XDCTF Writeup 天枢

MISC 100

这里不得不说主办方的脑洞很大，对着 B 平面撸了半天，发现图像最开始的一段被篡改成了 0xf6-0Xf8、0xfb-0xff，但是不知道代表什么。后来主办方提示 braintools，去 github 上找到了：https://github.com/mbikovitsky/BrainTools

编译了一下，然后运行：

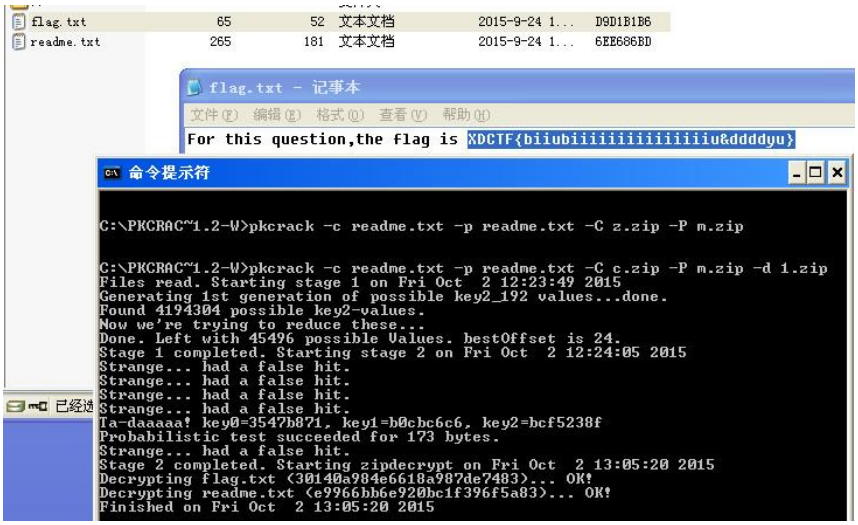
```
F:\Hades\CTF\xdctf2015\BrainTools-master\BrainTools-master\BrainTools\bin\Release>bftools decode braincopter F:\Hades\CT
F\xdctf2015\zzzzzzyu.png -o 1.txt

F:\Hades\CTF\xdctf2015\BrainTools-master\BrainTools-master\BrainTools\bin\Release>bftools run 1.txt
XDCTF{ji910-dad9jq0-iopuno}
```

本题 Flag：XDCTF{ji910-dad9jq0-iopuno}

MISC 200

首先利用 foremost 进行提取，得到两个 zip 包：

最开始以为都是伪加密，去了密码之后显示 CRC 校验错误，看来是有密码的。通过 readme.txt 解压之后的大小，判断是同一个文件。通过 PKCrack 进行明文密码攻击：



得到 flag：XDCTF{biiubiiiiiiiiiiiiiiiu&ddddyu}

PWN 100

这里给的是一个 word2007 rtf 文件的一个栈溢出漏洞样本，通过样本关键字可以搜索到 POC 相关分析的文档和漏洞号：

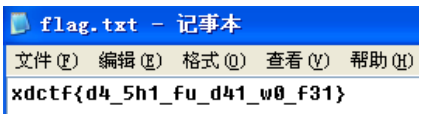
http://www.2cto.com/Article/201301/185786.html

在虚拟机里面安装对应版本的 office，先用 windbg 搜索 90909090，定位到 shellcode 的位置。然用 OD 挂载进程，可以在内存中看到 shellcode 的情况：

```
0012AC64   90              nop
0012AC65   90              nop
0012AC66   90              nop
0012AC67   90              nop
0012AC68   90              nop
0012AC69   90              nop
0012AC6A   90              nop
0012AC6B   90              nop
0012AC6C   4A              dec     edx
0012AC6D   D6              salc
0012AC6E   90              nop
0012AC6F   96              xchg    eax, esi
0012AC70   96              xchg    eax, esi
```

```
0012ADE9  8B 34 8B 01 D6 31 FF AC C1 CF 0D 01 C7 38 E0 75  ???ÿ绎?趋那
0012ADF9  F6 03 7D F8 3B 7D 24 75 E4 58 8B 58 24 01 D3 66  ?)?)$u绢婦$.邇
0012AE09  8B 0C 4B 8B 58 1C 01 D3 8B 04 8B 01 D0 89 44 24  ?K婦■虫┣ ?祕D$
0012AE19  24 5B 5B 61 59 5A 51 FF E0 5F 5F 5A 8B 12 EB 8D  $[[aYZQÿ绯_Z?霓
0012AE29  5D 6A 01 8D 85 B2 00 00 00 50 68 31 8B 6F 87 FF  ]j.癣?..Ph1婧?
0012AE39  D5 BB F0 B5 A2 56 68 A6 95 BD 9D FF D5 3C 06 7C  栈鸟  h  結ÿ?■|
0012AE49  0A 80 FB E0 75 05 BB 47 13 72 6F 6A 00 53 FF D5  .■  u`籗■roj.Sÿ∠
0012AE59  63 6D 64 2E 65 78 65 20 2F 63 20 65 63 68 6F 20  cmd.exe /c echo
0012AE69  78 64 63 74 66 7B 64 34 5F 35 68 31 5F 66 75 5F  xdctf{d4_5h1_fu_
0012AE79  64 34 31 5F 77 30 5F 66 33 31 7D 20 3E 20 63 3A  d41_w0_f31} > c:
0012AE89  5C 66 6C 61 67 2E 74 78 74 20 26 26 20 61 74 74  \flag.txt && att
0012AE99  72 69 62 20 63 3A 5C 66 6C 61 67 2E 74 78 74 20  rib c:\flag.txt
```

得到最终 flag：



Reverse 100

这题有个假的函数验证函数 sub_4008E1：

```
if ( (unsigned int)sub_400626(0x601328LL, 0xCu) == 1 )
{
  qword_601338 = aCongratulation;
  for ( i = 0; i < strlen((const char *)&good_crack_base64); ++i )
    byte_601310[(signed __int64)i] = pwd[i
                                       - 12
                                       * ((unsigned __int64)((unsigned __int128)(0xAAAAAAAAAAAAAAABLL * i) >> 64) >> 3)] ^ L
  sub_4006D5(byte_601310, 24, 12);
  for ( j = 0; j <= 0x17; ++j )
  {
    if ( byte_601310[(signed __int64)(signed int)j] <= 31 )
      byte_601310[(signed __int64)(signed int)j] += 32;
  }
  v1 = 1;
  for ( k = 0; k <= 0x17; ++k )
  {
    if ( byte_601310[(signed __int64)(signed int)k] != byte_6012A0[(signed __int64)(signed int)k] )
      v1 = 0;
  }
```

基本算法逻辑是异或上一段 base64 然后再异或 0x07，解出来之后一直不对，后来发现真正的逻辑在 sub_400787 处，唯一的区别是少异或了一个 0x07，写脚本暴力跑一下：

```
for ii in range(0, a2/2):
    v3 = result[ii]
    result[ii] = result[a2 - ii - a3/2 -1]
    result[a2 - ii - a3/2 -1] = v3

print result

for ii in range(0, 24):
    if result[ii] > 31:
        s1.append(result[ii]-32)
    else:
        s1.append(result[ii])

for ii in range(0, 24):
    r1 = chr(result[ii] ^ ord(a[ii]))
    r2 = chr(s1[ii] ^ ord(a[ii]))

    print hex(ord(r1)), r1, hex(ord(r2)), r2
```

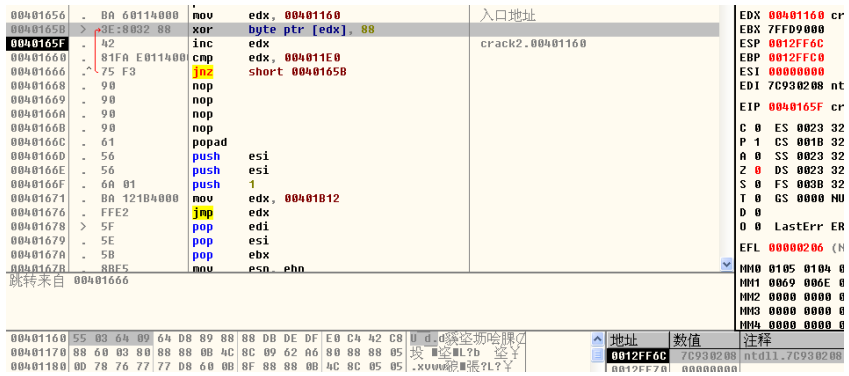
```
0x75 u 0x55 U
0x27 ' 0x47 G
0x72 r 0x52 R
0x45 E 0x65 e
0x7f  0x5f _
0x61 a 0x41 A
0x77 w 0x17 
0x65 e 0x45 E
0x53 S 0x73 s
0x30 0 0x50 P
0x6d m 0x4d M
0x45 E 0x65 e
0x75 u 0x55 U
0x7  0x27 '
0x72 r 0x52 R
0x45 E 0x65 e
0x7f  0x5f _
0x41 A 0x61 a
0x77 w 0x57 W
0x45 E 0x65 e
0x53 S 0x73 s
0x10  0x30 0
0x6d m 0xd
0x65 e 0x5 
```

根据主办方的提示，都是小写字母，所以最后的 flag：XDCTF{u’re_awes0me}

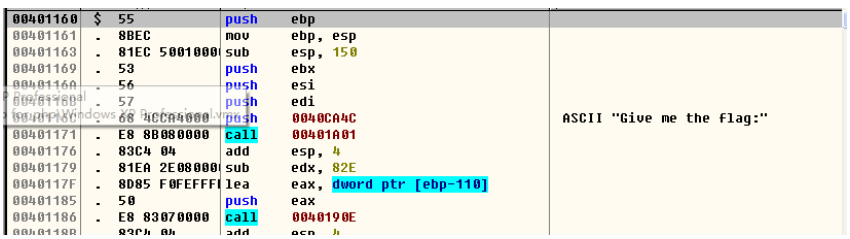
Reverse 200

这题是在入口点之后，会有一段代码解密的模块，对 0x401166 模块的代码进行解码，需要在解密模块之后分析代码功能，还有一堆花指令。

对 0x401166 下硬件访问断点，然后会断在这里：



跳过这个循环，然后看看 00401160 处的代码：



这样就看到了核心代码，往下跟一跟输入：

```
004011EB  .  83BD E8FEFFFF cmp     dword ptr [ebp-118], 19          25
004011F2  .  7D 09         jge     short 004011FD
004011F4  .  83BD E8FEFFFF cmp     dword ptr [ebp-118], 0
004011FB  .  7F 1A         jg      short 00401217
004011FD  >  68 60CA4000   push    0040CA60                         ASCII "You shall not pass",LF
00401202  .  E8 FA070000   call    00401A01
```

长度要要求小于 25，并且以 XDCTF{开头：

```
004012DA  .  8B95 D0FEFFFF mov     edx, dword ptr [ebp-130]
004012E0  .  0FBE8415 F0FE movsx   eax, byte ptr [ebp+edx-110]
004012E8  .  3BC8          cmp     ecx, eax
004012EA  .  74 1A         je      short 00401306                   XDCTF{
004012EC  .  68 7CCA4000   push    0040CA7C                         ASCII "You shall not pass!"
004012F1  .  E8 0B070000   call    00401A01
004012F6     83C4 04       add     esp, 4
```

接下来会判断第 24 个字符是不是}，并且指出_和$的位置：

```
0040130F  .  83F9 7D       cmp     ecx, 7D
00401312  .  74 1A         je      short 0040132E
00401314  .  68 7CCA4000   push    0040CA7C                         ASCII "You shall not pass!"
00401319  .  E8 E3060000   call    00401A01
0040131E  .  83C4 04       add     esp, 4
00401321  .  E8 A5070000   call    00401ACB
00401326  .  83C8 FF       or      eax, FFFFFFFF
00401329  .  E9 4A030000   jmp     00401678
0040132E  >  0FBE95 FCFEFF movsx   edx, byte ptr [ebp-104]
00401335  .  83FA 5F       cmp     edx, 5F
00401338  .  75 0C         jnz     short 00401346
0040133A  .  0FBE85 02FFFF movsx   eax, byte ptr [ebp-FE]
00401341  .  83F8 24       cmp     eax, 24
```

所以使用 XDCTF{123456_7890a$bcde}进行测试，找到第一部分的 check 代码：

```
004014D5  .  51            push    ecx
004014D6  .  8B95 E4FEFFFF mov     edx, dword ptr [ebp-11C]
004014DC  .  52            push    edx                              first_part
004014DD  .  8B85 ECFEFFFF mov     eax, dword ptr [ebp-114]
004014E3  .  50            push    eax                              XDCTF_
004014E4  .  E8 57FBFFFF   call    00401040
```

```
004010B6  .  2BC2          sub     eax, edx
004010B8  .  8B4D F8       mov     ecx, dword ptr [ebp-8]
004010BB  .  8B55 10       mov     edx, dword ptr [ebp+10]
004010BE  .  3B448A 08     cmp     eax, dword ptr [edx+ecx*4+8]
004010C2  .  74 04         je      short 004010C8
004010C4  .  32C0          xor     al, al
004010C6  .  EB 04         jmp     short 004010CC
004010C8  >^ EB C6         jmp     short 00401090
004010CA  >  B0 01         mov     al, 1
004010CC  >  8BE5          mov     esp, ebp
堆栈 ds:[0012FEF8]=00000015
eax=00000027

0012FEF8  15 00 00 00 D5 FF FF FF D5 FF FF FF ED FF FF FF  ■...?ÿÿ?ÿÿ?ÿÿ
0012FF08  D4 FF FF FF FE FF FF FF CC FF FF FF FE FF FF FF  ?ÿÿ?ÿÿ?ÿÿ?ÿÿ
```

循环相减 XDCTF_，等于固定的数值，推出第一部分 Congra，接下来是第二部分的 check：

```
004010EE  .  8D55 F8       lea     edx, dword ptr [ebp-8]
004010F1  .  8955 F4       mov     dword ptr [ebp-C], edx
004010F4  .  8B45 08       mov     eax, dword ptr [ebp+8]
004010F7  .  8945 F0       mov     dword ptr [ebp-10], eax
004010FA  >  8B4D F0       mov     ecx, dword ptr [ebp-10]
004010FD  .  8A11          mov     dl, byte ptr [ecx]
004010FF  .  8855 EF       mov     byte ptr [ebp-11], dl
00401102  .  8B45 F4       mov     eax, dword ptr [ebp-C]
00401105  .  3A10          cmp     dl, byte ptr [eax]
00401107  .  75 2E         jnz     short 00401137
00401109  .  807D EF 00    cmp     byte ptr [ebp-11], 0
0040110D  .  74 1F         je      short 0040112E
0040110F  .  8B4D F0       mov     ecx, dword ptr [ebp-10]
堆栈地址=0012FE08, (ASCII "tUlat")
edx=000109AB
跳转来自 004010E9, 004010EB
```

这里是直接字符串比较，第二部分是 tUlat，接下来是第三部分：

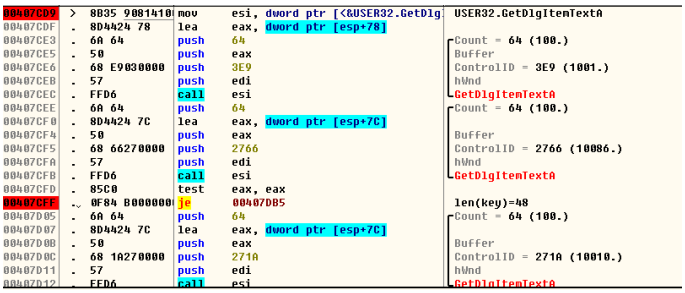
```
004015E4  .  75 3F         jnz     short 00401625
004015E6  .  0FBE85 DDFEFF movsx   eax, byte ptr [ebp-123]
004015ED  .  0FBE8D F2FEFF movsx   ecx, byte ptr [ebp-10E]
004015F4  .  33C1          xor     eax, ecx
004015F6  .  83F8 3A       cmp     eax, 3A
004015F9  .  75 2A         jnz     short 00401625
004015FB  .  0FBE95 DEFEFF movsx   edx, byte ptr [ebp-122]
00401602  .  0FBE85 F1FEFF movsx   eax, byte ptr [ebp-10F]
00401609  .  33D0          xor     edx, eax
0040160B  .  83FA 0B       cmp     edx, 0B
0040160E  .  75 15         jnz     short 00401625
00401610  .  0FBE8D DFFEFF movsx   ecx, byte ptr [ebp-121]
00401617  .  0FBE95 F0FEFF movsx   edx, byte ptr [ebp-110]
0040161E  .  33CA          xor     ecx, edx
00401620  .  83F9 2D       cmp     ecx, 2D
00401623  .  74 12         je      short 00401637
```

得到最后的 Flag：XDCTF{Congra_tUlat$eyOu}

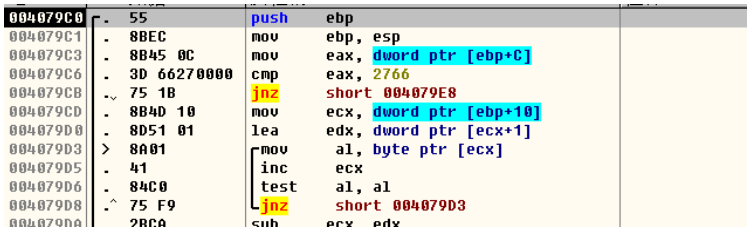
Reverse 500

这题首先考虑如何断下来，点击注册后，会弹出注册码格式错误的对话框，于是使用运行+弹框+“ALF+F9”的方法回到用户代码：

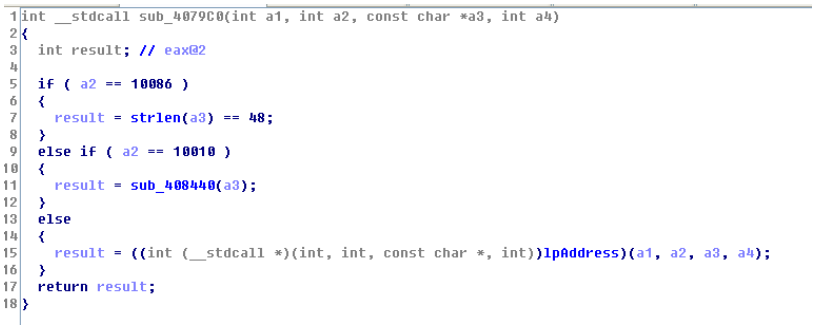
往上找找看到按钮函数的入口：



看到这里愣了半天，因为只有两个 Edit 框，但是 ID 却有两个不是控件的 ID，而且上来就会挂在这个函数的返回值中……后来手动 fuzz 了一下，发现输入 47 个 1 和 1 个 0 会成功跳过两个判断……然而，有次手贱跟进去才发现，似乎有个 jmp 到了一个奇怪的地方：



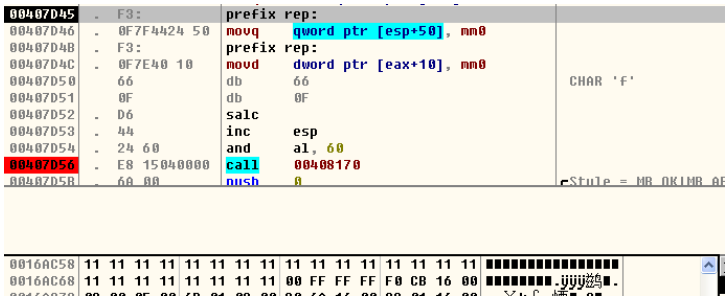
IDA 看看：



```
int __stdcall sub_4079C0(int a1, int a2, const char *a3, int a4)
{
  int result; // eax@2

  if ( a2 == 10086 )
  {
    result = strlen(a3) == 48;
  }
  else if ( a2 == 10010 )
  {
    result = sub_408440(a3);
  }
  else
  {
    result = ((int (__stdcall *)(int, int, const char *, int))lpAddress)(a1, a2, a3, a4);
  }
  return result;
}
```

原来是这个套路，其他两个 GetDlgItemTextA 是用来做长度判断和内容检测，要求长度为 48 字节且为数字或者大写字母……

然后对数据输入下硬件访问断点，可以看到调用了 sub_408480 函数，做了 hex2bin 的转换：



然后进入解密函数：

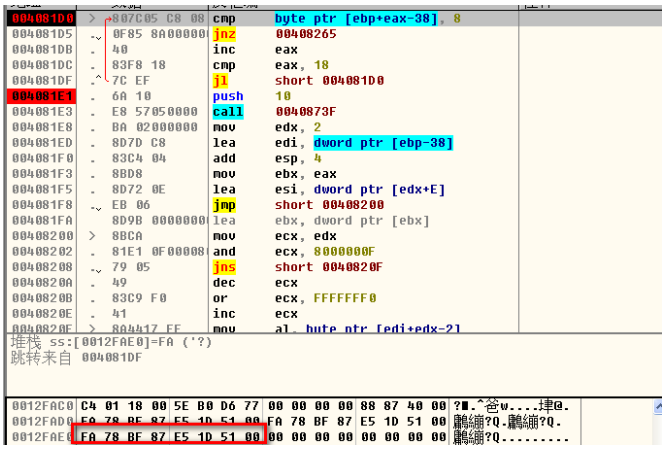
```
v15 = 0x99864534;
v16 = 0xA5CD4B1A;
v9.m128i_i8[0] = 0;
_mm_storel_epi64((__m128i *)&v10, 0i64);
_mm_storeu_si128((__m128i *)((char *)&v9 + 1), 0i64);
v11 = 0;
v12 = 0;
v13 = 0;
sub_408010((int)&v15, (int)&v9);
v0 = 16;
```

这里将 8 字节密钥 push 进了 sub_408010，IDA 进去看看：

```
sub_408690("解密之前: ");
v4 = 0;
do
  sub_408690("%.2X", v11.m128i_i8[v4++]);
while ( v4 < 24 );
sub_408690("\n");
v5 = a2;
v6 = &v11.m128i_i8[-a2];
v7 = 3;
do
{
  v19 = *(_DWORD *)&v6[v5];
  v20 = *(_DWORD *)&v6[v5 + 4];
  sub_4015C0((int)&v19, (int)v23, (int)&v10, 0);
  v5 += 8;
  *(_DWORD *)(v5 - 8) = *(_DWORD *)v23;
  *(_DWORD *)(v5 - 4) = *(_DWORD *)&v23[4];
  --v7;
```

这里写了是做了解密，同时用处理过的密钥 v10 进行解密，每次解密 8 字节，放到 v5 中。用 PEID 插件看下：

唔……结合 64 位明文组和 8 字节密钥，猜想是 DES ECB 模式解密，写了个 python 脚本对比了下解密结果，发现符合解密情况，那就好办了。接下来看看还进行了什么操作：



取最后一组解密结果与 0808080808080808 进行比较，所以先将该组加密结果作为注册

码的最后 16 个 16 进制位，即 970BB701CAF237F3。

接下来：

```
do
{
  v5 = v2 % 16;
  v6 = v9.m128i_i8[v2++ - 2];
  *((_BYTE *)v3->m128i_i32 + v5) = v6;
  --v4;
}
while ( v4 );
_mm_storeu_si128((__m128i *)&v14, _mm_loadu_si128(v3));
free(v3->m128i_i32);
v7 = 0;
_mm_storeu_si128(&v9, _mm_xor_si128(_mm_loadu_si128((const __m128i *)&v14), (__m128i)xmmword_42BD90));
while ( *((_BYTE *)&xmmword_430BC0 + v7) == v9.m128i_i8[v7] )
{
  ++v7;
  if ( v7 >= 16 )
    return 1;
```

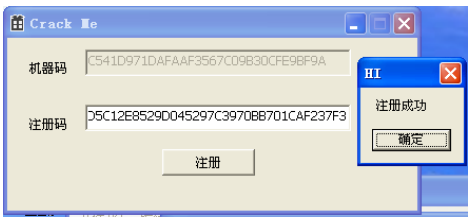
这个逻辑就比较清晰了，前 16 字节首先进行移位，赋值给 v14，然后与 0x42bd90 处做异或，然后与 0x430bc0 处（机器码 16 进制字节）做比较，相等即通过。

用 python 写个 keygen 脚本：

```
import pyDes
import binascii

mechine = 'C541D971DAFAAF3567C09B30CFE9BF9A'.decode('hex')

xor_m = ''

for m in mechine:
    xor_m += chr(ord(m)^0xe4)

result = ''

s = 2
for i in range(0, 16):
    result += xor_m[(i+s)%16]

result += ('0808080808080808'.decode('hex'))
#print binascii.b2a_hex(result)

key = '344586991A4BCDA5'.decode('hex')
k = pyDes.des(key)
d = k.encrypt(result)

flag = binascii.b2a_hex(d)
Flag = ''

for f in flag:
    if ord(f) in range(ord('a'), ord('z')+1):
        Flag += chr(ord(f)-ord('a')+ord('A'))
    else:
        Flag += f

print Flag
```

算出注册码：282FDA1AD8521D5C12E8529D045297C3970BB701CAF237F3

注册成功，提交之后获得 flag：



Crypt 200

这题是 AES CBC 字节翻转攻击：

http://cryptopals.com/sets/2/challenges/16/

http://drops.wooyun.org/tips/7828

基本套路是修改前一个分组对应位置的密文，使用异或之后的结果出现;，绕过加密时的限制，但是返回的密文开头并不是第一个字节，所以往后爆破了一下，脚本如下：

```
send_buf = 'mkprof:'
data1 = '@admin=true'

prefix = "comment1=wowsuch%20CBC;userdata="
suffix = ";coment2=%20suchsafe%20very%20encryptwowww"

print len(prefix),len(suffix),len(data1)

sock.send(send_buf + data1 + '\n')
# 接收消息
response_data = sock.recv(1024)
print response_data, len(response_data)
print data1

for i in range(0, 32):
    first = response_data[2*i:2*i+2].decode('hex')

    change_bit = chr(ord(first) ^ ord(';') ^ ord('@'))
    data2 = response_data[0:2*i] + change_bit.encode('hex') + response_data[2*i+

    print data2

    send_buf = 'parse:'
    sock.send(send_buf + data2 + '\n')
    ret = sock.recv(1024)
    #print data2
    print ret


# 接收消息
response_data = sock.recv(1024)
print 'recv:' + response_data
```

最后跑出来 flag：

```
Parsing Profile...You are a normal user.

684299166a05383e6eaa9139f8d8f5848cda560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ea1cb9c7b4ea7896428d

Parsing Profile...You are a normal user.

684299166a05383e6eaa9139f8d8f5fff7da560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ea1cb9c7b4ea7896428d

Parsing Profile...Congratulations!
The FLAG is: XDCTF{WelcometToCyberEngineeringSchool2333}
684299166a05383e6eaa9139f8d8f5ff8ca1560698b1987eb2092534397496b7f49b2991e0d7cfd5
0863843548ef0f9a6924069ce5564a8c832f3f9d36f49990b85a449a7a5234fba0315c69e63323d3
996be3d42536ea1cb9c7b4ea7896428d
```

Web1 100

神奇的 MD5。。。刚开始以为真的让碰撞。。。然而因为没猜到 test=是让干嘛，就猜可能是能拿到代码，试了一些可能导致代码泄漏的方式试验，最后发现 test.php~可以看到源码，但是源码是被加密的，百度搜了一个解密的网站 http://tool.lu/php/

解密之后看到代码，发现要绕过 $test=md5($test); if($test=='0') ，这个可以利用 PHP 的科学计数法绕过

搜到这个帖子

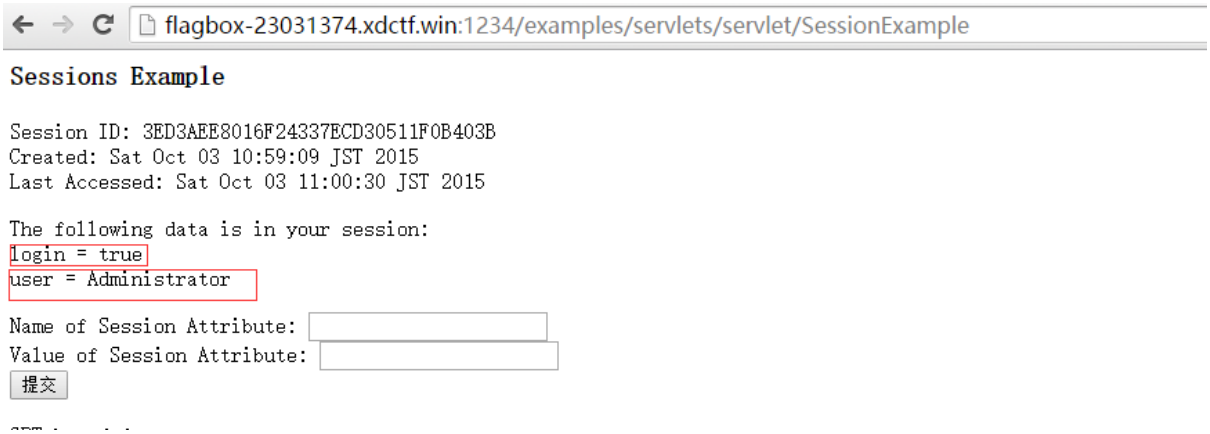
http://stackoverflow.com/questions/22140204/why-md5240610708-is-equal-to-md5qnkcdzo

payload:QNKCDZO

Web1 200

查看网站页面代码，发现/examples/目录

Tomcat 的 examples 页面未删除，导致 session 可以被设置,根据 examples 页面的表单猜出 user=Administrator，设置访问后发现说没有 login，然后设置 login=true



重新访问 http://flagbox-23031374.xdctf.win:1234/examples/
即可拿到 flag

Web1 400

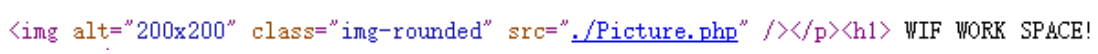
能输入的就一个登录，其他啥东西都没有。还有个 SleepzzzZZZ 是什么鬼！

Robots.txt 中发现提示：
Tips:

Brute-force cracking is not usefull!
thinking again and again! Good luck!

恩，和没说一样。。。。

仔细查看页面源码，发现 freebuf 这个图片源自 picture.php



访问之，发现图片末尾的提示：
<!--Please input the ID as parameter with numeric value-->

http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?ID=1
id=1 时返回图片，=2 时返回 not found，猜测为 sql 注入

根据提示 双引号

构造 payload：

2″ or 1=1%23　返回图片
2″ or 1=2%23　返回 not found
证实注入。

测试发现，后台暴力过滤了 select union and sleep benchmark ascii substr instr mid left right 等一系列关键字，出现黑名单中的关键字就会直接返回 ID=1 时的图片，不执行查询。

(ノ゜Д゜)ノ ┻━┻
要不要这样。。。。把 select 去掉我就不会玩了好么。。。

使用各种姿势尝试绕过 select 过滤无果后，惊奇地发现原本查询的表中有 username 字段：
ID=1″ %26%26 username is not null%23 返回图片
同样的姿势发现 password 字段

盲注猜之，
ID=1″ %26%26 username =’admin’ %23
ID=1″ %26%26 password =’ 5832f4251cb6f43917df’ %23

20 位 md5，估计不是登录密码。
按 dedecms 加密格式，去掉前三位后一位，向土豪乞讨 1 毛，解码 2f4251cb6f43917d
得到 lu5631209

登录之，获得 flag：XDCTF{e0a345cadaba033073d88d2cc5dce2f7}

Web2 200

源码泄漏，用 ript-git.pl

拉回一堆.git 下的文件

在 README 中发现提示

All source files are in git tag 1.0

去题目站点下载.git\refs\tags\1.0 丢进对应目录

导出打了 tag 1.0 的文件

$ git archive --format=zip --output=1.zip 1.0

获得源码，源码中 index.php 包含 flag

```
<?php
/*
```

Congratulation, this is the [XDSEC-CMS] flag 1

XDCTF-{raGWvWahqZjww4RdHN90}

*/

echo "Hello World";
?>

Web2 100

提示为逻辑漏洞

重置密码的逻辑位于：

xdsec_app\front_app\controllers\Auth.php

```
public function handle_resetpwd()
{
    if(empty($_GET["email"]) || empty($_GET["verify"])) {
        $this->error("Bad request", site_url("auth/forgetpwd"));
    }
    $user = $this->user->get_user(I("get.email"), "email");
    if(I('get.verify') != $user['verify']) {
        $this->error("Your verify code is error", site_url('auth/forgetpwd'));
    }
    if($this->input->method() == "post") {
        $password = I("post.password");
        if(!$this->confirm_password($password)) {
            $this->error("Confirm password error");
        }
        if(!$this->complex_password($password)) {
            $this->error("Password must have at least one alpha and one number");
        }
        if(strlen($password) < 8) {
            $this->error("The Password field must be at least 8 characters in length");
        }
        $this->user->update_userinfo([
            "password" => $password,
            "verify" => null
        ], $user["uid"]);
        $this->success("Password update successful!", site_url("auth/login"));
    } else {
        $url = site_url("auth/resetpwd") . "?email={$user['email']}&verify={$user['verify']}";
        $this->view("resetpwd.html", ["form_url" => $url]);
    }
}
```

重置密码链接由邮箱+随机 verify 组成,但重置成功时会将 verify 重置为 null

题目网站右键源码可以发现作者的邮箱地址：
<meta name="author" content="xdsec-cms@xdctf.com"/>

于是访问链接：
http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com &verify=%20
可重置管理员密码（貌似管理员的 verify 一直是 null 不会改变）

登录后可查看 flag 文件：

Congratulation, this is the [XDSEC-CMS] flag 2

XDCTF-{i32mX4WK1gwEE9S9Oxd2}

hint:
admin url is /th3r315adm1n.php