CVE-2014-3153漏洞分析与利用

一次内核漏洞利用实战 -- lieanu

0x01 漏洞概述

- CVE-2014-3153
- 本地提权漏洞
- 最早在2014年5月份由comex(Pinkie Pie)曝出[1]，影响内核版本3.14.5之前
- GeoHot(tomcr00se)很快拿出利用代码，towelroot[2]
- 漏洞代码位置在kernel/futex.c

[1] https://hackerone.com/reports/13388

[2] https://towelroot.com/

0x02 调试环境准备

- 内核调试方法[1]：kdump/systemtap/lttng
- kdump：捕获系统崩溃现场
- systemtap：动态监控/跟踪linux内核，需要编写脚本，语法类C
- lttng：开源的内核/库/应用的跟踪框架(vs. strace)
- gdb + qemu：双机联调，单步跟踪
- 该应用场景下，gdb+qemu的方法更有效

[1] https://wiki.ubuntu.com/Kernel/Debugging

0x02 调试环境准备

- gdb + qemu + ubuntu14.04环境搭建
- Tips：

  cpu支持kvm的话，打开-enable-kvm

  加上-vga std,可以将系统崩溃信息打到屏幕上

  -s 在本地1234port,监听gdb连接

- `$ qemu-system-i386 -enable-kvm -s -vga std -hda ubuntu140432.img`

0x02 调试环境准备

- 符号表与源码准备
- 源码推荐ubuntu官方git版：
  git://kernel.ubuntu.com/ubuntu/ubuntu-trusty.git
- 符号表：

```
echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main restricted universe multiverse
deb http://ddebs.ubuntu.com $(lsb_release -cs)-updates main restricted universe multiverse
deb http://ddebs.ubuntu.com $(lsb_release -cs)-security main restricted universe multiverse
deb http://ddebs.ubuntu.com $(lsb_release -cs)-proposed main restricted universe multiverse
" | sudo tee -a /etc/apt/sources.list.d/ddebs.list
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 428D7C01
sudo apt-get update
sudo apt-get install linux-image-$(uname -r)-dbgsym
```

- gdb中file path/to/vmlinux导入符号表。文件太大，ida打开需要时间

0x03 漏洞原理

- futex(Fast Userspace mutex)
- 其设计目的是加速glibc层的互斥访问速度
- Glibc库的实现：

  pthread_mutex_lock()

  pthread_mutex_unlock()

- 我们通过调用syscall(...)来触发系统调用

  如gettid()是个系统调用，glibc没有wrap，那如何使用？

  syscall(__NR_gettid)

0x03 漏洞原理

- 使用syscall调用futex相关函数
- 如futex_lock_pi():

  ```
  syscall(__NR_futex, &uaddr2, FUTEX_LOCK_PI, 1, 0, NULL, 0)
  ```

- 其它：

  FUTEX_WAIT_REQUEUE_PI

  FUTEX_CMP_REQUEUE_PI

  ...

- 其接口是do_futex

0x03 漏洞原理

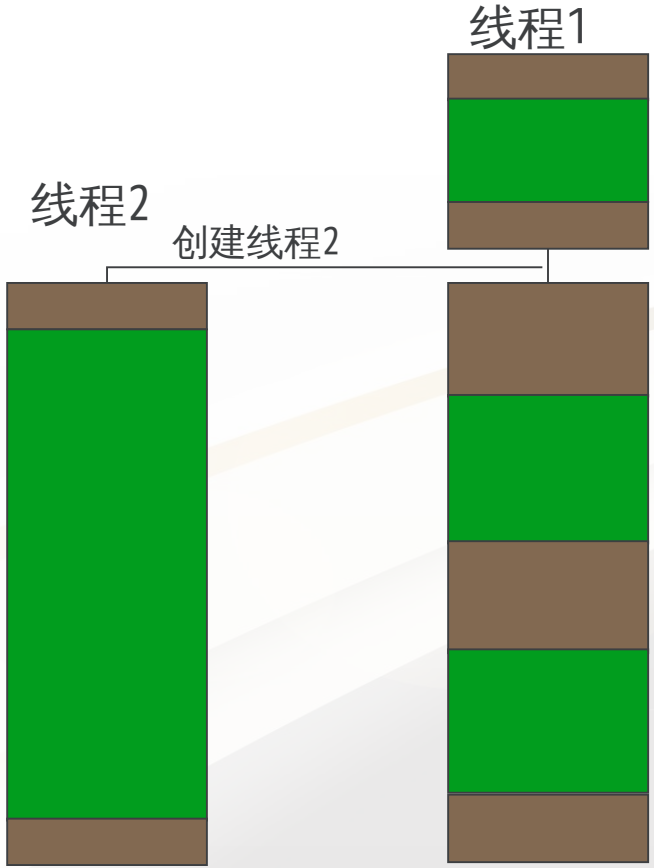
- Relock问题，考虑下面一段代码：

```
8  #define futex_lock_pi(mutex) syscall(__NR_futex, mutex, FUTEX_LOCK_PI, 1, 0, NULL, 0)
9  int mutex = 0;
10
11 void *thread(void *arg)
12 {
13     futex_lock_pi(&mutex);
14 }
15 int main(int argc, char *argv[])
16 {
17     pthread_t t;
18     futex_lock_pi(&mutex);
19     mutex = 0;
20     pthread_create(&t, NULL, thread, NULL);
21     return 0;
22 }
```

- Demo（testcode.c）

0x03 漏洞原理

- requeue漏洞:



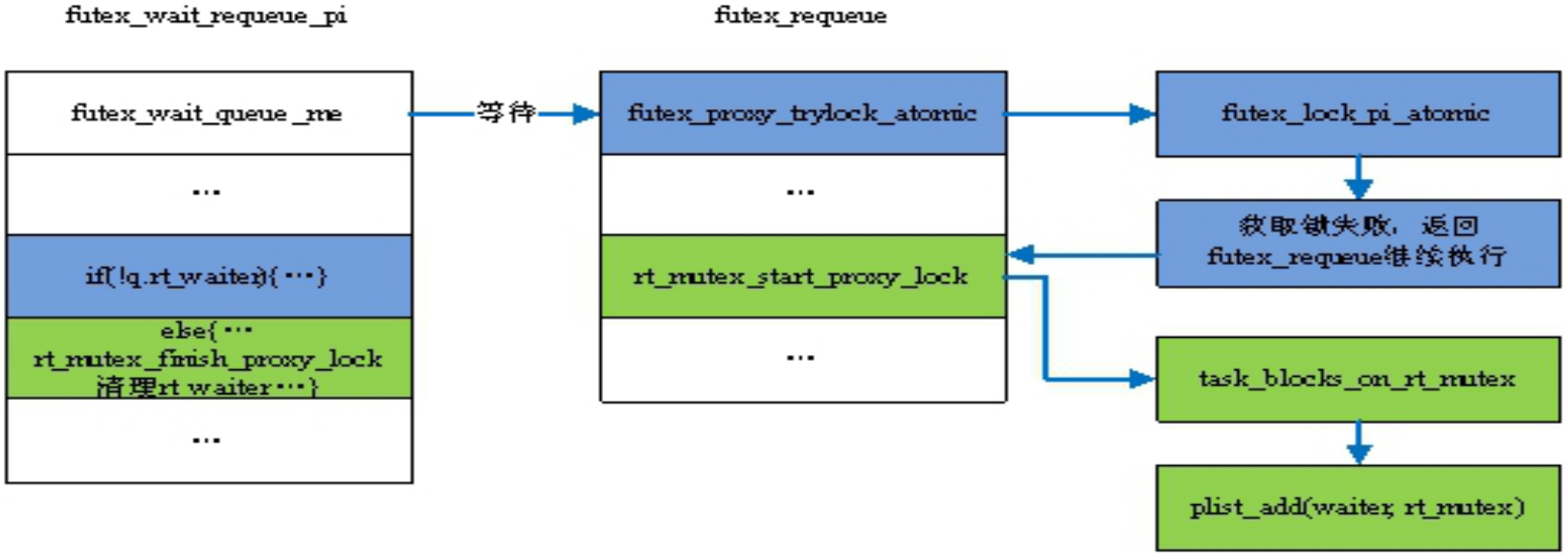
[2] futex_wait_requeue_pi (uaddr1, uaddr2)

栈上的rt_waiter = NULL
线程退出时回收链表,
造成崩溃

Demo(exp-dos.c)

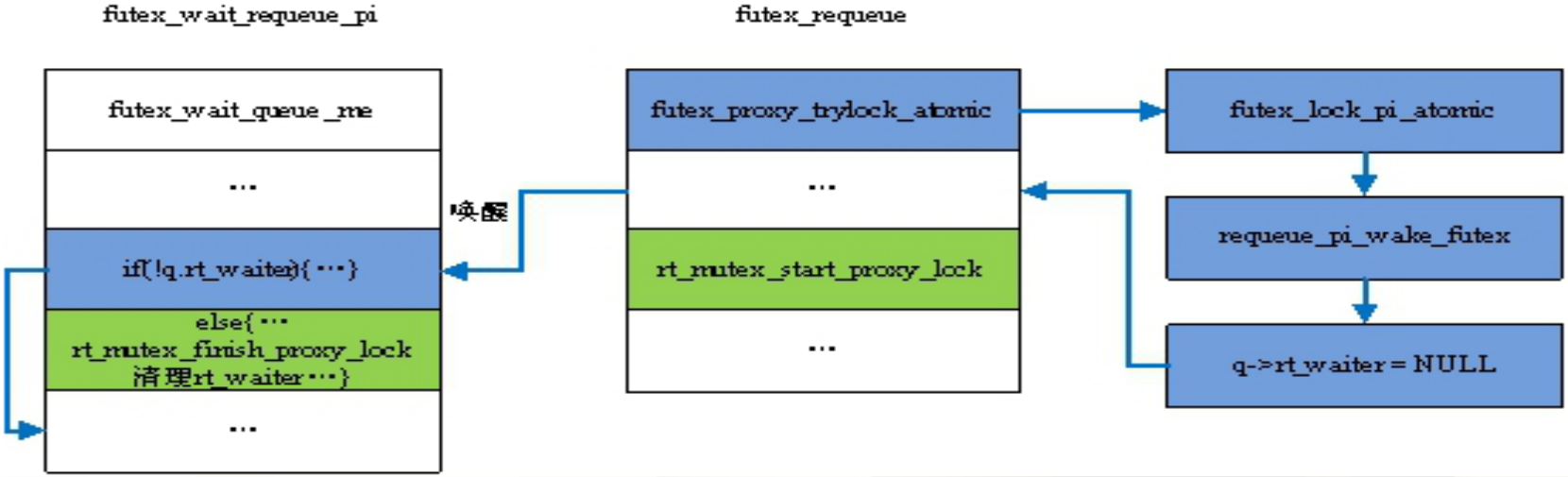
0x03 漏洞原理

- 细节：


futex_wait_requeue_pi
futex_requeue
futex_wait_queue_me
...
if(!q.rt_waiter){...}
else{...
rt_mutex_finish_proxy_lock
清理rt_waiter...}
...
等待
futex_proxy_trylock_atomic
...
rt_mutex_start_proxy_lock
...
futex_lock_pi_atomic
获取锁失败，返回
futex_requeue继续执行
task_blocks_on_rt_mutex
plist_add(waiter, rt_mutex)

0x03 漏洞原理

- 细节：


futex_wait_requeue_pi
futex_requeue
futex_wait_queue_me
...
if(!q.rt_waiter){...}
else{...
rt_mutex_finish_proxy_lock
清理rt_waiter...}
...
唤醒
futex_proxy_trylock_atomic
...
rt_mutex_start_proxy_lock
...
futex_lock_pi_atomic
requeue_pi_wake_futex
q->rt_waiter = NULL

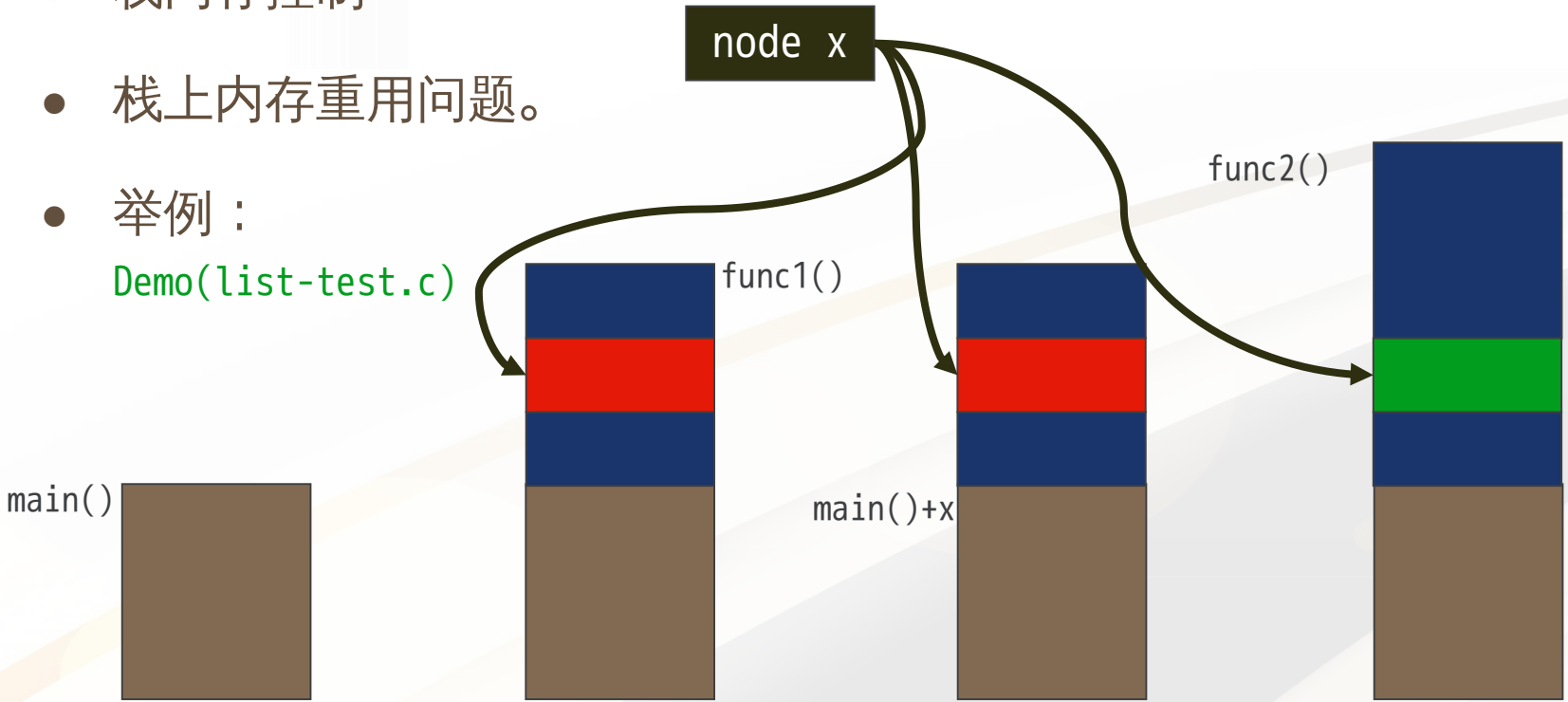
0x03 漏洞原理

- relock + requeue，两个漏洞的组合构造出这个cve的形成机理
- 目前，能做到对内核进行崩溃
- 重点是栈上的rt_waiter可否再利用？
- 接下来，我们看栈内存控制问题。。。

0x04 漏洞利用

- 栈内存控制
- 栈上内存重用问题。
- 举例：

  Demo(list-test.c)



0x04 漏洞利用

- 内核的线程栈如何重用？
- futex_wait_requeue_pi()的栈深度有216
- 需要不小于216深度的系统调用？rt_waiter不是最后一个元素，不需要这么深。

```
» objdump -d vmlinux | ./scripts/checkstack.pl i386
0xc104064e identity_mapped [vmlinux]:                    4096
0xc1188bb6 do_sys_poll [vmlinux]:                        928
0xc14cc4c6 xhci_reserve_bandwidth [vmlinux]:             828
.....
0xc155ad46 __sys_sendmmsg [vmlinux]:                     184
```

0x04 漏洞利用

- rt_waiter使用的数据结构
- linux kernel 3.14之前是plist， 之后是rbtree.

```
rt_waiter = {
    list_entry = {
        prio = 0x82,
        prio_list = {
            next = 0xc10788f2 ,
            prev = 0xc78840b0
        },
        node_list = {
            next = 0xc09c0cf0,
            prev = 0x2d081fc1
        }
    },
...
```

0x04 漏洞利用

- prio_list & node_list
- prio_list优先级和链表已知节点不相同，加入链表
- 利用prio_list，达到任意地址写的目的

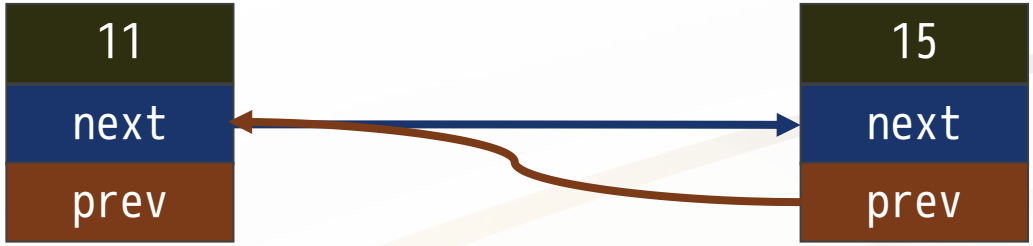
  Step 0x01: 在用户态空间申请内存，留做新的节点。

  Step 0x02: 利用栈内存重写，将rt_waiter替换为，在用户态空间伪造好的

  Step 0x03: 利用优先级，在链表指定位置插入新节点功能，达到任意地址写的目的

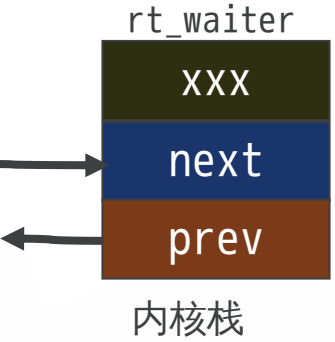
0x04 漏洞利用

- 申请用户态节点，指定地址mmap()，做如下初始化：



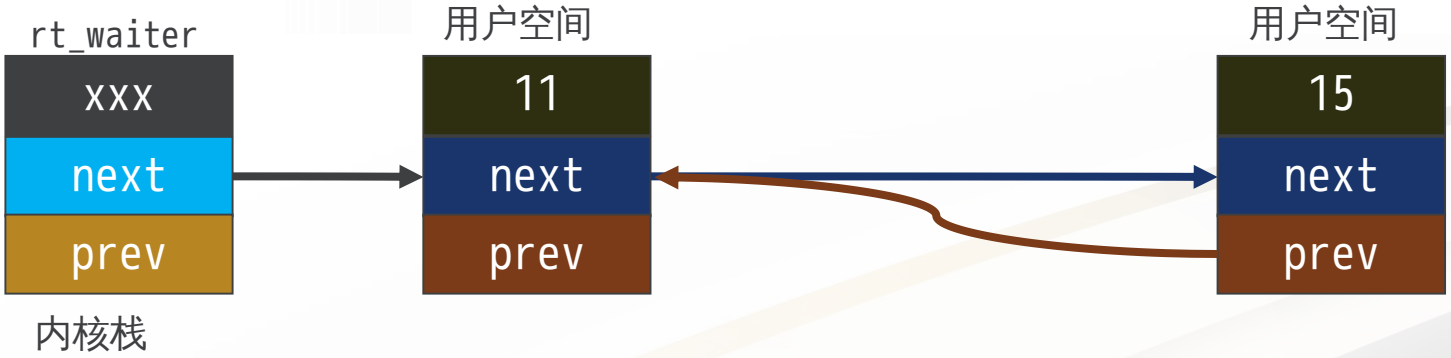
0x04 漏洞利用

- 利用sendmmsg()，覆盖栈上rt_waiter，达到追加新结点的目的。



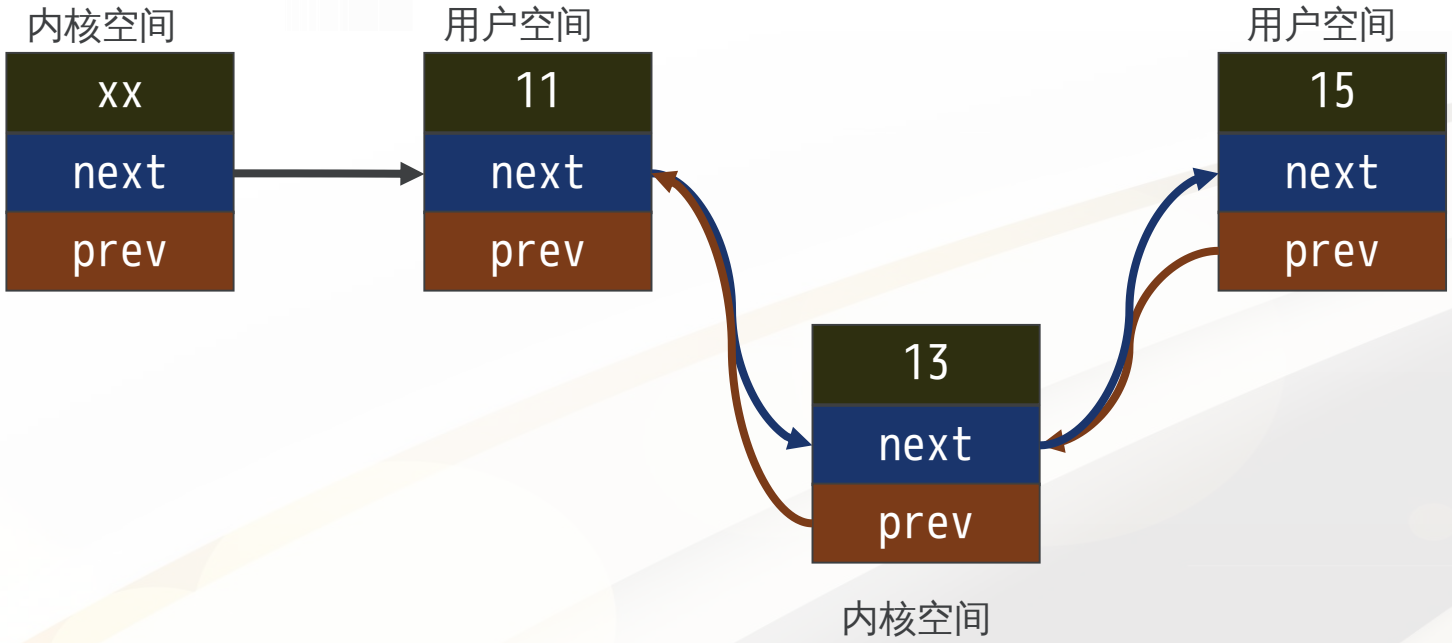
0x04 漏洞利用

- 利用sendmmsg()，覆盖栈上rt_waiter，达到追加新结点的目的。



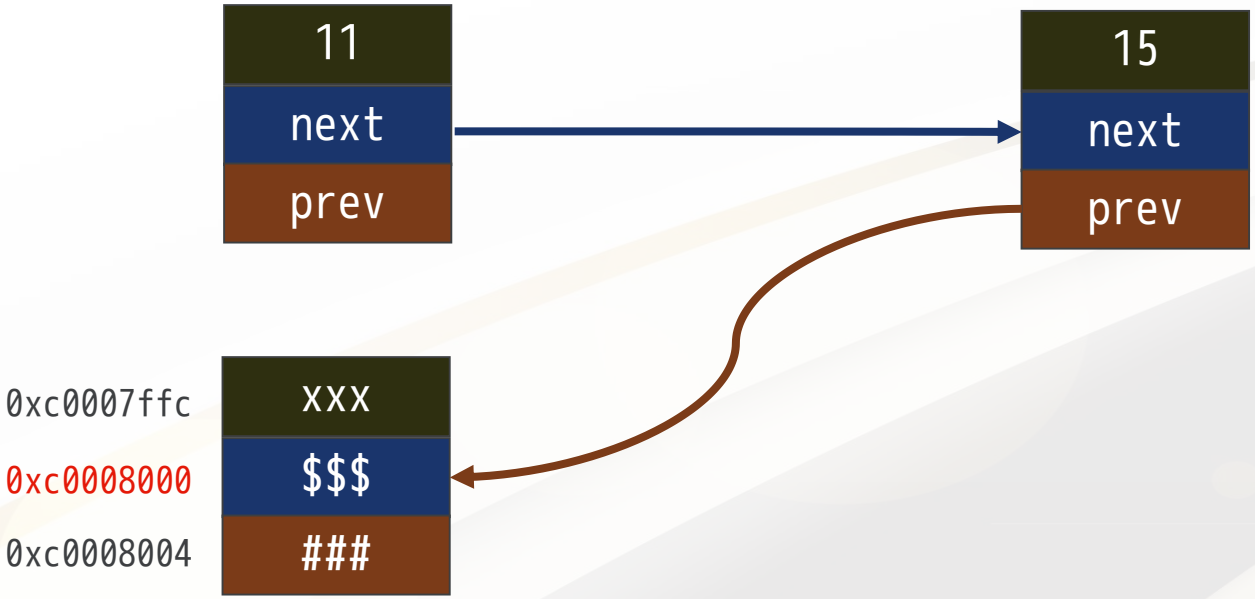
0x04 漏洞利用

- 利用futex_lock_pi()增加一个新节点，优先级排序，泄露了内核栈的值。



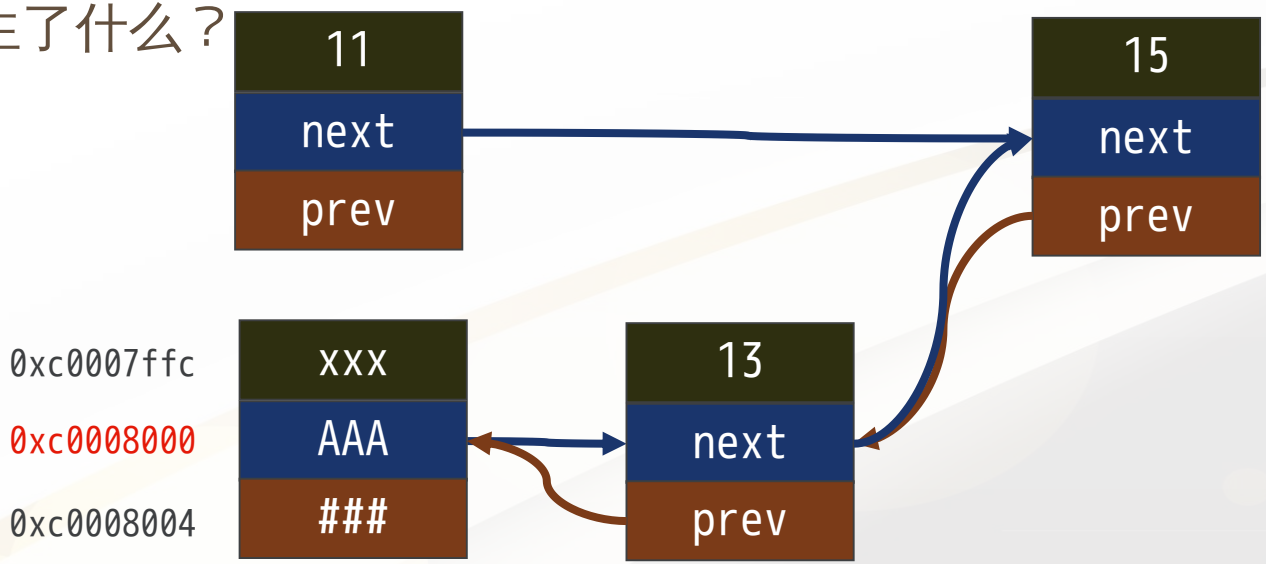
0x04 漏洞利用

- 要求更改0xc0008000的值。先修改prio15的节点的next为0xc0008000。



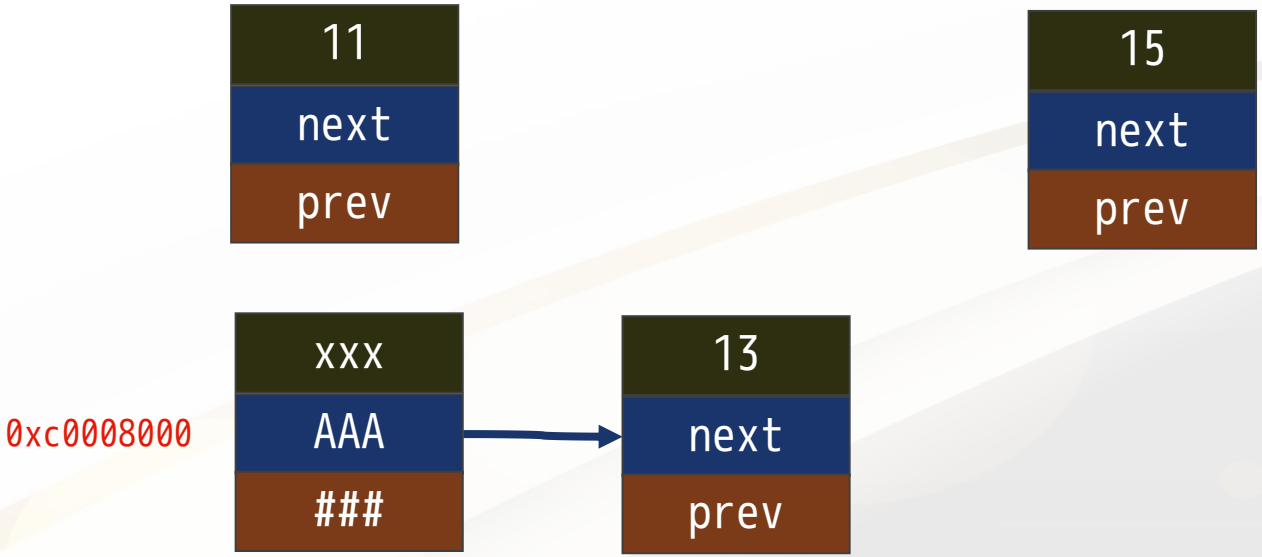
0x04 漏洞利用

- 利用futex_lock_pi()增加一个新节点。
- 发生了什么？



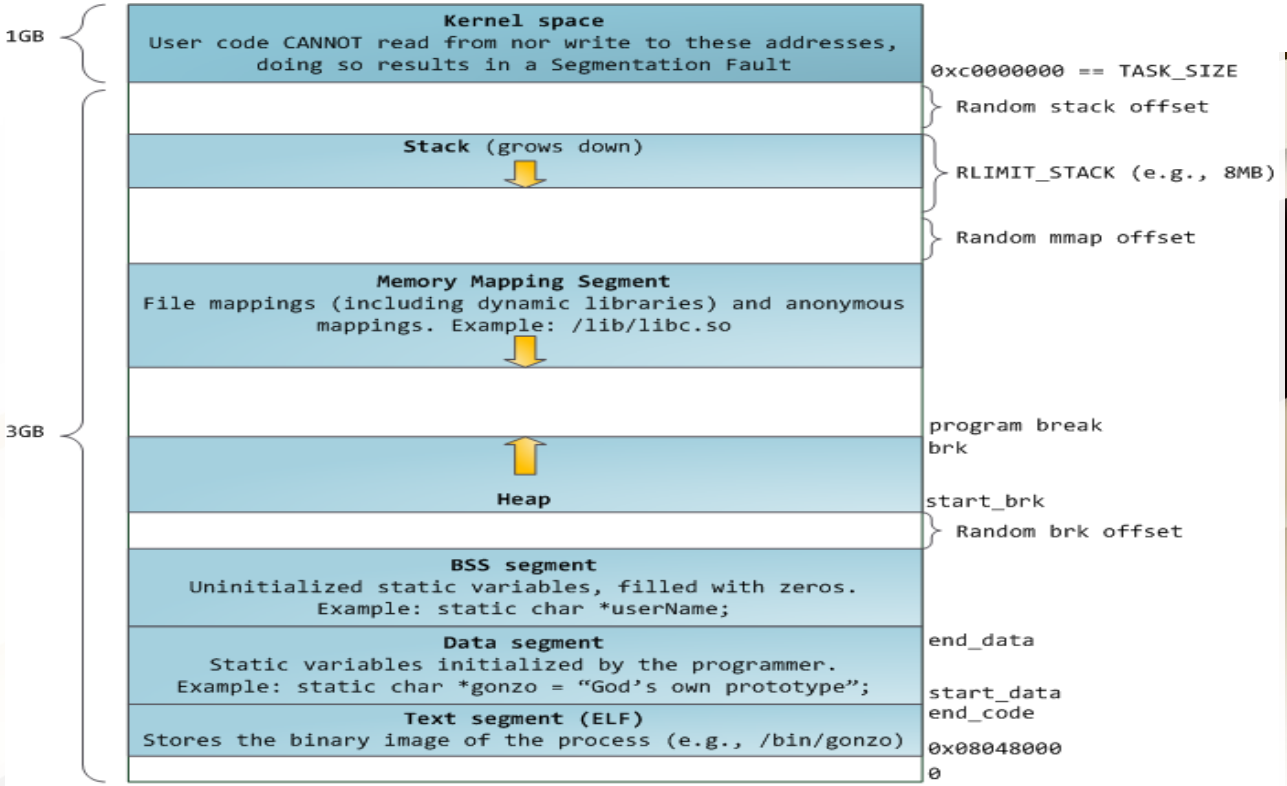
0x04 漏洞利用

- 只留我们关注的。prio13这个节点的地址是多少？



0x04 漏洞利用

- 因为是内核空间，并且rt_waiter在内核线程栈上申请，所以在0xc0000000之上。
- 现在我们已经可以内核中，任意地址写了
- 但所写的值，我们不可控，只能确保在一个范围。
- 能做什么？



0x04 漏洞利用

- 问题0x01：我最终要做什么？

  设置thread_info->task_struct->cred结构的各种信息，使之属于root。

```
cred = {
    usage = {counter = 0x1c},
    uid = {val = 0x3e8},
    gid = {val = 0x3e8},
    suid = {val = 0x3e8},
    /*...*/
    securebits = 0x0,
    cap_inheritable = {cap = {0x0, 0x0}},
    cap_permitted = {cap = {0x0, 0x0}},
    /*...*/}
```

0x04 漏洞利用

- 问题0x02: 要找cred, 先找thread_info, 那它在哪里？

```
static inline struct thread_info *current_thread_info(void)
{
return (struct thread_info *)
(current_stack_pointer & ~(THREAD_SIZE - 1));
}
```

- THREAD_SIZE为8192。因此thread_info = $sp & 0xffffe000

0x04 漏洞利用

- 问题0x03: thread_info在内核栈的栈基址开始，无法从用户态往内核态写内容啊，怎么办？

  为什么不能写？

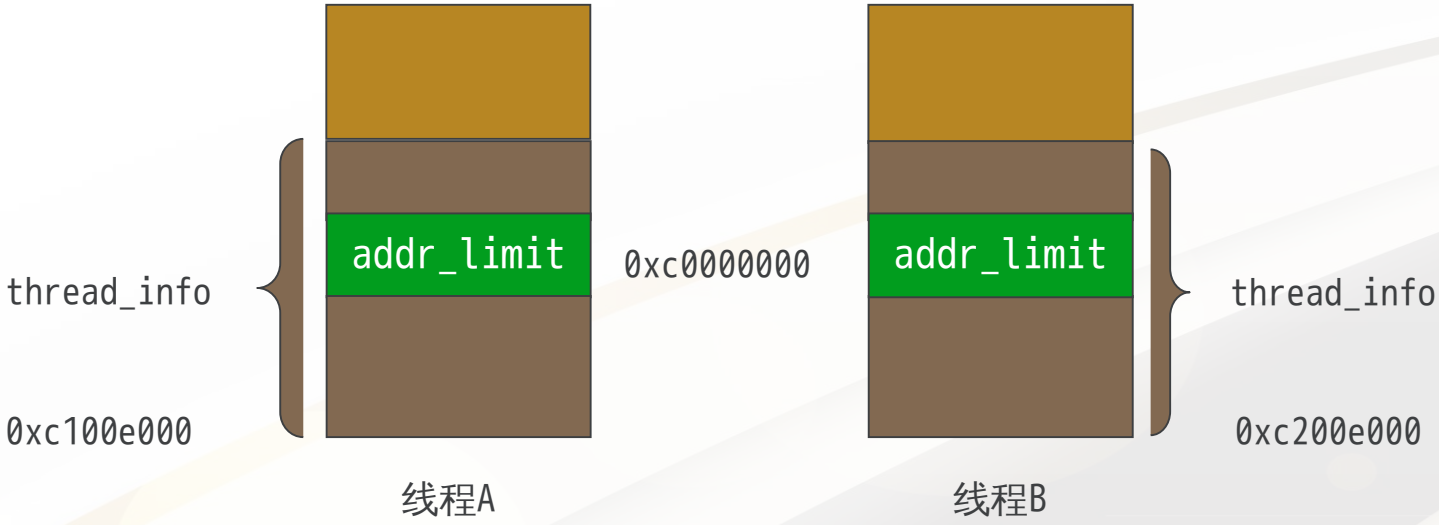
  thread_info->addr_limit限制，默认是0xc0000000

  我们想要整个内核都可写，怎么办？addr_limit=0xffffffff

- 利用我们刚才总结的链表利用方法，内核任意地址写，go on。。。

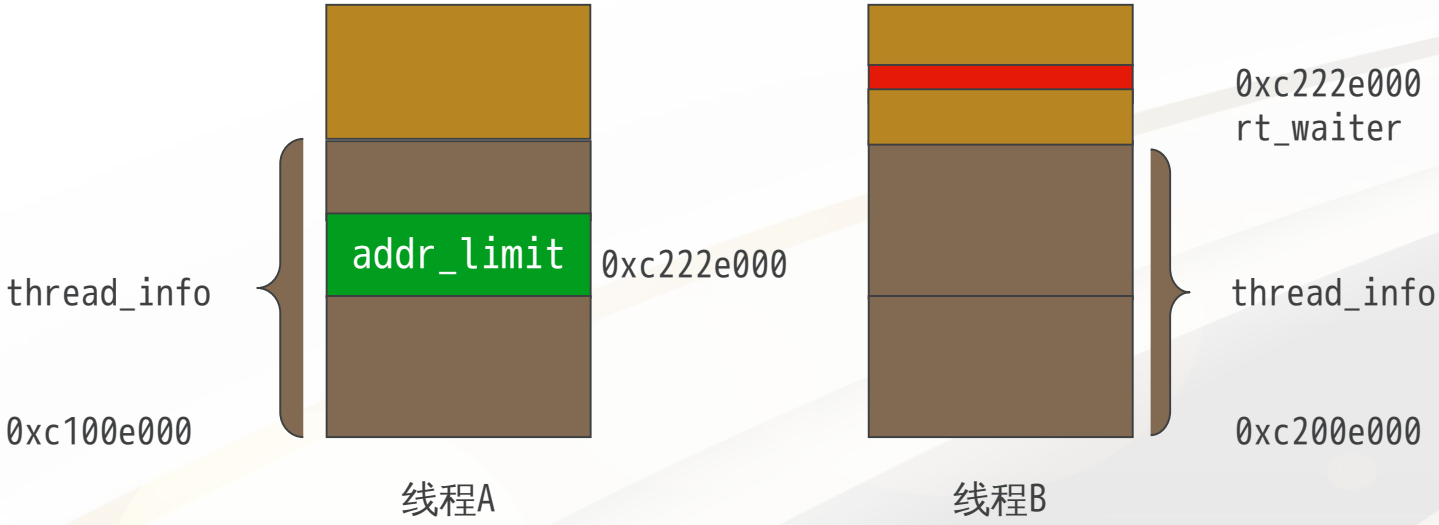
0x04 漏洞利用

- 线程A，B（tips：内核线程共享地址空间）



0x04 漏洞利用

- 线程B在栈上申请新的rt_waiter，加到优先级链表中，更改线程A的thread_info->addr_limit



- 现在线程A，可以对其thread_info结构，从用户态写数据了。

0x04 漏洞利用

- 修改thread_info->task_struct->cred相关信息
- system("/bin/sh")

0x04 漏洞利用

- 目前，没有针对ubuntu x86的利用放出
- arm与x86利用的区别在哪里？

  仅仅是thread_info这个结构体不同而已。
- 经测试ubuntu14.04 和 ubuntu12.04.2成功拿到shell。
- Demo

0x04 漏洞利用

Demo

0x05 参考

[1] http://tinyhack.com/2014/07/07/exploiting-the-futex-bug-and-uncovering-towelroot/

[2] http://blog.topsec.com.cn/ad_lab/cve2014-3153/

[3] https://jon.oberheide.org/files/stackjacking-infiltrate11.pdf

[4] https://github.com/timwr/CVE-2014-3153 ARM架构下的利用

[5] https://github.com/lieanu/CVE2014-3153 移植到X86架构下