PhantomMap: GPU-Assisted Kernel Exploitation

Jiayi Hu, Qi Tang, Xingkai Wang, Jinmeng Zhou, Rui Chang, Wenbo Shen ✉

浙江大學
ZHEJIANG UNIVERSITY

吉林大学
JILIN UNIVERSITY

Outline

1. **Motivation**
2. **New Findings**
3. **PhantomMap Technique**
    a) **Exploit Chains Identification**
    b) **Exploitability Evaluation**
4. **Lightweight Mitigation**
5. **Takeaways**

Motivation: CPU vs GPU

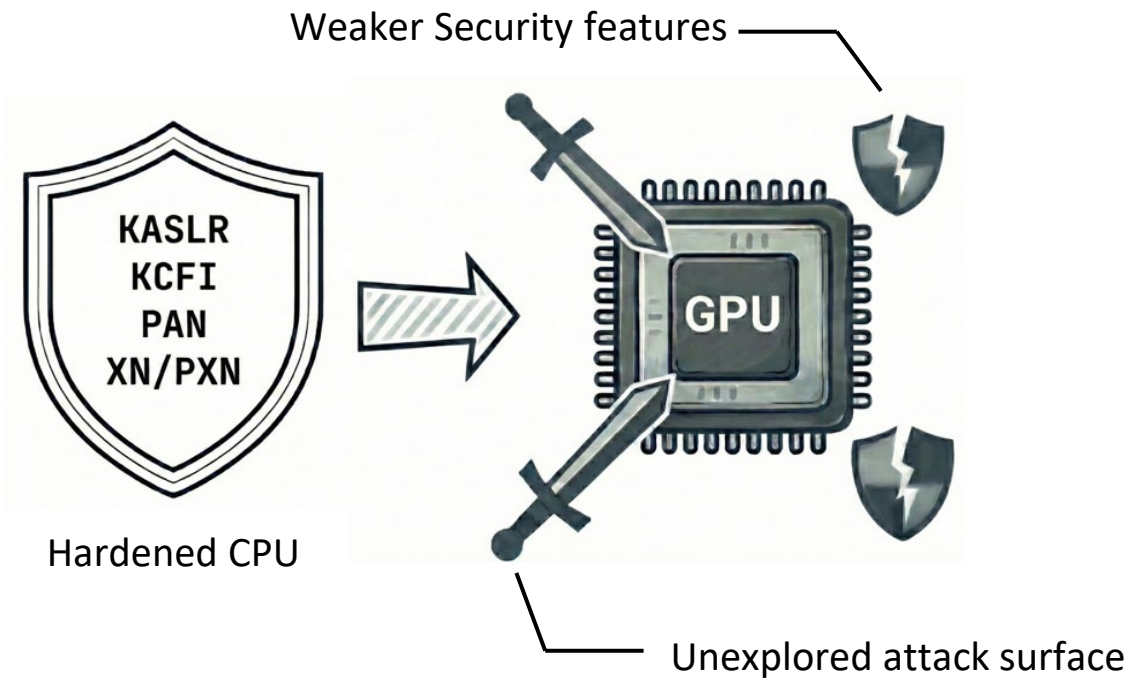
Robust CPU have raised the bar for direct kernel exploitation, forcing attackers to seek other targets.

Hardened CPU

Motivation: CPU vs GPU

Robust CPU have raised the bar for direct kernel exploitation, forcing attackers to seek other targets.

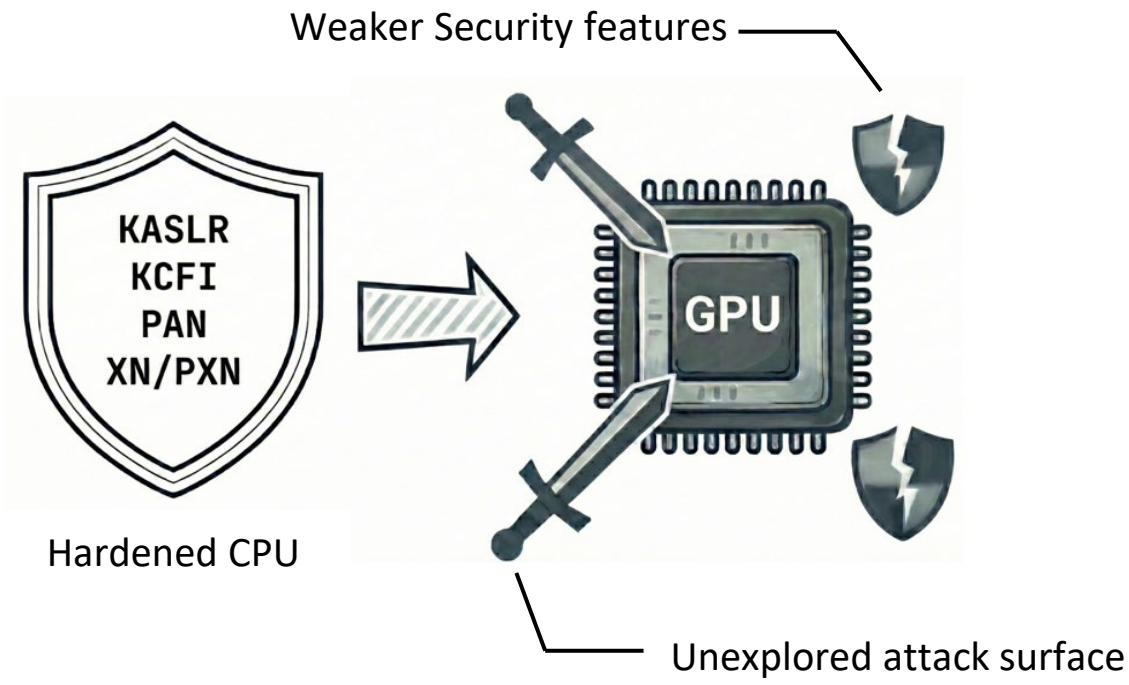
GPU have become a preferred target for attackers.



Motivation: CPU vs GPU

Robust CPU have raised the bar for direct kernel exploitation, forcing attackers to seek other targets.

GPU have become a preferred target for attackers.



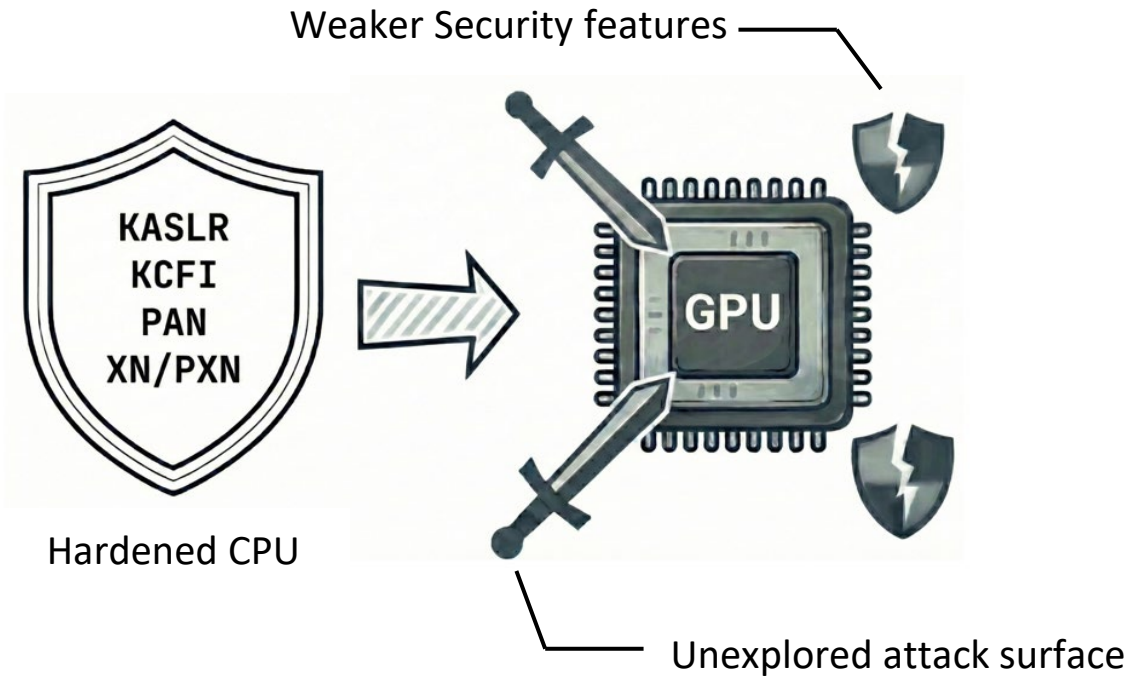
- **Weaker Security Features**

  e.g., unlike CPU applications, GPU applications often execute outside the sandbox, exposing the interfaces to kernel exploitation.

Motivation: CPU vs GPU

Robust CPU have raised the bar for direct kernel exploitation, forcing attackers to seek other targets.

GPU have become a preferred target for attackers.

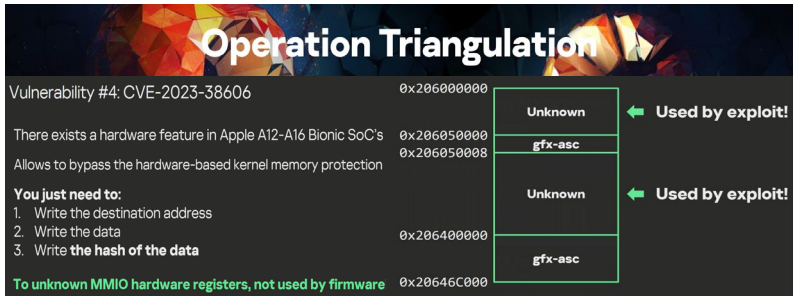


- **Weaker Security Features**

  e.g., unlike CPU applications, GPU applications often execute outside the sandbox, exposing the interfaces to kernel exploitation.

- **Unexplored/Potential Attack Surface**

  e.g., Operation Triangulation leveraged undocumented MMIO registers within the GPU coprocessor to bypass kernel protections and escalate privileges in iOS.



Motivation: Mali GPU popularity

Dominant SoC Adoption: integrated into MediaTek and Unisoc platforms.

Massive Device Reach: Deployed in Google Pixels, Samsung, Xiaomi, OPPO, and Vivo, etc.

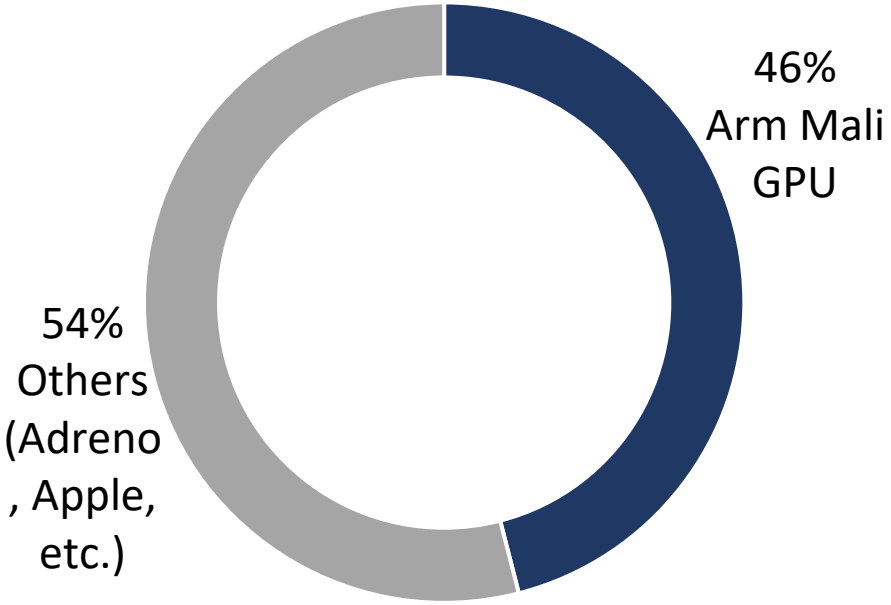
Brands	Q3 2024	Q4 2024	Q1 2025	Q2 2025	Q3 2025
MediaTek	37%	31%	38%	36%	34%
Qualcomm	24%	25%	27%	26%	24%
Apple	17%	21%	15%	15%	18%
UNISOC	13%	14%	10%	13%	14%
Samsung	5%	4%	5%	6%	6%
HiSilicon (Huawei)	4%	4%	4%	4%	3%
Others	1%	1%	1%	1%	1%

Motivation: Mali GPU popularity

Dominant SoC Adoption: integrated into MediaTek and Unisoc platforms.

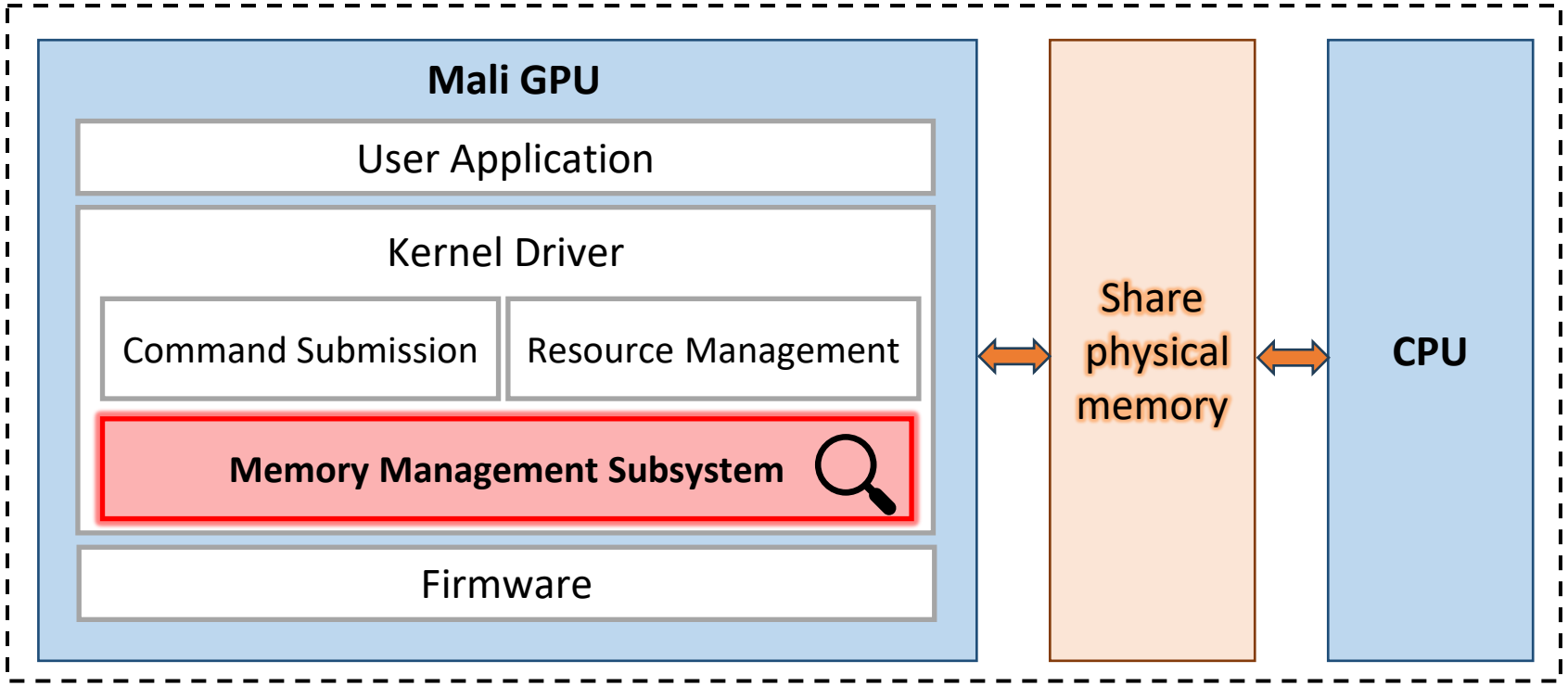
Massive Device Reach: Deployed in Google Pixels, Samsung, Xiaomi, OPPO, and Vivo, etc.

Global Smartphone GPU Market Share → **Arm Mali: 46% Global Market Share**



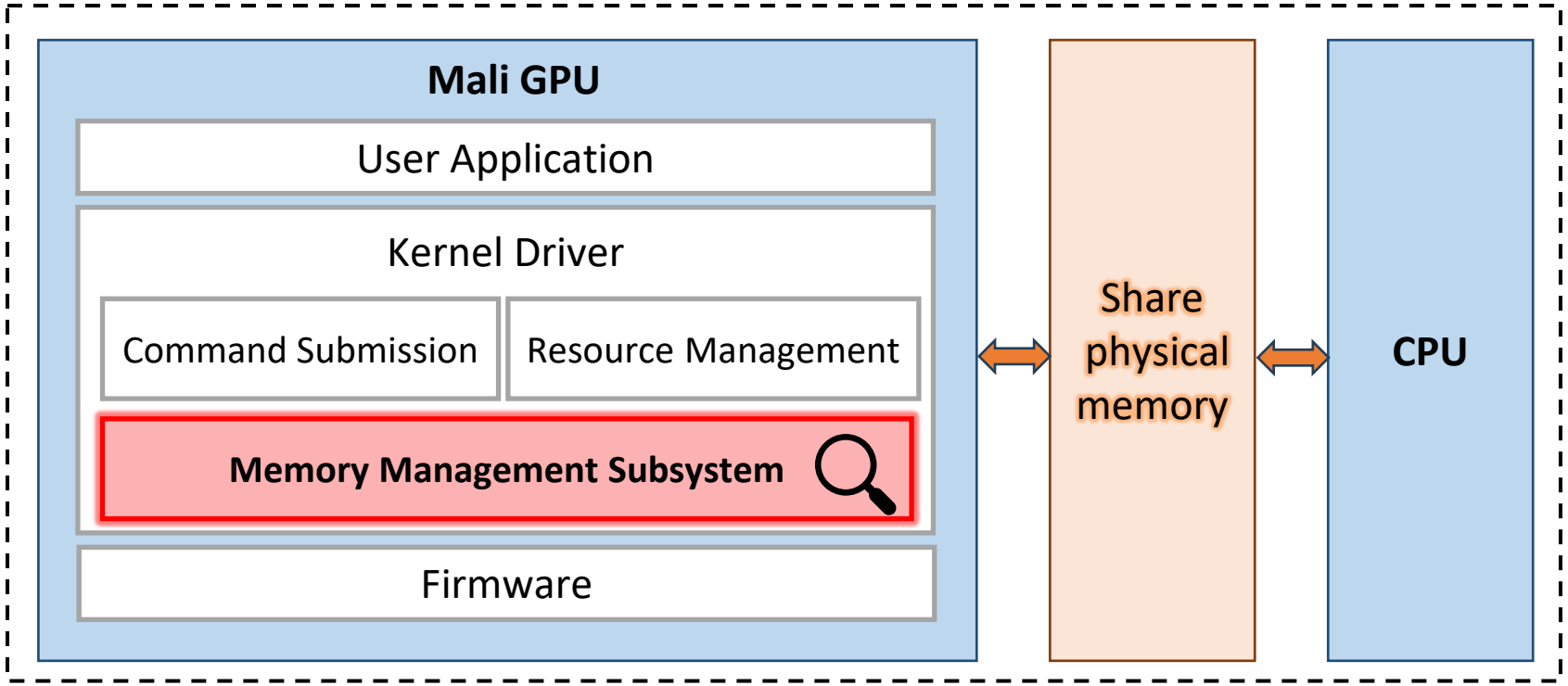
Brands	Q3 2024	Q4 2024	Q1 2025	Q2 2025	Q3 2025
MediaTek	37%	31%	38%	36%	34%
Qualcomm	24%	25%	27%	26%	24%
Apple	17%	21%	15%	15%	18%
UNISOC	13%	14%	10%	13%	14%
Samsung	5%	4%	5%	6%	6%
HiSilicon (Huawei)	4%	4%	4%	4%	3%
Others	1%	1%	1%	1%	1%

Motivation: unexplored attack surface



- **Insight1:** Mali GPU & CPU share the whole physical memory
- **Insight2:** The memory management subsystem of Mali GPU are critical. But security analysis has been ad-hoc and manual

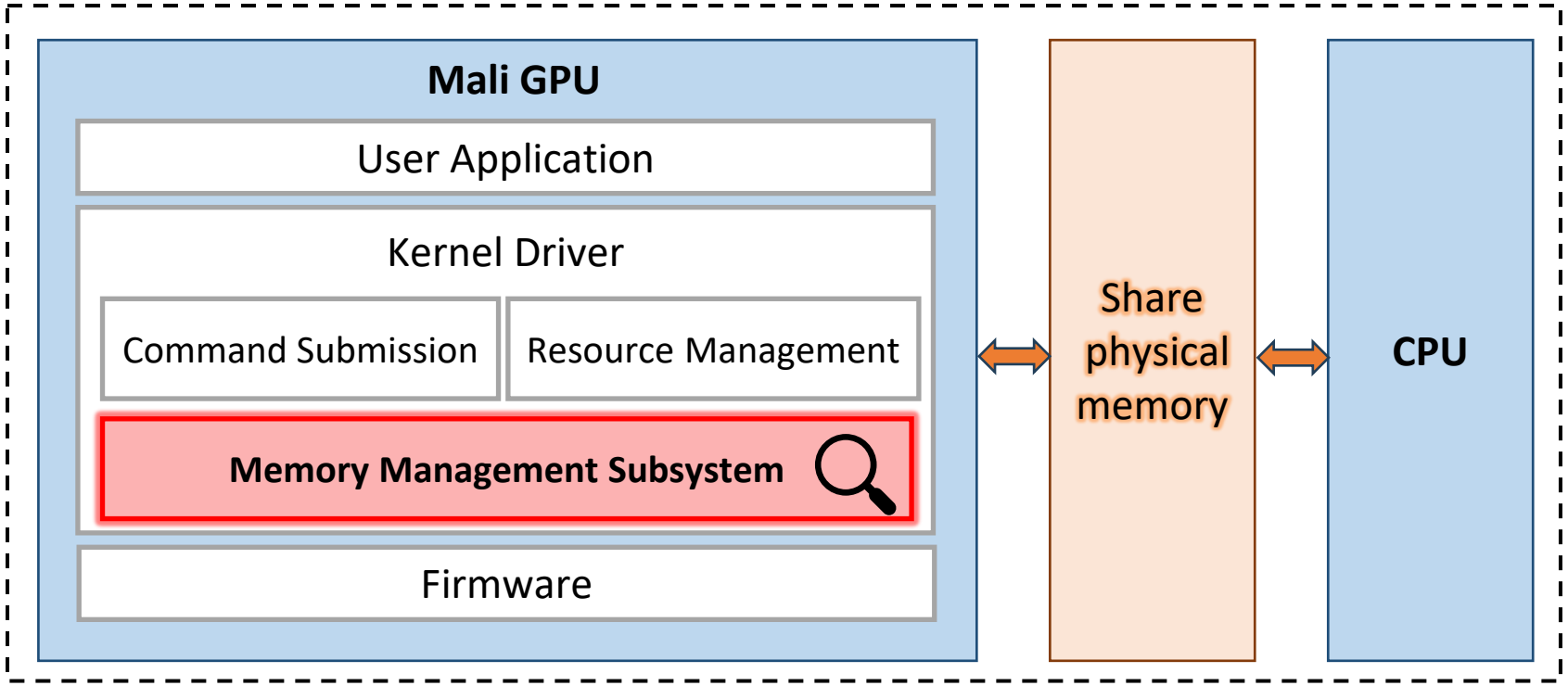
Motivation: unexplored attack surface



- **Insight1:** Mali GPU & CPU share the whole physical memory
- **Insight2:** The memory management subsystem of Mali GPU are critical. But security analysis has been ad-hoc and manual

⚠ Especially, there is a **gap** in understanding the memory mapping mechanism of Mali GPU.

Motivation: unexplored attack surface

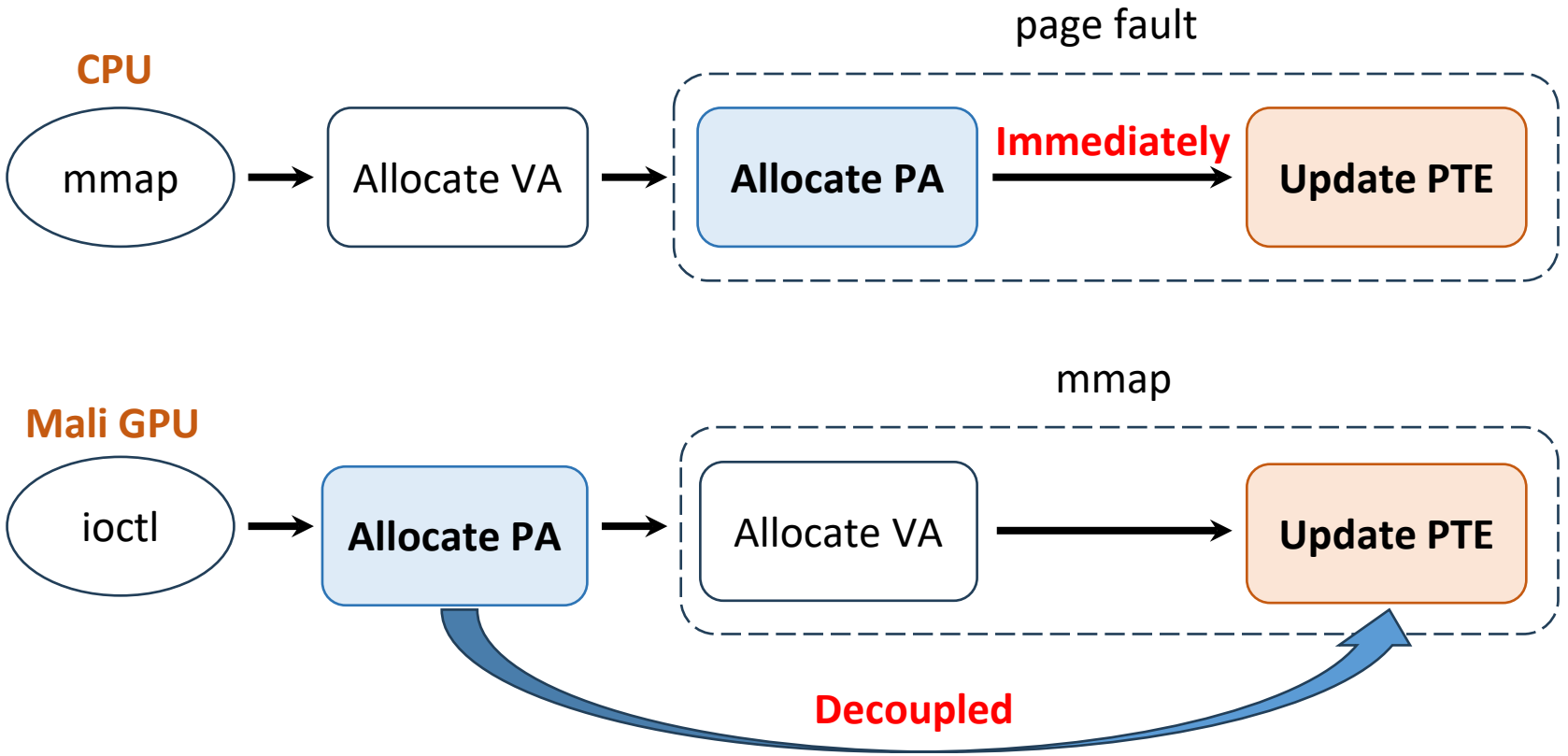


- **Insight1:** Mali GPU & CPU share the whole physical memory
- **Insight2:** The memory management subsystem of Mali GPU are critical. But security analysis has been ad-hoc and manual

⚠️ Especially, there is a **gap** in understanding the memory mapping mechanism of Mali GPU.

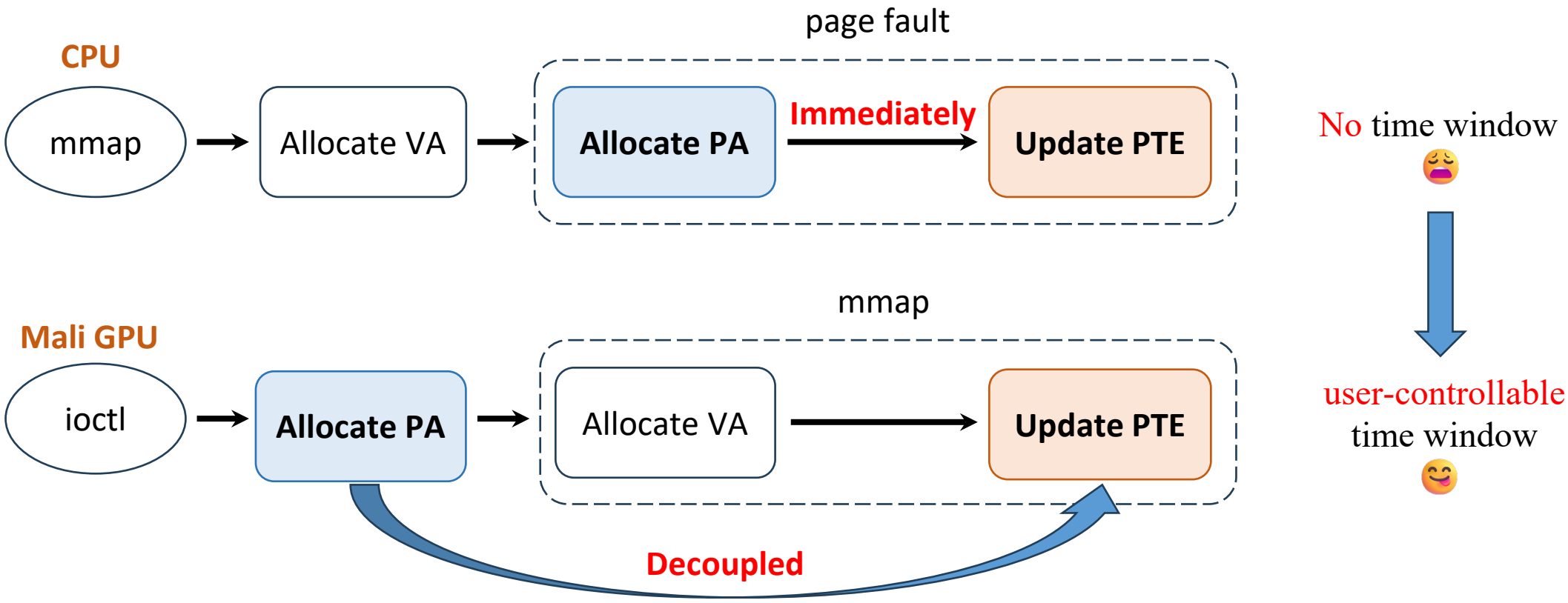
✅ We conduct the first systematic analysis of memory mapping mechanism of Mali GPU.

New Findings: decoupled memory operations



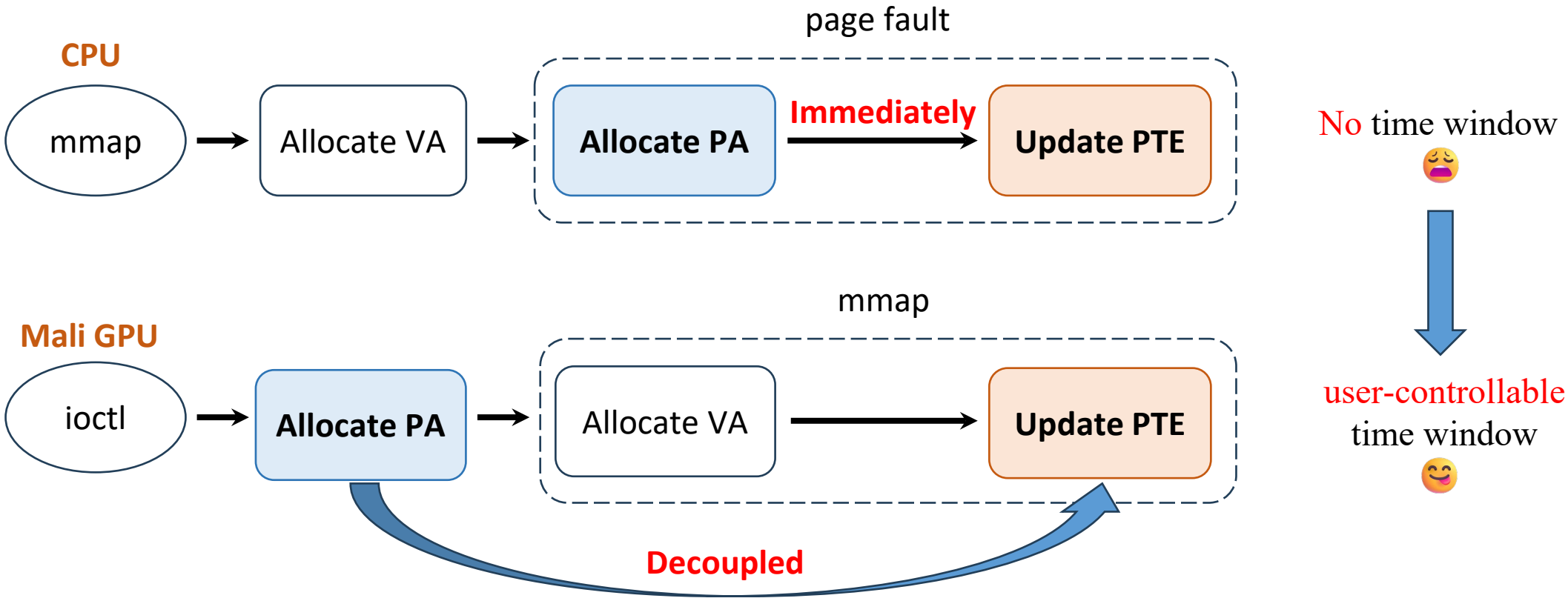
No time window 😩

New Findings: decoupled memory operations



New Findings: decoupled memory operations

Finding-1: The Mali GPU driver decouples physical memory allocation from page table updation, creating a user-controllable time window that can be exploited to remap physical memory.



New Findings: missing GPU physical address validation

In CPU-mapping process, essential checks ensure page validity before inserting physical addresses into and updating the CPU page-table.

```
int insert_page(struct vm_area_struct *vma,...struct
↪  page *page,...) {
    ...
    retval = validate_page_before_insert(vma, page);
    if (retval)
        goto out;
    ...
    retval = insert_page_into_pte_locked(page, ...);
    ...
}
```

New Findings: missing GPU physical address validation

In CPU-mapping process, essential checks ensure page validity before inserting physical addresses into and updating the CPU page-table.

```
int insert_page(struct vm_area_struct *vma,...struct
↪  page *page,...) {
    ...
    retval = validate_page_before_insert(vma, page);
    if (retval)
        goto out;
    ...
    retval = insert_page_into_pte_locked(page, ...);
    ...
}
```

```
int validate_page_before_insert(...,struct page*page){
    struct folio *folio = page_folio(page);
    if (!folio_ref_count(folio))
        return -EINVAL;
    ...
    if (folio_test_anon(folio) ||
    ↪  folio_test_slab(folio) || page_has_type(page))
        return -EINVAL;
    ...
}
```

New Findings: missing GPU physical address validation

In CPU-mapping process, essential checks ensure page validity before inserting physical addresses into and updating the CPU page-table.

```
int insert_page(struct vm_area_struct *vma,...struct
↪  page *page,...) {
    ...
    retval = validate_page_before_insert(vma, page);
    if (retval)
        goto out;
    ...
    retval = insert_page_into_pte_locked(page, ...);
    ...
}
```

```
int validate_page_before_insert(...,struct page*page){
    struct folio *folio = page_folio(page);
    if (!folio_ref_count(folio))
        return -EINVAL;
    ...
    if (folio_test_anon(folio) ||
    ↪  folio_test_slab(folio) || page_has_type(page))
        return -EINVAL;
    ...
}
```

- Check if the reference count for the page's folio is zero. ≫ Prevent mapping UAF page.
- Ensure the page is not of an anonymous or slab type. ≫ Avoid mapping critical regions to user-space.

New Findings: missing GPU physical address validation

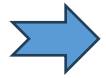
Arm Mali GPU driver implements custom memory mapping and pagefault handling interfaces for GPU-specific Virtual MemoryAreas (VMAs).

```
int kbase_mmu_insert_pages(struct tagged_addr *phys,
↪   ...) {
    ...
    err = mmu_insert_pages_no_flush(..., phys, ...);
    ...
}
```

New Findings: missing GPU physical address validation

Arm Mali GPU driver implements custom memory mapping and pagefault handling interfaces for GPU-specific Virtual MemoryAreas (VMAs).

```
int kbase_mmu_insert_pages(struct tagged_addr *phys,
↪   ...) {
    ...
    err = mmu_insert_pages_no_flush(..., phys, ...);
    ...
}
```

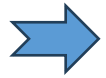


Without any validation of provided physical address

New Findings: missing GPU physical address validation

Arm Mali GPU driver implements custom memory mapping and pagefault handling interfaces for GPU-specific Virtual MemoryAreas (VMAs).

```
int kbase_mmu_insert_pages(struct tagged_addr *phys,
↪   ...) {
    ...
    err = mmu_insert_pages_no_flush(..., phys, ...);
    ...
}
```



Without any validation of provided physical address

😨 Driver blindly inserts whatever physical address into GPU page-table !

New Findings: missing GPU physical address validation

Finding-2: The Mali GPU driver doesn't validate physical pages before mapping, allowing arbitrary or sensitive kernel memory to be remapped into user space without any security checks.

Arm Mali GPU driver implements custom memory mapping and pagefault handling interfaces for GPU-specific Virtual MemoryAreas (VMAs).

```
int kbase_mmu_insert_pages(struct tagged_addr *phys,
↪   ...) {
    ...
    err = mmu_insert_pages_no_flush(..., phys, ...);
    ...
}
```

Without any validation of provided physical address

😨 Driver blindly inserts whatever physical address into GPU page-table !

PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap** 

The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.

PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.

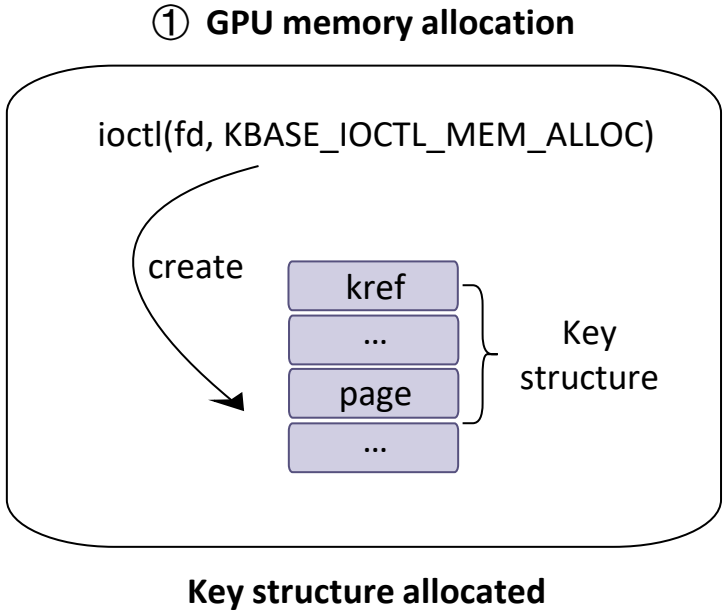
① **GPU memory allocation**

ioctl(fd, KBASE_IOCTL_MEM_ALLOC)

PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

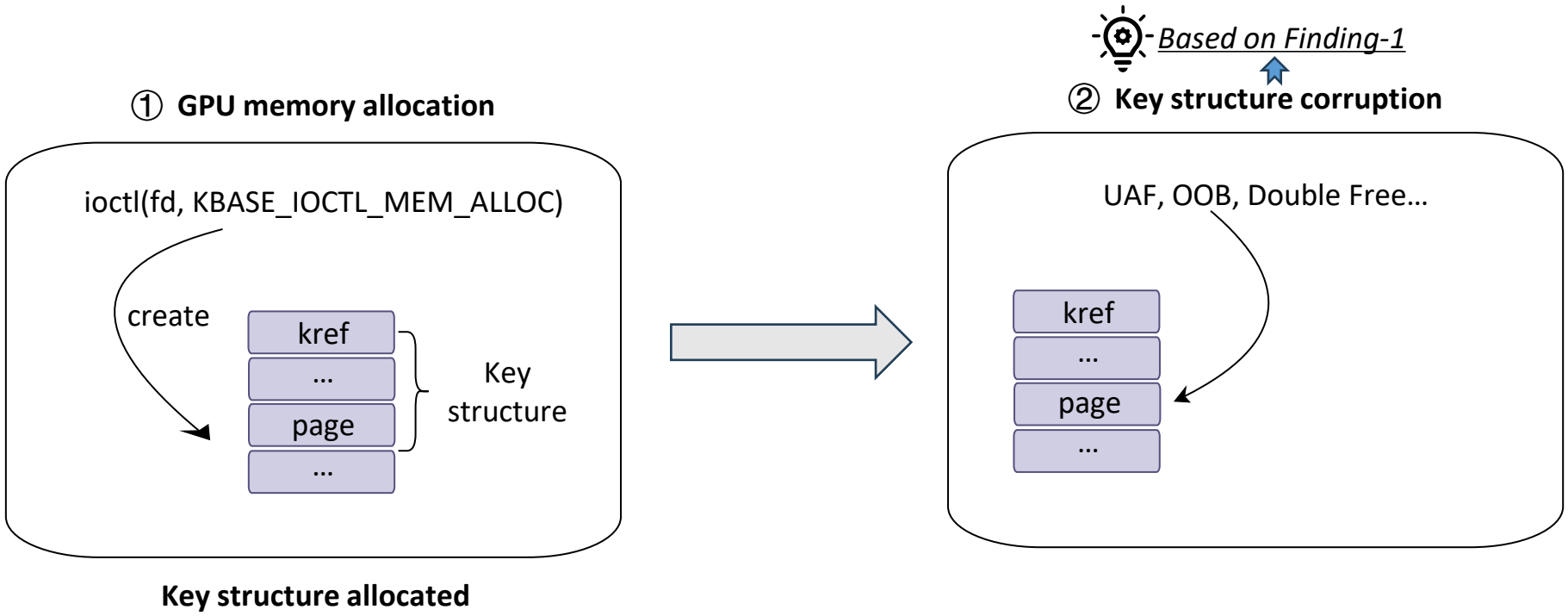
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

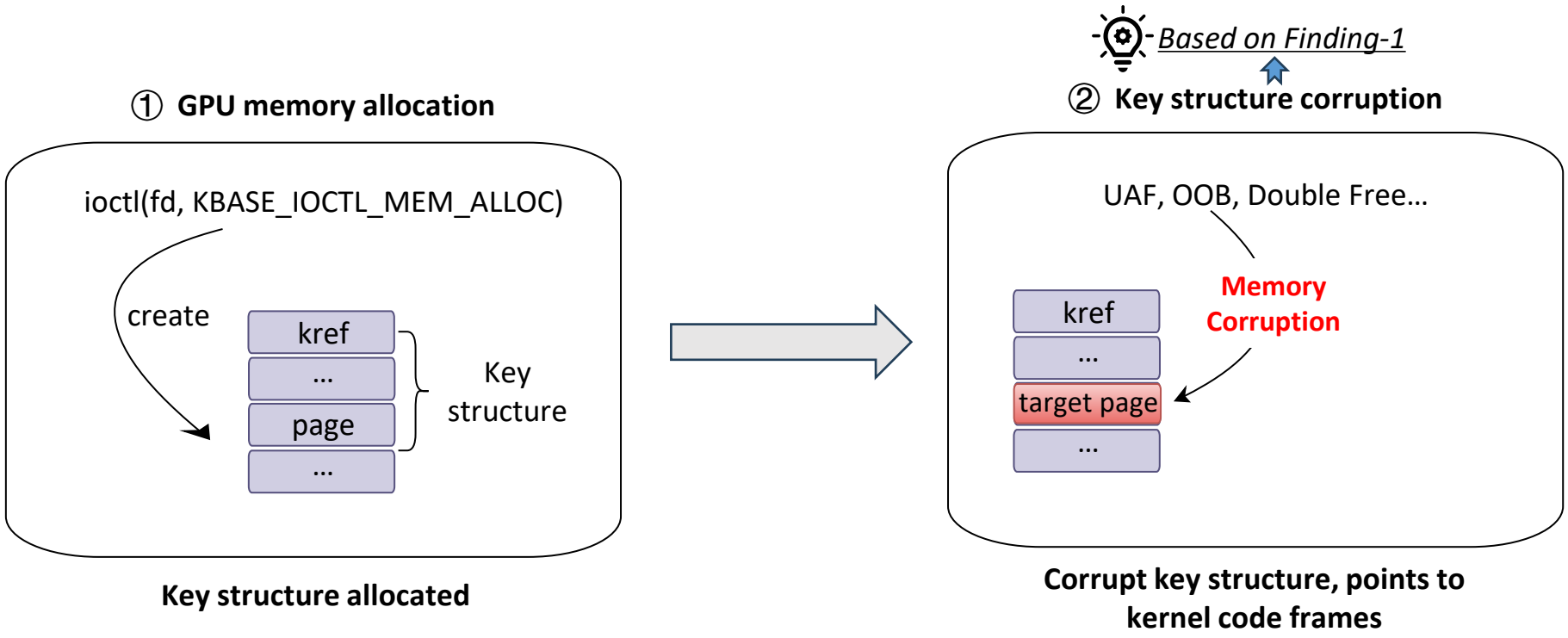
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

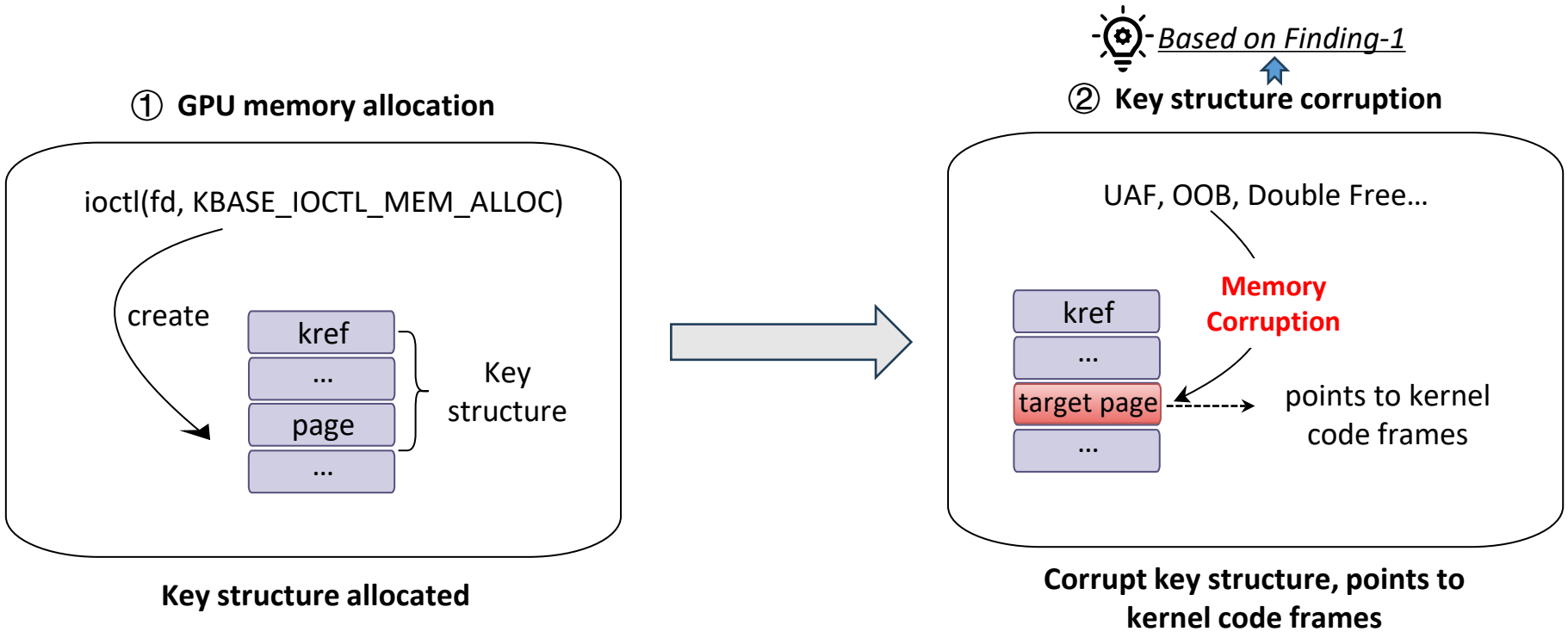
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

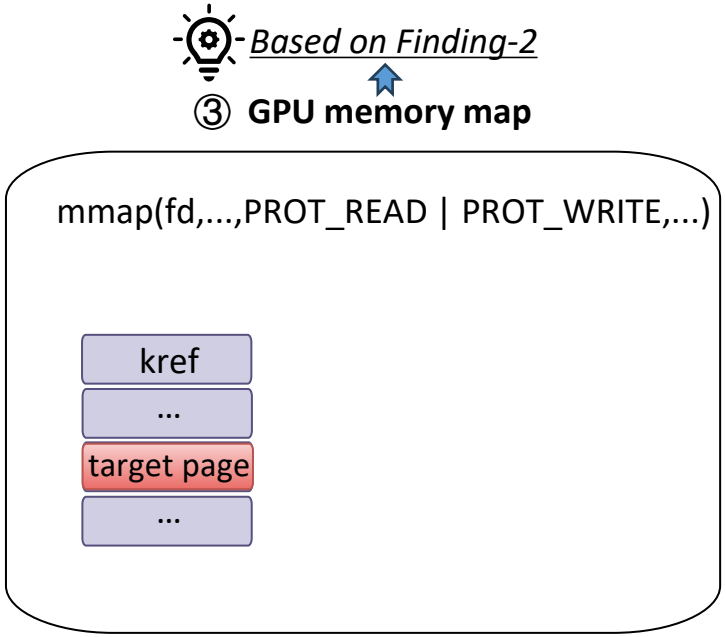
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

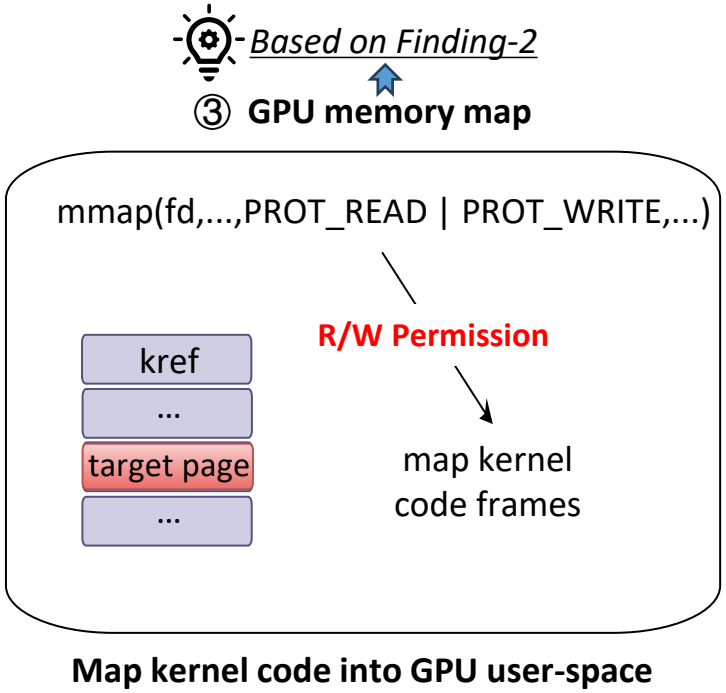
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

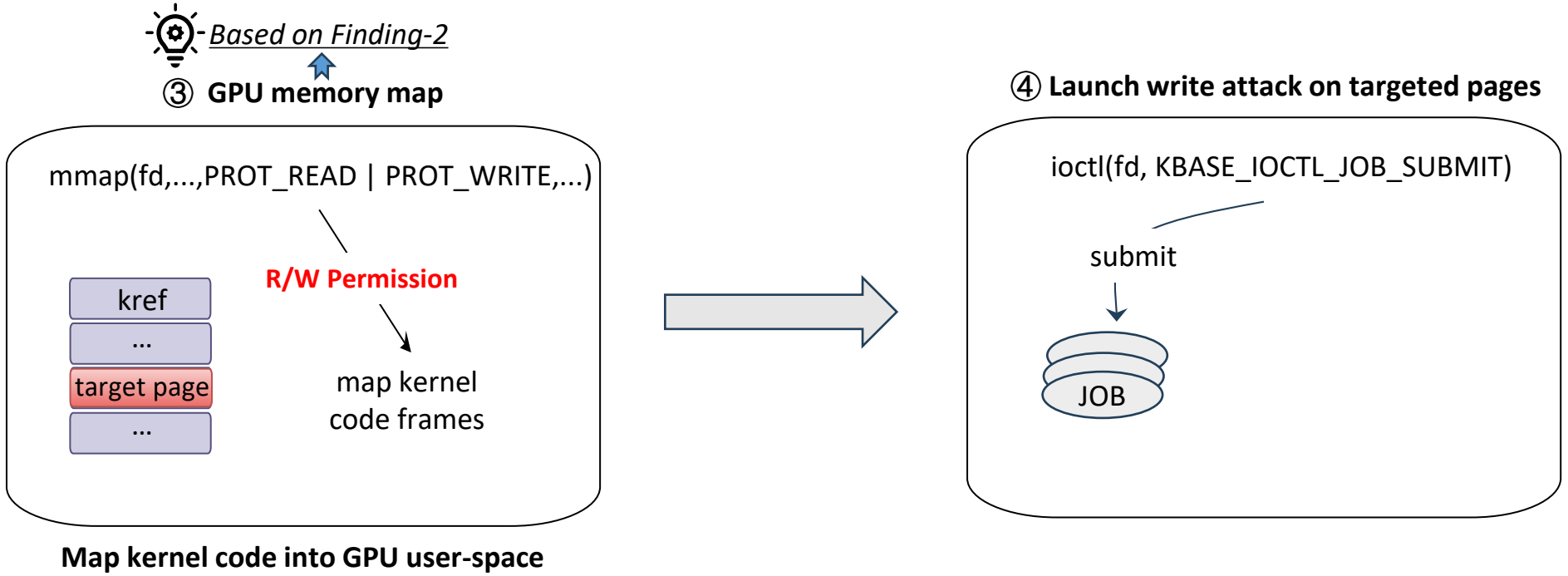
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

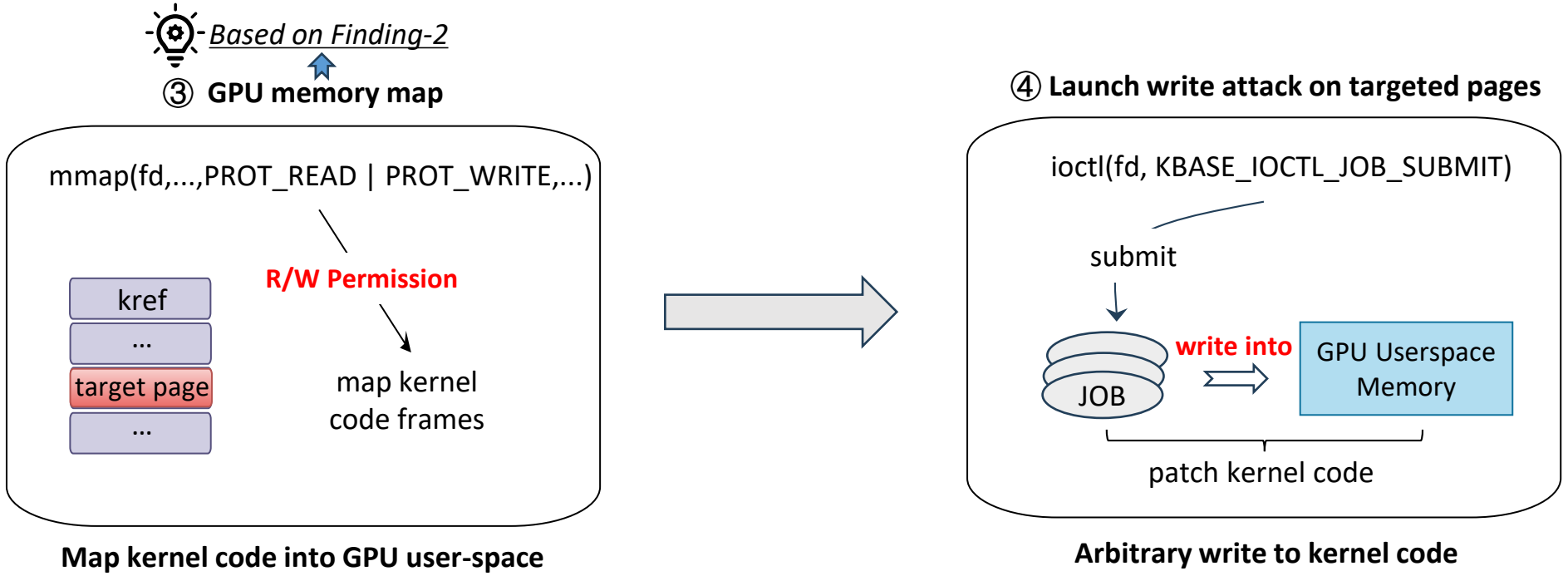
The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



PhantomMap Technique

Based on the two findings, we proposed a novel GPU-assisted exploitation technique: **PhantomMap**

The key idea of PhantomMap: exploit Mali's decoupled allocation-and-mapping workflow by corrupting GPU key structures, then the driver remaps arbitrary physical memory into GPU user-space without validation.



Map kernel code into GPU user-space

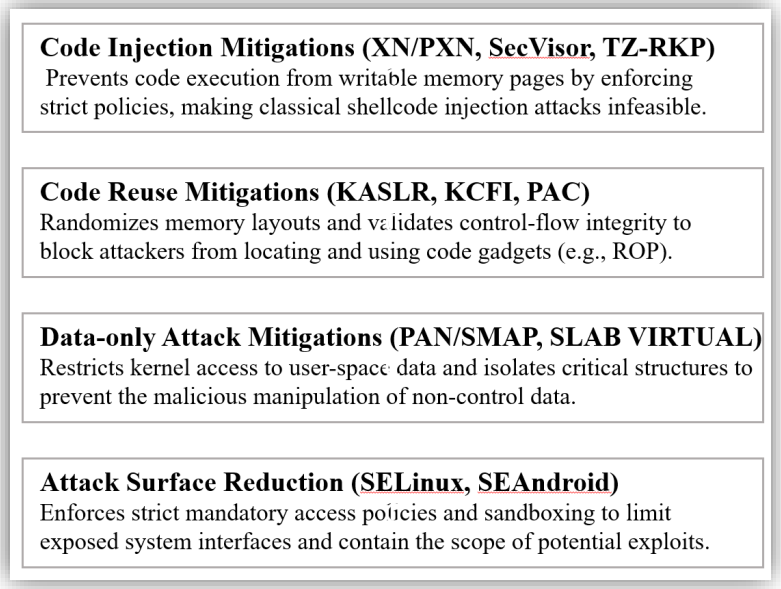
Arbitrary write to kernel code

PhantomMap Advantages

- ***Practical !***
  - Operate entirely without extra linux kernel capabilities(e.g., CAP_BPF, CAP_NET_RAW).

PhantomMap Advantages

- ***Practical !***
  - Operate entirely without extra linux kernel capabilities(e.g., CAP_BPF, CAP_NET_RAW).
- ***Powerful !***
  - Provide code injection primitives directly without complex pivot steps.
  - Bypass all modern mitigations in Android. ⚔️



PhantomMap Advantages

- ***Practical !***
  - Operate entirely without extra linux kernel capabilities(e.g., CAP_BPF, CAP_NET_RAW).
- ***Powerful !***
  - Provide code injection primitives directly without complex pivot steps.
  - Bypass all modern mitigations in Android. ⚔️
- ***General !***
  - Applicable to almost all heap vulnerability capabilities.

PhantomMap Advantages

- ***Practical !***
  - Operate entirely without extra linux kernel capabilities(e.g., CAP_BPF, CAP_NET_RAW).
- ***Powerful !***
  - Provide code injection primitives directly without complex pivot steps.
  - Bypass all modern mitigations in Android. ⚔️
- ***General !***
  - Applicable to almost all heap vulnerability capabilities.
- ***Reliable & Simpler !***
  - Don’t need any info-leak during exploitation.

Exploit Chains Identification

The dedicated analyzer identified **15 distinct end-to-end exploit chains** and **6 unique key structures**.

Allocation Entry	Syscall	Mapping Entry	Syscall	Related Key Structure	Read/Write
kbase_mem_alloc	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_mem_commit	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_cpu_mmap	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_alloc_import_user_buf	●
kbase_mem_import	ioctl	kbase_mem_flags_change	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_cpu_mmap	mmap	kbase_mem_phy_alloc	◐
kbase_create_context	open	kbase_gpu_mmap	mmap	kbase_context	◐
kbase_create_context	ioctl	kbase_map_external_resource	mmap	kbase_mem_phy_alloc	●
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_reg_page	mmap	kbase_device	◐
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_device	●
kbase_csf_doorbell_mapping_init†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_csf_device	●
kbase_csf_alloc_command_stream_user_pages†	ioctl	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_queue	●

Exploit Chains Identification

The dedicated analyzer identified **15 distinct end-to-end exploit chains** and **6 unique key structures**.

Allocation Entry	Syscall	Mapping Entry	Syscall	Related Key Structure	Read/Write
kbase_mem_alloc	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_mem_commit	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_cpu_mmap	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_alloc_import_user_buf	●
kbase_mem_import	ioctl	kbase_mem_flags_change	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_cpu_mmap	mmap	kbase_mem_phy_alloc	◐
kbase_create_context	open	kbase_gpu_mmap	mmap	kbase_context	◐
kbase_create_context	ioctl	kbase_map_external_resource	mmap	kbase_mem_phy_alloc	●
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_reg_page	mmap	kbase_device	◐
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_device	●
kbase_csf_doorbell_mapping_init†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_csf_device	●
kbase_csf_alloc_command_stream_user_pages†	ioctl	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_queue	●

R/W permission?

Exploit Chains Identification

The dedicated analyzer identified **15 distinct end-to-end exploit chains** and **6 unique key structures**.

Allocation Entry	Syscall	Mapping Entry	Syscall	Related Key Structure	Read/Write
kbase_mem_alloc	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_mem_commit	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_cpu_mmap	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_alloc_import_user_buf	●
kbase_mem_import	ioctl	kbase_mem_flags_change	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_cpu_mmap	mmap	kbase_mem_phy_alloc	◐
kbase_create_context	open	kbase_gpu_mmap	mmap	kbase_context	◐
kbase_create_context	ioctl	kbase_map_external_resource	mmap	kbase_mem_phy_alloc	●
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_reg_page	mmap	kbase_device	◐
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_device	●
kbase_csf_doorbell_mapping_init†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_csf_device	●
kbase_csf_alloc_command_stream_user_pages†	ioctl	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_queue	●

R/W permission?

Attack window?

Exploit Chains Identification

The dedicated analyzer identified **15 distinct end-to-end exploit chains** and **6 unique key structures**.

Allocation Entry	Syscall	Mapping Entry	Syscall	Related Key Structure	Read/Write
kbase_mem_alloc	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_mem_commit	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_cpu_mmap	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_alloc_import_user_buf	●
kbase_mem_import	ioctl	kbase_mem_flags_change	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_cpu_mmap	mmap	kbase_mem_phy_alloc	◐
kbase_create_context	open	kbase_gpu_mmap	mmap	kbase_context	◐
kbase_create_context	ioctl	kbase_map_external_resource	mmap	kbase_mem_phy_alloc	●
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_reg_page	mmap	kbase_device	◐
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_device	●
kbase_csf_doorbell_mapping_init†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_csf_device	●
kbase_csf_alloc_command_stream_user_pages†	ioctl	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_queue	●

R/W permission?

Attack window?

Exploit Chains Identification

The dedicated analyzer identified **15 distinct end-to-end exploit chains** and **6 unique key structures**.

Allocation Entry	Syscall	Mapping Entry	Syscall	Related Key Structure	Read/Write
kbase_mem_alloc	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_mem_commit	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alloc	ioctl	kbase_cpu_mmap	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_alloc_import_user_buf	●
kbase_mem_import	ioctl	kbase_mem_flags_change	ioctl	kbase_mem_phy_alloc	●
kbase_mem_import	ioctl	kbase_map_external_resource	ioctl	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_gpu_mmap	mmap	kbase_mem_phy_alloc	●
kbase_mem_alias	ioctl	kbase_cpu_mmap	mmap	kbase_mem_phy_alloc	◐
kbase_create_context	open	kbase_gpu_mmap	mmap	kbase_context	◐
kbase_create_context	ioctl	kbase_map_external_resource	mmap	kbase_mem_phy_alloc	●
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_reg_page	mmap	kbase_device	◐
kbase_platform_device_probe†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_device	●
kbase_csf_doorbell_mapping_init†	—	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_csf_device	●
kbase_csf_alloc_command_stream_user_pages†	ioctl	kbase_csf_cpu_mmap_user_io_pages	mmap	kbase_queue	●

R/W permission?

Attack window?

Futhermore, we developed 15 end-to-end exploits for each of the exploit chains to confirm their real-world exploitability using CVE-2022-20409 and CVE-2023-48409.

Exploitability Evaluation

		Existing Exploits		PhantomMap Exploit	
CVE ID	Type	No Leak Required	Bypass Mitigations	No Leak Required	Bypass Mitigations
CVE-2025-21836	UAF	–	–	✔	✔
CVE-2024-46740	UAF	✔	✔	✔	✔
CVE-2024-26582	Double Free	✗	✔	✔	✔
CVE-2023-32882	OOB	✗	✔	✔	✔
CVE-2023-6560	OOB	✔	✗	✔	✔
CVE-2023-32837	OOB	✗	✗	✔	✔
CVE-2023-32832	UAF	✗	✔	✔	✔
CVE-2023-48409	OOB	✗	✔	✔	✔
CVE-2023-20938*	UAF	✗	✗	✔	✗
CVE-2023-0266	UAF	✗	✔	✔	✔
CVE-2022-38181	UAF	✔	✔	✔	✔
CVE-2022-20421*	UAF	✗	✗	✔	✗
CVE-2022-20409	Double Free	✗	✗	✔	✔

- 13 real-world CVEs, covering all common all common vulnerability types.
- 15 end-to-end privilege escalation exploits on Pixel6, Pixel7, Samsung A71.
- Successfully exploits CVE-2025-21836, for which no public exploit is known to date.

Exploitability Evaluation

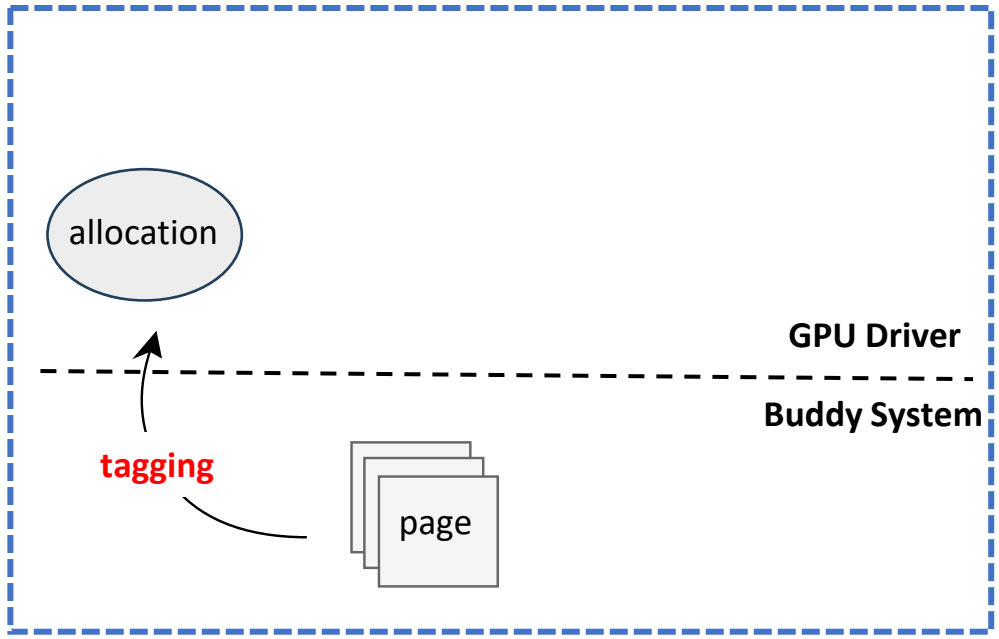
		Existing Exploits		PhantomMap Exploit	
CVE ID	Type	No Leak Required	Bypass Mitigations	No Leak Required	Bypass Mitigations
CVE-2025-21836	UAF	–	–	✔	✔
CVE-2024-46740	UAF	✔	✔	✔	✔
CVE-2024-26582	Double Free	✗	✔	✔	✔
CVE-2023-32882	OOB	✗	✔	✔	✔
CVE-2023-6560	OOB	✔	✗	✔	✔
CVE-2023-32837	OOB	✗	✗	✔	✔
CVE-2023-32832	UAF	✗	✔	✔	✔
CVE-2023-48409	OOB	✗	✔	✔	✔
CVE-2023-20938*	UAF	✗	✗	✔	✗
CVE-2023-0266	UAF	✗	✔	✔	✔
CVE-2022-38181	UAF	✔	✔	✔	✔
CVE-2022-20421*	UAF	✗	✗	✔	✗
CVE-2022-20409	Double Free	✗	✗	✔	✔

- 13 real-world CVEs, covering all common all common vulnerability types.
- 15 end-to-end privilege escalation exploits on Pixel6, Pixel7, Samsung A71.
- Successfully exploits CVE-2025-21836, for which no public exploit is known to date.

PhantomMap significantly outperforms existing methods: works without info-leak dependencies and bypasses stronger mitigations where prior exploits failed.

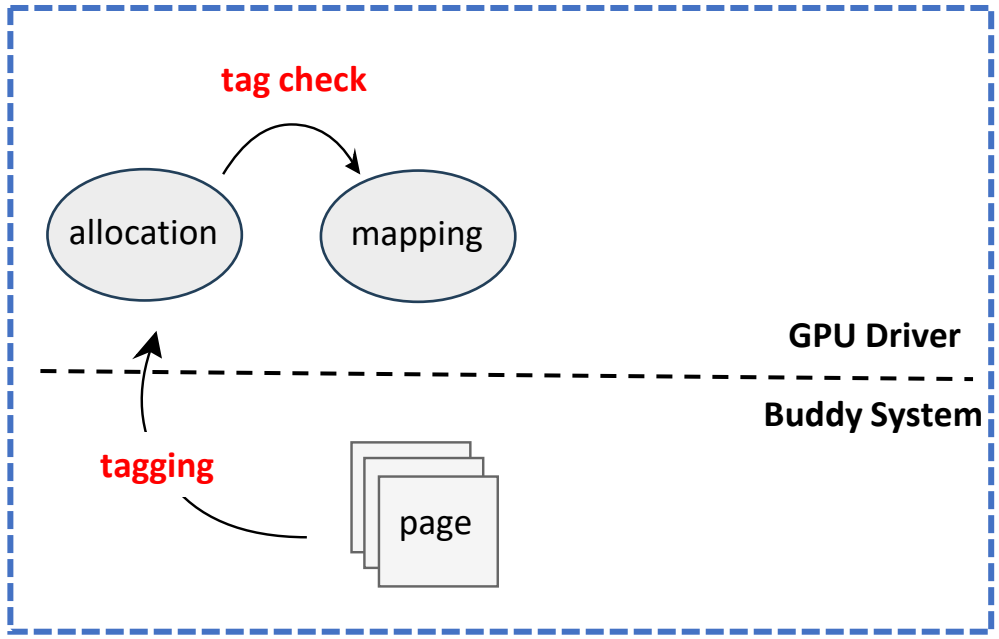
Lightweight Mitigation

Strategy



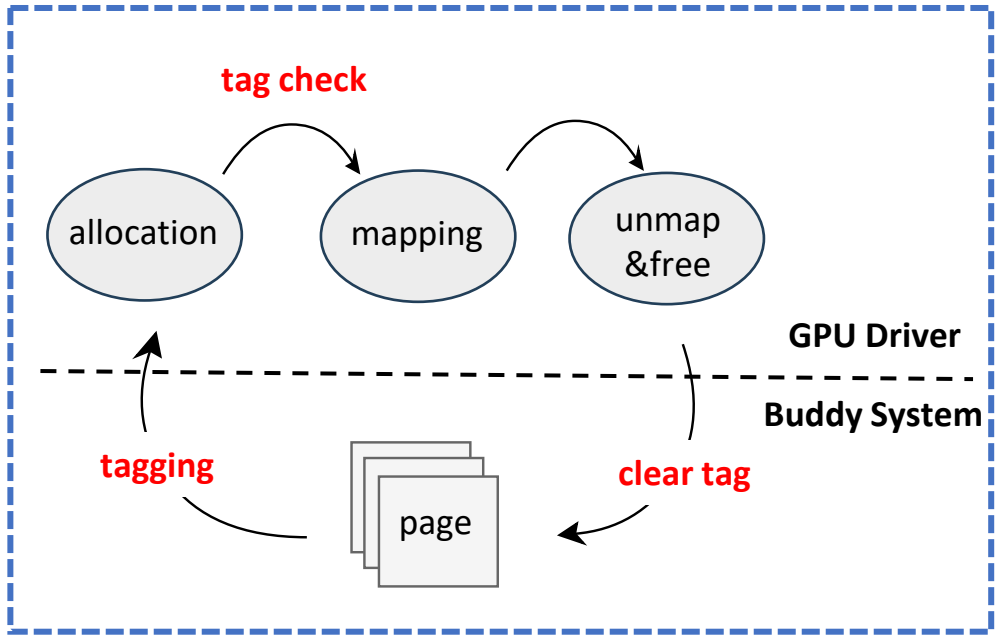
Lightweight Mitigation

Strategy



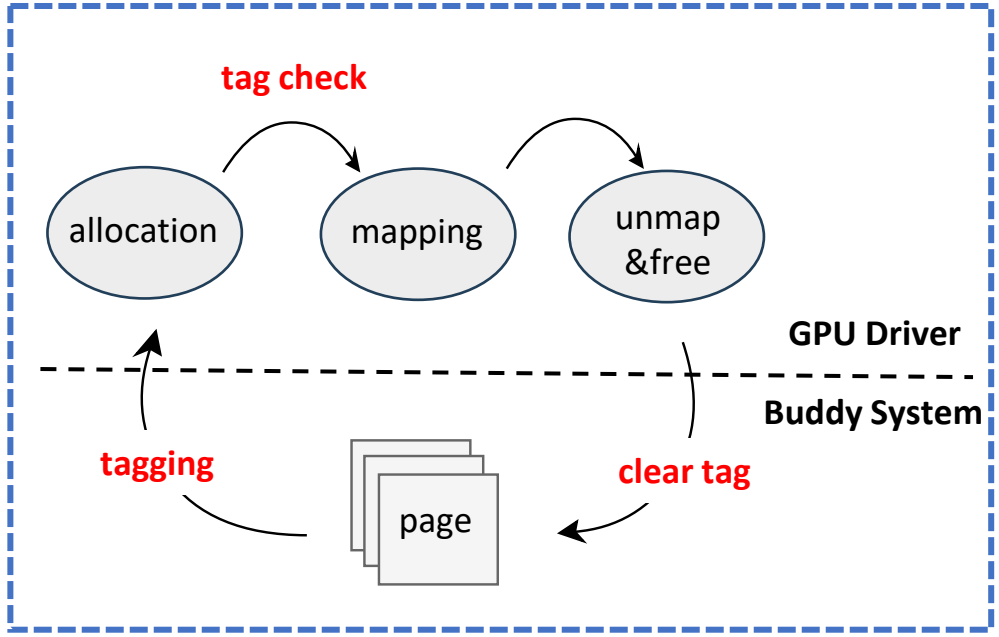
Lightweight Mitigation

Strategy



Lightweight Mitigation

Strategy

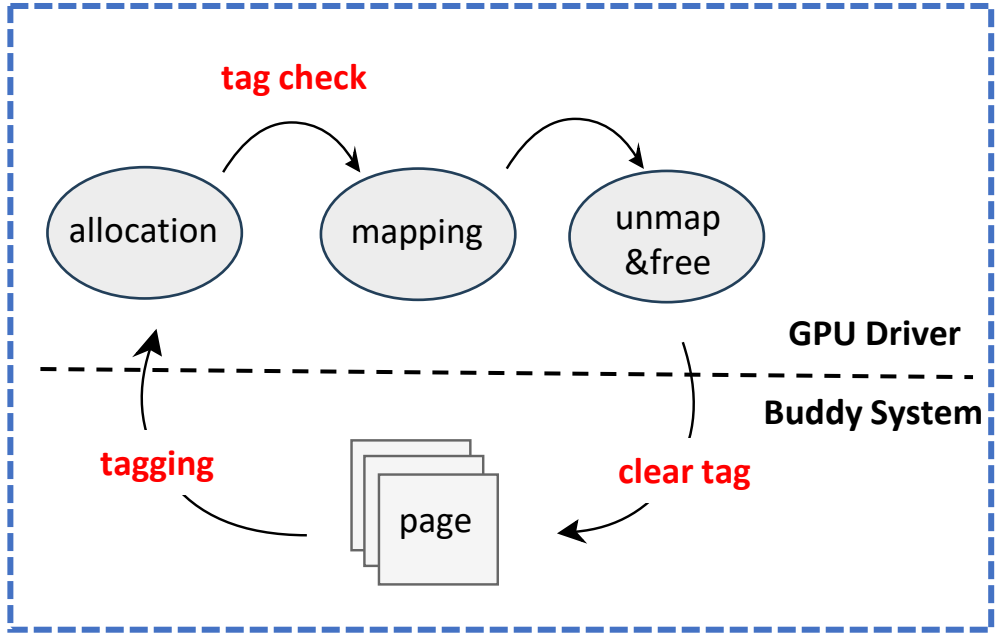


Implementation

```
**THE FIX**

// In kbase_csf_user_io_pages_vm_fault:


struct page *page = pfn_to_page(input_page_pfn);
if(!PageGpu(page)){
    ret = VM_FAULT_SIGBUS;
    goto exit;
}

// proceed to update GPU page table
ret = mgm_dev->ops.
mgm_vmf_insert_pfn_prot(..., input_page_pfn, ...);
```

Lightweight Mitigation

Strategy



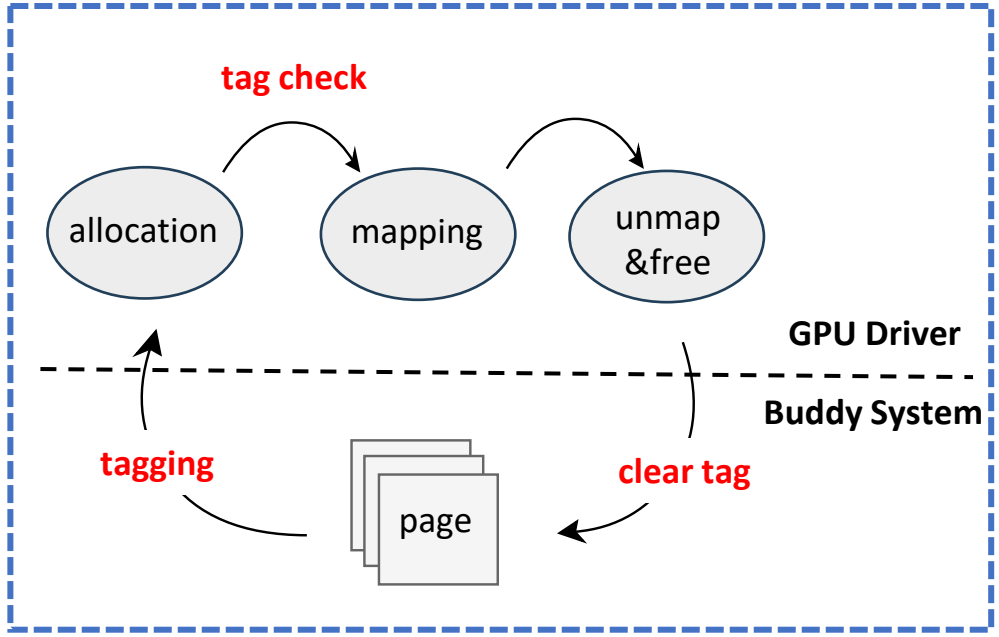
Implementation

```
**THE FIX**

// In kbase_csf_user_io_pages_vm_fault:

struct page *page = pfn_to_page(input_page_pfn);
if(!PageGpu(page)){
    ret = VM_FAULT_SIGBUS;
    goto exit;
}

// proceed to update GPU page table
ret = mgm_dev->ops.
mgm_vmf_insert_pfn_prot(..., input_page_pfn, ...);
```

Blocks mapping of any non-gpu memory

Lightweight Mitigation

Strategy



Implementation

```
**THE FIX**

// In kbase_csf_user_io_pages_vm_fault:

struct page *page = pfn_to_page(input_page_pfn);
if(!PageGpu(page)){
    ret = VM_FAULT_SIGBUS;
    goto exit;
}

// proceed to update GPU page table
ret = mgm_dev->ops.
mgm_vmf_insert_pfn_prot(..., input_page_pfn, ...);
```

Blocks mapping of any non-gpu memory

Re-enforcing stricter page validation !

Lightweight Mitigation

✅ **Lightweight**

- 21 representative GPU benchmarks tested.
- Evaluated on:
  - Pixel 6 (JM architecture)
  - Pixel 7 (CSF architecture)
- Performance Overhead:
  - Pixel 6: **0.56%** average overhead
  - Pixel 7: **0.34%** average overhead

Lightweight Mitigation

✅ Lightweight

- 21 representative GPU benchmarks tested.
- Evaluated on:
  - Pixel 6 (JM architecture)
  - Pixel 7 (CSF architecture)
- Performance Overhead:
  - Pixel 6: **0.56%** average overhead
  - Pixel 7: **0.34%** average overhead

☑ Robustness

- LTP test on Pixel 6 and Pixel 7 after enabling our mitigation. All test cases passed.
- Our mitigation introduces no functional regression.

Lightweight Mitigation

✅ Lightweight

- 21 representative GPU benchmarks tested.
- Evaluated on:
  - Pixel 6 (JM architecture)
  - Pixel 7 (CSF architecture)
- Performance Overhead:
  - Pixel 6: **0.56%** average overhead
  - Pixel 7: **0.34%** average overhead

☑ Robustness

- LTP test on Pixel 6 and Pixel 7 after enabling our mitigation. All test cases passed.

- Our mitigation introduces no functional regression.

Lightweight Mitigation

✅ Lightweight

- 21 representative GPU benchmarks tested.
- Evaluated on:
  - Pixel 6 (JM architecture)
  - Pixel 7 (CSF architecture)
- Performance Overhead:
  - Pixel 6: **0.56%** average overhead
  - Pixel 7: **0.34%** average overhead

☑ Robustness

- LTP test on Pixel 6 and Pixel 7 after enabling our mitigation. All test cases passed.

- Our mitigation introduces no functional regression.

🥳 Powerful

- Strictly prevents mapping of **all non-GPU pages**, to effectively mitigate PhantomMap.

Takeaways

1. **First systematic analysis of the Mali GPU memory-mapping mechanism, revealing two security weaknesses.**
2. **A novel GPU-assisted kernel exploitation technique.**
3. **Comprehensive real-world exploitability evaluation.**
4. **A dedicated static analyzer to identify all viable exploit chains.**
5. **A lightweight and effective mitigation.**