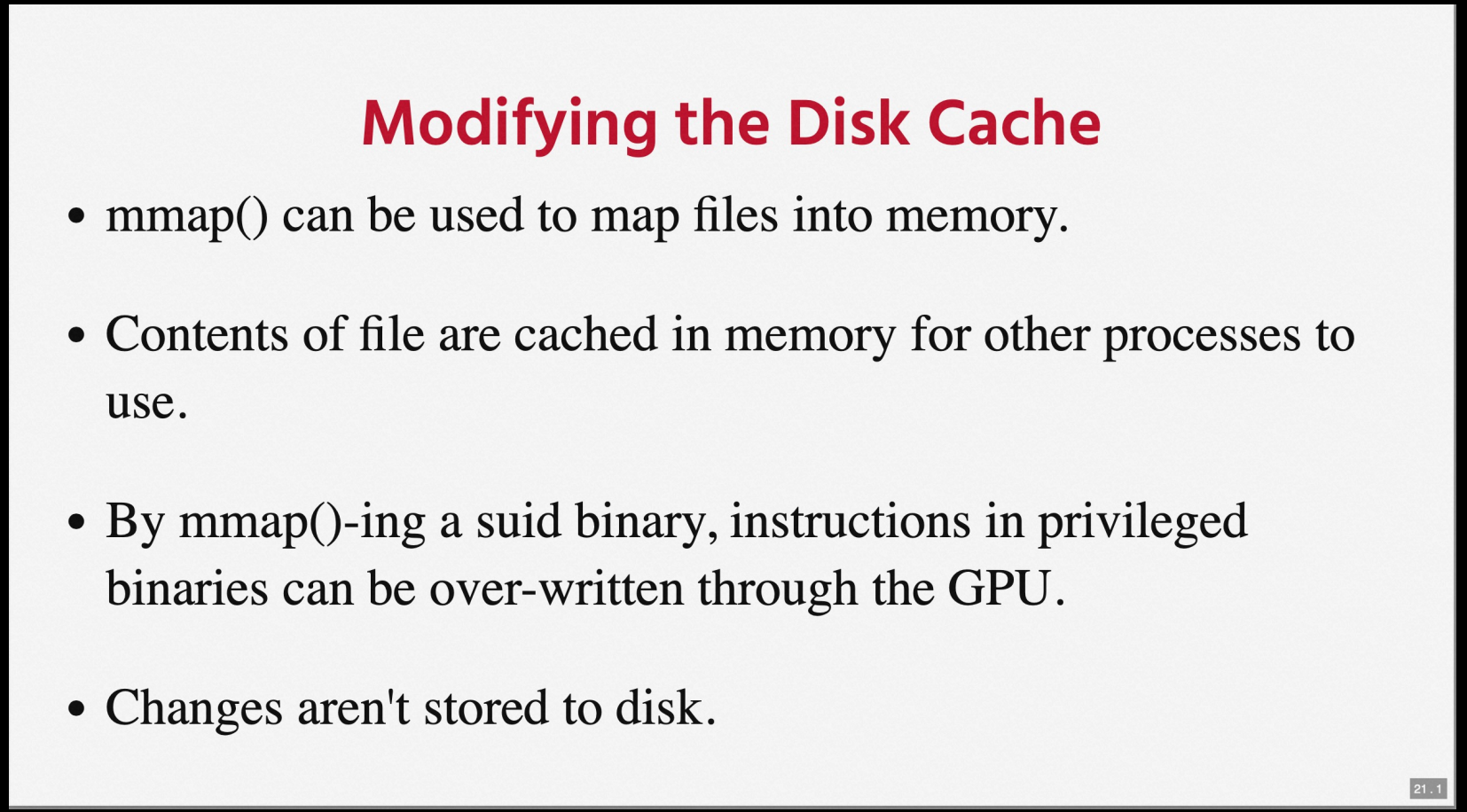
- Introduction
- Case study
- *Exploitation*
- Conclusion

Three categories

- Elevates CPU RO pages to writable
- Page UAF
- Page UAF in GPU MMU

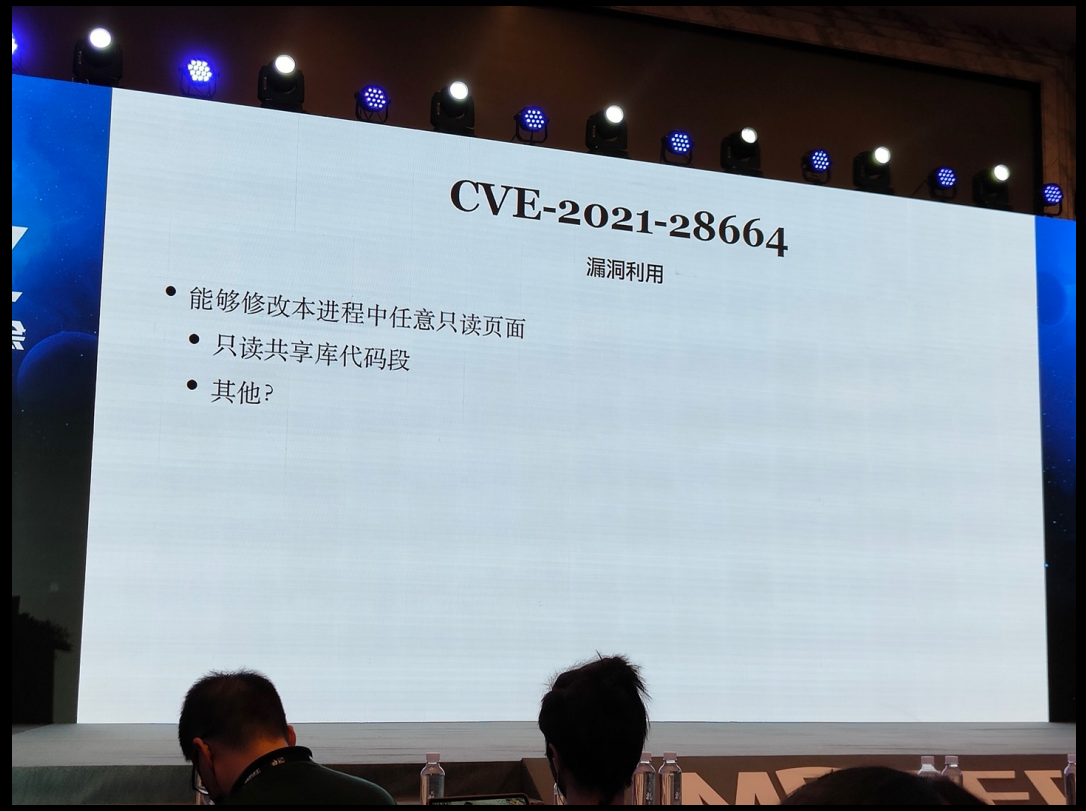
Elevates CPU RO pages to writable

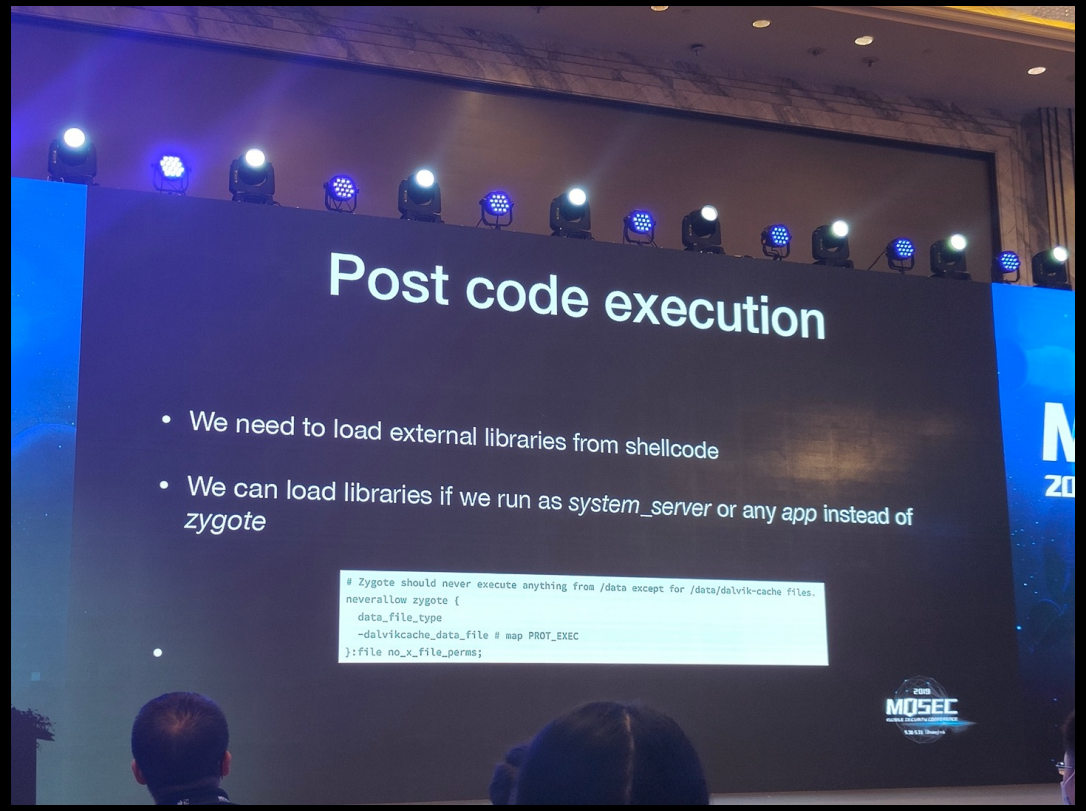
- Old way



https://www.blackhat.com/docs/eu-16/materials/eu-16-Taft-GPU-Security-Exposed.pdf

Elevates CPU RO pages to writable





Elevates CPU RO pages to writable

- Not GKI (<kernel 5.10)
- Module signing enabled

```
CONFIG_RT_MUTEXES=y
CONFIG_BASE_SMALL=0
CONFIG_MODULE_SIG_FORMAT=y
CONFIG_MODULES=y
# CONFIG_MODULE_FORCE_LOAD is not set
CONFIG_MODULE_UNLOAD=y
# CONFIG_MODULE_FORCE_UNLOAD is not set
CONFIG_MODVERSIONS=y
CONFIG_ASM_MODVERSIONS=y
# CONFIG_MODULE_SRCVERSION_ALL is not set
CONFIG_MODULE_SCMVERSION=y
CONFIG_MODULE_SIG=y
CONFIG_MODULE_SIG_FORCE=y
CONFIG_MODULE_SIG_ALL=y
# CONFIG_MODULE_SIG_SHA1 is not set
# CONFIG_MODULE_SIG_SHA224 is not set
# CONFIG_MODULE_SIG_SHA256 is not set
# CONFIG_MODULE_SIG_SHA384 is not set
CONFIG_MODULE_SIG_SHA512=y
CONFIG_MODULE_SIG_HASH="sha512"
# CONFIG_MODULE_COMPRESS is not set
# CONFIG_MODULE_ALLOW_MISSING_NAMESPACE_IMPORTS is not set
# CONFIG_UNUSED_SYMBOLS is not set
CONFIG_TRIM_UNUSED_KSYMS=y
```

Elevates CPU RO pages to writable

- Read only memory
- vm_insert_page
  - User buffer can be imported

```
static int binder_mmap(struct file *filp, struct vm_area_struct *vma)
{
    struct binder_proc *proc = filp->private_data;

    if (proc->tsk != current->group_leader)
        return -EINVAL;

    if (vma->vm_flags & FORBIDDEN_MMAP_FLAGS) { // VM_WRITE
        pr_err("%s: %d %lx-%lx %s failed %d\n", __func__,
               proc->pid, vma->vm_start, vma->vm_end, "bad vm_flags", -EPERM);
        return -EPERM;
    }
    vma->vm_flags |= VM_DONTCOPY | VM_MIXEDMAP;
    vma->vm_flags &= ~VM_MAYWRITE;

    vma->vm_ops = &binder_vm_ops;
    vma->vm_private_data = proc;

    return binder_alloc_mmap_handler(&proc->alloc, vma);
}
```

```
page->page_ptr = alloc_page(GFP_KERNEL |
                            __GFP_HIGHMEM |
                            __GFP_ZERO);
if (!page->page_ptr) {
    pr_err("%d: binder_alloc_buf failed for p
           alloc->pid, page_addr);
    goto err_alloc_page_failed;
}
page->alloc = alloc;
INIT_LIST_HEAD(&page->lru);

user_page_addr = (uintptr_t)page_addr;
ret = vm_insert_page(vma, user_page_addr, page[0].page_ptr);
```

Elevates CPU RO pages to writable

- Craft the flat_binder_object by modifying the binder's user buffer

```
struct flat_binder_object {
    struct binder_object_header hdr;
    __u32                  flags;

    /* 8 bytes of data. */
    union {
        binder_uintptr_t    binder; /* local object */
        __u32               handle; /* remote object */
    };

    /* extra data associated with local object */
    binder_uintptr_t    cookie;
};
```

```
hdr = &object.hdr;
switch (hdr->type) {
case BINDER_TYPE_BINDER:
case BINDER_TYPE_WEAK_BINDER: {
    struct flat_binder_object *fp;
    struct binder_node *node;

    fp = to_flat_binder_object(hdr);
    node = binder_get_node(proc, fp->binder);
    if (node == NULL) {
        pr_err("transaction release %d bad node %016llx\n",
                debug_id, (u64)fp->binder);
        break;
    }
    binder_debug(BINDER_DEBUG_TRANSACTION,
                 "        node %d u%016llx\n",
                 node->debug_id, (u64)node->ptr);
    binder_dec_node(node, hdr->type == BINDER_TYPE_BINDER,
                    0);
    binder_put_node(node);
} break;
```

Elevates CPU RO pages to writable

- Exploit the UAF bug like CVE-2020-0041
  - https://labs.bluefrostsecurity.de/blog/2020/04/08/cve-2020-0041-part-2-escalating-to-root/

```
if (t->buffer->target_node) {
    struct binder_node *target_node = t->buffer->target_node;
    struct binder_priority node_prio;

    trd->target.ptr = target_node->ptr;
    trd->cookie =  target_node->cookie;
    node_prio.sched_policy = target_node->sched_policy;
    node_prio.prio = target_node->min_priority;
    binder_transaction_priority(current, t, node_prio,
                                target_node->inherit_rt);
    cmd = BR_TRANSACTION;
} else {
    trd->target.ptr = 0;
    trd->cookie = 0;
    cmd = BR_REPLY;
}
trd->code = t->code;
trd->flags = t->flags;
trd->sender_euid = from_kuid(current_user_ns(), t->sender_euid);
```

Elevates CPU RO pages to writable

- Exploit the UAF bug like CVE-2020-0041
  - https://labs.bluefrostsecurity.de/blog/2020/04/08/cve-2020-0041-part-2-escalating-to-root/

```
} else {
    BUG_ON(!list_empty(&node->work.entry));
    spin_lock(&binder_dead_nodes_lock);
    /*
     * tmp_refs could have changed so
     * check it again
     */
    if (node->tmp_refs) {
        spin_unlock(&binder_dead_nodes_lock);
        return false;
    }
    hlist_del(&node->dead_node);
    spin_unlock(&binder_dead_nodes_lock);
    binder_debug(BINDER_DEBUG_INTERNAL_REFS,
```

Elevates CPU RO pages to writable

- New way

Loadable Kernel Modules

As part of the module kernel requirements introduced in Android 8.0, all system-on-chip (SoC) kernels must support loadable kernel modules.

Kernel configuration options

To support loadable kernel modules, android-base.cfg in all common kernels includes the following ker options (or their kernel-version equivalent):

```
CONFIG_MODULES=y
CONFIG_MODULE_UNLOAD=y
CONFIG_MODVERSIONS=y
```

All device kernels must enable these options. Kernel modules should also support unloading and reloading whenever possible.

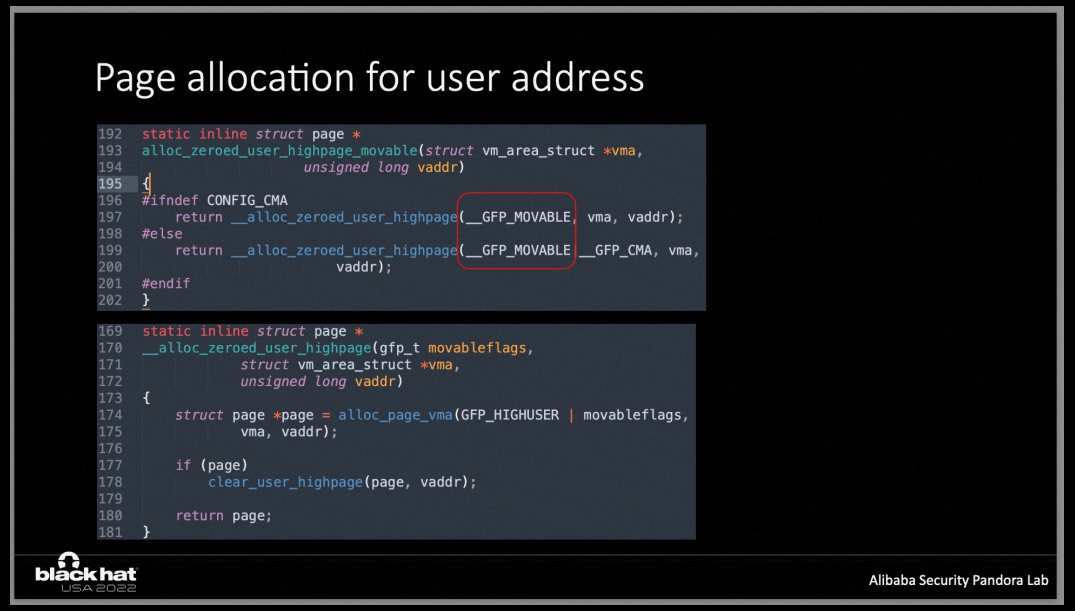
Module signing

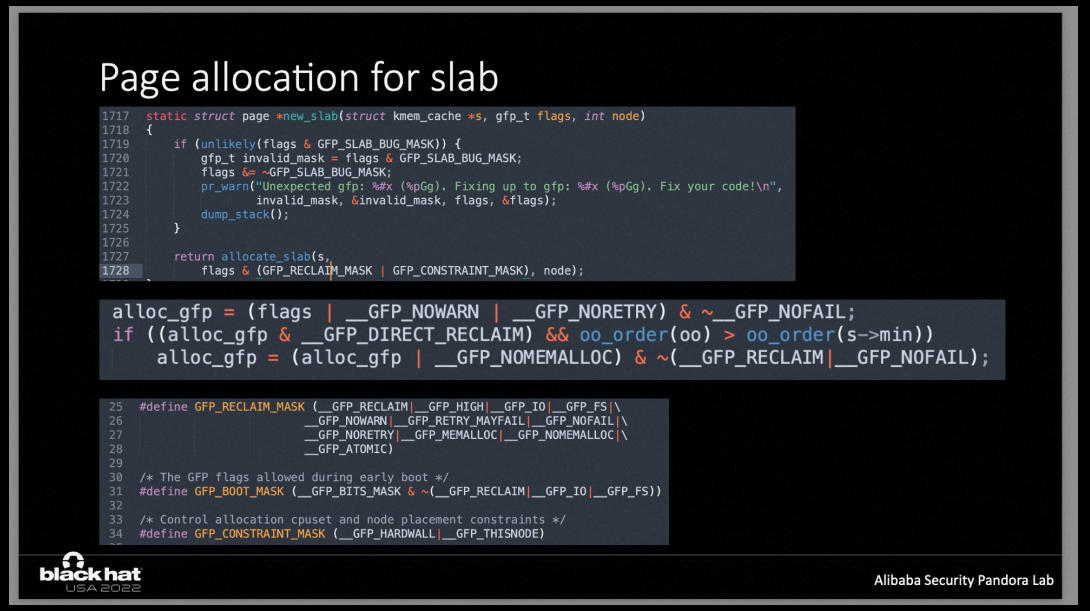
Module-signing is not supported for GKI vendor modules. On devices required to support verified boot, Android requires kernel modules to be in the partitions that have dm-verity enabled. This removes the need for signing individual modules for their authenticity. Android 13 introduced the concept of GKI modules. GKI modules use the kernel's build time signing infrastructure to differentiate between GKI and other modules at run time. Unsigned modules are allowed to load as long as they only use symbols appearing on the allowlist or provided by other unsigned modules. To facilitate GKI modules signing during GKI build using kernel's build time key pair, GKI kernel config has enabled `CONFIG_MODULE_SIG_ALL=y`. To avoid signing non-GKI modules during device kernel builds, you must add `# CONFIG_MODULE_SIG_ALL is not set` as part of your kernel config fragments.

https://source.android.com/docs/core/architecture/kernel/loadable-kernel-modules

Page UAF

- Physical pages Use-After-Free
  - In theory, all the pages within the free state can be imported and reused
- Hijack a kernel object
  - MIGRATE_UNMOVABLE VS MIGRATE_MOVABLE





http://i.blackhat.com/USA-22/Thursday/US-22-WANG-Ret2page-The-Art-of-Exploiting-Use-After-Free-Vulnerabilities-in-the-Dedicated-Cache.pdf

Page UAF in GPU MMU

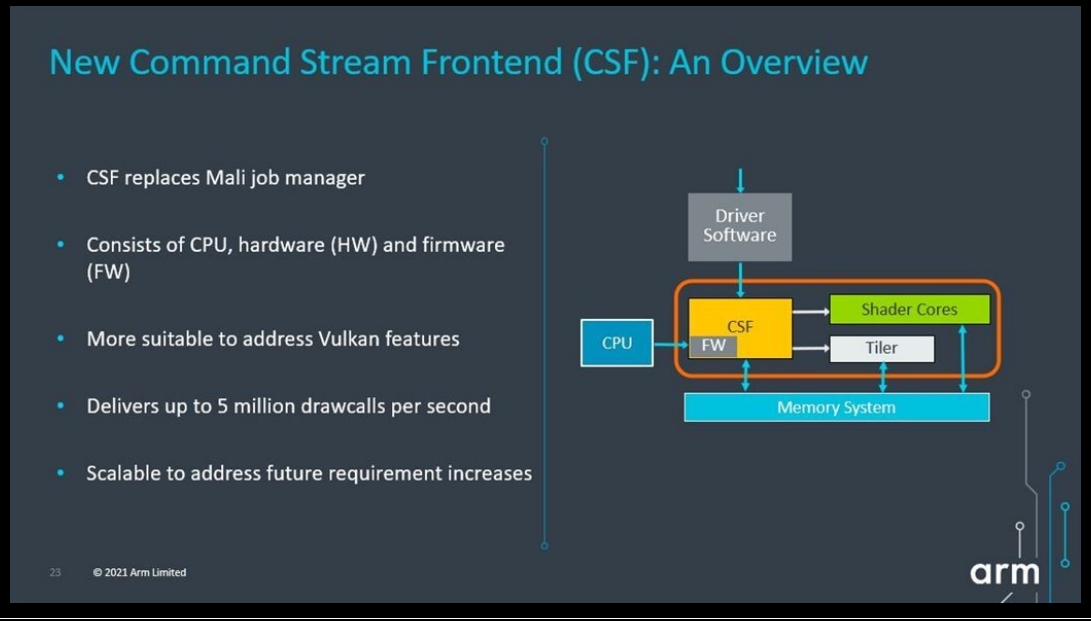
CVE-2021-28663

GPU_WR ^ CPU_WR

- 使用GPU读写内存?
  - CVE-2021-28663/Mali
    - 被引用的NATIVE内存被释放后仍然能够通过ALIAS内存访问
    - ALIAS内存还可以映射到用户态,但是禁用了KBASE_REG_CPU_WR
    - 在驱动中找不到使用GPU读写内存的指令😭
  - CVE-2021-28664/Mali
    - 映射用户态只读页面为可写🤔

GPU READ/WRITE

- Require reverse-engineering the GPU instruction sets
  - https://gitlab.freedesktop.org/panfrost
  - https://github.blog/2022-07-27-corrupting-memory-without-memory-corruption/
- New features and enhancements




```
#if MALI_USE_CSF
    case KBASE_IOCTL_CS_EVENT_SIGNAL:
        KBASE_HANDLE_IOCTL(KBASE_IOCTL_CS_EVENT_SIGNAL,
                kbasep_cs_event_signal,
                kctx);
        break;
    case KBASE_IOCTL_CS_QUEUE_REGISTER:
        KBASE_HANDLE_IOCTL_IN(KBASE_IOCTL_CS_QUEUE_REGISTER,
                kbasep_cs_queue_register,
                struct kbase_ioctl_cs_queue_register,
                kctx);
        break;
    case KBASE_IOCTL_CS_QUEUE_REGISTER_EX:
        KBASE_HANDLE_IOCTL_IN(KBASE_IOCTL_CS_QUEUE_REGISTER_EX,
                kbasep_cs_queue_register_ex,
                struct kbase_ioctl_cs_queue_register_ex,
                kctx);
        break;
```

GPU READ/WRITE

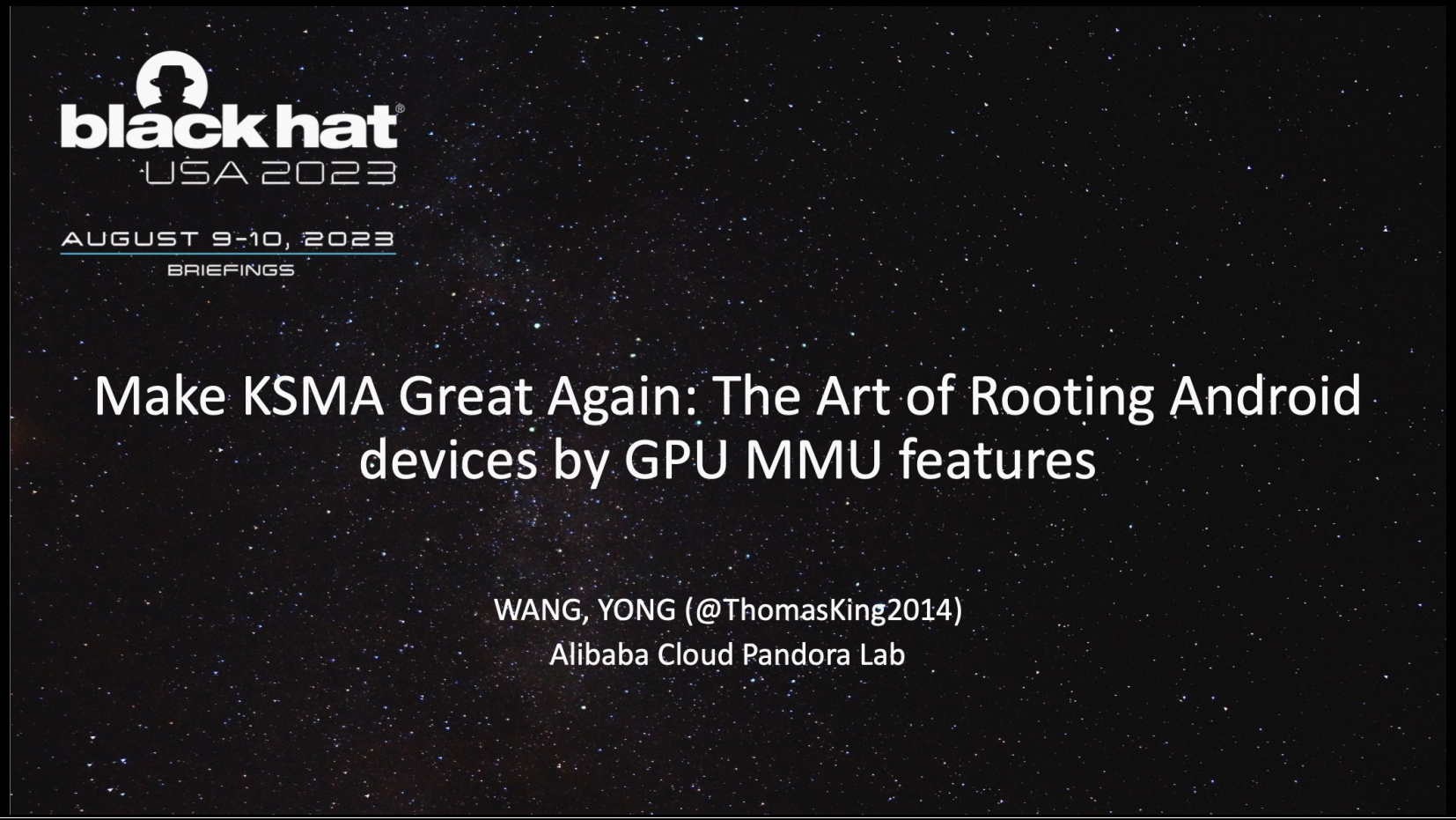
- OpenCL(Open Computing Language)
  - A framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators
  - Provides a standard interface for parallel computing using task- and data-based parallelism
  - Specifies programming languages (based on C99, C++14 and C++17) for programming abovementioned devices

GPU READ/WRITE

```
const char* gpu_code =
    "__kernel void rw_mem(__global unsigned long *p0, __global unsigned long *p1, __global unsigned long *p2) {”  // p0 – dest, p1 – src, p2 – rw_flag
    "   size_t idx = get_global_id(0);"
    "   if (p2[idx]) {” // write
    "       __global unsigned long *addr = (__global unsigned long)(p0[idx]);"
    "       addr[0] = p1[idx];"
    "   } else {” // read
    "       __global unsigned long *addr = (__global unsigned long *)(p1[idx]);"
    "       p0[idx] = addr[0];"
    "   }"
    "}";
```

Page UAF in GPU MMU

- How to craft the valid entry



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;
    }
```

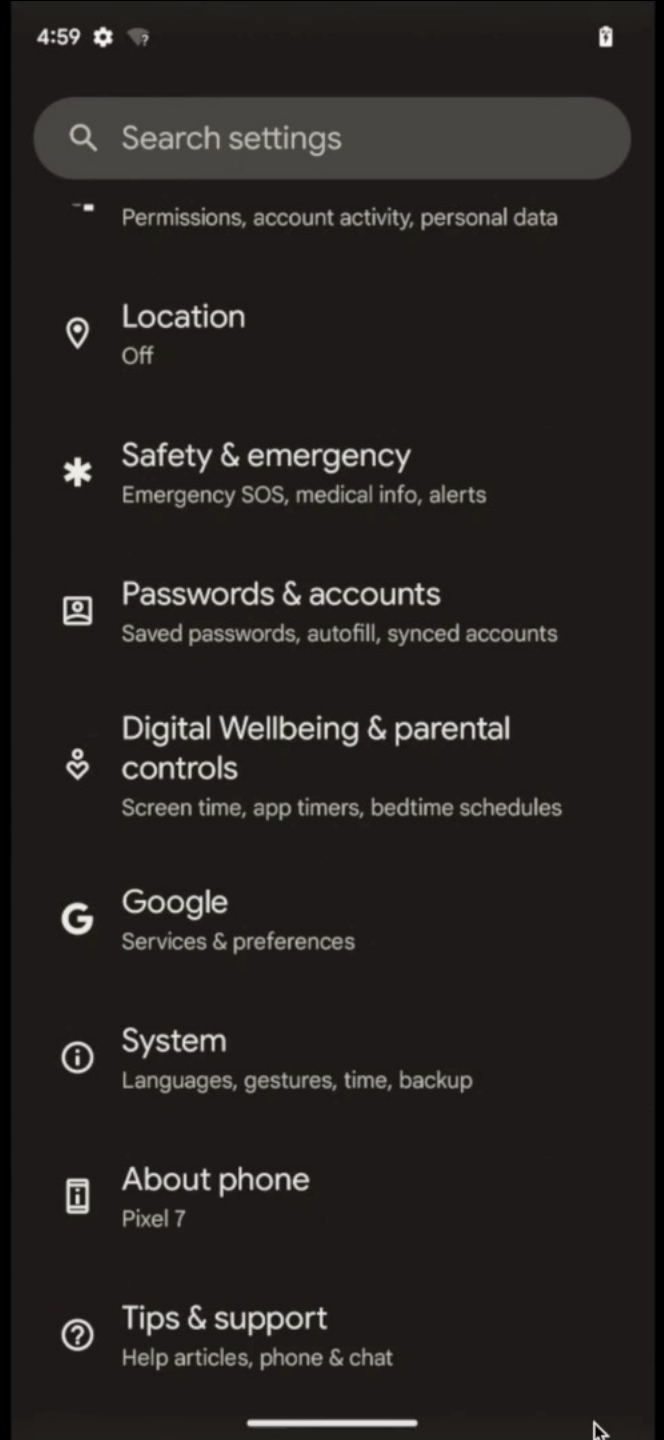
```
static phys_addr_t kbase_mmu_alloc_pgd(struct kbase_device *kbdev,
        struct kbase_mmu_table *mmut)
{
    u64 *page;
    struct page *p;
    phys_addr_t pgd;

    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;
```


4:59
Search settings
Permissions, account activity, personal data
Location
Off
Safety & emergency
Emergency SOS, medical info, alerts
Passwords & accounts
Saved passwords, autofill, synced accounts
Digital Wellbeing & parental controls
Screen time, app timers, bedtime schedules
Google
Services & preferences
System
Languages, gestures, time, backup
About phone
Pixel 7
Tips & support
Help articles, phone & chat

Agenda

- Introduction
- Bug #1
- Bug #2
- *Conclusion*

Takeaways

- Analyzing the old bug is always an efficient way to find a new one.
- Memory corruption is good, logic bug is better.
- Even with more and more both hardware and software mitigations, Android rooting is still possible.

References

[1] https://www.blackhat.com/docs/eu-16/materials/eu-16-Taft-GPU-Security-Exposed.pdf

[2] https://googleprojectzero.github.io/0days-in-the-wild//0day-RCAs/2021/CVE-2021-39793.html

[3] https://source.android.com/docs/core/architecture/kernel/loadable-kernel-modules

[4] https://labs.bluefrostsecurity.de/blog/2020/04/08/cve-2020-0041-part-2-escalating-to-root/

[5] https://i.blackhat.com/USA-22/Thursday/US-22-WANG-Ret2page-The-Art-of-Exploiting-Use-After-Free-Vulnerabilities-in-the-Dedicated-Cache.pdf

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

[7] <<Android GPU内存管理拾遗补阙>>

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