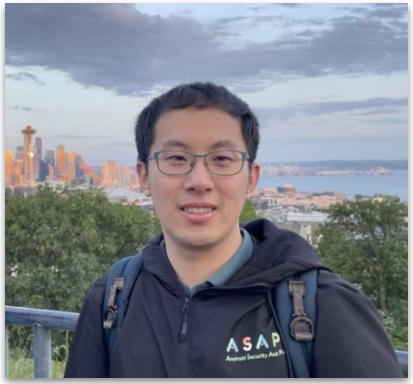
black hat USA 2022

Android Universal Root
Exploiting xPU Drivers

Xingyu Jin
Richard Neal
Jon Bottarini

Who are we?



Xingyu Jin
@1ce0ear

Richard Neal
@ExploitDr0id

Jon Bottarini
@jon_bottarini

Google

android security

What are we talking about?

- Rooting exploits on Android
  - In-the-wild
  - Internal research
- Android Partner Vulnerability Initiative

Some old driver vulns

```
= {SAM_EXP,        "/dev/exynos-mem",
= {ARAGORN_EXP,    "/dev/video1",
= {GIMLI_EXP,      "/dev/DspBridge",
= {MERRY_EXP,      "/dev/s5p-smem",
= {FRODO_EXP,      "/dev/exynos-mem",
= {LEGOLAS_EXP,    "/dev/graphics/fb5",
= {GANDALF_EXP,    "/dev/msm_camera/config0",
= {BOROMIR_EXP,    "/dev/camera-isp",
= {BOROMIR2_EXP,   "/dev/camera-eis",
= {BOROMIR3_EXP,   "/dev/camera-sysram",
```

mtk-su Exploit

Mobile

First Binder Exploit Linked to SideWinder APT Group

We found malicious apps that work together to compromise devices and collect user data. One of the apps, called Camero, exploits CVE-2019-2215, a flaw that exists in Binder. This is the first instance in the wild that exploits said UAF vulnerability.

We were able to download five exploits from the C&C server during our investigation. They use the vulnerabilities CVE-2019-2215 and MediaTek-SU to get root privilege.

XDA Developers

Rapid Temporary Root for HD 8 & HD 10

diplomatic · Feb 26, 2019 · fire hd 10 | fire hd 8 | root

Software root method for Mediatek MT816x, MT817x and MT67xx!

A tool that gives you a temporary root shell with Selinux permissive to do with as you please

diplomatic
Senior Member

Software root method for Mediatek MT816x, MT817x and MT67xx!

A tool that gives you a temporary root shell with Selinux permissive to do with as you please

STATUS

Confirmed Working

Fire HD 8 8th gen (2018) (thanks @xyz`) -- up to Fire OS 6.3.0.1 only
Fire HD 8 7th gen (2017) -- up to Fire OS 5.6.4.0 build 636558520 only
Fire HD 8 6th gen (2016) (thanks @bibikalka) -- up to Fire OS 5.3.6.4 build 626536720
Fire HD 10 7th gen (2017) (thanks @bibikalka) -- up to Fire OS 5.6.4.0 build 636558520 only
Fire TV 2 2015 (mt8173-based) (thanks @el7145) -- up to Fire OS 5.2.6.9 only
Fire 7 9th gen (2019) (thanks @Michajin) -- up to Fire OS 6.3.1.2 build 0002517050244 only
Fire HD 10 9th gen (2019) -- up to Fire OS 7.3.1.0 only
Various phones and tablets up to Android 9.x (see link below for full list)
Note that for Fire OS 5, OS version 5.3.x.x is newer than 5.6.x.x.

XDA Developers

Amazing Temp Root for MediaTek ARMv8 [2020-08-24]

diplomatic · Apr 17, 2019 · mediatek mt67xx root

Forums › General Development › Android Development and Hacking › **Miscellaneous Android Development**

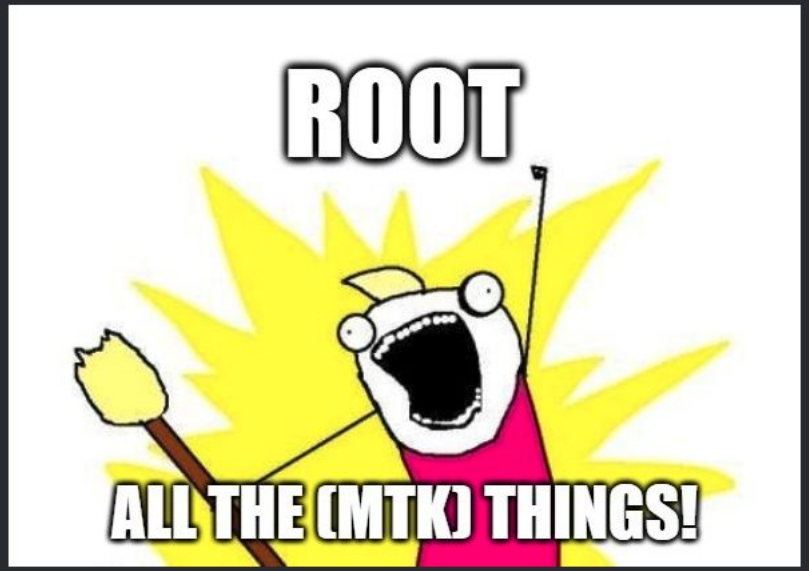
1 2 3 ... 81

Search This t

D

diplomatic
Senior Member

Apr 17, 2019



```
SL05:/ $ /data/local/tmp/mtk-su -v
```

```
Building symbol table
kallsyms_addresses pa 0x40ed5c50
kallsyms_num_syms 49679, addr_count 49679
kallsyms_names pa 0x40f064a0, size 642508
kallsyms_markers pa 0x40fa3270
kallsyms_token_table pa 0x40fa3580
kallsyms_token_index pa 0x40fa3910
```

```
Parsing current_is_single_threaded
c04aa5f8: LDR Rt, [PC, #0x80]  ; 0xc04aa680
Possible list_head tasks at offset 0x2b0
comm swapper/0 at offset 0x44c
Found own task_struct at node 1
cred VA: 0xcafa2780
init_task VA: 0xc140c4f8
```

```
Parsing avc_denied
c044cfc8: LDR Rt, [PC, #0x54]  ; 0xc044d024
selinux_enforcing VA: 0xc15af4dc
```

Reverse engineering the exploit

```
6f35a3ff: /dev/ion
4b4ab49b: /dev/mtk_cmdq
6d7a00fc: /proc/mtk_cmdq
```

```
SL05:/ $ ls -lZ /dev/mtk_cmdq
crw-r--r-- 1 system system u:object_r:mtk_cmdq_device:s0 250,   0 2010-01-03 01:
00 /dev/mtk_cmdq
SL05:/ $
```

```
$ sesearch --allow -t mtk_cmdq_device policy | grep appdomain
allow appdomain mtk_cmdq_device:chr_file { ioctl open read };
$
```

CVE-2020-0069

- CMDQ_IOCTL_EXEC_COMMAND
  - Send a buffer of opcodes from user -> kernel
  - Opcodes - CMDQ_CODE_ENUM

```
245 enum CMDQ_CODE_ENUM {
246         /* these are actual HW op code */
247         CMDQ_CODE_READ = 0x01,
248         CMDQ_CODE_MOVE = 0x02,
249         CMDQ_CODE_WRITE = 0x04,
250         CMDQ_CODE_POLL = 0x08,
251         CMDQ_CODE_JUMP = 0x10,
252         CMDQ_CODE_WFE = 0x20,   /* wait for event and clear */
253         CMDQ_CODE_EOC = 0x40,   /* end of command */
```

SELinux Fix

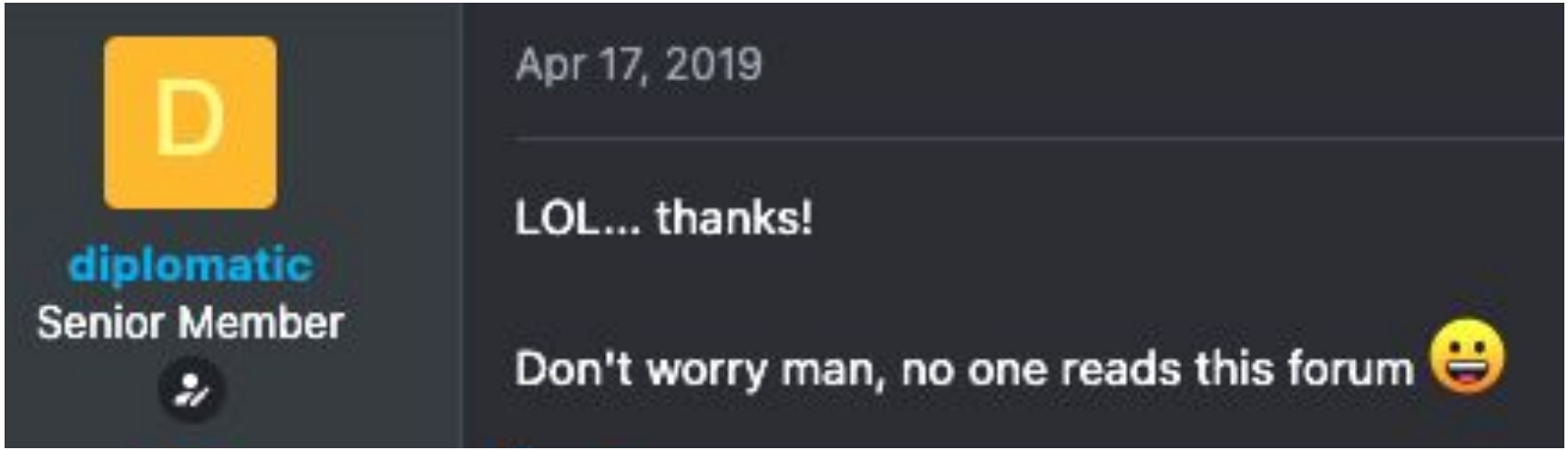
- Restrict access via SELinux policy:

```
1d0
< allow appdomain mtk_cmdq_device:chr_file { ioctl open read };
< allow init mtk_cmdq_device:chr_file { open read setattr };
---
> allow init mtk_cmdq_device:chr_file setattr;
```

- Can't open device node:

```
$ ./mtk-su
Failed critical init step 1
```

Lessons Learned



GPU Driver: Perfect Local Attack Surface

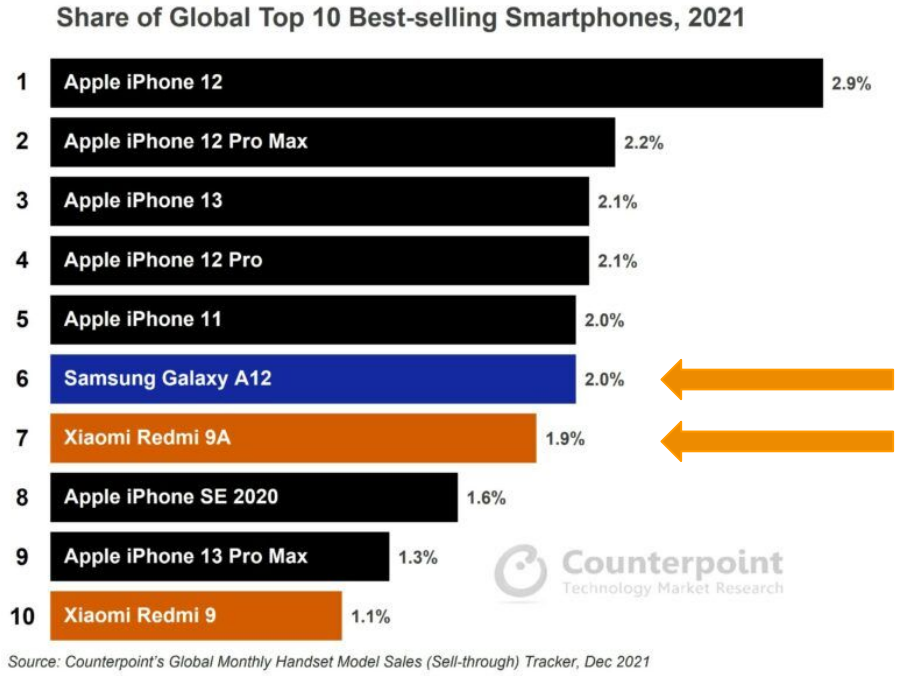
- It's impossible to restrict unprivileged users from accessing GPU driver
- GPU device driver exports a lot of functionality to userspace
- Mobile GPU
  - ARM: Mali
  - Qualcomm: Adreno
  - Imagination Technologies: PowerVR

Google received a lot of security reports

Only 1 in 2019

PowerVR GPU is everywhere

- PowerVR may have the biggest market share in numbers (MediaTek / UniSoc)



Use PowerVR GPU

Intro to Bridge APIs

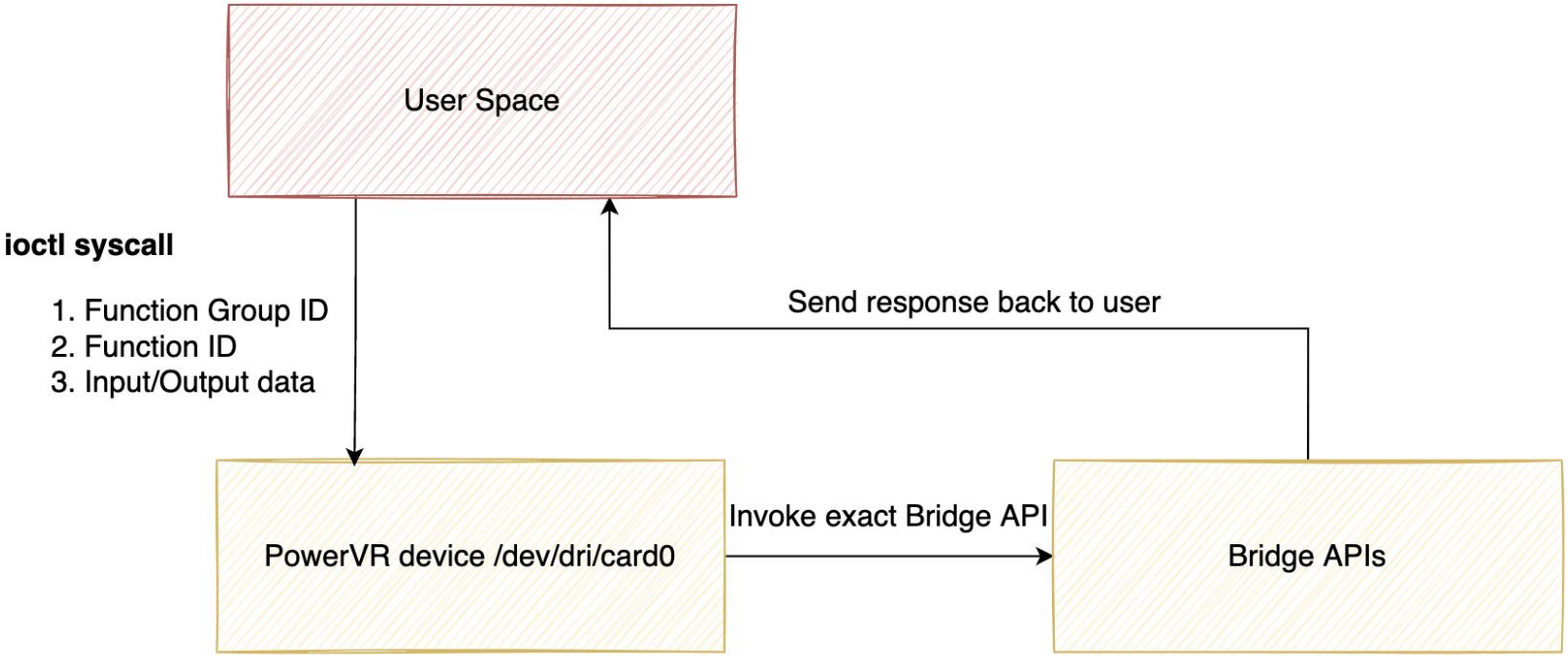
- No surprise, unprivileged user can talk to PowerVR GPU driver

- PowerVR GPU driver exports hundreds of kernel functions to userspace
  - Exported functions are called “Bridge functions” by PowerVR developers

- Three steps
  - Open device
  - Send `ioctl` code and arguments
  - Get response

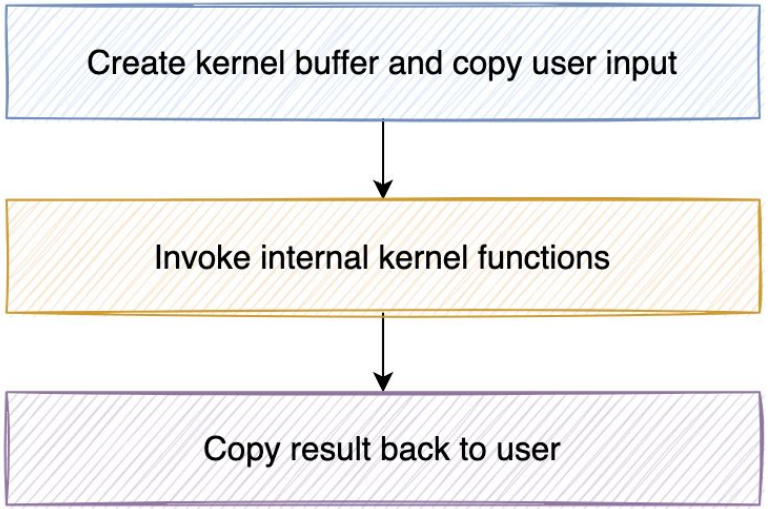
Intro to Bridge APIs

- Bridge APIs



Intro to Bridge APIs

- Overview of how Bridge APIs work
- Every step is buggy



Heap overflow in Bridge APIs

- Call Bridge API `PVRSRVBridgeSyncPrimOpTake`
  - Group ID 2, Function ID 9

```
SetDispatchTableEntry(PVRSRV_BRIDGE_SYNC, // Group id 2
                      PVRSRV_BRIDGE_SYNC_SYNCPRIMOPTAKE, // Function id 9
                      PVRSRVBridgeSyncPrimOpTake, NULL, bUseLock);
```

- Input data structure `PVRSRV_BRIDGE_IN_SYNCPRIMOPTAKE`
- Calculate kernel buffer size based on user input
  - Integer overflow

```
IMG_UINT32 ui32BufferSize =
    (psSyncPrimOpTakeIN->ui32ClientSyncCount * sizeof(IMG_UINT32)) +
    (psSyncPrimOpTakeIN->ui32ClientSyncCount * sizeof(IMG_UINT32)) +
    (psSyncPrimOpTakeIN->ui32ClientSyncCount * sizeof(IMG_UINT32)) +
    (psSyncPrimOpTakeIN->ui32ServerSyncCount * sizeof(IMG_UINT32)) + 0;

pArrayArgsBuffer = OSAllocMemNoStats(ui32BufferSize);
```

Heap overflow in Bridge APIs

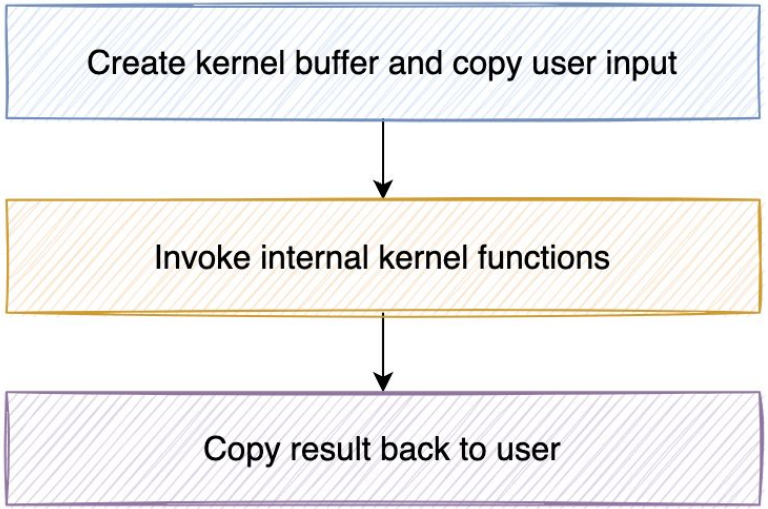
- GPU driver does have a a lot of “sanity checks”
  - Always by checking if an unsigned integer is above 0 :-/

```
/* Copy the data over */
if (psSyncPrimOpTakeIN->ui32ClientSyncCount * sizeof(IMG_UINT32) > 0)
```

- Massive trouble: every bridge API is written in this way
  - Several CVEs are assigned for this issue.
  - Good news: sometimes integer overflow cancels itself

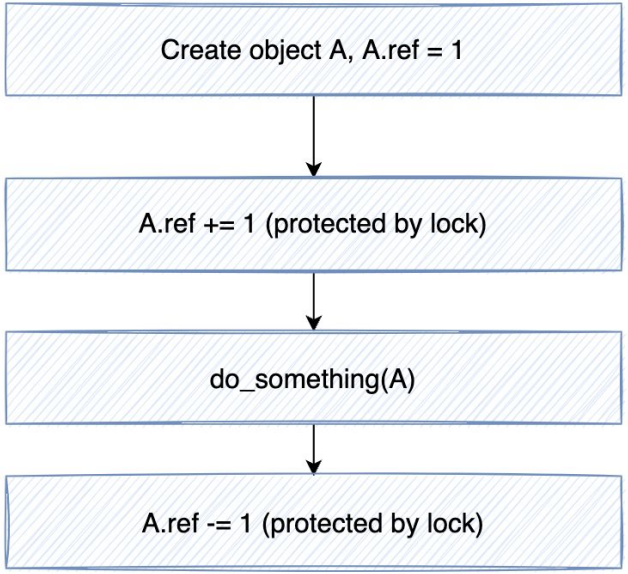
Race Condition in Bridge APIs

- Invoke internal kernel functions
  - Create object
  - Use object
  - Release object

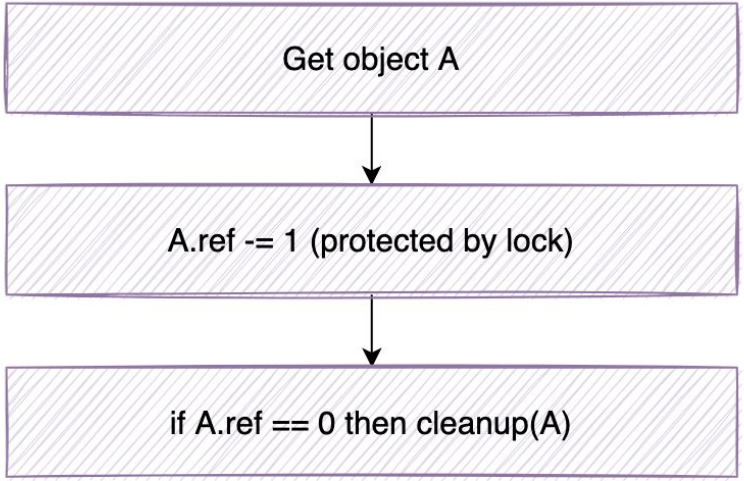


Race Condition in Bridge APIs

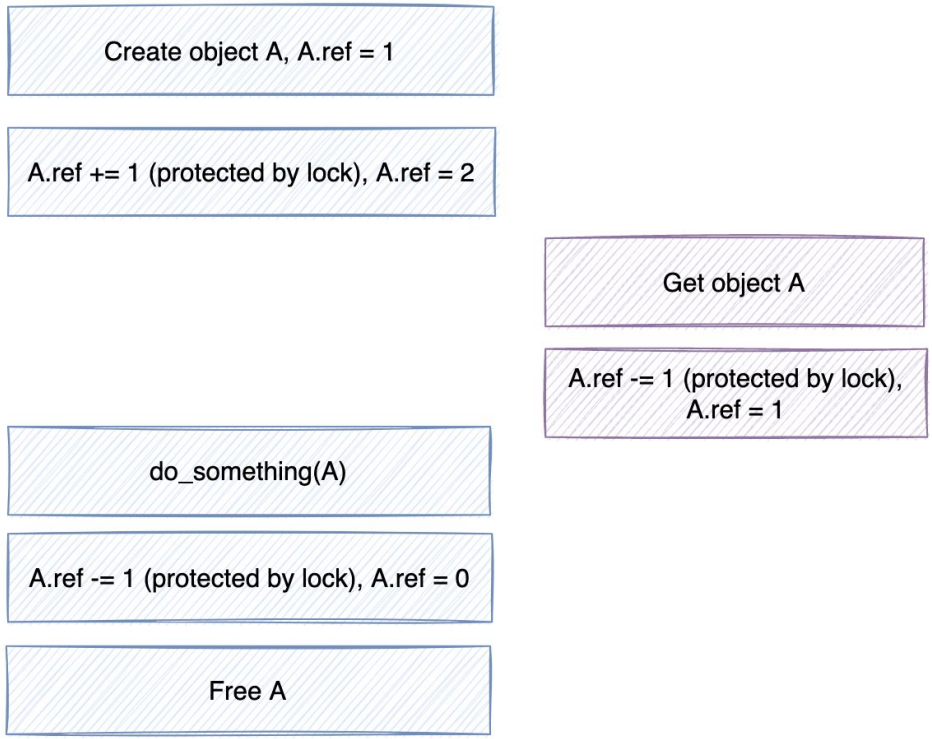
- Bridge API X
  - Create internal kernel object
  - Use kernel object for computation
  - Return a handle to user



- Bridge API Y
  - Find kernel object by handle
  - Decrement reference count



Race Condition in Bridge APIs



Race Condition in Bridge APIs

Create object A, A.ref = 1

A.ref += 1 (protected by lock), A.ref = 2

Get object A

A.ref -= 1 (protected by lock), A.ref = 1

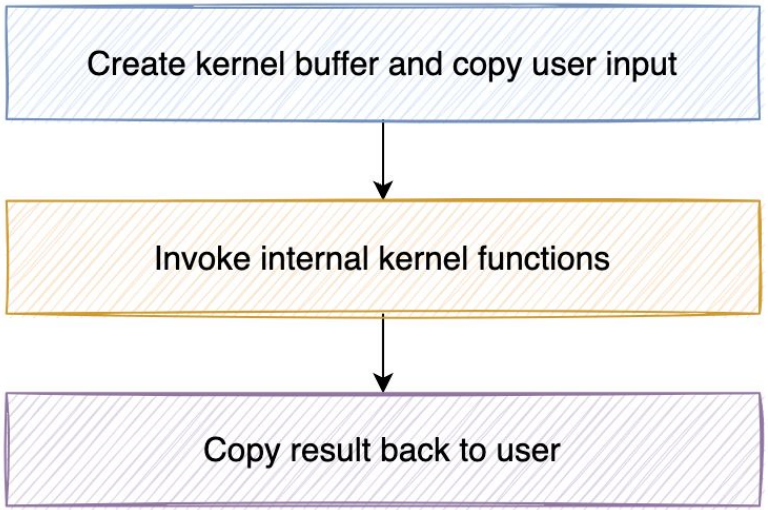
Get object A

A.ref -= 1 (protected by lock), A.ref = 0, Free A

do_something(A) <- UAF

Read Uninitialized Heap Memory in Bridge APIs

- Copy result back to user
  - Initialize the kernel object
  - Copy kernel object data back to user space



Read Uninitialized Heap Memory in Bridge APIs

- Copy result to user back
  - do_something(A)
  - A->data is copy back to user space

Fail early -> A is not initialized

- Easy leak kernel heap pointer / KASLR bypass (Arbitrary slab size)

```
00000000  20 20 20 20 20 09 30 09 09 30 09 09 30 09 09 30  |     .0..0..0..0|
00000010  09 09 30 09 09 30 09 09 30 09 09 39 36 35 09 09  |..0..0..0..965..|
00000020  30 0a 63 63 63 69 5f 69 70 63 5f 38 20 20 20 20  |0.ccci_ipc_8    |
00000030  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  |                |
00000040  20 20 09 30 09 09 30 09 09 30 09 09 30 09 09 30  |  .0..0..0..0..0|
00000050  09 09 30 09 09 30 09 09 39 36 35 09 09 30 0a 63  |..0..0..965..0.c|
00000060  63 63 69 5f 69 70 63 5f 37 20 20 20 20 20 20 20  |cci_ipc_7       |
00000070  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 09  |               .|
00000080  30 09 09 30 09 09 30 09 09 30 09 09 30 09 09 30  |0..0..0..0..0..0|
00000090  09 09 30 09 09 39 36 35 09 09 30 0a 63 63 63 69  |..0..965..0.ccci|
000000a0  5f 69 70 63 5f 36 20 20 20 20 20 20 20 20 20 20  |_ipc_6          |
000000b0  20 20 20 20 20 20 20 20 20 20 20 20 09 30 09 09  |             .0..|
000000c0  30 09 09 30 09 09 30 09 09 30 09 09 30 09 09 30  |0..0..0..0..0..0|
000000d0  09 09 39 36 34 09 09 30 0a 63 63 63 69 5f 69 70  |..964..0.ccci_ip|
000000e0  63 5f 35 20 20 20 20 20 20 20 20 20 20 20 20 20  |c_5             |
000000f0  20 20 20 20 20 20 20 20 20 09 30 09 09 30 09 09  |          .0..0..|
00000100  30 09 09 30 09 09 30 09 09 30 09 09 30 09 09 39  |0..0..0..0..0..9|
00000110  36 34 09 09 30 0a 63 63 63 69 5f 69 70 63 5f 34  |64..0.ccci_ipc_4|
00000120  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  |                |
00000130  20 20 20 20 20 20 09 30 09 09 30 09 09 30 09 09  |      .0..0..0..|
00000140  30 09 09 30 09 09 30 09 09 30 09 09 39 36 34 09  |0..0..0..0..964.|
00000150  09 30 0a 63 63 63 69 5f 69 70 63 5f 33 20 20 20  |.0.ccci_ipc_3   |
```

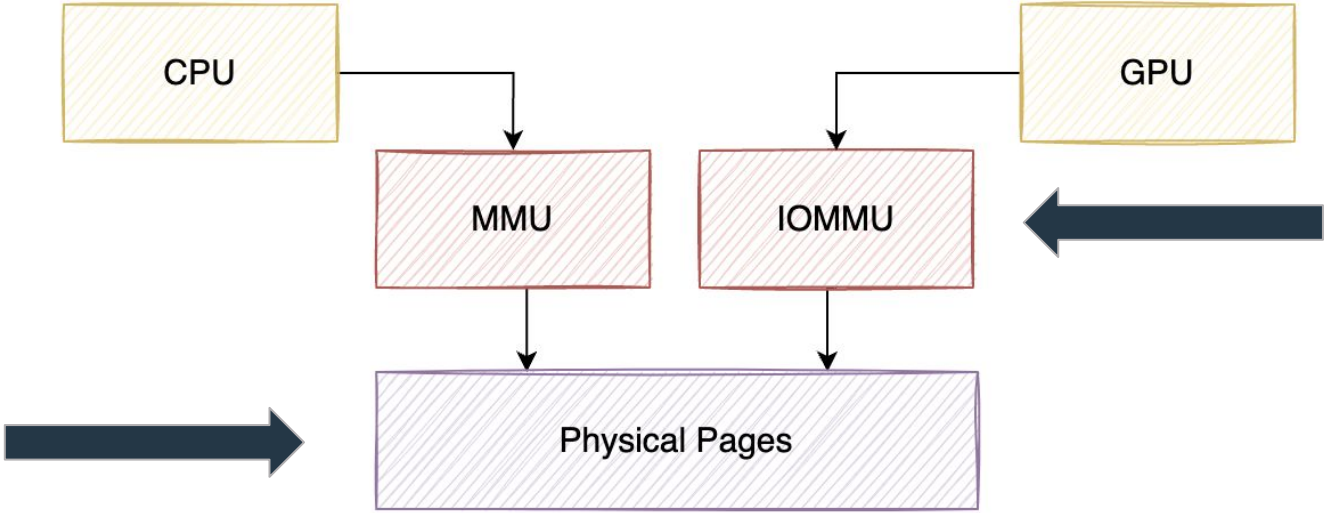
Bridge APIs: Summary

- There are other issues
  - Reference count overflow
  - Arbitrary stack overflow
  - Unlink UAF
  - … Please see also our reference slide later

- Classic memory corruption / race condition bugs
- These bugs are exploitable for rooting PowerVR devices
  - No more introduction, we have something … much more powerful

PowerVR Memory Management: GPU VA <-> PA

- CPU VA <-> PA
- GPU VA <-> PA
- GPU VA <-> PA <-> CPU VA



PowerVR Memory Management: GPU VA <-> PA

- **PMR** (Physical Memory Resource)
  - Call Bridge APIs => obtain a PMR handle
  - Use PMR handle for mapping GPU / CPU virtual memory

Create PMR

- How many physical pages? ← **-2 pages**
- How many virtual pages?
- GPU/CPU memory properties
- Sparse memory mapping table
- …

PowerVR Memory Management: GPU VA <-> PA

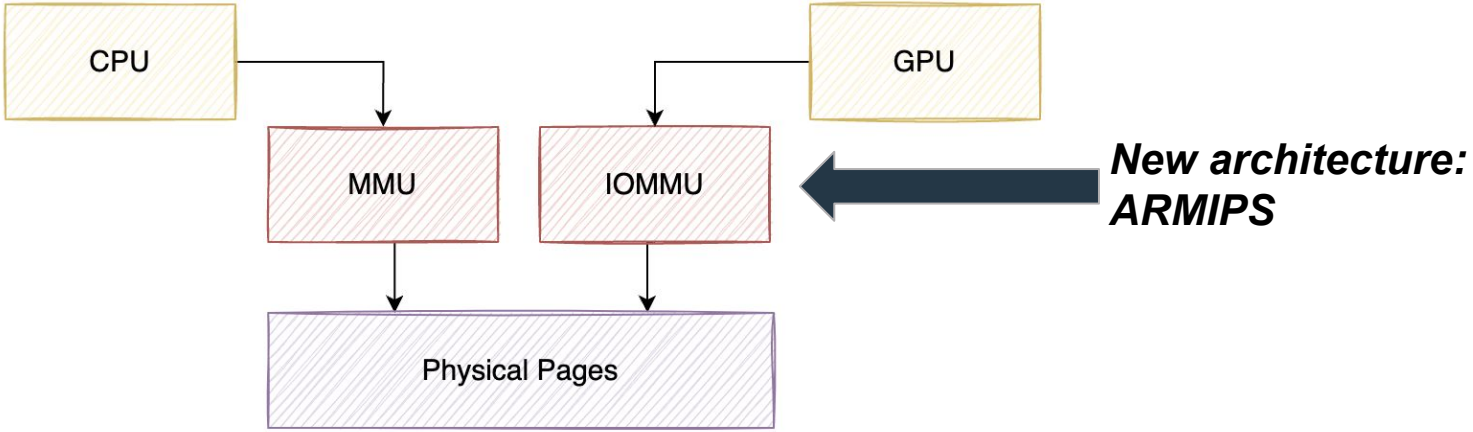
- Corrupt GPU VA mapping
  - Map to arbitrary physical memory we want

GPU VA mapping

- All kinds of object handles
- The number of mapped pages
- Physical page offset
- Memory properties
- …

PowerVR Memory Management: GPU VA <-> PA

- If we want to operate `ARM64` IOMMU
  - Then operate the `IOMMU` in an `ARM64` way for sure
- User space can invoke a typical Bridge API and point out to the driver that we want to operate the ARM64 IOMMU in…
  - ***MIPS way first…***



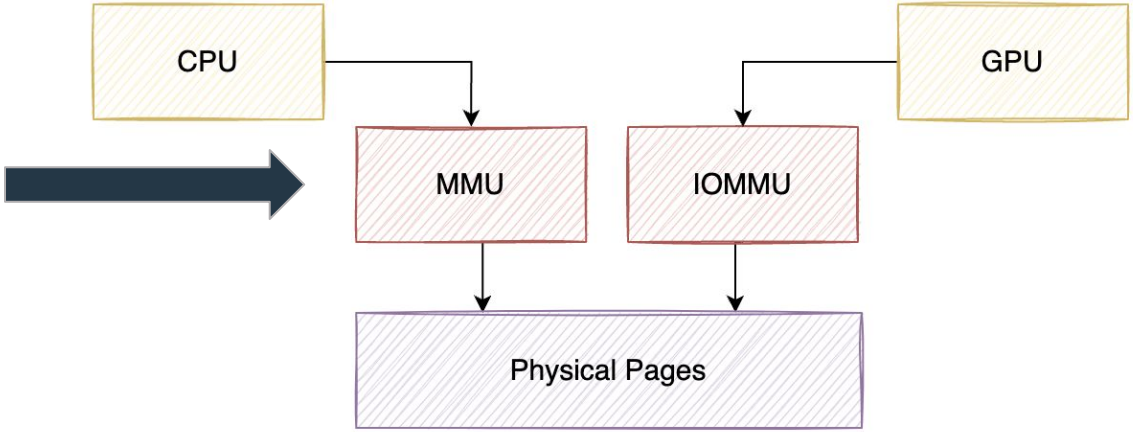
PowerVR Memory Management: GPU VA <-> PA

- Now we can operate the `IOMMU` in ***MIPS*** way on ***ARM64*** architecture
- Completely corrupt the page tables
  - E.g. viewing weird data from mmap syscall immediately

```
00000000  10 34 2b 04 10 34 2b 04 10 34 2b 04 10 34 2b 04  |.4+..4+..4+..4+.|
00000010  10 34 2b 04 10 34 2b 04 10 34 2b 04 10 34 2b 04  |.4+..4+..4+..4+.|
00000020  10 34 2b 04 10 34 2b 04 10 34 2b 04 10 34 2b 04  |.4+..4+..4+..4+.|
00000030  10 34 2b 04 10 34 2b 04 10 34 2b 04 10 34 2b 04  |.4+..4+..4+..4+.|
```

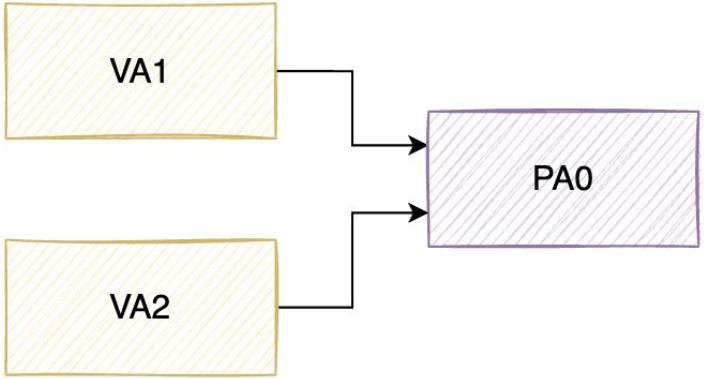
PowerVR Memory Management: CPU VA <-> PA

- Now let's take a look at the CPU side
  - CPU VA <-> PA



PowerVR Memory Management: CPU VA <-> PA

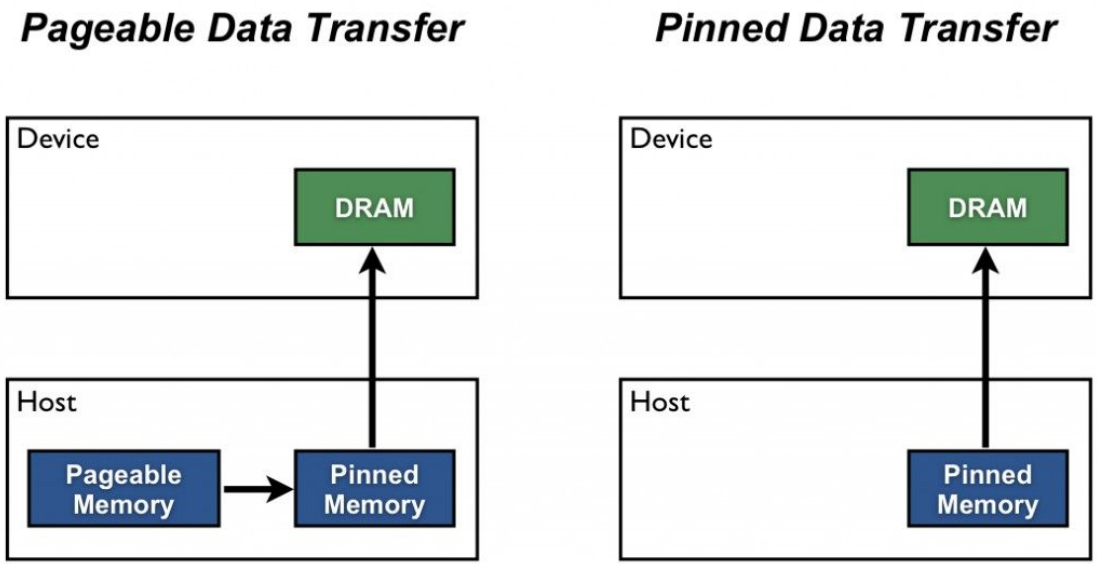
- Reserve physical pages, get `PMR` handle `hPMR`
- Get mapped CPU virtual memory by `mmap`
  - `mmap(hDev, ..., hPMR << PAGE_SHIFT)`
- Tracking the number of memory mappings for security purposes by `PMR->iRefCount`




```
PMR->iRefCount = 3
  (1 by default)
```

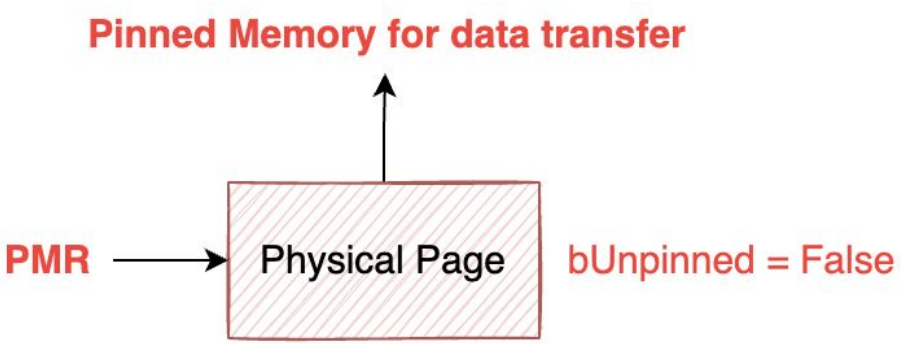
PowerVR Memory Management: Pinned Mem

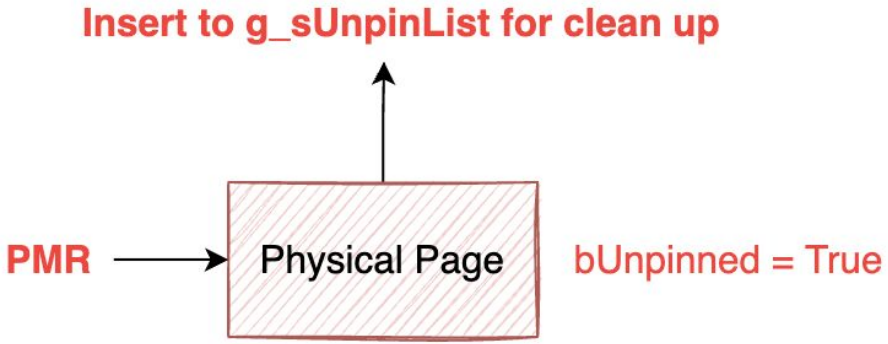
- Pinned Memory (According to Nvidia official web page)
  - Pinned memory is used for data transfers from the device to the host.
  - Allocate physical memory => Avoid cost in data transfer



PowerVR Memory Management: Pinned Mem

- "Pinned" physical pages reserved by `PMR`
  - `PVRSRVBridgeDevmemInt`**`Pin`**
- "Unpinned" physical pages reserved by `PMR`
  - `PVRSRVBridgeDevmemInt`**`Unpin`**





PowerVR Memory Management: Pinned Mem

- Allocate page A, B, C
- Unpin page A, B, C
  - `g_sUnpinList: {A, B, C}`

PowerVR Memory Management: Pinned Mem

- Register Linux kernel shrinker
- A shrinker is an internal kernel callback routine
  - When memory is tight => free pages from `g_sUnpinList`

```
static struct shrinker g_sShrinker =
{
  .count_objects = _CountObjectsInPagePool,
  .scan_objects = _ScanObjectsInPagePool,
  .seeks = DEFAULT_SEEKS
};
```

PowerVR Memory Management: Pinned Mem

- Allocate page A, B, C
- Unpin page A, B, C => `g_sUnpinList: {A, B, C}`

- Unpin page A, B, C => `g_sUnpinList: {A, B, C, A, B, C}`

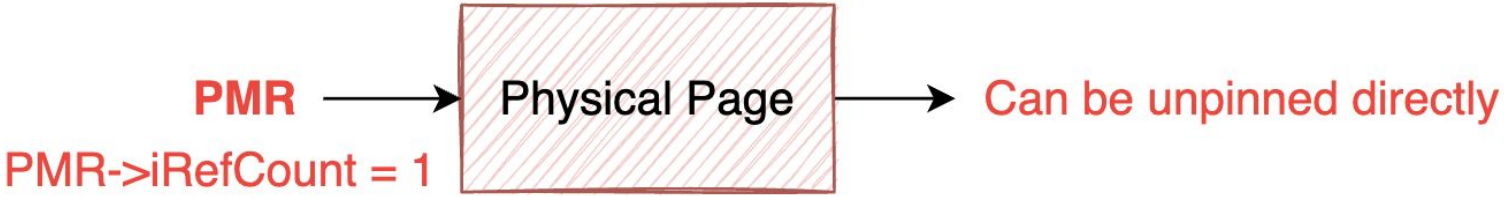
PowerVR Memory Management: Pinned Mem

- CVE-2022-20122 Free arbitrary page arbitrary time by Unpin API
  - `PVR_ASSERT(psOSPageArrayData->bUnpinned == IMG_FALSE)`
  - `PVR_ASSERT` is not enabled in production :-/

- PVR_ASSERT is enabled for code static analysis checker only
  - `#if defined(__KLOCWORK__)`
  - In production, it does nothing :-/

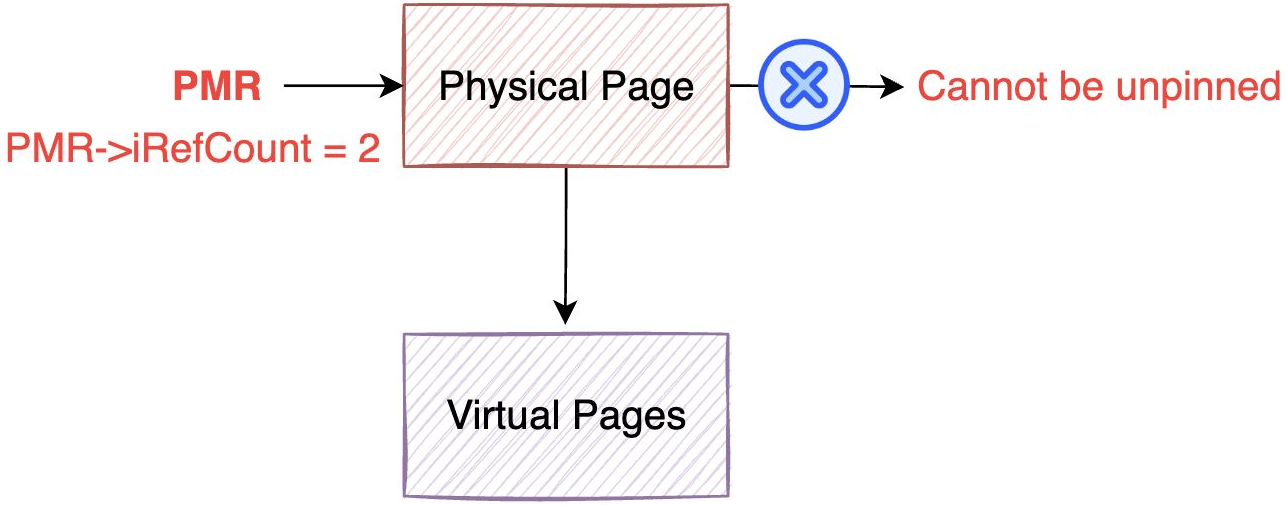
PowerVR Memory Management: Pinned Mem

- Let's visit the existing security checks
  - If physical pages are mapped to somewhere else (`PMR->iRefCount` > 1)
  - The physical pages are not allowed to be unpinned



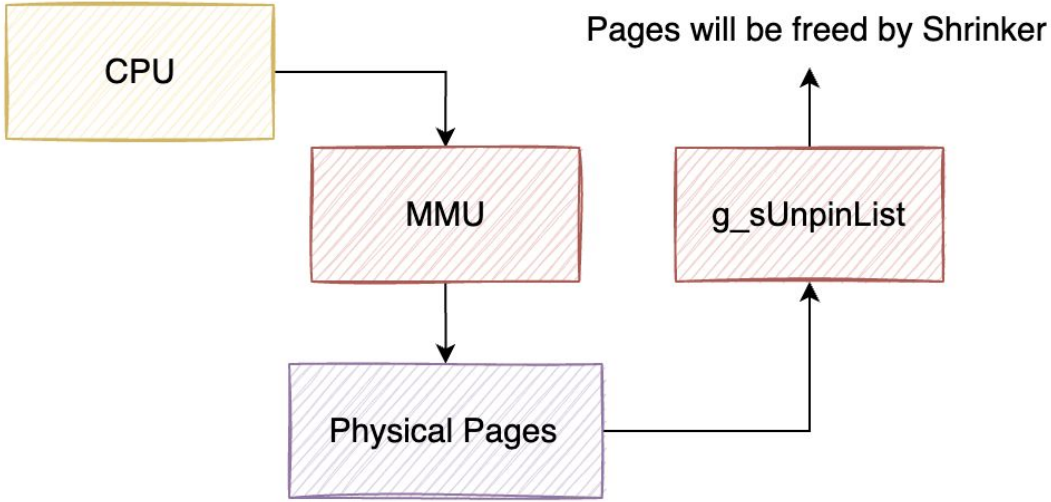
PowerVR Memory Management: Pinned Mem

- The following request is **illegal**
  - `hPMR = create_pmr() // PMR->iRefCount == 1`
  - `mmap(..., hPMR) // PMR->iRefCount == 2`
  - `unpin_mem(hPMR) // Failed because iRefCount > 1`



PowerVR Memory Management: Pinned Mem

- The following request is **legal**
  - `hPMR = create_pmr() // PMR->iRefCount == 1`
  - `unpin_mem(hPMR) // Succeed, move pages from PMR to shrinker`
  - `mmap(..., hPMR << PAGE_SHIFT)`



PowerVR Memory Management: Pinned Mem

- Effectively trigger shrinker callback
  -
```
for (int i = 0; i < 0x40000; i++) {
    hPMR = create_pmr();
    unpin_mem(hPMR);
    va[i] = mmap(..., hPMR << PAGE_SHIFT);
}
```

- CVE-2021-39815 (Discovered in late Feb, 2022)
  - A subtle logic bug
  - User space can R/W arbitrary freed physical pages!!

- PowerRoot

PowerRoot: Bypass SELinux

- Some devices don't allow you to `mmap` GPU device
  - allow appdomain dri_device (chr_file (ioctl read write open)) => no `mmap`

- Who has the privilege to `mmap` GPU device?
  - allow **hal_graphics_composer_default** dri_device (chr_file (ioctl read write getattr lock append map open watch watch_reads))

- Create OpenCL project
  - `void* svm_mem = cl::SVMAlloc(...)`
  - SVM (Shared Virtual Memory) still works

PowerRoot: Bypass SELinux

- `opencl.so`
  - Implemented by several vendor specific libraries
    - PVR + MTK libraries

- Reverse engineering these vendor libraries
  - Rigorous security checks on user space library

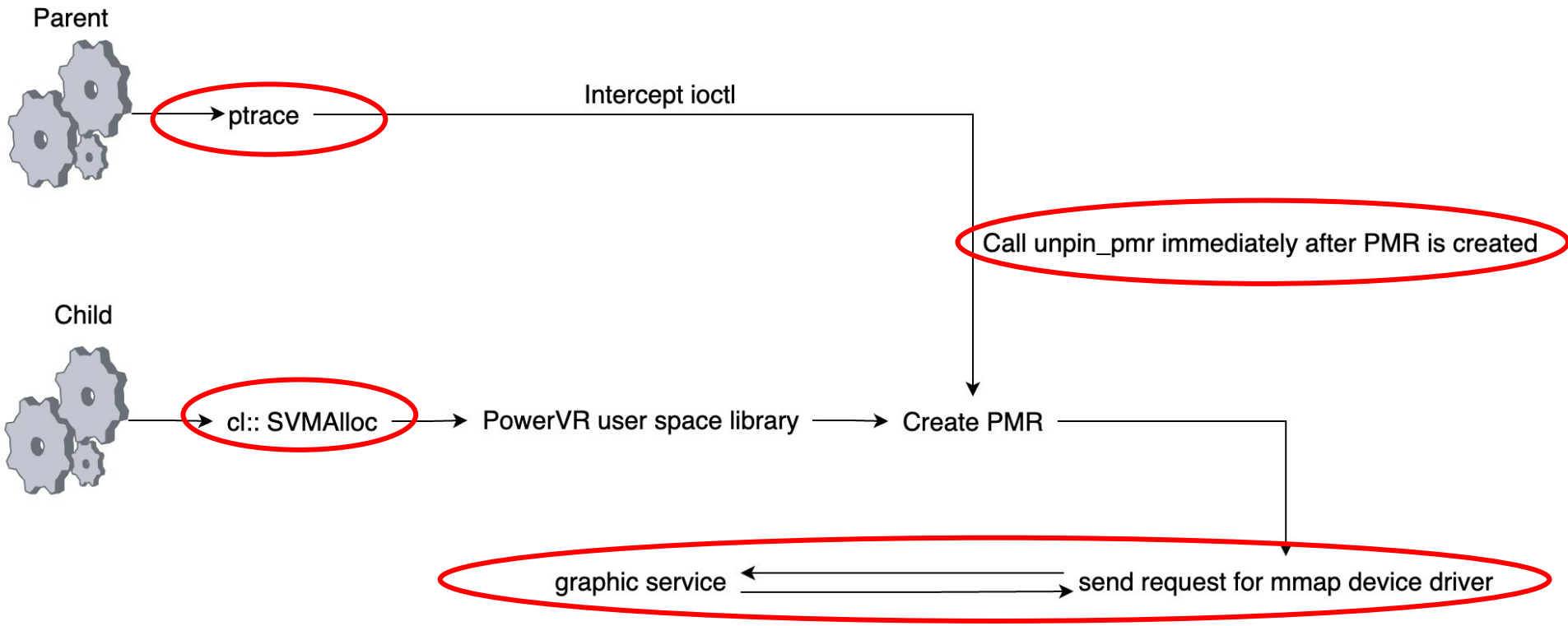
Pseudocode-A | Stack of sub_2D5B0

```
int __fastcall do_PVRSRVAcquireCPUMapping(unsigned int *a1, int *a2)
{
  unsigned int v2; // r5
  int v5; // r6
  unsigned int v6; // r0
  int cpu_va; // r0
  int v8; // r6
  unsigned int *v10; // r0
  int *v11; // r8
  unsigned int v12; // r1
  int v13; // r0

  v2 = *a1;
  pthread_mutex_lock(**(pthread_mutex_t ***)(*a1 + 40));
  v5 = *(_DWORD *)(v2 + 28);
  pthread_mutex_unlock(**(pthread_mutex_t ***)(v2 + 40));
  if ( (v5 & 0x48) != 0 )
  {
    PVRSRVDebugPrintf(
      2,
      (char *)&unk_1682D,
      2548,
      "%s: Allocation is currently unpinned or a secure buffer. Not possible to map to CPU!",
      "DevmemAcquireCpuVirtAddr");
    return 131;
  }
  if ( (v5 & 0x100) != 0 )
  {
    PVRSRVDebugPrintf(
      2,
      (char *)&unk_1682D,
      2557,
      "%s: CPU Mapping is not possible on this allocation!",
      "DevmemAcquireCpuVirtAddr");
    return 131;
  }
  pthread_mutex_lock(*(pthread_mutex_t **)a1[16]);
  v6 = a1[15];
  a1[15] = v6 + 1;
  if ( v6 )
  {
    cpu_va = a1[14];
LABEL_13:
    *a2 = cpu_va;
```

0004C7C0 do_PVRSRVAcquireCPUMapping:1 (4D7C0)

PowerVR GPU: Bypass SELinux



PowerRoot: Root

- Similar to root devices by dirtypipe vulnerability
  - You may load a kernel module (@iGio90) by dirtypipe vulnerability
  - PowerRoot: more powerful than dirtypipe vulnerability

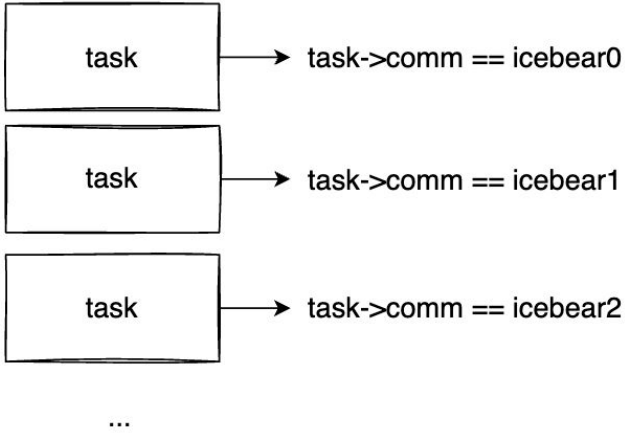
- A lot of methods to root devices by CVE-2021-39815
  - Corrupt page tables
  - Corrupt binaries…
  - Attack kernel in a “memory corruption” way

PowerRoot: Root

- Search `task_struct` by name
  - Find cred by `task_struct->cred`
- Spam files for arb R/W
  - Search file structure by `file->f_cred`
  - E.g. control ashmem `file->private_data`

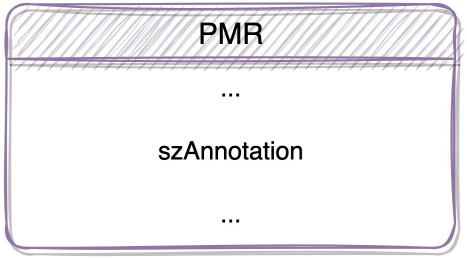
```
static long ashmem_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
  struct ashmem_area *asma = file->private_data;
  long ret = -ENOTTY;

  switch (cmd) {
  ...
  case ASHMEM_SET_PROT_MASK:
    ret = set_prot_mask(asma, arg);
    break;
  case ASHMEM_GET_PROT_MASK:
    ret = asma->prot_mask;
    break;
```



PowerRoot: Root

- Dump kernel image
  - `PMR->szAnnotation` field (`char[]`) specified by user space



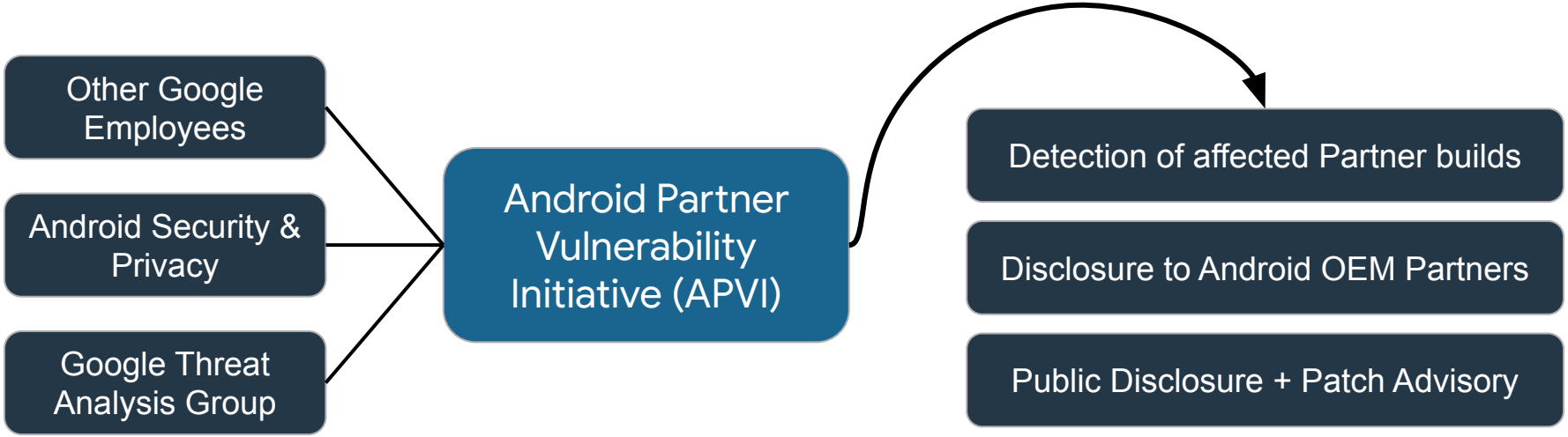
PMR contains .data, .text, .rodata pointers

- Dumping enough kernel data => find where is the possible address of `selinux_enforcing` (e.g. possible signs of SELinux avc structures)

- Overwrite `cred`
  - Write combined page => snoop CPU

Android Partner Vulnerability Initiative (APVI)

- Launched in late 2020 - Google-discovered security issues outside of AOSP code that could potentially affect the security posture of an Android device or its users
- 52 security issues publicly disclosed - https://bugs.chromium.org/p/apvi
- *Any* person or team within Google that discovers a Android partner security issue



Vendor Patch + Disclosure Process

Disclosure to Imagination Technologies

Ensure that ImgTec can remediate security issues + issue patches

Disclosure to Android OEM Partners

Ensure that partners have the ability to implement patches

Public Disclosure on APVI Issue Tracker

Ensure that the public is aware of these issues, to increase transparency

Statement from Imagination Technologies:

"Imagination Technologies supports and is appreciative of initiatives to improve our products. We will continue to engage with Google's APVI program, and others in the security community, to benefit our whole industry."

Driver Developers

- Security review of the driver purpose and design
- Fuzz testing and code review
- Lock down to minimal required access

Researchers

- xPUs introduce CPU/xPU memory-visibility issues
- PowerVR seems under-researched
- Vulnerability research is interesting, fun, and frustrating

Resources

Find a bug that affects a driver in a Pixel device?

- Submit that bug here: https://bughunters.google.com/report

NPU drivers with memory mapping issues

- CVE-2021-1940, CVE-2021-1968, CVE-2021-1969
- CVE-2020-28343, blog

Fuzzing Kernel Drivers with Interface Awareness:

- https://www.blackhat.com/docs/eu-17/materials/eu-17-Corina-Difuzzing-Android-Kernel-Drivers.pdf