GRAYSHIFT

Analysis and Exploitation of CVE-2021-28664 for Android Privilege Escalation

Bernard Lampe, Ph.D.

Android Vulnerability Researcher and Manager

October 28, 2021

Who am I

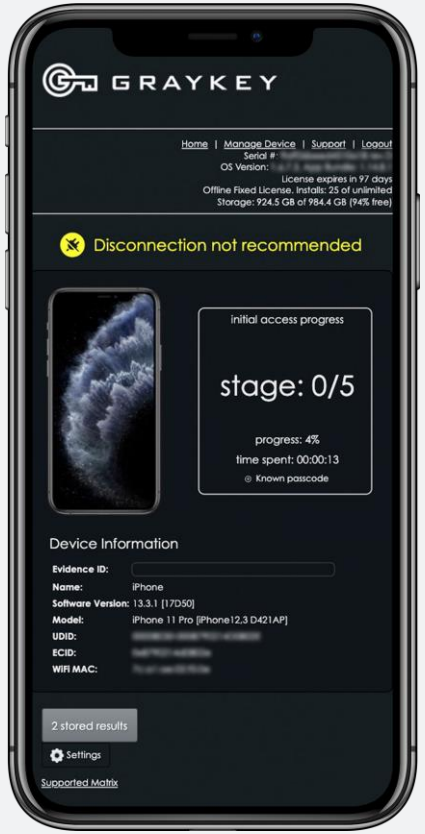
- Professional Vulnerability Researcher
  - Joined Grayshift May 2020
  - Android Vulnerability Manager
  - Work at home
- Vulnerability Research Contractor for 10 years
  - Browsers, Embedded Devices, Linux, Android
  - Work in SCIF life
- Academic Researcher Electrical Engineering
  - Hyperspectral and Compressive Sensing

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

GRAYKEY

- ACCESS LAWFULLY.
- DISCOVER SWIFTLY.
- PROTECT JUSTLY.
- Grayshift has developed GrayKey, a state-of-the-art mobile forensic access tool, that extracts encrypted or inaccessible data from mobile devices.
- Android coverage growing each release



GRAYSHIFT

© 2021. Grayshift, LLC. All rights reserved.

Vulnerability Researchers at Grayshift

- Our team of vulnerability researchers is dedicated to building innovative technology to gain “device access.”
- Develop N-day and researching novel 0-day
  - Initial Access
  - Elevating Privileges
  - Cryptoanalysis
- Goal is forensic data extraction and to preserve chain of custody of data

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Bug chains

- IA + Priv + Cryptoanalysis represents defense in depth
- Address each problem separately or break into subproblems
- Chain capabilities together to get data extraction
- Don’t crash, reset, or modify the phone
- Today’s talk focus on privesc on the application processor (AP)
  - Linux version linux-4.19.135-mali, arm64, and mali version v_r20p0.
- Only taking you on part of the exploit chain journey today

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

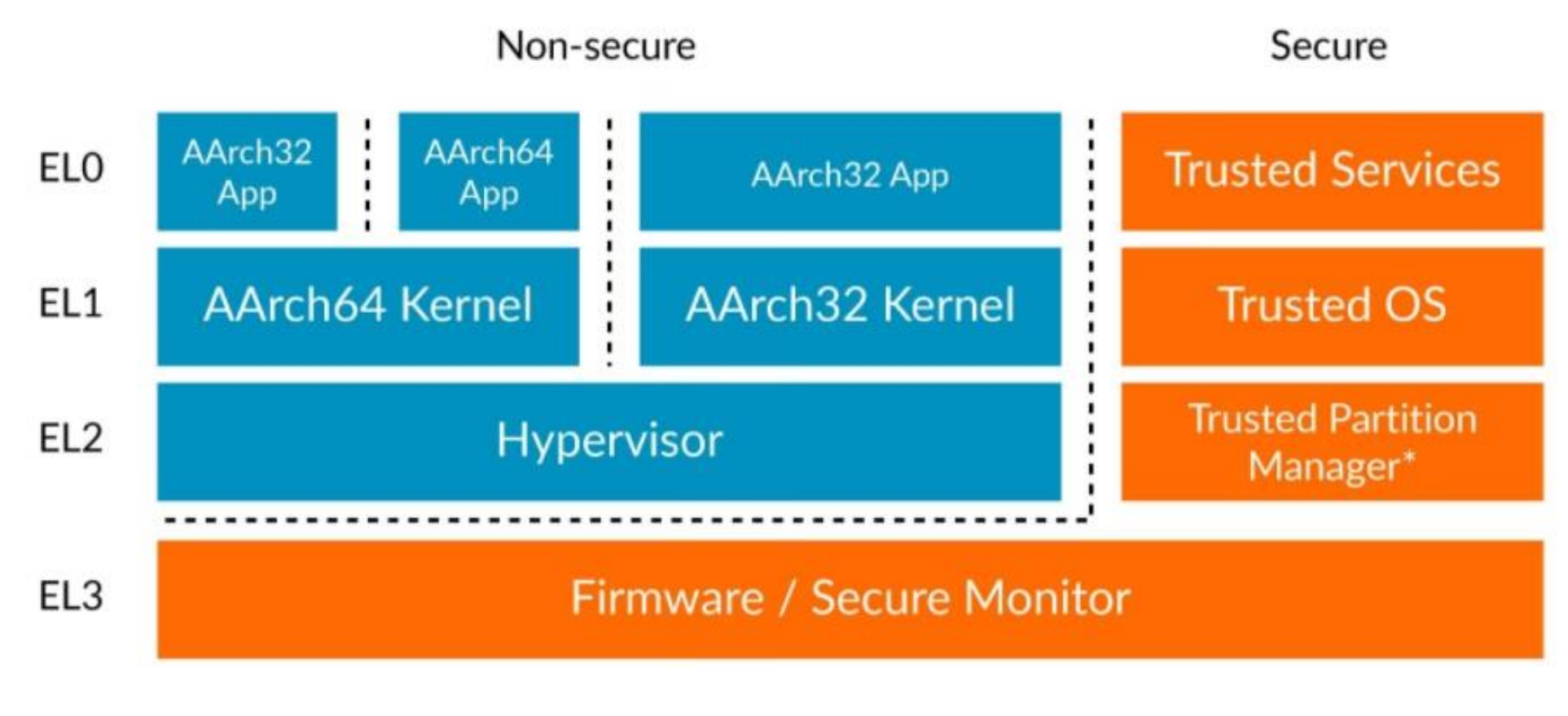
Privesc Attack Surfaces

- Go from low to high privs, but what is in our way?
- Linux process isolation and memory isolation
- Linux process and file privileges
  - Linux file and process permissions, Linux capabilities
  - SELinux: u:r:untrusted_app:s0, u:r:isolated_app:s0, u:r:shell:s0, u:r:vendor_shell:s0, many many more…
  - SECCOMP syscall filtering
  - Vendor specific protections, RKP, DEFEX, TIMA, etc, etc
- Starting context in peripheral device
  - USB, Modem, Bluetooth, NFC
  - IOMMUs enforce virtual memory isolation
  - Execution isolation achieved by being on a separate piece of hardware

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

ARM AP Exception Levels

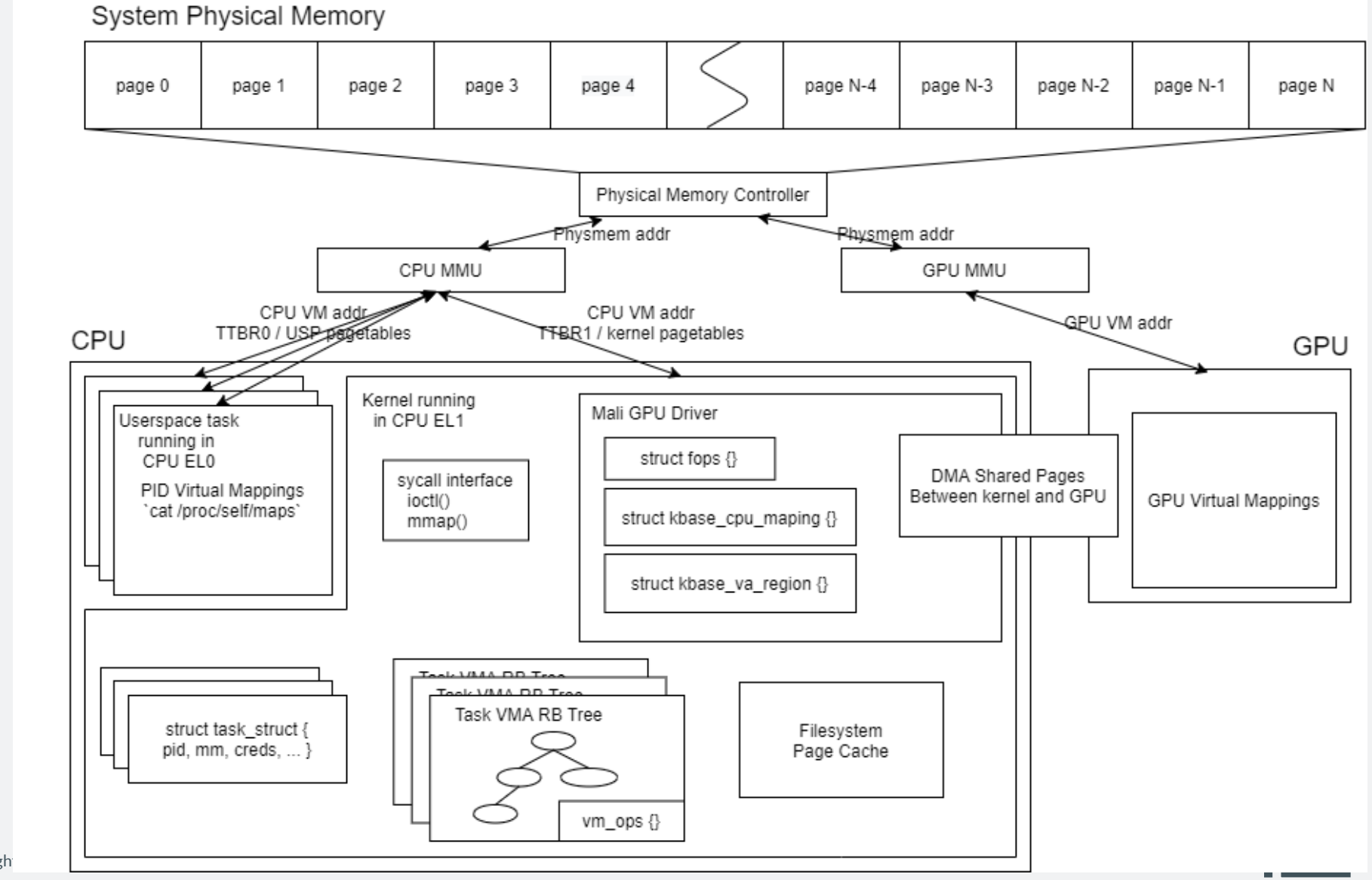


https://developer.arm.com/documentation/102412/0100/Execution-and-Security-states

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Background System Concepts



© 2021. Grayshift, LLC. All righ

GRAYSHIFT

Syscall Absolute Minimum

- Syscalls are a way to call kernel functions from userspace securely to enforce process isolation

On boot setup VBAR_EL1

```
// Exception vectors.
ENTRY(vectors)
<snip>
    kernel_ventry   0, sync         // Synchronous 64-bit EL0
    kernel_ventry   0, irq          // IRQ 64-bit EL0
    kernel_ventry   0, fiq_invalid  // FIQ 64-bit EL0
    kernel_ventry   0, error        // Error 64-bit EL0
<snip>
END(vectors)
```

```
arch/arm64/kernel/head.S
------------------------------
adr_l   x8, vectors // load VBAR_EL1 with virtual
msr vbar_el1, x8   // vector table address
isb
```

During EL0 exec

```
asm{
    mov x8, __NR_ioctl,
    svc #0
};
```

```
arch/arm64/kernel/entry.S: el0_sync()
  arch/arm64/kernel/entry.S:el0_svc()
    arch/arm64/kernel/syscall.c:el0_svc_handler(struct pt_regs *regs)
      arch/arm64/kernel/syscall.c:el0_svc_common(struct pt_regs *regs, …)
        arch/arm64/kernel/syscall.c:invoke_syscall(struct pt_regs *regs, …)
          arch/arm64/kernel/syscall.c:__invoke_syscall(struct pt_regs *regs, …)
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Page Fault Absolute Minimum

- Virtual memory is a way to isolate process memory from each other
- Kernel tracks memory regions by struct vma_area_struct *vma
- When you see vma, think kernel record and vm_ops for faulting

```
arch/arm64/kernel/entry.S: el0_sync()
    arch/arm64/kernel/entry.S:el0_da()
        arch/arm64/mm/fault.c:do_mem_abort(unsigned long addr, unsigned int esr, struct pt_regs *regs)
            arch/arm64/mm/fault.c:do_translation_fault(unsigned long addr, unsigned int esr, struct pt_regs *regs)
                arch/arm64/mm/fault.c:do_page_fault(unsigned long addr, unsigned int esr, struct pt_regs *regs)
                    arch/arm64/mm/fault.c:__do_page_fault(struct mm_struct *mm, unsigned long addr, ...)
                        handle_mm_fault(struct vm_area_struct *vma, unsigned long address, unsigned int flags)
                            __handle_mm_fault(struct vm_area_struct *vma, unsigned long address, unsigned int flags)
                            handle_pte_fault(struct vm_fault *vmf)
                                do_anonymous_page(struct vm_fault *vmf)
                                do_fault(struct vm_fault *vmf)
                                    do_shared_fault()
                                        __do_fault(struct vm_fault *vmf)
                                            vma->vm_ops->fault(vmf)
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Kernel Record of VMA

- Kernels keeps lists of virtual memory areas that are mapped in your processes
- Keep access permission in flags fields
- Keep vm_ops function pointers around to populate memory when needed

```
struct vm_area_struct
{
    unsigned long vm_start; /* Our start address within vm_mm. */
    unsigned long vm_end;   /* The first byte after our end address within vm_mm. */
<snip>
    const struct vm_operations_struct *vm_ops;
<snip>
    pgprot_t vm_page_prot;   /* Access permissions of pte's assigned to this VMA. */
    unsigned long vm_flags;  /* Arch independent flags such as VM_READ, VM_WRITE
    unsigned long vm_pgoff;  /* Offset (within vm_file) in PAGE_SIZE units */
    struct file * vm_file;   /* File we map to (can be NULL). */
<snip>
}
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Driver Record of VMA called kbase_va_region

- Mali driver keeps lists of virtual memory areas that are imported
- Keeps separate access permission in flags fields

```
// mali kbase_va_region struct from mali_kbase_mem.h
 struct kbase_va_region {
 <snip>

     u64 start_pfn;      /* The PFN in GPU space */
     size_t nr_pages;

 <snip>

     unsigned long flags;

 <snip>

     struct kbase_mem_phy_alloc *cpu_alloc; /* the one alloc object we mmap to the CPU when mapping this region */
     struct kbase_mem_phy_alloc *gpu_alloc; /* the one alloc object we mmap to the GPU when mapping this region */

 <snip>

     int    va_refcnt; /* number of users of this va */
 };
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Driver Record of VMA Mapping

- Used when a GPU maps things back out
- Does not keep access permission flags
- Physical mappings keep track of pages and associated userspace virtual memory addresses

```
 /**
  * A CPU mapping
  */
  struct kbase_cpu_mapping {
     struct   list_head mappings_list;
     struct   kbase_mem_phy_alloc
*alloc;
     struct   kbase_context *kctx;
     struct   kbase_va_region *region;
     int      count;
     int      free_on_close;
  };
```

```
/**
  * A physical mapping
  */
struct kbase_mem_phy_alloc {
    union {
<snip>
        struct
kbase_alloc_import_user_buf {
            unsigned long address;
            unsigned long size;
            unsigned long nr_pages;
            struct page **pages;
 <snip>
            dma_addr_t *dma_addrs;
        } user_buf;
    } imported;
};
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Summary of Prerequisite Knowledge

- Userspace programs running at EL0
- Kernel and driver running at EL1
- Programs at EL0 talk to the kernel in EL1 using exceptions (syscalls and page faults)
- vm_area_structs (vmas) are how kernel keeps track of userspace program memory
- kbase_va_region are how the driver keeps track of userspace memory imported
- kbase_cpu_mapping are how the driver keeps track of userspace memory mappings
- kbase_mem_phy_alloc are how driver keeps track of physical pages
- **Foreshadowing:** There's a lot of separate bookkeepers of the userspace memory

I hope they all do accounting consistently

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Why GPU Vulnerabilities? – Ben Hawkes

- “From an attacker’s perspective, maintaining an Android exploit capability is a question of covering the widest possible range of Android ecosystem in the most cost-effective way possible.”
- “[…] there are only two implementations of GPU hardware that are particularly popular in Android devices: ARM Mali and Qualcomm Adreno.”
- “This means that if an attacker can find a nicely exploitable bug in these two GPU implementations, […]”
- “[…] GPU are highly complex with significant amount of closed-source components […]”

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Why GPU Vulnerabilities?

- Android kernels (and vendors) deploy exploit mitigations in layers
  - SEAndroid, SECCOMP, DEFEX, RKP, TIMA, etc
- GPUs are complicated attack surface
  - Many fops and vm_ops {open, ioctl, mmap, etc}
  - Shared memory is hard to get right
- DAC and MAC make it accessible from u:r:untrusted_app:s0, and u:r:shell:s0
- Kernel drivers historically poor code quality

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

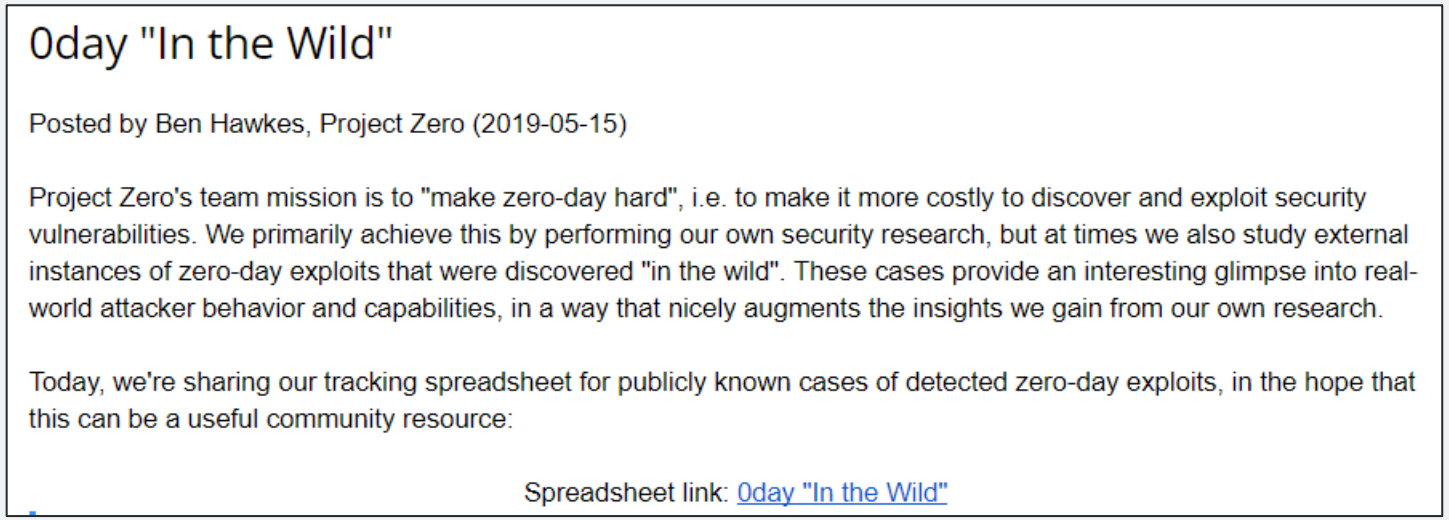
Other GPU Vulnerabilities – Background Research

- Ben Hawkes, Project Zero, Qualcomm Adreno GPU memory mapping use-after-free
  - CVE-2021-1905, UAF of struct could lead to freeing used pages
  - https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2021/CVE-2021-1905.html
- Ben Hawkes, Project Zero, Attacking the Qualcomm Adreno GPU
  - CVE-2020-11179, Global memory mapping tricks GPU again, allowing kernel r/w
  - https://googleprojectzero.blogspot.com/2020/09/attacking-qualcomm-adreno-gpu.html
- Guang Gong, An Exploit Chain to Remotely Root Modern Android Devices
  - CVE-2019-10567, Global memory mapping corrupts GPU, allowing kernel r/w
  - https://i.blackhat.com/USA-20/Thursday/us-20-Gong-TiYunZong-An-Exploit-Chain-To-Remotely-Root-Modern-Android-Devices-wp.pdf
- Justin Taft, NCC Group, GPU Security Exposed
  - CVE-2016-2067, Access flag logic error allow userspace ro pages r/w
  - https://www.blackhat.com/docs/eu-16/materials/eu-16-Taft-GPU-Security-Exposed.pdf

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Tracking 0-day “In the Wild”



0day "In the Wild"

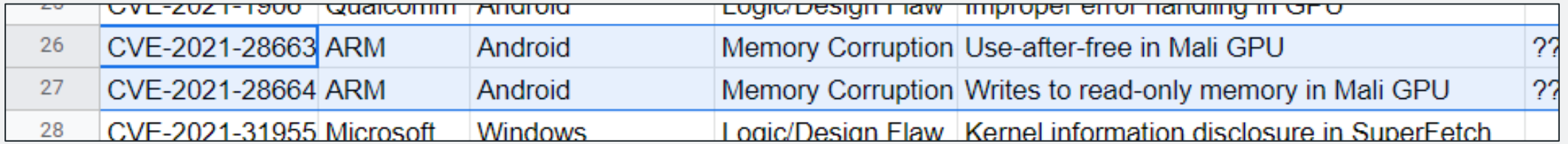
Posted by Ben Hawkes, Project Zero (2019-05-15)

Project Zero's team mission is to "make zero-day hard", i.e. to make it more costly to discover and exploit security vulnerabilities. We primarily achieve this by performing our own security research, but at times we also study external instances of zero-day exploits that were discovered "in the wild". These cases provide an interesting glimpse into real-world attacker behavior and capabilities, in a way that nicely augments the insights we gain from our own research.

Today, we're sharing our tracking spreadsheet for publicly known cases of detected zero-day exploits, in the hope that this can be a useful community resource:

Spreadsheet link: 0day "In the Wild"

https://googleprojectzero.blogspot.com/p/0day.html



26	CVE-2021-28663	ARM	Android	Memory Corruption	Use-after-free in Mali GPU	??
27	CVE-2021-28664	ARM	Android	Memory Corruption	Writes to read-only memory in Mali GPU	??
28	CVE-2021-31955	Microsoft	Windows	Logic/Design Flaw	Kernel information disclosure in SuperFetch	

https://docs.google.com/spreadsheets/d/1lkNJ0uQwbeC1ZTRrxdtuPLCIl7mlUreoKfSIgajnSyY/edit#gid=2129022708

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Google Android Security Bulletin

ARM components

These vulnerabilities affect ARM components and further details are available directly from ARM. The severity assessment of these issues is provided directly by ARM.

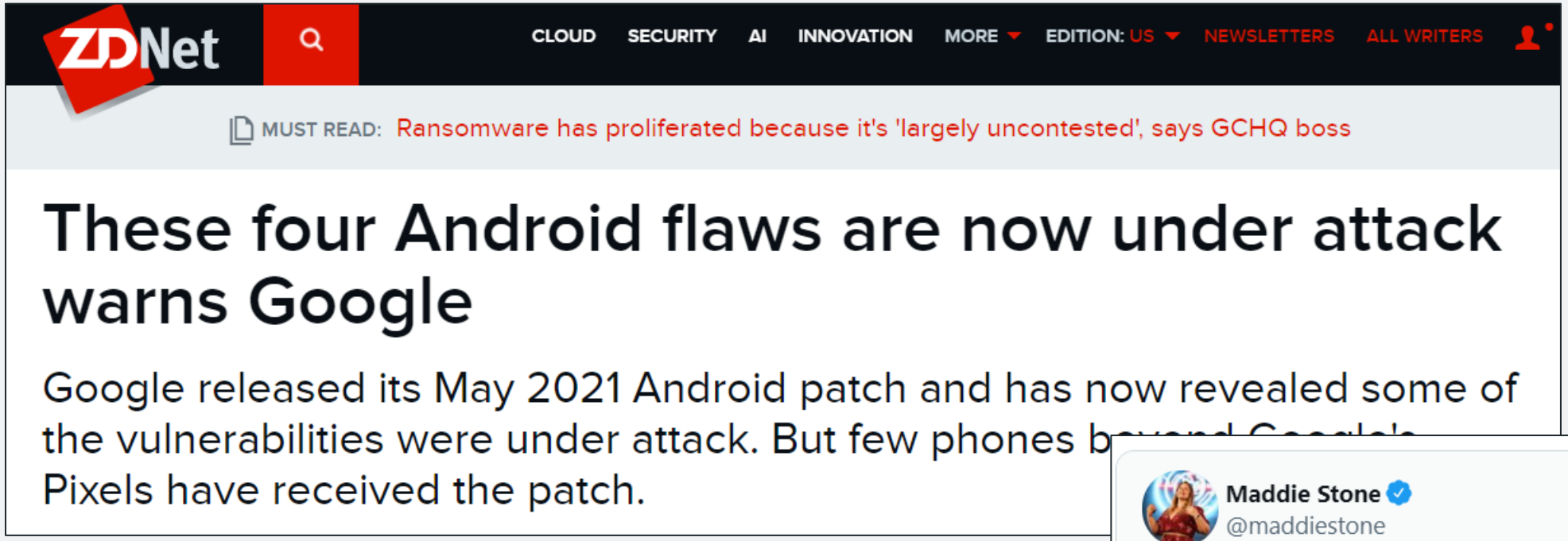
CVE	References	Severity	Component
CVE-2021-28663	A-174259860*	High	Mali
CVE-2021-28664	A-174588870*	High	Mali

https://source.android.com/security/bulletin/2021-05-01

© 2021. Grayshift, LLC. All rights reserved.

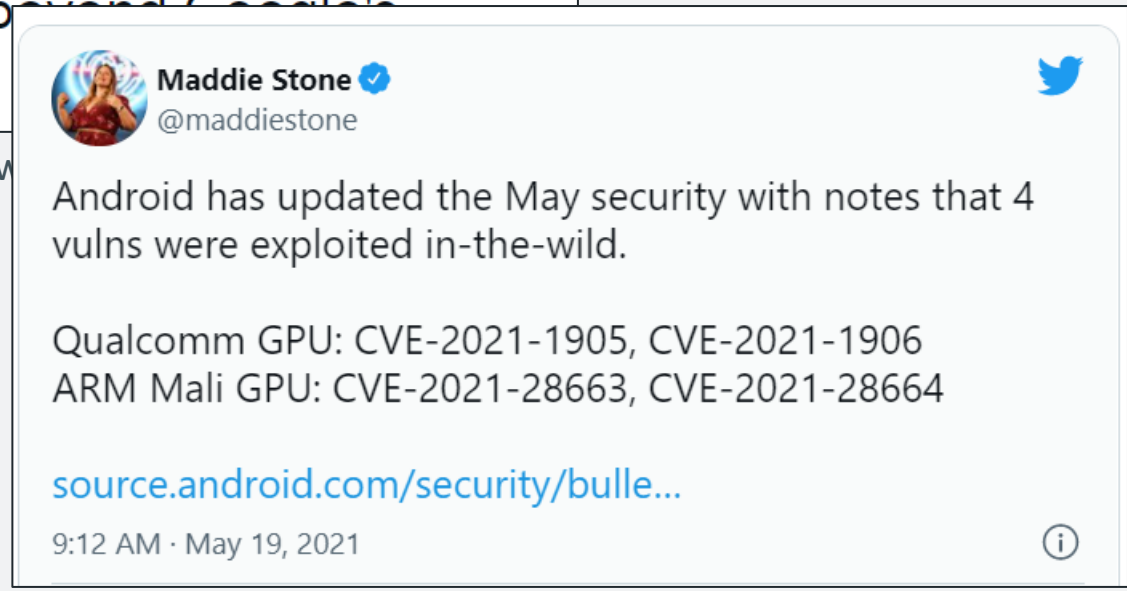
GRAYSHIFT

GPU Bugs in the Press



https://www.zdnet.com/article/google-warns-these-four-android-flaws-are-now

https://twitter.com/maddiestone/status/1395004346996248586?s=20



© 2021. Grayshift, LLC. All rights reserved.

ARM Vuln Report CVE-2021-28664

arm Developer

Overview Report Security Vulnerabilities

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 – r28p0 Valhall GPU Kernel Driver: All versions from r19p0 - r28p0
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 r29p0. It will be fixed in future Midgard release. Users are recommended to upgrade if they are impacted by this issue.
Credit	n/a

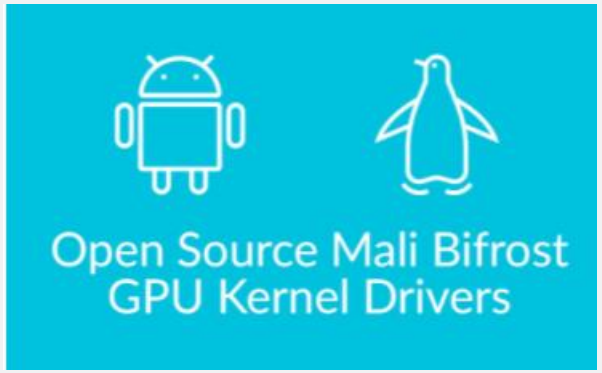
https://developer.arm.com/support/arm-security-updates/mali-gpu-kernel-driver

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Mali GPU Source Code

- Utgard, Miggard, Bifrost GPU Versions and Drivers



Download GPU Kernel Device Drivers

By downloading the packages below you acknowledge that you accept the End User License Agreement for the Mali GPUs Kernel Device Drivers Source Code.

BX304L01B-SW-99002-r33p0-01eac0.tar	Kernel Device Driver for Linux r33p0-01eac0. Released on 20th September 2021	4.70 MB
BX304L01B-SW-99002-r32p0-01eac0.tar	Kernel Device Driver for Linux r32p0-01eac0. Released on 22nd July 2021	4.60 MB
BX301A01B-SW-99002-r31p0-01eac0.tar	Kernel Device Driver for Linux r31p0-01eac0. Released on 17th June 2021	4.50 MB
BX304L01B-SW-99002-r31p0-	Kernel Device Driver for Android r31p0-01eac0. Released on 17th	4.50

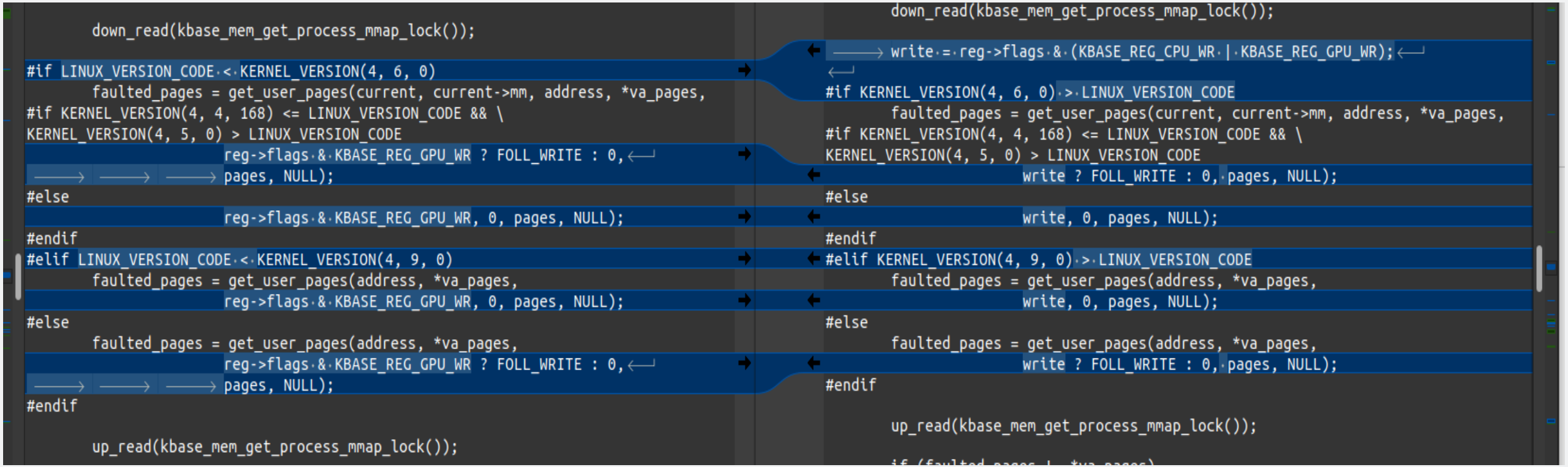
https://developer.arm.com/tools-and-software/graphics-and-gaming/mali-drivers/bifrost-kernel

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Mali GPU Source Patch Diff

mali_kbase_mem_linux.c:



```
        down_read(kbase_mem_get_process_mmap_lock());

#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

        up_read(kbase_mem_get_process_mmap_lock());
```

```
        down_read(kbase_mem_get_process_mmap_lock());

        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

        up_read(kbase_mem_get_process_mmap_lock());

        if (faulted_pages != *va_pages)
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Mali GPU Source Patch Diff

mali_kbase_mem_linux.c: struct kbase_va_region * kbase_mem_from_user_buffer(kctx, address, size, *va_pages, *flags)

BX304L01B-SW-99002-r28p0-01eac0-android

```
  down_read(kbase_mem_get_process_mmap_lock());

#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

    up_read(kbase_mem_get_process_mmap_lock());
```

BX304L01B-SW-99002-r31p0-01eac0

```
  down_read(kbase_mem_get_process_mmap_lock());

    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

    up_read(kbase_mem_get_process_mmap_lock());
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Mali GPU Source Patch Diff

mali_kbase_mem_linux.c: struct kbase_va_region * kbase_mem_from_user_buffer(kctx, address, size, *va_pages, *flags)

BX304L01B-SW-99002-r28p0-01eac0-android

```
down_read(kbase_mem_get_process_mmap_lock());

  faulted_pages = get_user_pages(address, *va_pages,
          reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
          pages, NULL);

  up_read(kbase_mem_get_process_mmap_lock());
```

BX304L01B-SW-99002-r31p0-01eac0

```
down_read(kbase_mem_get_process_mmap_lock());

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

  faulted_pages = get_user_pages(address, *va_pages,
          write ? FOLL_WRITE : 0, pages, NULL);

  up_read(kbase_mem_get_process_mmap_lock());
```

```
/**
 * get_user_pages() - pin user pages in memory
 * @start:      starting user address
 * @nr_pages:   number of pages from start to pin
 * @gup_flags:  flags modifying lookup behaviour
 * @pages:      array that receives pointers to the pages pinned.
 *              Should be at least nr_pages long. Or NULL, if caller
 *              only intends to ensure the pages are faulted in.
 * @vmas:       array of pointers to vmas corresponding to each page.
 *              Or NULL if the caller does not require them.
 *.
 */
long get_user_pages(unsigned long start, unsigned long nr_pages,
        unsigned int gup_flags, struct page **pages,
        struct vm_area_struct **vmas)
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

get_user_pages()

- “get_...” typically means to take a reference to mark a user and prevent free
- Looking up kernel docs on GUP:
  - get_user_pages walks a process's page tables and takes a reference to each struct page that each user address corresponds to at a given instant.
  - If write = 0, the page must not be written to.
- The write flag will check the page protections if set. Will not check if the page protections if not set.

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Implications of the Patch

- What is the purpose of encapsulating call?
  - Create a kbase_va_region tracking userspace memory
  - Copy in address, size and permission flags
  - Return the region and store it for later and give back a cookie for mmap

```
static struct kbase_va_region *kbase_mem_from_user_buffer(
        struct kbase_context *kctx, unsigned long address,
        unsigned long size, u64 *va_pages, u64 *flags)
```

```
int kbase_update_region_flags(
     struct kbase_context *kctx,
     struct kbase_va_region *reg,
     unsigned long flags)
{
<snip>
    if (flags & BASE_MEM_PROT_CPU_WR)
        reg->flags |= KBASE_REG_CPU_WR;

    if (flags & BASE_MEM_PROT_CPU_RD)
        reg->flags |= KBASE_REG_CPU_RD;

    if (flags & BASE_MEM_PROT_GPU_WR)
        reg->flags |= KBASE_REG_GPU_WR;

    if (flags & BASE_MEM_PROT_GPU_RD)
        reg->flags |= KBASE_REG_GPU_RD;
<snip>
}
```

```
/* return a cookie */
kctx->pending_regions[*gpu_va] = reg; <-- puts the region into array
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Implications of the Patch

- Sanity check is supposed to check if the pages have the right access permissions
- Developer assumption is that the write=0 flag enforces a RO check
  - In reality it just skips the check all together
- Therefore:
  - Attacker sends memory import flags to cause KBASE_REG_GPU_WR unset and KBASE_REG_CPU_WR set when copying to new region struct
- **The CPU write is recorded in the region, but not validated**

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

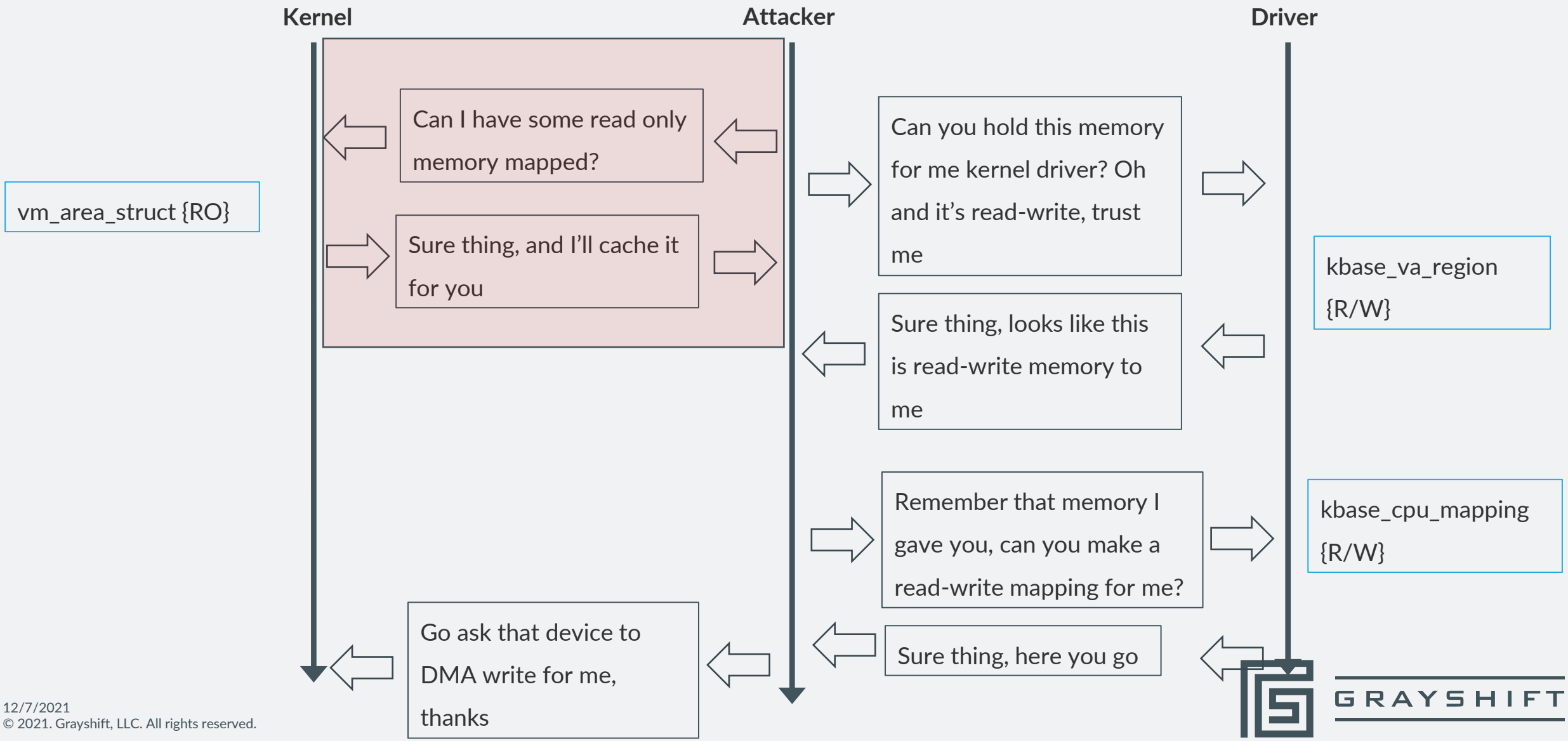
Call Trace to Patched Code – Calls and Args

```
drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:kbase_ioctl(struct file *filp,
                                                 unsigned int cmd,
                                                 unsigned long arg)

    drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:kbase_api_mem_import(struct kbase_context *kctx,
                                                                         union kbase_ioctl_mem_import *import)

        drivers/gpu/arm/v_r20p0/mali_kbase_mem_linux.c:kbase_mem_import(struct kbase_context *kctx,
                                                                        enum base_mem_import_type type,
                                                                        void __user *phandle,
                                                                        u32 padding,
                                                                        u64 *gpu_va,
                                                                        u64 *va_pages,
                                                                        u64 *flags)

            drivers/gpu/arm/v_r20p0/mali_kbase_mem_linux.c:kbase_mem_from_user_buffer(struct kbase_context *kctx,
                                                                                      unsigned long address,
                                                                                      unsigned long size,
                                                                                      u64 *va_pages,
                                                                                      u64 *flags)
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

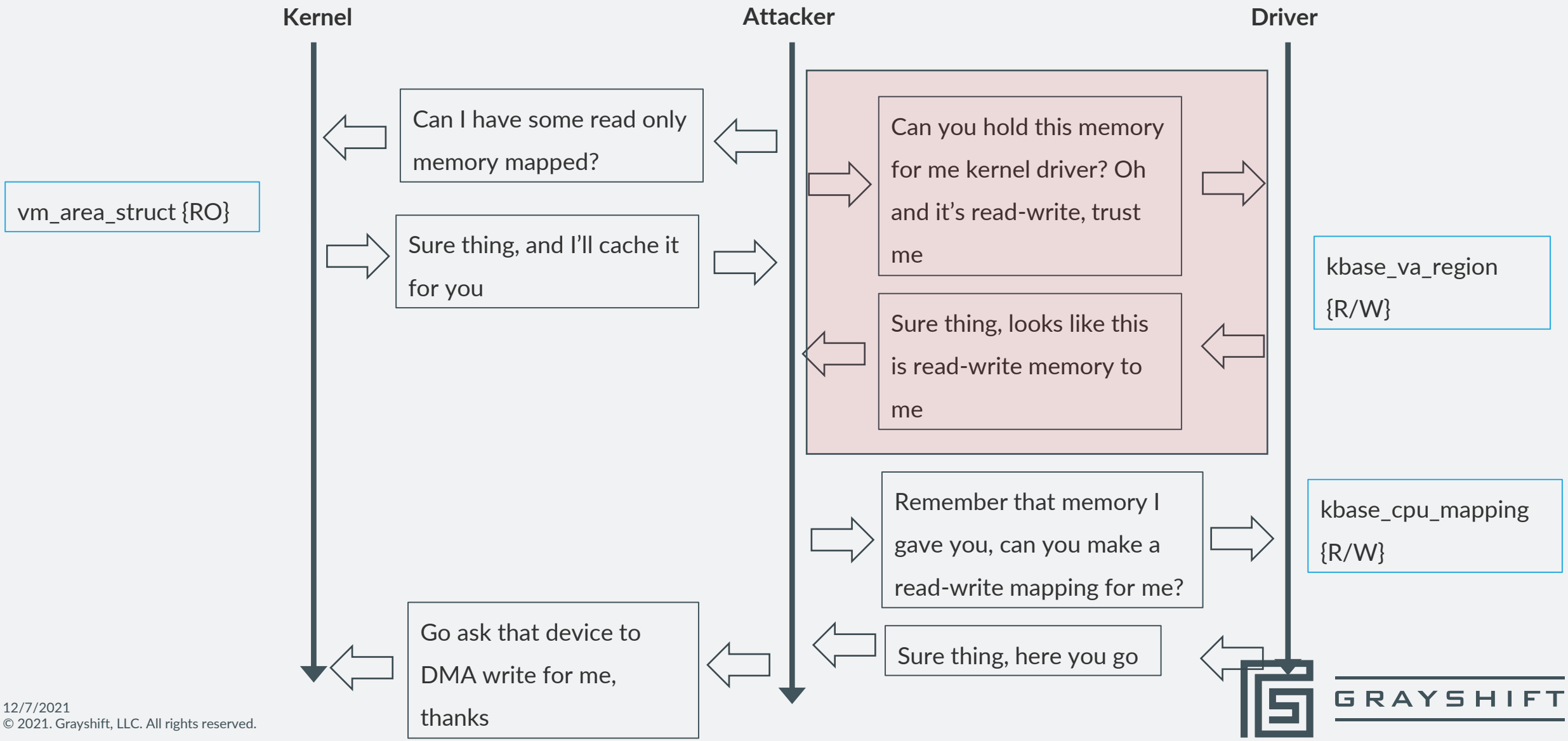
Where are we going?



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

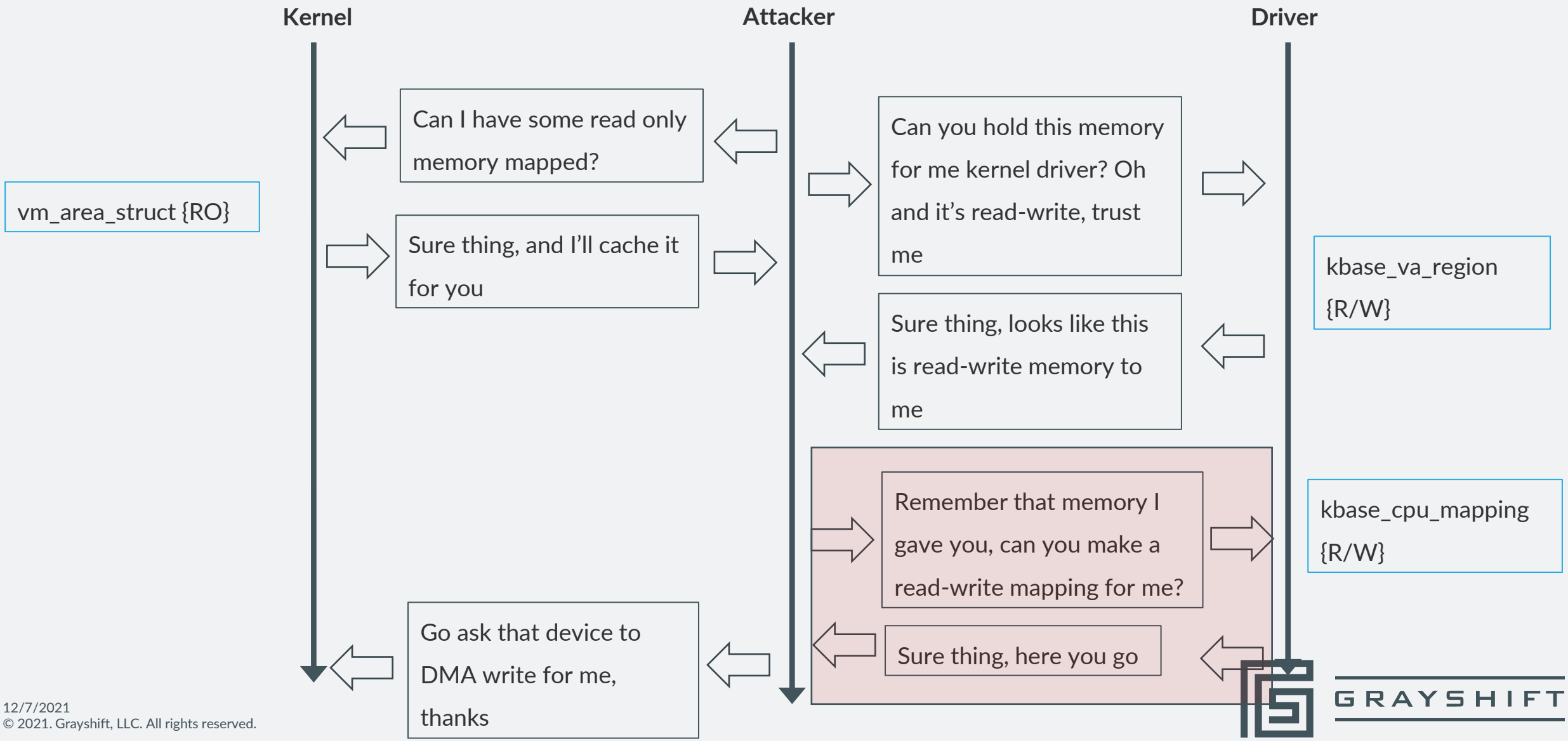
Where are we going?



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

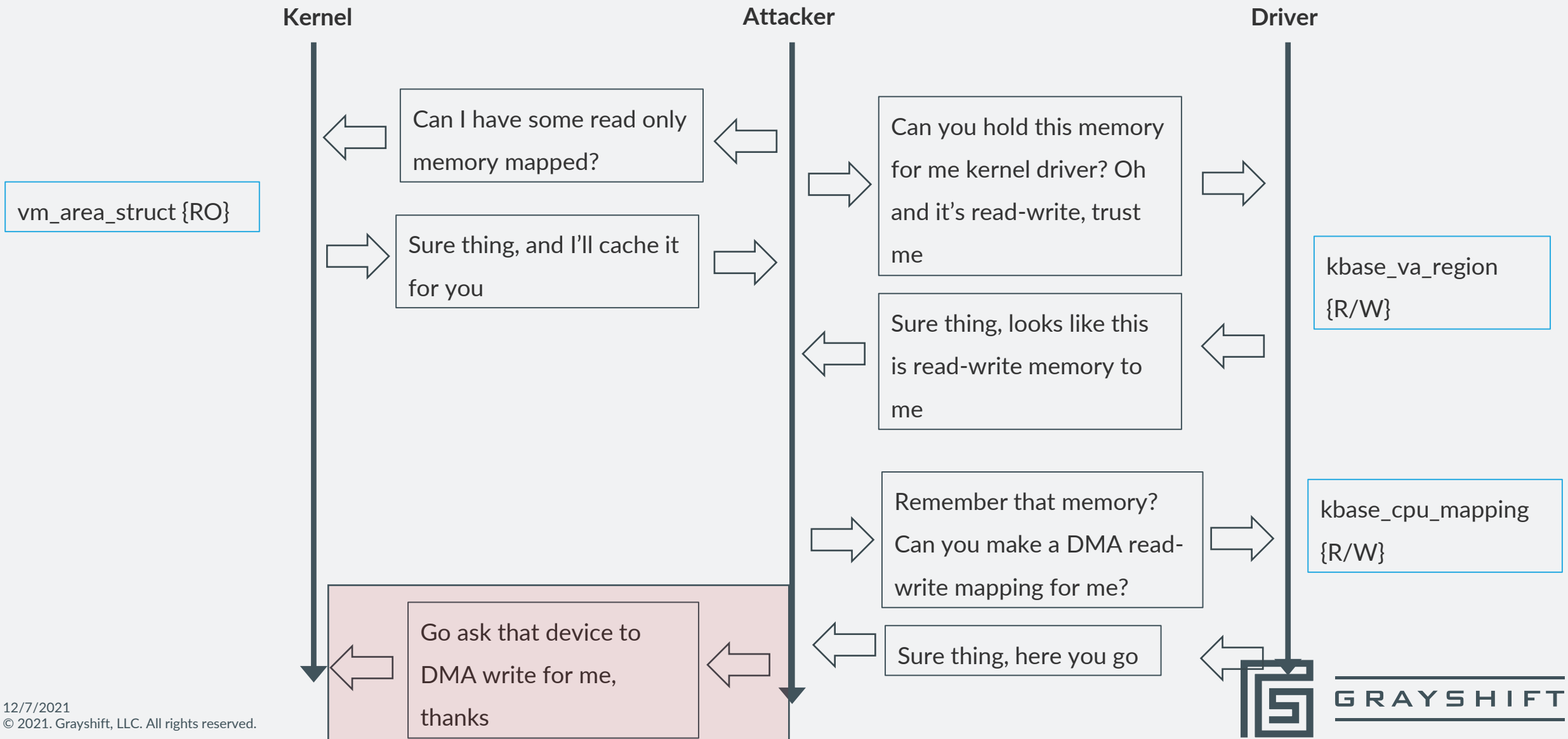
Where are we going?



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

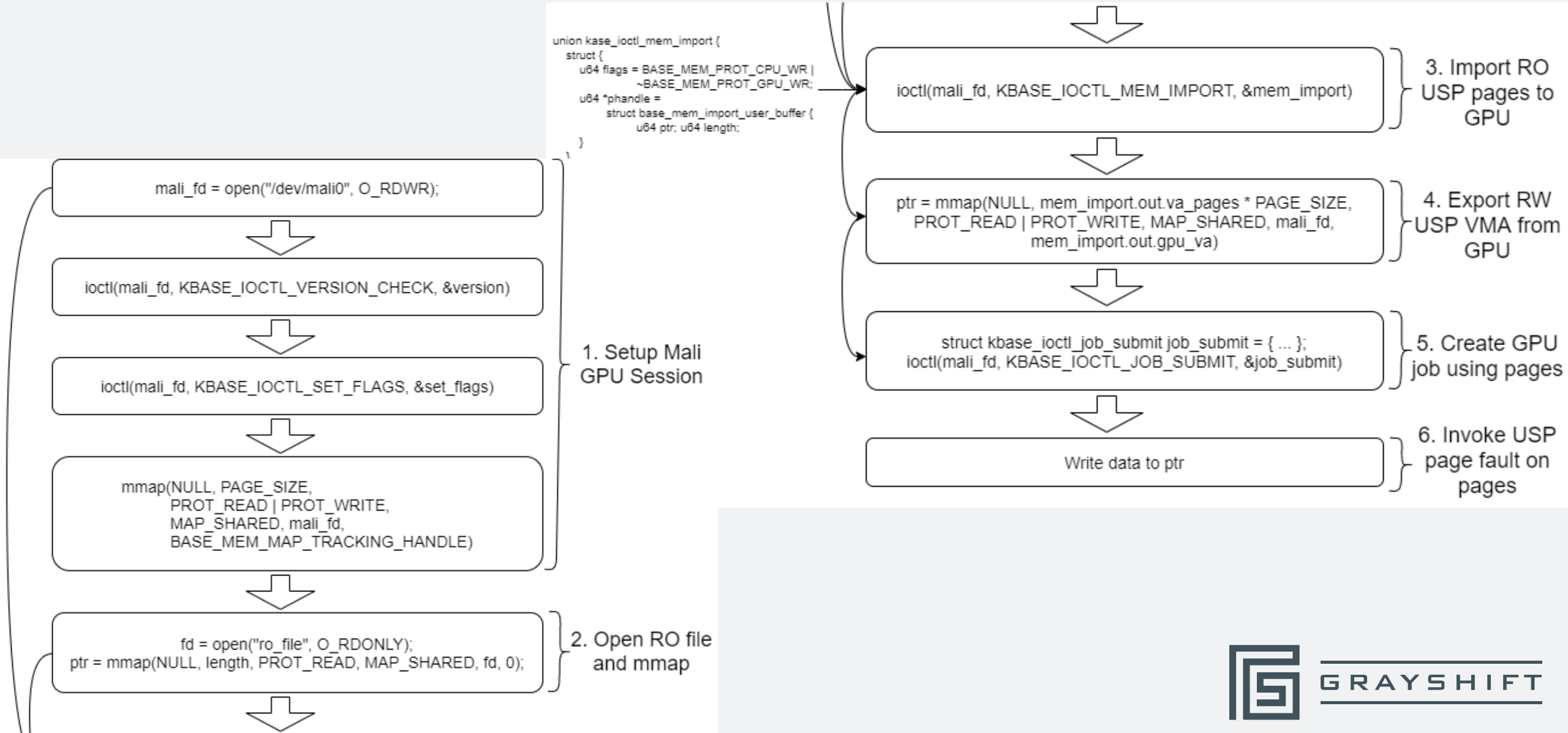
Where are we going?



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Primitive Userspace Calls



GRAYSHIFT

NoMali

- ARM has a hardware emulator
- Some ARM researchers profiled the driver without hardware and released the nomali simulator code for GEM5
  - https://github.com/ARM-software/nomali-model
  - https://ieeexplore.ieee.org/document/7482100

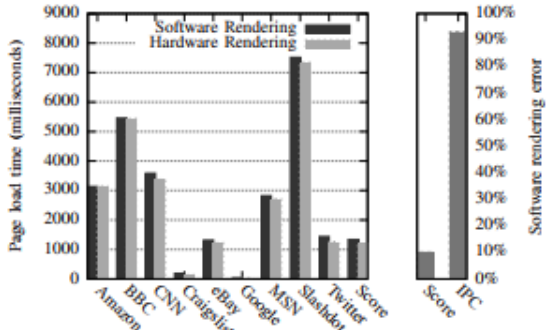
NoMali: Simulating a Realistic Graphics Driver Stack Using a Stub GPU

René de Jong
ARM Research
Cambridge
rene.dejong@arm.com

Andreas Sandberg
ARM Research
Cambridge
andreas.sandberg@arm.com

***Abstract*—Since the advent of the smartphone, all high-end mobile devices have required graphics acceleration in the form of a GPU. Today, even low-power devices such as smart-watches use GPUs for rendering and composition. However, the computer architecture community has largely ignored these developments when evaluating new architecture proposals.**

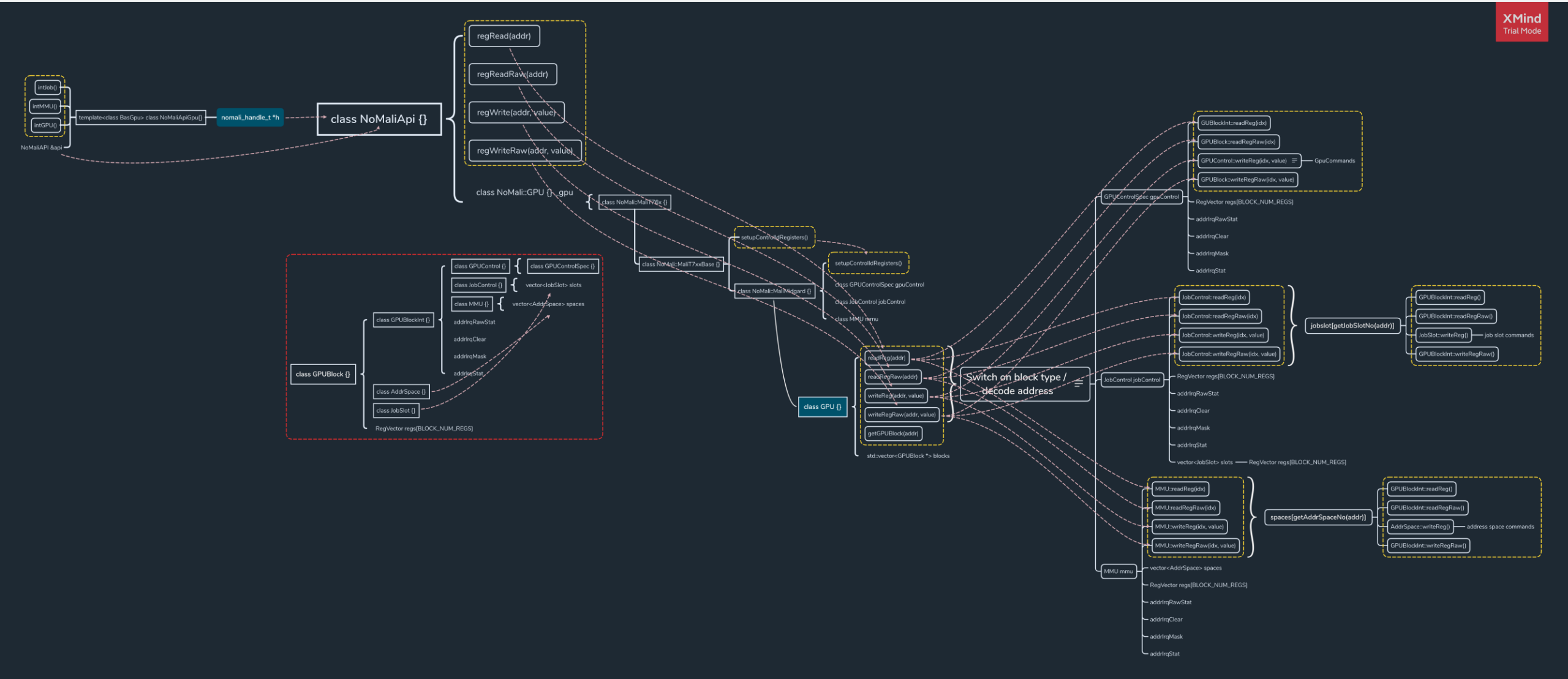
A common approach when evaluating CPU designs for the mobile space has been to use software rendering instead of a GPU model. However, due to the ubiquity of GPUs in mobile devices, they are used in both 3D applications *and* 2D applications. For example, when running a 2D application such as the web browser in Android with a software renderer instead of a GPU, the CPU ends up executing twice as many instructions. Both the CPU characteristics and the memory



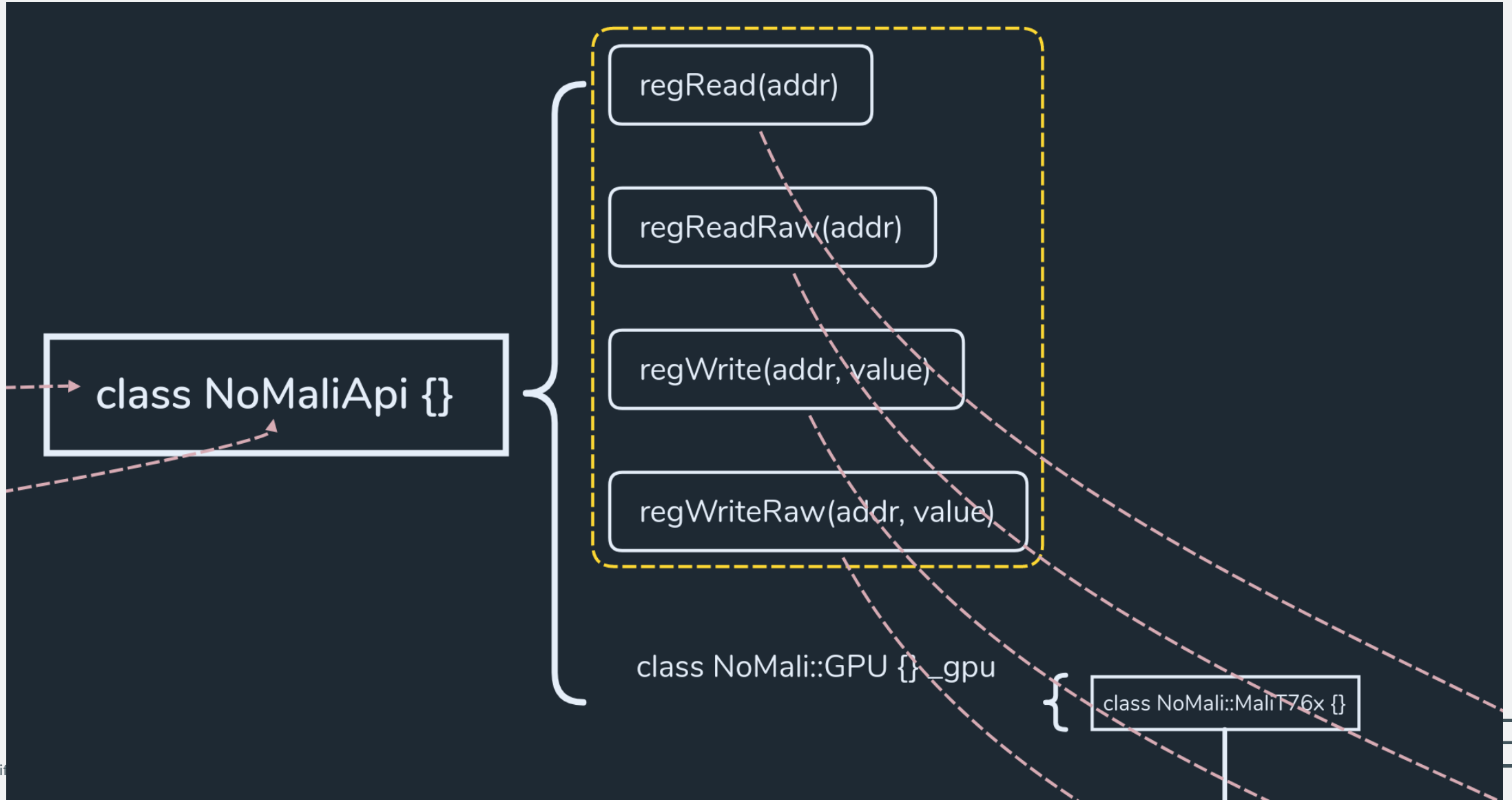
© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

NoMali – Xmind Map Code

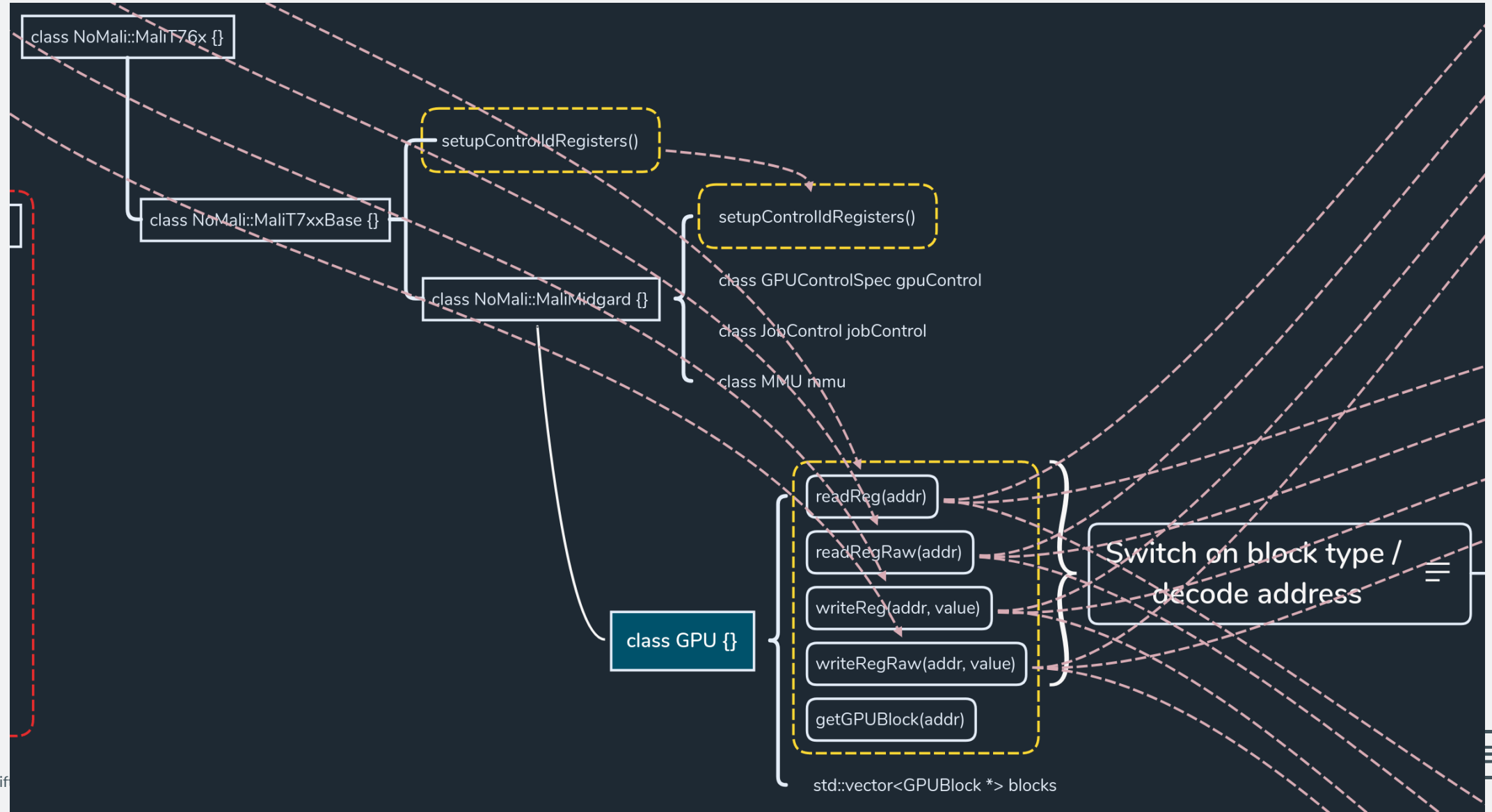


NoMali – Xmind Map Code



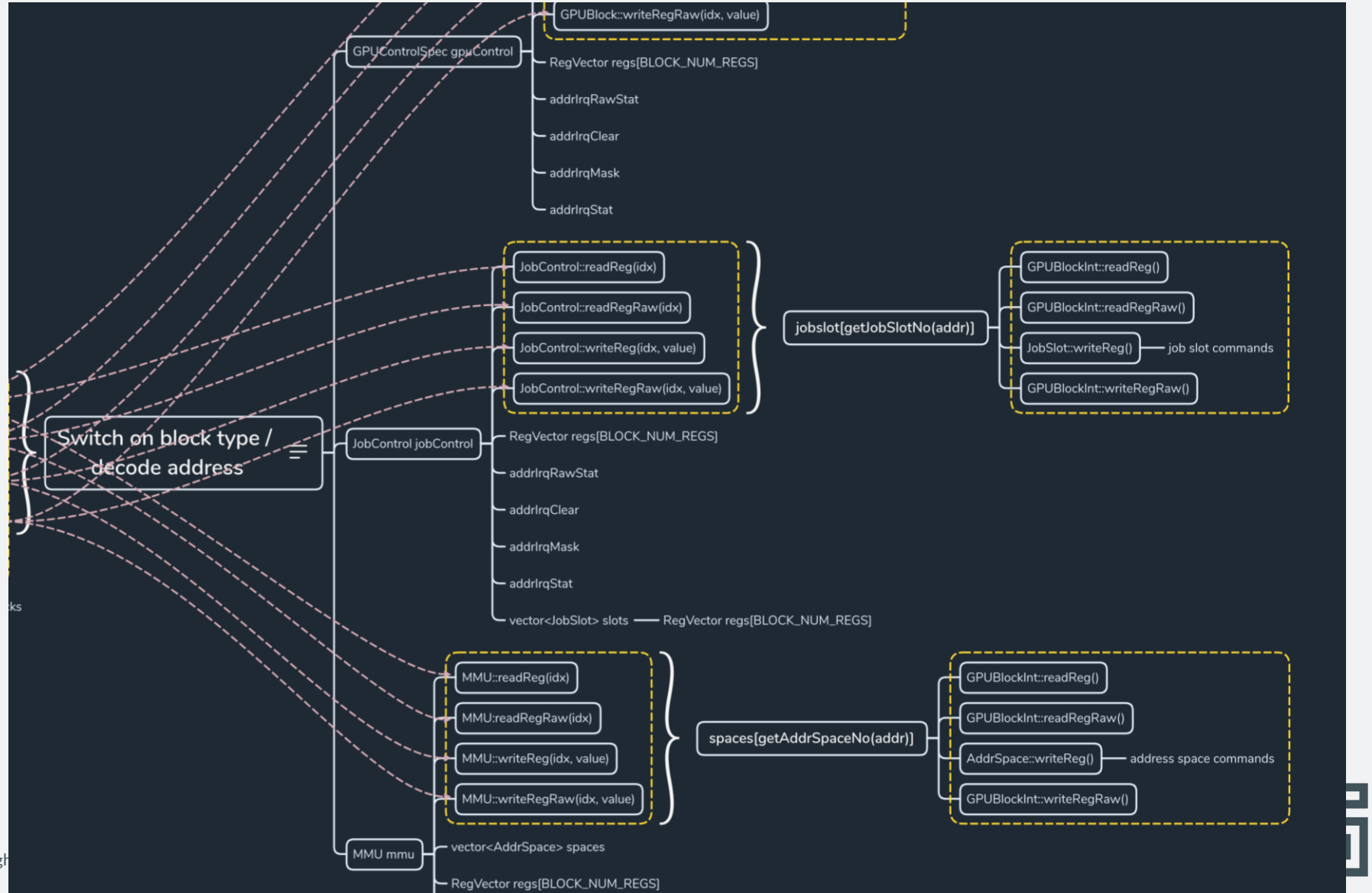
© 2021. Grayshif

NoMali – Xmind Map Code



© 2021. Grayshift

NoMali – Xmind Map Code



© 2021. Grayshift, LLC. All righ

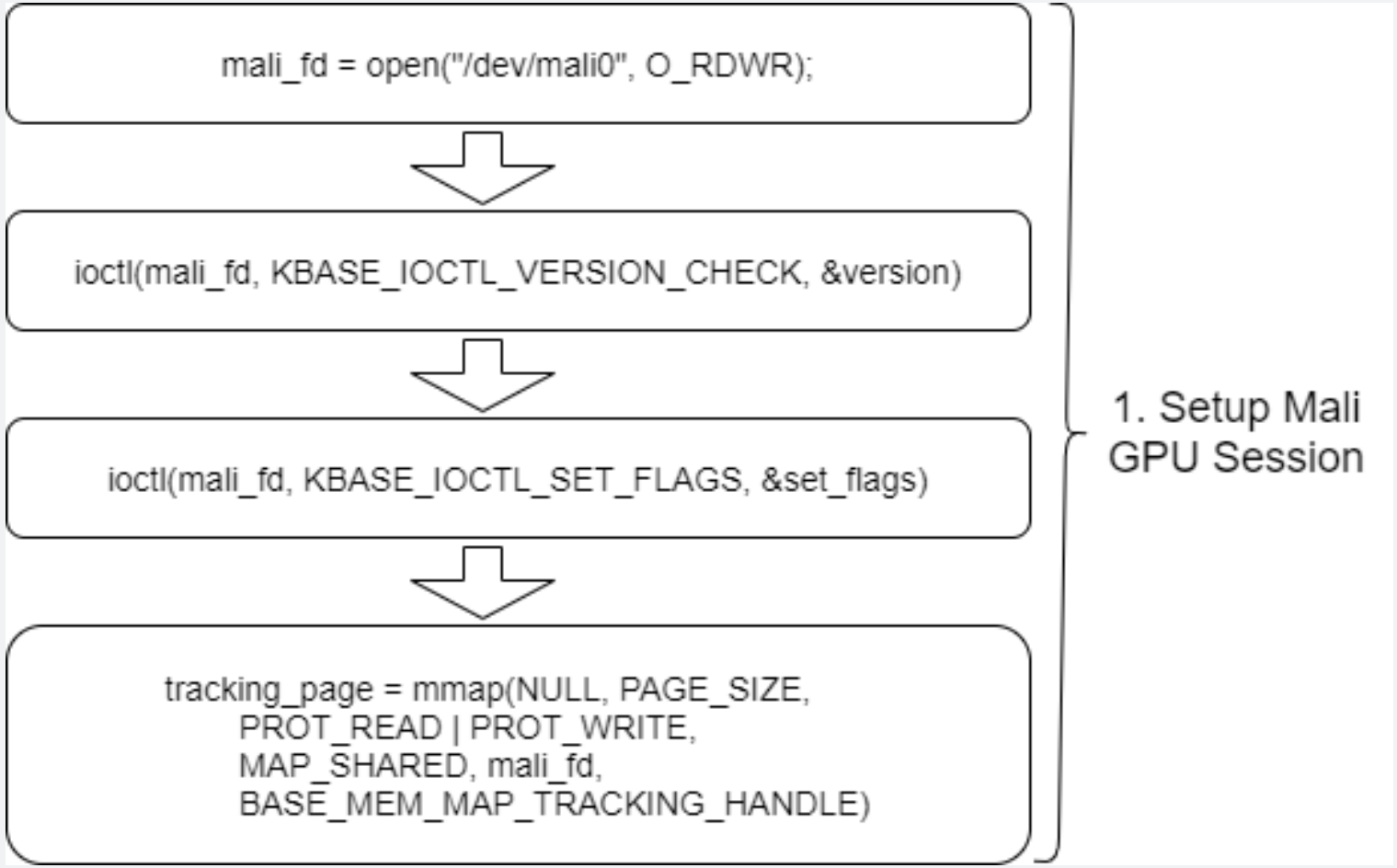
GRAYSHIFT

Emulator

- Import driver code into Linux source build system
- Hack up the PMIC code, temperature sensor code, and hardcode NoMali initialization in the driver

```
Breakpoint 1, kbase_mmap (filp=0xffff888139ff8d00, vma=0xffff88813af8ba80) at drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:1783
1783    {
(gdb) bt
#0  kbase_mmap (filp=0xffff888139ff8d00, vma=0xffff88813af8ba80) at drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:1783
#1  0xffffffff8116f608 in call_mmap (vma=<optimized out>, file=<optimized out>) at ./include/linux/fs.h:1826
#2  mmap_region (file=<optimized out>, addr=140062181584896, len=<optimized out>, vm_flags=<optimized out>, pgoff=<optimized out>, u
f=<optimized out>) at mm/mmap.c:1757
#3  0xffffffff8116fcd7 in do_mmap (file=<optimized out>, addr=<optimized out>, len=4096, prot=3, flags=1, vm_flags=251, pgoff=3, pop
ulate=0xffffc90000f27e88, uf=0xffffc90000f27e90) at mm/mmap.c:1530
#4  0xffffffff81155203 in do_mmap_pgoff (uf=<optimized out>, populate=<optimized out>, pgoff=<optimized out>, flags=<optimized out>,
 prot=<optimized out>, len=<optimized out>, addr=<optimized out>, file=<optimized out>) at ./include/linux/mm.h:2326
#5  vm_mmap_pgoff (file=0xffff888139ff8d00, addr=<optimized out>, len=<optimized out>, prot=3, flag=1, pgoff=3) at mm/util.c:357
#6  0xffffffff8116d9a8 in ksys_mmap_pgoff (addr=<optimized out>, len=4096, prot=3, flags=<optimized out>, fd=<optimized out>, pgoff=
3) at mm/mmap.c:1580
#7  0xffffffff8101ecf1 in __do_sys_mmap (off=<optimized out>, fd=<optimized out>, flags=<optimized out>, prot=<optimized out>, len=<
optimized out>, addr=<optimized out>) at arch/x86/kernel/sys_x86_64.c:100
#8  __se_sys_mmap (off=<optimized out>, fd=<optimized out>, flags=<optimized out>, prot=<optimized out>, len=<optimized out>, addr=<
optimized out>) at arch/x86/kernel/sys_x86_64.c:91
#9  __x64_sys_mmap (regs=<optimized out>) at arch/x86/kernel/sys_x86_64.c:91
#10 0xffffffff81002433 in do_syscall_64 (nr=<optimized out>, regs=0xffffc90000f27f58) at arch/x86/entry/common.c:293
#11 0xffffffff81a00078 in entry_SYSCALL_64 () at arch/x86/entry/entry_64.S:238
#12 0x0000000000000000 in ?? ()
(gdb)
```

How to talk to the GPU? - userspace



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

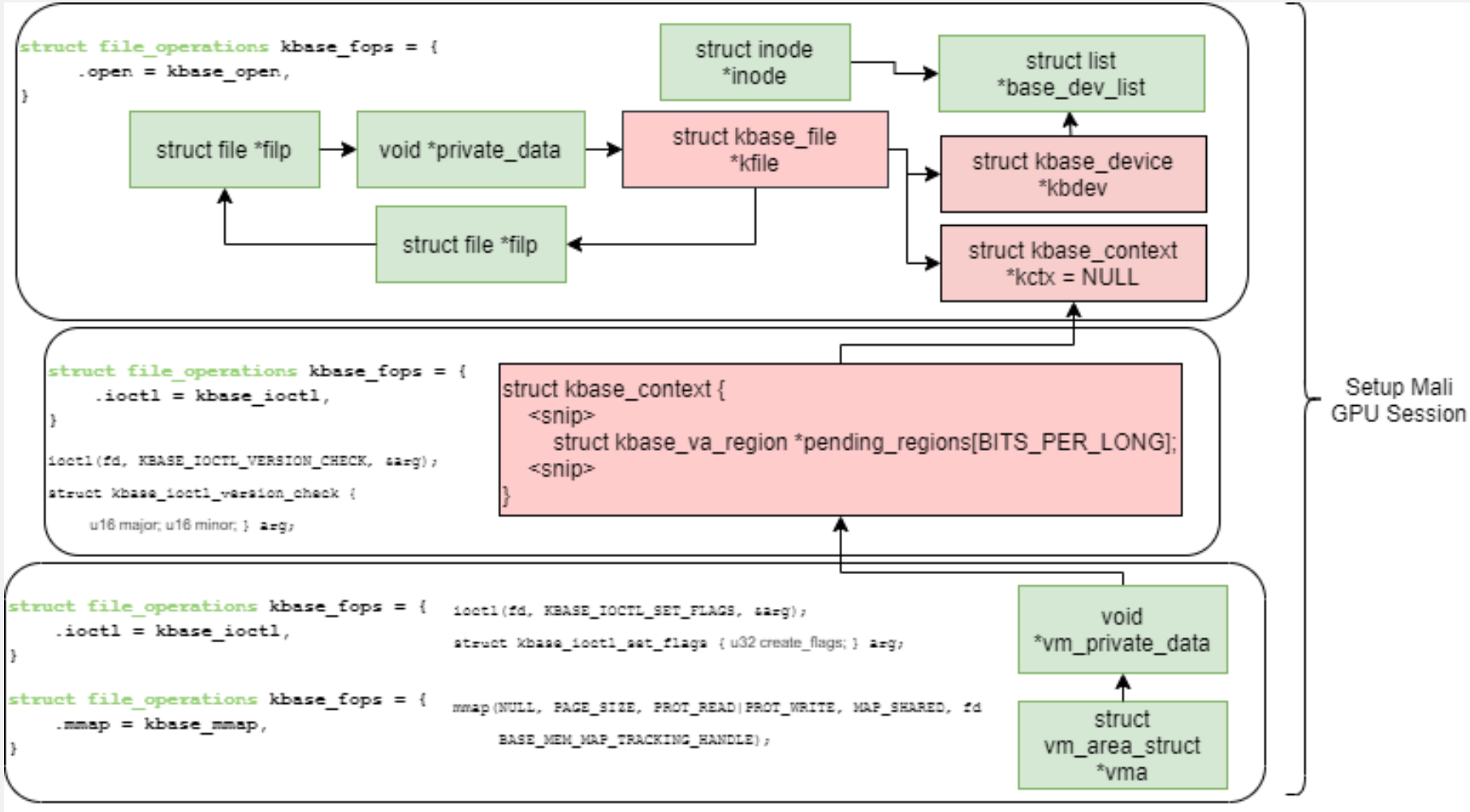
How to talk to the GPU? – driver entry

```
// drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c
static const struct file_operations kbase_fops = {
    .owner = THIS_MODULE,
    .open = kbase_open,
    .release = kbase_release,
    .read = kbase_read,
    .poll = kbase_poll,
    .unlocked_ioctl = kbase_ioctl,
    .compat_ioctl = kbase_ioctl,
    .mmap = kbase_mmap,
    .check_flags = kbase_check_flags,
    .get_unmapped_area = kbase_get_unmapped_area,
};
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

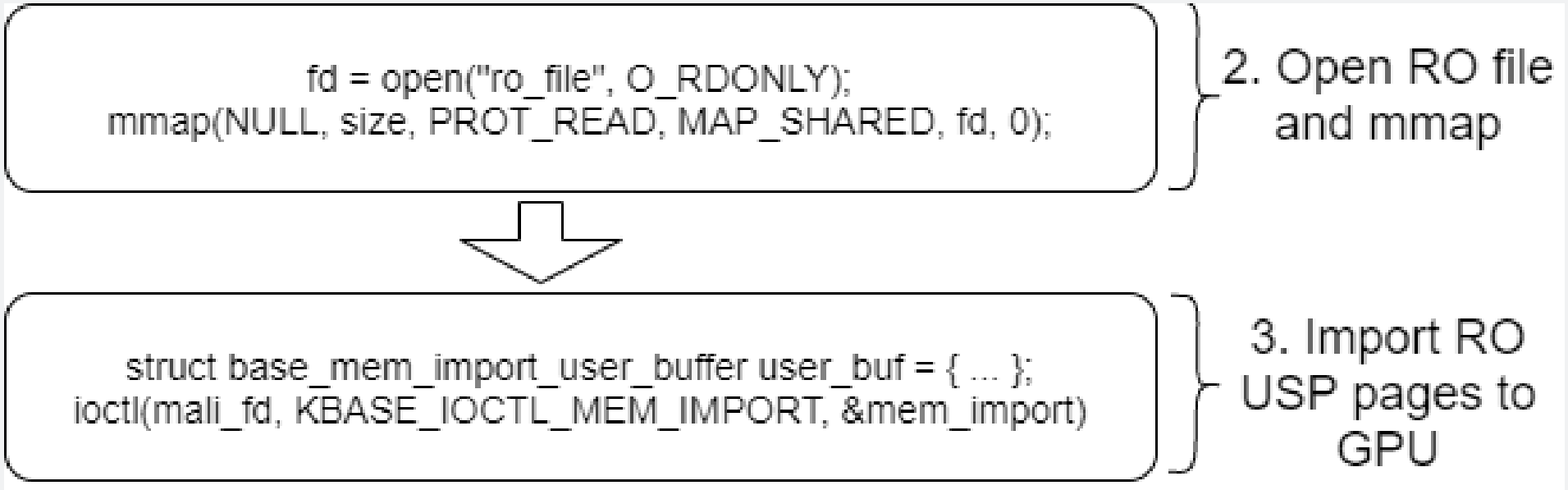
How to talk to the GPU? – kernel space



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

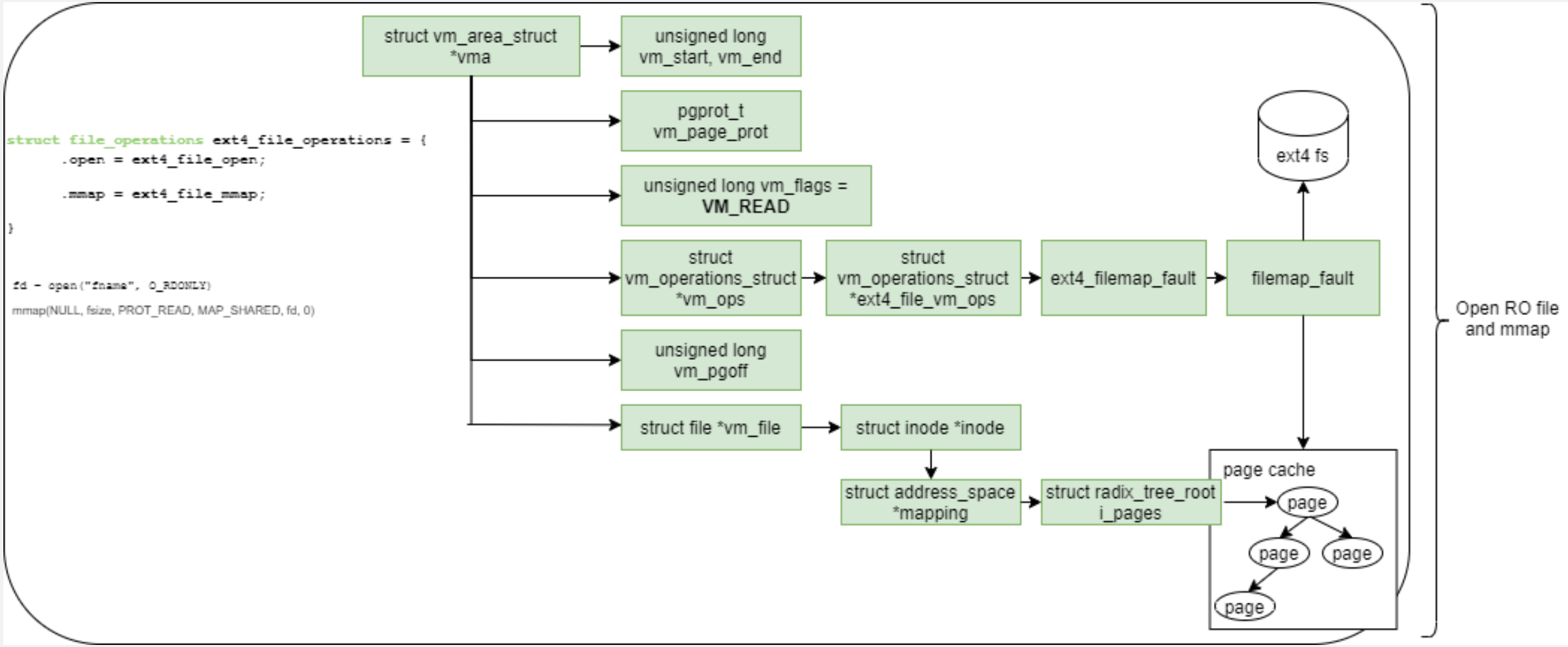
Importing VMA to GPU - userspace



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

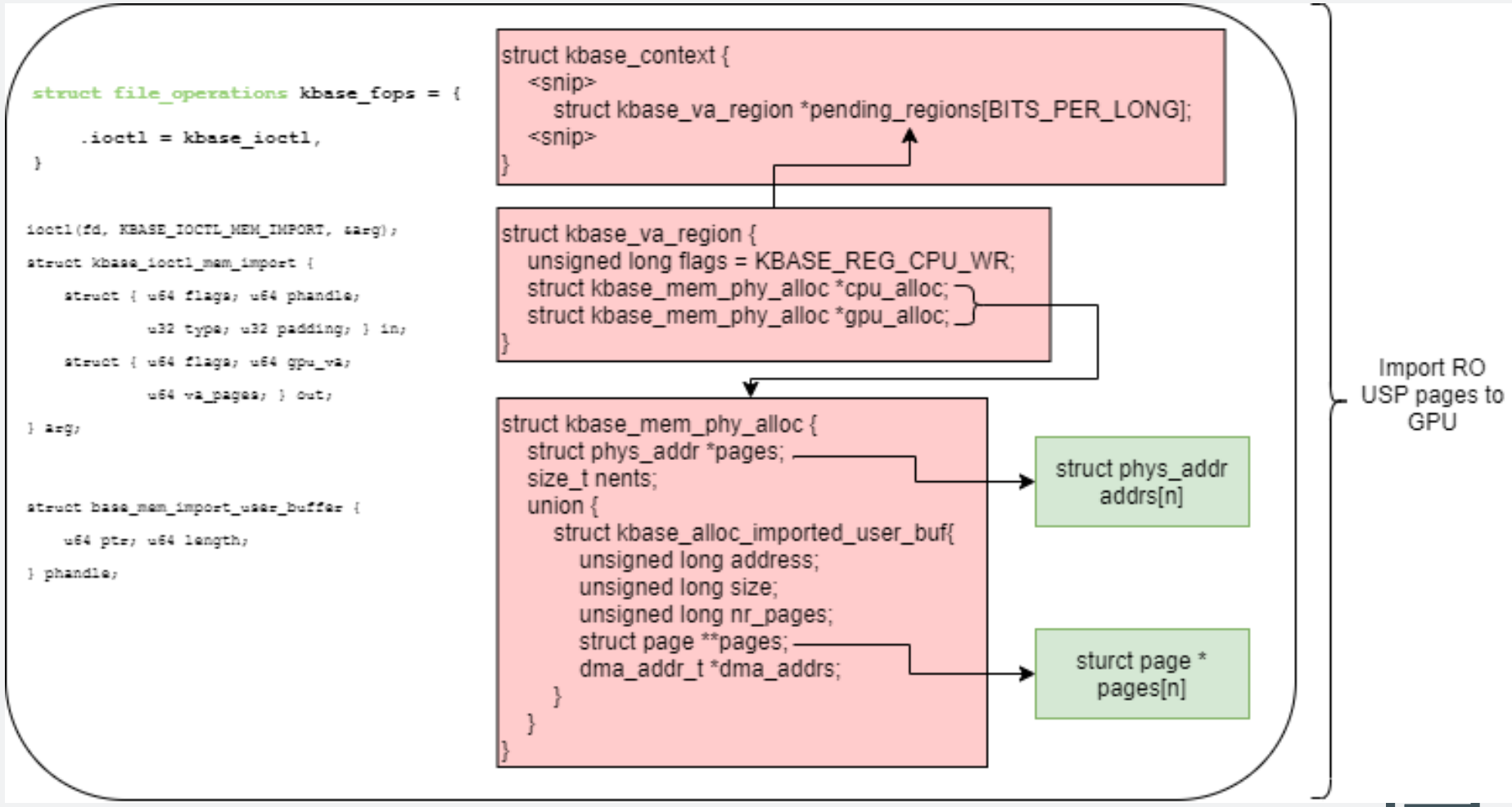
Importing VMA to GPU – kernel space



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

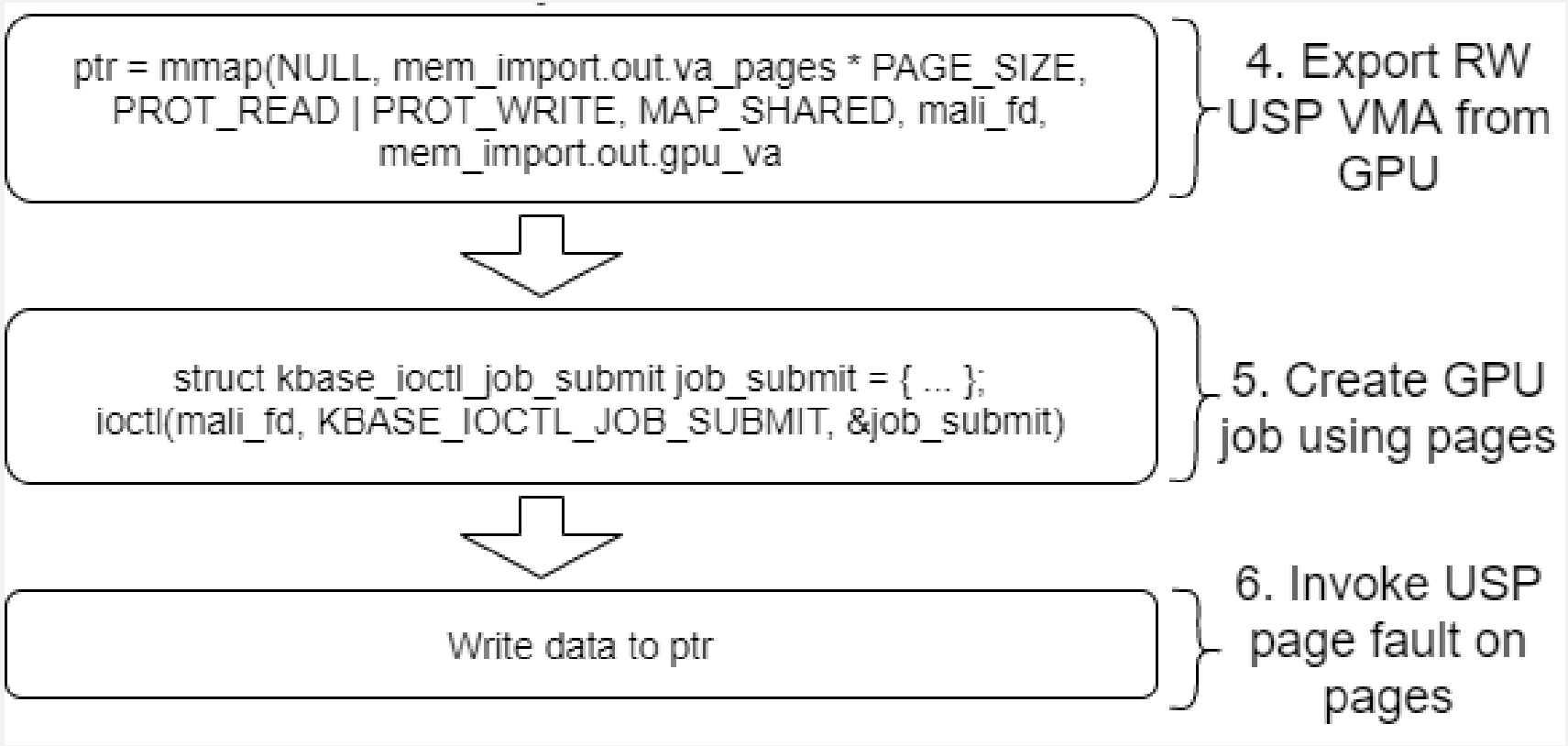
Importing VMA to GPU – kernel space



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

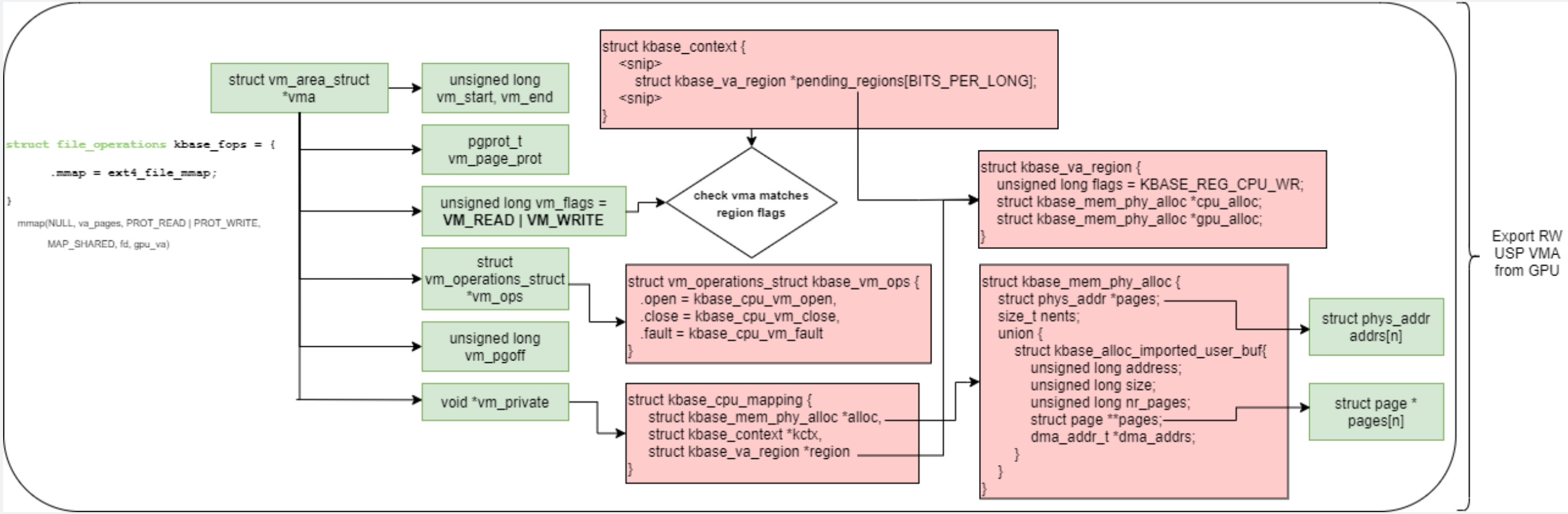
Exporting Region out of GPU - userspace



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Exporting Region out of GPU – kernel space



© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Submitting a Job to Pin Pages

```
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;
<snip>
    pinned_pages = get_user_pages(NULL, mm,
            address,
            alloc->imported.user_buf.nr_pages,
            reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,
            pages, NULL, NULL);
<snip>
    if (pinned_pages <= 0)
        return pinned_pages;
<snip>
    alloc->nents = pinned_pages;
}
```

```
static int kbase_jd_user_buf_map(struct kbase_context *kctx,
        struct kbase_va_region *reg)
{
<snip>
    int err = kbase_jd_user_buf_pin_pages(kctx, reg);
<snip>
    for (i = 0; i < pinned_pages; i++) {
        dma_addr_t dma_addr;
        unsigned long min;

        min = MIN(PAGE_SIZE - offset, local_size);
        dma_addr = dma_map_page(dev, pages[i],
                offset, min, DMA_BIDIRECTIONAL);
        if (dma_mapping_error(dev, dma_addr))
            goto unwind;

        alloc->imported.user_buf.dma_addrs[i] = dma_addr;
        pa[i] = as_tagged(page_to_phys(pages[i]));

        local_size -= min;
        offset = 0;
    }
<snip>
}
```

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

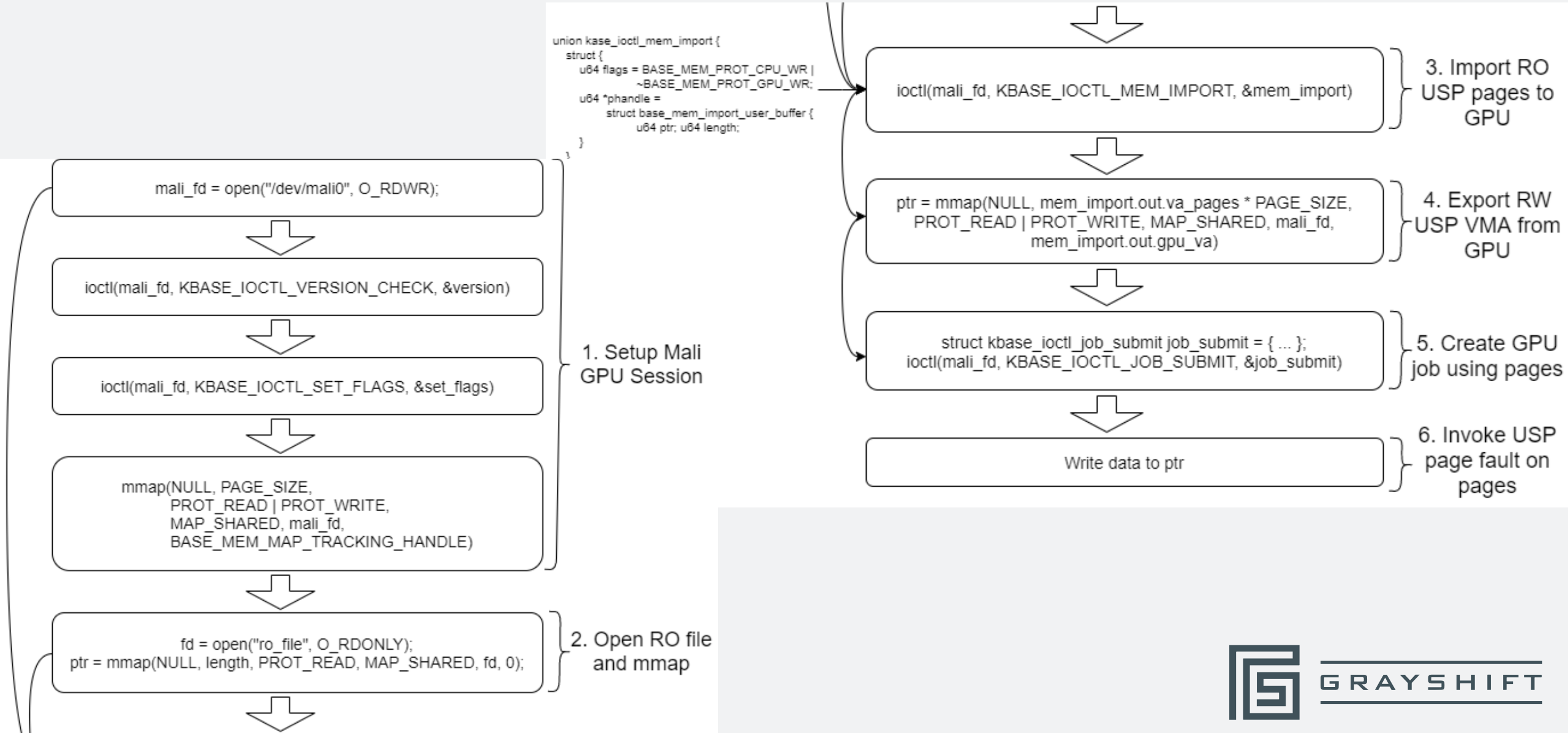
Fault the MMU

```
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;
<snip>
        nents = map->alloc->nents;
        pages = map->alloc->pages;
<snip>
    fault_pgoff = map_start_pgoff + (vmf->pgoff - vma->vm_pgoff);

    if (fault_pgoff >= nents)
        goto exit;
<snip>
    i = map_start_pgoff;
    addr = (pgoff_t)(vma->vm_start >> PAGE_SHIFT);
    mgm_dev = map->kctx->kbdev->mgm_dev;
    while (i < nents && (addr < vma->vm_end >> PAGE_SHIFT)) {

        ret = mgm_dev->ops.mgm_vmf_insert_pfn_prot(mgm_dev,
            map->alloc->group_id, vma, addr << PAGE_SHIFT,
            PFN_DOWN(as_phys_addr_t(pages[i])), vma->vm_page_prot);

        i++; addr++;
    }
}
```


© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Primitive Userspace Calls



GRAYSHIFT

Primitive

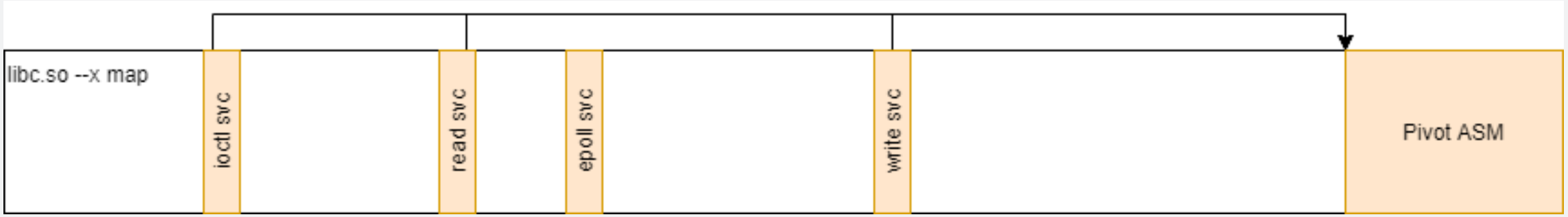
- Write to RO page caches
- What to write to?
  - mmap anything in /system/lib[64]
  - libc.so
- Process inject code into anything dynamically linked
  - Not init

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Pivot ASM

- Hand assemble hooks and pivot code for libc



- Pivot ASM Steps
  gettid(), create rwx memory allocation, open payload, read payload into memory, blr to payload
- Payload can be in memory loader to exec other elfs
- SELinux still enforcing, SECCOMP still filtering

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Further Attack Surfaces

- Pivot allows exec in vold, zygote, system_server
- Access to vold allows direct access to disk block devices
- Still need to contend with encryption

- Pivot allows further research to other DAC and MAC accessible surfaces
- Ideally a kernel pivot with goal of disabling SELinux

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Demo: Galaxy S10, SM-G973F, Security Patch Level 2021-01-01, PDA: G973FXXU9EUA4

```
beyond1:/ $ getprop |grep ro.build.version.security_patch
[ro.build.version.security_patch]: [2021-01-01]
beyond1:/ $ uname -a
Linux localhost 4.14.113-20606340 #1 SMP PREEMPT Wed Jan 20 11:28:48 KST 2021 aarch64
beyond1:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
beyond1:/ $ logcat|grep TESTBUG
10-27 12:05:28.880  9453  9453 I TESTBUG : [+] vold pid = 421
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] version: major = 0xb, minor = 0xe
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] setup_flags = 0
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] tracking page addr = 0x77fcbf3000
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] ddk version return str = K:r26p0-01eac0(GPL)
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] mem_port.in.cpu_addr = 0x7509ebc000
10-27 12:05:28.956  9453  9453 I TESTBUG : [+] mem_port.in.length = 0x11e4000
10-27 12:05:28.960  9453  9453 I TESTBUG : [+] mem_port.out.flags = 0x4007
10-27 12:05:28.960  9453  9453 I TESTBUG : [+] mem_port.out.gpu_va = 0x41000
10-27 12:05:28.961  9453  9453 I TESTBUG : [+] mem_port.out.va_pages = 0x11e4
10-27 12:05:28.961  9453  9453 I TESTBUG : [+] mmap ptr = 0x7508c00000
10-27 12:05:28.965  9453  9453 I TESTBUG : [+] found libc.so addrs, saddr = 0x77fb28d000, eaddr = 0x77fb309000
10-27 12:05:28.967  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2edeb0
10-27 12:05:28.968  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2ee9b0
10-27 12:05:28.970  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2eef70
10-27 12:05:28.971  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2edc30
10-27 12:05:28.973  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2edc50
10-27 12:05:28.975  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2ee650
10-27 12:05:28.976  9453  9453 I TESTBUG : [+] found sig_ptr = 0x77fb2eee70
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] version: major = 0xb, minor = 0xe
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] setup_flags = 0
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] tracking page addr = 0x77fba64000
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] ddk version return str = K:r26p0-01eac0(GPL)
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mem_port.in.cpu_addr = 0x77fb28d000
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mem_port.in.length = 0x7c000
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mem_port.out.flags = 0x4007
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mem_port.out.gpu_va = 0x41000
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mem_port.out.va_pages = 0x7c
10-27 12:05:28.982  9453  9453 I TESTBUG : [+] mmap ptr = 0x7509e40000
10-27 12:05:32.908 10078 10078 I TESTBUG : [./stage1.c:19] STAGE1 pid = 10078, uid = 0, gid = 0, context = u:r:vold:s0
^C
130|beyond1:/ $
```

© 2021. Grayshift, LLC. Al

GRAYSHIFT

Why does the Vulnerability Exist?

- Speculate:
  - The mali code is complicated
    - Large functions, copied code
    - Large data structs
    - Until recently, all the source files (hundreds) in the same directory
  - Mali setup is not trivial
    - Need to perform ioctls and mmap before interaction
    - Lack of public system emulator
  - GPU devs are not focused on security

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

How Bug was Discovered?

- Speculation:
  - Could have been discovered by matching the pattern from CVE-2016-2067 from KGSL/Adreno driver to Mali
  - Could have been discovered by pure auditing noticing the flags deficiency
  - Probably not fuzzing

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Why I Work at Grayshift

- Work from home
- Regular success stories from the field, close to the mission
- Work with highly skilled and experienced vulnerability researchers
- Rewards individual performance
- Access to tools, training, and education

© 2021. Grayshift, LLC. All rights reserved.

GRAYSHIFT

Questions? / Comments

GRAYSHIFT