窥探Android内核： Crash & Treasure

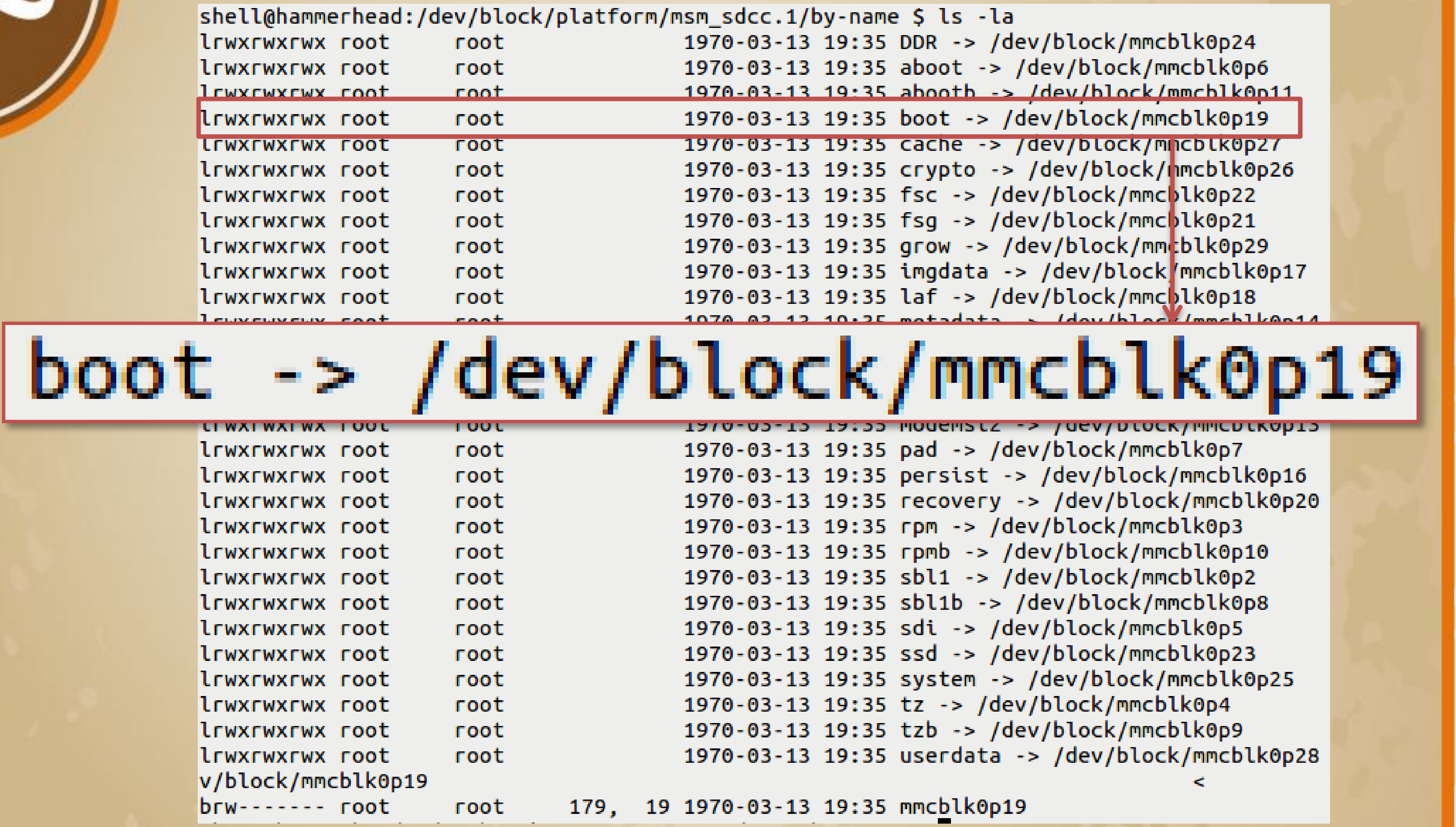
Apr 16,17 2014 Shanghai

{xKungfoo 2014}

Android Kernel ≈ Linux Kernel

- 我们关心的区别:
  - 没有udev
  - Init负责创建 /dev
  - 没有pagefile ☺
- 其他区别:
  - 特别的内存管理（ashmem， pmem）
  - 特别的电源管理（wakelock, alarm）
  - ……

静态分析Android Kernel

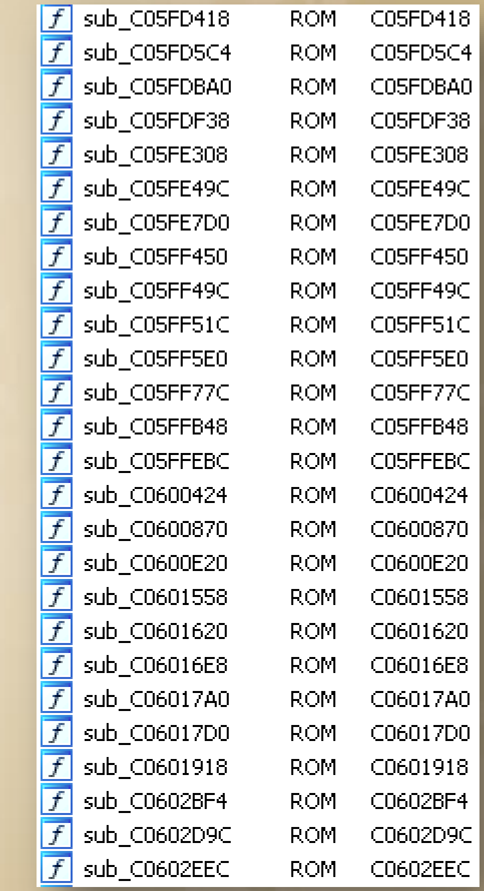


```
shell@hammerhead:/dev/block/platform/msm_sdcc.1/by-name $ ls -la
lrwxrwxrwx root     root              1970-03-13 19:35 DDR -> /dev/block/mmcblk0p24
lrwxrwxrwx root     root              1970-03-13 19:35 aboot -> /dev/block/mmcblk0p6
lrwxrwxrwx root     root              1970-03-13 19:35 abootb -> /dev/block/mmcblk0p11
lrwxrwxrwx root     root              1970-03-13 19:35 boot -> /dev/block/mmcblk0p19
lrwxrwxrwx root     root              1970-03-13 19:35 cache -> /dev/block/mmcblk0p27
lrwxrwxrwx root     root              1970-03-13 19:35 crypto -> /dev/block/mmcblk0p26
lrwxrwxrwx root     root              1970-03-13 19:35 fsc -> /dev/block/mmcblk0p22
lrwxrwxrwx root     root              1970-03-13 19:35 fsg -> /dev/block/mmcblk0p21
lrwxrwxrwx root     root              1970-03-13 19:35 grow -> /dev/block/mmcblk0p29
lrwxrwxrwx root     root              1970-03-13 19:35 imgdata -> /dev/block/mmcblk0p17
lrwxrwxrwx root     root              1970-03-13 19:35 laf -> /dev/block/mmcblk0p18
lrwxrwxrwx root     root              1970-03-13 19:35 metadata -> /dev/block/mmcblk0p14
lrwxrwxrwx root     root              1970-03-13 19:35 modemst2 -> /dev/block/mmcblk0p13
lrwxrwxrwx root     root              1970-03-13 19:35 pad -> /dev/block/mmcblk0p7
lrwxrwxrwx root     root              1970-03-13 19:35 persist -> /dev/block/mmcblk0p16
lrwxrwxrwx root     root              1970-03-13 19:35 recovery -> /dev/block/mmcblk0p20
lrwxrwxrwx root     root              1970-03-13 19:35 rpm -> /dev/block/mmcblk0p3
lrwxrwxrwx root     root              1970-03-13 19:35 rpmb -> /dev/block/mmcblk0p10
lrwxrwxrwx root     root              1970-03-13 19:35 sbl1 -> /dev/block/mmcblk0p2
lrwxrwxrwx root     root              1970-03-13 19:35 sbl1b -> /dev/block/mmcblk0p8
lrwxrwxrwx root     root              1970-03-13 19:35 sdi -> /dev/block/mmcblk0p5
lrwxrwxrwx root     root              1970-03-13 19:35 ssd -> /dev/block/mmcblk0p23
lrwxrwxrwx root     root              1970-03-13 19:35 system -> /dev/block/mmcblk0p25
lrwxrwxrwx root     root              1970-03-13 19:35 tz -> /dev/block/mmcblk0p4
lrwxrwxrwx root     root              1970-03-13 19:35 tzb -> /dev/block/mmcblk0p9
lrwxrwxrwx root     root              1970-03-13 19:35 userdata -> /dev/block/mmcblk0p28
v/block/mmcblk0p19                                                            <
brw------- root     root     179,  19 1970-03-13 19:35 mmcblk0p19
```

boot -> /dev/block/mmcblk0p19

静态分析Android Kernel

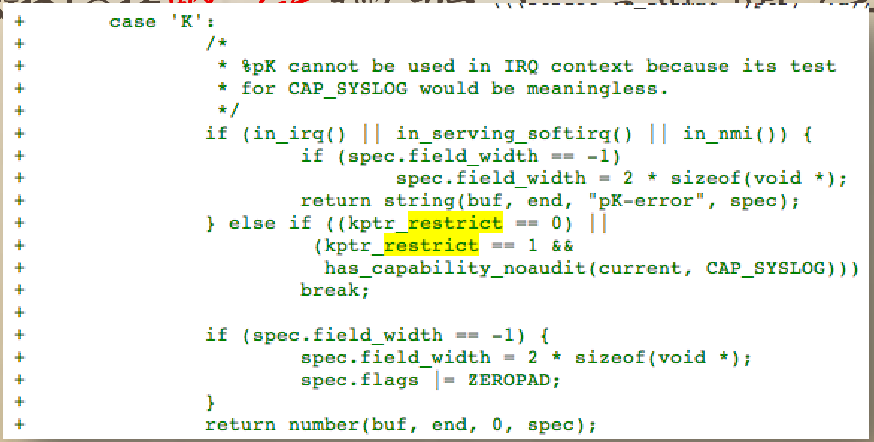
- 不是ELF文件
- 也不存在ELF bundle
  - 区别于iOS kernel
  - 平面结构
- IDA会尝试分析并区分其中的函数
- 效果非常不理想 ➔



sub_C05FD418	ROM	C05FD418
sub_C05FD5C4	ROM	C05FD5C4
sub_C05FDBA0	ROM	C05FDBA0
sub_C05FDF38	ROM	C05FDF38
sub_C05FE308	ROM	C05FE308
sub_C05FE49C	ROM	C05FE49C
sub_C05FE7D0	ROM	C05FE7D0
sub_C05FF450	ROM	C05FF450
sub_C05FF49C	ROM	C05FF49C
sub_C05FF51C	ROM	C05FF51C
sub_C05FF5E0	ROM	C05FF5E0
sub_C05FF77C	ROM	C05FF77C
sub_C05FFB48	ROM	C05FFB48
sub_C05FFEBC	ROM	C05FFEBC
sub_C0600424	ROM	C0600424
sub_C0600870	ROM	C0600870
sub_C0600E20	ROM	C0600E20
sub_C0601558	ROM	C0601558
sub_C0601620	ROM	C0601620
sub_C06016E8	ROM	C06016E8
sub_C06017A0	ROM	C06017A0
sub_C06017D0	ROM	C06017D0
sub_C0601918	ROM	C0601918
sub_C0602BF4	ROM	C0602BF4
sub_C0602D9C	ROM	C0602D9C
sub_C0602EEC	ROM	C0602EEC

静态分析Android Kernel

- /proc/kallsyms可以提供所有的kernel symbol
- 早先的一个patch加入/proc/sys/kernel/kptr_restrict，默认为1，隐藏symbol
- 对exploit略微增加了一些阻碍

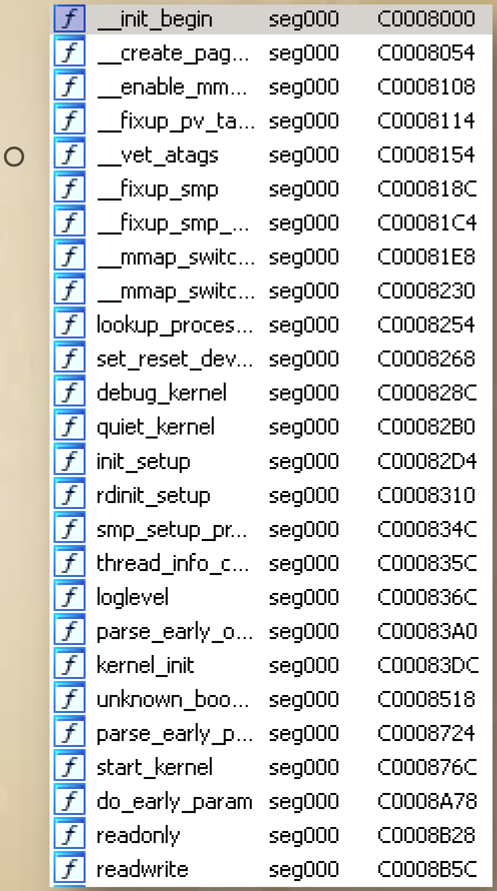


```
+       case 'K':
+               /*
+                * %pK cannot be used in IRQ context because its test
+                * for CAP_SYSLOG would be meaningless.
+                */
+               if (in_irq() || in_serving_softirq() || in_nmi()) {
+                       if (spec.field_width == -1)
+                               spec.field_width = 2 * sizeof(void *);
+                       return string(buf, end, "pK-error", spec);
+               } else if ((kptr_restrict == 0) ||
+                          (kptr_restrict == 1 &&
+                           has_capability_noaudit(current, CAP_SYSLOG)))
+                       break;
+
+               if (spec.field_width == -1) {
+                       spec.field_width = 2 * sizeof(void *);
+                       spec.flags |= ZEROPAD;
+               }
+               return number(buf, end, 0, spec);
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=455cd5ab305c90ffc422dd2e0fb634730942b257

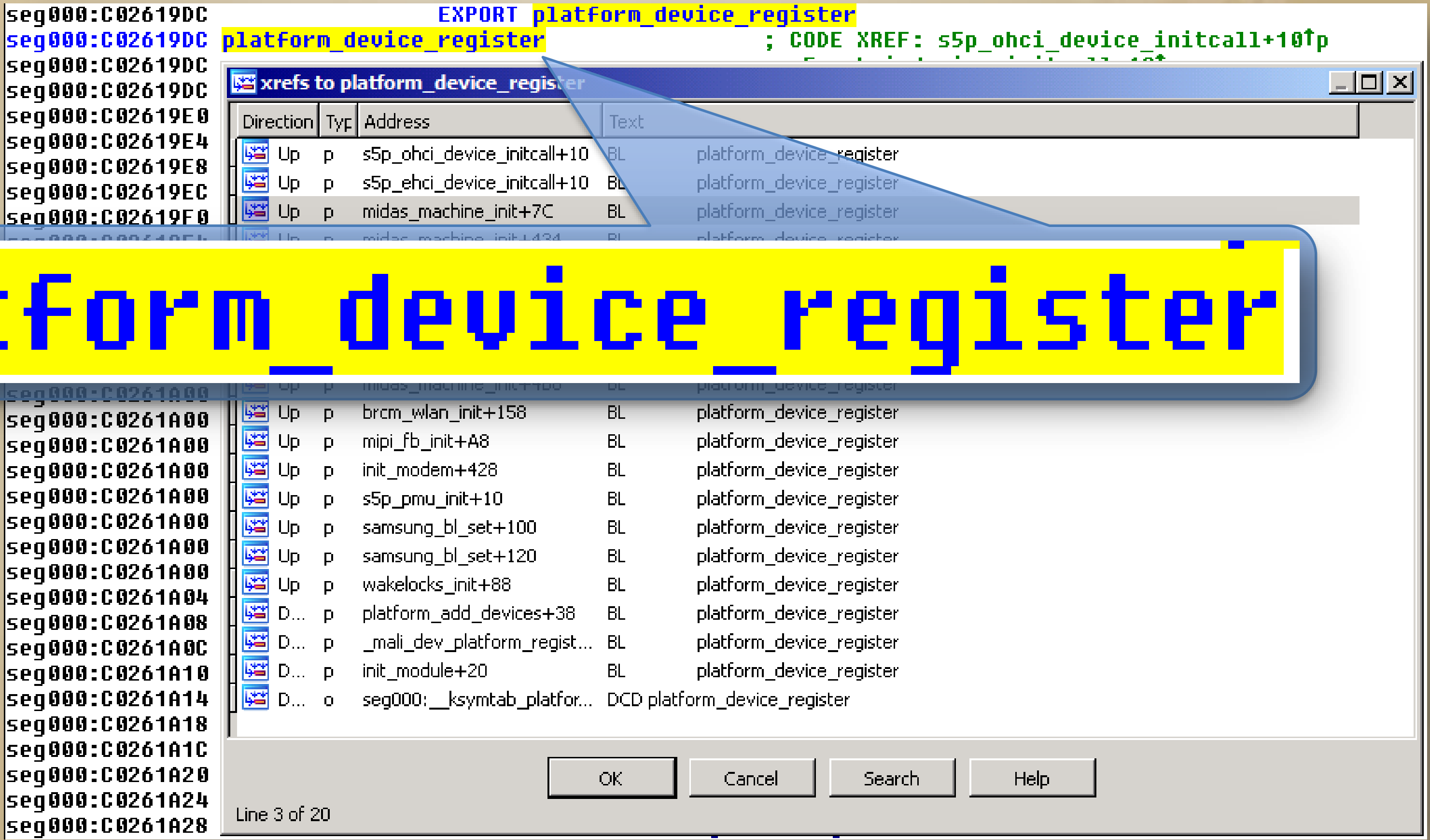
静态分析Android Kernel

- /proc/sys/kernel/kptr_restrict置为0即可正常输出。
- 创建一个IDA loader
- 配合kallsyms输出来创建函数
- 效果非常理想➔



__init_begin	seg000	C0008000
__create_pag...	seg000	C0008054
__enable_mm...	seg000	C0008108
__fixup_pv_ta...	seg000	C0008114
__vet_atags	seg000	C0008154
__fixup_smp	seg000	C000818C
__fixup_smp_...	seg000	C00081C4
__mmap_switc...	seg000	C00081E8
__mmap_switc...	seg000	C0008230
lookup_proces...	seg000	C0008254
set_reset_dev...	seg000	C0008268
debug_kernel	seg000	C000828C
quiet_kernel	seg000	C00082B0
init_setup	seg000	C00082D4
rdinit_setup	seg000	C0008310
smp_setup_pr...	seg000	C000834C
thread_info_c...	seg000	C000835C
loglevel	seg000	C000836C
parse_early_o...	seg000	C00083A0
kernel_init	seg000	C00083DC
unknown_boo...	seg000	C0008518
parse_early_p...	seg000	C0008724
start_kernel	seg000	C000876C
do_early_param	seg000	C0008A78
readonly	seg000	C0008B28
readwrite	seg000	C0008B5C

设备注册



```
seg000:C02619DC                      EXPORT platform_device_register
seg000:C02619DC platform_device_register                 ; CODE XREF: s5p_ohci_device_initcall+10↑p
seg000:C02619DC
seg000:C02619DC
seg000:C02619E0
seg000:C02619E4
seg000:C02619E8
seg000:C02619EC
seg000:C02619F0
seg000:C02619F4
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A00
seg000:C0261A04
seg000:C0261A08
seg000:C0261A0C
seg000:C0261A10
seg000:C0261A14
seg000:C0261A18
seg000:C0261A1C
seg000:C0261A20
seg000:C0261A24
seg000:C0261A28
```

xrefs to platform_device_register

Direction	Typ	Address	Text
Up	p	s5p_ohci_device_initcall+10	BL platform_device_register
Up	p	s5p_ehci_device_initcall+10	BL platform_device_register
Up	p	midas_machine_init+7C	BL platform_device_register
Up	p	midas_machine_init+434	BL platform_device_register
Up	p	midas_machine_init+[illegible]	BL platform_device_register
Up	p	brcm_wlan_init+158	BL platform_device_register
Up	p	mipi_fb_init+A8	BL platform_device_register
Up	p	init_modem+428	BL platform_device_register
Up	p	s5p_pmu_init+10	BL platform_device_register
Up	p	samsung_bl_set+100	BL platform_device_register
Up	p	samsung_bl_set+120	BL platform_device_register
Up	p	wakelocks_init+88	BL platform_device_register
D...	p	platform_add_devices+38	BL platform_device_register
D...	p	_mali_dev_platform_regist...	BL platform_device_register
D...	p	init_module+20	BL platform_device_register
D...	o	seg000:__ksymtab_platfor...	DCD platform_device_register

Line 3 of 20

platform_device_register

Fuzzing设备驱动

- 为什么从驱动入手?
- 已知漏洞
  - mmap() logic issue [Framaroot]
  - memory corruption [Qualcomm MSM]
- Dumb Fuzzing

软柿子

- Android碎片化严重
- 芯片厂商代码良莠不齐
- 驱动难道不是最简单的root方案么?

BROADCOM.
SPREADTRUM
HISILICON
MEDIATEK
QUALCOMM
SAMSUNG
MARVELL
TEXAS INSTRUMENTS

已知漏洞 – mmap边界检查

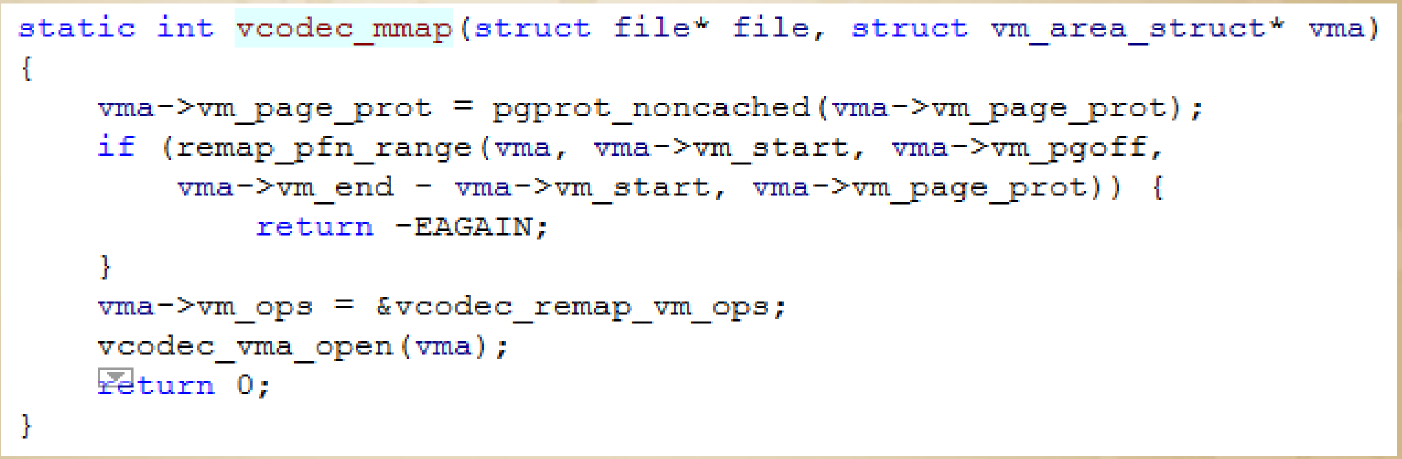
- Framaroot v1.9.1,包含多个可root漏洞，针对以下设备列表.

```
/dev/exynos-mem "Sam"
/dev/DspBridge "Gemli"
/dev/s5p-smem "Merry"
/dev/exynos-mem "Frodo"
/dev/video1 "Aragorn"
/dev/graphics/fb "Legolas"
/dev/msm_camera "Gandalf"
/dev/camera-isp "Boromir"
/dev/memalloc "Pippin"
/dev/amjpegdec "Gollum"
/dev/camera-sysr "Faramir"
/dev/Vcodec "Barahir"
```

下载地址：http://forum.xda-developers.com/showthread.php?t=2130276

已知漏洞 – mmap边界检查

- 以/dev/Vcodec为例，mmap具有读写权限且没有边界检查，导致用户态可以任意地址读写内核数据.

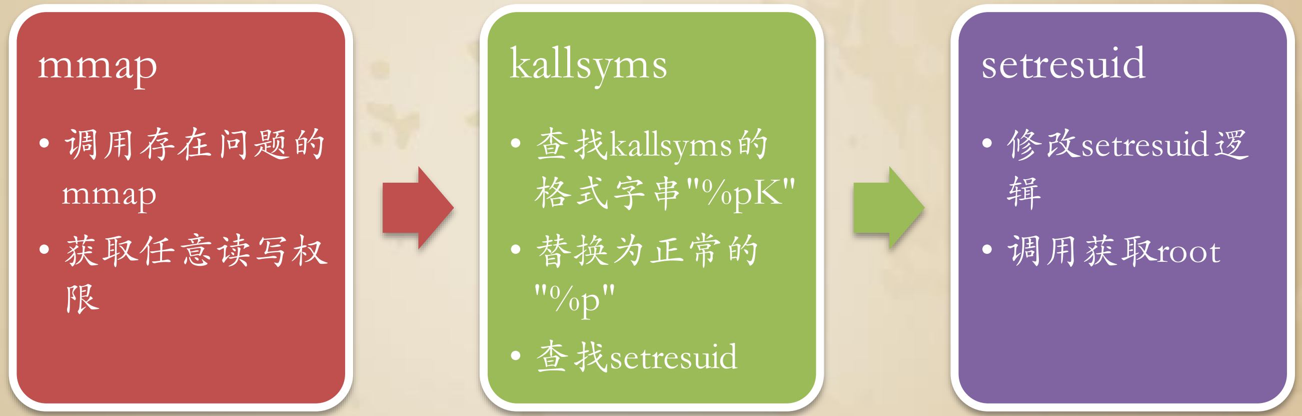


```
static int vcodec_mmap(struct file* file, struct vm_area_struct* vma)
{
    vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);
    if (remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff,
        vma->vm_end - vma->vm_start, vma->vm_page_prot)) {
            return -EAGAIN;
    }
    vma->vm_ops = &vcodec_remap_vm_ops;
    vcodec_vma_open(vma);
    return 0;
}
```

/mediatek/platform/mt6582/kernel/drivers/videocodec/videocodec_kernel_driver.c

已知漏洞 – mmap边界检查

- Root流程



mmap

- 调用存在问题的mmap
- 获取任意读写权限

kallsyms

- 查找kallsyms的格式字串"%pK"
- 替换为正常的"%p"
- 查找setresuid

setresuid

- 修改setresuid逻辑
- 调用获取root

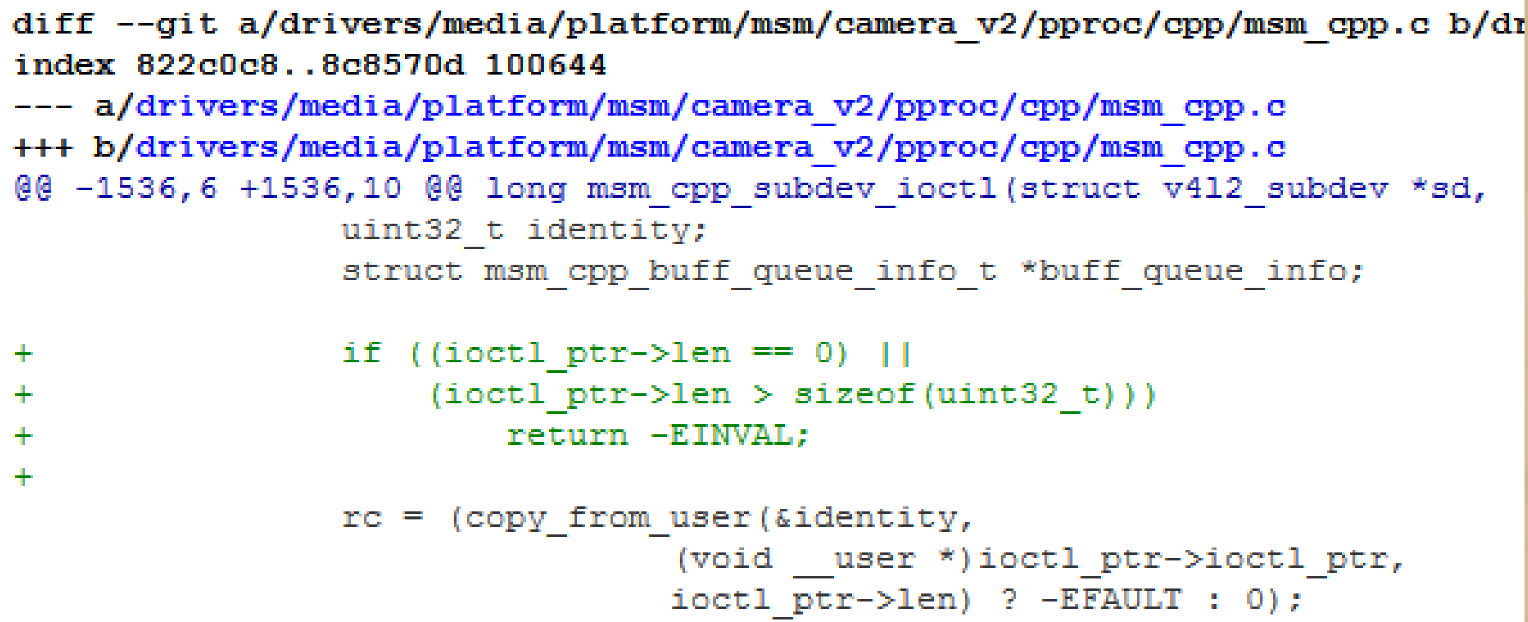
已知漏洞 – 内存破坏

- Qualcomm MSM代码存在多个内存破坏漏洞
  - CVE-2013-2596
  - CVE-2013-2597
  - CVE-2013-4738
  - CVE-2013-4739
  - CVE-2013-6123
  - …

已知漏洞 – 内存破坏

- CVE-2013-4738

  栈上的四字节变量被覆盖为超长数据，导致栈溢出。



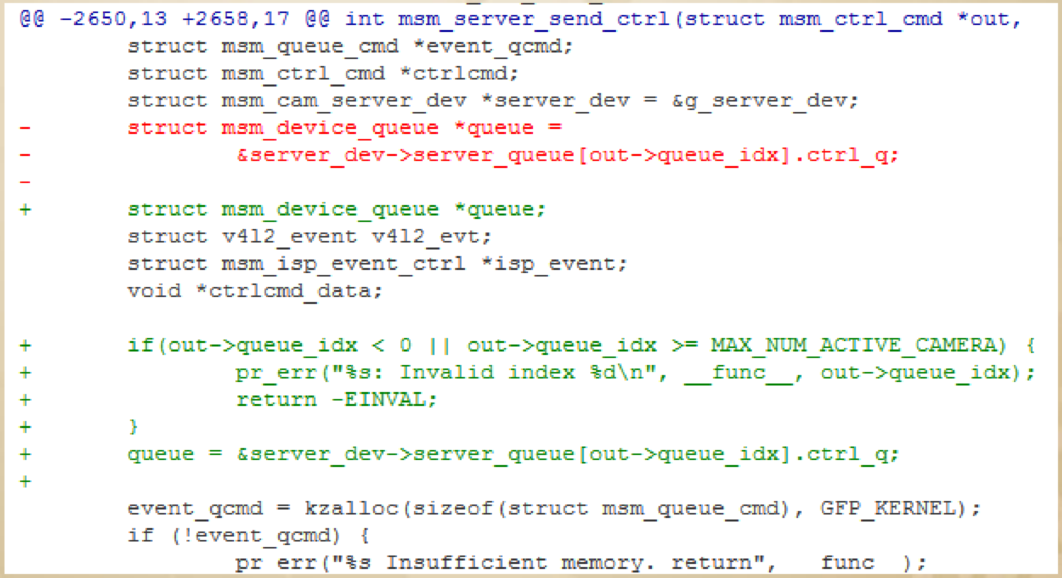
```
diff --git a/drivers/media/platform/msm/camera_v2/pproc/cpp/msm_cpp.c b/dr
index 822c0c8..8c8570d 100644
--- a/drivers/media/platform/msm/camera_v2/pproc/cpp/msm_cpp.c
+++ b/drivers/media/platform/msm/camera_v2/pproc/cpp/msm_cpp.c
@@ -1536,6 +1536,10 @@ long msm_cpp_subdev_ioctl(struct v4l2_subdev *sd,
                uint32_t identity;
                struct msm_cpp_buff_queue_info_t *buff_queue_info;

+               if ((ioctl_ptr->len == 0) ||
+                   (ioctl_ptr->len > sizeof(uint32_t)))
+                       return -EINVAL;
+
                rc = (copy_from_user(&identity,
                                (void __user *)ioctl_ptr->ioctl_ptr,
                                ioctl_ptr->len) ? -EFAULT : 0);
```

已知漏洞 – 内存破坏

- CVE-2013-6123

  读写地址均可由用户态传入数据指定，导致任意地址写任意数据。



```
@@ -2650,13 +2658,17 @@ int msm_server_send_ctrl(struct msm_ctrl_cmd *out,
        struct msm_queue_cmd *event_qcmd;
        struct msm_ctrl_cmd *ctrlcmd;
        struct msm_cam_server_dev *server_dev = &g_server_dev;
-       struct msm_device_queue *queue =
-               &server_dev->server_queue[out->queue_idx].ctrl_q;
-
+       struct msm_device_queue *queue;
        struct v4l2_event v4l2_evt;
        struct msm_isp_event_ctrl *isp_event;
        void *ctrlcmd_data;

+       if(out->queue_idx < 0 || out->queue_idx >= MAX_NUM_ACTIVE_CAMERA) {
+               pr_err("%s: Invalid index %d\n", __func__, out->queue_idx);
+               return -EINVAL;
+       }
+       queue = &server_dev->server_queue[out->queue_idx].ctrl_q;
+
        event_qcmd = kzalloc(sizeof(struct msm_queue_cmd), GFP_KERNEL);
        if (!event_qcmd) {
                pr_err("%s Insufficient memory. return", __func__);
```

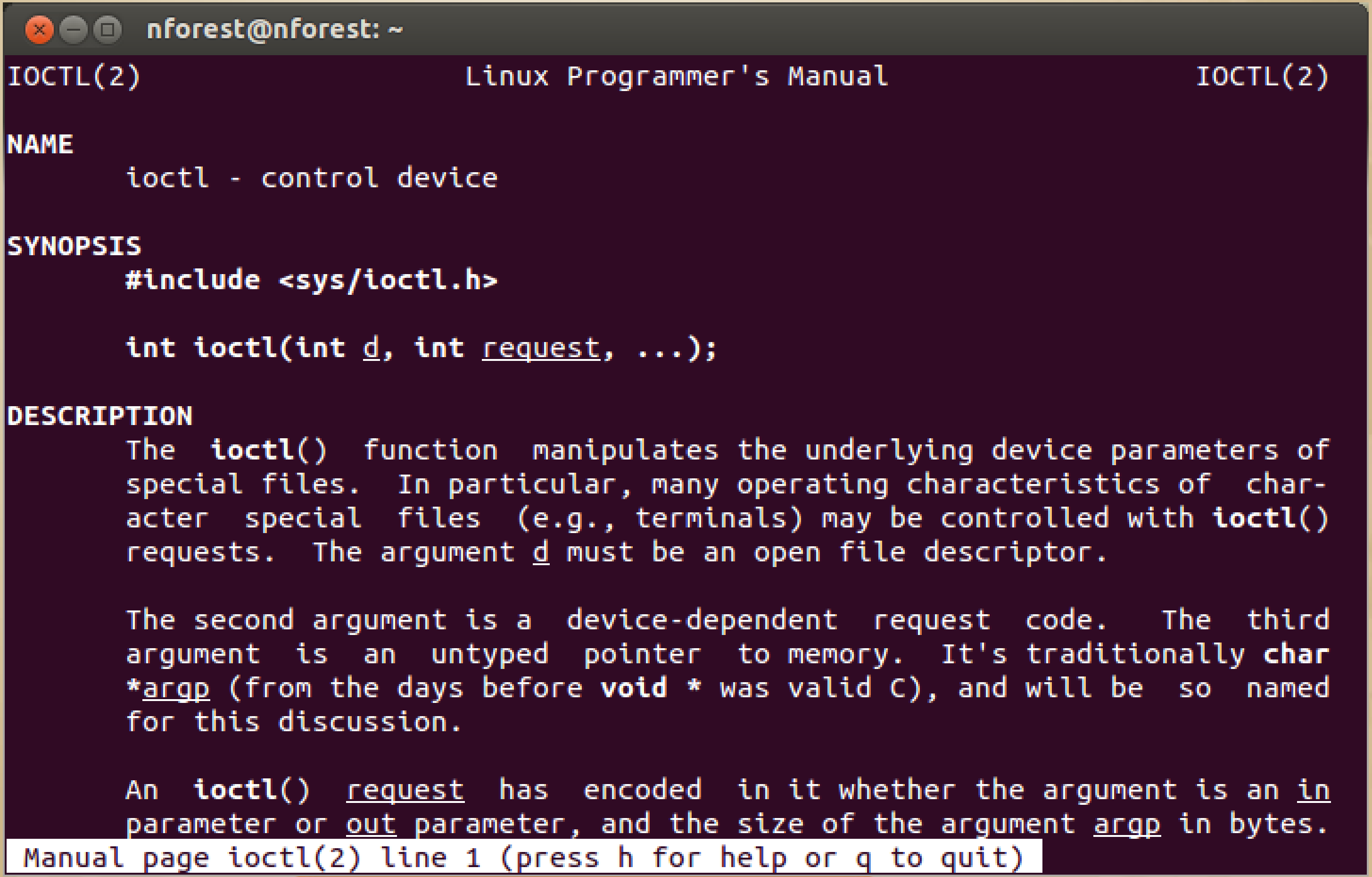
Dumb Fuzzing

- 相比iOS，构造更简洁
- 三个API：

```
1. ioctl (fd, cmd, arg)
2. copy_from_user(*to, *from, length)
3. copy_to_user(*to, *from, length)
```

Dumb Fuzzing

int ioctl(int fd, int cmd, ...)



```
IOCTL(2)                  Linux Programmer's Manual                  IOCTL(2)

NAME
       ioctl - control device

SYNOPSIS
       #include <sys/ioctl.h>

       int ioctl(int d, int request, ...);

DESCRIPTION
       The  ioctl()  function  manipulates the underlying device parameters of
       special files.  In particular, many operating characteristics of  char-
       acter  special  files  (e.g., terminals) may be controlled with ioctl()
       requests.  The argument d must be an open file descriptor.

       The second argument is a  device-dependent  request  code.   The  third
       argument  is  an  untyped  pointer  to memory.  It's traditionally char
       *argp (from the days before void * was valid C), and will be  so  named
       for this discussion.

       An  ioctl()  request  has  encoded  in it whether the argument is an in
       parameter or out parameter, and the size of the argument argp in bytes.
 Manual page ioctl(2) line 1 (press h for help or q to quit)
```

Dumb Fuzzing

- copy_from_user() & copy_to_user()
- 无须处理page fault ☺

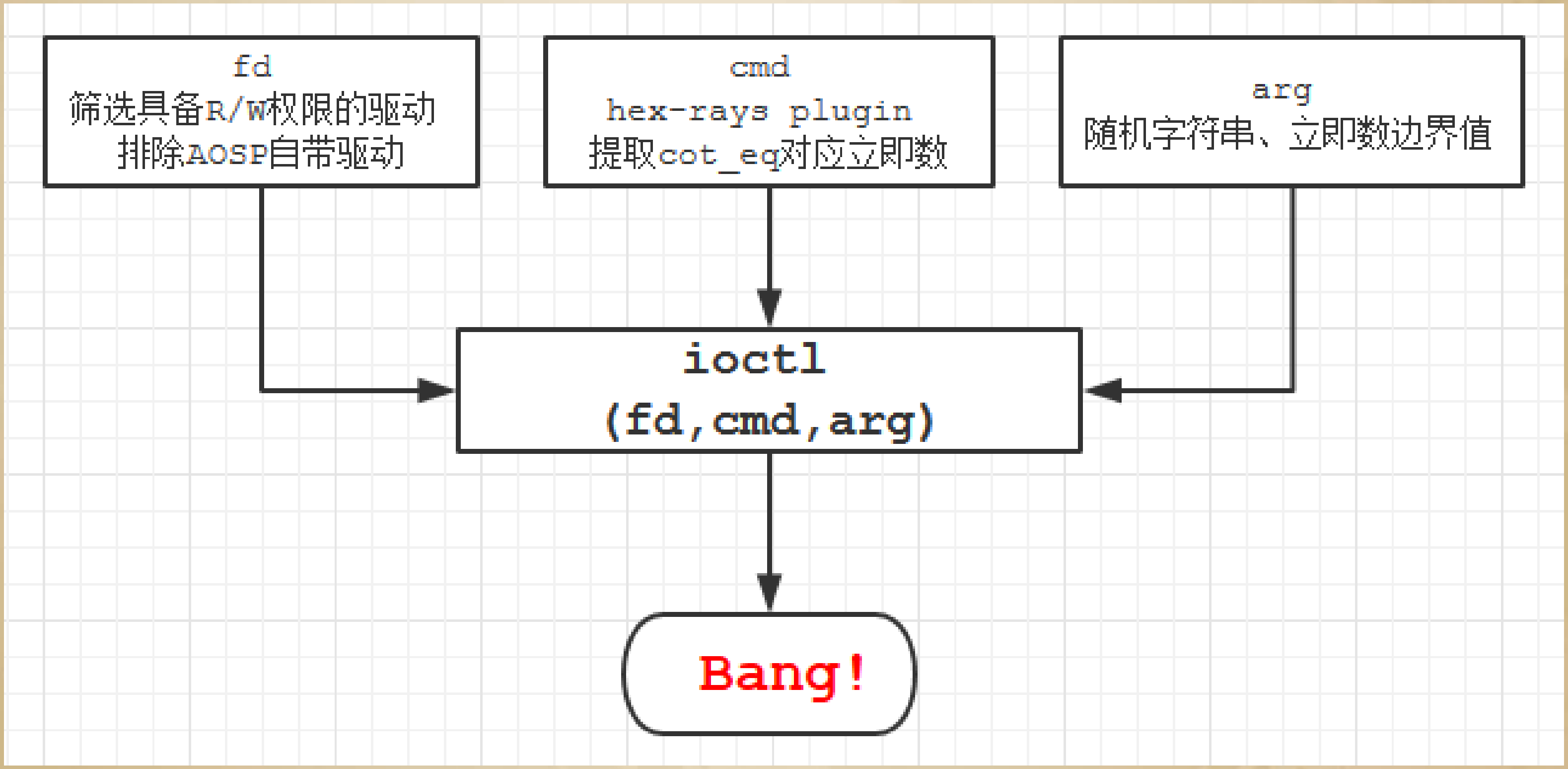
```
static inline unsigned long __must_check copy_from_user(void *to, const void __user *from,
unsigned long n)
{
        if (access_ok(VERIFY_READ, from, n))
                n = __copy_from_user(to, from, n);
        return n;
}

static inline unsigned long __must_check copy_to_user(void __user *to, const void *from,
unsigned long n)
{
        if (access_ok(VERIFY_WRITE, to, n))
                n = __copy_to_user(to, from, n);
        return n;
}

#define __copy_from_user(to,from,n)      (memcpy(to, (void __force *)from, n), 0)
#define __copy_to_user(to,from,n)        (memcpy((void __force *)to, from, n), 0)
```

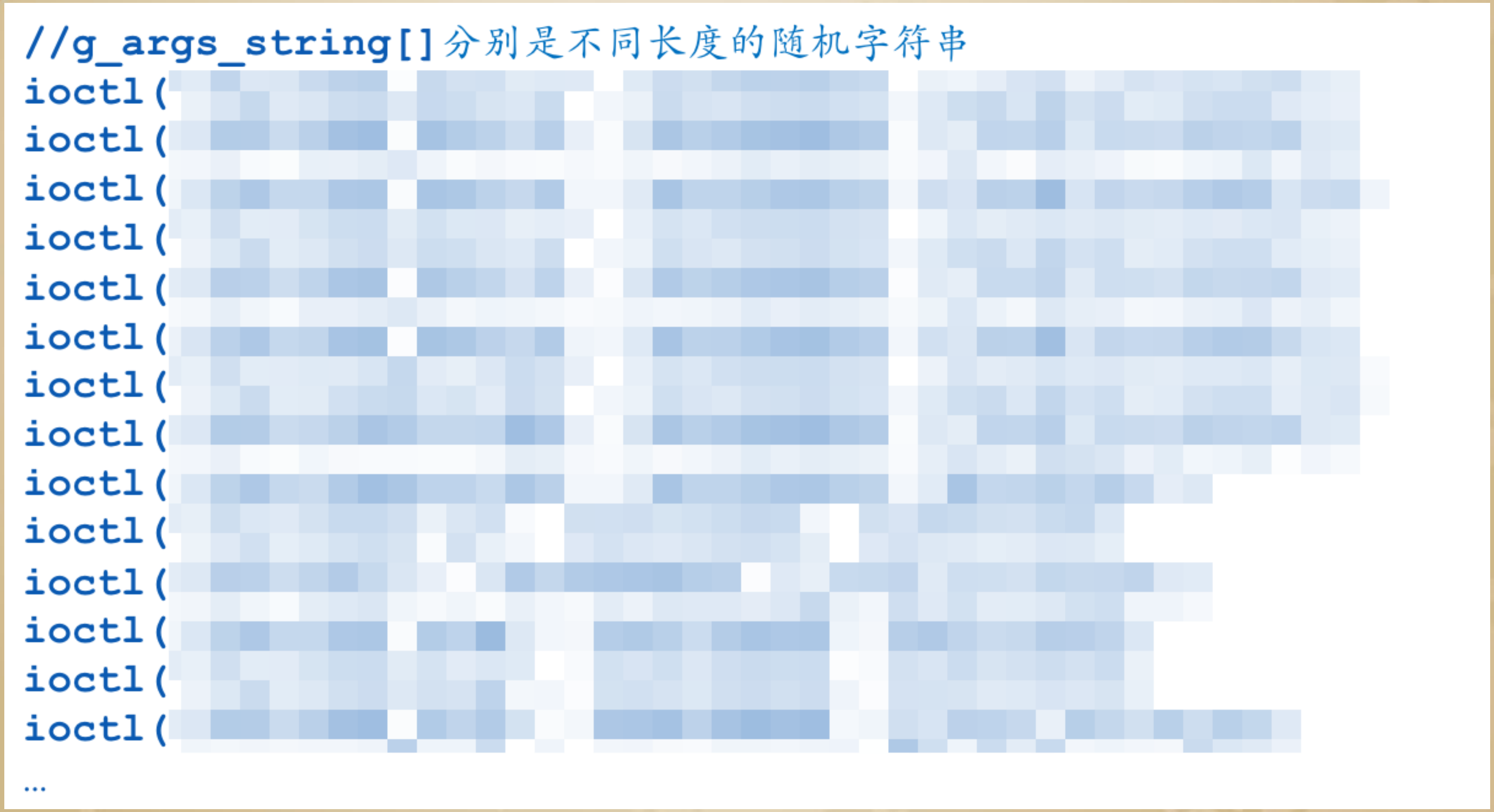
Dumb Fuzzing

- Dumb Fuzzer实现流程



Crash，更多的Crash

- Crash太多以至于我们不知道该怎么办



```
//g_args_string[]分别是不同长度的随机字符串
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
ioctl([redacted]
…
```

Dumb Fuzzing的问题

- Android内核难以调试
  - last_kmsg
- Crash过多，反而影响测试效率
- 大量Crash由Pointer Dereference造成，可用性较低。

Dumb Fuzzing的问题

"懒惰是科技发展的原动力"

为了减少人工分析成本:

- 更精确的识别ioctl cmd
- 尽可能的还原ioctl中arg的数据类型
- 确定cmd和arg的对应关系

HexRaysCodeXplorer

- HexRaysCodeXplorer
  - 基于Hex-Rays SDK实现，其类型重建功能可以依据代码中对于指针的引用情况自动生成对应的结构体类型.
  - 原作者开发此插件用于分析Win32/Gapz Bootkit[REcon'13、 ZeroNights'13]

http://rehints.com/2013-09-02-Type-REconstruction-in-HexRaysCodeXplorer.html

HexRaysCodeXplorer

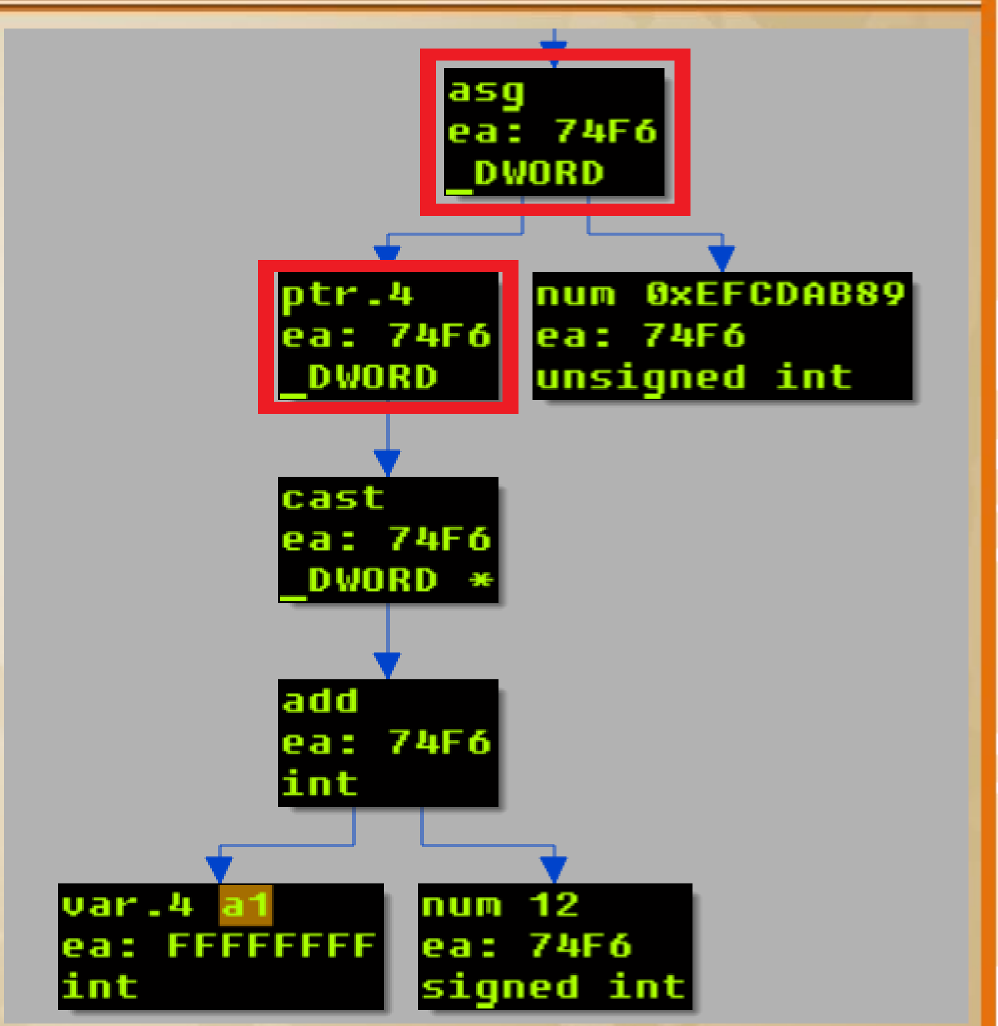
- 为恶意代码分析设计，有其不完善的地方
- 如何改进它的输出？
  - 利用Hex Rays输出的一些特性，“模糊处理”
  - 继而改进HexRaysCodeXplorer的输出

Hex-Rays SDK

- Hex-Rays SDK简要介绍
  - 函数在反编译过程中，Hex-Rays内部维护了一个ctree结构，针对此结构的遍历和修改提供了一系列数据结构和API供插件开发者使用。
  - ctree的每个节点是citem_t结构，该结构体包含一个ctype_t的字段,指示当前item的类型。

Hex-Rays SDK

```
*(DWORD *)(a1 + 12) =
0xEFCDAB89;
```



```
asg
ea: 74F6
_DWORD
```

```
ptr.4
ea: 74F6
_DWORD
```

```
num 0xEFCDAB89
ea: 74F6
unsigned int
```

```
cast
ea: 74F6
_DWORD *
```

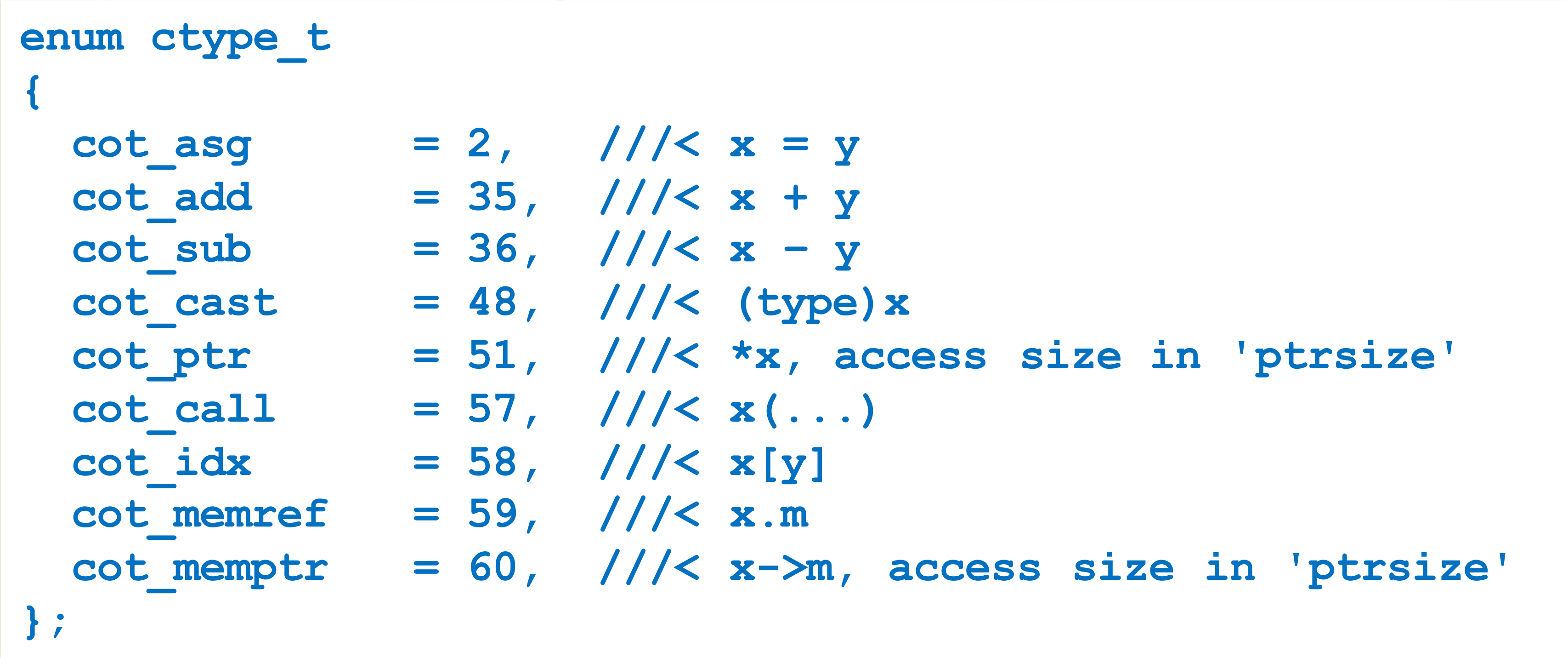
```
add
ea: 74F6
int
```

```
var.4 a1
ea: FFFFFFFF
int
```

```
num 12
ea: 74F6
signed int
```

Hex-Rays SDK

- citem_t类型有80+种
- 类型重建中可能用到的citem_t类型有



```
enum ctype_t
{
  cot_asg       = 2,   ///< x = y
  cot_add       = 35,  ///< x + y
  cot_sub       = 36,  ///< x - y
  cot_cast      = 48,  ///< (type)x
  cot_ptr       = 51,  ///< *x, access size in 'ptrsize'
  cot_call      = 57,  ///< x(...)
  cot_idx       = 58,  ///< x[y]
  cot_memref    = 59,  ///< x.m
  cot_memptr    = 60,  ///< x->m, access size in 'ptrsize'
};
```

HexRaysCodeXplorer的不足

- 存在的问题
  - 没有考虑变量依赖关系
  - 没有充分利用类型转换信息
  - 没有函数间的类型重建能力

改进方案

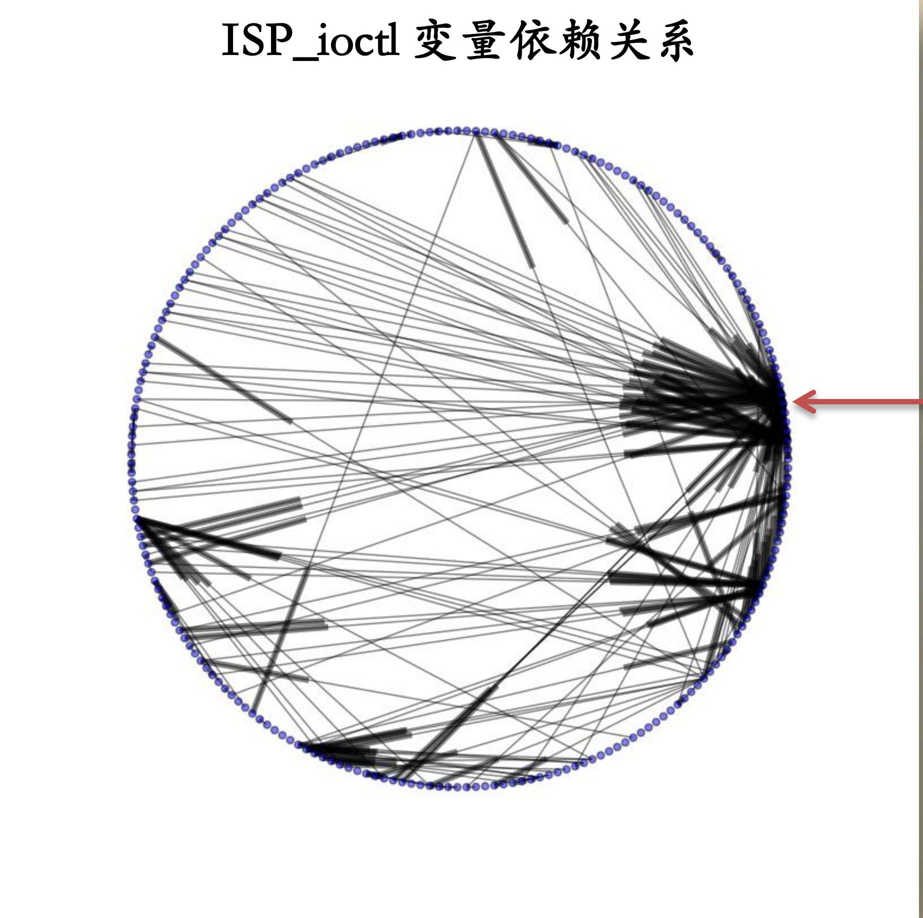
- 问题1：没有考虑变量依赖关系

```
v6 = arg;
v8 = _copy_from_user((int)&v209, (void *)v6, 4);
if ( !v8 )
{
    _xlog_printk((int)off_C002EFCC, (int)((char
*)off_C002EFCC - 728), v209, v79);
    clkmux_sel(1u, v209, (int)off_C002EFD0, v80);
    return v8;
}
```

改进方案

- 解决方案：处理变量依赖
  1. 对待分析变量的所有赋值操作进行处理
  2. 被赋值变量和待分析变量纳入同一个集合
  3. 对该集合里所有变量同时进行类型重建，且所有结果映射到原待分析变量

改进方案



ISP_ioctl 变量依赖关系

输入参数

改进方案

- 问题2：没有充分利用类型转换信息
  - HexRaysCodeXplorer不会利用该表达式获取信息
  - 但根据该表达式可以分析出v4指向的缓冲区长度大于等于16,且(v4+16)对应一个指针变量。

```
v69 = (DWORD*)((char *)v4 + 16);
```

改进方案

- 解决方案：充分利用类型转换信息
  1. 处理cot_add、cot_sub，获取变量的长度重建信息
  2. 处理cot_cast，获取变量的类型重建信息

```
case cot_add:
case cot_sub:
{
    if (expr->y->op == cot_num)
    {
      field.offset = expr->op==cot_add ?
      int32(expr->y->numval()) : 0-int32(expr->y->numval());
    }
}
```

改进方案

- 问题3：无函数间的类型重建能力

```
v4 = arg;
v154 = _copy_from_user((int)&v232, (void *)v4, 44);
if ( !v154 )
{
    m4u_query_mva(v232, v233, v234, (int)&v235, v3);
    //todo...
}
```

改进方案

- 解决方案：函数间类型重建
  1. cot_call可以看作变量依赖的特殊情况，即待分析变量和子函数参数的依赖关系
  2. 对于原函数中的每一个cot_call，判断其参数是否包含待分析变量，如果包含则建立映射关系
  3. 对子函数进行分析，若为特定函数如memcpy、memzero、copy_from_user，则特殊处理，否则4
  4. 对子函数进行反编译，递归的进行类型重建操作

改进方案

- cot_call处理
  - 从原函数分析获得的变量信息为实参
  - 对子函数反编译得到的是形参
  - 如果实参为待分析变量，则对应的子函数形参为待分析变量(一种较为特殊的变量依赖关系)

改进方案

- 特殊函数处理，以memcpy为例

```
Field field = {0};
if (strcmp(s,"memcpy") == 0)
{
  carg_t arg = arglist->at(2);
  if (arg.op == cot_num)
  {
    field.offset = 0;
    field.typesize = 1;
    field.count = int32(arg.numval());
  }
  cfield->fields.insert(field);
}
```

谢谢！

Q & A