XDCTF_WRITEUP

By 在下叶良辰

Reverse

reverse100

这题还真的让不了解 linux 的我涨了个大姿势。刚拿到本题，就想着是道简单的逆向，直接把算法逆出来，然后根据提示提交合适的 flag，然后发现怎么交怎么错，后来在程序中发现了这两个东西：

```
.init_array:0000000000601000 _init_array     segment para public 'DATA' use64
.init_array:0000000000601000                 assume cs:_init_array
.init_array:0000000000601000                 ;org 601000h
.init_array:0000000000601000 off_601000      dq offset sub_400600    ; DATA XREF: .text:000000000040AB1↑o
.init_array:0000000000601008                 dq offset sub_400669
.init_array:0000000000601008 _init_array     ends
.init_array:0000000000601008
.fini_array:0000000000601010 ; ===========================================================================
.fini_array:0000000000601010
.fini_array:0000000000601010 ; Segment type: Pure data
.fini_array:0000000000601010 ; Segment permissions: Read/Write
.fini_array:0000000000601010 ; Segment alignment 'qword' can not be represented in assembly
.fini_array:0000000000601010 _fini_array     segment para public 'DATA' use64
.fini_array:0000000000601010                 assume cs:_fini_array
.fini_array:0000000000601010                 ;org 601010h
.fini_array:0000000000601010 off_601010      dq offset sub_4005E0    ; DATA XREF: .text:000000000040AB9↑o
.fini_array:0000000000601018                 dq offset Real
.fini_array:0000000000601018 _fini_array     ends
.fini_array:0000000000601018
```

上网查了一下，得知 init_array 的函数会在程序装载完成后被执行，fini_array 的函数会在程序终止时被执行。这下清楚了。老老实实写段解密，然后根据提示提交正确的 flag 即可。

flag : XDCTF{u're_awes0me}

reverse200

首先有一个 TIs 的回调函数，里面有个反调试，吾爱破解的 OD 可以直接过了

```
void __stdcall TlsCallback_0(int a1, int a2, int a3)
{
  if ( a2 == 1 && IsDebuggerPresent() )
  {
    MessageBoxA(0, "You shall not pass", "ERROR", 0);
    ExitProcess(0xFFFFFFFF);
  }
}
```

然后是对代码部分进行解密

```
60              pushad
BA 60114000     mov edx,1.00401160
3E:8032 88      xor byte ptr ds:[edx],0x88
42              inc edx
81FA E0114000   cmp edx,1.004011E0
75 F3           jnz short 1.0040165B
```

解密后来到这里：

```
55              push ebp
8BEC            mov ebp,esp
81EC 50010000   sub esp,0x150
53              push ebx
56              push esi
57              push edi
68 4CCA4000     push 1.0040CA4C                 ASCII "Give me the flag:"
E8 8B080000     call 1.00401A01
83C4 04         add esp,0x4
81EA 2E080000   sub edx,0x82E
8D85 F0FEFFFF   lea eax,dword ptr ss:[ebp-0x110]
50              push eax
E8 83070000     call 1.0040190E
83C4 04         add esp,0x4
8D8D F0FEFFFF   lea ecx,dword ptr ss:[ebp-0x110]
898D CCFEFFFF   mov dword ptr ss:[ebp-0x134],ecx
8B95 CCFEFFFF   mov edx,dword ptr ss:[ebp-0x134]
83C2 01         add edx,0x1
```

然后是先判断输入长度为 0x19

然后依次判断格式

先确定格式是这样的：XDCTF{012345678901234567} ，先用数字填充后面依次替换

先替换个别位得到：XDCTF{012345_789012$4567}

```
cmp ecx,0x7D
je short 1.0040132E
push 1.0040CA7C
call 1.00401A01
add esp,0x4
call 1.00401ACB
or eax,-0x1
jmp 1.00401678
movsx edx,byte ptr ss
cmp edx,0x5F
jnz short 1.00401346
movsx eax,byte ptr ss
cmp eax,0x24
```

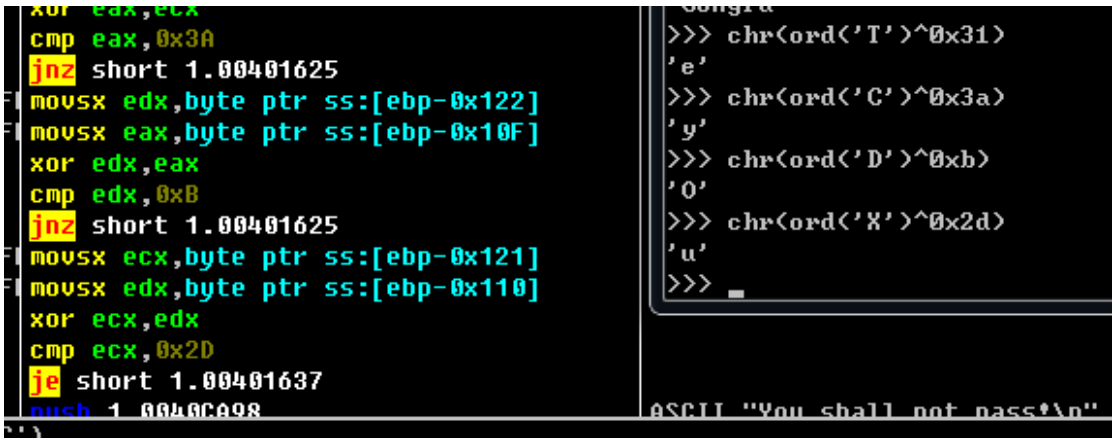
然后：“XDCTF_”和“0123456”依次相减，得到{0x15，-0x2b,-0x2b,-0x13,-0x2c,-2,-0x34}

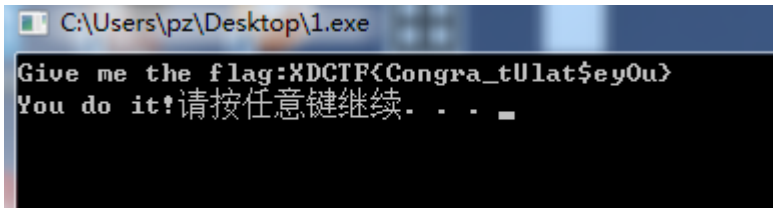
XDCTF{ Congra _789012$4567}

然后替换另一段：XDCTF{ Congra _789012$4567}

```
8B55 E8         mov edx,dword ptr ss:[ebp-
018FDCC]=0018FDD0, (ASCII "tUlat")
0, (ASCII "tUlat")
```

然后是一段异或，得到最终 flag：XDCTF{Congra_tUlat$eyOu}

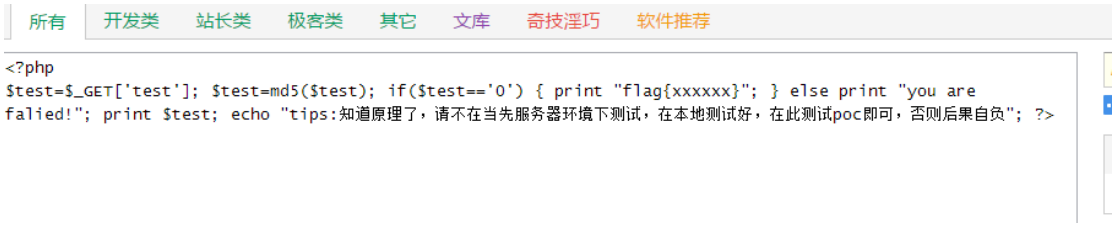




Web1

Web1-100:

在目录下发现 index.php~，查看源码发现被混淆加密的 phpjm，zgoogle 一下 phpjm 解密，找到一个网址 http://tool.lu/php/，解密之后为：

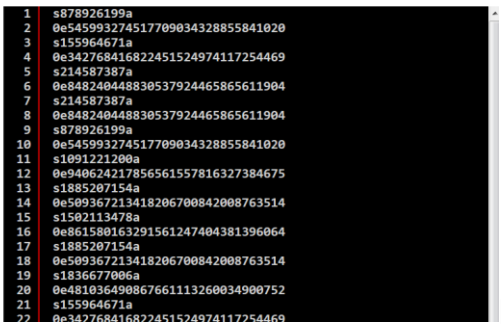


关键在于 test 在 MD5 加密之后为 0，google 搜索一下，发现 http://www.cnblogs.com/xia0zhiwei/archive/2015/08/08/4712646.html

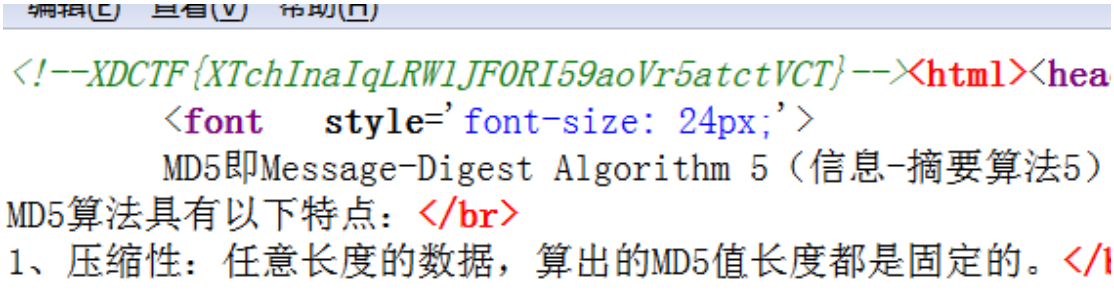
在这片文章中提到，php 对 MD5 的识别存在弱比较漏洞，对 hash 以 0e 开头的会被解释为 0，继续 google 以 0e 开头的 MD5，找到 http://www.219.me/posts/2884.html，

oe开头MD5值小结

2015 年 6 月 18 日 Code、PHP Admin



随便选取一个，构造 url:http://133.130.90.172/5008e9a6ea2ab282a9d646befa70d53a/index.php?test=s1502113478a 得到 flag



XDCTF{XTchInaIqLRW1JFORI59aoVr5atctVCT}

Web1-200

简单探测发现是 php 的，awvs 简单扫一下发现一个 tomcat 的 examples 目录，想起曾经在 m00n 上看到说 tomcat 的 examples 目录下如果存在一个文件可以修改任意 cookie 实现绕过，在上面找了一下，找到了这个：

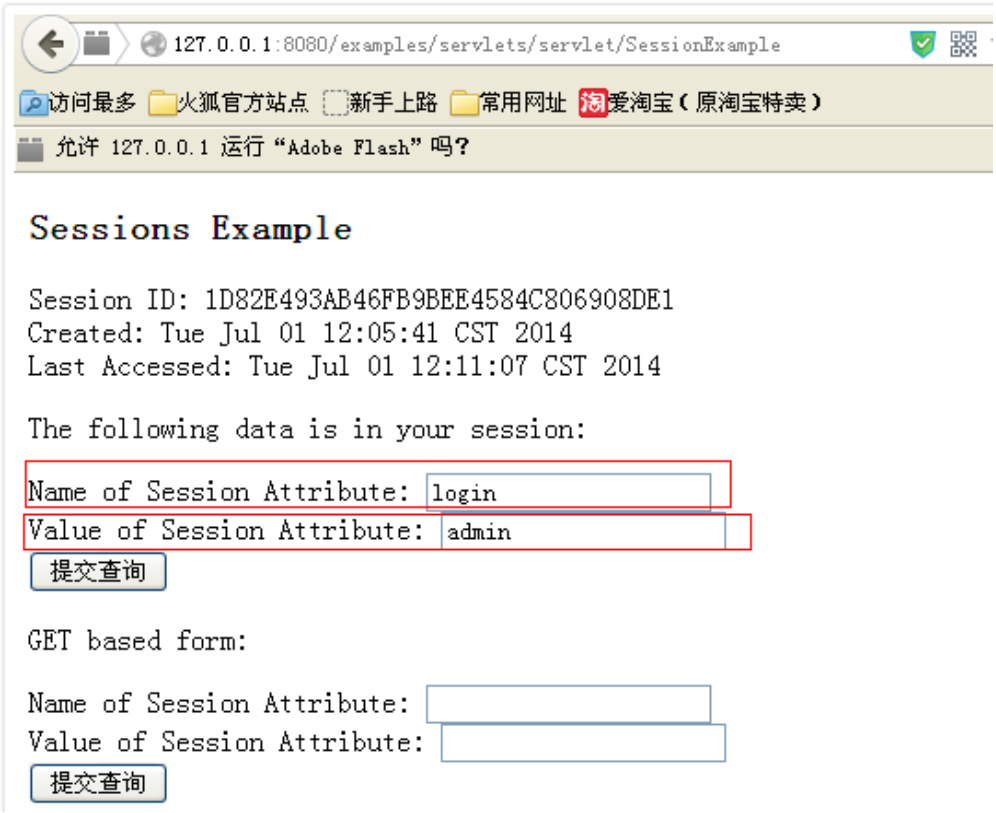
session进入网站后台

打开SessionExample

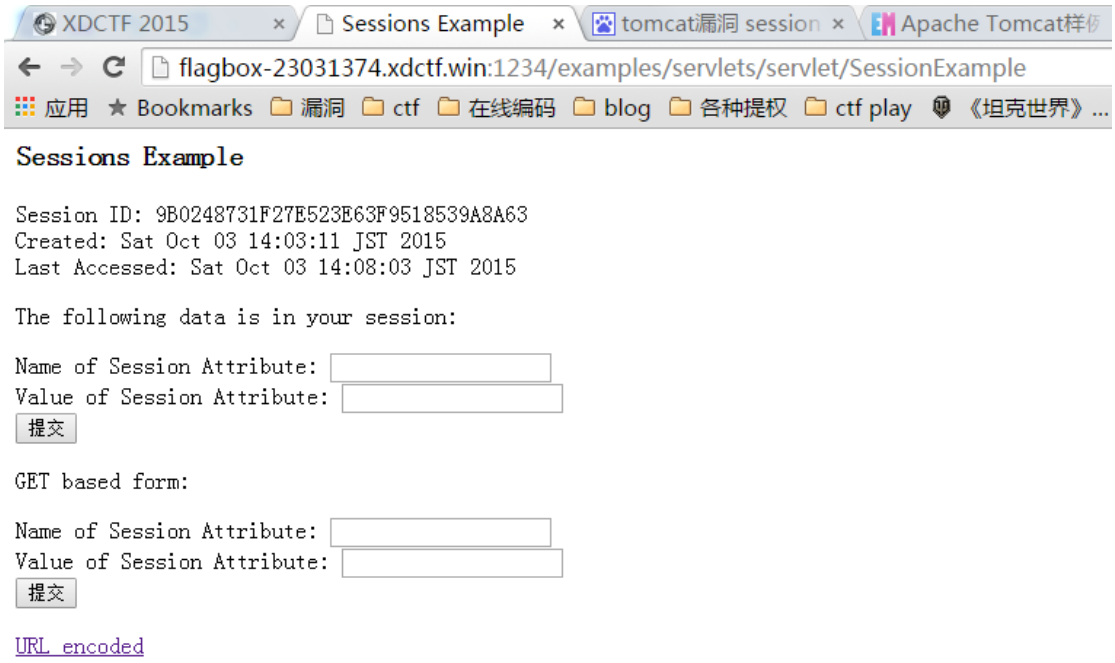
http://127.0.0.1:8080/examples/servlets/servlet/SessionExample

在Name of Session Attribute: 里输入login

在Value of Session Attribute:里输入admin

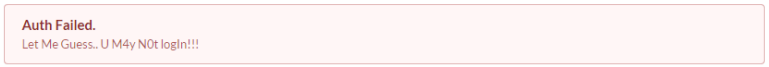


迅速调到这个页面

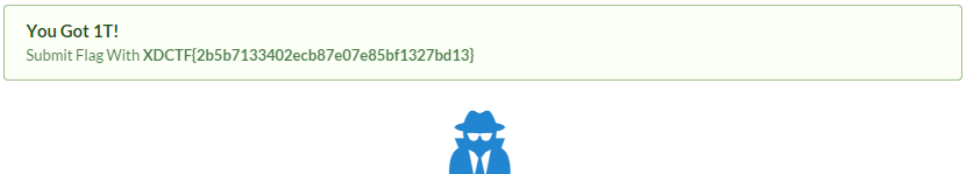


开始尝试伪造 cookie，开始是 login---amdin，发现并不是，主页提示

猜测是 Administrator，之后各种尝试，login=Administrator, Administrator=login, Auth=Administrator, Administrator=Auth, 直到输入 user= Administrator ，发现页面改了。。。于是开始考虑登录问题。。。

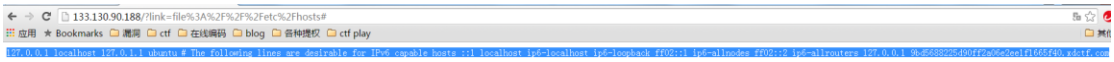


在 session 补上 login=true。。。发现过了



Web1-300

神奇的题目，进去之后一顿包含，包含了各种乱七八糟的东西无果，想到了 gctf 的一个题目，于是翻翻 gctf 的 writeup，找到了那个 ssrf 的题目，于是开始寻找内网地址，考虑到 linux 下地址解析与 windows 下相似的地点是存在 hosts 文件，尝试包含，发现了内网地址，



9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com，ssrf 访问回显为 1，尝试一下常见端口，22 是 ssh 正常，　　80 回显为 1。。唯一有不同的地方。。在 3389 端口发现了一个 discuz 论坛。。查查 discuz 的漏洞。。发现在 faq 有注入。。找了一大堆 payload 都不行。。最后。。在 csdn 的一篇博客找到了一个 payload 爆管理员密码的尝试可用

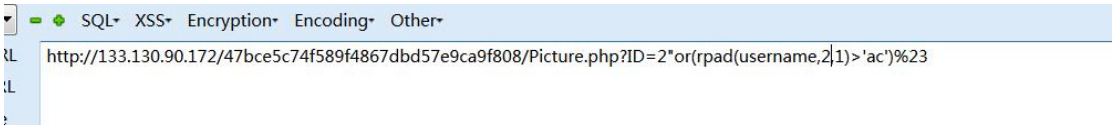
得到 flag。。。吐槽一下，这题目 lfi+ssrf+sqli。。真的好么。。。。

Web1-400

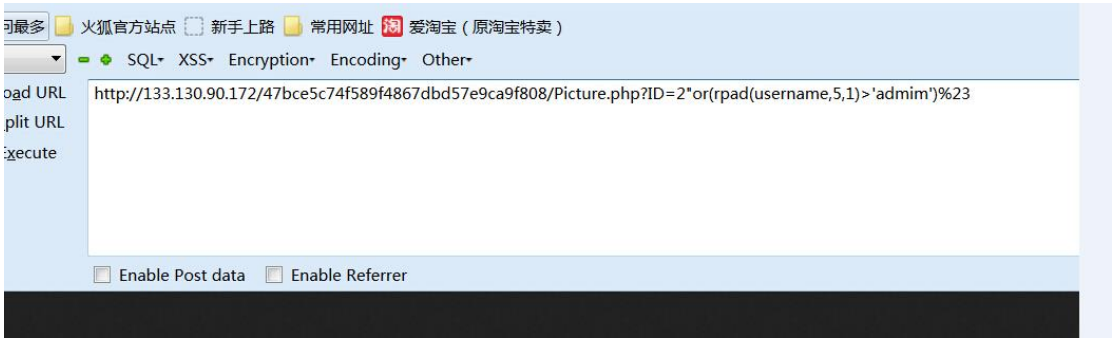
很恶心的题目，进去找了很久没找到题目考的是啥。。。把所有的东西 down 回来一个一个看。。在小马的图片后面发现有文字。。。知己访问发现似乎小马被解析成了 php，按照要求我们在后面用 ID 传参。尝试多次无果之后，准备放弃，直到出了 hint，提示“，再次尝试。。发现所有关键词都被过滤了。。在网上找找。。。google 一下关于 select，substring 被过滤怎么办。。找到了这个：

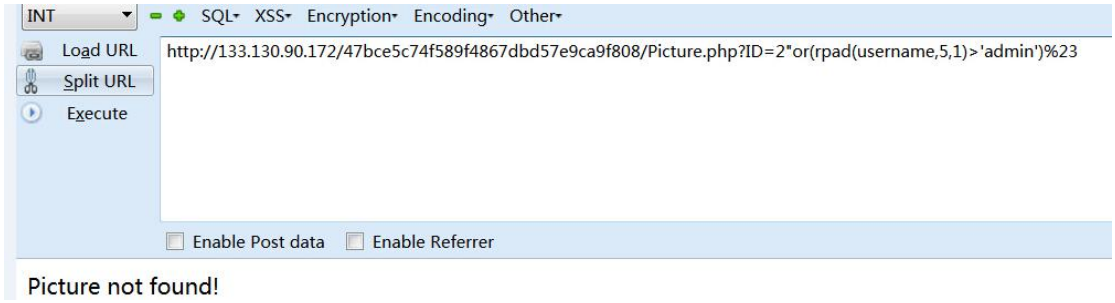
```
@@new: 0
@@log_bin: 1
#提取子字符串substr('abc',1,1) = 'a'
substr('abc' from 1 for 1) = 'a'
substring('abc',1,1) = 'a'
substring('abc' from 1 for 1) = 'a'
mid('abc',1,1) = 'a'
mid('abc' from 1 for 1) = 'a'
lpad('abc',1,space(1)) = 'a'
rpad('abc',1,space(1)) = 'a'
left('abc',1) = 'a'
reverse(right(reverse('abc'),1)) = 'a'
insert(insert('abc',1,0,space(0)),2,222,space(0)) = 'a'
space(0) = trim(version()from(version()))
```

可以用 lpad 和 rpad 写。。。之后手动注入几次，

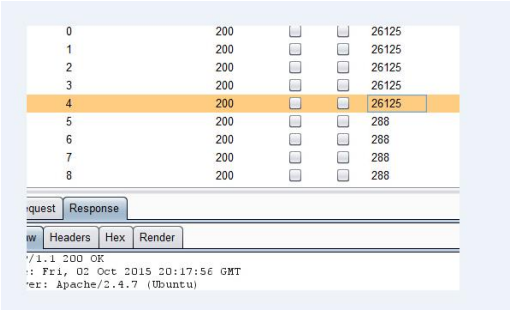


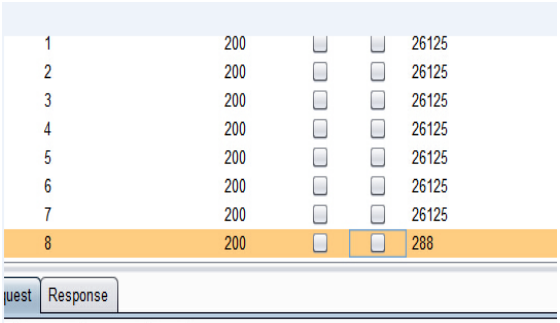
猜测用户名为 admin，

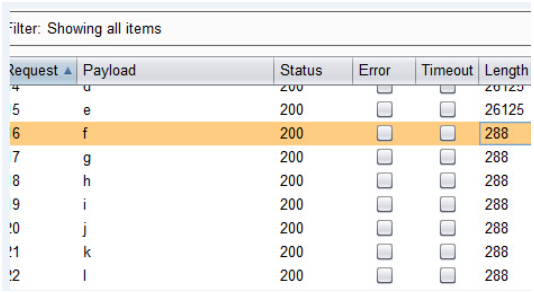


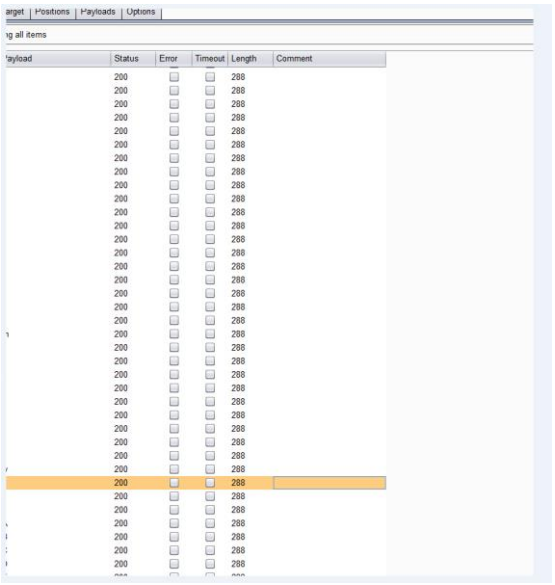


得到证实。。。于是证明方法可行，爆破密码，写了个脚本跑了半天也不知道啥错误什么都不显示。。。没办法 brupsuit 大法。。。手动跑。。做好了准备跑个 30 40 次。。。

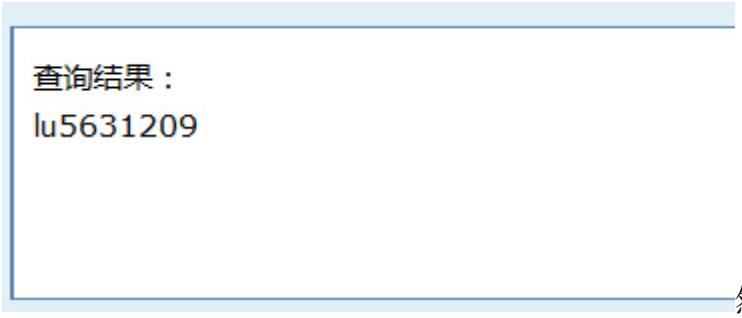








跑了 20 次，第二十一次都是错误。。再跑 22 页都是错误。。。发现一共才 20 位。。。。  ，之后就是尝试解密。。20 位既不是 md5 也不是十六进制。。纠结了很久想到。。应该是织梦独特的加密方式，前去三后去一的 md5.。尝试一下果然解出来了



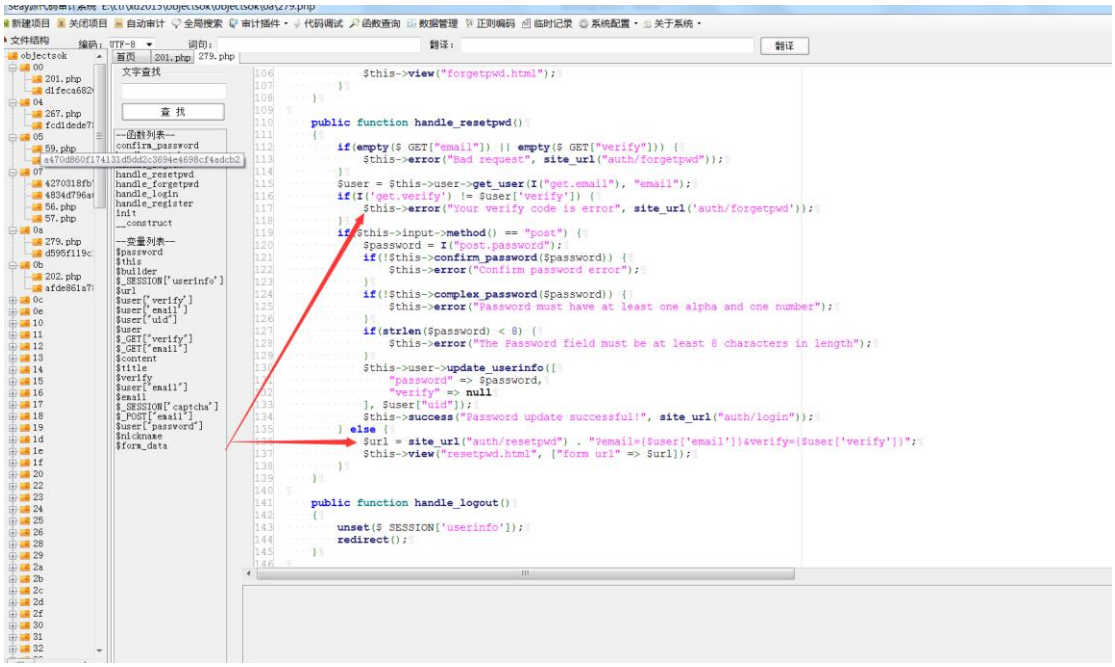
然后登陆进去。。找到 flag

Flag：XDCTF{e0a345cadaba033073d88d2cc5dce2f7}

Web2

Web2-100

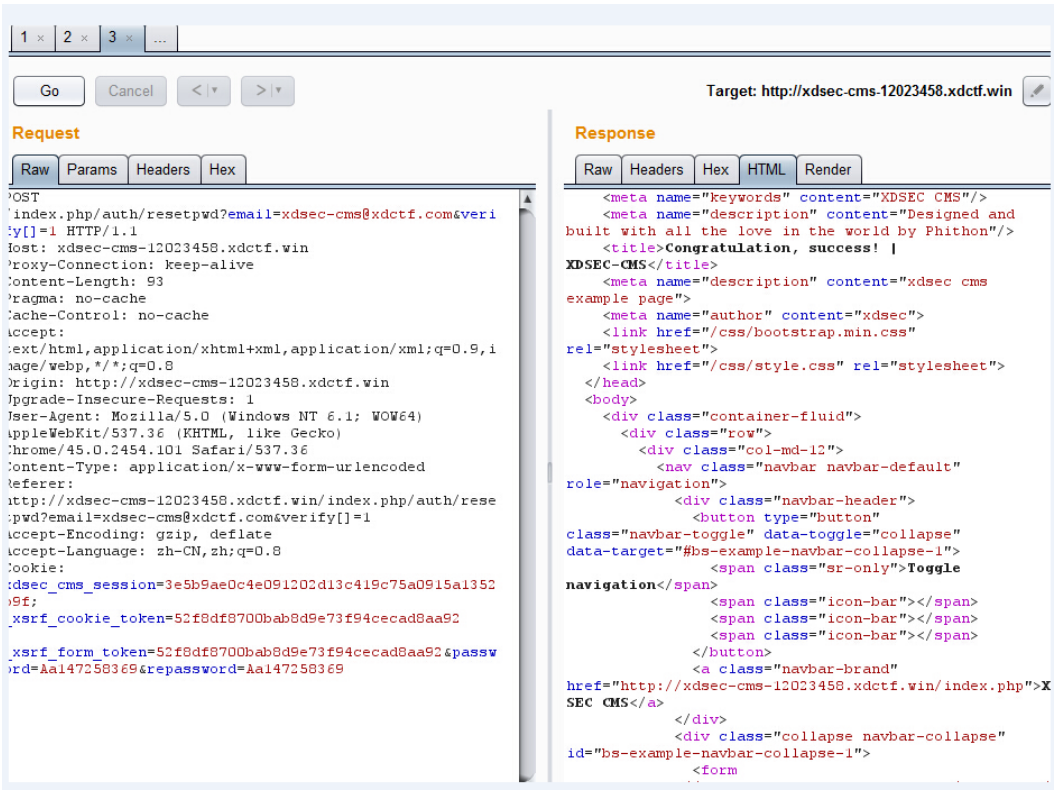
吐槽一下 100 分的题目这么难。。。。。。。真是  ，回归正题，题目给的是源码审计。。。。上一步的源码一顿解密，得到了大量的数字开头的 php，慢慢看，用



```
        $this->view("forgetpwd.html");
    }

    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]);
        }
    }

    public function handle_logout()
    {
        unset($_SESSION['userinfo']);
        redirect();
    }
```

审计一下，找到了一个关于修改密码的解析漏洞，存在于说可以 email 后跟着明文邮箱，加

上之前或许到的管理员邮箱 xdsec-cms@xdctf.com，我们可以尝试直接修改管理员的密码。。。。之后重点集中在了绕过 verify 的验证上。。。。尝试了很多方法，最后想起来之前在好像是国外还是哪里的比赛中做过一个关于绕过 password 的题目，利用 php 的特性构造 password[]=1 绕过，在本题同样尝试，verify[]=1，果然进入了修改密码的界面。。但是提交之后弹出错误的 verify。。。于是改用抓包方式。。成功修改密码，进入管理员后台，在个人信息中找到 flag



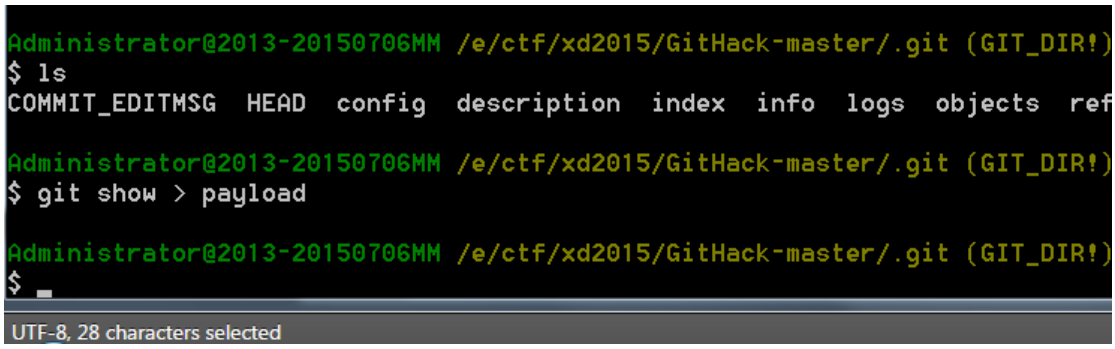
Congratulation, this is the [XDSEC-CMS] flag 2

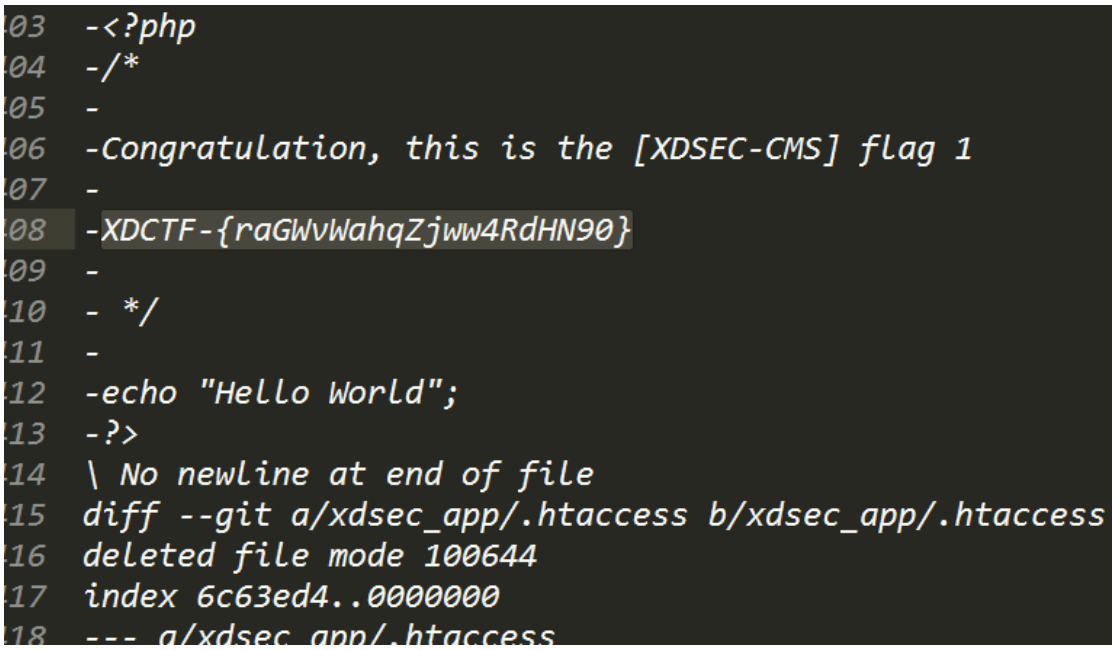
XDCTF-{i32mX4WK1gwEE9S9Oxd2}

hint:
admin url is /th3r315admin.php

Web2-200

进去根据提示/.git 源码泄露，GitHack 弄了半天只有两个文件。。。然后 github 上继续找。找到三四个工具吧。。。当时用的是 https://github.com/kost/dvcs-ripper 这个中的 rip-git.pl，linux 下运行 perl rip-git.pl *-v -u http://* *http://xdsec-cms-12023458.xdctf.win/.git/*得到大量的乱码文件。。。之后尝试解码失败，在晚上翻一番 *git* 的命令，找到了一个 *git show*。。。在 *linux* 下怎么运行都不行。。提示 *head* 错误。。。最后尝试在 *windows* 下运行，成功。。如下图。。。*git show > payload*，*sublime-Text* 打开。。看了一会儿找到了 *flag*

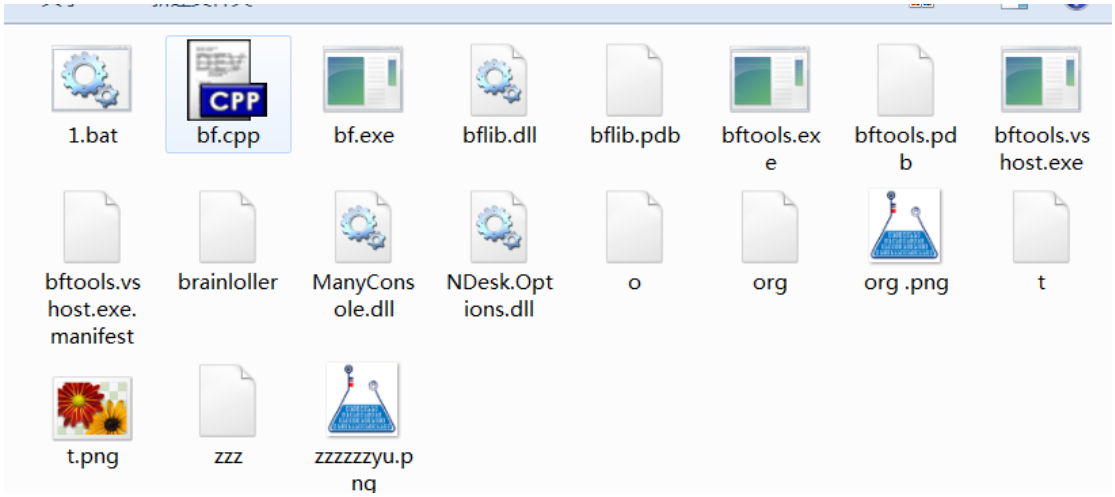




Misc

Misc100

第一天，没人做出来。。。提示了 brain 还是不懂。。最后提示 braintools。。。google 很久。。终于在 github 上找到了一个开源的项目



下下来编译下，得到这个

```
E:\ctf\xd2015\Debug>bftools.exe  decode braincopter "org .png" -o org

E:\ctf\xd2015\Debug>bftools.exe  decode braincopter zzzzzzyu.png -o zzz
```

然后 zzz 的东西是 brainfuck 的代码。。。org 的没找到。。。。下个 c 语言的 brainfuck 编译器，如第一张图片的 bf.cpp 和编译生成的 bf.exe，

```
E:\ctf\xd2015\Debug>bf.exe t
XDCTF{ji910-dad9jq0-iopuno}
E:\ctf\xd2015\Debug>
```

运行得到 flag

Misc200

拿到文件完全不知道是什么文件，binwalk –e 跑一下得到了一个压缩包有密码，里头有两个文件，flag.txt 和 readme.txt，还跑出来一个文本文档 readme.txt 可以直接打开，当然还有一大堆东西。。。然后在网上找了半天，找到了这个：

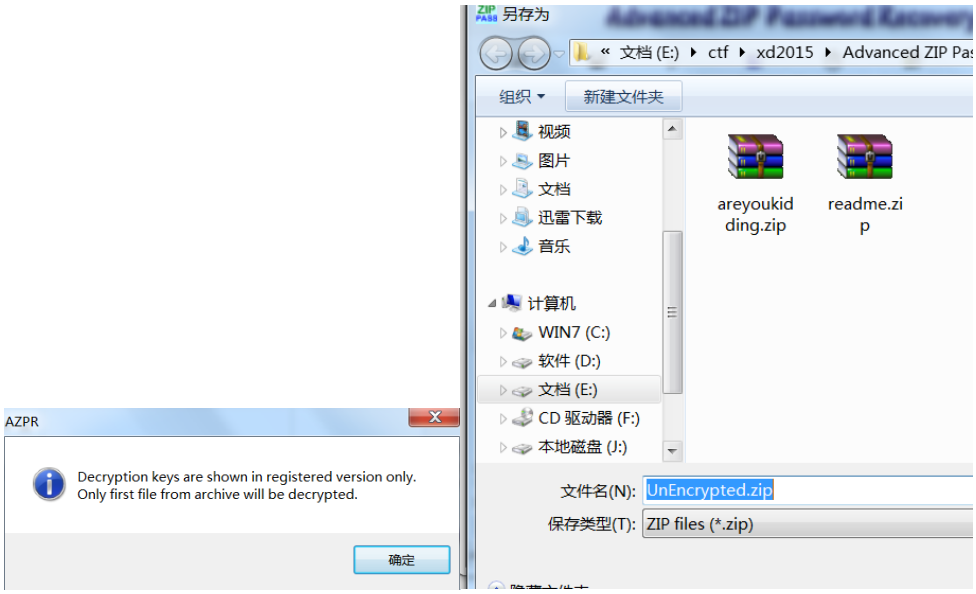
我自己找到答案了。

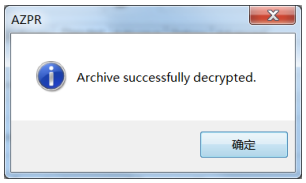
有一种破解方法叫做Known plaintext attack。市面上的密码破解软件几乎都带有这个功能。操作方法就是找到加密压缩包中的任意一个文件，用同样的压缩软件同样的压缩方式压缩成一个不加密的包，然后把这两个压缩包进行比较，这样就能把整个加密的压缩包全部还原成未加密的形式。

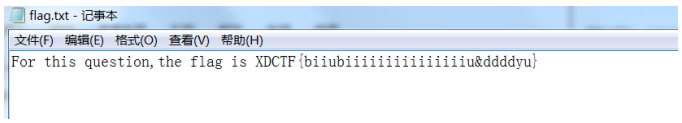
原理是这样的：你输入的密码，首先被转换成3个32bit的key，所以可能的key的组合是2^96，这是个天文数字，如果用暴力穷举的方式是不太可能的，除非你的密码比较短或者有个厉害的字典。压缩软件用这3个key加密所有包中的文件，这也就是说，所有文件的key是一样的，如果我们能够找到这个key，就能解开所有的文件。如果我们找到加密压缩包中的任意一个文件，这个文件和压缩包里的文件是一样的，我们把这个文件用同样的压缩软件同样的压缩方式进行无密码的压缩，得到的文件就是我们的Known plaintext。用这个无密码的压缩包和有密码的压缩包进行比较，分析两个包中相同的那个文件，抽取出两个文件的不同点，就是那3个key了，如此就能得到key。两个相同文件在压缩包中的字节数应该相差12个byte，就是那3个key了。虽然我们还是无法通过这个key还原出密码，但是我们已经可以用这个key解开所有的文件，所以已经满足我的要求了，而且要以前的密码也没什么用呀，我只要文件。

其实很简单，只要你能找到一个相同的文件，就万事大吉了！
所以我用Advanced ZIP Password Recovery（Crack了的）选择Known plaintext attack。用了2分钟就找到了key，然后用了20分钟把我的那个700M的压缩包解开，搞定！

http://bbs.csdn.net/topics/10444536，然后压缩 readme.txt，破解，得到 flag ,用时一分钟







flag: XDCTF{biiubiiiiiiiiiiiiiiu&ddddyu}

CRYPT

CRYPT200

这道题上来没什么思路，后来队友说在网上有这题的源码，就去看了看，得知本题考察了“CBC 字节翻转攻击技术”。后来我在网上找到了实现这项攻击的源码，修改了一下脚本的源码，就得到了包含”;admin=true”的密文。
脚本核心部分：

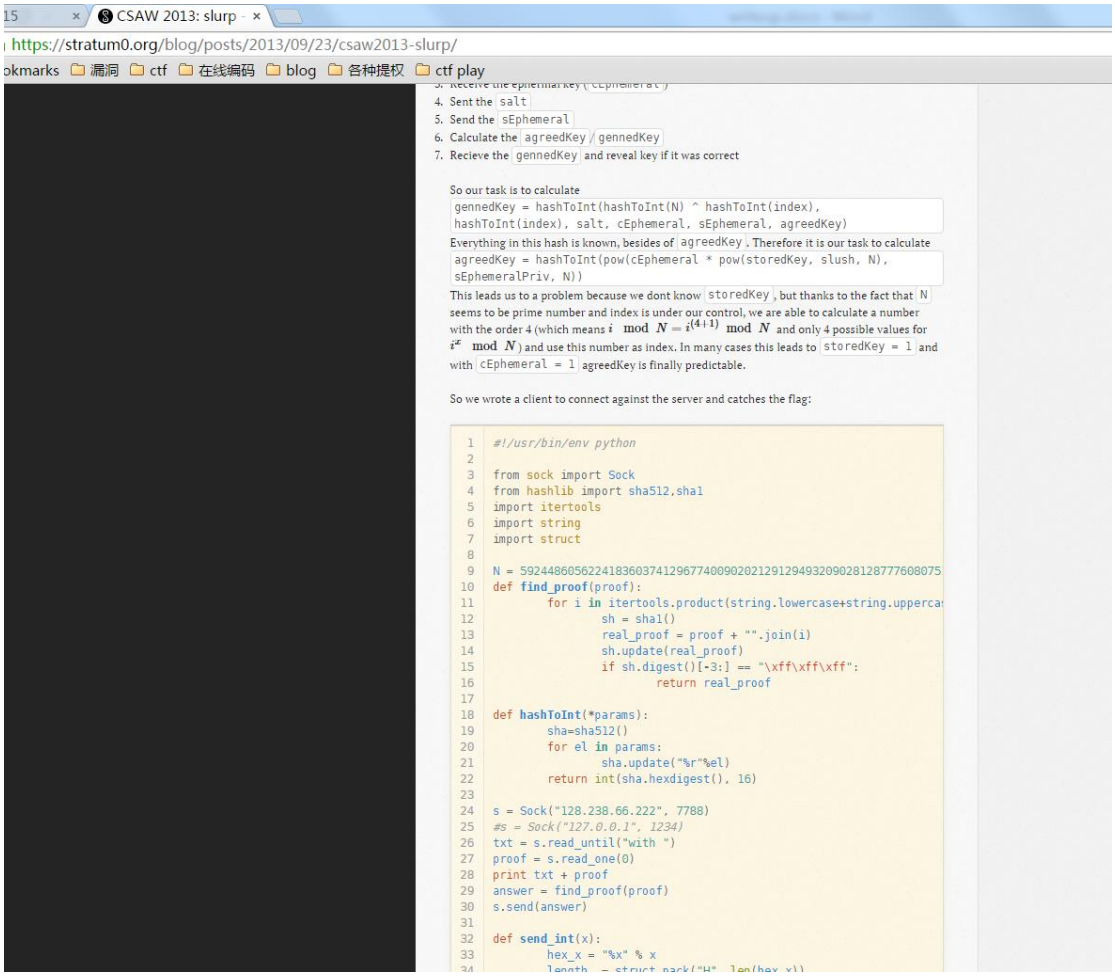
```
$badData = 'aaaaaaaaaaaaaaaabbbb;admin=true';
$goodData = 'aaaaaaaaaaaaaaaabbbbbbbbbbbbbbb';
//print $badData^$goodData;
$bitMask = substr($badData ^ $goodData, 16);
//print "$bitMask";

//      $query = getQuery($goodData, $key, $iv);
$query =
hex2bin("684299166a05383e6eaa9139f8d8f5ff8cda560698b1987eb2092534397496b777ea1db5
75b235603602ad563bac34301952f5c1a7a546fa40b5d47361ecf5a0c91068b6ab2193f3718da5ce
8698eb86aa1ed0ee6dcd83c11fea98b1ef00a9a8a4f4256564f5246a2d4e8f8b1ea9c6ea");

for ($i = 32; $i < 47; $i++) {
      $query[$i] = $query[$i] ^ $bitMask[$i - 32];
}

print "Querystring has admin=true:\n";
//print isAdmin($query, $key, $iv) ? "Yes\n\n" : "No :(";
print bin2hex($query);
```

```
D:\Mytools>nc 133.130.52.128 6666
parse:684299166a05383e6eaa9139f8d8f5ff8cda560698b1987eb2092534397496b777ea1db52c
b1336f3d0ef2402bbb33301952f5c1a7a546fa40b5d47361ecf5a0c91068b6ab2193f3718da5ce86
98eb86aa1ed0ee6dcd83c11fea98b1ef00a9a8a4f4256564f5246a2d4e8f8b1ea9c6ea
Parsing Profile...Congratulations!
The FLAG is: XDCTF{WelcometToCyberEngineeringSchool2333}
```

flag : XDCTF{WelcometToCyberEngineeringSchool2333}

CRYPT300

同样是源码，CSAW2013 的题目，以前做过，拿过来什么都不用改，把网上的 sock 模块改成 socket，N 一变直接得到 flag



。。。源码在网上找的 https://stratum0.org/blog/posts/2013/09/23/csaw2013-slurp/

Flag; XDCTF{alohauuuup^yourniversity2333}

PWN

pwn100

拿到样本，用 Winhex 打开看到：

```
00000000  7B 5C 72 74 66 31 7B 5C  66 6F 6E 74 74 62 6C 7B  {\rtf1{\fonttbl{
00000010  5C 66 30 5C 66 6E 69 6C  5C 66 63 68 61 72 73 65  \f0\fnil\fcharse
00000020  74 30 20 56 65 72 64 61  6E 61 3B 7D 7D 5C 76 69  t0 Verdana;}}\vi
00000030  65 77 6B 69 6E 64 34 5C  75 63 31 5C 70 61 72 64  ewkind4\uc1\pard
00000040  5C 73 62 31 30 30 5C 73  61 31 30 30 5C 6C 61 6E  \sb100\sa100\lan
00000050  67 39 5C 66 30 5C 66 73  32 32 5C 70 61 72 5C 70  g9\f0\fs22\par\p
00000060  61 72 64 5C 73 61 32 30  30 5C 73 6C 32 37 36 5C  ard\sa200\sl276\
00000070  73 6C 6D 75 6C 74 31 5C  6C 61 6E 67 39 5C 66 73  slmult1\lang9\fs
00000080  32 32 5C 70 61 72 7B 5C  6F 62 6A 65 63 74 5C 6F  22\par{\object\o
00000090  62 6A 6F 63 78 7B 5C 2A  5C 6F 62 6A 64 61 74 61  bjocx{\*\objdata
000000A0  0A 30 31 30 35 30 30 30  30 30 30 32 30 30 30 30  .010500000200000
000000B0  30 31 42 30 30 30 30 30  30 34 44 35 33 34 33 36  01B0000004D53436
```

得知样本是一个 rtf 文件，上网搜索与 rtf 有关的漏洞，得到以下网址：https://www.exploit-db.com/exploits/18780/

根据网页中的代码，将 shellcode 提取出来，跑完解密部分即可得到 flag：

```
00428387  E0 75 05 BB 47 13 72 6F  6A 00 53 FF D5 63 6D 64  鄒禳■roj.Sÿ舖md
00428397  2E 65 78 65 20 2F 63 20  65 63 68 6F 20 78 64 63  .exe /c echo xdc
004283A7  74 66 7B 64 34 5F 35 68  31 5F 66 75 5F 64 34 31  tf{d4_5h1_fu_d41
004283B7  5F 77 30 5F 66 33 31 7D  20 3E 20 63 3A 5C 66 6C  _w0_f31} > c:\fl
004283C7  61 67 2E 74 78 74 20 26  26 20 61 74 74 72 69 62  ag.txt && attrib
004283D7  20 63 3A 5C 66 6C 61 67  2E 74 78 74 20 2B 68 00   c:\flag.txt +h.
```

flag: xdctf{d4_5h1_fu_d41_w0_f31}

pwn300

类似于堆溢出的一个双向链表操作导致的任意地址可写。该程序支持以下几种操作：

```
switch ( Operation )
{
  case 1:
    add();
    break;
  case 2:
    delete();
    break;
  case 3:
    edit();
    break;
  case 4:
    show();
    break;
  case 5:
    exit(0);
    return result;
  default:
    puts(byte_8048E9D);
    break;
}
```

add 操作用于增加一个节点，delete 操作用于删除一个节点（但是写的有些错误，可能是故意为之？），edit 用于修改节点数据，show 用于显示节点数据。

这四种操作中，edit 没有对用户输入进行检查，导致可以覆盖下一个节点的头部。
于是有了以下思路：
增加三个节点，用节点 0 的数据覆盖节点 1 的头部（这里的数据将改变程序中一个数组的内容，这个数组保存着所有节点数据地址的起点，将其中索引为 0 的值修改为固定地址 0x0804b110，此处为 shellcode 的起点），用节点 1 的数据覆盖节点 2 的头部（这里的数据将改变位于 0x0804b00c 处的函数表的地址，此处我覆盖了 puts 的地址，这样程序返回执行 puts 时将跳到 shellcode 中），然后 delete1 号节点，写入 shellcode，然后 delete2 号节点，这样就拿到了 shell。拿到 shell 后 ls 发现 flag 就在当前目录下，cat flag 拿到 flag。

```
Give the Girl id to delete:
2

ls
ex3
flag
cat flag
XDCTF{Chu_ren_CEO_y1ng_Qu_b4i_fu_M31}
```

flag: XDCTF{Chu_ren_CEO_y1ng_Qu_b4i_fu_M31}

pwn400

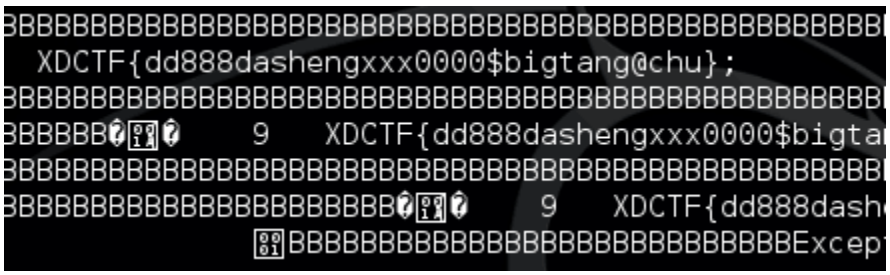
拿到这个程序，发现它将 flag 读入到内存里，就想到有没有可能是内存泄露。
继续分析程序，

```
loc_8048FD1:                            ; CODE XREF: commands+259↑j
                mov     eax, [ebp+src_startpk12]
                add     eax, 1Ch
                mov     [ebp+var_68], eax
                lea     eax, [ebp+var_68]
                mov     [esp], eax
                call    getfnlen
                mov     [ebp+fnlen], ax
                movzx   eax, [ebp+fnlen]
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aFileNameLength ; "file name length is %d\n"
                call     printf
```

发现此处获取了文件名的长度，存在栈中。继续往下：

```
movzx   eax, [ebp+fnlen]
add     eax, 2
mov     [ebp+var_20], ax
movzx   eax, [ebp+var_20]
mov     ecx, [ebp+buf]
mov     edx, [ebp+src_startpk12]
sub     ecx, edx
mov     edx, [ebp+datasize]
add     edx, ecx
sub     edx, 2Eh
cmp     eax, edx
jle     short loc_8049037
```

注意到此处存放到 var_20 中的是 ax，再以零扩展的方式放入 eax 中。所以，如果我们构造一个特殊的请求，使得 fnlen+2 产生加法溢出（此处我使 fnlen=0xFFFF），即可触发此漏洞。
效果如下：



flag : XDCTF{dd888dashengxxx0000$bigtang@chu}

感谢西电让我们体会到了这么一场高水平的比赛，无论是比赛官方出题还是管理员都很负责，谢谢。